10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Feature Engineering
+ Regularization
+ Neural Networks

Matt Gormley
Lecture 11
Feb. 19, 2020

Reminders

- Homework 4: Logistic Regression
  - Out: Wed, Feb. 19
  - Due: Fri, Feb. 28 at 11:59pm
- Today's In-Class Poll
  - http://p11.mlcourse.org

MULTINOMIAL LOGISTIC REGRESSION

Multinomial Logistic Regression

Chalkboard

- Background: Multinomial distribution
- Definition: Multi-class classification
- Geometric intuitions
- Multinomial logistic regression model
- Generative story
- Reduction to binary logistic regression
- Partial derivatives and gradients
- Applying Gradient Descent and SGD
- Implementation w/ sparse features

Debug that Program!

In-Class Exercise: *Think-Pair-Share*

Debug the following program which is (incorrectly) attempting to run SGD for multinomial logistic regression

Buggy Program:

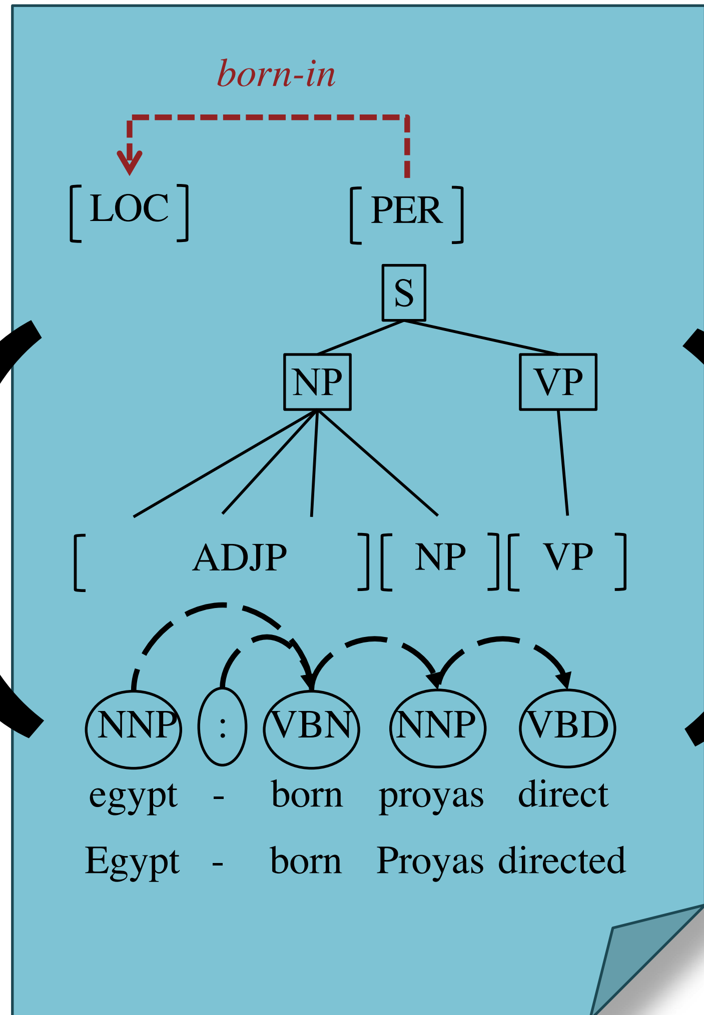
```
while not converged:
  for i in shuffle([1,…,N]):
    for k in [1,…,K]:
      theta[k] = theta[k] - lambda * grad(x[i], y[i], 
theta, k)
```

Assume: `grad(x[i], y[i], theta, k)` returns the gradient of the negative log-likelihood of the training example (x[i],y[i]) with respect to vector `theta[k]`. `lambda` is the learning rate. N = # of examples. K = # of output classes. M = # of features. `theta` is a K by M matrix.

FEATURE ENGINEERING

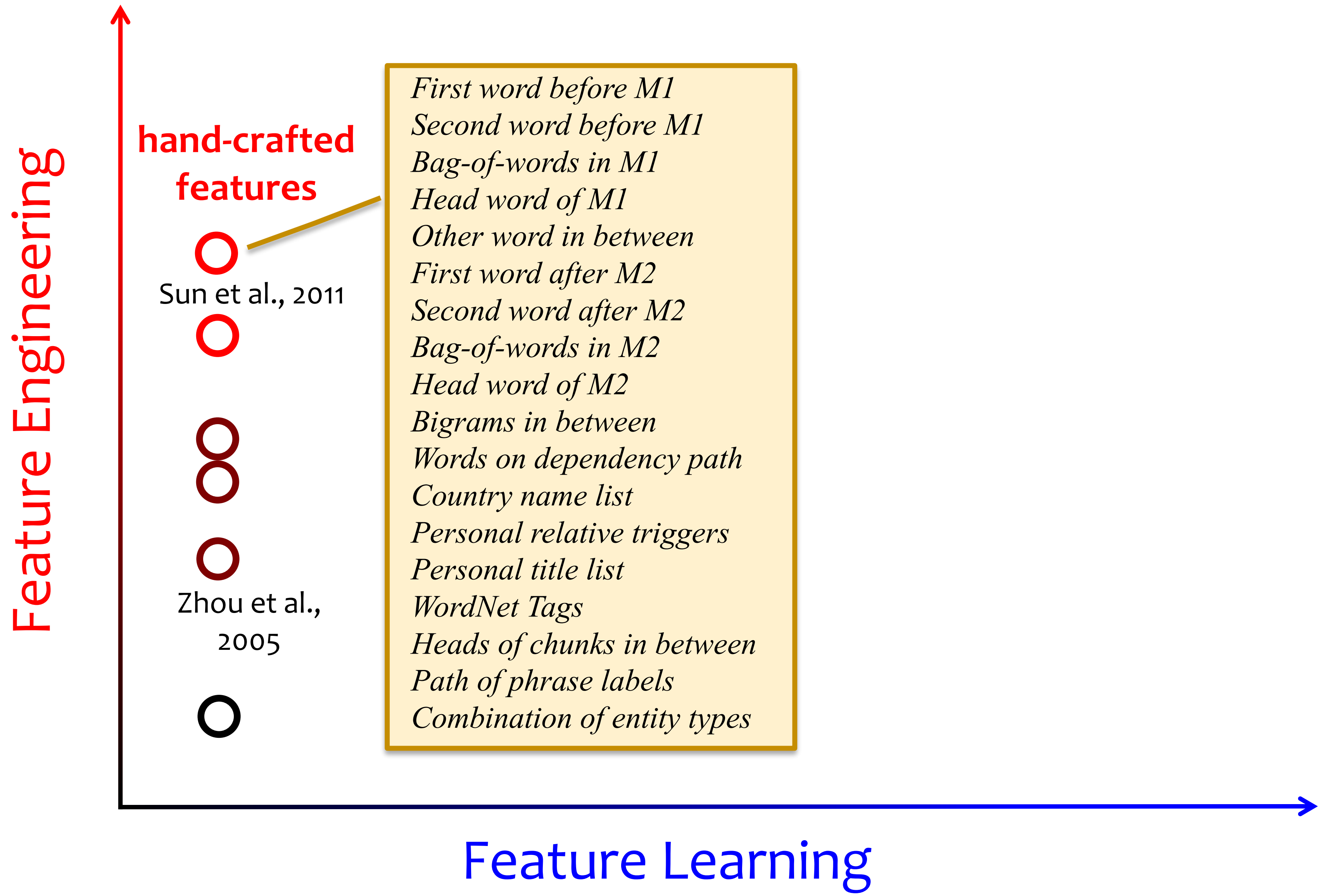
Handcrafted Features

$$p(\textcolor{red}{y}|x) \propto \exp(\Theta_{\textcolor{red}{y}} \bullet f($$

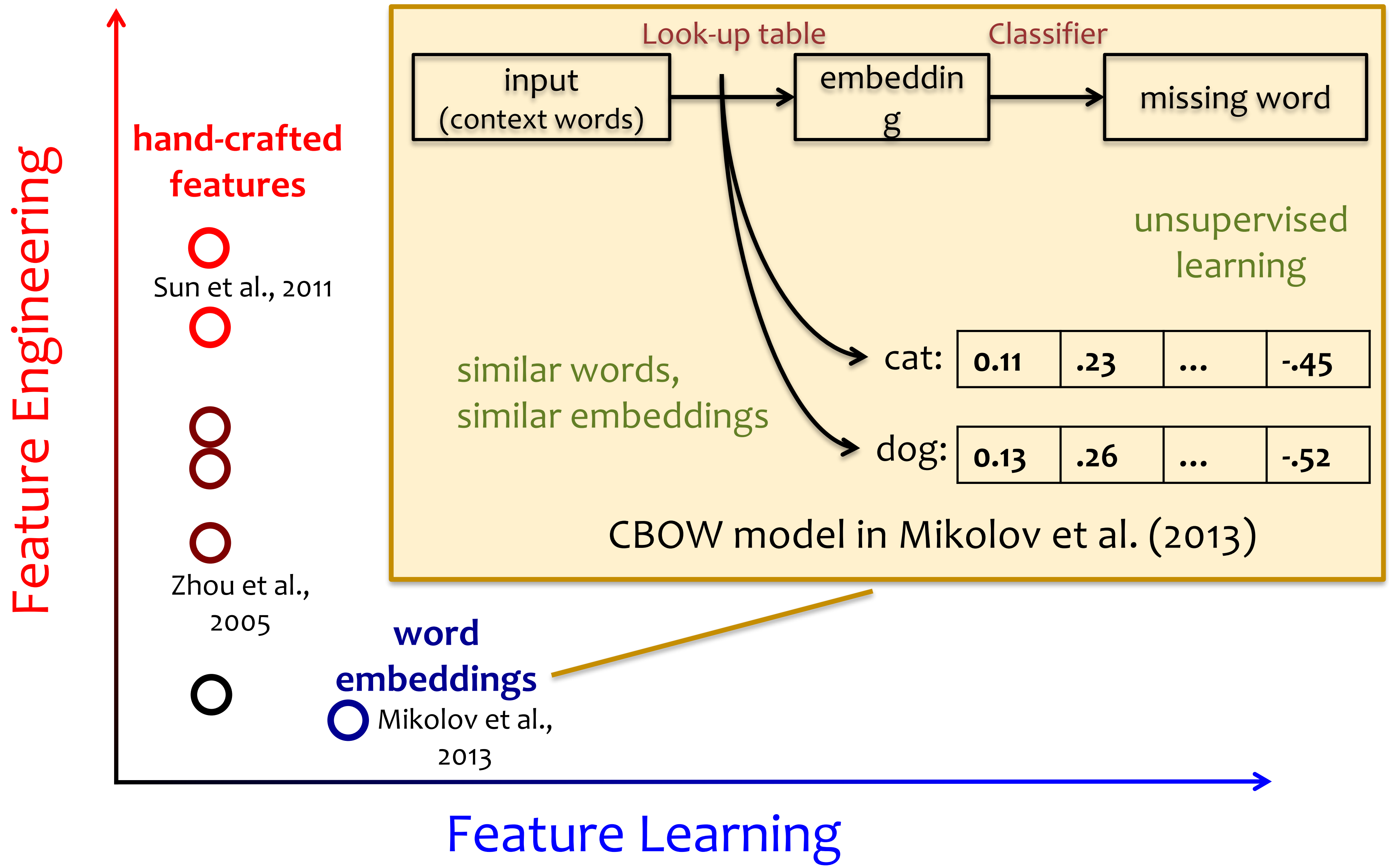


$$))$$

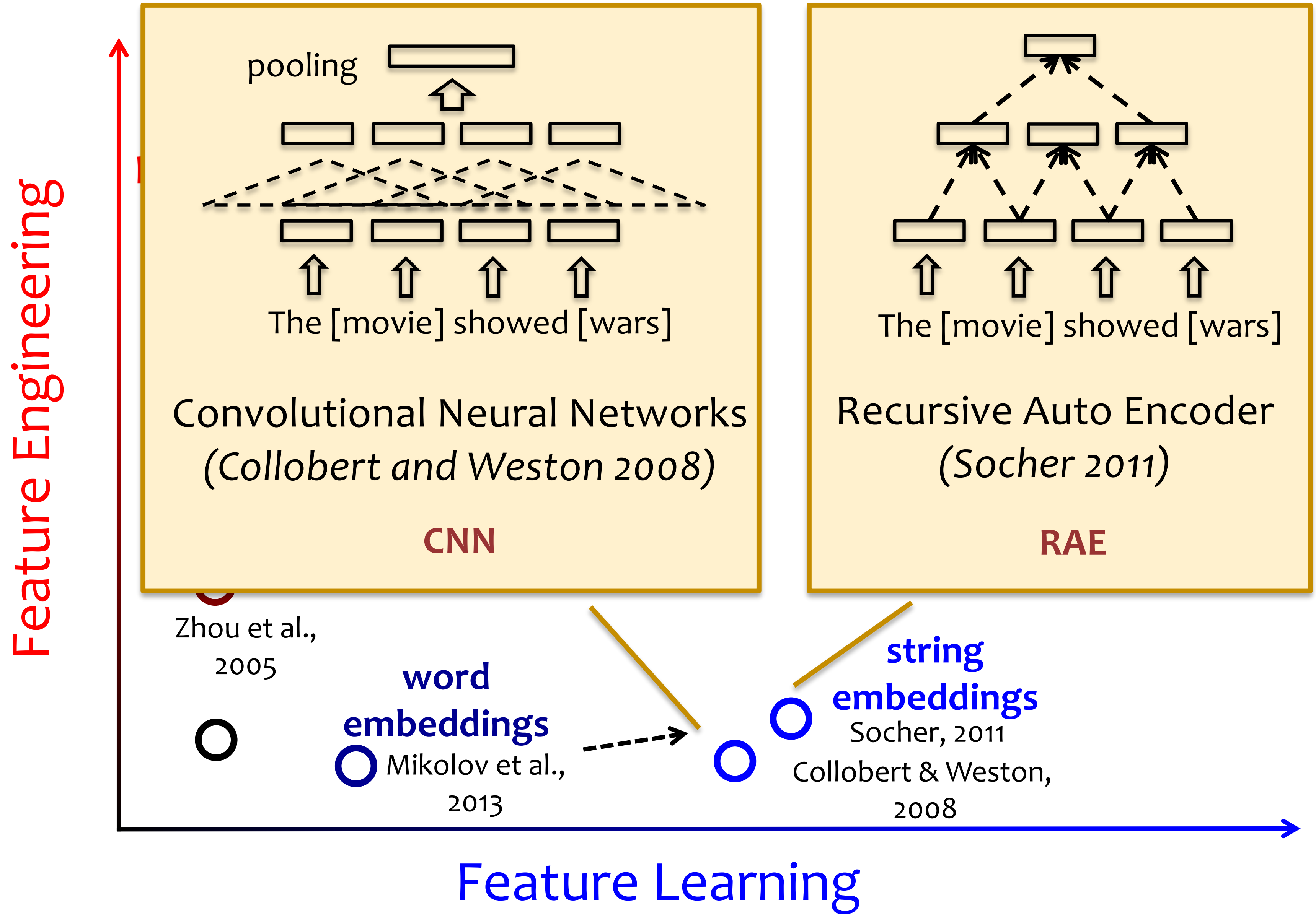
Where do features come from?



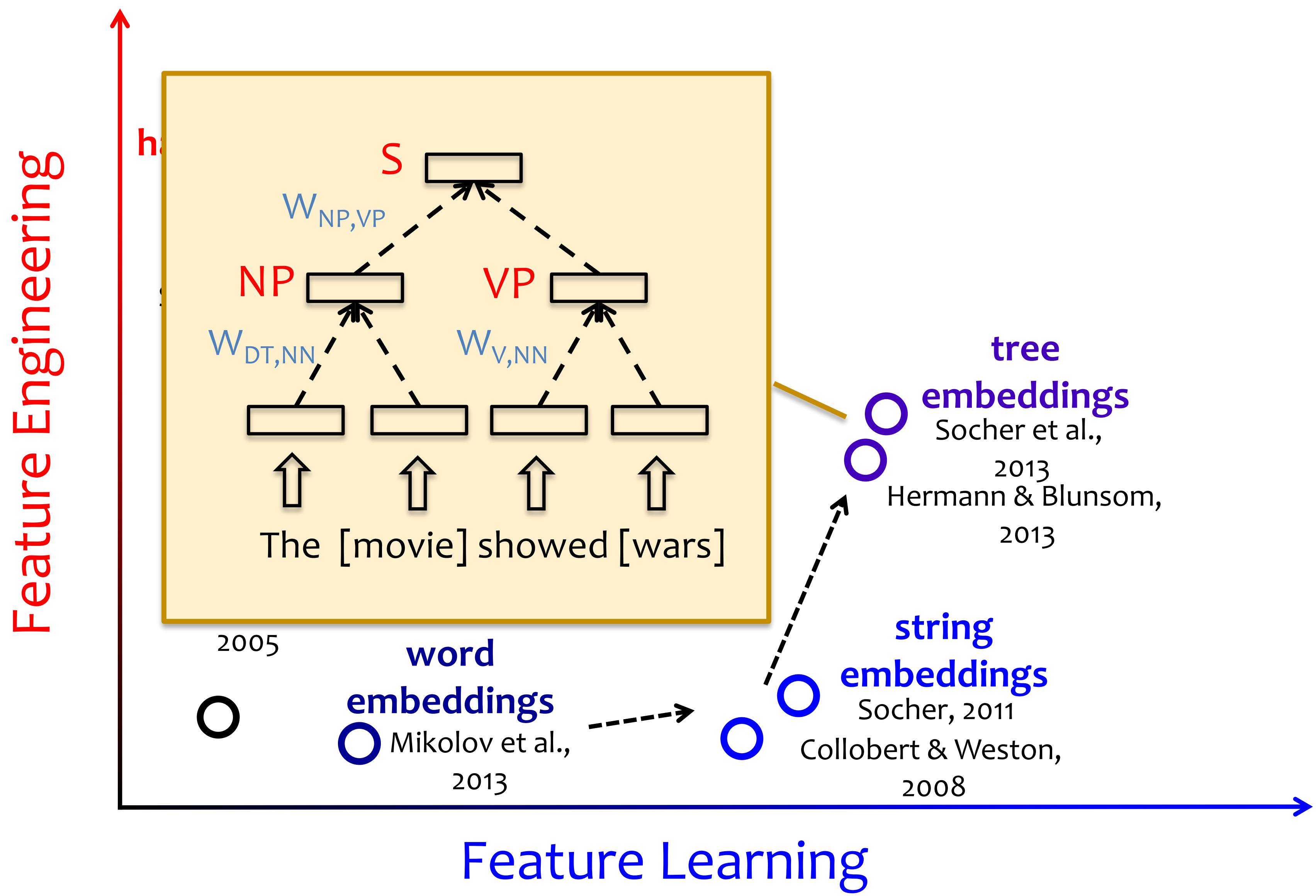
Where do features come from?



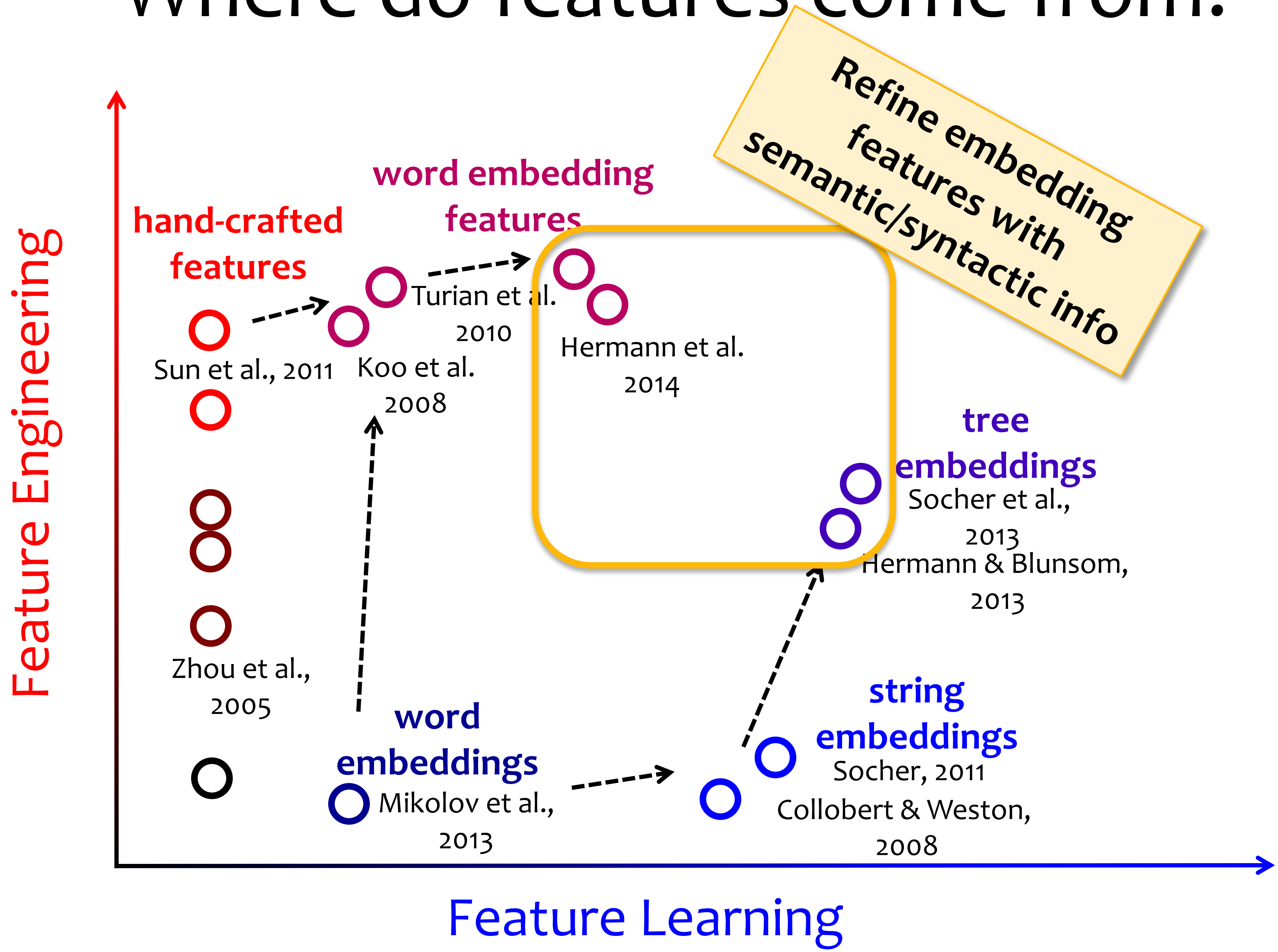
Where do features come from?



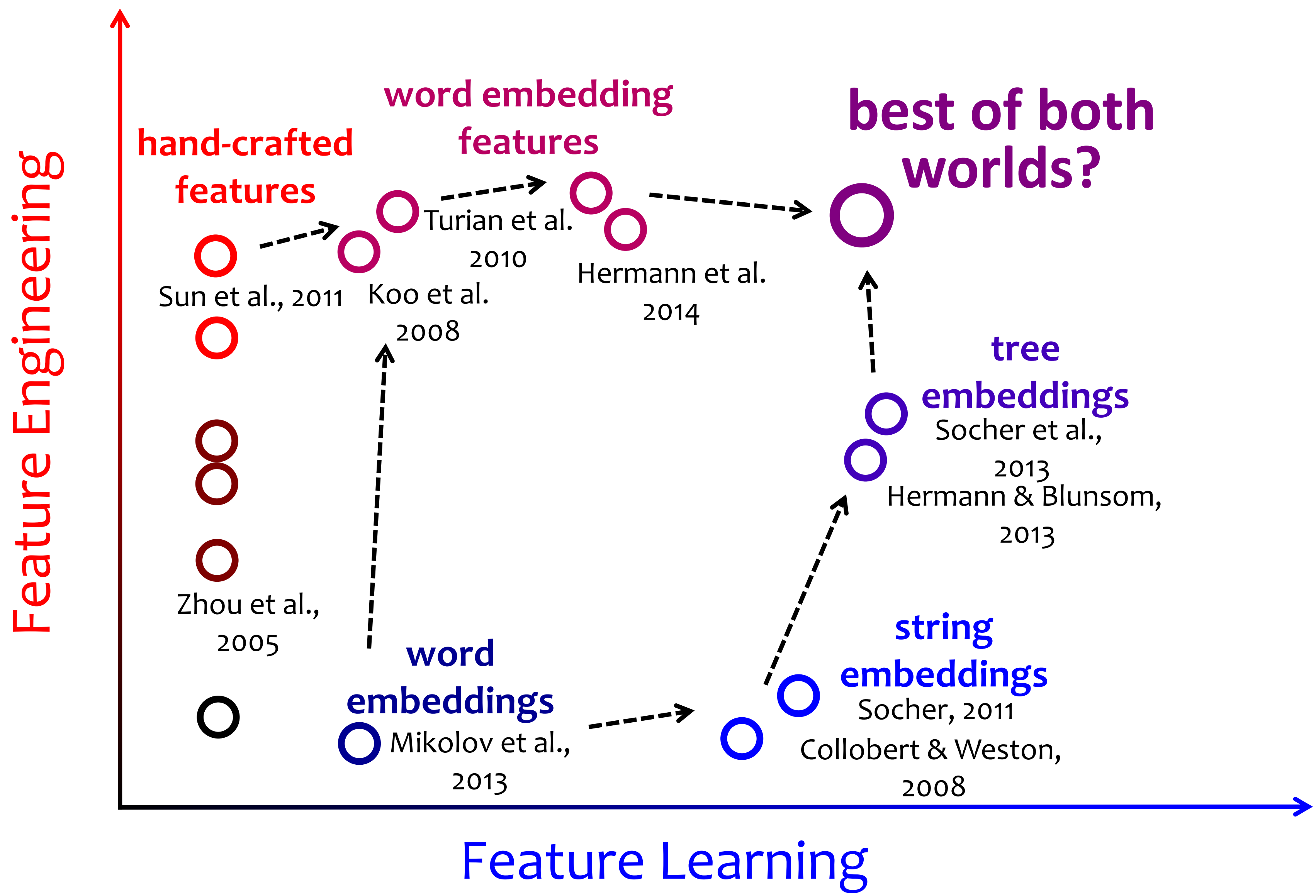
Where do features come from?



Where do features come from?



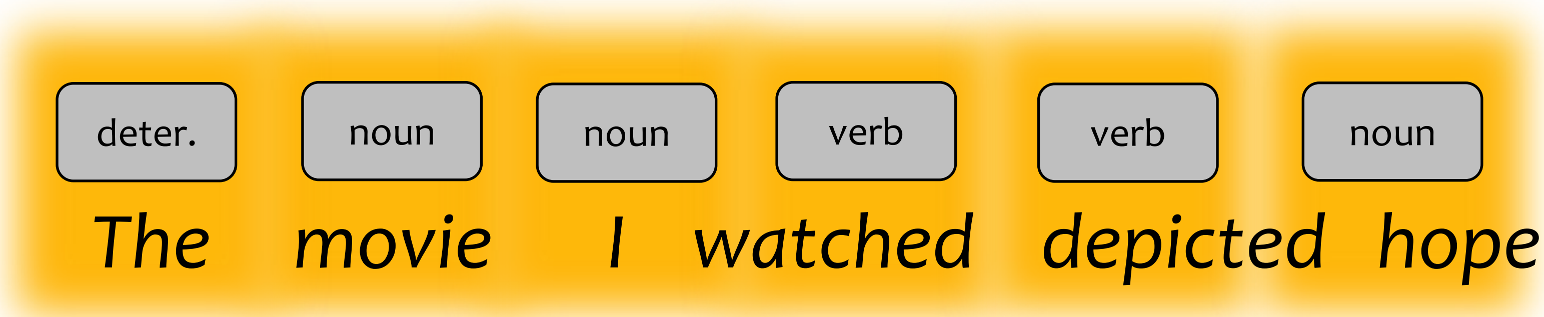
Where do features come from?



Feature Engineering for NLP

Suppose you build a logistic regression model to predict a part-of-speech (POS) tag for each word in a sentence.

What features should you use?



Feature Engineering for NLP

Per-word Features:

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$
`is-capital(w_i)`	1	0	1	0	0	0
`endswith(w_i,"e")`	1	1	0	0	0	1
`endswith(w_i,"d")`	0	0	0	1	1	0
`endswith(w_i,"ed")`	0	0	0	1	1	0
`w_i == "aardvark"`	0	0	0	0	0	0
`w_i == "hope"`	0	0	0	0	0	1
…	…	…	…	…	…	…

deter.	noun	noun	verb	verb	noun
The	*movie*	*I*	*watched*	*depicted*	*hope*

Feature Engineering for NLP

Context Features:

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$
…	…	…	…	…	…	…
`w_i == "watched"`	0	0	0	1	0	0
`w_{i+1} == "watched"`	0	0	1	0	0	0
`w_{i-1} == "watched"`	0	0	0	0	1	0
`w_{i+2} == "watched"`	0	1	0	0	0	0
`w_{i-2} == "watched"`	0	0	0	0	0	1
…	…	…	…	…	…	…

deter.	noun	noun	verb	verb	noun
The	*movie*	*I*	*watched*	*depicted*	*hope*

Feature Engineering for NLP

Context Features:

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$
…	…	…	…	…	…	…
`w`$_{\texttt{i}}$ `== "I"`	0	0	1	0	0	0
`w`$_{\texttt{i+1}}$ `== "I"`	0	1	0	0	0	0
`w`$_{\texttt{i-1}}$ `== "I"`	0	0	0	1	0	0
`w`$_{\texttt{i+2}}$ `== "I"`	1	0	0	0	0	0
`w`$_{\texttt{i-2}}$ `== "I"`	0	0	0	0	1	0
…	…	…	…	…	…	…

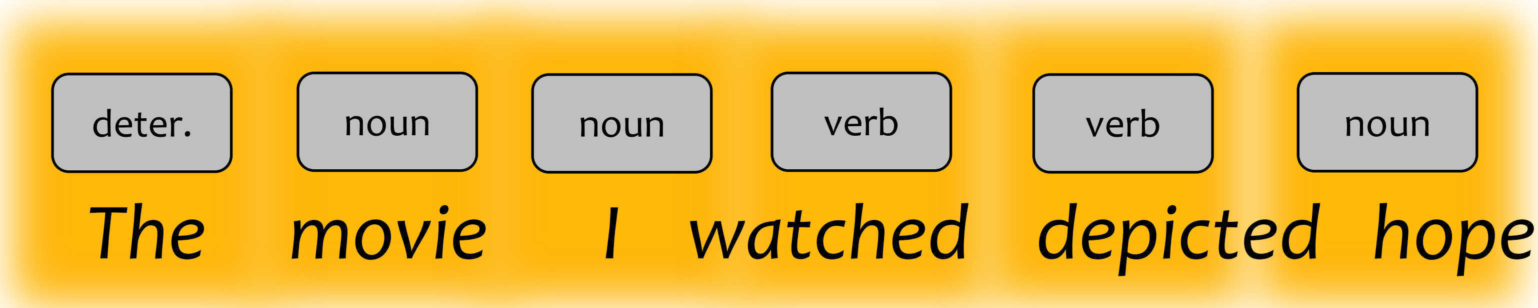
deter.	noun	noun	verb	verb	noun
The	*movie*	*I*	*watched*	*depicted*	*hope*

Table from Manning (2011)

Feature Engineering for NLP

Table 3. Tagging accuracies with different feature templates and other changes on the *WSJ* 19-21 development set.

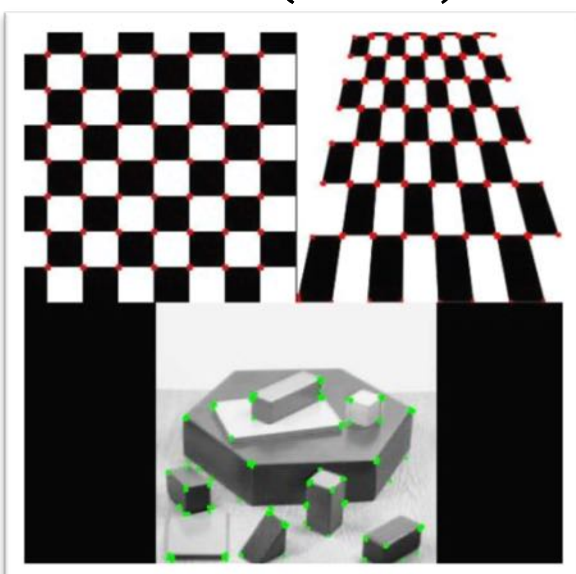
Model	Feature Templates	# Feats	Sent. Acc.	Token Acc.	Unk. Acc.
3GRAMMEMM	See text	248,798	52.07%	96.92%	88.99%
NAACL 2003	See text and [1]	460,552	55.31%	97.15%	88.61%
Replication	See text and [1]	460,551	55.62%	97.18%	88.92%
Replication′	+rareFeatureThresh = 5	482,364	55.67%	97.19%	88.96%
5W	$+\langle t_0, w_{-2}\rangle, \langle t_0, w_2\rangle$	730,178	56.23%	97.20%	89.03%
5WSHAPES	$+\langle t_0, s_{-1}\rangle, \langle t_0, s_0\rangle, \langle t_0, s_{+1}\rangle$	731,661	56.52%	97.25%	89.81%
5WSHAPESDS	+ distributional similarity	737,955	56.79%	97.28%	90.46%



Feature Engineering for CV

Edge detection (Canny)

Corner Detection (Harris)



Figures from http://opencv.org

Feature Engineering for CV

Scale Invariant Feature Transform (SIFT)

Figure 3: Model images of planar objects are shown in the top row. Recognition results below show model outlines and image keys used for matching.

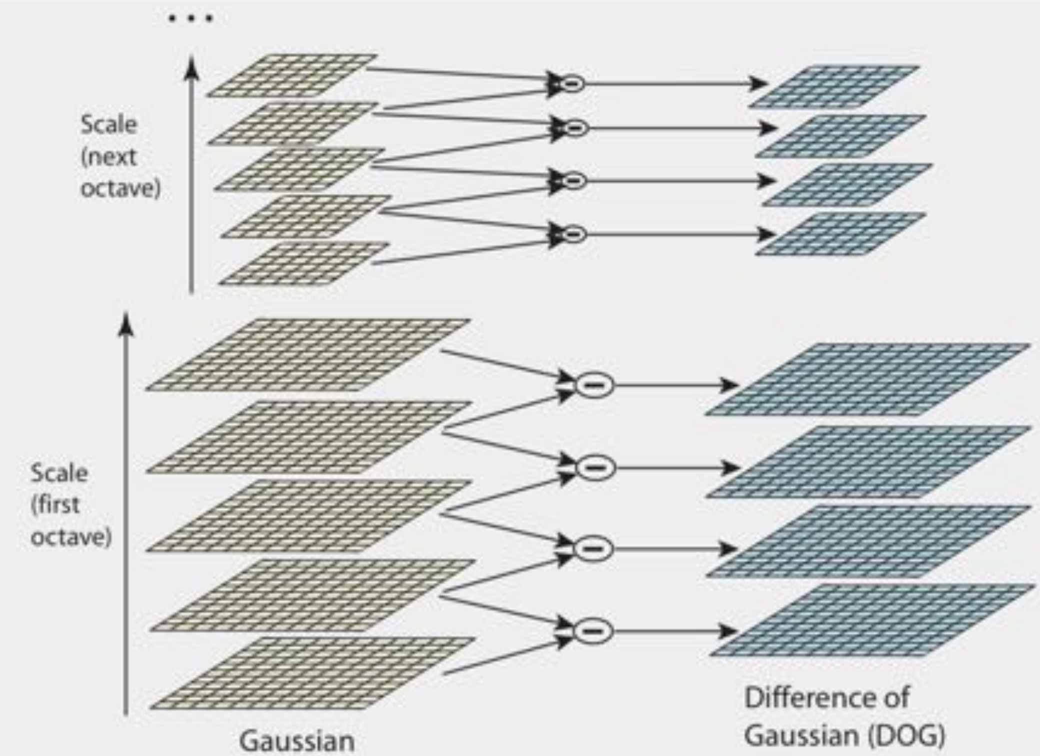


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

Figure from Lowe (1999) and Lowe (2004)

NON-LINEAR FEATURES

Nonlinear Features

- aka. "nonlinear basis functions"
- So far, input was always $\mathbf{x} = [x_1, \ldots, x_M]$
- **Key Idea:** let input be some function of **x**
  - original input: $\mathbf{x} \in \mathbb{R}^M$
  - new input: $\mathbf{x}' \in \mathbb{R}^{M'}$

    where $M' > M$ (usually)
  - define $\mathbf{x}' = b(\mathbf{x}) = [b_1(\mathbf{x}), b_2(\mathbf{x}), \ldots, b_{M'}(\mathbf{x})]$

    where $b_i : \mathbb{R}^M \rightarrow \mathbb{R}$ is any function
- **Examples:** (M = 1)

polynomial	$b_j(x) = x^j \quad \forall j \in \{1, \ldots, J\}$
radial basis function	$b_j(x) = \exp\left(\frac{-(x-\mu_j)^2}{2\sigma_j^2}\right)$
sigmoid	$b_j(x) = \frac{1}{1+\exp(-\omega_j x)}$
log	$b_j(x) = \log(x)$

For a linear model: still a linear function of b(**x**) even though a nonlinear function of **x**

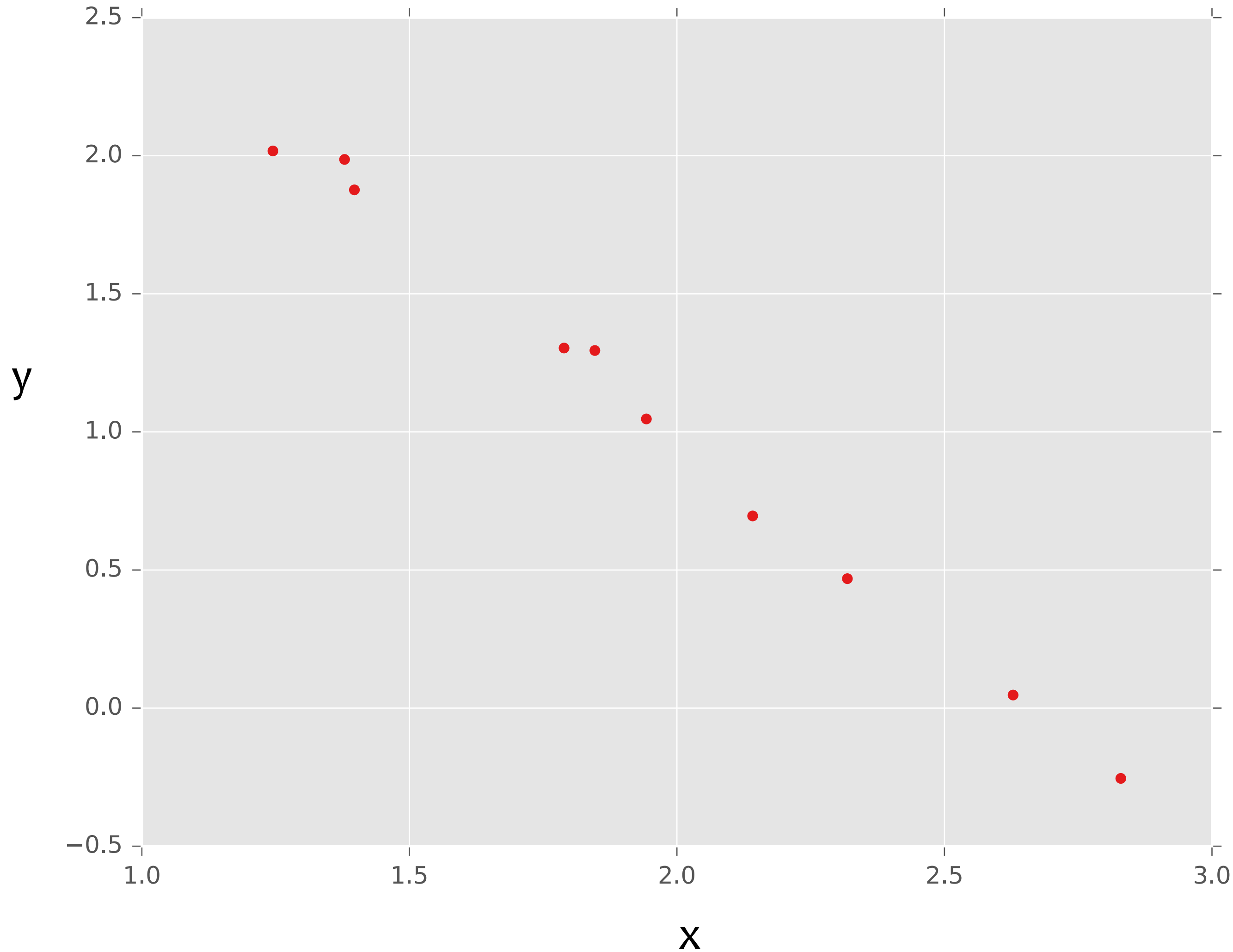
Examples:
- Perceptron
- Linear regression
- Logistic regression

Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b
where f(.) is a polynomial
basis function

y	x
2.0	1.2
1.3	1.7
0.1	2.7
1.1	1.9

true "unknown" target function is linear with negative slope and gaussian noise

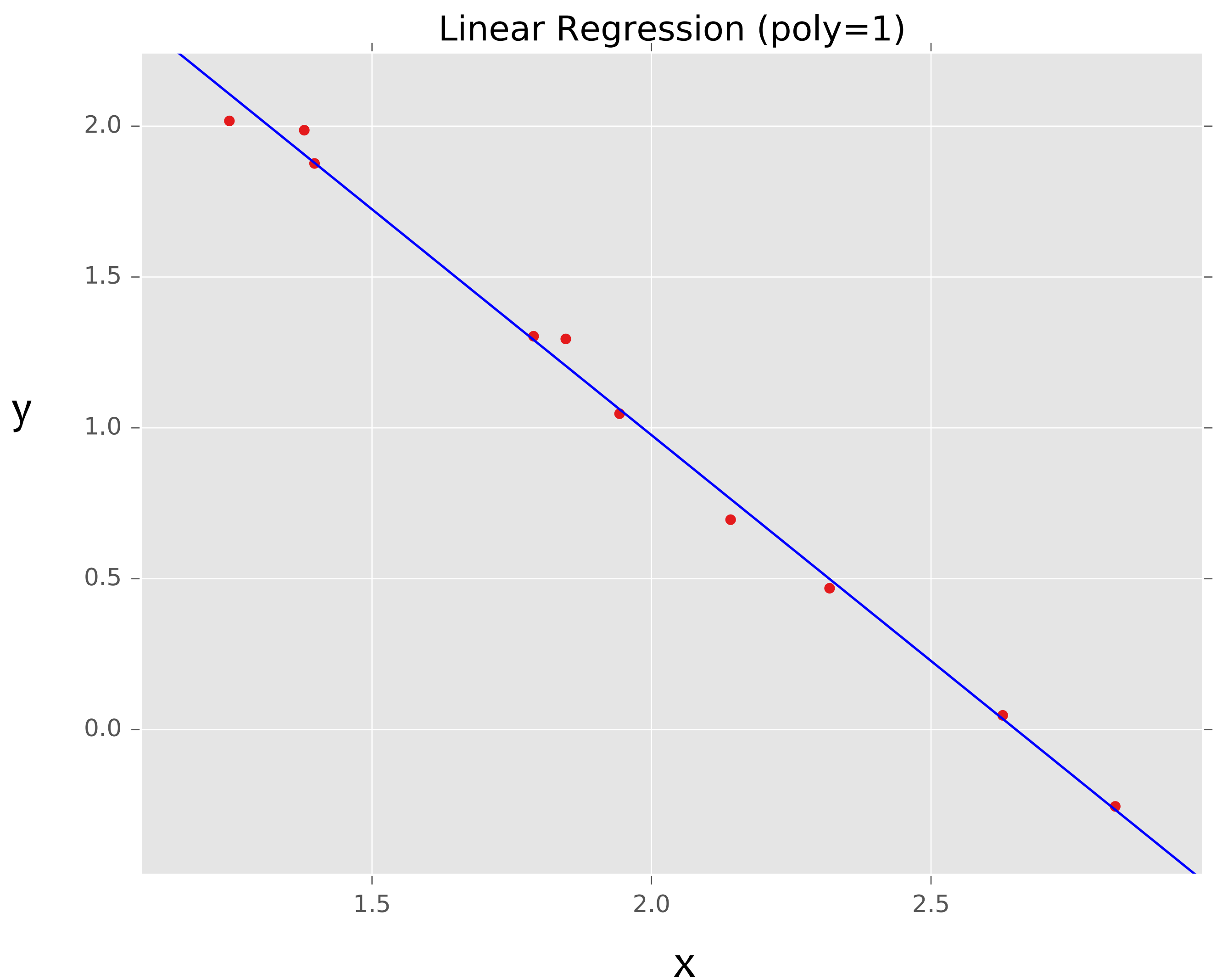


Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T f(\mathbf{x}) + b$
where f(.) is a polynomial
basis function

y	x
2.0	1.2
1.3	1.7
0.1	2.7
1.1	1.9

true "unknown" target function is linear with negative slope and gaussian noise

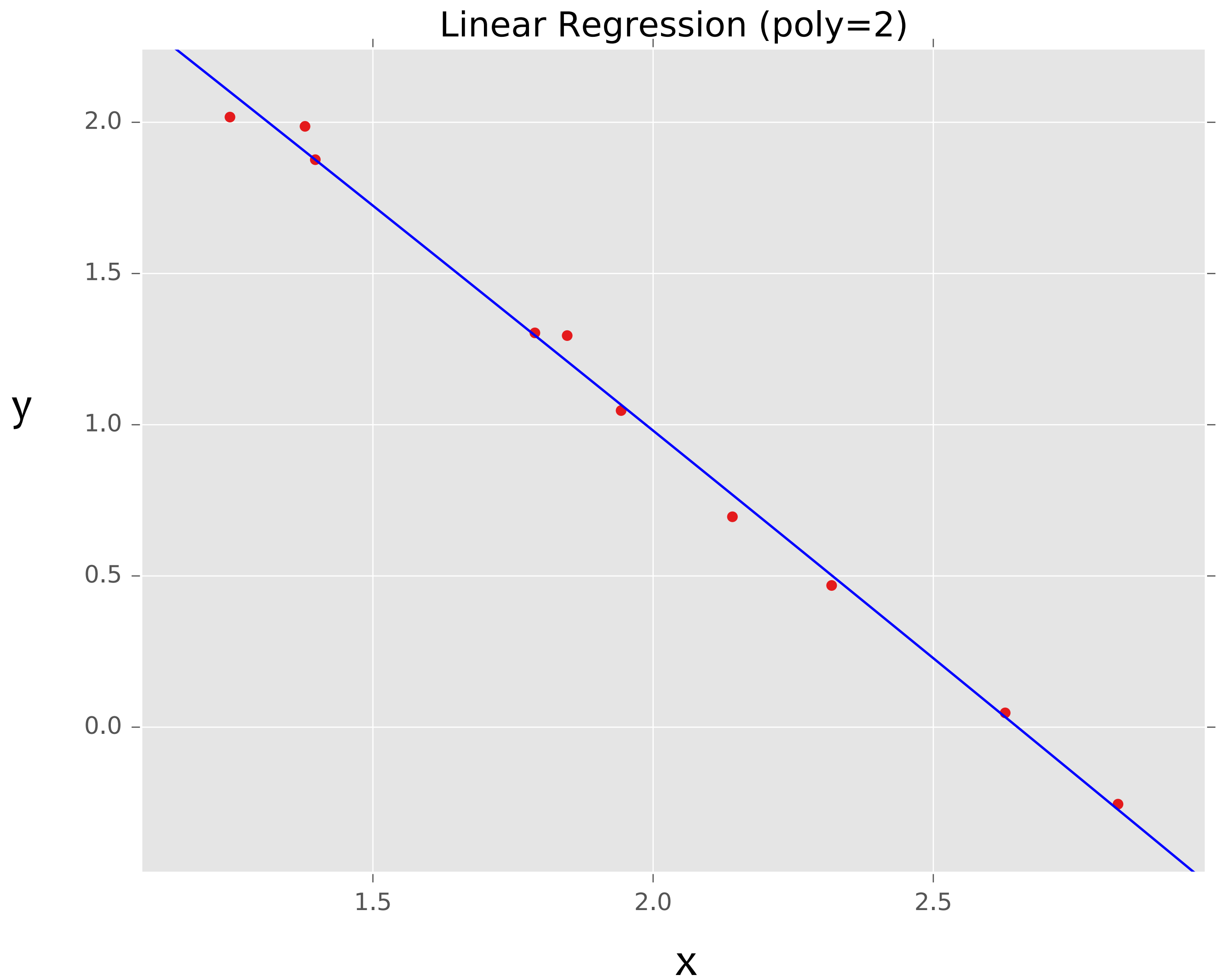


Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T f(\mathbf{x}) + b$
where f(.) is a polynomial basis function

y	x	x^2
2.0	1.2	$(1.2)^2$
1.3	1.7	$(1.7)^2$
0.1	2.7	$(2.7)^2$
1.1	1.9	$(1.9)^2$

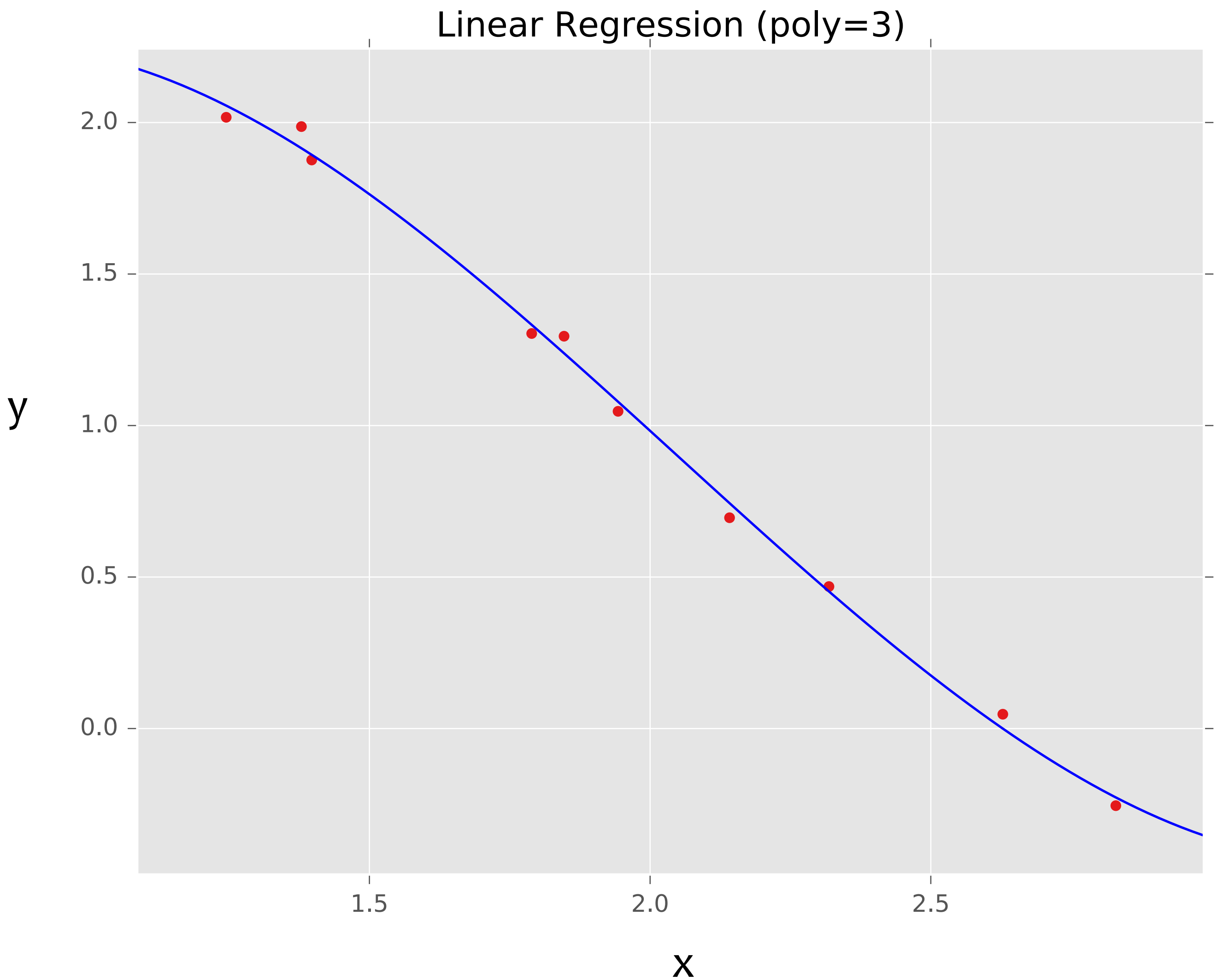
true "unknown" target function is linear with negative slope and gaussian noise



Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T f(\mathbf{x}) + b$ where f(.) is a polynomial basis function

y	x	x^2	x^3
2.0	1.2	$(1.2)^2$	$(1.2)^3$
1.3	1.7	$(1.7)^2$	$(1.7)^3$
0.1	2.7	$(2.7)^2$	$(2.7)^3$
1.1	1.9	$(1.9)^2$	$(1.9)^3$



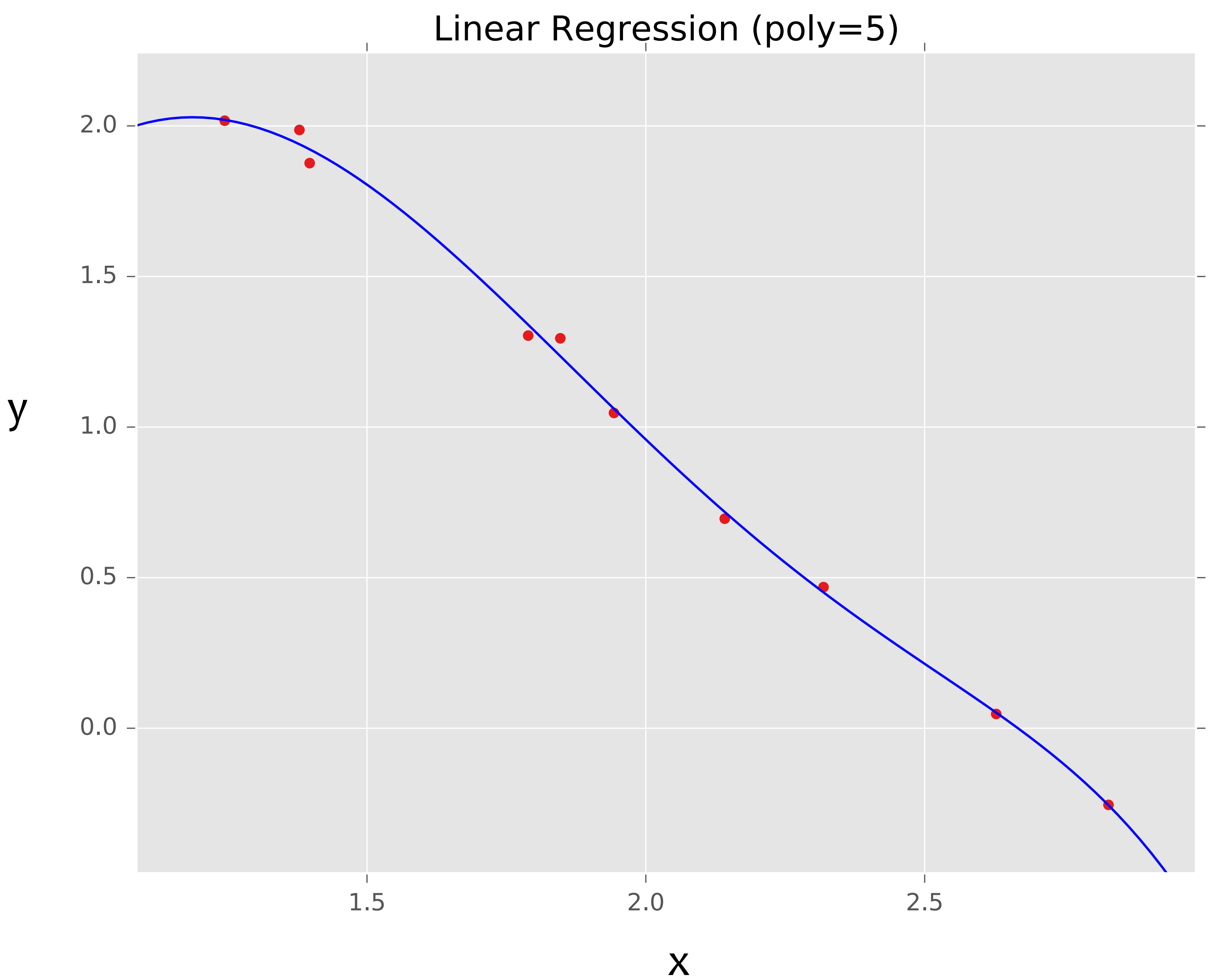
true "unknown" target function is linear with negative slope and gaussian noise

Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b
where f(.) is a polynomial basis function

y	x	x^2	…	x^5
2.0	1.2	$(1.2)^2$	…	$(1.2)^5$
1.3	1.7	$(1.7)^2$	…	$(1.7)^5$
0.1	2.7	$(2.7)^2$	…	$(2.7)^5$
1.1	1.9	$(1.9)^2$	…	$(1.9)^5$

true "unknown" target function is linear with negative slope and gaussian noise

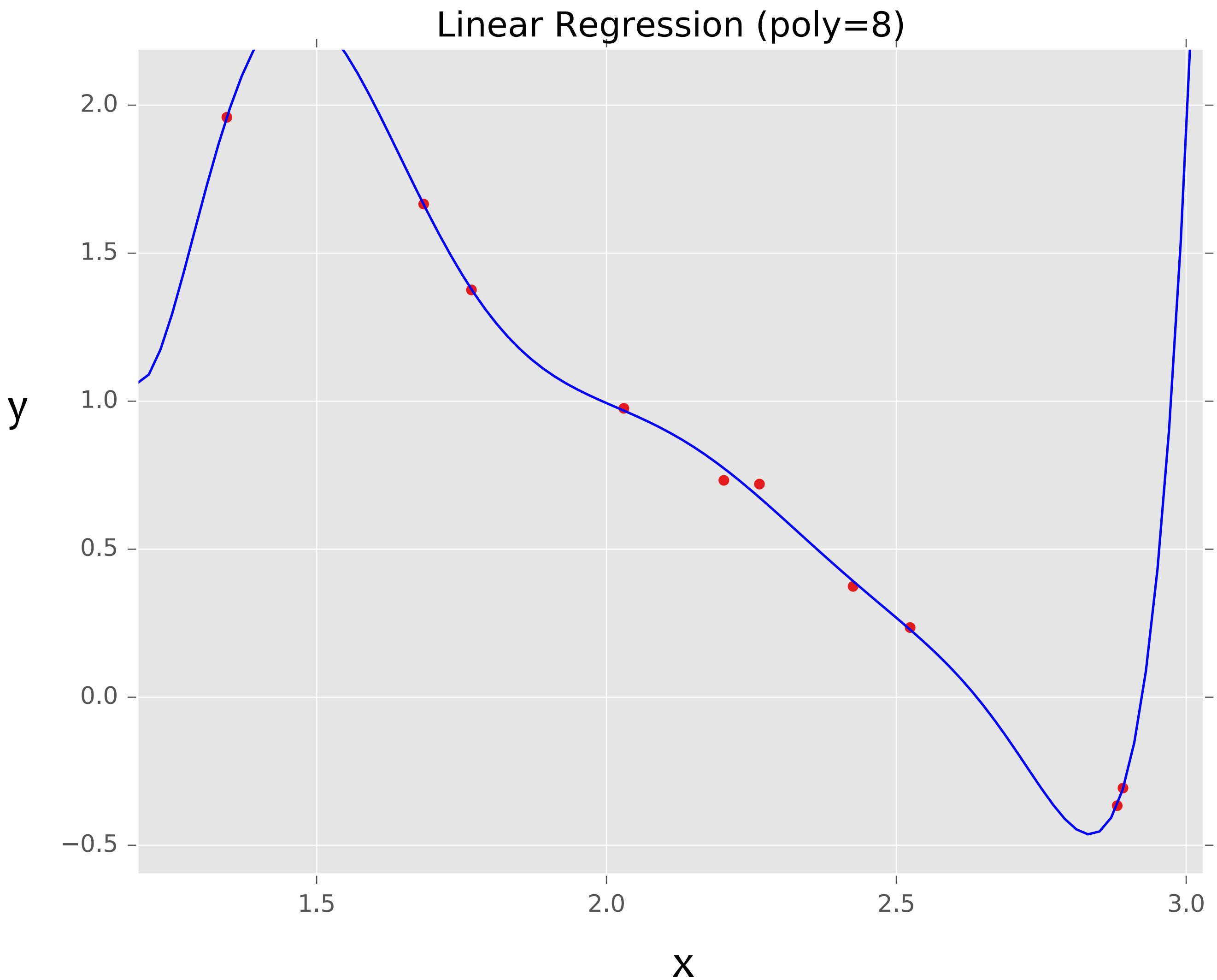


Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b
where f(.) is a polynomial basis function

y	x	x^2	…	x^8
2.0	1.2	$(1.2)^2$	…	$(1.2)^8$
1.3	1.7	$(1.7)^2$	…	$(1.7)^8$
0.1	2.7	$(2.7)^2$	…	$(2.7)^8$
1.1	1.9	$(1.9)^2$	…	$(1.9)^8$

true "unknown" target function is linear with negative slope and gaussian noise

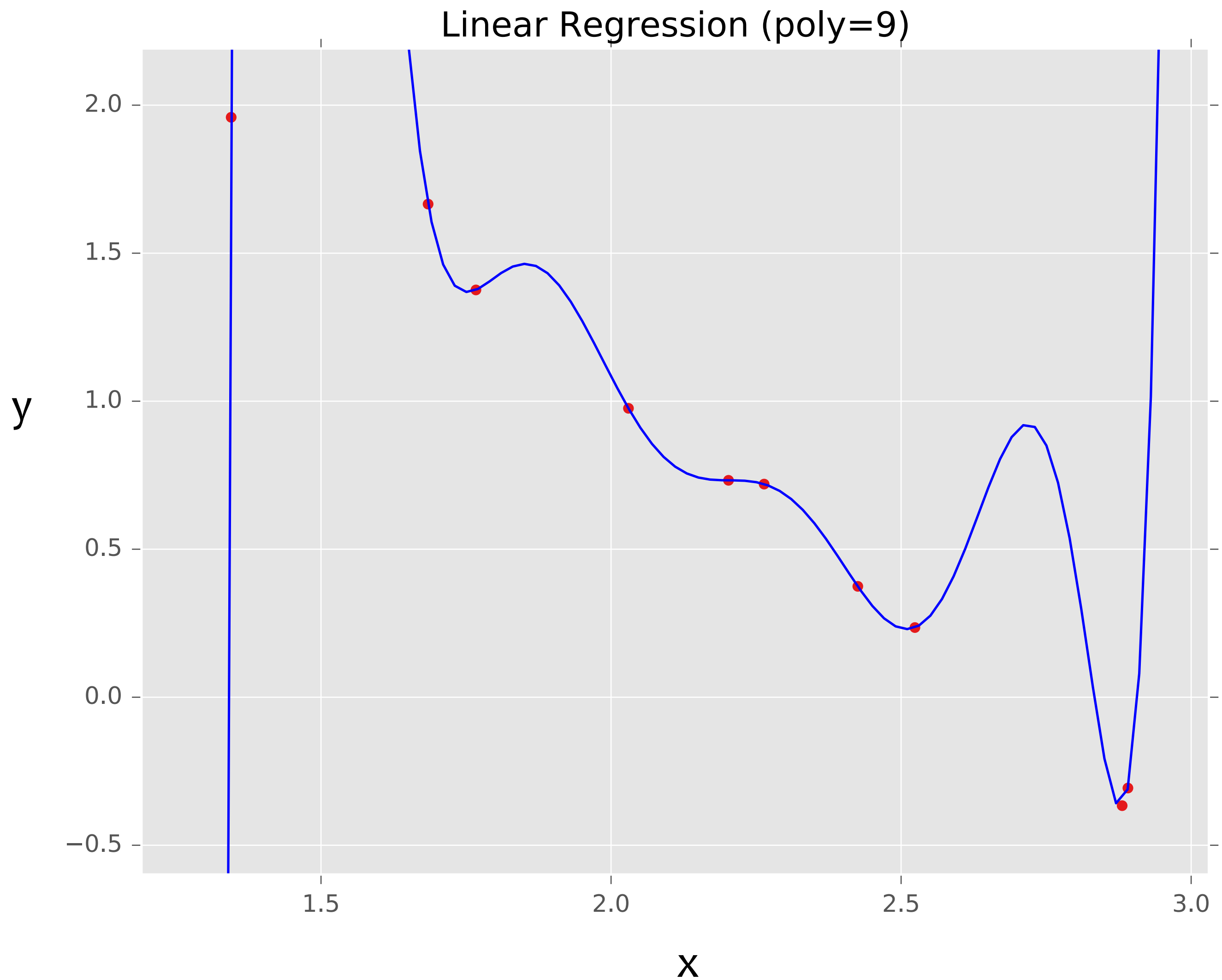


Example: Linear Regression

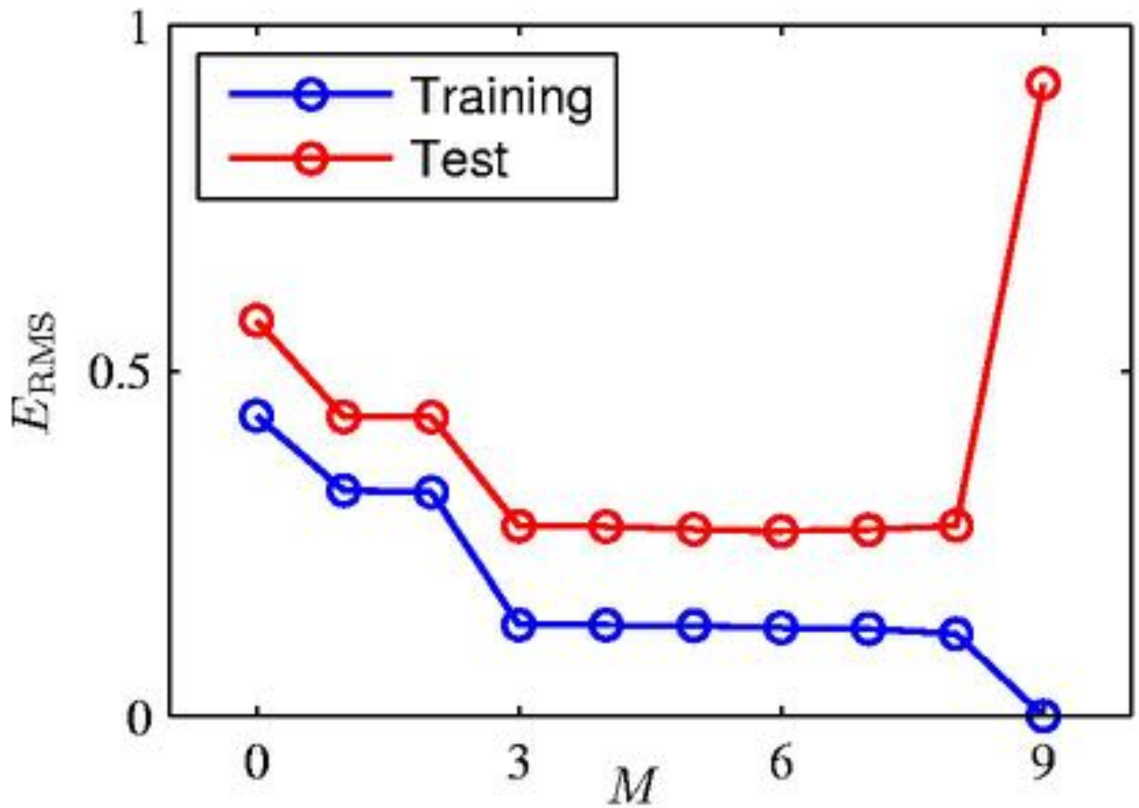
Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b
where f(.) is a polynomial
basis function

y	x	x^2	…	x^9
2.0	1.2	$(1.2)^2$	…	$(1.2)^9$
1.3	1.7	$(1.7)^2$	…	$(1.7)^9$
0.1	2.7	$(2.7)^2$	…	$(2.7)^9$
1.1	1.9	$(1.9)^2$	…	$(1.9)^9$

true "unknown" target function is linear with negative slope and gaussian noise



Over-fitting



Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

Slide courtesy of William Cohen

Polynomial Coefficients

	$M=0$	$M=1$	$M=3$	$M=9$
θ_0	0.19	0.82	0.31	0.35
θ_1		-1.27	7.99	232.37
θ_2			-25.43	-5321.83
θ_3			17.37	48568.31
θ_4				-231639.30
θ_5				640042.26
θ_6				-1061800.52
θ_7				1042400.18
θ_8				-557682.99
θ_9				125201.43

Slide courtesy of William Cohen

Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b where f(.) is a polynomial basis function

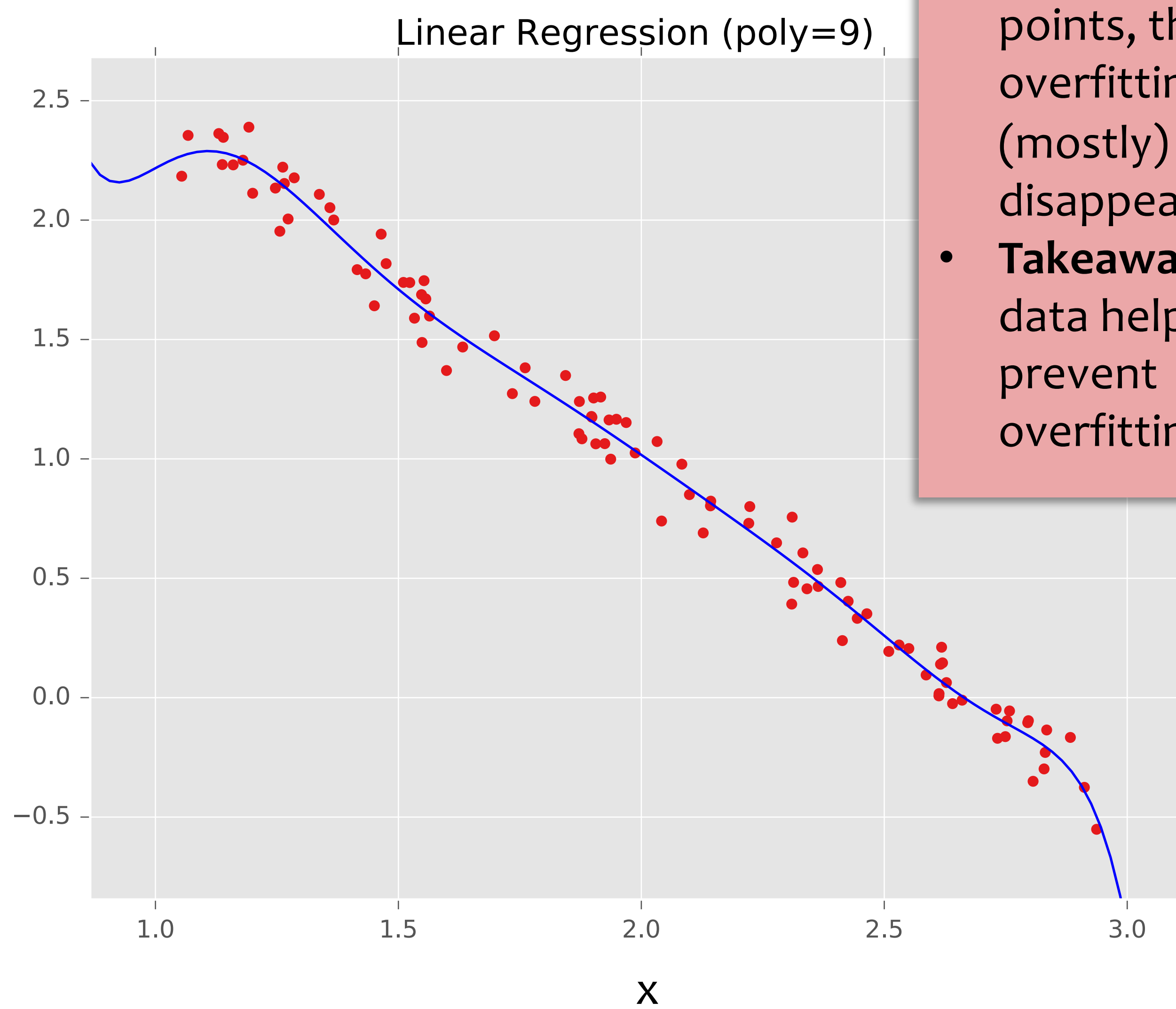
i	y	x	…	x^9
1	2.0	1.2	…	$(1.2)^9$
2	1.3	1.7	…	$(1.7)^9$
…	…	…	…	…
10	1.1	1.9	…	$(1.9)^9$

- With just N = 10 points we overfit!
- But with N = 100 points, the overfitting (mostly) disappears
- **Takeaway:** more data helps prevent overfitting

Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T f(\mathbf{x}) + b$ where f(.) is a polynomial basis function

i	y	x	…	x^9
1	2.0	1.2	…	$(1.2)^9$
2	1.3	1.7	…	$(1.7)^9$
3	0.1	2.7	…	$(2.7)^9$
4	1.1	1.9	…	$(1.9)^9$
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
98	…	…	…	…
99	…	…	…	…
100	0.9	1.5	…	$(1.5)^9$



- With just N = 10 points we overfit!
- But with N = 100 points, the overfitting (mostly) disappears
- **Takeaway:** more data helps prevent overfitting

REGULARIZATION

Overfitting

Definition: The problem of **overfitting** is when the model captures the noise in the training data instead of the underlying structure

Overfitting can occur in all the models we’ve seen so far:

- Decision Trees (e.g. when tree is too deep)
- KNN (e.g. when k is small)
- Perceptron (e.g. when sample isn’t representative)
- Linear Regression (e.g. with nonlinear features)
- Logistic Regression (e.g. with many rare features)

Motivation: Regularization

Example: Stock Prices

- Suppose we wish to predict Google's stock price at time t+1
- **What features should we use?** (putting all computational concerns aside)
  - Stock prices of all other stocks at times t, t-1, t-2, … , t - k
  - Mentions of Google with positive / negative sentiment words in all newspapers and social media outlets

- Do we believe that **all** of these features are going to be useful?

Motivation: Regularization

- **Occam's Razor:** prefer the simplest hypothesis

- What does it mean for a hypothesis (or model) to be **simple**?
  1. small number of features (**model selection**)
  2. small number of "important" features (**shrinkage**)

Regularization

- **Given** objective function: J(θ)
- **Goal** is to find: $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$
- **Key idea:** Define regularizer r(θ) s.t. we tradeoff between fitting the data and keeping the model simple
- **Choose form of r(θ):**
  - Example: q-norm (usually p-norm) $r(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_q = \left[\sum_{m=1}^{M} ||\theta_m||^q\right]^{(\frac{1}{q})}$

q	$r(\boldsymbol{\theta})$	yields parameters that are...	name	optimization notes
0	$\|\|\boldsymbol{\theta}\|\|_0 = \sum \mathbb{1}(\theta_m \neq 0)$	zero values	L0 reg.	no good computational solutions
1	$\|\|\boldsymbol{\theta}\|\|_1 = \sum \|\theta_m\|$	zero values	L1 reg.	subdifferentiable
2	$(\|\|\boldsymbol{\theta}\|\|_2)^2 = \sum \theta_m^2$	small values	L2 reg.	differentiable

Regularization

Question:

Suppose we are minimizing J'(θ) where

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$$

As λ increases, the minimum of J'(θ) will…

A. … move towards the midpoint between J'(θ) and r(θ)

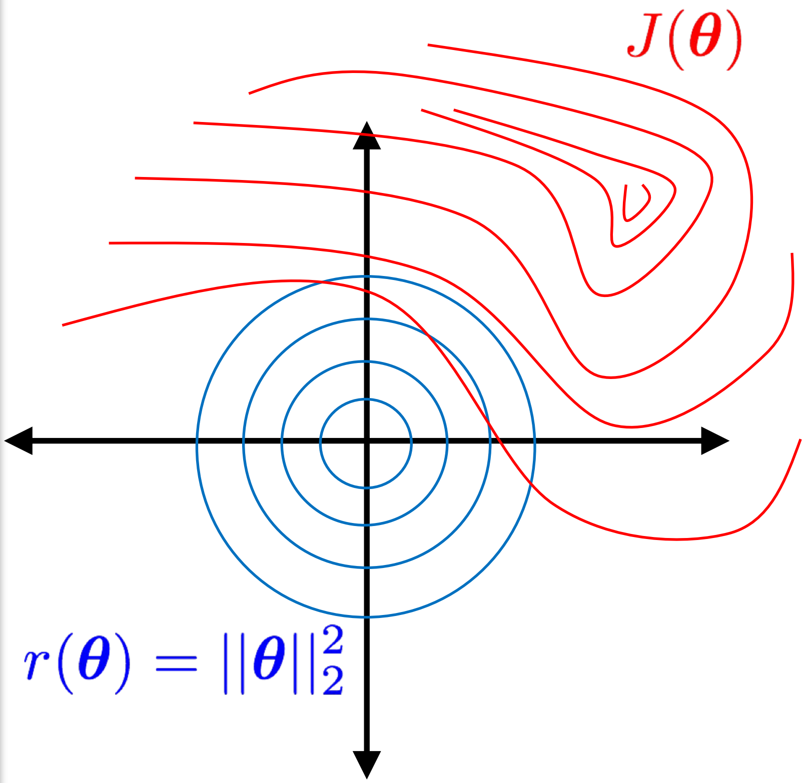
B. … move towards the minimum of J(θ)

C. … move towards the minimum of r(θ)

D. … move towards a theta vector of positive infinities

E. … move towards a theta vector of negative infinities

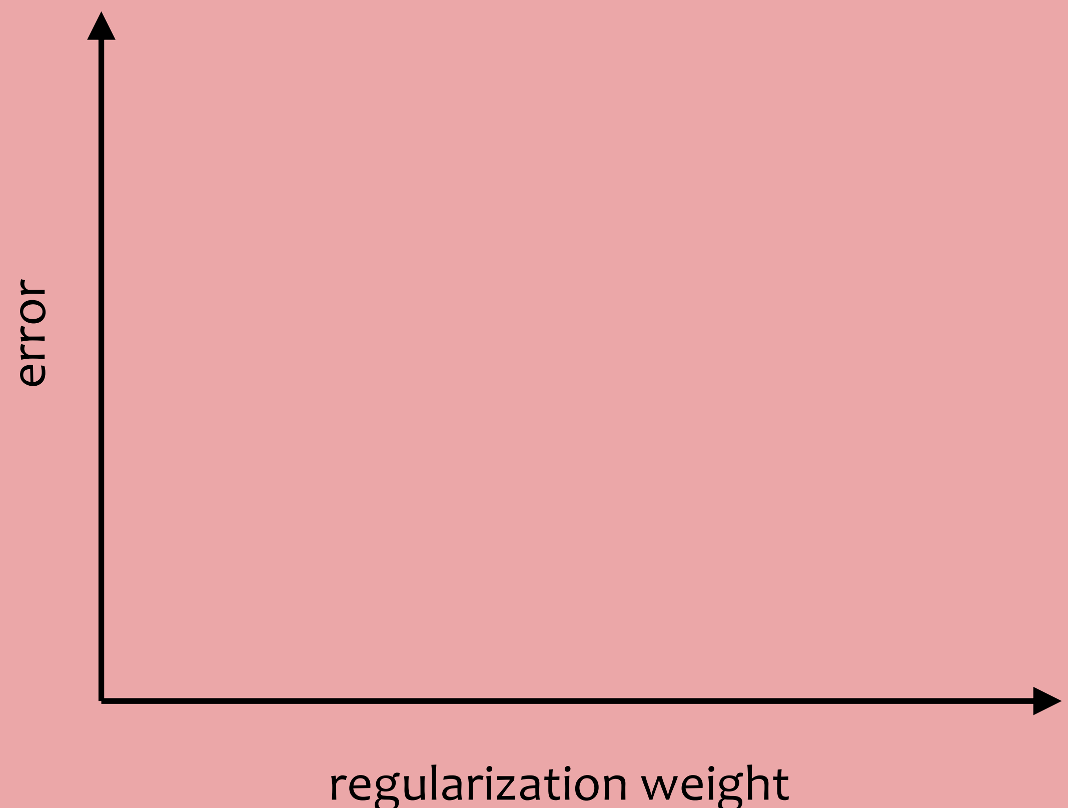
F. … stay the same



Regularization Exercise

In-class Exercise

1. Plot train error vs. regularization weight (cartoon)
2. Plot test error vs . regularization weight (cartoon)



Regularization

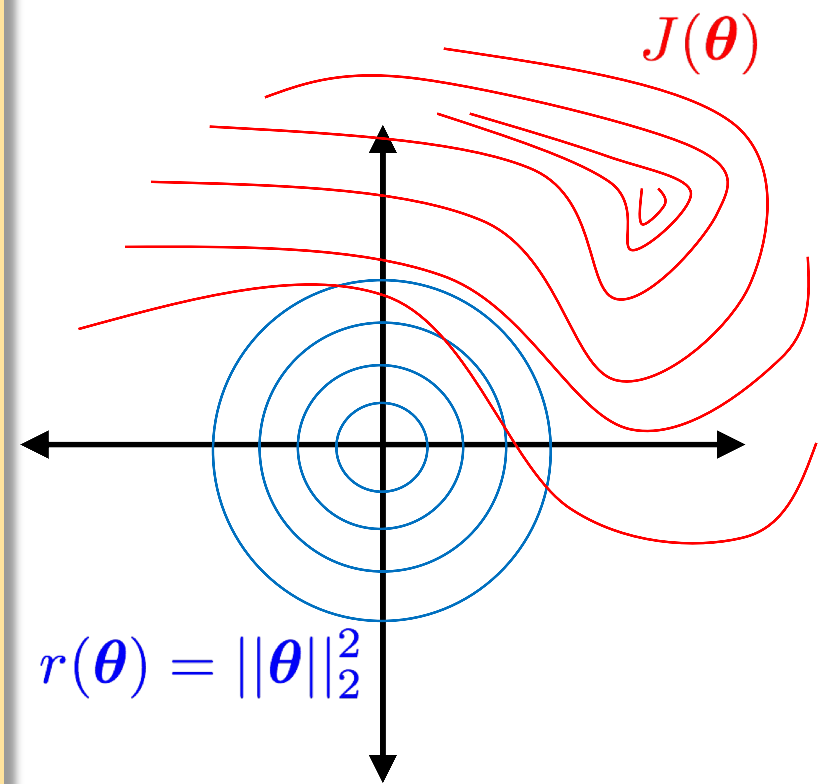
Question:

Suppose we are minimizing J'(θ) where

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$$

As we increase λ from 0, the the validation error will…

A. …increase
B. …decrease
C. …first increase, then decrease
D. …first decrease, then increase
E. …stay the same



Regularization

Don't Regularize the Bias (Intercept) Parameter!

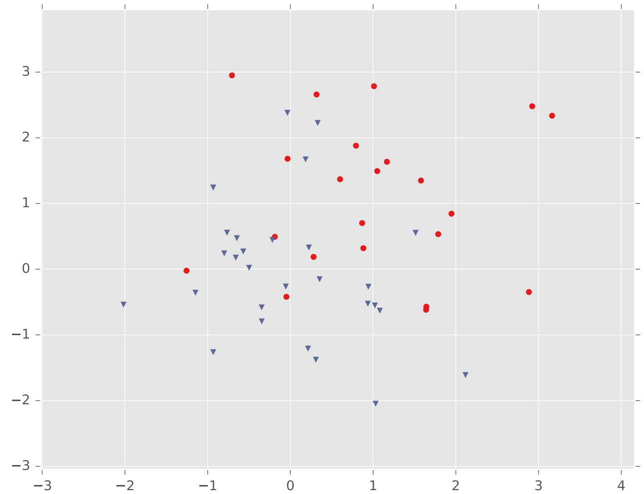
- In our models so far, the bias / intercept parameter is usually denoted by θ_0 -- that is, the parameter for which we fixed $x_0 = 1$
- Regularizers always avoid penalizing this bias / intercept parameter
- Why? Because otherwise the learning algorithms wouldn't be invariant to a shift in the y-values

Whitening Data

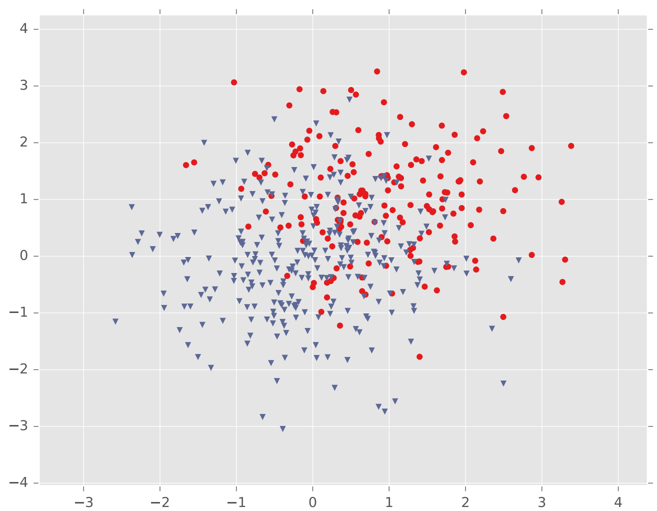
- It's common to *whiten* each feature by subtracting its mean and dividing by its variance
- For regularization, this helps all the features be penalized in the same units
  (e.g. convert both centimeters and kilometers to z-scores)

Example: Logistic Regression

Training Data

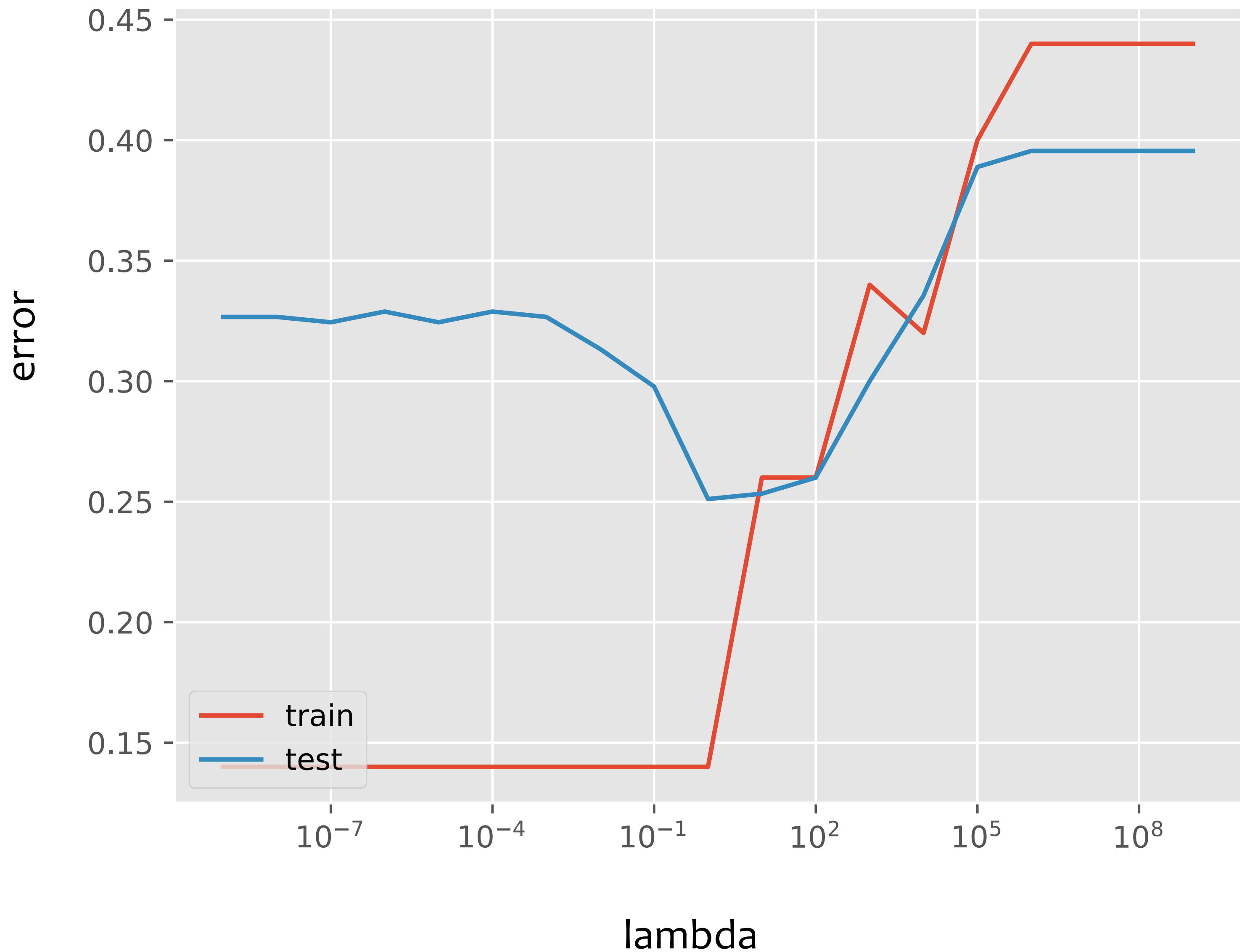


Test Data

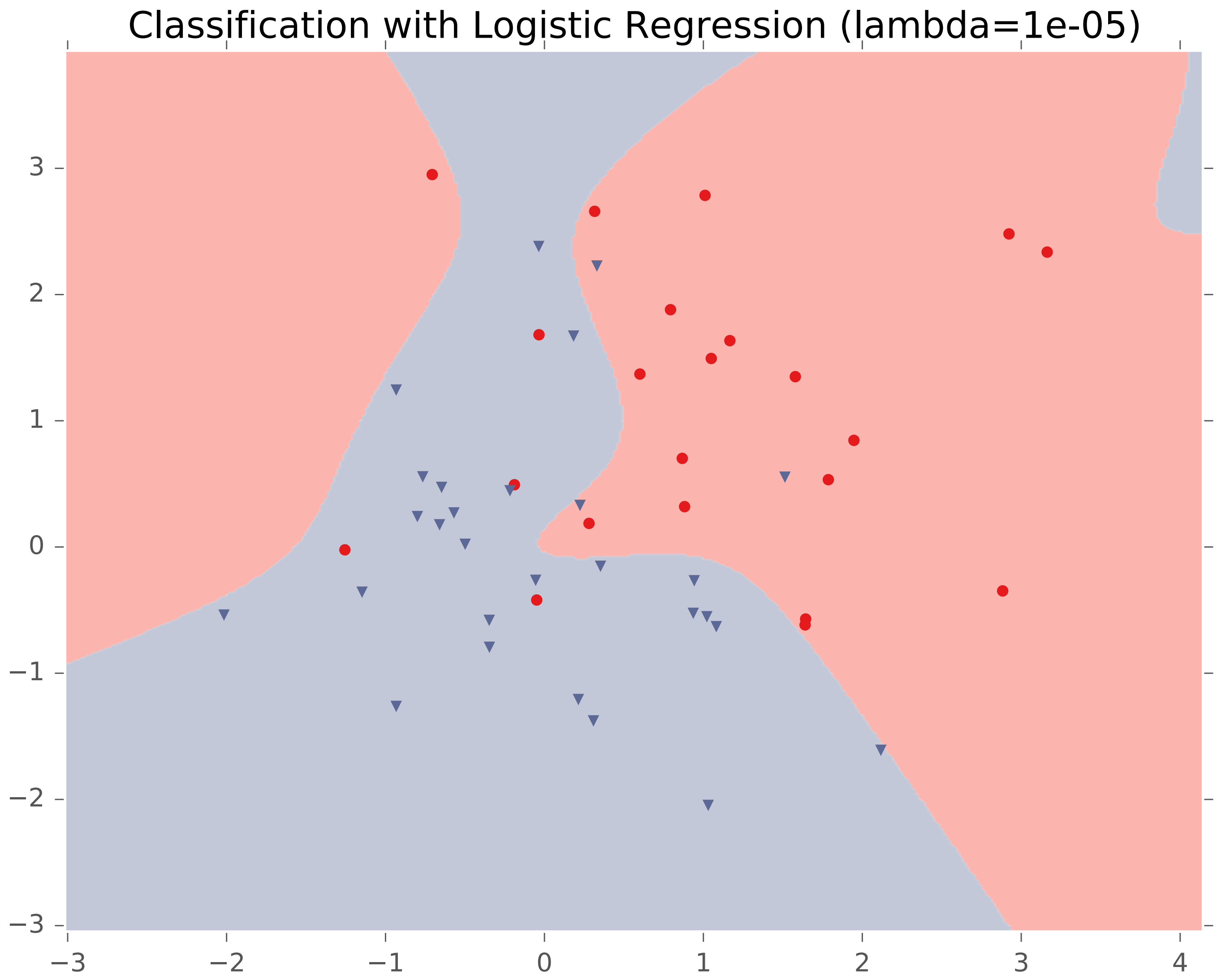


- For this example, we construct **nonlinear features** (i.e. feature engineering)
- Specifically, we add **polynomials up to order 9** of the two original features x_1 and x_2
- Thus our classifier is **linear** in the **high-dimensional feature space,** but the decision boundary is **nonlinear** when visualized in **low-dimensions** (i.e. the original two dimensions)

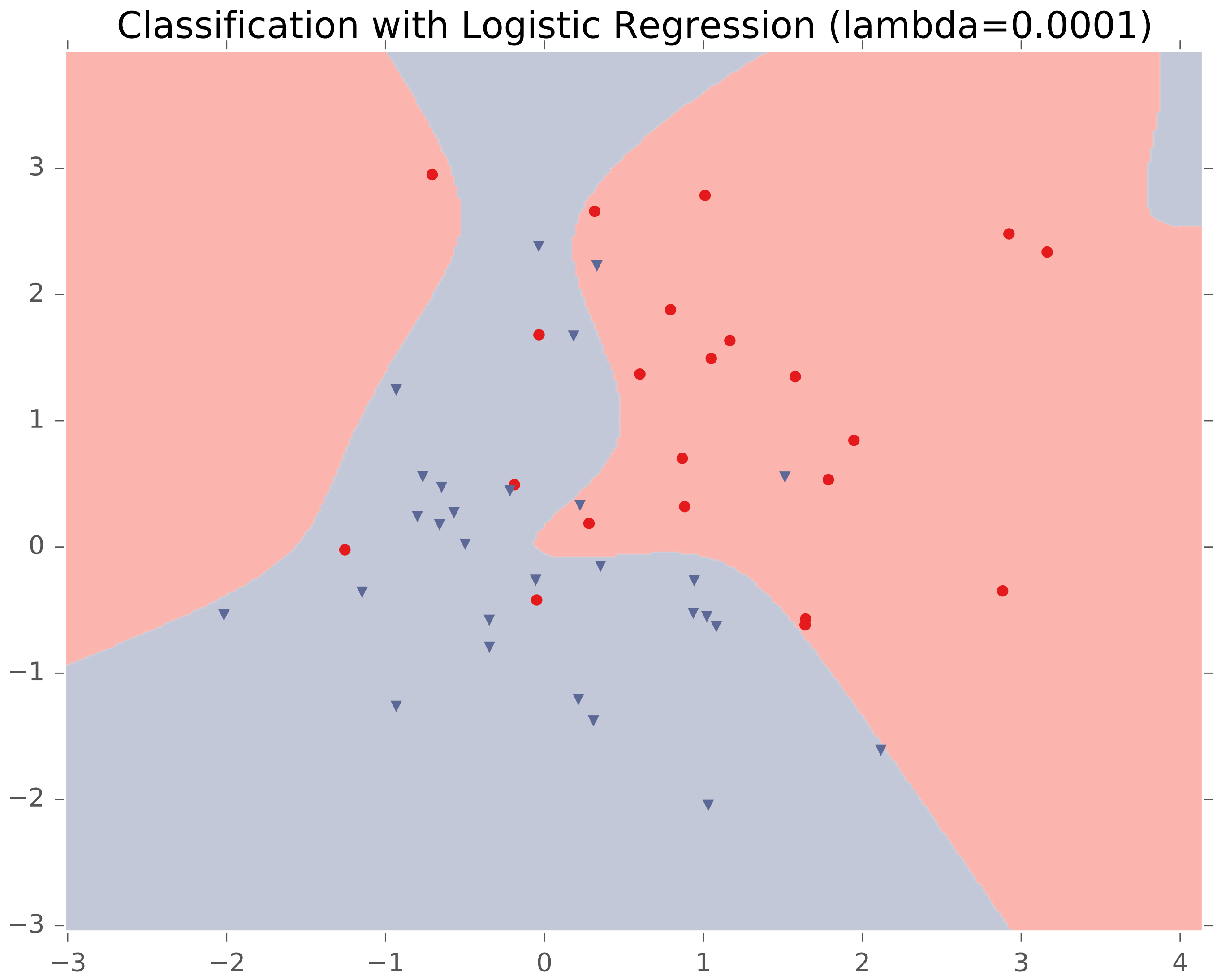
Example: Logistic Regression



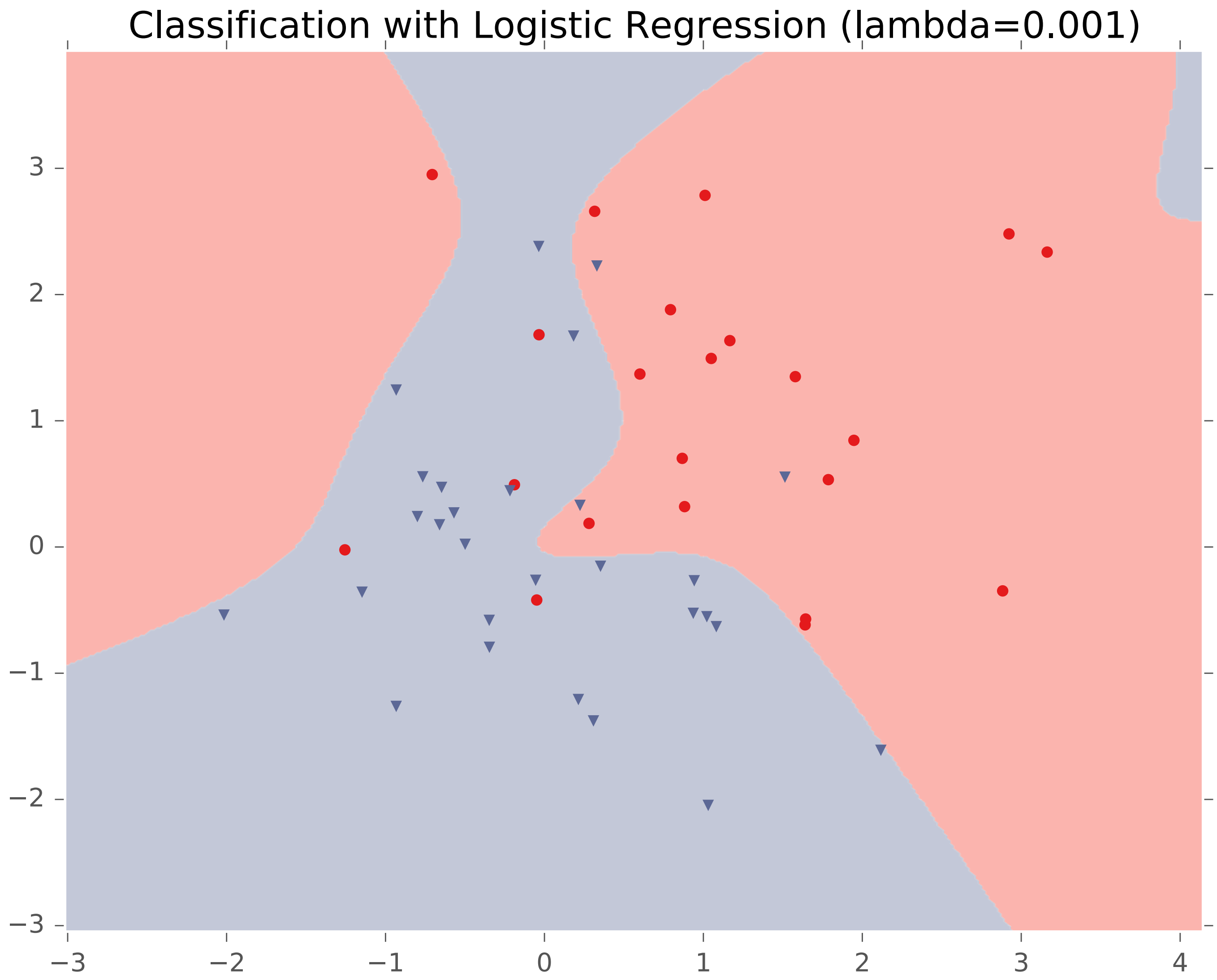
Example: Logistic Regression



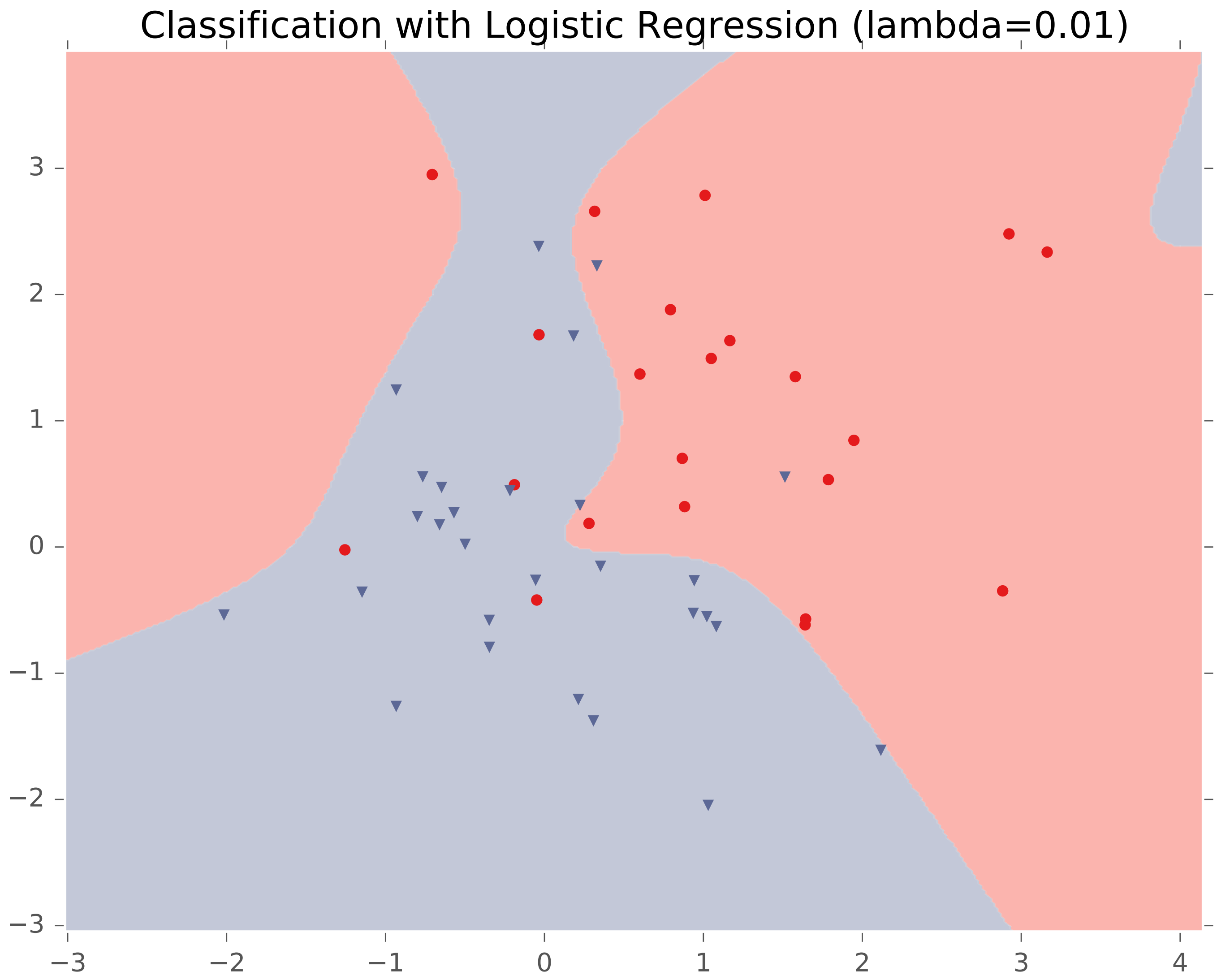
Example: Logistic Regression



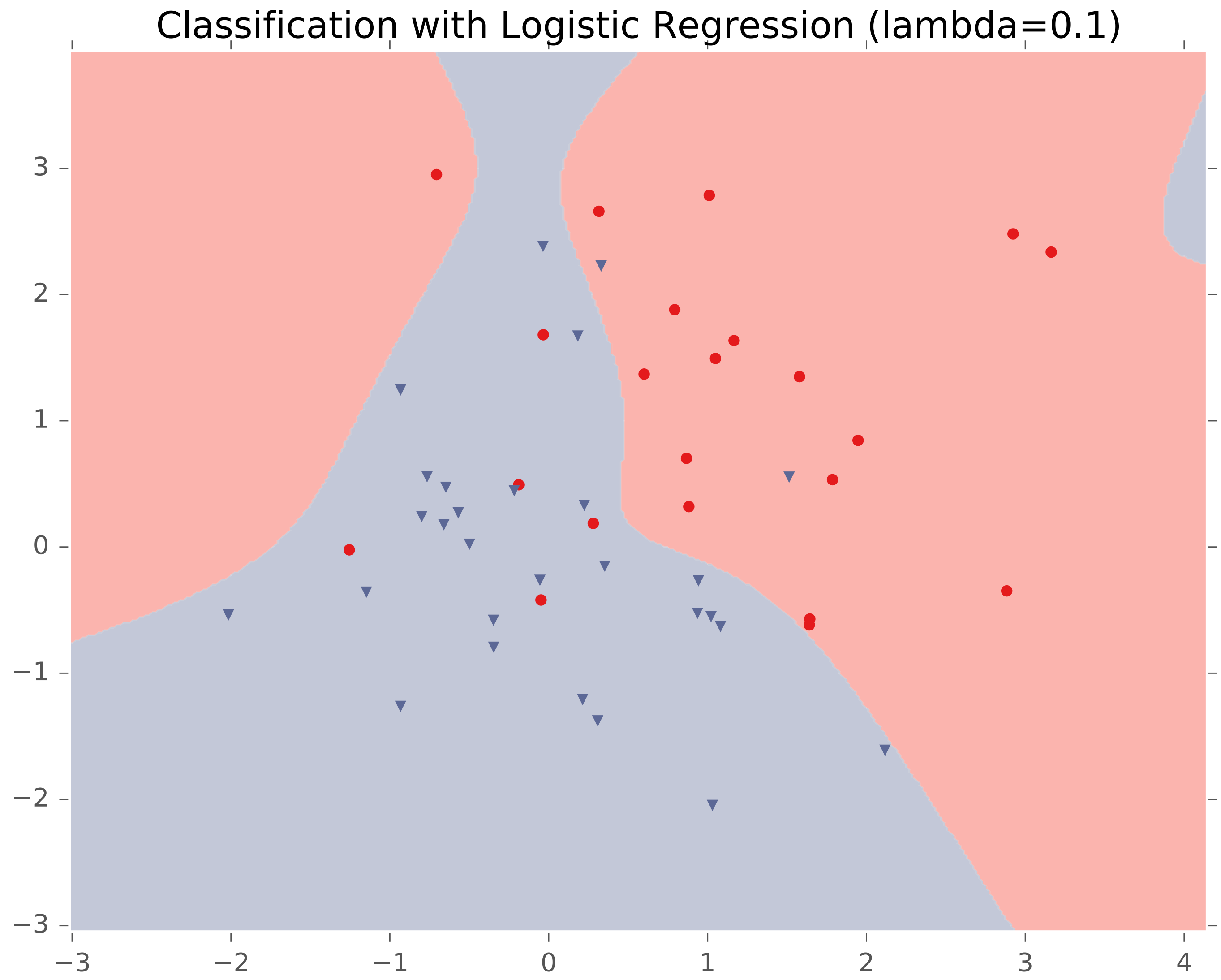
Example: Logistic Regression



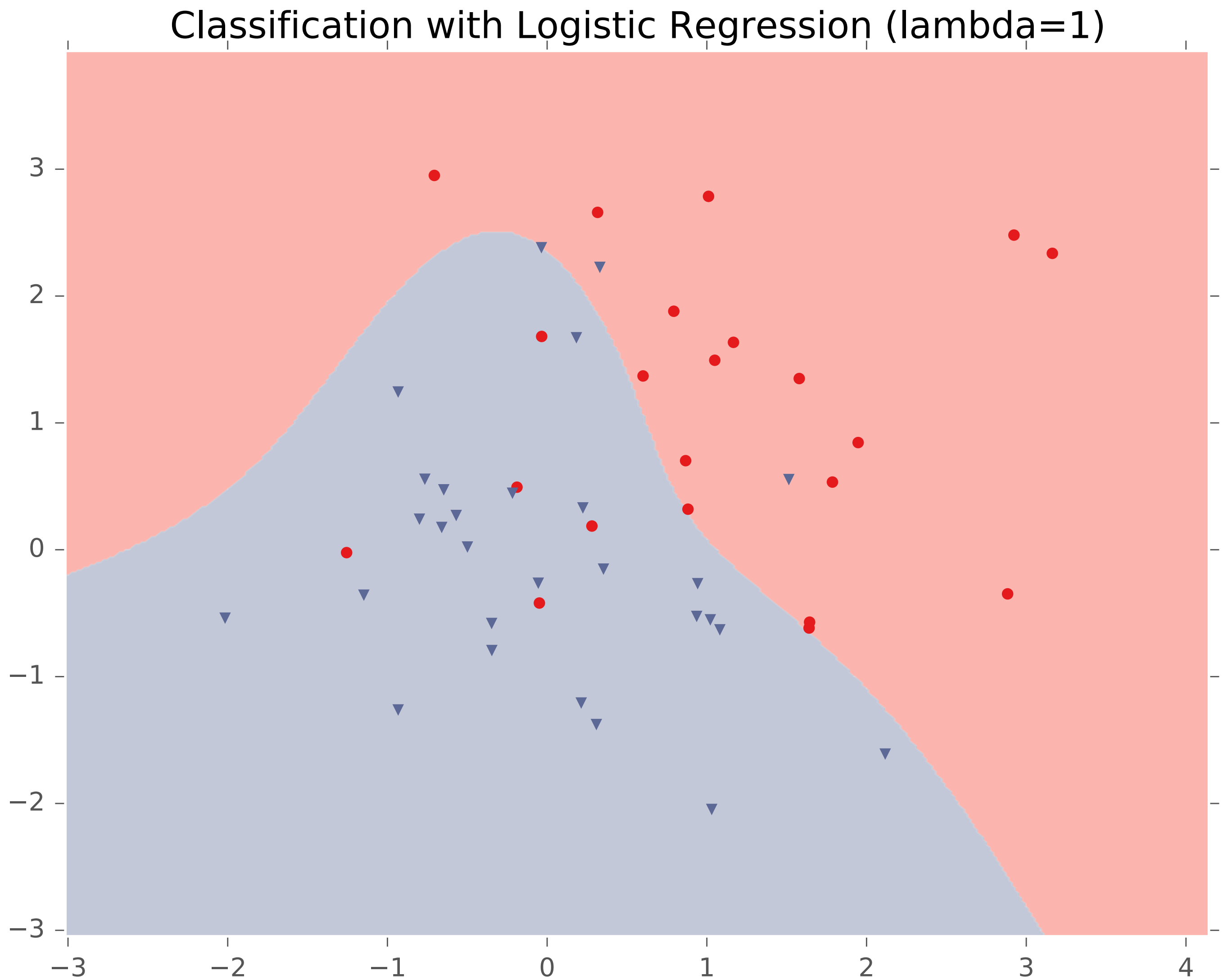
Example: Logistic Regression



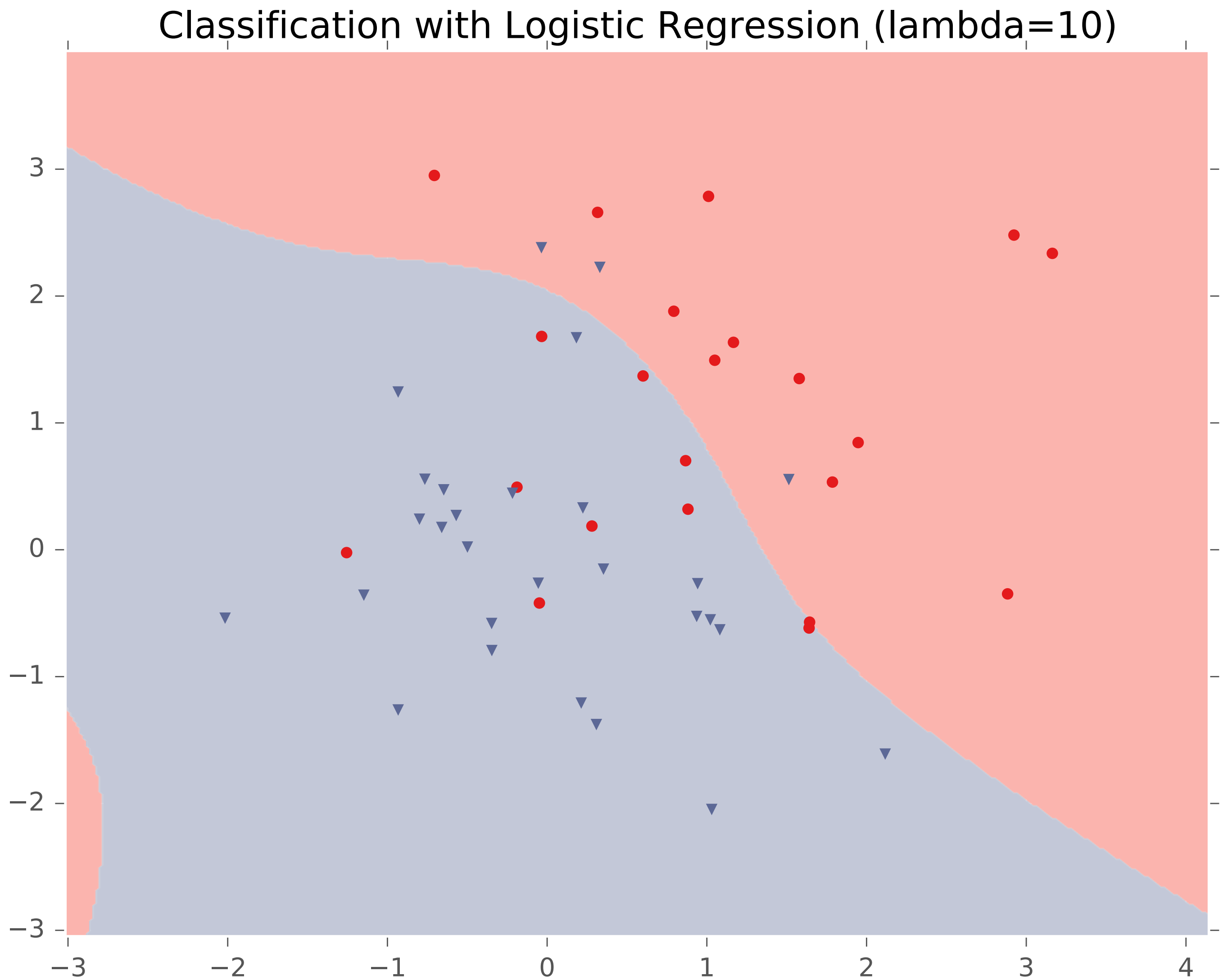
Example: Logistic Regression



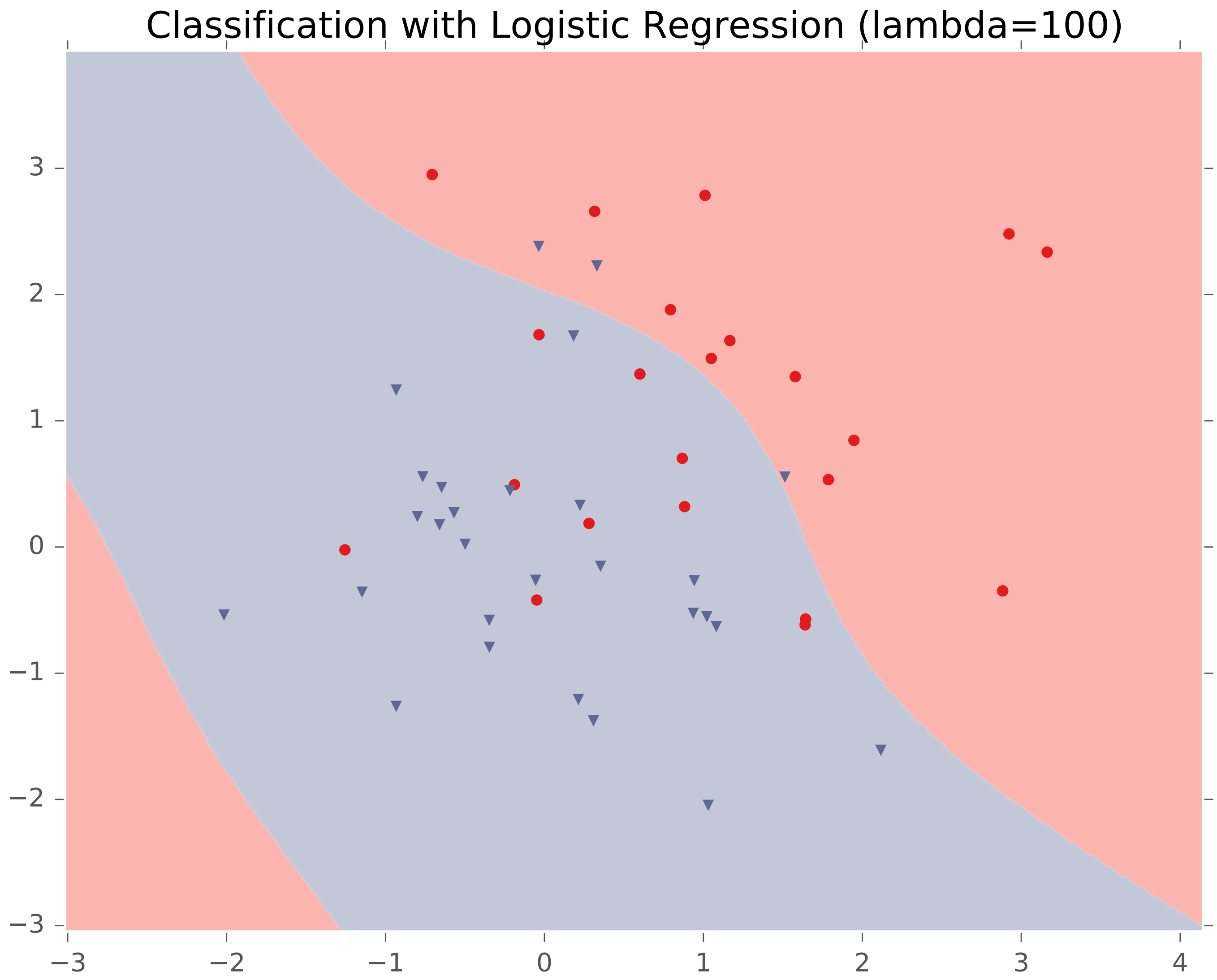
Example: Logistic Regression



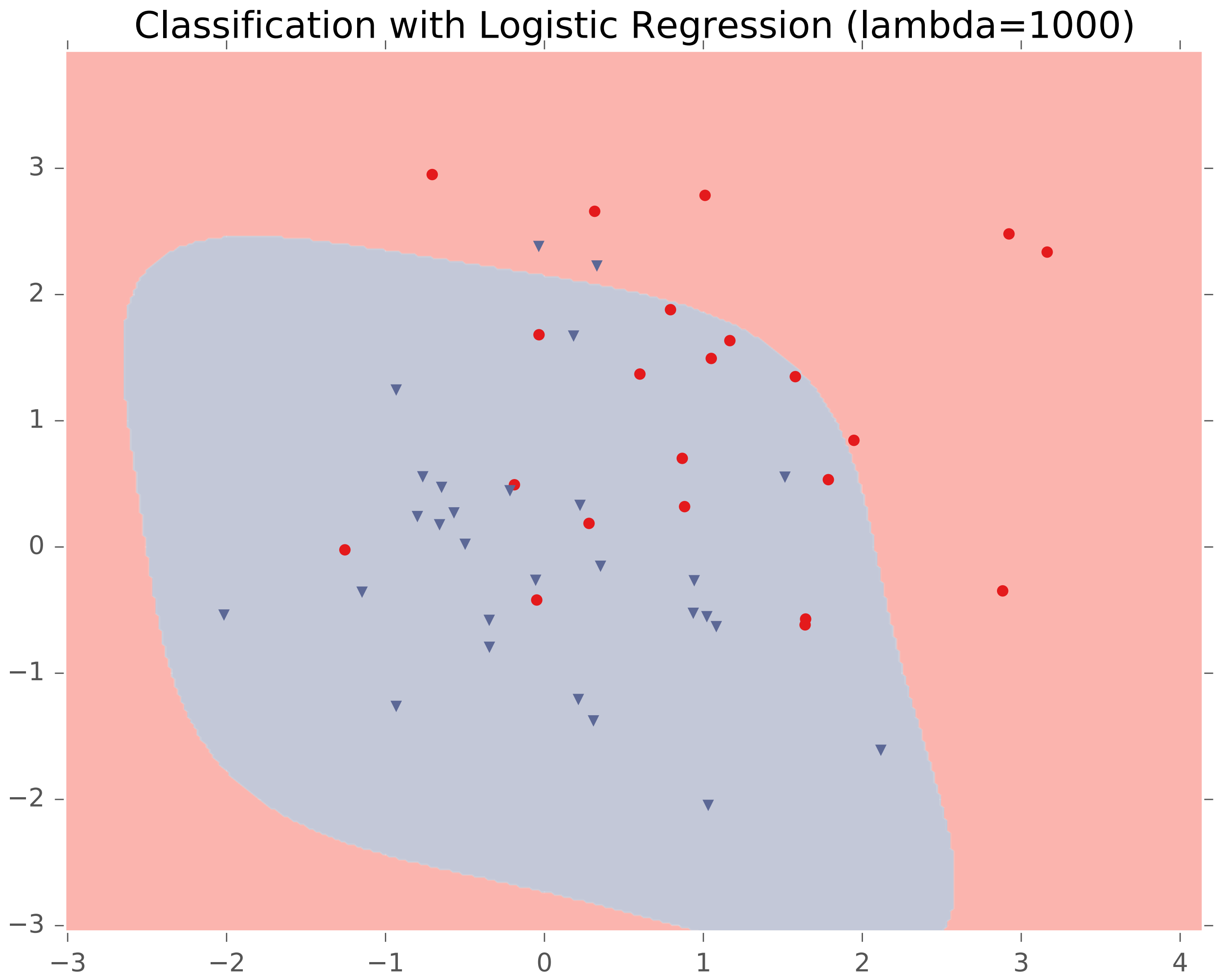
Example: Logistic Regression



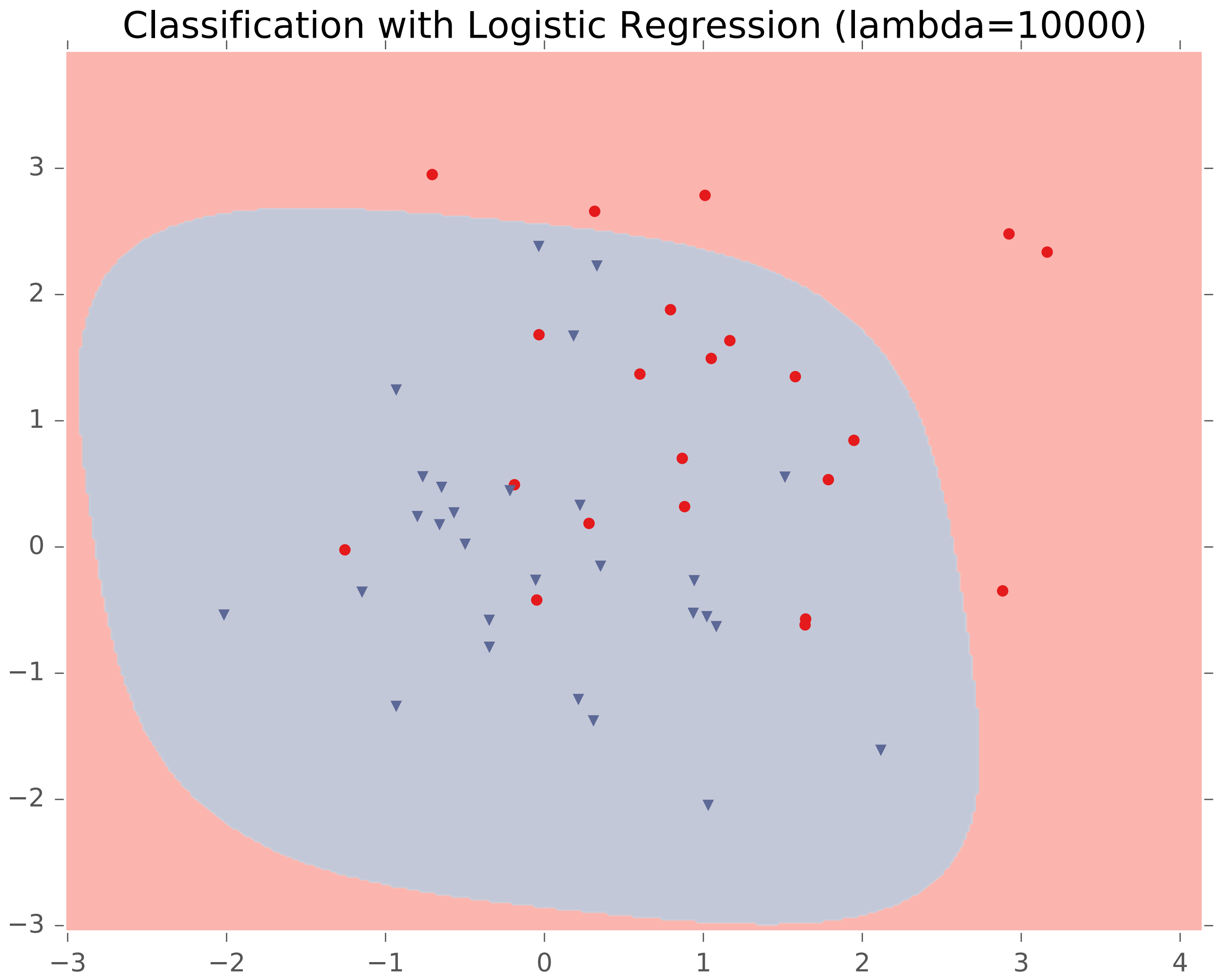
Example: Logistic Regression



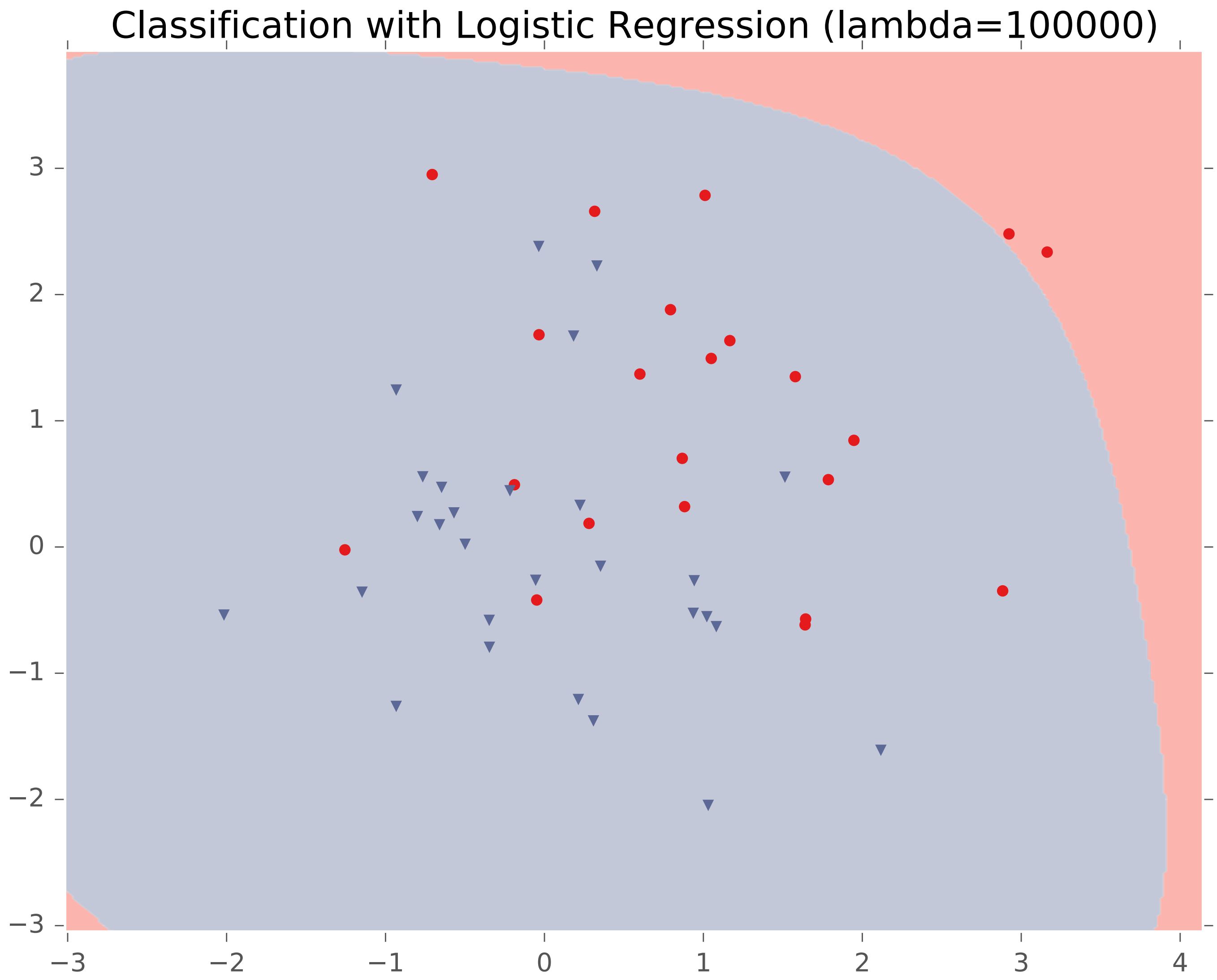
Example: Logistic Regression



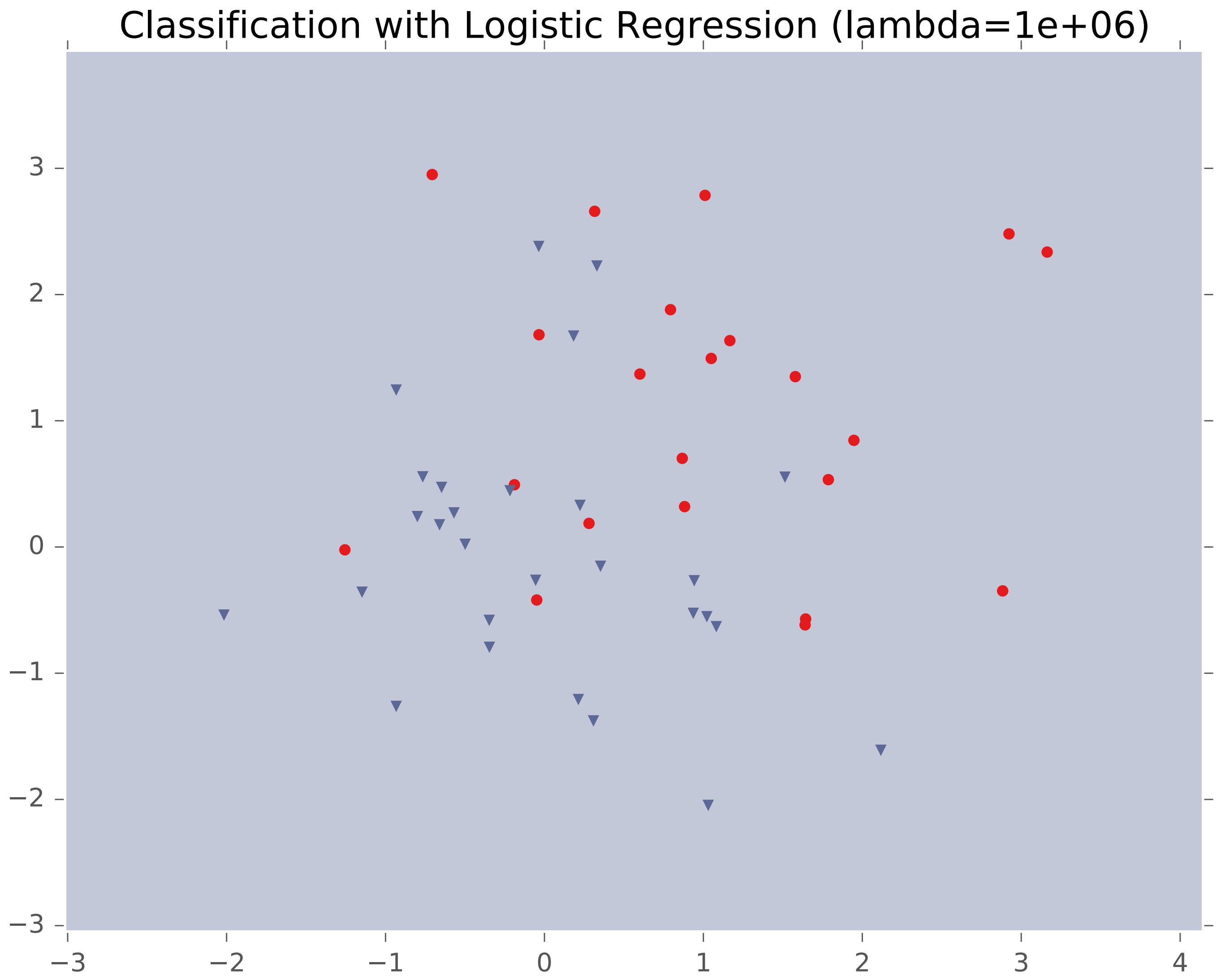
Example: Logistic Regression



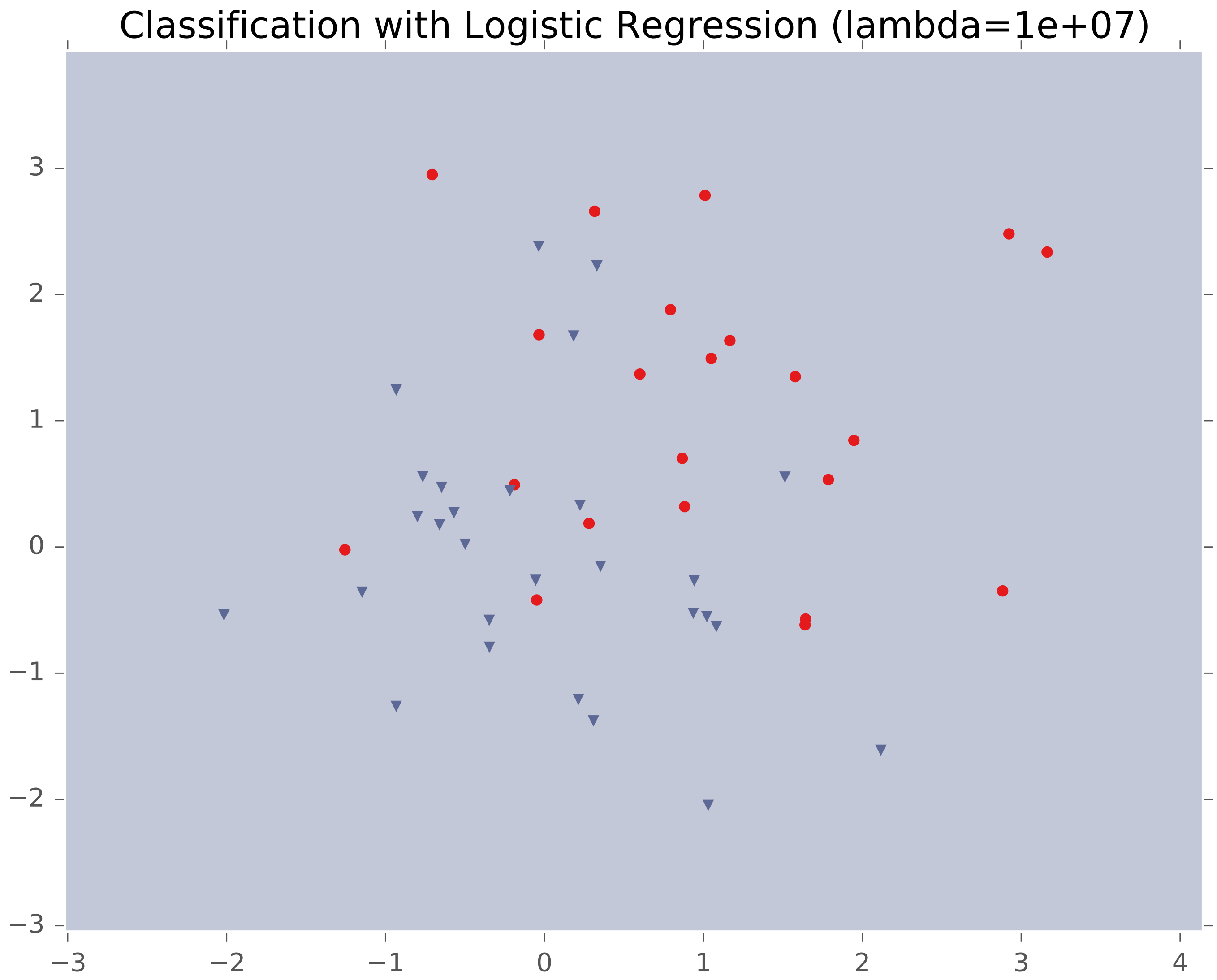
Example: Logistic Regression



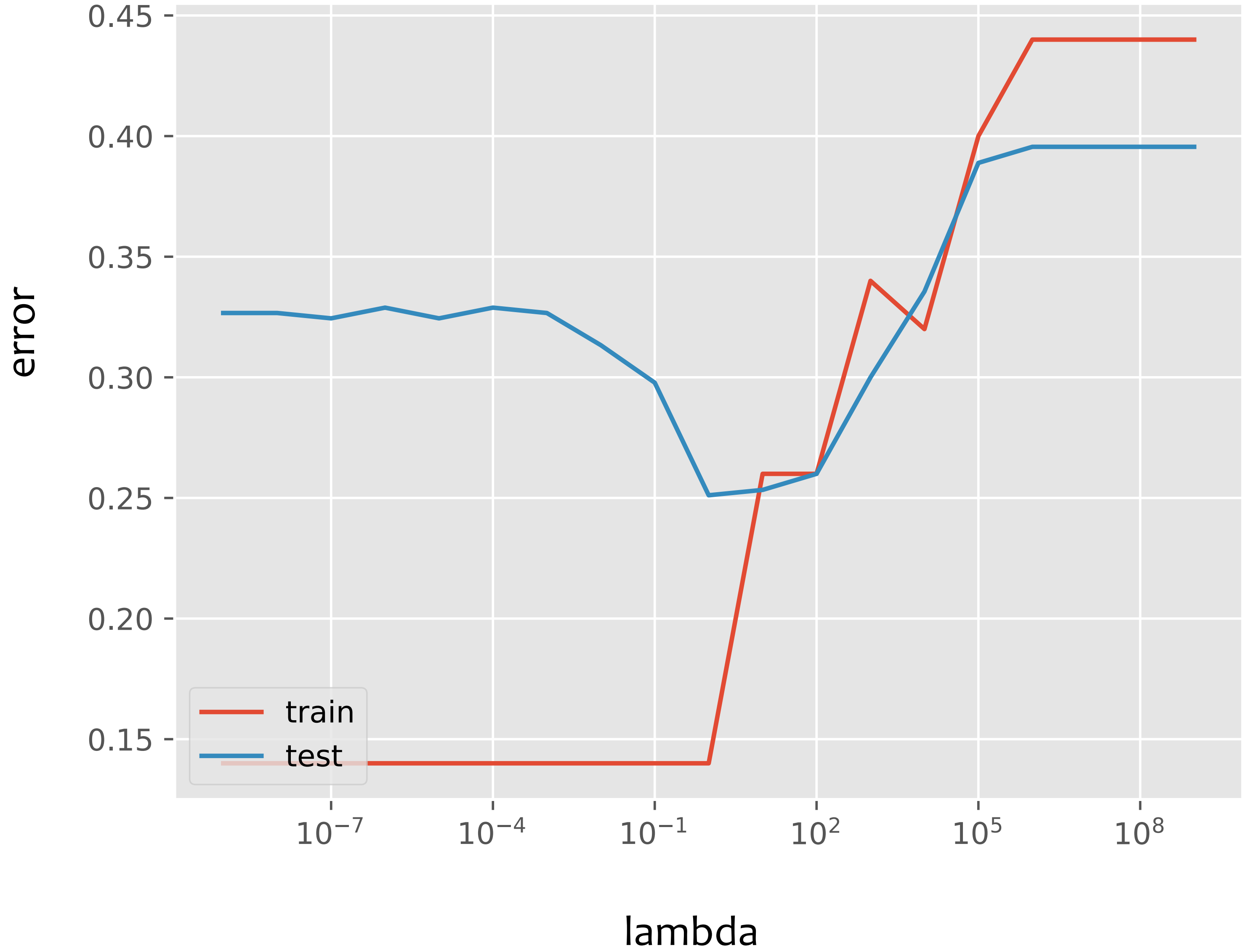
Example: Logistic Regression



Example: Logistic Regression



Example: Logistic Regression



Regularization as MAP

- L1 and L2 regularization can be interpreted as **maximum a-posteriori (MAP) estimation** of the parameters
- To be discussed later in the course…

Takeaways

1. **Nonlinear basis functions** allow **linear models** (e.g. Linear Regression, Logistic Regression) to capture **nonlinear** aspects of the original input
2. Nonlinear features are **require no changes to the model** (i.e. just preprocessing)
3. **Regularization** helps to avoid **overfitting**
4. **Regularization** and **MAP estimation** are equivalent for appropriately chosen priors

Feature Engineering / Regularization Objectives

You should be able to…

- Engineer appropriate features for a new task
- Use feature selection techniques to identify and remove irrelevant features
- Identify when a model is overfitting
- Add a regularizer to an existing objective in order to combat overfitting
- Explain why we should **not** regularize the bias term
- Convert linearly inseparable dataset to a linearly separable dataset in higher dimensions
- Describe feature engineering in common application areas

Neural Networks Outline

- **Logistic Regression (Recap)**
  - Data, Model, Learning, Prediction
- **Neural Networks**
  - A Recipe for Machine Learning
  - Visual Notation for Neural Networks
  - Example: Logistic Regression Output Surface
  - 2-Layer Neural Network
  - 3-Layer Neural Network
- **Neural Net Architectures**
  - Objective Functions
  - Activation Functions
- **Backpropagation**
  - Basic Chain Rule (of calculus)
  - Chain Rule for Arbitrary Computation Graph
  - Backpropagation Algorithm
  - Module-based Automatic Differentiation (Autodiff)

NEURAL NETWORKS

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

Face

Face

Not a face

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

Background

A Recipe for

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of th

– Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

opp site the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Goals for Today's Lecture

1. Explore a **new class of decision functions** (Neural Networks)
2. Consider **variants of this recipe** for training

A Recipe for

2. Choose each of these:
   - Decision function

     $$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

     $$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

4. Train with SGD: (take small steps opposite the gradient)

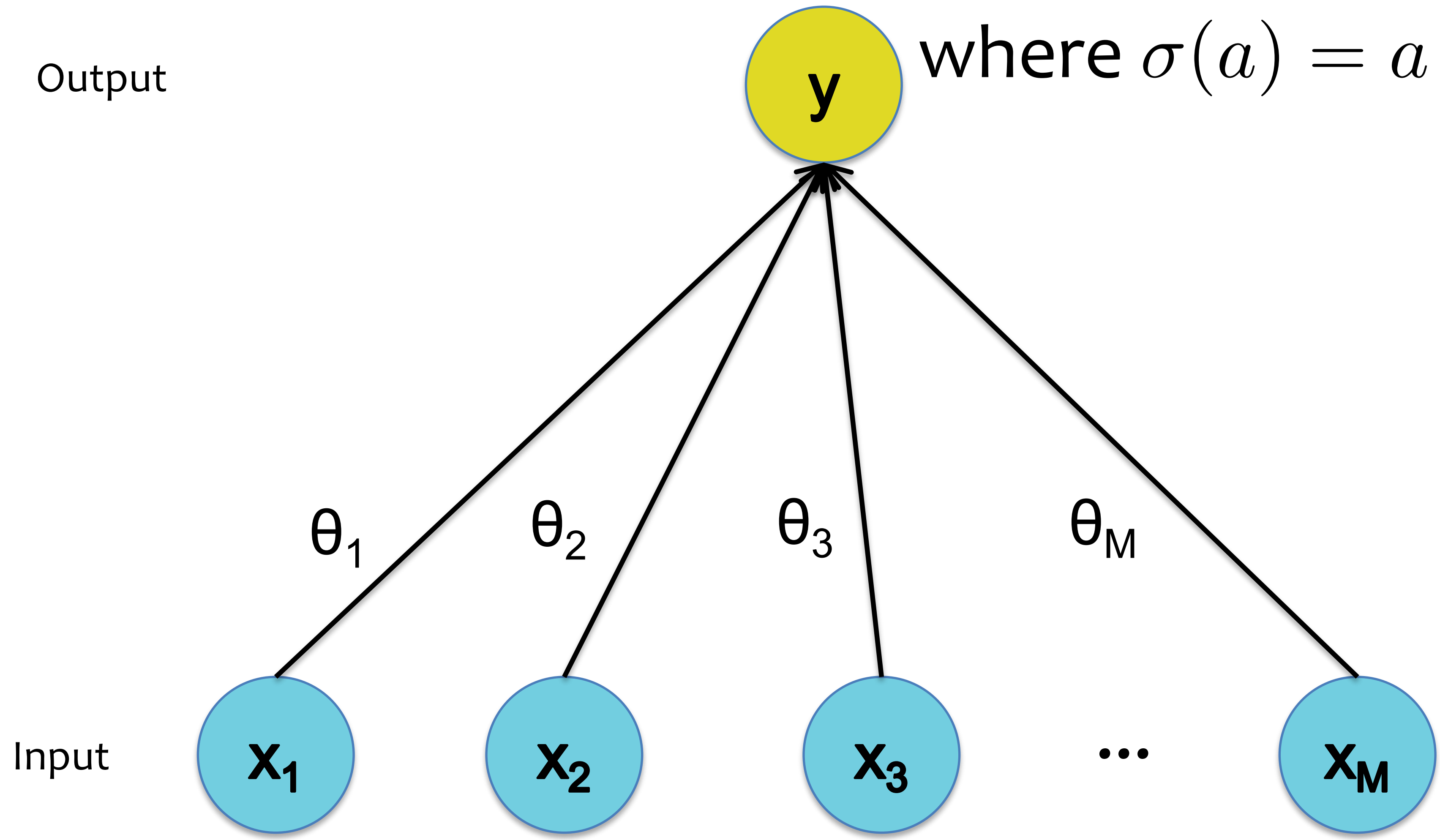
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Decision Functions

Linear Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = a$

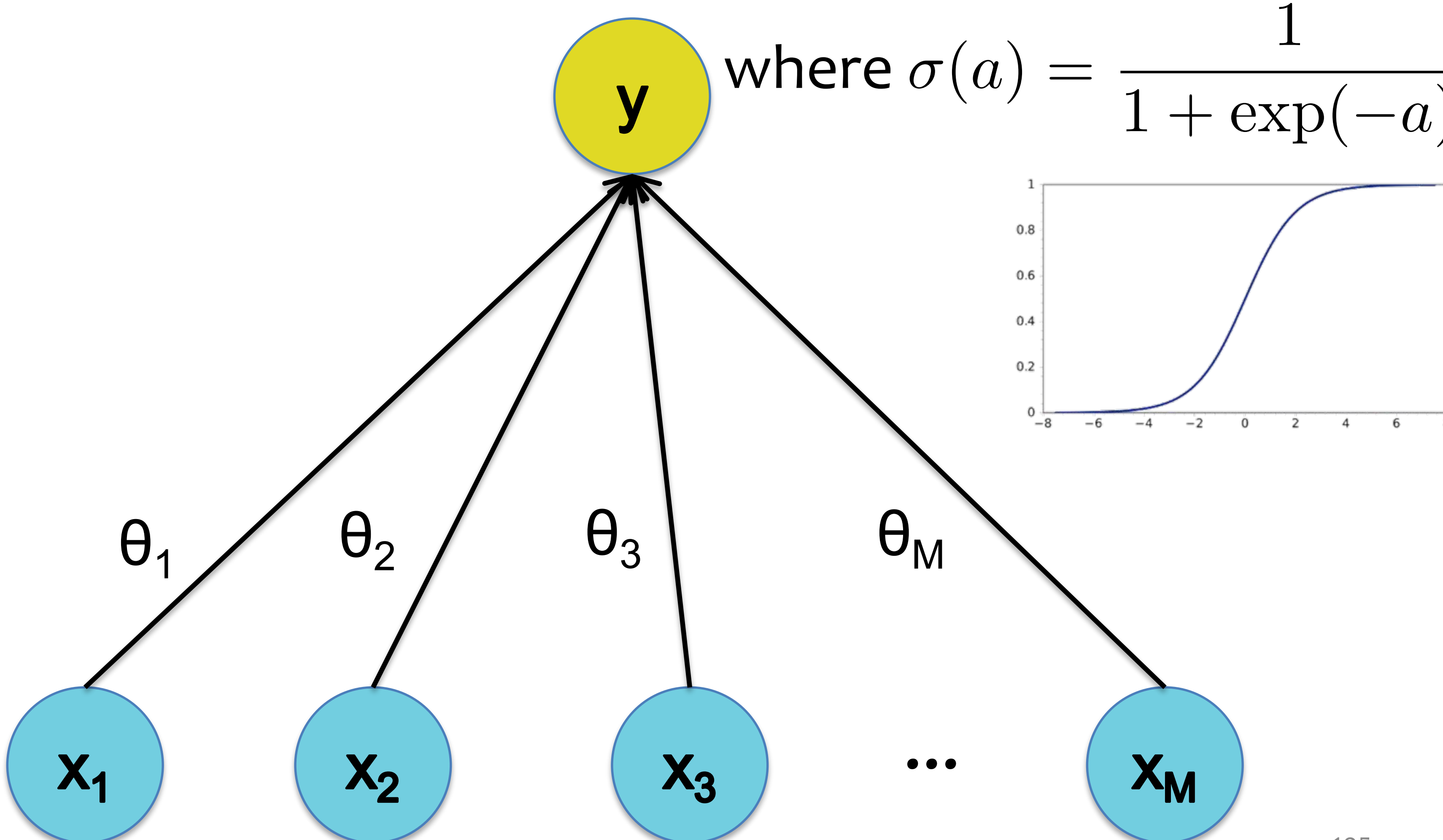


Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

Output

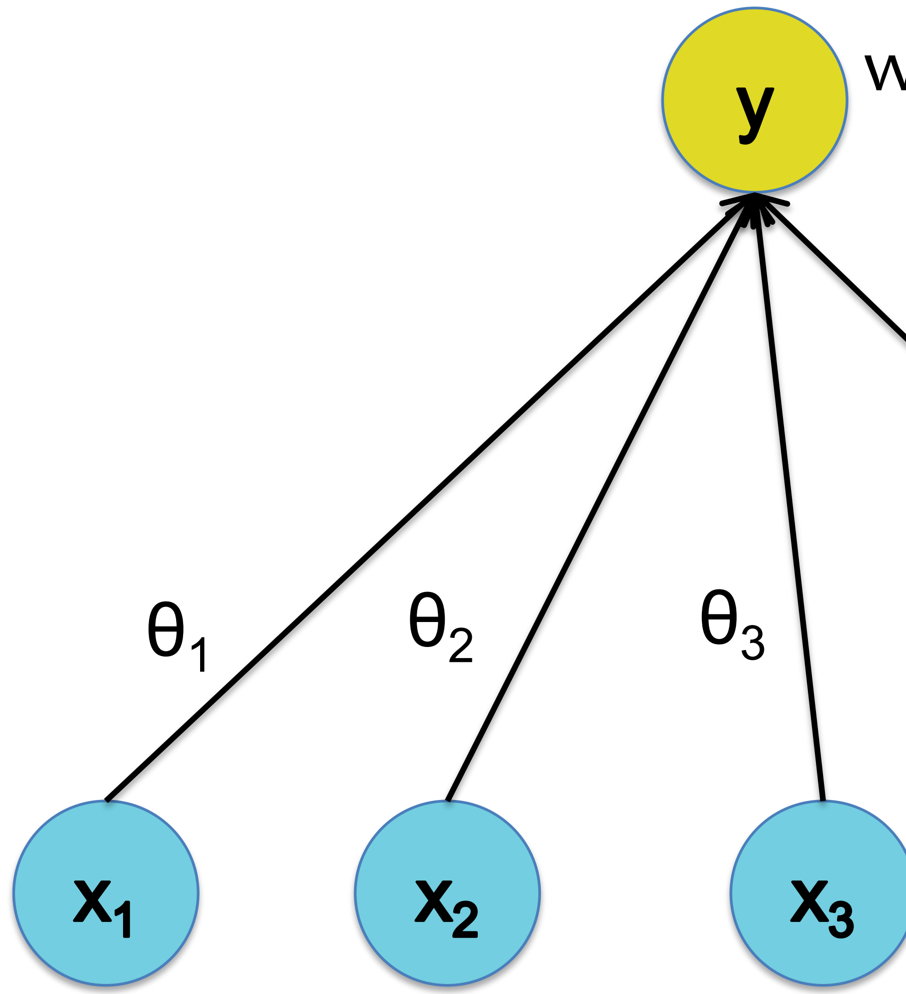


Input

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

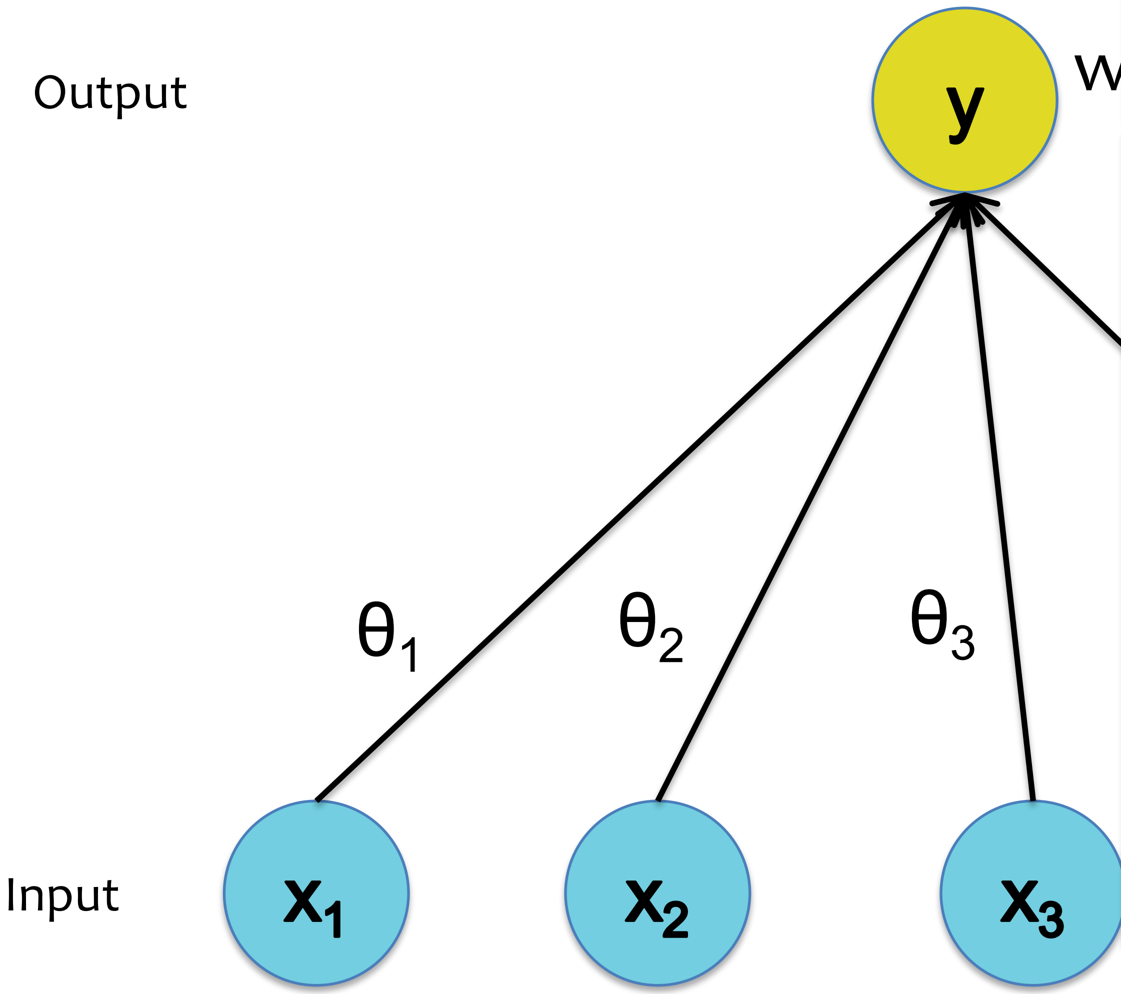
where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



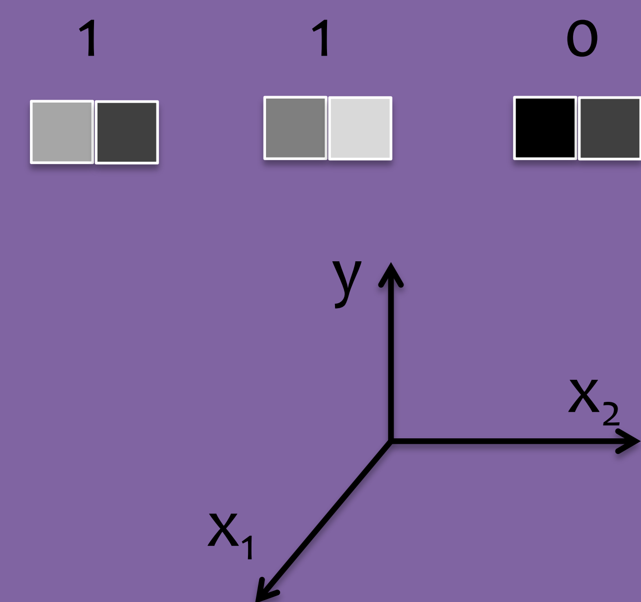
Decision Functions

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$



In-Class Example

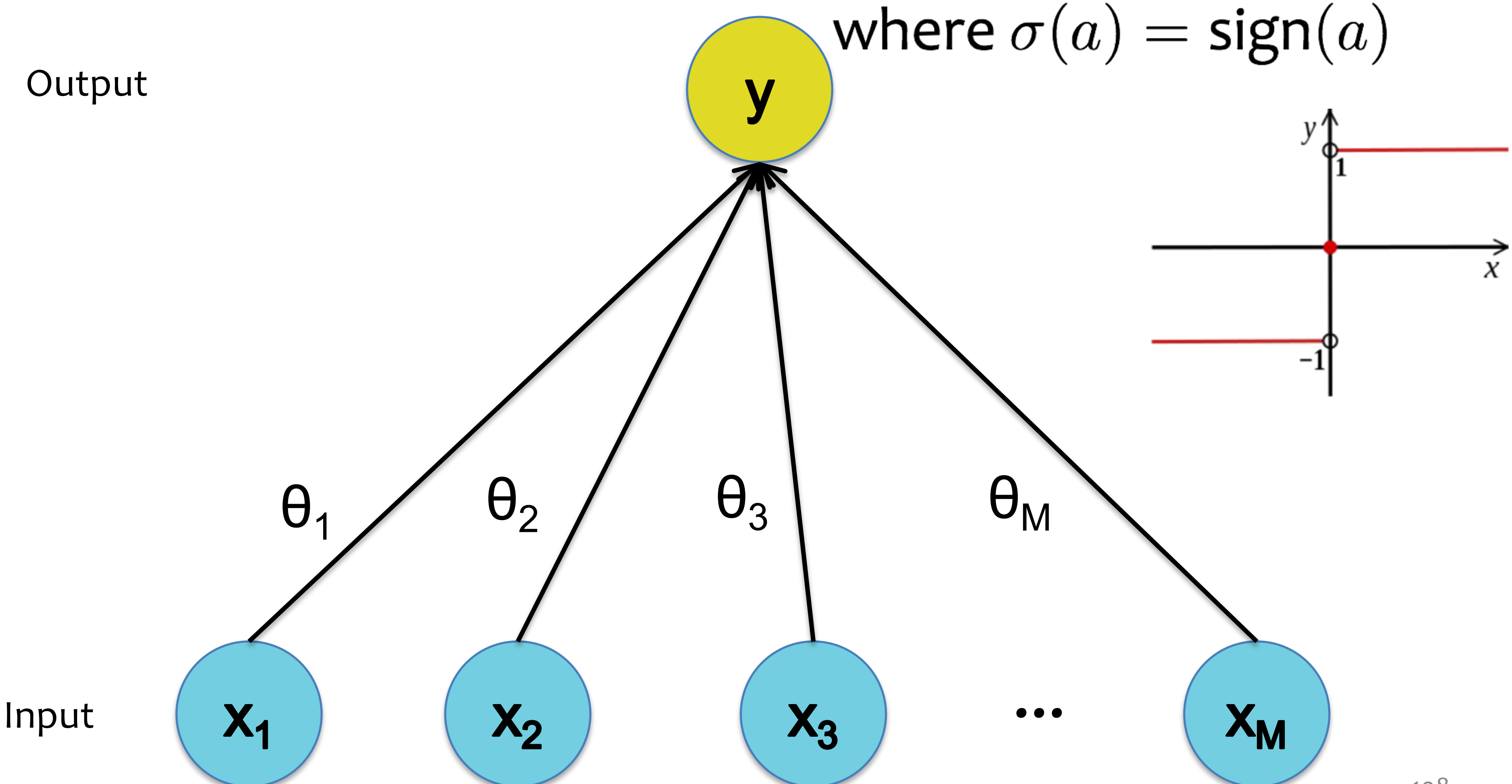


Decision Functions

Perceptron

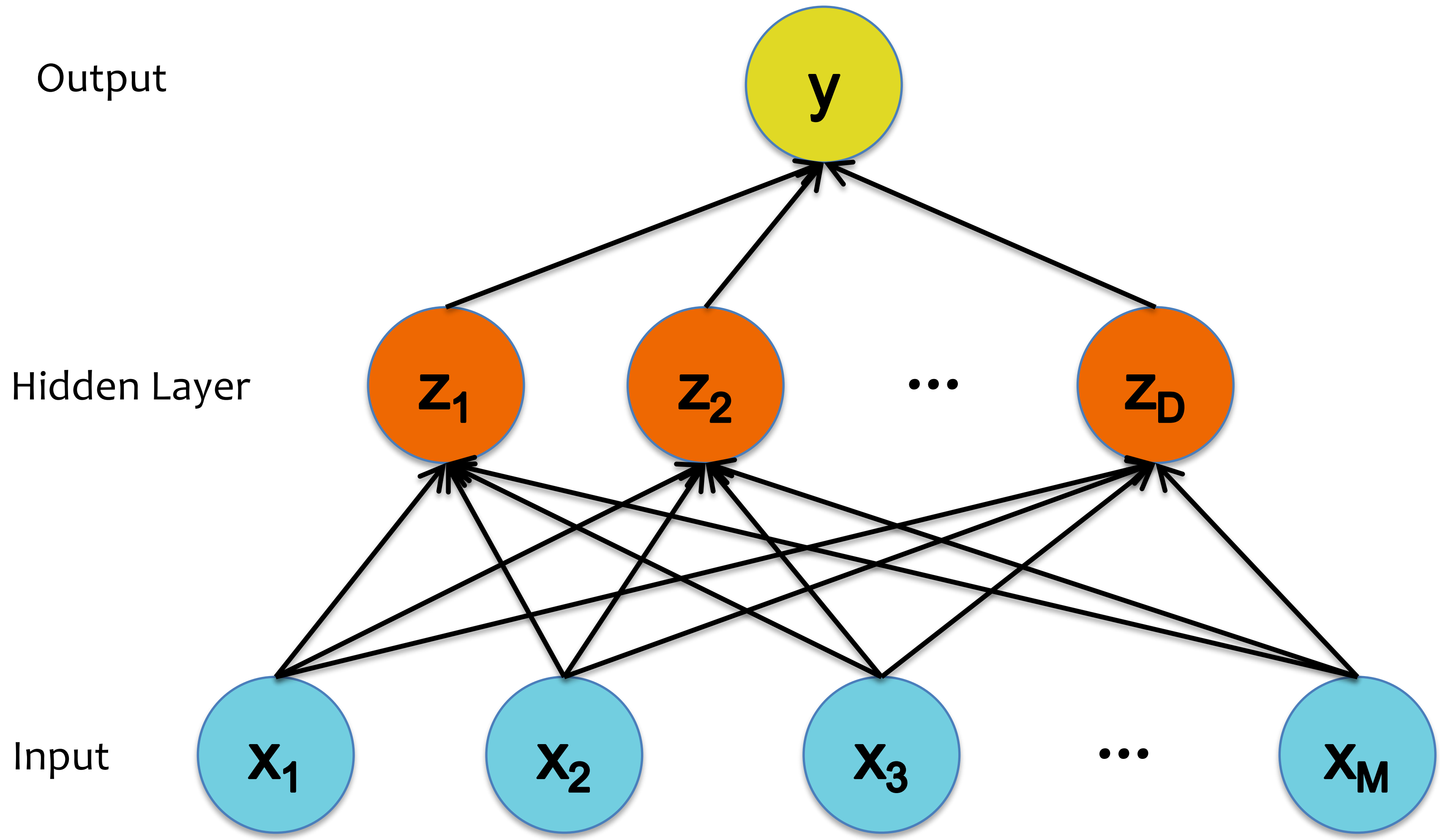
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \text{sign}(a)$

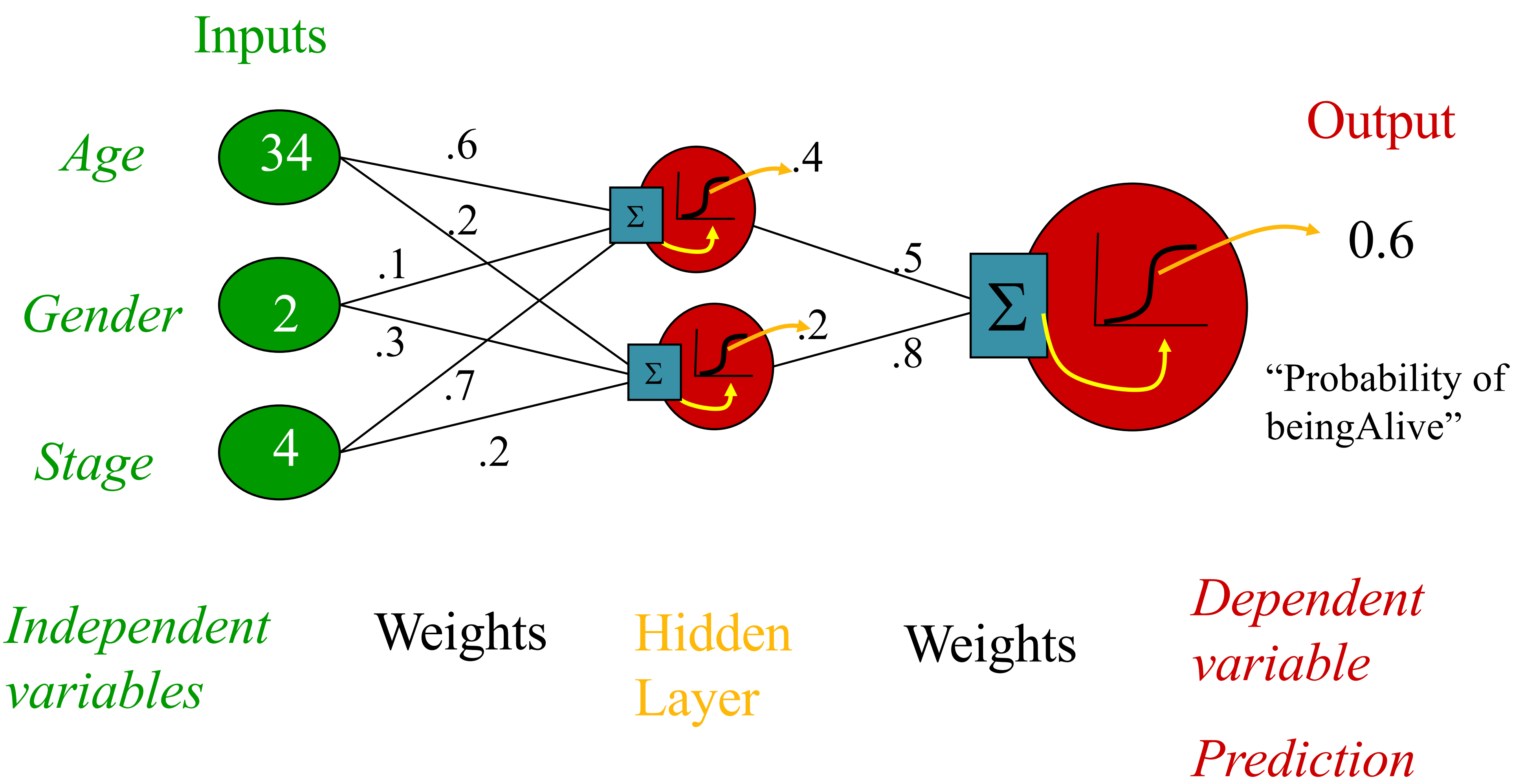


Decision Functions

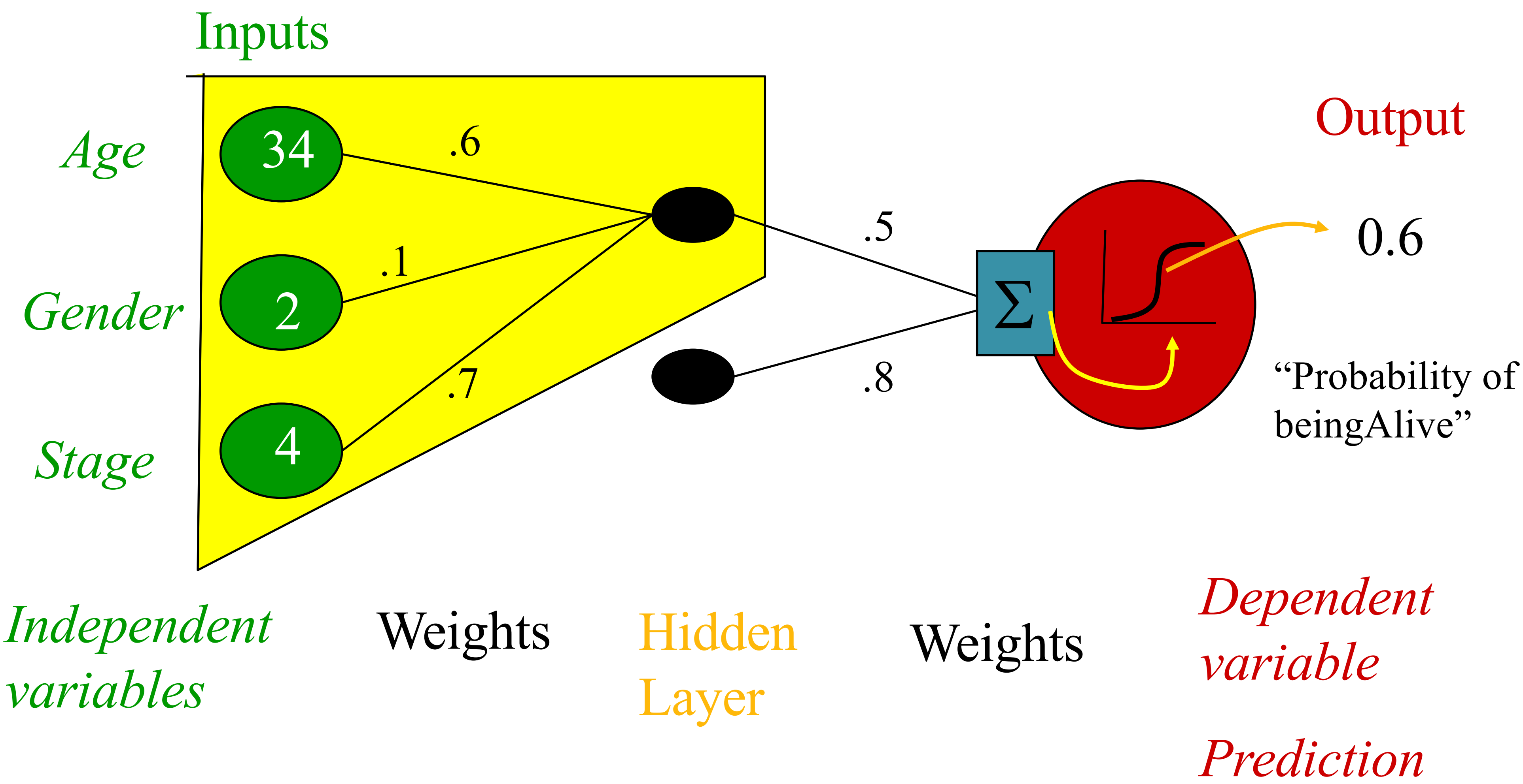
Neural Network

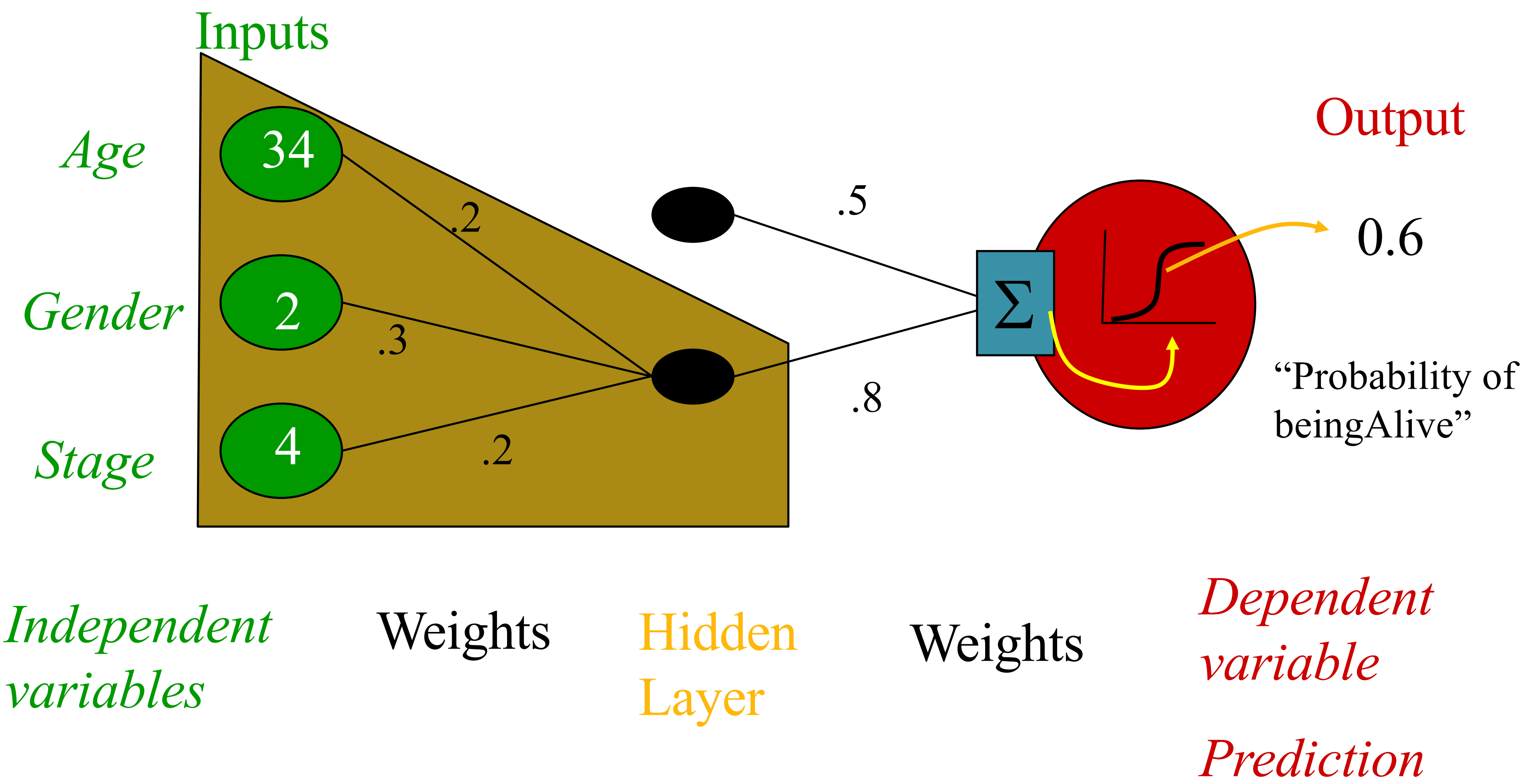


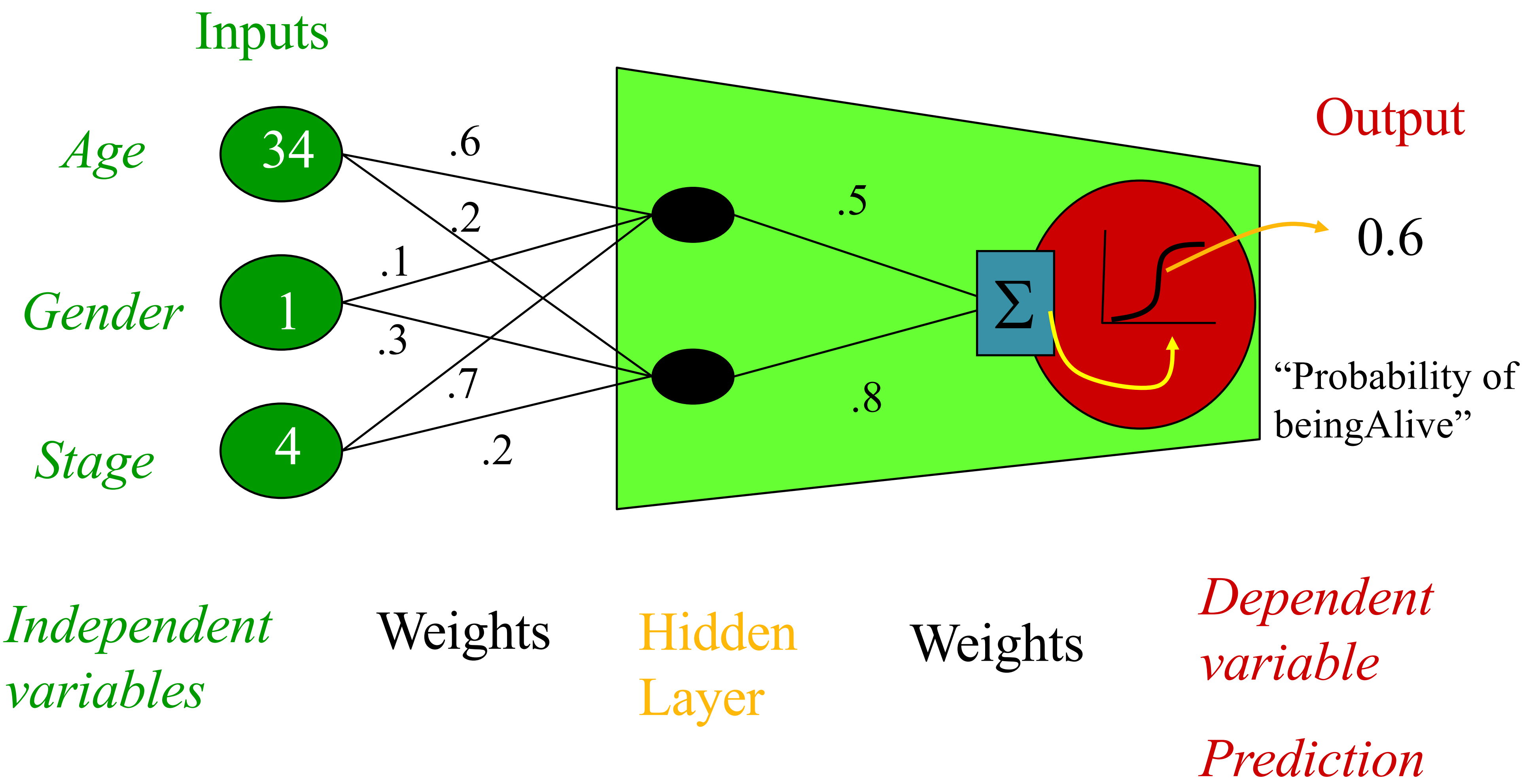
Neural Network Model



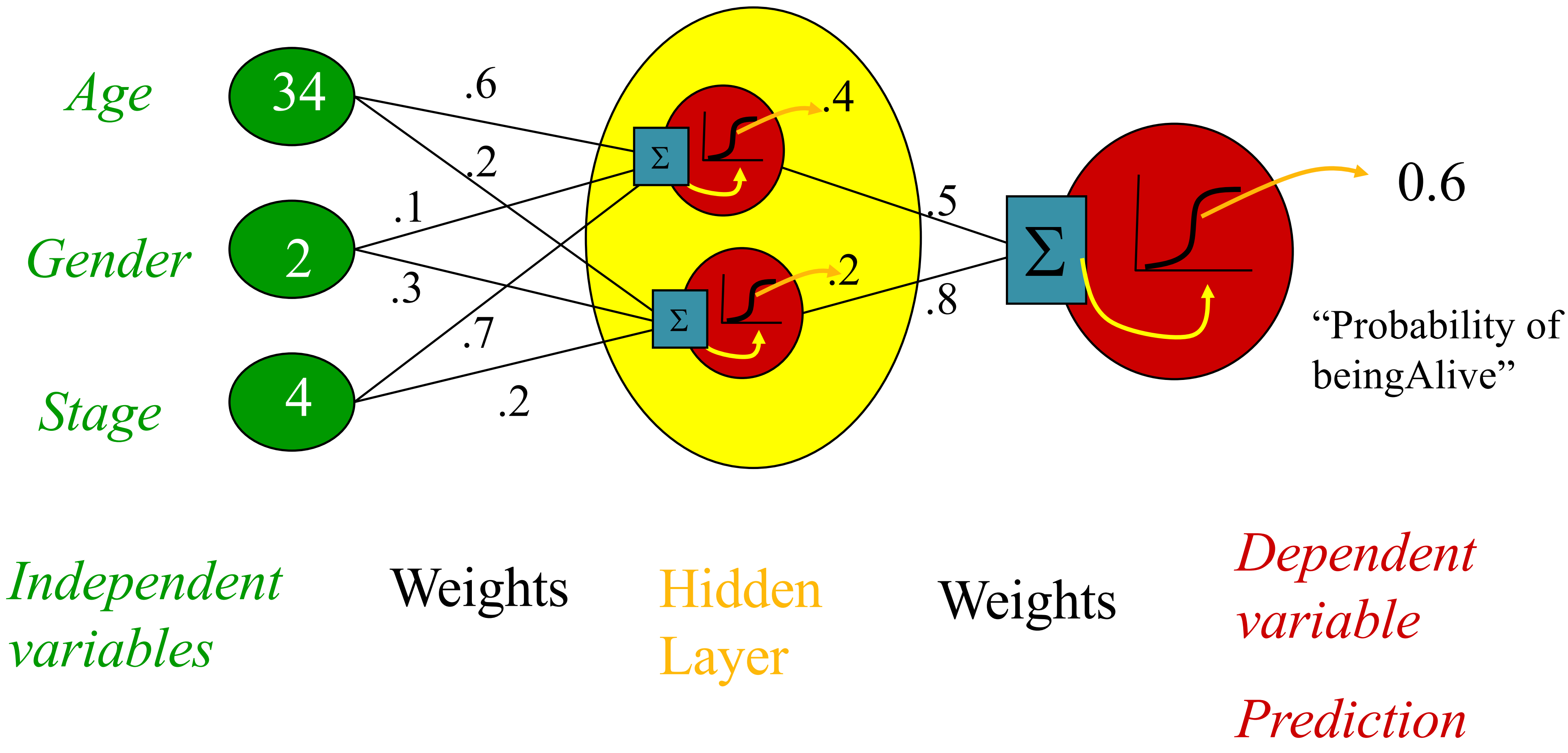
“Combined logistic models”




Inputs
Age
Gender
Stage
34
2
4
.2
.3
.2
.5
.8
Σ
Output
0.6
"Probability of beingAlive"
Independent variables
Weights
Hidden Layer
Weights
Dependent variable
Prediction

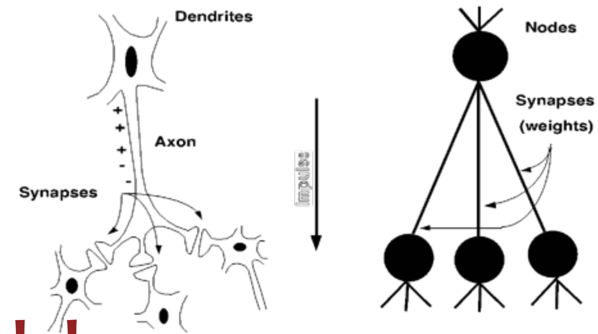

Inputs
Age
34
Gender
1
Stage
4
.6
.2
.1
.3
.7
.2
.5
.8
Σ
Output
0.6
"Probability of beingAlive"
Independent variables
Weights
Hidden Layer
Weights
Dependent variable
Prediction

Not really,
no target for hidden units...



From Biological to Artificial

The motivation for Artificial Neural Networks comes from biology…



Biological "Model"

- **Neuron:** an excitable cell
- **Synapse:** connection between neurons
- A *neuron* sends an **electrochemical pulse** along its *synapses* when a sufficient voltage change occurs
- **Biological Neural Network:** collection of neurons along some pathway through the brain

Artificial Model

- **Neuron:** node in a directed acyclic graph (DAG)
- **Weight:** multiplier on each edge
- **Activation Function:** nonlinear thresholding function, which allows a neuron to "fire" when the input value is sufficiently high
- **Artificial Neural Network:** collection of neurons into a DAG, which define some differentiable function

Biological "Computation"

- Neuron switching time : ~ 0.001 sec
- Number of neurons: ~ 10^{10}
- Connections per neuron: ~ 10^{4-5}
- Scene recognition time: ~ 0.1 sec

Artificial Computation

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed processes

Neural Networks

Chalkboard

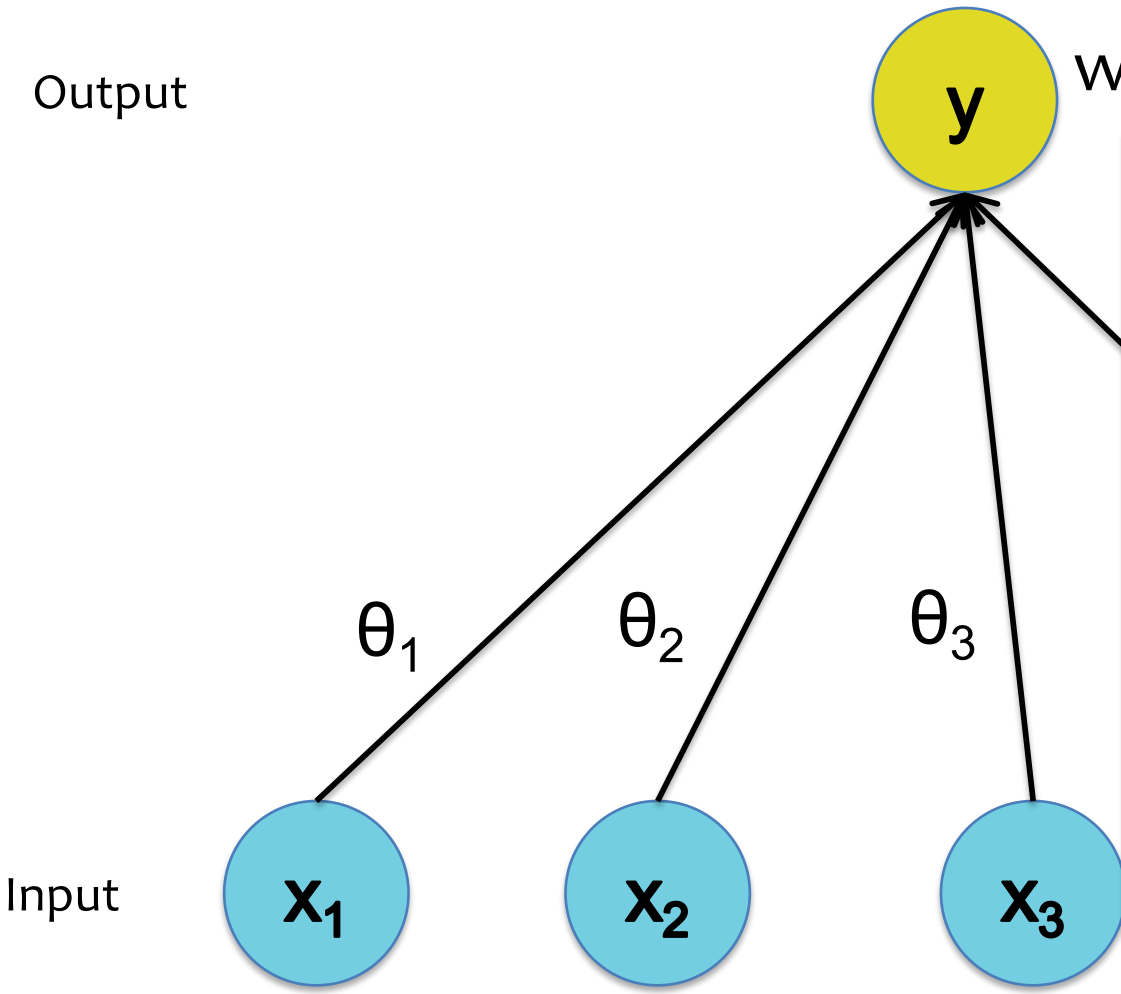
- Example: Neural Network w/1 Hidden Layer
- Example: Neural Network w/2 Hidden Layers
- Example: Feed Forward Neural Network

Decision Functions

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

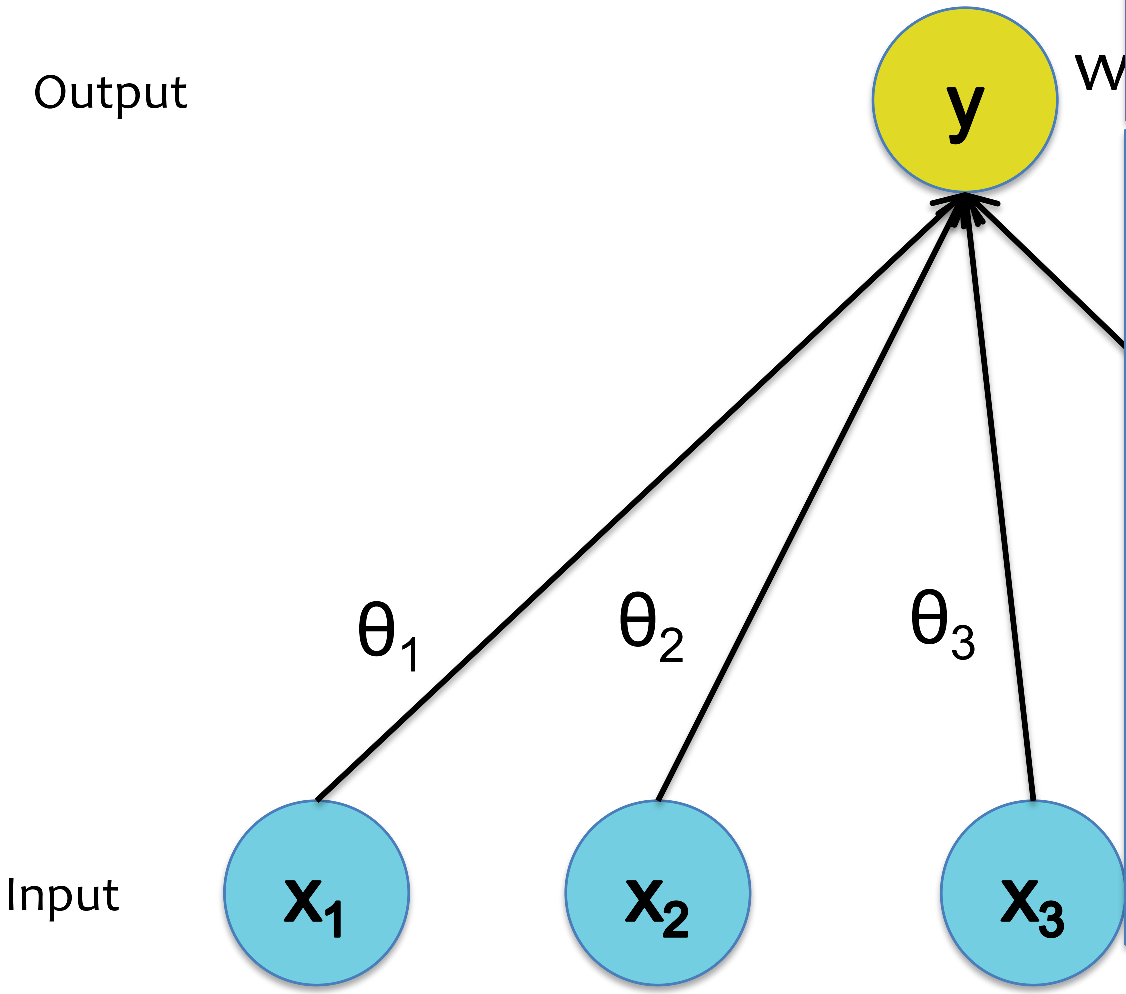
$$\text{where } \sigma(a) = \frac{1}{1 + \exp(-a)}$$



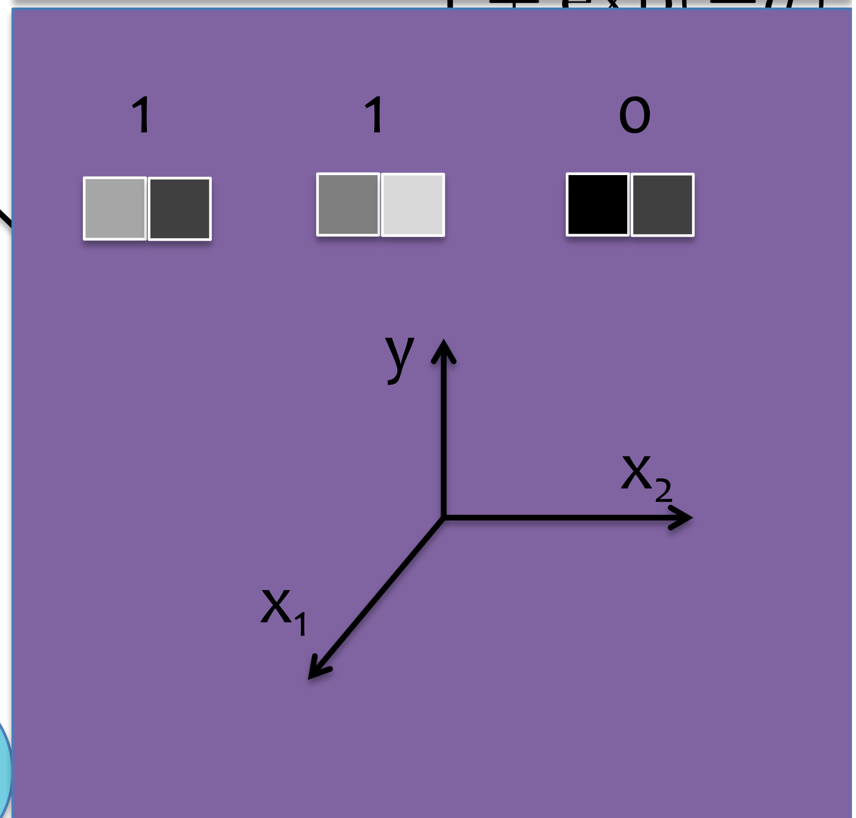
Decision Functions

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$



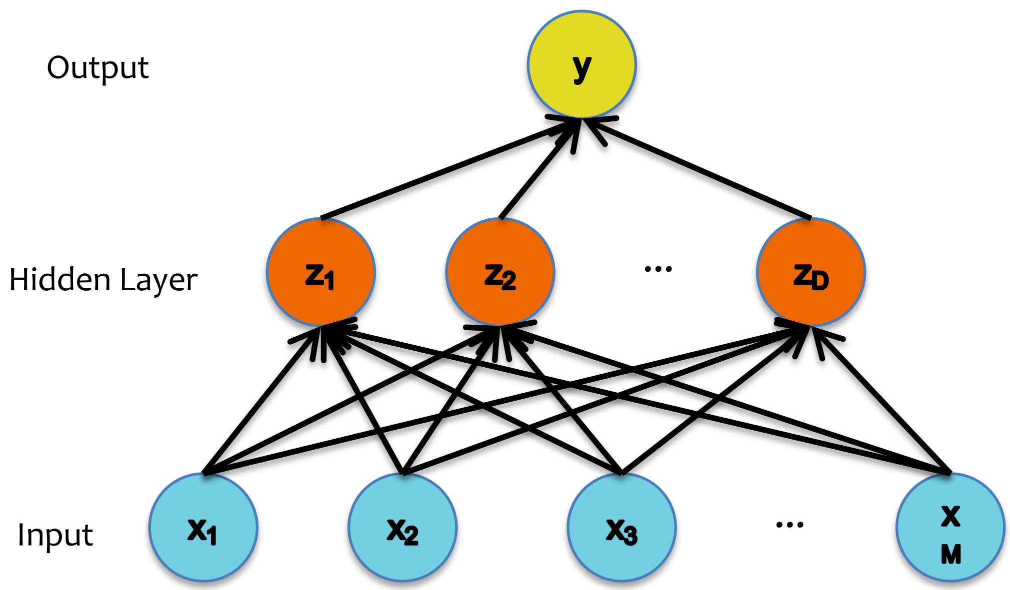
In-Class Example



Decision Functions

Neural Network

Neural Network for Classification



(E) Output (sigmoid)
$$y = \frac{1}{1+\exp(-b)}$$

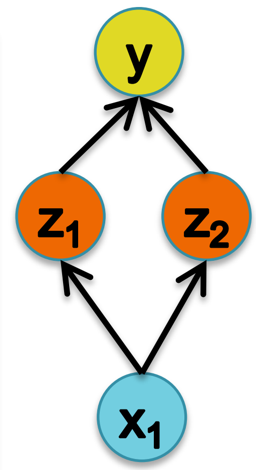
(D) Output (linear)
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) Hidden (sigmoid)
$$z_j = \frac{1}{1+\exp(-a_j)}, \ \ \forall j$$

(B) Hidden (linear)
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \ \forall j$$

(A) Input
Given $x_i, \ \ \forall i$

Neural Network Parameters



Question:

Suppose you are training a one-hidden layer neural network with sigmoid activations for binary classification.

True or False: There is a unique set of parameters that maximize the likelihood of the dataset above.

Answer: