School of Computer Science
Carnegie Mellon

10-701 Introduction to Machine Learning

Logistic Regression

Readings:
Bishop 4.2-4.3
Murphy Ch. 8.1-3, 8.6
Elken (2014) Notes

Matt Gormley
Lecture 5
September 21, 2016

Reminders

- Homework 1:
  - due 9/26/16
- Project Proposal:
  - due 10/3/16
  - start early!

Outline

- **Motivation:**
  - Choosing the right classifier
  - Example: Image Classification
- **Logistic Regression**
  - Background: Hyperplanes
  - Data, Model, Learning, Prediction
  - Log-odds
  - Bernoulli interpretation
  - Maximum Conditional Likelihood Estimation
- **Gradient descent for Logistic Regression**
  - Stochastic Gradient Descent (SGD)
  - Computing the gradient
  - Details (learning rate, finite differences)
- **Logistic Regression and Overfitting**
  - (non-stochastic) Gradient Descent
  - Difference of expectations
- **Newton's Method for Logistic Regression**
  - Taylor Series approximation
  - Hessian matrix
  - Newton's Method
  - Iteratively Reweighted Least Squares
- **Discriminative vs. Generative Classifiers**

Classifiers

Which classification method should we use?

1. The one that gives the best predictions…
   - on the training data
   - on the (unseen) test data
   - on the (held-out) validation data
2. The one that is computationally efficient…
   - during training
   - during classification
3. The most interpretable one…
   - in terms of its parameters
   - as a model
4. The one that is easiest to implement…
   - for learning
   - for classification

Classifiers

Which classification method should we use?

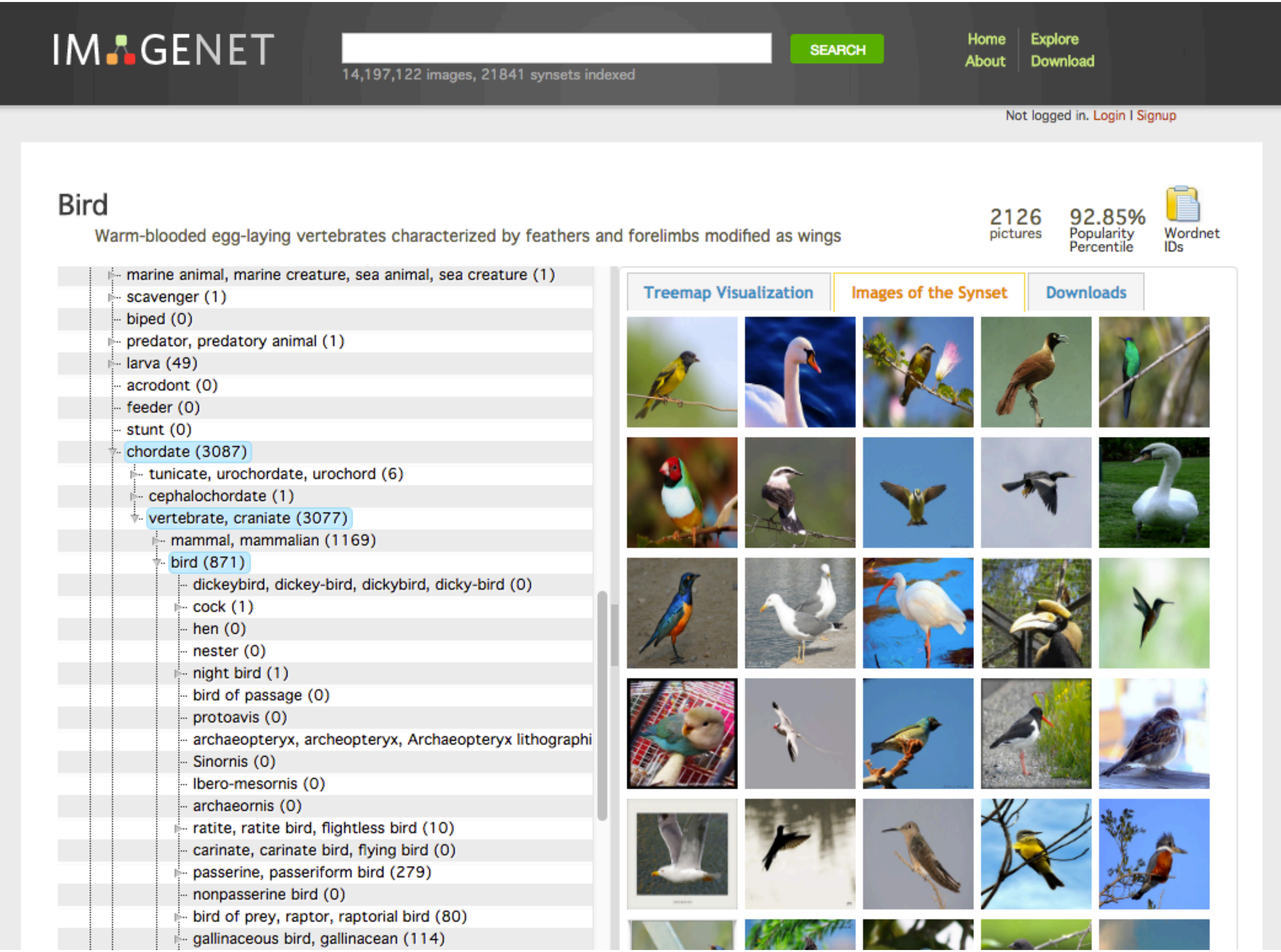
Naïve Bayes defined a generative model $p(\boldsymbol{x}, y)$ of the features $\boldsymbol{x}$ and the class y.

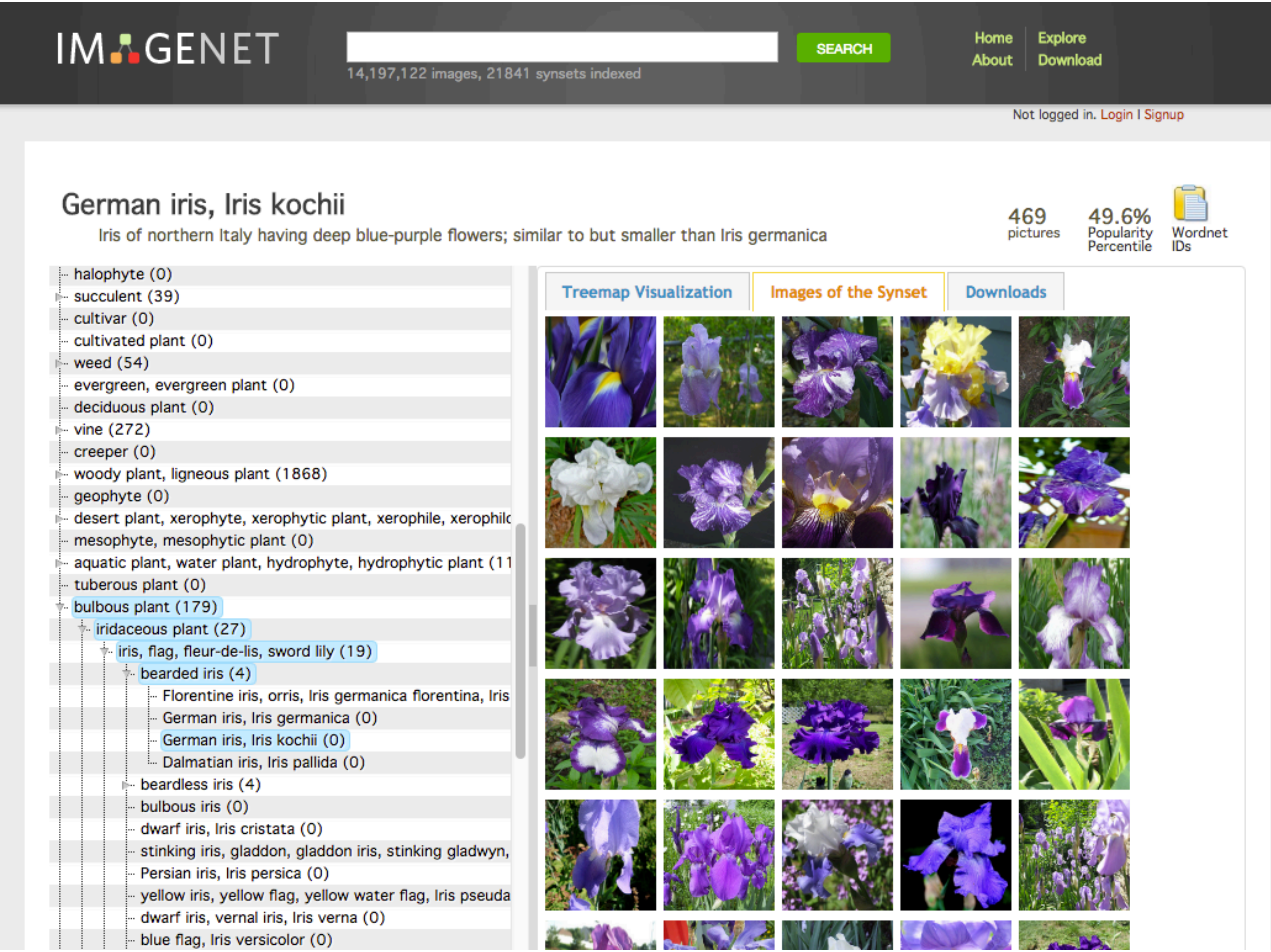
Why should we define a model of $p(\boldsymbol{x}, y)$ at all?

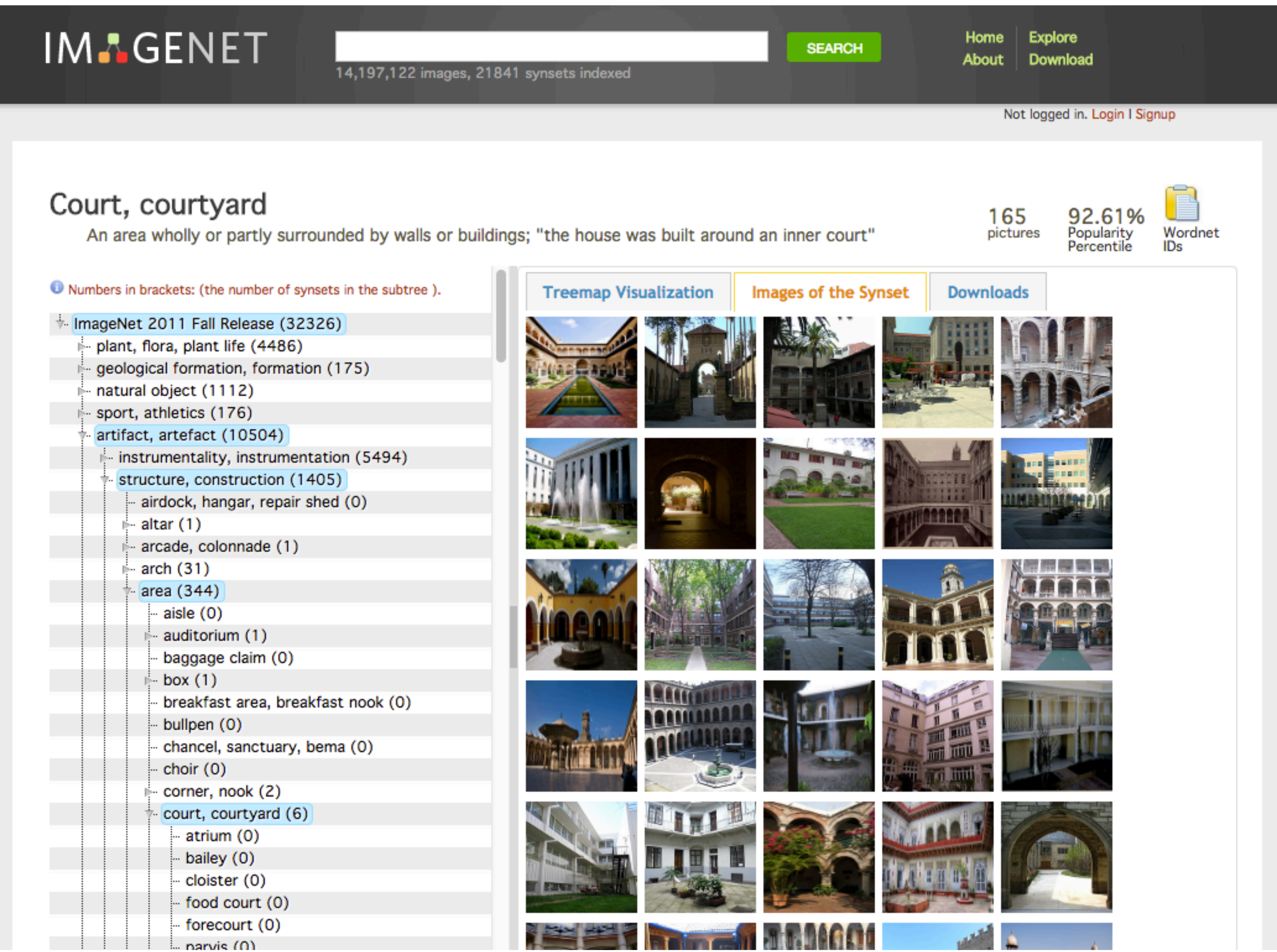
Why not directly model $p(y \mid \boldsymbol{x})$?

Example: Image Classification

- ImageNet LSVRC-2010 contest:
  - **Dataset:** 1.2 million labeled images, 1000 classes
  - **Task**: Given a new image, label it with the correct class
  - **Multiclass** classification problem
- Examples from http://image-net.org/


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
Bird
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings
2126 pictures
92.85% Popularity Percentile
Wordnet IDs
marine animal, marine creature, sea animal, sea creature (1)
scavenger (1)
biped (0)
predator, predatory animal (1)
larva (49)
acrodont (0)
feeder (0)
stunt (0)
chordate (3087)
tunicate, urochordate, urochord (6)
cephalochordate (1)
vertebrate, craniate (3077)
mammal, mammalian (1169)
bird (871)
dickeybird, dickey-bird, dickybird, dicky-bird (0)
cock (1)
hen (0)
nester (0)
night bird (1)
bird of passage (0)
protoavis (0)
archaeopteryx, archeopteryx, Archaeopteryx lithographi
Sinornis (0)
Ibero-mesornis (0)
archaeornis (0)
ratite, ratite bird, flightless bird (10)
carinate, carinate bird, flying bird (0)
passerine, passeriform bird (279)
nonpasserine bird (0)
bird of prey, raptor, raptorial bird (80)
gallinaceous bird, gallinacean (114)
Treemap Visualization
Images of the Synset
Downloads


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login | Signup
German iris, Iris kochii
Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica
469
pictures
49.6%
Popularity
Percentile
Wordnet
IDs
halophyte (0)
succulent (39)
cultivar (0)
cultivated plant (0)
weed (54)
evergreen, evergreen plant (0)
deciduous plant (0)
vine (272)
creeper (0)
woody plant, ligneous plant (1868)
geophyte (0)
desert plant, xerophyte, xerophytic plant, xerophile, xerophilc
mesophyte, mesophytic plant (0)
aquatic plant, water plant, hydrophyte, hydrophytic plant (11
tuberous plant (0)
bulbous plant (179)
iridaceous plant (27)
iris, flag, fleur-de-lis, sword lily (19)
bearded iris (4)
Florentine iris, orris, Iris germanica florentina, Iris
German iris, Iris germanica (0)
German iris, Iris kochii (0)
Dalmatian iris, Iris pallida (0)
beardless iris (4)
bulbous iris (0)
dwarf iris, Iris cristata (0)
stinking iris, gladdon, gladdon iris, stinking gladwyn,
Persian iris, Iris persica (0)
yellow iris, yellow flag, yellow water flag, Iris pseuda
dwarf iris, vernal iris, Iris verna (0)
blue flag, Iris versicolor (0)
Treemap Visualization
Images of the Synset
Downloads


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
Court, courtyard
An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"
165
pictures
92.61%
Popularity
Percentile
Wordnet
IDs
Numbers in brackets: (the number of synsets in the subtree).
ImageNet 2011 Fall Release (32326)
plant, flora, plant life (4486)
geological formation, formation (175)
natural object (1112)
sport, athletics (176)
artifact, artefact (10504)
instrumentality, instrumentation (5494)
structure, construction (1405)
airdock, hangar, repair shed (0)
altar (1)
arcade, colonnade (1)
arch (31)
area (344)
aisle (0)
auditorium (1)
baggage claim (0)
box (1)
breakfast area, breakfast nook (0)
bullpen (0)
chancel, sanctuary, bema (0)
choir (0)
corner, nook (2)
court, courtyard (6)
atrium (0)
bailey (0)
cloister (0)
food court (0)
forecourt (0)
parvis (0)
Treemap Visualization
Images of the Synset
Downloads

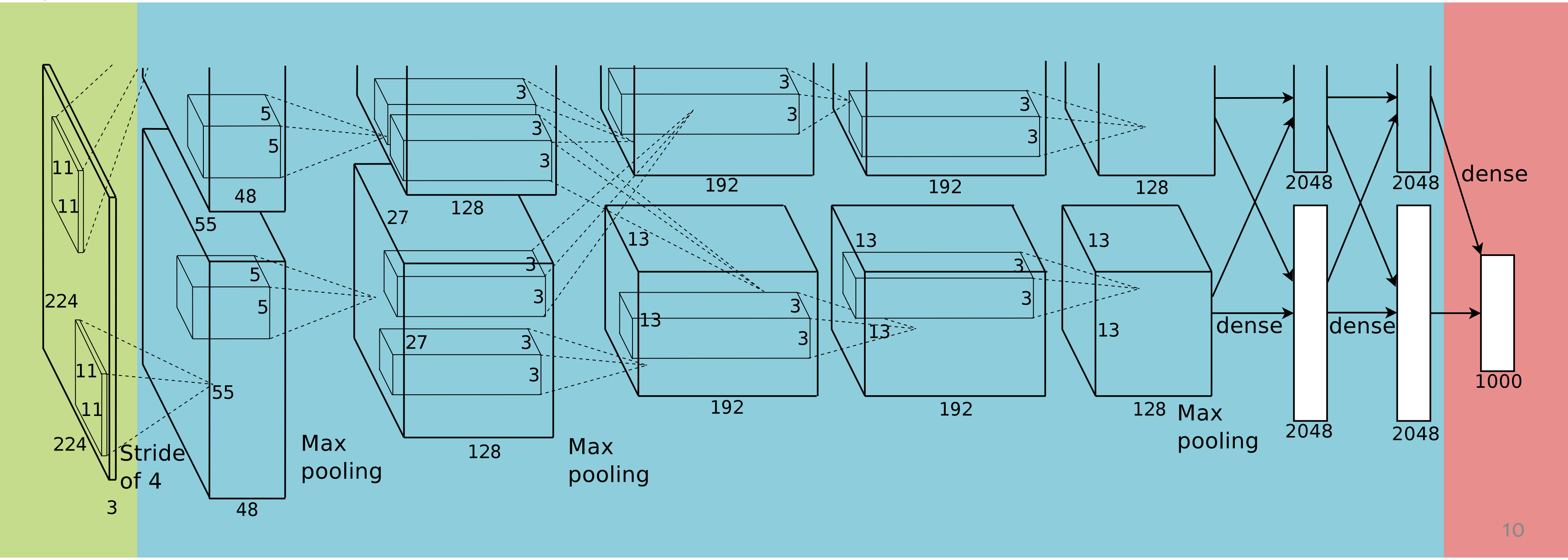
Example: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2011)
17.5% error on ImageNet LSVRC-2010 contest

Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



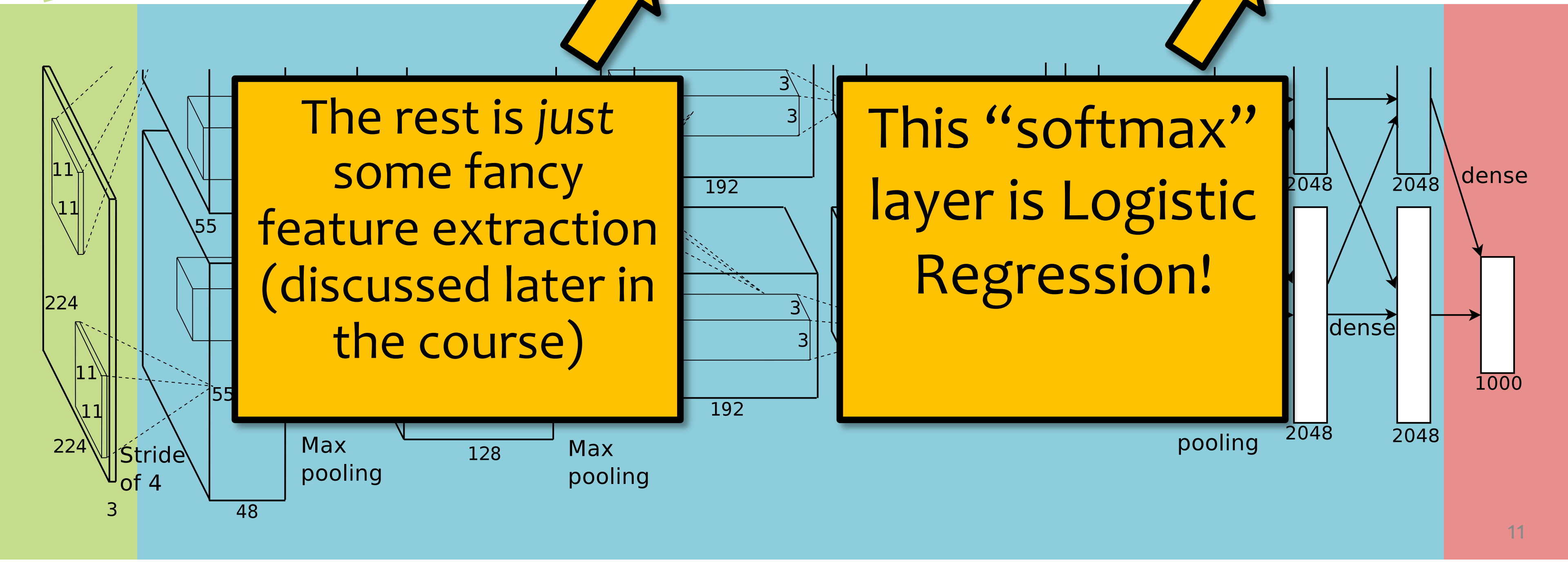
Example: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2011)
17.5% error on ImageNet LSVRC-2010 contest

Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



LOGISTIC REGRESSION

Logistic Regression

Data: Inputs are continuous vectors of length K. Outputs are discrete.

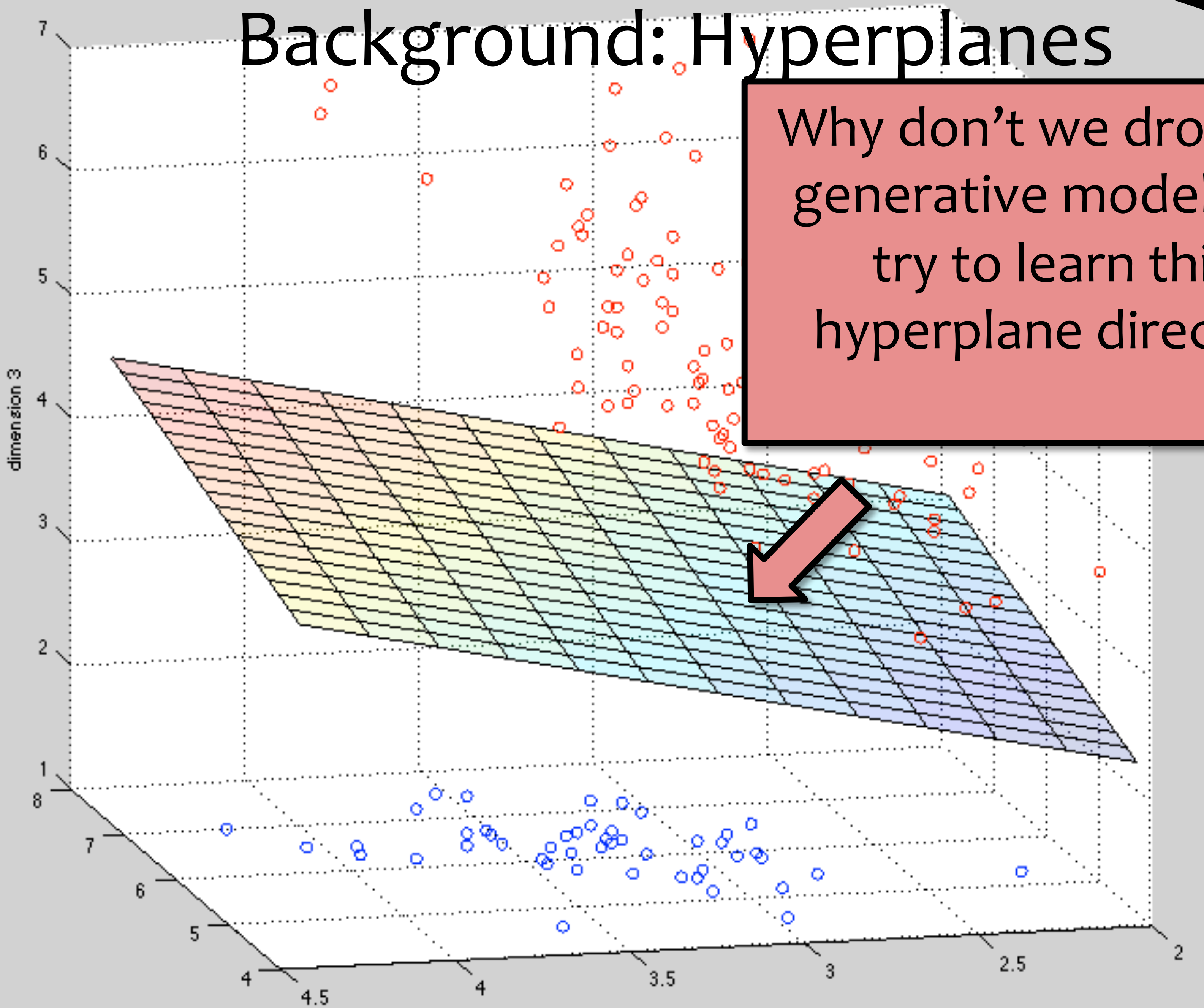
$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

We are back to classification.

Despite the name logistic **regression.**

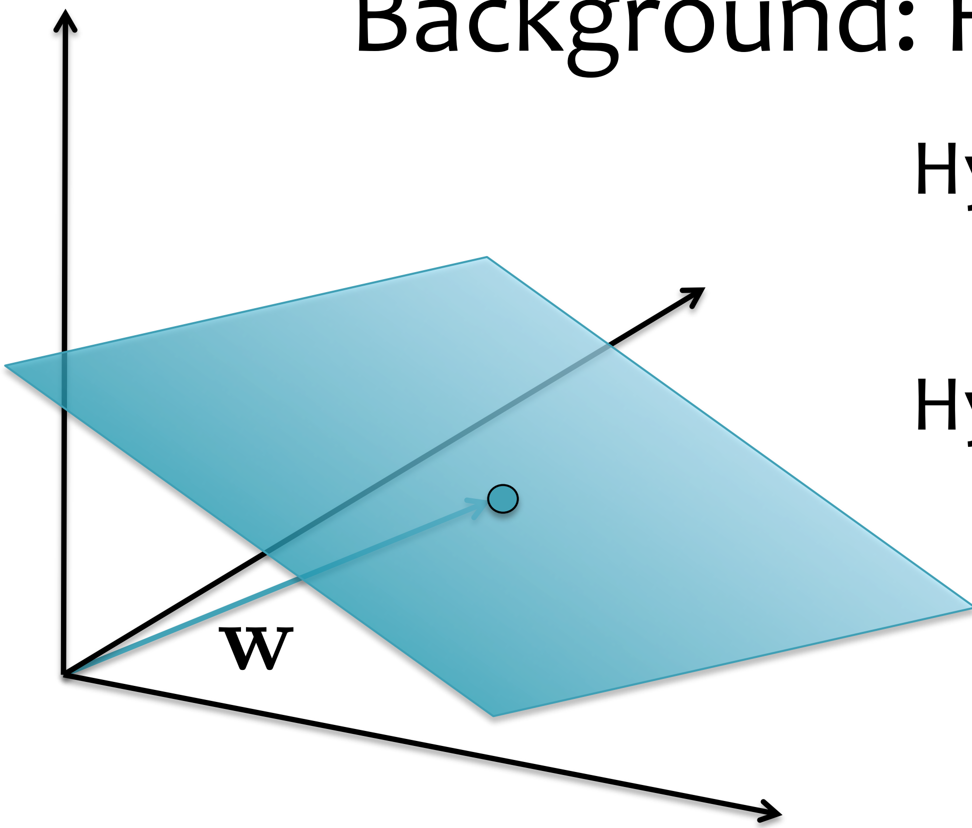
Background: Hyperplanes

Recall…



Why don’t we drop the generative model and try to learn this hyperplane directly?

Background: Hyperplanes



Hyperplane (Definition 1):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = b\}$$

Hyperplane (Definition 2):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0 \text{ and } x_1 = 1\}$$

Half-spaces:

$$\mathcal{H}^+ = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} > 0 \text{ and } x_1 = 1\}$$

$$\mathcal{H}^- = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} < 0 \text{ and } x_1 = 1\}$$

Recall…

d: Hyperplanes

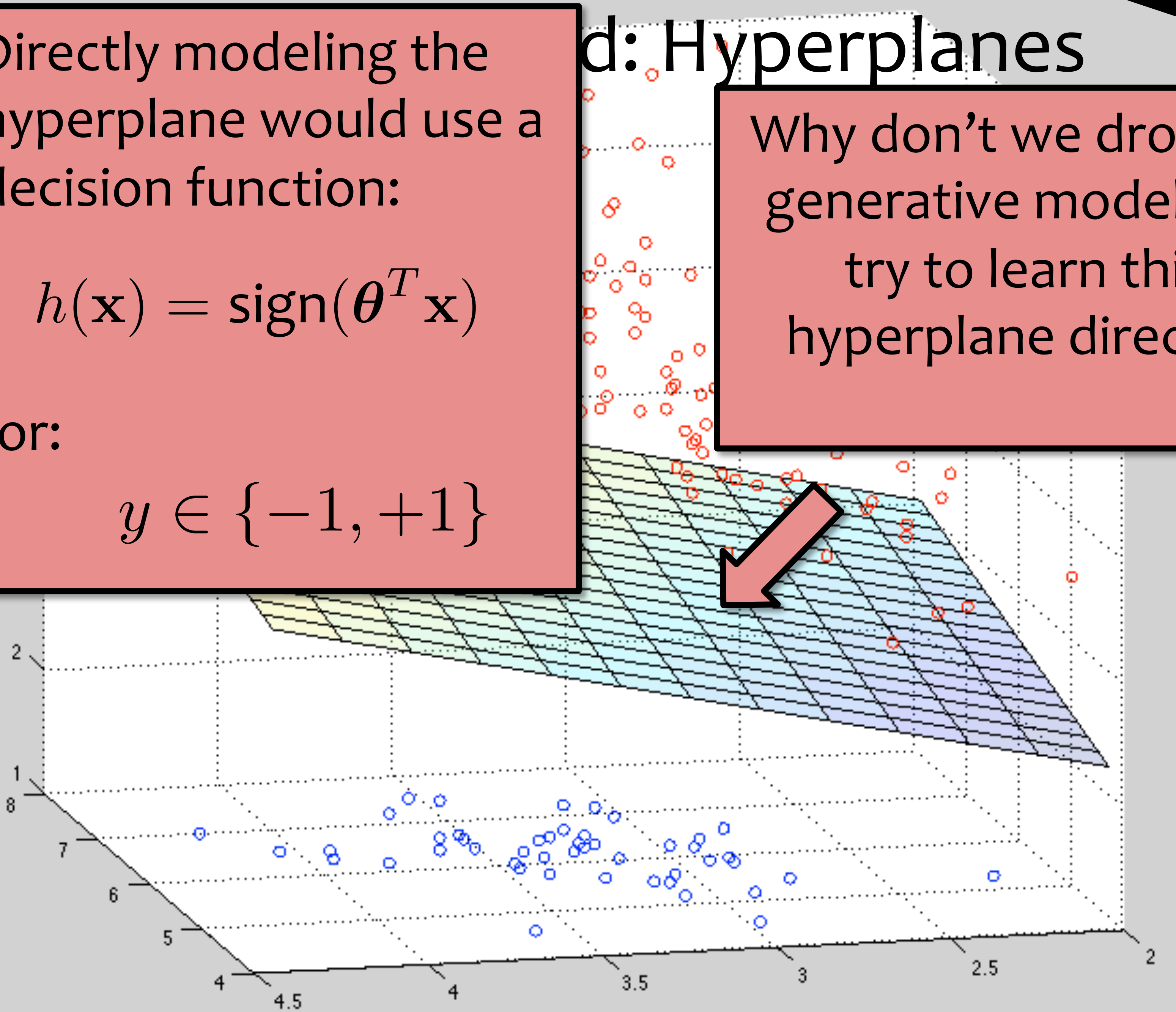
Directly modeling the hyperplane would use a decision function:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

for:

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

Why don't we drop the generative model and try to learn this hyperplane directly?



Using gradient ascent for linear classifiers

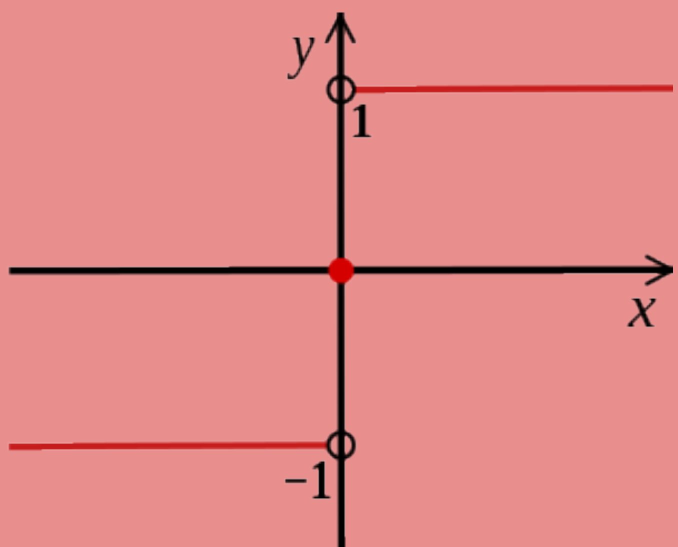
Key idea behind today's lecture:

1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Using gradient ascent for linear classifiers

This decision function isn't differentiable:

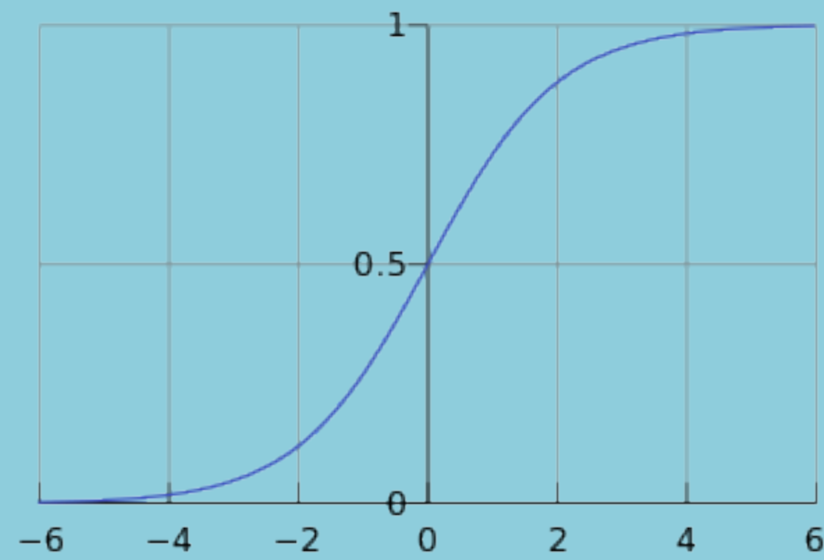
$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T\mathbf{x})$$



sign(x)

Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y=1|\mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^T\mathbf{x})}$$

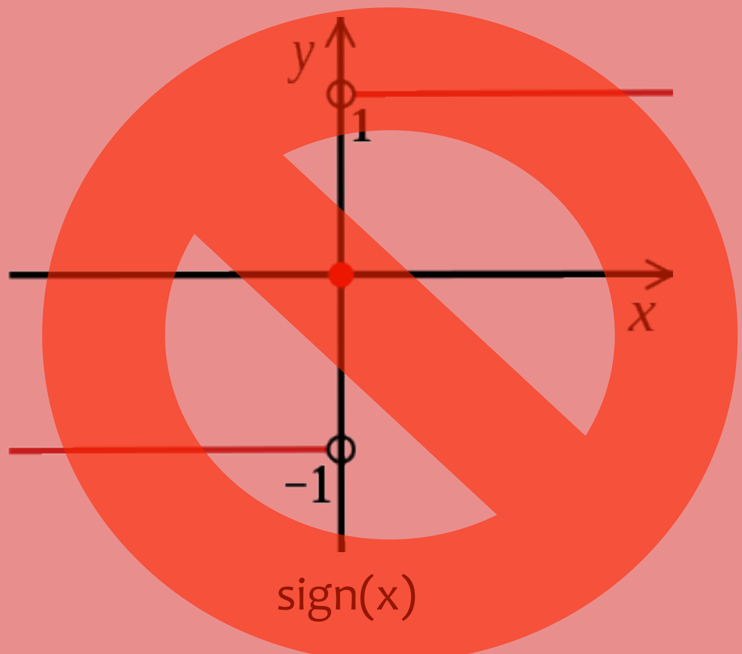


$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

Using gradient ascent for linear classifiers

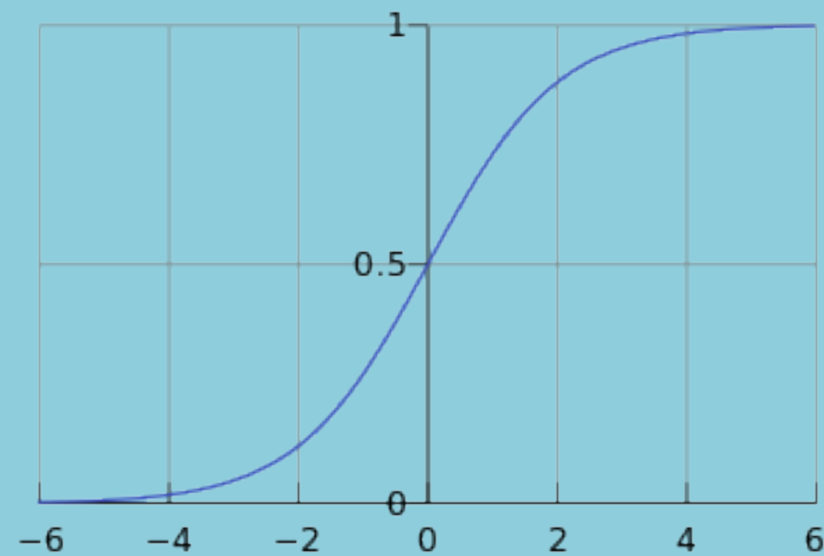
This decision function isn't differentiable:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$



Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$



$$\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}$$

Logistic Regression

Data: Inputs are continuous vectors of length K. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function. $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$

Prediction: Output is the most probable class.

$$\hat{y} = \underset{y \in \{0,1\}}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

Whiteboard

- Log-odds
- Bernoulli interpretation

Maximum **Conditional** Likelihood Estimation

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

We minimize the *negative* log conditional likelihood:

$$J(\boldsymbol{\theta}) = -\log \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y^{(i)}|\mathbf{x}^{(i)})$$

Why?

1. We can't maximize likelihood (as in Naïve Bayes) because we don't have a joint model p(x,y)
2. It worked well for Linear Regression (least squares is MCLE)

Maximum **Conditional** Likelihood Estimation

Learning: Four approaches to solving $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmin}}\, J(\boldsymbol{\theta})$

Approach 1: Gradient Descent
(take larger – more certain – steps opposite the gradient)

Approach 2: Stochastic Gradient Descent (SGD)
(take many small steps opposite the gradient)

Approach 3: Newton's Method
(use second derivatives to better follow curvature)

Approach 4: Closed Form???
(set derivatives equal to zero and solve for parameters)

Maximum **Conditional** Likelihood Estimation

Learning: Four approaches to solving $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$

Approach 1: Gradient Descent
(take larger – more certain – steps opposite the gradient)

Approach 2: Stochastic Gradient Descent (SGD)
(take many small steps opposite the gradient)

Approach 3: Newton's Method
(use second derivatives to better follow curvature)

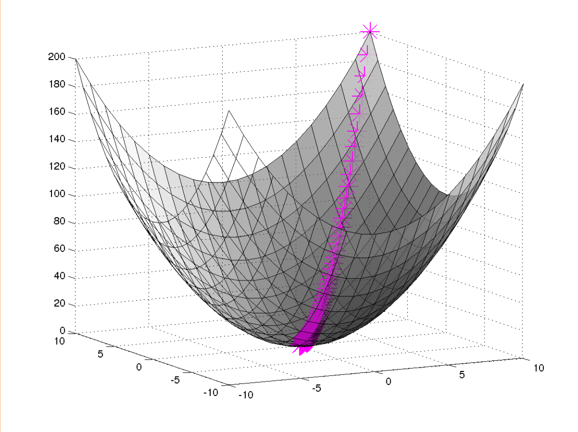
~~**Approach 4:** Closed Form???~~
~~(set derivatives equal to zero and solve for parameters)~~

Recall...

Gradient Descent

Algorithm 1 Gradient Descent

$$
\begin{array}{l}
\textbf{procedure } \mathrm{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)}) \\
\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)} \\
\quad \textbf{while } \text{not converged } \textbf{do} \\
\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\
\quad \textbf{return } \boldsymbol{\theta}
\end{array}
$$



In order to apply GD to Logistic Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

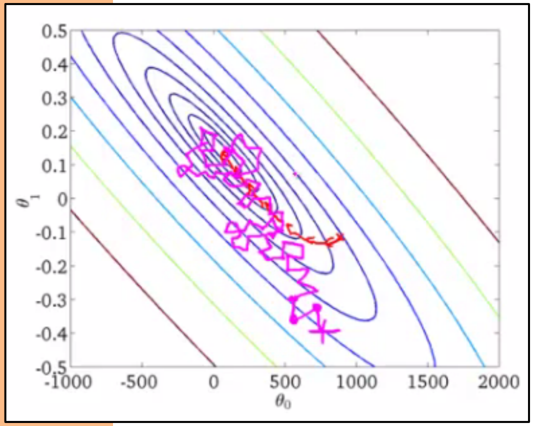
$$
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_N} J(\boldsymbol{\theta}) \end{bmatrix}
$$

Recall...

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

procedure SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
  while not converged **do**
    for $i \in \text{shuffle}(\{1, 2, \ldots, N\})$ **do**
      for $k \in \{1, 2, \ldots, K\}$ **do**
        $\theta_k \leftarrow \theta_k + \lambda \frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta})$
  return $\boldsymbol{\theta}$



We can also apply SGD to solve the MCLE problem for Logistic Regression.

We need a per-example objective:

$$\text{Let } J(\boldsymbol{\theta}) = \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$$
$$\text{where } J^{(i)}(\boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(y^i | \mathbf{x}^i).$$

Optimization for Linear Reg. vs. Logistic Reg.

- Can use the same tricks for both:
  - regularization
  - tuning learning rate on development data
  - shuffle examples out-of-core (if can't fit in memory) and stream over them
  - local hill climbing yields global optimum (both problems are convex)
  - etc.
- But Logistic Regression does not have a closed form solution for MLE parameters.

GRADIENT FOR LOGISTIC REGRESSION

Likelihood on one example is:

$$\log P(Y=y|X=\mathbf{x},\mathbf{w}) = \begin{cases} \log p & \text{if } y=1 \\ \log(1-p) & \text{if } y=0 \end{cases}$$

$$p \equiv \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} = \frac{1}{1+\exp(-\sum_j x^j w^j)}$$

We're going to dive into this thing here: d/dw(p)

$$(\log f)' = \frac{1}{f} f'$$

$$\frac{\partial}{\partial w^j} \log P(Y=y|X=\mathbf{x},\mathbf{w}) = \begin{cases} \frac{1}{p} \boxed{\frac{\partial}{\partial w^j} p} & \text{if } y=1 \\ \frac{1}{1-p}\left(-\frac{\partial}{\partial w^j} p\right) & \text{if } y=0 \end{cases}$$

Slide courtesy of William Cohen

$$p \equiv \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} = \frac{1}{1+\exp(-\sum_j x^j w^j)}$$

$$1-p = \frac{1+\exp(-\sum_j x^j w^j)}{1+\exp(-\sum_j x^j w^j)} - \frac{1}{1+\exp(-\sum_j x^j w^j)} = \boxed{\frac{\exp(-\sum_j x^j w^j)}{1+\exp(-\sum_j x^j w^j)}}$$

$$\boxed{\frac{\partial}{\partial w^j} p} = \frac{\partial}{\partial w^j}\left(1+\exp(-\sum_j x^j w^j)\right)^{-1}$$

$$(f^n)' = nf^{n-1} \cdot f'$$

$$(e^f)' = e^f f'$$

$$= (-1)\left(1+\exp(-\sum_j x^j w^j)\right)^{-2} \frac{\partial}{\partial w^j} \exp(-\sum_j x^j w^j)$$

$$= (-1)\left(1+\exp(-\sum_j x^j w^j)\right)^{-2} \exp(-\sum_j x^j w^j)(-x^j)$$

p

$$= \boxed{\frac{1}{1+\exp(-\sum_j x^j w^j)}}\boxed{\frac{\exp(-\sum_j x^j w^j)}{1+\exp(-\sum_j x^j w^j)}} x^j$$

$$\frac{\partial}{\partial w^j} p = p(1-p)x^j$$

Slide courtesy of William Cohen

$$p \equiv \frac{1}{1+e^{-\mathbf{x}\cdot\mathbf{w}}} = \frac{1}{1+\exp(-\sum_j x^j w^j)}$$

$$\boxed{\frac{\partial}{\partial w^j} p} = p(1-p)x^j$$

$$\log P(Y=y|X=\mathbf{x},\mathbf{w}) = \begin{cases} \log p & \text{if } y=1 \\ \log(1-p) & \text{if } y=0 \end{cases}$$

$$\frac{\partial}{\partial w^j}\log P(Y=y|X=\mathbf{x},\mathbf{w}) = \begin{cases} \frac{1}{p}\boxed{\frac{\partial}{\partial w^j}p} & \text{if } y=1 \\ \frac{1}{1-p}(-\frac{\partial}{\partial w^j}p) & \text{if } y=0 \end{cases}$$

$$\boxed{\frac{\partial}{\partial w^j}p = p(1-p)x^j}$$

$$\frac{\partial}{\partial w^j}\log P(Y=y|X=\mathbf{x},\mathbf{w}) = \begin{cases} \cancel{\frac{1}{p}p}(1-p)x^j = (1-p)x^j & \text{if } y=1 \\ \cancel{\frac{1}{1-p}}(-1)p\cancel{(1-p)}x^j = -px^j & \text{if } y=0 \end{cases}$$

$$\frac{\partial}{\partial w^j}\log P(Y=y|X=\mathbf{x},\mathbf{w}) = (y-p)x^j$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \lambda(y-p)\mathbf{x}$$

Slide courtesy of William Cohen

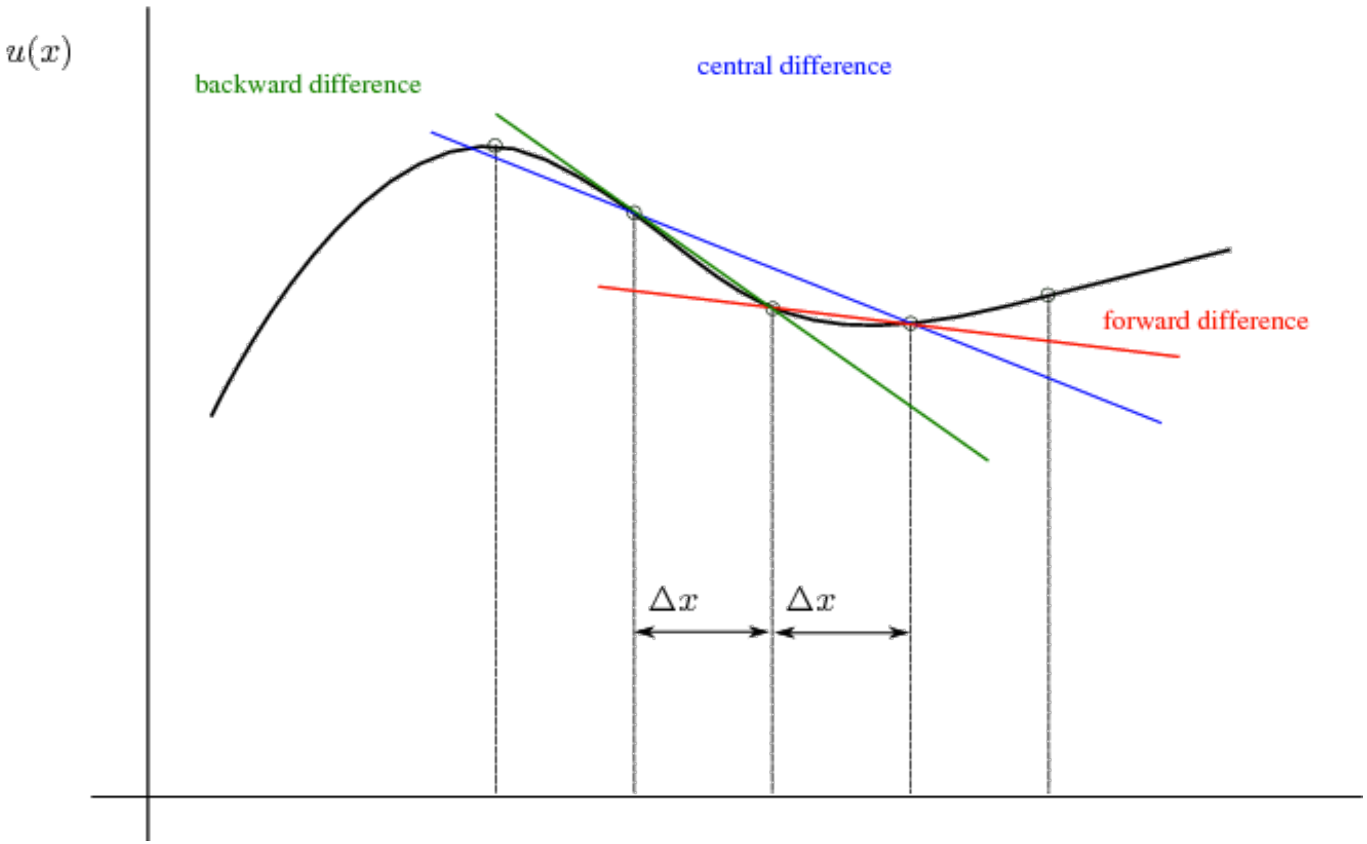
Details: Picking learning rate

- Use grid-search in log-space over small values on a tuning set:
  - e.g., 0.01, 0.001, …
- Sometimes, decrease after each pass:
  - e.g factor of $1/(1 + dt)$, t=epoch
  - sometimes $1/t^2$
- Fancier techniques I won't talk about:
  - Adaptive gradient: scale gradient differently for each dimension (Adagrad, ADAM, ….)

Slide courtesy of William Cohen

Details: Debugging

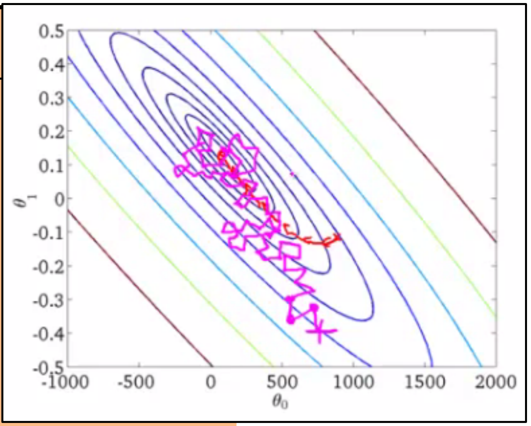
- Check that gradient is indeed a locally good approximation to the likelihood
  - "finite difference test"



Slide courtesy of William Cohen

SGD for Logistic Regression

Algorithm 1 SGD for Logistic Regression



procedure SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
    while not converged **do**
        for $i \in$ shuffle$(\{1, 2, \ldots, N\})$ **do**
            for $k \in \{1, 2, \ldots, K\}$ **do**
                $\theta_k \leftarrow \theta_k + \lambda(\mu^{(i)} - y^{(i)})x_k^{(i)}$
                where $\mu^{(i)} := h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = 1/(1 + \exp(-\boldsymbol{\theta}^T \mathbf{x}))$
    return $\boldsymbol{\theta}$

We can also apply SGD to solve the MCLE problem for Logistic Regression.

We need a per-example objective:

$$\text{Let } J(\boldsymbol{\theta}) = \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$$
$$\text{where } J^{(i)}(\boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(y^i|\mathbf{x}^i).$$

LOGISTIC REGRESSION: OVERFITTING

Convexity and logistic regression

This LCL function is *convex:* there is only one local minimum.

So gradient descent will give the *global* minimum.

Non-stochastic gradient descent

$$\frac{\partial}{\partial w^j} \log P(Y = y | X = \mathbf{x}, \mathbf{w}) = (y - p)x^j$$

- In batch gradient descent, average the gradient over all the examples $D=\{(x_1,y_1)\ldots,(x_n, y_n)\}$

$$\frac{\partial}{\partial w^j} \log P(D|\mathbf{w}) = \frac{1}{n}\sum_i (y_i - p_i)x_i^j =$$

$$= \boxed{\frac{1}{n}\sum_{i:x_i^j=1} y_i} - \boxed{\frac{1}{n}\sum_{i:x_i^j=1} p_i}$$

Slide courtesy of William Cohen

Non-stochastic gradient descent

- This can be interpreted as a difference between the expected value of *$y|x^j$=1 in the data* and the expected value of *$y|x^j$=1 as predicted by the model*
- Gradient ascent tries to make those equal

$$\frac{\partial}{\partial w^j} \log P(D|\mathbf{w}) = \frac{1}{n}\sum_i (y_i - p_i)x_i^j =$$

$$= \boxed{\frac{1}{n}\sum_{i:x_i^j=1} y_i} - \boxed{\frac{1}{n}\sum_{i:x_i^j=1} p_i}$$

Slide courtesy of William Cohen

This LCL function "overfits"

- This can be interpreted as a difference between the expected value of $y|x^j{=}1$ *in the data* and the expected value of $y|x^j{=}1$ *as predicted by the model*
- Gradient ascent tries to make those equal

$$\frac{\partial}{\partial w^j} \log P(D|\mathbf{w}) = \frac{1}{n}\sum_i (y_i - p_i)x_i^j = \frac{1}{n}\sum_{i:x_i^j=1} y_i - \frac{1}{n}\sum_{i:x_i^j=1} p_i$$

- That's impossible for some w^j !
    - e.g., if $x^j = 1$ only in positive examples, the gradient is always positive

Slide courtesy of William Cohen

This LCL function "overfits"

- This can be interpreted as a difference between the expected value of *y|x^j=1 in the data* and the expected value of *y|x^j=1 as predicted by the model*
- Gradient ascent tries to make those equal

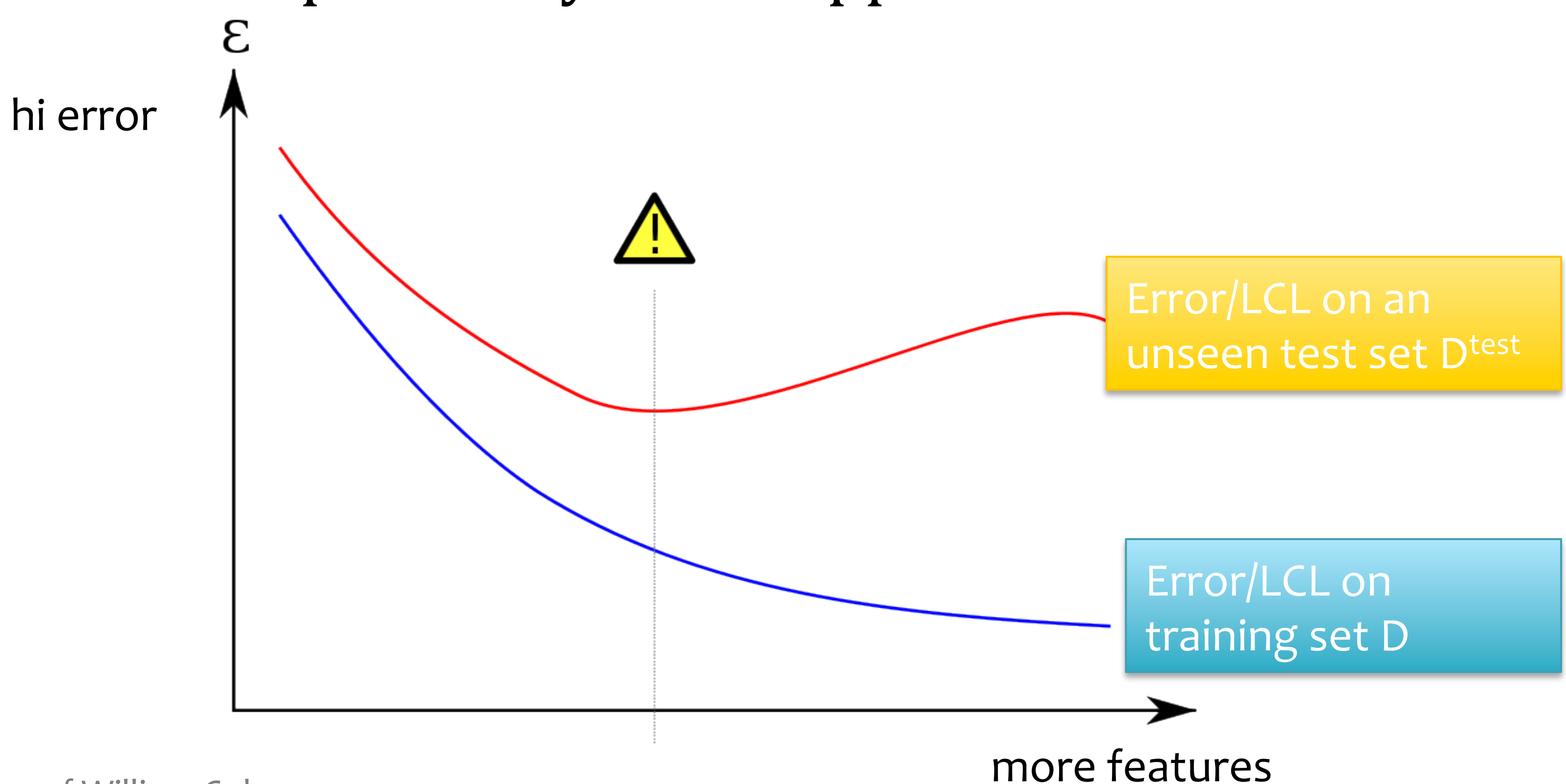
$$\frac{\partial}{\partial w^j} \log P(D|\mathbf{w}) = \frac{1}{n} \sum_i (y_i - p_i) x_i^j = \frac{1}{n} \sum_{i:x_i^j=1} y_i - \frac{1}{n} \sum_{i:x_i^j=1} p_i$$

- That's impossible for some w^j e.g., if they appear only in positive examples, gradient is always possible.
- Using this LCL function for text: practically, it's important to *discard* rare features to get good results.

Slide courtesy of William Cohen

This LCL function "overfits"

- Overfitting is often a problem in supervised learning.
  - When you fit the data (minimize LCL) are you fitting "real structure" in the data or "noise" in the data?
  - Will the patterns you see appear in a test set or not?



Slide courtesy of William Cohen

Iteratively Reweighted Least Squares (IRLS)

NEWTON'S METHOD FOR LOGISTIC REGRESSION

Newton's Method

- From linear regression, we know that we can find the **minimizer** to a **quadratic function** analytically (i.e. **closed form**).

- Yet gradient descent may take **many steps to converge** to that optimum.

- The motivation behind **Newton's method** is to use a **quadratic approximation** of our function to make a good guess where we should step next.

Background: Taylor Series

How can we approximate a function in 1-dimension?

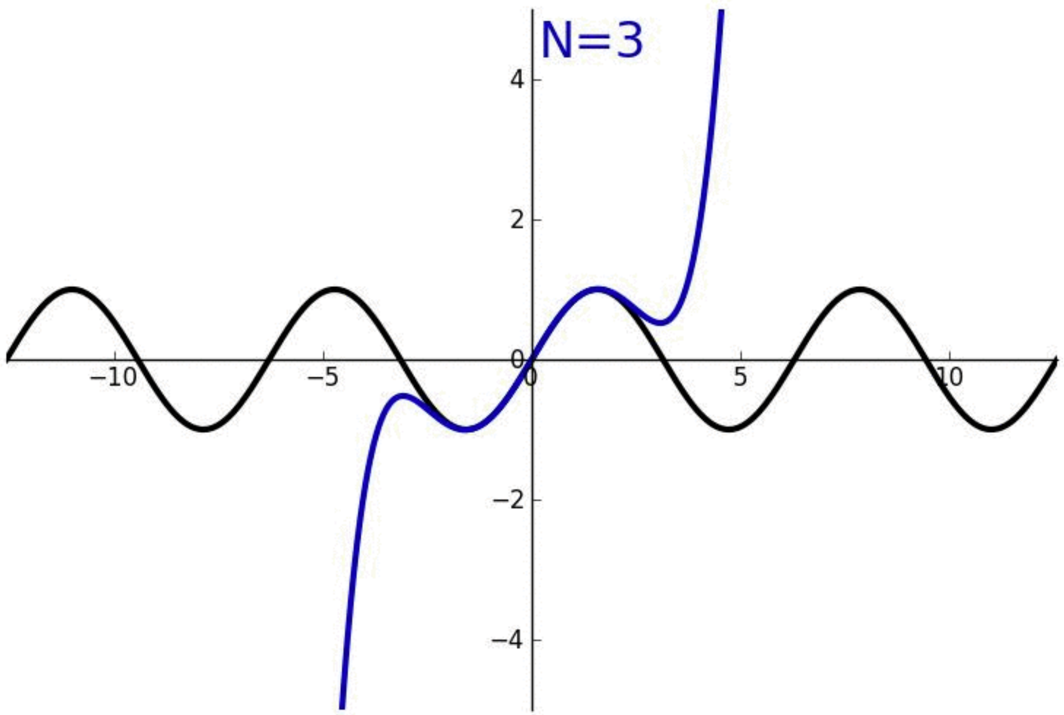
The **Taylor series expansion** for an infinitely differentiable function $f(x)$, $x \in \mathbb{R}$, about a point $v \in \mathbb{R}$ is:

$$f(x) = f(v) + \frac{(x-v)f'(x)}{1!} + \frac{(x-v)^2 f''(x)}{2!} + \frac{(x-v)^3 f'''(x)}{3!} + \ldots$$

The **2nd-order Taylor series approximation** cuts off the expansion after the quadratic term:

$$f(x) \approx f(v) + \frac{(x-v)f'(x)}{1!} + \frac{(x-v)^2 f''(x)}{2!}$$

Background: Taylor Series



https://upload.wikimedia.org/wikipedia/commons/c/cc/Sine_GIF.gif

Hessian Matrix

Definition: the **Hessian** of a K-dimensional function is the matrix of partial second derivatives with respect to each pair of dimensions.

$$H_f(\mathbf{x}) := \nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_K} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K^2} \end{bmatrix}$$

Background: Taylor Series

How can we approximate a function in K-dimensions?

The **Taylor series expansion** for an infinitely differentiable function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^K$, about a point $\mathbf{v} \in \mathbb{R}^K$ is:

$$f(\mathbf{x}) = f(\mathbf{v}) + \frac{(\mathbf{x}-\mathbf{v})^T \nabla f(\mathbf{x})}{1!} + \frac{(\mathbf{x}-\mathbf{v})^T \nabla^2 f(\mathbf{x})(\mathbf{x}-\mathbf{v})}{2!} + \ldots$$

The **2nd-order Taylor series approximation** cuts off the expansion after the quadratic term:

$$f(\mathbf{x}) \approx f(\mathbf{v}) + \frac{(\mathbf{x}-\mathbf{v})^T \nabla f(\mathbf{x})}{1!} + \frac{(\mathbf{x}-\mathbf{v})^T \nabla^2 f(\mathbf{x})(\mathbf{x}-\mathbf{v})}{2!}$$

Background: Taylor Series

How can we approximate a function in K-dimensions?

The **2nd-order Taylor series approximation** cuts off the expansion after the quadratic term:

$$f(\mathbf{x}) \approx \boxed{\tilde{f}(\mathbf{x})} := f(\mathbf{v}) + \frac{(\mathbf{x}-\mathbf{v})^T \nabla f(\mathbf{x})}{1!} + \frac{(\mathbf{x}-\mathbf{v})^T \nabla^2 f(\mathbf{x})(\mathbf{x}-\mathbf{v})}{2!}$$

Taking the derivative of $\tilde{f}(\mathbf{v})$ and setting to $\mathbf{0}$ gives us the closed form minimizer of this (convex) quadratic function:

$$\operatorname*{argmin}_{\mathbf{x}} \boxed{\tilde{f}(\mathbf{x})} = \mathbf{x} - (\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$$

The addend $\nabla \mathbf{x}_{nt} = -(\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$ is called Newton's step.

Newton's Method

Goal: $\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$

1. Approximate the function with the 2nd-order Taylor series

$$f(\mathbf{x}) \approx \tilde{f}(\mathbf{x}) := f(\mathbf{v}) + \frac{(\mathbf{x}-\mathbf{v})^T \nabla f(\mathbf{x})}{1!} + \frac{(\mathbf{x}-\mathbf{v})^T \nabla^2 f(\mathbf{x})(\mathbf{x}-\mathbf{v})}{2!}$$

2. Compute its minimizer

$$\underset{\mathbf{x}}{\operatorname{argmin}} \tilde{f}(\mathbf{x}) = \mathbf{x} - (\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$$

3. Step to that minimizer

$$\mathbf{x} \leftarrow \mathbf{x} - (\nabla^2 f(\mathbf{x}))^{-1} \nabla f(\mathbf{x})$$

4. Repeat

Also called the Newton-Raphson method

Newton's Method

Intuition

A. If $f(\mathbf{x})$ is quadratic, $x + \nabla x_{nt}$ exactly maximizes f.

B. $\tilde{f}(\mathbf{x})$ is a good *quadratic* approximation to the function f near the point $\mathbf{v}$. So if $f(\mathbf{x})$ is *locally* quadratic, then $f(\mathbf{x})$ is locally well approximated by $\tilde{f}(\mathbf{x})$.

Whiteboard

- Example in 1D
- Comparison with Gradient Descent

Newton's Method for Log. Reg.

Algorithm 1 Newton-Raphson Method

procedure NR($\mathcal{D}, \boldsymbol{\theta}^{(0)}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$ ▷ Initialize parameters
  while not converged **do**
    $\mathbf{g} \leftarrow \nabla J(\boldsymbol{\theta})$ ▷ Compute gradient
    $\mathbf{H} \leftarrow \nabla^2 J(\boldsymbol{\theta})$ ▷ Compute Hessian
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$ ▷ Update parameters
  return $\boldsymbol{\theta}$

Now we can apply this to MLE for **Logistic Regression.**

We just need the **gradient** and **Hessian.**

Logistic Regression

Recall...

Data: Inputs are continuous vectors of length K. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function. $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$

Prediction: Output is the most probable class.

$$\hat{y} = \underset{y \in \{0,1\}}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

Maximum **Conditional** Likelihood Estimation

Recall…

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

We minimize the *negative* log conditional likelihood:

$$J(\boldsymbol{\theta}) = -\log \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y^{(i)}|\mathbf{x}^{(i)})$$

Why?

1. We can't maximize likelihood (as in Naïve Bayes) because we don't have a joint model p(x,y)
2. It worked well for Linear Regression (least squares is MCLE)

Maximum **Conditional** Likelihood Estimation

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= -\log \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y^{(i)} | \mathbf{x}^{(i)}) \\
&= -\log \prod_{i=1}^{N} h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})^{y^{(i)}} (1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^{(1-y^{(i)})} \\
&= -\sum_{i=1}^{N} y^{(i)} \log h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) \\
&= -\sum_{i=1}^{N} y^{(i)} \log \mu^{(i)} + (1 - y^{(i)}) \log(1 - \mu^{(i)})
\end{aligned}
$$

where $\mu^{(i)} := h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = 1/(1 + \exp(-\boldsymbol{\theta}^T \mathbf{x}))$

Gradient / Hessian for Log. Reg.

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{N} y^{(i)} \log \mu^{(i)} + (1 - y^{(i)}) \log(1 - \mu^{(i)})$$

where $\mu^{(i)} := h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = 1/(1 + \exp(-\boldsymbol{\theta}^T \mathbf{x}))$

$$\mathbf{g} := \nabla J(\boldsymbol{\theta}) = \sum_{i=1}^{N} (\mu^{(i)} - y^{(i)}) \mathbf{x}^{(i)}$$

$$= \mathbf{X}^T (\boldsymbol{\mu} - \mathbf{y})$$

$$\mathbf{H} := \nabla^2 J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mu^{(i)} (1 - \mu^{(i)}) \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^T$$

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

where $\mathbf{S} = \text{diag}(\mu^{(i)}(1 - \mu^{(i)}))$

Newton's Method for Log. Reg.

Algorithm 1 Newton-Raphson Method

```
procedure NR(𝒟, θ⁽⁰⁾)
    θ ← θ⁽⁰⁾                        ▷ Initialize parameters
    while not converged do
        g ← ∇J(θ)                    ▷ Compute gradient
        H ← ∇²J(θ)                   ▷ Compute Hessian
        θ ← θ − H⁻¹g                 ▷ Update parameters
    return θ
```

1: **procedure** NR($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$ ▷ Initialize parameters
3: **while** not converged **do**
4: $\mathbf{g} \leftarrow \nabla J(\boldsymbol{\theta})$ ▷ Compute gradient
5: $\mathbf{H} \leftarrow \nabla^2 J(\boldsymbol{\theta})$ ▷ Compute Hessian
6: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$ ▷ Update parameters
7: **return** $\boldsymbol{\theta}$

For **Logistic Regression:**

$$-\mathbf{H}^{-1}\mathbf{g} = -(\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}(\mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}))$$

Newton's Method for Log. Reg.

Algorithm 1 Newton-Raphson Method

1: **procedure** NR($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: **while** not converged **do**
4: $\mathbf{g} \leftarrow \nabla J(\boldsymbol{\theta})$
5: $\mathbf{H} \leftarrow \nabla^2 J(\boldsymbol{\theta})$
6: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$ ▷ Update parameters
7: **return** $\boldsymbol{\theta}$

Question: How does Newton step compare **computationally** to solving **Least Squares** in **closed form**

For **Logistic Regression:**

$$-\mathbf{H}^{-1}\mathbf{g} = -(\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}(\mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}))$$

Newton's Method for Log. Reg.
(Iteratively Reweighted Least Squares)

Question: How does Newton step compare **computationally** to solving **Least Squares** in **closed form**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$$

$$= \boldsymbol{\theta} - (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}(\mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}))$$

By substituting in $\mathbf{H}$ and $\mathbf{g}$

$$= (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\left((\mathbf{X}^T\mathbf{S}\mathbf{X})\boldsymbol{\theta} - (\mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}))\right)$$

By factoring out the inverse term

$$= (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\left(\mathbf{S}\mathbf{X}\boldsymbol{\theta} - (\boldsymbol{\mu} - \mathbf{y})\right)$$

By factoring out $\mathbf{X}^T$

$$= (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}\left(\mathbf{X}\boldsymbol{\theta} - \mathbf{S}^{-1}(\boldsymbol{\mu} - \mathbf{y})\right)$$

By factoring out $\mathbf{S}$

$$= (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}\mathbf{z}$$

where $\mathbf{z} = \mathbf{X}\boldsymbol{\theta} - \mathbf{S}^{-1}(\boldsymbol{\mu} - \mathbf{y})$

Recall LMS

- Cost function in matrix form:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$
$$= \frac{1}{2}\left(\mathbf{X}\theta - \vec{y}\right)^T\left(\mathbf{X}\theta - \vec{y}\right)$$

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1 & -- \\ -- & \mathbf{x}_2 & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n & -- \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- To minimize *J(θ)*, take derivative and set to zero:

$$\nabla_\theta J = \frac{1}{2}\nabla_\theta \mathrm{tr}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)$$
$$= \frac{1}{2}\left(\nabla_\theta \mathrm{tr}\,\theta^T X^T X\theta - 2\nabla_\theta \mathrm{tr}\,\vec{y}^T X\theta + \nabla_\theta \mathrm{tr}\,\vec{y}^T\vec{y}\right)$$
$$= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right)$$
$$= X^T X\theta - X^T\vec{y} = 0$$

$$\Rightarrow \boxed{X^T X\theta = X^T\vec{y}}$$

The normal equations

$$\Downarrow$$

$$\theta^* = \left(X^T X\right)^{-1} X^T\vec{y}$$

Newton's Method for Log. Reg.
(Iteratively Reweighted Least Squares)

Question: How does Newton step compare **computationally** to solving **Least Squares** in **closed form**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$$
$$= (\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}\mathbf{X}^T\mathbf{S}\mathbf{z}$$
where $\mathbf{z} = \mathbf{X}\boldsymbol{\theta} - \mathbf{S}^{-1}(\boldsymbol{\mu} - \mathbf{y})$

The above update yields the minimizer for the **weighted least squares problem:**

$$\boldsymbol{\theta}^* \leftarrow \underset{\boldsymbol{\theta}}{\text{argmin}} (\mathbf{z} - \mathbf{X}\boldsymbol{\theta})^T\mathbf{S}(\mathbf{z} - \mathbf{X}\boldsymbol{\theta})$$

$$= \underset{\boldsymbol{\theta}}{\text{argmin}} \sum_{i=1}^{N} S_{ii}(z_i - \boldsymbol{\theta}^T\mathbf{x}^{(i)})^2$$

where S_{ii} is the weight of the ith "training example" consisting of the pair $(\mathbf{x}^{(i)}, z_i)$.

Newton's Method for Log. Reg.

Algorithm 1 Newton-Raphson Method

procedure NR($\mathcal{D}, \boldsymbol{\theta}^{(0)}$)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
    while not converged **do**
        $\mathbf{g} \leftarrow \nabla J(\boldsymbol{\theta})$
        $\mathbf{H} \leftarrow \nabla^2 J(\boldsymbol{\theta})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$ ▷ Update parameters
    return $\boldsymbol{\theta}$

Question: How does Newton step compare **computationally** to solving **Least Squares** in **closed form**

Answer: It's solving a weighted version of the same problem.

Hence the name "Iteratively Reweighted Least Squares (IRLS)".

For **Logistic Regression:**

$$-\mathbf{H}^{-1}\mathbf{g} = -(\mathbf{X}^T\mathbf{S}\mathbf{X})^{-1}(\mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}))$$

Newton's Method for Linear Regression

Newton's method applied to Linear Regression (or any convex quadratic function) **converges in exactly 1-step** to the true optimum.

This is **equivalent** to solving the Normal Equations

Matching Game

Goal: Match the Algorithm to its Update Rule

1. SGD for Logistic Regression $h_{\boldsymbol{\theta}}(\mathbf{x}) = p(y\|x)$	**4.** $\theta_k \leftarrow \theta_k + (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})$
2. Least Mean Squares $h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$	**5.** $\theta_k \leftarrow \theta_k + \frac{1}{1 + \exp \lambda(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})}$
3. Perceptron (next lecture) $h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$	**6.** $\theta_k \leftarrow \theta_k + \lambda(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})x_k^{(i)}$

A. 1=5, 2=4, 3=6
B. 1=5, 2=6, 3=4
C. 1=6, 2=4, 3=4
D. 1=5, 2=6, 3=6
E. 1=6, 2=6, 3=6

DISCRIMINATIVE AND GENERATIVE CLASSIFIERS

Generative vs. Discriminative

- **Generative Classifiers:**
  - Example: Naïve Bayes
  - Define a joint model of the observations **x** and the labels y: $p(\boldsymbol{x}, y)$
  - Learning maximizes (joint) likelihood
  - Use Bayes' Rule to classify based on the posterior:

    $$p(y|\mathbf{x}) = p(\mathbf{x}|y)p(y)/p(\mathbf{x})$$
- **Discriminative Classifiers:**
  - Example: Logistic Regression
  - Directly model the conditional: $p(y|\mathbf{x})$
  - Learning maximizes conditional likelihood

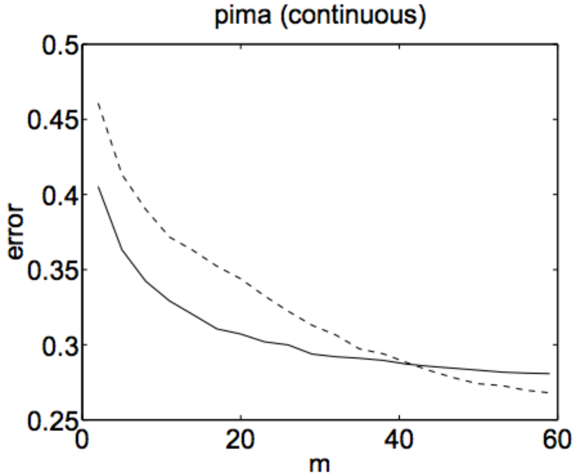
Generative vs. Discriminative

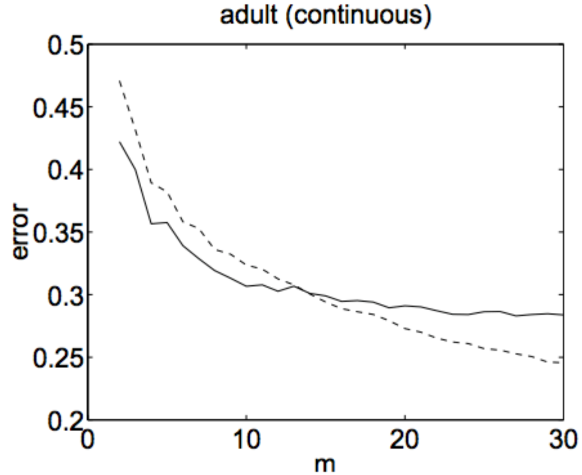
Finite Sample Analysis (Ng & Jordan, 2002)

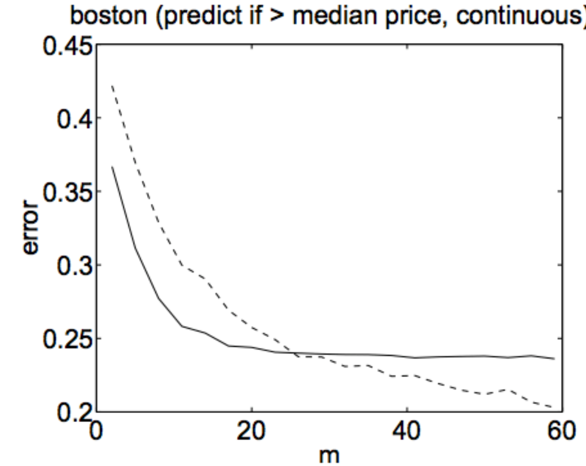
[Assume that we are learning from a finite training dataset]

If model assumptions are correct: Naive Bayes is a more efficient learner (requires fewer samples) than Logistic Regression

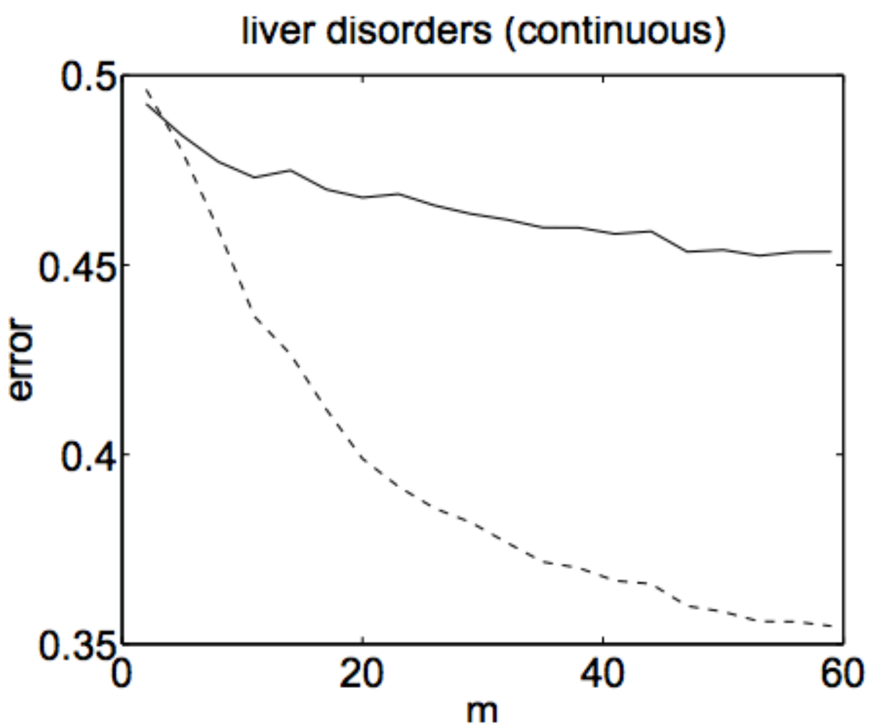
If model assumptions are incorrect: Logistic Regression has lower asymtotic error, and does better than Naïve Bayes

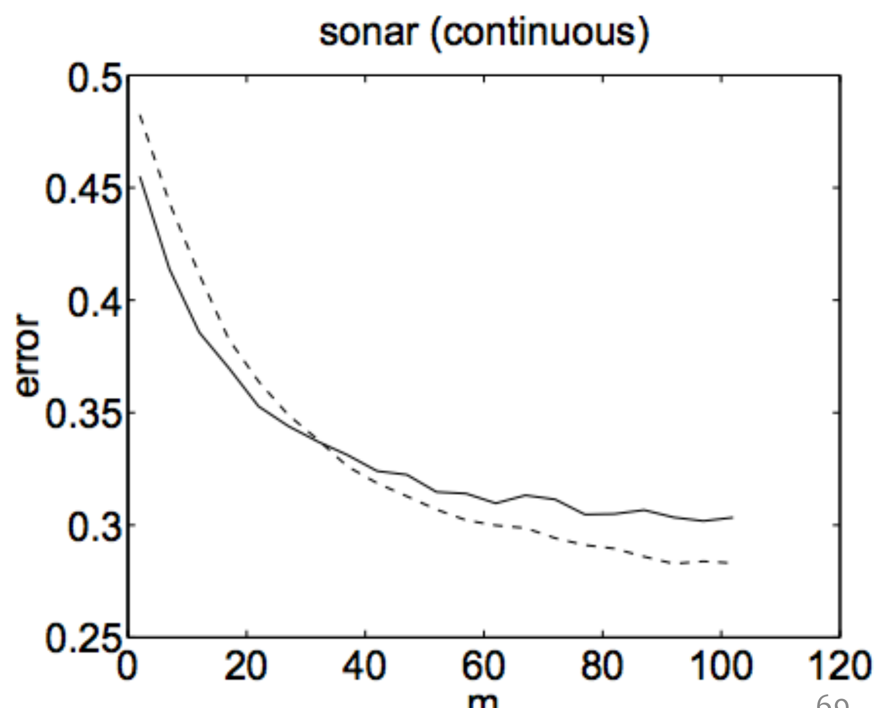




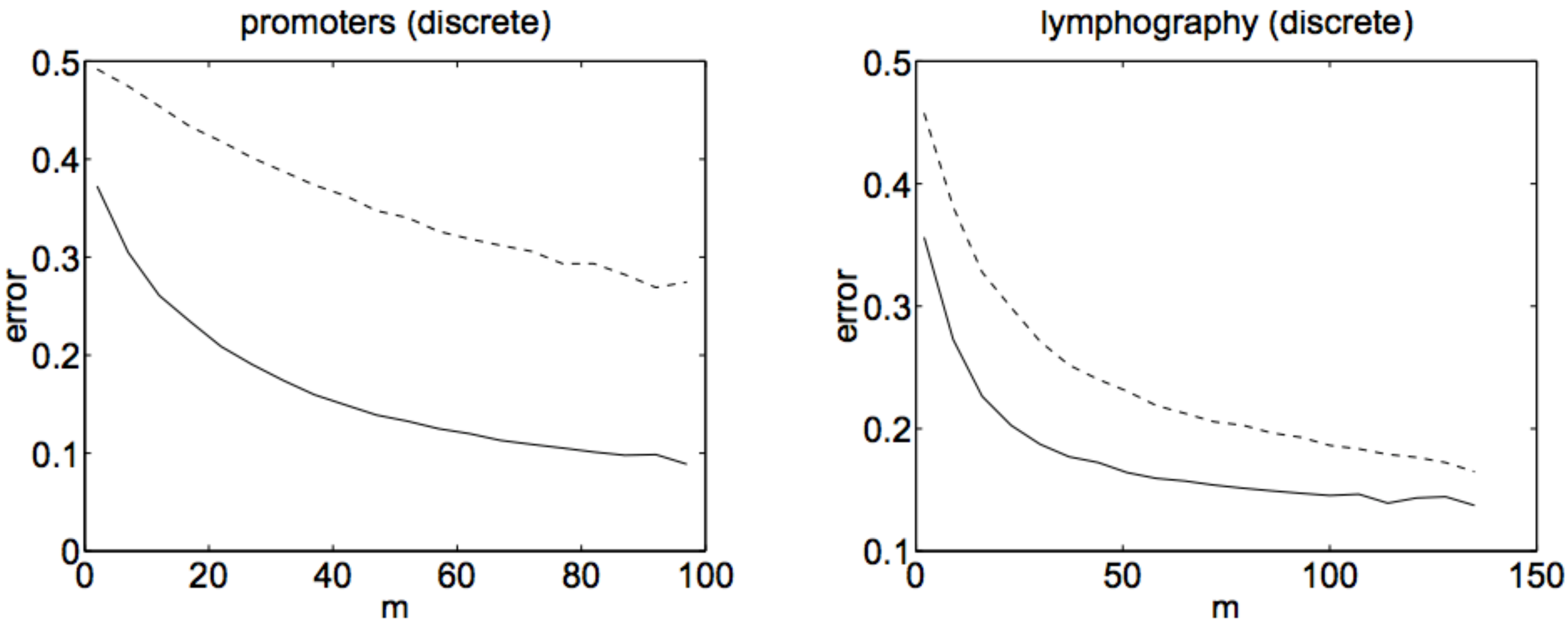


solid: NB dashed: LR





solid: NB dashed: LR



Naïve Bayes makes stronger assumptions about the data but needs fewer examples to estimate the parameters

"On Discriminative vs Generative Classifiers: …." Andrew Ng and Michael Jordan, NIPS 2001.

Slide courtesy of William Cohen

Generative vs. Discriminative

Learning (Parameter Estimation)

Naïve Bayes:
Parameters are decoupled → Closed form solution for MLE

Logistic Regression:
Parameters are coupled → No closed form solution – must use iterative optimization techniques instead

Naïve Bayes vs. Logistic Reg.

Learning (MAP Estimation of Parameters)

Bernoulli Naïve Bayes:
Parameters are probabilities → Beta prior (usually) pushes probabilities away from zero / one extremes

Logistic Regression:
Parameters are not probabilities → Gaussian prior encourages parameters to be close to zero

(effectively pushes the probabilities away from zero / one extremes)

Naïve Bayes vs. Logistic Reg.

Features

Naïve Bayes:
Features x are assumed to be conditionally independent given y. (i.e. Naïve Bayes Assumption)

Logistic Regression:
No assumptions are made about the form of the features x. They can be dependent and correlated in any fashion.

Summary

1. Discriminative classifiers directly model the **conditional**, p(y|x)
2. Logistic regression is a **simple linear classifier**, that retains a **probabilistic** semantics
3. Parameters in LR are learned by **iterative optimization** (e.g. SGD)
4. **Regularization** helps to avoid **overfitting**