

ColumbiaX: Machine Learning

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

ColumbiaX: Machine Learning

Lecture 1

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

OVERVIEW

This class will cover model-based techniques for extracting information from data with an end-task in mind. Such tasks include:

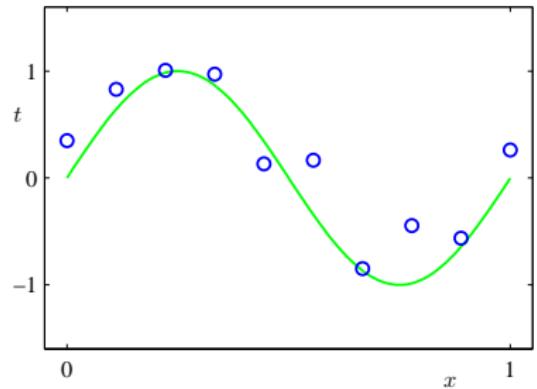
- ▶ predicting an unknown “output” given its corresponding “input”
- ▶ uncovering information within the data to better understand it
- ▶ data-driven recommendation, grouping, classification, ranking, etc.

There are a few ways we can divide up the material as we go along, e.g.,

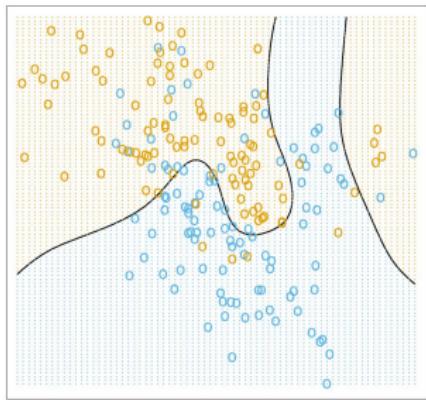
supervised learning		unsupervised learning
probabilistic models		non-probabilistic models
modeling approach		optimization techniques

We'll adopt the first method and work in the second two along the way.

OVERVIEW: SUPERVISED LEARNING



(a) Regression



(b) Classification

Regression: Using set of inputs, predict real-valued output.

Classification: Using set of inputs, predict a discrete label (aka class).

EXAMPLE CLASSIFICATION PROBLEM

Given a set of inputs characterizing an item, assign it a label.

Is this spam?

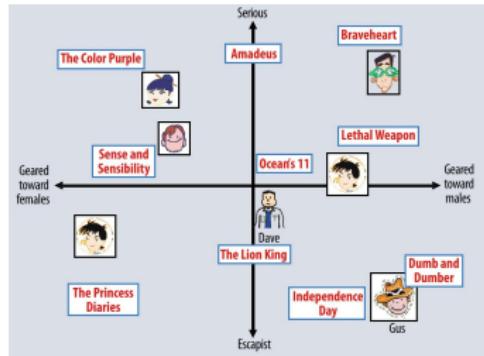
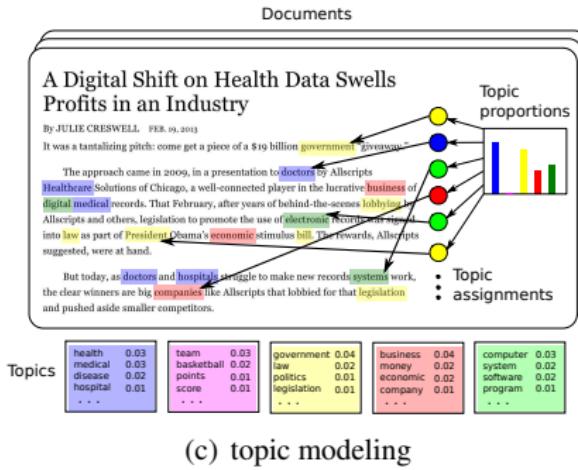
hi everyone,

i saw that close to my hotel there is a pub with bowling
(it's on market between 9th and 10th avenue). meet
there at 8:30?

What about this?

Enter for a chance to win a trip to Universal Orlando to
celebrate the arrival of Dr. Seuss's The Lorax on Movies
On Demand on August 21st! [Click here now!](#)

OVERVIEW: UNSUPERVISED LEARNING



With unsupervised learning our goal is often to uncover structure in the data. This helps with predictions, recommendations, efficient data exploration.

¹ Figure from Koren, Y., Robert B., and Volinsky, C. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

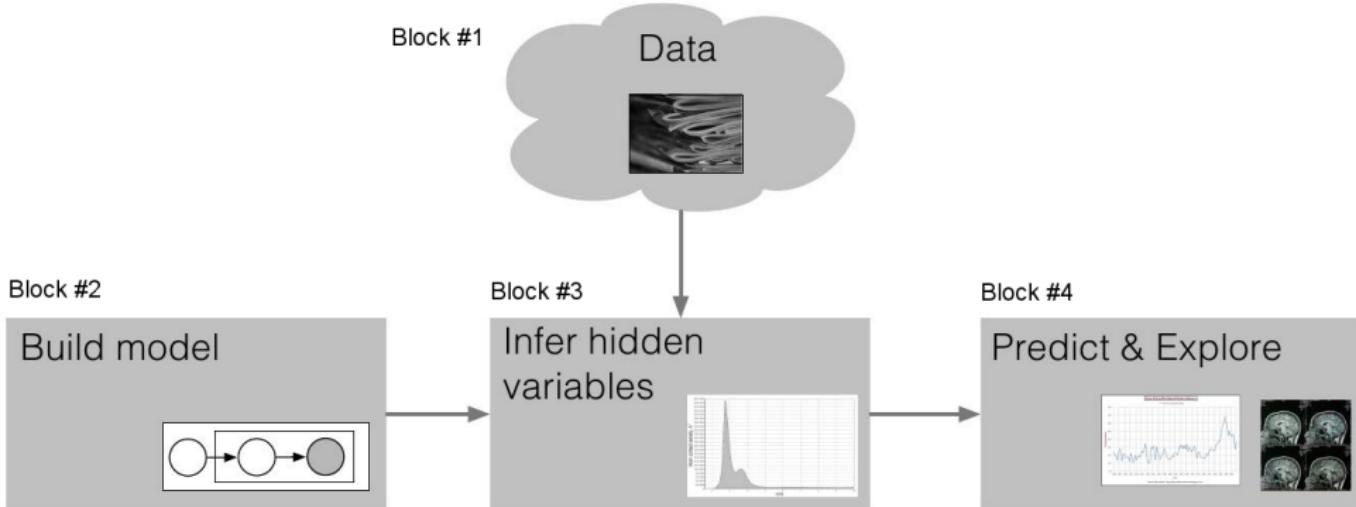
EXAMPLE UNSUPERVISED PROBLEM

Goal: Learn the dominant topics from a set of news articles.

The New York Times

music band songs rock album jazz pop song singer night	book life novel story books man stories love children family	art museum show exhibition artist artists paintings painting century works	game Knicks nets points team season play games night coach	show film television movie series says life man character know
theater play production show stage street broadway director musical directed	clinton bush campaign gore political republican dole presidential senator house	stock market percent fund investors funds companies stocks investment trading	restaurant sauce menu food dishes street dining dinner chicken served	budget tax governor county mayor billion taxes plan legislature fiscal

DATA MODELING

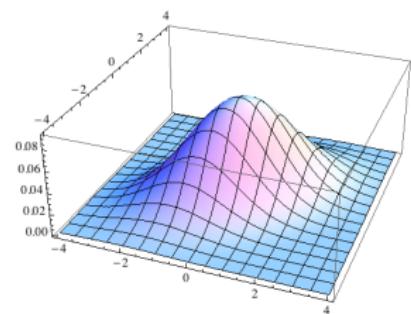


- ▶ Supervised vs. unsupervised: Blocks #1 and #4
- ▶ Probabilistic vs. non-probabilistic: Primarily Block #2 (Some Block #3)
- ▶ Model development (Block #2) vs. Optimization techniques (Block #3)

GAUSSIAN DISTRIBUTION (MULTIVARIATE)

Gaussian density in d dimensions

- ▶ Block #1: Data x_1, \dots, x_n . Each $x_i \in \mathbb{R}^d$
- ▶ Block #2: An i.i.d. Gaussian model
- ▶ Block #3: Maximum likelihood
- ▶ Block #4: Leave undefined



The density function is

$$p(x|\mu, \Sigma) := \frac{1}{(2\pi)^{\frac{d}{2}} \sqrt{\det(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

The central moments are:

$$\mathbb{E}[x] = \int_{\mathbb{R}^d} x p(x|\mu, \Sigma) dx = \mu,$$

$$\text{Cov}(x) = \mathbb{E}[(x - \mathbb{E}[x])(x - \mathbb{E}[x])^T] = \mathbb{E}[xx^T] - \mathbb{E}[x]\mathbb{E}[x]^T = \Sigma.$$

BLOCK #2: A PROBABILISTIC MODEL

Probabilistic Models

- ▶ A *probabilistic model* is a set of probability distributions, $p(x|\theta)$.
- ▶ We pick the *distribution family* $p(\cdot)$, but don't know the parameter θ .

Example: Model data with a Gaussian distribution $p(x|\theta)$, $\theta = \{\mu, \Sigma\}$.

The i.i.d. assumption

Assume data is *independent and identically distributed (iid)*. This is written

$$x_i \stackrel{iid}{\sim} p(x|\theta), \quad i = 1, \dots, n.$$

Writing the density as $p(x|\theta)$, then the *joint* density decomposes as

$$p(x_1, \dots, x_n | \theta) = \prod_{i=1}^n p(x_i | \theta).$$

BLOCK #3: MAXIMUM LIKELIHOOD ESTIMATION

Maximum Likelihood approach

We now need to find θ . *Maximum likelihood* seeks the value of θ that maximizes the likelihood function:

$$\hat{\theta}_{\text{ML}} := \arg \max_{\theta} p(x_1, \dots, x_n | \theta),$$

This value best explains the data according to the chosen distribution family.

Maximum Likelihood equation

The analytic criterion for this maximum likelihood estimator is:

$$\nabla_{\theta} \prod_{i=1}^n p(x_i | \theta) = 0.$$

Simply put, the maximum is at a peak. There is no “upward” direction.

BLOCK #3: LOGARITHM TRICK

Logarithm trick

Calculating $\nabla_{\theta} \prod_{i=1}^n p(x_i|\theta)$ can be complicated. We use the fact that the logarithm is monotonically increasing on \mathbb{R}_+ , and the equality

$$\ln\left(\prod_i f_i\right) = \sum_i \ln(f_i).$$

Consequence: Taking the logarithm does not change the *location* of a maximum or minimum:

$$\max_y \ln g(y) \neq \max_y g(y) \quad \text{The } value \text{ changes.}$$

$$\arg \max_y \ln g(y) = \arg \max_y g(y) \quad \text{The } location \text{ does not change.}$$

BLOCK #3: ANALYTIC MAXIMUM LIKELIHOOD

Maximum likelihood and the logarithm trick

$$\hat{\theta}_{\text{ML}} = \arg \max_{\theta} \prod_{i=1}^n p(x_i|\theta) = \arg \max_{\theta} \ln \left(\prod_{i=1}^n p(x_i|\theta) \right) = \arg \max_{\theta} \sum_{i=1}^n \ln p(x_i|\theta)$$

To then solve for $\hat{\theta}_{\text{ML}}$, find

$$\nabla_{\theta} \sum_{i=1}^n \ln p(x_i|\theta) = \sum_{i=1}^n \nabla_{\theta} \ln p(x_i|\theta) = 0.$$

Depending on the choice of the model, we will be able to solve this

1. analytically (via a simple set of equations)
2. numerically (via an iterative algorithm using different equations)
3. approximately (typically when #2 converges to a local optimal solution)

EXAMPLE: MULTIVARIATE GAUSSIAN MLE

Block #2: Multivariate Gaussian data model

Model: Set of all Gaussians on \mathbb{R}^d with unknown mean $\mu \in \mathbb{R}^d$ and covariance $\Sigma \in \mathbb{S}_{++}^d$ (positive definite $d \times d$ matrix).

We assume that x_1, \dots, x_n are i.i.d. $p(x|\mu, \Sigma)$, written $x_i \stackrel{iid}{\sim} p(x|\mu, \Sigma)$.

Block #3: Maximum likelihood solution

We have to solve the equation

$$\sum_{i=1}^n \nabla_{(\mu, \Sigma)} \ln p(x_i|\mu, \Sigma) = 0$$

for μ and Σ . (Try doing this without the log to appreciate it's usefulness.)

EXAMPLE: GAUSSIAN MEAN MLE

First take the gradient with respect to μ .

$$\begin{aligned} 0 &= \nabla_{\mu} \sum_{i=1}^n \ln \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(x_i - \mu)^T \Sigma^{-1} (x_i - \mu)\right) \\ &= \nabla_{\mu} \sum_{i=1}^n -\frac{1}{2} \ln(2\pi)^d |\Sigma| - \frac{1}{2}(x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \\ &= -\frac{1}{2} \sum_{i=1}^n \nabla_{\mu} \left(x_i^T \Sigma^{-1} x_i - 2\mu^T \Sigma^{-1} x_i + \mu^T \Sigma^{-1} \mu\right) = -\Sigma^{-1} \sum_{i=1}^n (x_i - \mu) \end{aligned}$$

Since Σ is positive definite, the only solution is

$$\sum_{i=1}^n (x_i - \mu) = 0 \quad \Rightarrow \quad \hat{\mu}_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n x_i$$

Since this solution is independent of Σ , it doesn't depend on $\hat{\Sigma}_{\text{ML}}$.

EXAMPLE: GAUSSIAN COVARIANCE MLE

Now take the gradient with respect to Σ .

$$\begin{aligned} 0 &= \nabla_{\Sigma} \sum_{i=1}^n -\frac{1}{2} \ln(2\pi)^d |\Sigma| - \frac{1}{2} (x_i - \mu)^T \Sigma^{-1} (x_i - \mu) \\ &= -\frac{n}{2} \nabla_{\Sigma} \ln |\Sigma| - \frac{1}{2} \nabla_{\Sigma} \text{trace} \left(\Sigma^{-1} \sum_{i=1}^n (x_i - \mu)(x_i - \mu)^T \right) \\ &= -\frac{n}{2} \Sigma^{-1} + \frac{1}{2} \Sigma^{-2} \sum_{i=1}^n (x_i - \mu)(x_i - \mu)^T \end{aligned}$$

Solving for Σ and plugging in $\mu = \hat{\mu}_{\text{ML}}$,

$$\hat{\Sigma}_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu}_{\text{ML}})(x_i - \hat{\mu}_{\text{ML}})^T.$$

EXAMPLE: GAUSSIAN MLE (SUMMARY)

So if we have data x_1, \dots, x_n in \mathbb{R}^d that we hypothesize is i.i.d. Gaussian, the maximum likelihood values of the mean and covariance matrix are

$$\hat{\mu}_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \hat{\Sigma}_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu}_{\text{ML}})(x_i - \hat{\mu}_{\text{ML}})^T.$$

Are we done? There are many assumptions/issues with this approach that makes finding the “best” parameter values not a complete victory.

- ▶ We made a model assumption (multivariate Gaussian).
- ▶ We made an i.i.d. assumption.
- ▶ We assumed that maximizing the likelihood is the best thing to do.

Comment: We often use θ_{ML} to make predictions about x_{new} (Block #4).

How does θ_{ML} generalize to x_{new} ?

If $x_{1:n}$ don’t “capture the space” well, θ_{ML} can *overfit* the data.

ColumbiaX: Machine Learning

Lecture 4

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

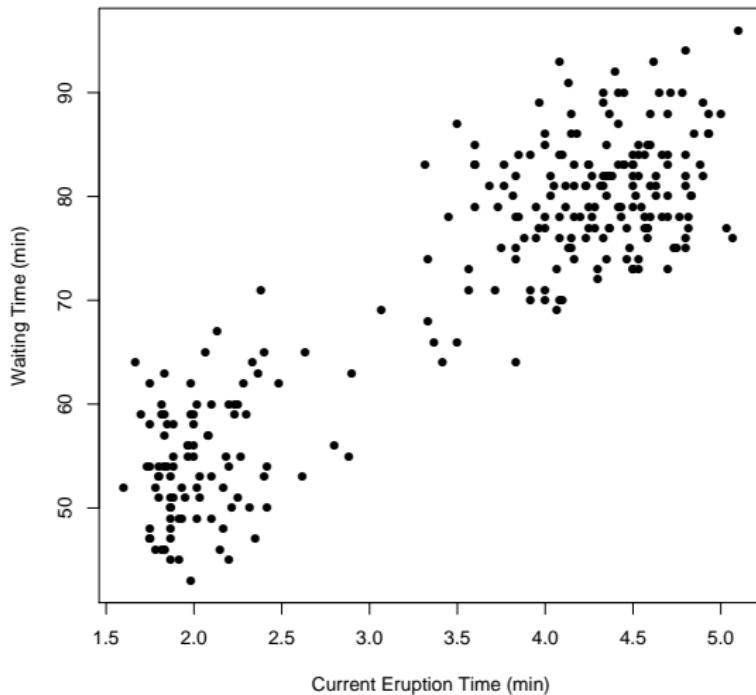
Columbia University

LINEAR REGRESSION

EXAMPLE: OLD FAITHFUL

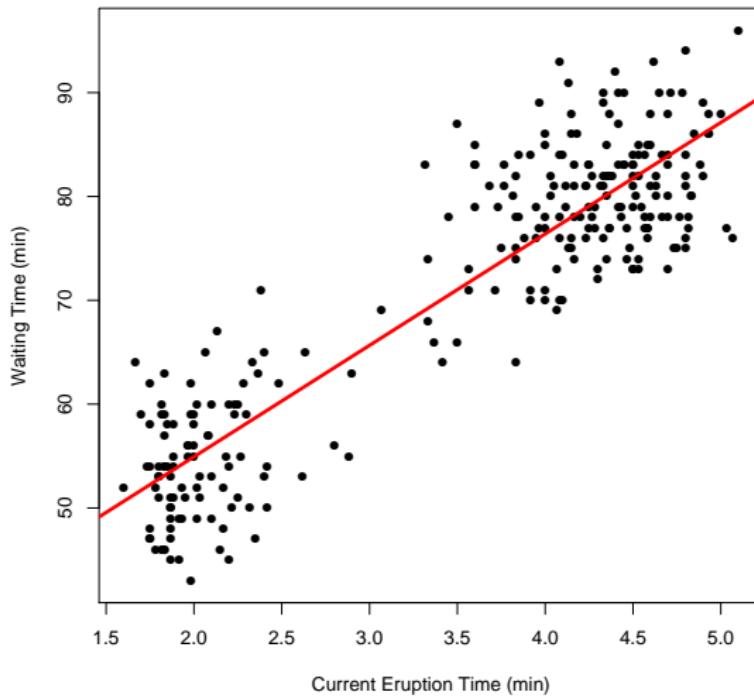


EXAMPLE: OLD FAITHFUL



Can we meaningfully predict the time between eruptions only using the duration of the last eruption?

EXAMPLE: OLD FAITHFUL



Can we meaningfully predict the time between eruptions only using the duration of the last eruption?

EXAMPLE: OLD FAITHFUL

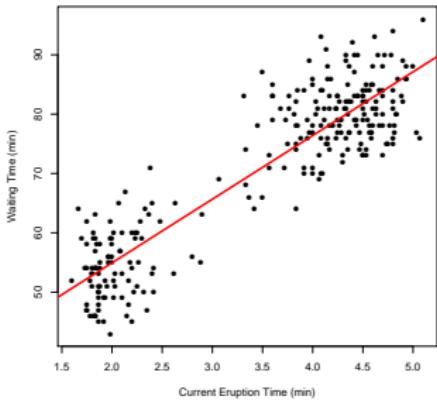
One model for this

$$(\text{wait time}) \approx w_0 + (\text{last duration}) \times w_1$$

- ▶ w_0 and w_1 are to be learned.
- ▶ This is an example of linear regression.

Refresher

w_1 is the slope, w_0 is called the intercept, bias, shift, offset.

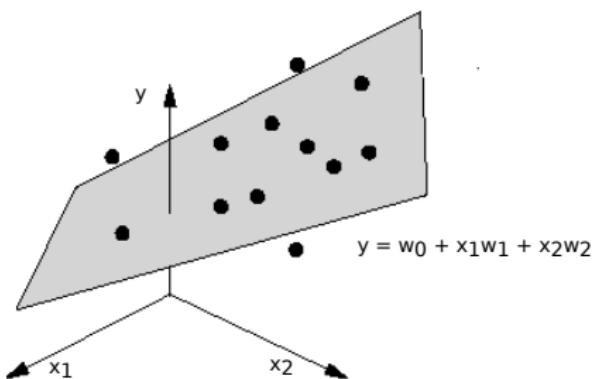


HIGHER DIMENSIONS

Two inputs

$$(\text{output}) \approx w_0 + (\text{input 1}) \times w_1 + (\text{input 2}) \times w_2$$

With two inputs the intuition
is the same →



REGRESSION: PROBLEM DEFINITION

Data

Input: $x \in \mathbb{R}^d$ (i.e., measurements, covariates, features, indepen. variables)

Output: $y \in \mathbb{R}$ (i.e., response, dependent variable)

Goal

Find a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ such that $y \approx f(x; w)$ for the data pair (x, y) .

$f(x; w)$ is called a *regression function*. Its free parameters are w .

Definition of linear regression

A regression method is called *linear* if the prediction f is a linear function of the unknown parameters w .

LEAST SQUARES LINEAR REGRESSION MODEL

Model

The linear regression model we focus on now has the form

$$y_i \approx f(x_i; w) = w_0 + \sum_{j=1}^d x_{ij}w_j.$$

Model learning

We have the set of *training data* $(x_1, y_1) \dots (x_n, y_n)$. We want to use this data to learn a w such that $y_i \approx f(x_i; w)$. But we first need an *objective function* to tell us what a “good” value of w is.

Least squares

The *least squares* objective tells us to pick the w that minimizes the sum of squared errors

$$w_{\text{LS}} = \arg \min_w \sum_{i=1}^n (y_i - f(x_i; w))^2 \equiv \arg \min_w \mathcal{L}.$$

LEAST SQUARES IN PICTURES

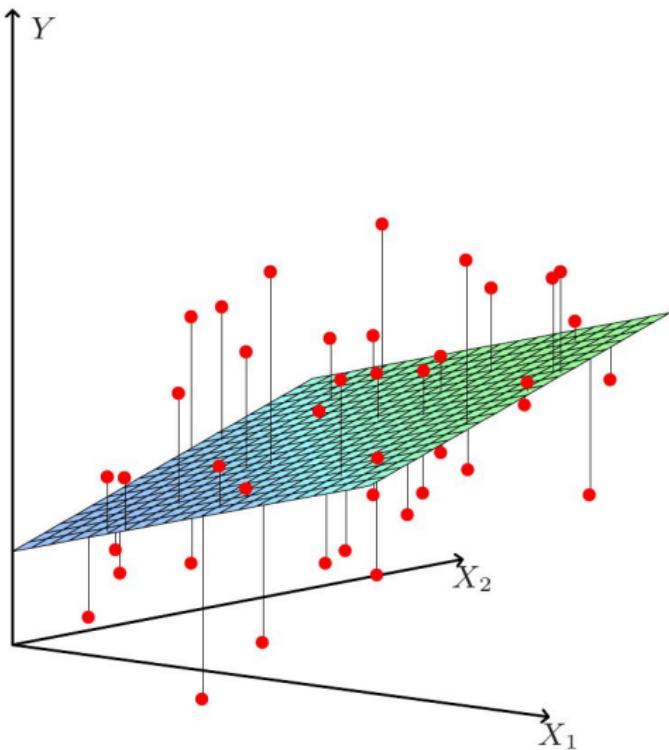
Observations:

Vertical length is error.

The objective function \mathcal{L} is the sum of all the squared lengths.

Find weights (w_1, w_2) plus an offset w_0 to minimize \mathcal{L} .

(w_0, w_1, w_2) defines this plane.



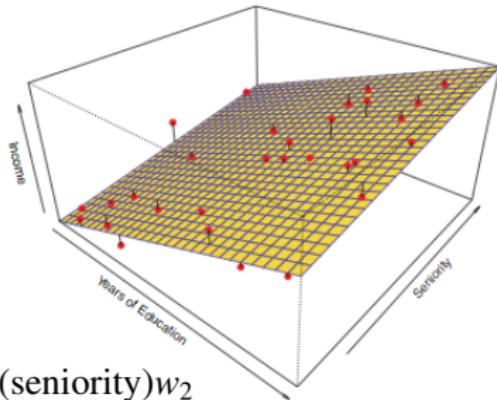
EXAMPLE: EDUCATION, SENIORITY AND INCOME

2-dimensional problem

Input: $(\text{education}, \text{seniority}) \in \mathbb{R}^2$.

Output: $(\text{income}) \in \mathbb{R}$

Model: $(\text{income}) \approx w_0 + (\text{education})w_1 + (\text{seniority})w_2$



Question: Both $w_1, w_2 > 0$. What does this tell us?

Answer: As education and/or seniority goes up, income tends to go up.

(Caveat: This is a statement about correlation, not causation.)

LEAST SQUARES LINEAR REGRESSION MODEL

Thus far

We have data pairs (x_i, y_i) of measurements $x_i \in \mathbb{R}^d$ and a response $y_i \in \mathbb{R}$.
We believe there is a linear relationship between x_i and y_i ,

$$y_i = w_0 + \sum_{j=1}^d x_{ij}w_j + \epsilon_i$$

and we want to minimize the objective function

$$\mathcal{L} = \sum_{i=1}^n \epsilon_i^2 = \sum_{i=1}^n (y_i - w_0 - \sum_{j=1}^d x_{ij}w_j)^2$$

with respect to (w_0, w_1, \dots, w_d) .

Can math notation make this easier to look at/work with?

NOTATION: VECTORS AND MATRICES

We think of data with d dimensions as a *column* vector:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{bmatrix} \quad (\text{e.g.}) \Rightarrow \begin{bmatrix} \text{age} \\ \text{height} \\ \vdots \\ \text{income} \end{bmatrix}$$

A set of n vectors can be stacked into a matrix:

$$\mathbf{X} = \begin{bmatrix} x_{11} & \dots & x_{1d} \\ x_{21} & \dots & x_{2d} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} -x_1^T- \\ -x_2^T- \\ \vdots \\ -x_n^T- \end{bmatrix}$$

Assumptions for now:

- ▶ All features are treated as continuous-valued ($x \in \mathbb{R}^d$)
- ▶ We have more observations than dimensions ($d < n$)

NOTATION: REGRESSION (AND CLASSIFICATION)

Usually, for linear regression (and classification) we include an intercept term w_0 that doesn't interact with any element in the vector $x \in \mathbb{R}^d$.

It will be convenient to attach a 1 to the first dimension of each vector x_i (which we indicate by $x_i \in \mathbb{R}^{d+1}$) and in the first column of the matrix X:

$$x_i = \begin{bmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & x_{21} & \dots & x_{2d} \\ \vdots & & \vdots & \\ 1 & x_{n1} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 - x_1^T - \\ 1 - x_2^T - \\ \vdots \\ 1 - x_n^T - \end{bmatrix}.$$

We also now view $w = [w_0, w_1, \dots, w_d]^T$ as $w \in \mathbb{R}^{d+1}$.

LEAST SQUARES IN VECTOR FORM

Original least squares objective function: $\mathcal{L} = \sum_{i=1}^n (y_i - w_0 - \sum_{j=1}^d x_{ij}w_j)^2$

Using vectors, this can now be written: $\mathcal{L} = \sum_{i=1}^n (y_i - x_i^T w)^2$

Least squares solution (vector version)

We can find w by setting,

$$\nabla_w \mathcal{L} = 0 \quad \Rightarrow \quad \sum_{i=1}^n \nabla_w (y_i^2 - 2w^T x_i y_i + w^T x_i x_i^T w) = 0.$$

Solving gives,

$$-\sum_{i=1}^n 2y_i x_i + \left(\sum_{i=1}^n 2x_i x_i^T \right) w = 0 \quad \Rightarrow \quad w_{\text{LS}} = \left(\sum_{i=1}^n x_i x_i^T \right)^{-1} \left(\sum_{i=1}^n y_i x_i \right).$$

LEAST SQUARES IN MATRIX FORM

Least squares solution (matrix version)

Least squares in matrix form is even cleaner.

Start by organizing the y_i in a column vector, $y = [y_1, \dots, y_n]^T$. Then

$$\mathcal{L} = \sum_{i=1}^n (y_i - x_i^T w)^2 = \|y - Xw\|^2 = (y - Xw)^T (y - Xw).$$

If we take the gradient with respect to w , we find that

$$\nabla_w \mathcal{L} = 2X^T Xw - 2X^T y = 0 \quad \Rightarrow \quad w_{\text{LS}} = (X^T X)^{-1} X^T y.$$

RECALL FROM LINEAR ALGEBRA

Recall: Matrix \times vector ($X^T y = \sum_{i=1}^n y_i x_i$)

$$\begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = y_1 \begin{bmatrix} | \\ x_1 \\ | \end{bmatrix} + y_2 \begin{bmatrix} | \\ x_2 \\ | \end{bmatrix} + \dots + y_n \begin{bmatrix} | \\ x_n \\ | \end{bmatrix}$$

Recall: Matrix \times matrix ($X^T X = \sum_{i=1}^n x_i x_i^T$)

$$\begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} -x_1^T- \\ -x_2^T- \\ \vdots \\ -x_n^T- \end{bmatrix} = x_1 x_1^T + \dots + x_n x_n^T.$$

LEAST SQUARES LINEAR REGRESSION: KEY EQUATIONS

Two notations for the *key equation*

$$w_{\text{LS}} = \left(\sum_{i=1}^n x_i x_i^T \right)^{-1} \left(\sum_{i=1}^n y_i x_i \right) \iff w_{\text{LS}} = (X^T X)^{-1} X^T y.$$

Making Predictions

We use w_{LS} to make predictions.

Given x_{new} , the least squares prediction for y_{new} is

$$y_{\text{new}} \approx x_{\text{new}}^T w_{\text{LS}}$$

LEAST SQUARES SOLUTION

Potential issues

Calculating $w_{\text{LS}} = (X^T X)^{-1} X^T y$ assumes $(X^T X)^{-1}$ exists.

When doesn't it exist?

Answer: When $X^T X$ is not a full rank matrix.

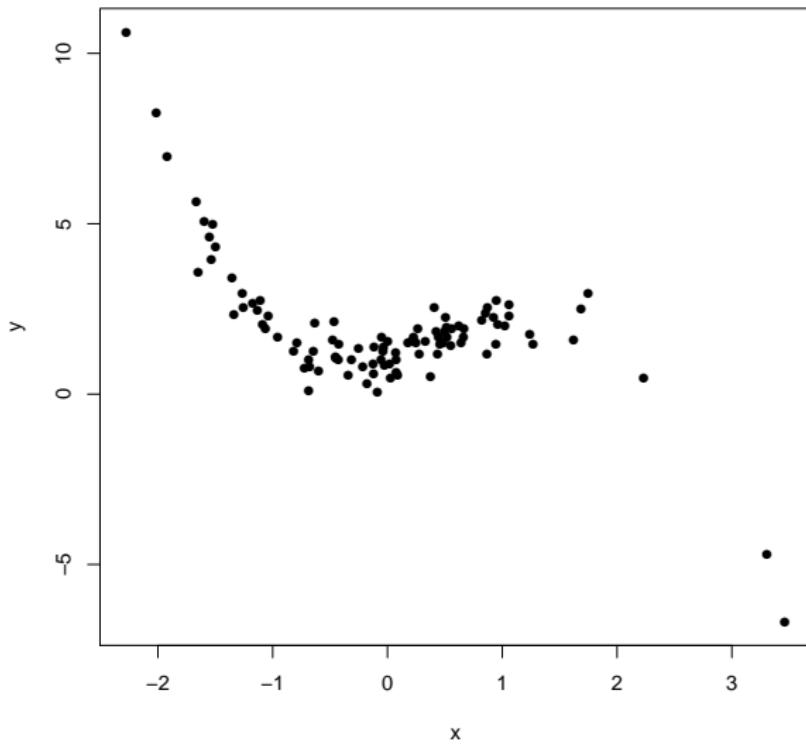
When is $X^T X$ full rank?

Answer: When the $n \times (d + 1)$ matrix X has at least $d + 1$ *linearly independent* rows. This means that any point in \mathbb{R}^{d+1} can be reached by a weighted combination of $d + 1$ rows of X .

Obviously if $n < d + 1$, we can't do least squares. If $(X^T X)^{-1}$ doesn't exist, there are an infinite number of possible solutions.

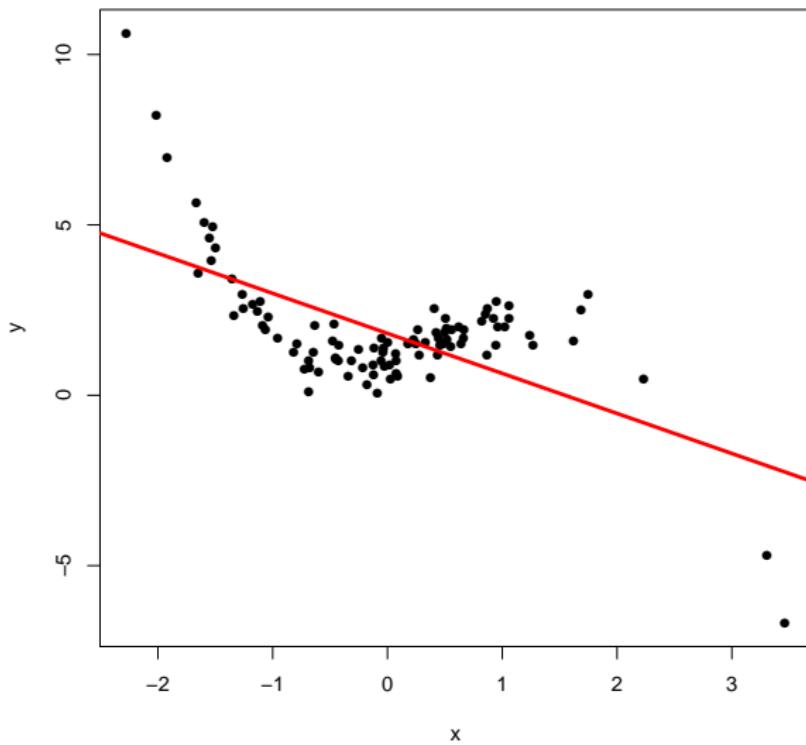
Takeaway: We want $n \gg d$ (i.e., X is “tall and skinny”).

BROADENING LINEAR REGRESSION



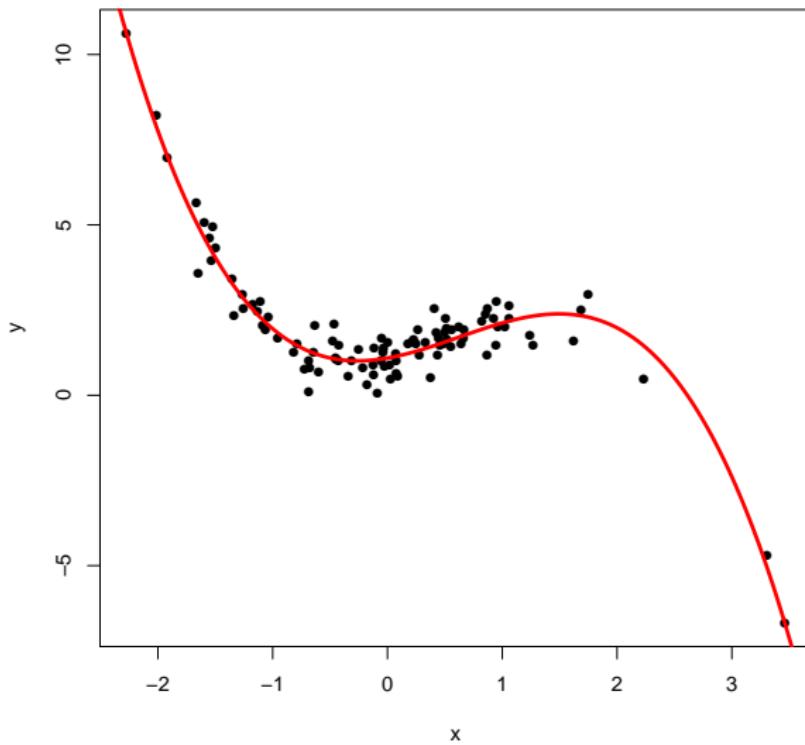
BROADENING LINEAR REGRESSION

$$y = w_0 + w_1 x$$



BROADENING LINEAR REGRESSION

$$y = w_0 + w_1x + w_2x^2 + w_3x^3$$



POLYNOMIAL REGRESSION IN \mathbb{R}

Recall: Definition of linear regression

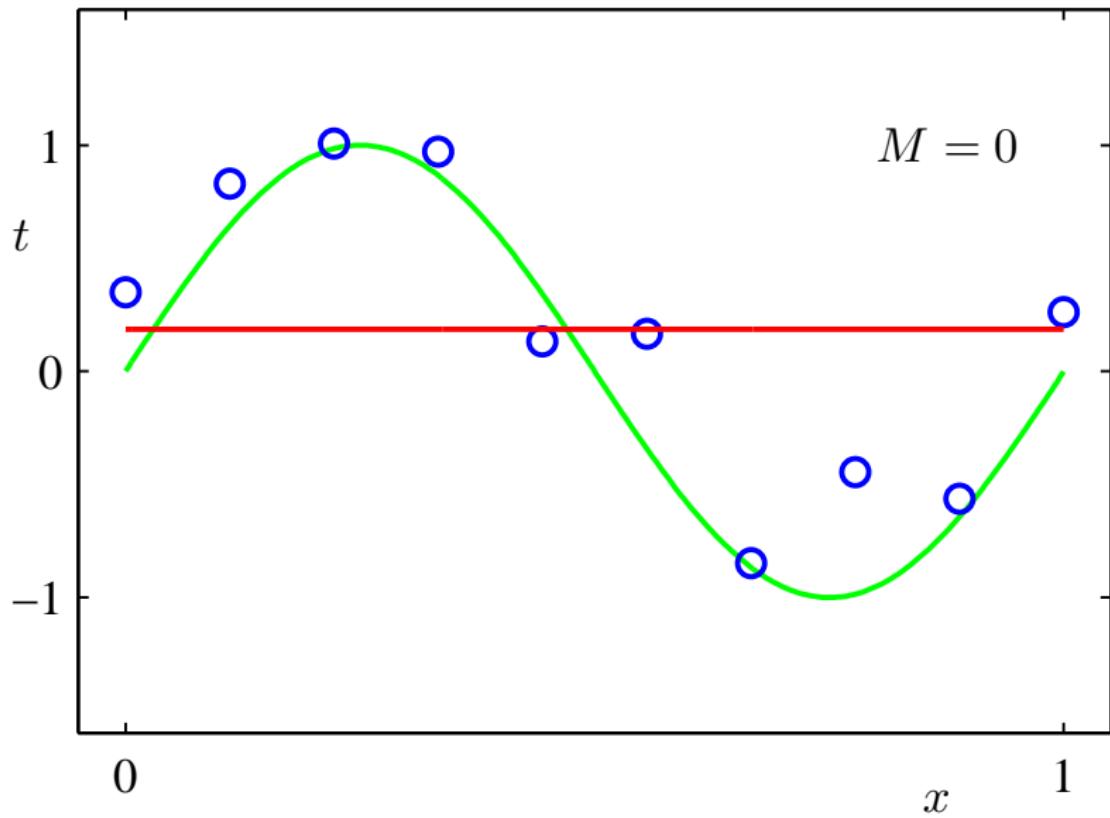
A regression method is called *linear* if the prediction f is a linear function of the unknown parameters w .

- ▶ Therefore, a function such as $y = w_0 + w_1x + w_2x^2$ is *linear* in w .
The LS solution is the same, only the preprocessing is different.
- ▶ E.g., Let $(x_1, y_1) \dots (x_n, y_n)$ be the data, $x \in \mathbb{R}$, $y \in \mathbb{R}$. For a p th-order polynomial approximation, construct the matrix

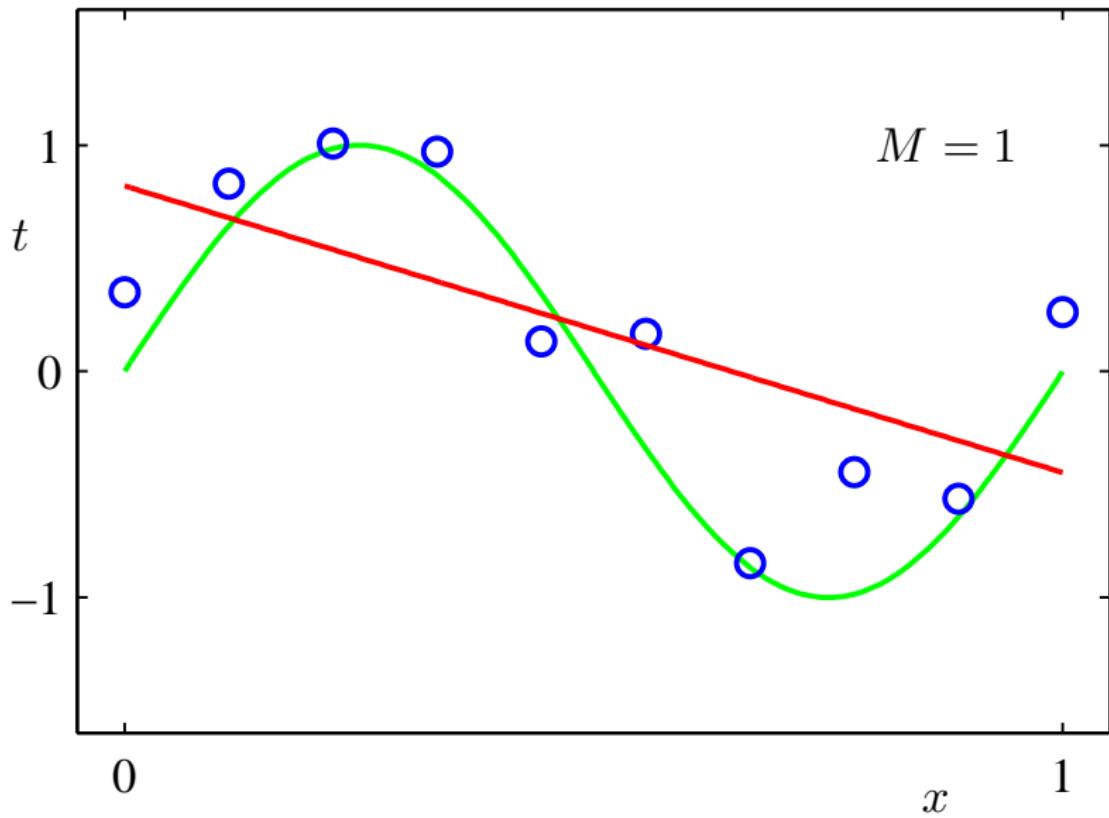
$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & \dots & x_2^p \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{bmatrix}$$

- ▶ Then solve exactly as before: $w_{\text{LS}} = (X^T X)^{-1} X^T y$.

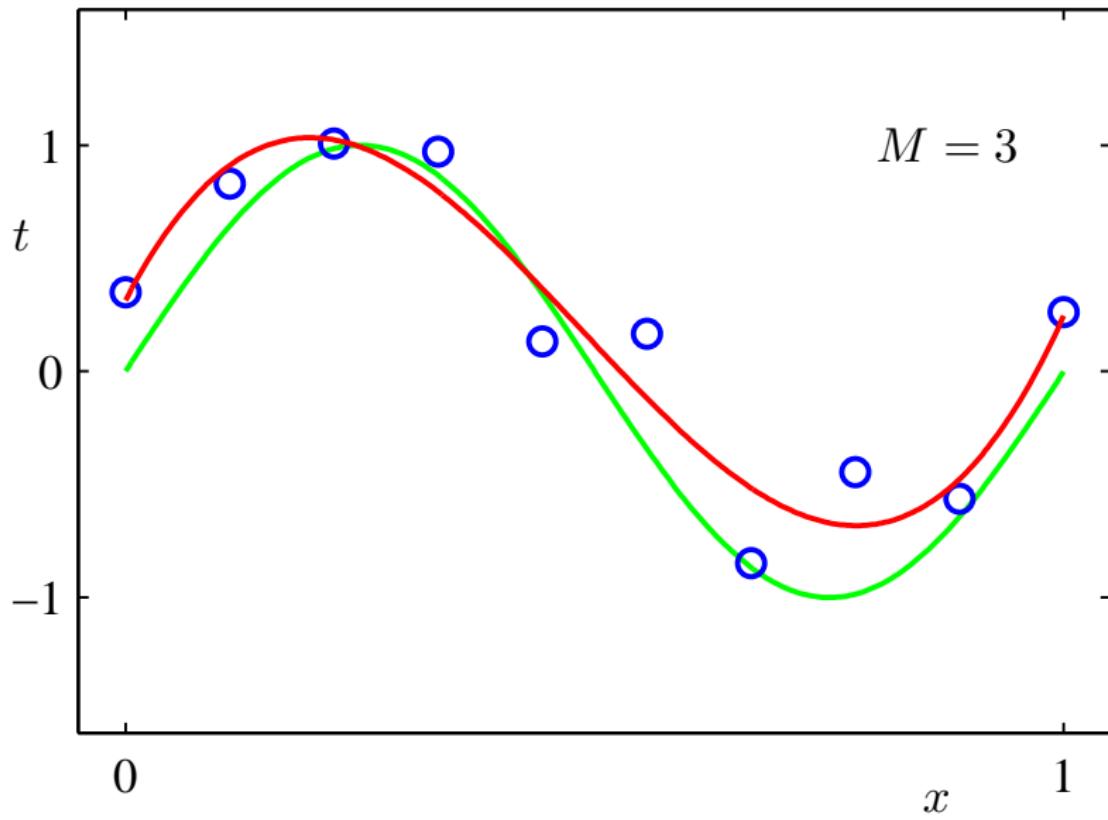
POLYNOMIAL REGRESSION (M TH ORDER)



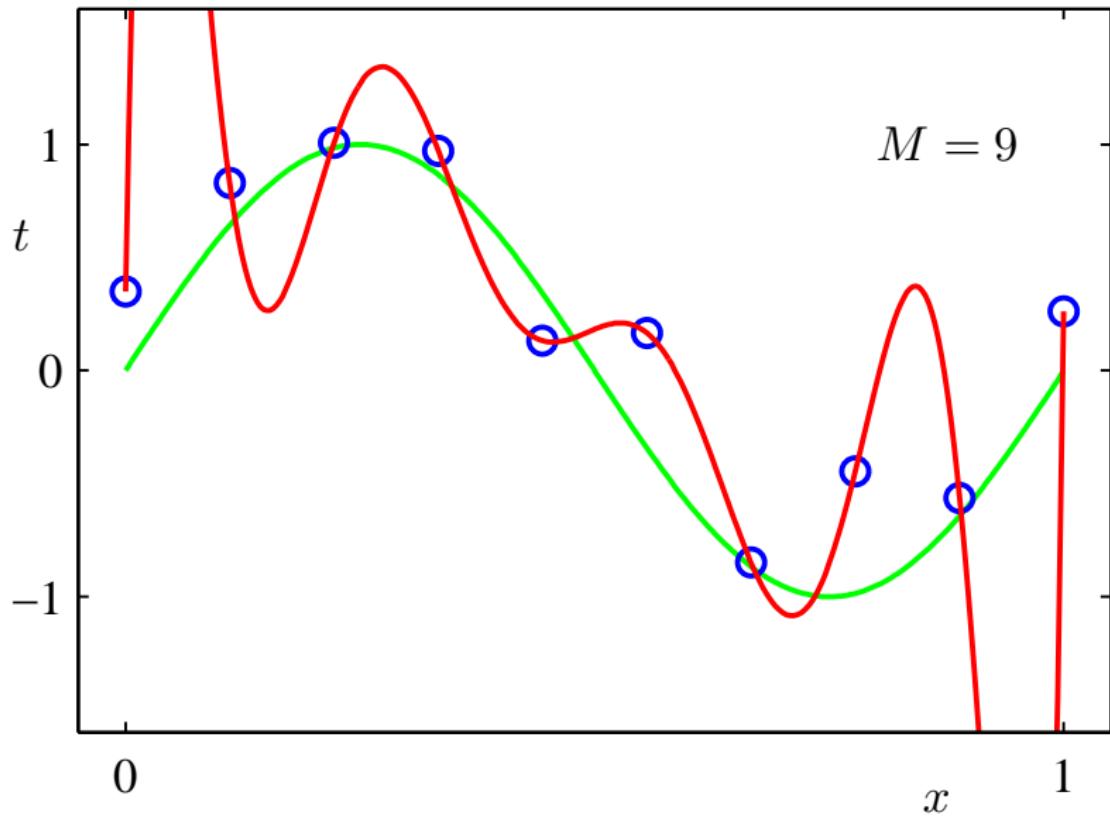
POLYNOMIAL REGRESSION (M TH ORDER)



POLYNOMIAL REGRESSION (M TH ORDER)



POLYNOMIAL REGRESSION (M TH ORDER)



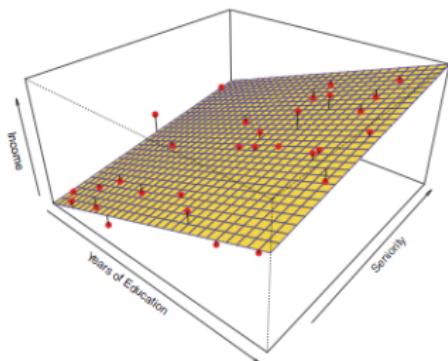
POLYNOMIAL REGRESSION IN TWO DIMENSIONS

Example: 2nd and 3rd order polynomial regression in \mathbb{R}^2

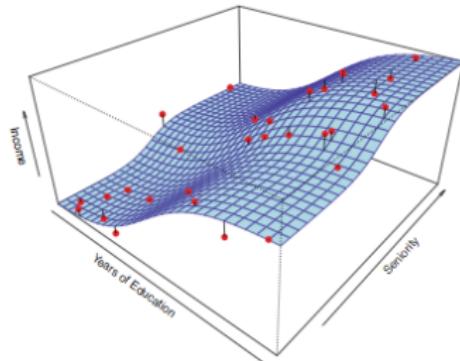
The width of X grows as (order) \times (dimensions) + 1.

2nd order: $y_i = w_0 + w_1x_{i1} + w_2x_{i2} + w_3x_{i1}^2 + w_4x_{i2}^2$

3rd order: $y_i = w_0 + w_1x_{i1} + w_2x_{i2} + w_3x_{i1}^2 + w_4x_{i2}^2 + w_5x_{i1}^3 + w_6x_{i2}^3$



(a) 1st order



(b) 3rd order

FURTHER EXTENSIONS

More generally, for $x_i \in \mathbb{R}^{d+1}$ least squares linear regression can be performed on functions $f(x_i; w)$ of the form

$$y_i \approx f(x_i, w) = \sum_{s=1}^S g_s(x_i) w_s.$$

For example,

$$\begin{aligned}g_s(x_i) &= x_{ij}^2 \\g_s(x_i) &= \log x_{ij} \\g_s(x_i) &= \mathbb{I}(x_{ij} < a) \\g_s(x_i) &= \mathbb{I}(x_{ij} < x_{ij'})\end{aligned}$$

As long as the function is *linear* in w_1, \dots, w_S , we can construct the matrix X by putting the transformed x_i on row i , and solve $w_{LS} = (X^T X)^{-1} X^T y$.

One caveat is that, as the number of functions increases, we need more data to avoid overfitting.

GEOMETRY OF LEAST SQUARES REGRESSION

Thinking geometrically about least squares regression helps a lot.

- ▶ We want to minimize $\|y - Xw\|^2$. Think of the vector y as a point in \mathbb{R}^n . We want to find w in order to get the product Xw close to y .
- ▶ If X_j is the j th *column* of X , then $Xw = \sum_{j=1}^{d+1} w_j X_j$.
- ▶ That is, we weight the columns in X by values in w to approximate y .
- ▶ The LS solutions returns w such that Xw is as close to y as possible in the Euclidean sense (i.e., intuitive “direct-line” distance).

GEOMETRY OF LEAST SQUARES REGRESSION

$$\arg \min_w \|y - Xw\|^2 \quad \Rightarrow \quad w_{\text{LS}} = (X^T X)^{-1} X^T y.$$

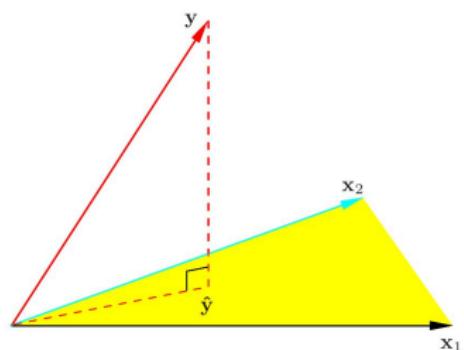
The columns of X define a $d + 1$ -dimensional subspace in the higher dimensional \mathbb{R}^n .

The closest point in that subspace is the *orthonormal projection* of y into the *column space* of X .

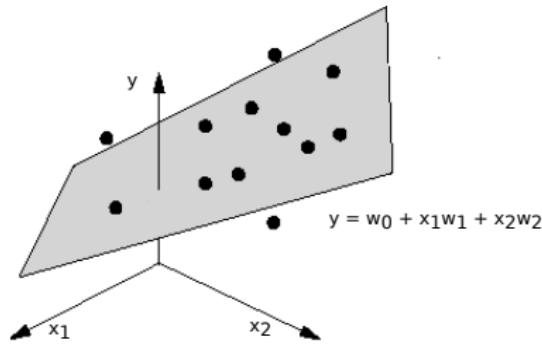
Right: $y \in \mathbb{R}^3$ and data $x_i \in \mathbb{R}$.

$$X_1 = [1, 1, 1]^T \text{ and } X_2 = [x_1, x_2, x_3]^T$$

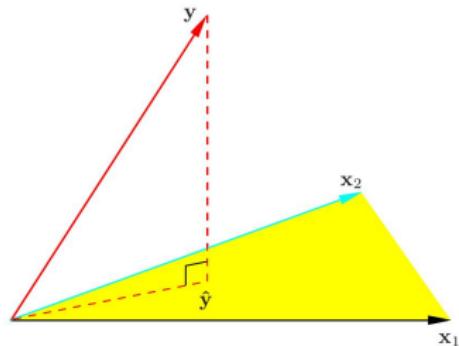
The approximation is $\hat{y} = Xw_{\text{LS}} = X(X^T X)^{-1} X^T y$.



GEOMETRY OF LEAST SQUARES REGRESSION



(a) $y_i \approx w_0 + x_i^T w$ for $i = 1, \dots, n$



(b) $y \approx Xw$

There are some key differences between (a) and (b) worth highlighting as you try to develop the corresponding intuitions.

(a) Can be shown for all n , but only for $x_i \in \mathbb{R}^2$ (not counting the added 1).

(b) This corresponds to $n = 3$ and one-dimensional data: $X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \end{bmatrix}$.

ColumbiaX: Machine Learning

Lecture 3

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

REGRESSION: PROBLEM DEFINITION

Data

Measured pairs (x, y) , where $x \in \mathbb{R}^{d+1}$ (input) and $y \in \mathbb{R}$ (output)

Goal

Find a function $f : \mathbb{R}^{d+1} \rightarrow \mathbb{R}$ such that $y \approx f(x; w)$ for the data pair (x, y) .
 $f(x; w)$ is the *regression function* and the vector w are its parameters.

Definition of linear regression

A regression method is called *linear* if the prediction f is a linear function of the unknown parameters w .

LEAST SQUARES (CONTINUED)

LEAST SQUARES LINEAR REGRESSION

Least squares solution

Least squares finds the w that minimizes the sum of squared errors. The least squares objective in the most basic form where $f(x; w) = x^T w$ is

$$\mathcal{L} = \sum_{i=1}^n (y_i - x_i^T w)^2 = \|y - Xw\|^2 = (y - Xw)^T (y - Xw).$$

We defined $y = [y_1, \dots, y_n]^T$ and $X = [x_1, \dots, x_n]^T$.

Taking the gradient with respect to w and setting to zero, we find that

$$\nabla_w \mathcal{L} = 2X^T Xw - 2X^T y = 0 \quad \Rightarrow \quad w_{\text{LS}} = (X^T X)^{-1} X^T y.$$

In other words, w_{LS} is the vector that minimizes \mathcal{L} .

PROBABILISTIC VIEW

- ▶ Last class, we discussed the geometric interpretation of least squares.
- ▶ Least squares also has an insightful probabilistic interpretation that allows us to analyze its properties.
- ▶ That is, given that we pick this model as reasonable for our problem, we can ask: What kinds of assumptions are we making?

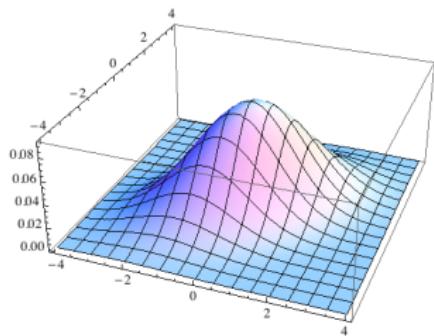
PROBABILISTIC VIEW

Recall: Gaussian density in n dimensions

Assume a diagonal covariance matrix $\Sigma = \sigma^2 I$. The density is

$$p(y|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma^2}(y - \mu)^T(y - \mu)\right).$$

What if we restrict the mean to $\mu = Xw$
and find the *maximum likelihood*
solution for w ?



PROBABILISTIC VIEW

Maximum likelihood for Gaussian linear regression

Plug $\mu = Xw$ into the multivariate Gaussian distribution and solve for w using maximum likelihood.

$$\begin{aligned} w_{\text{ML}} &= \arg \max_w \ln p(y|\mu = Xw, \sigma^2) \\ &= \arg \max_w -\frac{1}{2\sigma^2} \|y - Xw\|^2 - \frac{n}{2} \ln(2\pi\sigma^2). \end{aligned}$$

Least squares (LS) and maximum likelihood (ML) share the same solution:

$$\text{LS: } \arg \min_w \|y - Xw\|^2 \Leftrightarrow \text{ML: } \arg \max_w -\frac{1}{2\sigma^2} \|y - Xw\|^2$$

PROBABILISTIC VIEW

- ▶ Therefore, in a sense we are making an *independent Gaussian noise* assumption about the error, $\epsilon_i = y_i - x_i^T w$.
- ▶ Other ways of saying this:
 - 1) $y_i = x_i^T w + \epsilon_i$, $\epsilon_i \stackrel{iid}{\sim} N(0, \sigma^2)$, for $i = 1, \dots, n$,
 - 2) $y_i \stackrel{ind}{\sim} N(x_i^T w, \sigma^2)$, for $i = 1, \dots, n$,
 - 3) $y \sim N(Xw, \sigma^2 I)$, as on the previous slides.
- ▶ Can we use this probabilistic line of analysis to better understand the maximum likelihood (i.e., least squares) solution?

PROBABILISTIC VIEW

Expected solution

Given: The *modeling assumption* that $y \sim N(Xw, \sigma^2 I)$.

We can calculate the expectation of the ML solution under this distribution,

$$\begin{aligned}\mathbb{E}[w_{\text{ML}}] &= \mathbb{E}[(X^T X)^{-1} X^T y] \quad \left(= \int [(X^T X)^{-1} X^T y] p(y|X, w) dy \right) \\ &= (X^T X)^{-1} X^T \mathbb{E}[y] \\ &= (X^T X)^{-1} X^T X w \\ &= w\end{aligned}$$

Therefore w_{ML} is an *unbiased* estimate of w , i.e., $\mathbb{E}[w_{\text{ML}}] = w$.

REVIEW: AN EQUALITY FROM PROBABILITY

- ▶ Even though the “expected” maximum likelihood solution is the correct one, should we actually expect to get something near it?

REVIEW: AN EQUALITY FROM PROBABILITY

- ▶ Even though the “expected” maximum likelihood solution is the correct one, should we actually expect to get something near it?
- ▶ We should also look at the covariance. Recall that if $y \sim N(\mu, \Sigma)$, then

$$\text{Var}[y] = \mathbb{E}[(y - \mathbb{E}[y])(y - \mathbb{E}[y])^T] = \Sigma.$$

REVIEW: AN EQUALITY FROM PROBABILITY

- ▶ Even though the “expected” maximum likelihood solution is the correct one, should we actually expect to get something near it?
- ▶ We should also look at the covariance. Recall that if $y \sim N(\mu, \Sigma)$, then

$$\text{Var}[y] = \mathbb{E}[(y - \mathbb{E}[y])(y - \mathbb{E}[y])^T] = \Sigma.$$

- ▶ Plugging in $\mathbb{E}[y] = \mu$, this is equivalently written as

$$\begin{aligned}\text{Var}[y] &= \mathbb{E}[(y - \mu)(y - \mu)^T] \\ &= \mathbb{E}[yy^T - y\mu^T - \mu y^T + \mu\mu^T] \\ &= \mathbb{E}[yy^T] - \mu\mu^T\end{aligned}$$

- ▶ Immediately we also get $\mathbb{E}[yy^T] = \Sigma + \mu\mu^T$.

PROBABILISTIC VIEW

Variance of the solution

Returning to least squares linear regression, we wish to find

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])^T] \\ &= \mathbb{E}[w_{\text{ML}} w_{\text{ML}}^T] - \mathbb{E}[w_{\text{ML}}]\mathbb{E}[w_{\text{ML}}]^T.\end{aligned}$$

¹ Aside: For matrices A , B and vector c , recall that $(ABC)^T = c^T B^T A^T$.

PROBABILISTIC VIEW

Variance of the solution

Returning to least squares linear regression, we wish to find

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])^T] \\ &= \mathbb{E}[w_{\text{ML}} w_{\text{ML}}^T] - \mathbb{E}[w_{\text{ML}}]\mathbb{E}[w_{\text{ML}}]^T.\end{aligned}$$

The sequence of equalities follows:¹

$$\text{Var}[w_{\text{ML}}] = \mathbb{E}[(X^T X)^{-1} X^T y y^T X (X^T X)^{-1}] - w w^T$$

¹ Aside: For matrices A, B and vector c , recall that $(ABC)^T = c^T B^T A^T$.

PROBABILISTIC VIEW

Variance of the solution

Returning to least squares linear regression, we wish to find

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])^T] \\ &= \mathbb{E}[w_{\text{ML}} w_{\text{ML}}^T] - \mathbb{E}[w_{\text{ML}}]\mathbb{E}[w_{\text{ML}}]^T.\end{aligned}$$

The sequence of equalities follows:¹

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(X^T X)^{-1} X^T y y^T X (X^T X)^{-1}] - w w^T \\ &= (X^T X)^{-1} X^T \mathbb{E}[y y^T] X (X^T X)^{-1} - w w^T\end{aligned}$$

¹ Aside: For matrices A, B and vector c , recall that $(ABC)^T = c^T B^T A^T$.

PROBABILISTIC VIEW

Variance of the solution

Returning to least squares linear regression, we wish to find

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])^T] \\ &= \mathbb{E}[w_{\text{ML}} w_{\text{ML}}^T] - \mathbb{E}[w_{\text{ML}}]\mathbb{E}[w_{\text{ML}}]^T.\end{aligned}$$

The sequence of equalities follows:¹

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(X^T X)^{-1} X^T y y^T X (X^T X)^{-1}] - w w^T \\ &= (X^T X)^{-1} X^T \mathbb{E}[y y^T] X (X^T X)^{-1} - w w^T \\ &= (X^T X)^{-1} X^T (\sigma^2 I + X w w^T X^T) X (X^T X)^{-1} - w w^T\end{aligned}$$

¹ Aside: For matrices A, B and vector c , recall that $(ABC)^T = c^T B^T A^T$.

PROBABILISTIC VIEW

Variance of the solution

Returning to least squares linear regression, we wish to find

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])^T] \\ &= \mathbb{E}[w_{\text{ML}} w_{\text{ML}}^T] - \mathbb{E}[w_{\text{ML}}]\mathbb{E}[w_{\text{ML}}]^T.\end{aligned}$$

The sequence of equalities follows:¹

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(X^T X)^{-1} X^T y y^T X (X^T X)^{-1}] - w w^T \\ &= (X^T X)^{-1} X^T \mathbb{E}[y y^T] X (X^T X)^{-1} - w w^T \\ &= (X^T X)^{-1} X^T (\sigma^2 I + X w w^T X^T) X (X^T X)^{-1} - w w^T \\ &= (X^T X)^{-1} X^T \sigma^2 I X (X^T X)^{-1} + \dots \\ &\quad (X^T X)^{-1} X^T X w w^T X^T X (X^T X)^{-1} - w w^T\end{aligned}$$

¹ Aside: For matrices A, B and vector c , recall that $(ABC)^T = c^T B^T A^T$.

PROBABILISTIC VIEW

Variance of the solution

Returning to least squares linear regression, we wish to find

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])(w_{\text{ML}} - \mathbb{E}[w_{\text{ML}}])^T] \\ &= \mathbb{E}[w_{\text{ML}} w_{\text{ML}}^T] - \mathbb{E}[w_{\text{ML}}]\mathbb{E}[w_{\text{ML}}]^T.\end{aligned}$$

The sequence of equalities follows:¹

$$\begin{aligned}\text{Var}[w_{\text{ML}}] &= \mathbb{E}[(X^T X)^{-1} X^T y y^T X (X^T X)^{-1}] - w w^T \\ &= (X^T X)^{-1} X^T \mathbb{E}[y y^T] X (X^T X)^{-1} - w w^T \\ &= (X^T X)^{-1} X^T (\sigma^2 I + X w w^T X^T) X (X^T X)^{-1} - w w^T \\ &= (X^T X)^{-1} X^T \sigma^2 I X (X^T X)^{-1} + \dots \\ &\quad (X^T X)^{-1} X^T X w w^T X^T X (X^T X)^{-1} - w w^T \\ &= \sigma^2 (X^T X)^{-1}\end{aligned}$$

¹ Aside: For matrices A , B and vector c , recall that $(ABC)^T = c^T B^T A^T$.

PROBABILISTIC VIEW

- ▶ We've shown that, under the Gaussian assumption $y \sim N(Xw, \sigma^2 I)$,

$$\mathbb{E}[w_{\text{ML}}] = w, \quad \text{Var}[w_{\text{ML}}] = \sigma^2 (X^T X)^{-1}.$$

- ▶ When there are very large values in $\sigma^2 (X^T X)^{-1}$, the values of w_{ML} are very sensitive to the measured data y (more analysis later).
- ▶ This is bad if we want to analyze and predict using w_{ML} .

RIDGE REGRESSION

REGULARIZED LEAST SQUARES

- ▶ We saw how with least squares, the values in w_{ML} may be huge.
- ▶ In general, when developing a model for data we often wish to *constrain* the model parameters in some way.
- ▶ There are many models of the form

$$w_{\text{OPT}} = \arg \min_w \|y - Xw\|^2 + \lambda g(w).$$

- ▶ The added terms are
 1. $\lambda > 0$: a regularization parameter,
 2. $g(w) > 0$: a penalty function that encourages desired properties about w .

RIDGE REGRESSION

Ridge regression is one $g(w)$ that addresses variance issues with w_{ML} .

It uses the squared penalty on the regression coefficient vector w ,

$$w_{\text{RR}} = \arg \min_w \|y - Xw\|^2 + \lambda \|w\|^2$$

The term $g(w) = \|w\|^2$ penalizes large values in w .

However, there is a *tradeoff* between the first and second terms that is controlled by λ .

- ▶ Case $\lambda \rightarrow 0$: $w_{\text{RR}} \rightarrow w_{\text{LS}}$
- ▶ Case $\lambda \rightarrow \infty$: $w_{\text{RR}} \rightarrow \vec{0}$

RIDGE REGRESSION SOLUTION

Objective: We can solve the ridge regression problem using exactly the same procedure as for least squares,

$$\begin{aligned}\mathcal{L} &= \|y - Xw\|^2 + \lambda\|w\|^2 \\ &= (y - Xw)^T(y - Xw) + \lambda w^T w.\end{aligned}$$

Solution: First, take the gradient of \mathcal{L} with respect to w and set to zero,

$$\nabla_w \mathcal{L} = -2X^T y + 2X^T Xw + 2\lambda w = 0$$

Then, solve for w to find that

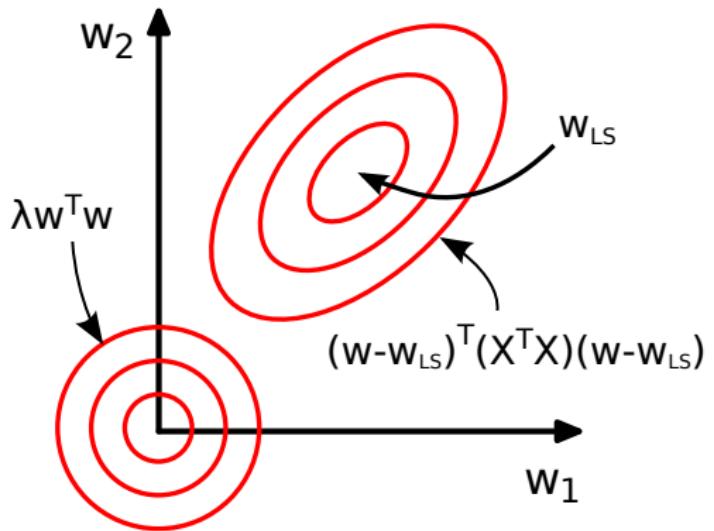
$$w_{\text{RR}} = (\lambda I + X^T X)^{-1} X^T y.$$

RIDGE REGRESSION GEOMETRY

There is a tradeoff between squared error and penalty on w .

We can write both in terms of *level sets*: Curves where function evaluation gives the same number.

The sum of these gives a new set of levels with a unique minimum.



You can check that we can write:

$$\|y - Xw\|^2 + \lambda \|w\|^2 = (w - w_{LS})^T (X^T X) (w - w_{LS}) + \lambda w^T w + (\text{const. w.r.t. } w).$$

DATA PREPROCESSING

Ridge regression is one possible regularization scheme. For this problem, we first assume the following *preprocessing* steps are done:

1. The mean is subtracted off of y :

$$y \leftarrow y - \frac{1}{n} \sum_{i=1}^n y_i.$$

2. The dimensions of x_i have been *standardized* before constructing X :

$$x_{ij} \leftarrow (x_{ij} - \bar{x}_{\cdot j}) / \hat{\sigma}_j, \quad \hat{\sigma}_j = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_{\cdot j})^2}.$$

i.e., subtract the empirical mean and divide by the empirical standard deviation for each dimension.

3. We can show that there is no need for the dimension of 1's in this case.

SOME ANALYSIS OF RIDGE REGRESSION

RIDGE REGRESSION VS LEAST SQUARES

The solutions to least squares and ridge regression are clearly very similar,

$$w_{\text{LS}} = (X^T X)^{-1} X^T y \quad \Leftrightarrow \quad w_{\text{RR}} = (\lambda I + X^T X)^{-1} X^T y.$$

- ▶ We can use linear algebra and probability to compare the two.
- ▶ This requires the *singular value decomposition*, which we review next.

REVIEW: SINGULAR VALUE DECOMPOSITIONS

- ▶ We can write any $n \times d$ matrix X (assume $n > d$) as $X = USV^T$, where
 1. $U: n \times d$ and orthonormal in the columns, i.e. $U^T U = I$.
 2. $S: d \times d$ non-negative diagonal matrix, i.e. $S_{ii} \geq 0$ and $S_{ij} = 0$ for $i \neq j$.
 3. $V: d \times d$ and orthonormal, i.e. $V^T V = VV^T = I$.
- ▶ From this we have the immediate equalities

$$X^T X = (USV^T)^T (USV^T) = VS^2 V^T, \quad XX^T = US^2 U^T.$$

- ▶ Assuming $S_{ii} \neq 0$ for all i (i.e., “ X is full rank”), we also have that

$$(X^T X)^{-1} = (VS^2 V^T)^{-1} = VS^{-2} V^T.$$

Proof: Plug in and see that it satisfies definition of inverse

$$(X^T X)(X^T X)^{-1} = VS^2 V^T VS^{-2} V^T = I.$$

LEAST SQUARES AND THE SVD

Using the SVD we can rewrite the variance,

$$\text{Var}[w_{\text{LS}}] = \sigma^2(X^T X)^{-1} = \sigma^2 V S^{-2} V^T.$$

This inverse becomes huge when S_{ii} is very small for some values of i .
(Aside: This happens when columns of X are highly correlated.)

The least squares prediction for new data is

$$y_{\text{new}} = x_{\text{new}}^T w_{\text{LS}} = x_{\text{new}}^T (X^T X)^{-1} X^T y = x_{\text{new}}^T V S^{-1} U^T y.$$

When S^{-1} has very large values, this can lead to unstable predictions.

RIDGE REGRESSION VS LEAST SQUARES I

Relationship to least squares solution

Recall for two symmetric matrices, $(AB)^{-1} = B^{-1}A^{-1}$.

$$\begin{aligned} w_{\text{RR}} &= (\lambda I + X^T X)^{-1} X^T y \\ &= (\lambda I + X^T X)^{-1} (X^T X) \underbrace{(X^T X)^{-1} X^T y}_{w_{\text{LS}}} \\ &= [(X^T X)(\lambda(X^T X)^{-1} + I)]^{-1} (X^T X) w_{\text{LS}} \\ &= (\lambda(X^T X)^{-1} + I)^{-1} (X^T X)^{-1} (X^T X) w_{\text{LS}} \\ &= (\lambda(X^T X)^{-1} + I)^{-1} w_{\text{LS}} \end{aligned}$$

Can use this to prove that the solution shrinks toward zero: $\|w_{\text{RR}}\|_2 \leq \|w_{\text{LS}}\|_2$.

RIDGE REGRESSION VS LEAST SQUARES II

Continue analysis with the SVD: $X = USV^T \rightarrow (X^T X)^{-1} = VS^{-2}V^T$:

$$\begin{aligned} w_{\text{RR}} &= (\lambda(X^T X)^{-1} + I)^{-1} w_{\text{LS}} \\ &= (\lambda VS^{-2}V^T + I)^{-1} w_{\text{LS}} \\ &= V(\lambda S^{-2} + I)^{-1} V^T w_{\text{LS}} \\ &:= VMV^T w_{\text{LS}} \end{aligned}$$

M is a diagonal matrix with $M_{ii} = \frac{S_{ii}^2}{\lambda + S_{ii}^2}$. We can pursue this to show that

$$w_{\text{RR}} = VS_{\lambda}^{-1}U^T y, \quad S_{\lambda}^{-1} = \begin{bmatrix} \frac{S_{11}}{\lambda + S_{11}^2} & & 0 \\ & \ddots & \\ 0 & & \frac{S_{dd}}{\lambda + S_{dd}^2} \end{bmatrix}$$

Compare with $w_{\text{LS}} = VS^{-1}U^T y$, which is the case where $\lambda = 0$ above.

RIDGE REGRESSION VS LEAST SQUARES III

Ridge regression can also be seen as a special case of least squares.

Define $\hat{y} \approx \hat{X}w$ in the following way,

$$\begin{bmatrix} y \\ 0 \\ \vdots \\ 0 \end{bmatrix} \approx \begin{bmatrix} - & X & - \\ \sqrt{\lambda} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\lambda} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$$

If we solved w_{LS} for *this* regression problem, we find w_{RR} of the *original* problem: Calculating $(\hat{y} - \hat{X}w)^T(\hat{y} - \hat{X}w)$ in two parts gives

$$\begin{aligned} (\hat{y} - \hat{X}w)^T(\hat{y} - \hat{X}w) &= (y - Xw)^T(y - Xw) + (\sqrt{\lambda}w)^T(\sqrt{\lambda}w) \\ &= \|y - Xw\|^2 + \lambda\|w\|^2 \end{aligned}$$

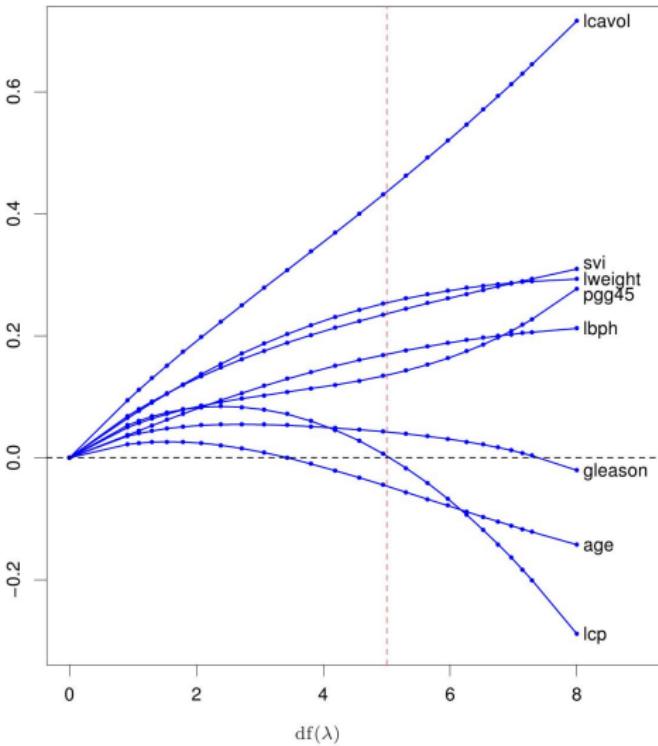
SELECTING λ

Degrees of freedom:

$$\begin{aligned} df(\lambda) &= \text{trace} [X(X^T X + \lambda I)^{-1} X^T] \\ &= \sum_{i=1}^d \frac{S_{ii}^2}{\lambda + S_{ii}^2} \end{aligned}$$

This gives a way of visualizing relationships.

We will discuss methods for picking λ later.



ColumbiaX: Machine Learning

Lecture 4

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

REGRESSION WITH/WITHOUT REGULARIZATION

Given:

A data set $(x_1, y_1), \dots, (x_n, y_n)$, where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. We standardize such that each dimension of x is zero mean unit variance, and y is zero mean.

Model:

We define a model of the form

$$y \approx f(x; w).$$

We particularly focus on the case where $f(x; w) = x^T w$.

Learning:

We can learn the model by minimizing the objective (aka, “loss”) function

$$\mathcal{L} = \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda w^T w \quad \Leftrightarrow \quad \mathcal{L} = \|y - Xw\|^2 + \lambda \|w\|^2$$

We’ve focused on $\lambda = 0$ (least squares) and $\lambda > 0$ (ridge regression).

BIAS-VARIANCE TRADE-OFF

BIAS-VARIANCE FOR LINEAR REGRESSION

We can go further and hypothesize a *generative* model $y \sim N(Xw, \sigma^2 I)$ and some true (but unknown) underlying value for the parameter vector w .

- ▶ We saw how the least squares solution, $w_{\text{LS}} = (X^T X)^{-1} X^T y$, is unbiased but potentially has high variance:

$$\mathbb{E}[w_{\text{LS}}] = w, \quad \text{Var}[w_{\text{LS}}] = \sigma^2 (X^T X)^{-1}.$$

- ▶ By contrast, the ridge regression solution is $w_{\text{RR}} = (\lambda I + X^T X)^{-1} X^T y$. Using the same procedure as for least squares, we can show that

$$\mathbb{E}[w_{\text{RR}}] = (\lambda I + X^T X)^{-1} X^T X w, \quad \text{Var}[w_{\text{RR}}] = \sigma^2 Z (X^T X)^{-1} Z^T,$$

where $Z = (I + \lambda (X^T X)^{-1})^{-1}$.

BIAS-VARIANCE FOR LINEAR REGRESSION

The expectation and covariance of w_{LS} and w_{RR} gives insight into how well we can hope to learn w in the case where our model assumption is correct.

- ▶ Least squares solution: unbiased, but potentially high variance
- ▶ Ridge regression solution: biased, but lower variance than LS

So which is preferable?

Ultimately, we really care about how well our solution for w generalizes to new data. Let (x_0, y_0) be future data for which we have x_0 , but not y_0 .

- ▶ Least squares predicts $y_0 = x_0^T w_{\text{LS}}$
- ▶ Ridge regression predicts $y_0 = x_0^T w_{\text{RR}}$

BIAS-VARIANCE FOR LINEAR REGRESSION

In keeping with the square error measure of performance, we could calculate the expected squared error of our prediction:

$$\mathbb{E} [(y_0 - x_0^T \hat{w})^2 | X, x_0] = \int_{\mathbb{R}} \int_{\mathbb{R}^n} (y_0 - x_0^T \hat{w})^2 p(y|X, w) p(y_0|x_0, w) dy dy_0.$$

- ▶ The estimate \hat{w} is either w_{LS} or w_{RR} .
- ▶ The distributions on y, y_0 are Gaussian with the true (but unknown) w .
- ▶ We condition on knowing x_0, x_1, \dots, x_n .

In words this is saying:

- ▶ Imagine I know X, x_0 and assume some true underlying w .
- ▶ I generate $y \sim N(Xw, \sigma^2 I)$ and approximate w with $\hat{w} = w_{\text{LS}}$ or w_{RR} .
- ▶ I then predict $y_0 \sim N(x_0^T w, \sigma^2)$ using $y_0 \approx x_0^T \hat{w}$.

What is the expected squared error of my prediction?

BIAS-VARIANCE FOR LINEAR REGRESSION

We can calculate this as follows (assume conditioning on x_0 and X),

$$\mathbb{E}[(y_0 - x_0^T \hat{w})^2] = \mathbb{E}[y_0^2] - 2\mathbb{E}[y_0]x_0^T \mathbb{E}[\hat{w}] + x_0^T \mathbb{E}[\hat{w}\hat{w}^T]x_0$$

- ▶ Since y_0 and \hat{w} are independent, $\mathbb{E}[y_0\hat{w}] = \mathbb{E}[y_0]\mathbb{E}[\hat{w}]$.
- ▶ Remember: $\mathbb{E}[\hat{w}\hat{w}^T] = \text{Var}[\hat{w}] + \mathbb{E}[\hat{w}]\mathbb{E}[\hat{w}]^T$

$$\mathbb{E}[y_0^2] = \sigma^2 + (x_0^T w)^2$$

BIAS-VARIANCE FOR LINEAR REGRESSION

We can calculate this as follows (assume conditioning on x_0 and X),

$$\mathbb{E}[(y_0 - x_0^T \hat{w})^2] = \mathbb{E}[y_0^2] - 2\mathbb{E}[y_0]x_0^T \mathbb{E}[\hat{w}] + x_0^T \mathbb{E}[\hat{w}\hat{w}^T]x_0$$

- ▶ Since y_0 and \hat{w} are independent, $\mathbb{E}[y_0\hat{w}] = \mathbb{E}[y_0]\mathbb{E}[\hat{w}]$.
- ▶ Remember: $\mathbb{E}[\hat{w}\hat{w}^T] = \text{Var}[\hat{w}] + \mathbb{E}[\hat{w}]\mathbb{E}[\hat{w}]^T$
 $\mathbb{E}[y_0^2] = \sigma^2 + (x_0^T w)^2$

Plugging these values in:

$$\begin{aligned}\mathbb{E}[(y_0 - x_0^T \hat{w})^2] &= \sigma^2 + (x_0^T w)^2 - 2(x_0^T w)(x_0^T \mathbb{E}[\hat{w}]) + (x_0^T \mathbb{E}[\hat{w}])^2 + x_0^T \text{Var}[\hat{w}]x_0 \\ &= \sigma^2 + x_0^T (w - \mathbb{E}[\hat{w}]) (w - \mathbb{E}[\hat{w}])^T x_0 + x_0^T \text{Var}[\hat{w}]x_0\end{aligned}$$

BIAS-VARIANCE FOR LINEAR REGRESSION

We have shown that if

1. $y \sim N(Xw, \sigma^2)$ and $y_0 \sim N(x_0^T w, \sigma^2)$, and
2. we approximate w with \hat{w} according to some algorithm,

then

$$\mathbb{E}[(y_0 - x_0^T \hat{w})^2 | X, x_0] = \underbrace{\sigma^2}_{\text{noise}} + \underbrace{x_0^T (w - \mathbb{E}[\hat{w}]) (w - \mathbb{E}[\hat{w}])^T x_0}_{\text{squared bias}} + \underbrace{x_0^T \text{Var}[\hat{w}] x_0}_{\text{variance}}$$

We see that the *generalization error* is a combination of three factors:

1. Measurement noise – we can't control this given the model.
2. Model bias – how close to the solution we expect to be on average.
3. Model variance – how sensitive our solution is to the data.

We saw how we can find $\mathbb{E}[\hat{w}]$ and $\text{Var}[\hat{w}]$ for the LS and RR solutions.

BIAS-VARIANCE TRADE-OFF

This idea is more general:

- ▶ Imagine we have a model: $y = f(x; w) + \epsilon$, $\mathbb{E}(\epsilon) = 0$, $\text{Var}(\epsilon) = \sigma^2$
- ▶ We approximate f by minimizing a loss function: $\hat{f} = \arg \min_f \mathcal{L}_f$.
- ▶ We apply \hat{f} to new data, $y_0 \approx \hat{f}(x_0) \equiv \hat{f}_0$.

Then integrating everything out (y, X, y_0, x_0):

$$\begin{aligned}\mathbb{E}[(y_0 - \hat{f}_0)^2] &= \mathbb{E}[y_0^2] - 2\mathbb{E}[y_0 \hat{f}_0] + \mathbb{E}[\hat{f}_0^2] \\ &= \sigma^2 + f_0^2 - 2f_0 \mathbb{E}[\hat{f}_0] + \mathbb{E}[\hat{f}_0]^2 + \text{Var}[\hat{f}_0] \\ &= \underbrace{\sigma^2}_{\text{noise}} + \underbrace{(f_0 - \mathbb{E}[\hat{f}_0])^2}_{\text{squared bias}} + \underbrace{\text{Var}[\hat{f}_0]}_{\text{variance}}\end{aligned}$$

This is interesting in principle, but is deliberately vague (What is f ?) and usually can't be calculated (What is the distribution on the data?)

CROSS-VALIDATION

An easier way to evaluate the model is to use cross-validation.

The procedure for K -fold cross-validation is very simple:

1. Randomly split the data into K roughly equal groups.
2. Learn the model on $K - 1$ groups and predict the held-out K th group.
3. Do this K times, holding out each group once.
4. Evaluate performance using the cumulative set of predictions.

For the case of the regularization parameter λ , the above sequence can be run for several values with the best-performing value of λ chosen.

The data you test the model on should never be used to train the model!



BAYES RULE

PRIOR INFORMATION/BELIEF

Motivation

We've discussed the ridge regression objective function

$$\mathcal{L} = \sum_{i=1}^n (y_i - x_i^T w)^2 + \lambda w^T w.$$

The regularization term $\lambda w^T w$ was imposed to penalize values in w that are large. This reduced potential high-variance predictions from least squares.

In a sense, we are imposing a “prior belief” about what values of w we consider to be good.

Question: Is there a mathematical way to formalize this?

Answer: Using probability we can frame this via Bayes rule.

REVIEW: PROBABILITY STATEMENTS

Imagine we have two events, A and B , that may or may not be related, e.g.,

- ▶ A = “It is raining”
- ▶ B = “The ground is wet”

We can talk about probabilities of these events,

- ▶ $P(A)$ = Probability it is raining
- ▶ $P(B)$ = Probability the ground is wet

We can also talk about their *conditional* probabilities,

- ▶ $P(A|B)$ = Probability it is raining *given* that the ground is wet
- ▶ $P(B|A)$ = Probability the ground is wet *given* that it is raining

We can also talk about their *joint* probabilities,

- ▶ $P(A, B)$ = Probability it is raining *and* the ground is wet

CALCULUS OF PROBABILITY

There are simple rules from moving from one probability to another

1. $P(A, B) = P(A|B)P(B) = P(B|A)P(A)$
2. $P(A) = \sum_b P(A, B = b)$
3. $P(B) = \sum_a P(A = a, B)$

Using these three equalities, we automatically can say

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{P(B|A)P(A)}{\sum_a P(B|A = a)P(A = a)}$$

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} = \frac{P(A|B)P(B)}{\sum_b P(A|B = b)P(B = b)}$$

This is known as “Bayes rule.”

BAYES RULE

Bayes rule lets us quantify what we don't know. Imagine we want to say something about the probability of B given that A happened.

Bayes rule says that the probability of B after knowing A is:

$$\underbrace{P(B|A)}_{posterior} = \underbrace{P(A|B)}_{likelihood} \underbrace{P(B)}_{prior} / \underbrace{P(A)}_{marginal}$$

Notice that with this perspective, these probabilities take on new meanings.

That is, $P(B|A)$ and $P(A|B)$ are both “conditional probabilities,” but they have different significance.

BAYES RULE WITH CONTINUOUS VARIABLES

Bayes rule generalizes to continuous-valued random variables as follows. However, instead of *probabilities* we work with *densities*.

- ▶ Let θ be a continuous-valued model parameter.
- ▶ Let X be data we possess. Then by Bayes rule,

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta} = \frac{p(X|\theta)p(\theta)}{p(X)}$$

In this equation,

- ▶ $p(X|\theta)$ is the likelihood, known from the model definition.
- ▶ $p(\theta)$ is a prior distribution that we define.
- ▶ Given these two, we can (in principle) calculate $p(\theta|X)$.

EXAMPLE: COIN BIAS

We have a coin with bias π towards “heads”. (Encode: heads = 1, tails = 0)

We flip the coin many times and get a sequence of n numbers (x_1, \dots, x_n) . Assume the flips are independent, meaning

$$p(x_1, \dots, x_n | \pi) = \prod_{i=1}^n p(x_i | \pi) = \prod_{i=1}^n \pi^{x_i} (1 - \pi)^{1-x_i}.$$

We choose a prior for π which we define to be a beta distribution,

$$p(\pi) = Beta(\pi | a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \pi^{a-1} (1-\pi)^{b-1}.$$

What is the posterior distribution of π given x_1, \dots, x_n ?

EXAMPLE: COIN BIAS

From Bayes rule,

$$p(\pi|x_1, \dots, x_n) = \frac{p(x_1, \dots, x_n|\pi)p(\pi)}{\int_0^1 p(x_1, \dots, x_n|\pi)p(\pi)d\pi}.$$

There is a trick that is often useful:

- ▶ The denominator only normalizes the numerator, doesn't depend on π .
- ▶ We can write $p(\pi|x) \propto p(x|\pi)p(\pi)$. (" \propto " → "proportional to")
- ▶ Multiply the two and see if we recognize anything:

$$\begin{aligned} p(\pi|x_1, \dots, x_n) &\propto \left[\prod_{i=1}^n \pi^{x_i} (1-\pi)^{1-x_i} \right] \left[\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \pi^{a-1} (1-\pi)^{b-1} \right] \\ &\propto \pi^{\sum_{i=1}^n x_i + a - 1} (1-\pi)^{\sum_{i=1}^n (1-x_i) + b - 1} \end{aligned}$$

We recognize this as $p(\pi|x_1, \dots, x_n) = Beta(\sum_{i=1}^n x_i + a, \sum_{i=1}^n (1-x_i) + b)$.

MAXIMUM A POSTERIORI

LIKELIHOOD MODEL

Least squares and maximum likelihood

When we modeled data pairs (x_i, y_i) with a linear model, $y_i \approx x_i^T w$, we saw that the least squares solution,

$$w_{\text{LS}} = \arg \min_w (y - Xw)^T (y - Xw),$$

was equivalent to the maximum likelihood solution when $y \sim N(Xw, \sigma^2 I)$.

The question now is whether a similar probabilistic connection can be made for the ridge regression problem.

PRIOR MODEL

Ridge regression and Bayesian modeling

The likelihood model is $y \sim N(Xw, \sigma^2 I)$. What about a prior for w ?

Let us assume that the prior for w is Gaussian, $w \sim N(0, \lambda^{-1} I)$. Then

$$p(w) = \left(\frac{\lambda}{2\pi}\right)^{\frac{d}{2}} e^{-\frac{\lambda}{2} w^T w}.$$

We can now try to find a w that satisfies both the data likelihood, and our prior conditions about w .

MAXIMUM A POSERIORI ESTIMATION

Maximum *a posteriori* (MAP) estimation seeks the most probable value w under the posterior:

$$\begin{aligned} w_{\text{MAP}} &= \arg \max_w \ln p(w|y, X) \\ &= \arg \max_w \ln \frac{p(y|w, X)p(w)}{p(y|X)} \\ &= \arg \max_w \ln p(y|w, X) + \ln p(w) - \ln p(y|X) \end{aligned}$$

- ▶ Contrast this with ML, which only focuses on the likelihood.
- ▶ The normalizing constant term $\ln p(y|X)$ doesn't involve w . Therefore, we can maximize the first two terms alone.
- ▶ In many models we don't know $\ln p(y|X)$, so this fact is useful.

MAP FOR LINEAR REGRESSION

MAP using our defined prior gives:

$$\begin{aligned} w_{\text{MAP}} &= \arg \max_w \ln p(y|w, X) + \ln p(w) \\ &= \arg \max_w -\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw) - \frac{\lambda}{2}w^Tw + \text{const.} \end{aligned}$$

Calling this objective \mathcal{L} , then as before we find w such that

$$\nabla_w \mathcal{L} = \frac{1}{\sigma^2}X^Ty - \frac{1}{\sigma^2}X^TXw - \lambda w = 0$$

- ▶ The solution is $w_{\text{MAP}} = (\lambda\sigma^2I + X^TX)^{-1}X^Ty$.
- ▶ Notice that $w_{\text{MAP}} = w_{\text{RR}}$ (modulo a switch from λ to $\lambda\sigma^2$)
- ▶ RR maximizes the posterior, while LS maximizes the likelihood.

ColumbiaX: Machine Learning

Lecture 5

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

BAYESIAN LINEAR REGRESSION

Model

Have vector $y \in \mathbb{R}^n$ and covariates matrix $X \in \mathbb{R}^{n \times d}$. The i th row of y and X correspond to the i th observation (y_i, x_i) .

In a Bayesian setting, we model this data as:

$$\textbf{Likelihood} : \quad y \sim N(Xw, \sigma^2 I)$$

$$\textbf{Prior} : \quad w \sim N(0, \lambda^{-1} I)$$

The unknown model variable is $w \in \mathbb{R}^d$.

- ▶ The “likelihood model” says how well the observed data agrees with w .
- ▶ The “model prior” is our prior belief (or constraints) on w .

This is called Bayesian linear regression because we have defined a prior on the unknown parameter and will try to learn its posterior.

REVIEW: MAXIMUM A POSTERIORI INFERENCE

MAP solution

MAP inference returns the maximum of the log joint likelihood.

$$\textbf{Joint Likelihood} : \quad p(y, w|X) = p(y|w, X)p(w)$$

Using Bayes rule that this point also maximizes the *posterior* of w .

$$\begin{aligned} w_{\text{MAP}} &= \arg \max_w \ln p(w|y, X) \\ &= \arg \max_w \ln p(y|w, X) + \ln p(w) \\ &= \arg \max_w -\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw) - \frac{\lambda}{2}w^Tw + \text{const.} \end{aligned}$$

We saw that this solution for w_{MAP} is the same as for ridge regression:

$$w_{\text{MAP}} = (\lambda\sigma^2 I + X^T X)^{-1} X^T y \quad \Leftrightarrow \quad w_{\text{RR}}$$

POINT ESTIMATES VS BAYESIAN INFERENCE

Point estimates

w_{MAP} and w_{ML} are referred to as *point estimates* of the model parameters.

They find a specific value (point) of the vector w that maximizes an objective function (MAP or ML).

- ▶ **ML:** Only consider data model: $p(y|w, X)$.
- ▶ **MAP:** Takes into account model prior: $p(y, w|X) = p(y|w, X)p(w)$.

Bayesian inference

Bayesian inference goes one step further by characterizing uncertainty about the values in w using Bayes rule.

BAYES RULE AND LINEAR REGRESSION

Posterior calculation

Since w is a continuous-valued random variable in \mathbb{R}^d , Bayes rule says that the *posterior* distribution of w given y, X is

$$p(w|y, X) = \frac{p(y|w, X)p(w)}{\int_{\mathbb{R}^d} p(y|w, X)p(w) dw}$$

That is, we get an updated distribution on w through the transition

prior \rightarrow likelihood \rightarrow posterior

Quote: “The posterior of is proportional to the likelihood times the prior.”

FULLY BAYESIAN INFERENCE

Bayesian linear regression

In this case, we can update the posterior distribution $p(w|y, X)$ analytically.

We work with the proportionality first:

$$\begin{aligned} p(w|y, X) &\propto p(y|w, X)p(w) \\ &\propto \left[e^{-\frac{1}{2\sigma^2}(y-Xw)^T(y-Xw)} \right] \left[e^{-\frac{\lambda}{2}w^Tw} \right] \\ &\propto e^{-\frac{1}{2}\{w^T(\lambda I + \sigma^{-2}X^T X)w - 2\sigma^{-2}w^T X^T y\}} \end{aligned}$$

The \propto sign lets us multiply and divide this by anything *as long as it doesn't contain w*. We've done this in two lines above.

BAYESIAN INFERENCE FOR LINEAR REGRESSION

We need to normalize:

$$p(w|y, X) \propto e^{-\frac{1}{2}\{w^T(\lambda I + \sigma^{-2}X^T X)w - 2w^T X^T y\}}$$

There are two key terms in the exponent:

$$\underbrace{w^T(\lambda I + \sigma^{-2}X^T X)w}_{\text{quadratic in } w} - \underbrace{2w^T X^T y / \sigma^2}_{\text{linear in } w}$$

We can conclude that $p(w|y, X)$ is Gaussian. Why?

1. We can multiply and divide by anything not involving w .
2. A Gaussian has $(w - \mu)^T \Sigma^{-1} (w - \mu)$ in the exponent.
3. We can “complete the square” by adding terms not involving w .

BAYESIAN INFERENCE FOR LINEAR REGRESSION

Compare: In other words, a Gaussian looks like:

$$p(w|\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(w^T \Sigma^{-1} w - 2w^T \Sigma^{-1} \mu + \mu^T \Sigma^{-1} \mu)}$$

and we've shown for some setting of Z that

$$p(w|y, X) = \frac{1}{Z} e^{-\frac{1}{2}(w^T (\lambda I + \sigma^{-2} X^T X) w - 2w^T X^T y / \sigma^2)}$$

Conclude: What happens if in the above Gaussian we define:

$$\Sigma^{-1} = (\lambda I + \sigma^{-2} X^T X), \quad \Sigma^{-1} \mu = X^T y / \sigma^2 ?$$

Using these specific values of μ and Σ we only need to set

$$Z = (2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}} e^{\frac{1}{2} \mu^T \Sigma^{-1} \mu}$$

BAYESIAN INFERENCE FOR LINEAR REGRESSION

The posterior distribution

Therefore, the posterior distribution of w is:

$$p(w|y, X) = N(w|\mu, \Sigma),$$

$$\Sigma = (\lambda I + \sigma^{-2} X^T X)^{-1},$$

$$\mu = (\lambda \sigma^2 I + X^T X)^{-1} X^T y \Leftarrow w_{\text{MAP}}$$

Things to notice:

- ▶ $\mu = w_{\text{MAP}}$ after a redefinition of the regularization parameter λ .
- ▶ Σ captures uncertainty about w as $\text{Var}[w_{\text{LS}}]$ and $\text{Var}[w_{\text{RR}}]$ did before.
- ▶ However, now we have a full probability distribution on w .

USES OF THE POSTERIOR DISTRIBUTION

Understanding w

We saw how we could calculate the variance of w_{LS} and w_{RR} . Now we have an entire distribution. Some questions we can ask are:

Q: Is $w_i > 0$ or $w_i < 0$? Can we confidently say $w_i \neq 0$?

A: Use the *marginal posterior distribution*: $w_i \sim N(\mu_i, \Sigma_{ii})$.

Q: How do w_i and w_j relate?

A: Use their joint marginal posterior distribution:

$$\begin{bmatrix} w_i \\ w_j \end{bmatrix} \sim N \left(\begin{bmatrix} \mu_i \\ \mu_j \end{bmatrix}, \begin{bmatrix} \Sigma_{ii} & \Sigma_{ij} \\ \Sigma_{ji} & \Sigma_{jj} \end{bmatrix} \right)$$

Predicting new data

The posterior $p(w|y, X)$ is perhaps most useful for predicting new data.

PREDICTING NEW DATA

PREDICTING NEW DATA

Recall: For a new pair (x_0, y_0) with x_0 measured and y_0 unknown, we can predict y_0 using x_0 and the LS or RR (i.e., ML or MAP) outputs:

$$y_0 \approx x_0^T w_{\text{LS}} \quad \text{or} \quad y_0 \approx x_0^T w_{\text{RR}}$$

With Bayes rule, we can make a *probabilistic* statement about y_0 :

$$\begin{aligned} p(y_0|x_0, y, X) &= \int_{\mathbb{R}^d} p(y_0, w|x_0, y, X) dw \\ &= \int_{\mathbb{R}^d} p(y_0|w, x_0, y, X) p(w|x_0, y, X) dw \end{aligned}$$

Notice that *conditional independence* lets us write

$$p(y_0|w, x_0, y, X) = \underbrace{p(y_0|w, x_0)}_{\text{likelihood}} \quad \text{and} \quad p(w|x_0, y, X) = \underbrace{p(w|y, X)}_{\text{posterior}}$$

PREDICTING NEW DATA

Predictive distribution (intuition)

This is called the *predictive distribution*:

$$p(y_0|x_0, y, X) = \int_{\mathbb{R}^d} \underbrace{p(y_0|x_0, w)}_{likelihood} \underbrace{p(w|y, X)}_{posterior} dw$$

Intuitively, we evaluate the likelihood of a new y_0 for a particular w and observed x_0 , and weight it by our current belief about w given data (y, X) .

We then sum (integrate) over all possible values of w .

PREDICTING NEW DATA

We know from the model and Bayes rule that

$$\begin{aligned}\text{Model: } p(y_0|x_0, w) &= N(y_0|x_0^T w, \sigma^2), \\ \text{Bayes rule: } p(w|y, X) &= N(w|\mu, \Sigma).\end{aligned}$$

With μ and Σ calculated on a previous slide.

The predictive distribution can be calculated exactly with these distributions.
Again we get a Gaussian distribution:

$$\begin{aligned}p(y_0|x_0, y, X) &= N(y_0|\mu_0, \sigma_0^2), \\ \mu_0 &= x_0^T \mu, \\ \sigma_0^2 &= \sigma^2 + x_0^T \Sigma x_0.\end{aligned}$$

Notice that the expected value is the MAP prediction since $\mu = x_0^T w_{\text{MAP}}$, but we now quantify our confidence in this prediction with the variance σ_0^2 .

ACTIVE LEARNING

PRIOR → POSTERIOR → PRIOR

Bayesian learning is naturally thought of as a sequential process. That is, the posterior after seeing some data becomes the prior for the next data.

Let y and X be “old data” and y_0 and x_0 be some “new data”. By Bayes rule

$$p(w|y_0, x_0, y, X) \propto p(y_0|w, x_0)p(w|y, X).$$

The posterior after (y, X) has become the prior for (y_0, x_0) .

Simple modifications can be made sequentially:

$$p(w|y_0, x_0, y, X) = N(w|\mu, \Sigma),$$

$$\Sigma = (\lambda I + \sigma^{-2}(x_0 x_0^T + \sum_{i=1}^n x_i x_i^T))^{-1},$$

$$\mu = (\lambda \sigma^2 I + (x_0 x_0^T + \sum_{i=1}^n x_i x_i^T)^{-1}(x_0 y_0 + \sum_{i=1}^n x_i y_i)).$$

INTELLIGENT LEARNING

Of course, we could also have written

$$p(w|y_0, x_0, y, X) \propto p(y_0, y|w, X, x_0)p(w)$$

but often we want to use the sequential aspect of inference to help us learn.

Learning w and making predictions for new y_0 is a two-step procedure:

- ▶ Form the predictive distribution $p(y_0|x_0, y, X)$.
- ▶ Update the posterior distribution $p(w|y, X, y_0, x_0)$.

Question: Can we learn $p(w|y, X)$ intelligently?

That is, if we're in the situation where we can pick which y_i to measure with the knowledge of $\mathcal{D} = \{x_1, \dots, x_n\}$, can we come up with a good strategy?

ACTIVE LEARNING

An “active learning” strategy

Imagine we already have a measured dataset (y, X) and posterior $p(w|y, X)$. We can construct the predictive distribution for every remaining $x_0 \in \mathcal{D}$.

$$\begin{aligned} p(y_0|x_0, y, X) &= N(y_0|\mu_0, \sigma_0^2), \\ \mu_0 &= x_0^T \mu, \\ \sigma_0^2 &= \sigma^2 + x_0^T \Sigma x_0. \end{aligned}$$

For each x_0 , σ_0^2 tells how confident we are. This suggests the following:

1. Form predictive distribution $p(y_0|x_0, y, X)$ for all unmeasured $x_0 \in \mathcal{D}$
2. Pick the x_0 for which σ_0^2 is largest and measure y_0
3. Update the posterior $p(w|y, X)$ where $y \leftarrow (y, y_0)$ and $X \leftarrow (X, x_0)$
4. Return to #1 using the updated posterior

ACTIVE LEARNING

Entropy (i.e., uncertainty) minimization

When devising a procedure such as this one, it's useful to know what *objective function* is being optimized in the process.

We introduce the concept of the *entropy* of a distribution. Let $p(z)$ be a continuous distribution, then its (differential) entropy is:

$$\mathcal{H}(p) = - \int p(z) \ln p(z) dz.$$

This is a measure of the spread of the distribution. Larger values correspond to a more “uncertain” distribution (more variance).

The entropy of a multivariate Gaussian is

$$\mathcal{H}(N(w|\mu, \Sigma)) = \frac{d}{2} \ln \left(2\pi e |\Sigma| \right).$$

ACTIVE LEARNING

The entropy of a Gaussian changes with its covariance matrix. With sequential Bayesian learning, the covariance transitions from

$$\begin{array}{ccc} \text{Prior : } & (\lambda I + \sigma^{-2} X^T X)^{-1} & \equiv \Sigma \\ & \Downarrow & \\ \text{Posterior : } & (\lambda I + \sigma^{-2} (x_0 x_0^T + X^T X))^{-1} & \equiv (\Sigma^{-1} + \sigma^{-2} x_0 x_0^T)^{-1} \end{array}$$

Using a rank-one update property of the determinant, the entropy of the prior $\mathcal{H}_{\text{prior}}$ is related to the entropy of the posterior $\mathcal{H}_{\text{post}}$ as follows:

$$\mathcal{H}_{\text{post}} = \mathcal{H}_{\text{prior}} - \frac{d}{2} \ln(1 + \sigma^{-2} x_0^T \Sigma x_0)$$

Therefore, the x_0 that minimizes $\mathcal{H}_{\text{post}}$ also maximizes $\sigma^2 + x_0^T \Sigma x_0$. We are minimizing \mathcal{H} myopically, so this is called a “greedy algorithm”.

MODEL SELECTION

SELECTING λ

We've discussed λ as a "nuisance" parameter that can impact performance.

Bayes rule gives a principled way to do this via *evidence maximization*:

$$p(w|y, X, \lambda) = \underbrace{p(y|w, X)}_{likelihood} \underbrace{p(w|\lambda)}_{prior} / \underbrace{p(y|X, \lambda)}_{evidence}.$$

The "evidence" gives the likelihood of the data with w integrated out. It's a measure of how good our model and parameter assumptions are.

SELECTING λ

If we want to set λ , we can also do it by maximizing the evidence.

$$\hat{\lambda} = \arg \max_{\lambda} \ln p(y|X, \lambda).$$

We can show that the distribution of y is $p(y|X, \lambda) = N(y|0, \sigma^2 I + \lambda^{-1} X^T X)$. This requires an algorithm to maximize over λ .

We notice that this looks exactly like maximum likelihood, and it is:

Type-I ML: Maximize the likelihood over the “main parameter” (w).

Type-II ML: Integrate out “main parameter” (w) and maximize over the “hyperparameter” (λ). Also called *empirical Bayes*.

The difference is only in their perspective.

This approach requires that we can solve this integral, but often we can't for more complex models. Cross-validation is the method that always works.

ColumbiaX: Machine Learning

Lecture 6

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

UNDERDETERMINED LINEAR EQUATIONS

We now consider the regression problem $y = Xw$ where $X \in \mathbb{R}^{n \times d}$ is “fat” (i.e., $d \gg n$). This is called an “underdetermined” problem.

- ▶ There are more dimensions than observations.
- ▶ w now has an infinite number of solutions satisfying $y = Xw$.

$$\begin{bmatrix} y \\ \vdots \end{bmatrix} = \begin{bmatrix} & & & \\ & X & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} w \\ \vdots \end{bmatrix}$$

These sorts of high-dimensional problems often come up:

- ▶ In gene analysis there are 1000's of genes but only 100's of subjects.
- ▶ Images can have millions of pixels.
- ▶ Even polynomial regression can quickly lead to this scenario.

MINIMUM ℓ_2 REGRESSION

ONE SOLUTION (LEAST NORM)

One possible solution to the underdetermined problem is

$$w_{\text{ln}} = X^T(XX^T)^{-1}y \quad \Rightarrow \quad Xw_{\text{ln}} = XX^T(XX^T)^{-1}y = y.$$

We can construct another solution by adding to w_{ln} a vector $\delta \in \mathbb{R}^d$ that is in the *null space* \mathcal{N} of X :

$$\delta \in \mathcal{N}(X) \quad \Rightarrow \quad X\delta = 0 \text{ and } \delta \neq 0$$

and so $X(w_{\text{ln}} + \delta) = Xw_{\text{ln}} + X\delta = y + 0$.

In fact, there are an infinite number of possible δ , because $d > n$.

We can show that w_{ln} is the solution with smallest ℓ_2 norm. We will use the proof of this fact as an excuse to introduce two general concepts.

TOOLS: ANALYSIS

We can use *analysis* to prove that w_{ln} satisfies the optimization problem

$$w_{\text{ln}} = \arg \min_w \|w\|^2 \quad \text{subject to} \quad Xw = y.$$

(Think of mathematical analysis as the use of inequalities to prove things.)

Proof: Let w be another solution to $Xw = y$, and so $X(w - w_{\text{ln}}) = 0$. Also,

$$\begin{aligned}(w - w_{\text{ln}})^T w_{\text{ln}} &= (w - w_{\text{ln}})^T X^T (X X^T)^{-1} y \\&= \underbrace{(X(w - w_{\text{ln}}))^T}_{= 0} (X X^T)^{-1} y = 0\end{aligned}$$

As a result, $w - w_{\text{ln}}$ is *orthogonal* to w_{ln} . It follows that

$$\|w\|^2 = \|w - w_{\text{ln}} + w_{\text{ln}}\|^2 = \|w - w_{\text{ln}}\|^2 + \|w_{\text{ln}}\|^2 + 2 \underbrace{(w - w_{\text{ln}})^T w_{\text{ln}}}_{= 0} > \|w_{\text{ln}}\|^2$$

TOOLS: LAGRANGE MULTIPLIERS

Instead of starting from the solution, start from the problem,

$$w_{\text{ln}} = \arg \min_w w^T w \quad \text{subject to} \quad Xw = y.$$

- ▶ Introduce Lagrange multipliers: $\mathcal{L}(w, \eta) = w^T w + \eta^T (Xw - y)$.
- ▶ Minimize \mathcal{L} over w maximize over η . If $Xw \neq y$, we can get $\mathcal{L} = +\infty$.
- ▶ The optimal conditions are

$$\nabla_w \mathcal{L} = 2w + X^T \eta = 0, \quad \nabla_\eta \mathcal{L} = Xw - y = 0.$$

We have everything necessary to find the solution:

1. From first condition: $w = -X^T \eta / 2$
2. Plug into second condition: $\eta = -2(XX^T)^{-1}y$
3. Plug this back into #1: $w_{\text{ln}} = X^T(XX^T)^{-1}y$

SPARSE ℓ_1 REGRESSION

LS AND RR IN HIGH DIMENSIONS

Usually not suited for high-dimensional data

- ▶ Modern problems: Many dimensions/features/predictors
- ▶ Only a few of these may be important or relevant for predicting y
- ▶ Therefore, we need some form of “feature selection”

- ▶ Least squares and ridge regression:
 - ▶ Treat all dimensions equally without favoring subsets of dimensions
 - ▶ The relevant dimensions are averaged with irrelevant ones
 - ▶ Problems: Poor generalization to new data, interpretability of results

REGRESSION WITH PENALTIES

Penalty terms

Recall: General ridge regression is of the form

$$\mathcal{L} = \sum_{i=1}^n (y_i - f(x_i; w))^2 + \lambda \|w\|^2$$

We've referred to the term $\|w\|^2$ as a *penalty term* and used $f(x_i; w) = x_i^T w$.

Penalized fitting

The general structure of the optimization problem is

total cost = goodness-of-fit term + penalty term

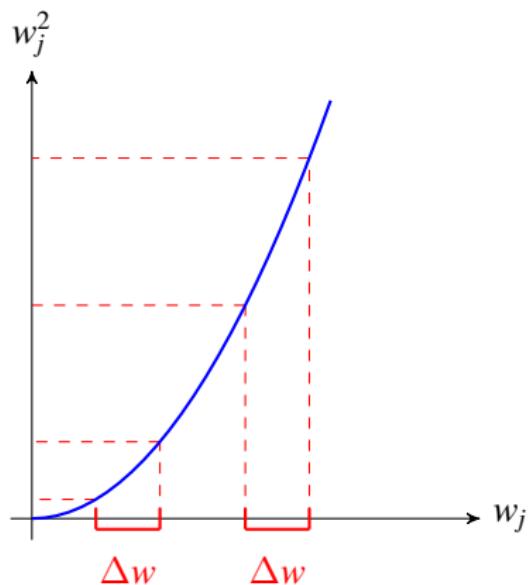
- ▶ Goodness-of-fit measures how well our model f approximates the data.
- ▶ Penalty term makes the solutions we don't want more “expensive”.

What kind of solutions does the choice $\|w\|^2$ favor or discourage?

QUADRATIC PENALTIES

Intuitions

- ▶ Quadratic penalty: Reduction in cost depends on $|w_j|$.
- ▶ Suppose we reduce w_j by Δw .
The effect on \mathcal{L} depends on the starting point of w_j .
- ▶ Consequence: We should favor vectors w whose entries are of similar size, preferably small.



SPARSITY

Setting

- ▶ Regression problem with n data points $x \in \mathbb{R}^d$, $d \gg n$.
- ▶ Goal: Select a small subset of the d dimensions and switch off the rest.
- ▶ This is sometimes referred to as “feature selection”.

What does it mean to “switch off” a dimension?

- ▶ Each entry of w corresponds to a dimension of the data x .
- ▶ If $w_k = 0$, the prediction is

$$f(x, w) = x^T w = w_1 x_1 + \cdots + 0 \cdot x_k + \cdots + w_d x_d,$$

so the prediction does not depend on the k th dimension.

- ▶ Feature selection: Find a w that (1) predicts well, and (2) has only a small number of non-zero entries.
- ▶ A w for which most dimensions = 0 is called a *sparse* solution.

SPARSITY AND PENALTIES

Penalty goal

Find a penalty term which encourages sparse solutions.

Quadratic penalty vs sparsity

- ▶ Suppose w_k is large, all other w_j are very small but non-zero
- ▶ Sparsity: Penalty should keep w_k , and push other w_j to zero
- ▶ Quadratic penalty: Will favor entries w_j which all have similar size, and so it will push w_k towards small value.

Overall, a quadratic penalty favors many small, but non-zero values.

Solution

Sparsity can be achieved using *linear* penalty terms.

LASSO

Sparse regression

LASSO: Least Absolute Shrinkage and Selection Operator

With the LASSO, we replace the ℓ_2 penalty with an ℓ_1 penalty:

$$w_{\text{lasso}} = \arg \min_w \|y - Xw\|_2^2 + \lambda \|w\|_1$$

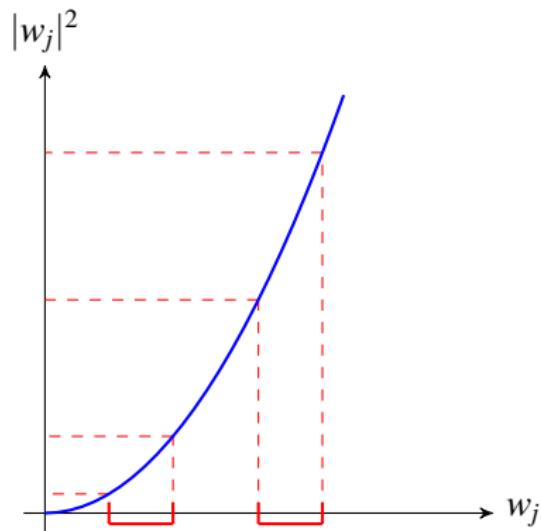
where

$$\|w\|_1 = \sum_{j=1}^d |w_j|.$$

This is also called ℓ_1 -regularized regression.

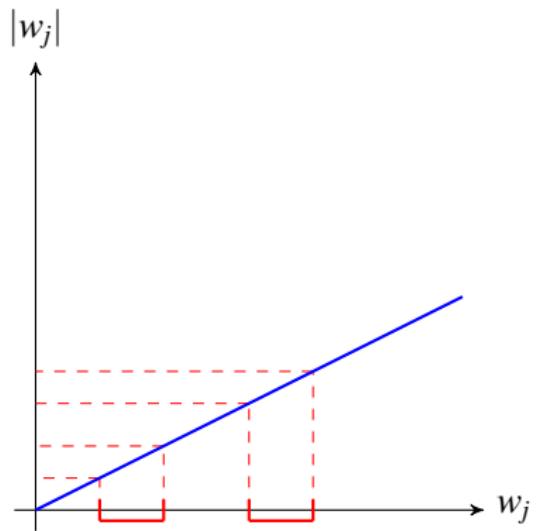
QUADRATIC PENALTIES

Quadratic penalty



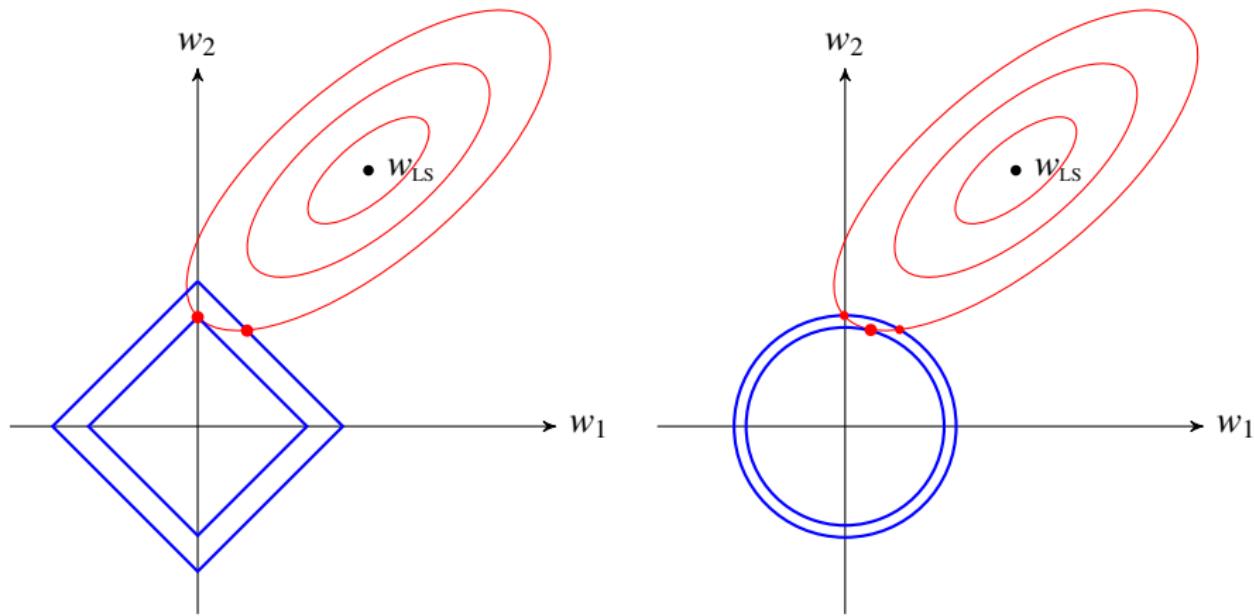
Reducing a large value w_j achieves a larger cost reduction.

Linear penalty



Cost reduction does not depend on the magnitude of w_j .

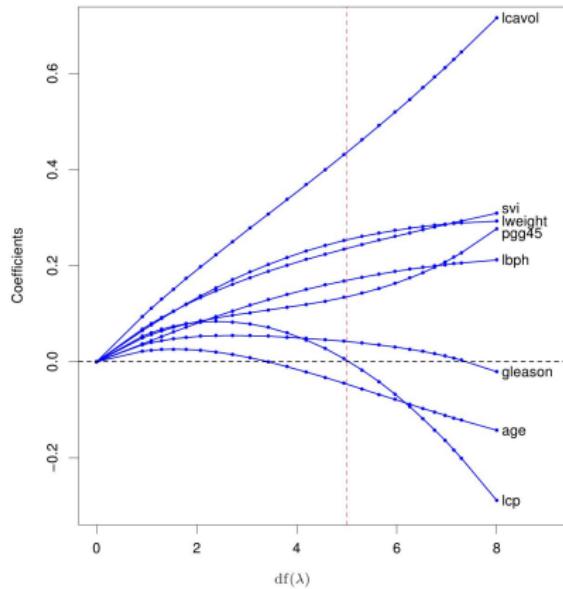
RIDGE REGRESSION VS LASSO



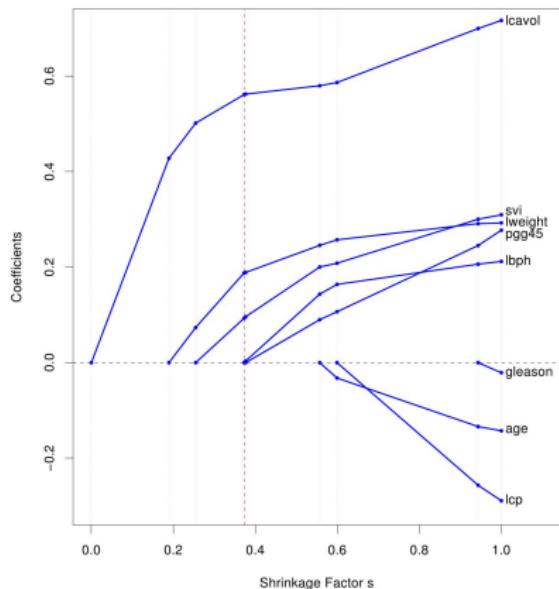
This figure applies to $d < n$, but gives intuition for $d \gg n$.

- ▶ Red: Contours of $(w - w_{LS})^T (X^T X) (w - w_{LS})$ (see Lecture 3)
- ▶ Blue: (left) Contours of $\|w\|_1$, and (right) contours of $\|w\|_2^2$

COEFFICIENT PROFILES: RR VS LASSO



(a) $\|w\|_2$ penalty



(b) $\|w\|_1$ penalty

ℓ_p REGRESSION

ℓ_p -norms

These norm-penalties can be extended to all norms:

$$\|w\|_p = \left(\sum_{j=1}^d |w_j|^p \right)^{\frac{1}{p}} \quad \text{for } 0 < p \leq \infty$$

ℓ_p -regression

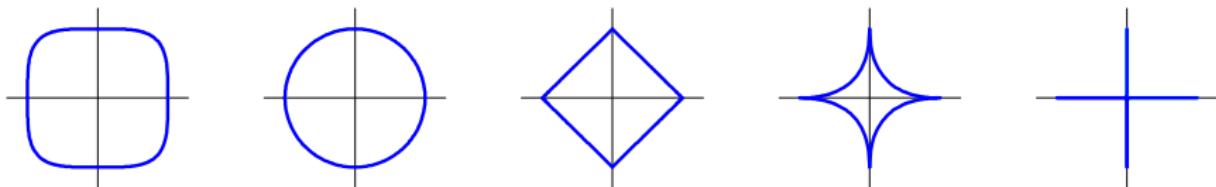
The ℓ_p -regularized linear regression problem is

$$w_{\ell_p} := \arg \min_w \|y - Xw\|_2^2 + \lambda \|w\|_p^p$$

We have seen:

- ▶ ℓ_1 -regression = LASSO
- ▶ ℓ_2 -regression = ridge regression

ℓ_p PENALIZATION TERMS



$$p = 4$$

$$p = 2$$

$$p = 1$$

$$p = 0.5$$

$$p = 0.1$$

p	Behavior of $\ \cdot\ _p$
$p = \infty$	Norm measures largest absolute entry, $\ w\ _\infty = \max_j w_j $
$p > 2$	Norm focuses on large entries
$p = 2$	Large entries are expensive; encourages similar-size entries
$p = 1$	Encourages sparsity
$p < 1$	Encourages sparsity as for $p = 1$, but contour set is not convex (i.e., no “line of sight” between every two points inside the shape)
$p \rightarrow 0$	Simply records whether an entry is non-zero, i.e. $\ w\ _0 = \sum_j \mathbb{I}\{w_j \neq 0\}$

COMPUTING THE SOLUTION FOR ℓ_p

Solution of ℓ_p problem

ℓ_2 aka ridge regression. Has a closed form solution

ℓ_p ($p \geq 1, p \neq 2$) — By “convex optimization”. We won’t discuss convex analysis in detail in this class, but two facts are important

- ▶ There are no “local optimal solutions” (i.e., local minimum of \mathcal{L})
- ▶ The true solution can be found *exactly* using iterative algorithms

($p < 1$) — We can only find an approximate solution (i.e., the best in its “neighborhood”) using iterative algorithms.

Three techniques formulated as optimization problems

Method	Good-o-fit	penalty	Solution method
Least squares	$\ y - Xw\ _2^2$	none	Analytic solution exists if $X^T X$ invertible
Ridge regression	$\ y - Xw\ _2^2$	$\ w\ _2^2$	Analytic solution exists always
LASSO	$\ y - Xw\ _2^2$	$\ w\ _1$	Numerical optimization to find solution

ColumbiaX: Machine Learning

Lecture 7

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

CLASSIFICATION

TERMINOLOGY AND NOTATION

Input: As with regression, in a *classification problem* we start with measurements x_1, \dots, x_n in an input space \mathcal{X} . (Again think $\mathcal{X} = \mathbb{R}^d$)

Output: The *discrete* output space \mathcal{Y} is composed of K possible *classes*:

- ▶ $\mathcal{Y} = \{-1, +1\}$ or $\{0, 1\}$ is called binary classification.
- ▶ $\mathcal{Y} = \{1, \dots, K\}$ is called multiclass classification

Instead of a real-valued response, classification assigns x to a category.

- ▶ Regression: For pair (x, y) , y is the response of x .
- ▶ Classification: For pair (x, y) , y is the class of x .

CLASSIFICATION PROBLEM

Defining a classifier

Classification uses a function f (called a *classifier*) to map input x to class y .

$$y = f(x) : f \text{ takes in } x \in \mathcal{X} \text{ and declares its class to be } y \in \mathcal{Y}$$

As with regression, the problem is two-fold:

- ▶ Define the classifier f and its parameters.
- ▶ Learn the classification rule using a training set of “labeled data.”

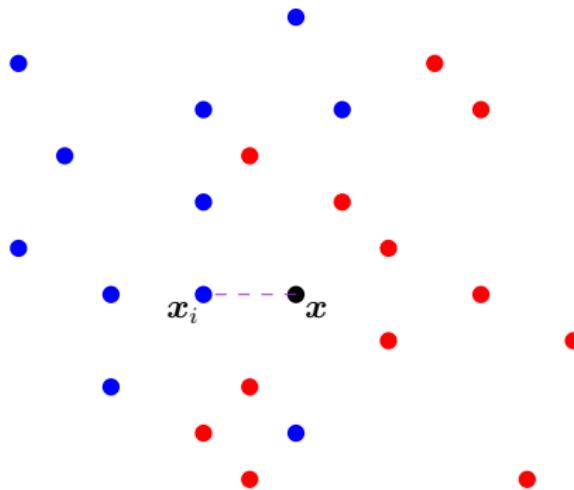
NEAREST NEIGHBOR CLASSIFIERS

NEAREST NEIGHBOR (NN) CLASSIFIER

Given data $(x_1, y_1), \dots, (x_n, y_n)$, construct classifier $\hat{f}(x) \rightarrow y$ as follows:

For an input x not in the training data,

1. Let x_i be the point among x_1, x_2, \dots, x_n that is “closest” to x .
2. Return its label y_i .



DISTANCES

Question: How should we measure distance between points?

The default distance for data in \mathbb{R}^d is the Euclidean one:

$$\|u - v\|_2 = \left(\sum_{i=1}^d (u_i - v_i)^2 \right)^{\frac{1}{2}} \quad (\text{line-of-sight distance})$$

But there are other options that may sometimes be better:

- ▶ ℓ_p for $p \in [1, \infty]$: $\|u - v\|_p = \left(\sum_{i=1}^d |u_i - v_i|^p \right)^{\frac{1}{p}}$.
- ▶ Edit distance (for strings): How many add/delete/substitutions are required to transform one string to the other.
- ▶ Correlation distance (for signal): Measures how correlated two vectors are for signal detection.

EXAMPLE: OCR WITH NN CLASSIFIER

- ▶ **Handwritten digits data:** grayscale 28×28 images, treated as vectors in \mathbb{R}^{784} , with labels indicating the digit they represent.



- ▶ Split into training set \mathcal{S} (60K points) and testing set \mathcal{T} (10K points).
- ▶ **Training error:** $\text{err}(\hat{f}, \mathcal{S}) = 0 \leftarrow$ declare its class to be its own class!
Test error: $\text{err}(\hat{f}, \mathcal{T}) = 0.0309 \leftarrow$ using ℓ_2 distance
- ▶ Examples of mistakes: (left) test point, (right) nearest neighbor in \mathcal{S} :



- ▶ **Observation:** First mistake might have been avoided by looking at three nearest neighbors (whose labels are '8', '2', '2') ...



test point three nearest neighbors

k -NEAREST NEIGHBORS CLASSIFIER

Given data $(x_1, y_1), \dots, (x_n, y_n)$, construct the k -NN classifier as follows:

For a new input x ,

1. Return the k points closest to x , indexed as x_{i_1}, \dots, x_{i_k} .
2. Return the majority-vote of $y_{i_1}, y_{i_2}, \dots, y_{i_k}$.

(Break ties in both steps arbitrarily.)

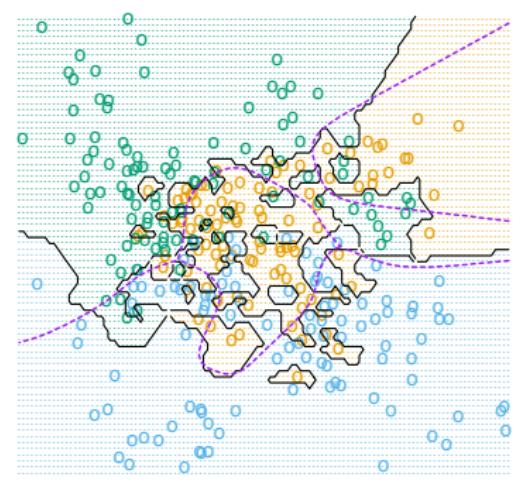
Example: OCR with k -NN classifier

k	1	3	5	7	9
$\text{err}(\hat{f}_k, T)$	0.0309	0.0295	0.0312	0.0306	0.0341

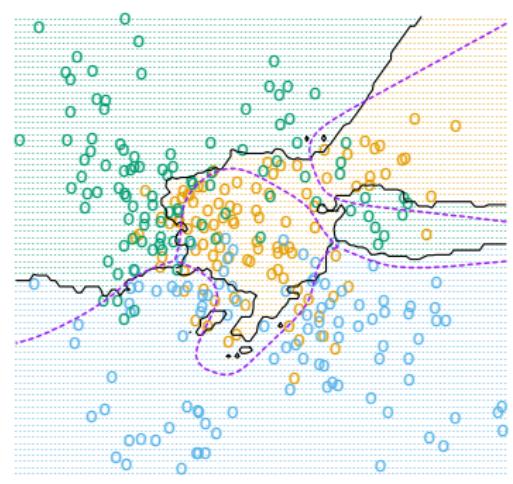
EFFECT OF k

In general:

- ▶ Smaller $k \Rightarrow$ smaller training error.
- ▶ Larger $k \Rightarrow$ predictions are more “stable” due to voting.



1-NN



15-NN

Purple dotted lines : Can ignore for now.

Black solid lines : k -NN’s decision boundaries.

STATISTICAL SETTING

How do we measure the quality of a classifier?

For any classifier we care about two sides of the same coin:

- ▶ Prediction accuracy: $P(f(x) = y)$.
- ▶ Prediction error: $\text{err}(f) = P(f(x) \neq y)$.

To calculate these values, we assume there is a distribution \mathcal{P} over the space of labeled examples generating the data

$$(x_i, y_i) \stackrel{iid}{\sim} \mathcal{P}, \quad i = 1, \dots, n.$$

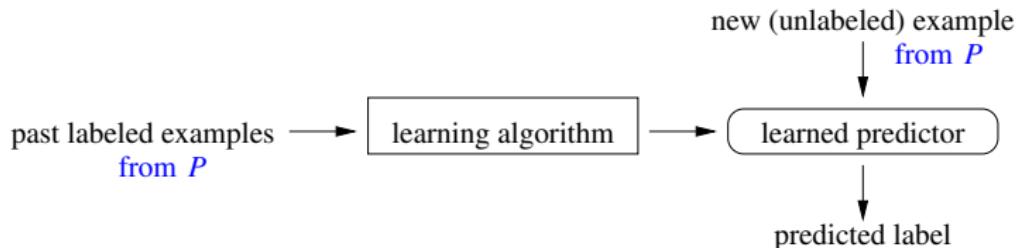
We don't know what \mathcal{P} is, but can still talk about it in abstract terms.

STATISTICAL LEARNING

When is there any hope for finding an accurate classifier?

Key assumption: Data $(x_1, y_1), \dots, (x_n, y_n)$ are i.i.d. random labeled examples with distribution \mathcal{P} .

This assumption allows us to say that the past should look like the future.



Regression makes similar assumptions.

BAYES CLASSIFIERS

OPTIMAL CLASSIFIERS

Can we talk about what an “optimal” classifier looks like?

Assume that $(X, Y) \stackrel{iid}{\sim} \mathcal{P}$. (Again, we don’t know \mathcal{P})

Some probability equalities with \mathcal{P} :

1. The expectation of an indicator of an event is the probability of the event, e.g.,

$$\mathbb{E}_P[\mathbb{1}(Y = 1)] = P(Y = 1), \quad \leftarrow \mathbb{1}(\cdot) = 0 \text{ or } 1 \text{ depending if } \cdot \text{ is true}$$

2. Conditional expectations can be random variables, and their expectations remove the randomness,

$C = \mathbb{E}[A | B]$: A and B are *both* random, so C is random

$\mathbb{E}[C] = \mathbb{E}[\mathbb{E}[A | B]] = \mathbb{E}[A]$ “tower property” of expectation

OPTIMAL CLASSIFIERS

For any classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$, its prediction error is

$$P(f(X) \neq Y) = \mathbb{E}[\mathbf{1}(f(X) \neq Y)] = \mathbb{E}\left[\underbrace{\mathbb{E}[\mathbf{1}(f(X) \neq Y) | X]}_{\text{a random variable}}\right] \quad (\dagger)$$

OPTIMAL CLASSIFIERS

For any classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$, its prediction error is

$$P(f(X) \neq Y) = \mathbb{E}[\mathbb{1}(f(X) \neq Y)] = \mathbb{E}\left[\underbrace{\mathbb{E}[\mathbb{1}(f(X) \neq Y) | X]}_{\text{a random variable}}\right] \quad (\dagger)$$

For each $x \in \mathcal{X}$,

$$\mathbb{E}[\mathbb{1}(f(X) \neq Y) | X = x] = \sum_{y \in \mathcal{Y}} P(Y = y | X = x) \cdot \mathbb{1}(f(x) \neq y), \quad (\ddagger)$$

OPTIMAL CLASSIFIERS

For any classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$, its prediction error is

$$P(f(X) \neq Y) = \mathbb{E}[\mathbb{1}(f(X) \neq Y)] = \mathbb{E}\left[\underbrace{\mathbb{E}[\mathbb{1}(f(X) \neq Y) | X]}_{\text{a random variable}}\right] \quad (\dagger)$$

For each $x \in \mathcal{X}$,

$$\mathbb{E}[\mathbb{1}(f(X) \neq Y) | X = x] = \sum_{y \in \mathcal{Y}} P(Y = y | X = x) \cdot \mathbb{1}(f(x) \neq y), \quad (\ddagger)$$

The above quantity (\ddagger) is minimized for this particular $x \in \mathcal{X}$ when

$$f(x) = \arg \max_{y \in \mathcal{Y}} P(Y = y | X = x). \quad (\star)$$

The classifier f with property (\star) for all $x \in \mathcal{X}$ is called the *Bayes classifier*, and it has the smallest prediction error (\dagger) among *all classifiers*.

THE BAYES CLASSIFIER

Under the assumption $(X, Y) \stackrel{iid}{\sim} \mathcal{P}$, the optimal classifier is

$$f^*(x) := \arg \max_{y \in \mathcal{Y}} P(Y = y | X = x).$$

From Bayes rule we equivalently have

$$f^*(x) = \arg \max_{y \in \mathcal{Y}} \underbrace{P(Y = y)}_{\text{class prior}} \times \underbrace{P(X = x | Y = y)}_{\text{data likelihood} | \text{class}}.$$

- ▶ $P(Y = y)$ is called the *class prior*.
- ▶ $P(X = x | Y = y)$ is called the *class conditional distribution* of X .
- ▶ In practice we don't know either of these, so we approximate them.

Aside: If X is a continuous-valued random variable, replace $P(X = x | Y = y)$ with *class conditional density* $p(x | Y = y)$.

EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES

Suppose $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \{0, 1\}$, and the distribution \mathcal{P} of (X, Y) is as follows.

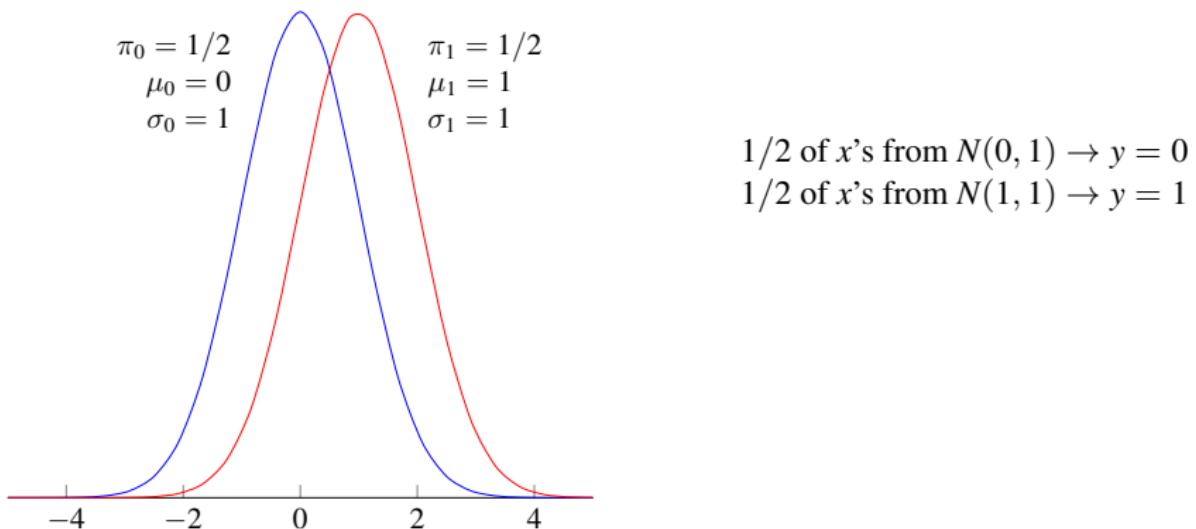
- ▶ **Class prior:** $P(Y = y) = \pi_y, \quad y \in \{0, 1\}.$
- ▶ **Class conditional density** for class $y \in \{0, 1\}$: $p_y(x) = N(x|\mu_y, \sigma_y^2).$
- ▶ **Bayes classifier:**

$$\begin{aligned} f^\star(x) &= \operatorname{argmax}_{y \in \{0,1\}} p(X = x|Y = y)P(Y = y) \\ &= \begin{cases} 1 & \text{if } \frac{\pi_1}{\sigma_1} \exp\left[-\frac{(x - \mu_1)^2}{2\sigma_1^2}\right] > \frac{\pi_0}{\sigma_0} \exp\left[-\frac{(x - \mu_0)^2}{2\sigma_0^2}\right] \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

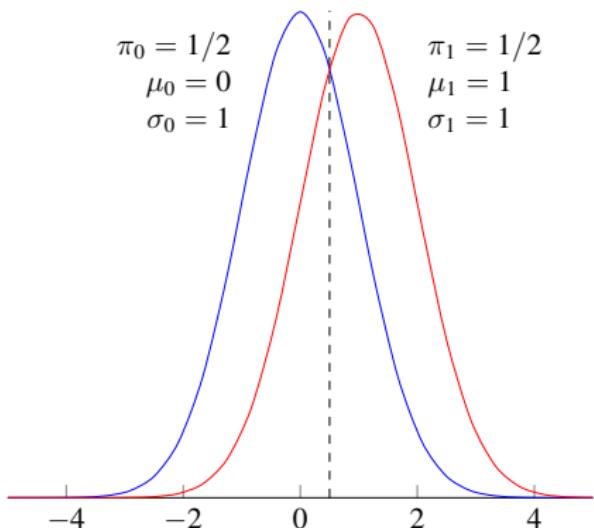
This type of classifier is called a *generative* model.

- ▶ **Generative model:** Model x and y with distributions.
- ▶ **Discriminative model:** Plug x into a distribution on y (used thus far).

EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES



EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES

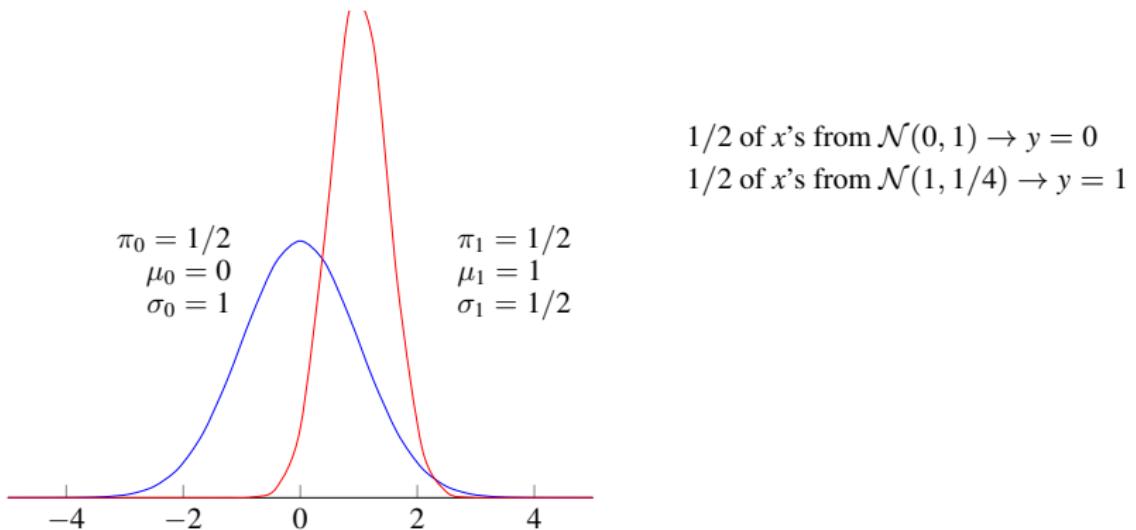


1/2 of x 's from $N(0, 1) \rightarrow y = 0$
1/2 of x 's from $N(1, 1) \rightarrow y = 1$

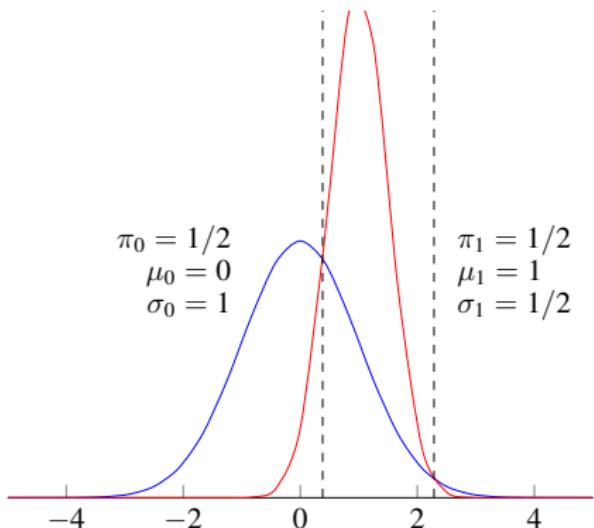
Bayes classifier:

$$f^*(x) = \begin{cases} 1 & \text{if } x > 1/2; \\ 0 & \text{otherwise.} \end{cases}$$

EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES



EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES



1/2 of x 's from $\mathcal{N}(0, 1) \rightarrow y = 0$
1/2 of x 's from $\mathcal{N}(1, 1/4) \rightarrow y = 1$

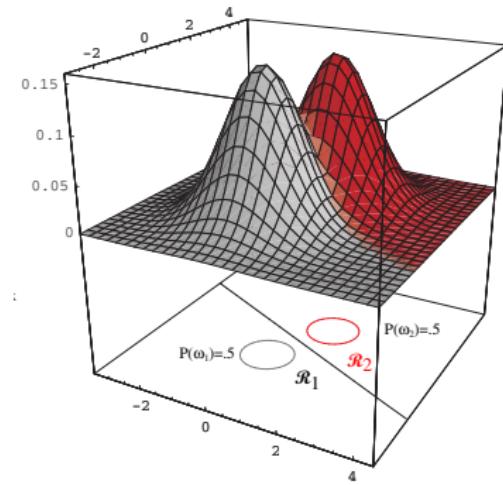
Bayes classifier:

$$f^*(x) = \begin{cases} 1 & \text{if } x \in [0.38, 2.29]; \\ 0 & \text{otherwise.} \end{cases}$$

EXAMPLE: MULTIVARIATE GAUSSIANS

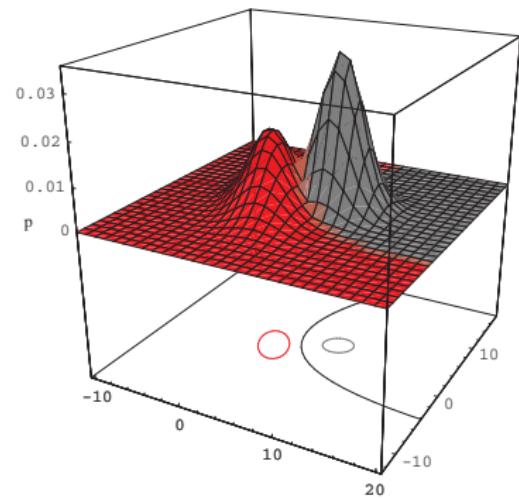
Data: $\mathcal{X} = \mathbb{R}^2$, Label: $\mathcal{Y} = \{0, 1\}$

Class conditional densities are Gaussians in \mathbb{R}^2 with covariance Σ_0 and Σ_1 .



$$\Sigma_0 = \Sigma_1$$

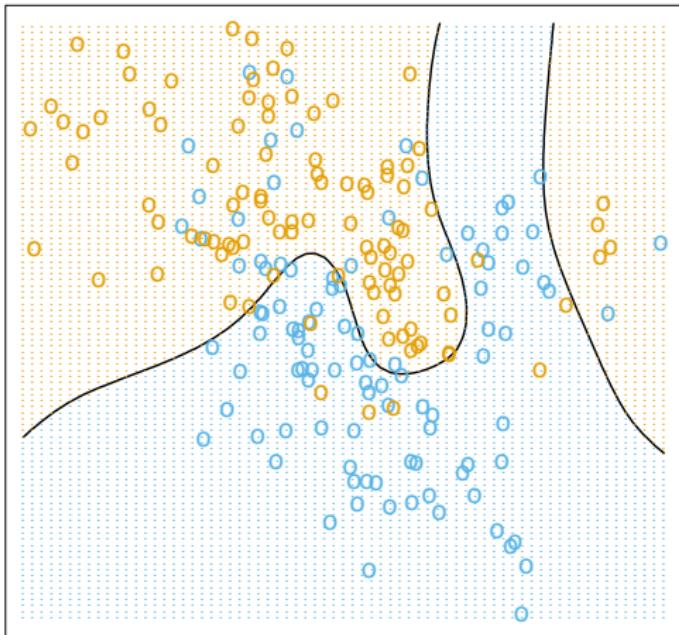
Bayes classifier:
linear separator



$$\Sigma_0 \neq \Sigma_1$$

Bayes classifier:
quadratic separator

BAYES CLASSIFIER IN GENERAL



In general, the Bayes classifier may be rather complicated!
This one uses more than a single Gaussian for the class-conditional density.

PLUG-IN CLASSIFIERS

Bayes classifier

The Bayes classifier has the smallest prediction error of *all* classifiers.

Problem: We can't construct the Bayes classifier without knowing \mathcal{P} .

- ▶ What is $P(Y = y|X = x)$, or equiv., $P(X = x|Y = y)$ and $P(Y = y)$?
- ▶ All we have are labeled examples drawn from the distribution \mathcal{P} .

Plug-in classifiers

Use the available data to approximate $P(Y = y)$ and $P(X = x|Y = y)$.

- ▶ Of course, the result may no longer give the best results among all the classifiers we can choose from (e.g., k -NN and those discussed later).

EXAMPLE: GAUSSIAN CLASS CONDITIONAL DENSITIES

Here, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, \dots, K\}$. Estimate Bayes classifier via MLE:

- ▶ **Class priors:** The MLE estimate of π_y is $\hat{\pi}_y = \frac{1}{n} \sum_{i=1}^n \mathbb{1}(y_i = y)$.
- ▶ **Class conditional density:** Choose $p(x|Y = y) = N(x|\mu_y, \Sigma_y)$.
The MLE estimate of (μ_y, Σ_y) is

$$\hat{\mu}_y = \frac{1}{n_y} \sum_{i=1}^n \mathbb{1}(y_i = y) x_i,$$

$$\hat{\Sigma}_y = \frac{1}{n_y} \sum_{i=1}^n \mathbb{1}(y_i = y) (x_i - \hat{\mu}_y)(x_i - \hat{\mu}_y)^T.$$

This is just the empirical mean and covariance of class y .

- ▶ **Plug-in classifier:**

$$\hat{f}(x) = \arg \max_{y \in \mathcal{Y}} \hat{\pi}_y |\hat{\Sigma}_y|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (x - \hat{\mu}_y)^T \hat{\Sigma}_y^{-1} (x - \hat{\mu}_y) \right\}.$$

EXAMPLE: SPAM FILTERING

Representing emails

- ▶ **Input:** x , a vector of word counts. For example, if index $\{j \rightarrow \text{"car"}\}$ $x(j) = 3$ means that the word “car” occurs three times in the email.
- ▶ **Output:** $\mathcal{Y} = \{-1, +1\}$. Map {email $\rightarrow -1$, spam $\rightarrow +1$ }

Example dimensions

	george	you	your	hp	free	work	!	our	re	click	remove
spam	0	4	1	0	4	0	5	5	1	3	2
email	1	3	4	1	1	4	0	1	1	0	0

Using a Bayes classifier

$$f(x) = \operatorname{argmax}_{y \in \{-1, +1\}} p(x|Y=y)P(Y=y)$$

NAIVE BAYES

We have to *define* $p(X = x|Y = y)$.

Simplifying assumption

Naive Bayes is a Bayes classifier that makes the assumption

$$p(X = x|Y = y) = \prod_{j=1}^d p_j(x(j)|Y = y),$$

i.e., it treats the dimensions of X as *conditionally independent* given y .

In spam example

- ▶ Correlations between words is ignored.
- ▶ Can help make it easier to define the distribution.

ESTIMATION

Class prior

The distribution $P(Y = y)$ is again easy to estimate from the training data:

$$P(Y = y) = \frac{\#\text{observations in class } y}{\#\text{observations}}$$

Class-conditional distributions

For the spam model we define

$$P(X = x|Y = y) = \prod_j p_j(x(j)|Y = y) = \prod_j \text{Poisson}(x(j)|\lambda_j^{(y)})$$

We then approximate each $\lambda_j^{(y)}$ from the data. For example, the MLE is

$$\lambda_j^{(y)} = \frac{\#\text{unique uses of word } j \text{ in observations from class } y}{\#\text{observations in class } y}$$

ColumbiaX: Machine Learning

Lecture 8

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

LINEAR CLASSIFICATION

BINARY CLASSIFICATION

We focus on binary classification, with input $x_i \in \mathbb{R}^d$ and output $y_i \in \{\pm 1\}$.

- ▶ We define a *classifier* f , which makes prediction $y_i = f(x_i, \Theta)$ based on a function of x_i and parameters Θ . In other words $f : \mathbb{R}^d \rightarrow \{-1, +1\}$.

Last lecture, we discussed the **Bayes classification** framework.

- ▶ Here, Θ contains:
 - (1) class prior probabilities on y ,
 - (2) parameters for class-dependent distribution on x .

This lecture we'll introduce the **linear classification** framework.

- ▶ In this approach the prediction is linear in the parameters Θ .
- ▶ In fact, there is an intersection between the two that we discuss next.

A BAYES CLASSIFIER

Bayes decisions

With the Bayes classifier we predict the class of a new x to be the most probable label given the model and training data $(x_1, y_1), \dots, (x_n, y_n)$.

In the binary case, we declare class $y = 1$ if

$$p(x|y=1) \underbrace{P(y=1)}_{\pi_1} > p(x|y=0) \underbrace{P(y=0)}_{\pi_0}$$
$$\Updownarrow$$

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} > 0$$

This second line is referred to as the *log odds*.

A BAYES CLASSIFIER

Gaussian with shared covariance

Let's look at the log odds for the special case where

$$p(x|y) = N(x|\mu_y, \Sigma)$$

(i.e., a single Gaussian with a shared covariance matrix)

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} = \underbrace{\ln \frac{\pi_1}{\pi_0} - \frac{1}{2}(\mu_1 + \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0)}_{\text{a constant, call it } w_0} + x^T \underbrace{\Sigma^{-1}(\mu_1 - \mu_0)}_{\text{a vector, call it } w}$$

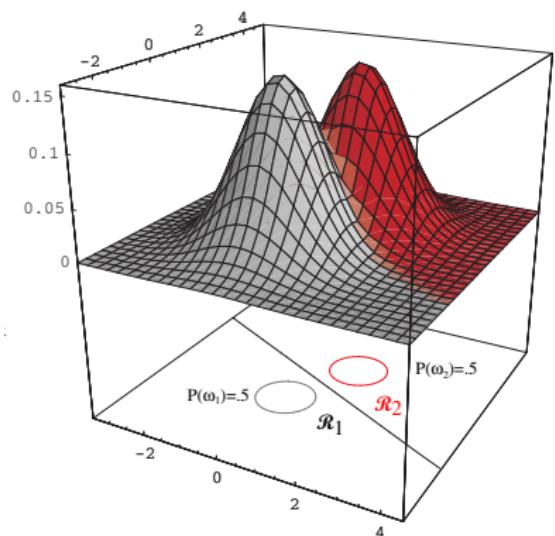
This is also called “linear discriminant analysis” (used to be called LDA).

A BAYES CLASSIFIER

So we can write the decision rule for this Bayes classifier as a linear one:

$$f(x) = \text{sign}(x^T w + w_0).$$

- ▶ This is what we saw last lecture
(but now class 0 is called -1)
- ▶ The Bayes classifier produced a linear decision boundary in the data space when $\Sigma_1 = \Sigma_0$.
- ▶ w and w_0 are obtained through a specific equation.



LINEAR CLASSIFIERS

This Bayes classifier is one instance of a linear classifier

$$f(x) = \text{sign}(x^T w + w_0)$$

where

$$\begin{aligned}w_0 &= \ln \frac{\pi_1}{\pi_0} - \frac{1}{2}(\mu_1 + \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0) \\w &= \Sigma^{-1}(\mu_1 - \mu_0)\end{aligned}$$

With MLE used to find values for π_y, μ_y and Σ .

Setting w_0 and w this way may be too restrictive:

- ▶ This Bayes classifier assumes single Gaussian with shared covariance.
- ▶ Maybe if we relax what values w_0 and w can take we can do better.

LINEAR CLASSIFIERS (BINARY CASE)

Definition: Binary linear classifier

A *binary linear classifier* is a function of the form

$$f(x) = \text{sign}(x^T w + w_0),$$

where $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$. Since the goal is to learn w, w_0 from data, we are assuming that *linear separability* in x is an accurate property of the classes.

Definition: Linear separability

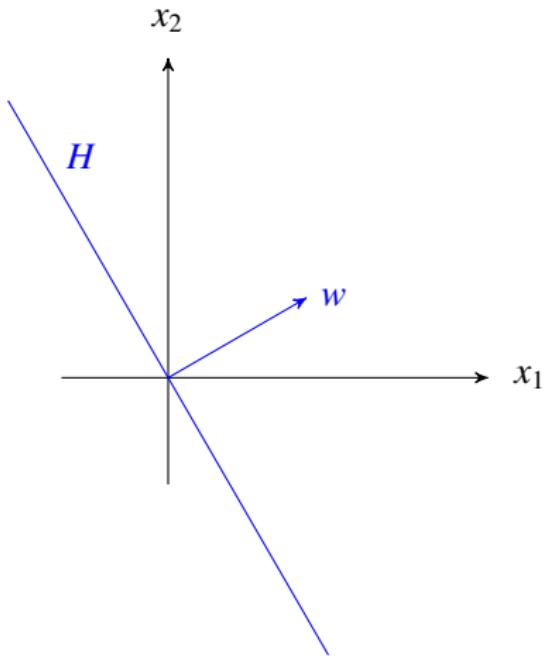
Two sets $A, B \subset \mathbb{R}^d$ are called *linearly separable* if

$$x^T w + w_0 \quad \begin{cases} > 0 & \text{if } x \in A \text{ (e.g, class +1)} \\ < 0 & \text{if } x \in B \text{ (e.g, class -1)} \end{cases}$$

The pair (w, w_0) defines an *affine hyperplane*. It is important to develop the right geometric understanding about what this is doing.

HYPERPLANES

Geometric interpretation of linear classifiers:



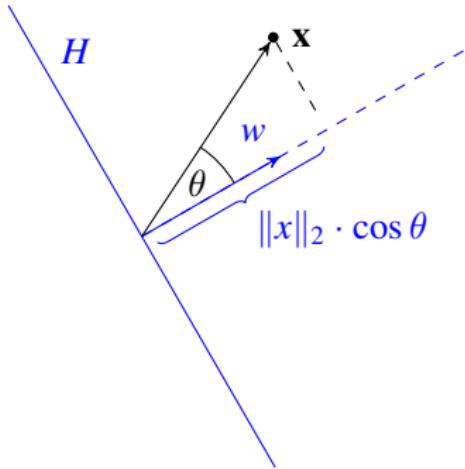
A *hyperplane* in \mathbb{R}^d is a linear subspace of dimension $(d - 1)$.

- ▶ A \mathbb{R}^2 -hyperplane is a line.
- ▶ A \mathbb{R}^3 -hyperplane is a plane.
- ▶ As a linear subspace, a hyperplane always contains the origin.

A hyperplane H can be represented by a vector w as follows:

$$H = \left\{ x \in \mathbb{R}^d \mid x^T w = 0 \right\} .$$

WHICH SIDE OF THE PLANE ARE WE ON?



Distance from the plane

- ▶ How close is a point x to H ?
- ▶ Cosine rule: $x^T w = \|x\|_2 \|w\|_2 \cos \theta$
- ▶ The distance of x to the hyperplane is

$$\|x\|_2 \cdot |\cos \theta| = |x^T w| / \|w\|_2.$$

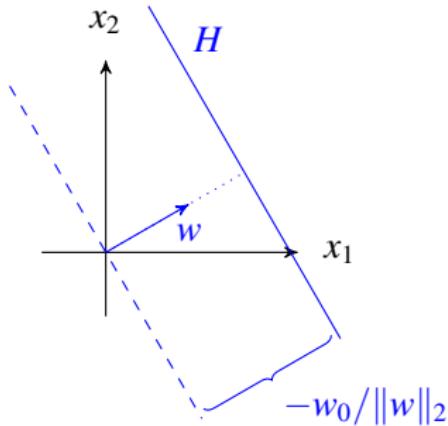
So $|x^T w|$ gives a sense of distance.

Which side of the hyperplane?

- ▶ The cosine satisfies $\cos \theta > 0$ if $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$.
- ▶ So the sign of $\cos(\cdot)$ tells us the side of H , and by the cosine rule

$$\text{sign}(\cos \theta) = \text{sign}(x^T w).$$

AFFINE HYPERPLANES



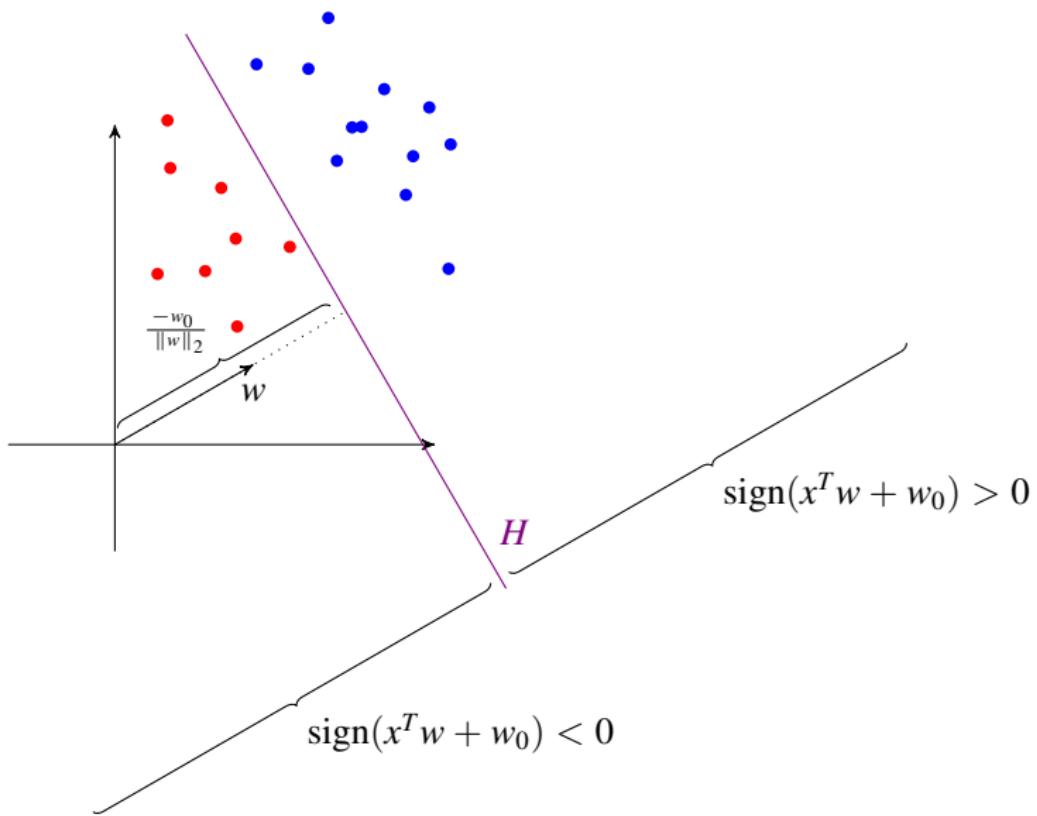
Affine Hyperplanes

- ▶ An *affine hyperplane* H is a hyperplane translated (shifted) using a scalar w_0 .
- ▶ Think of: $H = x^T w + w_0 = 0$.
- ▶ Setting $w_0 > 0$ moves the hyperplane in the *opposite* direction of w . ($w_0 < 0$ in figure)

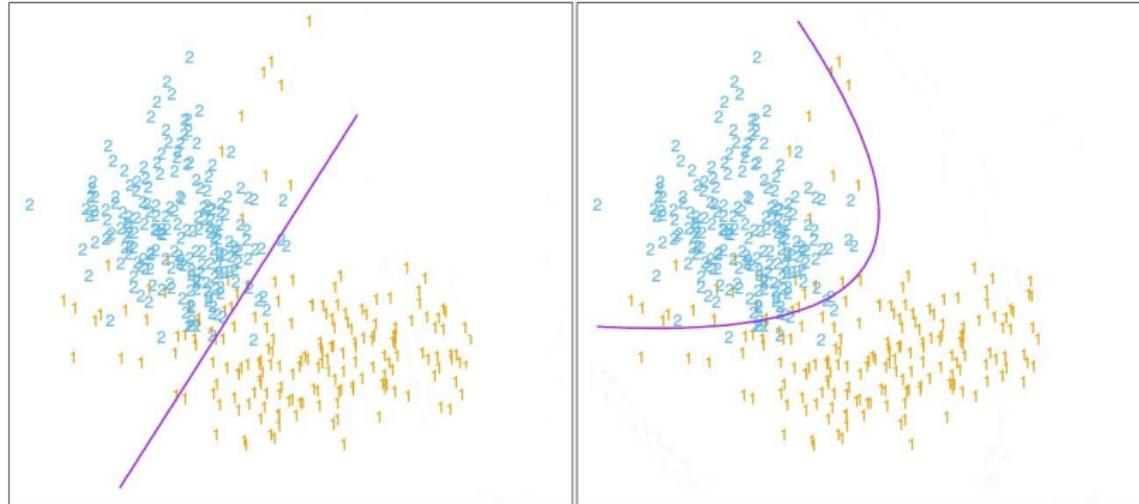
Which side of the hyperplane now?

- ▶ The plane has been shifted by distance $\frac{-w_0}{\|w\|_2}$ in the direction w .
- ▶ For a given w , w_0 and input x the inequality $x^T w + w_0 > 0$ says that x is on the far side of an affine hyperplane H in the direction w points.

CLASSIFICATION WITH AFFINE HYPERPLANES



POLYNOMIAL GENERALIZATIONS



The same generalizations from regression also hold for classification:

- ▶ (left) A linear classifier using $x = (x_1, x_2)$.
- ▶ (right) A linear classifier using $x = (x_1, x_2, x_1^2, x_2^2)$.
The decision boundary is linear in \mathbb{R}^4 , but isn't when plotted in \mathbb{R}^2 .

ANOTHER BAYES CLASSIFIER

Gaussian with different covariance

Let's look at the log odds for the general case where $p(x|y) = N(x|\mu_y, \Sigma_y)$ (i.e., now each class has its own covariance)

$$\ln \frac{p(x|y=1)P(y=1)}{p(x|y=0)P(y=0)} = \underbrace{\text{something complicated not involving } x}_{\text{a constant}} + \underbrace{x^T(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0)}_{\text{a part that's linear in } x} + \underbrace{x^T(\Sigma_0^{-1}/2 - \Sigma_1^{-1}/2)x}_{\text{a part that's quadratic in } x}$$

Also called “quadratic discriminant analysis,” but it’s *linear* in the weights.

ANOTHER BAYES CLASSIFIER

- ▶ We also saw this last lecture.

- ▶ Notice that

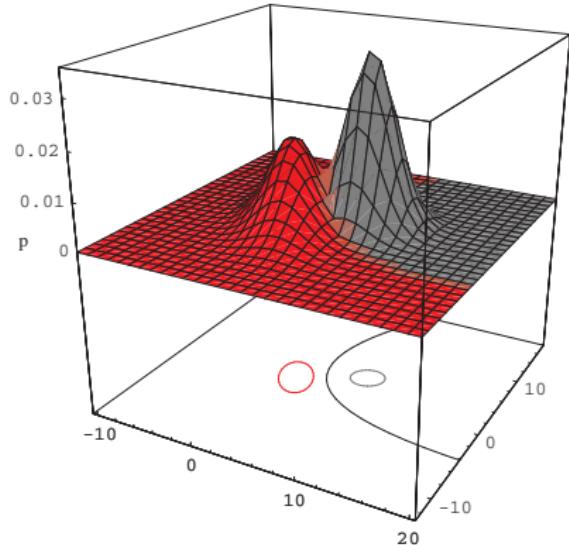
$$f(x) = \text{sign}(x^T A x + x^T b + c)$$

is linear in A, b, c .

- ▶ When $x \in \mathbb{R}^2$, rewrite as

$$x \leftarrow (x_1, x_2, 2x_1x_2, x_1^2, x_2^2)$$

and do linear classification in \mathbb{R}^5 .



Whereas the Bayes classifier with shared covariance is a version of linear classification, using different covariances is like polynomial classification.

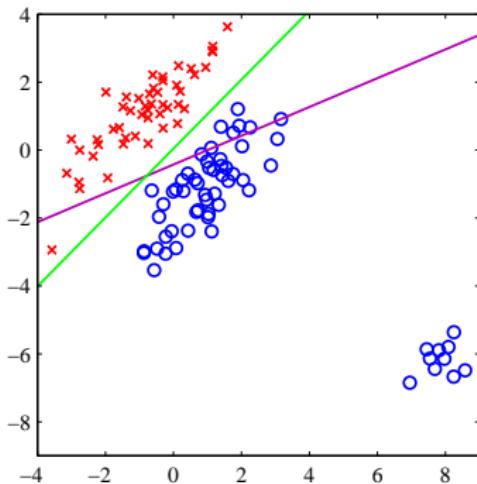
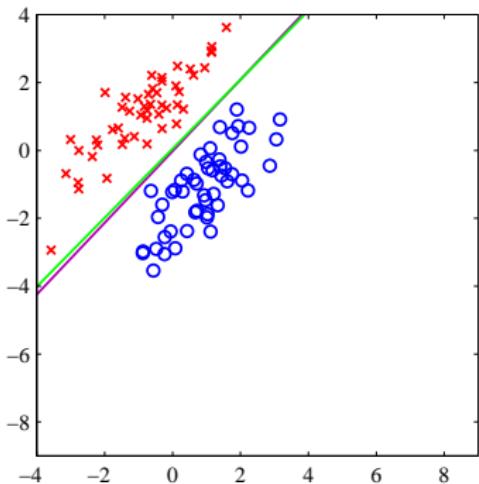
LEAST SQUARES ON $\{-1, +1\}$

How do we define more general classifiers of the form

$$f(x) = \text{sign}(x^T w + w_0) ?$$

- ▶ One simple idea is to treat classification as a regression problem:
 1. Let $y = (y_1, \dots, y_n)^T$, where $y_i \in \{-1, +1\}$ is the class of x_i .
 2. Add dimension equal to 1 to x_i and construct the matrix $X = [x_1, \dots, x_n]^T$.
 3. Learn the least squares weight vector $w = (X^T X)^{-1} X^T y$.
 4. For a new point x_0 declare $y_0 = \text{sign}(x_0^T w) \leftarrow w_0$ is included in w .
- ▶ Another option: Instead of LS, use ℓ_p regularization.
- ▶ These are “baseline” options. We can use them, along with k -NN, to get a quick sense what performance we’re aiming to beat.

SENSITIVITY TO OUTLIERS

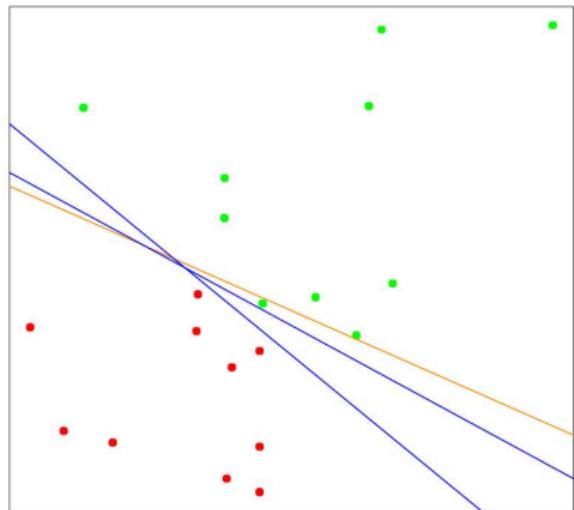


Least squares can do well, but it is sensitive to outliers. In general we can find better classifiers that focus more on the decision boundary.

- ▶ (left) Least squares (purple) does well compared with another method
- ▶ (right) Least squares does poorly because of outliers

THE PERCEPTRON ALGORITHM

EASY CASE: LINEARLY SEPARABLE DATA



(Assume data x_i has a 1 attached.)

Suppose there is a linear classifier with zero *training* error:

$$y_i = \text{sign}(x_i^T w), \text{ for all } i.$$

Then the data is “linearly separable”

Left: Can separate classes with a line.
(Can find an infinite number of lines.)

PERCEPTRON (ROSENBLATT, 1958)



Using the linear classifier

$$y = f(x) = \text{sign}(x^T w),$$

the Perceptron seeks to minimize

$$\mathcal{L} = - \sum_{i=1}^n (y_i \cdot x_i^T w) \mathbb{1}\{y_i \neq \text{sign}(x_i^T w)\}.$$

Because $y \in \{-1, +1\}$,

$$y_i \cdot x_i^T w \quad \text{is} \quad \begin{cases} > 0 & \text{if } y_i = \text{sign}(x_i^T w) \\ < 0 & \text{if } y_i \neq \text{sign}(x_i^T w) \end{cases}$$

By minimizing \mathcal{L} we're trying to always predict the correct label.

LEARNING THE PERCEPTRON

- ▶ Unlike other techniques we've talked about, we can't find the minimum of \mathcal{L} by taking a derivative and setting to zero:

$$\nabla_w \mathcal{L} = 0 \text{ cannot be solved for } w \text{ analytically.}$$

However $\nabla_w \mathcal{L}$ does tell us the direction in which \mathcal{L} is *increasing* in w .

- ▶ Therefore, for a sufficiently small η , if we update

$$w' \leftarrow w - \eta \nabla_w \mathcal{L},$$

then $\mathcal{L}(w') < \mathcal{L}(w)$ — i.e., we have a better value for w .

- ▶ This is a very general method for optimizing an objective functions called **gradient descent**. Perceptron uses a “stochastic” version of this.

LEARNING THE PERCEPTRON

Input: Training data $(x_1, y_1), \dots, (x_n, y_n)$ and a positive step size η

1. Set $w^{(1)} = \vec{0}$

2. For step $t = 1, 2, \dots$ do

a) Search for all examples $(x_i, y_i) \in \mathcal{D}$ such that $y_i \neq \text{sign}(x_i^T w^{(t)})$

b) If such a (x_i, y_i) exists, randomly pick one and update

$$w^{(t+1)} = w^{(t)} + \eta y_i x_i,$$

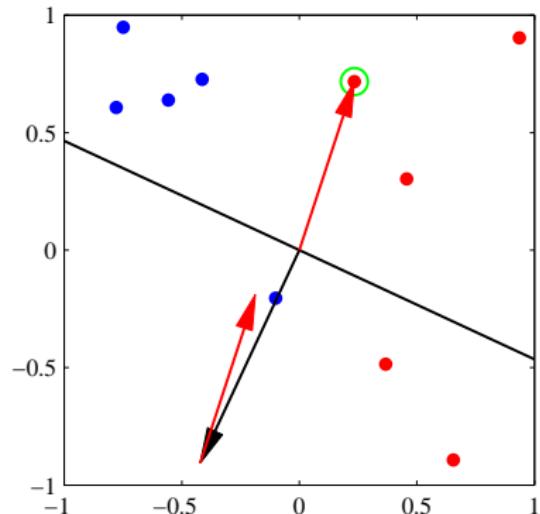
Else: Return $w^{(t)}$ as the solution since everything is classified correctly.

If \mathcal{M}_t indexes the misclassified observations at step t , then we have

$$\mathcal{L} = - \sum_{i=1}^n (y_i \cdot x_i^T w) \mathbb{1}\{y_i \neq \text{sign}(x_i^T w)\}, \quad \nabla_w \mathcal{L} = - \sum_{i \in \mathcal{M}_t} y_i x_i.$$

The full gradient step is $w^{(t+1)} = w^{(t)} - \eta \nabla_w \mathcal{L}$. Stochastic optimization just picks out one element in $\nabla_w \mathcal{L}$ —we could have also used the full summation.

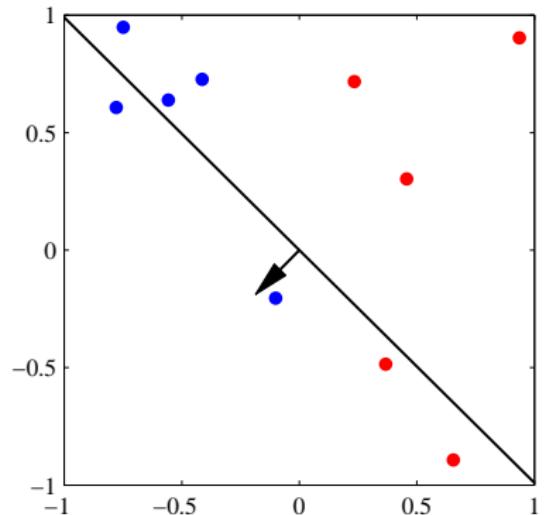
LEARNING THE PERCEPTRON



red = +1, blue = -1, $\eta = 1$

1. Pick a misclassified (x_i, y_i)
2. Set $w \leftarrow w + \eta y_i x_i$

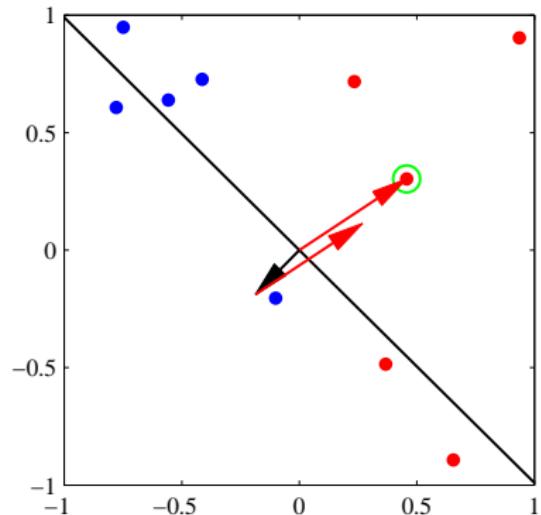
LEARNING THE PERCEPTRON



red = +1, blue = -1, $\eta = 1$

The update to w defines a new decision boundary (hyperplane)

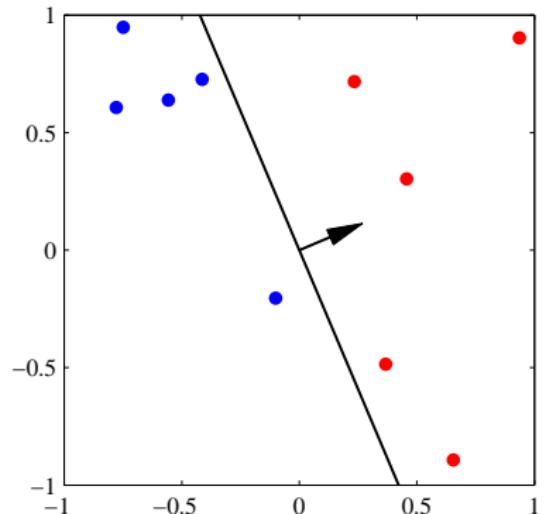
LEARNING THE PERCEPTRON



red = +1, blue = -1, $\eta = 1$

1. Pick another misclassified (x_j, y_j)
2. Set $w \leftarrow w + \eta y_j x_j$

LEARNING THE PERCEPTRON



red = +1, blue = -1, $\eta = 1$

Again update w , i.e., the hyperplane
This time we're done.

DRAWBACKS OF PERCEPTRON

The perceptron represents a first attempt at linear classification by directly learning the hyperplane defined by w . It has some drawbacks:

1. When the data is separable, there are an infinite # of hyperplanes.
 - ▶ We may think some are better than others, but this algorithm doesn't take "quality" into consideration. It converges to the first one it finds.
2. When the data isn't separable, the algorithm doesn't converge. The hyperplane of w is always moving around.
 - ▶ It's hard to detect this since it can take a long time for the algorithm to converge when the data is separable.

Later, we will discuss algorithms that use the same idea of directly learning the hyperplane w , but alters the objective function to fix these problems.

ColumbiaX: Machine Learning

Lecture 9

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

LOGISTIC REGRESSION

BINARY CLASSIFICATION

Linear classifiers

Given: Data $(x_1, y_1), \dots, (x_n, y_n)$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$

A **linear classifier** takes a vector $w \in \mathbb{R}^d$ and scalar $w_0 \in \mathbb{R}$ and predicts

$$y_i = f(x_i; w, w_0) = \text{sign}(x_i^T w + w_0).$$

We discussed two methods last time:

- ▶ Least squares: Sensitive to outliers
- ▶ Perceptron: Convergence issues, assumes linear separability

Can we combine the separating hyperplane idea with probability to fix this?

BAYES LINEAR CLASSIFICATION

Linear discriminant analysis

We saw an example of a linear classification rule using a Bayes classifier.

For the model $y \sim \text{Bern}(\pi)$ and $x | y \sim N(\mu_y, \Sigma)$, declare $y = 1$ given x if

$$\ln \frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)} > 0.$$

In this case, the *log odds* is equal to

$$\begin{aligned} \ln \frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)} &= \underbrace{\ln \frac{\pi_1}{\pi_0} - \frac{1}{2}(\mu_1 + \mu_0)^T \Sigma^{-1} (\mu_1 - \mu_0)}_{\text{a constant } w_0} \\ &\quad + \underbrace{x^T \Sigma^{-1} (\mu_1 - \mu_0)}_{\text{a vector } w} \end{aligned}$$

LOG ODDS AND BAYES CLASSIFICATION

Original formulation

Recall that originally we wanted to declare $y = 1$ given x if

$$\ln \frac{p(y=1|x)}{p(y=0|x)} > 0$$

We didn't have a way to define $p(y|x)$, so we used Bayes rule:

- ▶ Use $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$ and let the $p(x)$ cancel each other in the fraction
- ▶ Define $p(y)$ to be a Bernoulli distribution (coin flip distribution)
- ▶ Define $p(x|y)$ however we want (e.g., a single Gaussian)

Now, we want to directly define $p(y|x)$. We'll use the log odds to do this.

LOG ODDS AND BAYES CLASSIFICATION

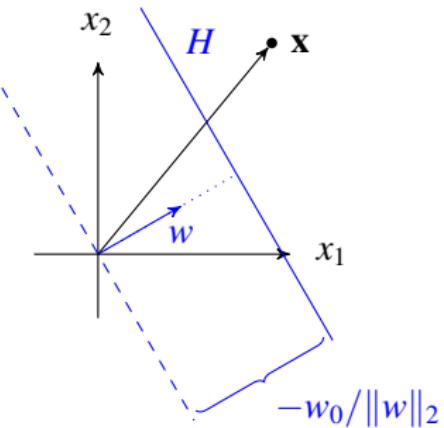
Log odds and hyperplanes

Classifying x based on the log odds

$$L = \ln \frac{p(y = +1|x)}{p(y = -1|x)},$$

we notice that

1. $L \gg 0$: more confident $y = +1$,
2. $L \ll 0$: more confident $y = -1$,
3. $L = 0$: can go either way



The linear function $x^T w + w_0$ captures these three objectives:

- The distance of x to a hyperplane H defined by (w, w_0) is $|\frac{x^T w}{\|w\|_2} + \frac{w_0}{\|w\|_2}|$.
- The sign of the function captures which side x is on.
- As x moves away/towards H , we become more/less confident.

LOG ODDS AND HYPERPLANES

Logistic link function

We can directly plug in the hyperplane representation for the log odds:

$$\ln \frac{p(y = +1|x)}{p(y = -1|x)} = x^T w + w_0$$

Question: What is different from the previous Bayes classifier?

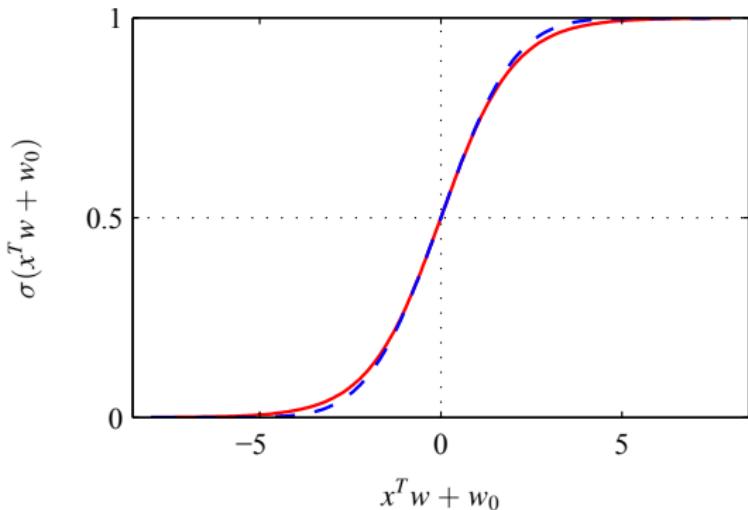
Answer: There was a formula for calculating w and w_0 based on the prior model and data x . Now, we put no restrictions on these values.

Setting $p(y = -1|x) = 1 - p(y = +1|x)$, solve for $p(y = +1|x)$ to find

$$p(y = +1|x) = \frac{\exp\{x^T w + w_0\}}{1 + \exp\{x^T w + w_0\}} = \sigma(x^T w + w_0).$$

- ▶ This is called the *sigmoid function*.
- ▶ We have chosen $x^T w + w_0$ as the *link function* for the log odds.

LOGISTIC SIGMOID FUNCTION



- ▶ Red line: Sigmoid function $\sigma(x^T w + w_0)$, which maps x to $p(y = +1|x)$.
- ▶ The function $\sigma(\cdot)$ captures our desire to be more confident as we move away from the separating hyperplane, defined by the x -axis.
- ▶ (Blue dashed line: On a later slide.)

LOGISTIC REGRESSION

As with regression, absorb the offset: $w \leftarrow \begin{bmatrix} w_0 \\ w \end{bmatrix}$ and $x \leftarrow \begin{bmatrix} 1 \\ x \end{bmatrix}$.

Definition

Let $(x_1, y_1), \dots, (x_n, y_n)$ be a set of binary labeled data with $y \in \{-1, +1\}$.
Logistic regression models each y_i as independently generated, with

$$P(y_i = +1 | x_i, w) = \sigma(x_i^T w), \quad \sigma(x_i; w) = \frac{e^{x_i^T w}}{1 + e^{x_i^T w}}.$$

Discriminative vs Generative classifiers

- ▶ This is a *discriminative* classifier because x is not directly modeled.
- ▶ Bayes classifiers are known as *generative* because x is modeled.

$$\text{Discriminative: } p(y|x) \qquad \text{Generative: } p(x|y)p(y).$$

LOGISTIC REGRESSION LIKELIHOOD

Data likelihood

Define $\sigma_i(w) = \sigma(x_i^T w)$. The joint likelihood of y_1, \dots, y_n is

$$\begin{aligned} p(y_1, \dots, y_n | x_1, \dots, x_n, w) &= \prod_{i=1}^n p(y_i | x_i, w) \\ &= \prod_{i=1}^n \sigma_i(w)^{\mathbb{1}(y_i=+1)} (1 - \sigma_i(w))^{\mathbb{1}(y_i=-1)} \end{aligned}$$

- ▶ Notice that each x_i modifies the probability of success for its y_i .
- ▶ Predicting new data is the same:
 - ▶ If $x^T w > 0$, then $\sigma(x^T w) > 1/2$ and predict $y = +1$, and vice versa.
 - ▶ We now get a confidence in our prediction via the probability $\sigma(x^T w)$.

LOGISTIC REGRESSION AND MAXIMUM LIKELIHOOD

More notation changes

Use the following fact to condense the notation:

$$\underbrace{\frac{e^{y_i x_i^T w}}{1 + e^{y_i x_i^T w}}}_{\sigma_i(y_i \cdot w)} = \underbrace{\left(\frac{e^{x_i^T w}}{1 + e^{x_i^T w}} \right)}_{\sigma_i(w)} \underbrace{\left(1 - \frac{e^{x_i^T w}}{1 + e^{x_i^T w}} \right)}_{1 - \sigma_i(w)} \mathbb{1}(y_i = +1)$$

therefore, the data likelihood can be written compactly as

$$p(y_1, \dots, y_n | x_1, \dots, x_n, w) = \prod_{i=1}^n \sigma_i(y_i \cdot w)$$

We want to maximize this over w .

LOGISTIC REGRESSION AND MAXIMUM LIKELIHOOD

Maximum likelihood

The maximum likelihood solution for w can be written

$$\begin{aligned} w_{\text{ML}} &= \arg \max_w \sum_{i=1}^n \ln \sigma_i(y_i \cdot w) \\ &= \arg \max_w \mathcal{L} \end{aligned}$$

As with the Perceptron, we can't directly set $\nabla_w \mathcal{L} = 0$, so we need an iterative algorithm. At step t , we can update

$$w^{(t+1)} = w^{(t)} + \eta \nabla_w \mathcal{L}, \quad \nabla_w \mathcal{L} = \sum_{i=1}^n (1 - \sigma_i(y_i \cdot w)) y_i x_i.$$

We will see that this results in an algorithm similar to the Perceptron.

LOGISTIC REGRESSION ALGORITHM (STEEPEST ASCENT)

Input: Training data $(x_1, y_1), \dots, (x_n, y_n)$ and step size $\eta > 0$

1. Set $w^{(1)} = \vec{0}$
2. For step $t = 1, 2, \dots$ do

- Update $w^{(t+1)} = w^{(t)} + \eta \sum_{i=1}^n (1 - \sigma_i(y_i \cdot w)) y_i x_i$

Perceptron: Search for misclassified (x_i, y_i) , update $w^{(t+1)} = w^{(t)} + \eta y_i x_i$.

Logistic regression: Something similar except we sum over all data.

- ▶ Recall that $\sigma_i(y_i \cdot w)$ picks out the probability assigned to the observed y_i .
- ▶ Therefore $1 - \sigma_i(y_i \cdot w)$ is the probability assigned to the *wrong* value.
- ▶ Perceptron is “all-or-nothing.” Either it’s correctly or incorrectly classified.
- ▶ Logistic regression has a probability “fudge-factor.”

BAYESIAN LOGISTIC REGRESSION

Problem: If a hyperplane can separate all training data, then $\|w_{\text{ML}}\|_2 \rightarrow \infty$. This drives $\sigma_i(y_i \cdot w) \rightarrow 1$ for each (x_i, y_i) .

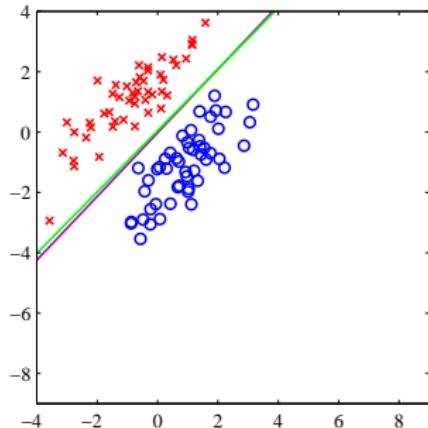
Even for nearly separable data it might get a few very wrong in order to be more confident about the rest. This is a case of “over-fitting.”

A solution: Regularize w with $\lambda w^T w$:

$$w_{\text{MAP}} = \arg \max_w \sum_{i=1}^n \ln \sigma_i(y_i \cdot w) - \lambda w^T w$$

We've seen how this corresponds to a Gaussian prior distribution on w .

How about the posterior $p(w|x, y)$?



LAPLACE APPROXIMATION

BAYESIAN LOGISTIC REGRESSION

Posterior calculation

Define the prior distribution on w to be $w \sim N(0, \lambda^{-1}I)$. The posterior is

$$p(w|x, y) = \frac{p(w) \prod_{i=1}^n \sigma_i(y_i \cdot w)}{\int p(w) \prod_{i=1}^n \sigma_i(y_i \cdot w) dw}$$

This is not a “standard” distribution and we can’t calculate the denominator.

Therefore we can’t actually say what $p(w|x, y)$ is.

Can we approximate $p(w|x, y)$?

LAPLACE APPROXIMATION

One strategy

Pick a distribution to approximate $p(w|x, y)$. We will say

$$p(w|x, y) \approx \text{Normal}(\mu, \Sigma).$$

Now we need a method for setting μ and Σ .

Laplace approximations

Using a condensed notation, notice from Bayes rule that

$$p(w|x, y) = \frac{e^{\ln p(y, w|x)}}{\int e^{\ln p(y, w|x)} dw}.$$

We will approximate $\ln p(y, w|x)$ in the numerator and denominator.

LAPLACE APPROXIMATION

Let's define $f(w) = \ln p(y, w|x)$.

Taylor expansions

We can approximate $f(w)$ with a **second order Taylor expansion**.

Recall that $w \in \mathbb{R}^{d+1}$. For any point $z \in \mathbb{R}^{d+1}$,

$$f(w) \approx f(z) + (w - z)^T \nabla f(z) + \frac{1}{2} (w - z)^T (\nabla^2 f(z)) (w - z)$$

The notation $\nabla f(z)$ is short for $\nabla_w f(w)|_z$, and similarly for the matrix of second derivatives. We just need to pick z .

The Laplace approximation defines $z = w_{\text{MAP}}$.

LAPLACE APPROXIMATION (SOLVING)

Recall $f(w) = \ln p(y, w|x)$ and $z = w_{\text{MAP}}$. From Bayes rule and the Laplace approximation we now have

$$\begin{aligned} p(w|x, y) &= \frac{e^{f(w)}}{\int e^{f(w)} dw} \\ &\approx \frac{e^{f(z) + (w-z)^T \nabla f(z) + \frac{1}{2}(w-z)^T (\nabla^2 f(z))(w-z)}}{\int e^{f(z) + (w-z)^T \nabla f(z) + \frac{1}{2}(w-z)^T (\nabla^2 f(z))(w-z)} dw} \end{aligned}$$

This can be simplified in two ways,

1. The term $e^{f(w_{\text{MAP}})}$ in the numerator and denominator can be viewed as a constant since it doesn't vary in w . It therefore cancels out.
2. By definition of how we find w_{MAP} , the vector $\nabla_w \ln p(y, w|x)|_{w_{\text{MAP}}} = 0$.

LAPLACE APPROXIMATION (SOLVING)

We're therefore left with the approximation

$$p(w|x, y) \approx \frac{e^{-\frac{1}{2}(w-w_{MAP})^T(-\nabla^2 \ln p(y, w_{MAP}|x))(w-w_{MAP})}}{\int e^{-\frac{1}{2}(w-w_{MAP})^T(-\nabla^2 \ln p(y, w_{MAP}|x))(w-w_{MAP})} dw}$$

The solution comes by observing that this is a multivariate normal,

$$p(w|x, y) \approx \text{Normal}(\mu, \Sigma),$$

where

$$\mu = w_{MAP}, \quad \Sigma = (-\nabla^2 \ln p(y, w_{MAP}|x))^{-1}$$

We can take the second derivative (Hessian) of the log joint likelihood to find

$$\nabla^2 \ln p(y, w_{MAP}|x) = -\lambda I - \sum_{i=1}^n \sigma(y_i \cdot x_i^T w_{MAP}) (1 - \sigma(y_i \cdot x_i^T w_{MAP})) x_i x_i^T$$

BAYESIAN LOGISTIC REGRESSION

Laplace approximation for logistic regression

Given labeled data $(x_1, y_1), \dots, (x_n, y_n)$ and the model

$$P(y_i|x_i, w) = \sigma(y_i x_i^T w), \quad w \sim N(0, \lambda^{-1} I), \quad \sigma(y_i x_i^T w) = \frac{e^{y_i x_i^T w}}{1 + e^{y_i x_i^T w}}$$

1. Find: $w_{\text{MAP}} = \arg \max_w \sum_{i=1}^n \ln \sigma(y_i x_i^T w_{\text{MAP}}) - \frac{\lambda}{2} w^T w$
2. Set: $-\Sigma^{-1} = -\lambda I - \sum_{i=1}^n \sigma(y_i x_i^T w_{\text{MAP}}) (1 - \sigma(y_i x_i^T w_{\text{MAP}})) x_i x_i^T$
3. Approximate: $p(w|x, y) = N(w_{\text{MAP}}, \Sigma)$.

ColumbiaX: Machine Learning

Lecture 10

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

FEATURE EXPANSIONS

FEATURE EXPANSIONS

Feature expansions (also called **basis expansions**) are names given to a technique we've already discussed and made use of.

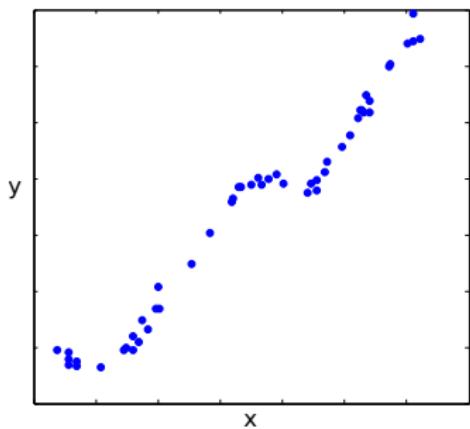
Problem: A linear model on the original feature space $x \in \mathbb{R}^d$ doesn't work.

Solution: Map the features to a higher dimensional space $\phi(x) \in \mathbb{R}^D$, where $D > d$, and do linear modeling there.

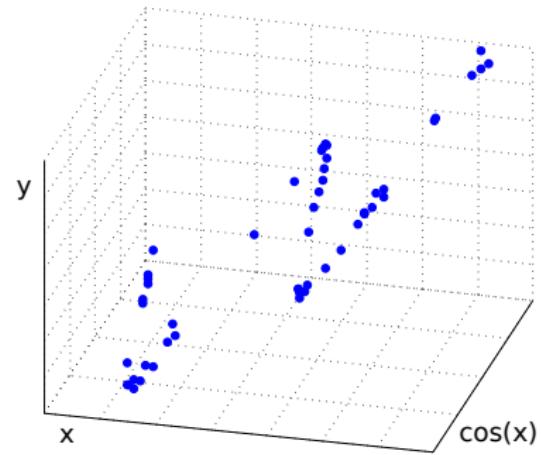
Examples

- ▶ For polynomial regression on \mathbb{R} , we let $\phi(x) = (x, x^2, \dots, x^p)$.
- ▶ For jump discontinuities, $\phi(x) = (x, \mathbb{1}\{x < a\})$.

MAPPING EXAMPLE FOR REGRESSION



(a) Data for linear regression



(b) Same data mapped to higher dimension

High-dimensional maps can transform the data so output is linear in inputs.

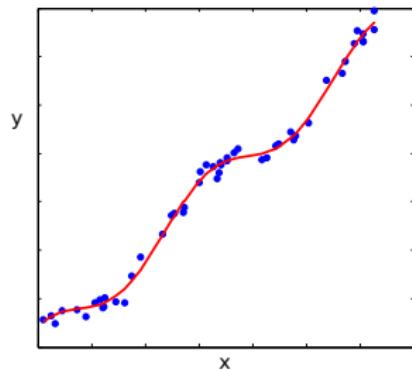
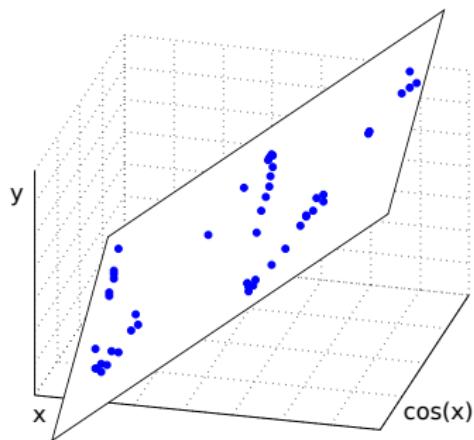
Left: Original $x \in \mathbb{R}$ and response y .

Right: x mapped to \mathbb{R}^2 using $\phi(x) = (x, \cos x)^T$.

MAPPING EXAMPLE FOR REGRESSION

Using the mapping $\phi(x) = (x, \cos x)^T$, learn the linear regression model

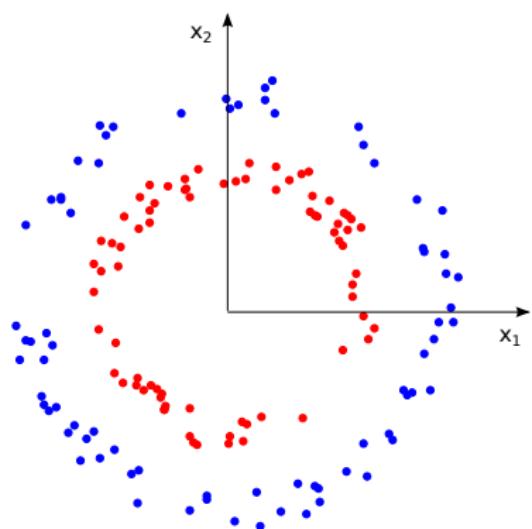
$$\begin{aligned}y &\approx w_0 + \phi(x)^T w \\&\approx w_0 + w_1 x + w_2 \cos x.\end{aligned}$$



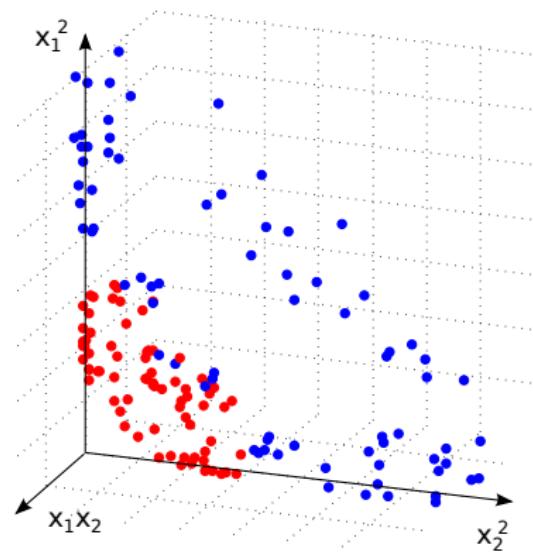
Left: Learn (w_0, w_1, w_2) to approximate data on the left with a plane.

Right: For each point x , map to $\phi(x)$ and predict y . Plot as a function of x .

MAPPING EXAMPLE FOR CLASSIFICATION



(e) Data for binary classification



(f) Same data mapped to higher dimension

High-dimensional maps can transform data so it becomes linearly separable.

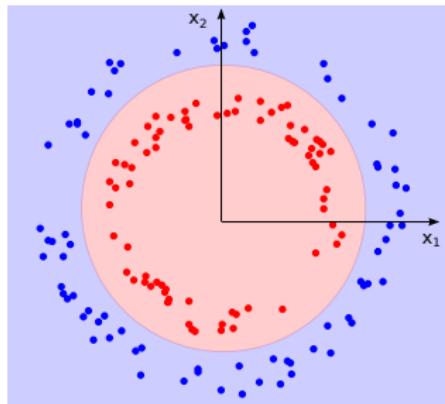
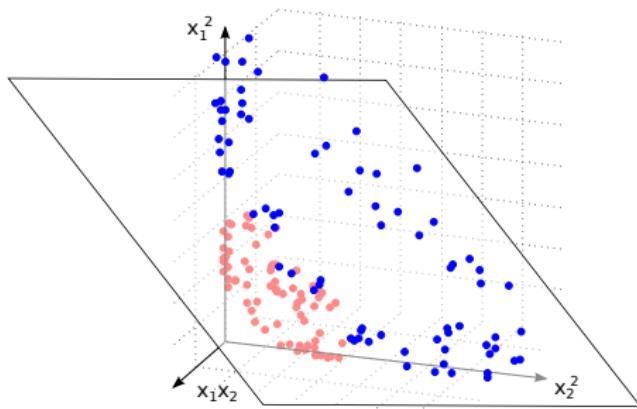
Left: Original data in \mathbb{R}^2 .

Right: Data mapped to \mathbb{R}^3 using $\phi(x) = (x_1^2, x_1x_2, x_2^2)^T$.

MAPPING EXAMPLE FOR CLASSIFICATION

Using the mapping $\phi(x) = (x_1^2, x_1x_2, x_2^2)^T$, learn a linear classifier

$$\begin{aligned}y &= \text{sign}(w_0 + \phi(x)^T w) \\&= \text{sign}(w_0 + w_1x_1^2 + w_2x_1x_2 + w_3x_2^2).\end{aligned}$$



Left: Learn (w_0, w_1, w_2, w_3) to linearly separate classes with hyperplane.

Right: For each point x , map to $\phi(x)$ and classify. Color decision regions in \mathbb{R}^2 .

FEATURE EXPANSIONS AND DOT PRODUCTS

What expansion should I use?

This is not obvious. The illustrations required knowledge about the data that we likely won't have (especially if it's in high dimensions).

One approach is to use the “kitchen sink”: If you can think of it, then use it.
Select the useful features with an ℓ_1 penalty

$$w_{\ell_1} = \arg \min_w \sum_{i=1}^n f(y_i, \phi(x_i), w) + \lambda \|w\|_1.$$

We know that this will find a sparse subset of the dimensions of $\phi(x)$ to use.

Often we only need to work with dot products $\phi(x_i)^T \phi(x_j) \equiv K(x_i, x_j)$. This is called a **kernel** and can produce some interesting results.

KERNELS

PERCEPTRON (SOME MOTIVATION)

Perceptron classifier

Let $x_i \in \mathbb{R}^{d+1}$ and $y_i \in \{-1, +1\}$ for $i = 1, \dots, n$ observations. We saw that the Perceptron constructs the hyperplane from data,

$$w = \sum_{i \in \mathcal{M}} y_i x_i,$$

where \mathcal{M} is the sequentially constructed set of misclassified examples.

Predicting new data

We also discussed how we can predict the label y_0 for a new observation x_0 :

$$y_0 = \text{sign}(x_0^T w) = \text{sign} \left(\sum_{i \in \mathcal{M}} y_i x_0^T x_i \right)$$

We've taken feature expansions for granted, but we can explicitly write it as

$$y_0 = \text{sign}(\phi(x_0)^T w) = \text{sign} \left(\sum_{i \in \mathcal{M}} y_i \phi(x_0)^T \phi(x_i) \right)$$

We can represent the decision using dot products between data points.

KERNELS

Kernel definition

A kernel $K(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a symmetric function defined as follows:

Definition: For any set of n data points $x_1, \dots, x_n \in \mathbb{R}^d$, the $n \times n$ matrix K , where $K_{ij} = K(x_i, x_j)$, is *positive semidefinite*.

Intuitively, this means K satisfies the properties of a covariance matrix.

Mercer's theorem

If the function $K(\cdot, \cdot)$ satisfies the above properties, then there exists a mapping $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j).$$

If we first define $\phi(\cdot)$ and then K , this is obvious. However, sometimes we first define $K(\cdot, \cdot)$ and avoid ever using $\phi(\cdot)$.

GAUSSIAN KERNEL (RADIAL BASIS FUNCTION)

By far the most popular kernel is the Gaussian kernel, also called the radial basis function (RBF),

$$K(x, x') = a \exp \left\{ -\frac{1}{b} \|x - x'\|^2 \right\}.$$

- ▶ This is a good, general-purpose kernel that usually works well.
- ▶ It takes into account proximity in \mathbb{R}^d . Things close together in space have larger value (as defined by kernel width b).

In this case, the the mapping $\phi(x)$ that produces the RBF kernel is *infinite dimensional* (it's a continuous function instead of a vector). Therefore

$$K(x, x') = \int \phi_t(x) \phi_t(x') dt.$$

- ▶ $K(x, x')$ is like a Gaussian on x with x' as the mean (or vice versa).
- ▶ $\phi_t(x)$ can be thought of as a function of t with parameter x .

KERNELS

Another kernel

Map : $\phi(x) = (1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, x_1^2, \dots, x_d^2, \dots, \sqrt{2}x_i x_j, \dots)$

Kernel : $\phi(x)^T \phi(x') = K(x, x') = (1 + x^T x')^2$

In fact, we can show: $K(x, x') = (1 + x^T x')^b$, for $b > 0$ is a kernel as well.

Kernel arithmetic

Certain functions of kernels can produce new kernels.

Let K_1 and K_2 be any two kernels, then constructing K in the following ways produces a new kernel (among many other ways):

$$K(x, x') = K_1(x, x')K_2(x, x')$$

$$K(x, x') = K_1(x, x') + K_2(x, x')$$

$$K(x, x') = \exp\{K_1(x, x')\}$$

KERNELIZED PERCEPTRON

Returning to the Perceptron

We write the feature-expanded decision as

$$\begin{aligned}y_0 &= \text{sign} \left(\sum_{i \in \mathcal{M}} y_i \phi(x_0)^T \phi(x_i) \right) \\&= \text{sign} \left(\sum_{i \in \mathcal{M}} y_i K(x_0, x_i) \right)\end{aligned}$$

We can pick the kernel we want to use. Let's pick the RBF (set $a = 1$). Then

$$y_0 = \text{sign} \left(\sum_{i \in \mathcal{M}} y_i e^{-\frac{1}{b} \|x_0 - x_i\|^2} \right)$$

Notice that we never actually need to calculate $\phi(x)$.

What is this doing?

- ▶ Notice $0 < K(x_0, x_i) \leq 1$, with bigger values when x_0 is closer to x_i .
- ▶ This is like a “soft voting” among the data picked by Perceptron.

KERNELIZED PERCEPTRON

Learning the kernelized Perceptron

Recall: Given a current vector $w^{(t)} = \sum_{i \in \mathcal{M}_t} y_i x_i$, we update it as follows,

1. Find a new x' such that $y' \neq \text{sign}(x'^T w^{(t)})$
2. Add the index of x' to \mathcal{M} and set $w^{(t+1)} = \sum_{i \in \mathcal{M}_{t+1}} y_i x_i$

Again we only need dot products, meaning these steps are equivalent to

1. Find a new x' such that $y' \neq \text{sign}(\sum_{i \in \mathcal{M}_t} y_i K(x', x_i))$
2. Add the index of x' to \mathcal{M} *but don't bother calculating $w^{(t+1)}$*

The trick is to realize that we never need to work with $\phi(x)$.

- ▶ We don't need $\phi(x)$ to do Step 1 above.
- ▶ We don't need $\phi(x)$ to classify new data (previous slide).
- ▶ We only ever need to calculate $K(x, x')$ between two points.

KERNEL k -NN

An extension

We can generalize kernelized Perceptron to *soft k*-NN with a simple change.
Instead of summing over misclassified data \mathcal{M} , sum over *all* the data:

$$y_0 = \text{sign} \left(\sum_{i=1}^n y_i e^{-\frac{1}{b} \|x_0 - x_i\|^2} \right).$$

Next, notice the *decision* doesn't change if we divide by a positive constant.

Let : $Z = \sum_{j=1}^n e^{-\frac{1}{b} \|x_0 - x_j\|^2}$

Construct : Vector $p(x_0)$, where $p_i(x_0) = \frac{1}{Z} e^{-\frac{1}{b} \|x_0 - x_i\|^2}$

Declare : $y_0 = \text{sign} \left(\sum_{i=1}^n y_i p_i(x_0) \right)$

- ▶ We let all data vote for the label based on a “confidence score” $p(x_0)$.
- ▶ Set b so that most $p_i(x_0) \approx 0$ to only focus on neighborhood around x_0 .

KERNEL REGRESSION

Nadaraya-Watson model

The developments are almost limitless.

Here's a regression example almost identical to the kernelized k -NN:

Before: $y \in \{-1, +1\}$

Now: $y \in \mathbb{R}$

Using the RBF kernel, for a new (x_0, y_0) predict

$$y_0 = \sum_{i=1}^n y_i \frac{K(x_0, x_i)}{\sum_{j=1}^n K(x_0, x_j)}.$$

What is this doing?

We're taking a locally weighted average of all y_i for which x_i is close to x_0 (as decided by the kernel width). *Gaussian processes* are another option. . .

GAUSSIAN PROCESSES

KERNELIZED BAYESIAN LINEAR REGRESSION

Regression setup: For n observations, with response vector $y \in \mathbb{R}^n$ and their feature matrix X , we define the likelihood and prior

$$y \sim N(Xw, \sigma^2 I), \quad w \sim N(0, \lambda^{-1} I).$$

Marginalizing: What if we integrate out w ? We can solve this,

$$p(y|X) = \int p(y|X, w)p(w)dw = N(0, \sigma^2 I + \lambda^{-1} X X^T).$$

Kernelization: Notice that $(X X^T)_{ij} = x_i^T x_j$. Replace each x with $\phi(x)$ after which we can say $(\phi(X)\phi(X)^T)_{ij} = K(x_i, x_j)$. We can define K directly, so

$$p(y|X) = \int p(y|X, w)p(w)dw = N(0, \sigma^2 I + \lambda^{-1} K).$$

This is called a *Gaussian process*. We never use w or $\phi(x)$, but just $K(x_i, x_j)$.

GAUSSIAN PROCESSES

Definition

- Let $f(x) \in \mathbb{R}$ and $x \in \mathbb{R}^d$.
- Define the *kernel* $K(x, x')$ between two points x and x' .
- Then $f(x)$ is a *Gaussian process* and $y(x)$ the noise-added process if

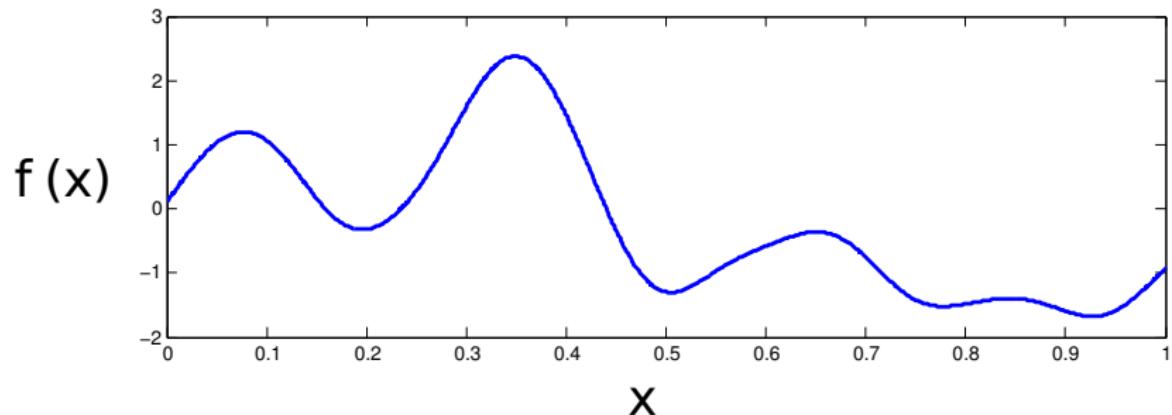
$$y | f \sim N(f, \sigma^2 I), \quad f \sim N(0, K) \quad \Leftrightarrow \quad y \sim N(0, \sigma^2 I + K)$$

where $y = (y_1, \dots, y_n)^T$ and K is $n \times n$ with $K_{ij} = K(x_i, x_j)$.

Comments:

- ▶ We combined the previous λ^{-1} with K (for notation only).
- ▶ Typical breakdown: $f(x)$ is the GP and $y(x)$ equals $f(x)$ plus i.i.d. noise.
- ▶ The kernel is what keeps this from being “just a Gaussian.”

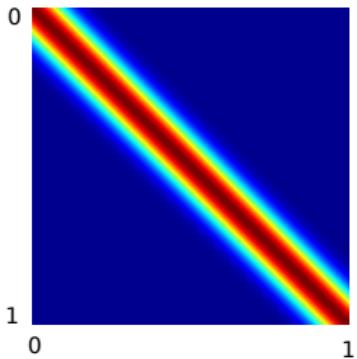
GAUSSIAN PROCESSES



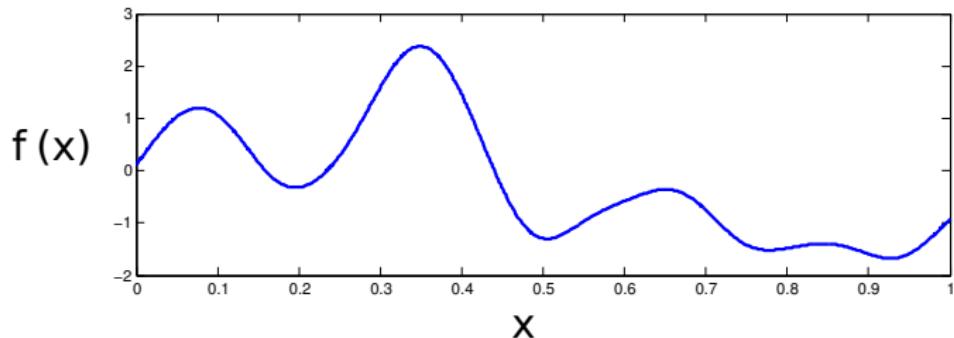
Above: A Gaussian process $f(x)$ generated using

$$K(x_i, x_j) = \exp \left\{ -\frac{\|x_i - x_j\|^2}{b} \right\}.$$

Right: The covariance of $f(x)$ defined by K .

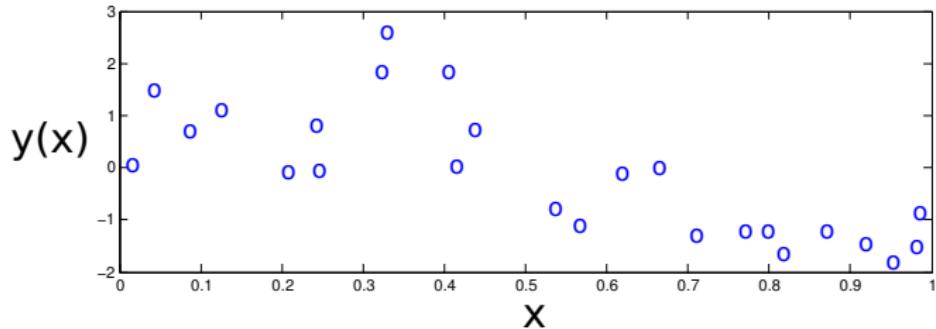


GAUSSIAN PROCESSES



Top: Unobserved underlying function,

Bottom: Noisy observed data sampled from this function



PREDICTIONS WITH GAUSSIAN VECTORS

Bayesian linear regression

Imagine we have n observation pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ and want to predict y_0 given x_0 . Integrating out w , the joint distribution is

$$\begin{bmatrix} y_0 \\ y \end{bmatrix} \sim \text{Normal}\left(\mathbf{0}, \sigma^2 I + \begin{bmatrix} x_0^T x_0 & (X x_0)^T \\ X x_0 & X X^T \end{bmatrix}\right)$$

We want to predict y_0 given \mathcal{D} and x_0 . Calculations can show that

$$y_0 | \mathcal{D}, x_0 \sim \text{Normal}(\mu_0, \sigma_0^2)$$

$$\mu_0 = (X x_0)^T (X X^T)^{-1} y$$

$$\sigma_0^2 = \sigma^2 + x_0^T x_0 - (X x_0)^T (X X^T)^{-1} (X x_0)$$

The since the infinite Gaussian process is only evaluated at a finite set of points, we can use this fact.

PREDICTIONS WITH GAUSSIAN PROCESSES

Predictive distribution of $y(x)$

Given measured data $\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$, the distribution of $y(x)$ can be calculated at any *new* x to make predictions.

Let $K(x, \mathcal{D}_n) = [K(x, x_1), \dots, K(x, x_n)]$ and K_n be the $n \times n$ kernel matrix restricted points in \mathcal{D}_n . Then we can show

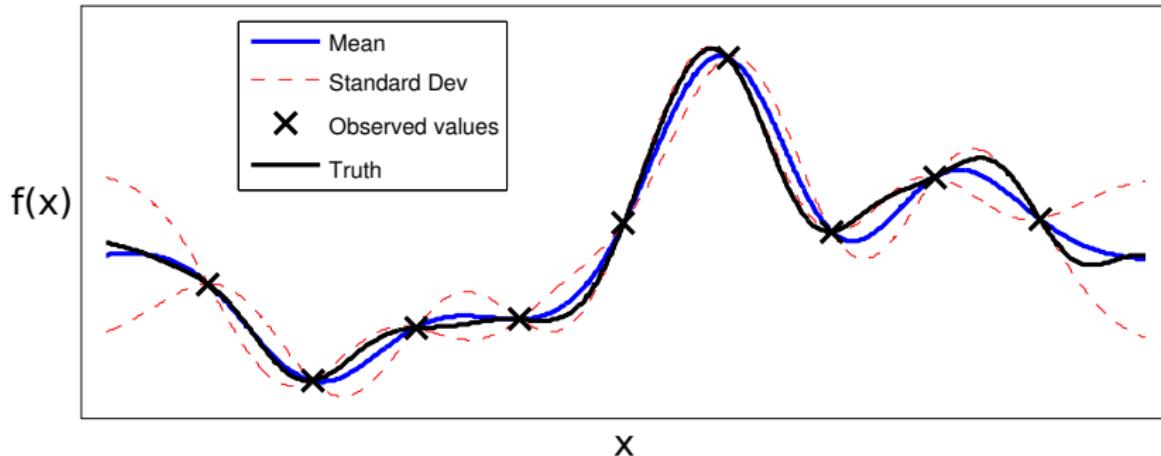
$$y(x)|\mathcal{D}_n \sim N(\mu(x), \Sigma(x)),$$

$$\mu(x) = K(x, \mathcal{D}_n)K_n^{-1}y,$$

$$\Sigma(x) = \sigma^2 + K(x, x) - K(x, \mathcal{D}_n)K_n^{-1}K(x, \mathcal{D}_n)^T$$

For the posterior of $f(x)$ instead of $y(x)$, just remove σ^2 .

GAUSSIAN PROCESSES POSTERIOR



What does the posterior distribution of $f(x)$ look like?

- ▶ We have data marked by an \times .
- ▶ These values pin down the function $f(x)$ nearby
- ▶ We get a mean and variance for every possible x from a previous slide.
- ▶ The distribution on $y(x)$ adds variance σ^2 (*very* small above) point-wise.

ColumbiaX: Machine Learning

Lecture 11

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

MAXIMUM MARGIN CLASSIFIERS

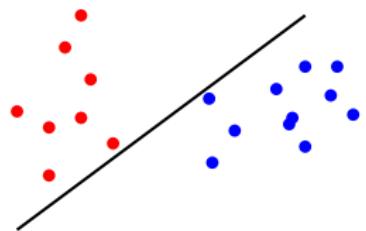
MAXIMUM MARGIN IDEA

Setting

Linear classification, two linearly separable classes.

Recall Perceptron

- ▶ Selects *some* hyperplane separating the classes.
- ▶ Selected hyperplane depends on several factors.

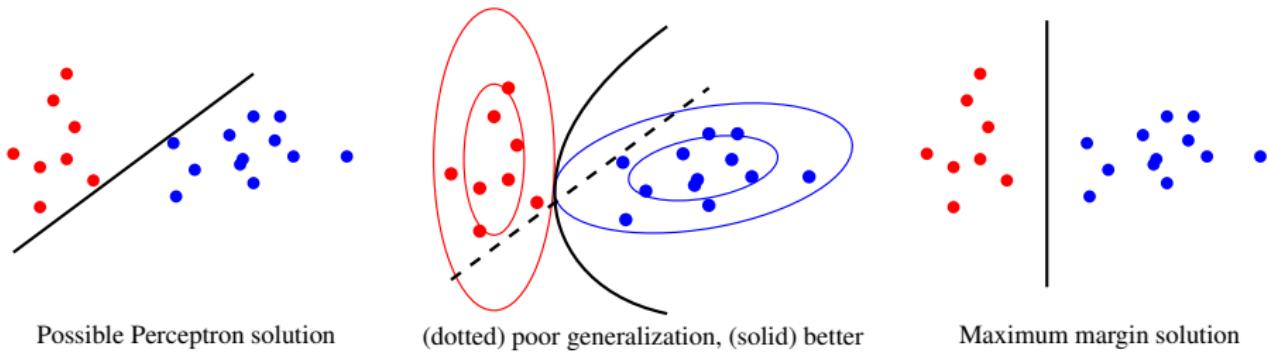


Maximum margin

To achieve good generalization (low prediction error), place the hyperplane “in the middle” between the two classes.

More precisely, choose a plane such that its distance to the closest point in each class is maximized. This distance is called the **margin**.

GENERALIZATION ERROR



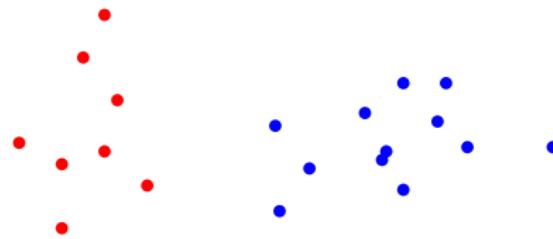
Example: Gaussian data

- ▶ Intuitively, the classifier on the left isn't good because sampling more data could lead to misclassifications.
- ▶ If we imagine the data from each class as Gaussian, we could frame the goal as to find a decision boundary that cuts into as little probability mass as possible.
- ▶ With no distribution assumptions, we can argue that max-margin is best.

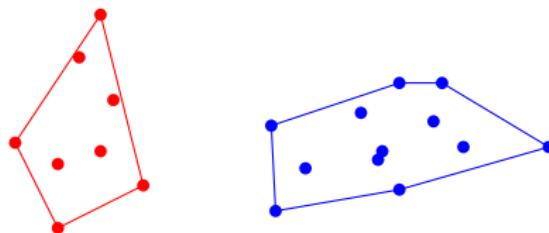
SUBSTITUTING CONVEX SETS

Observation

Where a separating hyperplane may be placed depends on the “outer” points on the sets. Points in the center do not matter.



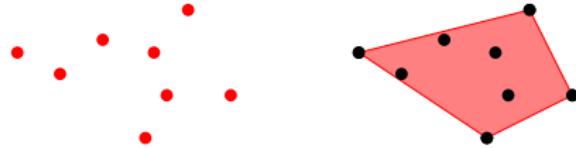
In geometric terms, we can represent each class by the smallest convex set which contains all point in the class. This is called a *convex hull*.



SUBSTITUTING CONVEX SETS

Convex hulls

The thing to remember for this lecture is that a convex hull is defined by all possible weighted averages of points in a set.



That is, let x_1, \dots, x_n be the above data coordinates. Every point x_0 in the shaded region – i.e., the convex hull – can be reached by setting

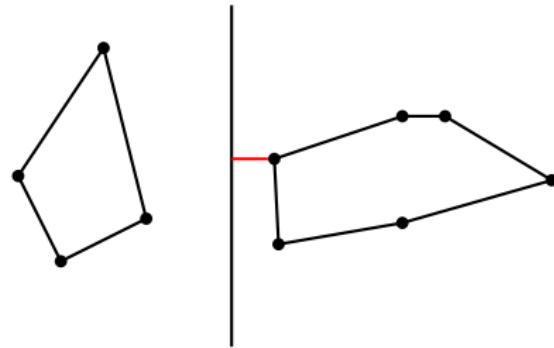
$$x_0 = \sum_{i=1}^n \alpha_i x_i, \quad \alpha_i \geq 0, \quad \sum_{i=1}^n \alpha_i = 1,$$

for some $(\alpha_1, \dots, \alpha_n)$. No point outside this region can be reached this way.

MARGIN

Definition

The *margin* of a classifying hyperplane H is the shortest distance between the plane and any point in either set (equivalently, the convex hull)



When we maximize this margin, H is “exactly in the middle” of the two convex hulls. Of course, the difficult part is how do we find this H ?

SUPPORT VECTOR MACHINES

SUPPORT VECTOR MACHINE

Finding the hyperplane

For n linearly separable points $(x_1, y_1), \dots, (x_n, y_n)$ with $y_i \in \{\pm 1\}$, solve:

$$\begin{aligned} & \min_{w, w_0} && \frac{1}{2} \|w\|^2 \\ & \text{subject to} && y_i(x_i^T w + w_0) \geq 1 \quad \text{for } i = 1, \dots, n \end{aligned}$$

Comments

- ▶ Recall that $y_i(x_i^T w + w_0) > 0$ if $y_i = \text{sign}(x_i^T w + w_0)$.
- ▶ If there exists a hyperplane H that separates the classes, we can scale w so that $y_i(x_i^T w + w_0) > 1$ for all i (this is useful later).
- ▶ The resulting classifier is called a *support vector machine*. This formulation only has a solution when the classes are linearly separable.
- ▶ It is not at all obvious why this maximizes the margin. This will become more clear when we look at the solution.

SUPPORT VECTOR MACHINE

[Skip to the end](#)

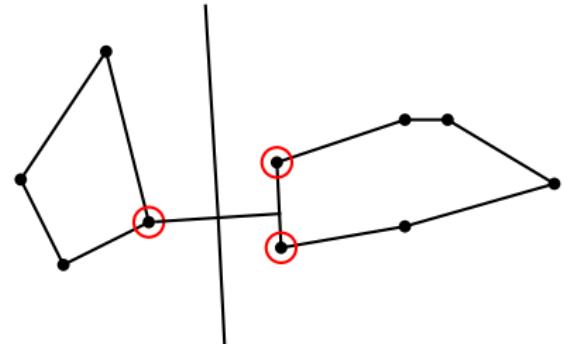
Q: First, can we intuitively say what the solution should *look* like?

A: Yes, but we won't give the proof.

1. Find the closest two points from the convex hulls of class $+1$ and -1 .
2. Connect them with a line and put a perpendicular hyperplane in the middle.
3. If S_1 and S_0 are the sets of x in class $+1$ and -1 respectively, we're looking for two *probability* vectors α_1 and α_0 such that we minimize

$$\left\| \underbrace{\left(\sum_{x_i \in S_1} \alpha_{1i} x_i \right)}_{\text{in conv. hull of } S_1} - \underbrace{\left(\sum_{x_i \in S_0} \alpha_{0i} x_i \right)}_{\text{in conv. hull of } S_0} \right\|_2$$

4. Then we define the hyperplane using the two points found with α_1 and α_0 .



PRIMAL AND DUAL PROBLEMS

Primal problem

The *primal* optimization problem is the one we defined:

$$\begin{aligned} \min_{w, w_0} \quad & \frac{1}{2} \|w\|^2 \\ \text{subject to} \quad & y_i(x_i^T w + w_0) \geq 1 \quad \text{for } i = 1, \dots, n \end{aligned}$$

This is tricky, so we use *Lagrange multipliers* to set up the “dual” problem.

Lagrange multipliers

Define Lagrange multipliers $\alpha_i > 0$ for $i = 1, \dots, n$. The Lagrangian is

$$\begin{aligned} \mathcal{L} &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i(x_i^T w + w_0) - 1) \\ &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i y_i (x_i^T w + w_0) + \sum_{i=1}^n \alpha_i \end{aligned}$$

We want to minimize \mathcal{L} over w and w_0 and maximize over $(\alpha_1, \dots, \alpha_n)$.

SETTING UP THE DUAL PROBLEM

First minimize over w and w_0 :

$$\mathcal{L} = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i y_i (x_i^T w + w_0) + \sum_{i=1}^n \alpha_i$$



$$\nabla_w \mathcal{L} = w - \sum_{i=1}^n \alpha_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^n \alpha_i y_i x_i$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = - \sum_{i=1}^n \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^n \alpha_i y_i = 0$$

Therefore,

1. We can plug the solution for w back into the problem.
2. We know that $(\alpha_1, \dots, \alpha_n)$ must satisfy $\sum_{i=1}^n \alpha_i y_i = 0$.

SVM DUAL PROBLEM

Lagrangian: $\mathcal{L} = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i y_i (x_i^T w + w_0) + \sum_{i=1}^n \alpha_i$

Dual problem

Plugging these values in from the previous slide, we get the dual problem

$$\begin{aligned} \max_{\alpha_1, \dots, \alpha_n} \quad & \mathcal{L} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\ \text{subject to} \quad & \sum_{i=1}^n \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

Comments

- ▶ Where did w_0 go? The condition $\sum_{i=1}^n \alpha_i y_i = 0$ gives $0 \cdot w_0$ in the dual.
- ▶ We now maximize over the α_i . This requires an algorithm that we won't discuss in class. Many good software implementations are available.

AFTER SOLVING THE DUAL

Solving the primal problem

Before discussing the solution of the dual, we ask:

After finding each α_i how do we predict a new $y_0 = \text{sign}(x_0^T w + w_0)$?

We have: $\mathcal{L} = \frac{1}{2}\|w\|^2 - \sum_{i=1}^n \alpha_i(y_i(x_i^T w + w_0) - 1)$

With conditions: $\alpha_i \geq 0, \quad y_i(x_i^T w + w_0) - 1 \geq 0$

Solve for w .

$$\nabla_w \mathcal{L} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^n \alpha_i y_i x_i \quad (\text{just plug in the learned } \alpha_i \text{'s})$$

What about w_0 ?

- ▶ We can show that at the solution, $\alpha_i(y_i(x_i^T w + w_0) - 1) = 0$ for all i .
- ▶ Therefore, pick i for which $\alpha_i > 0$ and solve $y_i(x_i^T w + w_0) - 1 = 0$ for w_0 using the solution for w (all possible i will give the same solution).

UNDERSTANDING THE DUAL

Dual problem

We can manipulate the dual problem to find out what it's trying to do.

$$\begin{aligned} \max_{\alpha_1, \dots, \alpha_n} \quad & \mathcal{L} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\ \text{subject to} \quad & \sum_{i=1}^n \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

Since $y_i \in \{-1, +1\}$

- ▶ $\sum_{i=1}^n \alpha_i y_i = 0 \Rightarrow C = \sum_{i \in S_1} \alpha_i = \sum_{j \in S_0} \alpha_j$
- ▶ $\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j (x_i^T x_j) = \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|^2 = C^2 \left\| \sum_{i \in S_1} \frac{\alpha_i}{C} x_i - \sum_{j \in S_0} \frac{\alpha_j}{C} x_j \right\|^2$

UNDERSTANDING THE DUAL

Dual problem

We can change notation to write the dual as

$$\max_{\alpha_1, \dots, \alpha_n} \quad \mathcal{L} = 2C - \frac{1}{2} C^2 \left\| \sum_{i \in S_1} \frac{\alpha_i}{C} x_i - \sum_{j \in S_0} \frac{\alpha_j}{C} x_j \right\|^2$$

$$\text{subject to} \quad C := \sum_{i \in S_1} \alpha_i = \sum_{j \in S_0} \alpha_j, \quad \alpha_i \geq 0$$

We observe that the maximum of this function satisfies

$$\min_{\alpha_1, \dots, \alpha_n} \left\| \underbrace{\left(\sum_{i \in S_1} \frac{\alpha_i}{C} x_i \right)}_{\text{in conv. hull of } S_1} - \underbrace{\left(\sum_{j \in S_0} \frac{\alpha_j}{C} x_j \right)}_{\text{in conv. hull of } S_0} \right\|^2$$

Therefore, the dual problem is trying to find the closest points in the convex hulls constructed from data in class +1 and -1.

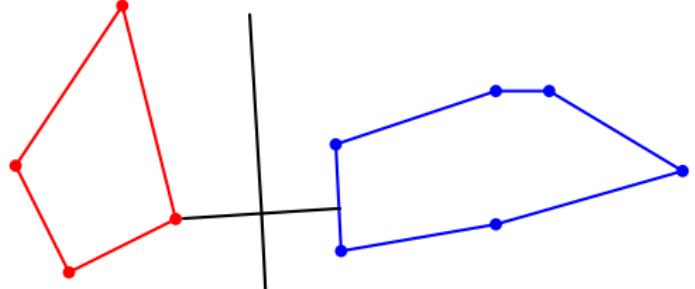
RETURNING TO THE PICTURE

Recall

We wanted to find:

$$\min_{\substack{u \in \mathcal{H}(S_1) \\ v \in \mathcal{H}(S_0)}} \|u - v\|^2$$

The direction of w is $u - v$.



We previously claimed we can find the max-margin hyperplane as follows:

1. Find shortest line connecting the convex hulls.
2. Place hyperplane orthogonal to line and exactly at the midpoint.

With the SVM we want to minimize $\|w\|^2$ and we can write this solution as

$$w = \sum_{i=1}^n \alpha_i y_i x_i = C \left(\sum_{i \in S_1} \frac{\alpha_i}{C} x_i - \sum_{j \in S_0} \frac{\alpha_j}{C} x_j \right)$$

SOFT-MARGIN SVM

Question: What if the data isn't linearly separable?

Answer: Permit training data be on wrong side of hyperplane, but at a cost.

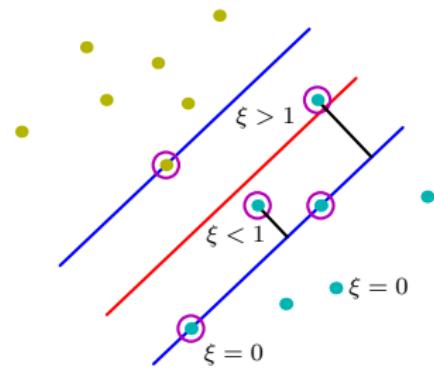
Slack variables

Replace the training rule $y_i(x_i^T w + w_0) \geq 1$ with

$$y_i(x_i^T w + w_0) \geq 1 - \xi_i,$$

with $\xi_i \geq 0$.

The ξ_i are called *slack variables*.



SOFT-MARGIN SVM

Soft-margin objective function

Adding the slack variables gives a new objective to optimize

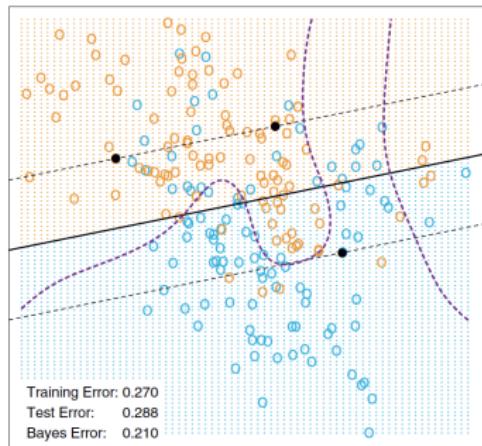
$$\begin{aligned} \min_{w, w_0, \xi_1, \dots, \xi_n} \quad & \frac{1}{2} \|w\|^2 + \lambda \sum_{i=1}^n \xi_i \\ \text{subject to} \quad & y_i(x_i^T w + w_0) \geq 1 - \xi_i \quad \text{for } i = 1, \dots, n \\ & \xi_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

We also have to choose the parameter $\lambda > 0$. We solve the dual as before.

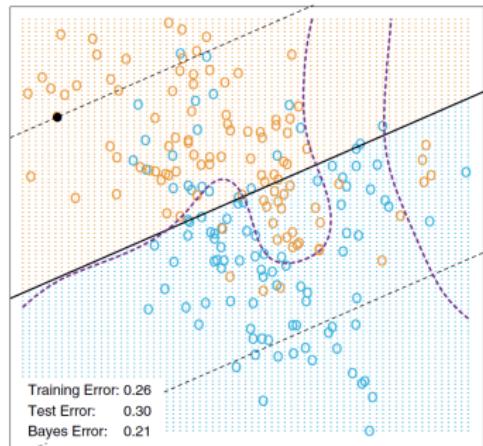
Role of λ

- ▶ Specifies the “cost” of allowing a point on the wrong side.
- ▶ If λ is very small, we’re happy to misclassify.
- ▶ For $\lambda \rightarrow \infty$, we recover the original SVM because we want $\xi_i = 0$.
- ▶ We can use cross-validation to choose it.

INFLUENCE OF MARGIN PARAMETER



$$\lambda = 100000$$



$$\lambda = 0.01$$

Hyperplane is sensitive to λ . Either way, a linear classifier isn't ideal . . .

KERNELIZING THE SVM

Primal problem with slack variables

Let's map the data into higher dimensions using the function $\phi(x_i)$,

$$\begin{aligned} \min_{w, w_0, \xi_1, \dots, \xi_n} \quad & \frac{1}{2} \|w\|^2 + \lambda \sum_{i=1}^n \xi_i \\ \text{subject to} \quad & y_i(\phi(x_i)^T w + w_0) \geq 1 - \xi_i \quad \text{for } i = 1, \dots, n \\ & \xi_i \geq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

Dual problem

Maximize over each (α_i, μ_i) and minimize over $w, w_0, \xi_1, \dots, \xi_n$

$$\mathcal{L} = \frac{1}{2} \|w\|^2 + \lambda \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i (y_i(\phi(x_i)^T w + w_0) - 1 + \xi_i) - \sum_{i=1}^n \mu_i \xi_i$$

$$\text{subject to } \alpha_i \geq 0, \quad \mu_i \geq 0, \quad y_i(\phi(x_i)^T w + w_0) - 1 + \xi_i \geq 0$$

KERNELIZING THE SVM

Dual problem

Minimizing for w , w_0 and each ξ_i , we find

$$w = \sum_{i=1}^n \alpha_i y_i x_i, \quad \sum_{i=1}^n \alpha_i y_i = 0, \quad \lambda - \alpha_i - \mu_i = 0$$

If we plug w and $\mu_i = \lambda - \alpha_i$ back into the \mathcal{L} , we have the dual problem

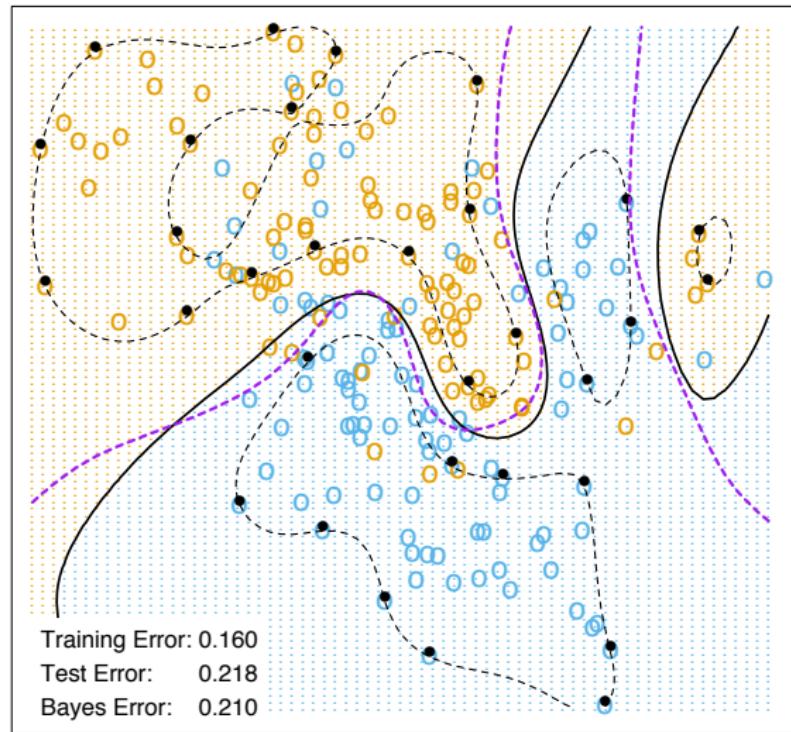
$$\max_{\alpha_1, \dots, \alpha_n} \quad \mathcal{L} = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \underbrace{\phi(x_i)^T \phi(x_j)}_{K(x_i, x_j)}$$

$$\text{subject to} \quad \sum_{i=1}^n \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq \lambda$$

Classification: Using the solution $w = \sum_{i=1}^n \alpha_i y_i \phi(x_i)$, declare

$$y_0 = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i \phi(x_0)^T \phi(x_i) + w_0\right) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(x_0, x_i) + w_0\right)$$

KERNELIZING THE SVM



Black solid line
SVM decision boundary

Classification rule

$$\text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(x_0, x_i) + w_0\right)$$

Dots
Support vectors ($\alpha_i > 0$)

Purple line
A Bayes classifier.

SUMMARY: SUPPORT VECTOR MACHINE

Basic SVM

- ▶ Linear classifier for linearly separable data.
- ▶ Position of affine hyperplane is determined to maximize the margin.
- ▶ The dual is a convex, so we can find exact solution with optimization.

Full-fledged SVM

Ingredient	Purpose
Maximum margin	Good generalization properties
Slack variables	Overlapping classes, robust against outliers
Kernel	Nonlinear decision boundary

Use in practice

- ▶ Software packages (many options)
- ▶ Choose a kernel function (e.g., RBF)
- ▶ Cross-validate λ parameter and RBF kernel width

ColumbiaX: Machine Learning

Lecture 12

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

DECISION TREES

DECISION TREES

A *decision tree* maps input $x \in \mathbb{R}^d$ to output y using binary decision rules:

- ▶ Each node in the tree has a *splitting rule*.
- ▶ Each leaf node is associated with an output value (outputs can repeat).

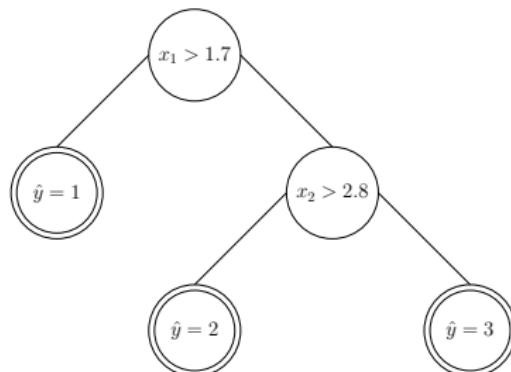
Each splitting rule is of the form

$$h(x) = \mathbb{1}\{x_j > t\}$$

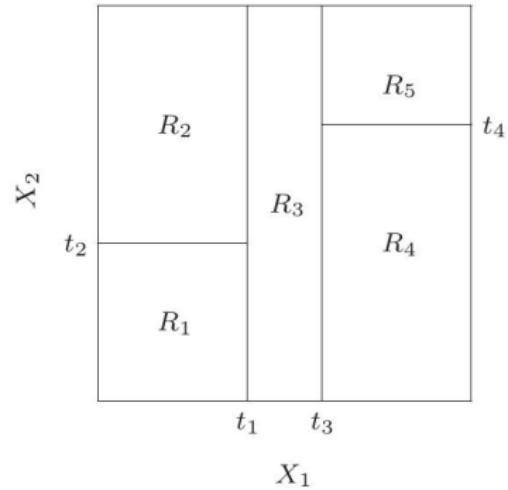
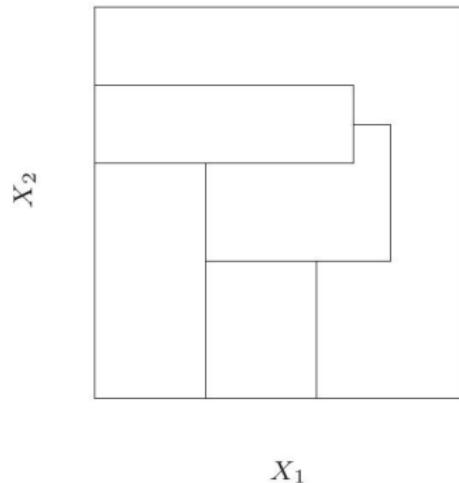
for some dimension j of x and $t \in \mathbb{R}$.

Using these transition rules, a path to a *leaf node* gives the prediction.

(One-level tree = *decision stump*)



REGRESSION TREES

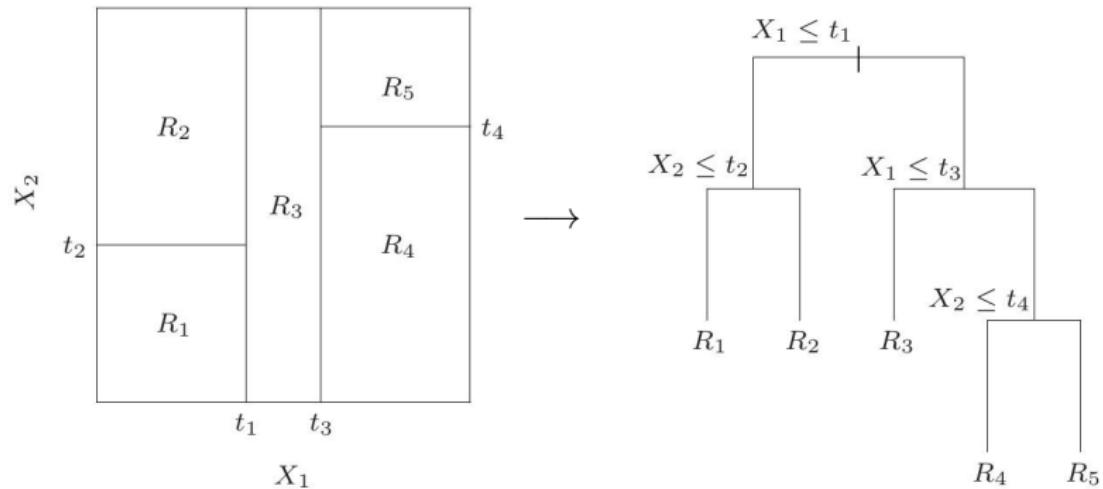


Motivation: Partition the space so that data in a region have same prediction

Left: Difficult to define a “rule”.

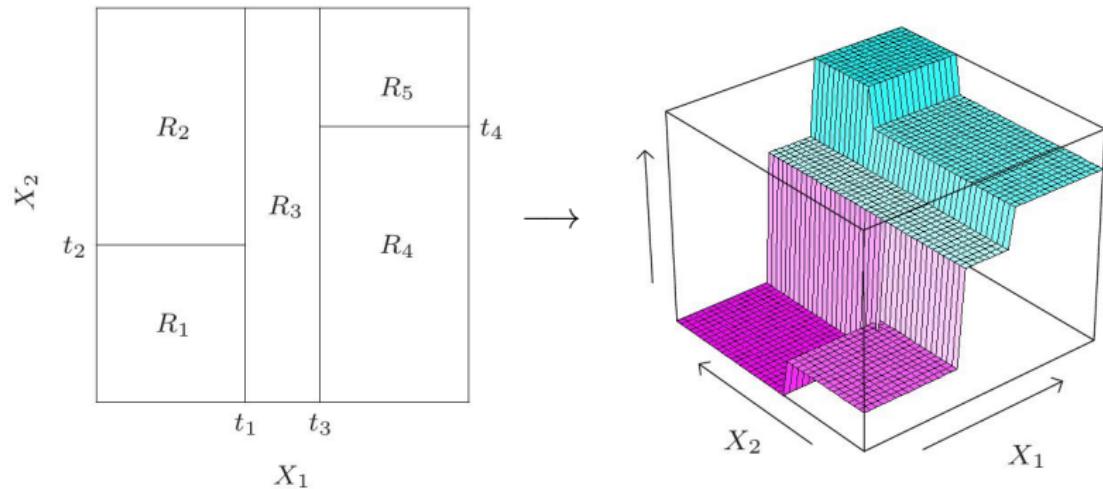
Right: Easy to define a recursive splitting rule.

REGRESSION TREES



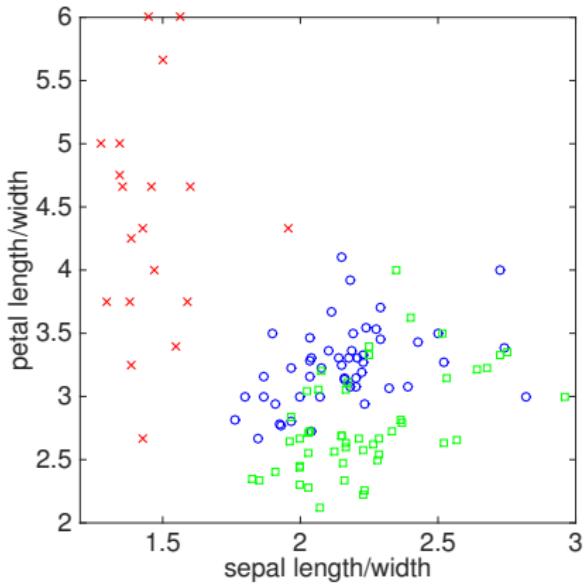
If we think in terms of trees, we can define a simple rule for partitioning the space. The left and right figures represent the same regression function.

REGRESSION TREES



Adding an output dimension to the figure (right), we can see how regression trees can learn a step function approximation to the data.

CLASSIFICATION TREES (EXAMPLE)

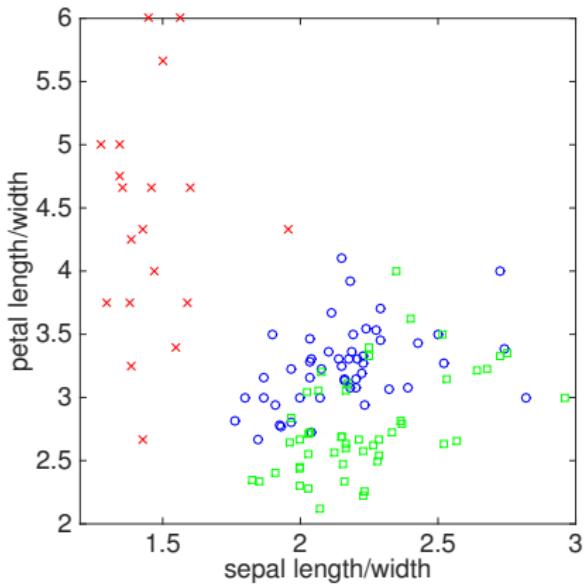


Classifying irises using sepal and petal measurements:

- ▶ $x \in \mathbb{R}^2, y \in \{1, 2, 3\}$
- ▶ $x_1 = \text{ratio of sepal length to width}$
- ▶ $x_2 = \text{ratio of petal length to width}$



CLASSIFICATION TREES (EXAMPLE)

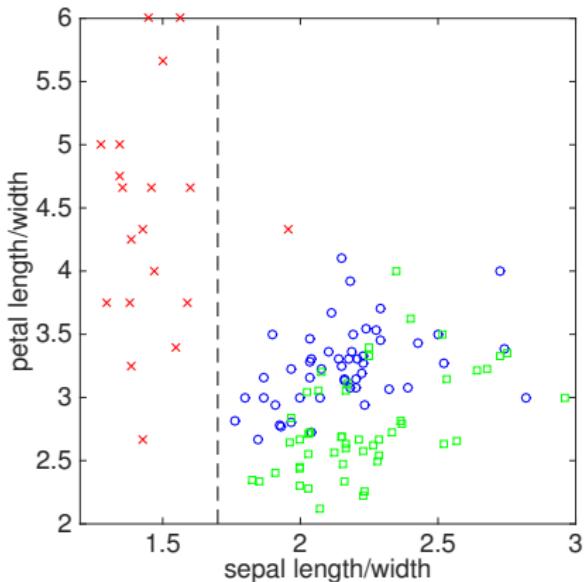


Classifying irises using sepal and petal measurements:

- ▶ $x \in \mathbb{R}^2, y \in \{1, 2, 3\}$
- ▶ $x_1 = \text{ratio of sepal length to width}$
- ▶ $x_2 = \text{ratio of petal length to width}$

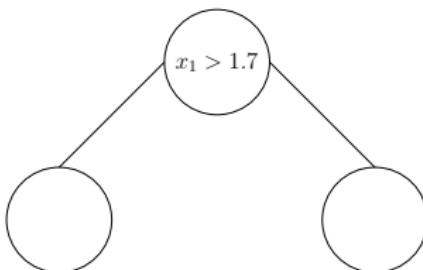
A circular diagram with a double-lined border. Inside the circle, the text $\hat{y} = 2$ is centered, representing a predicted class value.

CLASSIFICATION TREES (EXAMPLE)

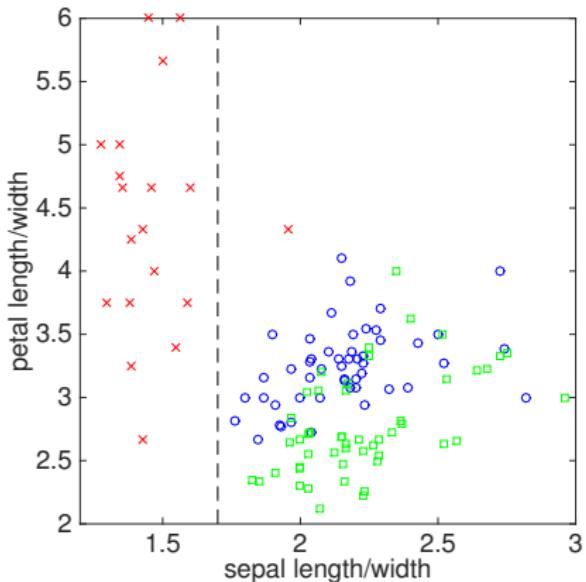


Classifying irises using sepal and petal measurements:

- ▶ $x \in \mathbb{R}^2, y \in \{1, 2, 3\}$
- ▶ x_1 = ratio of sepal length to width
- ▶ x_2 = ratio of petal length to width

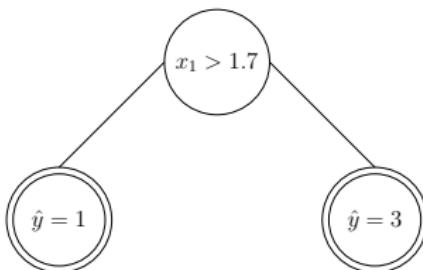


CLASSIFICATION TREES (EXAMPLE)

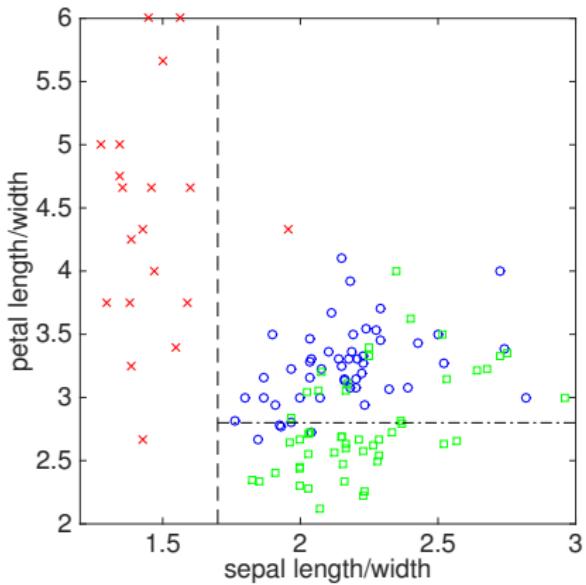


Classifying irises using sepal and petal measurements:

- ▶ $x \in \mathbb{R}^2, y \in \{1, 2, 3\}$
- ▶ $x_1 = \text{ratio of sepal length to width}$
- ▶ $x_2 = \text{ratio of petal length to width}$

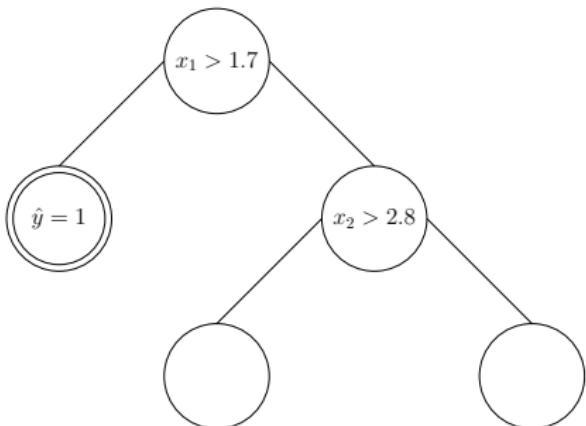


CLASSIFICATION TREES (EXAMPLE)

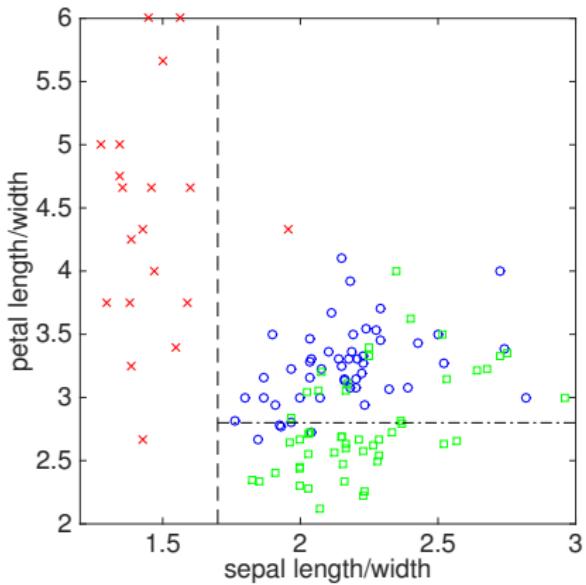


Classifying irises using sepal and petal measurements:

- ▶ $x \in \mathbb{R}^2, y \in \{1, 2, 3\}$
- ▶ $x_1 = \text{ratio of sepal length to width}$
- ▶ $x_2 = \text{ratio of petal length to width}$

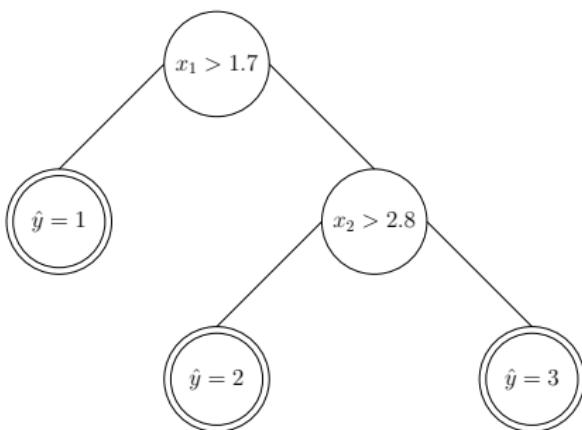


CLASSIFICATION TREES (EXAMPLE)

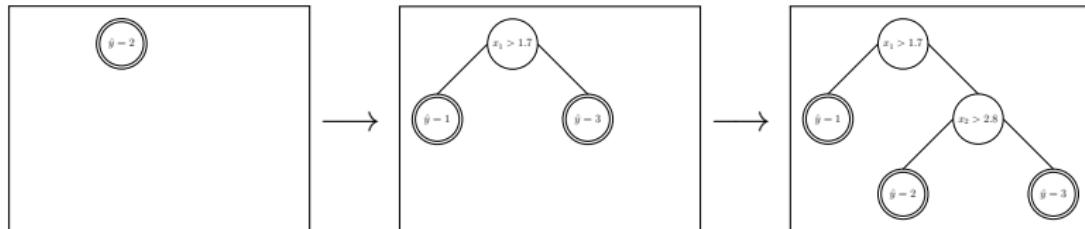


Classifying irises using sepal and petal measurements:

- ▶ $x \in \mathbb{R}^2, y \in \{1, 2, 3\}$
- ▶ $x_1 = \text{ratio of sepal length to width}$
- ▶ $x_2 = \text{ratio of petal length to width}$



BASIC DECISION TREE LEARNING ALGORITHM



The basic method for learning trees is with a top-down greedy algorithm.

- ▶ Start with a single leaf node containing all data
- ▶ Loop through the following steps:
 - ▶ Pick the leaf to split that reduces uncertainty the most.
 - ▶ Figure out the \leq decision rule on one of the dimensions.
- ▶ Stopping rule discussed later.

Label/response of the leaf is majority-vote/average of data assigned to it.

GROWING A REGRESSION TREE

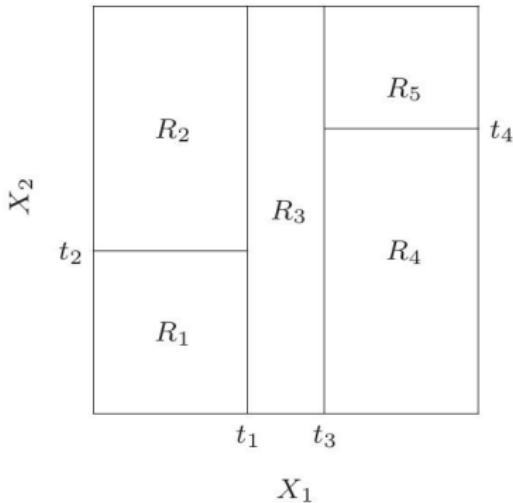
How do we grow a regression tree?

- For M regions of the space, R_1, \dots, R_M ,
the prediction function is

$$f(x) = \sum_{m=1}^M c_m \mathbb{1}\{x \in R_m\}.$$

So for a fixed M , we need R_m and c_m .

Goal: Try to minimize $\sum_i (y_i - f(x_i))^2$.



- Find c_m given R_m : Simply the average of all y_i for which $x_i \in R_m$.
- How do we find regions? Consider splitting region R at value s of dim j :
 - Define $R^-(j, s) = \{x_i \in R | x_i(j) \leq s\}$ and $R^+(j, s) = \{x_i \in R | x_i(j) > s\}$
 - For each dimension j , calculate the best splitting point s for that dimension.
 - Do this for each region (leaf node). Pick the one that reduces the objective most.

GROWING A CLASSIFICATION TREE

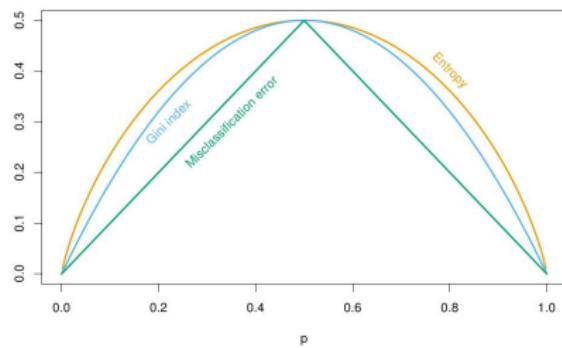
For regression: Squared error is a natural way to define the splitting rule.

For classification: Need some measure of how badly a region classifies data and how much it can improve if it's split.

K-class problem: For all $x \in R_m$, let p_k be empirical fraction labeled k .

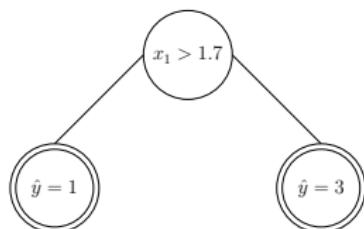
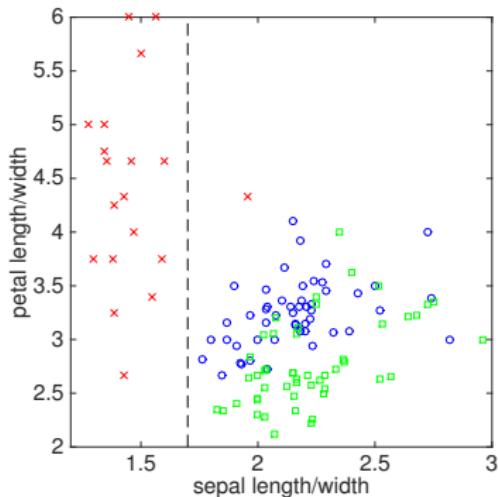
Measures of quality of R_m include

1. Classification error: $1 - \max_k p_k$
2. Gini index: $1 - \sum_k p_k^2$
3. Entropy: $-\sum_k p_k \ln p_k$



- ▶ These are all *maximized* when p_k is uniform on the K classes in R_m .
- ▶ These are *minimized* when $p_k = 1$ for some k (R_m only contains one class)

GROWING A CLASSIFICATION TREE



Search R_1 and R_2 for splitting options.

1. R_1 : $y = 1$ leaf classifies perfectly
2. R_2 : $y = 3$ leaf has Gini index

$$\begin{aligned} u(R_2) &= 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2 \\ &= 0.5098 \end{aligned}$$

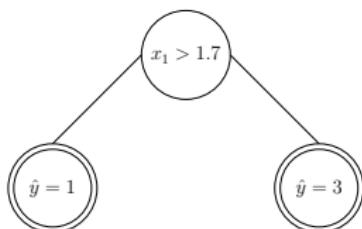
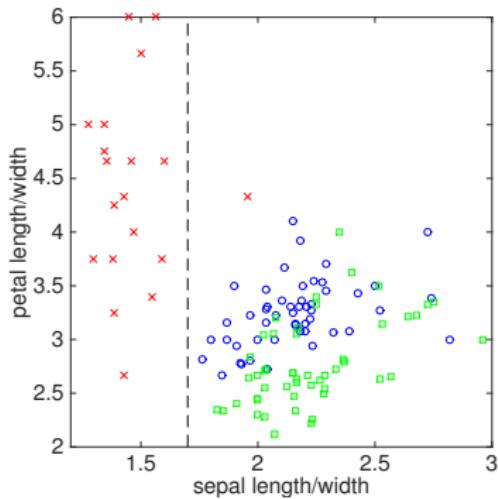
Gini improvement from split R_m to R_m^- & R_m^+ :

$$u(R_m) - \left(p_{R_m^-} \cdot u(R_m^-) + p_{R_m^+} \cdot u(R_m^+) \right)$$

$p_{R_m^+}$: Fraction of data in R_m split into R_m^+ .

$u(R_m^+)$: New quality measure in region R_m^+ .

GROWING A CLASSIFICATION TREE

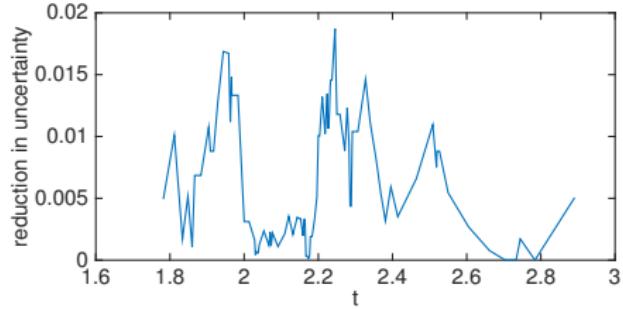


Search R_1 and R_2 for splitting options.

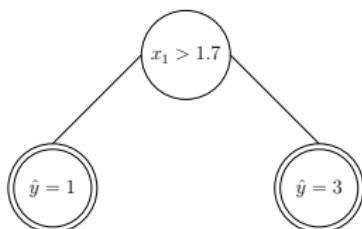
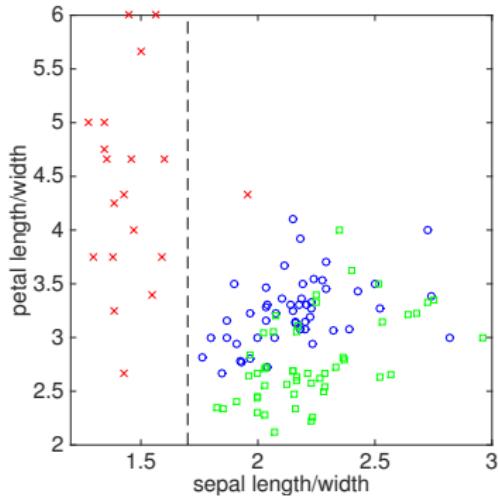
1. R_1 : $y = 1$ leaf classifies perfectly
2. R_2 : $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2 \\ = 0.5098$$

Check split R_2 with $\mathbb{1}\{x_1 > t\}$



GROWING A CLASSIFICATION TREE

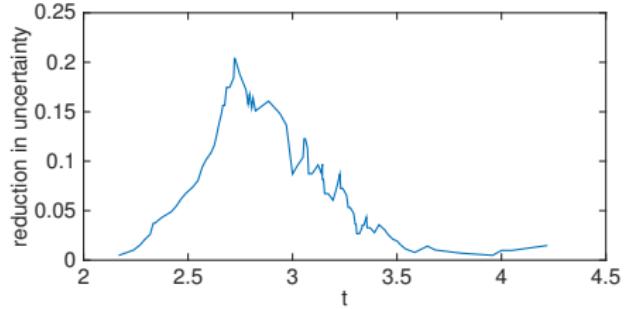


Search R_1 and R_2 for splitting options.

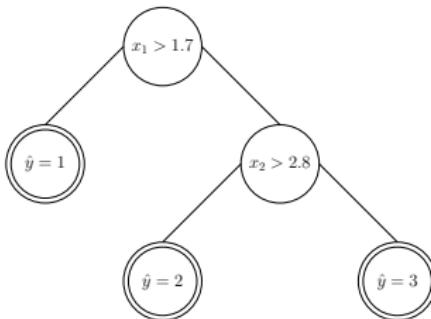
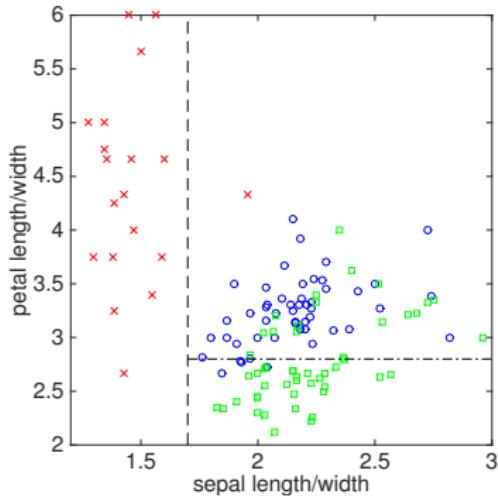
1. R_1 : $y = 1$ leaf classifies perfectly
2. R_2 : $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2 \\ = 0.5098$$

Check split R_2 with $\mathbb{1}\{x_2 > t\}$



GROWING A CLASSIFICATION TREE

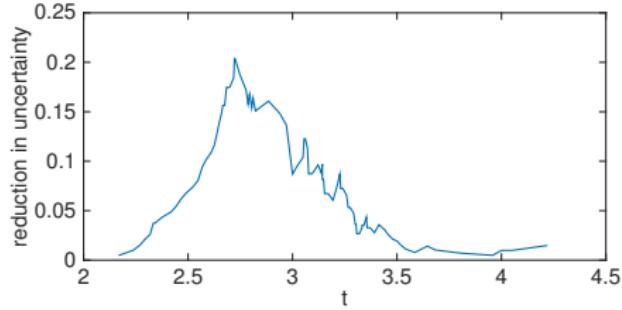


Search R_1 and R_2 for splitting options.

1. R_1 : $y = 1$ leaf classifies perfectly
2. R_2 : $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2 \\ = 0.5098$$

Check split R_2 with $\mathbb{1}\{x_2 > t\}$



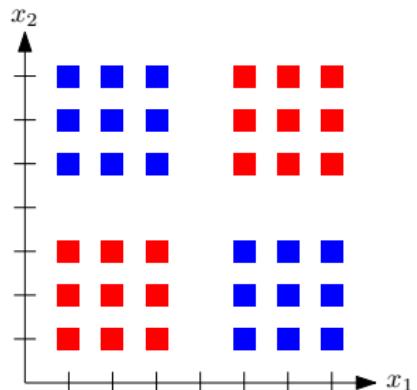
PRUNING A TREE

Q: When should we stop growing a tree?

A: Uncertainty reduction is not best way.

Example: Any split of x_1 or x_2 at right will show *zero* reduction in uncertainty.

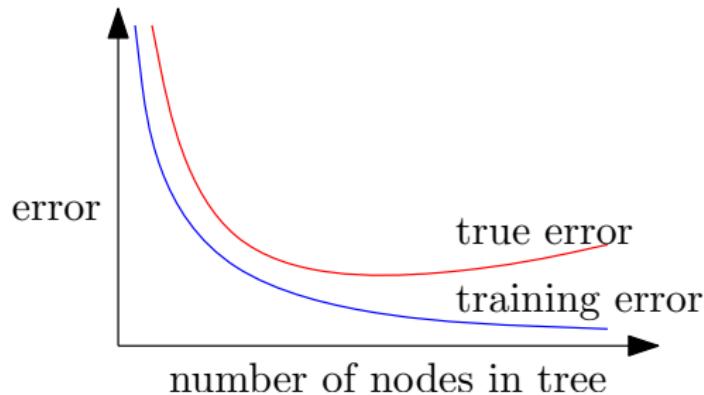
However, we can learn a perfect tree on this data by partitioning in quadrants.



Pruning is the method most often used. Grow the tree to a very large size. Then use an algorithm to trim it back.

(We won't cover the algorithm, but mention that it's non-trivial.)

OVERFITTING



- ▶ *Training error* goes to zero as size of tree increases.
- ▶ *Testing error* decreases, but then increases because of *overfitting*.

THE BOOTSTRAP

THE BOOTSTRAP: A RESAMPLING TECHNIQUE

We briefly present a technique called the *bootstrap*. This statistical technique is used as the basis for learning *ensemble classifiers*.

Bootstrap

Bootstrap (i.e., resampling) is a technique for improving estimators.

Resampling = Sampling from the empirical distribution of the data

Application to ensemble methods

- ▶ We will use resampling to generate many “mediocre” classifiers.
- ▶ We then discuss how “bagging” these classifiers improves performance.
- ▶ First, we cover the bootstrap in a simpler context.

BOOTSTRAP: BASIC ALGORITHM

Input

- ▶ A sample of data x_1, \dots, x_n .
- ▶ An estimation rule \hat{S} of a statistic S . For example, $\hat{S} = \text{med}(x_{1:n})$ estimates the true median S of the unknown distribution on x .

Bootstrap algorithm

1. Generate *bootstrap samples* $\mathcal{B}_1, \dots, \mathcal{B}_B$.
 - Create \mathcal{B}_b by picking points from $\{x_1, \dots, x_n\}$ randomly n times.
 - A particular x_i can appear in \mathcal{B}_b many times (it's simply duplicated).
2. Evaluate the estimator on each \mathcal{B}_b by pretending it's the data set:

$$\hat{S}_b := \hat{S}(\mathcal{B}_b)$$

3. Estimate the mean and variance of \hat{S} :

$$\mu_B = \frac{1}{B} \sum_{b=1}^B \hat{S}_b, \quad \sigma_B^2 = \frac{1}{B} \sum_{b=1}^B (\hat{S}_b - \mu_B)^2$$

EXAMPLE: VARIANCE ESTIMATION OF THE MEDIAN

- ▶ The median of x_1, \dots, x_n (for $x \in \mathbb{R}$) is found by simply sorting them and taking the middle one, or the average of the two middle ones.
 - ▶ How confident can we be in the estimate $\text{median}(x_1, \dots, x_n)$?
 - ▶ Find its variance.
 - ▶ But how? Answer: By bootstrapping the data.
1. Generate bootstrap data sets $\mathcal{B}_1, \dots, \mathcal{B}_B$.
 2. Calculate: (notice that \hat{S}_{mean} is the mean of the median)

$$\hat{S}_{\text{mean}} = \frac{1}{B} \sum_{b=1}^B \text{median}(\mathcal{B}_b), \quad \hat{S}_{\text{var}} = \frac{1}{B} \sum_{b=1}^B \left(\text{median}(\mathcal{B}_b) - \hat{S}_{\text{mean}} \right)^2$$

- ▶ The procedure is remarkably simple, but has a lot of theory behind it.

BAGGING AND RANDOM FORESTS

BAGGING

Bagging uses the bootstrap for regression or classification:

Bagging = Bootstrap aggregation

Algorithm

For $b = 1, \dots, B$:

1. Draw a bootstrap sample \mathcal{B}_b of size n from training data.
 2. Train a classifier or regression model f_b on \mathcal{B}_b .
- For a new point x_0 , compute:

$$f_{\text{avg}}(x_0) = \frac{1}{B} \sum_{b=1}^B f_b(x_0)$$

- For regression, $f_{\text{avg}}(x_0)$ is the prediction.
- For classification, view $f_{\text{avg}}(x_0)$ as an average over B votes. Pick the majority.

EXAMPLE: BAGGING TREES

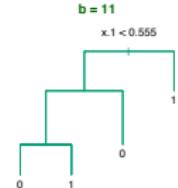
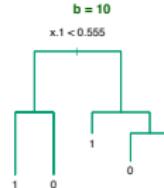
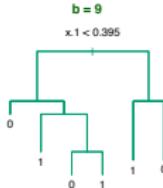
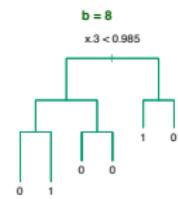
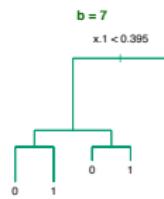
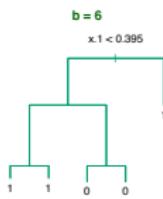
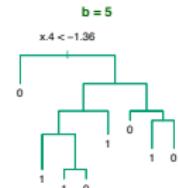
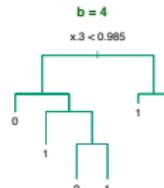
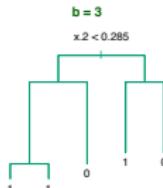
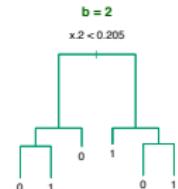
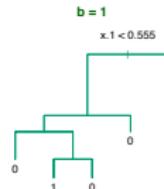
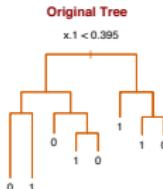
- ▶ Binary classification, $x \in \mathbb{R}^5$.

- ▶ Note the variation among bootstrapped trees.

- ▶ Take-home message:

With bagging, each tree doesn't have to be great, just "ok".

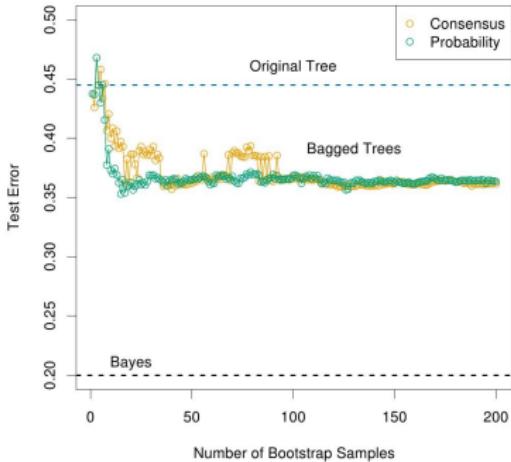
- ▶ Bagging often improves results *when the function is non-linear.*



RANDOM FORESTS

Drawbacks of Bagging

- ▶ Bagging works on trees because of the bias-variance tradeoff (\uparrow bias, \downarrow variance).
- ▶ However, the bagged trees are correlated.
- ▶ In general, when bootstrap samples are correlated, the benefit of bagging decreases.



Random Forests

Modification of bagging where trees are designed to reduce correlation.

- ▶ A very simple modification.
- ▶ Still learn a tree on each bootstrap set, \mathcal{B}_b .
- ▶ To split a region, only consider random subset of dimensions of $x \in \mathbb{R}^d$.

RANDOM FORESTS: ALGORITHM

Training

Input parameter: m — a positive integer with $m < d$, often $m \approx \sqrt{d}$

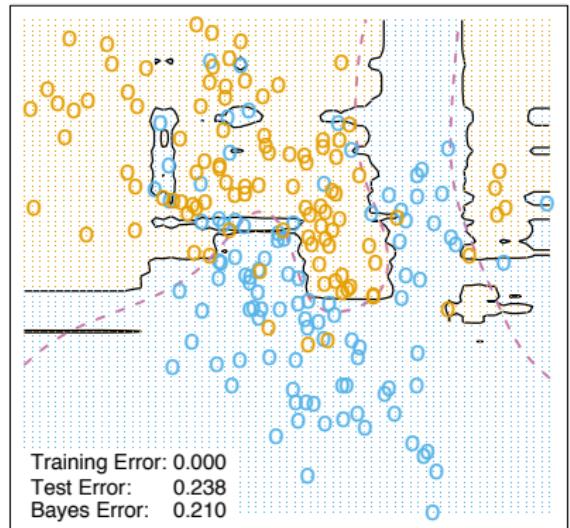
For $b = 1, \dots, B$:

1. Draw a bootstrap sample \mathcal{B}_b of size n from the training data.
 2. Train a tree classifier on \mathcal{B}_b , where each split is computed as follows:
 - ▶ Randomly select m dimensions of $x \in \mathbb{R}^d$, newly chosen for each b .
 - ▶ Make the best split restricted to that subset of dimensions.
- ▶ Bagging for trees: Bag trees learned using the original algorithm.
- ▶ Random forests: Bag trees learned using algorithm on this slide.

RANDOM FORESTS

Example problem

- ▶ Random forest classification.
- ▶ Forest size: A few hundred trees.
- ▶ Notice there is a tendency to align decision boundary with the axis.



ColumbiaX: Machine Learning

Lecture 13

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

BOOSTING

BAGGING CLASSIFIERS

Algorithm: Bagging binary classifiers

Given $(x_1, y_1), \dots, (x_n, y_n)$, $x \in \mathcal{X}$, $y \in \{-1, +1\}$

- ▶ For $b = 1, \dots, B$
 - ▶ Sample a bootstrap dataset \mathcal{B}_b of size n . For each entry in \mathcal{B}_b , select (x_i, y_i) with probability $\frac{1}{n}$. Some (x_i, y_i) will repeat and some won't appear in \mathcal{B}_b .
 - ▶ Learn a classifier f_b using data in \mathcal{B}_b .
- ▶ Define the classification rule to be

$$f_{bag}(x_0) = \text{sign} \left(\sum_{b=1}^B f_b(x_0) \right).$$

- ▶ With bagging, we observe that a *committee* of classifiers votes on a label.
- ▶ Each classifier is learned on a *bootstrap sample* from the data set.
- ▶ Learning a collection of classifiers is referred to as an *ensemble method*.

BOOSTING

How is it that a committee of blockheads can somehow arrive at highly reasoned decisions, despite the weak judgment of the individual members?

- Schapire & Freund, “Boosting: Foundations and Algorithms”

Boosting is another powerful method for ensemble learning. It is similar to bagging in that a set of classifiers are combined to make a better one.

It works for any classifier, but a “weak” one that is easy to learn is usually chosen. (weak = accuracy a little better than random guessing)

Short history

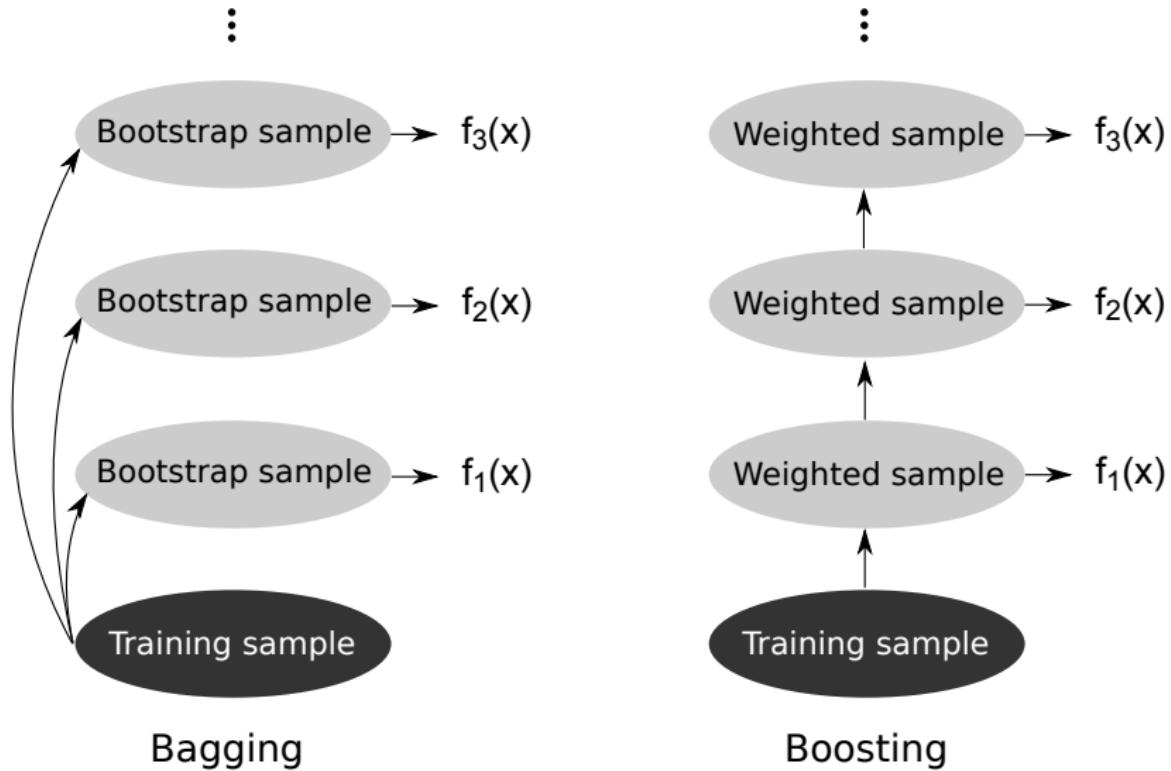
1984 : Leslie Valiant and Michael Kearns ask if “boosting” is possible.

1989 : Robert Schapire creates first boosting algorithm.

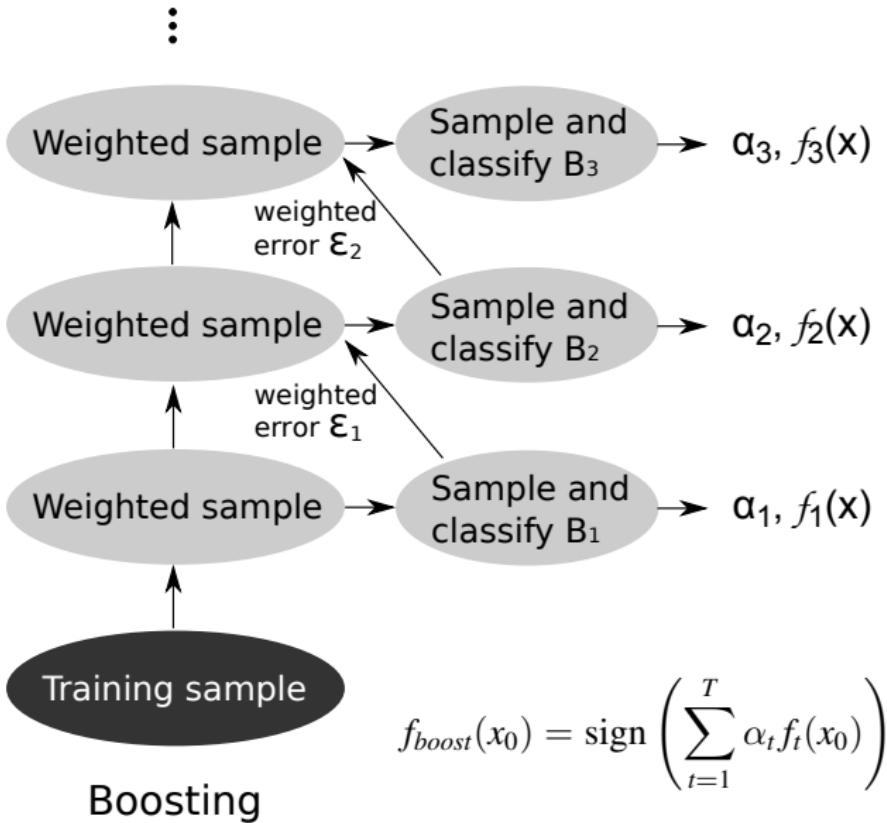
1990 : Yoav Freund creates an optimal boosting algorithm.

1995 : Freund and Schapire create AdaBoost (Adaptive Boosting),
the major boosting algorithm.

BAGGING VS BOOSTING (OVERVIEW)



THE ADABoost ALGORITHM (SAMPLING VERSION)



THE ADABoost ALGORITHM (SAMPLING VERSION)

Algorithm: Boosting a binary classifier

Given $(x_1, y_1), \dots, (x_n, y_n)$, $x \in \mathcal{X}$, $y \in \{-1, +1\}$, set $w_1(i) = \frac{1}{n}$

► For $t = 1, \dots, T$

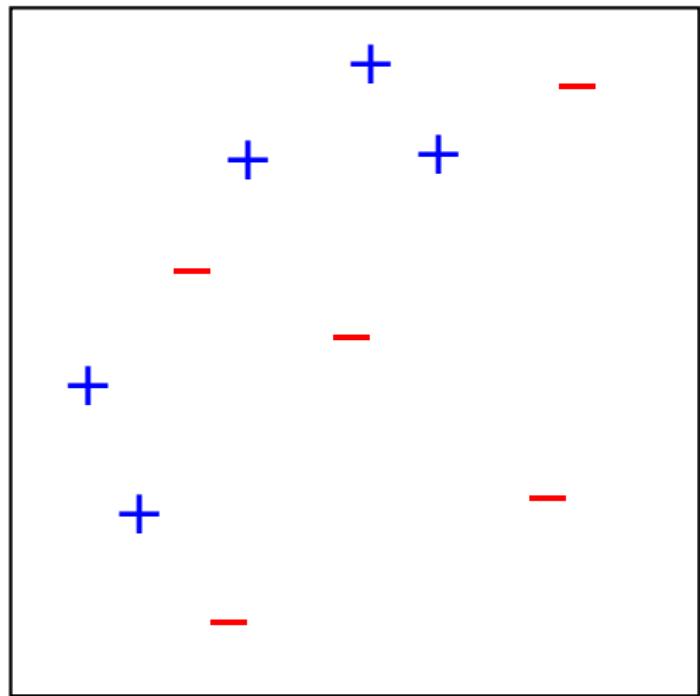
1. Sample a bootstrap dataset \mathcal{B}_t of size n according to distribution w_t .
Notice we pick (x_i, y_i) with probability $w_t(i)$ and not $\frac{1}{n}$.
2. Learn a classifier f_t using data in \mathcal{B}_t .
3. Set $\epsilon_t = \sum_{i=1}^n w_t(i) \mathbb{1}\{y_i \neq f_t(x_i)\}$ and $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$.
4. Scale $\hat{w}_{t+1}(i) = w_t(i) e^{-\alpha_t y_i f_t(x_i)}$ and set $w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)}$.

► Set the classification rule to be

$$f_{\text{boost}}(x_0) = \text{sign} \left(\sum_{t=1}^T \alpha_t f_t(x_0) \right).$$

Comment: Description usually simplified to “learn classifier f_t using distribution w_t .”

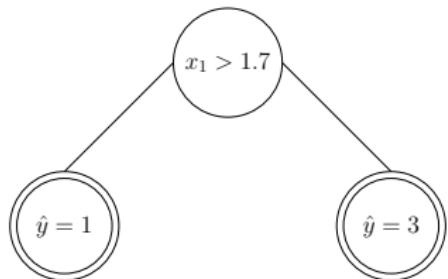
BOOSTING A DECISION STUMP (EXAMPLE 1)



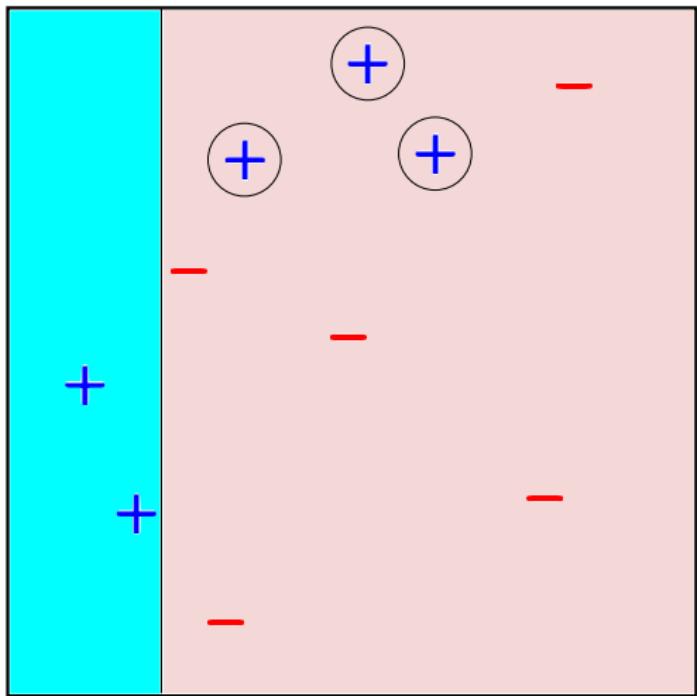
Original data

Uniform distribution, w_1
Learn *weak classifier*

Here: Use a decision stump



BOOSTING A DECISION STUMP (EXAMPLE 1)

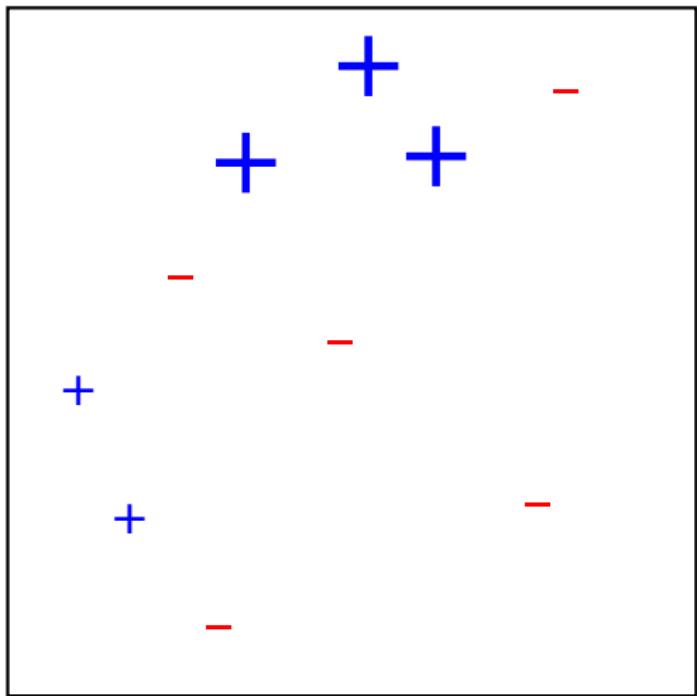


Round 1 classifier

Weighted error: $\epsilon_1 = 0.3$

Weight update: $\alpha_1 = 0.42$

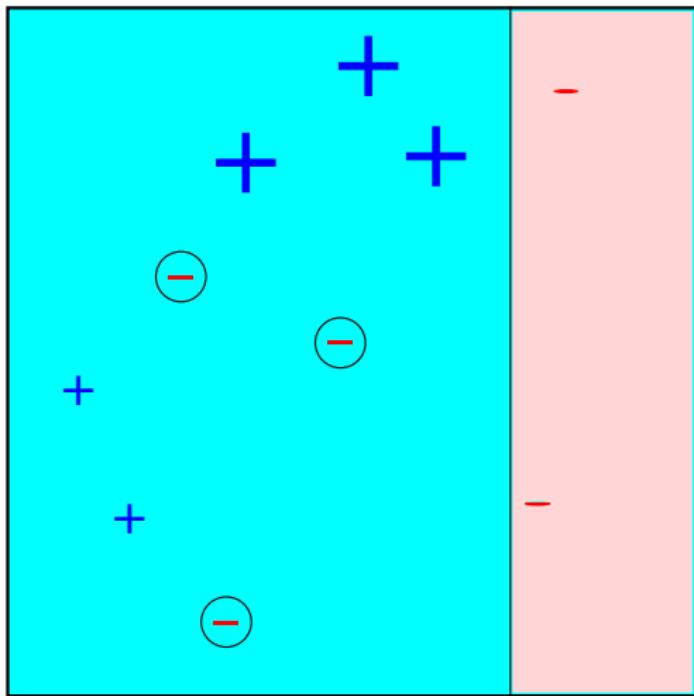
BOOSTING A DECISION STUMP (EXAMPLE 1)



Weighted data

After round 1

BOOSTING A DECISION STUMP (EXAMPLE 1)

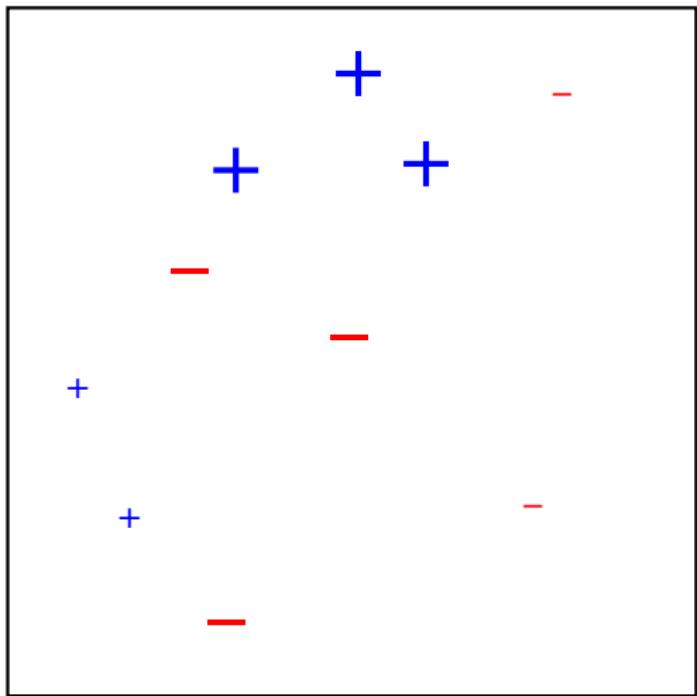


Round 2 classifier

Weighted error: $\epsilon_2 = 0.21$

Weight update: $\alpha_2 = 0.65$

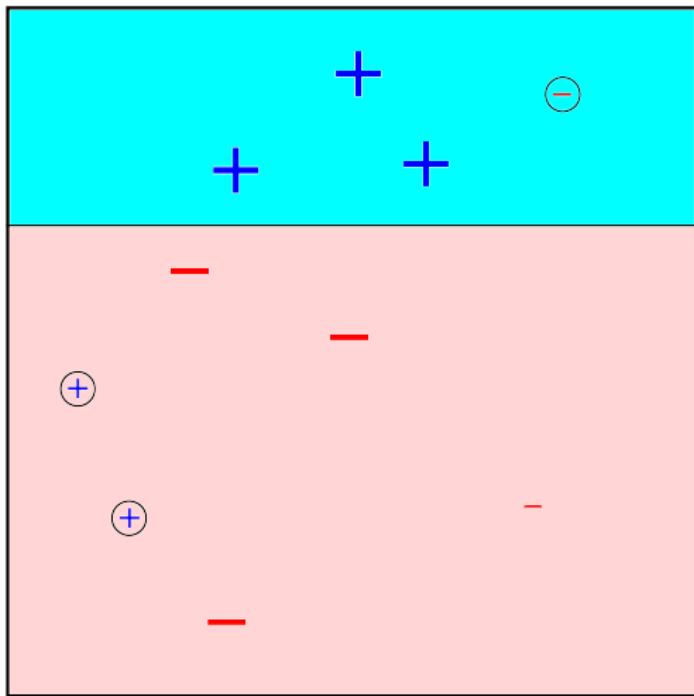
BOOSTING A DECISION STUMP (EXAMPLE 1)



Weighted data

After round 2

BOOSTING A DECISION STUMP (EXAMPLE 1)

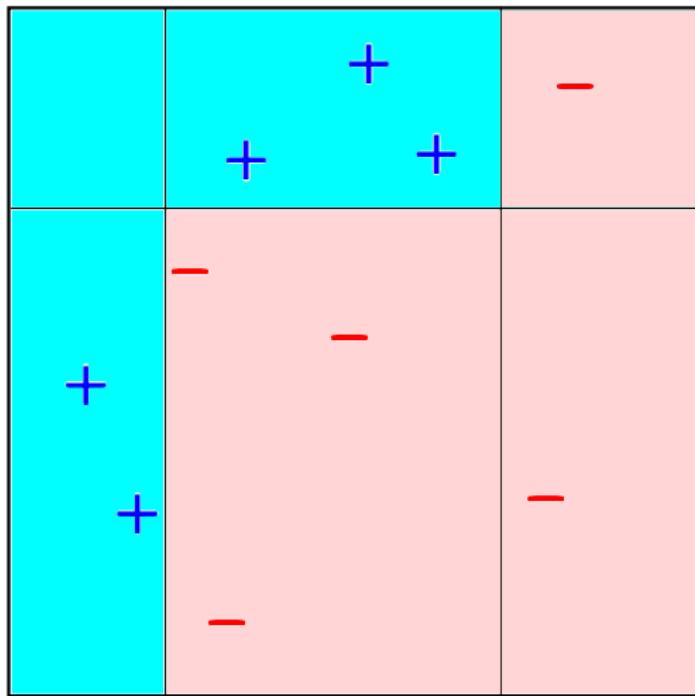


Round 2 classifier

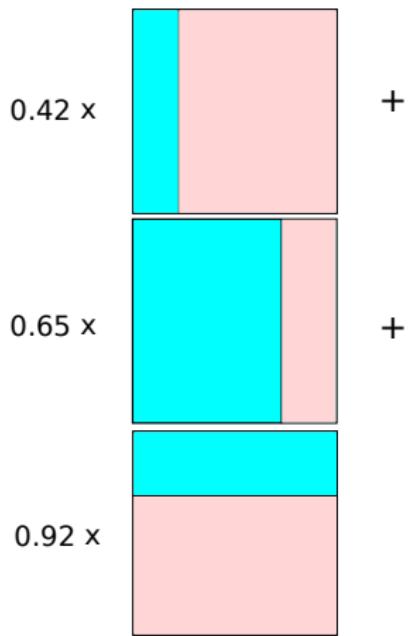
Weighted error: $\epsilon_3 = 0.14$

Weight update: $\alpha_3 = 0.92$

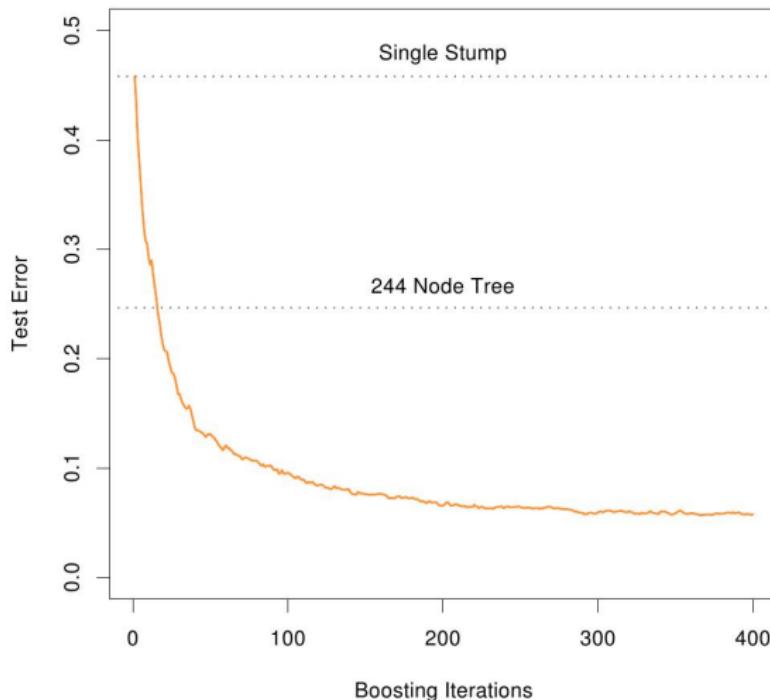
BOOSTING A DECISION STUMP (EXAMPLE 1)



Classifier after three rounds



BOOSTING A DECISION STUMP (EXAMPLE 2)



Example problem

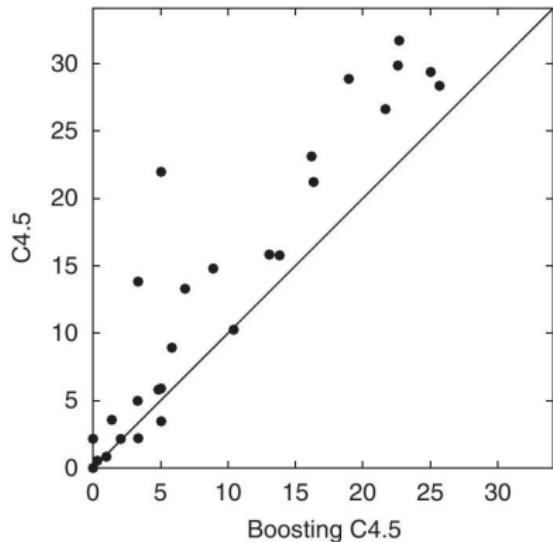
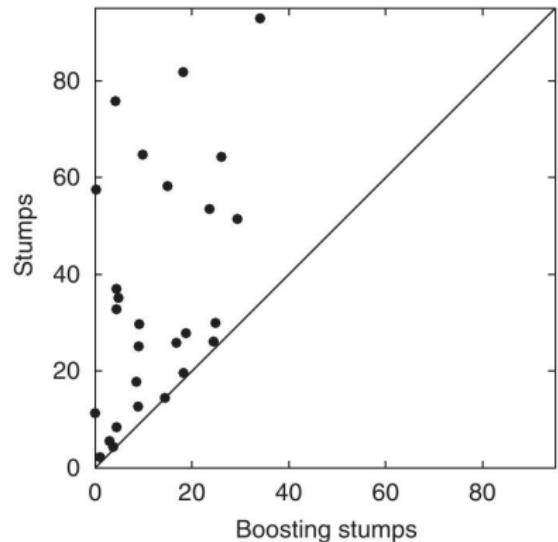
Random guessing
50% error

Decision stump
45.8% error

Full decision tree
24.7% error

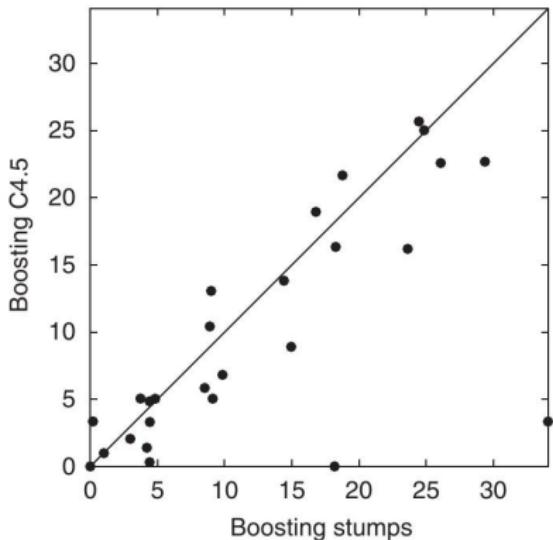
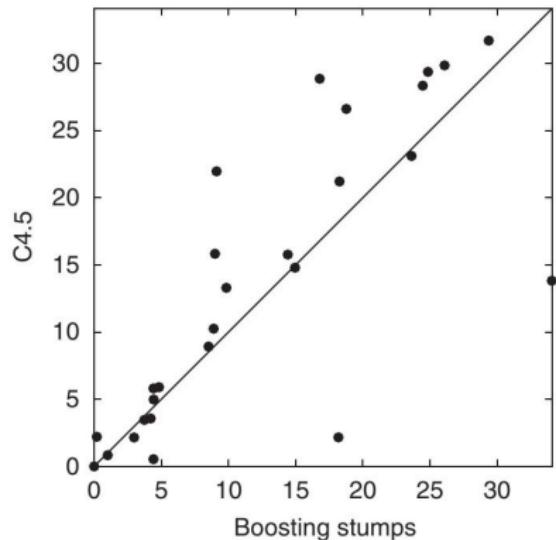
Boosted stump
5.8% error

BOOSTING



Point = one dataset. Location = error rate w/ and w/o boosting. The boosted version of the same classifier almost always produces better results.

BOOSTING



(left) Boosting a bad classifier is often better than not boosting a good one.
(right) Boosting a good classifier is often better, but can take more time.

BOOSTING AND FEATURE MAPS

Q: What makes boosting work so well?

A: This is a well-studied question. We will present one analysis later, but we can also give intuition by tying it in with what we've already learned.

The classification for a new x_0 from boosting is

$$f_{\text{boost}}(x_0) = \text{sign} \left(\sum_{t=1}^T \alpha_t f_t(x_0) \right).$$

Define $\phi(x) = [f_1(x), \dots, f_T(x)]^\top$, where each $f_t(x) \in \{-1, +1\}$.

- ▶ We can think of $\phi(x)$ as a high dimensional feature map of x .
- ▶ The vector $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_T]^\top$ corresponds to a hyperplane.
- ▶ So the classifier can be written $f_{\text{boost}}(x_0) = \text{sign}(\phi(x_0)^\top \boldsymbol{\alpha})$.
- ▶ Boosting learns the feature mapping and hyperplane simultaneously.

APPLICATION: FACE DETECTION

FACE DETECTION (VIOLA & JONES, 2001)

Problem: Locate the faces in an image or video.

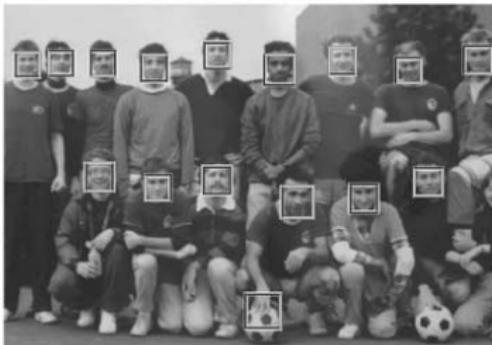
Processing: Divide image into patches of different scales, e.g., 24×24 , 48×48 , etc. Extract *features* from each patch.

Classify each patch as face or no face using a *boosted decision stump*. This can be done in real-time, for example by your digital camera (at 15 fps).



- ▶ One patch from a larger image. Mask it with many “feature extractors.”
- ▶ Each pattern gives one number, which is the sum of all pixels in black region minus sum of pixels in white region (total of 45,000+ features).

FACE DETECTION (EXAMPLE RESULTS)



ANALYSIS OF BOOSTING

ANALYSIS OF BOOSTING

Training error theorem

We can use *analysis* to make a statement about the accuracy of boosting *on the training data*.

Theorem: Under the AdaBoost framework, if ϵ_t is the weighted error of classifier f_t , then for the classifier $f_{\text{boost}}(x_0) = \text{sign}(\sum_{t=1}^T \alpha_t f_t(x_0))$,

$$\text{training error} = \frac{1}{n} \sum_{i=1}^n \mathbb{1}\{y_i \neq f_{\text{boost}}(x_i)\} \leq \exp\left(-2 \sum_{t=1}^T \left(\frac{1}{2} - \epsilon_t\right)^2\right).$$

Even if each ϵ_t is only a little better than random guessing, the sum over T classifiers can lead to a large negative value in the exponent when T is large.

For example, if we set:

$$\epsilon_t = 0.45, T = 1000 \rightarrow \text{training error} \leq 0.0067.$$

PROOF OF THEOREM

Setup

We break the proof into three steps. It is an application of the fact that

$$\text{if } \underbrace{a < b}_{\text{Step 2}} \text{ and } \underbrace{b < c}_{\text{Step 3}} \text{ then } \underbrace{a < c}_{\text{conclusion}}$$

- ▶ Step 1 calculates the value of b .
- ▶ Steps 2 and 3 prove the two inequalities.

Also recall the following step from AdaBoost:

- ▶ Update $\hat{w}_{t+1}(i) = w_t(i)e^{-\alpha_t y_i f_t(x_i)}$.
- ▶ Normalize $w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)}$ \longrightarrow Define $Z_t = \sum_j \hat{w}_{t+1}(j)$.

PROOF OF THEOREM $(a \leq b \leq c)$

Step 1

We first want to expand the equation of the weights to show that

$$w_{T+1}(i) = \frac{1}{n} \frac{e^{-y_i \sum_{t=1}^T \alpha_t f_t(x_i)}}{\prod_{t=1}^T Z_t} = \frac{1}{n} \frac{e^{-y_i f_{\text{boost}}^{(T)}(x_i)}}{\prod_{t=1}^T Z_t} \quad (f_{\text{boost}}^{(T)} \text{ is up to step } T)$$

Derivation of Step 1:

Notice the update rule: $w_{t+1}(i) = \frac{1}{Z_t} w_t(i) e^{-\alpha_t y_i f_t(x_i)}$

Do the same expansion for $w_t(i)$ and continue until reaching $w_1(i) = \frac{1}{n}$,

$$w_{T+1}(i) = w_1(i) \frac{e^{-\alpha_1 y_i f_1(x_i)}}{Z_1} \times \cdots \times \frac{e^{-\alpha_T y_i f_T(x_i)}}{Z_T}$$

The product $\prod_{t=1}^T Z_t$ is “b**” above.** We use this form of $w_{T+1}(i)$ in Step 2.

PROOF OF THEOREM $(a \leq b \leq c)$

Step 2

Next show that the training error of $f_{boost}^{(T)}$ after T steps is $\leq \prod_{t=1}^T Z_t$.

From Step 1: $w_{T+1}(i) = \frac{1}{n} \frac{e^{-y_i f_{boost}^{(T)}(x_i)}}{\prod_{t=1}^T Z_t} \implies w_{T+1}(i) \prod_{t=1}^T Z_t = \frac{1}{n} e^{-y_i f_{boost}^{(T)}(x_i)}$

Derivation of Step 2:

(Observe that $0 < e^{z_1}$ and $1 < e^{z_2}$ for any $z_1 < 0 < z_2$.)

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n \mathbb{1}\{y_i \neq f_{boost}^{(T)}(x_i)\} &\leq \frac{1}{n} \sum_{i=1}^n e^{-y_i f_{boost}^{(T)}(x_i)} \\ &= \sum_{i=1}^n w_{T+1}(i) \prod_{t=1}^T Z_t \\ &= \prod_{t=1}^T Z_t \end{aligned}$$

“a” is the training error – the quantity we care about.

PROOF OF THEOREM $(a \leq b \leq c)$

Step 3

The final step is to calculate an upper bound on Z_t , and by extension $\prod_{t=1}^T Z_t$.

Derivation of Step 3:

This step is slightly more involved. It also shows why $\alpha_t := \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$.

$$\begin{aligned} Z_t &= \sum_{i=1}^n w_t(i) e^{-\alpha_t y_i f_t(x_i)} \\ &= \sum_{i : y_i = f_t(x_i)} e^{-\alpha_t} w_t(i) + \sum_{i : y_i \neq f_t(x_i)} e^{\alpha_t} w_t(i) \\ &= e^{-\alpha_t} (1 - \epsilon_t) + e^{\alpha_t} \epsilon_t \end{aligned}$$

Remember we defined $\epsilon_t = \sum_{i : y_i \neq f_t(x_i)} w_t(i)$, the probability of error for w_t .

PROOF OF THEOREM $(a \leq b \leq c)$

Derivation of Step 3 (continued):

Remember from Step 2 that

$$\text{training error} = \frac{1}{n} \sum_{i=1}^n \mathbb{1}\{y_i \neq f_{\text{boost}}(x_i)\} \leq \prod_{t=1}^T Z_t.$$

and we just showed that $Z_t = e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t}\epsilon_t$.

We want the training error to be small, so we pick α_t to *minimize* Z_t .
Minimizing, we get the value of α_t used by AdaBoost:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right).$$

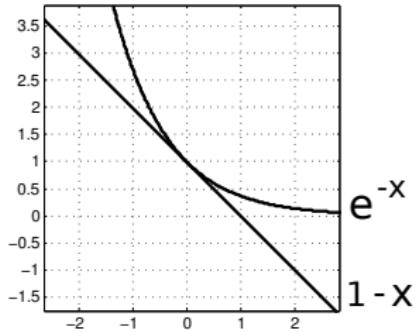
Plugging this value back in gives $Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$.

PROOF OF THEOREM $(a \leq b \leq c)$

Derivation of Step 3 (continued):

Next, re-write Z_t as

$$\begin{aligned} Z_t &= 2\sqrt{\epsilon_t(1 - \epsilon_t)} \\ &= \sqrt{1 - 4(\frac{1}{2} - \epsilon_t)^2} \end{aligned}$$



Then, use the inequality $1 - x \leq e^{-x}$ to conclude that

$$Z_t = \left(1 - 4\left(\frac{1}{2} - \epsilon_t\right)^2\right)^{\frac{1}{2}} \leq \left(e^{-4\left(\frac{1}{2} - \epsilon_t\right)^2}\right)^{\frac{1}{2}} = e^{-2\left(\frac{1}{2} - \epsilon_t\right)^2}.$$

PROOF OF THEOREM

Concluding the right inequality $(a \leq b \leq c)$

Because both sides of $Z_t \leq e^{-2(\frac{1}{2} - \epsilon_t)^2}$ are positive, we can say that

$$\prod_{t=1}^T Z_t \leq \prod_{t=1}^T e^{-2(\frac{1}{2} - \epsilon_t)^2} = e^{-2 \sum_{t=1}^T (\frac{1}{2} - \epsilon_t)^2}.$$

This concludes the “ $b \leq c$ ” portion of the proof.

Combining everything

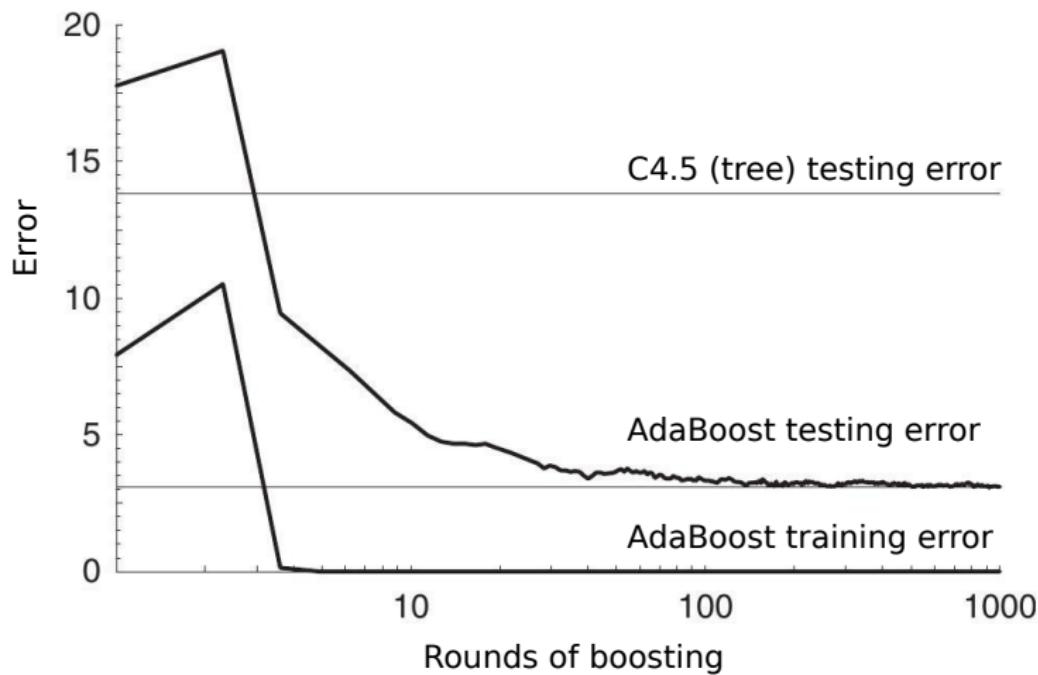
$$\text{training error} = \underbrace{\frac{1}{n} \sum_{i=1}^n \mathbb{1}\{y_i \neq f_{\text{boost}}(x_i)\}}_a \leq \underbrace{\prod_{t=1}^T Z_t}_b \leq \underbrace{e^{-2 \sum_{t=1}^T (\frac{1}{2} - \epsilon_t)^2}}_c.$$

We set out to prove “ $a < c$ ” and we did so by using “ b ” as a stepping-stone.

TRAINING VS TESTING ERROR

Q: Driving the training error to zero leads one to ask, does boosting overfit?

A: Sometimes, but very often it doesn't!



ColumbiaX: Machine Learning

Lecture 14

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

UNSUPERVISED LEARNING

SUPERVISED LEARNING

Framework of supervised learning

Given: Pairs $(x_1, y_1), \dots, (x_n, y_n)$. Think of x as input and y as output.

Learn: A function $f(x)$ that accurately predicts $y_i \approx f(x_i)$ on this data.

Goal: Use the function $f(x)$ to predict new y_0 given x_0 .

Probabilistic motivation

If we think of (x, y) as a random variable with joint distribution $p(x, y)$, then supervised learning seeks to learn the conditional distribution $p(y|x)$.

This can be done either directly or indirectly:

Directly: e.g., with logistic regression where $p(y|x) = \text{sigmoid function}$

Indirectly: e.g., with a Bayes classifier

$$y = \arg \max_k p(y=k|x) = \arg \max_k p(x|y=k)p(y=k)$$

UNSUPERVISED LEARNING

Some motivation

- ▶ The Bayes classifier factorizes the joint density as $p(x, y) = p(x|y)p(y)$.
- ▶ The joint density can also be written as $p(x, y) = p(y|x)p(x)$.
- ▶ *Unsupervised learning* focuses on the term $p(x)$ — learning $p(x|y)$ on a class-specific subset has the same “feel.” What should this be?
- ▶ This implies an underlying classification task, but often there isn’t one.

Unsupervised learning

Given: A data set x_1, \dots, x_n , where $x_i \in \mathcal{X}$, e.g., $\mathcal{X} = \mathbb{R}^d$

Define: Some model of the data (probabilistic or non-probabilistic).

Goal: Learn structure within the data set *as defined by the model*.

- ▶ Supervised learning has a clear performance metric: accuracy
- ▶ Unsupervised learning is often (but not always) more subjective

SOME TYPES OF UNSUPERVISED LEARNING

Overview of second half of course

We will discuss a few types of unsupervised learning approaches in the second half of the course.

Clustering models: Learn a partition of data x_1, \dots, x_n into groups.

- ▶ Image segmentation, data quantization, preprocessing for other models

Matrix factorization: Learn an underlying dot-product representation.

- ▶ User preference modeling, topic modeling

Sequential models: Learn a model based on sequential information.

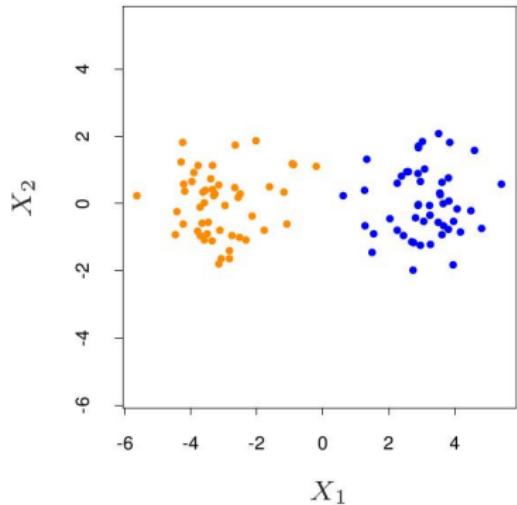
- ▶ Learn how to rank objects, target tracking

As will become evident, an unsupervised model can often be interpreted as a supervised model, or very easily turned into one.

CLUSTERING

Problem

- ▶ Given data x_1, \dots, x_n , partition it into groups called *clusters*.
- ▶ Find the clusters, given only the data.
- ▶ Observations in same group \Rightarrow “similar,” different groups \Rightarrow “different.”
- ▶ We will set how many clusters we learn.



Cluster assignment representation

For K clusters, encode cluster assignments as an indicator $c \in \{1, \dots, K\}$,

$$c_i = k \iff x_i \text{ is assigned to cluster } k$$

Clustering feels similar to classification in that we “label” an observation by its cluster assignment. The difference is that there is no ground truth.

THE K-MEANS ALGORITHM

CLUSTERING AND K-MEANS

K-means is the simplest and most fundamental clustering algorithm.

Input: x_1, \dots, x_n , where $x \in \mathbb{R}^d$.

Output: Vector \mathbf{c} of cluster assignments, and K mean vectors $\boldsymbol{\mu}$

- ▶ $\mathbf{c} = (c_1, \dots, c_n)$, $c_i \in \{1, \dots, K\}$
 - If $c_i = c_j = k$, then x_i and x_j are *clustered together* in cluster k .
- ▶ $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$, $\mu_k \in \mathbb{R}^d$ (same space as x_i)
 - Each μ_k (called a *centroid*) defines a cluster.

As usual, we need to define an *objective function*. We pick one that:

1. Tells us what are good \mathbf{c} and $\boldsymbol{\mu}$, and
2. That is easy to optimize.

K-MEANS OBJECTIVE FUNCTION

The K-means objective function can be written as

$$\boldsymbol{\mu}^*, \mathbf{c}^* = \arg \min_{\boldsymbol{\mu}, \mathbf{c}} \sum_{i=1}^n \sum_{k=1}^K \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$$

Some observations:

- ▶ K-means uses the squared Euclidean distance of x_i to the centroid μ_k .
- ▶ It only penalizes the distance of x_i to the centroid it's assigned to by c_i .

$$\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2 = \sum_{k=1}^K \sum_{i:c_i=k} \|x_i - \mu_k\|^2$$

- ▶ The objective function is “non-convex”
 - ▶ This means that we can't actually find the *optimal* $\boldsymbol{\mu}^*$ and \mathbf{c}^* .
 - ▶ We can only derive an *algorithm* for finding a *local optimum* (more later).

OPTIMIZING THE K-MEANS OBJECTIVE

Gradient-based optimization

We can't optimize the K-means objective function exactly by taking derivatives and setting to zero, so we use an iterative algorithm.

However, the algorithm we will use is different from gradient methods:

$$w \leftarrow w - \eta \nabla_w \mathcal{L} \quad (\text{gradient descent})$$

Recall: With gradient descent, when we update a parameter “ w ” we move in the direction that decreases the objective function, but

- ▶ It will almost certainly not move to the *best* value for that parameter.
- ▶ It may not even move to a better value if the step size η is too big.
- ▶ We also need the parameter w to be continuous-valued.

K-MEANS AND COORDINATE DESCENT

Coordinate descent

We will discuss a new and widely used optimization procedure in the context of K -means clustering. We want to minimize the objective function

$$\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2.$$

We split the variables into two unknown sets μ and c . We can't find their best values *at the same time* to minimize \mathcal{L} . However, we will see that

- ▶ Fixing μ we can find the best c exactly.
- ▶ Fixing c we can find the best μ exactly.

This optimization approach is called *coordinate descent*: Hold one set of parameters fixed, and optimize the other set. Then switch which set is fixed.

COORDINATE DESCENT

Coordinate descent (in the context of K-means)

Input: x_1, \dots, x_n where $x_i \in \mathbb{R}^d$. Randomly initialize $\mu = (\mu_1, \dots, \mu_K)$.

- ▶ Iterate back-and-forth between the following two steps:

1. Given μ , find the best value $c_i \in \{1, \dots, K\}$ for $i = 1, \dots, n$.
 2. Given c , find the best vector $\mu_k \in \mathbb{R}^d$ for $k = 1, \dots, K$.
-

There's a circular way of thinking about why we need to iterate:

1. Given a particular μ , we may be able to find *the best c*, but once we *change c* we can probably find a better μ .
2. Then find *the best μ* for the new-and-improved c found in #1, but now that we've changed μ , there is probably a better c .

We have to iterate because the values of μ and c *depend on each other*. This happens very frequently in unsupervised models.

K-MEANS ALGORITHM: UPDATING \boldsymbol{c}

Assignment step

Given $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$, update $\boldsymbol{c} = (c_1, \dots, c_n)$. By rewriting \mathcal{L} , we notice the independence of each c_i given $\boldsymbol{\mu}$,

$$\mathcal{L} = \underbrace{\left(\sum_{k=1}^K \mathbb{1}\{c_1 = k\} \|x_1 - \mu_k\|^2 \right)}_{\text{distance of } x_1 \text{ to its assigned centroid}} + \cdots + \underbrace{\left(\sum_{k=1}^K \mathbb{1}\{c_n = k\} \|x_n - \mu_k\|^2 \right)}_{\text{distance of } x_n \text{ to its assigned centroid}}.$$

We can minimize \mathcal{L} with respect to each c_i by minimizing each term above separately. The solution is to assign x_i to the closest centroid

$$c_i = \arg \min_k \|x_i - \mu_k\|^2.$$

Because there are only K options for each c_i , there are no derivatives. Simply calculate all the possible values for c_i and pick the best (smallest) one.

K-MEANS ALGORITHM: UPDATING μ

Update step

Given $c = (c_1, \dots, c_n)$, update $\mu = (\mu_1, \dots, \mu_K)$. For a given c , we can break \mathcal{L} into K clusters defined by c so that each μ_i is independent.

$$\mathcal{L} = \underbrace{\left(\sum_{i=1}^N \mathbb{1}\{c_i = 1\} \|x_i - \mu_1\|^2 \right)}_{\text{sum squared distance of data in cluster \#1}} + \dots + \underbrace{\left(\sum_{i=1}^N \mathbb{1}\{c_i = K\} \|x_i - \mu_K\|^2 \right)}_{\text{sum squared distance of data in cluster \#K}}.$$

For each k , we then optimize. Let $n_k = \sum_{i=1}^n \mathbb{1}\{c_i = k\}$. Then

$$\mu_k = \arg \min_{\mu} \sum_{i=1}^n \mathbb{1}\{c_i = k\} \|x_i - \mu\|^2 \quad \rightarrow \quad \mu_k = \frac{1}{n_k} \sum_{i=1}^n x_i \mathbb{1}\{c_i = k\}.$$

That is, μ_k is the *mean* of the data assigned to cluster k .

K-MEANS CLUSTERING ALGORITHM

Algorithm: K-means clustering

Given: x_1, \dots, x_n where each $x \in \mathbb{R}^d$

Goal: Minimize $\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$.

- ▶ Randomly initialize $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$.
- ▶ Iterate until \boldsymbol{c} and $\boldsymbol{\mu}$ stop changing

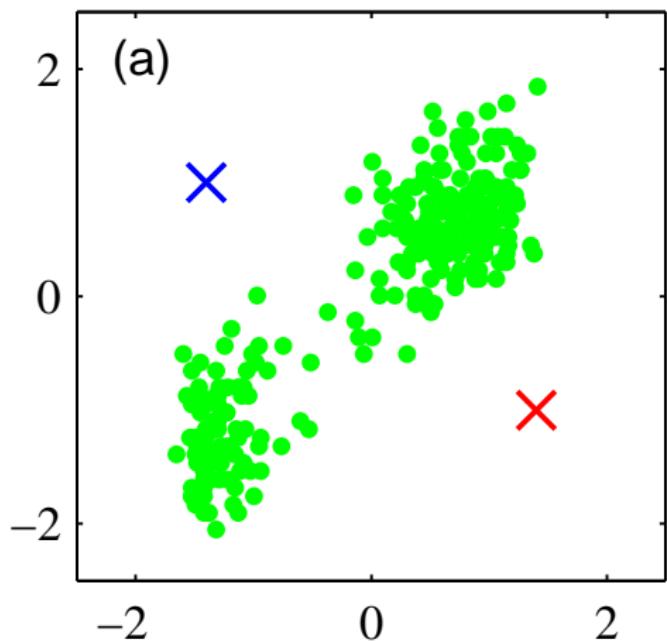
1. Update each c_i :

$$c_i = \arg \min_k \|x_i - \mu_k\|^2$$

2. Update each μ_k : Set

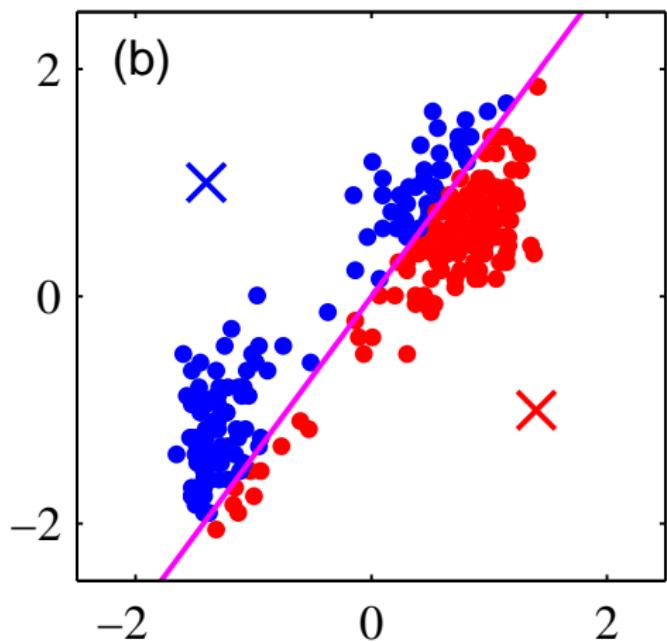
$$n_k = \sum_{i=1}^n \mathbb{1}\{c_i = k\} \quad \text{and} \quad \mu_k = \frac{1}{n_k} \sum_{i=1}^n x_i \mathbb{1}\{c_i = k\}$$

K-MEANS ALGORITHM: EXAMPLE RUN



A random initialization

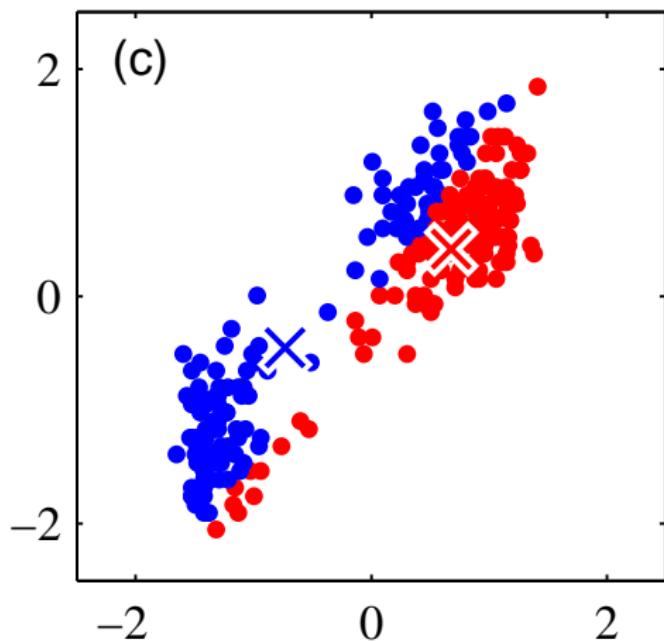
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 1

Assign data to clusters

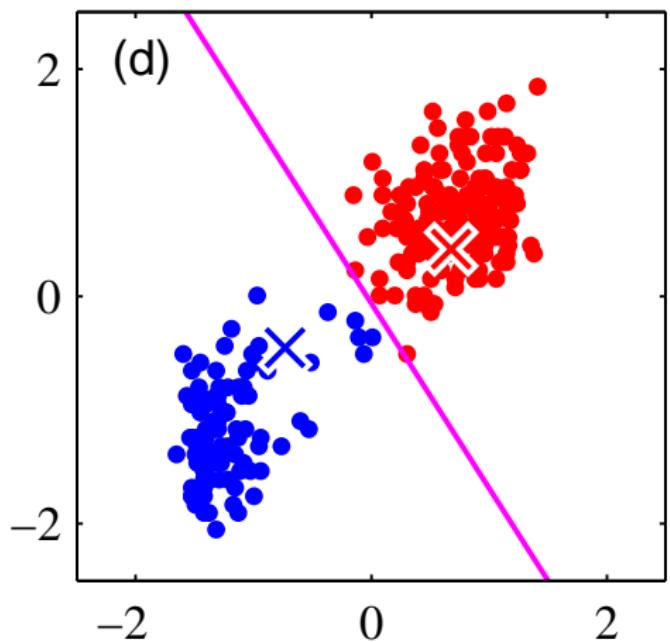
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 1

Update the centroids

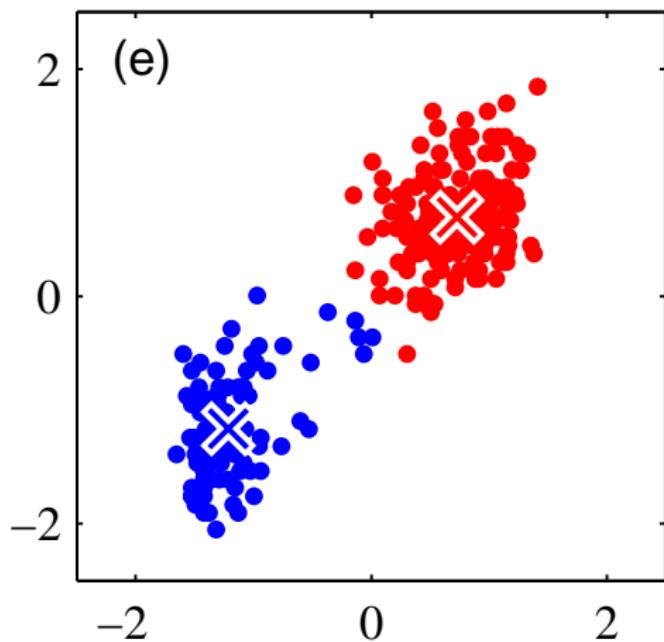
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 2

Assign data to clusters

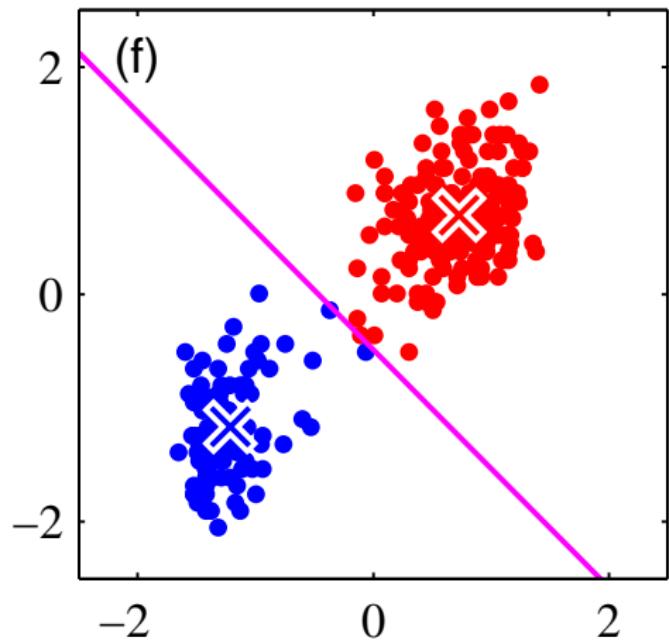
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 2

Update the centroids

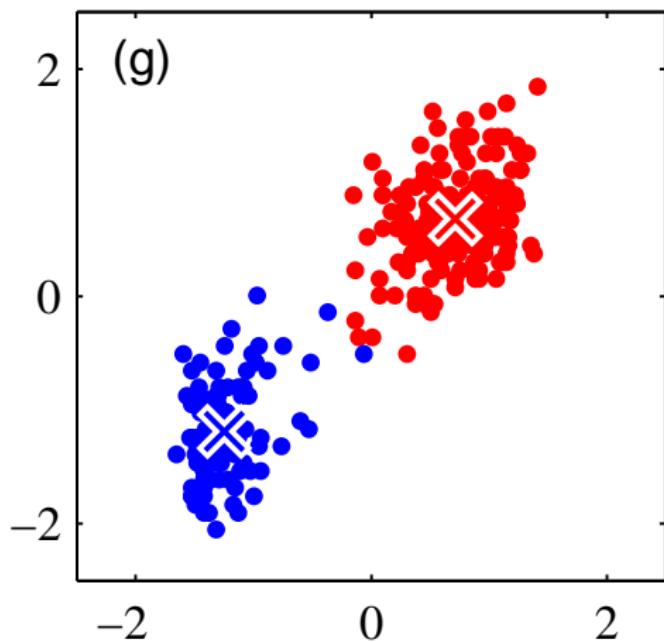
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 3

Assign data to clusters

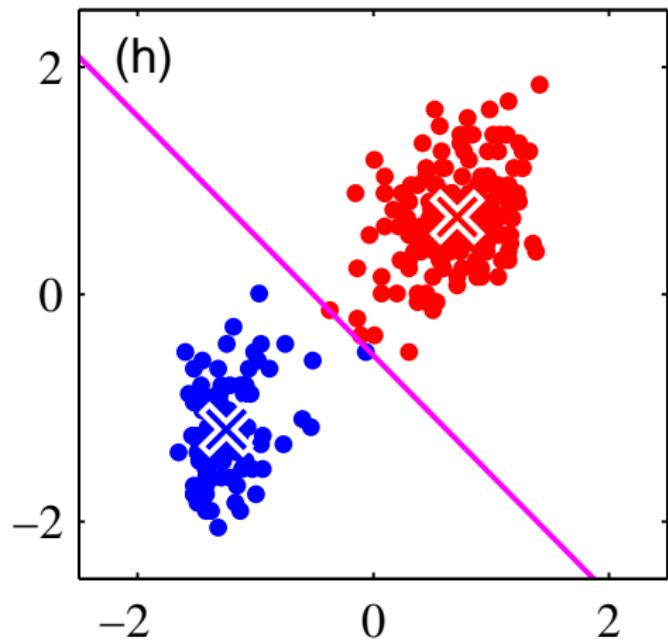
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 3

Update the centroids

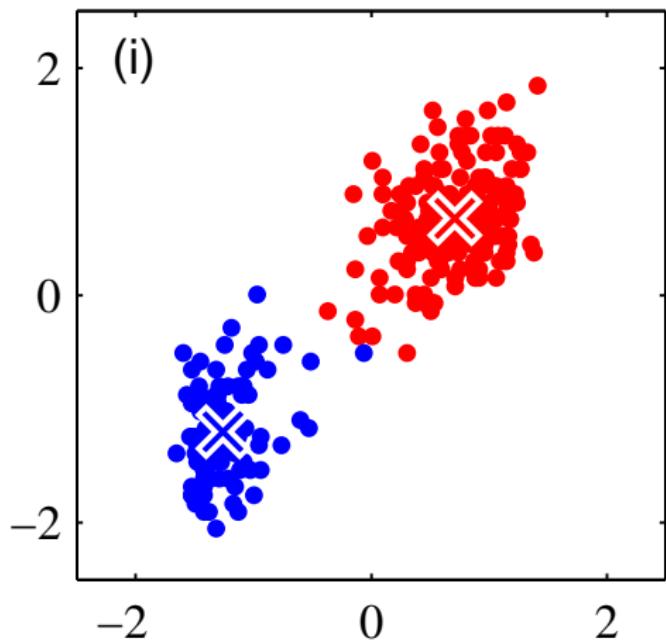
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 4

Assign data to clusters

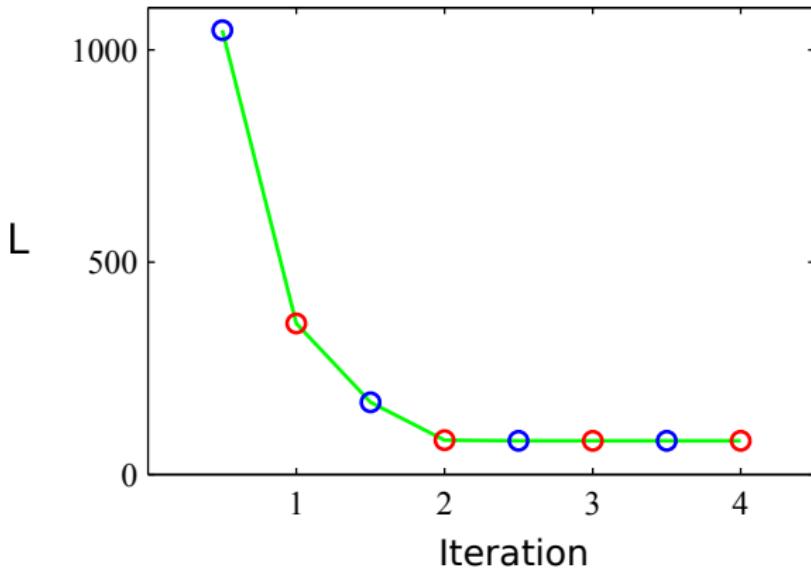
K-MEANS ALGORITHM: EXAMPLE RUN



Iteration 4

Update the centroids

CONVERGENCE OF K-MEANS



Objective function after

- ▶ the “assignment” step (blue: corresponding to c), and
- ▶ the “update” step (red: corresponding to μ).

CONVERGENCE OF K-MEANS

The outline of why this converges is straightforward:

1. Every update to c_i or μ_k decreases \mathcal{L} compared to the previous value.
2. Therefore, \mathcal{L} is *monotonically decreasing*.
3. $\mathcal{L} \geq 0$, so Step 1 converges to some point (but probably not to 0).

When c stops changing, the algorithm has converged to a *local* optimal solution. This is a result of \mathcal{L} not being convex.

Non-convexity means that different initializations will give different results:

- ▶ Often the results will be similar in quality, but no guarantees.
- ▶ In practice, the algorithm can be run multiple times with different initializations. Then use the result with the lowest \mathcal{L} .

SELECTING K

We don't know how many clusters there are, but selecting K is tricky.
The K-means objective function decreases as K increases,

$$\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2.$$

For example, if $K = n$ then let $\mu_k = x_k$ and as a result $\mathcal{L} = 0$.

Methods for choosing K include:

- ▶ Using advanced knowledge. e.g., if you want to split a set of tasks among K people, then you already know K .
- ▶ Looking at the *relative* decrease in \mathcal{L} . If K^* is best, then increasing K when $K \leq K^*$ should decrease \mathcal{L} much more than when $K > K^*$.
- ▶ Often the K-means result is part of a larger application. The main application may start to perform worse even though \mathcal{L} is decreasing.
- ▶ More advanced modeling techniques exist that address this issue.

TWO APPLICATIONS OF K-MEANS

Lossy data compression



Approach: Vectorize 2×2 patches from an image (so data is $x \in \mathbb{R}^4$) and cluster them with K-means. Replace each patch with its assigned centroid.

(left) Original 1024×1024 image requiring 8 bits/pixel (1MB total)

(middle) Approximation using 200 clusters (requires 239KB storage)

(right) Approximation using 4 clusters (requires 62KB storage)

Data preprocessing (side comment)

K-means is also very useful for *discretizing* data as a preprocessing step. This allows us to recast a continuous-valued problem as a discrete one.

EXTENSIONS: K-MEDOIDS

Algorithm: K-medoids clustering

Input: Data x_1, \dots, x_n and distance measure $D(x, \mu)$. Randomly initialize μ .

- ▶ Iterate until c is no longer changing

1. For each c_i : Set

$$c_i = \arg \min_k D(x_i, \mu_k)$$

2. For each μ_k : Set

$$\mu_k = \arg \min_{\mu} \sum_{i:c_i=k} D(x_i, \mu)$$

Comment: Step #2 may require an algorithm.

K-medoids is a straightforward extension of K-means where the distance measure isn't the squared error. That is,

- ▶ K-means uses $D(x, \mu) = \|x - \mu\|^2$.
- ▶ Could set $D(x, \mu) = \|x - \mu\|_1$, which would be more robust to outliers.
- ▶ If $x \notin \mathbb{R}^d$, we could define $D(x, \mu)$ to be more complex.

ColumbiaX: Machine Learning

Lecture 15

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

MAXIMUM LIKELIHOOD

APPROACHES TO DATA MODELING

Our approaches to modeling data thus far have been either probabilistic or non-probabilistic in motivation.

- ▶ Probabilistic models: Probability distributions defined on data, e.g.,
 1. Bayes classifiers
 2. Logistic regression
 3. Least squares and ridge regression (using ML and MAP interpretation)
 4. Bayesian linear regression
- ▶ Non-probabilistic models: No probability distributions involved, e.g.,
 1. Perceptron
 2. Support vector machine
 3. Decision trees
 4. K-means

In *every* case, we have some objective function we are trying to optimize (greedily vs non-greedily, locally vs globally).

MAXIMUM LIKELIHOOD

As we've seen, one *probabilistic* objective function is maximum likelihood.

Setup: In the most basic scenario, we start with

1. some set of model parameters θ
2. a set of data $\{x_1, \dots, x_n\}$
3. a probability distribution $p(x|\theta)$
4. an i.i.d. assumption, $x_i \stackrel{iid}{\sim} p(x|\theta)$

Maximum likelihood seeks to find the θ that maximizes the likelihood

$$\theta_{ML} = \arg \max_{\theta} p(x_1, \dots, x_n | \theta) \stackrel{(a)}{=} \arg \max_{\theta} \prod_{i=1}^n p(x_i | \theta) \stackrel{(b)}{=} \arg \max_{\theta} \sum_{i=1}^n \ln p(x_i | \theta)$$

- (a) follows from i.i.d. assumption.
(b) follows since $f(y) > f(x) \Rightarrow \ln f(y) > \ln f(x)$.

MAXIMUM LIKELIHOOD

We've discussed maximum likelihood for a few models, e.g., least squares linear regression and the Bayes classifier.

Both of these models were “nice” because we could find their respective θ_{ML} analytically by writing an equation and plugging in data to solve.

Gaussian with unknown mean and covariance

In the first lecture, we saw if $x_i \stackrel{iid}{\sim} N(\mu, \Sigma)$, where $\theta = \{\mu, \Sigma\}$, then

$$\nabla_{\theta} \ln \prod_{i=1}^n p(x_i | \theta) = 0$$

gives the following maximum likelihood values for μ and Σ :

$$\mu_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \Sigma_{\text{ML}} = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_{\text{ML}})(x_i - \mu_{\text{ML}})^T$$

COORDINATE ASCENT AND MAXIMUM LIKELIHOOD

In more complicated models, we might split the parameters into groups θ_1, θ_2 and try to maximize the likelihood over both of these,

$$\theta_{1,\text{ML}}, \theta_{2,\text{ML}} = \arg \max_{\theta_1, \theta_2} \sum_{i=1}^n \ln p(x_i | \theta_1, \theta_2),$$

Although we can solve one *given* the other, we can't solve it *simultaneously*.

Coordinate ascent (probabilistic version)

We saw how K-means presented a similar situation, and that we could optimize using coordinate ascent. This technique is generalizable.

Algorithm: For iteration $t = 1, 2, \dots$,

1. Optimize $\theta_1^{(t)} = \arg \max_{\theta_1} \sum_{i=1}^n \ln p(x_i | \theta_1, \theta_2^{(t-1)})$
2. Optimize $\theta_2^{(t)} = \arg \max_{\theta_2} \sum_{i=1}^n \ln p(x_i | \theta_1^{(t)}, \theta_2)$

COORDINATE ASCENT AND MAXIMUM LIKELIHOOD

There is a third (subtly) different situation, where we really want to find

$$\theta_{1,\text{ML}} = \arg \max_{\theta_1} \sum_{i=1}^n \ln p(x_i | \theta_1).$$

Except this function is “tricky” to optimize directly. However, we figure out that we can add a second variable θ_2 such that

$$\sum_{i=1}^n \ln p(x_i, \theta_2 | \theta_1) \quad (\text{Function 2})$$

is easier to work with. We’ll make this clearer later.

- ▶ Notice in this second case that θ_2 is on the *left* side of the conditioning bar. This implies a prior on θ_2 , (whatever “ θ_2 ” turns out to be).
- ▶ We will next discuss a fundamental technique called the EM algorithm for finding $\theta_{1,\text{ML}}$ by using Function 2 instead.

EXPECTATION-MAXIMIZATION ALGORITHM

A MOTIVATING EXAMPLE

Let $x_i \in \mathbb{R}^d$, be a vector with *missing data*. Split this vector into two parts:

1. x_i^o – observed portion (the sub-vector of x_i that is measured)
2. x_i^m – missing portion (the sub-vector of x_i that is still unknown)
3. The missing dimensions can be different for different x_i .

We assume that $x_i \stackrel{iid}{\sim} N(\mu, \Sigma)$, and want to solve

$$\mu_{\text{ML}}, \Sigma_{\text{ML}} = \arg \max_{\mu, \Sigma} \sum_{i=1}^n \ln p(x_i^o | \mu, \Sigma).$$

This is tricky. However, if we knew x_i^m (and therefore x_i), then

$$\mu_{\text{ML}}, \Sigma_{\text{ML}} = \arg \max_{\mu, \Sigma} \sum_{i=1}^n \underbrace{\ln p(x_i^o, x_i^m | \mu, \Sigma)}_{= p(x_i | \mu, \Sigma)}$$

is very easy to optimize (we just did it on a previous slide).

CONNECTING TO A MORE GENERAL SETUP

We will discuss a method for optimizing $\sum_{i=1}^n \ln p(x_i^o | \mu, \Sigma)$ and imputing its missing values $\{x_1^m, \dots, x_n^m\}$. This is a very general technique.

General setup

Imagine we have two parameter sets θ_1, θ_2 , where

$$p(x|\theta_1) = \int p(x, \theta_2|\theta_1) d\theta_2 \quad (\text{marginal distribution})$$

Example: For the previous example we can show that

$$p(x_i^o | \mu, \Sigma) = \int p(x_i^o, x_i^m | \mu, \Sigma) dx_i^m = N(\mu_i^o, \Sigma_i^o),$$

where μ_i^o and Σ_i^o are the sub-vector/sub-matrix of μ and Σ defined by x_i^o .

THE EM OBJECTIVE FUNCTION

We need to define a general *objective function* that gives us what we want:

1. It lets us optimize the marginal $p(x|\theta_1)$ over θ_1 ,
2. It uses $p(x, \theta_2|\theta_1)$ in doing so purely for computational convenience.

The EM objective function

Before picking it apart, we claim that this objective function is

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$$

Some immediate comments:

- ▶ $q(\theta_2)$ is *any* probability distribution (assumed continuous for now)
- ▶ We assume we know $p(\theta_2|x, \theta_1)$. That is, given the data x and fixed values for θ_1 , we can solve the conditional posterior distribution of θ_2 .

DERIVING THE EM OBJECTIVE FUNCTION

Let's show that this equality is actually true

$$\begin{aligned}\ln p(x|\theta_1) &= \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2 \\ &= \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)q(\theta_2)}{p(\theta_2|x, \theta_1)q(\theta_2)} d\theta_2\end{aligned}$$

Remember some rules of probability:

$$p(a, b|c) = p(a|b, c)p(b|c) \Rightarrow p(b|c) = \frac{p(a, b|c)}{p(a|b, c)}.$$

Letting $a = \theta_1$, $b = x$ and $c = \theta_1$, we conclude

$$\begin{aligned}\ln p(x|\theta_1) &= \int q(\theta_2) \ln p(x|\theta_1) d\theta_2 \\ &= \ln p(x|\theta_1)\end{aligned}$$

THE EM OBJECTIVE FUNCTION

The EM objective function splits our desired objective into two terms:

$$\ln p(x|\theta_1) = \underbrace{\int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2}_{\text{A function only of } \theta_1, \text{ we'll call it } \mathcal{L}} + \underbrace{\int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2}_{\text{Kullback-Leibler divergence}}$$

Some more observations about the right hand side:

1. The **KL divergence** is always ≥ 0 and only $= 0$ when $q = p$.
2. We are assuming that the integral in \mathcal{L} can be calculated, leaving a function only of θ_1 (for a particular setting of the distribution q).

BIGGER PICTURE

Q: What does it mean to iteratively optimize $\ln p(x|\theta_1)$ w.r.t. θ_1 ?

A: One way to think about it is that we want a method for generating:

1. A sequence of values for θ_1 such that $\ln p(x|\theta_1^{(t)}) \geq \ln p(x|\theta_1^{(t-1)})$.
2. We want $\theta_1^{(t)}$ to converge to a local maximum of $\ln p(x|\theta_1)$.

It doesn't matter how we generate the sequence $\theta_1^{(1)}, \theta_1^{(2)}, \theta_1^{(3)}, \dots$

We will show how EM generates #1 and just mention that EM satisfies #2.

THE EM ALGORITHM

The EM objective function

$$\ln p(x|\theta_1) = \underbrace{\int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2}_{\text{define this to be } \mathcal{L}(x, \theta_1)} + \underbrace{\int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2}_{\text{Kullback-Leibler divergence}}$$

Definition: The EM algorithm

Given the value $\theta_1^{(t)}$, find the value $\theta_1^{(t+1)}$ as follows:

E-step: Set $q_t(\theta_2) = p(\theta_2|x, \theta_1^{(t)})$ and calculate

$$\mathcal{L}_t(x, \theta_1) = \int q_t(\theta_2) \ln p(x, \theta_2|\theta_1) d\theta_2 - \underbrace{\int q_t(\theta_2) \ln q_t(\theta_2) d\theta_2}_{\text{can ignore this term}}.$$

M-step: Set $\theta_1^{(t+1)} = \arg \max_{\theta_1} \mathcal{L}_t(x, \theta_1)$.

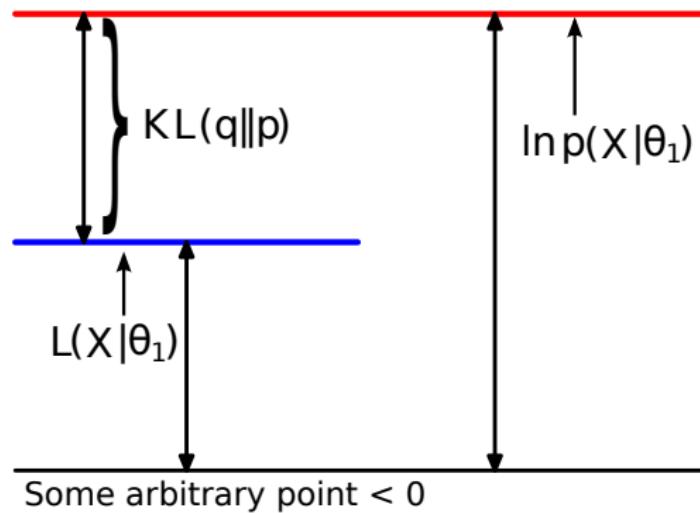
PROOF OF MONOTONIC IMPROVEMENT

Once we're comfortable with the moving parts, the proof that the sequence $\theta_1^{(t)}$ monotonically improves $\ln p(x|\theta_1)$ just requires *analysis*:

$$\begin{aligned}\ln p(x|\theta_1^{(t)}) &= \mathcal{L}(x, \theta_1^{(t)}) + \underbrace{KL\left(q(\theta_2) \| p(\theta_2|x_1, \theta_1^{(t)})\right)}_{= 0 \text{ by setting } q = p} \\ &= \mathcal{L}_t(x, \theta_1^{(t)}) \quad \leftarrow \text{E-step} \\ &\leq \mathcal{L}_t(x, \theta_1^{(t+1)}) \quad \leftarrow \text{M-step} \\ &\leq \mathcal{L}_t(x, \theta_1^{(t+1)}) + \underbrace{KL\left(q_t(\theta_2) \| p(\theta_2|x_1, \theta_1^{(t+1)})\right)}_{> 0 \text{ because } q \neq p} \\ &= \mathcal{L}(x, \theta_1^{(t+1)}) + KL\left(q(\theta_2) \| p(\theta_2|x_1, \theta_1^{(t+1)})\right) \\ &= \ln p(x|\theta_1^{(t+1)})\end{aligned}$$

ONE ITERATION OF EM

Start: Current setting of θ_1 and $q(\theta_2)$



For reference:

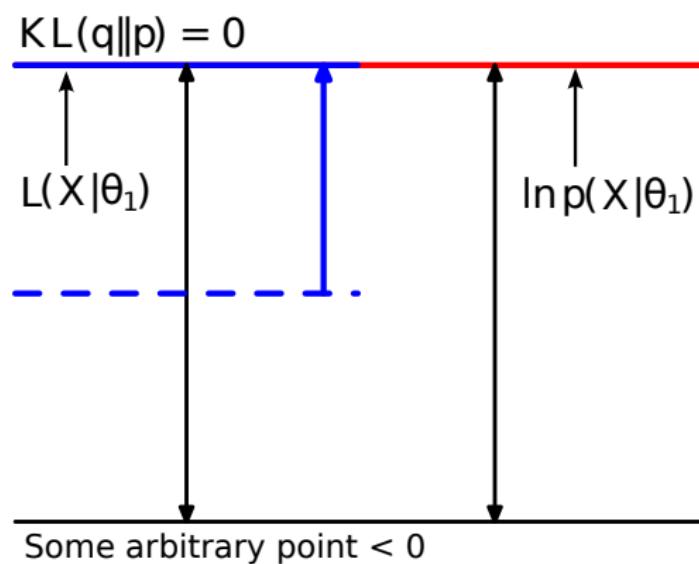
$$\ln p(x|\theta_1) = \mathcal{L} + KL$$

$$\mathcal{L} = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2$$

$$KL = \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$$

ONE ITERATION OF EM

E-step: Set $q(\theta_2) = p(\theta_2|x, \theta_1)$ and update \mathcal{L} .



For reference:

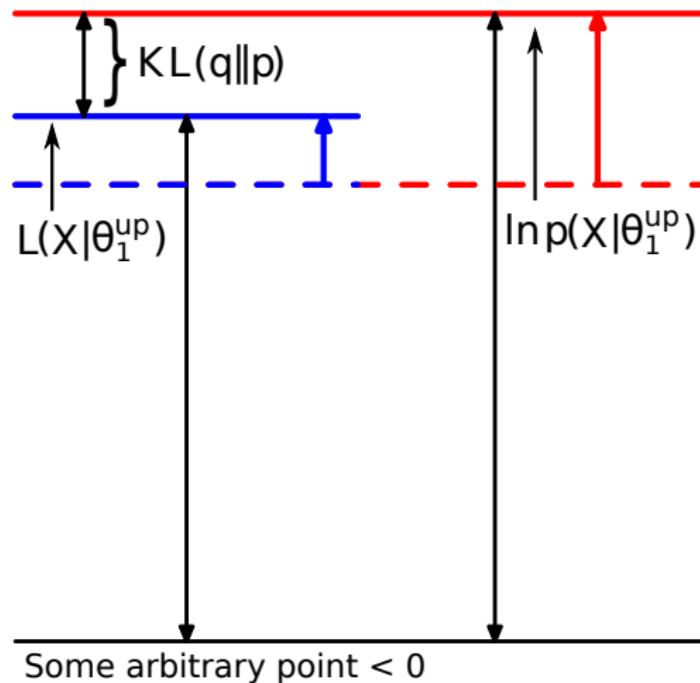
$$\ln p(x|\theta_1) = \mathcal{L} + KL$$

$$\mathcal{L} = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2$$

$$KL = \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$$

ONE ITERATION OF EM

M-step: Maximize \mathcal{L} wrt θ_1 . Now $q \neq p$.



For reference:

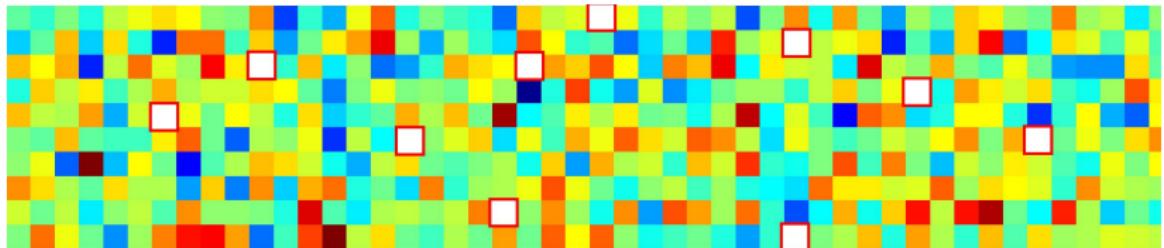
$$\ln p(x|\theta_1) = \mathcal{L} + KL$$

$$\mathcal{L} = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2$$

$$KL = \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$$

EM FOR MISSING DATA

THE PROBLEM



We have a data matrix with missing entries. We model the columns as

$$x_i \stackrel{iid}{\sim} N(\mu, \Sigma).$$

Our goal could be to

1. Learn μ and Σ using maximum likelihood
2. Fill in the missing values “intelligently” (e.g., using a model)
3. Both

We will see how to achieve both of these goals using the EM algorithm.

EM FOR SINGLE GAUSSIAN MODEL WITH MISSING DATA

The original, generic EM objective is

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$$

The EM objective for this specific problem and notation is

$$\begin{aligned} \sum_{i=1}^n \ln p(x_i^o | \mu, \Sigma) &= \sum_{i=1}^n \int q(x_i^m) \ln \frac{p(x_i^o, x_i^m | \mu, \Sigma)}{q(x_i^m)} dx_i^m + \\ &\quad \sum_{i=1}^n \int q(x_i^m) \ln \frac{q(x_i^m)}{p(x_i^m | x_i^o, \mu, \Sigma)} dx_i^m \end{aligned}$$

We can calculate everything required to do this.

E-STEP (PART ONE)

Set $q(x_i^m) = p(x_i^m|x_i^o, \mu, \Sigma)$ using current μ, Σ

Let x_i^o and x_i^m represent the observed and missing dimensions of x_i . For notational convenience, think

$$x_i = \begin{bmatrix} x_i^o \\ x_i^m \end{bmatrix} \sim N \left(\begin{bmatrix} \mu_i^o \\ \mu_i^m \end{bmatrix}, \begin{bmatrix} \Sigma_i^{oo} & \Sigma_i^{om} \\ \Sigma_i^{mo} & \Sigma_i^{mm} \end{bmatrix} \right)$$

Then we can show that $p(x_i^m|x_i^o, \mu, \Sigma) = N(\hat{\mu}_i, \hat{\Sigma}_i)$, where

$$\hat{\mu}_i = \mu_i^m + \Sigma_i^{mo}(\Sigma_i^{oo})^{-1}(x_i^o - \mu_i^o), \quad \hat{\Sigma}_i = \Sigma_i^{mm} - \Sigma_i^{mo}(\Sigma_i^{oo})^{-1}\Sigma_i^{om}.$$

It doesn't look nice, but these are just functions of sub-vectors of μ and sub-matrices of Σ using the relevant dimensions defined by x_i .

E-STEP (PART TWO)

E-step: $\mathbb{E}_{q(x_i^m)}[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$

For each i we will need to calculate the following term,

$$\begin{aligned}\mathbb{E}_q[(x_i - \mu)^T \Sigma^{-1} (x_i - \mu)] &= \mathbb{E}_q[\text{trace}\{\Sigma^{-1}(x_i - \mu)(x_i - \mu)^T\}] \\ &= \text{trace}\{\Sigma^{-1}\mathbb{E}_q[(x_i - \mu)(x_i - \mu)^T]\}\end{aligned}$$

The expectation is calculated using $q(x_i^m) = p(x_i^m | x_i^o, \mu, \Sigma)$. So only the x_i^m portion of x_i will be integrated.

To this end, recall $q(x_i^m) = N(\hat{\mu}_i, \hat{\Sigma}_i)$. We define

1. \hat{x}_i : A vector where we replace the missing values in x_i with $\hat{\mu}_i$.
2. \hat{V}_i : A matrix of 0's, plus sub-matrix $\hat{\Sigma}_i$ in the missing dimensions.

M-STEP

M-step: Maximize $\sum_{i=1}^n \mathbb{E}_q[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$

We'll omit the derivation, but the expectation can now be solved and

$$\mu_{\text{up}}, \Sigma_{\text{up}} = \arg \max_{\mu, \Sigma} \sum_{i=1}^n \mathbb{E}_q[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$$

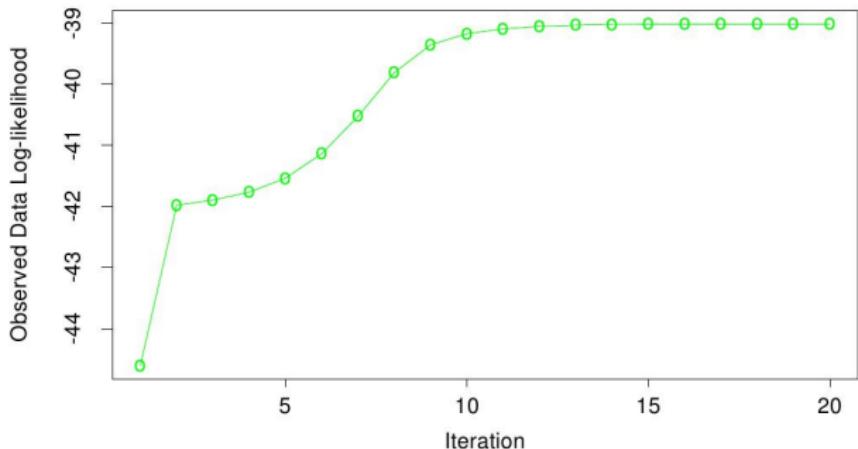
can be found. Recalling the $\hat{\cdot}$ notation,

$$\mu_{\text{up}} = \frac{1}{n} \sum_{i=1}^n \hat{x}_i,$$

$$\Sigma_{\text{up}} = \frac{1}{n} \sum_{i=1}^n \{(\hat{x}_i - \mu_{\text{up}})(\hat{x}_i - \mu_{\text{up}})^T + \hat{V}_i\}$$

Then return to the E-step to calculate the new $p(x_i^m | x_i^o, \mu_{\text{up}}, \Sigma_{\text{up}})$.

IMPLEMENTATION DETAILS



We need to initialize μ and Σ , for example, by setting missing values to zero and calculating μ_{ML} and Σ_{ML} . (We can also use random initialization.)

The EM objective function is then calculated after each update to μ and Σ and will look like the figure above. Stop when the change is “small.”

The output is μ_{ML} , Σ_{ML} and $q(x_i^m)$ for all missing entries.

ColumbiaX: Machine Learning

Lecture 16

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

SOFT CLUSTERING VS HARD CLUSTERING MODELS

HARD CLUSTERING MODELS

Review: K-means clustering algorithm

Given: Data x_1, \dots, x_n , where $x \in \mathbb{R}^d$

Goal: Minimize $\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$.

- ▶ Iterate until values no longer changing

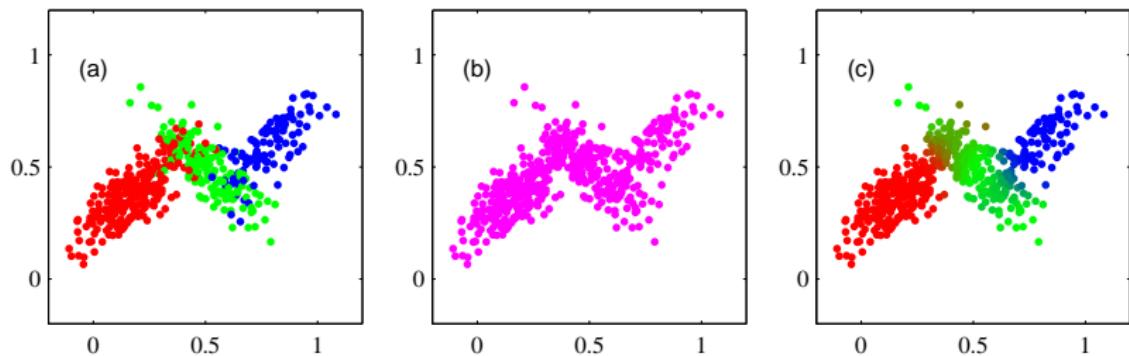
1. Update c : For each i , set $c_i = \arg \min_k \|x_i - \mu_k\|^2$
 2. Update μ : For each k , set $\mu_k = (\sum_i x_i \mathbb{1}\{c_i = k\}) / (\sum_i \mathbb{1}\{c_i = k\})$
-

K-means is an example of a *hard clustering* algorithm because it assigns each observation to only one cluster.

In other words, $c_i = k$ for some $k \in \{1, \dots, K\}$. There is no accounting for the “boundary cases” by hedging on the corresponding c_i .

SOFT CLUSTERING MODELS

A *soft clustering* algorithm breaks the data across clusters intelligently.



- (left) True cluster assignments of data from three Gaussians.
- (middle) The data as we see it.
- (right) A soft-clustering of the data accounting for borderline cases.

WEIGHTED K-MEANS (SOFT CLUSTERING EXAMPLE)

Weighted K-means clustering algorithm

Given: Data x_1, \dots, x_n , where $x \in \mathbb{R}^d$

Goal: Minimize $\mathcal{L} = \sum_{i=1}^n \sum_{k=1}^K \phi_i(k) \frac{\|x_i - \mu_k\|^2}{\beta}$ over ϕ_i and μ_k

Conditions: $\phi_i(k) > 0$ and $\sum_{k=1}^K \phi_i(k) = 1$. Set parameter $\beta > 0$.

► Iterate the following

1. Update ϕ : For each i , update the word allocation weights

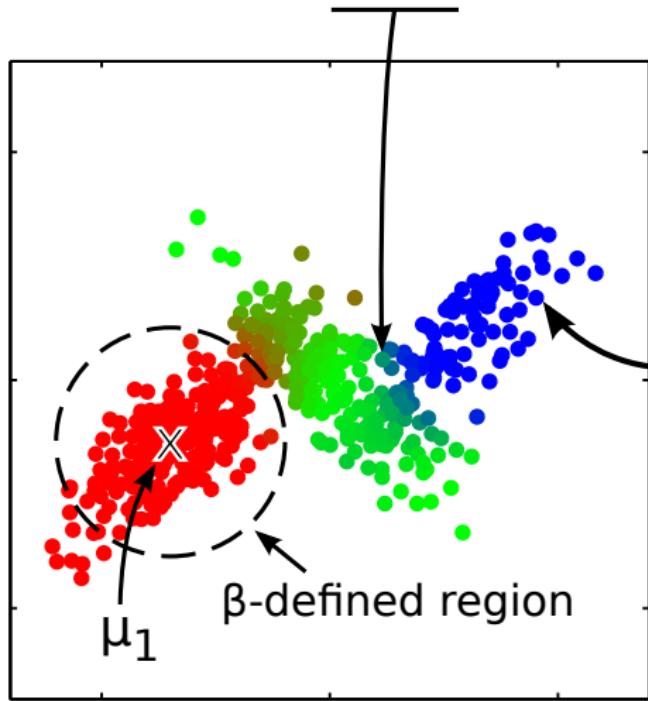
$$\phi_i(k) = \frac{\exp\left\{-\frac{1}{\beta}\|x_i - \mu_k\|^2\right\}}{\sum_j \exp\left\{-\frac{1}{\beta}\|x_i - \mu_j\|^2\right\}}, \text{ for } k = 1, \dots, K$$

2. Update μ : For each k , update μ_k with the *weighted* average

$$\mu_k = \frac{\sum_i x_i \phi_i(k)}{\sum_i \phi_i(k)}$$

SOFT CLUSTERING WITH WEIGHTED K-MEANS

$\phi_i = 0.75$ on green cluster & 0.25 blue cluster



$\phi_i = 1$ on blue cluster

When ϕ_i is binary,
we get back the hard
clustering model

MIXTURE MODELS

PROBABILISTIC SOFT CLUSTERING MODELS

Probabilistic vs non-probabilistic soft clustering

The weight vector ϕ_i is *like* a probability of x_i being assigned to each cluster.

A **mixture model** is a probabilistic model where ϕ_i actually *is* a probability distribution according to the model.

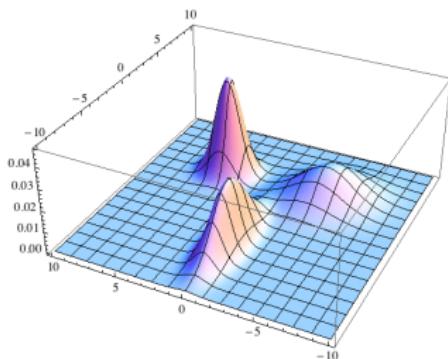
Mixture models work by defining:

- ▶ A prior distribution on the cluster assignment indicator c_i
- ▶ A likelihood distribution on observation x_i given the assignment c_i

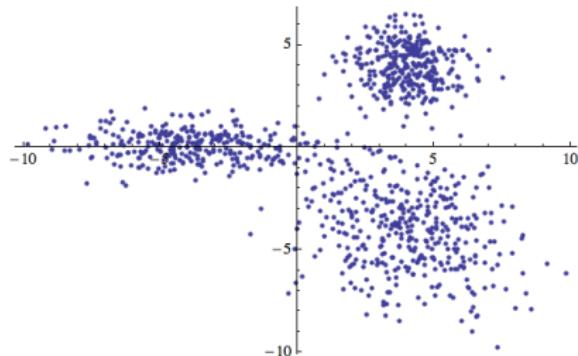
Intuitively we can connect a mixture model to the Bayes classifier:

- ▶ Class prior \rightarrow cluster prior. This time, we *don't* know the “label”
- ▶ Class-conditional likelihood \rightarrow cluster-conditional likelihood

MIXTURE MODELS



(a) A probability distribution on \mathbb{R}^2 .



(b) Data sampled from this distribution.

Before introducing math, some key features of a mixture model are:

1. It is a generative model (defines a probability distribution on the data)
2. It is a weighted combination of simpler distributions.
 - ▶ Each simple distribution is in the same distribution family (i.e., a Gaussian).
 - ▶ The “weighting” is defined by a discrete probability distribution.

MIXTURE MODELS

Generating data from a mixture model

Data: x_1, \dots, x_n , where each $x_i \in \mathcal{X}$ (can be complicated, but think $\mathcal{X} = \mathbb{R}^d$)

Model parameters: A K -dim distribution π and parameters $\theta_1, \dots, \theta_K$.

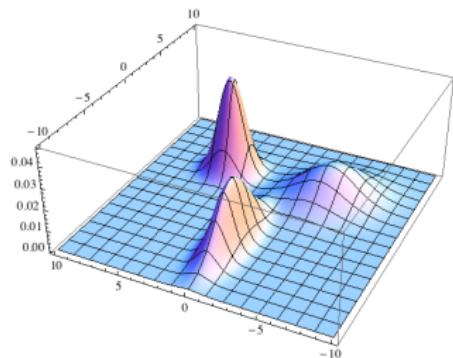
Generative process: For observation number $i = 1, \dots, n$,

1. Generate cluster assignment: $c_i \stackrel{iid}{\sim} \text{Discrete}(\pi) \Rightarrow \text{Prob}(c_i = k|\pi) = \pi_k$.
2. Generate observation: $x_i \sim p(x|\theta_{c_i})$.

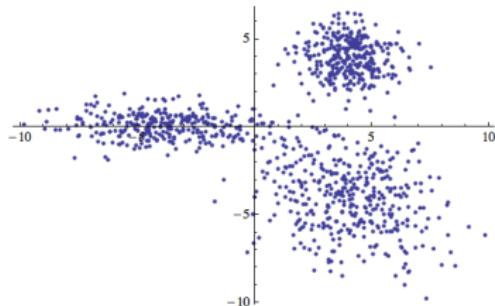
Some observations about this procedure:

- ▶ First, each x_i is randomly assigned to a cluster using distribution π .
- ▶ c_i indexes the cluster assignment for x_i
 - ▶ This picks out the index of the parameter θ used to generate x_i .
 - ▶ If two x 's share a parameter, they are clustered together.

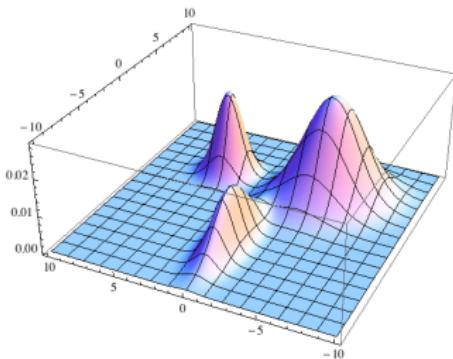
MIXTURE MODELS



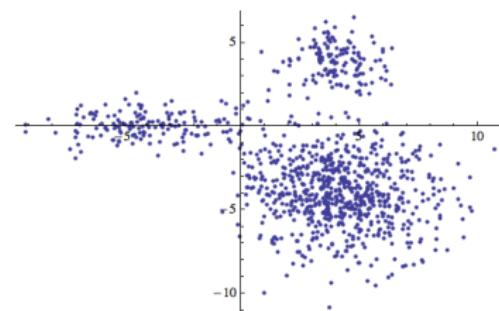
(a) Uniform mixing weights



(b) Data sampled from this distribution.



(c) Uneven mixing weights



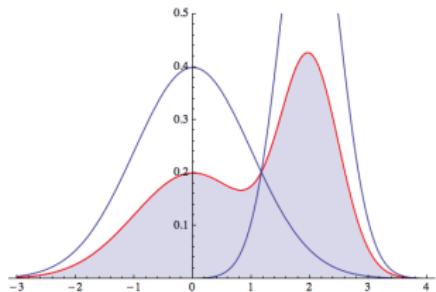
(d) Data sampled from this distribution.

GAUSSIAN MIXTURE MODELS

ILLUSTRATION

Gaussian mixture models are mixture models where $p(x|\theta)$ is Gaussian.

Mixture of two Gaussians



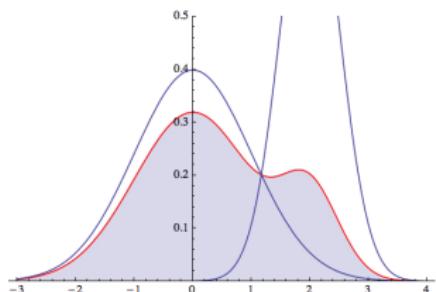
The red line is the density function.

$$\pi = [0.5, 0.5]$$

$$(\mu_1, \sigma_1^2) = (0, 1)$$

$$(\mu_2, \sigma_2^2) = (2, 0.5)$$

Influence of mixing weights



The red line is the density function.

$$\pi = [0.8, 0.2]$$

$$(\mu_1, \sigma_1^2) = (0, 1)$$

$$(\mu_2, \sigma_2^2) = (2, 0.5)$$

GAUSSIAN MIXTURE MODELS (GMM)

The model

Parameters: Let π be a K -dimensional probability distribution and (μ_k, Σ_k) be the mean and covariance of the k th Gaussian in \mathbb{R}^d .

Generate data: For the i th observation,

1. Assign the i th observation to a cluster, $c_i \sim \text{Discrete}(\pi)$
2. Generate the value of the observation, $x_i \sim N(\mu_{c_i}, \Sigma_{c_i})$

Definitions: $\mu = \{\mu_1, \dots, \mu_K\}$ and $\Sigma = \{\Sigma_1, \dots, \Sigma_K\}$.

Goal: We want to learn π, μ and Σ .

GAUSSIAN MIXTURE MODELS (GMM)

Maximum likelihood

Objective: Maximize the likelihood over model parameters π , μ and Σ by treating the c_i as auxiliary data using the EM algorithm.

$$p(x_1, \dots, x_n | \pi, \mu, \Sigma) = \prod_{i=1}^n p(x_i | \pi, \mu, \Sigma) = \prod_{i=1}^n \sum_{k=1}^K p(x_i, c_i = k | \pi, \mu, \Sigma)$$

The summation over values of each c_i “integrates out” this variable.

We can't simply take derivatives with respect to π , μ_k and Σ_k and set to zero to maximize this because there's no closed form solution.

We could use gradient methods, but EM is cleaner.

EM ALGORITHM

Q: Why not instead just include each c_i and maximize $\prod_{i=1}^n p(x_i, c_i | \pi, \mu, \Sigma)$ since (we can show) this is easy to do using coordinate ascent?

A: We would end up with a hard-clustering model where $c_i \in \{1, \dots, K\}$. Our goal here is to have soft clustering, which the sum effectively does.

EM and the GMM

We will not derive everything from scratch. However, we can treat c_1, \dots, c_n as the auxiliary data that we integrate out.

Therefore, we use EM to

$$\text{maximize } \sum_{i=1}^n \ln p(x_i | \pi, \mu, \Sigma) \quad \text{by using } \sum_{i=1}^n \ln p(x_i, c_i | \pi, \mu, \Sigma)$$

Let's look at the outlines of how to derive this.

THE EM ALGORITHM AND THE GMM

From the last lecture, the generic EM objective is

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$$

The EM objective for the Gaussian mixture model is

$$\begin{aligned} \sum_{i=1}^n \ln p(x_i|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \sum_{i=1}^n \sum_{k=1}^K q(c_i=k) \ln \frac{p(x_i, c_i=k|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{q(c_i=k)} + \\ &\quad \sum_{i=1}^n \sum_{k=1}^K q(c_i=k) \ln \frac{q(c_i=k)}{p(c_i|x_i, \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})} \end{aligned}$$

Because c_i is discrete, the integral becomes a sum.

EM SETUP (ONE ITERATION)

First: Set $q(c_i = k) \Leftarrow p(c_i = k|x_i, \pi, \mu, \Sigma)$ using Bayes rule:

$$p(c_i = k|x_i, \pi, \mu, \Sigma) \propto p(c_i = k|\pi)p(x_i|c_i = k, \mu, \Sigma)$$

We can solve the posterior of c_i given π, μ and Σ :

$$q(c_i = k) = \frac{\pi_k N(x_i|\mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i|\mu_j, \Sigma_j)} \implies \phi_i(k)$$

E-step: Take the expectation using the updated q 's

$$Q = \sum_{i=1}^n \sum_{k=1}^K \phi_i(k) \ln p(x_i, c_i = k|\pi, \mu_k, \Sigma_k) + \text{constant w.r.t. } \pi, \mu, \Sigma$$

M-step: Maximize Q with respect to π and each μ_k, Σ_k .

M-STEP CLOSE UP

Aside: How has EM made this easier?

Original objective function:

$$\mathcal{L} = \sum_{i=1}^n \ln \sum_{k=1}^K p(x_i, c_i = k | \pi, \mu_k, \Sigma_k) = \sum_{i=1}^n \ln \sum_{k=1}^K \pi_k N(x_i | \mu_k, \Sigma_k).$$

The log-sum form makes optimizing π , and each μ_k and Σ_k difficult.

Using EM here, we have the M-Step:

$$Q = \sum_{i=1}^n \sum_{k=1}^K \phi_i(k) \underbrace{\{\ln \pi_k + \ln N(x_i | \mu_k, \Sigma_k)\}}_{\ln p(x_i, c_i=k | \pi, \mu_k, \Sigma_k)} + \text{constant w.r.t. } \pi, \mu, \Sigma$$

The sum-log form is easier to optimize. We can take derivatives and solve.

EM FOR THE GMM

Algorithm: Maximum likelihood EM for the GMM

Given: x_1, \dots, x_n where $x \in \mathbb{R}^d$

Goal: Maximize $\mathcal{L} = \sum_{i=1}^n \ln p(x_i | \pi, \mu, \Sigma)$.

- ▶ Iterate until incremental improvement to \mathcal{L} is “small”

1. **E-step:** For $i = 1, \dots, n$, set

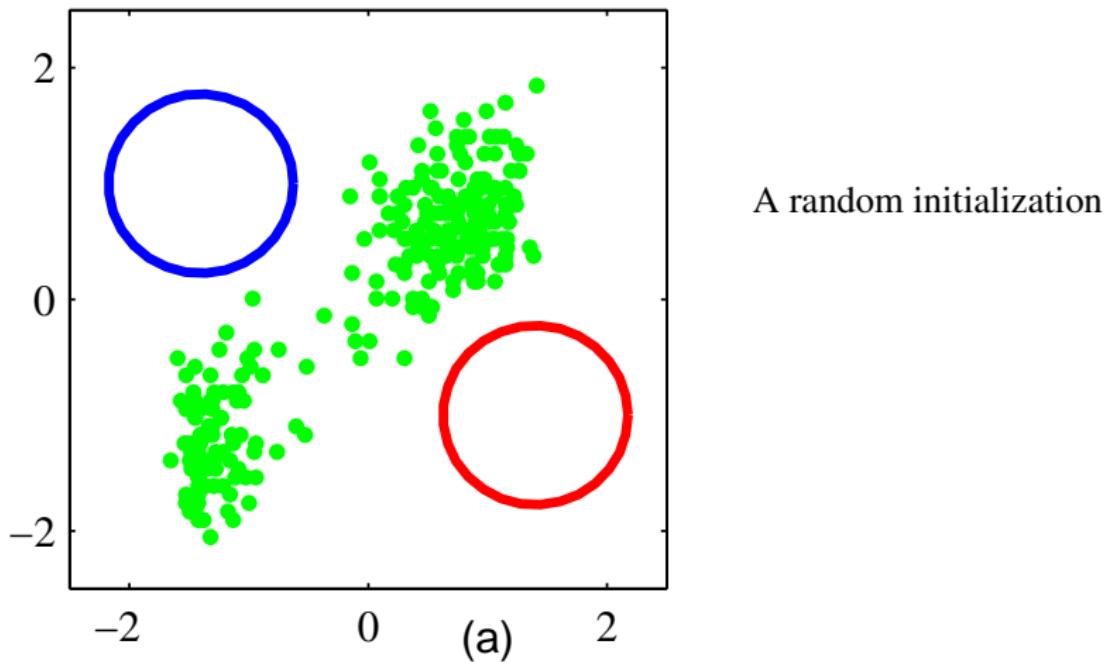
$$\phi_i(k) = \frac{\pi_k N(x_i | \mu_k, \Sigma_k)}{\sum_j \pi_j N(x_i | \mu_j, \Sigma_j)}, \quad \text{for } k = 1, \dots, K$$

2. **M-step:** For $k = 1, \dots, K$, define $n_k = \sum_{i=1}^n \phi_i(k)$ and update the values

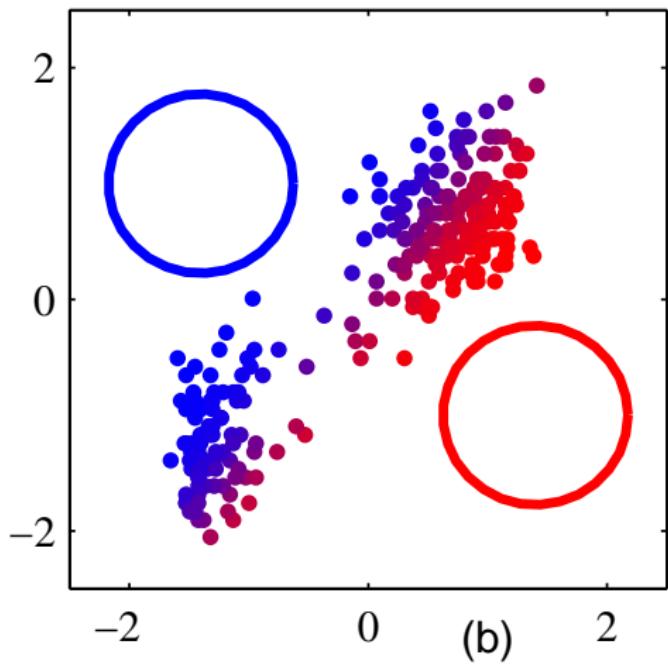
$$\pi_k = \frac{n_k}{n}, \quad \mu_k = \frac{1}{n_k} \sum_{i=1}^n \phi_i(k) x_i \quad \Sigma_k = \frac{1}{n_k} \sum_{i=1}^n \phi_i(k) (x_i - \mu_k)(x_i - \mu_k)^T$$

Comment: The updated value for μ_k is used when updating Σ_k .

GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



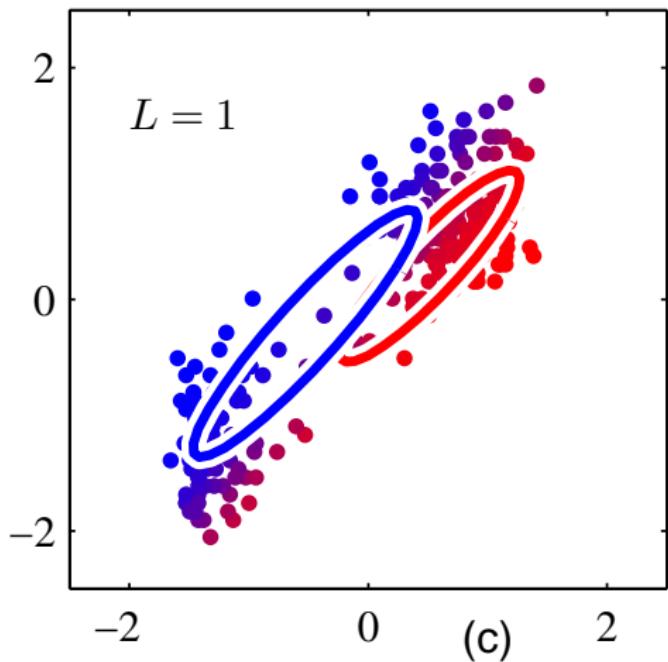
GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



Iteration 1 (E-step)

Assign data to clusters

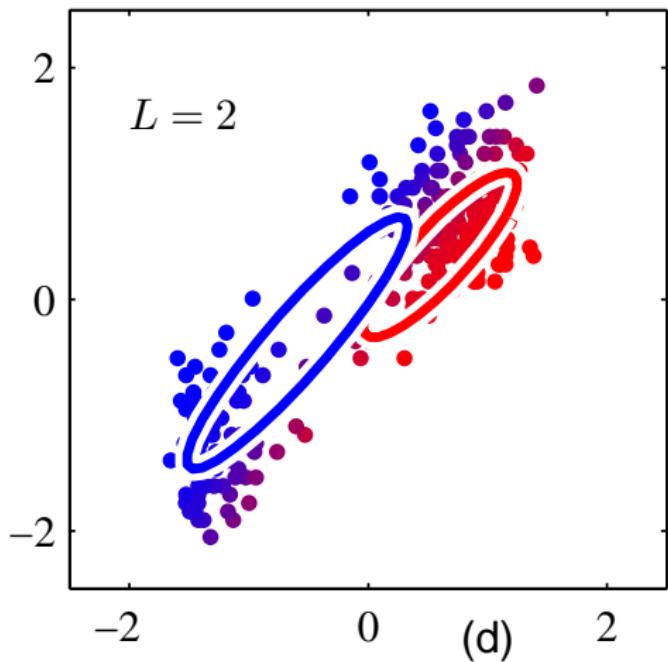
GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



Iteration 1 (M-step)

Update the Gaussians

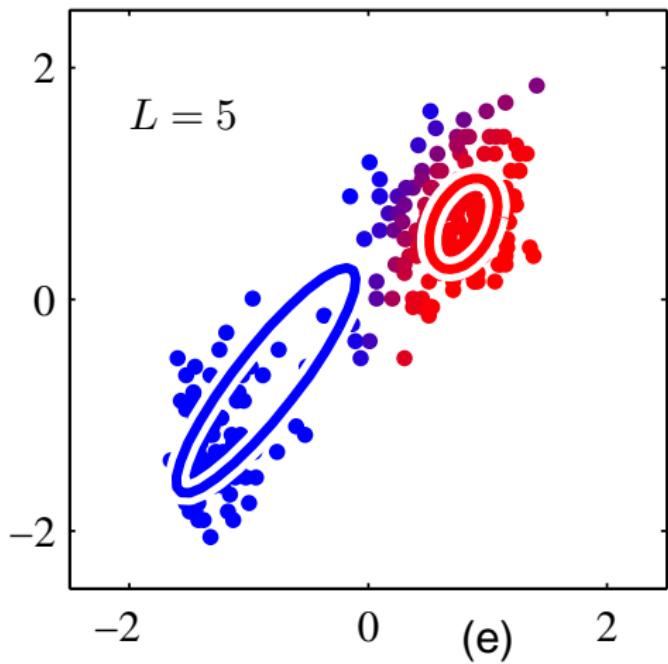
GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



Iteration 2

Assign data to clusters
and update the Gaussians

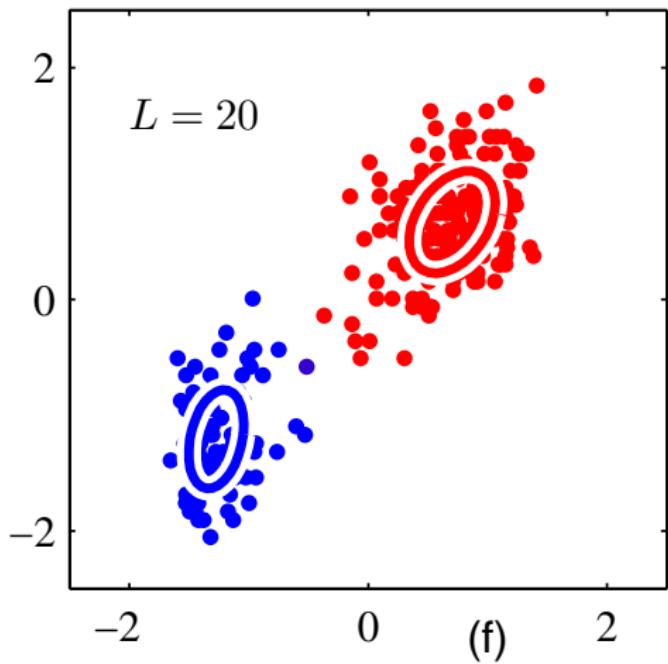
GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



Iteration 5 (skipping ahead)

Assign data to clusters
and update the Gaussians

GAUSSIAN MIXTURE MODEL: EXAMPLE RUN



Iteration 20 (convergence)

Assign data to clusters
and update the Gaussians

GMM AND THE BAYES CLASSIFIER

The GMM feels a lot like a K -class Bayes classifier, where the label of x_i is

$$\text{label}(x_i) = \arg \max_k \pi_k N(x_i | \mu_k, \Sigma_k).$$

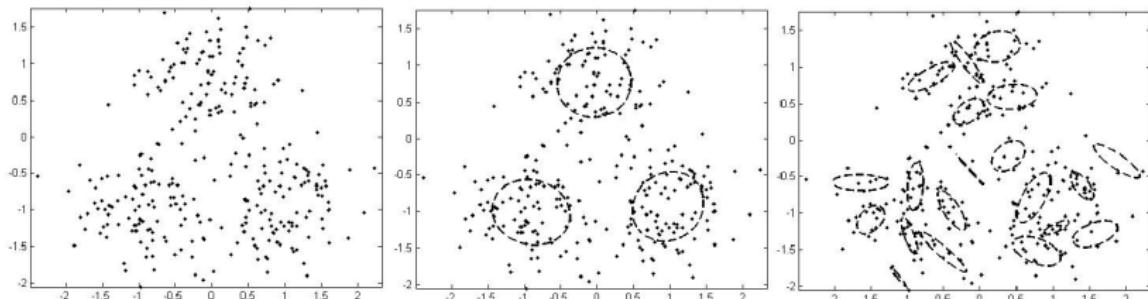
- ▶ π_k = class prior, and $N(\mu_k, \Sigma_k)$ = class-conditional density function.
- ▶ We learned π , μ and Σ using maximum likelihood here too.

For the Bayes classifier, we could find π , μ and Σ with a single equation because the class label was *known*. Compare with the GMM update:

$$\pi_k = \frac{n_k}{n}, \quad \mu_k = \frac{1}{n_k} \sum_{i=1}^n \phi_i(k) x_i \quad \Sigma_k = \frac{1}{n_k} \sum_{i=1}^n \phi_i(k) (x_i - \mu_k)(x_i - \mu_k)^T$$

They're almost identical. But since $\phi_i(k)$ is changing we have to update these values. With the Bayes classifier, " ϕ_i " encodes the label, so it was known.

CHOOSING THE NUMBER OF CLUSTERS



Maximum likelihood for the Gaussian mixture model can overfit the data. It will learn as many Gaussians as it's given.

There are a set of techniques for this based on the Dirichlet distribution.

A Dirichlet prior is used on π which encourages many Gaussians to disappear (i.e., not have any data assigned to them).

EM FOR A GENERIC MIXTURE MODEL

Algorithm: Maximum likelihood EM for mixture models

Given: Data x_1, \dots, x_n where $x \in \mathcal{X}$

Goal: Maximize $\mathcal{L} = \sum_{i=1}^n \ln p(x_i|\pi, \theta)$, where $p(x|\theta_k)$ is problem-specific.

- ▶ Iterate until incremental improvement to \mathcal{L} is “small”

1. **E-step:** For $i = 1, \dots, n$, set

$$\phi_i(k) = \frac{\pi_k p(x_i|\theta_k)}{\sum_j \pi_j p(x_i|\theta_j)}, \quad \text{for } k = 1, \dots, K$$

2. **M-step:** For $k = 1, \dots, K$, define $n_k = \sum_{i=1}^n \phi_i(k)$ and set

$$\pi_k = \frac{n_k}{n}, \quad \theta_k = \arg \max_{\theta} \sum_{i=1}^n \phi_i(k) \ln p(x_i|\theta)$$

Comment: Similar to generalization of the Bayes classifier for any $p(x|\theta_k)$.

ColumbiaX: Machine Learning

Lecture 17

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

COLLABORATIVE FILTERING

OBJECT RECOMMENDATION

Matching consumers to products is an important practical problem.

We can often make these connections using user feedback about subsets of products. To give some prominent examples:

- ▶ Netflix lets users to rate movies
- ▶ Amazon lets users to rate products and write reviews about them
- ▶ Yelp lets users to rate businesses, write reviews, upload pictures
- ▶ YouTube lets users like/dislike a videos and write comments

Recommendation systems use this information to help recommend new things to customers that they may like.

CONTENT FILTERING

One strategy for object recommendation is:

Content filtering: Use known information about the products and users to make recommendations. Create profiles based on

- ▶ Products: movie information, price information, product descriptions
- ▶ Users: demographic information, questionnaire information

Example: A fairly well known example is the online radio Pandora, which uses the “Music Genome Project.”

- ▶ An expert scores a song based on hundreds of characteristics
- ▶ A user also provides information about his/her music preferences
- ▶ Recommendations are made based on pairing these two sources

COLLABORATIVE FILTERING

Content filtering requires a lot of information that can be difficult and expensive to collect. Another strategy for object recommendation is:

Collaborative filtering (CF): Use previous users' input/behavior to make future recommendations. Ignore any *a priori* user or object information.

- ▶ CF uses the ratings of similar users to predict my rating.
- ▶ CF is a domain-free approach. It doesn't need to know what is being rated, just who rated what, and what the rating was.

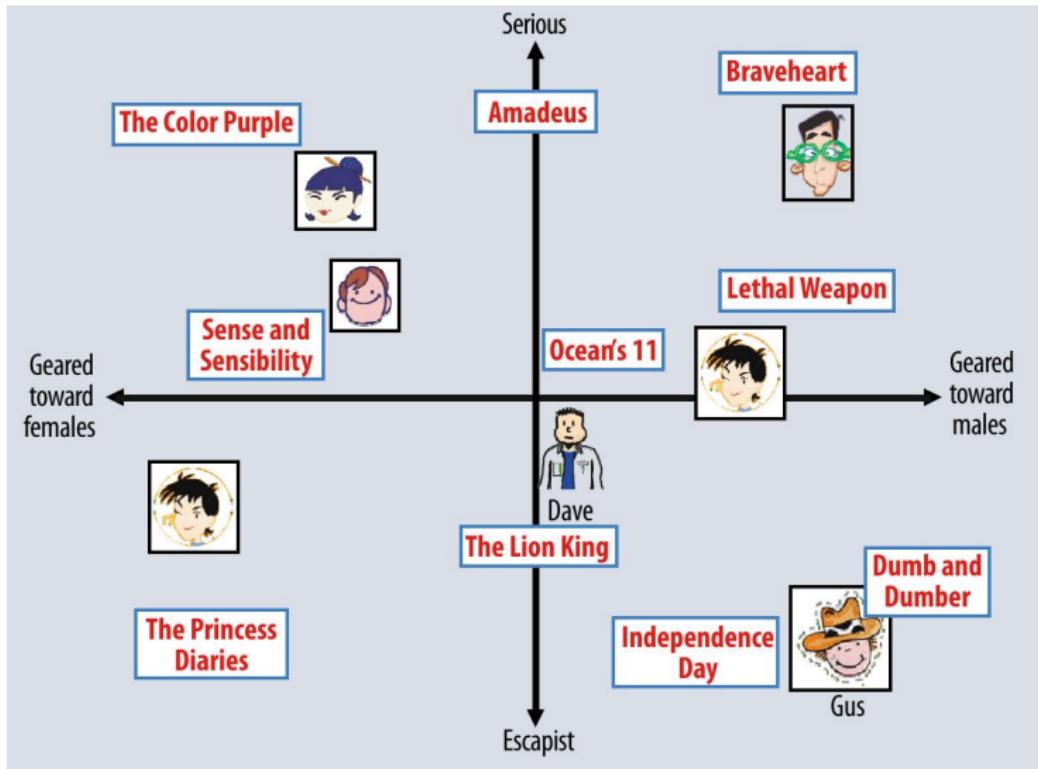
One CF method uses a *neighborhood-based* approach. For example,

1. define a similarity score between me and other users based on how much our overlapping ratings agree, then
2. based on these scores, let others "vote" on what I would like.

These filtering approaches are not mutually exclusive. Content information can be built into a collaborative filtering system to improve performance.

LOCATION-BASED CF METHODS (INTUITION)

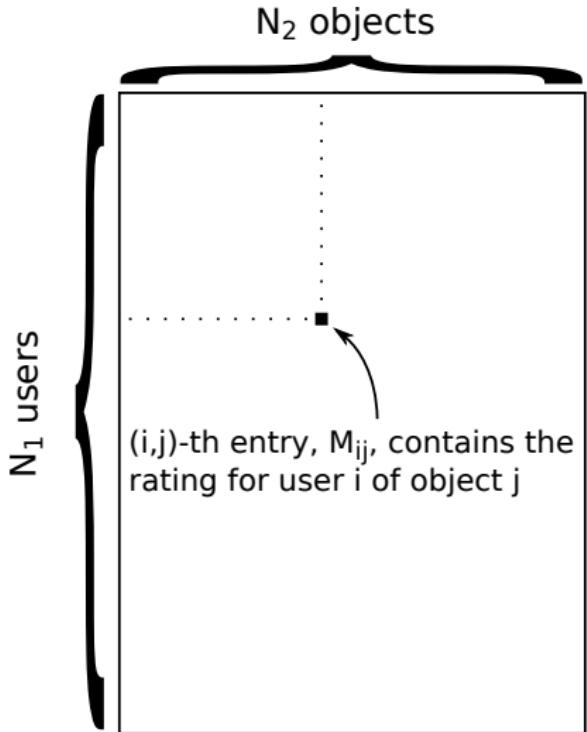
Location-based approaches embed users and objects into points in \mathbb{R}^d .



¹Koren, Y., Robert B., and Volinsky, C.. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

MATRIX FACTORIZATION

MATRIX FACTORIZATION



Matrix factorization (MF) gives a way to learn user and object locations.

First, form the rating matrix M :

- ▶ Contains every user/object pair.
- ▶ Will have many missing values.
- ▶ The goal is to fill in these missing values.

MF and recommendation systems:

- ▶ We have prediction of every missing rating for user i .
- ▶ Recommend the highly rated objects among the predictions.

SINGULAR VALUE DECOMPOSITION

Our goal is to factorize the matrix M . We've discussed one method already.

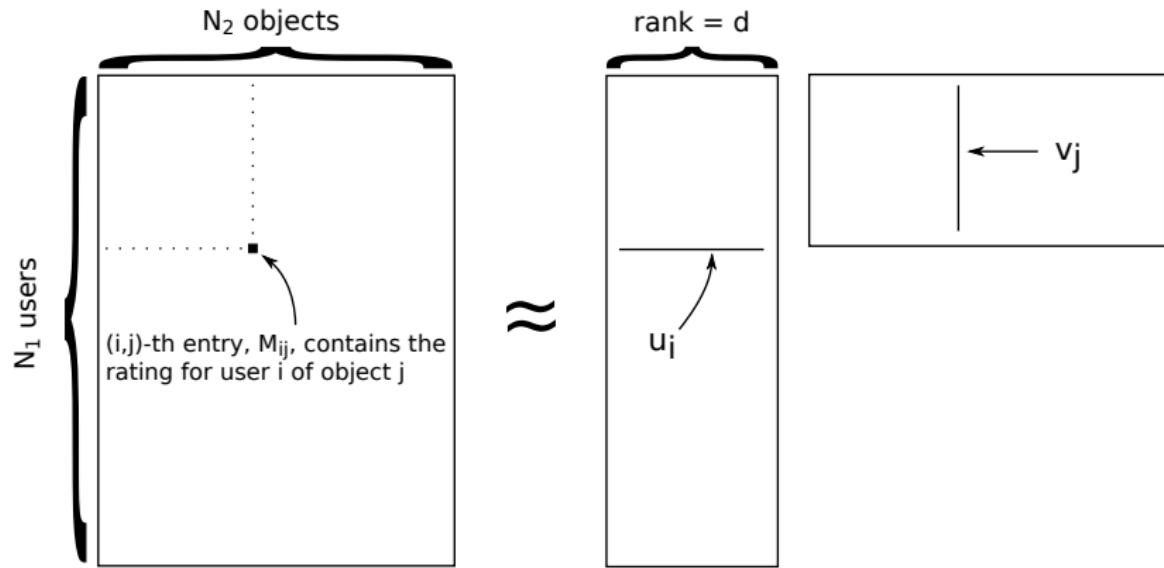
$$\begin{matrix} \text{[Large gray rectangle]} & = & \text{[Tall gray rectangle]} & \text{[Small gray square]} & \text{[Wide gray rectangle]} \\ \mathbf{M} & & \mathbf{U} & \mathbf{S} & \mathbf{V}^\top \\ (n \times d) & & (n \times r) & (r \times r) & (r \times d) \end{matrix}$$

Singular value decomposition: Every matrix M can be written as $M = USV^T$, where $U^T U = I$, $V^T V = I$ and S is diagonal with $S_{ii} \geq 0$.

$r = \text{rank}(M)$. When it's small, M has fewer “degrees of freedom.”

Collaborative filtering with matrix factorization is intuitively similar.

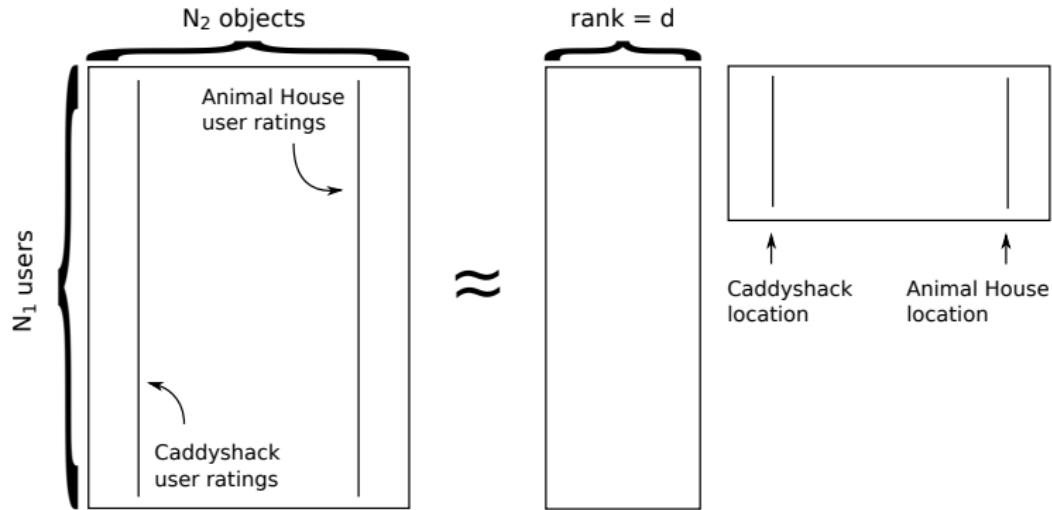
MATRIX FACTORIZATION



We will define a model for learning a low-rank factorization of M . It should:

1. Account for the fact that most values in M are missing
2. Be low-rank, where $d \ll \min\{N_1, N_2\}$ (e.g., $d \approx 10$)
3. Learn a location $u_i \in \mathbb{R}^d$ for user i and $v_j \in \mathbb{R}^d$ for object j

LOW-RANK MATRIX FACTORIZATION

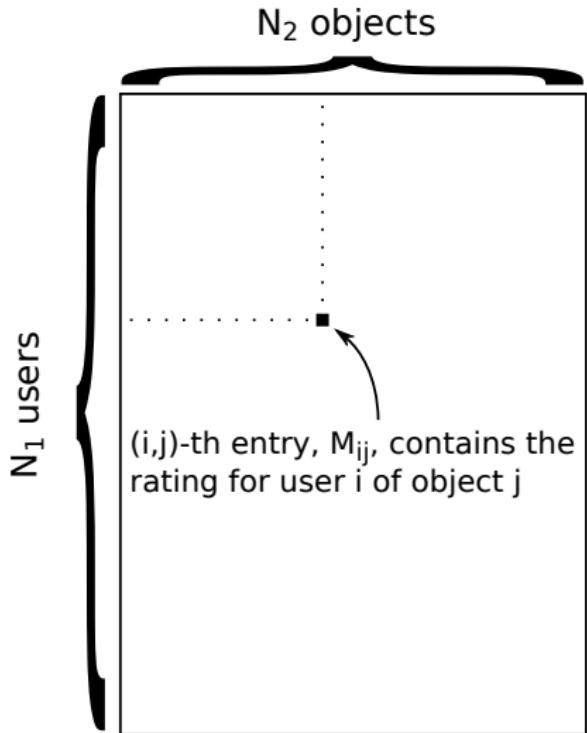


Why learn a low-rank matrix?

- ▶ We think that many columns should look similar. For example, movies like *Caddyshack* and *Animal House* should have **correlated** ratings.
- ▶ Low-rank means that the N_1 -dimensional columns don't "fill up" \mathbb{R}^{N_1} .
- ▶ Since $> 95\%$ of values may be missing, a low-rank restriction gives hope for filling in missing data because it models correlations.

PROBABILISTIC MATRIX FACTORIZATION

SOME NOTATION



- Let the set Ω contain the pairs (i, j) that are observed. In other words,
$$\Omega = \{(i, j) : M_{ij} \text{ is measured}\}.$$
So $(i, j) \in \Omega$ if user i rated object j .
- Let Ω_{u_i} be the index set of objects rated by user i .
- Let Ω_{v_j} be the index set of users who rated object j .

PROBABILISTIC MATRIX FACTORIZATION

Generative model

For N_1 users and N_2 objects, generate

$$\textbf{User locations: } u_i \sim N(0, \lambda^{-1}I), \quad i = 1, \dots, N_1$$

$$\textbf{Object locations: } v_j \sim N(0, \lambda^{-1}I), \quad j = 1, \dots, N_2$$

Given these locations the distribution on the data is

$$M_{ij} \sim N(u_i^T v_j, \sigma^2), \quad \text{for each } (i, j) \in \Omega.$$

Comments:

- ▶ Since M_{ij} is a rating, the Gaussian assumption is clearly wrong.
- ▶ However, the Gaussian is a convenient assumption. The algorithm will be easy to implement, and the model works well.

MODEL INFERENCE

Q: There are many missing values in the matrix M . Do we need some sort of EM algorithm to learn all the u 's and v 's?

- ▶ Let M_o be the part of M that is observed and M_m the missing part. Then

$$p(M_o|U, V) = \int p(M_o, M_m|U, V) dM_m.$$

- ▶ Recall that EM is a **tool** for maximizing $p(M_o|U, V)$ over U and V .
- ▶ Therefore, it is only needed when
 1. $p(M_o|U, V)$ is hard to maximize,
 2. $p(M_o, M_m|U, V)$ is easy to work with, and
 3. the posterior $p(M_m|M_o, U, V)$ is known.

A: If $p(M_o|U, V)$ doesn't present any problems for inference, then no.

(Similar conclusion in our MAP scenario, maximizing $p(M_o, U, V)$.)

MODEL INFERENCE

To test how hard it is to maximize $p(M_o, U, V)$ over U and V , we have to

1. Write out the joint likelihood
2. Take its natural logarithm
3. Take derivatives with respect to u_i and v_j and see if we can solve

The joint likelihood of $p(M_o, U, V)$ can be factorized as follows:

$$p(M_o, U, V) = \underbrace{\left[\prod_{(i,j) \in \Omega} p(M_{ij}|u_i, v_j) \right]}_{\text{conditionally independent likelihood}} \times \underbrace{\left[\prod_{i=1}^{N_1} p(u_i) \right] \left[\prod_{j=1}^{N_2} p(v_j) \right]}_{\text{independent priors}}.$$

By definition of the model, we can write out each of these distributions.

MAXIMUM A POSTERIORI

Log joint likelihood and MAP

The MAP solution for U and V is the maximum of the log joint likelihood

$$U_{\text{MAP}}, V_{\text{MAP}} = \arg \max_{U, V} \sum_{(i,j) \in \Omega} \ln p(M_{ij} | u_i, v_j) + \sum_{i=1}^{N_1} \ln p(u_i) + \sum_{j=1}^{N_2} \ln p(v_j)$$

Calling the MAP objective function \mathcal{L} , we want to maximize

$$\mathcal{L} = - \sum_{(i,j) \in \Omega} \frac{1}{2\sigma^2} \|M_{ij} - u_i^T v_j\|^2 - \sum_{i=1}^{N_1} \frac{\lambda}{2} \|u_i\|^2 - \sum_{j=1}^{N_2} \frac{\lambda}{2} \|v_j\|^2 + \text{constant}$$

The squared terms appear because all distributions are Gaussian.

MAXIMUM A POSTERIORI

To update each u_i and v_j , we take the derivative of \mathcal{L} and set to zero.

$$\nabla_{u_i} \mathcal{L} = \sum_{j \in \Omega_{u_i}} \frac{1}{\sigma^2} (M_{ij} - u_i^T v_j) v_j - \lambda u_i = 0$$

$$\nabla_{v_j} \mathcal{L} = \sum_{i \in \Omega_{v_j}} \frac{1}{\sigma^2} (M_{ij} - v_j^T u_i) u_i - \lambda v_i = 0$$

We can solve for each u_i and v_j individually (therefore EM isn't required),

$$u_i = \left(\lambda \sigma^2 I + \sum_{j \in \Omega_{u_i}} v_j v_j^T \right)^{-1} \left(\sum_{j \in \Omega_{u_i}} M_{ij} v_j \right)$$

$$v_j = \left(\lambda \sigma^2 I + \sum_{i \in \Omega_{v_j}} u_i u_i^T \right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i \right)$$

However, we can't solve for all u_i and v_j at once to find the MAP solution. Thus, as with K-means and the GMM, we use a coordinate ascent algorithm.

PROBABILISTIC MATRIX FACTORIZATION

MAP inference coordinate ascent algorithm

Input: An incomplete ratings matrix M , as indexed by the set Ω . Rank d .

Output: N_1 user locations, $u_i \in \mathbb{R}^d$, and N_2 object locations, $v_j \in \mathbb{R}^d$.

Initialize each v_j . For example, generate $v_j \sim N(0, \lambda^{-1}I)$.

for each iteration **do**

▶ **for** $i = 1, \dots, N_1$ **update user location**

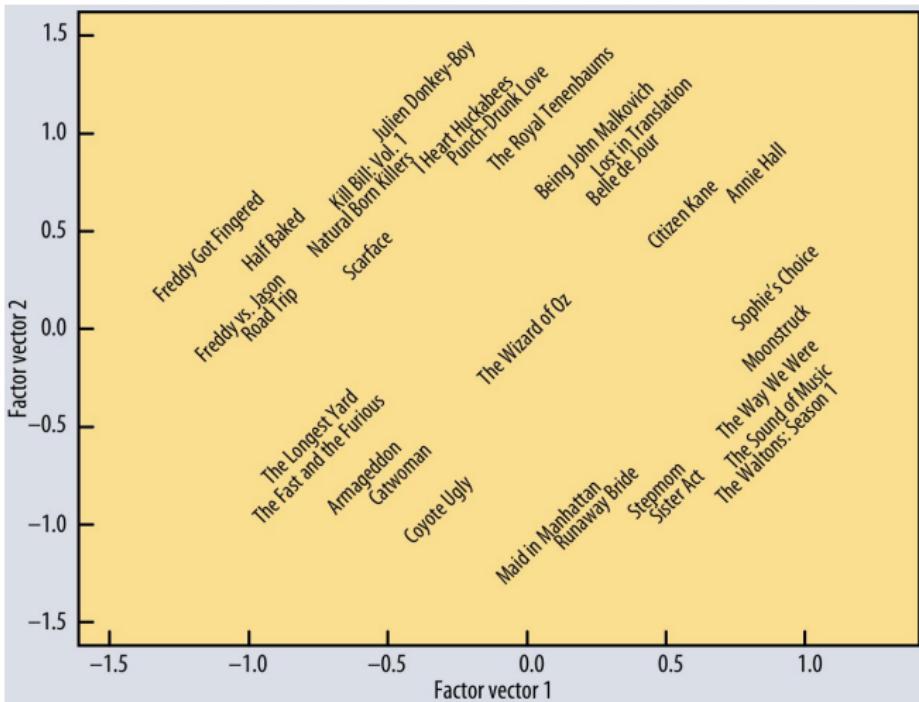
$$u_i = \left(\lambda\sigma^2 I + \sum_{j \in \Omega_{u_i}} v_j v_j^T \right)^{-1} \left(\sum_{j \in \Omega_{u_i}} M_{ij} v_j \right)$$

▶ **for** $j = 1, \dots, N_2$ **update object location**

$$v_j = \left(\lambda\sigma^2 I + \sum_{i \in \Omega_{v_j}} u_i u_i^T \right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i \right)$$

Predict that user i rates object j as $u_i^T v_j$ rounded to closest rating option

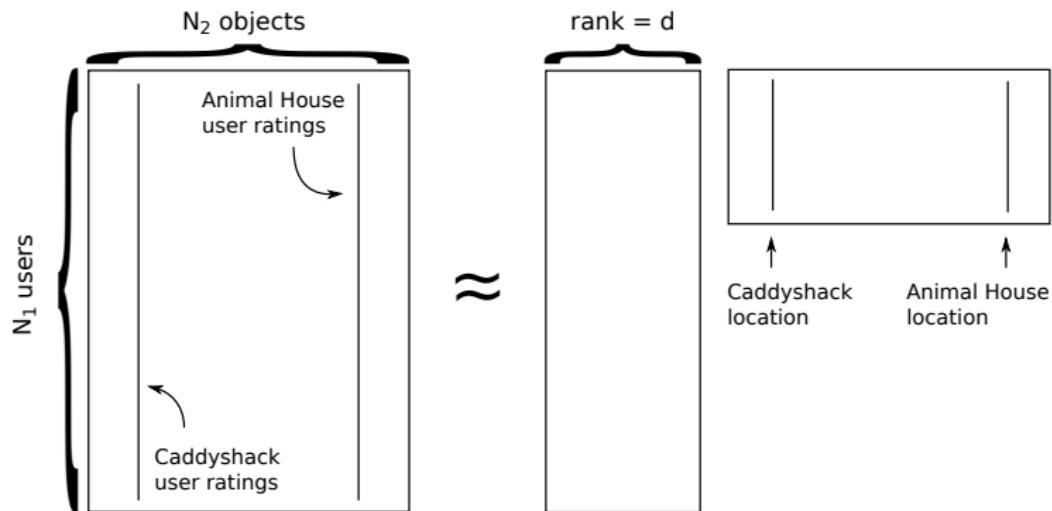
ALGORITHM OUTPUT FOR MOVIES



Hard to show in \mathbb{R}^2 , but we get locations for movies and users. Their relative locations captures relationships (that can be hard to explicitly decipher).

¹Koren, Y., Robert B., and Volinsky, C. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

ALGORITHM OUTPUT FOR MOVIES

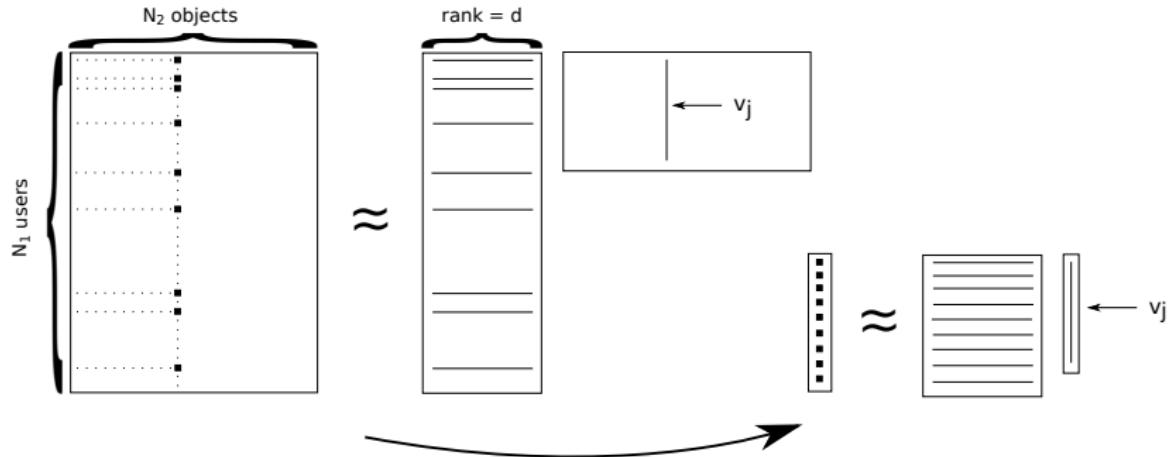


Returning to *Animal House* (j) and *Caddyshack* (j'), it's easy to understand the relationship between their locations v_j and $v_{j'}$:

- ▶ For these two movies to have similar rating patterns, their respective v 's must be similar (i.e., close to each other in \mathbb{R}^d).
- ▶ The same holds for users who have similar tastes across movies.

MATRIX FACTORIZATION AND RIDGE REGRESSION

MATRIX FACTORIZATION AND RIDGE REGRESSION



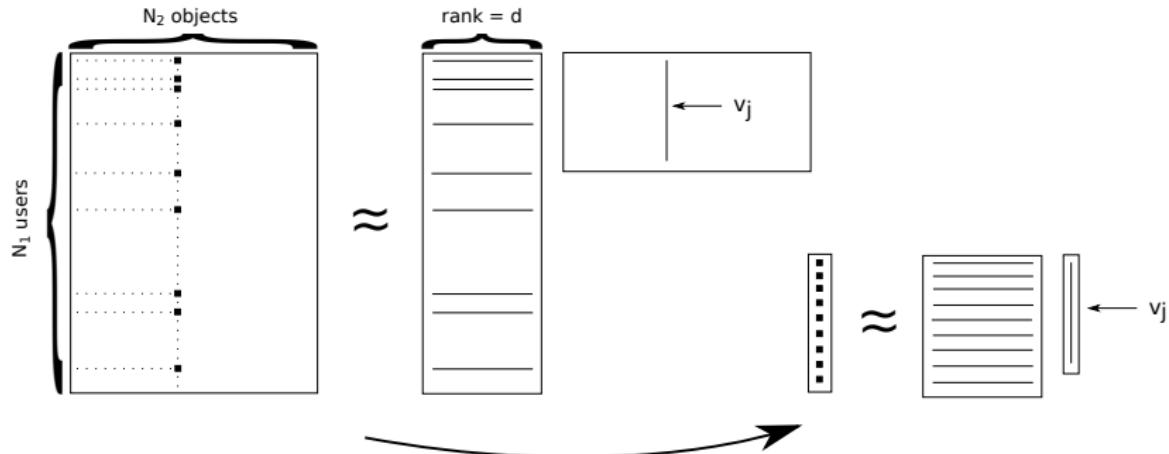
There is a close relationship between this algorithm and ridge regression.

- ▶ Think from the perspective of object location v_j .
- ▶ Minimize the sum squared error $\frac{1}{\sigma^2} (M_{ij} - u_i^T v_j)^2$ with penalty $\lambda \|v_j\|^2$.
- ▶ This is ridge regression for v_j , as the update also shows:

$$v_j = \left(\lambda \sigma^2 I + \sum_{i \in \Omega_{v_j}} u_i u_i^T \right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i \right)$$

- ▶ So this model is a set of $N_1 + N_2$ coupled ridge regression problems.

MATRIX FACTORIZATION AND LEAST SQUARES



We can also connect it to least squares.

- ▶ Remove the Gaussian priors on u_i and v_j . The update for, e.g., v_j is then

$$v_j = \left(\sum_{i \in \Omega_{v_j}} u_i u_i^T \right)^{-1} \left(\sum_{i \in \Omega_{v_j}} M_{ij} u_i \right)$$

- ▶ This is the least squares solution. It requires that every user has rated at least d objects and every object is rated by at least d users.
- ▶ This probably isn't the case, so we see why a prior is *necessary* here.

ColumbiaX: Machine Learning

Lecture 18

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

TOPIC MODELING

MODELS FOR TEXT DATA



Given text data we want to:

- ▶ Organize
- ▶ Visualize
- ▶ Summarize
- ▶ Search
- ▶ Predict
- ▶ Understand

Topic models allow us to

1. Discover themes in text
2. Annotate documents
3. Organize, summarize, etc.

TOPIC MODELING

The New York Times | <http://nyti.ms/WO91dx>

BUSINESS DAY

A Digital Shift on Health Data Swells Profits in an Industry

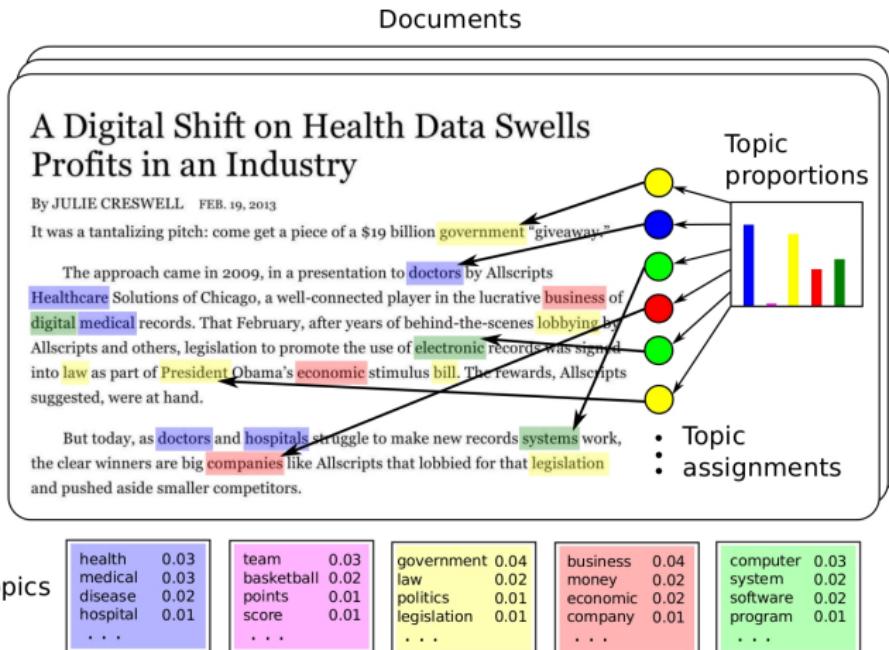
By JULIE CRESWELL FEB. 19, 2013

It was a tantalizing pitch: come get a piece of a \$19 billion government “giveaway.”

The approach came in 2009, in a presentation to doctors by Allscripts Healthcare Solutions of Chicago, a well-connected player in the lucrative business of digital medical records. That February, after years of behind-the-scenes lobbying by Allscripts and others, legislation to promote the use of electronic records was signed into law as part of President Obama’s economic stimulus bill. The rewards, Allscripts suggested, were at hand.

But today, as doctors and hospitals struggle to make new records systems work, the clear winners are big companies like Allscripts that lobbied for that legislation and pushed aside smaller competitors.

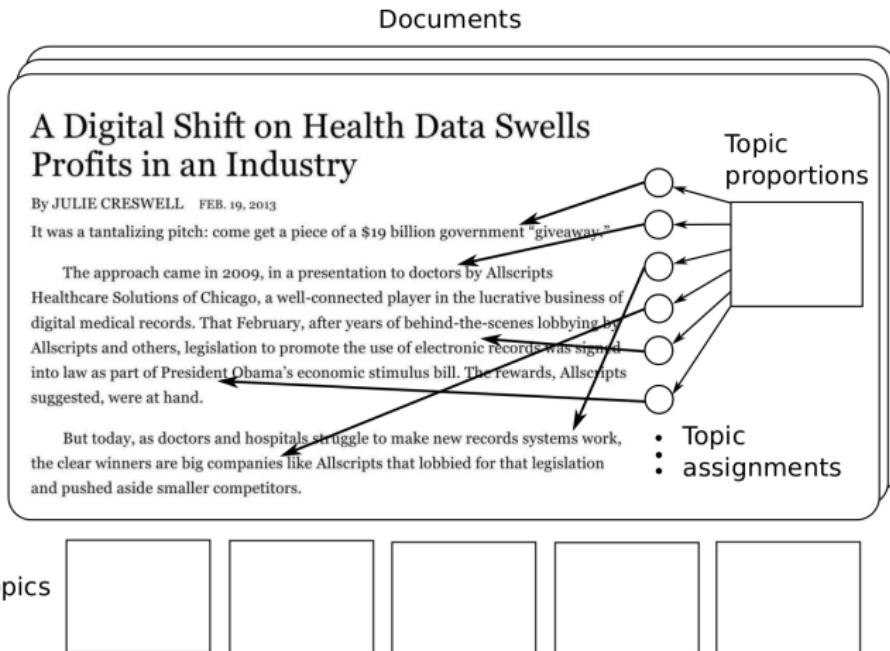
TOPIC MODELING



A probabilistic topic model

- ▶ Learns distributions on words called “topics” shared by documents
- ▶ Learns a distribution on topics for each document
- ▶ Assigns every word in a document to a topic

TOPIC MODELING



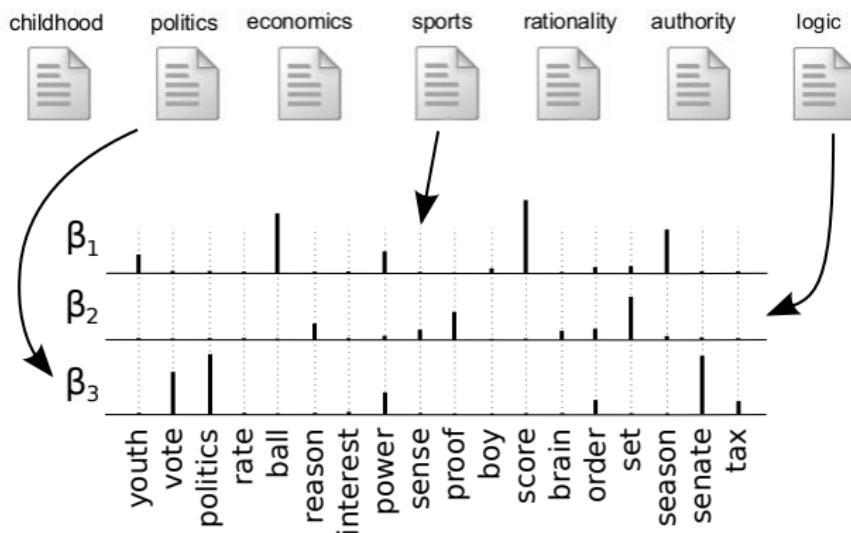
However, none of these things are known in advance and must be learned

- ▶ Each document is treated as a “bag of words”
- ▶ Need to define (1) a model, and (2) an algorithm to learn it
- ▶ We will review the standard topic model, but won’t cover inference

LATENT DIRICHLET ALLOCATION

There are two essential ingredients to latent Dirichlet allocation (LDA).

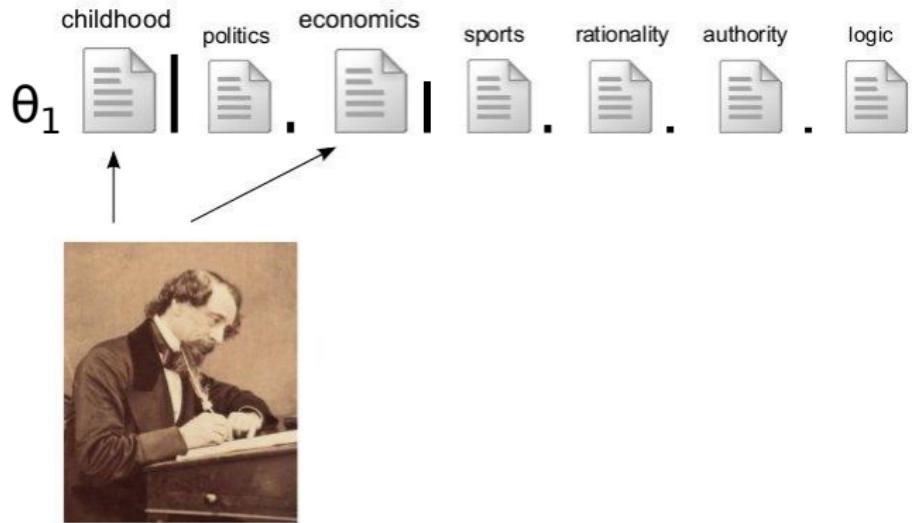
1. A collection of distributions on words (topics).
2. A distribution on topics for each document.



LATENT DIRICHLET ALLOCATION

There are two essential ingredients to latent Dirichlet allocation (LDA).

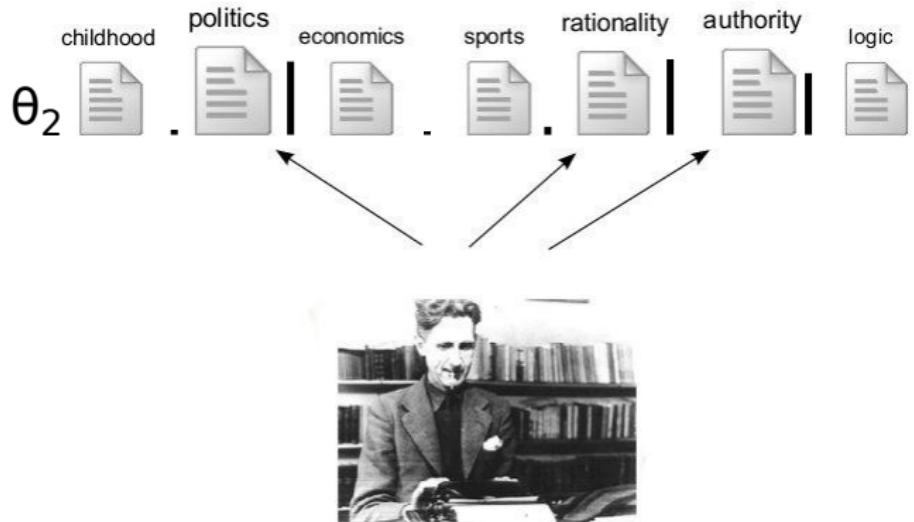
1. A collection of distributions on words (topics).
2. A distribution on topics for each document.



LATENT DIRICHLET ALLOCATION

There are two essential ingredients to latent Dirichlet allocation (LDA).

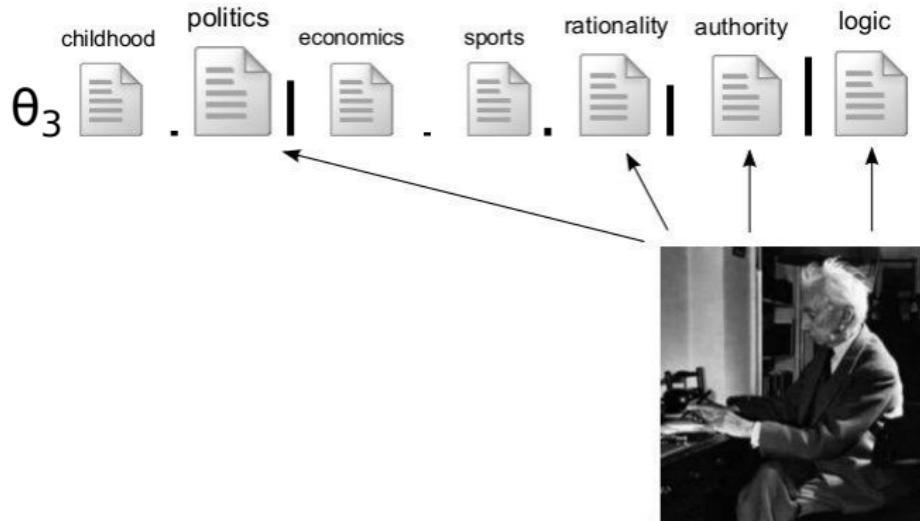
1. A collection of distributions on words (topics).
2. A distribution on topics for each document.



LATENT DIRICHLET ALLOCATION

There are two essential ingredients to latent Dirichlet allocation (LDA).

1. A collection of distributions on words (topics).
2. A distribution on topics for each document.



LATENT DIRICHLET ALLOCATION

There are two essential ingredients to latent Dirichlet allocation (LDA).

1. A collection of distributions on words (topics).
2. A distribution on topics for each document.

The generative process for LDA is:

1. Generate each topic, which is a distribution on words

$$\beta_k \sim \text{Dirichlet}(\gamma), \quad k = 1, \dots, K$$

2. For each document, generate a distribution on topics

$$\theta_d \sim \text{Dirichlet}(\alpha), \quad d = 1, \dots, D$$

3. For the n th word in the d th document,

- a) Allocate the word to a topic, $c_{dn} \sim \text{Discrete}(\theta_d)$
- b) Generate the word from the selected topic, $x_{dn} \sim \text{Discrete}(\beta_{c_{dn}})$

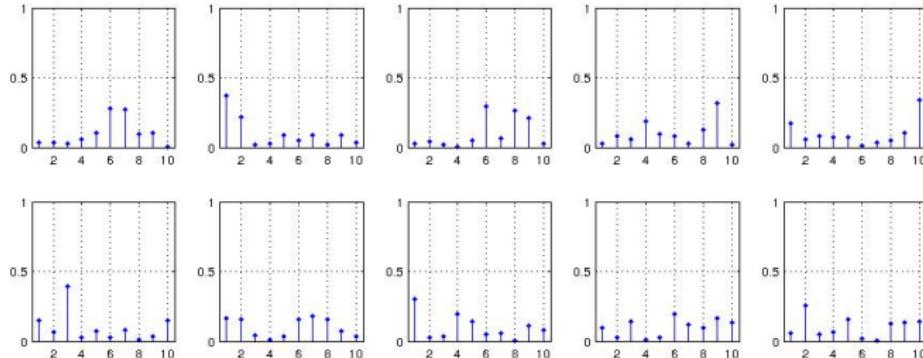
DIRICHLET DISTRIBUTION

A continuous distribution on discrete probability vectors. Let β_k be a probability vector and γ a positive parameter vector,

$$p(\beta_k | \gamma) = \frac{\Gamma(\sum_v \gamma_v)}{\prod_{v=1}^V \Gamma(\gamma_v)} \prod_{v=1}^V \beta_{k,v}^{\gamma_v - 1}$$

This defines the Dirichlet distribution. Some examples of β_k generated from this distribution for a constant value of γ and $V = 10$ are given below.

$$\gamma = 1$$



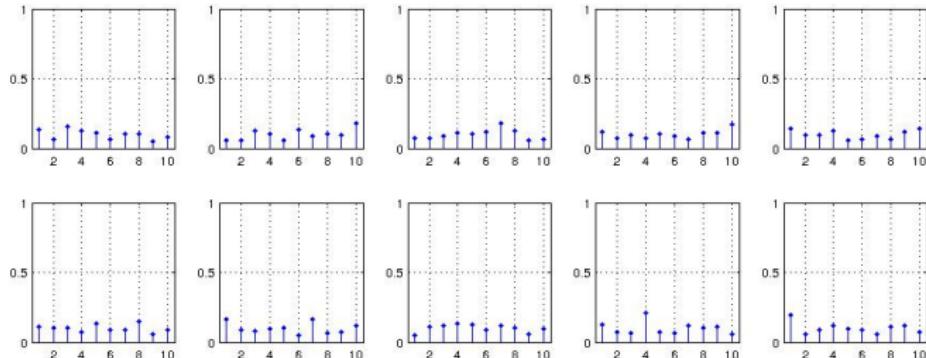
DIRICHLET DISTRIBUTION

A continuous distribution on discrete probability vectors. Let β_k be a probability vector and γ a positive parameter vector,

$$p(\beta_k | \gamma) = \frac{\Gamma(\sum_v \gamma_v)}{\prod_{v=1}^V \Gamma(\gamma_v)} \prod_{v=1}^V \beta_{k,v}^{\gamma_v - 1}$$

This defines the Dirichlet distribution. Some examples of β_k generated from this distribution for a constant value of γ and $V = 10$ are given below.

$\gamma = 10$

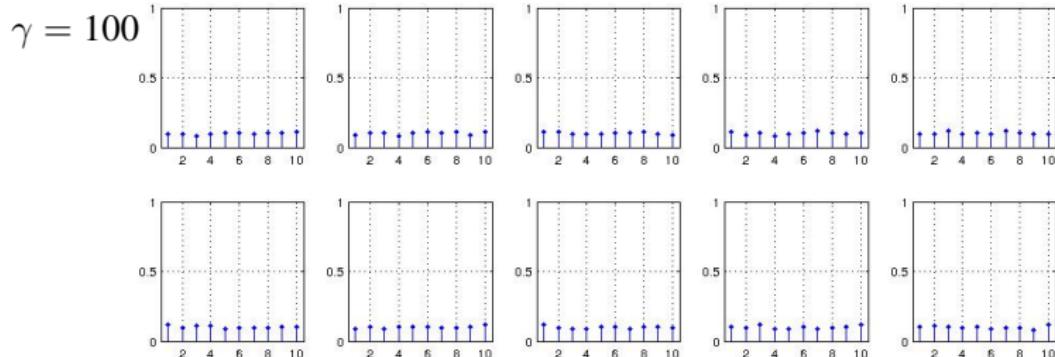


DIRICHLET DISTRIBUTION

A continuous distribution on discrete probability vectors. Let β_k be a probability vector and γ a positive parameter vector,

$$p(\beta_k | \gamma) = \frac{\Gamma(\sum_v \gamma_v)}{\prod_{v=1}^V \Gamma(\gamma_v)} \prod_{v=1}^V \beta_{k,v}^{\gamma_v - 1}$$

This defines the Dirichlet distribution. Some examples of β_k generated from this distribution for a constant value of γ and $V = 10$ are given below.



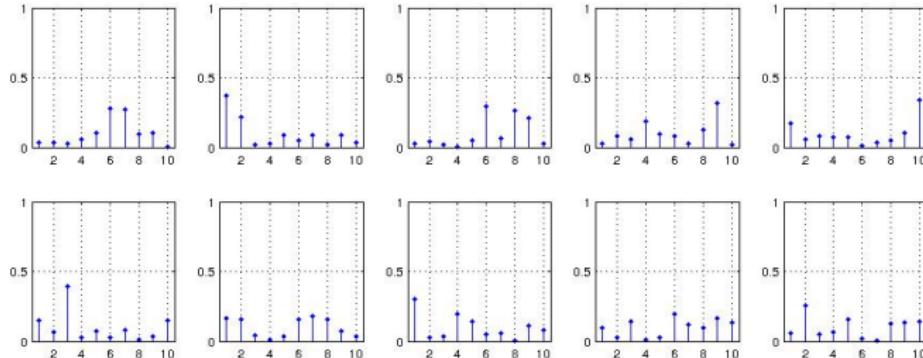
DIRICHLET DISTRIBUTION

A continuous distribution on discrete probability vectors. Let β_k be a probability vector and γ a positive parameter vector,

$$p(\beta_k | \gamma) = \frac{\Gamma(\sum_v \gamma_v)}{\prod_{v=1}^V \Gamma(\gamma_v)} \prod_{v=1}^V \beta_{k,v}^{\gamma_v - 1}$$

This defines the Dirichlet distribution. Some examples of β_k generated from this distribution for a constant value of γ and $V = 10$ are given below.

$$\gamma = 1$$



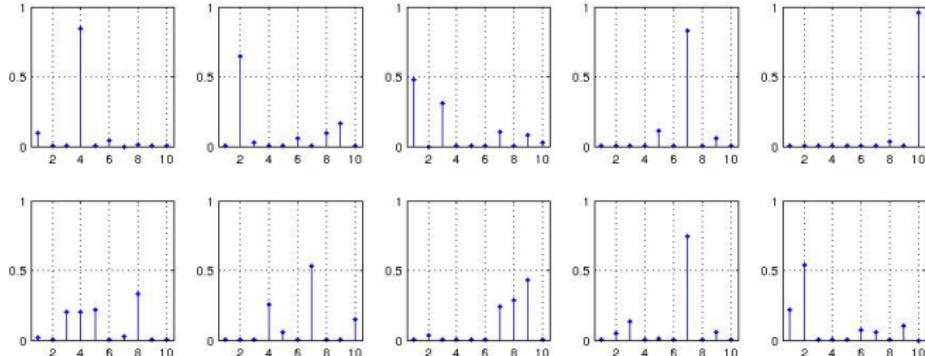
DIRICHLET DISTRIBUTION

A continuous distribution on discrete probability vectors. Let β_k be a probability vector and γ a positive parameter vector,

$$p(\beta_k | \gamma) = \frac{\Gamma(\sum_v \gamma_v)}{\prod_{v=1}^V \Gamma(\gamma_v)} \prod_{v=1}^V \beta_{k,v}^{\gamma_v - 1}$$

This defines the Dirichlet distribution. Some examples of β_k generated from this distribution for a constant value of γ and $V = 10$ are given below.

$\gamma = 0.1$

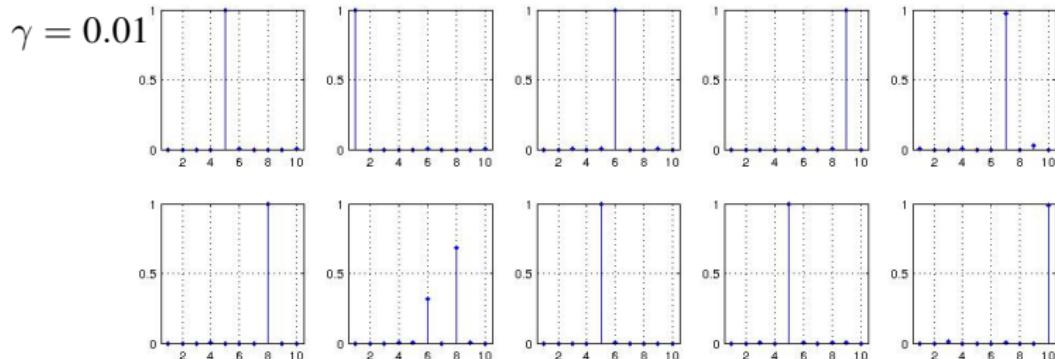


DIRICHLET DISTRIBUTION

A continuous distribution on discrete probability vectors. Let β_k be a probability vector and γ a positive parameter vector,

$$p(\beta_k | \gamma) = \frac{\Gamma(\sum_v \gamma_v)}{\prod_{v=1}^V \Gamma(\gamma_v)} \prod_{v=1}^V \beta_{k,v}^{\gamma_v - 1}$$

This defines the Dirichlet distribution. Some examples of β_k generated from this distribution for a constant value of γ and $V = 10$ are given below.



LDA OUTPUT

The New York Times

music band songs rock album jazz pop song singer night	book life novel story books man stories love children family	art museum show exhibition artist artists paintings painting century works	game Knicks nets points team season play games night coach	show film television movie series says life man character know
theater play production show stage street broadway director musical directed	clinton bush campaign gore political republican doe presidential senator house	stock market percent fund investors funds companies stocks investment trading	restaurant sauce menu food dishes street dining dinner chicken served	budget tax governor county mayor billion taxes plan legislature fiscal

LDA outputs two main things:

1. A set of distributions on words (topics). Shown above are ten topics from NYT data. We list the ten words with the highest probability.
2. A distribution on topics for each document (not shown). This indicates its thematic breakdown and provides a compact representation.

LDA AND MATRIX FACTORIZATION

Q: For a particular document, what is $P(x_{dn} = i|\beta, \theta_d)$?

A: Find this by integrating out the cluster assignment,

$$\begin{aligned} P(x_{dn} = i|\beta, \theta) &= \sum_{k=1}^K P(x_{dn} = i, c_{dn} = k|\beta, \theta_d) \\ &= \sum_{k=1}^K \underbrace{P(x_{dn} = i, | \beta, c_{dn} = k)}_{= \beta_{ki}} \underbrace{P(c_{dn} = k | \theta_d)}_{= \theta_{dk}} \end{aligned}$$

Let $B = [\beta_1, \dots, \beta_K]$ and $\Theta = [\theta_1, \dots, \theta_D]$, then $P(x_{dn} = i|\beta, \theta) = (B\Theta)_{id}$

In other words, we can read the probabilities from a matrix formed by taking the product of two matrices that have nonnegative entries.

NONNEGATIVE MATRIX FACTORIZATION

NONNEGATIVE MATRIX FACTORIZATION

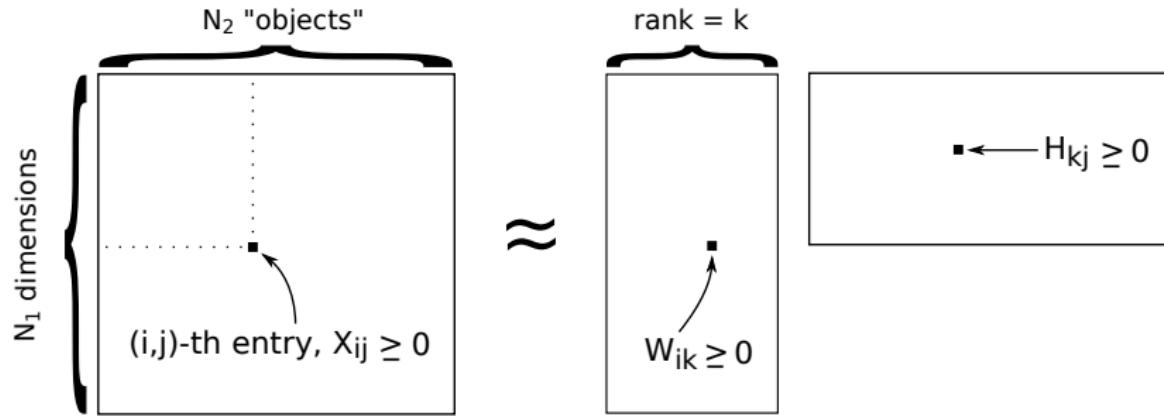
LDA can be thought of as an instance of nonnegative matrix factorization.

- ▶ It is a probabilistic model.
- ▶ Inference involves techniques not taught in this course.

We will discuss two other related models and their algorithms. These two models are called *nonnegative matrix factorization* (NMF)

- ▶ They can be used for the same tasks as LDA
- ▶ Though “nonnegative matrix factorization” is a general technique, “NMF” usually just refers to the following two methods.

NONNEGATIVE MATRIX FACTORIZATION



We use notation and think about the problem slightly differently from PMF

- ▶ Data X has nonnegative entries. None missing, but likely many zeros.
- ▶ The learned factorization W and H also have nonnegative entries.
- ▶ The value $X_{ij} \approx \sum_k W_{ik}H_{kj}$, but we won't write this with vector notation
- ▶ Later we interpret the output in terms of columns of W and H .

NONNEGATIVE MATRIX FACTORIZATION

What are some data modeling problems that can constitute X ?

- ▶ Text data:
 - ▶ Word term frequencies
 - ▶ X_{ij} contains the number of times word i appears in document j .
- ▶ Image data:
 - ▶ Face identification data sets
 - ▶ Put each *vectorized* $N \times M$ image of a face on a *column* of X .
- ▶ Other discrete grouped data:
 - ▶ Quantize *continuous* sets of features using K-means
 - ▶ X_{ij} counts how many times group j uses cluster i .
 - ▶ For example: group = song, features = $d \times n$ spectral information matrix

TWO OBJECTIVE FUNCTIONS

NMF minimizes one of the following two objective functions over W and H .

Choice 1: Squared error objective

$$\|X - WH\|^2 = \sum_i \sum_j (X_{ij} - (WH)_{ij})^2$$

Choice 2: Divergence objective

$$D(X \| WH) = - \sum_i \sum_j [X_{ij} \ln(WH)_{ij} - (WH)_{ij}]$$

- ▶ Both have the constraint that W and H contain nonnegative values.
- ▶ NMF uses a fast, simple algorithm for optimizing these two objectives.

MINIMIZATION AND MULTIPLICATIVE ALGORITHMS¹

Recall what we should look for in minimizing an objective “ $\min_h F(h)$ ”:

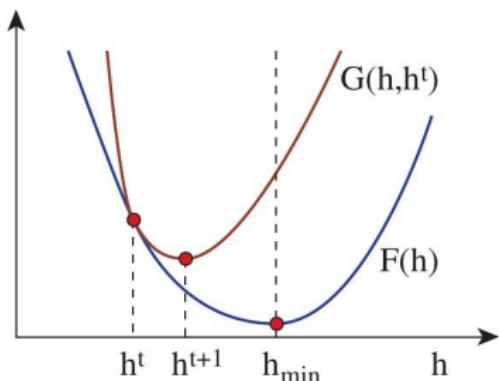
1. A way to generate a sequence of values h^1, h^2, \dots , such that

$$F(h^1) \geq F(h^2) \geq F(h^3) \geq \dots$$

2. Convergence of the sequence to a local minimum of F

The following algorithms fulfill these requirements. In this case:

- ▶ Minimization is done via an “auxiliary function.”
- ▶ Leads to a “multiplicative algorithm” for W and H .
- ▶ We’ll skip details (see reference).



¹For details, see D.D. Lee and H.S. Seung (2001). “Algorithms for non-negative matrix factorization.” *Advances in Neural Information Processing Systems*.

MULTIPLICATIVE UPDATE FOR $\|X - WH\|^2$

Problem

$$\min \sum_{ij} (X_{ij} - (WH)_{ij})^2 \quad \text{subject to } W_{ik} \geq 0, H_{kj} \geq 0.$$

Algorithm

- ▶ Randomly initialize H and W with nonnegative values.
- ▶ Iterate the following, first for all values in H , then all in W :

$$H_{kj} \leftarrow H_{kj} \frac{(W^T X)_{kj}}{(W^T W H)_{kj}},$$

$$W_{ik} \leftarrow W_{ik} \frac{(X H^T)_{ik}}{(W H H^T)_{ik}},$$

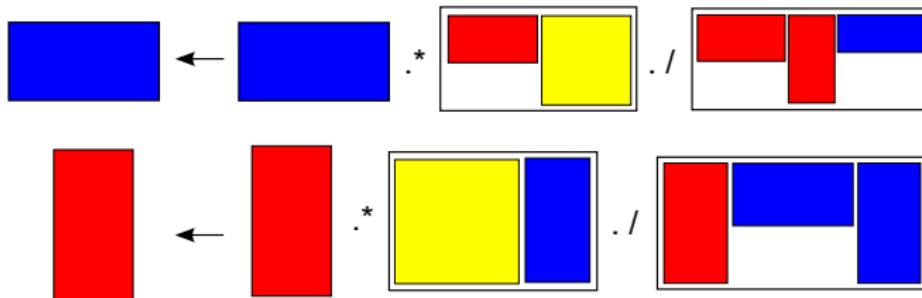
until the change in $\|X - WH\|^2$ is “small.”

VISUALIZATION AND MAXIMUM LIKELIHOOD

x ≈ w^H

A visualization that may be helpful. Use the color-coded definition above.

- ▶ Use element-wise multiplication/division across three columns below.
- ▶ Use matrix multiplication within each outlined box.



Probabilistically, the squared error penalty implies a Gaussian distribution,

$$X_{ij} \sim N(\sum_k W_{ik} H_{kj}, \sigma^2)$$

Since $X_{ij} \geq 0$ (and often isn't continuous), we are making an incorrect modeling assumption. Nevertheless, as with PMF it still works well.

MULTIPLICATIVE UPDATE FOR $D(X\|WH)$

Problem

$$\min \sum_{ij} \left[X_{ij} \ln \frac{1}{(WH)_{ij}} + (WH)_{ij} \right] \quad \text{subject to } W_{ik} \geq 0, H_{kj} \geq 0.$$

Algorithm

- ▶ Randomly initialize H and W with nonnegative values.
- ▶ Iterate the following, first for all values in H , then all in W :

$$H_{kj} \leftarrow H_{kj} \frac{\sum_i W_{ik} X_{ij} / (WH)_{ij}}{\sum_i W_{ik}},$$

$$W_{ik} \leftarrow W_{ik} \frac{\sum_j H_{kj} X_{ij} / (WH)_{ij}}{\sum_j H_{kj}},$$

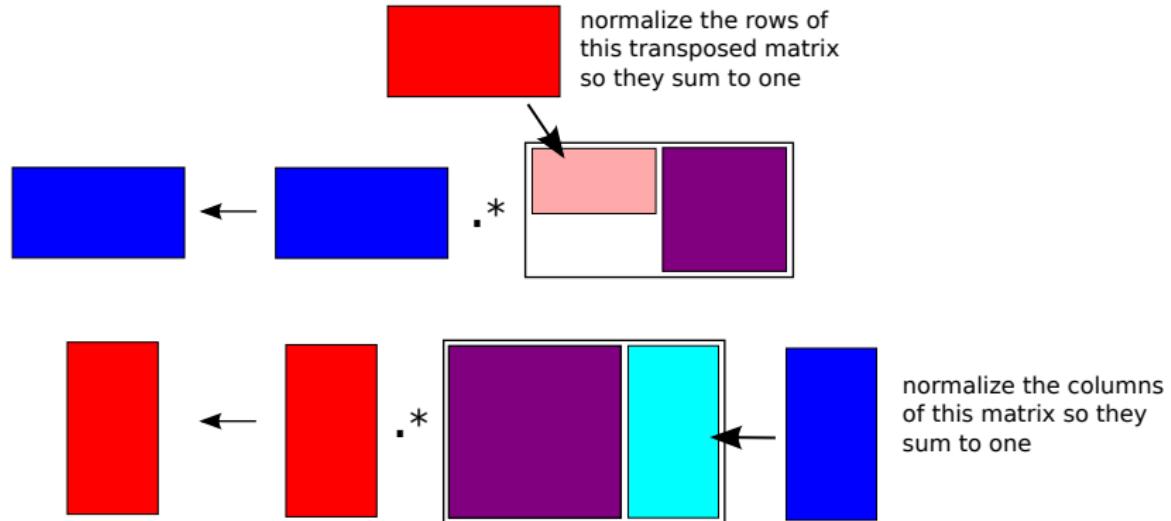
until the change in $D(X\|WH)$ is “small.”

VISUALIZATION

$$\text{purple} \stackrel{\text{def}}{=} \begin{matrix} \text{yellow} \\ \cdot / \end{matrix} \begin{matrix} \text{red} & \text{blue} \\ \text{H} \end{matrix}$$

Visualizing the update for the divergence penalty is more complicated.

- ▶ Use the color-coded definition above.
- ▶ “Purple” is the data matrix “dot-divided” by the approximation of it.



MAXIMUM LIKELIHOOD

The maximum likelihood interpretation of the divergence penalty is more interesting than for the squared error penalty.

If we model the data as independent Poisson random variables

$$X_{ij} \sim \text{Pois}((WH)_{ij}), \quad \text{Pois}(x|\lambda) = \frac{\lambda^x}{x!} e^{-\lambda}, \quad x \in \{0, 1, 2, \dots\},$$

then the negative divergence penalty is maximum likelihood for W and H .

$$\begin{aligned} -D(X \| WH) &= \sum_{ij} [X_{ij} \ln(WH)_{ij} - (WH)_{ij}] \\ &= \sum_{ij} \ln P(X_{ij} | W, H) + \text{constant} \end{aligned}$$

We use: $P(X|W, H) = \prod_{ij} P(X_{ij}|W, H) = \prod_{ij} \text{Pois}(X_{ij}|(WH)_{ij}).$

NMF AND TOPIC MODELING

As discussed, NMF can be used for topic modeling. In fact, one can show that the divergence penalty is closely related mathematically to LDA.

- Step 1. Form the term-frequency matrix X . ($X_{ij} = \#$ times word i in doc j)
- Step 2. Run NMF to learn W and H using $D(X\|WH)$ penalty
- Step 3. As an added step, after Step 2 is complete, for $k = 1, \dots, K$
 1. Set $a_k = \sum_i W_{ik}$
 2. Divide W_{ik} by a_k for all i
 3. Multiply H_{kj} by a_k for all j

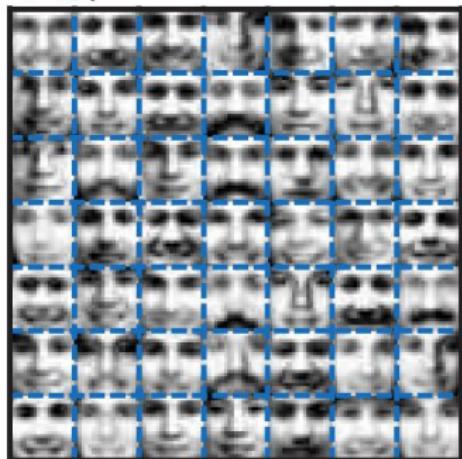
Notice that this does not change the matrix multiplication WH .

Interpretation: The k th *column* of W can be interpreted as the k th *topic*. The j th *column* of H can be interpreted as how much document j uses each topic.

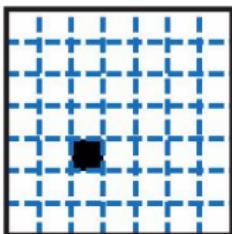
NMF AND FACE MODELING

For face modeling, put the face images along the columns of X and factorize. Show columns of W as image. Compare this with K-means and SVD.

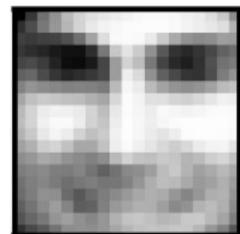
VQ



\times



$=$



K-means (i.e., VQ): Equivalent to each column of H having a single 1.
K-means learns averages of full faces.

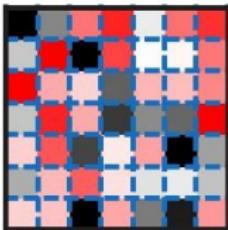
NMF AND FACE MODELING

For face modeling, put the face images along the columns of X and factorize. Show columns of W as image. Compare this with K-means and SVD.

SVD



\times



=



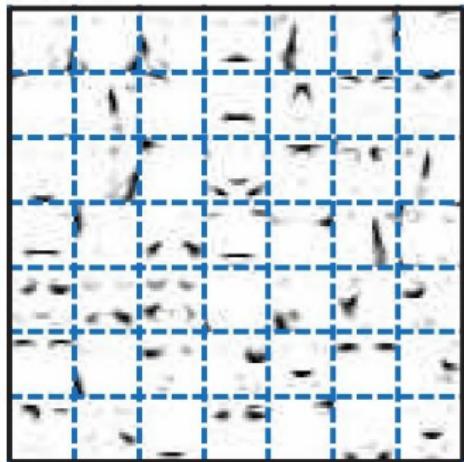
SVD: Finds the singular value decomposition of X .

Results not interpretable because of \pm values and orthogonality constraint

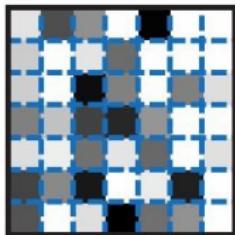
NMF AND FACE MODELING

For face modeling, put the face images along the columns of X and factorize. Show columns of W as image. Compare this with K-means and SVD.

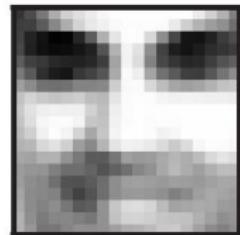
NMF



\times



$=$



NMF learns a “parts-based” representation. Each column captures something interpretable. This is a result of the nonnegativity constraint.

ColumbiaX: Machine Learning

Lecture 19

Prof. John Paisley

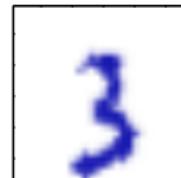
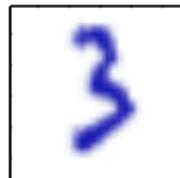
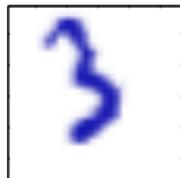
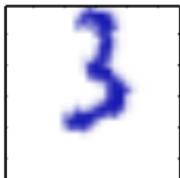
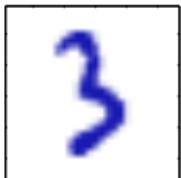
Department of Electrical Engineering
& Data Science Institute

Columbia University

PRINCIPAL COMPONENT ANALYSIS

DIMENSIONALITY REDUCTION

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



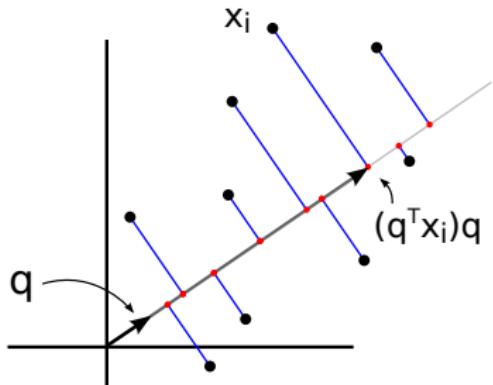
For example, we could represent the above images with three numbers since they have three degrees of freedom. Two for shifts and a third for rotation.

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

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

PRINCIPAL COMPONENT ANALYSIS

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



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

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

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

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

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

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

PCA : THE FIRST PRINCIPAL COMPONENT

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

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

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

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

This is the eigendecomposition problem:

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

PCA: GENERAL

The general form of PCA considers K eigenvectors,

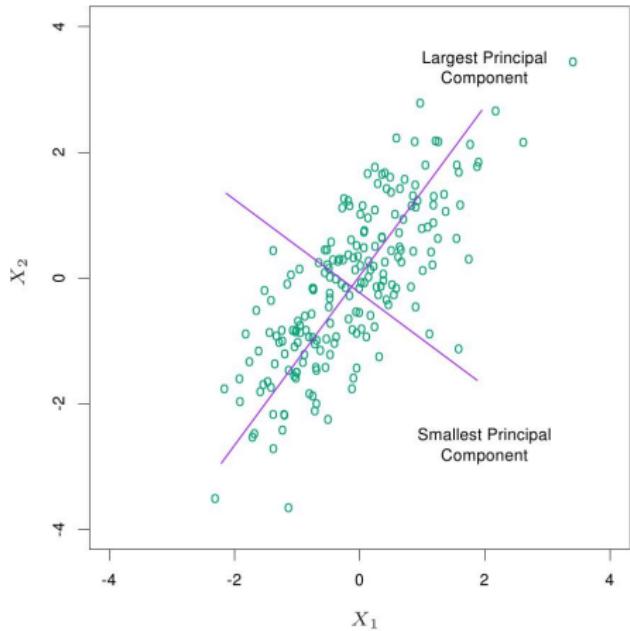
$$\begin{aligned} q &= \arg \min_q \sum_{i=1}^n \|x_i - \underbrace{\sum_{k=1}^K (x_i^T q_k) q_k}_{\text{approximates } x}\|^2 \quad \text{s.t. } q_k^T q_{k'} = \begin{cases} 1, & k = k' \\ 0, & k \neq k' \end{cases} \\ &= \arg \min_q \sum_{i=1}^n x_i^T x_i - \underbrace{\sum_{k=1}^K q_k^T \left(\sum_{i=1}^n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

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

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

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

EIGENVALUES, EIGENVECTORS AND THE SVD



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

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

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

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

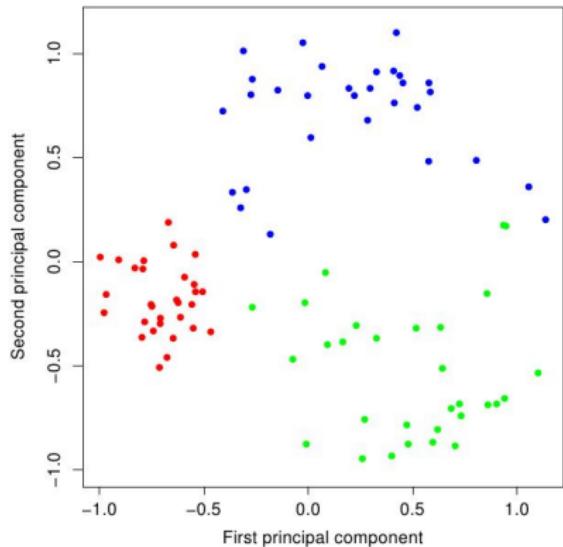
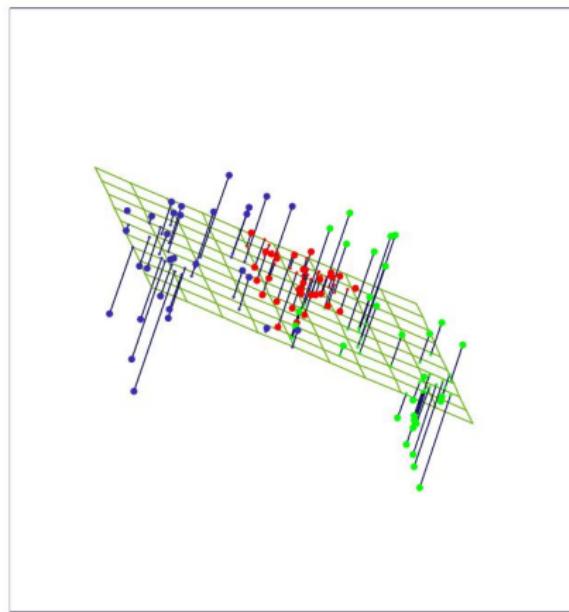
$$q_k^T q_k = 1, \quad q_k^T q_{k'} = 0$$

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

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

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

PCA: EXAMPLE OF PROJECTING FROM \mathbb{R}^3 TO \mathbb{R}^2



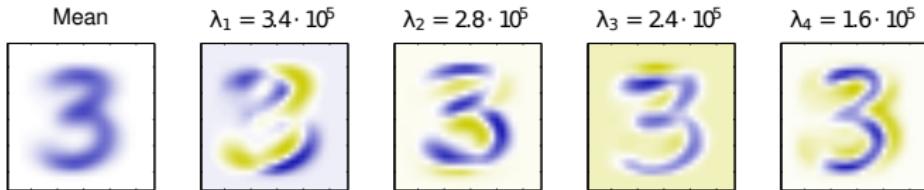
For this data, most information (structure in the data) can be captured in \mathbb{R}^2 .

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

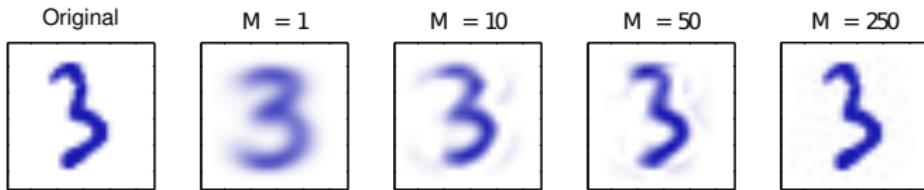
(right) The new coordinates for the data: $x_i \rightarrow x_i^{proj} = \begin{bmatrix} x_i^T q_1 \\ x_i^T q_2 \end{bmatrix}$.

EXAMPLE: DIGITS

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



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



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

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

PROBABILISTIC PCA

PCA AND THE SVD

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

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

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

Therefore,

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

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

A MODELING APPROACH TO PCA

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

This model will have the advantages of:

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

PROBABILISTIC PCA

In effect, this is a new matrix factorization model.

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

The generative process of Probabilistic PCA is

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

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

THE LIKELIHOOD

Maximum likelihood

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

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

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

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

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

EM FOR PROBABILISTIC PCA

Setup

The marginal log likelihood can be expressed using EM as

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

EM Algorithm: Remember that EM has two iterated steps

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

Again, for this to work well we need that

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

THE ALGORITHM

EM for Probabilistic PCA

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

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

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

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

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

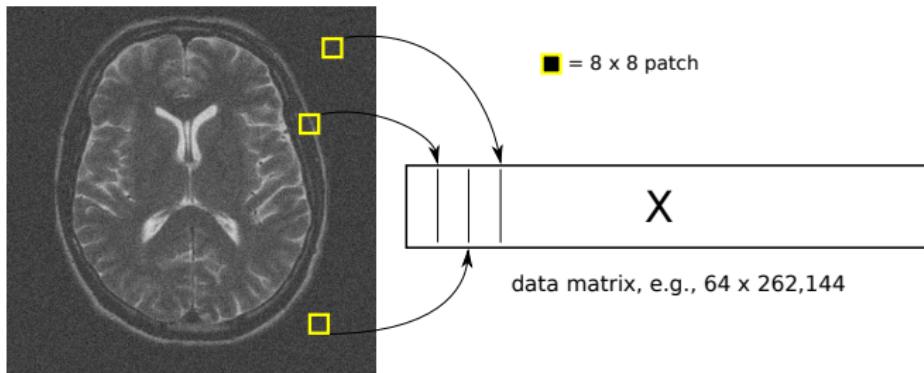
$$W = \left[\sum_{i=1}^n x_i \mu_i^T \right] \left[\sigma^2 I + \sum_{i=1}^n (\mu_i \mu_i^T + \Sigma_i) \right]^{-1}$$

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

Comment:

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

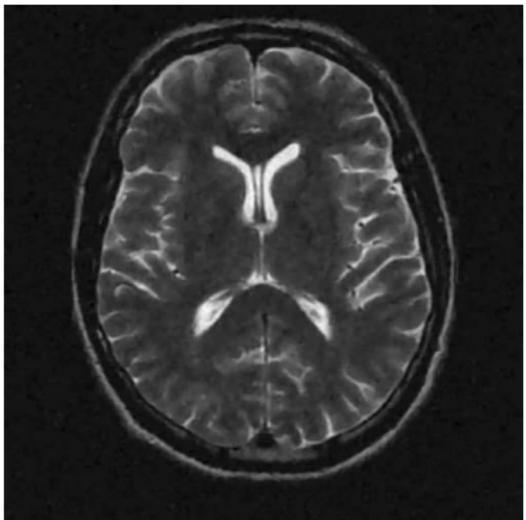
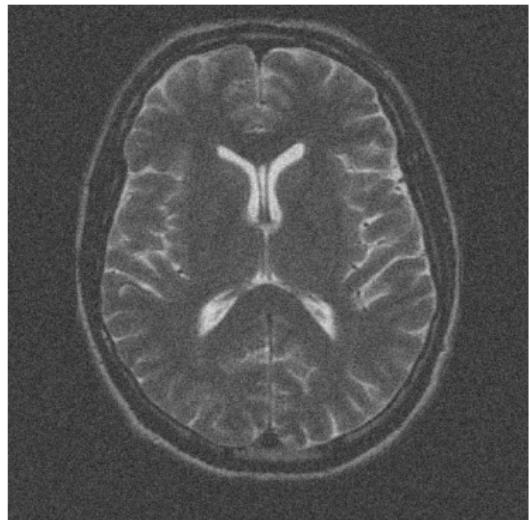
EXAMPLE: IMAGE PROCESSING



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

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

EXAMPLE: DENOISING



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

EXAMPLE: MISSING DATA



Another somewhat extreme example:

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

KERNEL PCA

KERNEL PCA

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

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

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

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

without having to work in the higher dimensional space.

- ▶ That is, how can we do PCA without explicitly using $\phi(\cdot)$ and q ?

KERNEL PCA

Notice that we can reorganize the operations of the eigendecomposition

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

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

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

Plug this equation for q_k back into the first equation:

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

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

KERNEL PCA

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

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

we get a new set of linear equations

$$K^2 \mathbf{a}_k = \lambda_k K \mathbf{a}_k \iff K \mathbf{a}_k = \lambda_k \mathbf{a}_k$$

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

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

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

KERNEL PCA ALGORITHM

Kernel PCA

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

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

Solve: The eigendecomposition

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

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

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

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

where a_{ki} is the i th dimension of the k th eigenvector \mathbf{a}_k .

KERNEL PCA AND NEW DATA

Q: How do we handle new data, x_0 ? Before, we could take the eigenvectors q_k and project $x_0^T q_k$, but a_k is different here.

A: Recall the relationship of a_k to q_k in kernel PCA is

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

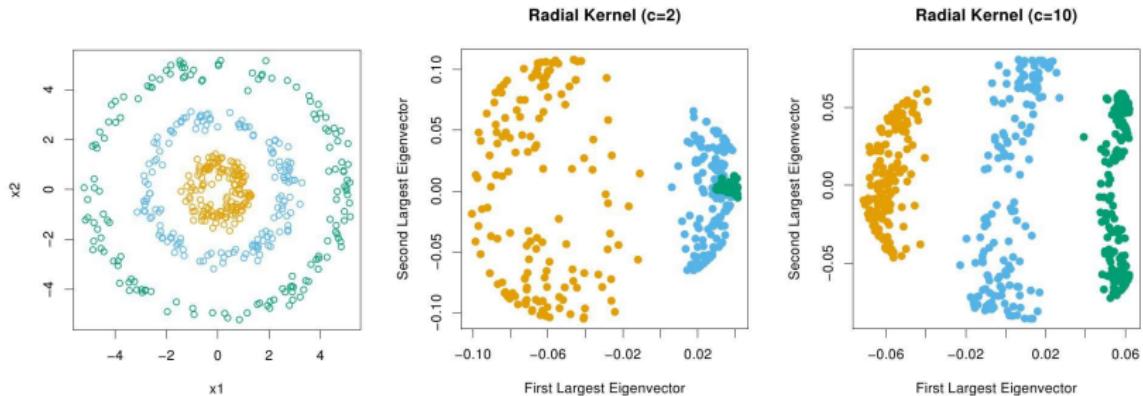
We used the “kernel trick” to avoid working with or even defining $\phi(x_i)$.

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

$$x_0 \xrightarrow{\text{projection}} \begin{bmatrix} \phi(x_0)^T q_1 \\ \vdots \\ \phi(x_0)^T q_r \end{bmatrix}$$

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

EXAMPLE RESULTS



An example of kernel PCA using the Gaussian kernel.

- (left) Original data, colored for reference (but may be classes)
- (middle) New coordinates using kernel width $c = 2$
- (right) New coordinates using kernel width $c = 10$

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

ColumbiaX: Machine Learning

Lecture 20

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

SEQUENTIAL DATA

So far, when thinking probabilistically we have focused on the i.i.d. setting.

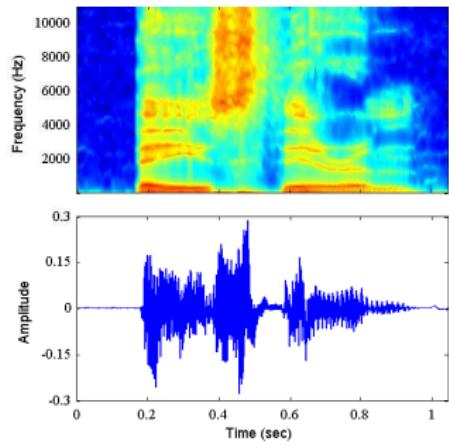
- ▶ All data are independent given a model parameter.
- ▶ This is often a reasonable assumption, but was also done for convenience.

In some applications this assumption is bad:

- ▶ Modeling rainfall as a function of hour
- ▶ Daily value of currency exchange rate
- ▶ Acoustic features of speech audio

The distribution on the next value clearly depends on the previous values.

A basic way to model sequential information is with a discrete, first-order Markov chain.



b	ey	z	th	ih	er	em
	Bayes'	Theorem				

MARKOV CHAINS

EXAMPLE: ZOMBIE WALKER¹



Imagine you see a zombie in an alley. Each time it moves forward it steps

(left, straight, right) with probability (p_l, p_s, p_r) ,

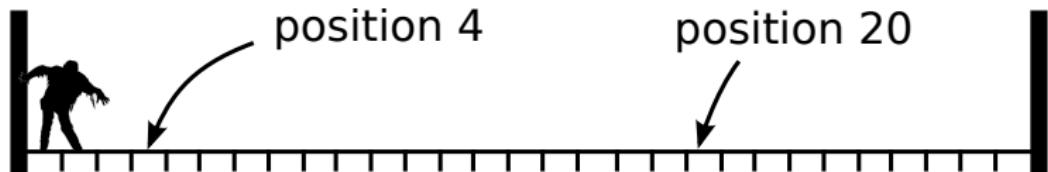
unless it's next to the wall, in which case it steps straight with probability p_s^w and toward the middle with probability p_m^w .

The distribution on the next location only depends on the current location.

¹This problem is often introduced with a “drunk,” so our maturity is textbook-level.

RANDOM WALK NOTATION

We simplify the problem by assuming there are only a finite number of positions the zombie can be in, and we model it as a random walk.



The distribution on the next position only depends on the current position. For example, for a position i away from the wall,

$$s_{t+1} \mid \{s_t = i\} = \begin{cases} i + 1 & \text{w.p. } p_r \\ i & \text{w.p. } p_s \\ i - 1 & \text{w.p. } p_l \end{cases}$$

This is called the *first-order Markov property*. It's the simplest type. A second-order model would depend on the previous two positions.

MATRIX NOTATION

A more compact notation uses a matrix.

For the random walk problem, imagine we have 6 different positions, called *states*. We can write the *transition matrix* as

$$M = \begin{bmatrix} p_s^w & p_m^w & 0 & 0 & 0 & 0 \\ p_l & p_s & p_r & 0 & 0 & 0 \\ 0 & p_l & p_s & p_r & 0 & 0 \\ 0 & 0 & p_l & p_s & p_r & 0 \\ 0 & 0 & 0 & p_l & p_s & p_r \\ 0 & 0 & 0 & 0 & p_m^w & p_s^w \end{bmatrix}$$

M_{ij} is the probability that the next position is j given the current position is i .

Of course we can jumble this matrix by moving rows and columns around in a correct way, as long as we can map the rows and columns to a position.

FIRST-ORDER MARKOV CHAIN (GENERAL)

Let $s \in \{1, \dots, S\}$. A sequence (s_1, \dots, s_t) is a *first-order Markov chain* if

$$p(s_1, \dots, s_t) \stackrel{(a)}{=} p(s_1) \prod_{u=2}^t p(s_u | s_1, \dots, s_{u-1}) \stackrel{(b)}{=} p(s_1) \prod_{u=2}^t p(s_u | s_{u-1})$$

From the two equalities above:

- (a) This equality is *always* true, regardless of the model (chain rule).
- (b) This simplification results from the Markov property assumption.

Notice the difference from the i.i.d. assumption

$$p(s_1, \dots, s_t) = \begin{cases} p(s_1) \prod_{u=2}^t p(s_u | s_{u-1}) & \text{Markov assumption} \\ \prod_{u=1}^t p(s_u) & \text{i.i.d. assumption} \end{cases}$$

From a modeling standpoint, this is a significant difference.

FIRST-ORDER MARKOV CHAIN (GENERAL)

Again, we encode this more general probability distribution in a matrix:

$$M_{ij} = p(s_t = j | s_{t-1} = i)$$

We will adopt the notation that rows are distributions.

- ▶ M is a *transition matrix*, or *Markov matrix*.
- ▶ M is $S \times S$ and each row sums to one.
- ▶ M_{ij} is the probability of transitioning to state j given we are in state i .

Given a starting state, s_0 , we generate a sequence (s_1, \dots, s_t) by sampling

$$s_t | s_{t-1} \sim \text{Discrete}(M_{s_{t-1}, :}).$$

We can model the starting state with its own separate distribution.

MAXIMUM LIKELIHOOD

Given a sequence, we can approximate the transition matrix using ML,

$$M_{\text{ML}} = \arg \max_M p(s_1, \dots, s_t | M) = \arg \max_M \sum_{u=1}^{t-1} \sum_{i,j}^S \mathbb{1}(s_u = i, s_{u+1} = j) \ln M_{ij}.$$

Since each row of M has to be a probability distribution, we can show that

$$M_{\text{ML}}(i,j) = \frac{\sum_{u=1}^{t-1} \mathbb{1}(s_u = i, s_{u+1} = j)}{\sum_{u=1}^{t-1} \mathbb{1}(s_u = i)}.$$

Empirically count how many times we observe a transition from $i \rightarrow j$ and divide by the total number of transitions from i .

Example: Model probability it rains (r) tomorrow given it rained today with observed fraction $\frac{\#\{r \rightarrow r\}}{\#\{r\}}$. Notice that $\#\{r\} = \#\{r \rightarrow r\} + \#\{r \rightarrow \text{no-}r\}$.

PROPERTY: STATE DISTRIBUTION

Q: Can we say at the beginning what state we'll be in at step $t + 1$?

A: Imagine at step t that we have a probability distribution on which state we're in, call it $p(s_t = u)$. Then the distribution on s_{t+1} is

$$p(s_{t+1} = j) = \sum_{u=1}^S \underbrace{p(s_{t+1} = j | s_t = u)}_{p(s_{t+1} = j, s_t = u)} p(s_t = u).$$

Represent $p(s_t = u)$ with the row vector w_t (the state distribution). Then

$$\underbrace{p(s_{t+1} = j)}_{w_{t+1}(j)} = \sum_{u=1}^S \underbrace{p(s_{t+1} = j | s_t = u)}_{M_{uj}} \underbrace{p(s_t = u)}_{w_t(u)}.$$

We can calculate this for all j with the matrix-vector product $w_{t+1} = w_t M$. Therefore, $w_{t+1} = w_1 M^t$ and w_1 can be indicator if starting state is known.

PROPERTY: STATIONARY DISTRIBUTION

Given current state distribution w_t , the distribution on the next state is

$$w_{t+1}(j) = \sum_{u=1}^S M_{uj} w_t(u) \iff w_{t+1} = w_t M$$

What happens if we project an infinite number of steps out?

Definition: Let $w_\infty = \lim_{t \rightarrow \infty} w_t$. Then w_∞ is the *stationary distribution*.

- ▶ There are many technical results that can be proved about w_∞ .
- ▶ Property: If the following are true, then w_∞ is the same vector for all w_0
 1. We can eventually reach any state starting from any other state,
 2. The sequence doesn't loop between states in a pre-defined pattern.
- ▶ Clearly $w_\infty = w_\infty M$ since w_t is converging and $w_{t+1} = w_t M$.

This last property is related to the first eigenvector of M^T :

$$M^T q_1 = \lambda_1 q_1 \implies \lambda_1 = 1, \quad w_\infty = \frac{q_1}{\sum_{u=1}^S q_1(u)}$$

A RANKING ALGORITHM

EXAMPLE: RANKING OBJECTS

We show an example of using the stationary distribution of a Markov chain to rank objects. The data are pairwise comparisons between objects.

For example, we might want to rank

- ▶ Sports teams or athletes competing against each other
- ▶ Objects being compared and selected by users

Our goal is to rank objects from “best” to “worst.”

- ▶ We will construct a random walk matrix on the objects.
- ▶ The stationary distribution will give us the ranking.
- ▶ Notice: We don’t consider the sequential information in the data itself.
The Markov chain is an artificial modeling construct.

EXAMPLE: TEAM RANKINGS

Problem setup

We want to construct a Markov chain where each team is a state.

- ▶ We encourage transitions from teams that lose to teams that win.
- ▶ Predicting the “state” (i.e., team) far in the future, we can interpret a more probable state as a better team.

One specific approach to this specific problem:

- ▶ Transitions only occur between teams that play each other.
- ▶ If Team A beats Team B, there should be a high probability of transitioning from $B \rightarrow A$ and small probability from $A \rightarrow B$.
- ▶ The strength of the transition can be linked to the score of the game.

EXAMPLE: TEAM RANKINGS

How about this?

Initialize \hat{M} to a matrix of zeros. For a particular game, let j_1 be the index of Team A and j_2 the index of Team B. Then update

$$\hat{M}_{j_1 j_1} \leftarrow \hat{M}_{j_1 j_1} + \mathbb{1}\{\text{Team A wins}\} + \frac{\text{points}_{j_1}}{\text{points}_{j_1} + \text{points}_{j_2}},$$

$$\hat{M}_{j_2 j_2} \leftarrow \hat{M}_{j_2 j_2} + \mathbb{1}\{\text{Team B wins}\} + \frac{\text{points}_{j_2}}{\text{points}_{j_1} + \text{points}_{j_2}},$$

$$\hat{M}_{j_1 j_2} \leftarrow \hat{M}_{j_1 j_2} + \mathbb{1}\{\text{Team B wins}\} + \frac{\text{points}_{j_2}}{\text{points}_{j_1} + \text{points}_{j_2}},$$

$$\hat{M}_{j_2 j_1} \leftarrow \hat{M}_{j_2 j_1} + \mathbb{1}\{\text{Team A wins}\} + \frac{\text{points}_{j_1}}{\text{points}_{j_1} + \text{points}_{j_2}}.$$

After processing all games, let M be the matrix formed by normalizing the rows of \hat{M} so they sum to 1.

EXAMPLE: 2014-2015 COLLEGE BASKETBALL SEASON

USA Today Coaches Poll			
RK	TEAM	RECORD	PTS
1	Villanova (30)	35-5	750
2	North Carolina	33-7	720
3	Kansas	33-5	657
4	Oklahoma	29-8	643
5	Virginia	29-8	631
6	Oregon	31-7	596
7	Michigan State	29-6	488
8	Miami	27-8	480
9	Indiana	27-8	456
10	Syracuse	23-14	446
11	Xavier	28-6	361
12	Texas A&M	28-9	358
12	Maryland	27-9	358
14	West Virginia	26-9	331
15	Iowa State	23-12	315
16	Kentucky	27-9	297
17	Notre Dame	24-12	285
18	Duke	25-11	263
19	Purdue	26-9	184
20	Utah	27-9	170
21	Gonzaga	28-8	157
22	Arizona	25-9	154
23	Wisconsin	22-13	149
24	Baylor	22-12	105
25	Iowa	22-11	82

Markov chain ranking		
RK	SCORE	TEAM
1	0.0090112	Villanova
2	0.0079282	Kansas
3	0.0074781	North Carolina
4	0.0067752	Virginia
5	0.0065791	Oklahoma
6	0.0063760	Oregon
7	0.0058095	Michigan St
8	0.0056623	Xavier OH
9	0.0055031	Miami FL
10	0.0049979	West Virginia
11	0.0047690	Utah
12	0.0047131	Kentucky
13	0.0046578	Indiana
14	0.0046482	Seton Hall
15	0.0046097	Texas A&M
16	0.0045635	Duke
17	0.0042596	Maryland
18	0.0042441	Purdue
19	0.0041866	Iowa St
20	0.0041599	St Joseph's PA
21	0.0040336	Notre Dame
22	0.0040017	Arizona
23	0.0039594	George Wash
24	0.0039369	Louisville
25	0.0039273	Providence RI

1,549 total teams

22,033 total games

SCORE = stationary distribution

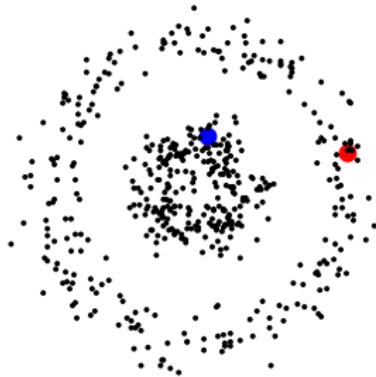
A CLASSIFICATION ALGORITHM

SEMI-SUPERVISED LEARNING

Imagine we have data with very few labels.

We want to use the structure in the dataset to help classify the unlabeled data.

We can do this with a Markov chain.



Semi-supervised learning uses partially labeled data to do classification.

- ▶ Many or most y_i will be missing in the pair (x_i, y_i) .
- ▶ Still, there is structure in x_1, \dots, x_n that we don't want to throw away.
- ▶ In the example above, we might want the inner ring to be one class (blue) and the outer ring another (red).

A RANDOM WALK CLASSIFIER

We will define a classifier where, starting from any data point x_i ,

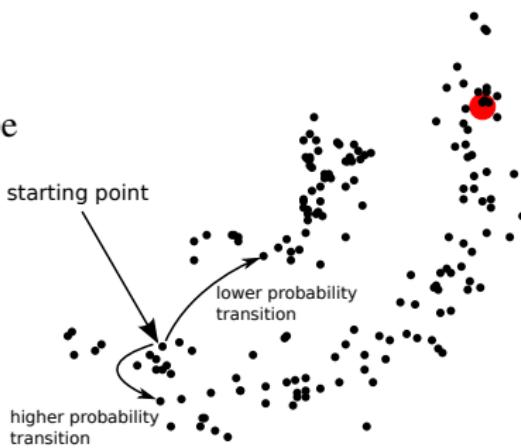
- ▶ A “random walker” moves around from point to point
- ▶ A transition between nearby points has higher probability
- ▶ A transition to a labeled point terminates the walk
- ▶ The label of a point x_i is the label of the terminal point

One possible random walk matrix

1. Let the *unnormalized* transition matrix be

$$\hat{M}_{ij} = \exp \left\{ -\frac{\|x_i - x_j\|^2}{b} \right\}$$

2. Normalize rows of \hat{M} to get M
3. If x_i has label y_i , re-define $M_{ii} = 1$



PROPERTY: ABSORBING STATES

Imagine we have S states. If $p(s_t = i | s_{t-1} = i) = 1$, then the i th state is called an **absorbing state** since we can never leave it.

Q: Given initial state $s_0 = j$ and set of absorbing states $\{i_1, \dots, i_k\}$, what is the probability a Markov chain terminates at a particular absorbing state?

- ▶ Aside: For the semi-supervised classifier, the answer gives the probability on the label of x_j .

A: Start a random walk at j and keep track of the distribution on states.

- ▶ w_0 is a vector of 0's with a 1 in entry j because we know $s_0 = j$
- ▶ If M is the transition matrix, we know that $w_{t+1} = w_t M$.
- ▶ So we want $w_\infty = w_0 M^\infty$.

PROPERTY: ABSORBING STATE DISTRIBUTION

Group the absorbing states and break up the transition matrix into quadrants:

$$M = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}$$

The bottom half contains the self-transitions of the absorbing states.

Observation: $w_{t+1} = w_t M = w_{t-1} M^2 = \dots = w_0 M^{t+1}$

So we need to understand what's going on with M^t . For the first two we have

$$M^2 = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A^2 & AB + B \\ 0 & I \end{bmatrix}$$

$$M^3 = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^2 & AB + B \\ 0 & I \end{bmatrix} = \begin{bmatrix} A^3 & A^2B + AB + B \\ 0 & I \end{bmatrix}$$

GEOMETRIC SERIES

Detour: We will use the matrix version of the following scalar equality.

Definition: Let $0 < r < 1$. Then $\sum_{u=0}^{t-1} r^u = \frac{1-r^t}{1-r}$ and so $\sum_{u=0}^{\infty} r^u = \frac{1}{1-r}$.

Proof: First define the top equality and create the bottom equality

$$\begin{aligned} C_t &= 1 + r + r^2 + \cdots + r^{t-1} \\ r C_t &= \qquad\qquad r + r^2 + \cdots + r^{t-1} + r^t \end{aligned}$$

and so

$$C_t - r C_t = 1 - r^t.$$

Therefore

$$C_t = \sum_{u=0}^{t-1} r^u = \frac{1-r^t}{1-r} \quad \text{and} \quad C_{\infty} = \frac{1}{1-r}.$$

PROPERTY: ABSORBING STATE DISTRIBUTION

A matrix version of the geometric series appears here. We see the pattern

$$M^t = \begin{bmatrix} A^t & \left(\sum_{u=0}^{t-1} A^u\right) B \\ 0 & I \end{bmatrix}.$$

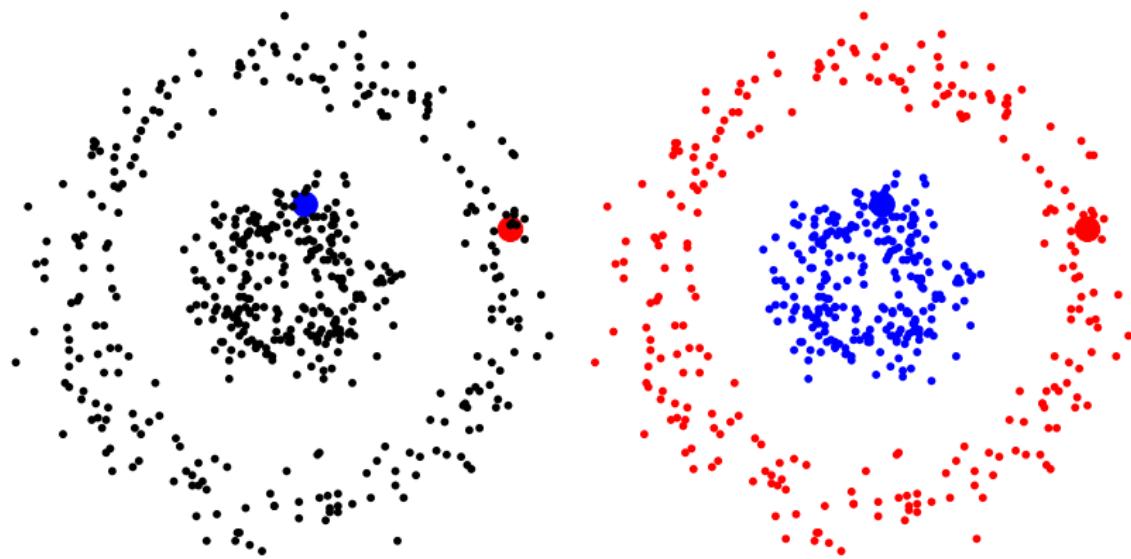
Two key things that can be shown are:

$$A^\infty = 0, \quad \sum_{u=0}^{\infty} A^u = (I - A)^{-1}$$

Summary:

- ▶ After an infinite # of steps, $w_\infty = w_0 M^\infty = w_0 \begin{bmatrix} 0 & (I - A)^{-1}B \\ 0 & I \end{bmatrix}$.
- ▶ The non-zero dimension of w_0 picks out a row of $(I - A)^{-1}B$.
- ▶ The probability that a random walk started at x_j terminates at the i th absorbing state is $[(I - A)^{-1}B]_{ji}$.

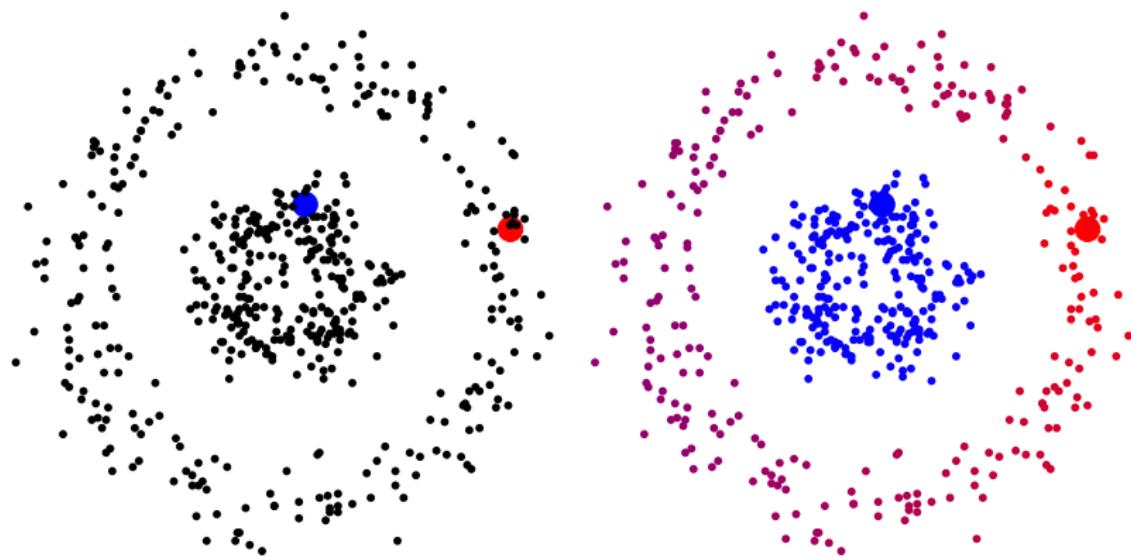
CLASSIFICATION EXAMPLE



Using a Gaussian kernel normalized on the rows. The color indicates the distribution on the terminal state for each starting point.

Kernel width was tuned to give this result.

CLASSIFICATION EXAMPLE



Using a Gaussian kernel normalized on the rows. The color indicates the distribution on the terminal state for each starting point.

Kernel width is larger here. Therefore, purple points may leap to the center.

ColumbiaX: Machine Learning

Lecture 21

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

HIDDEN MARKOV MODELS

OVERVIEW

Motivation

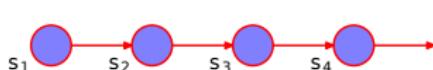
We have seen how Markov models can model sequential data.

- ▶ We assumed the observation was the sequence of states.
- ▶ Instead, each state may define a *distribution* on observations.

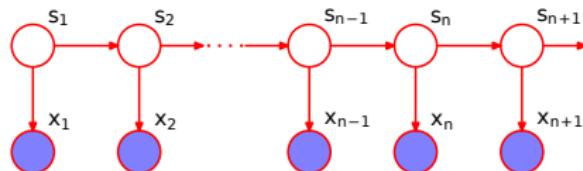
Hidden Markov model

A *hidden* Markov model treats a sequence of data slightly differently.

- ▶ Assume a hidden (i.e., unobserved, latent) sequence of states.
- ▶ An observation is drawn from the distribution associated with its state.



Markov model



hidden Markov model

MARKOV TO HIDDEN MARKOV MODELS

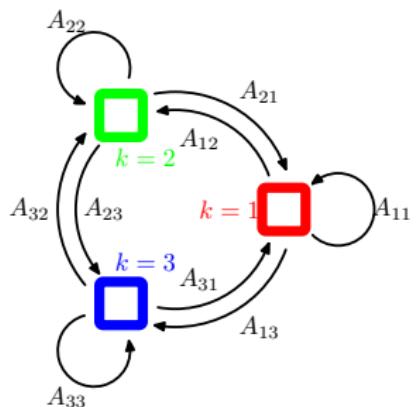
Markov models

Imagine we have three possible states in \mathbb{R}^2 .

The data is a sequence of these positions.

Since there are only three unique positions,
we can give an index in place of coordinates.

For example, the sequence $(1, 2, 1, 3, 2, \dots)$
would map to a sequence of 2-D vectors.



Using the notation of the figure, A is a 3×3 *transition matrix*. A_{ij} is the probability of transitioning from state i to state j .

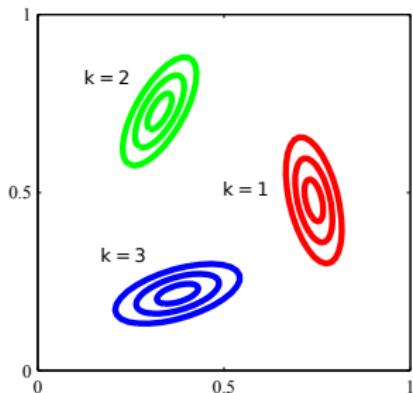
MARKOV TO HIDDEN MARKOV MODELS

Hidden Markov models

Now imagine the same three states, but each time the coordinates are randomly permuted.

The state sequence is still a set of indexes, e.g., $(1, 2, 1, 3, 2, \dots)$ of positions in \mathbb{R}^2 .

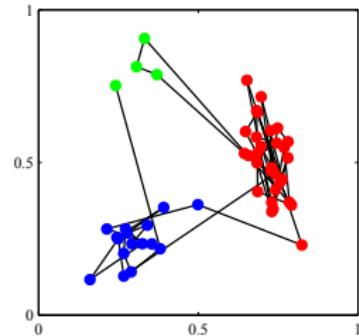
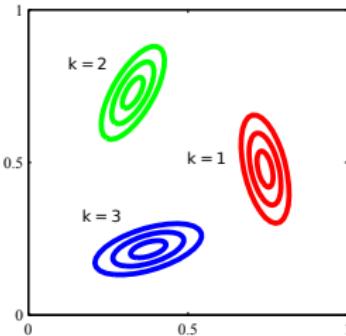
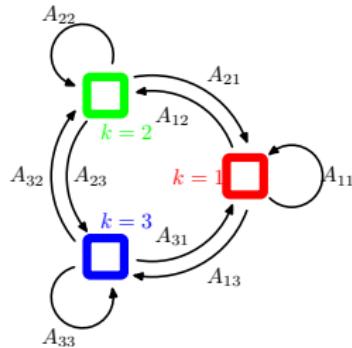
However, if μ_1 is the position of state #1, then we observe $x_i = \mu_1 + \epsilon_i$ if $s_i = 1$.



Exactly as before, we have a state transition matrix A (in this case 3×3).

However, the observed data is a sequence (x_1, x_2, x_3, \dots) where each $x \in \mathbb{R}^2$ is a random perturbation of the state it's assigned to $\{\mu_1, \mu_2, \mu_3\}$.

MARKOV TO HIDDEN MARKOV MODELS



A continuous hidden Markov model

This HMM is *continuous* because each $x \in \mathbb{R}^2$ in the sequence (x_1, \dots, x_T) .

- (left) A Markov state transition distribution for an unobserved sequence
- (middle) The state-dependent distributions used to generate observations
- (right) The data sequence. Colors indicate the distribution (state) used.

HIDDEN MARKOV MODELS

Definition

A *hidden Markov model (HMM)* consists of:

- ▶ An $S \times S$ Markov transition matrix A for transitioning between S states.
- ▶ An initial state distribution π for selecting the first state.
- ▶ A state-dependent *emission distribution*, $\text{Prob}(x_i|s_i = k) = p(x_i|\theta_{s_i})$.

The model generates a sequence $(x_1, x_2, x_3 \dots)$ by:

1. Sampling the first state $s_1 \sim \text{Discrete}(\pi)$ and $x_1 \sim p(x|\theta_{s_1})$.
2. Sampling the Markov chain of states, $s_i | \{s_{i-1} = k'\} \sim \text{Discrete}(A_{k',:})$, followed by the observation $x_i | s_i \sim p(x|\theta_{s_i})$.

Continuous HMM: $p(x|\theta_s)$ is a continuous distribution, often Gaussian.

Discrete HMM: $p(x|\theta_s)$ is a discrete distribution, θ_s a vector of probabilities.

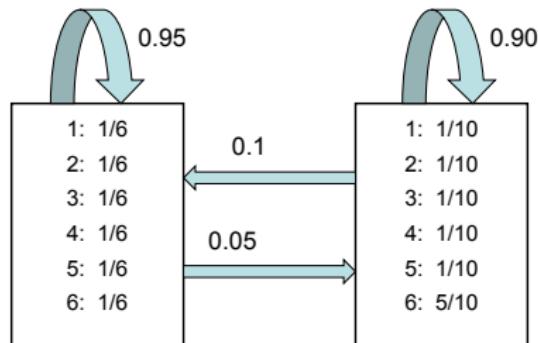
We focus on discrete case. Let B be a matrix, where $B_{s,:} = \theta_s$ (from above).

EXAMPLE: DISHONEST CASINO

Problem

Here is an example of a *discrete* hidden Markov model.

- ▶ Consider two dice, one is fair and one is unfair.
- ▶ At each roll, we either keep the current dice, or switch to the other one.
- ▶ The observation is the sequence of numbers rolled.



The transition matrix is

$$A = \begin{bmatrix} 0.95 & 0.05 \\ 0.10 & 0.90 \end{bmatrix}$$

The emission matrix is

$$B = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{2} \end{bmatrix}$$

Let $\pi = [\frac{1}{2} \ \frac{1}{2}]$.

SOME ESTIMATION PROBLEMS

State estimation

- ▶ **Given:** An HMM $\{\pi, A, B\}$ and observation sequence (x_1, \dots, x_T)
- ▶ **Estimate:** State probability for x_i using “forward-backward algorithm,”

$$p(s_i = k \mid x_1, \dots, x_T, \pi, A, B).$$

State sequence

- ▶ **Given:** An HMM $\{\pi, A, B\}$ and observation sequence (x_1, \dots, x_T)
- ▶ **Estimate:** Most probable state sequence using the “Viterbi algorithm,”

$$s_1, \dots, s_T = \arg \max_{\vec{s}} p(s_1, \dots, s_T \mid x_1, \dots, x_T, \pi, A, B).$$

Learn an HMM

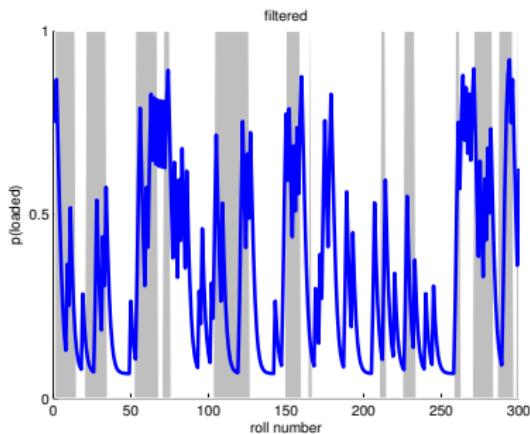
- ▶ **Given:** An observation sequence (x_1, \dots, x_T)
- ▶ **Estimate:** HMM parameters π, A, B using maximum likelihood

$$\pi_{\text{ML}}, A_{\text{ML}}, B_{\text{ML}} = \arg \max_{\pi, A, B} p(x_1, \dots, x_T \mid \pi, A, B)$$

EXAMPLES

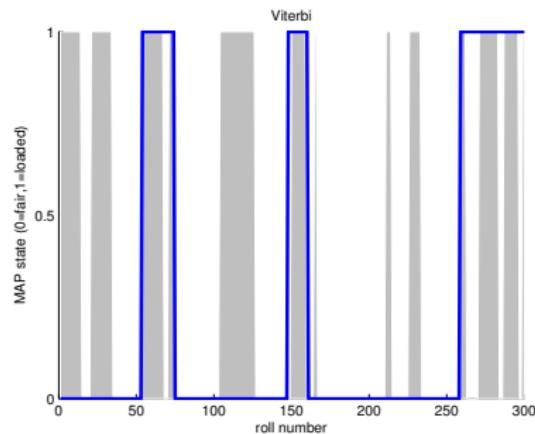
Before we look at the details, here are examples for the dishonest casino.

- ▶ Not shown is that π, A, B were learned first in order to calculate this.
- ▶ Notice that the right plot isn't just a rounding of the left plot.



State estimation result

Gray bars: Loaded dice used
Blue: Probability $p(s_i = \text{loaded} | x_{1:T}, \pi, A, B)$



State sequence result

Gray bars: Loaded dice used
Blue: Most probable state sequence

LEARNING THE HMM

LEARNING THE HMM: THE LIKELIHOOD

We focus on the discrete HMM. To learn the HMM parameters, maximize

$$\begin{aligned} p(\vec{x} | \pi, A, B) &= \sum_{s_1=1}^S \cdots \sum_{s_T=1}^S p(\vec{x}, s_1, \dots, s_T | \pi, A, B) \\ &= \sum_{s_1=1}^S \cdots \sum_{s_T=1}^S \prod_{i=1}^T p(x_i | s_i, B) p(s_i | s_{i-1}, \pi, A) \end{aligned}$$

- ▶ $p(x_i | s_i, B) = B_{s_i, x_i} \leftarrow s_i$ indexes the distribution, x_i is the observation
- ▶ $p(s_i | s_{i-1}, \pi, A) = A_{s_{i-1}, s_i}$ (or π_{s_1}) \leftarrow since s_1, \dots, s_T is a Markov chain

LEARNING THE HMM: THE LOG LIKELIHOOD

- ▶ Maximizing $p(\vec{x}|\pi, A, B)$ is hard since the objective has log-sum form

$$\ln p(\vec{x}|\pi, A, B) = \ln \sum_{s_1=1}^S \cdots \sum_{s_T=1}^S \prod_{i=1}^T p(x_i | s_i, B) p(s_i | s_{i-1}, \pi, A)$$

- ▶ However, if we had or learned \vec{s} it would be easy (remove the sums).
- ▶ In addition, we can calculate $p(\vec{s} | \vec{x}, \pi, A, B)$, though it's much more complicated than in previous models.
- ▶ Therefore, we can use the EM algorithm! The following is high-level.

LEARNING THE HMM: THE LOG LIKELIHOOD

E-step: Using $q(\vec{s}) = p(\vec{s} \mid \vec{x}, \pi, A, B)$, calculate

$$\mathcal{L}(\vec{x}, \pi, A, B) = \mathbb{E}_q [\ln p(\vec{x}, \vec{s} \mid \pi, A, B)].$$

M-Step: Maximize \mathcal{L} with respect to π, A, B .

This part is tricky since we need to take the expectation using $q(\vec{s})$ of

$$\begin{aligned}\ln p(\vec{x}, \vec{s} \mid \pi, A, B) &= \sum_{i=1}^T \sum_{k=1}^S \underbrace{\mathbb{1}(s_i = k) \ln B_{k,x_i}}_{\text{observations}} + \sum_{k=1}^S \underbrace{\mathbb{1}(s_1 = k) \ln \pi_k}_{\text{initial state}} \\ &\quad + \sum_{i=2}^T \sum_{j=1}^S \sum_{k=1}^S \underbrace{\mathbb{1}(s_{i-1} = j, s_i = k) \ln A_{j,k}}_{\text{Markov chain}}\end{aligned}$$

The following is an overview to help you better navigate the books/tutorials.¹

¹See the classic tutorial: Rabiner, L.R. (1989). “A tutorial on hidden Markov models and selected applications in speech recognition.” *Proceedings of the IEEE* 77(2), 257–285.

LEARNING THE HMM WITH EM

E-Step

Let's define the following conditional posterior quantities:

$\gamma_i(k)$ = the posterior probability that $s_i = k$

$\xi_i(j, k)$ = the posterior probability that $s_{i-1} = j$ and $s_i = k$

Therefore, γ_i is a vector and ξ_i is a matrix, both varying over i .

We can calculate both of these using the “forward-backward” algorithm.
(We won’t cover it in this class, but Rabiner’s tutorial is good.)

Given these values the E-step is:

$$\mathcal{L} = \sum_{k=1}^S \gamma_1(k) \ln \pi_k + \sum_{i=2}^T \sum_{j=1}^S \sum_{k=1}^S \xi_i(j, k) \ln A_{j,k} + \sum_{i=1}^T \sum_{k=1}^S \gamma_i(k) \ln B_{k,x_i}$$

This gives us everything we need to update π, A, B .

LEARNING THE HMM WITH EM

M-Step

The updates for the HMM parameters are:

$$\pi_k = \frac{\gamma_1(k)}{\sum_j \gamma_1(j)}, \quad A_{j,k} = \frac{\sum_{i=2}^T \xi_i(j, k)}{\sum_{i=2}^T \sum_{l=1}^S \xi_i(j, l)}, \quad B_{k,v} = \frac{\sum_{i=1}^T \gamma_i(k) \mathbb{1}\{x_i = v\}}{\sum_{i=1}^T \gamma_i(k)}$$

The updates can be understood as follows:

- ▶ $A_{j,k}$ is the expected fraction of transitions $j \rightarrow k$ when we start at j
 - ▶ Numerator: *Expected* count of transitions $j \rightarrow k$
 - ▶ Denominator: *Expected* total number of transitions from j
- ▶ $B_{k,v}$ is the expected fraction of data coming from state k and equal to v
 - ▶ Numerator: *Expected* number of observations $= v$ from state k
 - ▶ Denominator: *Expected* total number of observations from state k
- ▶ π has interpretation similar to A

LEARNING THE HMM WITH EM

M-Step: N sequences

Usually we'll have multiple sequences that are modeled by an HMM. In this case, the updates for the HMM parameters with N sequences are:

$$\pi_k = \frac{\sum_{n=1}^N \gamma_1^n(k)}{\sum_{n=1}^N \sum_j \gamma_1^n(j)}, \quad A_{j,k} = \frac{\sum_{n=1}^N \sum_{i=2}^{T_n} \xi_i^n(j, k)}{\sum_{n=1}^N \sum_{i=2}^{T_n} \sum_{l=1}^S \xi_i^n(j, l)},$$
$$B_{k,v} = \frac{\sum_{n=1}^N \sum_{i=1}^{T_n} \gamma_i^n(k) \mathbb{1}\{x_i = v\}}{\sum_{n=1}^N \sum_{i=1}^{T_n} \gamma_i^n(k)}$$

The modifications are:

- ▶ Each sequence can be of different length, T_n
- ▶ Each sequence has its own set of γ and ξ values
- ▶ Using this we sum over the sequences, with the interpretation the same.

APPLICATION: SPEECH RECOGNITION

APPLICATION: SPEECH RECOGNITION

Problem

Given speech in the form of an audio signal, determine the words spoken.

Method

- ▶ Words are broken down into small sound units (called *phonemes*). The states in the HMM are intended to represent phonemes.
- ▶ The incoming sound signal is transformed into a sequence of vectors (feature extraction). Each vector x_i is indexed by a time step i .
- ▶ The sequence $x_{1:T}$ of feature vectors is the data used to learn the HMM.

PHONEME MODELS

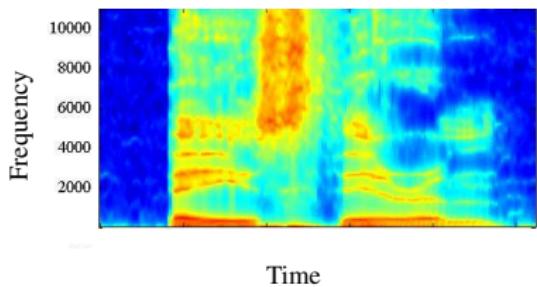
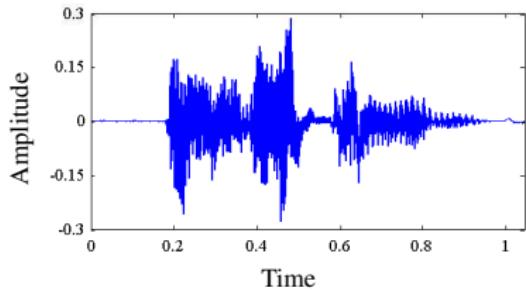
Phoneme

A phoneme is defined as the smallest unit of sound in a language that distinguishes between distinct meanings. English uses about 50 phonemes.

Example

Zero	Z IH R OW	Six	S IH K S
One	W AH N	Seven	S EH V AX N
Two	T UW	Eight	EY T
Three	TH R IY	Nine	N AY N
Four	F OW R	Oh	OW
Five	F AY V		

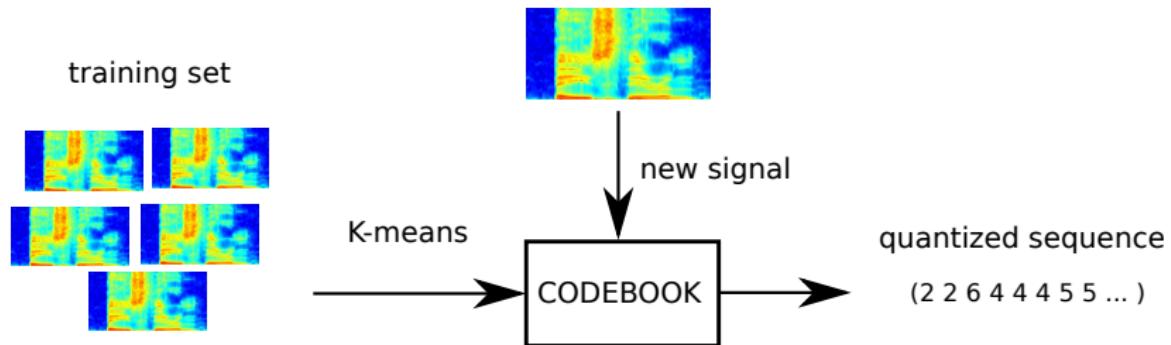
PREPROCESSING SPEECH



Feature extraction

- ▶ A speech signal is measured as amplitude over time.
- ▶ The signal is typically transformed into features by breaking down frequency content of the signal in a sliding time-window.
- ▶ (above) Each column is the frequency content of about 50 milliseconds (10,000+ dimensional). This can be further reduced to, e.g., 40 dims.

DATA QUANTIZATION



We could work directly with the extracted features and learn a Gaussian distribution for each state, i.e., a continuous HMM.

To transition to a discrete HMM, we can perform vector quantization using a codebook learned by K-means.

A SPEECH RECOGNITION MODEL

These models and problems can become more complex. For now, imagine a simple automated phone conversation using a question/answer format.

Training data: Quantized feature sequences of words, e.g., “yes,” “no”

Learn: An HMM for each word using all training sequences of that word

Predict: Let w index the word. Predict the word of a new sequence using

$$w_{new} = \arg \max_w p(\vec{x}_{new} | \pi_w, A_w, B_w) \leftarrow \text{requires forward-backward}$$

Notice that this is a Bayes classifier with a uniform prior on the word!

- ▶ We’re learning a class-conditional discrete HMM.
- ▶ We could try something else, e.g., a GMM instead of an HMM.
- ▶ If the GMM predicts better, then use it instead. (But we anticipate that it won’t since the HMM models sequential information.)

ColumbiaX: Machine Learning

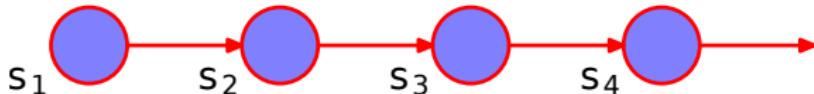
Lecture 22

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

MARKOV MODELS



The sequence (s_1, s_2, s_3, \dots) has the *Markov property*, if for all t

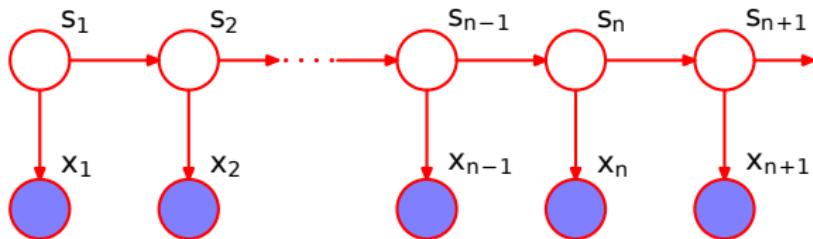
$$p(s_t | s_{t-1}, \dots, s_1) = p(s_t | s_{t-1}).$$

Our first encounter with Markov models assumed a finite state space, meaning we can define an indexing such that $s \in \{1, \dots, S\}$.

This allowed us to represent the transition probabilities in a matrix,

$$A_{ij} \Leftrightarrow p(s_t = j | s_{t-1} = i).$$

HIDDEN MARKOV MODELS



The hidden Markov model modified this by assuming the sequence of states was a *latent process* (i.e., unobserved).

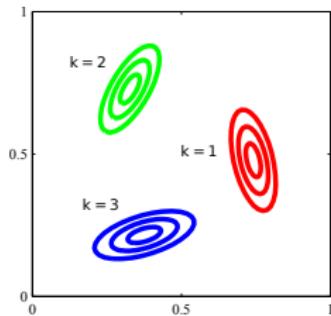
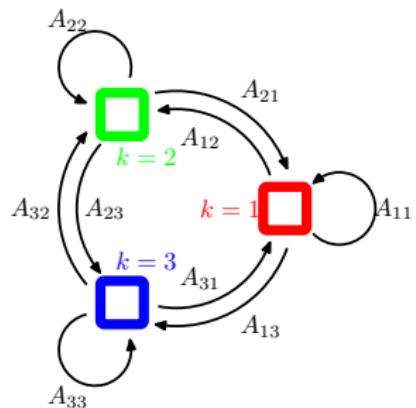
An observation x_t is associated with each s_t , where $x_t \mid s_t \sim p(x|\theta_{s_t})$.

Like a mixture model, this allowed for a few distributions to generate the data. It adds an extra transition rule between distributions.

DISCRETE STATE SPACES

In both cases, the *state space* was discrete and relatively small in number.

- ▶ For the Markov chain, we gave an example where states correspond to positions in \mathbb{R}^d .
- ▶ A continuous hidden Markov model might perturb the latent state of the Markov chain.
 - ▶ For example, each s_i can be modified by continuous-valued noise, $x_i = s_i + \epsilon_i$.
 - ▶ But $s_{1:T}$ is still a *discrete* Markov chain.



DISCRETE VS CONTINUOUS STATE SPACES

Markov and hidden Markov models both assume a discrete state space.

For Markov models:

- ▶ The state could be a data point x_i (Markov Chain classifier)
- ▶ The state could be an object (object ranking)
- ▶ The state could be the destination of a link (internet search engines)

For hidden Markov models we can simplify complex data:

- ▶ Sequences of discrete data may come from a few discrete distributions.
- ▶ Sequences of continuous data may come from a few distributions.

What if we model the states as continuous too?

CONTINUOUS-STATE MARKOV MODEL

Continuous Markov models extend the state space to a continuous domain. Instead of $s \in \{1, \dots, S\}$, s can take any value in \mathbb{R}^d .

Again compare:

- ▶ Discrete-state Markov models: The states live in a discrete space.
- ▶ Continuous-state Markov models: The states live in a continuous space.

The simplest example is the process

$$s_t = s_{t-1} + \epsilon_t, \quad \epsilon_t \sim N(0, aI).$$

Each successive state is a perturbed version of the current state.

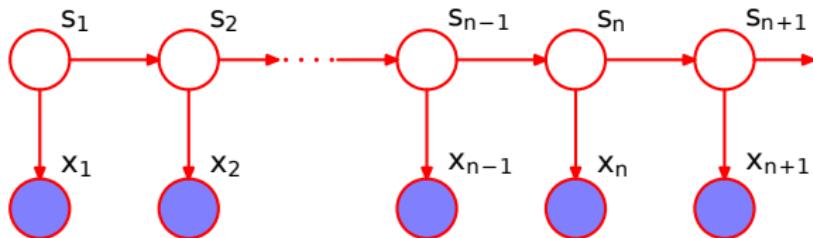
LINEAR GAUSSIAN MARKOV MODEL

The most basic continuous-state version of the hidden Markov model is called a *linear Gaussian Markov model* (also called the *Kalman filter*).

$$\underbrace{s_t = Cs_{t-1} + \epsilon_{t-1}}_{\text{latent process}}, \quad \underbrace{x_t = Ds_t + \varepsilon_t}_{\text{observed process}}$$

- ▶ $s_t \in \mathbb{R}^p$ is a continuous-state latent (unobserved) Markov process
- ▶ $x_t \in \mathbb{R}^d$ is a continuous-valued observation
- ▶ The process noise $\epsilon_t \sim N(0, Q)$
- ▶ The measurement noise $\varepsilon_t \sim N(0, V)$

EXAMPLE APPLICATIONS



Difference from HMM: s_t and x_t are *both* from continuous distributions.

The linear Gaussian Markov model (and its variants) has many applications.

- ▶ Tracking moving objects
- ▶ Automatic control systems
- ▶ Economics and finance (e.g., stock modeling)
- ▶ etc.

EXAMPLE: TRACKING

We get (very) noisy measurements of an object's position in time, $x_t \in \mathbb{R}^2$.

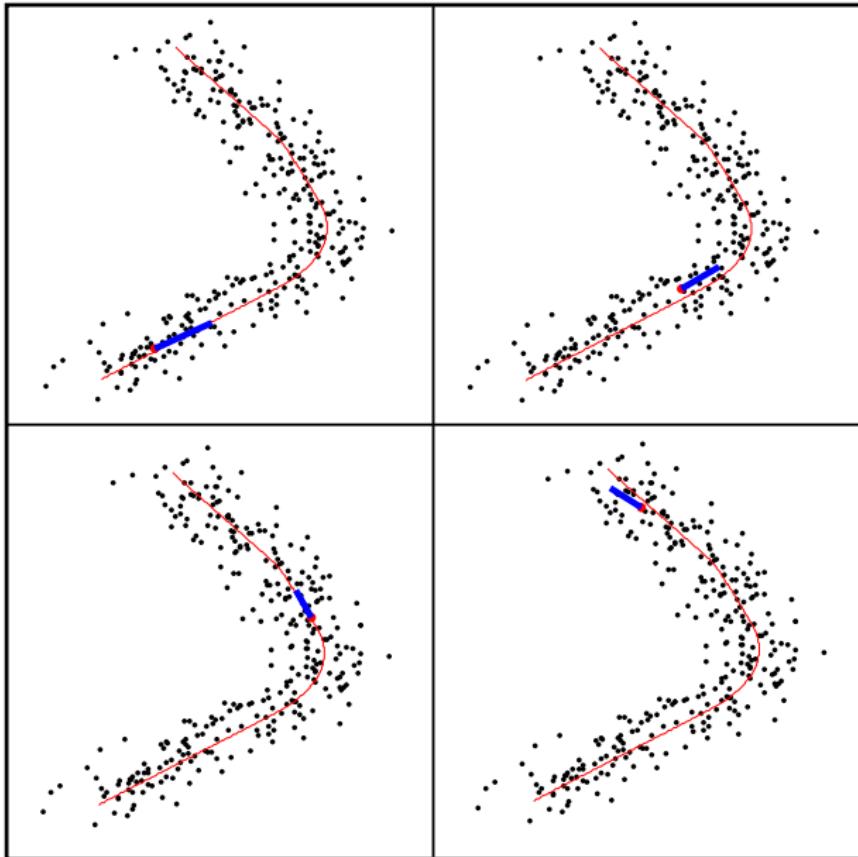
The time-varying state vector is $s = [\text{pos}_1 \text{ vel}_1 \text{ accel}_1 \text{ pos}_2 \text{ vel}_2 \text{ accel}_2]^T$.

Motivated by the underlying physics, we model this as:

$$s_{t+1} = \underbrace{\begin{bmatrix} 1 & \Delta t & \frac{1}{2}(\Delta t)^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & e^{-\alpha \Delta t} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{1}{2}(\Delta t)^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & e^{-\alpha \Delta t} \end{bmatrix}}_{\equiv C} s_t + \epsilon_t$$
$$x_{t+1} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}}_{\equiv D} s_{t+1} + \varepsilon_{t+1}$$

Therefore, s_t not only approximates where the target is, but where it's going.

EXAMPLE: TRACKING



THE LEARNING PROBLEM

As with the hidden Markov model, we're given the sequence (x_1, x_2, x_3, \dots) , where each $x \in \mathbb{R}^d$. The goal is to learn state sequence (s_1, s_2, s_3, \dots) .

All distributions are Gaussian,

$$p(s_{t+1} = s|s_t) = N(Cs_t, Q), \quad p(x_t = x|s_t) = N(Ds_t, V).$$

Notice that with the discrete HMM we wanted to learn π, A and B , where

- ▶ π is the initial state distribution
- ▶ A is the transition matrix among the discrete set of states
- ▶ B contains the state-dependent distributions on discrete-valued data

The situation here is very different.

THE LEARNING PROBLEM

No “B” to learn: In the linear Gaussian Markov model, each state is unique and so the distribution on x_t is different for each t .

No “A” to learn: In addition, each state transition is to a brand new state, so each s_t has its own unique probability distribution.

What we can learn are the two posterior distributions.

1. $p(s_t|x_1, \dots, x_t)$: A distribution on the current state given the past.
 2. $p(s_t|x_1, \dots, x_T)$: A distribution on each latent state in the sequence
-
- ▶ #1: Kalman *filtering* problem. We'll focus on this one today.
 - ▶ #2: Kalman *smoothing* problem. Requires extra step (not discussed).

THE KALMAN FILTER

Goal: Learn the sequence of distributions $p(s_t|x_1, \dots, x_t)$ given a sequence of data (x_1, x_2, x_3, \dots) and the model

$$s_{t+1} | s_t \sim N(Cs_t, Q), \quad x_t | s_t \sim N(Ds_t, V).$$

This is the (linear) Kalman filtering problem and is often used for tracking.

Setup: We can use Bayes rule to write

$$p(s_t|x_1, \dots, x_t) \propto p(x_t|s_t) p(s_t|x_1, \dots, x_{t-1})$$

and represent the prior as a marginal distribution

$$p(s_t|x_1, \dots, x_{t-1}) = \int p(s_t|s_{t-1}) p(s_{t-1}|x_1, \dots, x_{t-1}) ds_{t-1}$$

THE KALMAN FILTER

We've decomposed the problem into parts that we do and don't know (yet)

$$p(s_t|x_1, \dots, x_t) \propto \underbrace{p(x_t|s_t)}_{N(Ds_t, V)} \int \underbrace{p(s_t|s_{t-1})}_{N(Cs_{t-1}, Q)} \underbrace{p(s_{t-1}|x_1, \dots, x_{t-1})}_{?} ds_{t-1}$$

Observations and considerations:

1. The left is the posterior on s_t and the right has the posterior on s_{t-1} .
2. We want the integral to be in closed form and a known distribution.
3. We want the prior and likelihood terms to lead to a known posterior.
4. We want future calculations, e.g. for s_{t+1} , to be easy.

We will see how choosing the Gaussian distribution makes this all work.

THE KALMAN FILTER: STEP 1

Calculate the marginal for prior distribution

Hypothesize (temporarily) that the unknown distribution is Gaussian,

$$p(s_t|x_1, \dots, x_t) \propto \underbrace{p(x_t|s_t)}_{N(Ds_t, V)} \int \underbrace{p(s_t|s_{t-1})}_{N(Cs_{t-1}, Q)} \underbrace{p(s_{t-1}|x_1, \dots, x_{t-1})}_{N(\mu, \Sigma)} ds_{t-1}$$

by hypothesis

A property of the Gaussian is that marginals are still Gaussian,

$$\int N(s_t|Cs_{t-1}, Q)N(s_{t-1}|\mu, \Sigma)ds_{t-1} = N(s_t|C\mu, Q + C\Sigma C^T).$$

We know C and Q (by design) and μ and Σ (by hypothesis).

THE KALMAN FILTER: STEP 2

Calculate the posterior

We plug in the marginal distribution for the prior and see that

$$p(s_t|x_1, \dots, x_t) \propto N(x_t|Ds_t, V) N(s_t|C\mu, Q + C\Sigma C^T).$$

Though the parameters look complicated, the posterior is just a Gaussian

$$p(s_t|x_1, \dots, x_t) = N(s_t|\mu', \Sigma')$$

$$\Sigma' = [(Q + C\Sigma C^T)^{-1} + D^T V^{-1} D]^{-1}$$

$$\mu' = \Sigma' (D^T V^{-1} x_t + (Q + C\Sigma C^T)^{-1} C\mu)$$

We can plug the relevant values into these two equations.

ADDRESSING THE GAUSSIAN ASSUMPTION

By making the assumption of a Gaussian in the prior,

$$p(s_t|x_1, \dots, x_t) \propto \underbrace{p(x_t|s_t)}_{N(x_t|Ds_t, V)} \int \underbrace{p(s_t|s_{t-1})}_{N(s_t|Cs_{t-1}, Q)} \underbrace{p(s_{t-1}|x_1, \dots, x_{t-1})}_{N(\mu, \Sigma)} ds_{t-1}$$

by hypothesis

we found that the posterior is also Gaussian with a new mean and covariance.

- ▶ We therefore only need to define a Gaussian prior on the first state to keep things moving forward. For example,

$$p(s_0) \sim N(0, I).$$

Once this is done, all future calculations are in closed form.

KALMAN FILTER: ONE FINAL QUANTITY

Making predictions

We know how to update the sequence of state posterior distributions

$$p(s_t|x_1, \dots, x_t).$$

What about predicting x_{t+1} ?

$$\begin{aligned} p(x_{t+1}|x_1, \dots, x_t) &= \int p(x_{t+1}|s_{t+1})p(s_{t+1}|x_1, \dots, x_t)ds_{t+1} \\ &= \underbrace{\int p(x_{t+1}|s_{t+1})}_{N(x_{t+1}|Ds_{t+1}, V)} \underbrace{\int p(s_{t+1}|s_t)}_{N(s_{t+1}|Cs_t, Q)} \underbrace{p(s_t|x_1, \dots, x_t)}_{N(s_t|\mu', \Sigma')} ds_t ds_{t+1} \end{aligned}$$

Again, Gaussians are nice because these operations stay Gaussian.

This is a multivariate Gaussian that looks even more complicated than the previous one (omitted). Simply perform the previous integral twice.

ALGORITHM: KALMAN FILTERING

The Kalman filtering algorithm can be run in real time.

0. Set the initial state distribution $p(s_0) = N(0, I)$
1. Prior to observing each new $x_t \in \mathbb{R}^d$ predict

$$x_t \sim N(\mu_t^x, \Sigma_t^x) \quad (\text{using previously discussed marginalization})$$

2. After observing each new $x_t \in \mathbb{R}^d$ update

$$p(s_t | x_1, \dots, x_t) = N(\mu_t^s, \Sigma_t^s) \quad (\text{using equations on previous slide})$$

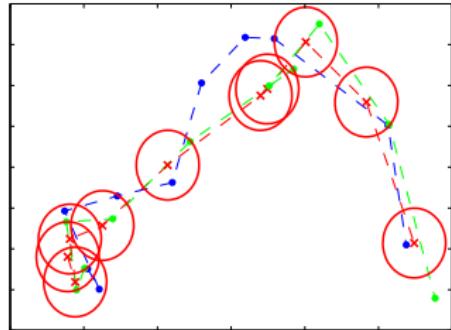
EXAMPLE

Learning state trajectory

Green: True trajectory

Blue: Observed trajectory

Red: State distribution



Intuitions about what this is doing:

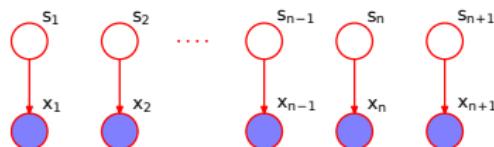
- ▶ In the prior distribution notice that we add Q to the covariance,

$$p(s_t|x_1, \dots, x_{t-1}) = N(s_t|C\mu, Q + C\Sigma C^T).$$

This allows the state s_t to “drift” away from s_{t-1} .

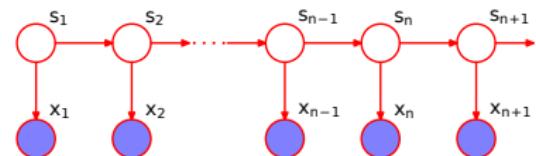
- ▶ In the posterior $p(s_t|x_1, \dots, x_t)$, x_t “pulls” the distribution away.

SOME FINAL MODEL COMPARISONS



Gaussian mixture model

- ▶ $s_t \sim \text{Discrete}(\pi)$
- ▶ $x_t | s_t \sim N(\mu_{s_t}, \Sigma_{s_t})$

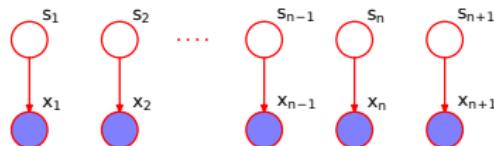


Continuous hidden Markov model

- ▶ $s_t | s_{t-1} \sim \text{Discrete}(A_{s_{t-1}})$
- ▶ $x_t | s_t \sim N(\mu_{s_t}, \Sigma_{s_t})$

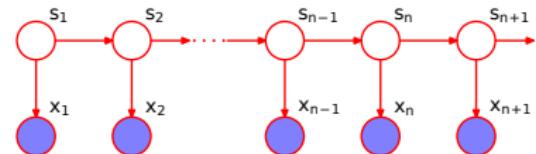
We saw how the transition from GMM \rightarrow HMM involves using a Markov chain to index the distribution on clusters.

SOME FINAL MODEL COMPARISONS



Probabilistic PCA

- ▶ $s_t \sim N(0, Q)$
- ▶ $x_t | s_t \sim N(Ds_t, V)$



Linear Gaussian Markov model

- ▶ $s_t | s_{t-1} \sim N(Cs_{t-1}, Q)$
- ▶ $x_t | s_t \sim N(Ds_t, V)$

There is a similar relationship between probabilistic PCA and the Kalman filter. (Probabilistic PCA also learns D , while the Kalman filter doesn't).

EXTENSIONS

There are a variety of extensions to this framework. The equations in the corresponding algorithms would all look familiar given our discussion.

Extended Kalman filter: *Nonlinear Kalman filters* use nonlinear function of the state, $h(s_t)$. The EKF approximates $h(s_t) \approx h(z) + \nabla h(z)(s_t - z)$

$$s_{t+1} | s_t \sim N(Ds_t, Q), \quad x_t | s_t \sim N(h(s_t), V).$$

Continuous time: Sometimes the time between observations varies. Let Δ_t be the time between observation x_t and x_{t+1} , then model

$$s_{t+1} | s_t \sim N(s_t, \Delta_t Q), \quad x_t | s_t \sim N(Ds_t, V).$$

Adding control: In dynamic models, we can add control to the state using a vector u_t whose values we choose (e.g., thrusters).

$$s_{t+1} | s_t \sim N(Cs_t + Gu_t, Q), \quad x_t | s_t \sim N(Ds_t, V).$$

ColumbiaX: Machine Learning

Lecture 23

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

ASSOCIATION ANALYSIS

SETUP

Many businesses have massive amounts of customer purchasing data.

- ▶ Amazon has your order history
- ▶ A grocery store knows objects purchased in each transaction
- ▶ Other retailers have data on purchases in their stores

Using this data, we may want to find sub-groups of products that tend to co-occur in purchasing or viewing behavior.

- ▶ Retailers can use this to cross-promote products through “deals”
- ▶ Grocery stores can use this to strategically place items
- ▶ Online retailers can use this to recommend content
- ▶ This is more general than finding purchasing patterns

MARKET BASKET ANALYSIS

Association analysis is the task of understanding these patterns.

For example consider the following “market baskets” of five customers.

<i>TID</i>	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Using such data, we want to analyze patterns of co-occurrence within it. We can use these patterns to define *association rules*. For example,

$$\{\text{diapers}\} \Rightarrow \{\text{beer}\}$$

ASSOCIATION ANALYSIS AND RULES

Imagine we have:

- ▶ p different objects indexed by $\{1, \dots, p\}$
- ▶ A collection of subsets of these objects $X_n \subset \{1, \dots, p\}$. Think of X_n as the index of things purchased by customer $n = 1, \dots, N$.

Association analysis: Find subsets of objects that often appear together. For example, if $\mathcal{K} \subset \{1, \dots, p\}$ indexes objects that frequently co-occur, then

$$P(\mathcal{K}) = \frac{\#\{n \text{ such that } \mathcal{K} \subseteq X_n\}}{N} \text{ is large relatively speaking}$$

Example: $\mathcal{K} = \{\text{peanut_butter}, \text{ jelly}, \text{ bread}\}$

Association rules: Learn correlations. Let A and B be disjoint sets. Then $A \Rightarrow B$ means purchasing A increases likelihood of also purchasing B .

Example: $\{\text{peanut_butter}, \text{ jelly}\} \Rightarrow \{\text{bread}\}$

PROCESSING THE BASKET

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Figure: An example of 5 baskets.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Figure: A binary representation of these 5 baskets for analysis.

PROCESSING THE BASKET

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Want to find subsets that occur with probability above some threshold.

For example, does {bread, milk} occur relatively frequently?

- ▶ Go to each of the 5 baskets and count the number that contain both.
- ▶ Divide this number by 5 to get the frequency.
- ▶ Aside: Notice that the basket might have more items in it.

When $N = 5$ and $p = 6$ as in this case, we can easily check every possible combination. However, real problems might have $N \approx 10^8$ and $p \approx 10^4$.

SOME COMBINATORICS

Some combinatorial analysis will show that brute-force search isn't possible.

Q: How many different subsets $\mathcal{K} \subseteq \{1, \dots, p\}$ are there?

A: Each subset can be represented by a binary indicator vector of length p .
The total number of possible vectors is 2^p .

Q: Nobody will have a basket with every item in it, so we shouldn't check every combination. How about if we only check up to k items?

A: The number of sets of size k picked from p items is $\binom{p}{k} = \frac{p!}{k!(p-k)!}$. For example, if $p = 10^4$ and $k = 5$, then $\binom{p}{k} \approx 10^{18}$.

Takeaway: Though the problem only requires counting, we need an algorithm that can tell us which \mathcal{K} we should count and which we can ignore.

QUANTITIES OF INTEREST

Before we find an efficient counting algorithm, what do we want to count?

- ▶ Again, let $\mathcal{K} \subset \{1, \dots, p\}$ and $A, B \subset \mathcal{K}$, where $A \cup B = \mathcal{K}, A \cap B = \emptyset$.

We're interested in the following empirically-calculated probabilities:

1. $P(\mathcal{K}) = P(A, B)$: The *prevalence* (or support) of items in set \mathcal{K} . We want to find which combinations co-occur often.
2. $P(B|A) = \frac{P(\mathcal{K})}{P(A)}$: The *confidence* that B appears in the basket given A is in the basket. We use this to define a *rule* $A \Rightarrow B$.
3. $L(A, B) = \frac{P(A, B)}{P(A)P(B)} = \frac{P(B|A)}{P(B)}$: The *lift* of the rule $A \Rightarrow B$. This is a measure of how much *more* confident we are in B given that we see A .

EXAMPLE

For example, let

$$\mathcal{K} = \{\text{peanut_butter}, \text{ jelly}, \text{ bread}\},$$

$$A = \{\text{peanut_butter}, \text{ jelly}\}, B = \{\text{bread}\}$$

- ▶ A *prevalence* of 0.03 means that peanut_butter, jelly and bread appeared together in 3% of baskets.
- ▶ A *confidence* of 0.82 means that when both peanut_butter and jelly were purchased, 82% of the time bread was also purchased.
- ▶ A *lift* of 1.95 means that it's 1.95 more probable that bread will be purchased given that peanut_butter and jelly were purchased.

APRIORI ALGORITHM

The goal of the **Apriori algorithm** is to quickly find all of the subsets $\mathcal{K} \subset \{1, \dots, p\}$ that have probability greater than a predefined threshold t .

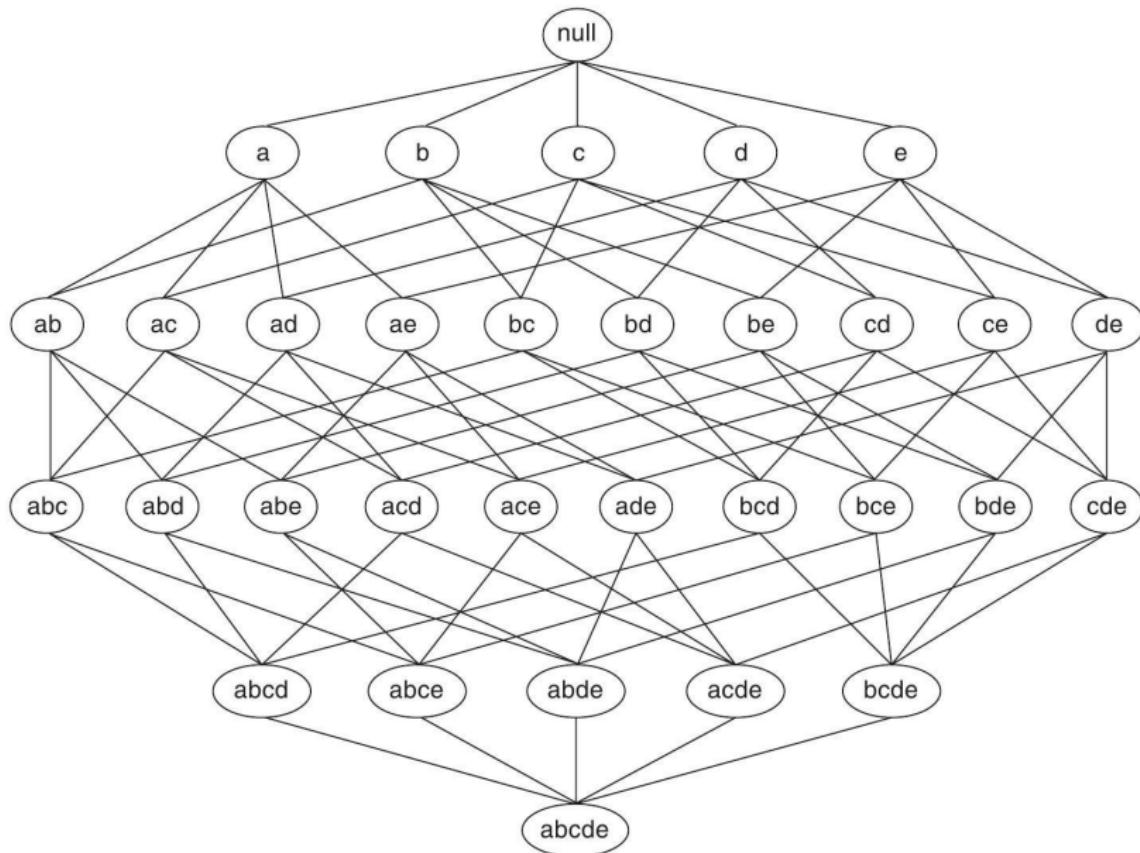
- ▶ Such a \mathcal{K} will contain items that appear in at least $N \cdot t$ of the N baskets.
- ▶ A small fraction of such \mathcal{K} should exist out of the 2^p possibilities.

Apriori uses properties about $P(\mathcal{K})$ to reduce the number of subsets that need to be checked to a small fraction of all 2^p sets.

- ▶ It starts with \mathcal{K} containing 1 item. It then moves to 2 items, etc.
- ▶ Sets of size $k - 1$ that “survive” help determine sets of size k to check.
- ▶ Important: Apriori finds *every* set \mathcal{K} such that $P(\mathcal{K}) > t$.

Next slide: The structure of the problem can be organized in a lattice.

LATTICE REPRESENTATION



FREQUENCY DEPENDENCE

We can use two properties to develop an algorithm for efficiently counting.

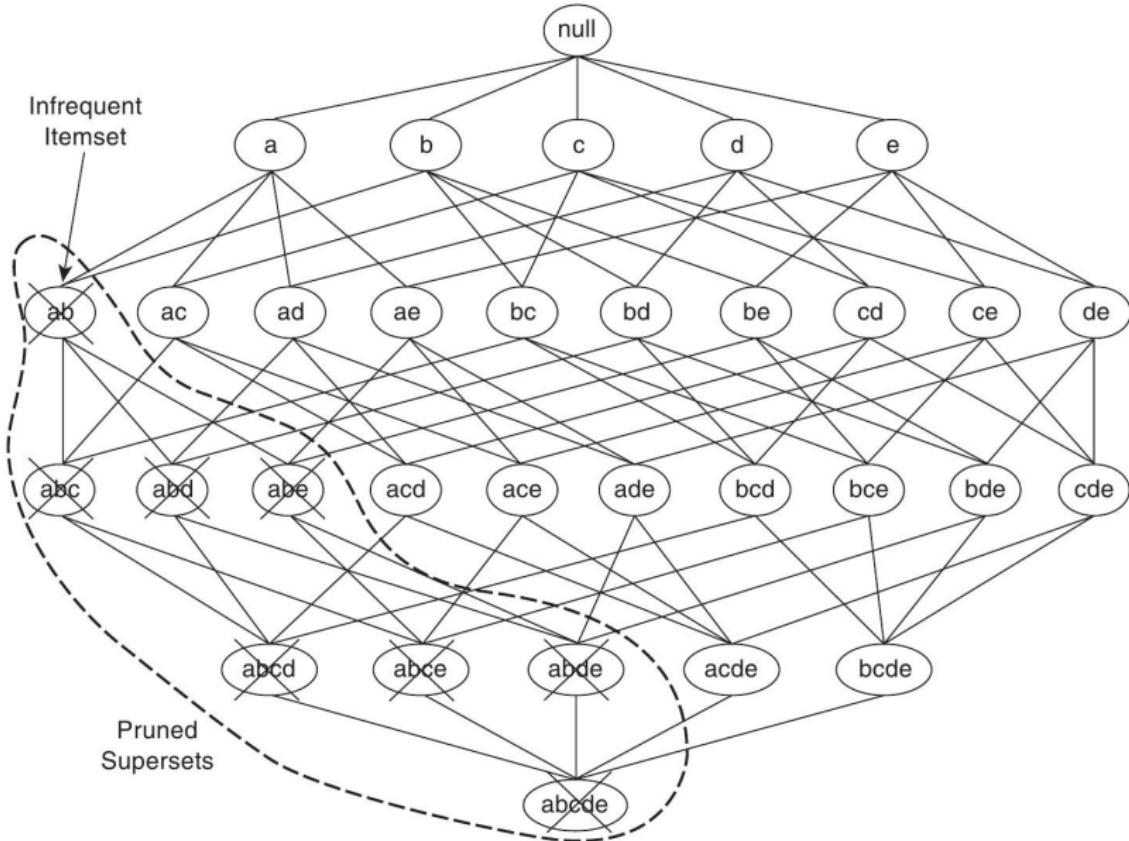
1. If the set \mathcal{K} is not big enough, then $\mathcal{K}' = \mathcal{K} \cup A$ with $A \subset \{1, \dots, p\}$ is not big enough. In other words: $P(\mathcal{K}) < t$ implies $P(\mathcal{K}') < t$

e.g., Let $\mathcal{K} = \{a, b\}$. If these items appear together in x baskets, then the set of items $\mathcal{K}' = \{a, b, c\}$ appears in $\leq x$ baskets since $\mathcal{K} \subset \mathcal{K}'$.

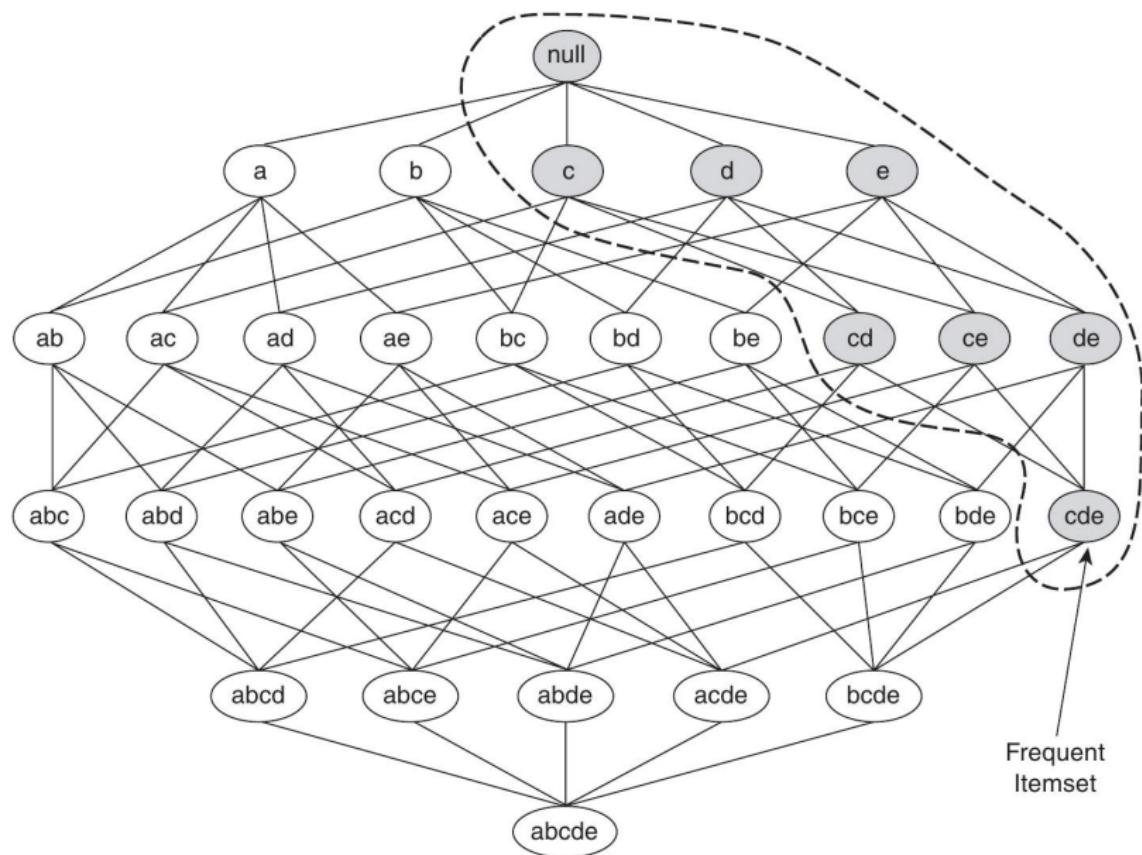
Mathematically: $P(\mathcal{K}') = P(\mathcal{K}, A) = P(A|\mathcal{K})P(\mathcal{K}) \leq P(\mathcal{K}) < t$

2. By the converse, if $P(\mathcal{K}) > t$ and $A \subset \mathcal{K}$, then $P(A) > P(\mathcal{K}) > t$.

FREQUENCY DEPENDENCE: PROPERTY 1



FREQUENCY DEPENDENCE: PROPERTY 2



APRIORI ALGORITHM (ONE VERSION)

Here is a basic version of the algorithm. It can be improved in clever ways.

Apriori algorithm

Set a threshold $N \cdot t$, where $0 < t < 1$ (but relatively small).

1. $|\mathcal{K}| = 1$: Check each object and keep those that appear in $\geq N \cdot t$ baskets.
 2. $|\mathcal{K}| = 2$: Check all pairs of objects that survived Step 1 and keep the sets that appear in $\geq N \cdot t$ baskets.
⋮
 - k. $|\mathcal{K}| = k$: Using all sets of size $k - 1$ that appear in $\geq N \cdot t$ baskets,
 - ▶ Increment each set with an object surviving Step 1 not already in the set.
 - ▶ Keep all sets that appear in $\geq N \cdot t$ baskets
-

It should be clear that as k increases, we can hope that the number of sets that survive decrease. At a certain $k < p$, no sets will survive and we're done.

MORE CONSIDERATIONS

1. We can show that this algorithm returns *every* set \mathcal{K} for which $P(\mathcal{K}) > t$.
 - ▶ Imagine we know every set of size $k - 1$ for which $P(\mathcal{K}) > t$. Then every potential set of size k that could have $P(\mathcal{K}) > t$ will be checked.
 - e.g. Let $k = 3$: The set $\{a, b, c\}$ appears in $> N \cdot t$ baskets. Will we check it?
 - Known:** $\{a, b\}$ and $\{c\}$ must appear in $> N \cdot t$ baskets.
 - Assumption:** We've found $\mathcal{K} = \{a, b\}$ as a set satisfying $P(\mathcal{K}) > t$.
 - Apriori algorithm:** We know $P(\{c\}) > t$ and so will check $\{a, b\} \cup \{c\}$.
 - Induction:** We have all $|\mathcal{K}| = 1$ by brute-force search (start induction).
2. As written, this can lead to duplicate sets for checking, e.g., $\{a, b\} \cup \{c\}$ and $\{a, c\} \cup \{b\}$. Indexing methods can ensure we create $\{a, b, c\}$ once.
3. For each proposed \mathcal{K} , should we iterate through each basket for checking? There are tricks to make this faster that takes structure into account.

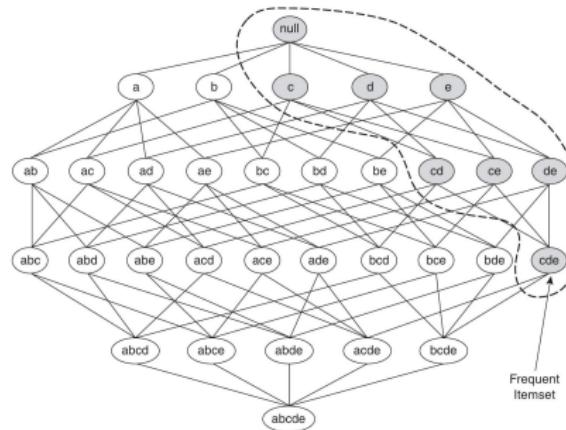
FINDING ASSOCIATION RULES

We've found all \mathcal{K} such that

$$P(\mathcal{K}) > t.$$

Now we want to find association rules.

These are of the form $P(A|B) > t_2$
where we split \mathcal{K} into subsets A and B .



Notice:

1. $P(A|B) = \frac{P(\mathcal{K})}{P(B)}$.
2. If $P(\mathcal{K}) > t$ and A and B partition \mathcal{K} , then $P(A) > t$ and $P(B) > t$.
3. Since Apriori found all \mathcal{K} such that $P(\mathcal{K}) > t$, it found $P(A)$ and $P(B)$, so we can calculate $P(A|B)$ without counting again.

EXAMPLE

Feature	Demographic	# Values	Type
1	Sex	2	Categorical
2	Marital status	5	Categorical
3	Age	7	Ordinal
4	Education	6	Ordinal
5	Occupation	9	Categorical
6	Income	9	Ordinal
7	Years in Bay Area	5	Ordinal
8	Dual incomes	3	Categorical
9	Number in household	9	Ordinal
10	Number of children	9	Ordinal
11	Householder status	3	Categorical
12	Type of home	5	Categorical
13	Ethnic classification	8	Categorical
14	Language in home	3	Categorical

Data

$N = 6876$ questionnaires

14 questions coded into $p = 50$ items

For example:

- ▶ ordinal (2 items): Pick the item based on value being \leqslant median
- ▶ categorical: item = category
 x categories $\rightarrow x$ items

- ▶ Based on the item encoding, it's clear that no "basket" can have every item.
- ▶ We see that association analysis extends to more than consumer analysis.

EXAMPLE

Association rule 1: Support 13.4%, confidence 80.8%, and lift 2.13.

$$\left[\begin{array}{l} \text{language in home} = \textit{English} \\ \text{householder status} = \textit{own} \\ \text{occupation} = \{\textit{professional}/\textit{managerial}\} \end{array} \right] \Downarrow \text{income} \geq \$40,000$$

Association rule 2: Support 26.5%, confidence 82.8% and lift 2.15.

$$\left[\begin{array}{l} \text{language in home} = \textit{English} \\ \text{income} < \$40,000 \\ \text{marital status} = \textit{not married} \\ \text{number of children} = 0 \end{array} \right] \Downarrow$$

$\text{education} \notin \{\textit{college graduate}, \textit{graduate study}\}$

ColumbiaX: Machine Learning

Lecture 24

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute
Columbia University

MODEL SELECTION

MODEL SELECTION

The model selection problem

We've seen how often model parameters need to be set in advance and discussed how this can be done using cross-validation.

Another type of model selection problem is learning model order.

Model order: The complexity of a class of models

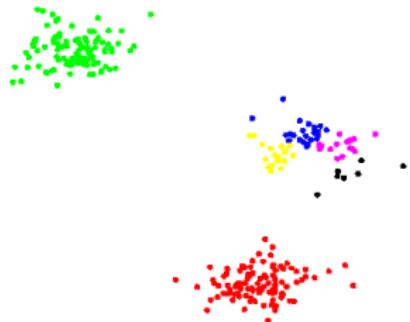
- ▶ Gaussian mixture model: How many Gaussians?
- ▶ Matrix factorization: What rank?
- ▶ Hidden Markov models: How many states?

In each of these problems, we can't simply look at the log-likelihood because a more complex model can always fit the data better.

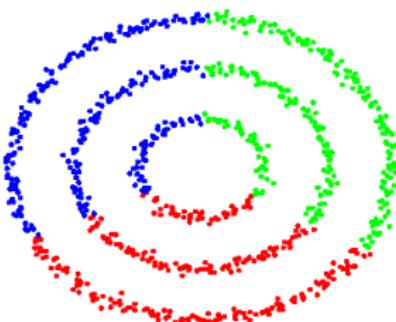
MODEL SELECTION

Model Order

We will discuss two methods for selecting an “appropriate” complexity of the model. This assumes a good model type was chosen to begin with.



(a) Inappropriate model order.



(b) Inappropriate model type.

EXAMPLE: MAXIMUM LIKELIHOOD

Notation

We write \mathcal{L} for the log-likelihood of a parameter under a model $p(x|\theta)$:

$$x_i \stackrel{iid}{\sim} p(x|\theta) \iff \mathcal{L} = \sum_{i=1}^N \log p(x_i|\theta)$$

The maximum likelihood solution is: $\theta_{ML} = \arg \max_{\theta} \mathcal{L}$.

Example: How many clusters? (wrong way)

The parameters θ could be those of a GMM. We could find θ_{ML} for different numbers of clusters and pick the one with the largest \mathcal{L} .

Problem: We can perfectly fit the data by putting each observation in its own cluster. Then shrink the variance of each Gaussian to zero.

NUMBER OF PARAMETERS

The general problem

- ▶ Models with more degrees of freedom are more prone to overfitting.
- ▶ The degrees of freedom is roughly the number of scalar parameters, K .
- ▶ By increasing K (done by increasing #clusters, rank, #states, etc.) the model can add more degrees of freedom.

Some common solutions

- ▶ **Stability:** Bootstrap sample the data, learn a model, calculate the likelihood on the original data set. Repeat and pick the best model.
- ▶ **Bayesian nonparametric methods:** Each possible value of K is assigned a prior probability. The posterior learns the best K .
- ▶ **Penalization approaches:** A penalty term makes adding parameters expensive. Must be overcome by a greater improvement in likelihood.

PENALIZING MODEL COMPLEXITY

General form

Define a *penalty function* on the number of model parameters. Instead of maximizing \mathcal{L} , minimize $-\mathcal{L}$ and add the defined penalty.

Two popular penalties are:

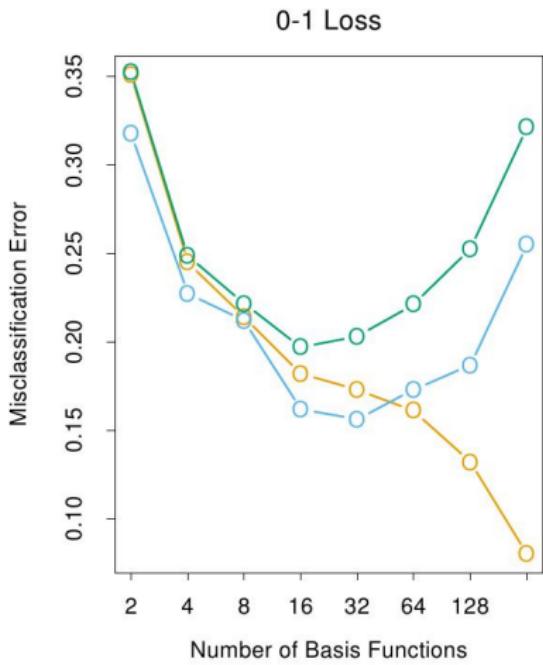
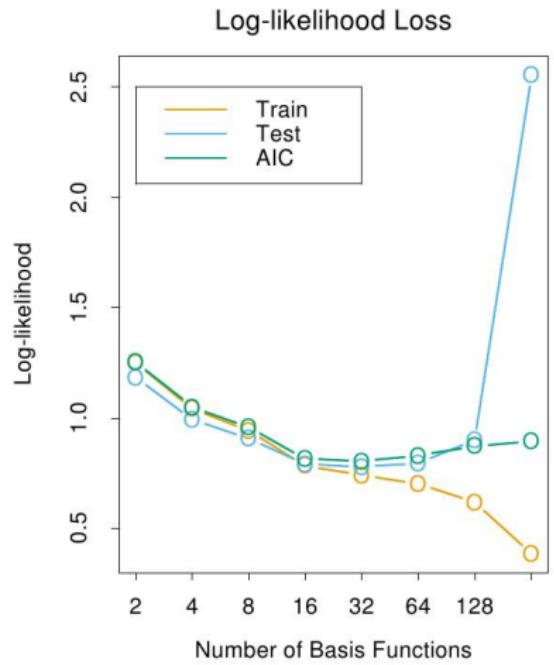
- ▶ **Akaike information criterion (AIC):** $-\mathcal{L} + K$
- ▶ **Bayesian information criterion (BIC):** $-\mathcal{L} + \frac{1}{2}K \ln N$

When $\frac{1}{2} \ln N > 1$, BIC encourages a simpler model (happens when $N \geq 8$).

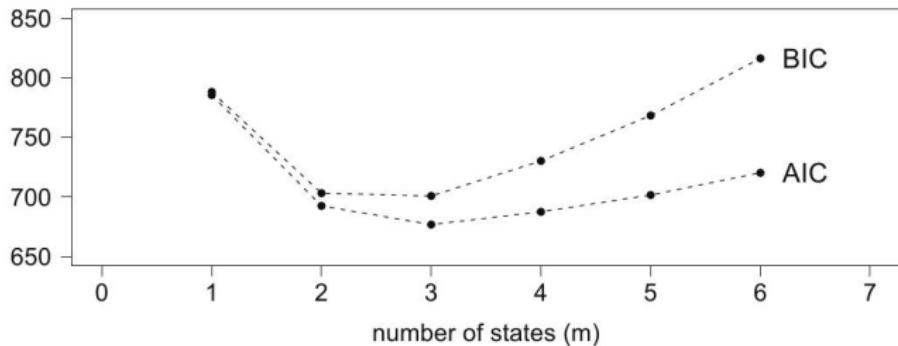
Example: For NMF with an $M_1 \times M_2$ matrix and rank R factorization,

$$\text{AIC} \rightarrow (M_1 + M_2)R, \quad \text{BIC} \rightarrow \frac{1}{2}(M_1 + M_2)R \ln(M_1 M_2)$$

EXAMPLE OF AIC OUTPUT



EXAMPLE: AIC vs BIC ON HMM



model	$-\log L$	AIC	BIC
'1-state HM'	391.9189	785.8	788.5
2-state HM	342.3183	692.6	703.3
3-state HM	329.4603	676.9	701.0
4-state HM	327.8316	687.7	730.4
5-state HM	325.9000	701.8	768.6
6-state HM	324.2270	720.5	816.7
indep. mixture (2)	360.3690	726.7	734.8
indep. mixture (3)	356.8489	723.7	737.1
indep. mixture (4)	356.7337	727.5	746.2

Notice:

- ▶ Likelihood is always improving
- ▶ Only compare location of AIC and BIC minima, not the values.

DERIVATION OF BIC

AIC AND BIC

Recall the two penalties:

- ▶ **Akaike information criterion (AIC):** $-\mathcal{L} + K$
- ▶ **Bayesian information criterion (BIC):** $-\mathcal{L} + \frac{1}{2}K \ln N$

Algorithmically, there is no extra work required:

1. Find the ML solution of the selected models and calculate \mathcal{L} .
2. Add the AIC or BIC penalty to get a score useful for picking a model.

Q: Where do these penalties come from? Currently they seem arbitrary.

A: We will derive BIC next. AIC also has a theoretical motivation, but we will not discuss that derivation.

DERIVING THE BIC

Imagine we have r candidate models, $\mathcal{M}_1, \dots, \mathcal{M}_r$. For example, r HMMs each having a different number of states.

We also have data $\mathcal{D} = \{x_1, \dots, x_N\}$. We want the posterior of each \mathcal{M}_i .

$$p(\mathcal{M}_i | \mathcal{D}) = \frac{p(\mathcal{D} | \mathcal{M}_i)p(\mathcal{M}_i)}{\sum_j p(\mathcal{D} | \mathcal{M}_j)p(\mathcal{M}_j)}$$

If we assume a uniform prior distribution on models, then because the denominator is constant in \mathcal{M}_i , we can pick

$$\mathcal{M} = \arg \max_{\mathcal{M}_i} \ln p(\mathcal{D} | \mathcal{M}_i) = \int \ln p(\mathcal{D} | \theta, \mathcal{M}_i)p(\theta | \mathcal{M}_i)d\theta$$

We're choosing the model with the largest *marginal likelihood* of the data by integrating out all parameters of the model. This is usually not solvable.

DERIVING THE BIC

We will see how the BIC arises from the approximation,

$$\mathcal{M} = \arg \max_{\mathcal{M}_i} \ln p(\mathcal{D}|\mathcal{M}_i) \approx \arg \max_{\mathcal{M}_i} \ln p(\mathcal{D}|\theta_{\text{ML}}, \mathcal{M}_i) - \frac{1}{2}K \ln N$$

Step 1: Recognize that the difficulty is with the integral

$$\ln p(\mathcal{D}|\mathcal{M}_i) = \ln \int p(\mathcal{D}|\theta)p(\theta)d\theta.$$

\mathcal{M}_i determines $p(\mathcal{D}|\theta)$, $p(\theta)$ —we will suppress this conditioning.

Step 2: Approximate this integral using a second-order Taylor expansion.

DERIVING THE BIC

1. We want to calculate:

$$\ln p(\mathcal{D}|\mathcal{M}) = \ln \int p(\mathcal{D}|\theta)p(\theta)d\theta = \ln \int \exp\{\ln p(\mathcal{D}|\theta)\}p(\theta)d\theta$$

2. We use a second-order Taylor expansion of $\ln p(\mathcal{D}|\theta)$ at the point θ_{ML} ,

$$\begin{aligned}\ln p(\mathcal{D}|\theta) &\approx \ln p(\mathcal{D}|\theta_{ML}) + (\theta - \theta_{ML})^T \underbrace{\nabla \ln p(\mathcal{D}|\theta_{ML})}_{= 0} \\ &+ \frac{1}{2}(\theta - \theta_{ML})^T \underbrace{\nabla^2 \ln p(\mathcal{D}|\theta_{ML})(\theta - \theta_{ML})}_{= -\mathcal{J}(\theta_{ML})}\end{aligned}$$

3. Approximate $p(\theta)$ as uniform and plug this approximation back in,

$$\ln p(\mathcal{D}|\mathcal{M}) \approx \ln p(\mathcal{D}|\theta_{ML}) + \ln \int \exp \left\{ -\frac{1}{2}(\theta - \theta_{ML})^T \mathcal{J}(\theta_{ML})(\theta - \theta_{ML}) \right\} d\theta$$

DERIVING THE BIC

Observation: The integral is the normalizing constant of a Gaussian,

$$\int \exp \left\{ -\frac{1}{2}(\theta - \theta_{\text{ML}})^T \mathcal{J}(\theta_{\text{ML}})(\theta - \theta_{\text{ML}}) \right\} d\theta = \left(\frac{2\pi}{|\mathcal{J}(\theta_{\text{ML}})|} \right)^{K/2}$$

Remember the definition that

$$-\mathcal{J}(\theta_{\text{ML}}) = \nabla^2 \ln p(\mathcal{D} | \theta_{\text{ML}}) \stackrel{(a)}{=} \underbrace{N \sum_{i=1}^N \frac{1}{N} \nabla^2 \ln p(x_i | \theta_{\text{ML}})}_{\text{converges as } N \text{ increases}}$$

(a) is by the i.i.d. model assumption made at the beginning of the lecture.

DERIVING THE BIC

4. Plugging this in,

$$\ln p(\mathcal{D}|\mathcal{M}) \approx \ln p(\mathcal{D}|\theta_{\text{ML}}) + \ln \left(\frac{2\pi}{|\mathcal{J}(\theta_{\text{ML}})|} \right)^{K/2}$$

$$\text{and } |\mathcal{J}(\theta_{\text{ML}})| = N \left| \sum_{i=1}^N \frac{1}{N} \nabla^2 \ln p(x_i|\theta_{\text{ML}}) \right|.$$

Therefore we arrive at the BIC,

$$\ln p(\mathcal{D}|\mathcal{M}) \approx \ln p(\mathcal{D}|\theta_{\text{ML}}) - \frac{1}{2} K \ln N + \underbrace{\text{something not growing with } N}_{O(1) \text{ term, so we ignore it}}$$

SOME NEXT STEPS

ICML SESSIONS (SUBSET)

The International Conference on Machine Learning (ICML) is a major ML conference. Many of the session titles should look familiar:

- ▶ Bayesian Optimization and Gaussian Processes
- ▶ PCA and Subspace Models
- ▶ Supervised Learning
- ▶ Matrix Completion and Graphs
- ▶ Clustering and Nonparametrics
- ▶ Active Learning
- ▶ Clustering
- ▶ Boosting and Ensemble Methods
- ▶ Matrix Factorization I & II
- ▶ Kernel Methods I & II
- ▶ Topic models
- ▶ Time Series and Sequences
- ▶ etc.

ICML SESSIONS (SUBSET)

Other sessions might not look so familiar:

- ▶ Reinforcement Learning I & II
- ▶ Bandits I & II
- ▶ Optimization I, II & III
- ▶ Bayesian nonparametrics I & II
- ▶ Online learning I & II
- ▶ Graphical Models I & II
- ▶ Neural Networks and Deep Learning I & II
- ▶ Metric Learning and Feature Selection
- ▶ etc.

Many of these topics are taught in advanced machine learning courses at Columbia in the CS, Statistics, IEOR and EE departments.