Lecture 12: Constrained Optimization

The “kernel trick”

- Many machine learning algorithms only involve the data through ***inner products***
- For many interesting feature maps Φ, the function

$$k(\mathbf{x}, \mathbf{x}') := \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

has a simple, closed form expression that can be evaluated ***without explicitly calculating*** $\Phi(\mathbf{x})$ and $\Phi(\mathbf{x}')$

Homogenous kernel ridge regression

For many kernels, $\Phi(\mathbf{x})$ already contains a constant component, in which case we often omit β_0

Examples

- inhomogenous polynomial kernel
- Gaussian kernel does not seem to require a constant

In this case, the kernel ridge regression solution becomes

$$\widehat{f}(\mathbf{x}) = \mathbf{y}^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x})$$

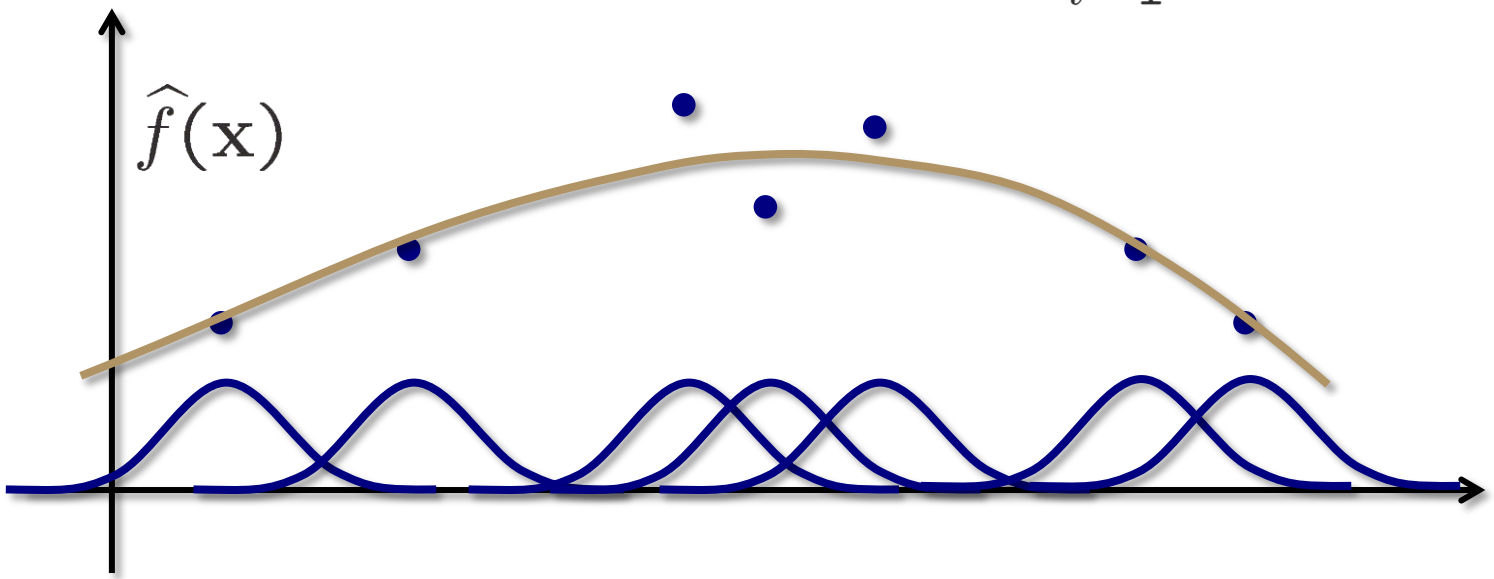
where

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \mathbf{k}(\mathbf{x}) = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x} \\ \vdots \\ \mathbf{x}_n^T\mathbf{x} \end{bmatrix}$$

Example: Gaussian kernel

$$k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|^2}{2\sigma^2}\right)$$

$$\begin{aligned}\widehat{f}(\mathbf{x}) &= \mathbf{y}^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}) \\ &= \boldsymbol{\alpha}^T\mathbf{k}(\mathbf{x}) = \sum_{i=1}^{n} \alpha(i)k(\mathbf{x}, \mathbf{x}_i)\end{aligned}$$



The “kernel trick” in classification

What about using kernels for designing classifiers?

It is possible to “kernelize”

- LDA
- Logistic regression
- Perceptron learning algorithm
- Maximum margin hyperplanes
  - ***support vector machines***

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \ \ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, n$$

$$\xi_i \geq 0 \quad i = 1, \ldots, n$$

Detour

In order to understand both

- how to kernelize the maximum margin optimization problem
- how to actually solve this optimization problem with a practical algorithm

we will need to spend a bit of time learning about constrained optimization

This stuff is super useful, even outside of this context

Constrained optimization

A general constrained optimization problem has the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 \quad i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0 \quad i = 1, \ldots, p \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^d$

We call $f(\mathbf{x})$ the ***objective function***

If $\mathbf{x}$ satisfies all the constraints, we say that $\mathbf{x}$ is ***feasible***

We will assume that $f(\mathbf{x})$ is defined for all feasible $\mathbf{x}$

Lagrangian duality

By considering the ***dual***, we can ***bound*** or ***solve*** the original optimization problem via a different optimization problem

The ***Lagrangian function*** is

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) := f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \sum_{i=1}^{p} \nu_i h_i(\mathbf{x})$$

$\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_m]^T$ and $\boldsymbol{\nu} = [\nu_1, \ldots, \nu_p]^T$ are called ***Lagrange multipliers*** or ***dual variables***

The ***(Lagrange) dual function*** is

$$L_D(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \min_{\mathbf{x}} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

The dual optimization problem

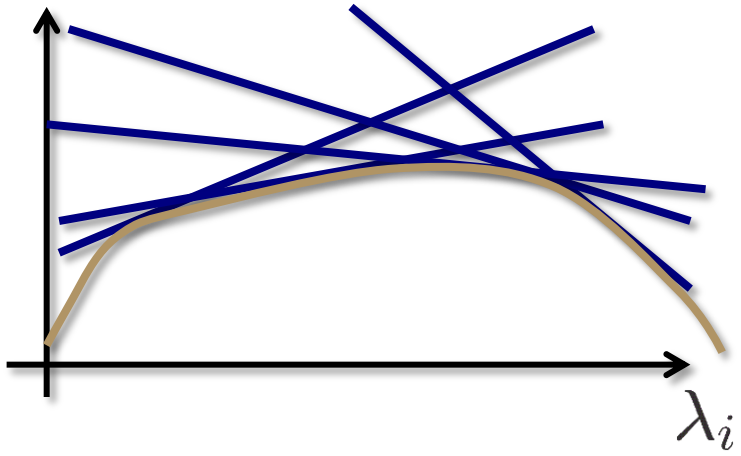
We can then define the ***dual optimization problem***

$$\max_{\boldsymbol{\lambda},\boldsymbol{\nu}} \; L_D(\boldsymbol{\lambda},\boldsymbol{\nu})$$
$$\text{s.t.} \;\; \lambda_i \geq 0 \quad i = 1,\ldots,m$$

Why do we constrain $\lambda_i \geq 0$?

Note: No matter the choice of $f, g_i,$ or h_i, $L_D(\boldsymbol{\lambda},\boldsymbol{\nu})$ is always ***concave***

It is the point-wise minimum of a family of affine functions



The primal optimization problem

The ***primal function*** is

$$L_P(\mathbf{x}) := \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

and the ***primal optimization problem*** is

$$\min_{\mathbf{x}} L_P(\mathbf{x}) = \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

Contrast this with the dual optimization problem

$$\max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

What is so "primal" about the primal?

$$\min_{\mathbf{x}} L_P(\mathbf{x}) = \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

Suppose $\mathbf{x}$ is feasible

$$L_P(\mathbf{x}) = \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \sum_{i=1}^{p} \nu_i h_i(\mathbf{x})$$

$$= f(\mathbf{x})$$

Suppose $\mathbf{x}$ is not feasible

$$L_P(\mathbf{x}) = \infty$$

The primal encompasses the original problem (i.e., they have the same solution), yet it is ***unconstrained***

Weak duality

Theorem

$$d^* := \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \min_{\mathbf{x}} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

$$\leq \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) =: p^*$$

Proof

Let $\widetilde{\mathbf{x}}$ be feasible. Then for any $\boldsymbol{\lambda}, \boldsymbol{\nu}$ with $\lambda_i \geq 0$,

$$L(\widetilde{\mathbf{x}}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\widetilde{\mathbf{x}}) + \sum_i \lambda_i g_i(\widetilde{\mathbf{x}}) + \sum_i \nu_i h_i(\widetilde{\mathbf{x}}) \leq f(\widetilde{\mathbf{x}})$$

Hence $L_D(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \min_{\mathbf{x}} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$

$$\leq \min_{\text{feasible } \tilde{\mathbf{x}}} \; L(\widetilde{\mathbf{x}}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

$$\leq \min_{\text{feasible } \tilde{\mathbf{x}}} \; f(\widetilde{\mathbf{x}}) = p^*$$

Weak duality

Theorem

$$d^* := \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \min_{\mathbf{x}} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

$$\leq \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) =: p^*$$

Proof

Thus, for any $\boldsymbol{\lambda}, \boldsymbol{\nu}$ with $\lambda_i \geq 0$, we have

$$\min_{\mathbf{x}} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^*$$

Since this holds for any $\boldsymbol{\lambda}, \boldsymbol{\nu}$, we can take the max to obtain

$$d^* = \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \min_{\mathbf{x}} \; L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^*$$

Duality gap

We have just shown that for any optimization problem, we can solve the dual (which is always a concave maximization problem), and obtain a lower bound d^* on p^* (the optimal value of the objective function for the original problem)

The difference $p^* - d^*$ is called the ***duality gap***

In general, all we can say about the duality gap is that $p^* - d^* \geq 0$, but sometimes we are lucky…

Strong duality

If $p^* = d^*$, we say that ***strong duality*** holds

Theorem

If the original problem is ***convex***, meaning that f and the g_i are convex and the h_i are affine, then under certain ***constraint qualifications***, $p^* = d^*$.

Example constraint qualifications

- All the g_i are affine
- Strict feasibility: There exists an $\mathbf{x}$ such that $h_i(\mathbf{x}) = 0$ for all i and $g_i(\mathbf{x}) < 0$ for all i

The KKT conditions

Assume that f, g_i, h_i are all differentiable

The Karush-Kuhn-Tucker (KKT) conditions are a set of properties that must hold under strong duality

The KKT conditions will allow us to translate between solutions of the primal and dual optimization problems

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} L_P(\mathbf{x})$$

$$(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \arg\max_{\boldsymbol{\lambda}, \boldsymbol{\nu}: \lambda_i \geq 0} L_D(\boldsymbol{\lambda}, \boldsymbol{\nu})$$

The KKT conditions

1. $\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(\mathbf{x}^*) = 0$

2. $g_i(\mathbf{x}^*) \leq 0, \quad i = 1, \ldots, m$

3. $h_i(\mathbf{x}^*) = 0, \quad i = 1, \ldots, p$

4. $\lambda_i^* \geq 0, \quad i = 1, \ldots, m$

5. $\lambda_i^* g_i(\mathbf{x}^*) = 0 \quad i = 1, \ldots, m$
   (complementary slackness)

KKT conditions: Necessity

Theorem

If $p^* = d^*$, $\mathbf{x}^*$ is primal optimal, and $(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ is dual optimal, then the KKT conditions hold.

Proof

- 2 and 3 hold because $\mathbf{x}^*$ must be feasible
- 4 holds by the definition of the dual problem
- To prove 5, note that from strong duality, we have

$$\begin{aligned} f(\mathbf{x}^*) &= L_D(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) \\ &= \min_{\mathbf{x}} \; f(\mathbf{x}) + \sum_i \lambda_i^* g_i(\mathbf{x}) + \sum_i \nu_i^* h_i(\mathbf{x}) \\ &\leq f(\mathbf{x}^*) + \sum_i \lambda_i^* g_i(\mathbf{x}^*) + \sum_i \nu_i^* h_i(\mathbf{x}^*) \\ &\leq f(\mathbf{x}^*) \quad \text{(by 2,3, and 4)} \end{aligned}$$

KKT conditions: Necessity

- This shows that the two inequalities are actually equalities, and hence the equality of the last two lines implies

$$f(\mathbf{x}^*) + \sum_i \lambda_i^* g_i(\mathbf{x}^*) + \sum_i \nu_i^* h_i(\mathbf{x}^*) = f(\mathbf{x}^*)$$

  which implies that $\sum_i \lambda_i^* g_i(\mathbf{x}^*) = 0$, which is 5

- Equality of the 2nd and 3rd lines implies that $\mathbf{x}^*$ is a minimizer of $L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ with respect to $\mathbf{x}$, and hence

$$\nabla L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = 0$$

  which establishes 1

KKT conditions: Sufficiency

Theorem

If the original problem is convex, meaning that f and the g_i are convex and the h_i are affine, and $(\widetilde{\mathbf{x}}, \widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$ satisfy the KKT conditions, then $\widetilde{\mathbf{x}}$ is primal optimal, $(\widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$ is dual optimal, and the duality gap is zero.

Proof

- 2 and 3 imply that $\widetilde{\mathbf{x}}$ is feasible
- 4 implies that $L(\mathbf{x}, \widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$ is convex in $\mathbf{x}$, and so 1 means that $\widetilde{\mathbf{x}}$ is a minimizer of $L(\mathbf{x}, \widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$
- Thus $L_D(\widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}}) = L(\widetilde{\mathbf{x}}, \widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$

$$= f(\widetilde{\mathbf{x}}) + \sum_i \widetilde{\lambda}_i g_i(\widetilde{\mathbf{x}}) + \sum_i \widetilde{\nu}_i h_i(\widetilde{\mathbf{x}})$$

$$= f(\widetilde{\mathbf{x}}) \quad \text{(by feasibility and 5)}$$

KKT conditions: Sufficiency

- If $L_D(\widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}}) = f(\widetilde{\mathbf{x}})$ then $d^* = \max_{\boldsymbol{\lambda}, \boldsymbol{\nu}: \lambda_i \geq 0} L_D(\boldsymbol{\lambda}, \boldsymbol{\nu})$
$$\geq L_D(\widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$$
$$= f(\widetilde{\mathbf{x}})$$
$$\geq p^*$$

- But earlier we proved that $d^* \leq p^*$, and hence it must actually be that $d^* = p^*$, and thus we have
  - zero duality gap
  - $\widetilde{\mathbf{x}}$ is primal optimal
  - $(\widetilde{\boldsymbol{\lambda}}, \widetilde{\boldsymbol{\nu}})$ is dual optimal

KKT conditions: The bottom line

If a constrained optimization problem is

- differentiable
- convex

then the KKT conditions are necessary and sufficient for primal/dual optimality (with zero duality gap)

In this case, we can use the KKT conditions to find a solution to our optimization problem

i.e., if we find $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ satisfying the conditions, we have found solutions to both the primal and dual problems

Lecture 13: SVMs

Constrained optimization

A general constrained optimization problem has the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 \quad i = 1, \ldots, m \\ & h_i(\mathbf{x}) = 0 \quad i = 1, \ldots, p \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^d$

The Lagrangian function is given by

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) := f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \sum_{i=1}^{p} \nu_i h_i(\mathbf{x})$$

Primal and dual optimization problems

Primal: $\min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$

Dual: $\max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$

Weak duality:

$$d^* := \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

$$\leq \min_{\mathbf{x}} \max_{\boldsymbol{\lambda},\boldsymbol{\nu}:\lambda_i \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) =: p^*$$

Strong duality: For convex problems with affine constraints

$$d^* = p^*$$

KKT conditions: The bottom line

If a constrained optimization problem is

- differentiable
- convex

then the KKT conditions are necessary and sufficient for primal/dual optimality (with zero duality gap)

In this case, we can use the KKT conditions to find a solution to our optimization problem

i.e., if we find $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ satisfying the conditions, we have found solutions to both the primal and dual problems

The KKT conditions

1. $\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\mathbf{x}^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(\mathbf{x}^*) = 0$

2. $g_i(\mathbf{x}^*) \leq 0, \quad i = 1, \ldots, m$

3. $h_i(\mathbf{x}^*) = 0, \quad i = 1, \ldots, p$

4. $\lambda_i^* \geq 0, \quad i = 1, \ldots, m$

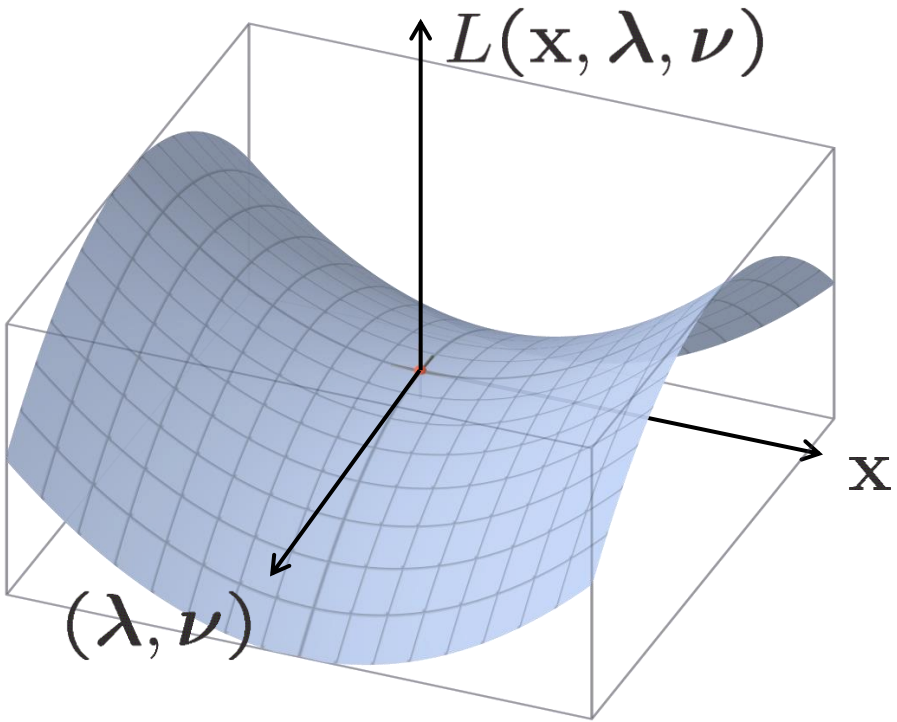
5. $\lambda_i^* g_i(\mathbf{x}^*) = 0 \quad i = 1, \ldots, m$
   (complementary slackness)

Saddle point property

If $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ are primal/dual optimal with zero duality gap, they are a ***saddle point*** of $L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$, i.e.,

$$L(\mathbf{x}^*, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq L(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) \leq (\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$$

for all $\mathbf{x} \in \mathbb{R}^d,\ \boldsymbol{\lambda} \in \mathbb{R}^m_+,\ \boldsymbol{\nu} \in \mathbb{R}^p$



Soft-margin classifier

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \ \ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, n$$

$$\xi_i \geq 0 \quad i = 1, \ldots, n$$

This optimization problem is differentiable and convex

- the KKT conditions and necessary and sufficient conditions for primal/dual optimality (with zero duality gap)
- we can use these conditions to find a relationship between the solutions of the primal and dual problems
- the dual optimization problem will be easy to “kernelize”

Forming the Lagrangian

Begin by converting our problem to the standard form

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \ \ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, n$$

$$\xi_i \geq 0 \quad i = 1, \ldots, n$$

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \ \ 1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0 \quad i = 1, \ldots, n$$

$$-\xi_i \leq 0 \quad i = 1, \ldots, n$$

Forming the Lagrangian

The Lagrangian function is then given by

Lagrange multipliers/dual variables

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$+ \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x}_i + b)) - \sum_{i=1}^{n}\beta_i\xi_i$$

Soft-margin dual

The Lagrangian dual is thus

$$L_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{w}, b, \boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

and the dual optimization problem is

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta}: \alpha_i, \beta_i \geq 0} L_D(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

Let's compute a simplified expression for $L_D(\boldsymbol{\alpha}, \boldsymbol{\beta})$

How?

By taking the gradient with respect to $(\mathbf{w}, b, \boldsymbol{\xi})$ and setting this equal to zero, and plugging the results back into the formula

Taking the gradient

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

$$= \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

$$+ \sum_{i=1}^{n}\alpha_i(1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x}_i + b)) - \sum_{i=1}^{n}\beta_i\xi_i$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{w} - \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = -\sum_{i=1}^{n}\alpha_i y_i = 0$$

$$\frac{\partial}{\partial \xi_i} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{C}{n} - \alpha_i - \beta_i = 0$$

Plugging this in

The dual function is thus

$$L_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_i \alpha_i$$

And the dual optimization problem can be written as

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_i \alpha_i$$

$$\text{s.t.} \quad \sum_i \alpha_i y_i = 0$$

$$\alpha_i + \beta_i = \frac{C}{n} \qquad \alpha_i, \beta_i \geq 0 \qquad i = 1, \ldots, n$$

Soft-margin dual quadratic program

We can eliminate β to obtain

$$\max_{\alpha} \quad -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_i \alpha_i$$

$$\text{s.t.} \quad \sum_i \alpha_i y_i = 0$$

$$0 \le \alpha_i \le \frac{C}{n} \qquad i = 1, \ldots, n$$

Note: Input patterns are only involved via ***inner products***

Recovering $\mathbf{w}^*$

Given α^* (the solution to the soft-margin dual), can we recover the optimal $\mathbf{w}^*$ and b?

Yes! Use the KKT conditions

From KKT condition 1, we know that

$$\mathbf{w}^* - \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i = 0$$

And thus the optimal normal vector is just a linear combination of our input patterns

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i$$

b is a little less obvious – we'll return to this in a minute

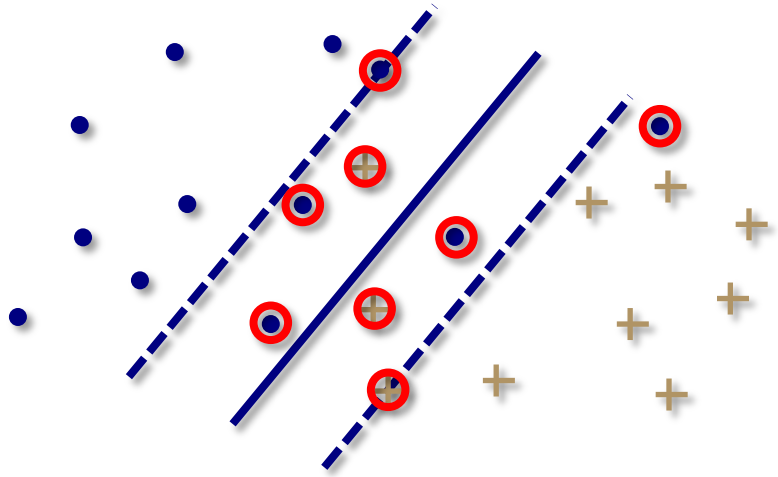
Support vectors

From KKT condition 5 (complementary slackness) we also have that for all i,

$$\alpha_i^* \left(1 - \xi_i^* - y_i \left(\mathbf{w}^{*T}\mathbf{x}_i + b^*\right)\right) = 0$$

The $\mathbf{x}_i$ for which $y_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) = 1 - \xi_i^*$ are called ***support vectors***

These are the points on or inside the margin of separation



Useful fact:

By the KKT condition, $\alpha_i^* \neq 0$ if and only if $\mathbf{x}_i$ is a support vector!

Empirical fact

It has been widely demonstrated (empirically) that in typical learning problems, only a small fraction of the training input patterns are support vectors

Thus, support vector machines produce a hyperplane with a ***sparse*** representation

$$\mathbf{w}^* = \sum_{\substack{\text{support} \\ \text{vectors}}} \alpha_i^* y_i \mathbf{x}_i$$

This is advantageous for efficient storage and evaluation

What about b^*?

Another consequence of the KKT conditions (condition 5) is that for all i, $\beta_i^* \xi_i^* = 0$

Since $\alpha_i^* + \beta_i^* = \frac{C}{n}$, this implies that if $\alpha_i^* < \frac{C}{n}$, then $\xi_i^* = 0$

Recall that if $\alpha_i^* > 0$ we also have that $\mathbf{x}_i$ is a support vector, and hence

$$y_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) = 1 - \xi_i^*$$

How can we combine these two facts to determine b^* ?

Recovering b^*

For any i such that $0 < \alpha_i^* < \frac{C}{n}$, we have

$$y_i(\mathbf{w}^{*T}\mathbf{x}_i + b^*) = 1$$

$$b^* = y_i - \mathbf{w}^{*T}\mathbf{x}_i$$

In practice, it is common to average over several such i to counter numerical imprecision

Support vector machines

Given an inner product kernel k, we can write the SVM classifier as

$$f(\mathbf{x}) = \text{sign}\left(\sum_i \alpha_i^* y_i k(\mathbf{x}, \mathbf{x}_i) + b^*\right)$$

where α^* is the solution of

$$\max_{\alpha} \; -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + \sum_i \alpha_i$$

$$\text{s.t.} \; \sum_i \alpha_i y_i = 0, \quad 0 \le \alpha_i \le \frac{C}{n} \quad i = 1, \ldots, n$$

and $b^* = y_i - \sum_j \alpha_j^* y_j k(\mathbf{x}_i, \mathbf{x}_j)$ for some i s.t. $0 < \alpha_i^* < \frac{C}{n}$

Remarks

- The final classifier depends only on the $\mathbf{x}_i$ with $\alpha_i > 0$, i.e., the ***support vectors***

- The size (number of variables) of the dual QP is n, independent of the kernel k, the mapping Φ, or the space $\mathcal{F}$
  - remarkable, since the dimension of $\mathcal{F}$ can be ***infinite***

- The soft-margin hyperplane was the first machine learning algorithm to be "kernelized", but since then the idea has been applied to many, many other algorithms
  - kernel ridge regression
  - kernel PCA
  - …

Solving the quadratic program

How can we actually compute the solution to

$$\max_{\alpha} \; -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j q_{ij} + \sum_i \alpha_i$$

$$\text{s.t.} \sum_i \alpha_i y_i = 0, \quad 0 \le \alpha_i \le \frac{C}{n} \quad i = 1, \ldots, n$$

where $q_{ij} := y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$?

There are several general approaches to soling quadratic programs, and many can be applied to solve the SVM dual

We will focus on a particular example that is very efficient and capitalizes on some of the unique structure in the SVM dual, called ***sequential minimal optimization (SMO)***

Sequential minimal optimization

SMO is an example of a ***decomposition*** algorithm

Sequential minimal optimization

Initialize: $\alpha = 0$

Repeat until stopping criteria satisfied

(1) Select a pair $i, j, 1 \leq i, j \leq n$

(2) Update α_i and α_j by optimizing the dual QP, holding all other $\alpha_k, k \neq i, j$ fixed

The reason for decomposing this to a two-variable subproblem is that this subproblem can be solved ***exactly*** via a simple ***analytic*** update

The update step

Choose α_i and α_j to solve

$$\max_{\alpha_i,\alpha_j} \; -\frac{1}{2}\left(\alpha_i^2 q_{ii} + \alpha_j^2 q_{jj} + 2\alpha_i\alpha_j q_{ij}\right) + c_i\alpha_i + c_j\alpha_j$$

$$\text{s.t. } \alpha_i y_i + \alpha_j y_j = -\sum_{k\neq i,j} \alpha_k y_k$$

$$0 \leq \alpha_i, \alpha_j \leq \frac{C}{n}$$

where $c_i = 1 - \frac{1}{2}\sum_{k\neq i,j} \alpha_k q_{ik}$ and similarly for c_j

SMO in practice

- Several strategies have been proposed for selecting (i, j) at each iteration
- Typically based on heuristics (often using the KKT conditions) that predict which pair of variables will lead to the largest change in the objective function
- For many of these heuristics, the SMO algorithm is proven to converge to the global optimum after finitely many iterations
- The running time is $O(n^3)$ in the worst case, but tends to be more like $O(n^2)$ in practice

Lecture 14: Nearest Neighbors

Methods for classification

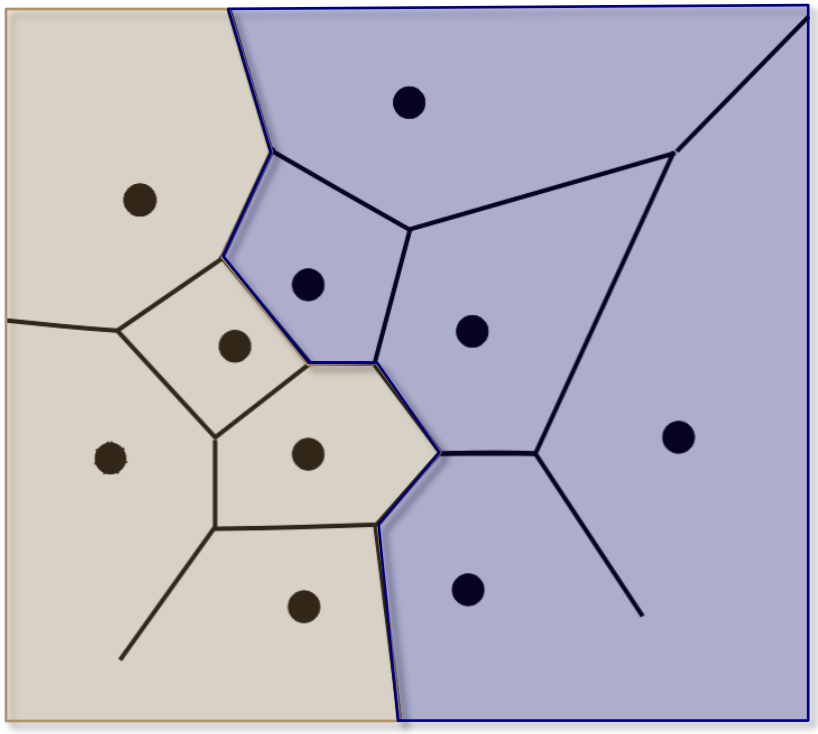
- Parametric methods
  - LDA
  - Logistic regression
  - Naïve bayes
- Nonparametric methods
  - PLA
  - support vector machines
- Nearest neighbor classifiers

Nearest neighbor classifier

The ***nearest neighbor classifier*** is easiest to state in words:

Assign $\mathbf{x}$ the same label as the closest training point $\mathbf{x}_i$ to $\mathbf{x}$

The nearest neighbor rule defines a ***Vornoi partition*** of the input space



Properties

The nearest neighbor classifier is

- nonparametric
- nonlinear
- easy to kernelize

A ***very simple*** alternative to support vector machines

Unfortunately, the nearest neighbor classifier performs rather poorly when the size of the training set n is small

k-nearest neighbors

For $k \geq 1$, the ***k-nearest neighbor*** rule generalizes the nearest neighbor rule:

Assign a label to $\mathbf{x}$ by taking a majority vote over the k training points $\mathbf{x}_i$ closest to $\mathbf{x}$

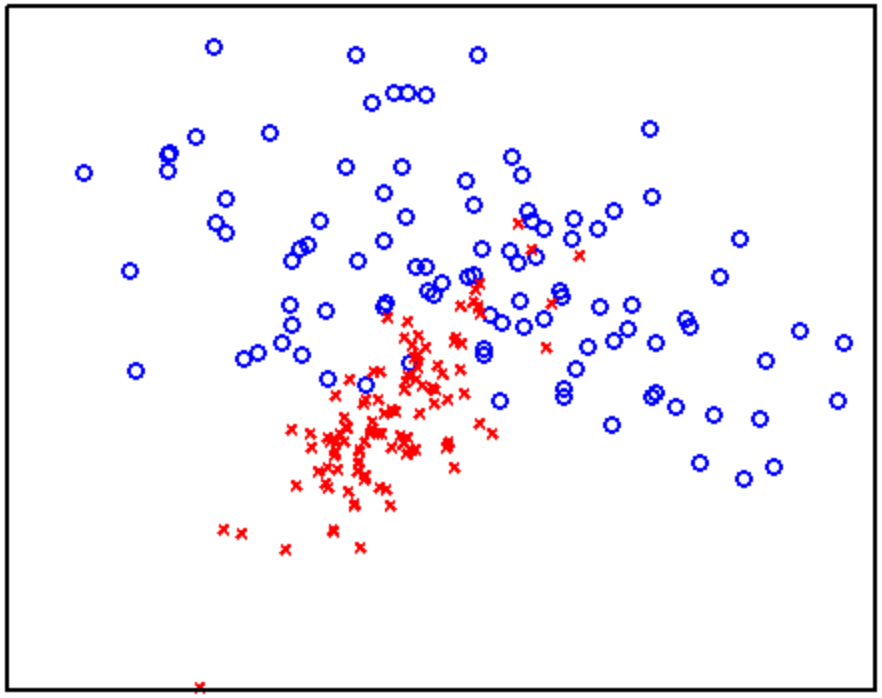
How do we define this more mathematically?

$I_k(\mathbf{x}) :=$ indices of the k training points closest to $\mathbf{x}$

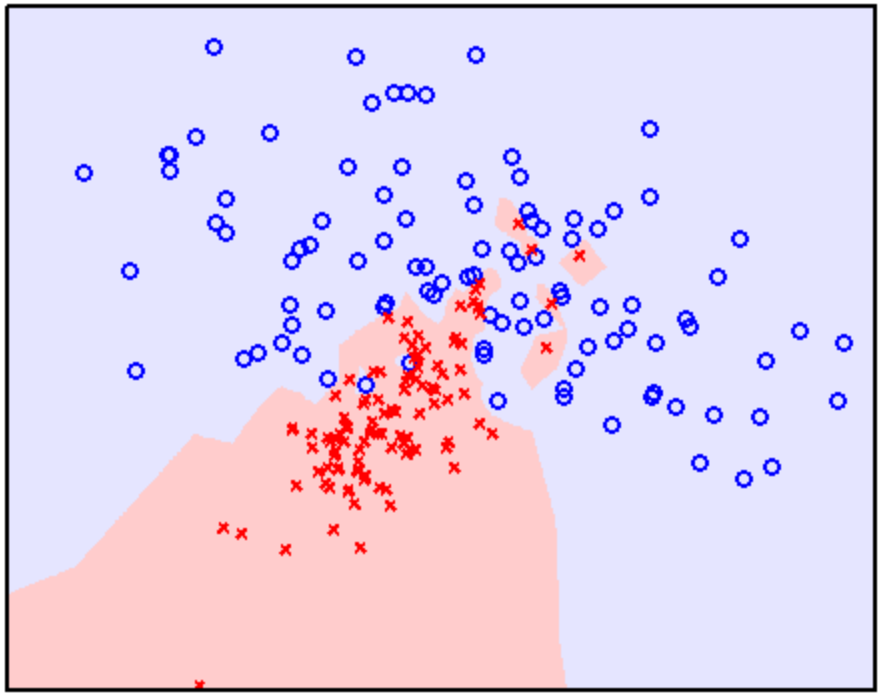
If $y_i = \pm 1$, then we can write the k-nearest neighbor classifier as

$$f_k(\mathbf{x}) := \text{sign}\left(\sum_{i \in I_k(\mathbf{x})} y_i\right)$$

Example

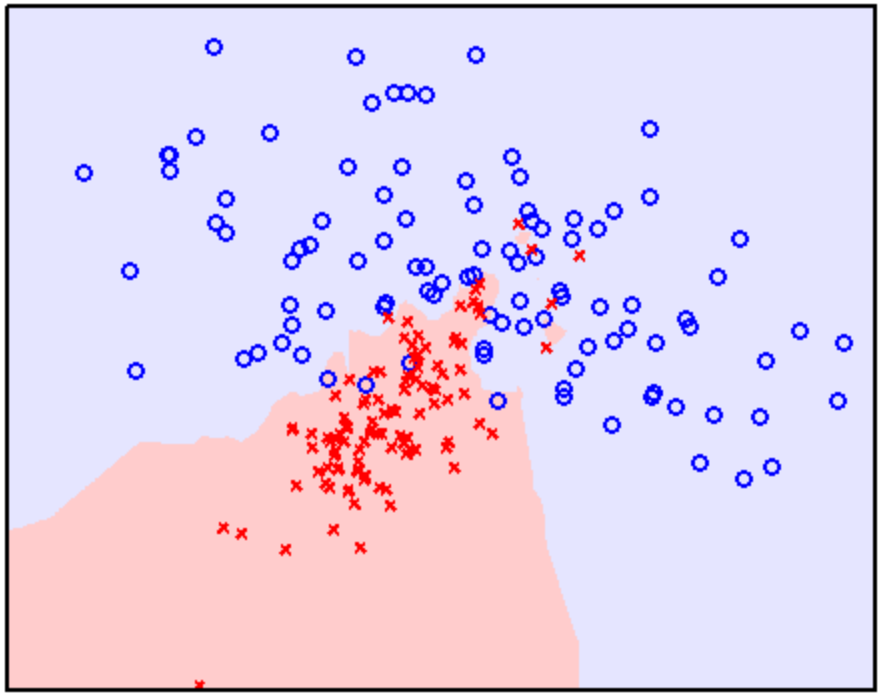


Example



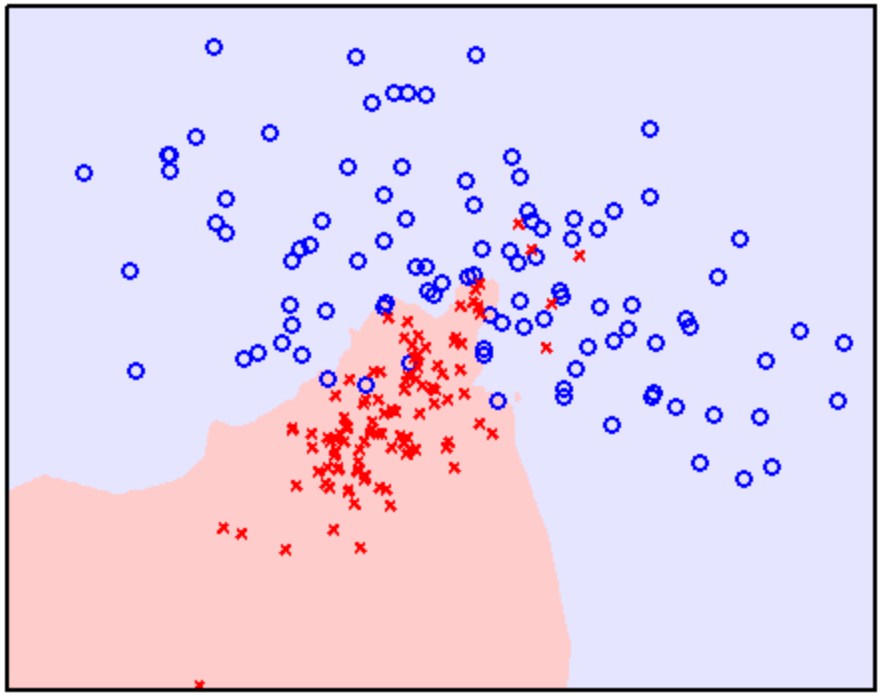
$k = 1$

Example



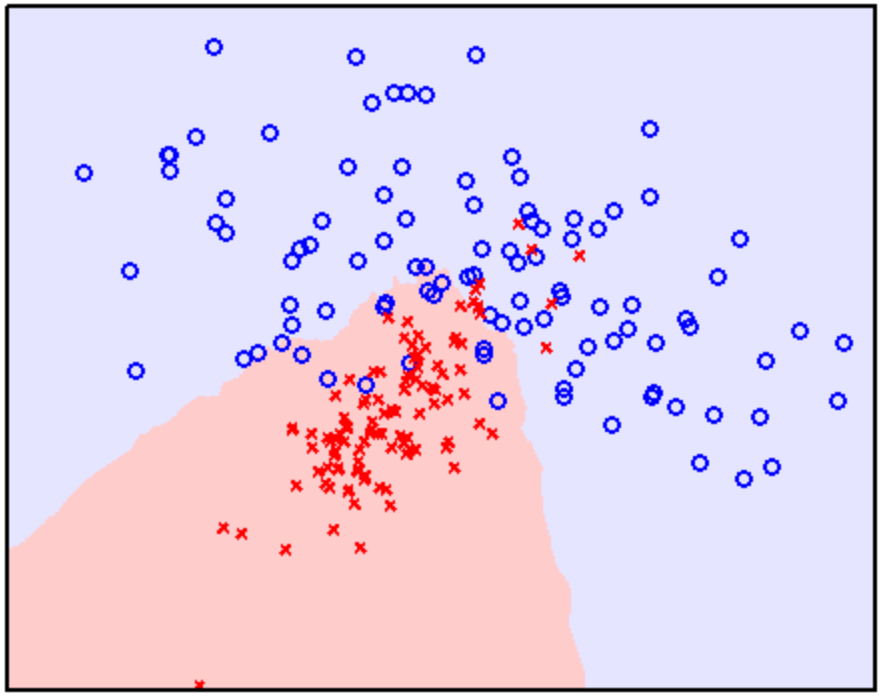
$k = 3$

Example



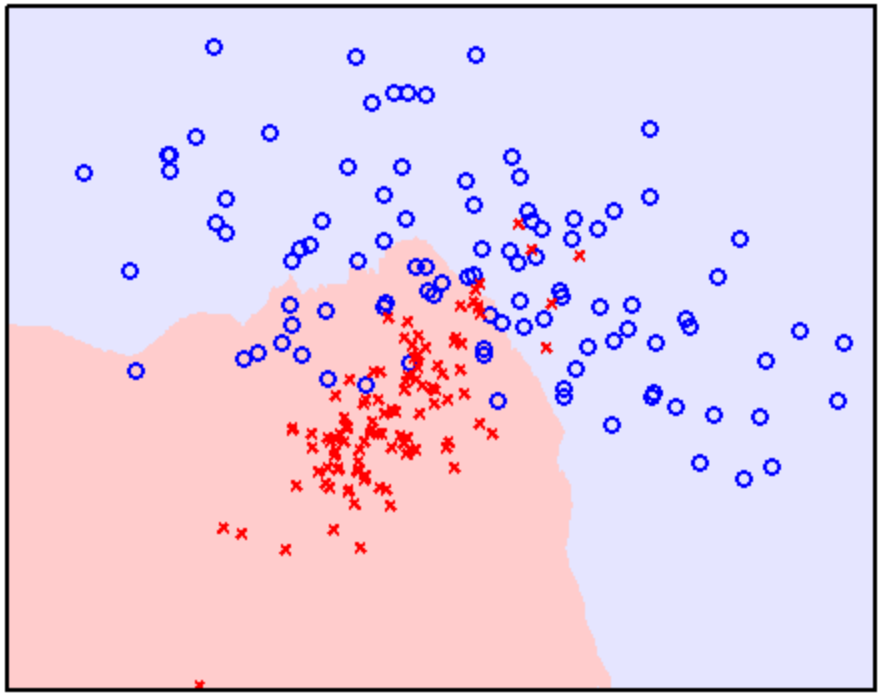
$k = 5$

Example



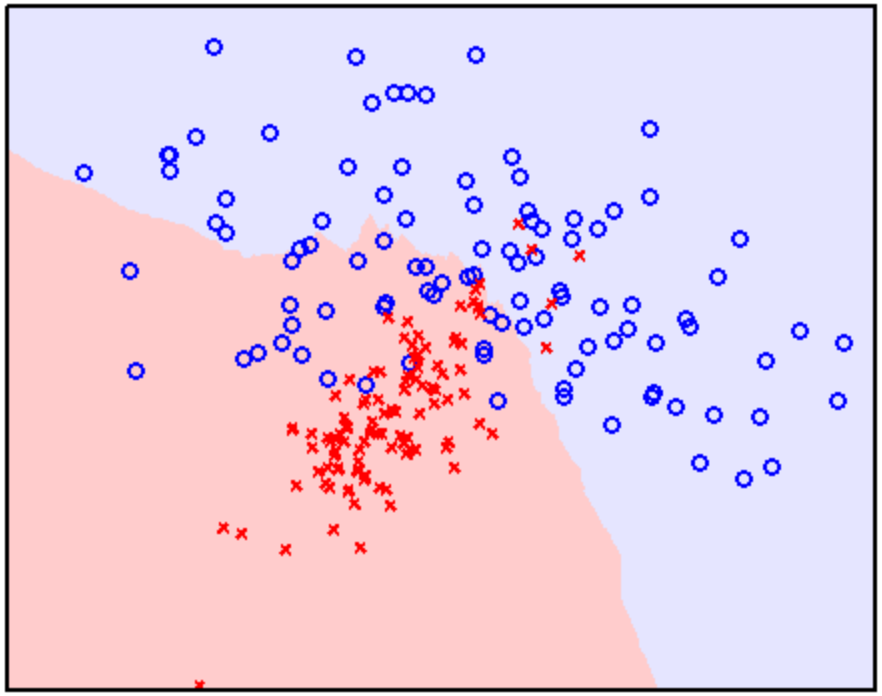
$k = 25$

Example



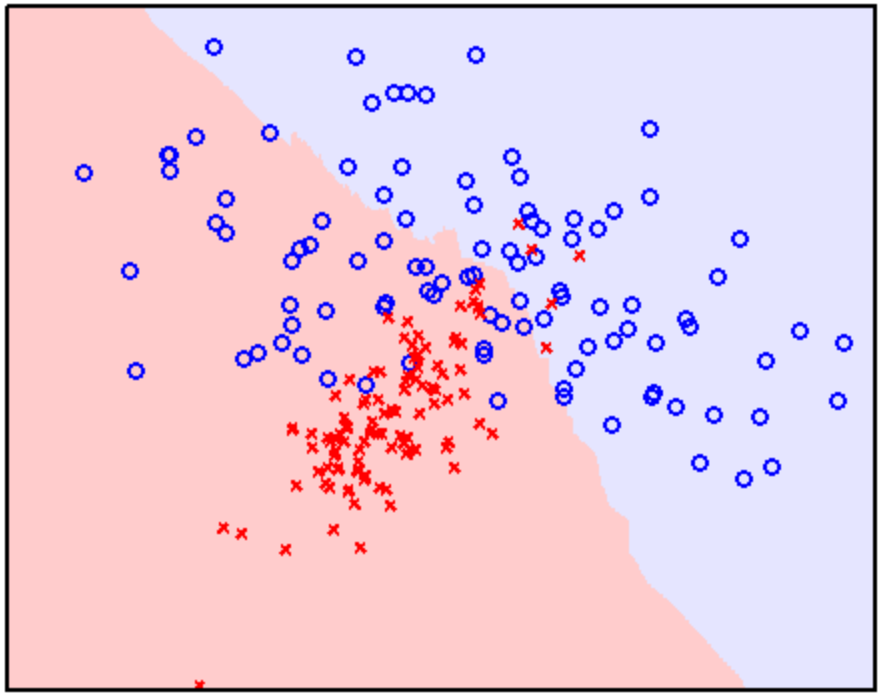
$k = 51$

Example



$k = 75$

Example



$k = 101$

Choosing the size of the neighborhood

Setting the parameter k is again a problem of ***model selection***

Setting k by trying to minimize the training error is a particularly bad idea

$$\widehat{R}_n(f_k) = \frac{1}{n}\sum_{i=1}^{n} 1_{\{f_k(\mathbf{x}_i) \neq y_i\}}$$

What is $\widehat{R}_n(f_1)$?

No matter what, we always have $\widehat{R}_n(f_1) = 0$

Not much practical guidance from the theory, so we typically must rely on estimates based on holdout sets or more sophisticated model selection techniques

Consistency

We say that a classification algorithm is consistent if when the size of the training set $n \to \infty$, we have that

$$\mathbb{E}\left[R(\widehat{f}_n)\right] \to R^*$$

where $\widehat{f}_n$ is a classifier learned from a training set of size and R^* is the Bayes risk

Theorem

Let $f_{k,n}$ denote the k-nearest neighbor rule with a training set of size n. If $n \to \infty$, $k \to \infty$, and $k/n \to 0$, then $f_{k,n}$ is consistent, i.e.,

$$\mathbb{E}\left[R(f_{k,n})\right] \to R^*$$

Summary

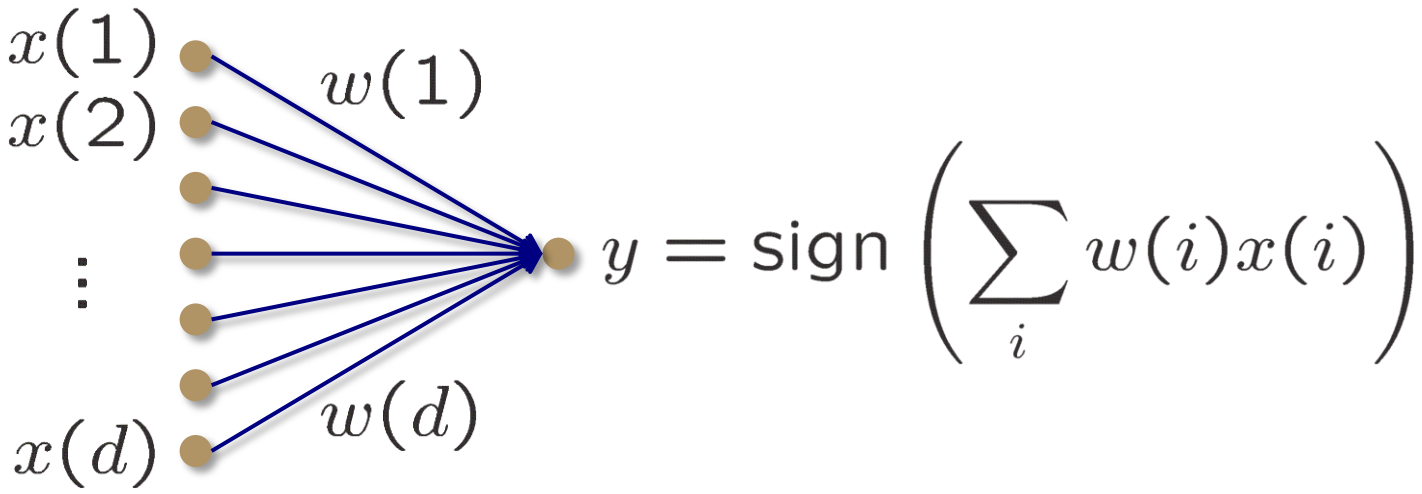
Given enough data, the k-nearest neighbor classifier will do just as well as pretty much any other method

Catch

- The amount of required data can be huge, especially if our feature space is high-dimensional
- The parameter k can matter a lot, so just like other methods, model selection will still be very important
- Finding the nearest neighbors out of a set of millions of examples is still pretty hard
  - there essentially no “learning” or “training” phase, but applying the classifier can be expensive
  - in contrast, for SVMs, training is expensive, but application is cheap

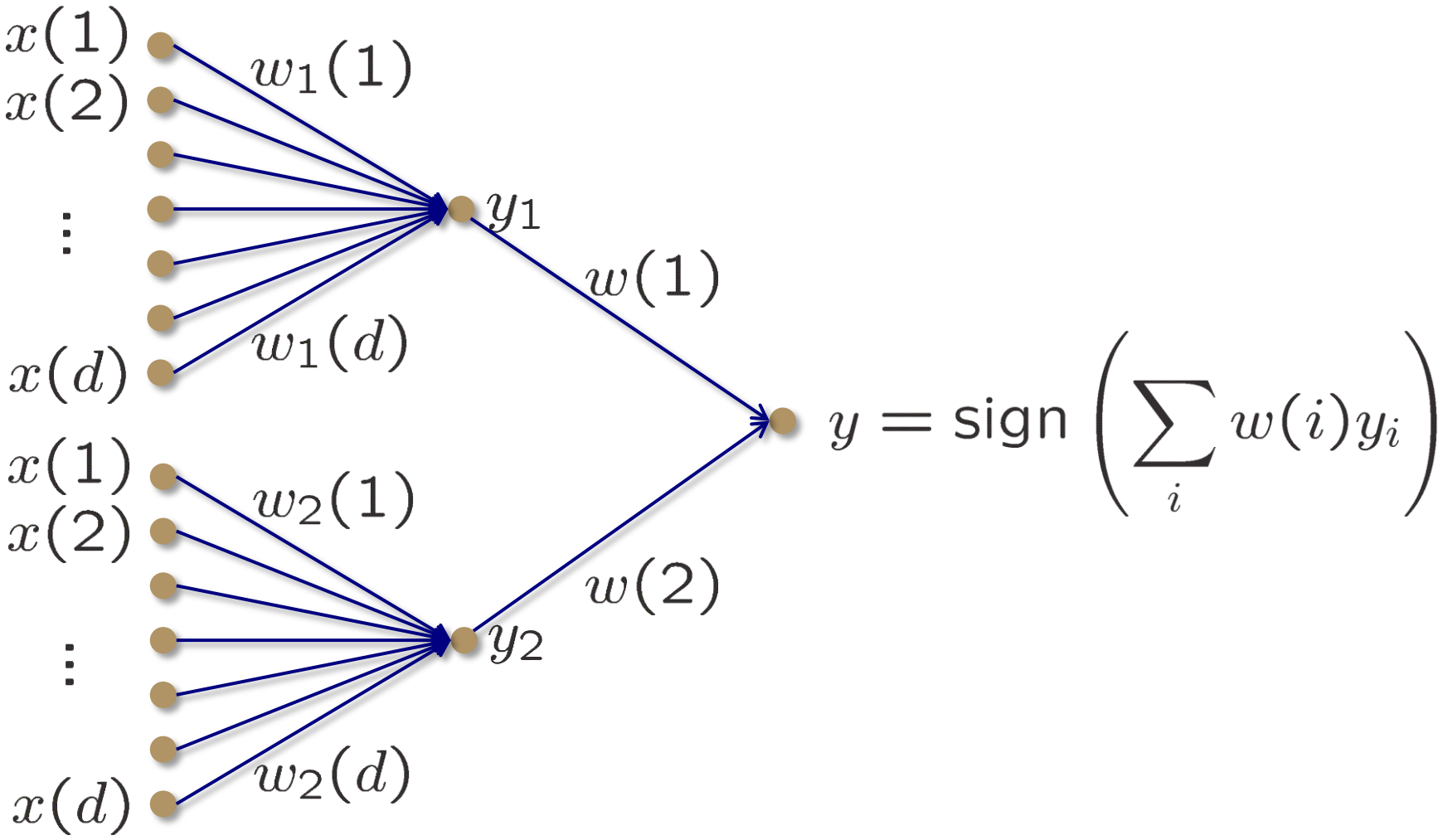
Feed-forward networks of classifiers

Another way to get a nonlinear classifier is to consider combining several linear classifiers together



Feed-forward networks of classifiers

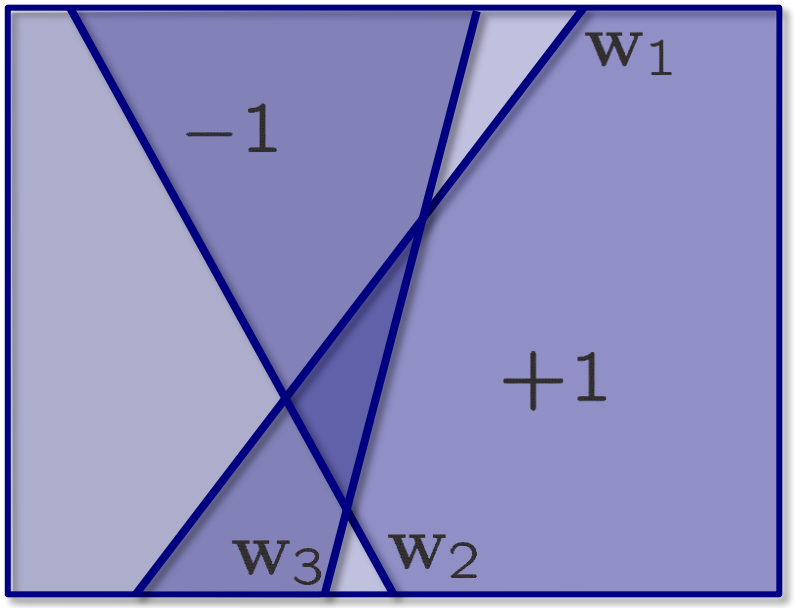
Another way to get a nonlinear classifier is to consider combining several linear classifiers together



What does this buy us?

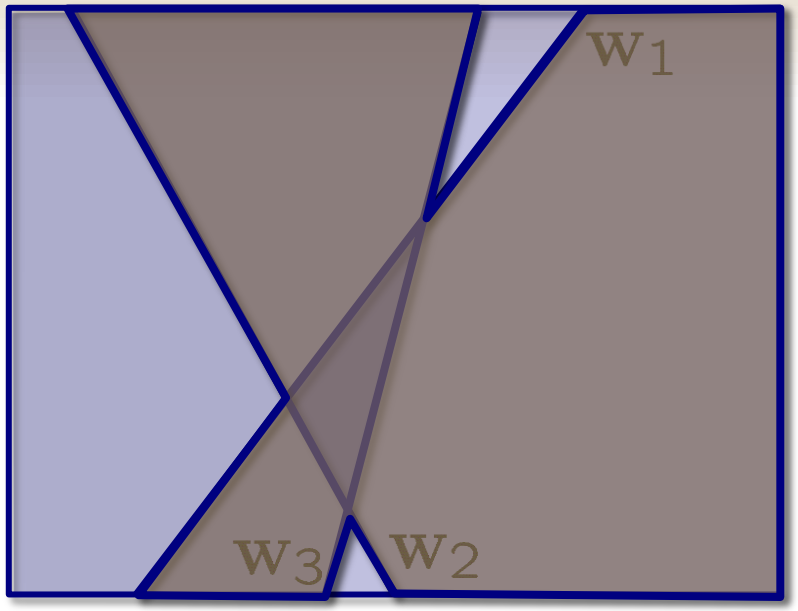
We showed previously that feeding the output of a linear classifier into another linear classifier doesn't introduce any new degrees of freedom

This is not true in the case of ***multiple*** classifiers



$$y_1 = \text{sign}(\mathbf{w}_1^T \mathbf{x})$$
$$y_2 = \text{sign}(\mathbf{w}_2^T \mathbf{x})$$
$$y_3 = \text{sign}(\mathbf{w}_3^T \mathbf{x})$$

Possible classifiers



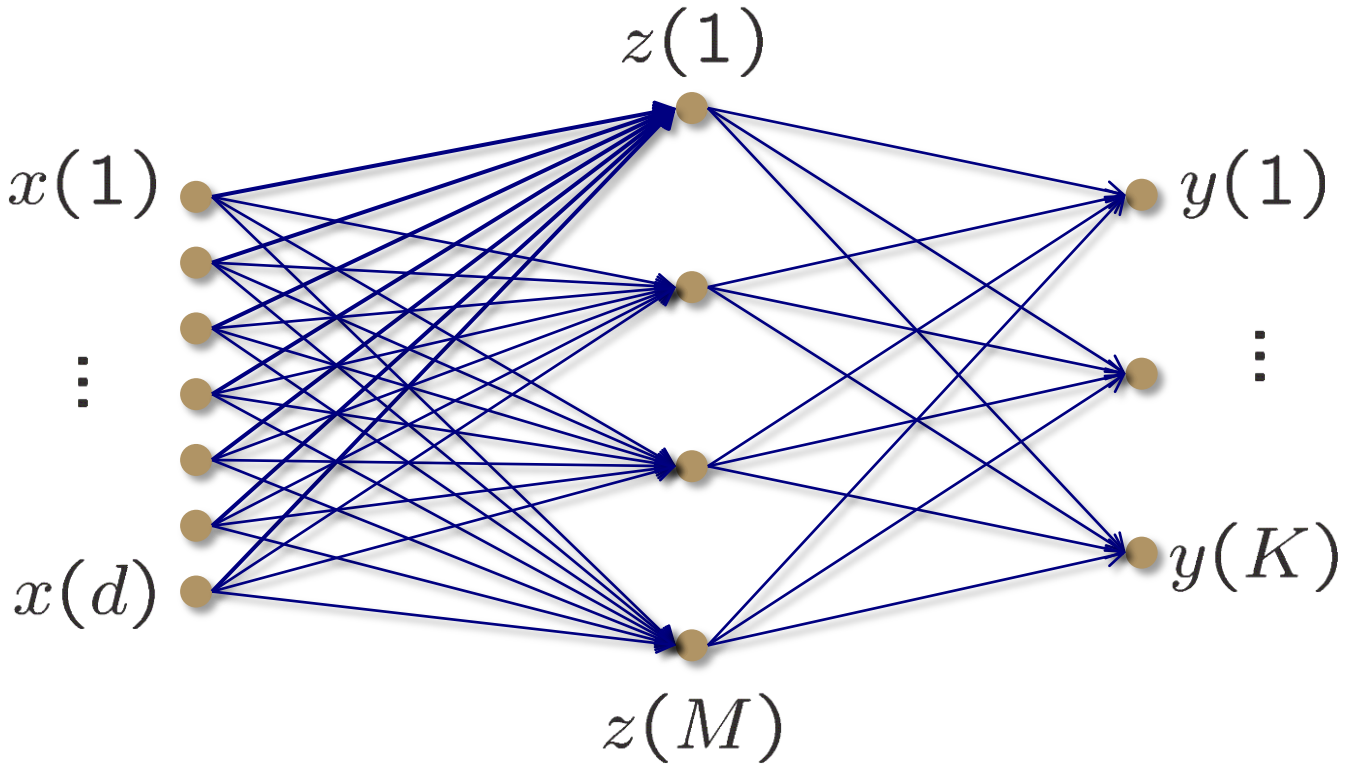
$$y_1 = \text{sign}(\mathbf{w}_1^T \mathbf{x})$$
$$y_2 = \text{sign}(\mathbf{w}_2^T \mathbf{x})$$
$$y_3 = \text{sign}(\mathbf{w}_3^T \mathbf{x})$$

$$y = \text{sign}(w(1)y_1 + w(2)y_2 + w(3)y_3)$$

Suppose that $w(1), w(2), w(3) > 0$
and $w(1) = w(2) = w(3)$

Neural networks

This is a particular example of a ***neural network***



input layer

hidden layer

output layer

Neural networks

Formally, a neural network is expressed mathematically via

$$z(m) = \sigma(\mathbf{w}_m^T \mathbf{x} + b_m), \quad m = 1, \ldots, M$$

$$y(k) = g(\boldsymbol{\beta}_k^T \mathbf{z} + c_k), \quad k = 1, \ldots, K$$

where σ, g are fixed functions and $\mathbf{w}_m, b_m, \boldsymbol{\beta}_k, c_k$ are parameters to be learned from the data

Example

The previous example fits this model with $K = 1$ and

$$\sigma(t) = g(t) = \text{sign}(t)$$

Typical neural networks

We will see next time that in general, learning the parameters for a neural network can be quite difficult

To make life easier, it is nice to choose σ, g to be differentiable

A common choice for σ is the sigmoid function

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

The choice of g depends somewhat on the application

Regression

Here, $K = 1$ since our output is a scalar, and we can take

$$g(t) = t$$

This leads to a regression model of

$$f(X) = \sum_{m=1}^{M} \beta(m)\sigma(\mathbf{w}_m^T\mathbf{x} + b_m) + c$$

Note: This is a linear model, but with nonlinear features

$$z_m = \sigma(\mathbf{w}_m^T\mathbf{x} + b_m)$$

Binary classification

We can again take $K = 1$

Take

$$g(t) = \frac{1 - e^{-t}}{1 + e^{-t}}$$

Our model is then given by

$$f(X) = g\left(\sum_{m=1}^{M} \beta(m)\sigma(\mathbf{w}_m^T\mathbf{x} + b_m) + c\right)$$

Again, this classifier is a linear model in the nonlinear features $z_m = \sigma(\mathbf{w}_m^T\mathbf{x} + b_m)$

Multiclass classification

Let K be the number of classes

If pattern $\mathbf{x}_i$ belongs to class k, then we define the desired output of our neural network to be

$$\mathbf{y}_i = [0 \cdots 0 1 0 \cdots 0]^T$$

k^{th} position

Take

$$g(t_k) = \frac{e^{t_k}}{\sum_{\ell=1}^{K} e^{t_k}}$$

where $t_k = \beta_k^T \mathbf{z} + c_k$, also called the ***softmax*** function

Multiclass classification

The idea is that $y(k) = g(t_k)$ models the posterior probability for class k

This results in a final classification given by

$$\widehat{y} = \arg\max_k \; y(k)$$

This roughly amounts to ***multiclass logistic regression*** on the nonlinear features $z_m = \sigma(\mathbf{w}_m^T \mathbf{x} + b_m)$

Remarks

- Like SVMs, neural networks fit a linear model in a nonlinear feature space
- Unlike SVMs, those nonlinear features are ***learned***
- Unfortunately, training involves ***nonconvex*** optimization
- Originally conceived as models for the brain
  - nodes are neurons
  - edges are synapses
  - if σ is a step function, this represents a neuron "firing" when the total incoming signal exceeds a certain threshold
- Neural networks with one hidden layer are ***universal*** approximators, provided that M can be arbitrarily large

Lecture 15: Neural Networks

Nearest neighbor classifiers

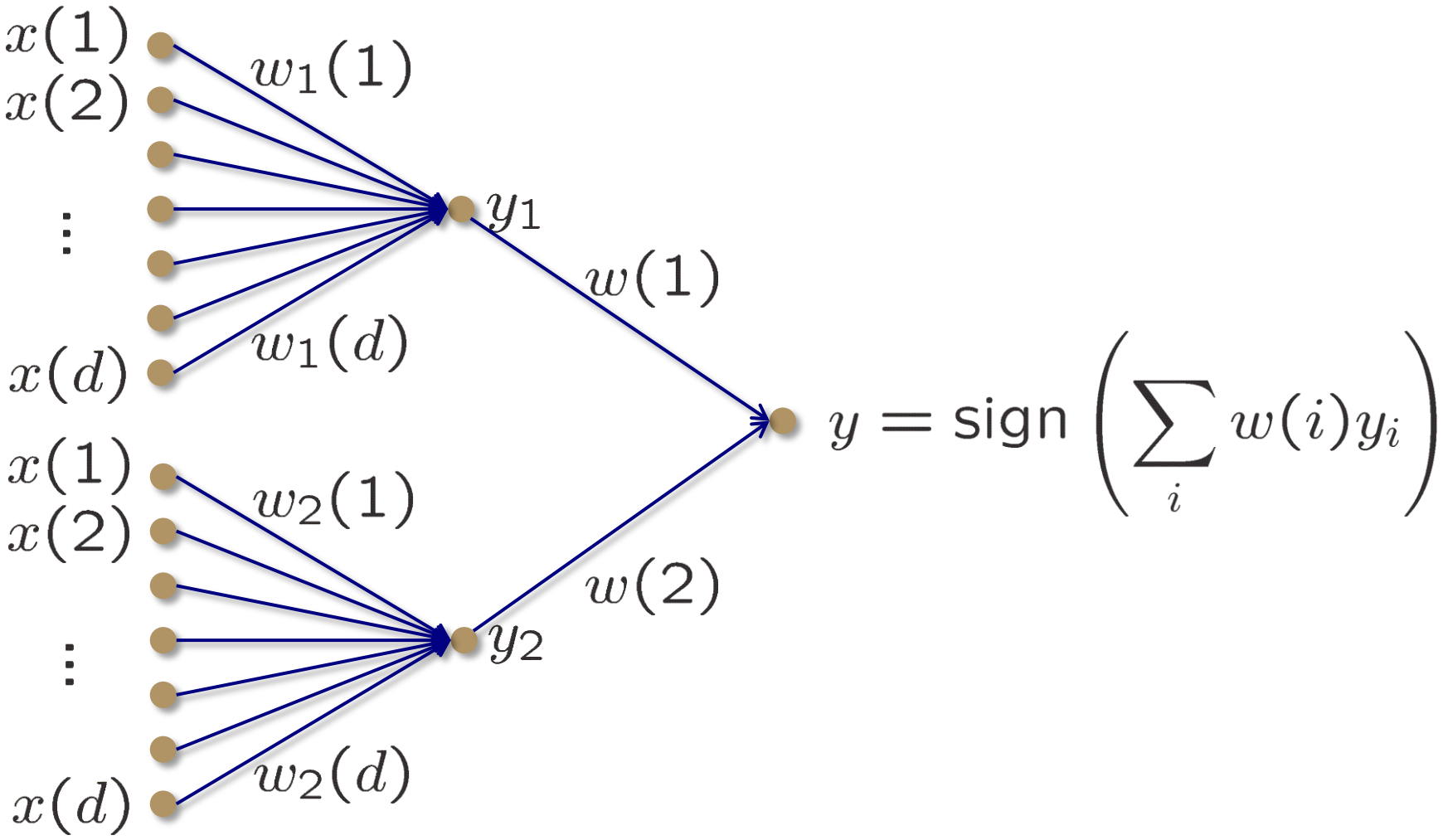
Given enough data, the k-nearest neighbor classifier will do just as well as pretty much any other method

Catch

- The amount of required data can be huge, especially if our feature space is high-dimensional
- The parameter k can matter a lot, so just like other methods, model selection will still be very important
- Finding the nearest neighbors out of a set of millions of examples is still pretty hard
  - there essentially no “learning” or “training” phase, but applying the classifier can be expensive
  - in contrast, for SVMs, training is expensive, but application is cheap

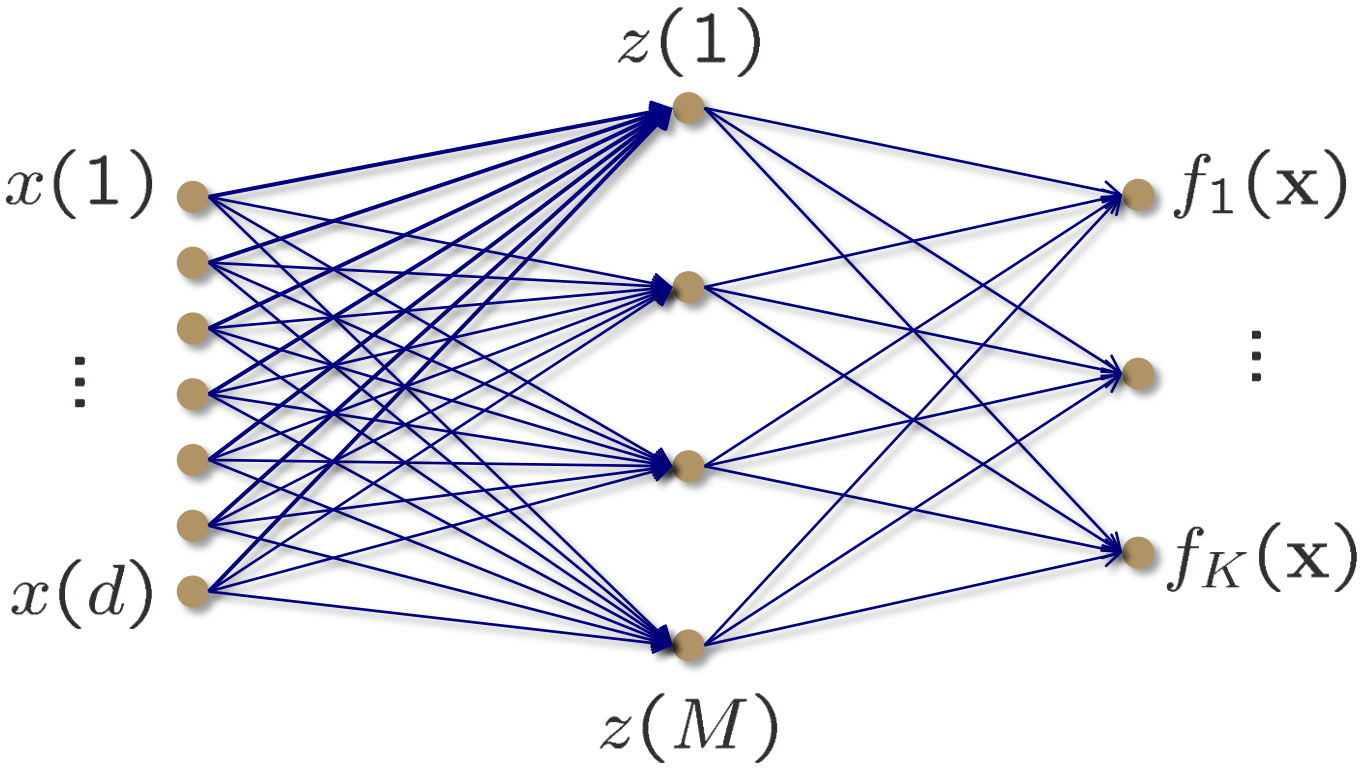
Feed-forward networks of classifiers

Another way to get a nonlinear classifier is to consider combining several linear classifiers together



Neural networks

This is a particular example of a ***neural network***



input layer

hidden layer

output layer

Neural networks

Formally, a neural network is expressed mathematically via

$$z(m) = \sigma(\mathbf{w}_m^T \mathbf{x} + b_m), \quad m = 1, \ldots, M$$

$$f_k(\mathbf{x}) = g(\boldsymbol{\beta}_k^T \mathbf{z} + c_k), \quad k = 1, \ldots, K$$

where σ, g are fixed functions and $\mathbf{w}_m, b_m, \boldsymbol{\beta}_k, c_k$ are parameters to be learned from the data

Example

The previous example fits this model with $K = 1$ and

$$\sigma(t) = g(t) = \text{sign}(t)$$

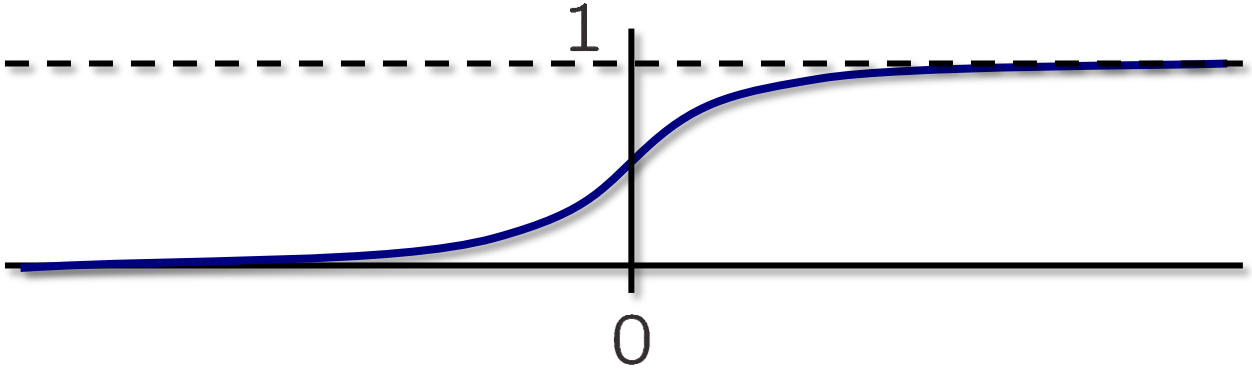
Typical neural networks

In general, learning the parameters for a neural network can be quite difficult

To make life easier, it is nice to choose σ, g to be differentiable

A common choice for σ is the logistic/sigmoid function

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



The choice of g depends somewhat on the application

Typical neural networks

The choice of g depends somewhat on the application

Regression ($K = 1, y \in \mathbb{R}$)

$$g(t) = t$$

Binary classification ($K = 1, y \in \{-1, +1\}$)

$$g(t) = \frac{1 - e^{-t}}{1 + e^{-t}}$$

Multiclass classification ($\mathbf{y} = [0 \cdots 010 \cdots 0]^T$)

$$g(t_k) = \frac{e^{t_k}}{\sum_{j=1}^{K} e^{t_j}} \qquad t_k = \boldsymbol{\beta}_k^T \mathbf{z} + c_k$$

Remarks

- Like SVMs, neural networks fit a linear model in a nonlinear feature space
- Unlike SVMs, those nonlinear features are ***learned***
- Unfortunately, training involves ***nonconvex*** optimization
- Originally conceived as models for the brain
  - nodes are neurons
  - edges are synapses
  - if σ is a step function, this represents a neuron "firing" when the total incoming signal exceeds a certain threshold
  - note that our choices for σ, g are not exactly step functions, but smooth approximations – this also means our output is continuous (must be quantized in the case of classification)
- Neural networks with one hidden layer are ***universal*** approximators, provided that M can be arbitrarily large

Training neural networks

Given training data $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in \mathbb{R}^K$, we would like to estimate the parameters $\mathbf{w}_m, b_m, \boldsymbol{\beta}_k, c_k$

For simplicity, let $\boldsymbol{\theta}$ denote the complete set of parameters

$$\{\mathbf{w}_m, b_m\}_{m=1}^{M} \longrightarrow M(d+1) \text{ parameters}$$

$$\{\boldsymbol{\beta}_k, c_k\}_{k=1}^{K} \longrightarrow K(M+1) \text{ parameters}$$

Objective functions

We would like to choose θ to do a good job of predicting $\mathbf{y}$

We can quantify this by choosing an ***objective function*** which we will seek to minimize by picking θ appropriately

Regression or ***binary classification***

$$F(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2$$

or for vector-valued problems

$$F(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{i=1}^{n} (y_i(k) - f_k(\mathbf{x}_i))^2$$

Objective functions

We would like to choose θ to do a good job of predicting $\mathbf{y}$

We can quantify this by choosing an ***objective function*** which we will seek to minimize by picking θ appropriately

Classification

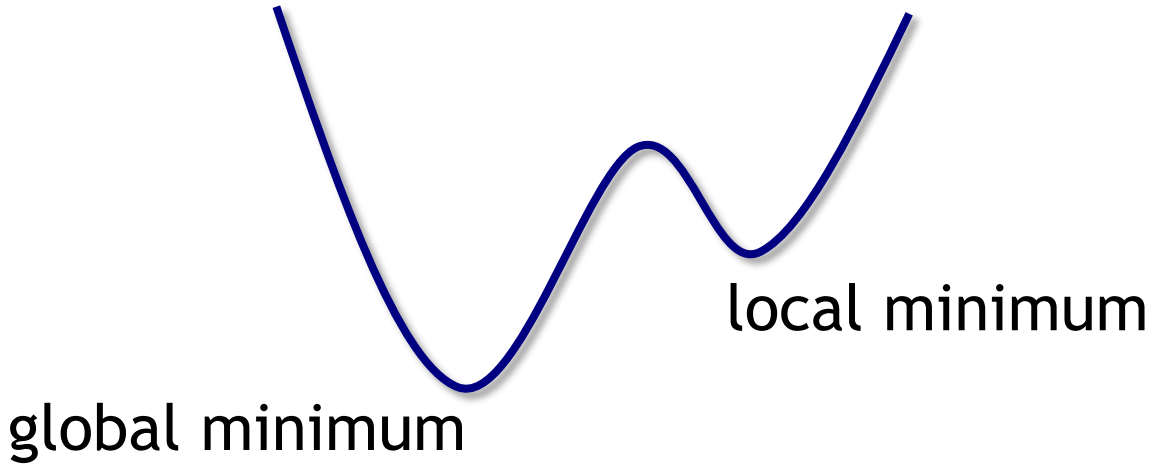
In the case where $\mathbf{y}_i = [0 \cdots 010 \cdots 0]^T$, a common choice is

$$F(\boldsymbol{\theta}) = -\sum_{k=1}^{K} \sum_{i=1}^{n} y_i(k) \log f_k(\mathbf{x}_i)$$

If we interpret the outputs of our neural network $f_k(\mathbf{x}_i)$ as class conditional probabilities, this is computing the negative log-likelihood

Nonconvex optimization

Because of the complex interactions between the parameters in θ, these objective functions do not lead to convex optimization problems



A ***local minimum*** is not necessarily a ***global minimum***

The best we can hope for is to try to find a local minimum, and hope that this gives us good performance in practice

Gradient descent

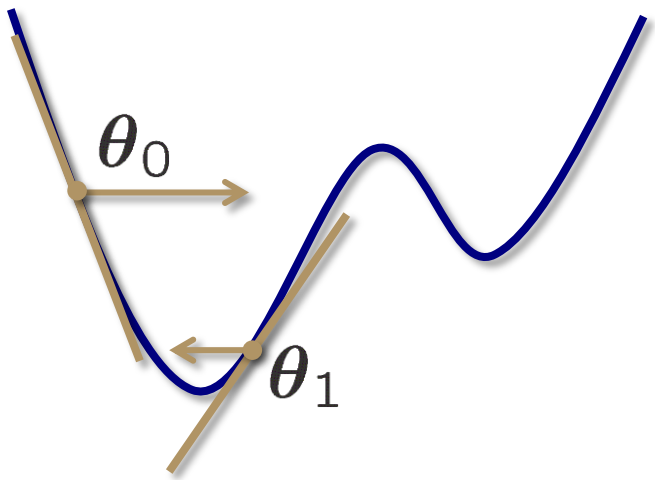
A simple way to try to find the minimum of our objective function is to iteratively *"roll downhill"*

From θ_0, take a step in the direction of the negative gradient

$$\theta_1 = \theta_0 - \alpha_0 \nabla F(\theta)|_{\theta=\theta_0} \qquad \alpha_0 : \text{"step size"}$$

$$\theta_2 = \theta_1 - \alpha_1 \nabla F(\theta)|_{\theta=\theta_1}$$

$$\vdots$$



Squared error loss

Today, we will focus on how to train to minimize the squared error loss

$$F(\boldsymbol{\theta}) = \sum_{k=1}^{K} \sum_{i=1}^{n} (y_i(k) - f_k(\mathbf{x}_i))^2 = \sum_{i=1}^{n} F_i$$

$$F_i = \sum_{k=1}^{K} (y_i(k) - f_k(\mathbf{x}_i))^2$$

For convenience, we will also use our usual trick and let

$$\mathbf{x}_i = [1, x_i(1), \ldots, x_i(d)]^T$$

$$\mathbf{w}_m = [b_m, w_m(1), \ldots, w_m(d)]^T$$

Notation

In this case, we can set $z(m) = \sigma(\mathbf{w}_m^T \mathbf{x})$

It will also be useful to define $z_i(m) = \sigma(\mathbf{w}_m^T \mathbf{x}_i)$ and consider the vectors

$$\mathbf{z}_i = [1, z_i(1), \ldots, z_i(M)]^T$$

$$\boldsymbol{\beta}_k = [c_k, \beta_k(1), \ldots, \beta_k(M)]^T$$

All together, this lets us write

$$f_k(\mathbf{x}_i) = g_k(\boldsymbol{\beta}_k^T \mathbf{z}_i)$$

$$= g_k(c_k + \sum_{m=1}^{M} \beta_k(m) \sigma(\mathbf{w}_m^T \mathbf{x}_i))$$

Computing the gradient

$$\frac{\partial F_i}{\partial \beta_k(m)} = \frac{\partial}{\partial \beta_k(m)} \sum_{j=1}^{K} \left(y_i(j) - g_j(\boldsymbol{\beta}_j^T \mathbf{z}_i)\right)^2$$

$$= -2(y_i(k) - g_k(\boldsymbol{\beta}_k^T \mathbf{z}_i)) g_k'(\boldsymbol{\beta}_k^T \mathbf{z}_i) z_i(m)$$

$$=: \delta_i(k) z_i(m)$$

$$\frac{\partial F_i}{\partial w_m(\ell)} = -2 \sum_{k=1}^{K} \Big((y_i(k) - g_k(\boldsymbol{\beta}_k^T \mathbf{z}_i)) g_k'(\boldsymbol{\beta}_k^T \mathbf{z}_i) \times \beta_k(m) \sigma'(\mathbf{w}_m^T \mathbf{x}_i) x_i(\ell) \Big)$$

$$=: s_i(m) x_i(\ell)$$

Gradient descent update

The update at iteration $r+1$ is

$$\beta_k^{(r+1)}(m) = \beta_k^{(r)}(m) - \alpha_r \sum_{i=1}^{n} \frac{\partial F_i}{\partial \beta_k(m)}(\boldsymbol{\theta}^{(r)})$$

$$w_m^{(r+1)}(\ell) = w_m^{(r)}(\ell) - \alpha_r \sum_{i=1}^{n} \frac{\partial F_i}{\partial w_m(\ell)}(\boldsymbol{\theta}^{(r)})$$

The weights in our formula for the partial derivatives depend on the previous parameter estimate $\boldsymbol{\theta}^{(r)}$

Of course, we could just go ahead and calculate these weights in a naïve way, add them all up, and compute our updates, but this will be ***slow***

Saving on computation?

Notice that

$$\delta_i(k) = -2(y_i(k) - f_k(\mathbf{x}_i))g_k'(\boldsymbol{\beta}_k^T \mathbf{z}_i)$$

$$s_i(m) = -2 \sum_{k=1}^{K} (y_i(k) - f_k(\mathbf{x}_i))g_k'(\boldsymbol{\beta}_k^T \mathbf{z}_i)\beta_k(m)\sigma'(\mathbf{w}_m^T \mathbf{x}_i)$$

$$= \sigma'(\mathbf{w}_m^T \mathbf{x}_i) \sum_{k=1}^{K} \delta_i(k)\beta_k(m)$$

This suggests an efficient two-pass algorithm

“Backpropagation”

Forward pass

Using current weights $\theta^{(r)}$, compute

$$f_k^{(r)}(\mathbf{x}_i), \quad k = 1, \ldots, K, i = 1, \ldots, n$$

Backward pass

Use $f_k^{(r)}(\mathbf{x}_i)$ to calculate the weights $\delta_i^{(r)}(k)$

$$\delta_i^{(r)}(k) = -2(y_i(k) - f_k^{(r)}(\mathbf{x}_i))g_k'((\boldsymbol{\beta}_k^{(r)})^T \mathbf{z}_i^{(r)})$$

Next, “back-propagate” these values to the previous layer

$$s_i^{(r)}(m) = \sigma'((\mathbf{w}_m^{(r)})^T \mathbf{x}_i) \sum_{k=1}^{K} \delta_i^{(r)}(k) \beta_k^{(r)}(m)$$

Compute $\theta^{(r+1)}$ and iterate

Convergence speed

In backpropagation, each hidden unit passes and receives information to and from only those units to which it is connected

Much faster update than a naïve approach

Lends itself naturally to ***parallel*** implementations

Gradient descent can still be slow to converge

Conjugate gradient and other methods can converge faster

Initialization

Initial conditions make a big difference

To avoid “saturation”, data should be pre-processed to have zero mean and unit variance

Since our objective function will have many local minima, it is generally a good idea to try several random starting values.

Assuming the above pre-processing, a reasonable initialization would be to choose parameter values

$$w_m(\ell), \beta_k(m) \in [-0.7, 0.7]$$

Online learning

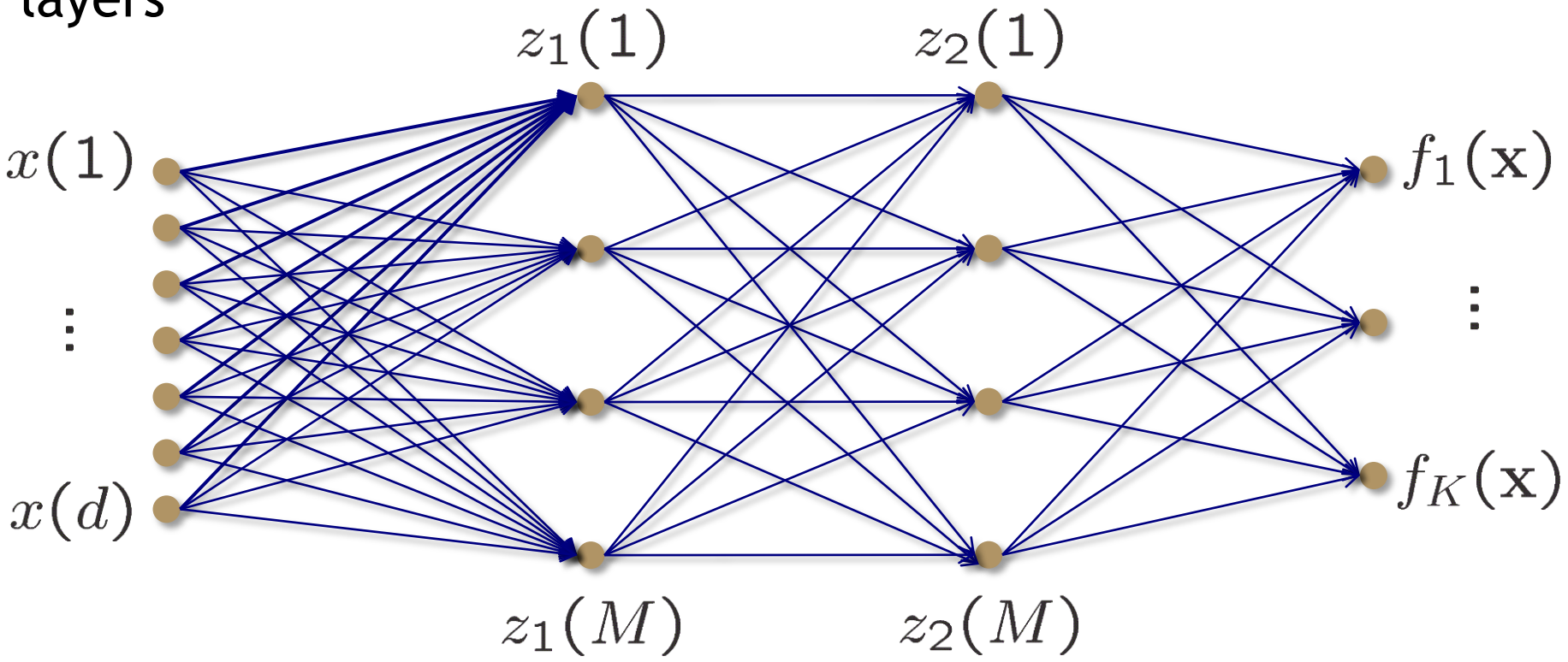
Neural networks are particularly possible in reinforcement learning, where we

- begin with some model
- get to try it out on a new piece of data
- receive some kind of reward (or penalty) based on how well our prediction matches the new piece of data
- want to update our model

Backpropagation and other gradient-based methods are easy to implement in this “online” setting, since we can update our parameters using only the most recent data if we choose to do so

Multi-layer neural networks

This framework, along with the backpropagation algorithm, can easily be extended to networks with multiple hidden layers



Has had a resurgence in recent years under the name ***"deep learning"***

Regularization

To avoid overfitting the data, regularization is crucial

Choose $\boldsymbol{\theta}$ to minimize

$$F(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|^2$$

This encourages small parameter values $\theta(i)$

Note that when the $\theta(i)$ are small, the effect of the nonlinear functions σ, g go away and are approximately linear

If we constrain all $\theta(i)$ to be small, the entire network approximately reduces to a simple linear classifier

Encouraging small parameter values $\theta(i)$ reduces the effective number of degrees of freedom

Other strategies for controlling overfitting

- Choosing the number of hidden layers and number of "neurons" in each layer is more art than science
- Various "constrained" models have been proposed that try to limit the number of degrees of freedom in various ways
  - e.g., require some of the weights to have the same value (convolutional neural networks)
- In general, it seems to be better to have too many parameters rather than too few, and rely on regularization to avoid overfitting
- Another common strategy is to intentionally stop the backpropagation before convergence – early stopping seems to act as a form of regularization and helps prevent overfitting

Topics to be covered on midterm

- Basic probability
  - binomial distribution
  - union bound
  - Hoeffding's inequality
- The Bayes classifier
  - how to calculate the classifier
  - how to calculate the risk
- Linear classifiers
  - difference between LDA, logistic regression, Naïve Bayes
  - PLA, maximum margin hyperplanes

Topics to be covered on midterm

- VC dimension
  - definition
  - how to calculate the VC dimension for a simple example
  - how to "guess" the VC dimension for complicated examples
  - implications (VC generalization bound)
- Bias-variance decomposition
  - how to calculate the "average hypothesis"
  - how to calculate the bias, variance, and total risk
- Regularization
  - why?
  - understand practical implications of regularization in both regression and classification

Topics to be covered on midterm

- Kernels
  - definition of a kernel
  - how to interpret a kernel as an inner product
  - how to use kernels to do nonlinear regression/classification
- Kernel ridge regression
  - how to interpret the resulting function
- SVMs
  - understand the derivation
  - equivalence and difference between primal/dual formulations
  - interpretation of support vectors and resulting classifier
- Nearest neighbors
  - understand what it is

Lecture 16: Model Selection and Validation

Model selection

In statistical learning, a ***model*** is a mathematical representation of a function such as a

- classifier
- regression function
- density
- …

In many cases, we have one (or more) “free parameters” that are not automatically determined by the learning algorithm

Often, the value chosen for these free parameters has a significant impact on the algorithm’s output

The problem of selecting values for these free parameters is called ***model selection***

Examples

Method	Parameter
• k-nearest neighbors	number of neighbors k
• SVM	margin violation cost C kernel parameters (e.g., σ)
• polynomial regression	polynomial degree d
• ridge regression	regularization parameter λ
• neural networks	regularization parameter λ stopping criteria

Model selection dilemma

We need to select appropriate values for the free parameters

All we have is the training data

We must use the training data to select the parameters

However, these free parameters usually control the balance between ***underfitting*** and ***overfitting***

They were left "free" precisely because we don't want to let the training data influence their selection, as this almost always leads to overfitting

- e.g., if we let the training data determine k in k-nearest neighbors, we will always choose $k = 1$

Big picture

For much of this class, we have focused on trying to understand learning via a decomposition of the form

$$R(h) = \widehat{R}_n(h) + \underbrace{\text{overfit penalty}}$$

VC dimension

regularization

Validation takes another approach

$$\underbrace{R(h)} = \widehat{R}_n(h) + \text{overfit penalty}$$

directly estimate this quantity

Validation

Suppose that in addition to our training data, we also have a validation set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_k, y_k)$

Use the validation set to form an estimate $\widehat{R}_{\text{val}}(h)$

$$\widehat{R}_{\text{val}}(h) := \frac{1}{k} \sum_{i=1}^{k} e(h(\mathbf{x}_i), y_i)$$

Examples

- Classification: $\widehat{R}_{\text{val}}(h) = \frac{1}{k} \sum_i 1_{\{h(\mathbf{x}_i) \neq y_i\}}$
- Regression: $\widehat{R}_{\text{val}}(h) = \frac{1}{k} \sum_i (h(\mathbf{x}_i) - y_i)^2$

  $\widehat{R}_{\text{val}}(h) = \frac{1}{k} \sum_i |h(\mathbf{x}_i) - y_i|$

  ⋮

Accuracy of validation

What can we say about the accuracy of $\widehat{R}_{\text{val}}(h)$?

In the case of classification, $e(h(\mathbf{x}_i), y_i) = 1_{\{h(\mathbf{x}_i) \neq y_i\}}$, which is just a Bernoulli random variable

Hoeffding: $\mathbb{P}\left[\left|\widehat{R}_{\text{val}}(h) - R(h)\right| > \epsilon\right] \leq 2e^{-2\epsilon^2 k}$

More generally, we always have

$$\mathbb{E}\left[\widehat{R}_{\text{val}}(h)\right] = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}\left[e(h(\mathbf{x}_i), y_i)\right] = R(h)$$

$$\text{var}\left[\widehat{R}_{\text{val}}(h)\right] = \frac{1}{k^2}\sum_{i=1}^{k} \text{var}\left[e(h(\mathbf{x}_i), y_i)\right] = \frac{\sigma^2}{k}$$

Accuracy of validation

In either case, this shows us that

$$\widehat{R}_{\text{val}}(h) = R(h) \pm O\left(\frac{1}{\sqrt{k}}\right)$$

Thus, we can get as accurate an estimate of $R(h)$ using a validation set as long as k is large enough

Remember, h is ultimately something we learned from training data

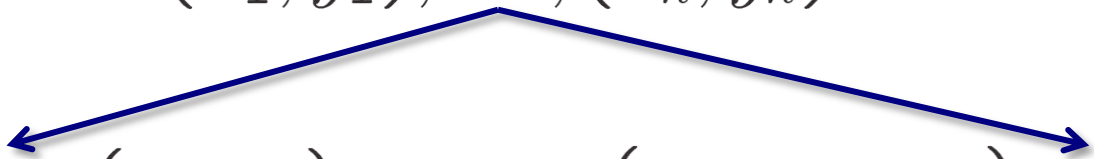
Where is this validation set coming from?


THERE IS NO TRAINING SET AND VALIDATION SET
THERE IS ONLY TRAINING SET

Validation vs training

We are given a data set

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$$



$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_k, y_k)$ — validation (holdout) set

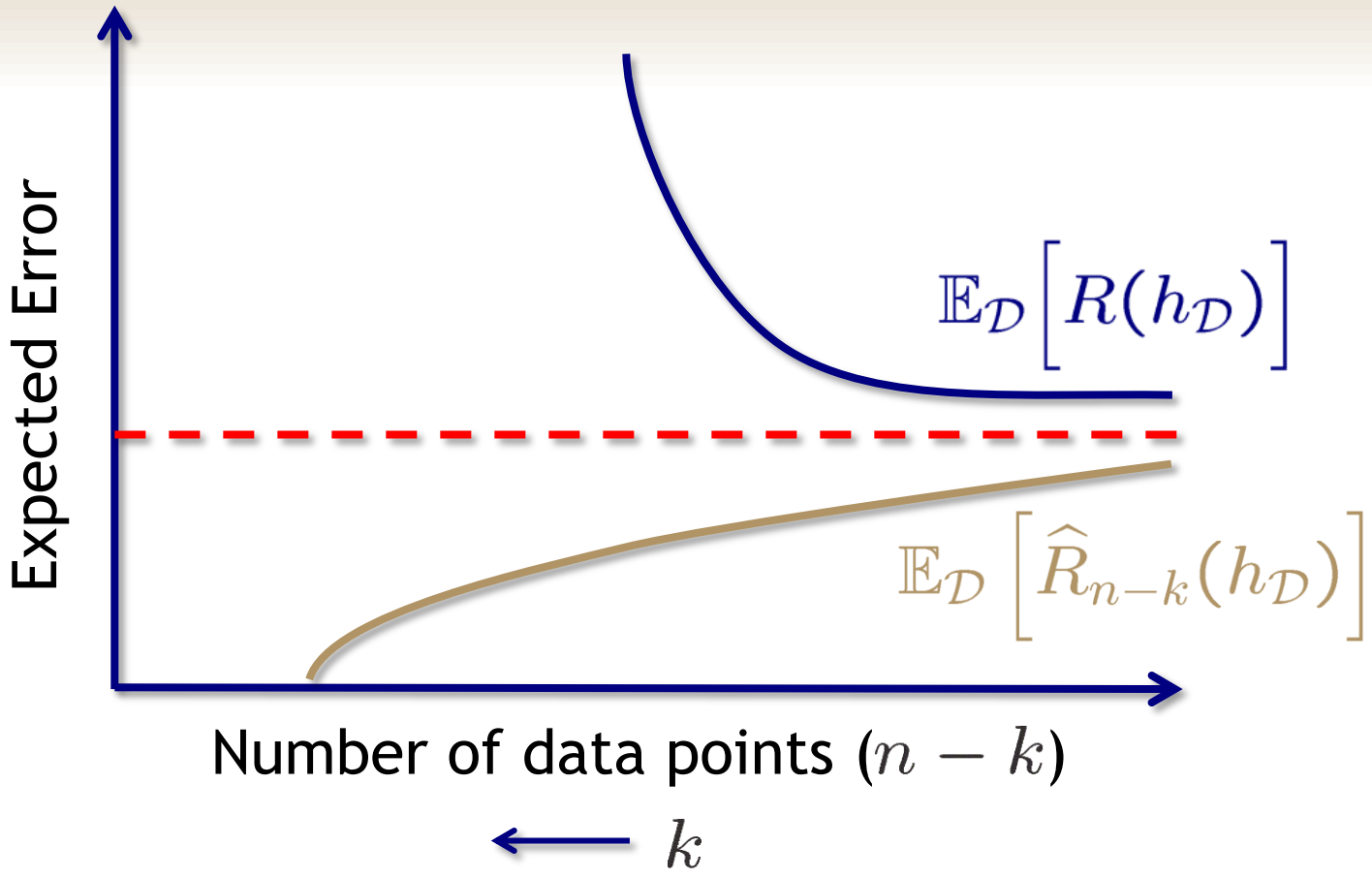
$(\mathbf{x}_{k+1}, y_{k+1}), \ldots, (\mathbf{x}_n, y_n)$ — training set

Validation error is $O(1/\sqrt{k})$:

Small k ➡ bad estimate

Large k ➡ accurate estimate, but of what?

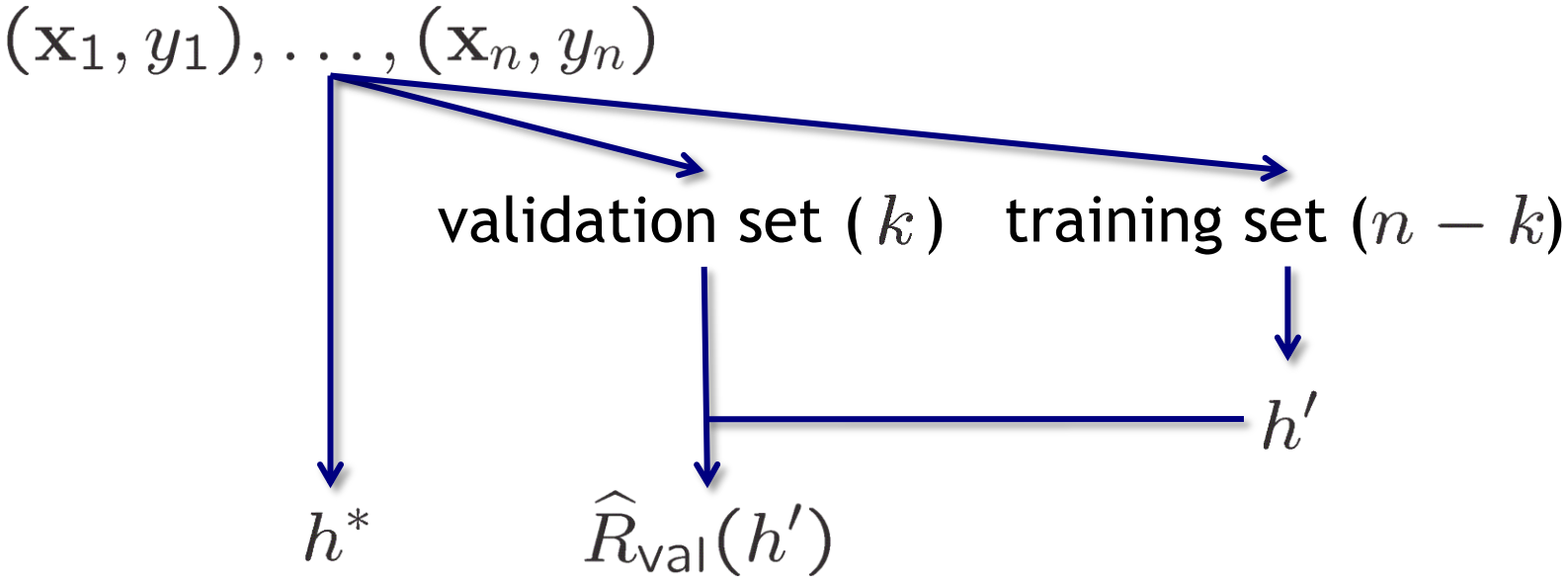
Learning curve



Large k lets us say: We are very confident that we have selected a terrible h

Can we have our cake and eat it too?

After we've used our validation set to estimate the error, re-train on the whole data set



Small k ➡ bad estimate of $R(h')$, but $R(h') \approx R(h)$

Large k ➡ good estimate of $R(h')$, but $R(h') \gg R(h)$

Rule of thumb: Set $k = n/5$

Validation vs testing

We call this “validation”, but how is it any different than simply “testing”?

Typically, $\widehat{R}_{\text{val}}$ is used to make learning choices

If an estimate of $R(h)$ affects learning, i.e., it impacts which h we choose, then it is no longer a ***test*** set

It becomes a ***validation*** set

What’s the difference?

- a test set is *unbiased*
- a validation set will have an (overly) *optimistic bias*

  (remember the coin tossing experiments?)

Example

Suppose we have two hypotheses h_1, h_2 and that

$$R(h_1) = R(h_2) = p$$

Moreover, suppose that our error estimates for h_1, h_2, denoted by $\widehat{R}_{\text{val}}(h_1)$ and $\widehat{R}_{\text{val}}(h_2)$, are such that

$$\mathbb{P}\left[\widehat{R}_{\text{val}}(h_i) > p\right] = \mathbb{P}\left[\widehat{R}_{\text{val}}(h_i) < p\right] = \frac{1}{2}$$

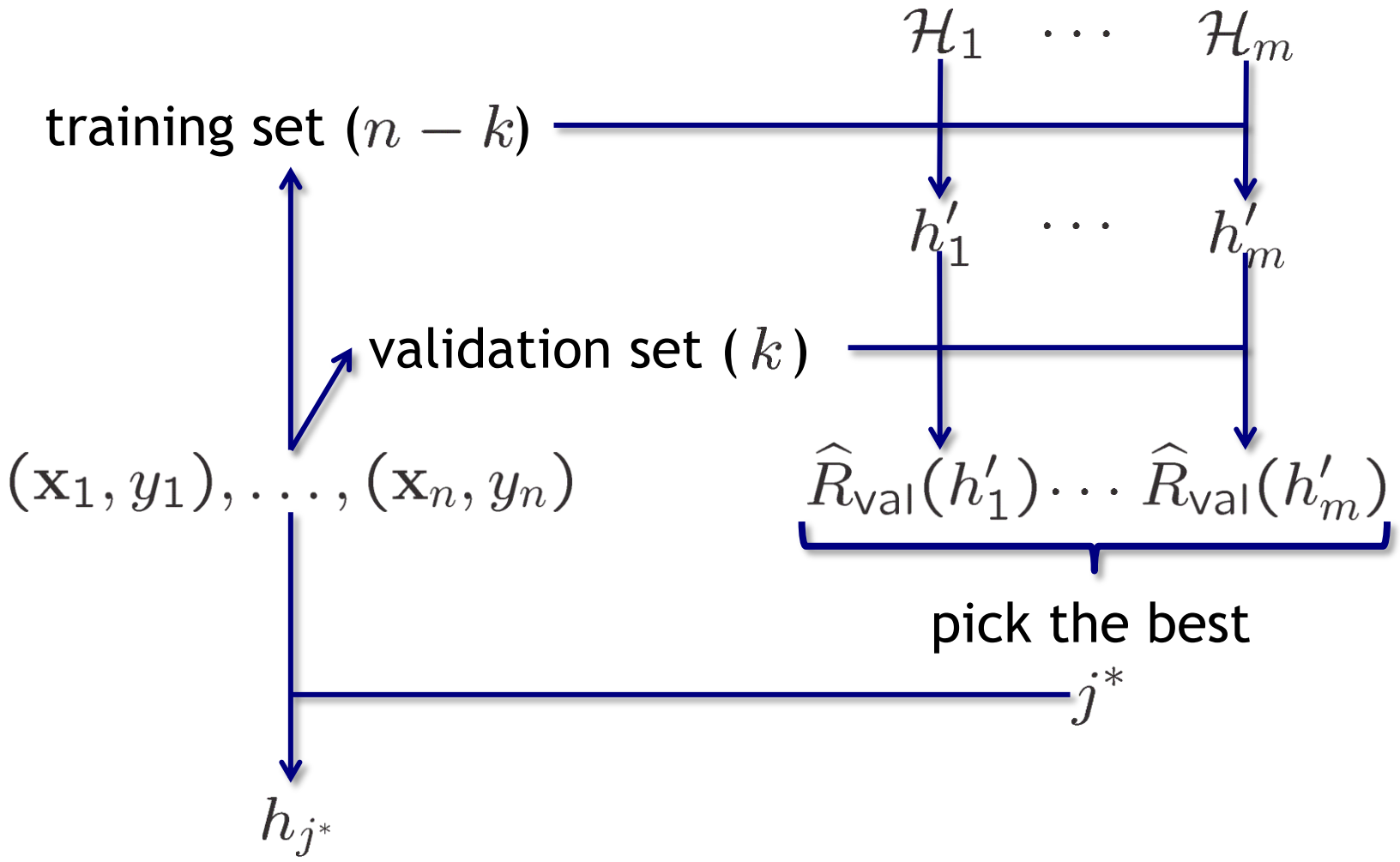
Pick $h \in \{h_1, h_2\}$ that minimizes $\widehat{R}_{\text{val}}(h)$

It is easy to argue that $\mathbb{E}\left[\widehat{R}_{\text{val}}(h)\right] < p$ ***optimistic bias***

Why? 75 % of the time, $\min\left(\widehat{R}_{\text{val}}(h_1), \widehat{R}_{\text{val}}(h_2)\right) < p$

Using validation for model selection

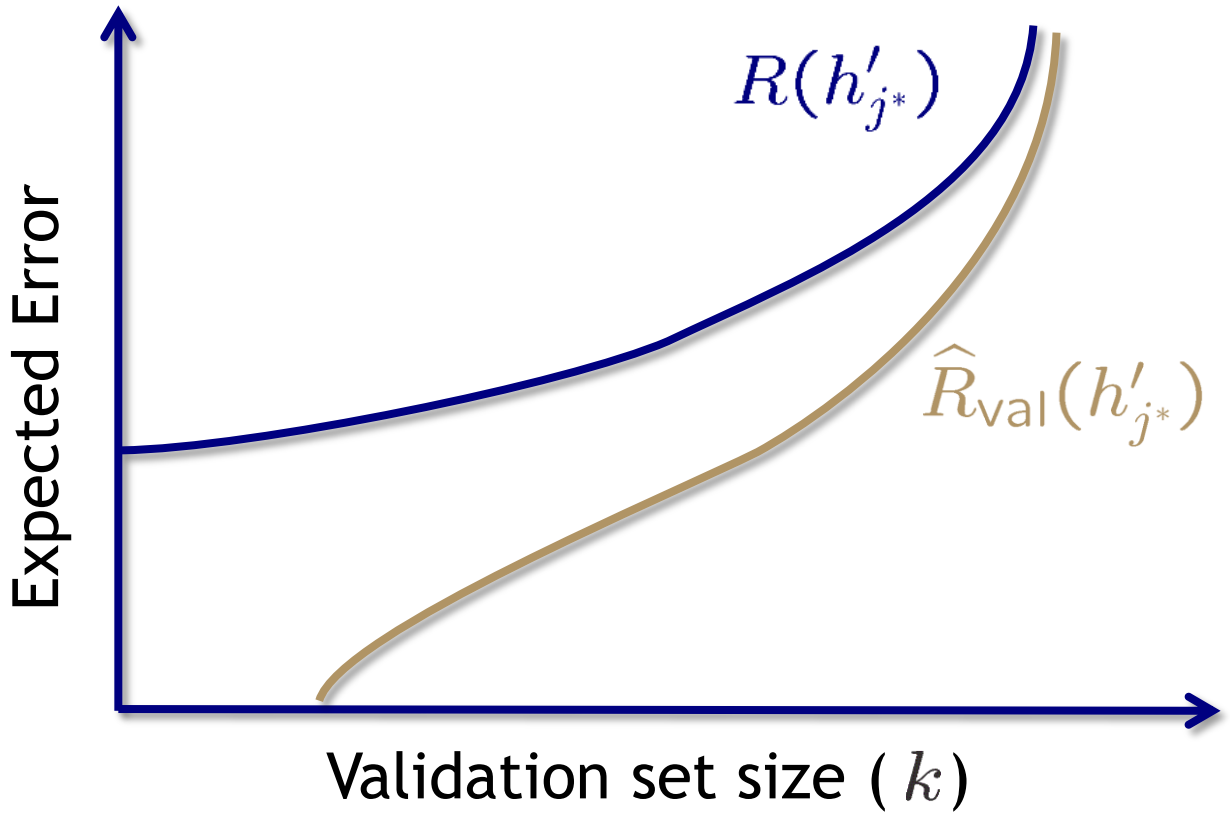
Suppose we have m models $\mathcal{H}_1, \ldots, \mathcal{H}_m$



The bias

We select the model $\mathcal{H}_{j^*}$ using the validation set

$\widehat{R}_{\text{val}}(h'_{j^*})$ is a biased estimate of $R(h'_{j^*})$ (and $R(h_{j^*})$)



Quantifying the bias

We've seen this before…

For m models $\mathcal{H}_1, \ldots, \mathcal{H}_m$, we use a data set of size k to pick the model that does best out of $\{h'_1, \ldots, h'_m\}$

Back to Hoeffding!

$$R(h'_{j^*}) \leq \widehat{R}_{\text{val}}(h'_{j^*}) + O\left(\sqrt{\frac{\log m}{k}}\right)$$

Or, if the $\mathcal{H}_j$ correspond to a few continuous parameters, we can use the VC approach to argue

$$R(h'_{j^*}) \leq \widehat{R}_{\text{val}}(h'_{j^*}) + O\left(\sqrt{\frac{\#\text{ of parameters}}{k}}\right)$$

Data contamination

We have now discussed three different kinds of estimates of the risk $R(h)$: $\widehat{R}_{\text{train}}(h), \widehat{R}_{\text{test}}(h), \widehat{R}_{\text{val}}(h)$

These three estimates have different degrees of "contamination" that manifests itself as a (deceptively) optimistic bias

- Training set: totally contaminated
- Testing set: totally clean
- Validation set: slightly contaminated

Validation dilemma

Back to our core dilemma in validation

We would like to argue that

$$R(h) \approx R(h') \approx \widehat{R}_{\text{val}}(h')$$

small k large k

All we need to do is set k so that it is simultaneously small and large

Can we do this?

Yes!

Leave one out

We need k to be small, so let's set $k = 1$!

$$\mathcal{D}_j = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_j, y_j), \ldots, (\mathbf{x}_n, y_n)$$

Select a hypothesis h'_j using the data set $\mathcal{D}_j$

Validation error $\widehat{R}_{\text{val}}(h'_j) = e(h'_j(\mathbf{x}_j), y_j) := e_j$

We set k to be too small, so this is a terrible estimate

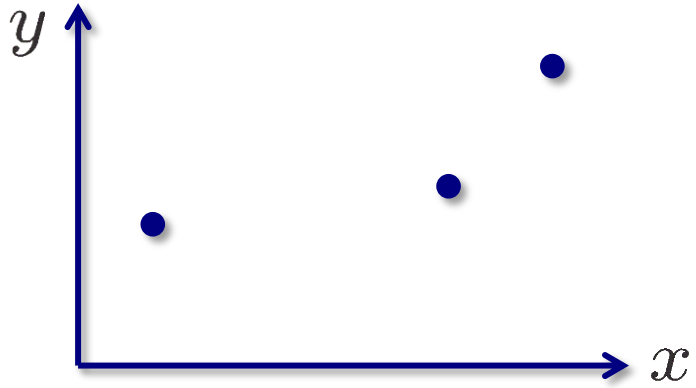
Repeat this for all possible choices of j and average!

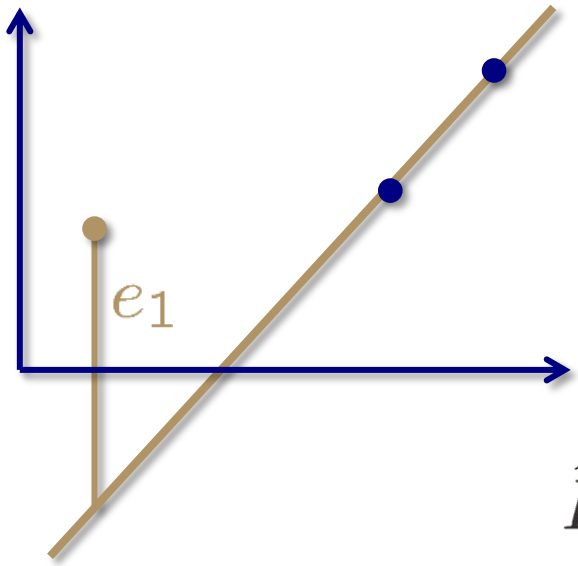
$$\widehat{R}_{\text{cv}} = \frac{1}{n} \sum_{j=1}^{n} e_j$$

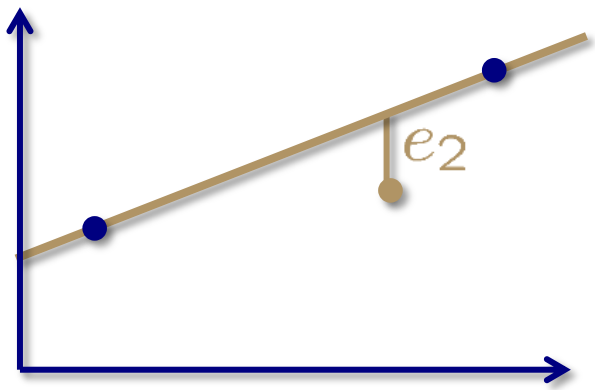
This is called the ***leave one out cross validation*** error

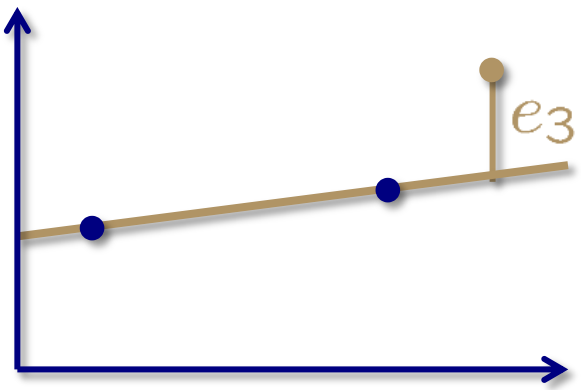
Example

Fitting a line to 3 data points









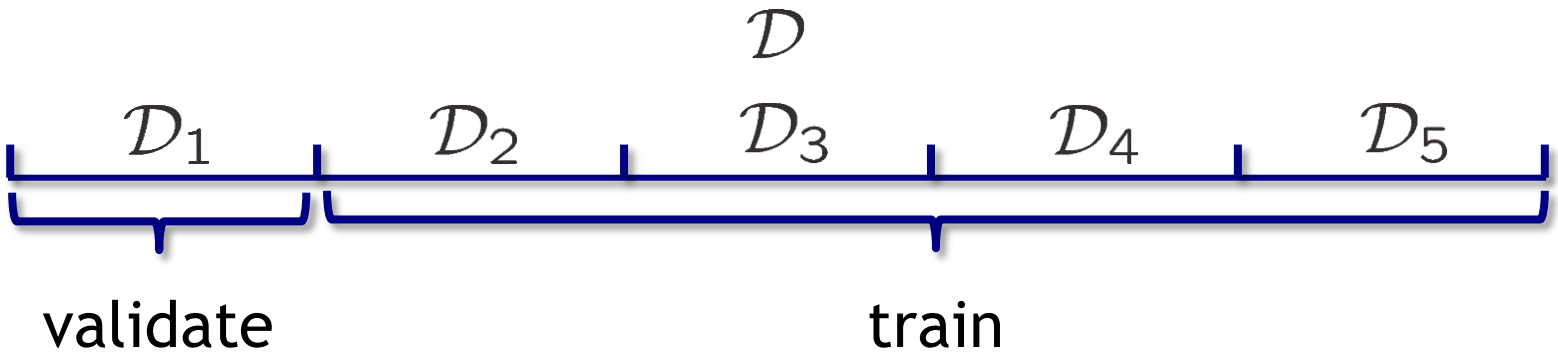
$$\widehat{R}_{\text{cv}} = \frac{1}{3}(e_1 + e_2 + e_3)$$

Leave more out

Leave one out: Train n times on $n-1$ points each

k-fold cross validation: Train $\frac{n}{k}$ times on $n-k$ points each

Example: $k = 5$



Iterate over all 5 choices of validation set and average

Common choices are $k = 5, 10$

Remarks

- For k-fold cross validation, the estimate depends on the particular choice of partition
- It is common to form several estimates based on different random partitions and then average them
- When using k-fold cross validation for classification, you should ensure that each of the sets $\mathcal{D}_j$ contain training data from each class in the same proportion as in the full data set

The bootstrap

What else can you do when your training set is really small?

You really need as much training data as possible to get reasonable results

Fix $B \geq 1$

For $b = 1, \ldots, B$, let $\mathcal{D}_b$ be a subset of size n obtained by ***sampling with replacement*** from the full data set $\mathcal{D}$

Example: $n = 5$

$$\mathcal{D}_1 = (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4), (\mathbf{x}_5, y_5), (\mathbf{x}_4, y_4), (\mathbf{x}_1, y_1)$$

$$\mathcal{D}_2 = (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_5, y_5), (\mathbf{x}_5, y_5), (\mathbf{x}_2, y_2)$$

$$\vdots$$

The bootstrap error estimate

Define $h_b :=$ model learned based on the data $\mathcal{D}_b$

$$\mathcal{D}_b^c := \mathcal{D} \setminus \mathcal{D}_b$$

Set $e_b = \dfrac{1}{|\mathcal{D}_b^c|} \displaystyle\sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_b^c} 1_{\{h_b(\mathbf{x}_i) \neq y_i\}}$

The ***bootstrap*** error estimate is then given by

$$\widehat{R}_{\mathsf{B}} := \frac{1}{B} \sum_{b=1}^{B} e_b$$

Bootstrap in practice

- Typically, B must be large (say, $B \approx 200$) for the estimate to be accurate
- Very computationally demanding
- $\widehat{R}_{\mathrm{B}}$ tends to be ***pessimistic***, so it is common to combine the training and bootstrap error estimates
- A common choice is the ***"0.632 bootstrap estimate"***

$$0.632\widehat{R}_{\mathrm{B}} + 0.368\widehat{R}_{\mathrm{train}}$$

- The "balanced" bootstrap chooses $\mathcal{D}_1, \ldots, \mathcal{D}_B$ such that each input-output pair appears exactly B times

Lecture 17: PCA

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

$$\mathbf{x}_i \rightarrow \theta_i \in \mathbb{R}^k$$

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)

Types of 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 $\mathbf{x} \rightarrow \theta$ linear or nonlinear?
4. Feature ***selection*** versus feature ***extraction***?

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

Principal component analysis (PCA)

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

The idea behind PCA is to find an approximation

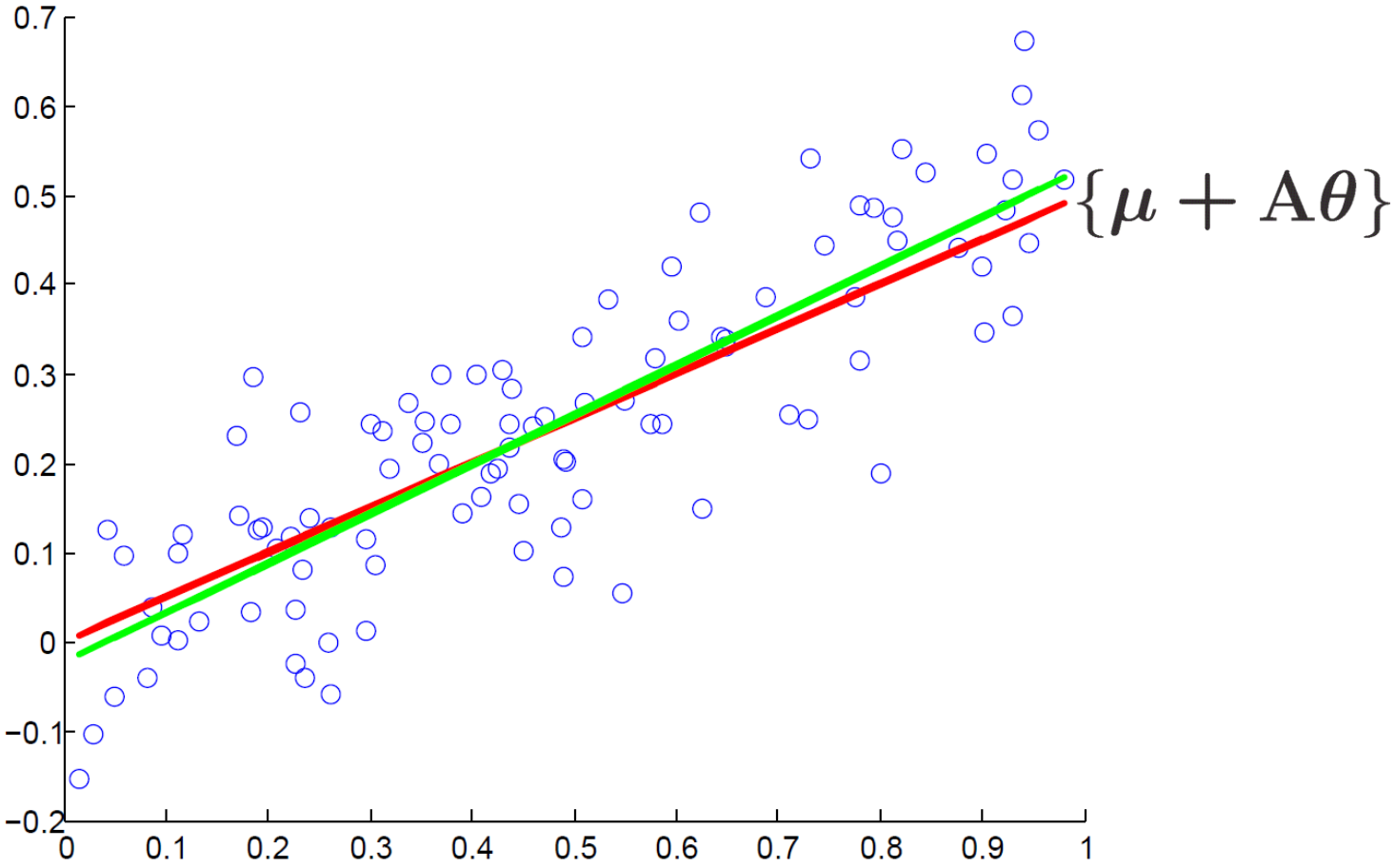
$$\mathbf{x}_i \approx \boldsymbol{\mu} + \mathbf{A}\boldsymbol{\theta}_i$$

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

$d = 2 \qquad k = 1$



Derivation of PCA

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

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

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

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

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

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

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

$$= \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^T \underbrace{(\mathbf{I} - \mathbf{A}\mathbf{A}^T)^T(\mathbf{I} - \mathbf{A}\mathbf{A}^T)}_{\mathbf{B}}(\mathbf{x}_i - \boldsymbol{\mu})$$

Determining μ

Taking the gradient with respect to μ and setting this equal to zero, we obtain

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

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

The choice of μ 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

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

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

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

Determining $\mathbf{A}$

Expanding this out, we obtain

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

$$= \sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i + \mathbf{x}_i^T\mathbf{A}\underbrace{\mathbf{A}^T\mathbf{A}}_{\mathbf{A}^T\mathbf{A}=\mathbf{I}}\mathbf{A}^T\mathbf{x}_i$$

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

Thus, we can instead focus on maximizing

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

Determining $\mathbf{A}$

Let $\mathbf{X}$ denote the $d \times n$ matrix with columns given by $\mathbf{x}_1, \ldots, \mathbf{x}_n$

Note that we can write

$$\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{A}\mathbf{A}^T \mathbf{x}_i = \text{Trace}(\mathbf{X}^T \mathbf{A}\mathbf{A}^T \mathbf{X})$$

$$= \text{Trace}(\mathbf{A}^T \mathbf{X}\mathbf{X}^T \mathbf{A})$$

$$(\text{Trace}(\mathbf{AB}) = \text{Trace}(\mathbf{BA}))$$

Define $\mathbf{S} = \mathbf{X}\mathbf{X}^T$

- 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 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

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

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

$\mathbf{a}$ 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 $\mathbf{S}$:

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

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

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

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

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

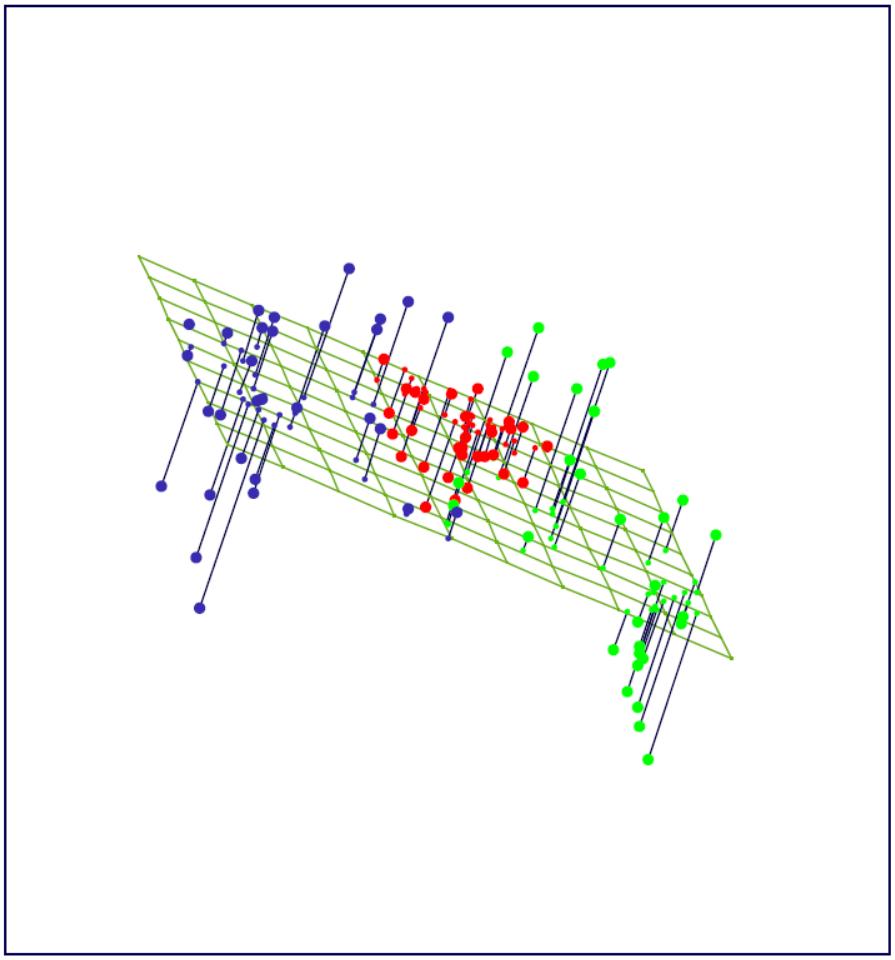
i.e., take the top k eigenvectors of $\mathbf{S}$

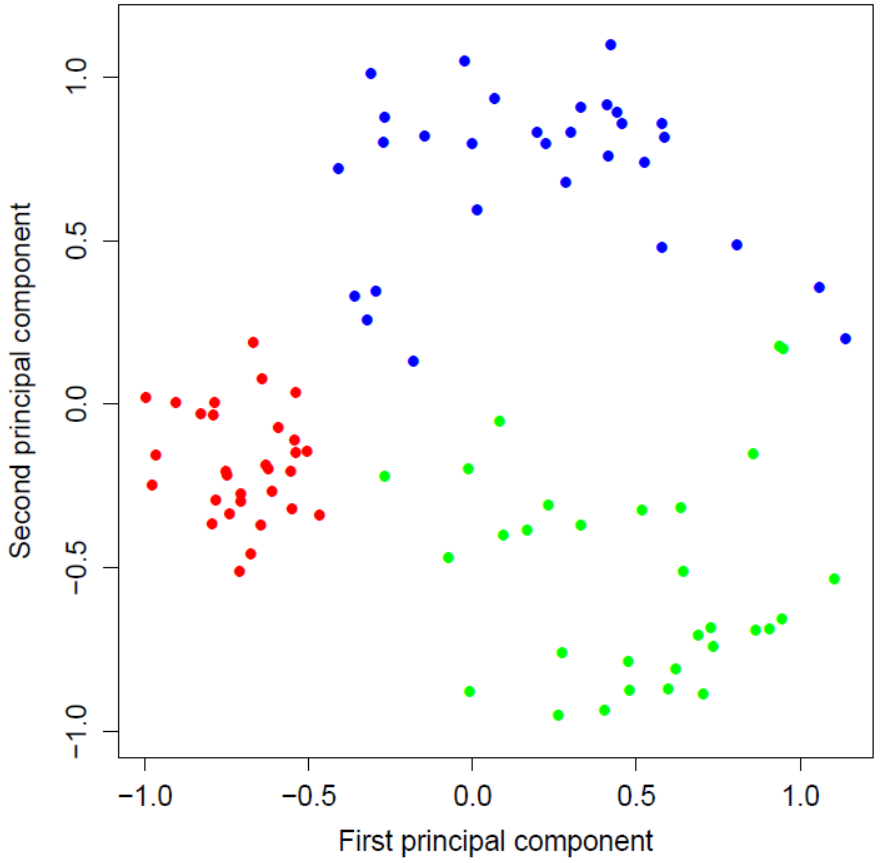
Terminology

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

Example

From Chapter 14 of Hastie, Tibshirani, and Friedman





Connection to SVD

Recall the singular value decomposition (SVD)

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

If $\mathbf{X}$ is a real $d \times n$ matrix

- $\mathbf{U}$ is a $d \times d$ orthonormal matrix
- $\mathbf{V}$ is an $n \times n$ orthonormal matrix
- $\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_r)$ is a $d \times n$ diagonal matrix where $r = \min(d, n)$ and

$$\sigma_i = i^{\text{th}} \text{ singular value}$$

$$= \text{square root of } i^{\text{th}} \text{ eigenvalue of } \mathbf{X}\mathbf{X}^T$$

The principal eigenvectors are the first k columns of $\mathbf{U}$

Practical matters

It is customary to ***center*** and ***scale*** a data set so that it has zero mean and unit variance along each feature

This puts all features on an "equal playing field"

These steps may be omitted when

- The data are known to be zero mean
- The data are known to have comparable units of measurement

To select k, we typically choose it to be large enough so that

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\theta}_i\|^2 = n(\lambda_{k+1} + \ldots + \lambda_d)$$

is sufficiently small

When to use PCA

- When the data form a single “point cloud” in space
- When the data are approximately Gaussian, or some other “elliptical” distribution
- When low-rank subspaces capture the majority of the variation

When not to use PCA?

When these assumptions are not justified

(i.e., pretty often)

Lecture 18: Multidimensional Scaling (MDS) and Nonlinear Embeddings

Principal component analysis (PCA)

Most common linear method for dimensionality reduction

The idea behind PCA is to find an approximation

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

where

- $\boldsymbol{\mu} = \frac{1}{n}\sum_i \mathbf{x}_i$
- $\boldsymbol{\theta}_i = \mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu}) \in \mathbb{R}^k$
- $\mathbf{A} \in \mathbb{R}^{d \times k}$ with orthonormal columns

If $\widetilde{\mathbf{X}}$ is the matrix with columns $\mathbf{x}_1 - \boldsymbol{\mu}, \ldots, \mathbf{x}_n - \boldsymbol{\mu}$ and we let $\widetilde{\mathbf{X}} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ denote the SVD of $\widetilde{\mathbf{X}}$, the optimal $\mathbf{A}$ is given by the first k columns of $\mathbf{U}$

Low-dimensional embedding

A closely related problem to dimensionality reduction is constructing ***low-dimensional embeddings***

Given an $n \times n$ ***dissimilarity matrix*** $\mathbf{D}$, find a dimension k and points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ such that $\rho(\mathbf{x}_i, \mathbf{x}_j) = d_{ij}$ for some distance function ρ

Applications

- dimensionality reduction
- visualization
- extend algorithms to non-Euclidean (or non-metric) data

Dissimilarity matrix

A dissimilarity matrix should satisfy

- $d_{ij} \geq 0$
- $d_{ij} = d_{ji}$
- $d_{ii} = 0$

Note that we do ***not*** require the triangle inequality, since in practice many measures of "dissimilarity" will not have this property

In general, a perfect embedding into the desired dimension will not exist

We will be interested mostly in ***approximate*** embeddings

Multidimensional scaling (MDS)

The problem of finding $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ such that $\rho(\mathbf{x}_i, \mathbf{x}_j)$ approximately agrees with d_{ij} is known as ***multidimensional scaling (MDS)***

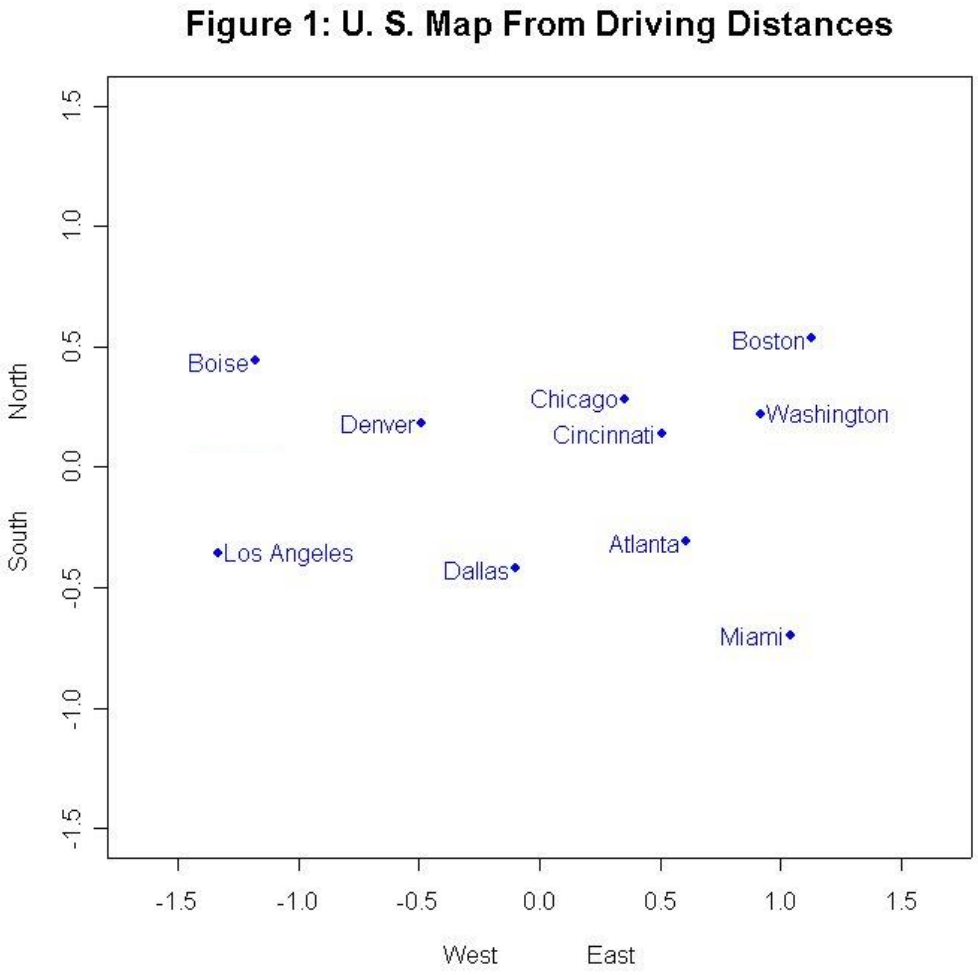
There are a number of variants of MDS based on

- our choice of distance function ρ
- how we quantify "approximately agrees with"
- whether we have access to all entries of $\mathbf{D}$

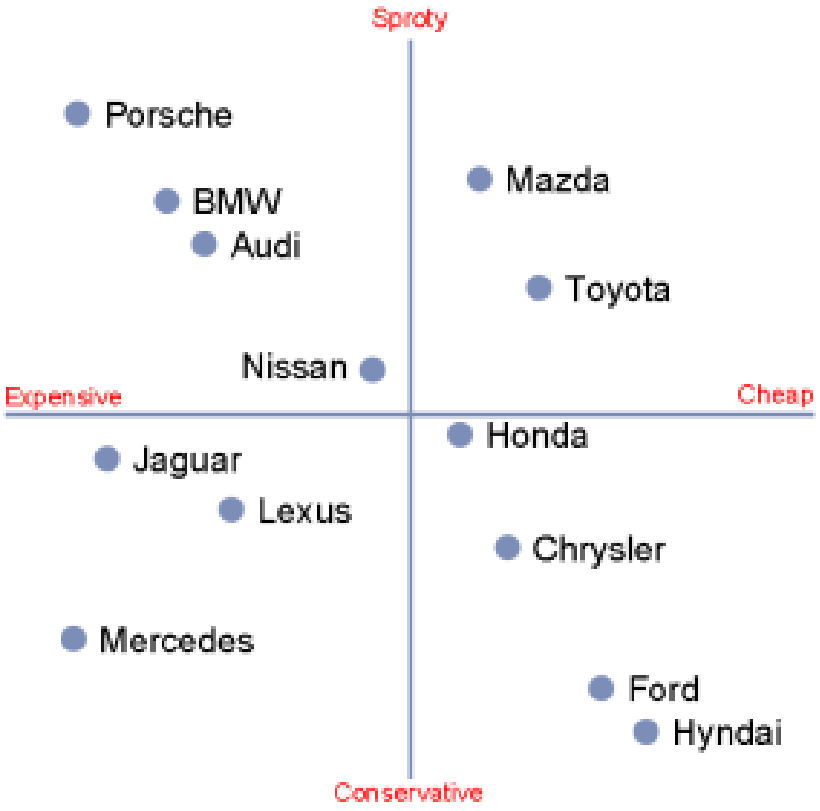
Main distinction

- ***metric*** methods attempt to ensure that $\rho(\mathbf{x}_i, \mathbf{x}_j) \approx d_{ij}$
- ***nonmetric*** methods only attempt to preserve rank ordering, i.e., if $d_{ij} \leq d_{\ell m}$ then nonmetric methods seek an embedding that satisfies $\rho(\mathbf{x}_i, \mathbf{x}_j) \leq \rho(\mathbf{x}_\ell, \mathbf{x}_m)$

Example: Creating a map



Example: Marketing



Example: Whisky

Euclidean embeddings

Today we will focus primarily on the metric case where we observe all of $\mathbf{D}$ and choose $\rho(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|$

To see how we might find an embedding, first consider the reverse process…

Given an (exact) embedding $\mathbf{X}$ (where the $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ form the columns of $\mathbf{X}$), how can we compute $\mathbf{D}$?

Consider $\mathbf{D}^2$, i.e., the matrix with entries

$$d_{ij}^2 = \|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i^T\mathbf{x}_j$$

$$\Rightarrow \mathbf{D}^2 = \mathbf{b}\mathbf{1}^T + \mathbf{1}\mathbf{b}^T - 2\mathbf{X}^T\mathbf{X}$$

where $\mathbf{b} = \left[\|\mathbf{x}_1\|^2, \ldots, \|\mathbf{x}_n\|^2\right]^T$ and $\mathbf{1} = [1, \ldots, 1]^T$

Finding the embedding

Thus, we know that

$$\mathbf{X}^T\mathbf{X} = \frac{1}{2}(\mathbf{b}\mathbf{1}^T + \mathbf{1}\mathbf{b}^T - \mathbf{D}^2)$$

We are given $\mathbf{D}^2$, but $\mathbf{b}$ is actually part of what we are trying to estimate, right?

Consider the ***"centering matrix"*** $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$

Observe that $\widetilde{\mathbf{X}} = \mathbf{X}\mathbf{H}$ is simply our data set $\mathbf{X}$ with the mean subtracted off

If all we know are distances between pairs of points, we have lost all information about any rigid transformation (e.g., a translation) of our data

Finding the embedding

We are free to enforce that our embedding is centered around the origin, in which case we are interested in

$$\begin{aligned}\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}} &= \mathbf{H}^T\mathbf{X}^T\mathbf{X}\mathbf{H} \\ &= \frac{1}{2}(\mathbf{H}\mathbf{b}\mathbf{1}^T\mathbf{H} + \mathbf{H}\mathbf{1}\mathbf{b}^T\mathbf{H} - \mathbf{H}\mathbf{D}^2\mathbf{H})\end{aligned}$$

Note that $1^T\mathbf{H} = \mathbf{H}1 = 0$

$$\Rightarrow \quad \widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H}$$

Given $\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}$, we can find $\widetilde{\mathbf{X}}$ by computing an eigendecomposition

Classical MDS

Even if a dissimilarity matrix $\mathbf{D}$ cannot be perfectly embedded into k dimensions, this suggests an approximate algorithm

1. Form $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H}$
2. Compute the eigendecomposition $\mathbf{B} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$
3. Set $\mathbf{V} = \mathbf{U}_k\mathbf{\Lambda}^{1/2}$, i.e., the matrix whose columns are given by $\mathbf{v}_i = \sqrt{\lambda_i}\mathbf{u}_i$ for $i = 1, \ldots, k$
4. Return $\mathbf{X} = \mathbf{V}^T$

In MATLAB: **cmdscale**

Note: The algorithm will crash if $\lambda_i < 0$ for some $i \leq k$

Relation to PCA

Suppose we have $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^d$ and set $\mathbf{D}$ to be such that

$$d_{ij} = \|\mathbf{z}_i - \mathbf{z}_j\|$$

Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ be the embedding obtained by applying classical MDS to the dissimilarity matrix $\mathbf{D}$

We can interpret $\mathbf{x}_i$ as the projection of $\widetilde{\mathbf{z}}_i = \mathbf{z}_i - \boldsymbol{\mu}$ onto the first k principal eigenvectors

In this sense, PCA and classical MDS are essentially equivalent

MDS can be applied in a broader variety of settings

Alternative formulations

Classical MDS finds the embedding that minimizes the loss function

$$\|\mathbf{X}^T\mathbf{X} - \mathbf{B}\|$$

Many other choices for loss function exist

Perhaps the most common alternative is the ***stress*** function

$$\sum_{i,j} w_{ij}(d_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)^2$$

where the w_{ij} are fixed weights, e.g., $w_{ij} \in \{0, 1\}$

Stress criteria are typically minimized by iterative procedures

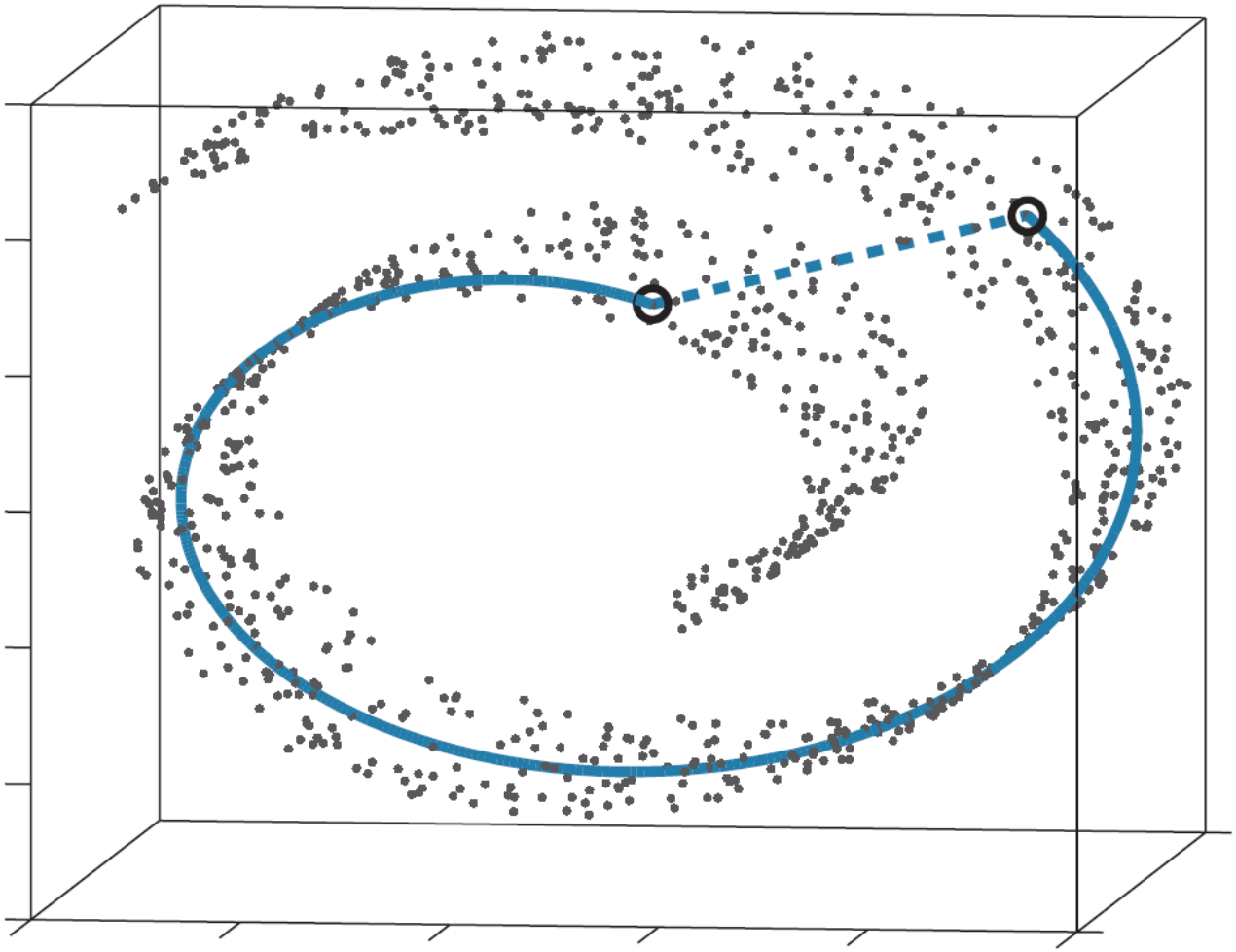
In MATLAB: **mdscale**

Nonlinear embeddings

The goal of embeddings/dimensionality reduction is to map a (potentially) high-dimensional dataset into a low-dimensional one in a way that preserves ***global*** and/or ***local*** geometric and topological properties are preserved

While PCA/MDS is the most popular method for this in practice, many high-dimensional data sets have ***nonlinear*** structure that is difficult to capture via linear methods

Example: Swiss roll



Isomap

Isometric feature **map**ping is essentially just the application of MDS to dissimilarities which are derived not from the raw data but based on shortest paths in a ***proximity graph***

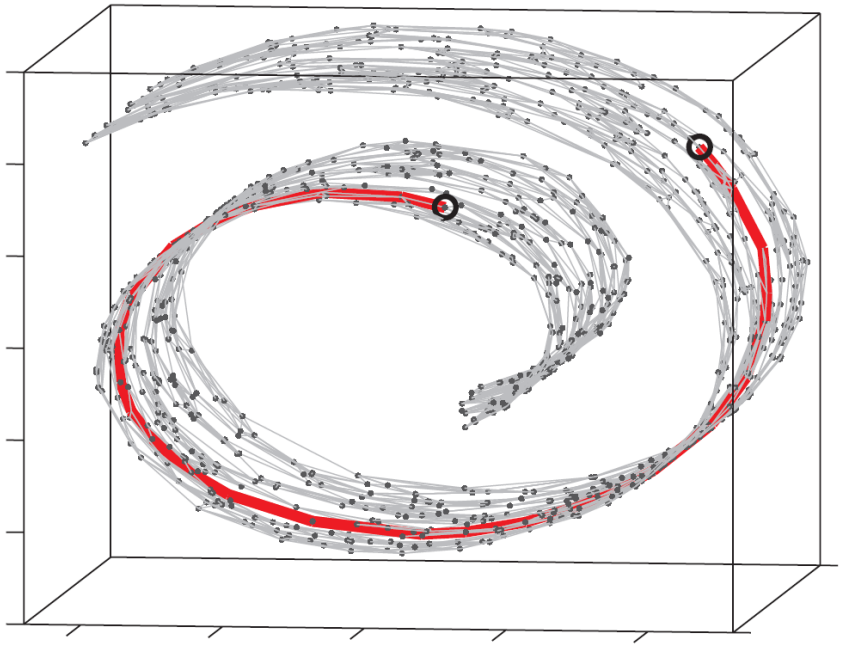
- e.g., a graph containing edges only between a data point and its nearest-neighbors

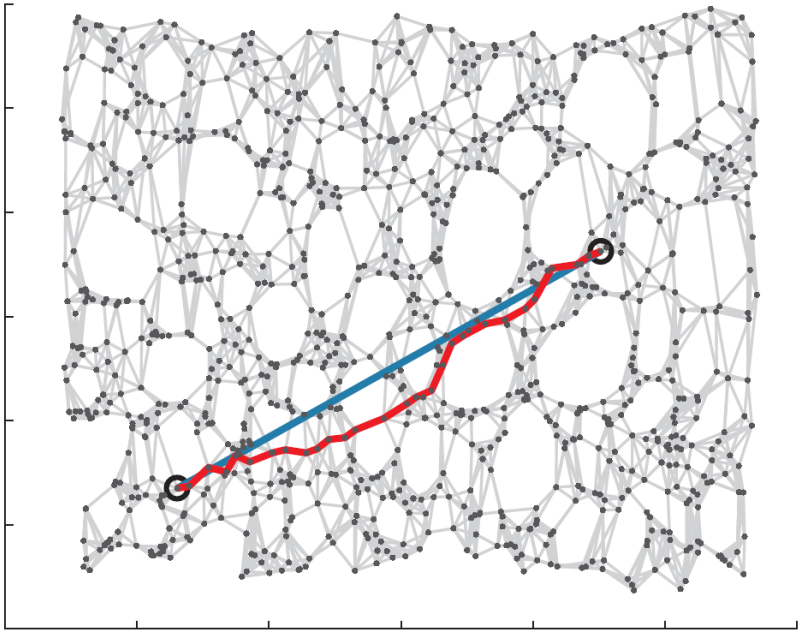
This path length is viewed as an approximation to a ***geodesic distance*** on the underlying data ***manifold***

Reference: Tennenbaum, de Silva, and Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction" *Science*, 2000.

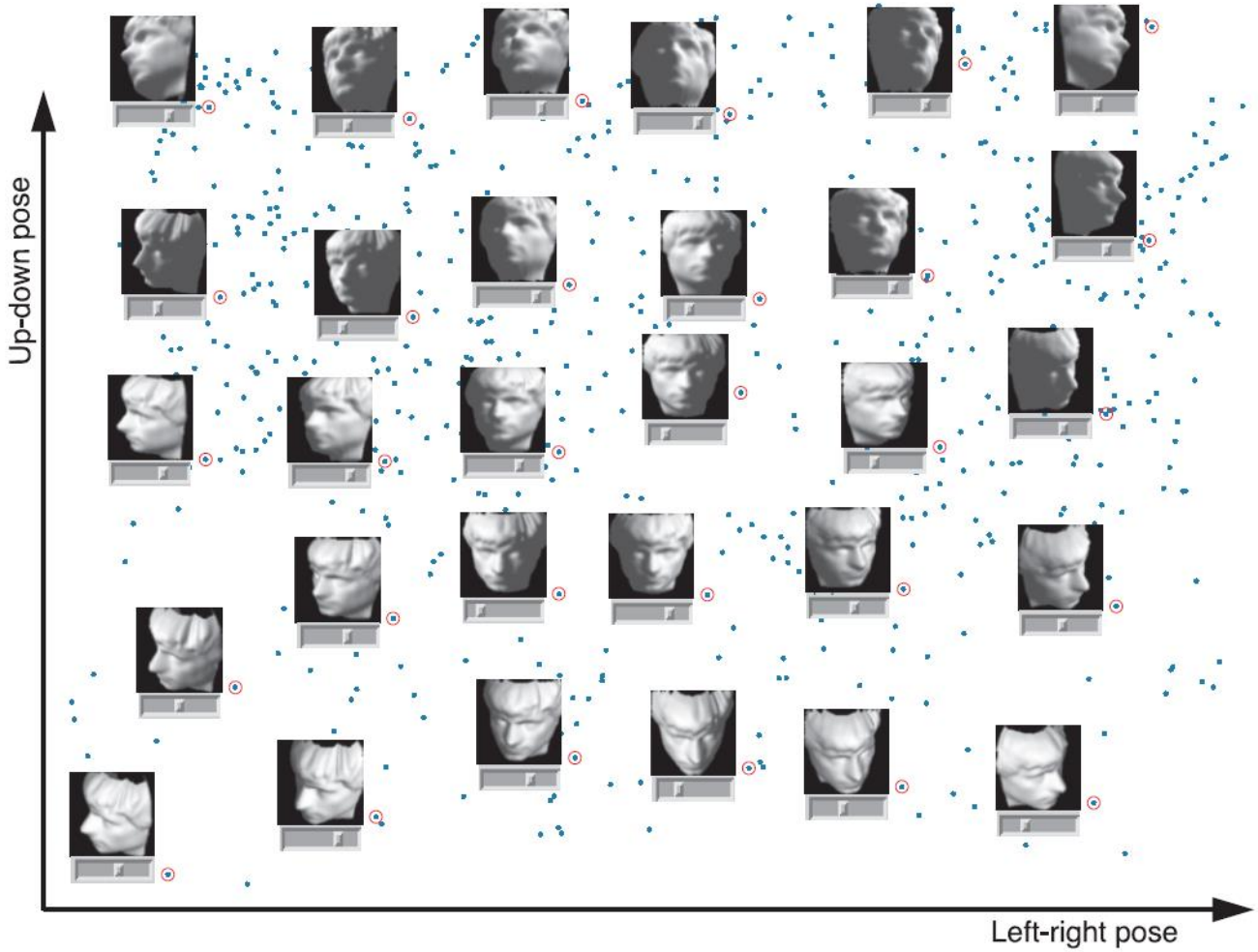
The idea behind Isomap is most easily seen by returning to our "Swiss roll" example

Example: Swiss roll

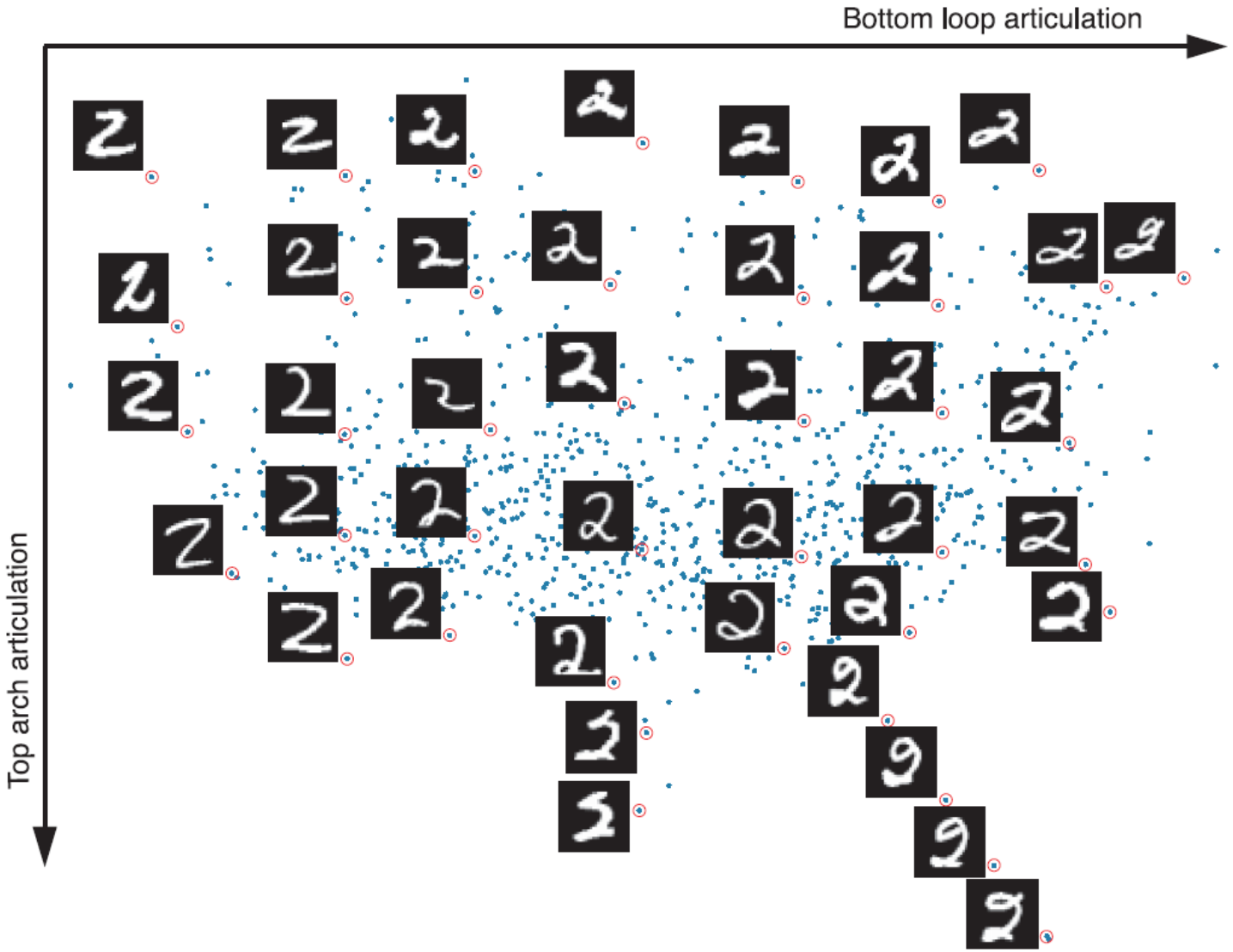




Example: Facial pose



Example: Handwritten digits



Locally linear embedding (LLE)

A potential challenge for Isomap is that estimates of the geodesic distance between points that are very far from each other on the manifold can grow increasingly inaccurate

Locally linear embedding (LLE) capitalizes on the intuition that a data manifold that is globally nonlinear will still appear linear in local pieces

LLE does not try to explicitly model global geodesic distances, but instead tries to preserve the structure in the data by trying to "patch together" local pieces of the manifold

Reference: Roweis and Saul, "Nonlinear dimensionality reduction by locally linear embedding" *Science*, 2000.

The LLE algorithm

Given high-dimensional data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, LLE consists of

1. For each $\mathbf{x}_i$, define a local neighborhood N_i
   e.g., the k-nearest neighbors

2. Solve

$$\min_{\{w_{ij}\}} \sum_{i=1}^{n} \|\mathbf{x}_i - \sum_{j \in N_i} w_{ij}\mathbf{x}_j\|^2$$
$$\text{s.t.} \quad \sum_j w_{ij} = 1 \qquad w_{ij} \geq 0$$

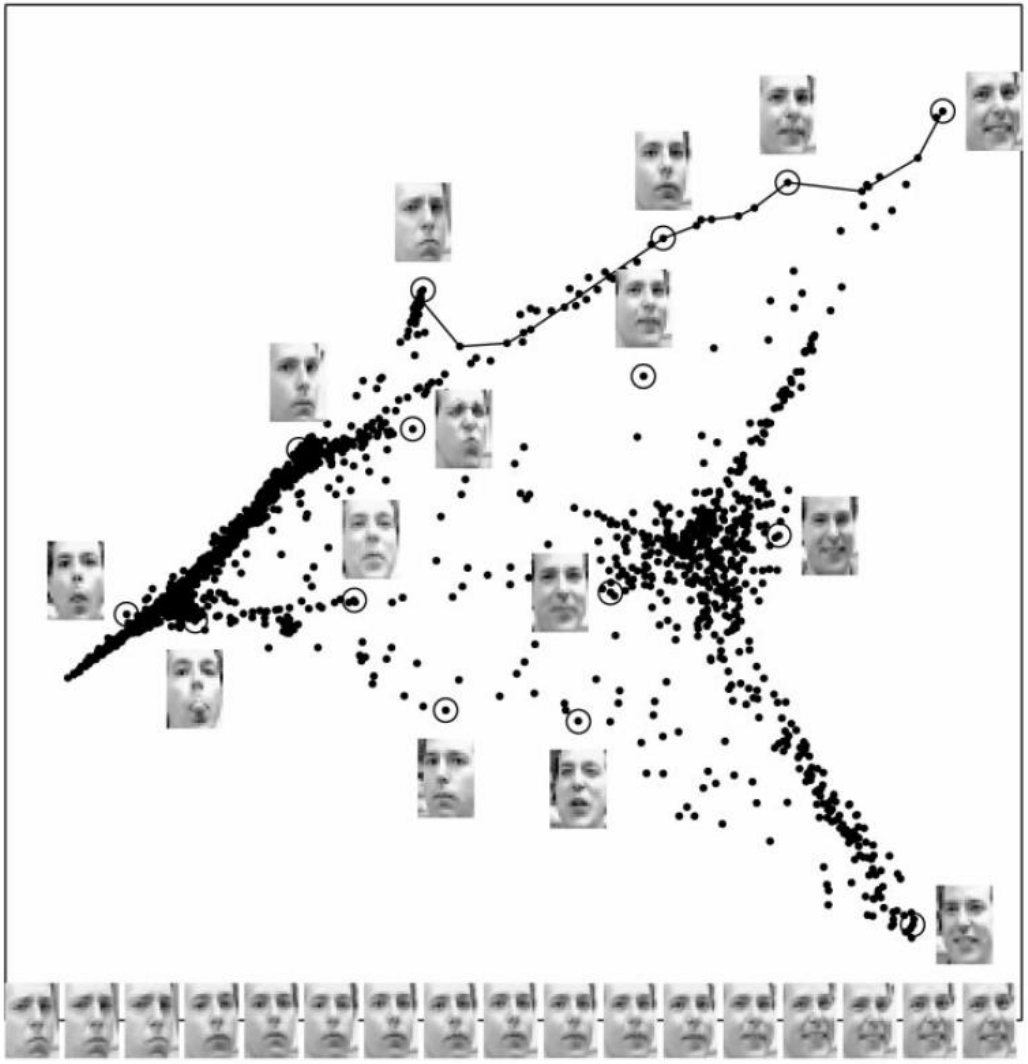
constrained least squares

3. Fix $\{w_{ij}\}$ and solve

$$\min_{\{\mathbf{y}_i\}} \sum_{i=1}^{n} \|\mathbf{y}_i - \sum_{j \in N_i} w_{ij}\mathbf{y}_j\|^2$$

eigenvalue problem

Example: Facial expression



Isomap versus LLE

- Isomap
  - emphasizes global distance preservation
  - can distort local geometry
- LLE
  - emphasizes local geometry preservation
  - can distort global geometry
    - far away points can get mapped close to each other
- Many other variants of nonlinear dimensionality reduction along these same lines have been developed
  - Laplacian eigenmaps, local tangent space alignment, Hessian LLE, diffusion maps, …

Kernel PCA

Yet another approach to nonlinear dimensionality reduction is to "kernelize" PCA in the same way that we did for SVMs

- Map the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$

$$\mathbf{x}_i = \begin{bmatrix} x_i(1) \\ \vdots \\ x_i(d) \end{bmatrix} \xrightarrow{\Phi} \Phi(\mathbf{x}_i) = \begin{bmatrix} \Phi^{(1)}(\mathbf{x}_i) \\ \\ \vdots \\ \\ \Phi^{(p)}(\mathbf{x}_i) \end{bmatrix}$$

where Φ is nonlinear and $d \ll p$

- Apply PCA to $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$

Summary of kernel PCA

Input: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, kernel k, desired dimension r

1. Form $\widetilde{\mathbf{K}} = \mathbf{HKH}$, where $\mathbf{K}$ is our kernel matrix and $\mathbf{H}$ is the centering matrix
2. Compute eigendecomposition $\widetilde{\mathbf{K}} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$
3. Set $\boldsymbol{\alpha}_j = \frac{1}{\sqrt{\lambda_j}}\mathbf{u}_j, j = 1, \ldots, r$

Output: A mapping $\mathbf{x} \to \mathbf{y} \in \mathbb{R}^r$ where

$$y(j) = \sum_{i=1}^{n} \alpha_j(i) k(\mathbf{x}, \mathbf{x}_i)$$

Lecture 19: Feature Selection

Multidimensional scaling (MDS)

The problem of finding $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^k$ such that $\rho(\mathbf{x}_i, \mathbf{x}_j)$ approximately agrees with d_{ij} is known as ***multidimensional scaling (MDS)***

There are a number of variants of MDS based on

- our choice of distance function ρ
- how we quantify "approximately agrees with"
- whether we have access to all entries of $\mathbf{D}$

Main distinction

- ***metric*** methods attempt to ensure that $\rho(\mathbf{x}_i, \mathbf{x}_j) \approx d_{ij}$
- ***nonmetric*** methods only attempt to preserve rank ordering, i.e., if $d_{ij} \leq d_{\ell m}$ then nonmetric methods seek an embedding that satisfies $\rho(\mathbf{x}_i, \mathbf{x}_j) \leq \rho(\mathbf{x}_\ell, \mathbf{x}_m)$

Classical MDS

Even if a dissimilarity matrix $\mathbf{D}$ cannot be perfectly embedded into k dimensions, this suggests an approximate algorithm

1. Form $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}^2\mathbf{H}$
2. Compute the eigendecomposition $\mathbf{B} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$
3. Set $\mathbf{V} = \mathbf{U}_k\mathbf{\Lambda}^{1/2}$, i.e., the matrix whose columns are given by $\mathbf{v}_i = \sqrt{\lambda_i}\mathbf{u}_i$ for $i = 1, \ldots, k$
4. Return $\mathbf{X} = \mathbf{V}^T$

In MATLAB: **cmdscale**

Note: The algorithm will crash if $\lambda_i < 0$ for some $i \leq k$

Nonlinear embeddings

- Isomap
  - emphasizes global distance preservation
  - can distort local geometry
- LLE
  - emphasizes local geometry preservation
  - can distort global geometry
    - far away points can get mapped close to each other
- Many other variants of nonlinear dimensionality reduction along these same lines have been developed
  - Laplacian eigenmaps, local tangent space alignment, Hessian LLE, diffusion maps, …

Kernel PCA

Yet another approach to nonlinear dimensionality reduction is to "kernelize" PCA in the same way that we did for SVMs

- Map the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$

$$\mathbf{x}_i = \begin{bmatrix} x_i(1) \\ \vdots \\ x_i(d) \end{bmatrix} \xrightarrow{\Phi} \Phi(\mathbf{x}_i) = \begin{bmatrix} \Phi^{(1)}(\mathbf{x}_i) \\ \\ \vdots \\ \\ \Phi^{(p)}(\mathbf{x}_i) \end{bmatrix}$$

where Φ is nonlinear and $d \ll p$

- Apply PCA to $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)$

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

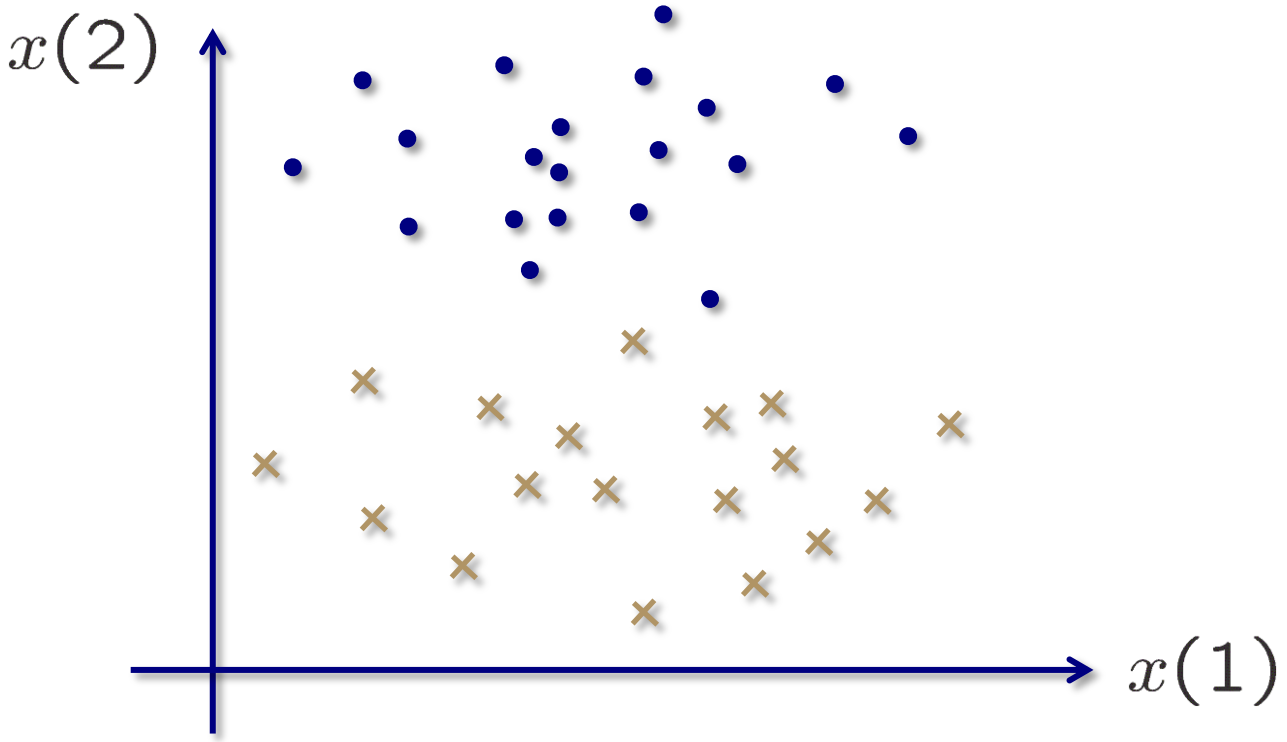
Disadvantage

- 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

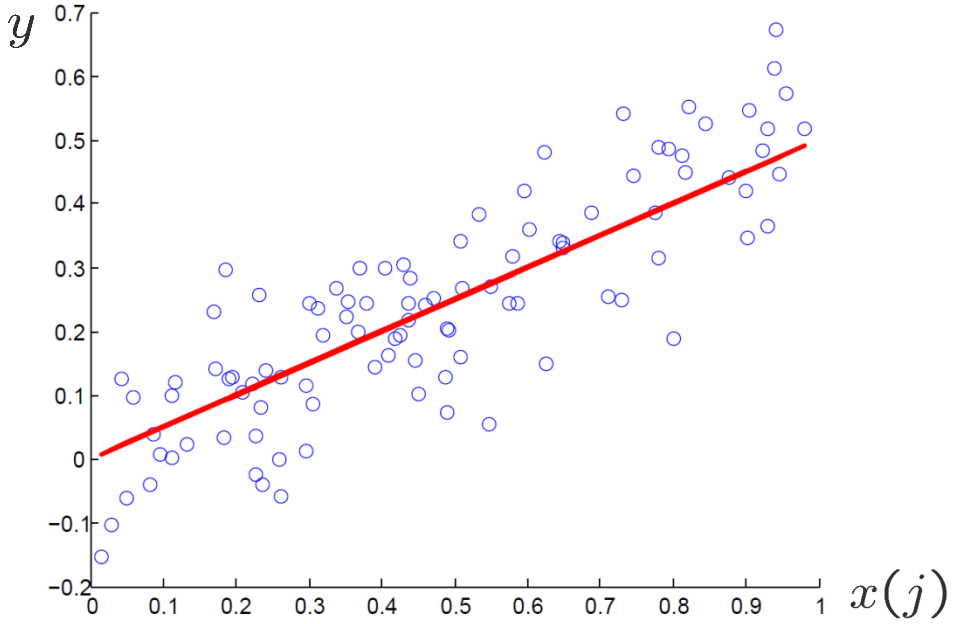
If the data is separable, then we can compute

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

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

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

Wrapper methods

Wrapper methods have three basic 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 ***very slow***

Wrapper methods derive their name from the fact that they "wrap around" a basic learning algorithm, calling it many times for different feature subsets

Examples

1. LDA, LR, SVM, nearest neighbors, neural nets, …
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 adding removing 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

This is easiest to describe through an example

The Lasso

The "Lasso" (Least absolute shrinkage and selection operator) is a method for linear regression

It is just like ridge regression, except that we use an ℓ_1 penalty instead of an ℓ_2 penalty

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \boldsymbol{\beta}^T \mathbf{x}_i)^2 + \lambda \|\boldsymbol{\beta}\|_1$$

where $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{d} |\beta(j)|$

Alternative formulations

Denoting

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_1(1) & \cdots & x_1(d) \\ x_2(1) & \cdots & x_2(d) \\ \vdots & \ddots & \vdots \\ x_n(1) & \cdots & x_n(d) \end{bmatrix}$$

we can re-express the Lasso in a constrained form

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$$
$$\text{s.t. } \|\boldsymbol{\beta}\|_1 \leq s$$

Solving the Lasso

$$\widehat{\beta} = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2$$
$$\text{s.t. } \|\beta\|_1 \leq s$$

The Lasso is a convex optimization problem

It can be solved via ***quadratic programming***

$$\widehat{\beta} = \arg\min_{\beta,\mathrm{t}} \|\mathbf{y} - \mathbf{X}\beta\|^2$$
$$\text{s.t. } -t(j) \leq \beta(j) \leq t(j)$$
$$\sum_j t(j) \leq s, t(j) \geq 0$$

The Lasso as embedded feature selection

Why can I claim that the Lasso is doing feature selection?

The Lasso solution is ***sparse*** (has only a few nonzeros)

➡ simultaneous regression and feature selection

The Lasso became popular because of the ***empirical*** observation that it usually produces sparse solutions

More recently, there has been a tremendous amount of research into ℓ_1 penalties, and there are now precise theoretical guarantees showing:

Under certain natural conditions, if only k features are significant, the Lasso will correctly identify them as long as we have $n \geq O(k \log d)$ observations

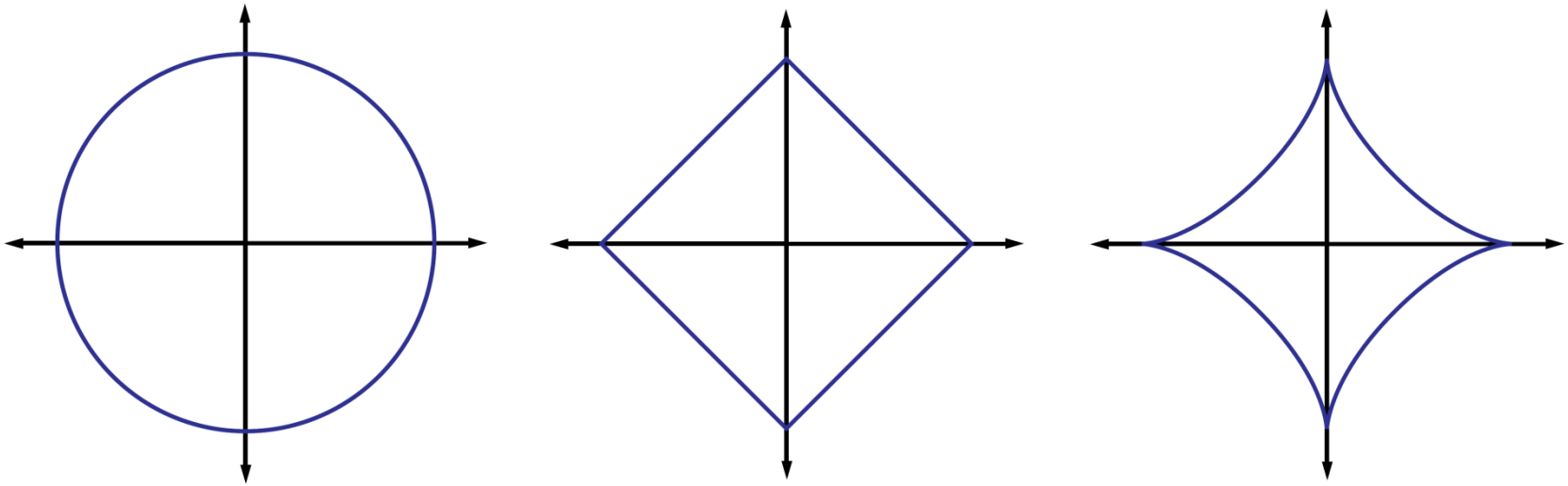
The geometry of the ℓ_1-norm

The reason why ℓ_1 penalties promote sparsity has to do with the shape of the "ℓ_1-ball"

$$B_p(s) := \{\boldsymbol{\beta} : \|\boldsymbol{\beta}\|_p = s\}$$

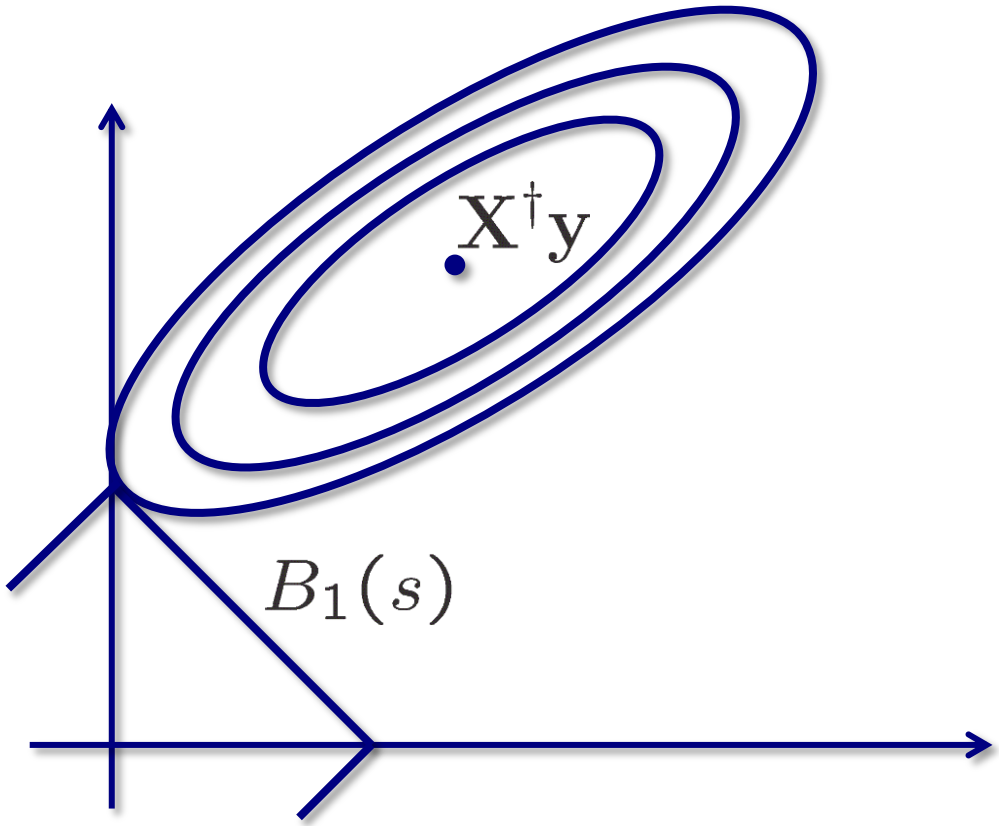
Examples

$B_2(1)$ $B_1(1)$ $B_{0.5}(1)$



The geometry of the Lasso

Recall that the set $\{\beta : \|\mathbf{y} - \mathbf{X}\beta\|^2 = c\}$ is an ellipse



The ellipse tends to intersect $B_1(s)$ at a corner, which is where some (most) of the $\beta(j)$ are zero

Remarks

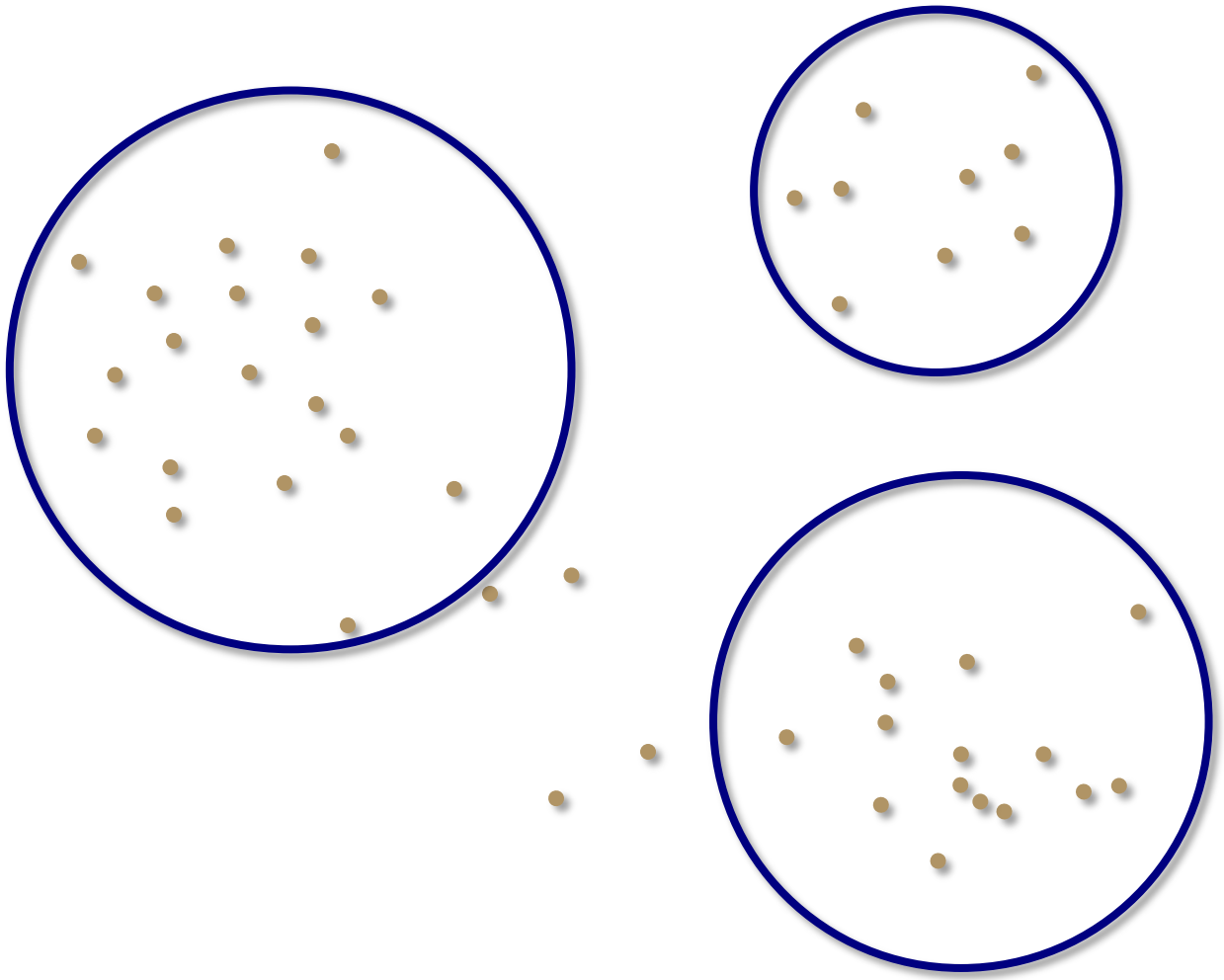
- The argument that ℓ_1 penalization leads to sparsity actually works for any ℓ_p with $p \leq 1$, but if $p < 1$ the optimization problem is ***nonconvex***

- Decreasing s (or increasing λ) leads to
  - sparser solutions (fewer features)
  - increasing squared error
  - parameter usually fixed using validation

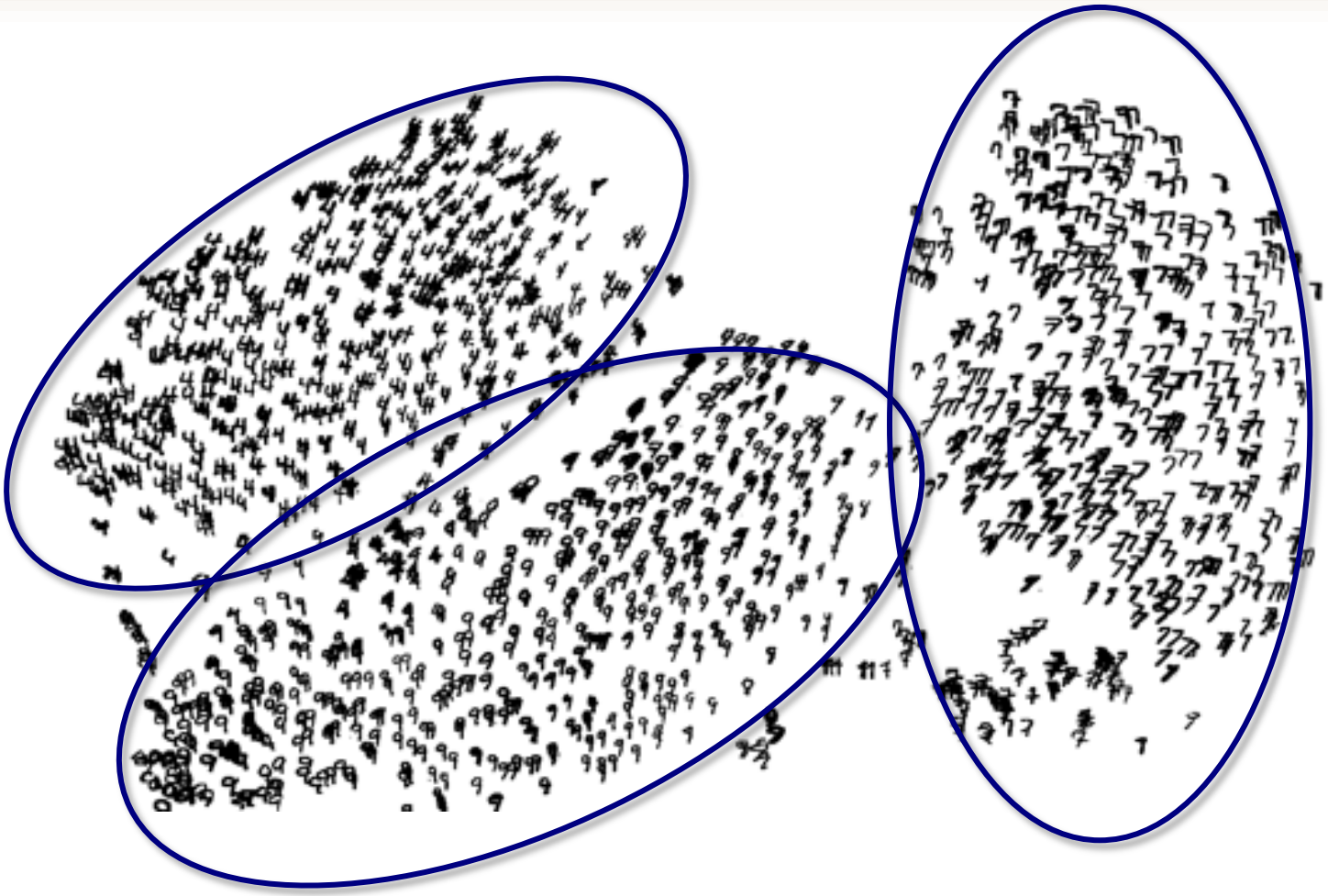
- There are lots of algorithms for efficiently solving the optimization problem, some of which give you the entire path of solutions as a function of s very efficiently
  - see Hastie, Tibshiarni, and Friedman for all the gory details

Lecture 20: K-means Clustering

Clustering



Example



Formal definition

Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

The goal of ***clustering*** is to assign the data to disjoint subsets called ***clusters***, so that points in the same cluster are more similar to each other than points in different clusters

A clustering can be represented by a cluster map, which is a function

$$C : \{1, \ldots, n\} \to \{1, \ldots, K\}$$

where K is the number of clusters

K-means criterion

Choose C to minimize

$$W(C) = \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

where

$$\boldsymbol{\mu}_k := \frac{1}{n_k} \sum_{i:C(i)=k} \mathbf{x}_i \qquad n_k = |\{i : C(i) = k\}|$$

Note that K is assumed fixed and known

$W(C)$ is sometimes called the "***within-cluster scatter***"

Within-cluster scatter

It is possible to show that

$$W(C) = \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

$$= \frac{1}{2} \sum_{k=1}^{K} \sum_{i:C(i)=k} \underbrace{\left[\frac{1}{n_k} \sum_{j:C(j)=k} \|\mathbf{x}_i - \mathbf{x}_j\|^2 \right]}$$

average distance between $\mathbf{x}_i$ and all other points in the same cluster

How many clusterings?

How many possible cluster maps C do we need to consider?

$$S(n, K) = \text{\# of clusterings of } n \text{ objects into } K \text{ clusters}$$
$$= S(n-1, K-1) + KS(n-1, K)$$

Solutions to this recurrence (with the natural boundary conditions) are called ***Stirling's numbers of the second kind***

$$S(n, K) = \frac{1}{K!} \sum_{k=1}^{K} (-1)^{K-k} \binom{K}{k} k^n$$

Examples

- $S(10, 4) = 34,105$
- $S(19, 4) \approx 10^{10}$

Minimizing the K-means criterion

There is no known efficient search strategy for this space

Can be solved (exactly) in time $O(n^{dK+1} \log n)$

Completely impractical unless both d and K are extremely small

- e.g., $d = 2, K = 3$ already results in $O(n^7 \log n)$

More formally, minimizing the K-means is a ***combinatorial*** optimization problem (NP-hard)

Instead, we resort to an ***iterative, suboptimal algorithm***

Another look at K-means

Recall that we want to find

$$C^* = \arg\min_{C} \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

Note that for fixed C and K

$$\boldsymbol{\mu}_k = \arg\min_{\mathbf{m}} \sum_{i:C(i)=k} \|\mathbf{x}_i - \mathbf{m}\|^2$$

Therefore, we can equivalently write

$$C^* = \arg\min_{C, \{\mathbf{m}_k\}_{k=1}^{K}} \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \mathbf{m}_k\|^2$$

An iterative algorithm

$$C^* = \arg\min_{C,\{\mathbf{m}_k\}_{k=1}^K} \underbrace{\sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \mathbf{m}_k\|^2}_{W(C,\{\mathbf{m}_k\}_{k=1}^K)}$$

This suggests an iterative algorithm

1. Given C, choose $\mathbf{m}_k$ to minimize $W(C, \{\mathbf{m}_k\}_{k=1}^K)$

2. Given $\mathbf{m}_k$, choose C to minimize $W(C, \{\mathbf{m}_k\}_{k=1}^K)$

K-means clustering algorithm

The solutions to each sub-problem are given by

1. $\mathbf{m}_k^* = \frac{1}{n_k} \sum_{i:C(i)=k} \mathbf{x}_i$

2. $C^*(i) = \arg\min_k \|\mathbf{x}_i - \mathbf{m}_k\|^2$

Algorithm

Initialize $\mathbf{m}_k, k = 1, \ldots, K$

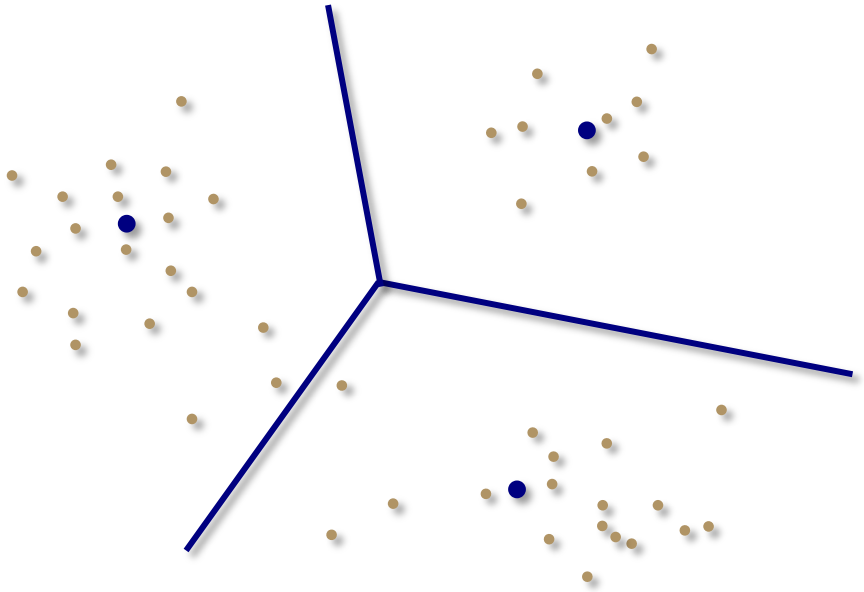
Repeat until clusters don't change

- $C(i) = \arg\min_k \|\mathbf{x}_i - \mathbf{m}_k\|^2$
- $\mathbf{m}_k = \frac{1}{n_k} \sum_{i:C(i)=k} \mathbf{x}_i$

Cluster geometry

Clusters are "nearest neighbor" regions or ***Vornoi cells*** defined with respect to the cluster means

Cluster boundaries are formed by the intersections of ***hyperplanes***



K-means will "fail" if clusters are ***nonconvex***

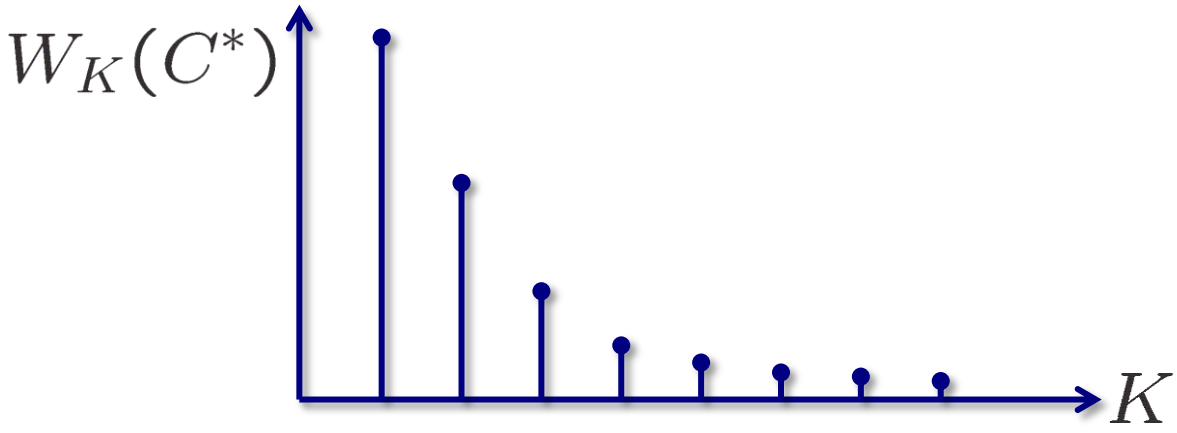
Remarks

- The algorithm is typically initialized by setting each $\mathbf{m}_k$ to be a ***random*** point
- Since the algorithm often finds a local minimum, several random initializations are recommended
- Algorithm originally discovered at Bell Labs as an approach to vector quantization
- If we replace the ℓ_2 norm with the ℓ_1 norm in our function $W(C)$, then
  - the geometry of our Vornoi regions will change
  - the "center" of each region is actually calculated via the median in each dimension
  - results in ***K-medians clustering***

Model selection for K-means

How to choose K ?

Let $W_K(C^*)$ be the within-cluster scatter based on K clusters



If the "right" number of clusters is K^*, we expect

- for $K < K^*$, $W_K(C^*) - W_{K-1}(C^*)$ will be ***large***
- for $K > K^*$, $W_K(C^*) - W_{K-1}(C^*)$ will be ***small***

This suggests choosing K to be near the "knee" of the curve

Beyond K-means clustering

In K-means clustering, by measuring the within-cluster scatter as

$$W(C) = \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

we are implicitly assuming that each cluster is roughly spherical in shape

This is probably unrealistic

But if we don't even know what the clusters are

we definitely don't know how they are shaped…

Clustering with Gaussian mixture models

One way to extend the basic idea behind K-means clustering to allow for more general cluster shapes is to assume

- the clusters are ***elliptical***
- each cluster can be modeled using a ***multivariate Gaussian density***
- the full data set is modeled using a ***Gaussian mixture model*** (GMM)

We can then perform clustering based on a maximum likelihood estimation of the GMM

Gaussian mixture models

Recall the multivariate Gaussian density

$$\phi(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{d}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

where $\mathbf{x} \in \mathbb{R}^d, \boldsymbol{\mu} \in \mathbb{R}^d, \boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}, \boldsymbol{\Sigma} \succeq 0$

A random variable X follows a ***Gaussian mixture model*** if its density has the form

$$f(\mathbf{x}) = \sum_{k=1}^{K} w_k \phi(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

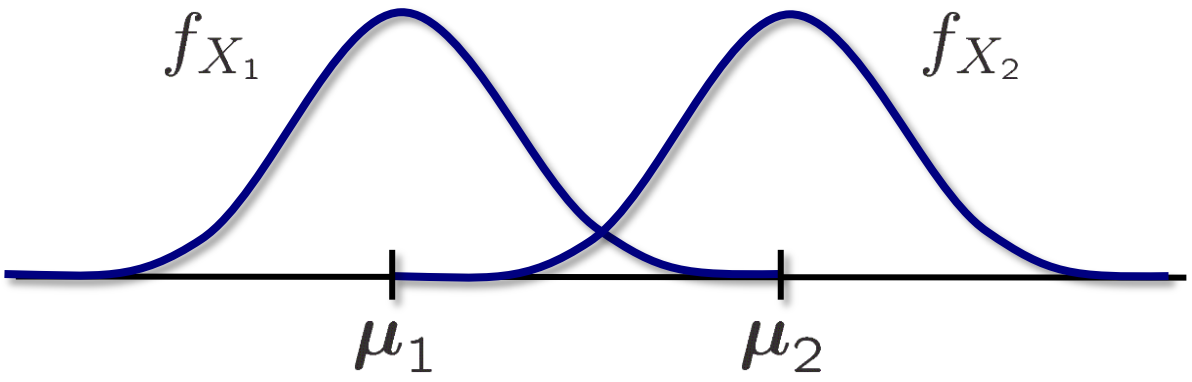
where $0 < w_k < 1, \sum_{k=1}^{K} w_k = 1$

$$\boldsymbol{\mu}_k \in \mathbb{R}^d$$

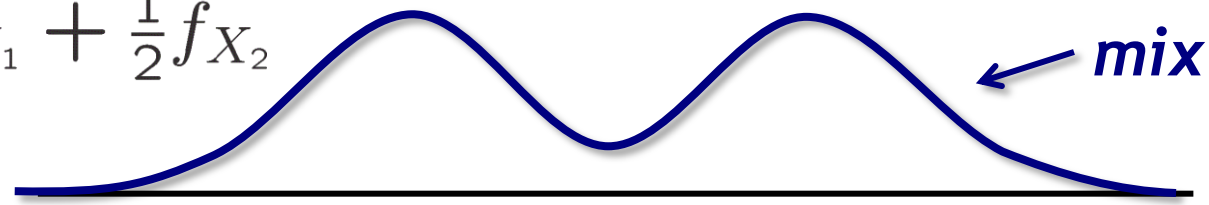
$$\boldsymbol{\Sigma}_k \in \mathbb{R}^{d \times d}, \boldsymbol{\Sigma}_k \succeq 0$$

Example

$X_1 \sim \mathcal{N}(\mu_1, \sigma^2)$ $X_2 \sim \mathcal{N}(\mu_2, \sigma^2)$

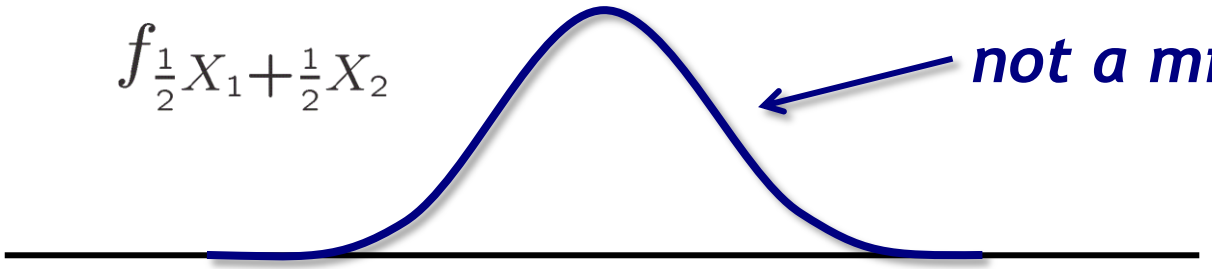


$\frac{1}{2}f_{X_1} + \frac{1}{2}f_{X_2}$



mixture

$f_{\frac{1}{2}X_1+\frac{1}{2}X_2}$



not a mixture

Simulating a GMM

Suppose we know

$$\theta = (w_1, \ldots, w_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_K)$$

How can we simulate a realization of the GMM?

Basic idea

1. Choose one of the "components" at random, weighted according to w_k
   - i.e., $\mathbb{P}[\text{picking component } k] = w_k$
2. Draw a realization from $\mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$

Why does this work?

Simulating a GMM

Let $S \in \{1, \ldots, K\}$ be a discrete random variable such that

$$\mathbb{P}[S = k] = w_k$$

Generate X as follows:

1. Generate a realization s of S
2. Generate $X \sim \mathcal{N}(\boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$

The density of X generated this way is

$$f(\mathbf{x}) = \sum_{k=1}^{K} f(\mathbf{x}|S = k) \cdot \mathbb{P}[S = k]$$

$$= \sum_{k=1}^{K} w_k \phi(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

GMMs for clustering

The variable S is called a (hidden) ***state variable***

We can imagine that every realization from a GMM is actually associated with a (hidden) realization of the state variable S

In the context of clustering, our objective is to estimate $\boldsymbol{\theta}$ and define clusters in terms of the estimate GMM

That is, we assume

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \sim f(\mathbf{x}; \boldsymbol{\theta})$$

and reduce clustering to a ***parameter estimation*** problem

Of course, we have to do all of this without observing the hidden states $s_1, \ldots, s_n$ associated with $\mathbf{x}_1, \ldots, \mathbf{x}_n$

Maximum likelihood estimation

We will approach this problem from the perspective of ***maximum likelihood estimation***

- $f(\mathrm{x};\boldsymbol{\theta})$: arbitrary density parameterized by $\boldsymbol{\theta}$
- n iid realizations $\mathrm{x}_1,\ldots,\mathrm{x}_n \sim f(\mathrm{x};\boldsymbol{\theta})$ denoted by $\mathbf{X} = (\mathrm{x}_1,\ldots,\mathrm{x}_n)$

The ***likelihood function*** of $\boldsymbol{\theta}$ is

$$\mathcal{L}(\boldsymbol{\theta};\mathbf{X}) := f(\mathbf{X};\boldsymbol{\theta}) = \prod_{i=1}^{n} f(\mathrm{x}_i;\boldsymbol{\theta})$$

The ***maximum likelihood estimator (MLE)*** is

$$\widehat{\boldsymbol{\theta}} = \widehat{\boldsymbol{\theta}}(\mathbf{X}) := \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta};\mathbf{X})$$

Example

Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ ➡ $\boldsymbol{\theta} = (\boldsymbol{\mu}, \Sigma)$

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{X}) = \prod_{i=1}^{n} \frac{e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_i - \boldsymbol{\mu})}}{\sqrt{(2\pi)^d |\Sigma|}}$$

Computationally, it is more convenient to maximize the ***log-likelihood***

$$\ell(\boldsymbol{\theta}; \mathbf{X}) = \log \mathcal{L}(\boldsymbol{\theta}; \mathbf{X})$$

$$= \sum_{i=1}^{n} -\frac{1}{2} [d \log(2\pi) + \log |\Sigma| + (\mathbf{x}_i - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_i - \boldsymbol{\mu})]$$

Example

By taking the gradient of $\ell(\boldsymbol{\theta}; \mathbf{X})$ and setting this to zero, we obtain that the MLE of the parameters is given by

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

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \widehat{\boldsymbol{\mu}})(\mathbf{x}_i - \widehat{\boldsymbol{\mu}})^T$$

MLE for a GMM

Now consider a GMM and assume that K is known

The likelihood function is

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{X}) = \prod_{i=1}^{n} f(\mathbf{x}_i; \boldsymbol{\theta}) = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} w_k \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

and the log likelihood is

$$\ell(\boldsymbol{\theta}; \mathbf{X}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} w_k \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

There is (unfortunately) no known closed-form maximizer

State variables

Just for kicks, let's suppose that we actually also had access to realizations of the state variables associated with $\mathbf{X}$

Notation: $\mathbf{s} = (s_1, \ldots, s_n),\ I_k = \{i : s_i = k\},\ n_k = |I_k|$

The likelihood function is given by

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{s}) &= \prod_{i=1}^{n} f(\mathbf{x}_i|\boldsymbol{\theta}, s_i) \cdot \mathbb{P}[S_i = s_i|\boldsymbol{\theta}] \\
&= \prod_{i=1}^{n} w_{s_i} \phi(\mathbf{x}_i; \boldsymbol{\mu}_{s_i}, \boldsymbol{\Sigma}_{s_i}) \\
&= \prod_{k=1}^{K} w_k^{n_k} \prod_{k=1}^{K} \prod_{i \in I_k} \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)
\end{aligned}$$

MLE with state variables

In this case the log-likelihood is given by

$$\ell(\boldsymbol{\theta}; \mathbf{X}, \mathbf{s}) = \sum_{k=1}^{K} n_k \log w_k + \sum_{k=1}^{K} \sum_{i \in I_k} \log \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Thus, we can maximize with respect to each $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ independently from the rest of the components

$$\widehat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i \in I_k} \mathbf{x}_i$$

$$\widehat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k} \sum_{i \in I_k} (\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)^T$$

MLE with state variables

To solve

$$\max_{\mathbf{w}} \sum_{k=1}^{K} n_k \log w_k$$

$$\text{s.t.} \sum_{k=1}^{K} w_k = 1$$

we can use Lagrange multipliers

The Lagrangian is

$$L(\mathbf{w}, \lambda) = \sum_{k=1}^{K} n_k \log w_k + \lambda \left(\sum_{k=1}^{K} w_k - 1 \right)$$

MLE with state variables

$$L(\mathbf{w}, \lambda) = \sum_{k=1}^{K} n_k \log w_k + \lambda \left(\sum_{k=1}^{K} w_k - 1 \right)$$

$$\rightarrow \quad \frac{\partial L(\mathbf{w}, \lambda)}{\partial w_k} = \frac{n_k}{w_k} + \lambda = 0 \quad \rightarrow \quad w_k = -\frac{n_k}{\lambda}$$

$$\rightarrow \quad \sum_{k=1}^{K} \left(-\frac{n_k}{\lambda} \right) = 1 \quad \rightarrow \quad \lambda = -\sum_{k=1}^{K} n_k = -n$$

$$\rightarrow \quad \widehat{w}_k = \frac{n_k}{n}$$

Incomplete data

The combined data $(\mathbf{X}, \mathbf{s})$ is sometimes called the ***"complete data"***

In general, "complete data" usually refers to the combination of what we observe together with the unobserved data that we would need in order to make the MLE tractable

Unfortunately, we never have the complete data, and so we must figure out how to infer the "missing data" in order calculate the MLE

More on how to do this next time…

Lecture 21: Expectation Maximization (EM) and Kernel Density Estimation (KDE)

K-means clustering algorithm

Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

The K-means algorithm tries to solve

$$\min_{C, \{\mathbf{m}_k\}_{k=1}^K} \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \mathbf{m}_k\|^2$$

Algorithm

Initialize $\mathbf{m}_k, k = 1, \ldots, K$

Repeat until clusters don't change

- $C(i) = \arg\min \|\mathbf{x}_i - \mathbf{m}_k\|^2$
- $\mathbf{m}_k = \frac{1}{n_k} \sum_{i:C(i)=k}^{k} \mathbf{x}_i$

Beyond K-means clustering

In K-means clustering, by measuring the within-cluster scatter as

$$W(C) = \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

we are implicitly assuming that each cluster is roughly spherical in shape

This is probably unrealistic

But if we don't even know what the clusters are

we definitely don't know how they are shaped…

Clustering with Gaussian mixture models

One way to extend the basic idea behind K-means clustering to allow for more general cluster shapes is to assume

- the clusters are ***elliptical***
- each cluster can be modeled using a ***multivariate Gaussian density***
- the full data set is modeled using a ***Gaussian mixture model*** (GMM)

We can then perform clustering based on a maximum likelihood estimation of the GMM

Gaussian mixture models

Recall the multivariate Gaussian density

$$\phi(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{d}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

where $\mathbf{x} \in \mathbb{R}^d, \boldsymbol{\mu} \in \mathbb{R}^d, \boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}, \boldsymbol{\Sigma} \succeq 0$

A random variable X follows a ***Gaussian mixture model*** if its density has the form

$$f(\mathbf{x}) = \sum_{k=1}^{K} w_k \phi(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where $0 < w_k < 1, \sum_{k=1}^{K} w_k = 1$

$$\boldsymbol{\mu}_k \in \mathbb{R}^d$$

$$\boldsymbol{\Sigma}_k \in \mathbb{R}^{d \times d}, \boldsymbol{\Sigma}_k \succeq 0$$

MLE with state variables

In this case the log-likelihood is given by

$$\ell(\boldsymbol{\theta}; \mathbf{X}, \mathbf{s}) = \sum_{k=1}^{K} n_k \log w_k + \sum_{k=1}^{K} \sum_{i \in I_k} \log \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

And the MLE is given by

$$\widehat{w}_k = \frac{n_k}{n}$$

$$\widehat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i \in I_k} \mathbf{x}_i$$

$$\widehat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k} \sum_{i \in I_k} (\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k)^T$$

MLE with "incomplete data"

Define the "indicator" variable

$$\Delta_{i,k} = \begin{cases} 1 & \text{if } s_i = k \\ 0 & \text{if } s_i \neq k \end{cases}$$

The ***complete data log-likelihood*** can be written as

$$\ell(\boldsymbol{\theta}; \mathbf{X}, \mathbf{s}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \Delta_{i,k} \cdot w_k \cdot \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{K} \Delta_{i,k} \left(\log w_k + \log \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

Expectation-Maximization (EM)

The ***expectation-maximization (EM)*** algorithm is an iterative algorithm that produces a sequence $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ of parameter estimates

E-step

Given $\boldsymbol{\theta}^{(j)}$, compute the ***expected complete data log-likelihood***:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) = \mathbb{E}_{\mathbf{S}|\mathbf{X}} \left[\ell(\boldsymbol{\theta}; \mathbf{X}, \mathbf{S}) | \mathbf{X}; \boldsymbol{\theta}^{(j)} \right]$$

In our case

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) = \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{i,k}(\boldsymbol{\theta}^{(j)}) \left(\log w_k + \log \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

where

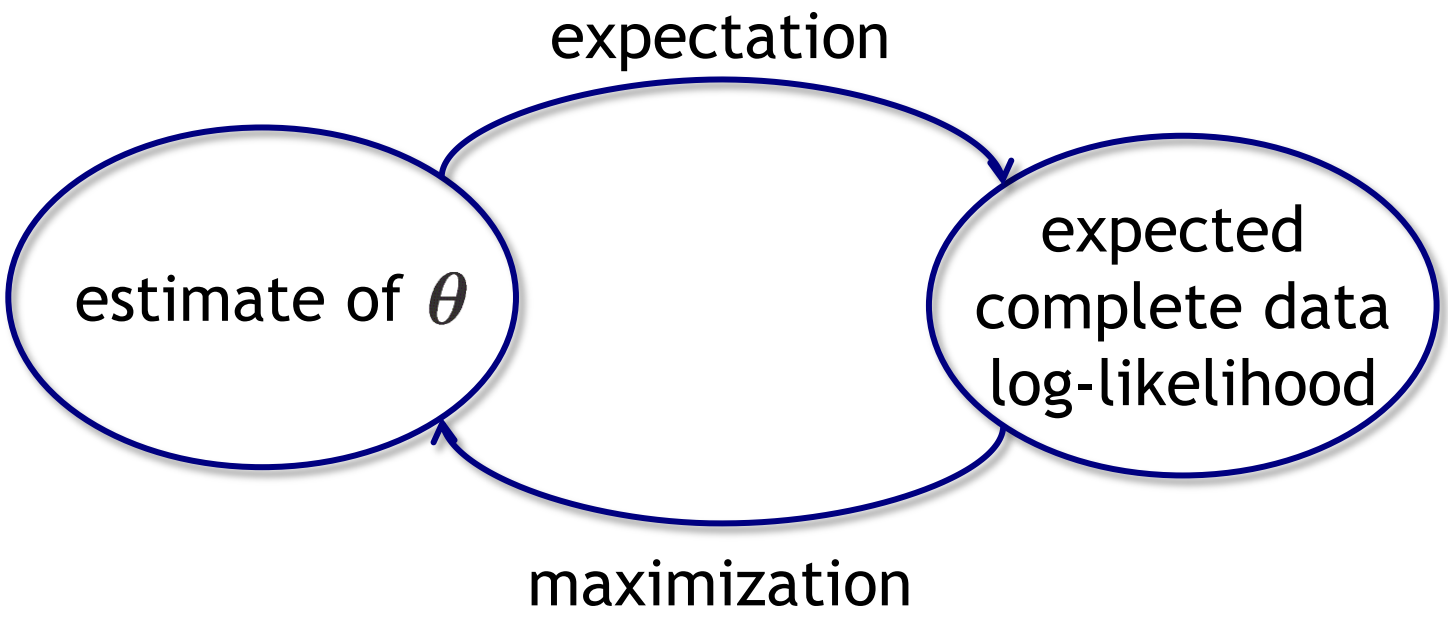
$$\gamma_{i,k}(\boldsymbol{\theta}^{(j)}) := \mathbb{E}_{\mathbf{S}|\mathbf{X}} \left[\Delta_{i,k} | \mathbf{X}; \boldsymbol{\theta}^{(j)} \right] = \mathbb{P} \left[S_i = k | \mathbf{X}; \boldsymbol{\theta}^{(j)} \right]$$

Expectation-Maximization (EM)

M-step

Given the expected complete data log-likelihood, compute the maximum likelihood estimate

$$\boldsymbol{\theta}^{(j+1)} = \arg\max_{\boldsymbol{\theta}} \; Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)})$$



EM algorithm for GMMs

Initialize $\boldsymbol{\theta}^{(0)}$

Repeat

E-Step: Compute

$$\gamma_{i,k}^{(j)} := \gamma_{i,k}(\boldsymbol{\theta}^{(j)}) = \mathbb{E}_{\mathbf{S}|\mathbf{X}}\left[\Delta_{i,k}|\mathbf{X};\boldsymbol{\theta}^{(j)}\right]$$

and form

$$Q(\boldsymbol{\theta},\boldsymbol{\theta}^{(j)}) = \mathbb{E}_{\mathbf{S}|\mathbf{X}}\left[\ell(\boldsymbol{\theta};\mathbf{X},\mathbf{S})|\mathbf{X};\boldsymbol{\theta}^{(j)}\right]$$

M-Step: Compute

$$\boldsymbol{\theta}^{(j+1)} = \arg\max_{\boldsymbol{\theta}} \; Q(\boldsymbol{\theta},\boldsymbol{\theta}^{(j)})$$

Until termination criterion satisfied

M-step for GMMs

Maximizing

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(j)}) = \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{i,k}^{(j)} \left(\log w_k + \log \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

with respect to $\boldsymbol{\theta} = (\{w_k\}, \{\boldsymbol{\mu}_k\}, \{\boldsymbol{\Sigma}_k\})$ yields

$$\widehat{w}_k = \frac{\sum_{i=1}^{n} \gamma_{i,k}^{(j)}}{n}$$

$$\widehat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{n} \gamma_{i,k}^{(j)} \mathbf{x}_i}{\sum_{i=1}^{n} \gamma_{i,k}^{(j)}}$$

$$\widehat{\boldsymbol{\Sigma}}_k = \frac{\sum_{i=1}^{n} \gamma_{i,k}^{(j)} (\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k^{(j+1)})(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k^{(j+1)})^T}{\sum_{i=1}^{n} \gamma_{i,k}^{(j)}}$$

E-step for GMMs

In the E-step we need to compute

$$\gamma_{i,k}^{(j)} = \mathbb{E}_{\mathbf{S}|\mathbf{X}}\left[\Delta_{i,k}|\mathbf{X};\boldsymbol{\theta}^{(j)}\right]$$

$$= \mathbb{P}\left[\Delta_{i,k}=1|\mathbf{X};\boldsymbol{\theta}^{(j)}\right]$$

$$= \mathbb{P}\left[S_i=k|\mathbf{x}_i;\boldsymbol{\theta}^{(j)}\right]$$

Bayes' rule

$$= \frac{f(\mathbf{x}_i|S_i=k;\boldsymbol{\theta}^{(j)})\mathbb{P}\left[S_i=k;\boldsymbol{\theta}^{(j)}\right]}{f(\mathbf{x}_i;\boldsymbol{\theta}^{(j)})}$$

$$= \frac{w_k^{(j)}\phi(\mathbf{x}_i;\boldsymbol{\mu}_k^{(j)},\boldsymbol{\Sigma}_k^{(j)})}{\sum_{\ell=1}^{K} w_\ell^{(j)}\phi(\mathbf{x}_i;\boldsymbol{\mu}_\ell^{(j)},\boldsymbol{\Sigma}_\ell^{(j)})}$$

Initialization and termination

In general, the likelihood has many local maxima, so a good initialization of the algorithm is critical

A good initialization for EM in the case of GMM is

$$w_k^{(0)} = \frac{1}{K}$$

$$\boldsymbol{\mu}_k^{(0)} = \text{ a randomly selected } \mathbf{x}_i$$

(sampled without replacement)

$$\boldsymbol{\Sigma}_k^{(0)} = \text{ the sample covariance}$$

Possible termination criteria are to stop iterating when

$$|\ell(\boldsymbol{\theta}^{(j+1)}; \mathbf{X}) - \ell(\boldsymbol{\theta}^{(j)}; \mathbf{X})| \leq \epsilon$$

or

$$|Q(\boldsymbol{\theta}^{(j+1)}, \boldsymbol{\theta}^{(j)}) - Q(\boldsymbol{\theta}^{(j)}, \boldsymbol{\theta}^{(j)})| \leq \epsilon$$

Defining clusters

Recall that

$$\gamma_{i,k}(\boldsymbol{\theta}^{(j)}) = \mathbb{P}\left[S_i = k|\mathbf{X};\boldsymbol{\theta}^{(j)}\right]$$

A reasonable "hard" assignment of points to clusters is given by

$$C(i) = \underset{k=1,\ldots,K}{\arg\max}\, \gamma_{i,k}(\widehat{\boldsymbol{\theta}})$$

Alternatively, one may simply take $\gamma_{i,k}(\widehat{\boldsymbol{\theta}})$ as a "soft" assignment that expresses the "affinity" of $\mathbf{x}_i$ for cluster k

EM in general

Nothing in the EM algorithm as we have stated it is specific to GMMs

EM is actually an extremely general algorithm for computing ML/MAP estimators

Applies whenever having knowledge of certain "hidden variables" renders an ML/MAP estimator tractable

See *Statistical Analysis with Missing Data* by Little and Rubin (2002) for an in-depth discussion of other applications

Convergence of EM

Theorem

For each $j = 1, 2, \ldots$

$$\ell(\boldsymbol{\theta}^{(j+1)}; \mathbf{X}) \geq \ell(\boldsymbol{\theta}^{(j)}; \mathbf{X})$$

A proof based on Jensen's inequality is available in Hastie, Tibshirani, and Friedman

The convergence rate is also of interest

It is typically linear, but with a rate that depends on the proportion of observed data

Connection to K-means

Consider a GMM where each $\Sigma_k = \sigma^2 \mathbf{I}$ for some fixed σ^2

The EM algorithm for computing the MLE of $\{w_k\}_{k=1}^K$ and $\{\boldsymbol{\mu}_k\}_{k=1}^K$ is to iterate

$$\gamma_{i,k} = \frac{w_k \phi(\mathbf{x}_i; \boldsymbol{\mu}_k, \sigma^2 \mathbf{I})}{\sum_{\ell=1}^K w_\ell \phi(\mathbf{x}_i; \boldsymbol{\mu}_\ell, \sigma^2 \mathbf{I})}$$

$$w_k = \frac{1}{n} \sum_{i=1}^n \gamma_{i,k}$$

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

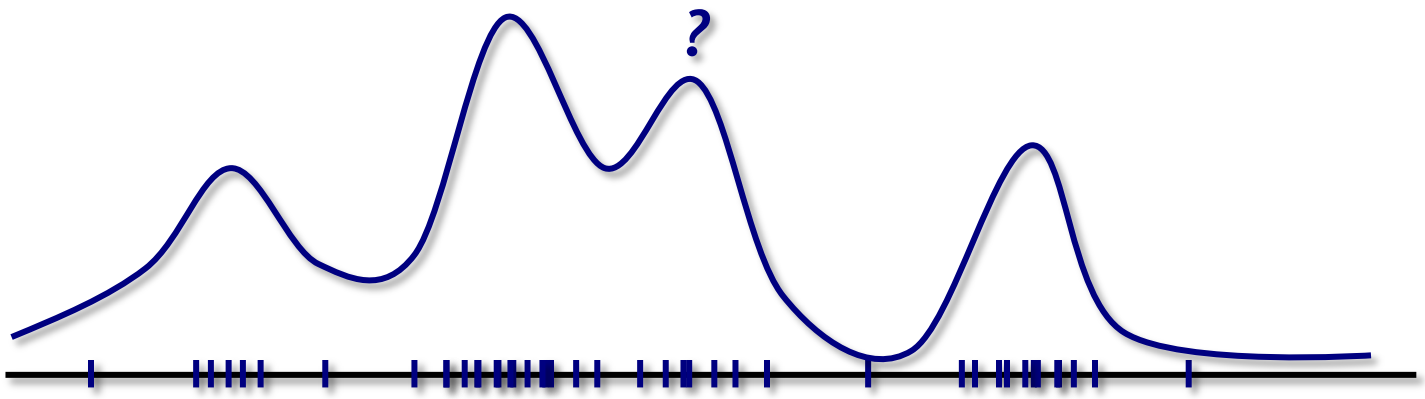
When $\sigma^2 \to 0$, $\gamma_{i,k} \to \begin{cases} 1 & \text{if } k = \arg\min_\ell \|\mathbf{x}_i - \boldsymbol{\mu}_\ell\| \\ 0 & \text{otherwise} \end{cases}$

so the algorithm reduces to K-means

Density estimation

In density estimation problems, we are given a random sample $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ from an unknown density $f(\mathbf{x})$

Our objective is to estimate $f(\mathbf{x})$



Applications

Classification

- If we estimate the density for each class, we can simply plug this in to the formula for the Bayes' classifier
- Density estimation (for each feature) is the key component in Naïve Bayes

Clustering

- Clusters can be defined by the density: given a point $\mathbf{x}$, climb the density until you reach a local maximum

Anomaly detection

- Given a density estimate $\widehat{f}(\mathbf{x})$, we can use the test

$$\widehat{f}(\mathbf{x}) \leqslant \gamma$$

to detect anomalies in future observations

Kernel density estimation

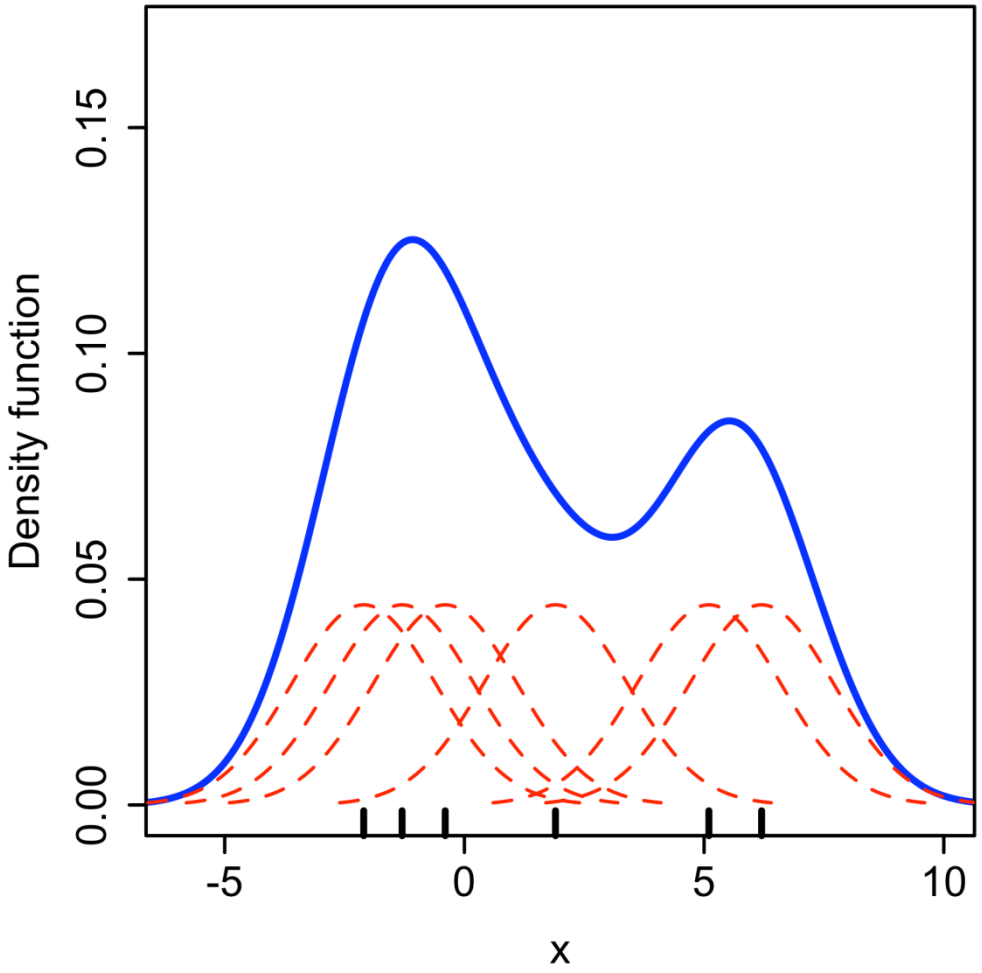
A ***kernel density estimate*** has the form

$$\widehat{f}(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^{n} k_\sigma(\mathbf{x} - \mathbf{x}_i)$$

where k_σ is called a ***kernel***

- A kernel density estimate is ***nonparametric***
- Another name for this is the ***Parzen window method***
- The σ parameter is called the ***bandwidth***
- Looks just like kernel ridge regression, but with equal weights
- Note that k_σ does not necessarily need to be an inner product kernel

Example



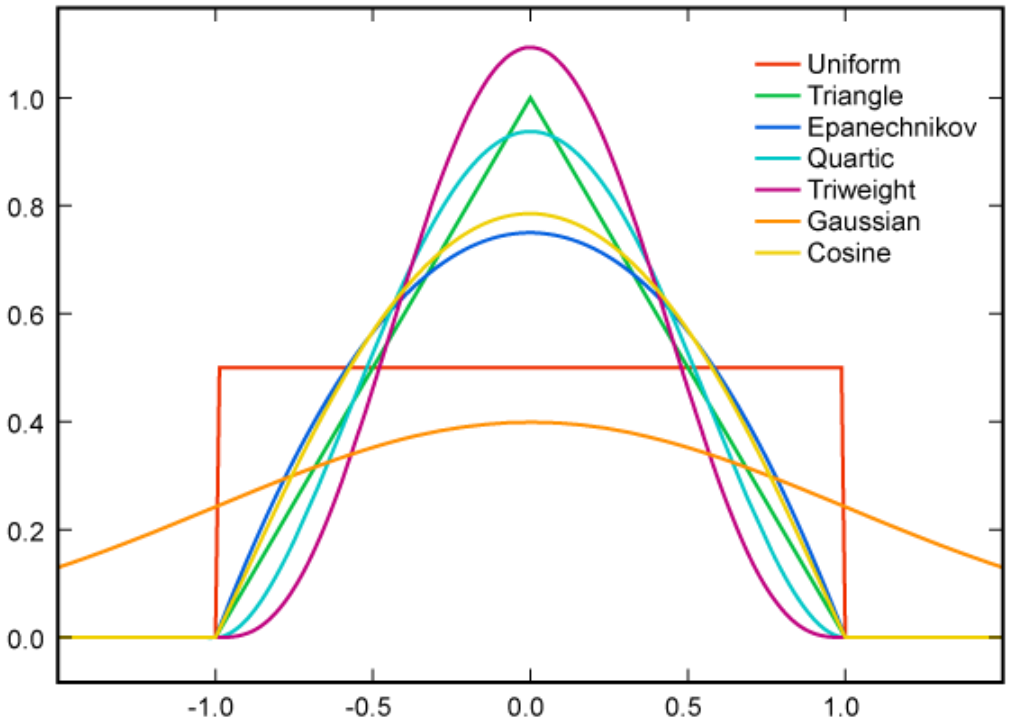
Kernels

In the context of density estimation, a kernel should satisfy

1. $\int k_\sigma(y)dy = 1$
2. $k_\sigma(y) \geq 0$
3. $k_\sigma(y) = \frac{1}{\sigma^d} D(\frac{\|y\|}{\sigma})$ for some D

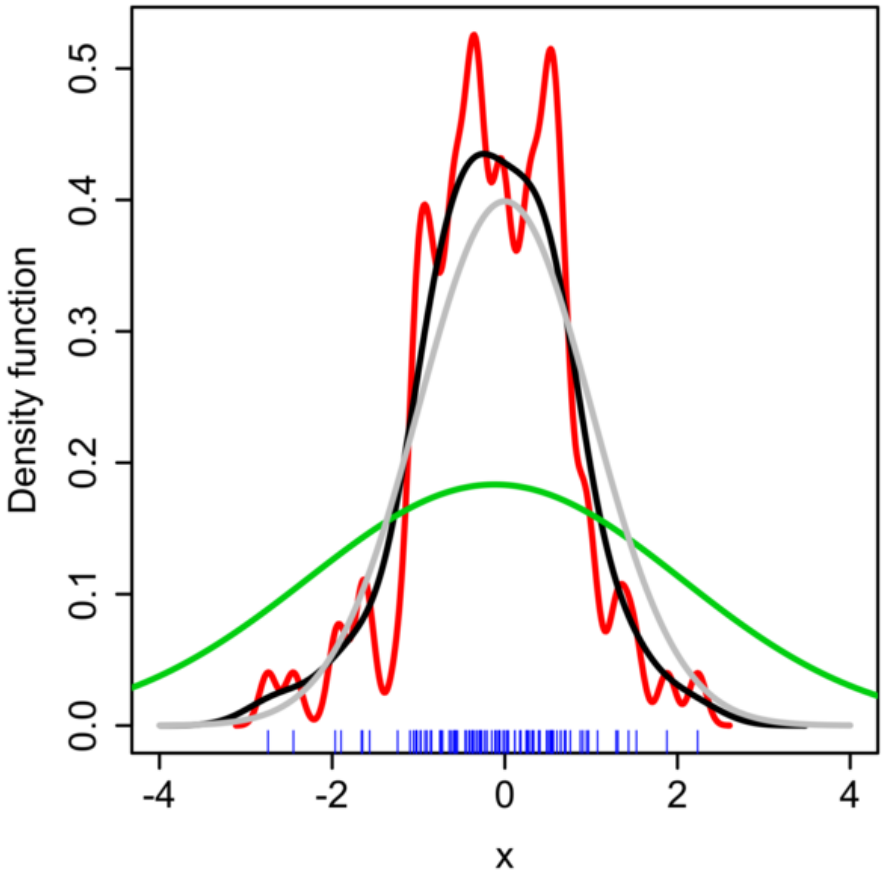
Examples (in $\mathbb{R}$)

- Uniform kernel
- Triangular kernel
- Epanichnikov kernel
- Gaussian
- …



Kernel bandwidth

The accuracy of a kernel density estimate depends critically on the bandwidth



Setting the bandwidth - Theory

Theorem

Let $\widehat{f}_\sigma(\mathbf{x})$ be a kernel density estimate based on the kernel k_σ

Suppose $\sigma = \sigma_n$ is such that

- $\sigma_n \to 0$ as $n \to \infty$
- $n\sigma_n^d \to \infty$ as $n \to \infty$

Then

$$\mathbb{E}\left[\int |\widehat{f}_\sigma(\mathbf{x}) - f(\mathbf{x})| d\mathbf{x}\right] \to 0$$

as $n \to \infty$, regardless of the true density $f(\mathbf{x})$

Proof: See Devroye and Lugosi, *Combinatorial Methods in Density Estimation* (1987)

Setting the bandwidth - Practice

Silverman's rule of thumb

If using the Gaussian kernel, a good choice for σ is

$$\sigma \approx 1.06 \widehat{\sigma} n^{-1/5}$$

where $\widehat{\sigma}$ is the standard deviation of the samples

How can we apply what we know about model selection to setting σ?

- Randomly split the data into two sets
- Obtain a kernel density estimate for the first
- Measure how well the second set fits this estimate
  - e.g., compute another kernel density estimate on the second set and calculate the KL divergence between the two
- Repeat over many random splits and average

Lecture 22: Matrix Completion and Robust PCA

Principal component analysis

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ denote a $d \times n$ "data matrix" whose columns are given by the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

Recall that we can compute the principal components simply by computing the singular value decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

The principal eigenvectors are the first k columns of $\mathbf{U}$

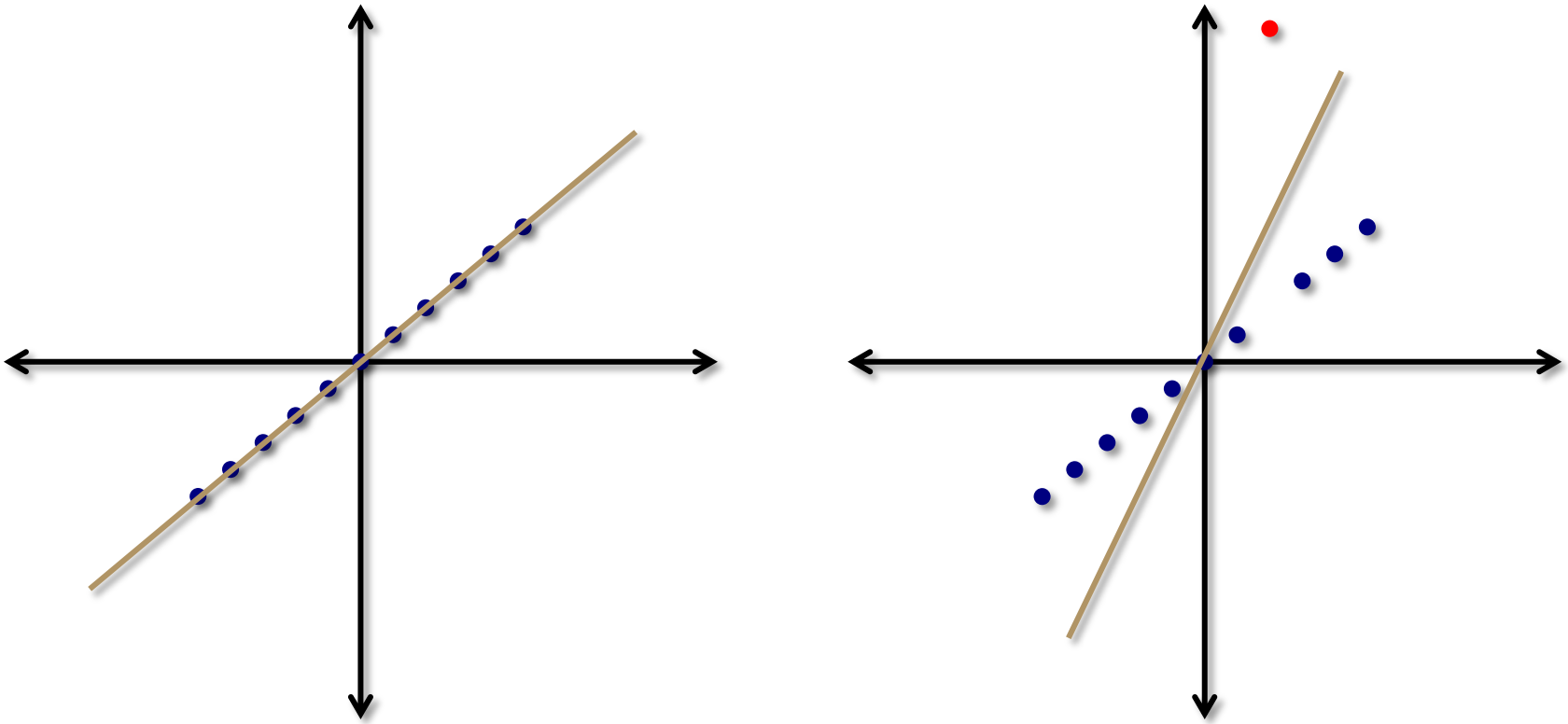
The inherent assumption in PCA is that $\mathbf{X}$ is (at least approximately) low rank

- i.e., most of the singular values are close to zero

Outliers

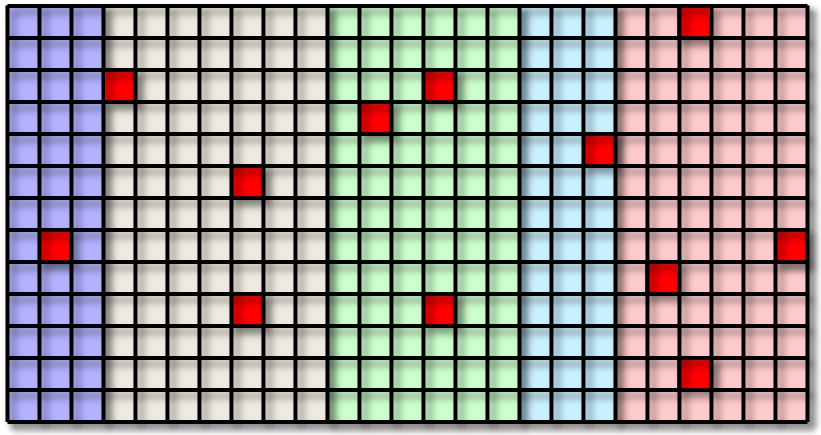
What happens if there are outliers in our data set?

PCA is ***extremely sensitive*** to outliers/corruptions



Modeling outliers

In the presence of outliers, our data matrix $\mathbf{X}$ is no longer low-rank because some of the entries have been corrupted



$$\mathbf{X} = \underbrace{\mathbf{L}}_{\text{low-rank}} + \underbrace{\mathbf{S}}_{\text{corruptions}}$$

Robust PCA

The problem of robust PCA boils down to a separation problem

Given $\mathbf{X} = \mathbf{L} + \mathbf{S}$, determine $\mathbf{L}$ (and hence $\mathbf{S}$)

Is this possible?

Example

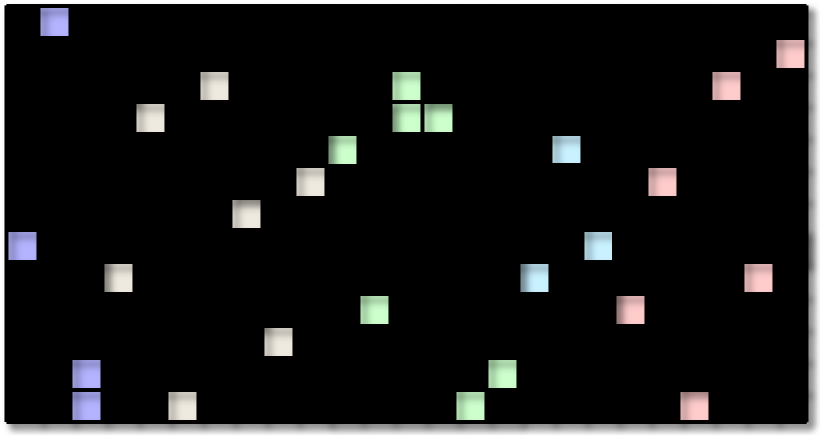
If I tell you $3 = x + y$, what are x and y?

We need to make some ***assumptions*** on $\mathbf{L}$ and $\mathbf{S}$ to make this problem tractable

- $\mathbf{L}$ is ***low-rank***
- $\mathbf{S}$ is ***sparse***

Missing data

Another (closely related) problem that arises in practice is that we may be missing some (or most) of the entries in our matrix



Given a set Ω of indices, we get to observe $\mathbf{Y} = \mathbf{X}_\Omega$, and we would like to recover $\mathbf{X}$

- also known as ***matrix completion***
- really just a special case of robust PCA

How to perform separation/recovery?

Robust PCA

$$\min_{\mathbf{L},\mathbf{S}} \ \text{rank}(\mathbf{L}) + \lambda \|\mathbf{S}\|_0$$
$$\text{s.t. } \mathbf{L} + \mathbf{S} = \mathbf{X}$$

Matrix completion

$$\min_{\mathbf{X}} \ \text{rank}(\mathbf{X})$$
$$\text{s.t. } \mathbf{X}_\Omega = \mathbf{Y}$$

What's the problem with these approaches?

- nonconvex, intractable, NP hard, etc.

Convex relaxation

We have already seen that an effective proxy for $\|\mathbf{S}\|_0$ that encourages sparsity is the "convex relaxation"

$$\|\mathbf{S}\|_1 := \sum_{i,j} |S_{i,j}|$$

What about a convex relaxation of $\text{rank}(\mathbf{L})$?

Observe that if we let $\boldsymbol{\sigma}$ denote the vector containing the singular values of $\mathbf{L}$, then $\text{rank}(\mathbf{L}) = \|\boldsymbol{\sigma}\|_0$

What if we replace $\text{rank}(\mathbf{L})$ with $\|\mathbf{L}\|_* = \|\boldsymbol{\sigma}\|_1$?

It turns out that $\|\mathbf{L}\|_*$ is indeed a convex function, and hence something we can potentially optimize efficiently

- typically called the ***nuclear norm*** or ***Shatten 1-norm***
- we can write $\|\mathbf{L}\|_* = \text{trace}(\sqrt{\mathbf{L}^T\mathbf{L}})$

Convex programs for separation/recovery

Robust PCA

$$\min_{\mathbf{L},\mathbf{S}} \ \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1$$

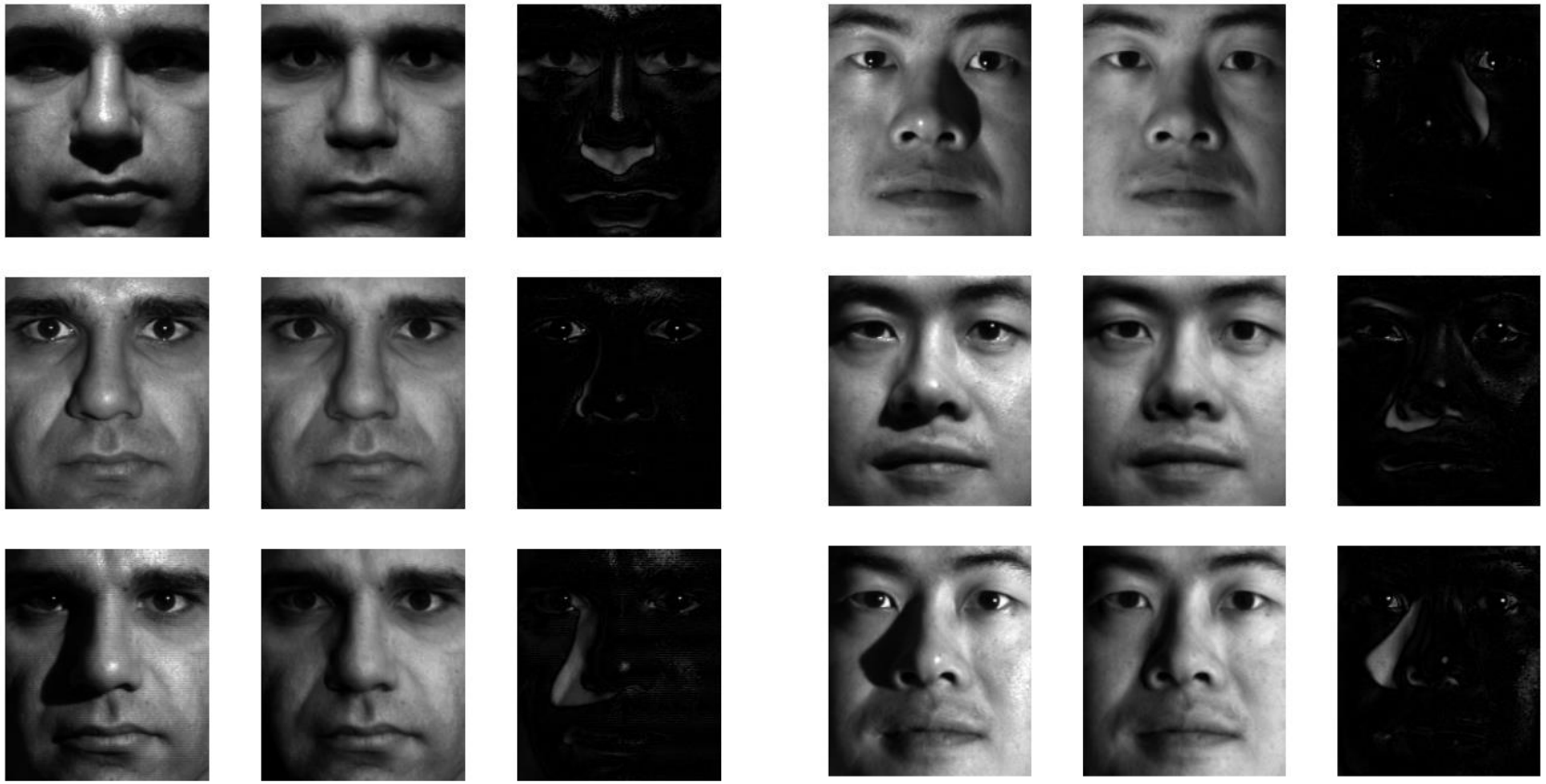
$$\text{s.t. } \mathbf{L} + \mathbf{S} = \mathbf{X}$$

Matrix completion

$$\min_{\mathbf{X}} \ \|\mathbf{X}\|_*$$

$$\text{s.t. } \mathbf{X}_\Omega = \mathbf{Y}$$

Application: Removing face illumination



[Candès et al., 2009]

Application: Background subtraction

X L S

[Candès et al., 2009]

Application: Collaborative filtering

The "Netflix Problem"

$X_{i,j} =$ how much user i likes movie j

Rank 1 model: $u_i =$ how much user i likes romantic movies

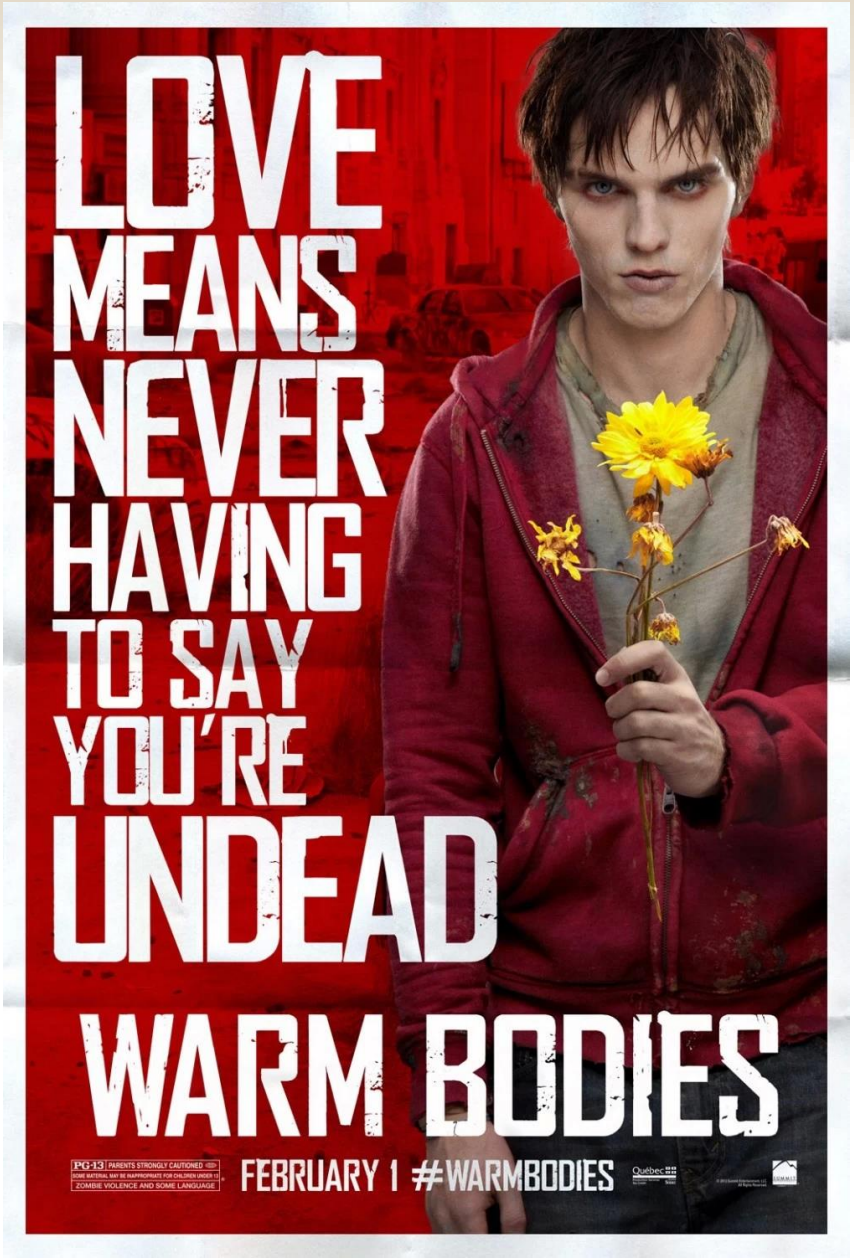
$v_j =$ amount of romance in movie j

$X_{i,j} = u_i v_j$

Rank 2 model: $w_i =$ how much user i likes zombie movies

$x_j =$ amount of zombies in movie j

$X_{i,j} = u_i v_j + w_i x_j$


LOVE MEANS NEVER HAVING TO SAY YOU'RE UNDEAD
WARM BODIES
PG-13 PARENTS STRONGLY CAUTIONED
SOME MATERIAL MAY BE INAPPROPRIATE FOR CHILDREN UNDER 13
ZOMBIE VIOLENCE AND SOME LANGUAGE
FEBRUARY 1 #WARMBODIES

Application: Response data

More generally, we might have

$$X_{i,j} = \text{response from person } i \text{ to question } j$$

Low-rank models are commonly applied to all kinds of "response" data

- surveys (census)
- standardized tests
- market research

If we have lots of questions and lots of participants

- we might not be able to ask all participants all possible questions
- we might need to deal with malicious responses

Application: Multidimensional scaling

Suppose that $\mathbf{X}$ represents the pairwise distances between a set of objects, i.e.,

$$X_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

Recall from before that in this case $\mathrm{rank}(\mathbf{X}) = d + 2$ where d is the dimension of the $\mathbf{x}_i$

If we have a large number of objects, we may not be able to observe/measure all possible pairwise distances

If these distances are calculated from similarity judgments by people, we may have outliers/corruptions

When does it work?

In general, we cannot always guarantee that we can solve the separation/recovery problems

Examples where separation/recovery might be impossible:

- too many of the entries are corrupted
  - in this case our corruptions are not really sparse
- we do not observe enough of the entries
  - e.g., we do not even observe some rows/columns
- the low-rank portion of the matrix is also sparse
  - we will not be able to separate it from sparse corruptions
  - we might not observe any of the relevant entries

How much data do we need?

Consider the matrix completion problem

To recover a matrix with rank k, how many observations do we expect to need?

If $\mathbf{X}$ is $n_1 \times n_2$, then there are

$$n_1 k - \frac{k(k+1)}{2} + n_2 k - \frac{k(k+1)}{2} + k = k(n_1 + n_2 - k)$$

degrees of freedom

We can reasonably expect that we will need to have

$$|\Omega| \gtrsim Ck(n_1 + n_2)$$

In fact, we need a little bit more: $n_{\max} = \max(n_1, n_2)$

$$|\Omega| \gtrsim Ckn_{\max} \log(n_{\max})$$

Incoherence

We also need to avoid the case where $\mathbf{X}$ is sparse

We will assume that $\mathbf{X}$ satisfies the incoherence condition:

If $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$, the incoherence condition with parameter μ states that

$$\max_i \|\mathbf{U}^T \mathbf{e}_i\|^2 \leq \frac{\mu k}{n_1} \qquad \max_i \|\mathbf{V}^T \mathbf{e}_i\|^2 \leq \frac{\mu k}{n_2}$$

$$\|\mathbf{U}\mathbf{V}^T\|_\infty \leq \sqrt{\frac{\mu k}{n_1 n_2}}$$

Matrix completion

Theorem

Suppose that $\mathbf{X}$ has rank k and satisfies the incoherence condition. If we observe m entries of $\mathbf{X}$ sampled uniformly at random, then as long as

$$m \gtrsim C_\mu k n_{\max} \log^2(n_{\max})$$

we will recover $\mathbf{X}$ ***exactly*** with high probability

See papers by Recht, Candès, Tao, and many others

Robust PCA

Theorem

Suppose that $\mathbf{X} = \mathbf{L} + \mathbf{S}$ where $\mathbf{L}$ satisfies the incoherence property, $\text{rank}(\mathbf{L}) \leq C'_\mu n_{\text{min}} / \log^2(n_{\text{max}})$, and the support set of $\mathbf{S}$ is chosen uniformly at random from all sets of size $m \leq C' n_1 n_2$.

Then if we set $\lambda = 1/\sqrt{n_{\text{min}}}$ we achieve exact separation of $\mathbf{L}$ and $\mathbf{S}$ with high probability.

See papers by Candès, Li, Ma, and Wright, and others