10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Stochastic Gradient Descent + Feature Engineering + Regularization

Matt Gormley & Henry Chai

Lecture 10

Sep. 25, 2024

Reminders

- Exam 1
  - Mon, Sep. 30, 6:30pm – 8:30pm
- Homework 4: Logistic Regression
  - Out: ~~Fri, Feb 17~~ 9/30
  - Due: ~~Sun, Feb. 26~~ at 11:59pm

10/??

STOCHASTIC GRADIENT DESCENT

Recall: Gradient Descent for Logistic Regression

- Input: training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ and step size γ

1. Initialize $\boldsymbol{\theta}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:

$$O(ND)\left\{\ \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)}) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)}\left(P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(t)}\right) - y^{(i)}\right)\right.$$

   b. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$
   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{\theta}^{(t)}$

Stochastic Gradient Descent (SGD)

- Input: training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ and step size γ

1. Initialize $\boldsymbol{\theta}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample a data point from $\mathcal{D}$, $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$
   b. Compute the pointwise gradient:

$$O(D) \left\{ \nabla_{\boldsymbol{\theta}} J^{(i)}\left(\boldsymbol{\theta}^{(t)}\right) = \boldsymbol{x}^{(i)}\left(P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(t)}\right) - y^{(i)}\right) \right.$$

   c. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla_{\boldsymbol{\theta}} J^{(i)}\left(\boldsymbol{\theta}^{(t)}\right)$
   d. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{\theta}^{(t)}$

Stochastic Gradient Descent (SGD)

- If the example is sampled uniformly at random, the expected value of the pointwise gradient is the same as the full gradient!

$$E\left[\nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})\right] = \sum_{i=1}^{N} \left(\underline{\text{probability of selecting } \boldsymbol{x}^{(i)}, y^{(i)}}\right) \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \left(\frac{1}{N}\right) \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta}) = \underbrace{\frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- In practice, the data set is randomly shuffled then looped through so that each data point is used equally often

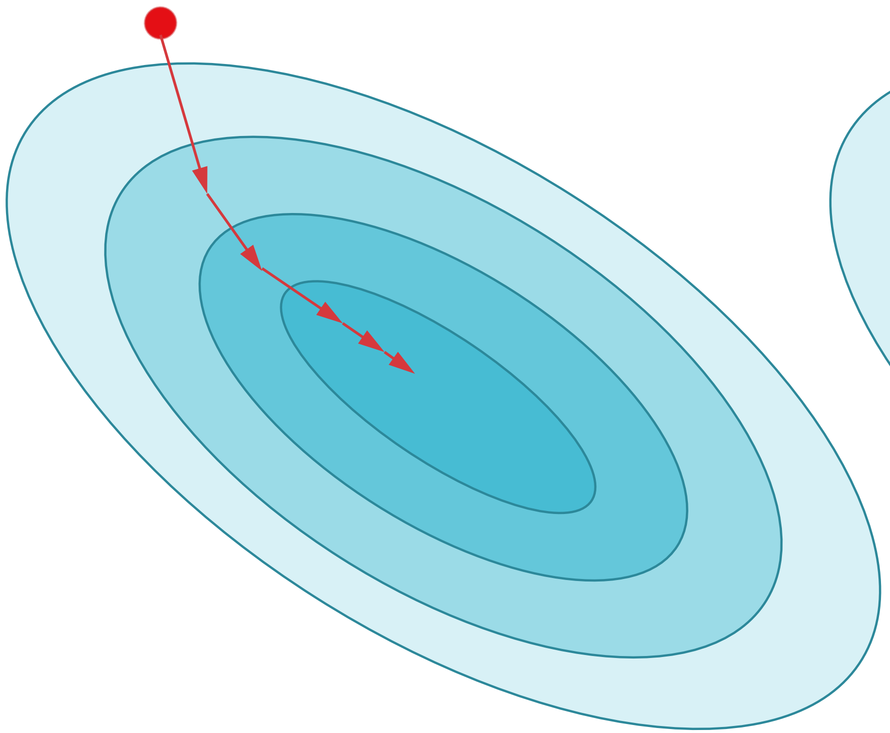
Stochastic Gradient Descent (SGD)

- Input: training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$ and step size γ

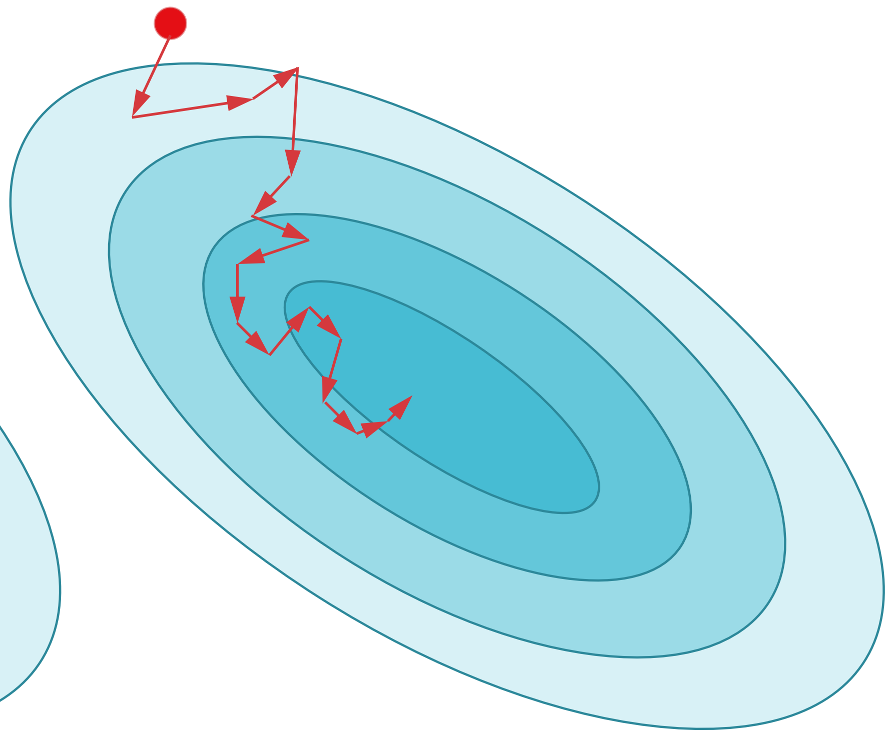
1. Initialize $\boldsymbol{\theta}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. For $i \in \text{shuffle}(\{1, \dots, N\})$
      i. Compute the pointwise gradient:
         $\nabla_{\boldsymbol{\theta}} J^{(i)}\left(\boldsymbol{\theta}^{(t)}\right) = \boldsymbol{x}^{(i)}\left(P\left(Y = 1 \middle| \boldsymbol{x}^{(i)}, \boldsymbol{\theta}^{(t)}\right) - y^{(i)}\right)$
      ii. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla_{\boldsymbol{\theta}} J^{(i)}\left(\boldsymbol{\theta}^{(t)}\right)$
      iii. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{\theta}^{(t)}$

Stochastic Gradient Descent vs. Gradient Descent



Gradient Descent



Stochastic Gradient Descent

Stochastic Gradient Descent vs. Gradient Descent

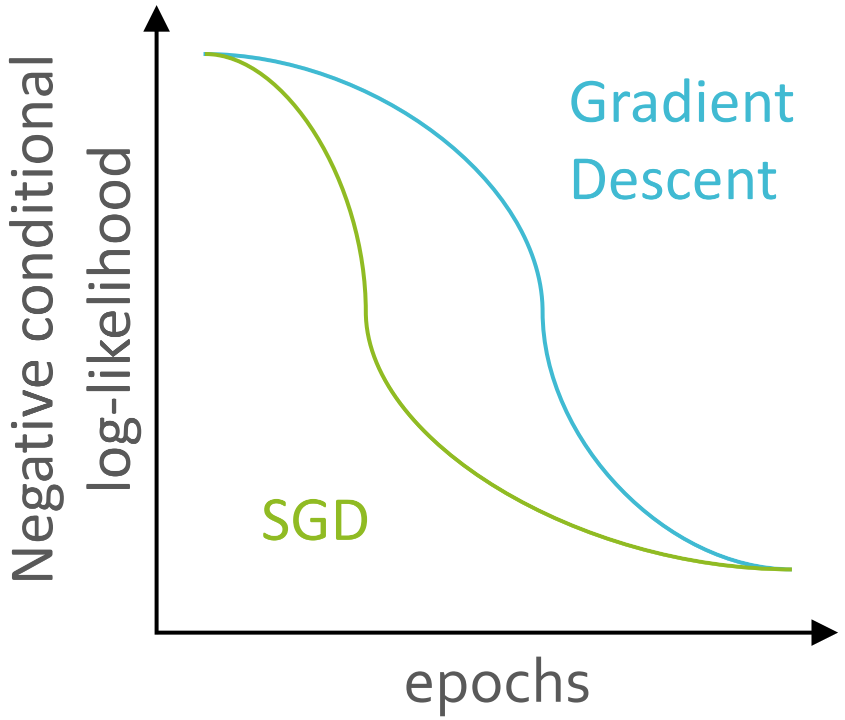
- An *epoch* is a single pass through the entire training dataset
  - Gradient descent updates the parameters once per epoch
  - SGD updates the parameters N times per epoch
- Theoretical comparison:
  - Define convergence to be when $J(\boldsymbol{\theta}^{(t)}) - J(\boldsymbol{\theta}^{*}) < \epsilon$

Method	Steps to Convergence	Computation per Step
Gradient descent	$O(\log {}^{1}/_{\epsilon})$	$O(ND)$
SGD	$O({}^{1}/_{\epsilon})$	$O(D)$

(with high probability under certain assumptions)

Stochastic Gradient Descent vs. Gradient Descent

- An *epoch* is a single pass through the entire training dataset
  - Gradient descent updates the parameters once per epoch
  - SGD updates the parameters N times per epoch



Empirically, SGD reduces the negative conditional log-likelihood much faster than gradient descent

Optimization for ML Learning Objectives

You should be able to…

- Apply gradient descent to optimize a function
- Apply stochastic gradient descent (SGD) to optimize a function
- Apply knowledge of zero derivatives to identify a closed-form solution (if one exists) to an optimization problem
- Distinguish between convex, concave, and nonconvex functions
- Obtain the gradient (and Hessian) of a (twice) differentiable function

Logistic Regression Learning Objectives

You should be able to…

- Apply the principle of maximum likelihood estimation (MLE) to learn the parameters of a probabilistic model
- Given a discriminative probabilistic model, derive the conditional log-likelihood, its gradient, and the corresponding Bayes Classifier
- Explain the practical reasons why we work with the log of the likelihood
- Implement logistic regression for binary (and multiclass) classification
- Prove that the decision boundary of binary logistic regression is linear

Linear Models

PERCEPTRON, LINEAR REGRESSION, AND LOGISTIC REGRESSION

Matching Game

$\vec{\theta} \leftarrow \vec{\theta} + \lambda(h_\theta(\vec{x}^{(i)}) - y^{(i)})\vec{x}^{(i)}$

Question: 1

Match the Algorithm to its Update Rule

1. SGD for Logistic Regression

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = p(y|x)$$

$y=1$

2. Least Mean Squares

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \underline{\boldsymbol{\theta}^T \mathbf{x}}$$

3. Perceptron

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

$\lambda = 1/2$

4.

$$\theta_k \leftarrow \theta_k + (h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})$$

5.

$$\theta_k \leftarrow \theta_k + \frac{1}{1 + \exp \lambda(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})}$$

6.

$$\theta_k \leftarrow \theta_k + \lambda(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})x_k^{(i)}$$

step size

Answer:

31% A. 1=5, 2=4, 3=6

21% B. 1=5, 2=6, 3=4

12% C. 1=6, 2=4, 3=4

9% D. 1=5, 2=6, 3=6

0% $\boxed{\text{E. 1=6, 2=6, 3=6}}$

0% F. 1=6, 2=5, 3=5

1% G. 1=5, 2=5, 3=5

5% H. 1=4, 2=5, 3=6

~~I. None of the above~~

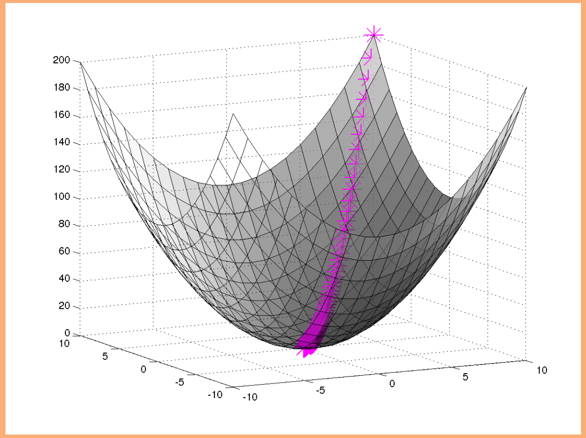
toxic

Recall…

Gradient Descent

Algorithm 1 Gradient Descent

> **procedure** GD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
>
> $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
>
> $\quad$ **while** not converged **do**
>
> $\qquad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
>
> $\quad$ **return** $\boldsymbol{\theta}$



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

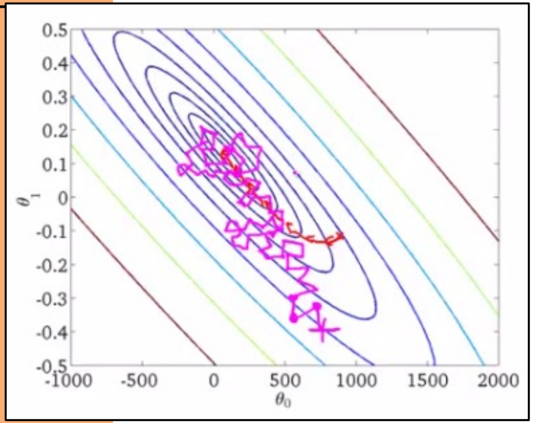
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M} J(\boldsymbol{\theta}) \end{bmatrix}$$

Recall…

Stochastic Gradient Descent (SGD)

Algorithm 1 Stochastic Gradient Descent (SGD)

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**
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
  return $\boldsymbol{\theta}$



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

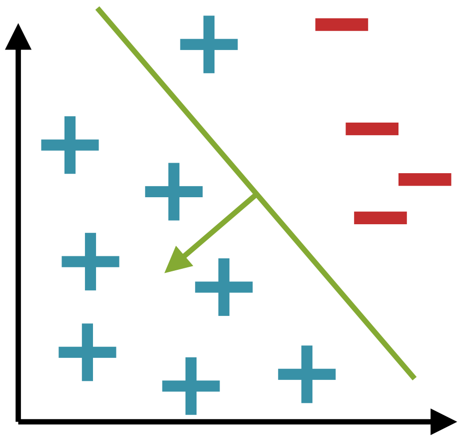
We need a per-example objective:

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

Logistic Regression vs. Perceptron

Question: 2

True or False: Just like Perceptron, **one step** (i.e. iteration) **of SGD for Logistic Regression** will result in a change to the parameters **only** if the current example is **incorrectly** classified.



Answer:

A = toxic

45% B = true

55% $\boxed{\text{C = false}}$

BAYES OPTIMAL CLASSIFIER

Bayes Optimal Classifier

Functio

Previous
was gen
function

Our goa
best app

Suppose you knew the distribution p*(y | **x**) or had a good approximation to it.

Question:

How would you design a function y = h(**x**) to predict a single label?

Answer:

You'd use the *Bayes optimal classifier*!

Probabilistic Learning

Today, we assume that our output is **sampled** from a conditional **probability distribution**:

$$\mathbf{x}^{(i)} \sim p^*(\cdot)$$

$$y^{(i)} \sim p^*(\cdot|\mathbf{x}^{(i)})$$

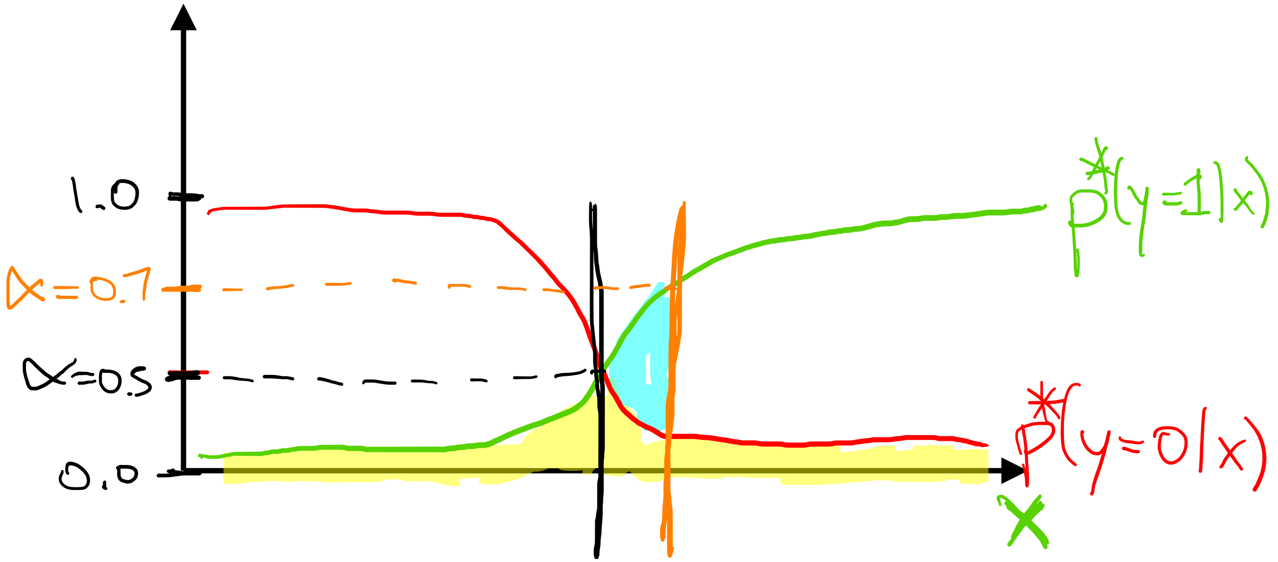
Our goal is to learn a probability distribution p(y|**x**) that best approximates p^*(y|**x**)

Bayes Optimal Classifier

Suppose you have an **oracle** that knows the data generating distribution, p*(y|x).
Q: What is the optimal classifier in this setting?
A: The Bayes optimal classifier! This is the best classifier for the distribution p* and the loss function.

$$\ell(y,\hat{y}) = \mathbb{1}(y \neq \hat{y})$$

$$\hat{y} = h(\vec{x}) = \begin{cases} 1 & \text{if } p(y=1 \mid \vec{x}) \geq \alpha \\ 0 & \text{otherwise} \end{cases}$$



$$\ell(y,\hat{y}) = \begin{cases} 1000 & \text{if } y \neq \hat{y} \text{ and } y=1 \\ 10 & \text{if } y \neq \hat{y} \text{ and } y=0 \\ 0 & \text{otherwise} \end{cases}$$

Definition: The **reducible error** is the expected loss of a hypothesis h(x) that could be reduced if we knew p*(y|x) and picked the optimal h(x) for that p*.

Definition: The **irreducible error** is the expected loss of a hypothesis h(x) that could **not** be reduced if we knew p*(y|x) and picked the optimal h(x) for that p*.

OPTIMIZATION METHOD #4: MINI-BATCH SGD

Mini-Batch SGD

- **Gradient Descent:**
  Compute true gradient exactly from all N examples
- **Stochastic Gradient Descent (SGD):**
  Approximate true gradient by the gradient of one randomly chosen example
- **Mini-Batch SGD:**
  Approximate true gradient by the average gradient of ~~k~~ S randomly chosen examples

Mini-Batch SGD

while not converged: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$

Three variants of first-order optimization:

Gradient Descent: $\mathbf{g} = \nabla J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \nabla J^{(i)}(\boldsymbol{\theta})$

SGD: $\mathbf{g} = \nabla J^{(i)}(\boldsymbol{\theta})$ where i sampled uniformly

Mini-batch SGD: $\mathbf{g} = \frac{1}{S}\sum_{s=1}^{S} \nabla J^{(i_s)}(\boldsymbol{\theta})$ where i_s sampled uniformly $\forall s$

$\{i_1, i_2, \ldots, i_S\} \subset \{1, \ldots, N\}$

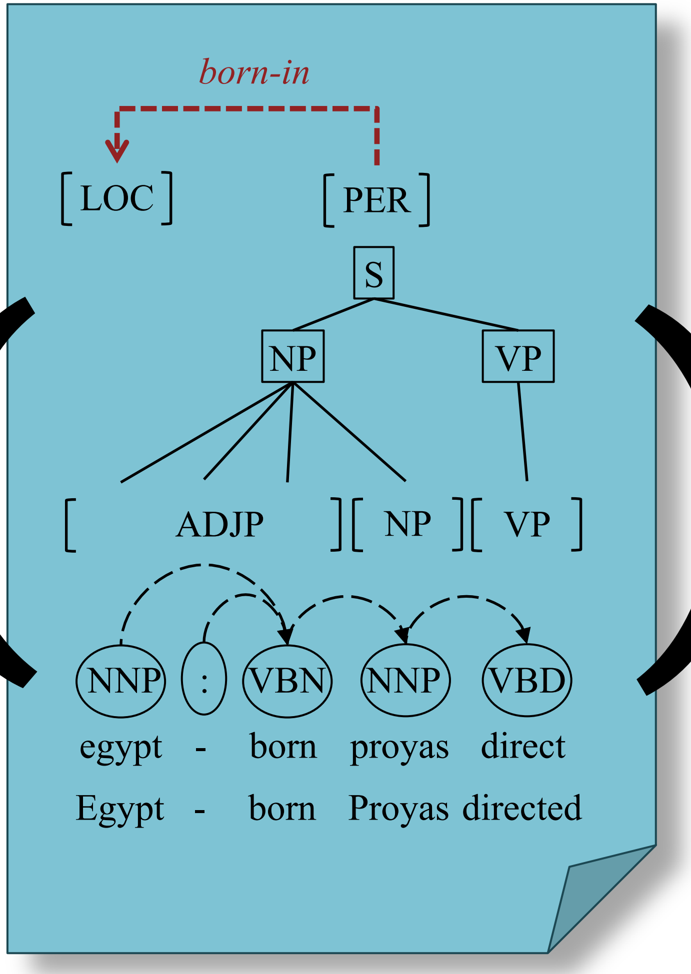
higher $S \Rightarrow$ lower variance

higher $S \Rightarrow$ higher memory

FEATURE ENGINEERING

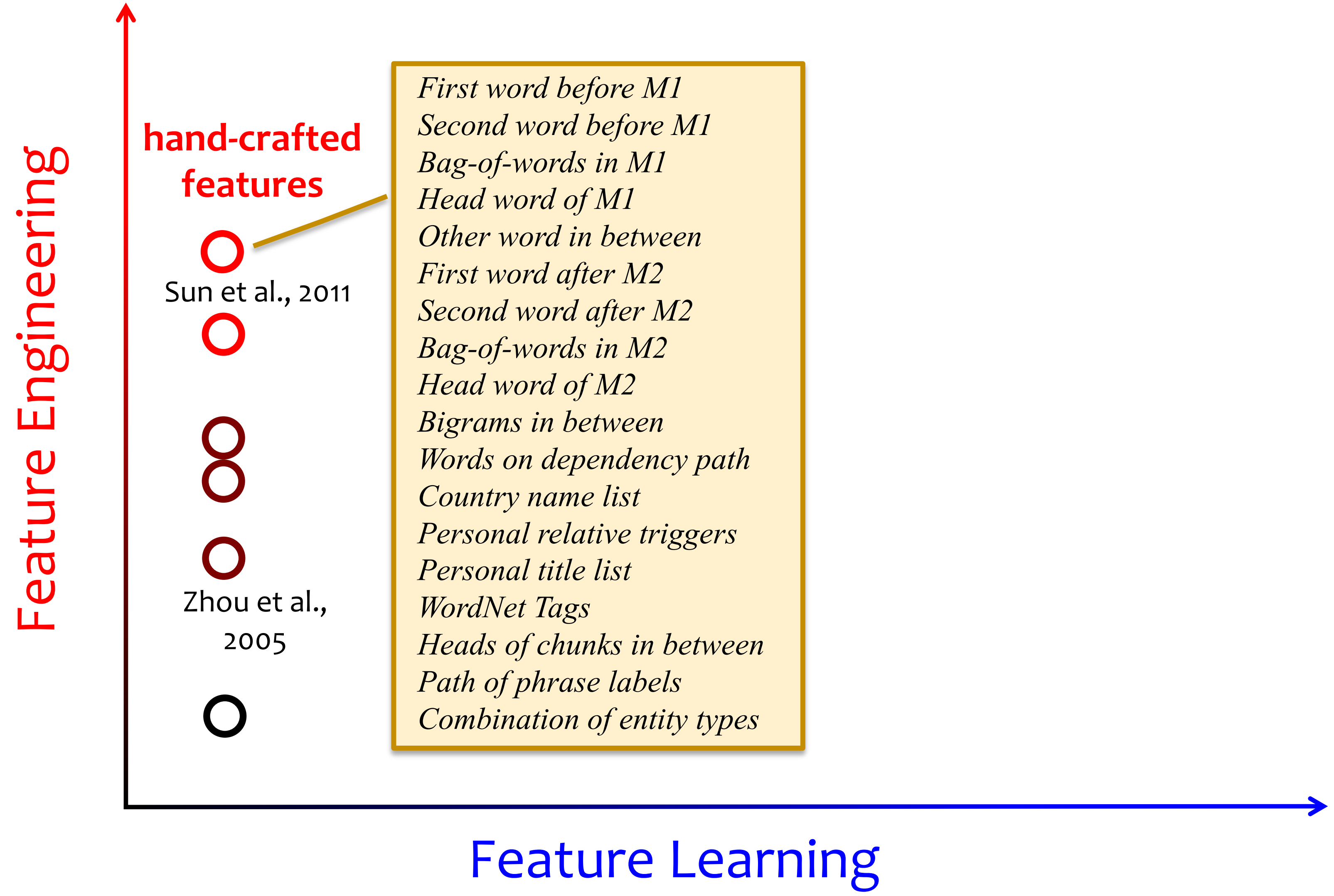
Handcrafted Features

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

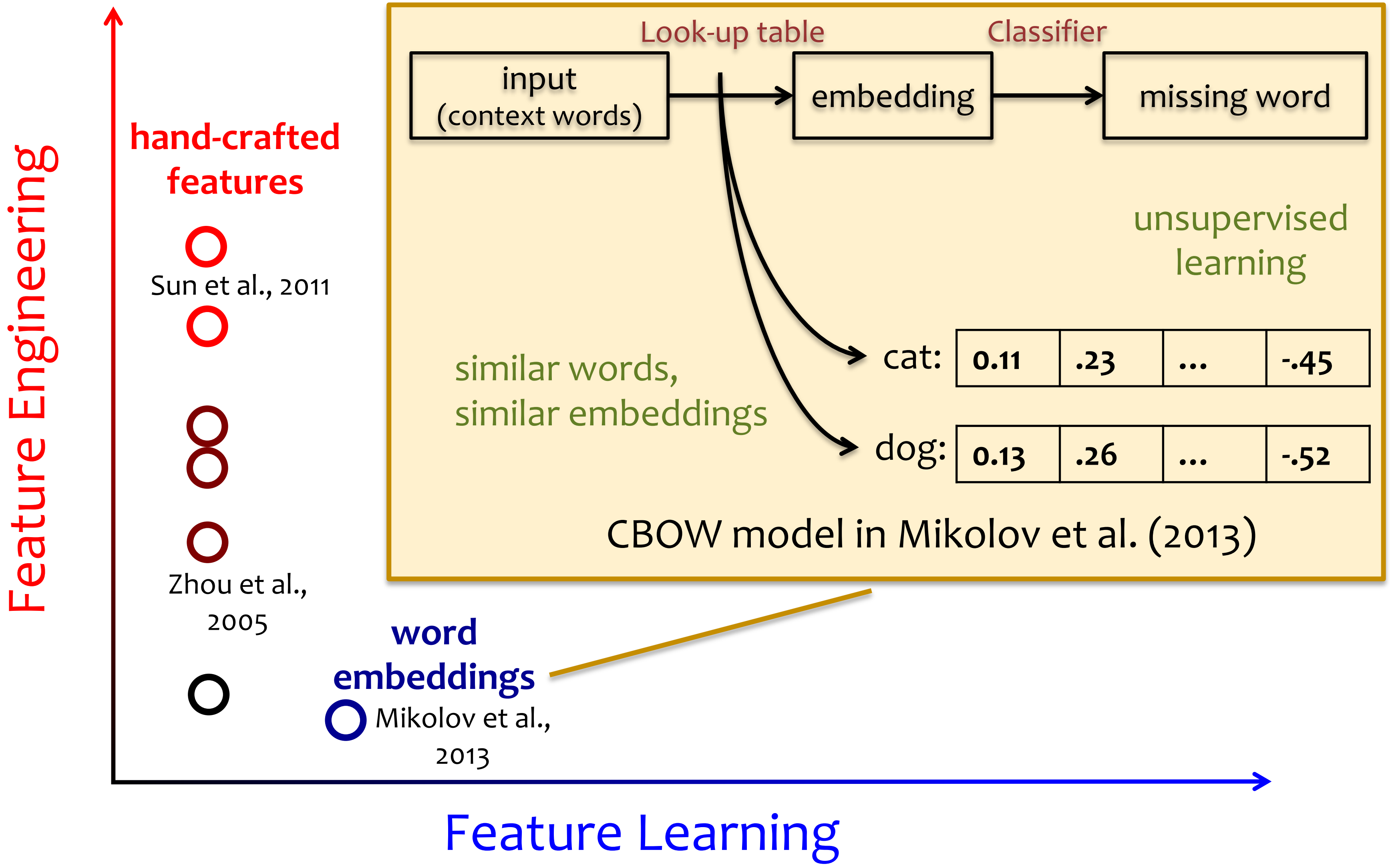


$$\Big))$$

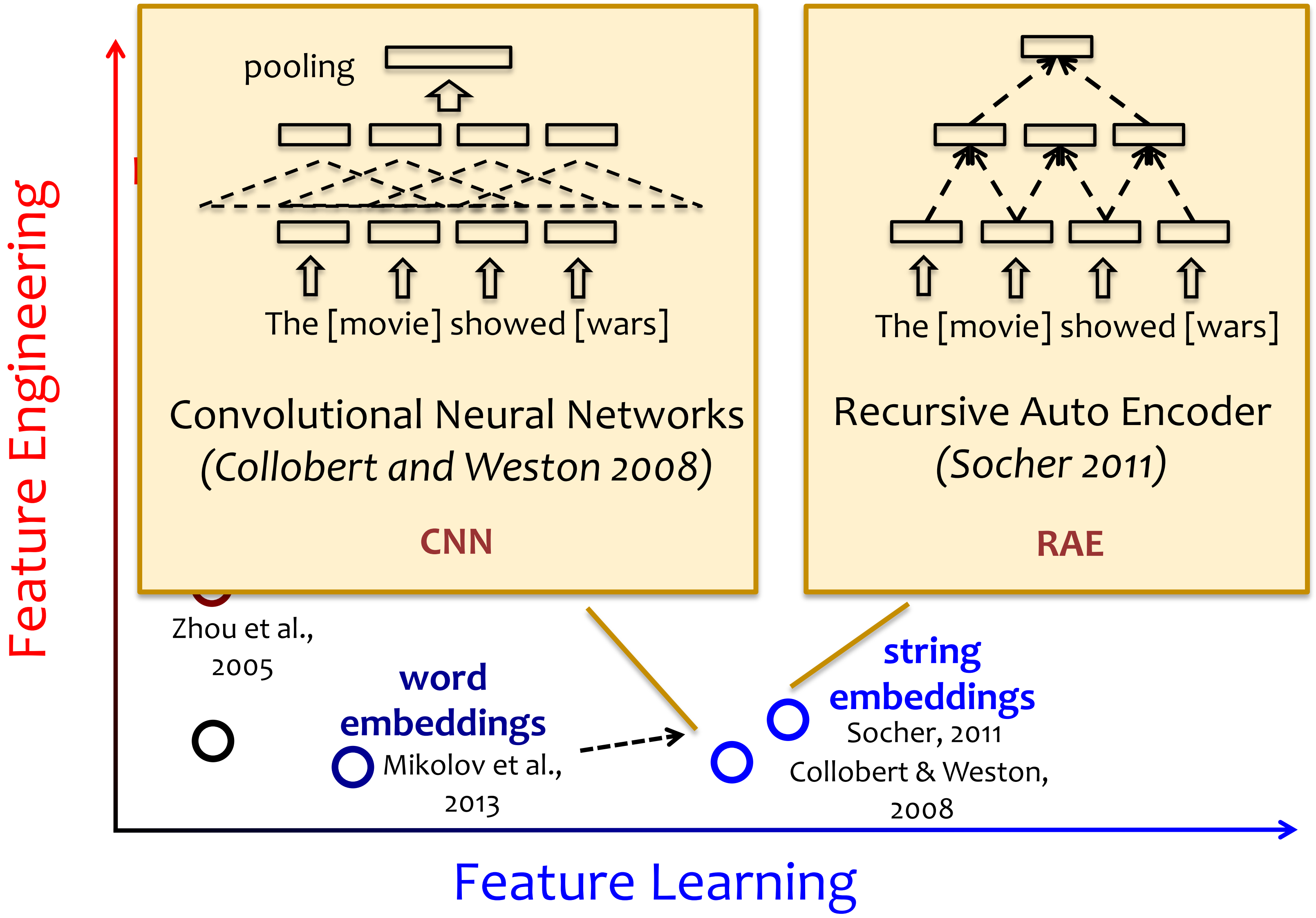
Where do features come from?



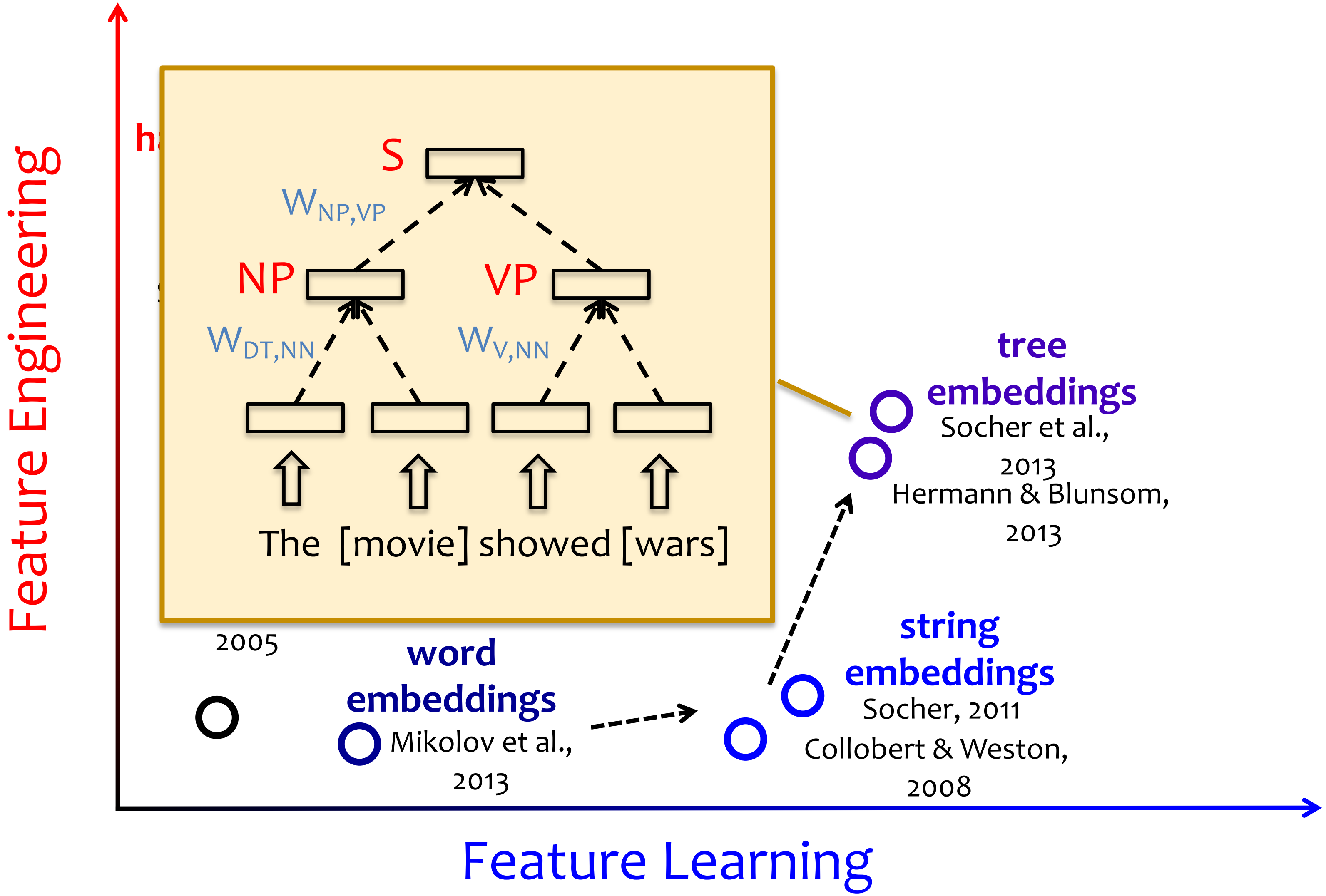
Where do features come from?



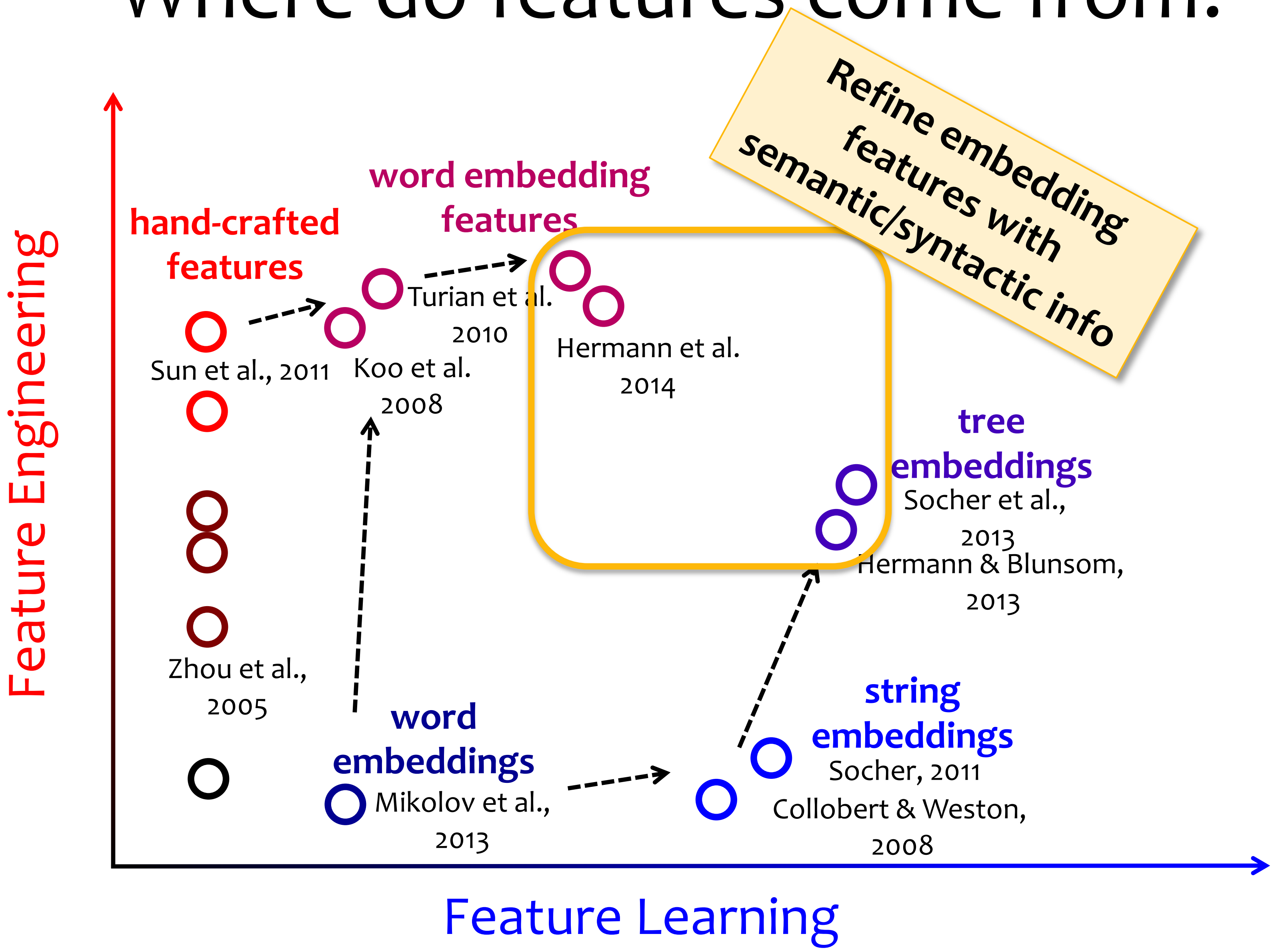
Where do features come from?



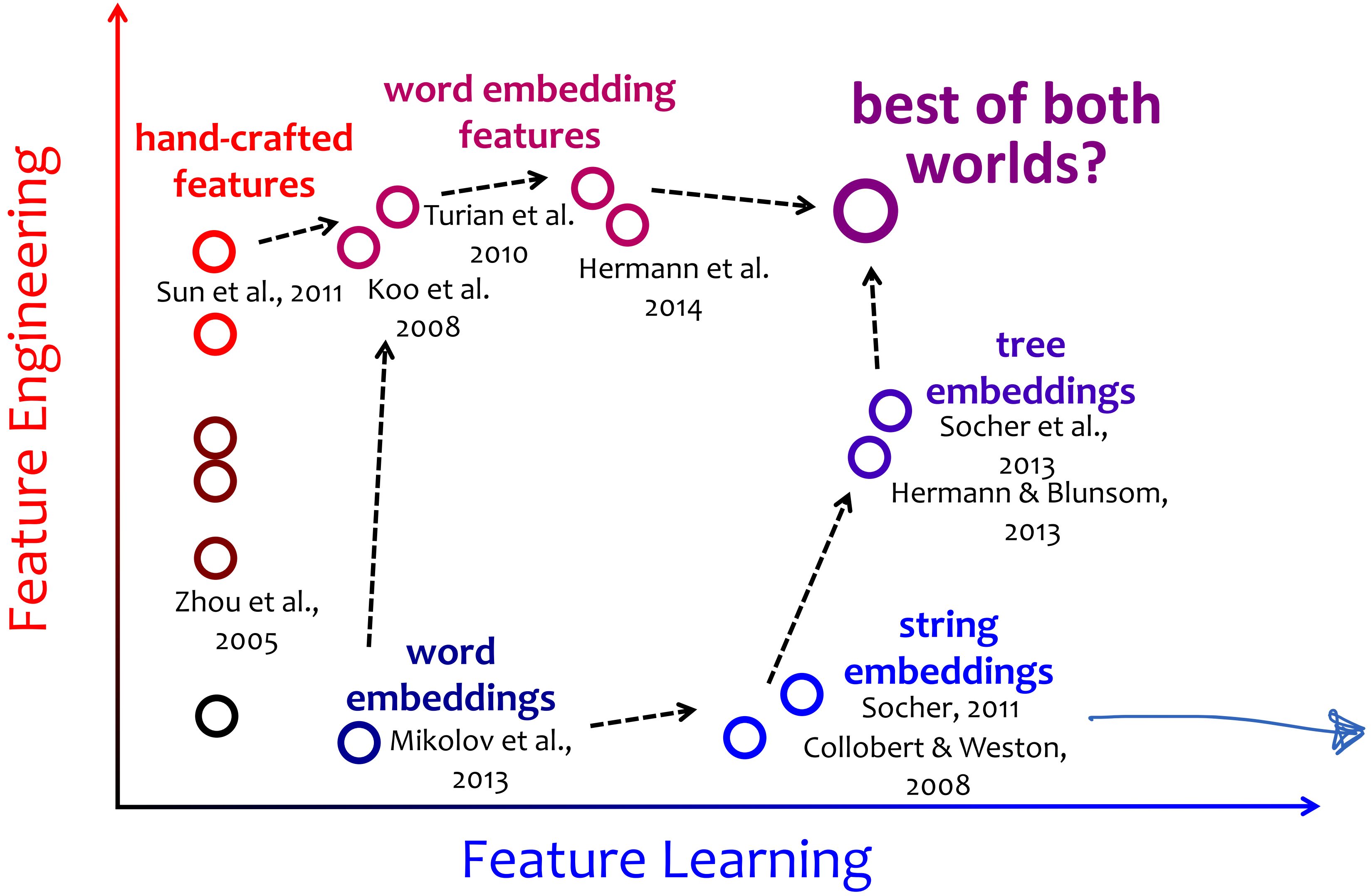
Where do features come from?



Where do features come from?



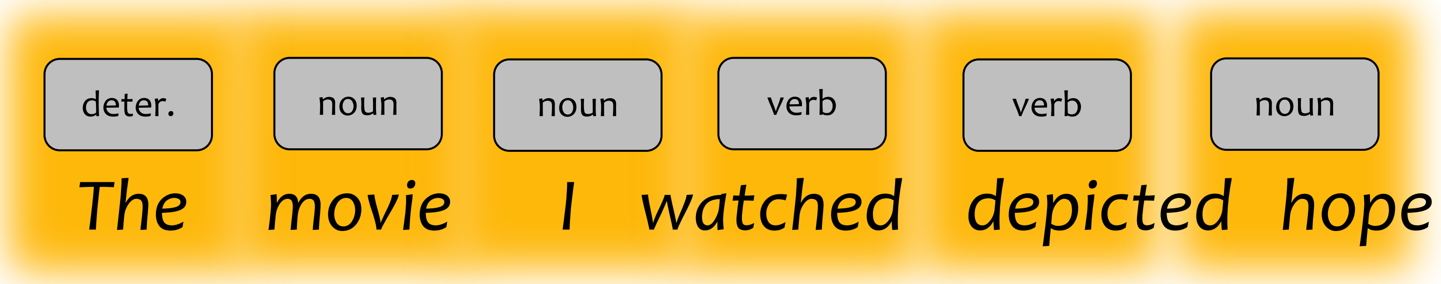
Where do features come from?



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:

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

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$
…	…	…	…	…	…	…
`wi == "I"`	0	0	1	0	0	0
`wi+1 == "I"`	0	1	0	0	0	0
`wi-1 == "I"`	0	0	0	1	0	0
`wi+2 == "I"`	1	0	0	0	0	0
`wi-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%

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

Background: Word Embeddings

One-hot vectors

- Standard representation of a word in NLP: 1-hot vector (aka. a string)
- Vectors representing related words share nothing in common

$M = 50{,}000$

	a	and	be	cat	dog		you	zebra
cat:	**0**	**0**	**0**	**1**	**0**	**…**	**0**	**0**
dog:	**0**	**0**	**0**	**0**	**1**	**…**	**0**	**0**

Word embeddings

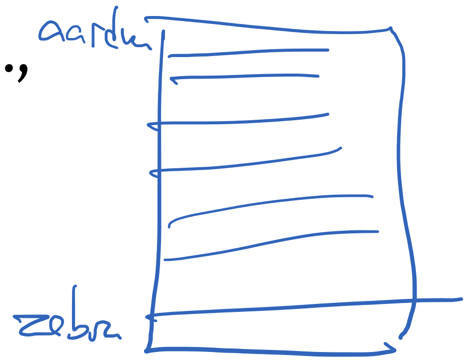
- Word embedding: real-valued vector representation of a word in *M* dimensions
- Related words have similar vectors
- Long history in NLP: Term-doc frequency matrices, Reduce dimensionality with {LSA, NNMF, CCA, PCA}, Brown clusters, Vector space models, Random projections, Neural networks / deep learning

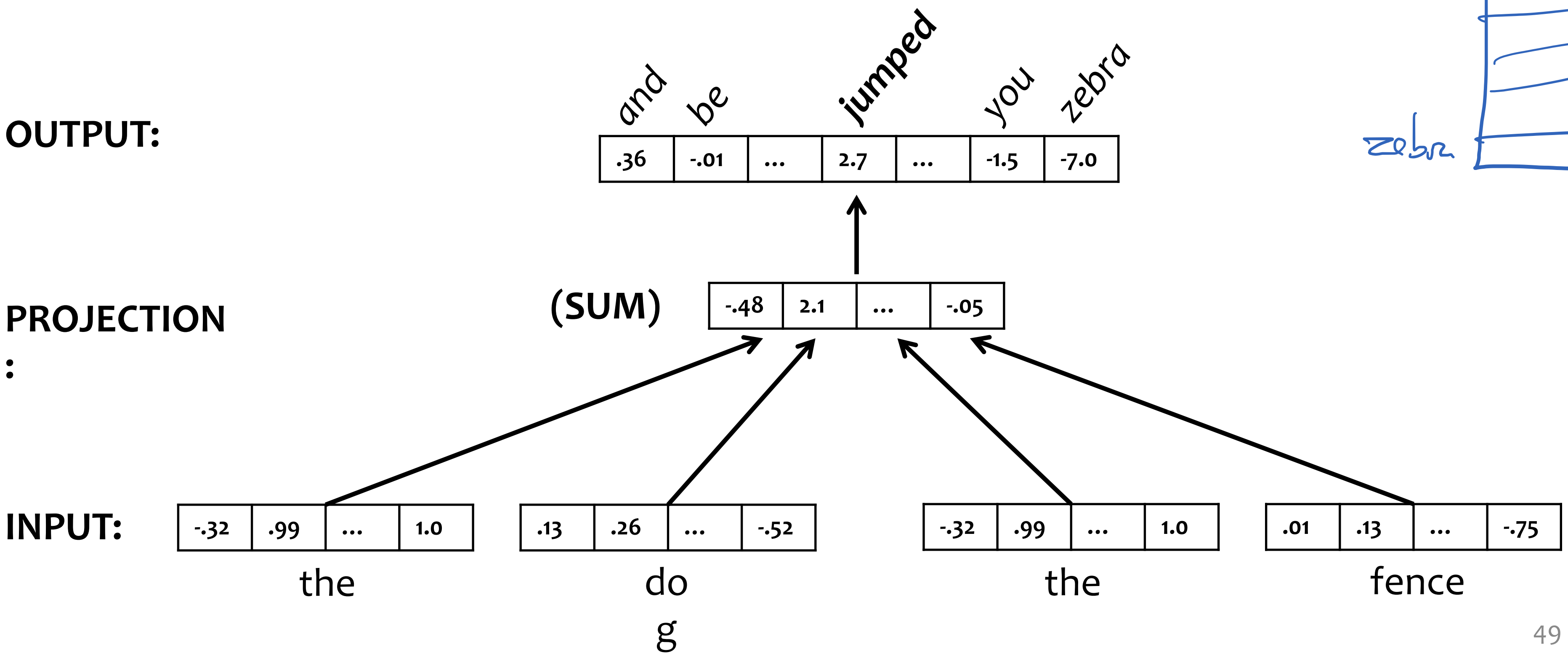
$\boxed{M = 512}$

cat:	**0.13**	**.26**	**…**	**-.52**
dog:	**0.11**	**.23**	**…**	**-.45**

Background: Word Embeddings

- It’s common to use **neural-network** trained embeddings
  - Key idea: learn embeddings which are good at reconstructing the context of a word
  - Popular across HLT (speech, NLP)
- The Continuous Bag-of-words Model (CBOW) (Mikolov et al., 2013) maximizes the likelihood of a word given its context:

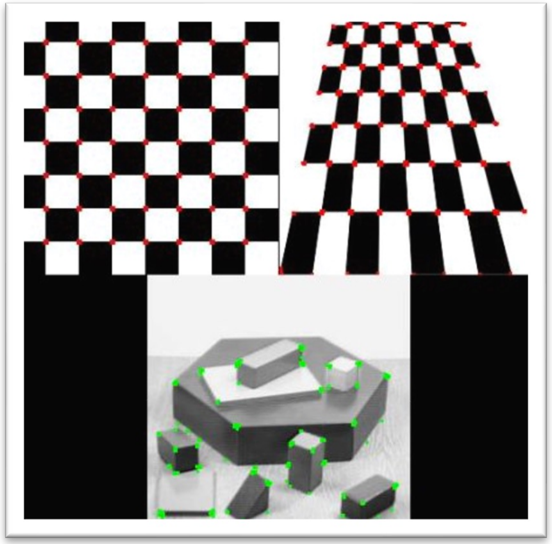




Feature Engineering for CV

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.

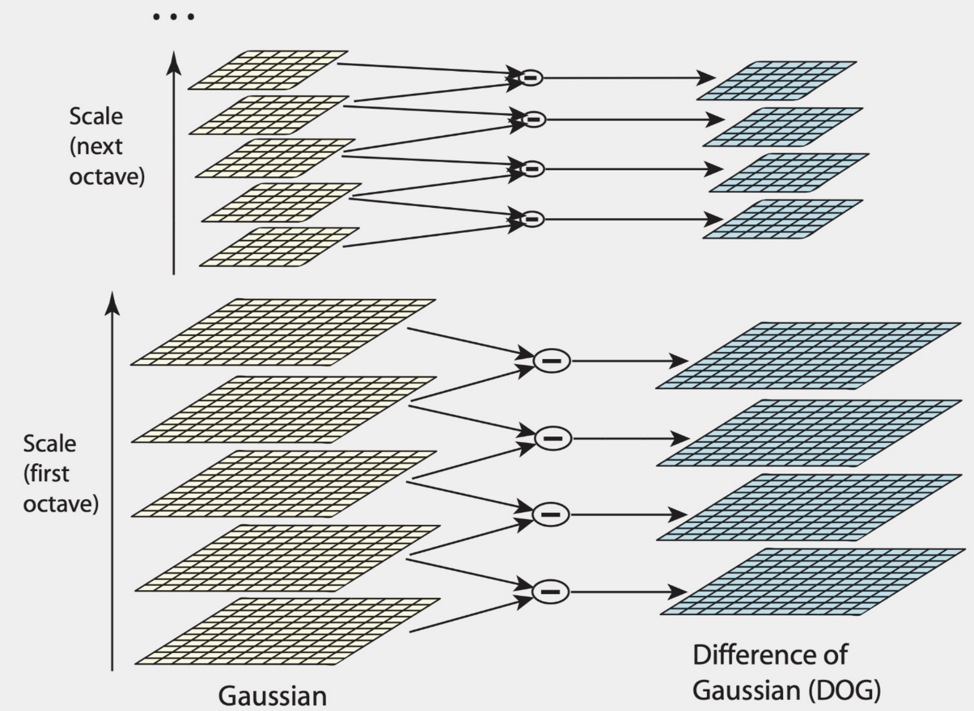


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled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

Figure from Lowe (1999) and Lowe (2004)

Feature Engineering

Question:

the time is 7:30am

Suppose you are building a classification model to predict the reason that CMU's campus has so many building entrances closed off by security this morning.

What features would you use?

Answer:

1. whereabouts of pres. cand.
2. time of year
3. # of police cars on Forbes
4. proxity to election
5. weather
6. PA swing state
7. recent crime reports
8. check reddit

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**

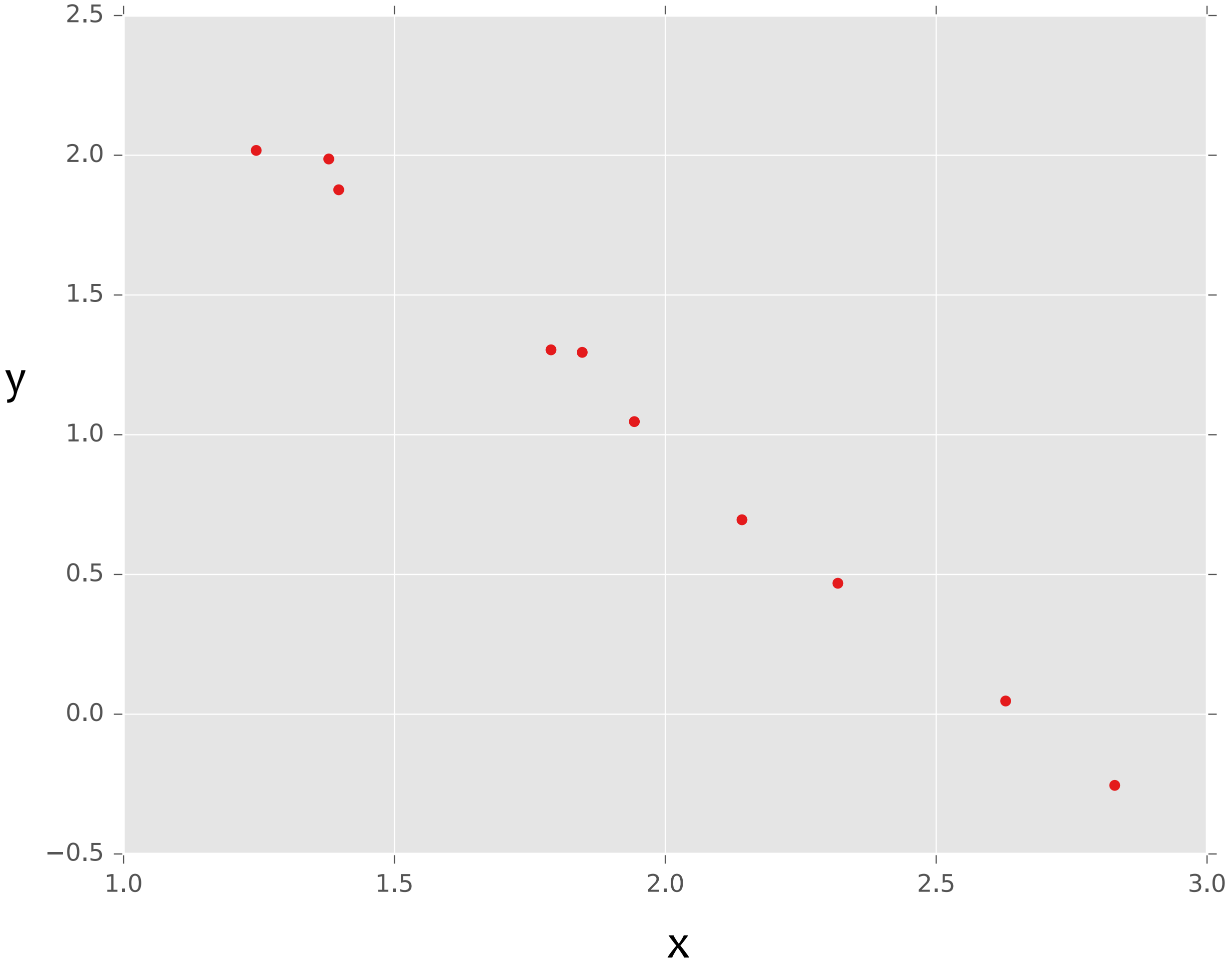
Examples:
- Perceptron
- Linear regression
- Logistic regression

Example: Linear Regression

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

i	y	x
1	2.0	1.2
2	1.3	1.7
…	…	…
10	1.1	1.9

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

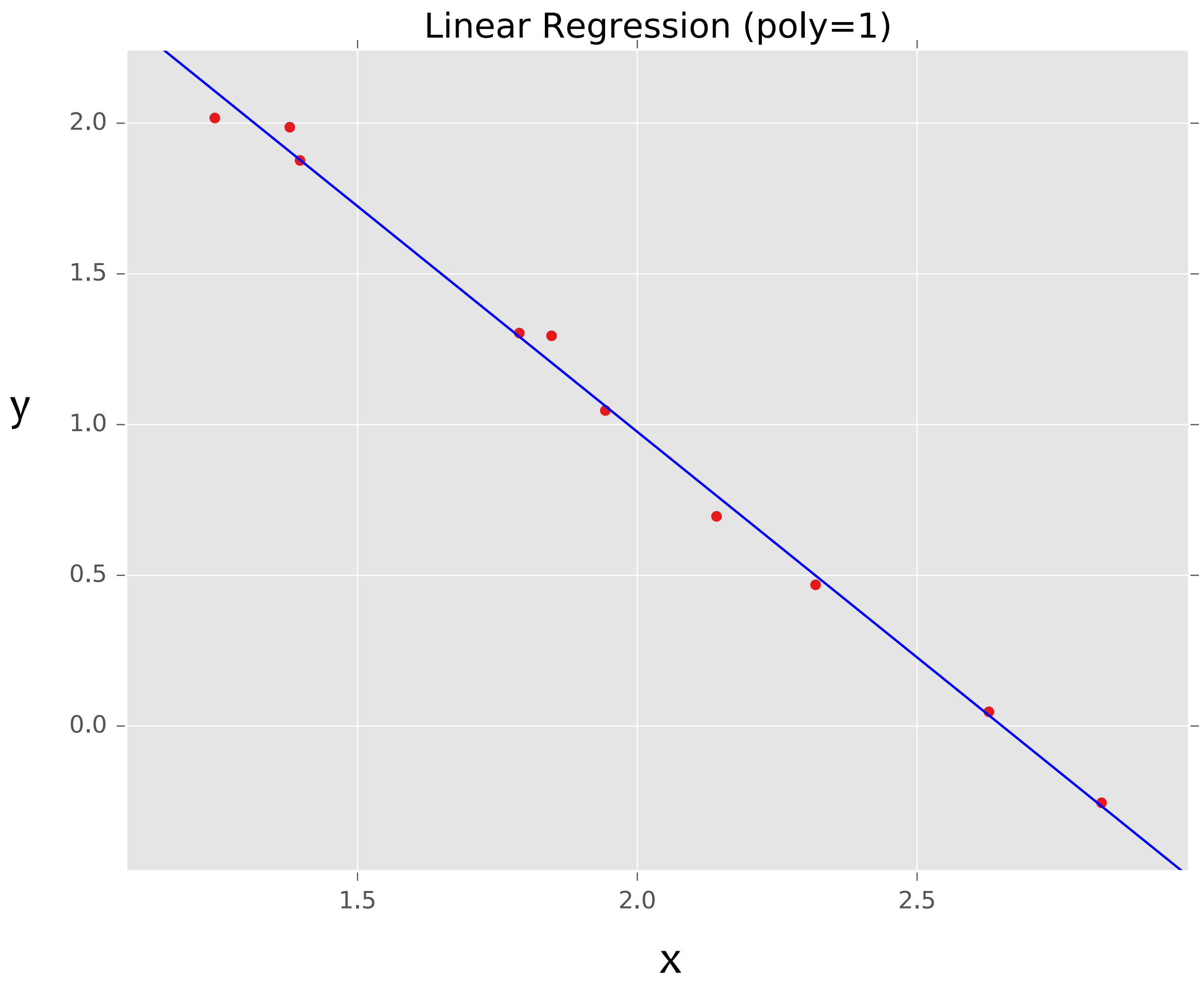


Example: Linear Regression

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

i	y	x
1	2.0	1.2
2	1.3	1.7
…	…	…
10	1.1	1.9

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

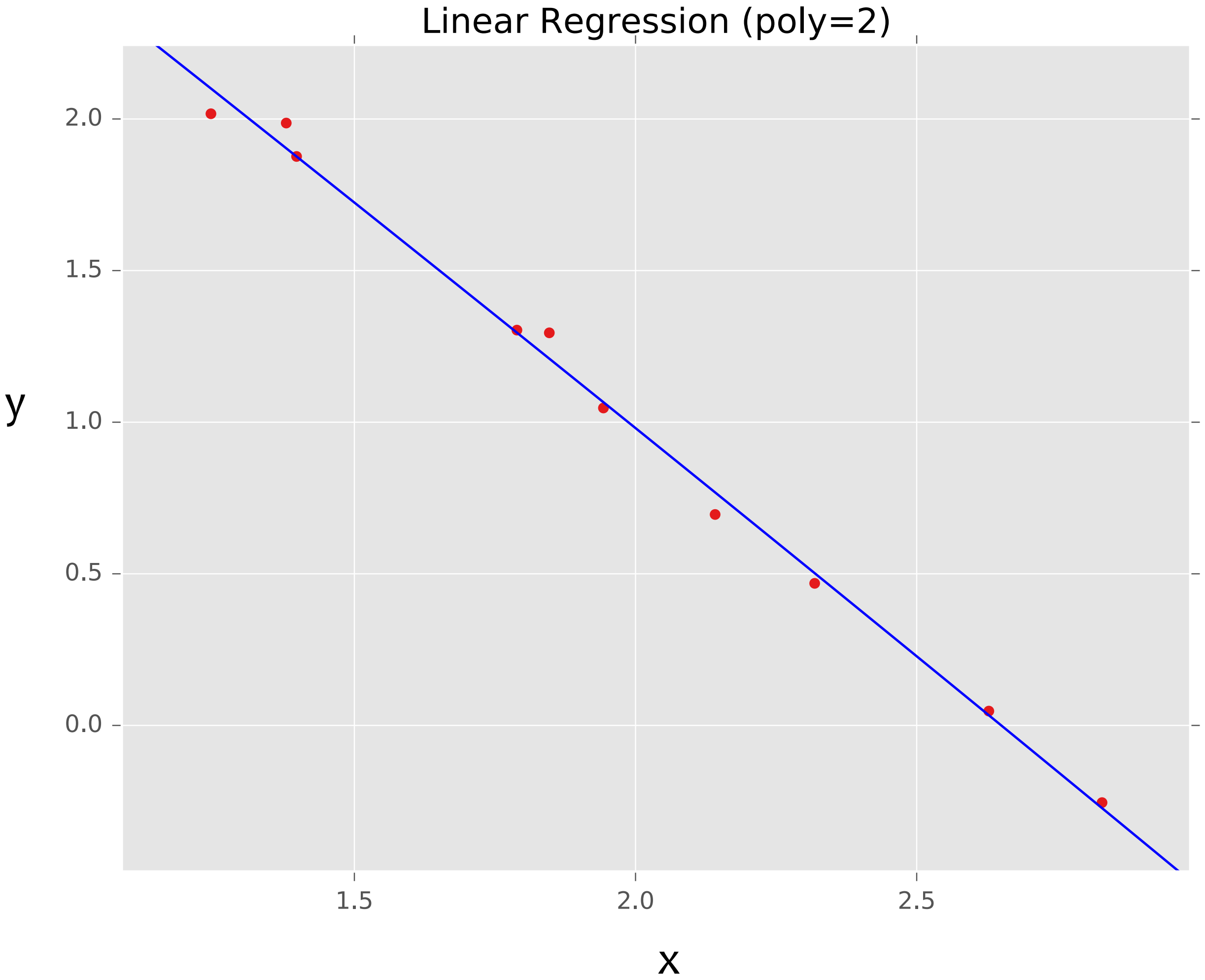


Example: Linear Regression

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

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

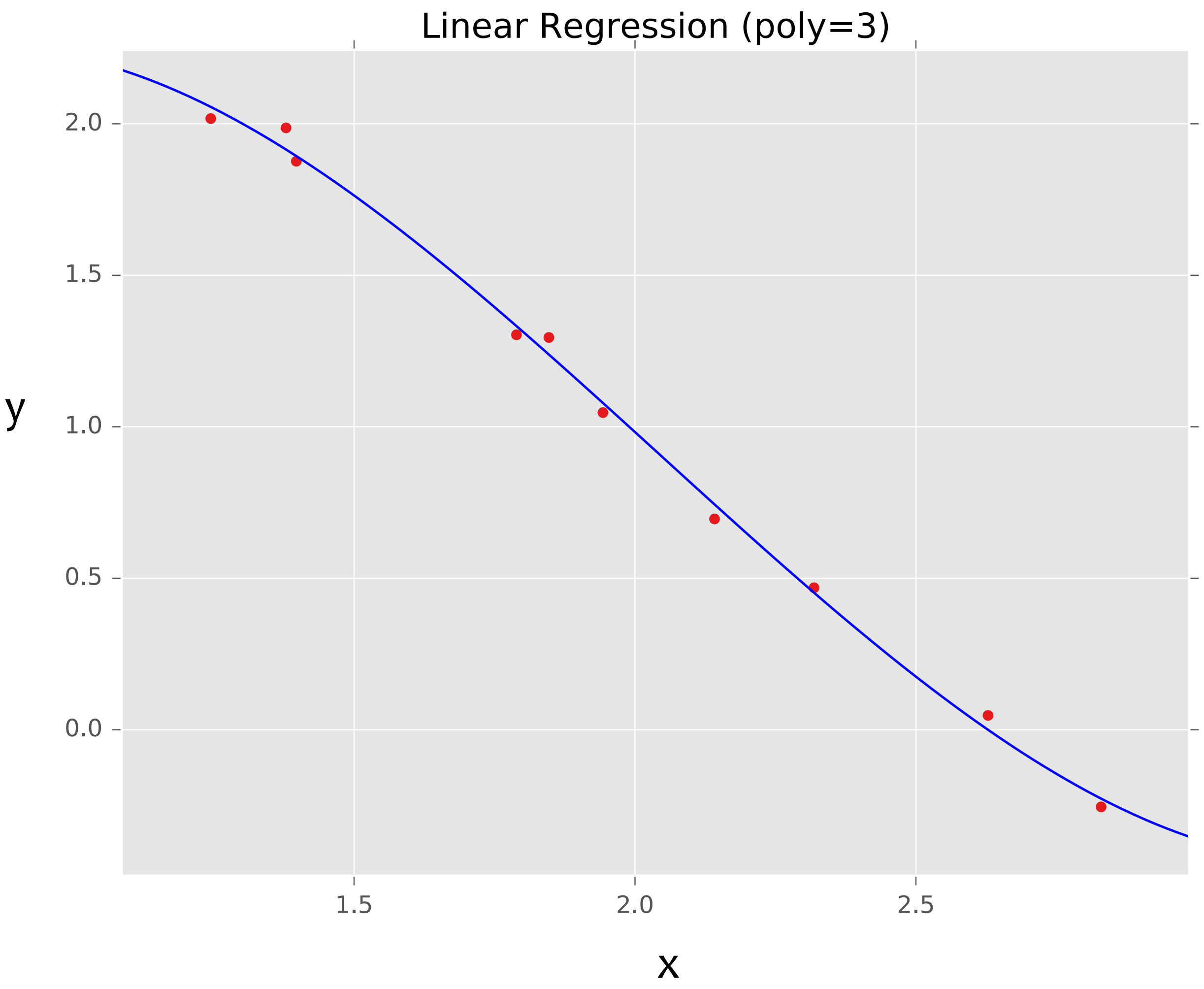
true "unknown" target function is linear with negative slope and gaussian noise



Example: Linear Regression

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

i	y	x	x^2	x^3
1	2.0	1.2	$(1.2)^2$	$(1.2)^3$
2	1.3	1.7	$(1.7)^2$	$(1.7)^3$
…	…	…	…	…
10	1.1	1.9	$(1.9)^2$	$(1.9)^3$



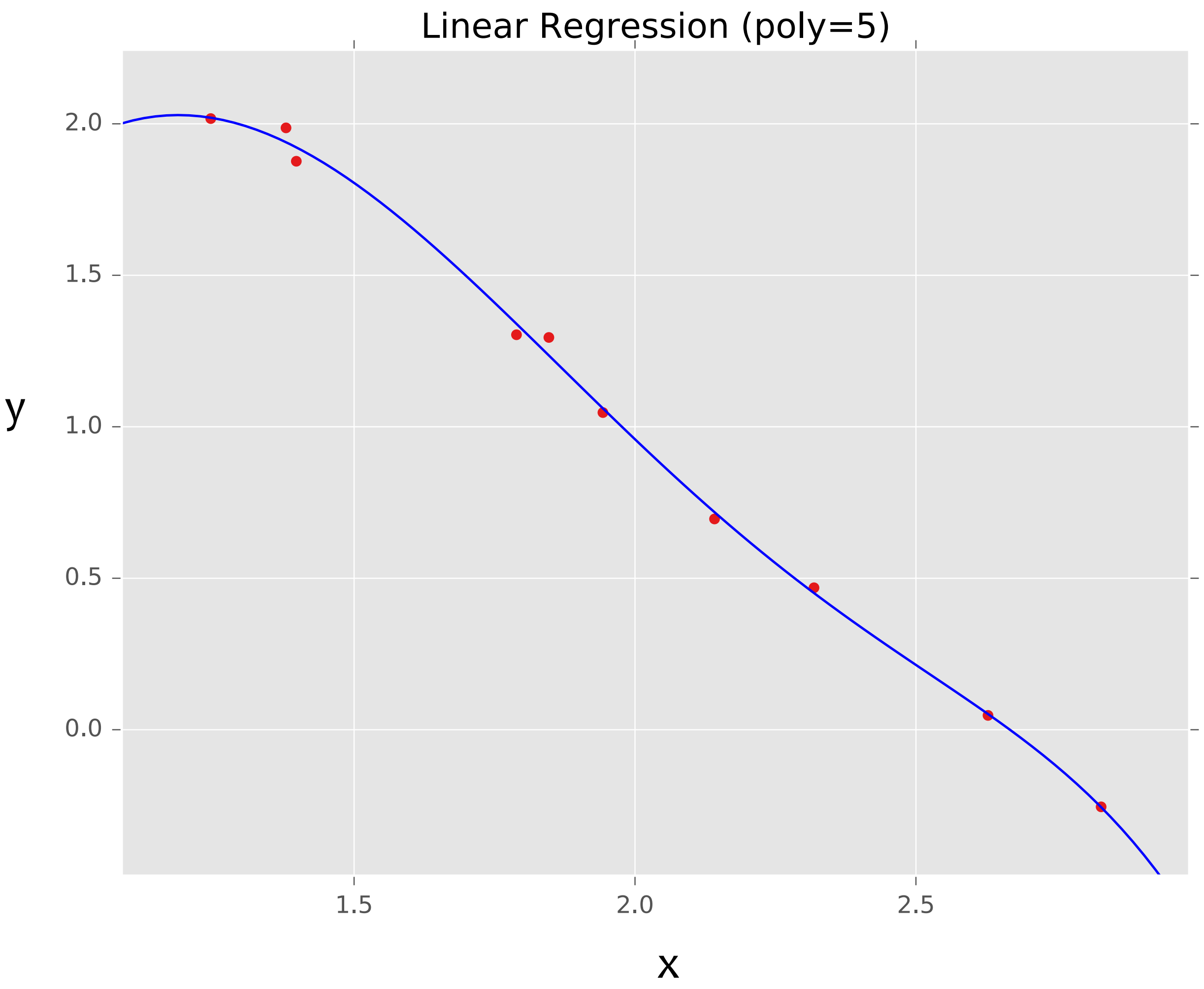
true "unknown" target function is linear with negative slope and gaussian noise

Example: Linear Regression

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

i	y	x	…	x^5
1	2.0	1.2	…	$(1.2)^5$
2	1.3	1.7	…	$(1.7)^5$
…	…	…	…	…
10	1.1	1.9	…	$(1.9)^5$

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

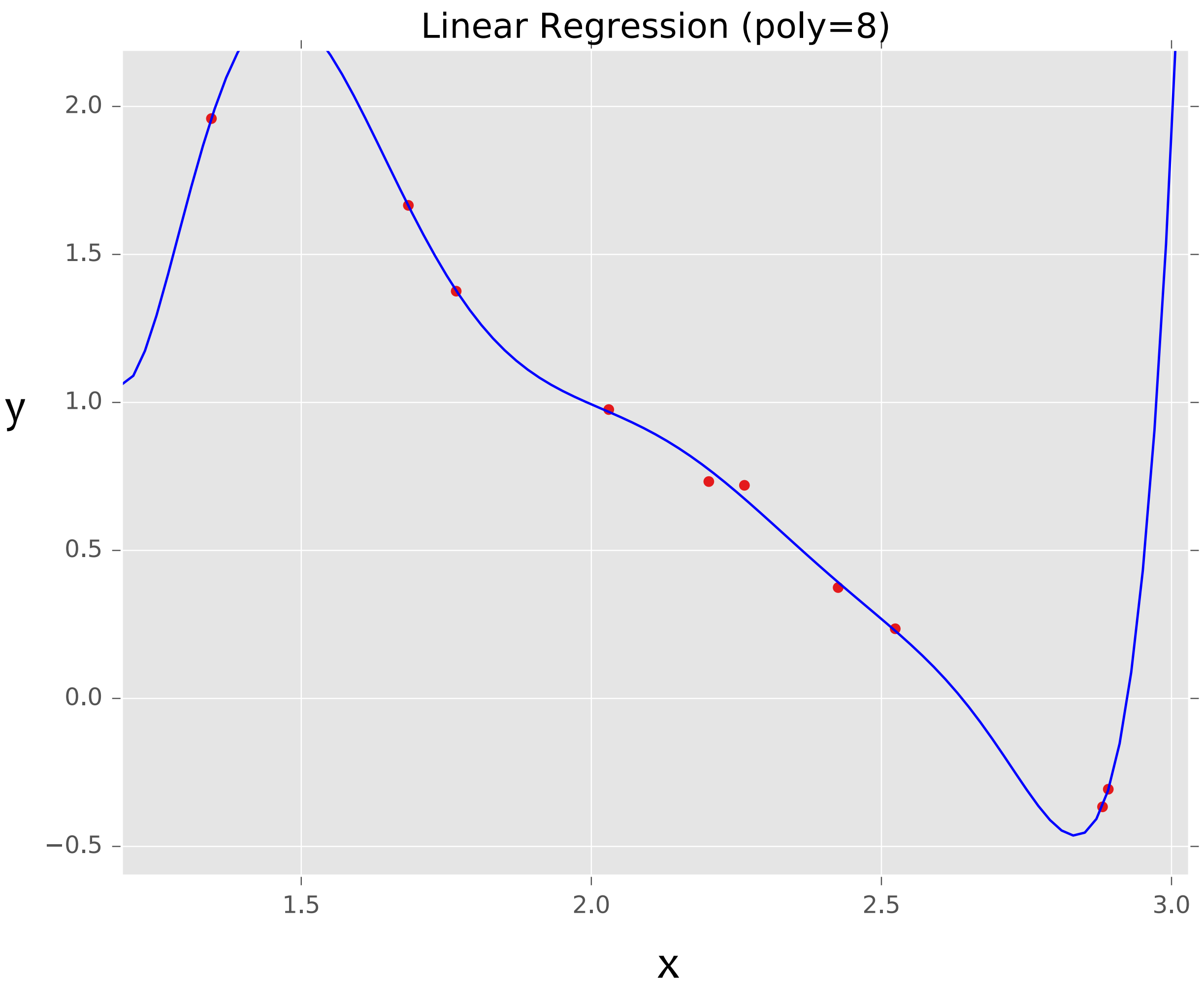


Example: Linear Regression

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

i	y	x	…	x^8
1	2.0	1.2	…	$(1.2)^8$
2	1.3	1.7	…	$(1.7)^8$
…	…	…	…	…
10	1.1	1.9	…	$(1.9)^8$

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

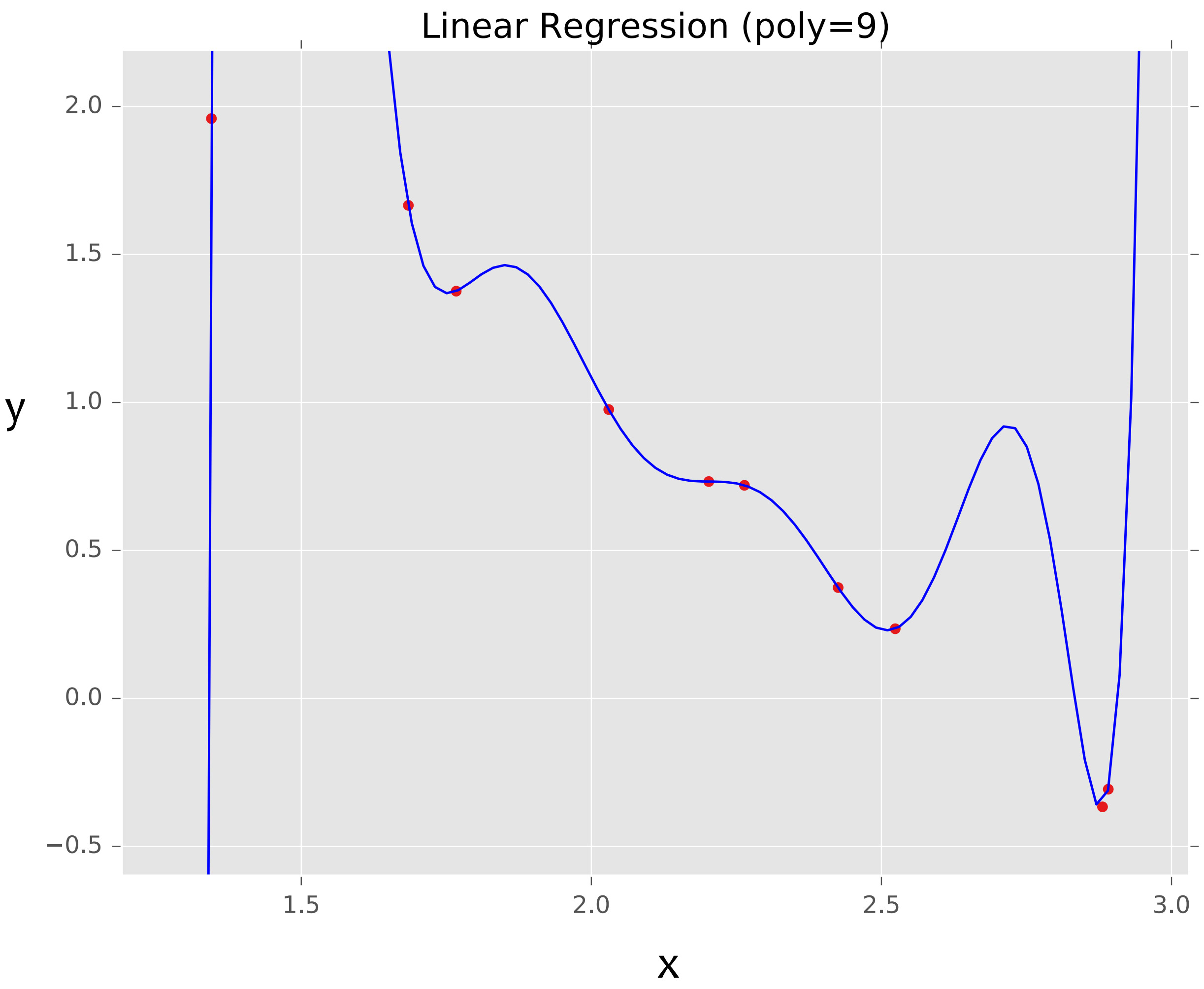


Example: Linear Regression

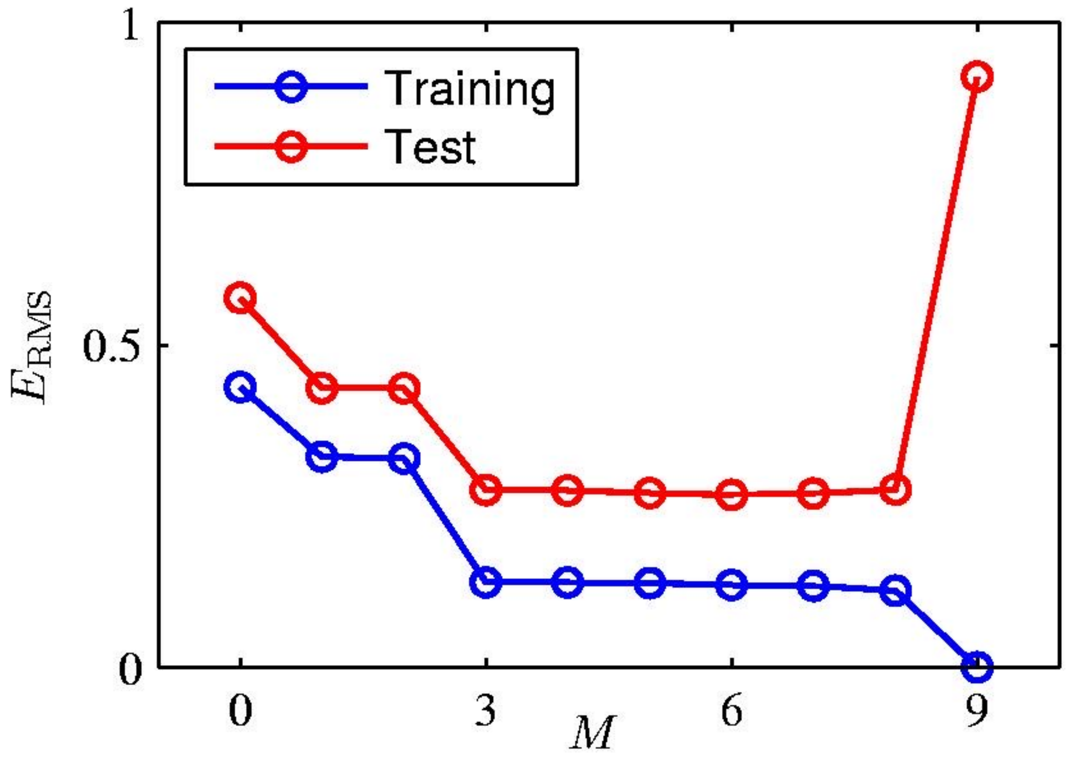
Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b
where f(.) is a polynomial
basis function

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	…	$(1.9)^9$

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



Over-fitting



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

Polynomial Coefficients

$\vec{\Theta}^T \vec{x}$

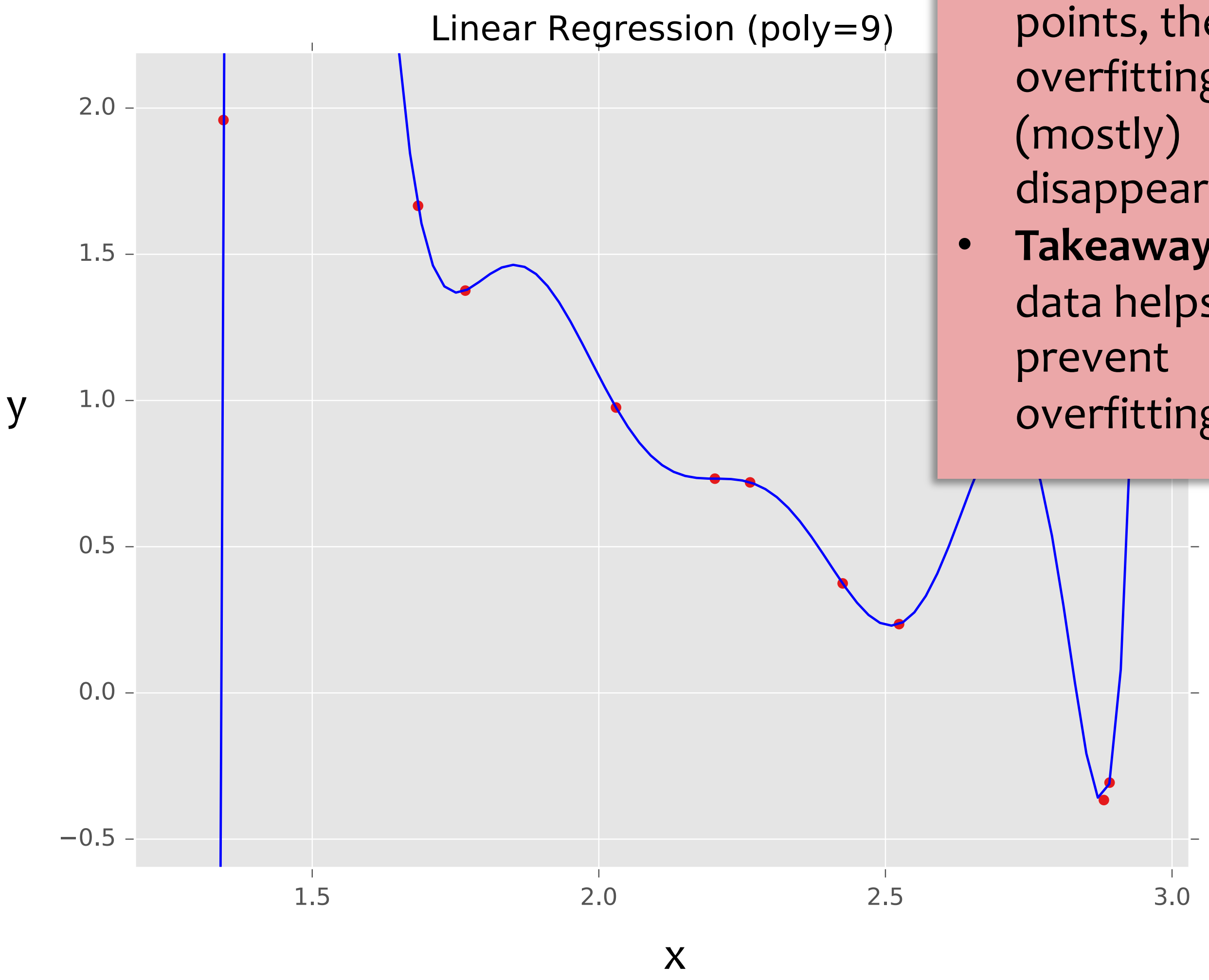
$\Theta_0 x_0 + \Theta_1 x_1$

		$M=0$	$M=1$	$M=3$	$M=9$
b	θ_0	0.19	0.82	0.31	0.35
w_1	θ_1		-1.27	7.99	232.37
w_2	θ_2		0	-25.43	-5321.83
w_3	θ_3		0	17.37	48568.31
	θ_4		0		-231639.30
	θ_5		0		640042.26
	θ_6		0		-1061800.52
	θ_7		0		1042400.18
	θ_8		0		-557682.99
w_9	θ_9		0		125201.43

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	…	$(1.9)^9$

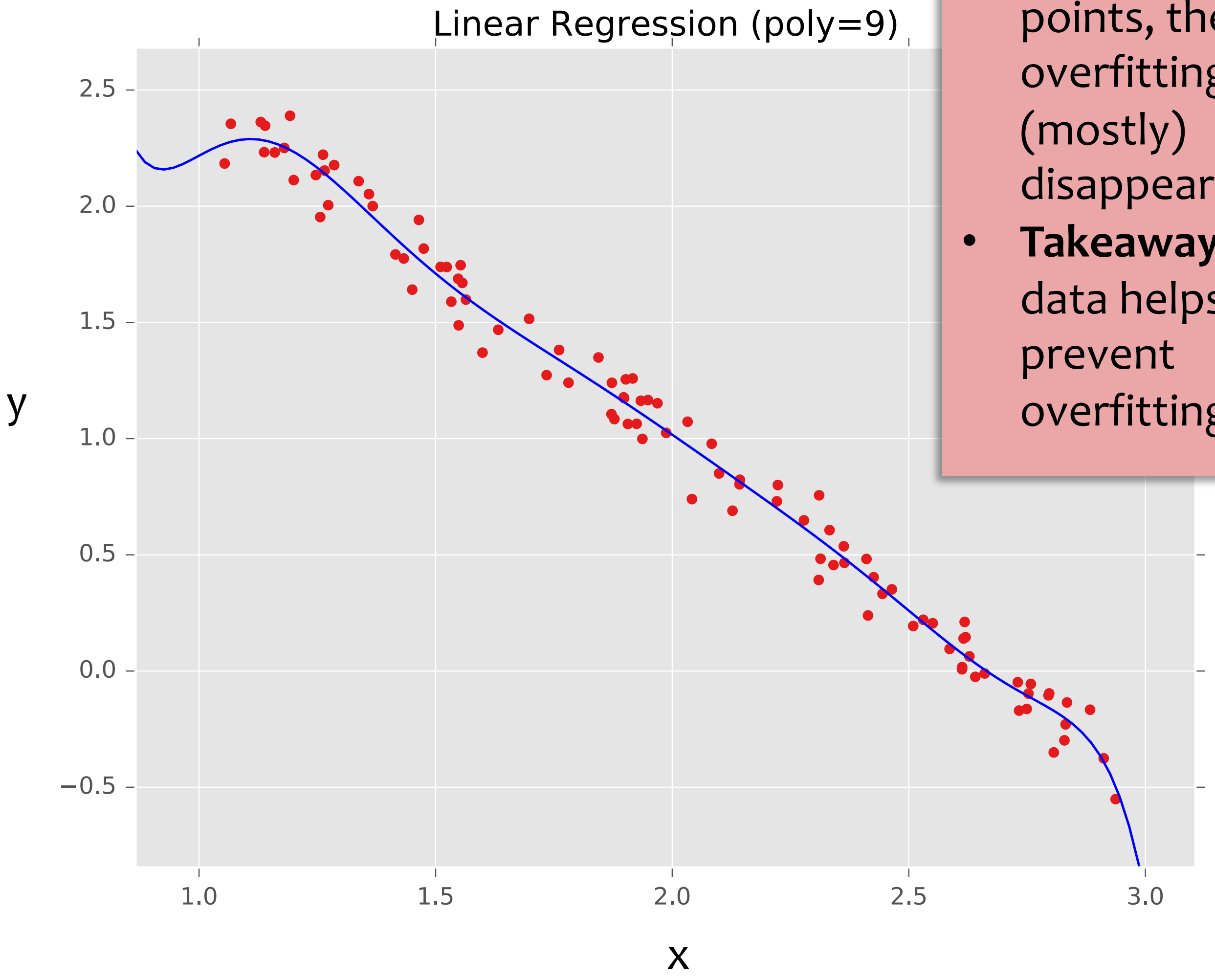


- With just N = 10 points we overfit!
- But with N = 100 points, the overfitting (mostly) disappears
- **Takeaway:** more data helps prevent overfitting

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

- **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

hyperparameter

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

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 Examples

Add an **L2 regularizer** to Linear Regression (aka. Ridge Regression)

$$J_{\text{RR}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_2^2}$$

$$= \underbrace{\frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2}_{\text{MSE}} + \underbrace{\boxed{\lambda \sum_{m=1}^{M} \theta_m^2}}_{\text{L2}}$$

Add an **L1 regularizer** to Linear Regression (aka. LASSO)

$$J_{\text{LASSO}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_1}$$

$$= \underbrace{\frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2}_{\text{MSE}} + \underbrace{\boxed{\lambda \sum_{m=1}^{M} |\theta_m|}}_{\text{L1}}$$

Regularization Examples

Add an **L2 regularizer** to Logistic Regression

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_2^2}$$

$$= \underbrace{\frac{1}{N} \sum_{i=1}^{N} - \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})}_{l.l.} + \boxed{\lambda \sum_{m=1}^{M} \theta_m^2}$$

Add an **L1 regularizer** to Logistic Regression

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_1}$$

$$= \underbrace{\frac{1}{N} \sum_{i=1}^{N} - \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})}_{l.l.} + \boxed{\lambda \sum_{m=1}^{M} |\theta_m|}$$

Regularization

$J''(\theta) = (1-\lambda)J(\theta) + \lambda r(\theta)$

Question: 3

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)

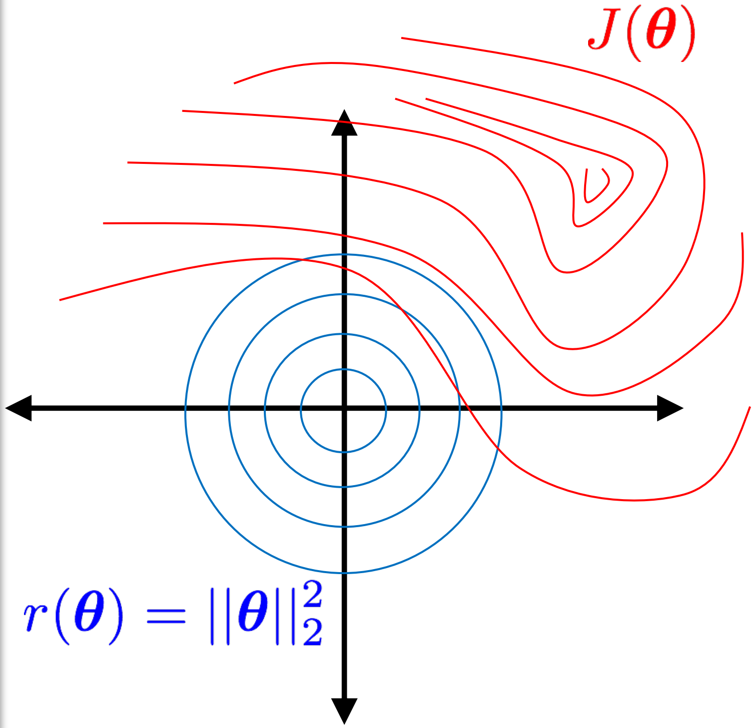
B. …move towards the minimum of J(θ)

70% 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~~ toxic



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

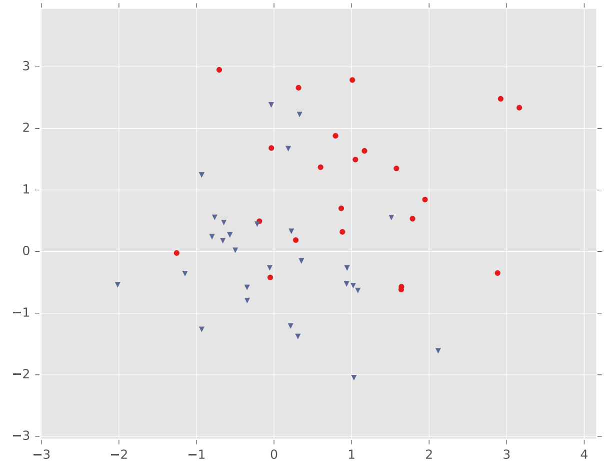
Standardizing Data

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

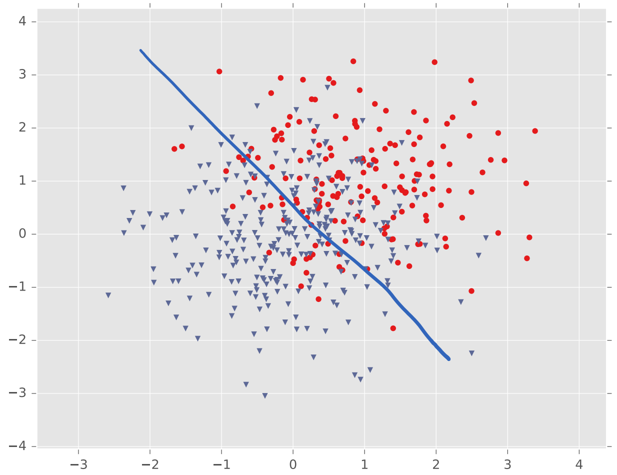
REGULARIZATION EXAMPLE: LOGISTIC REGRESSION

Example: Logistic Regression

Training Data

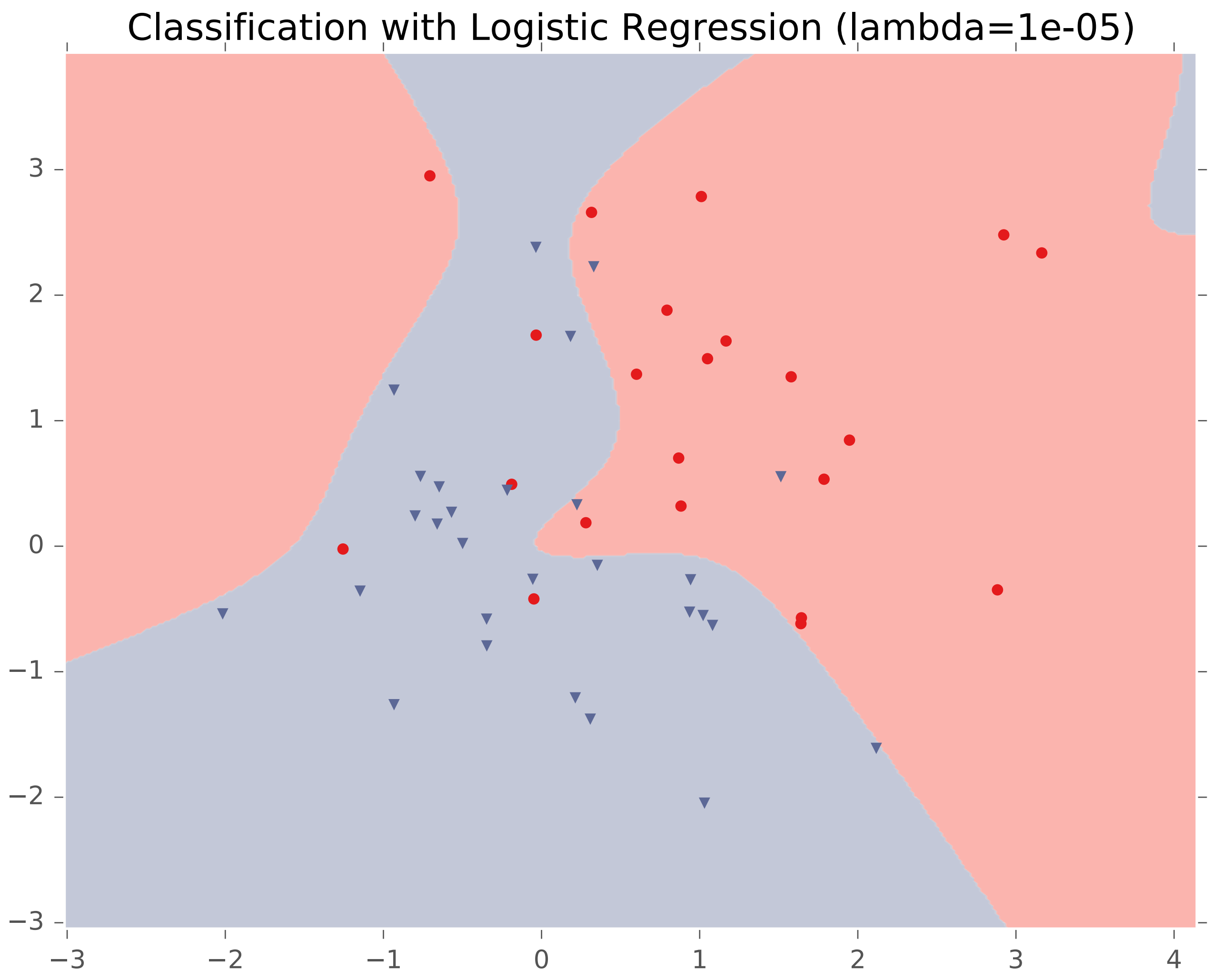


Test Data

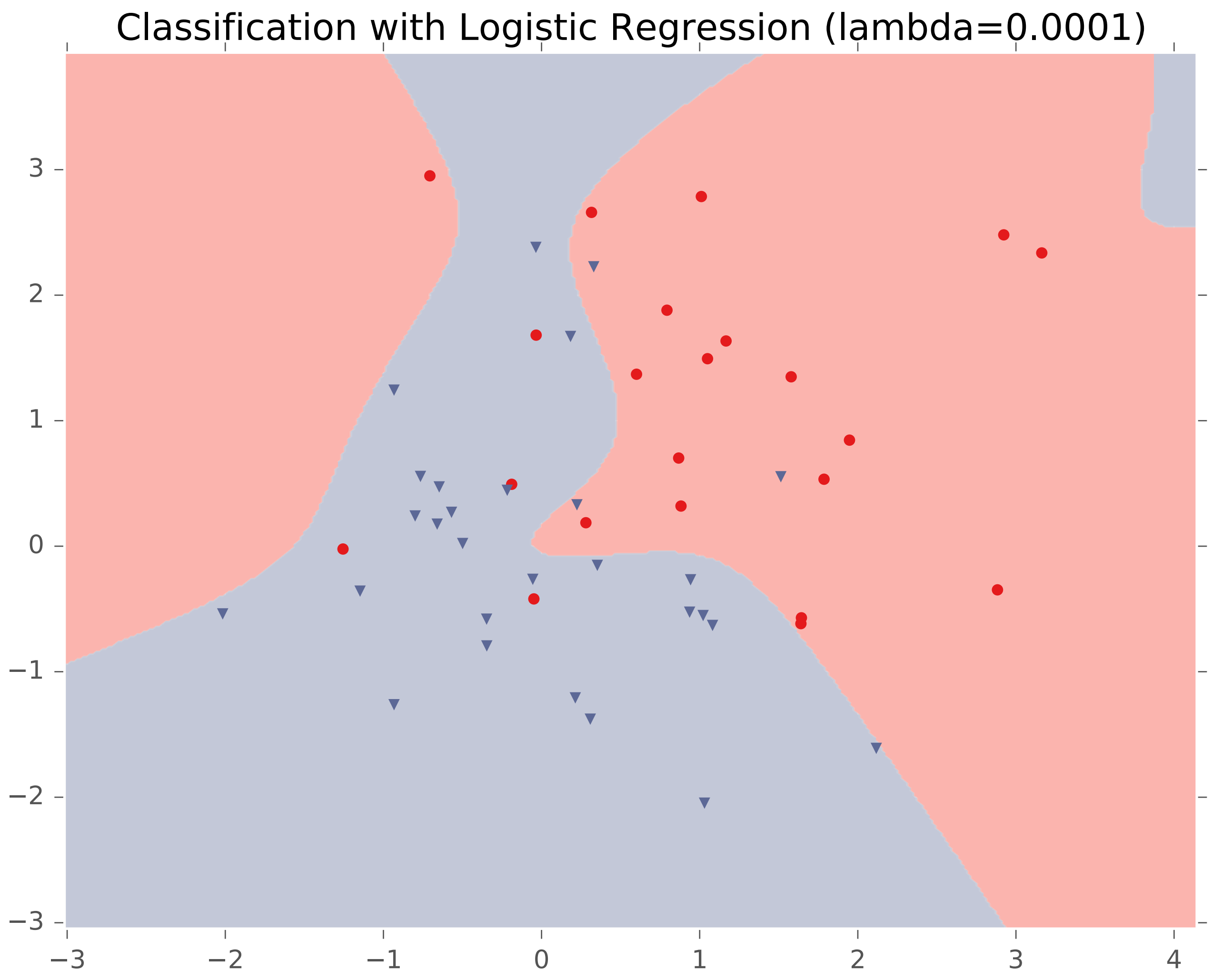


- For this example, we construct **nonlinear features** (i.e. feature engineering)
- Specifically, we add **polynomials up to order 9** of the two original features x_1 and x_2
- Thus our classifier is **linear** in the **high-dimensional feature space,** but the decision boundary is **nonlinear** when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

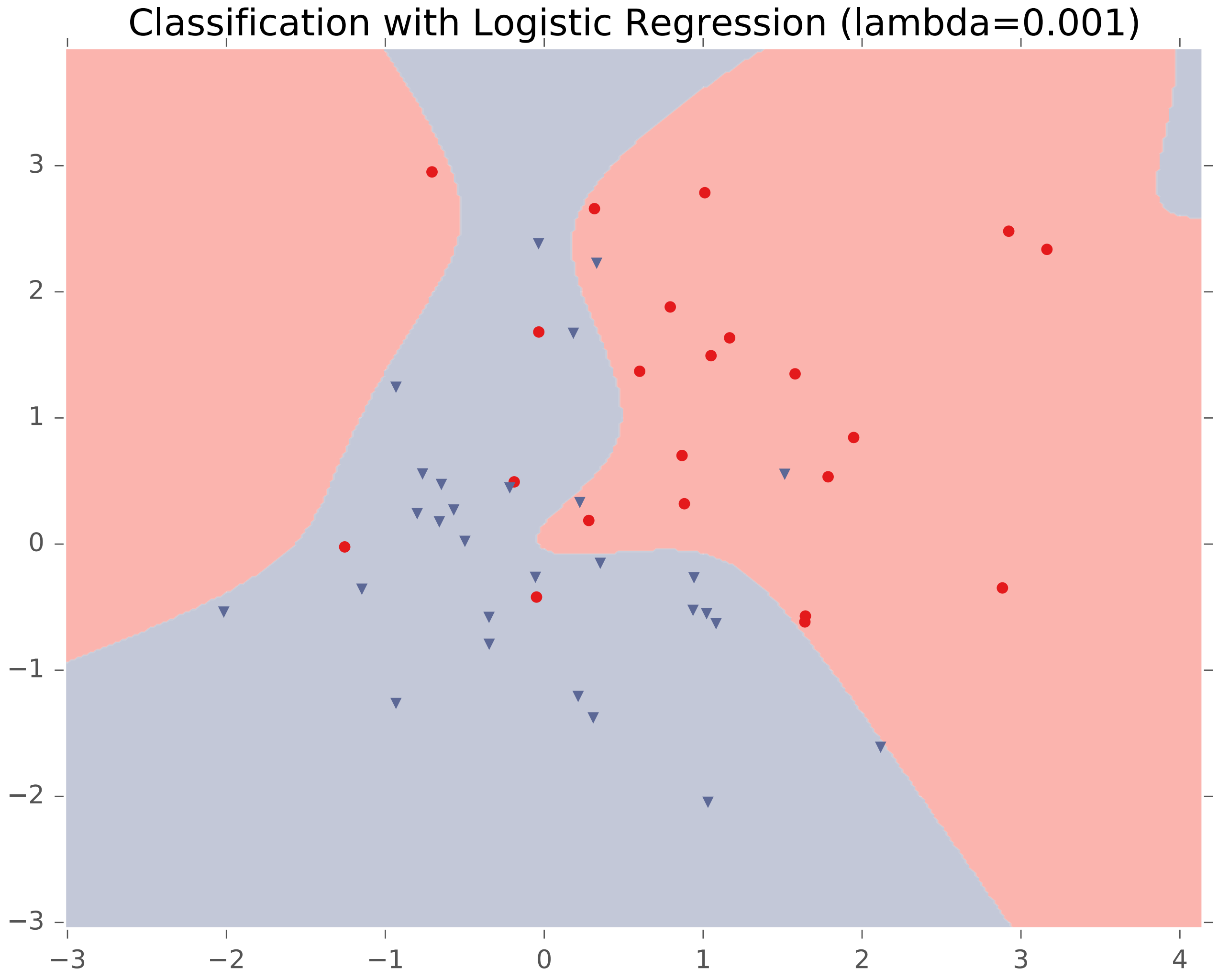
Example: Logistic Regression



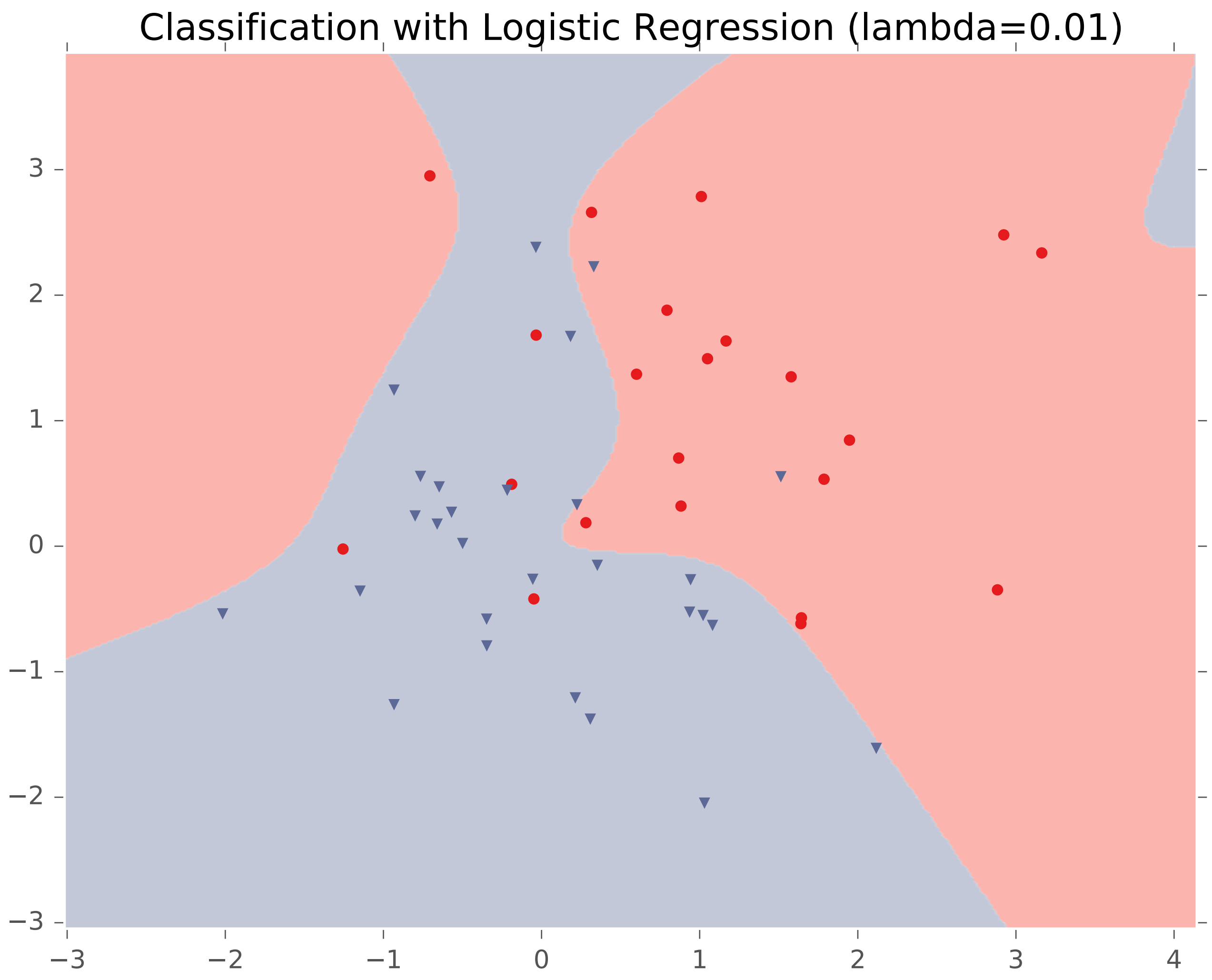
Example: Logistic Regression



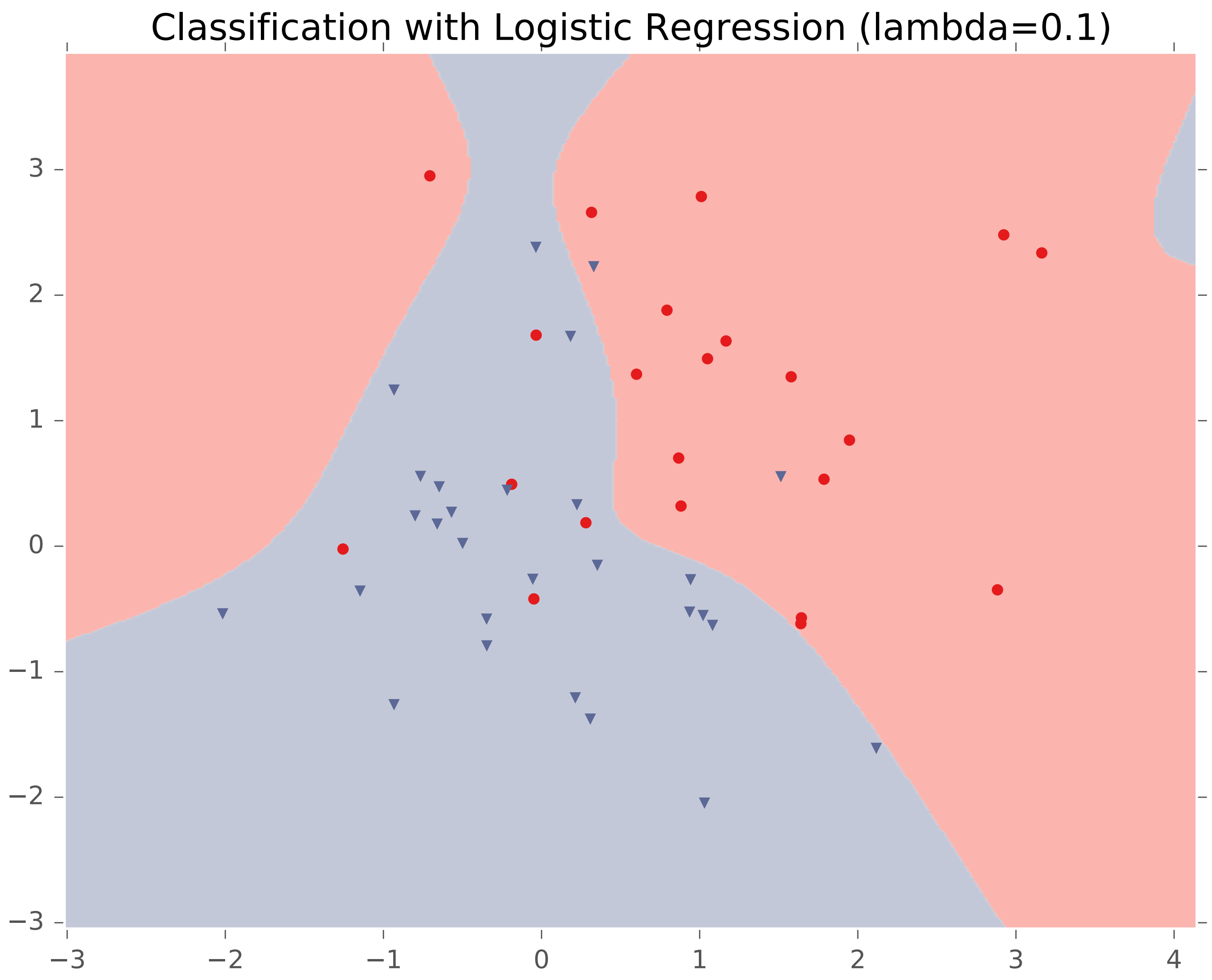
Example: Logistic Regression



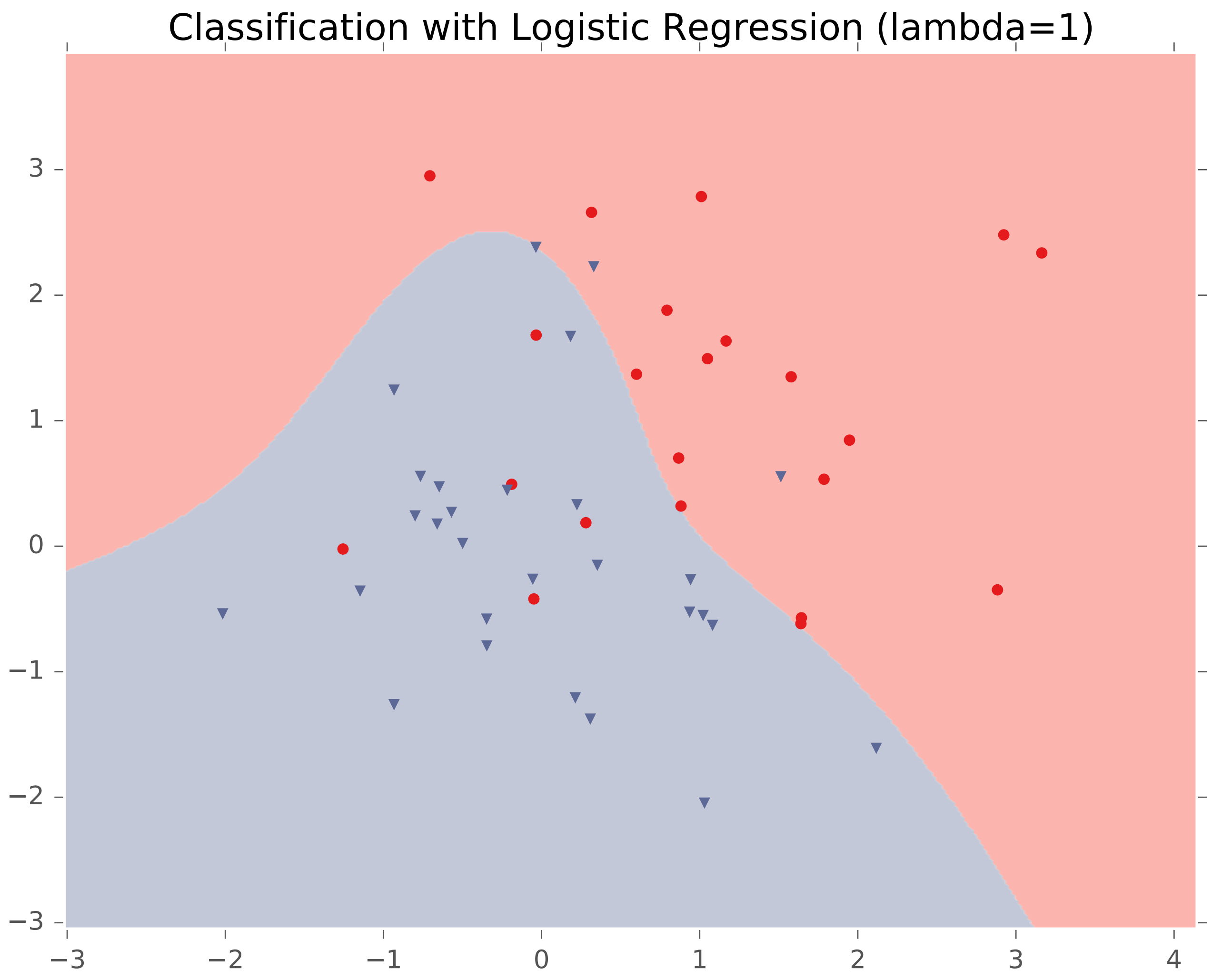
Example: Logistic Regression



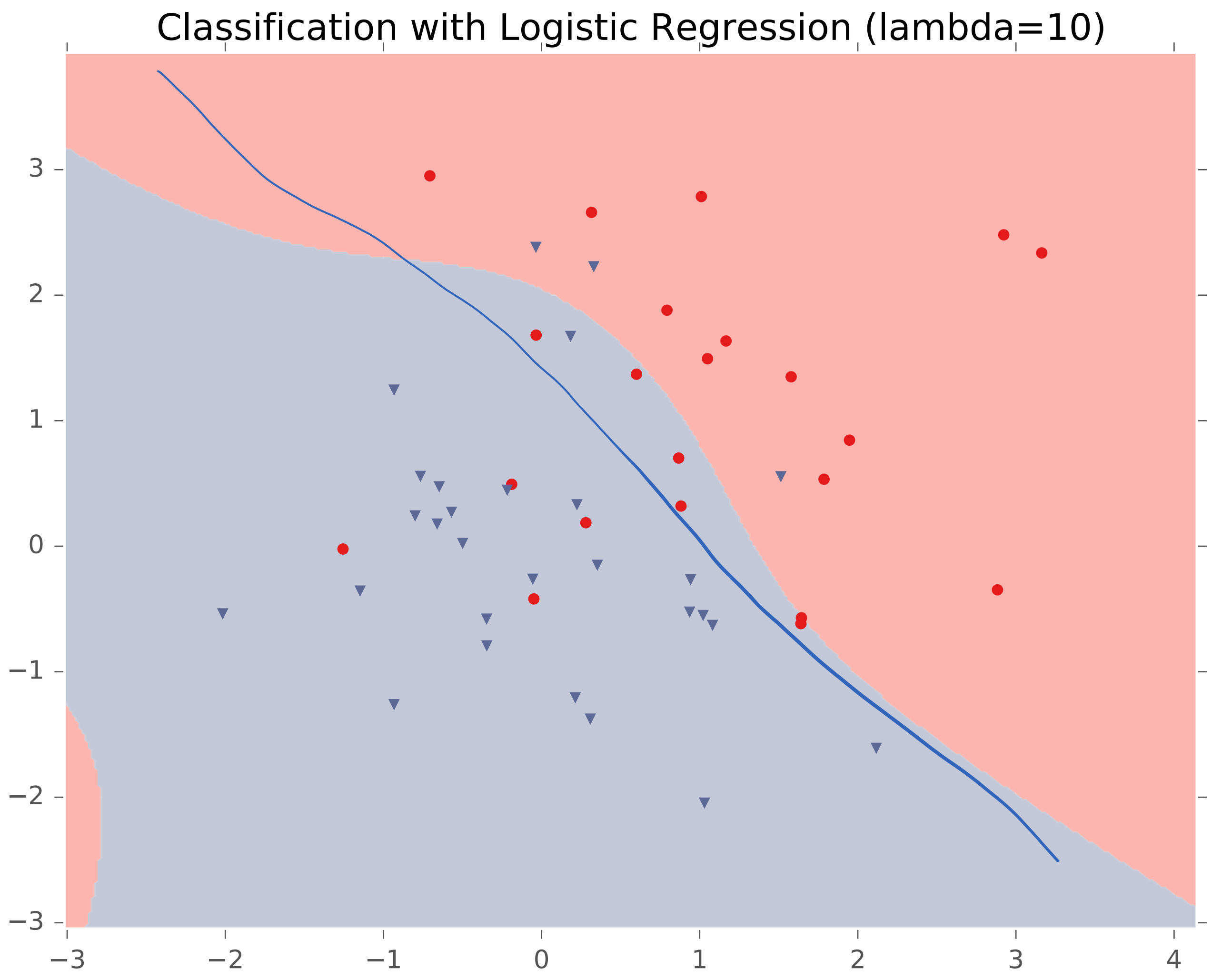
Example: Logistic Regression



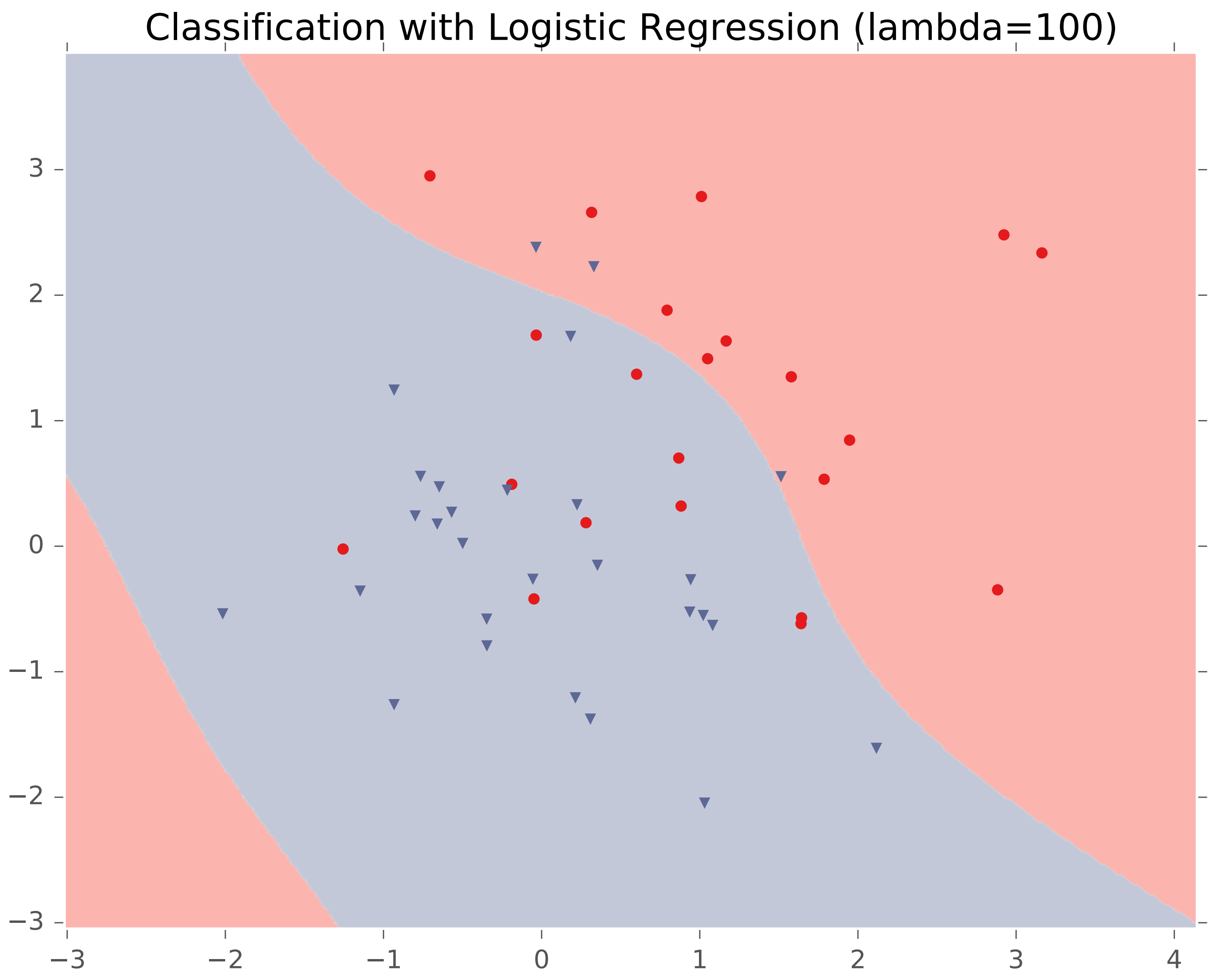
Example: Logistic Regression



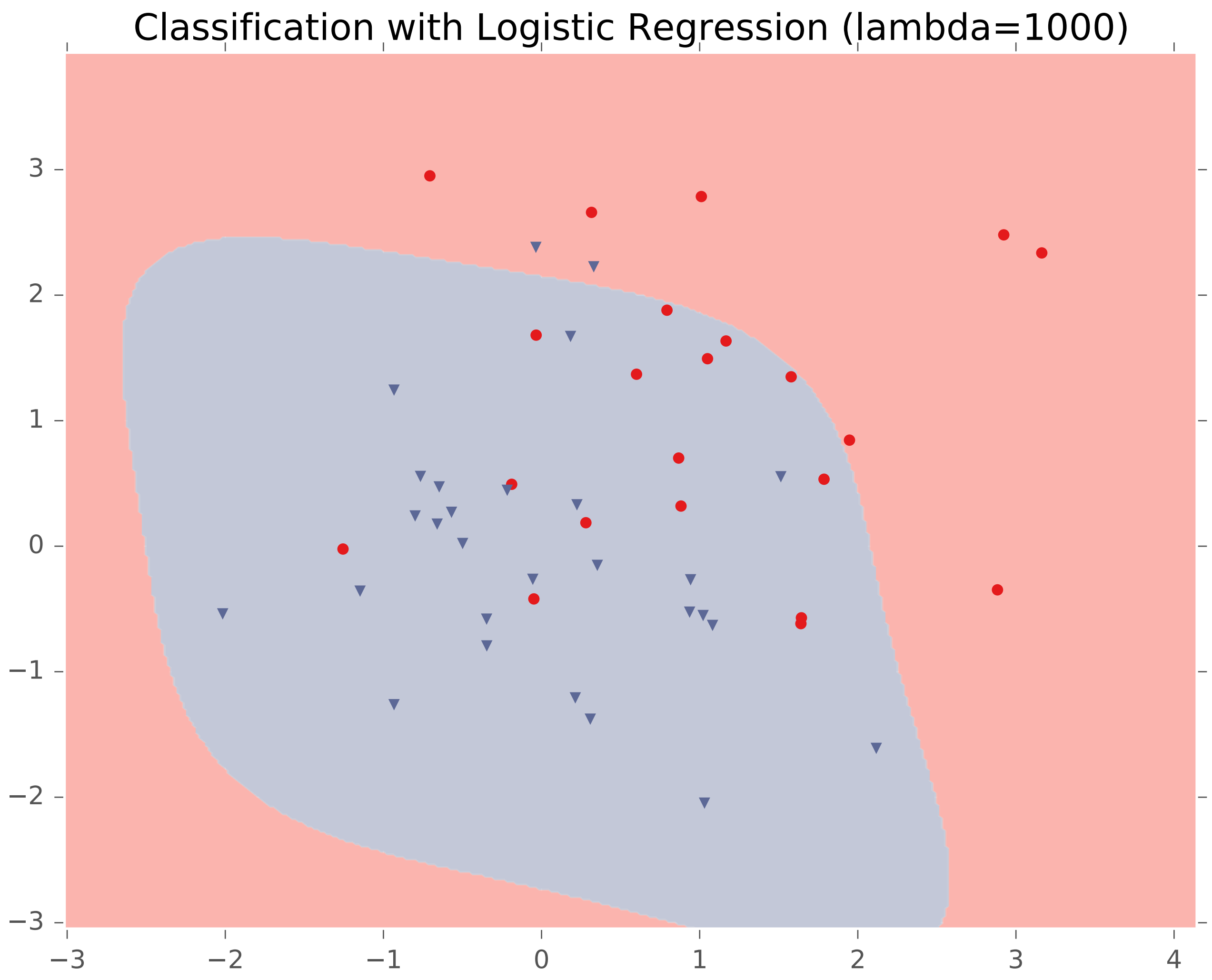
Example: Logistic Regression



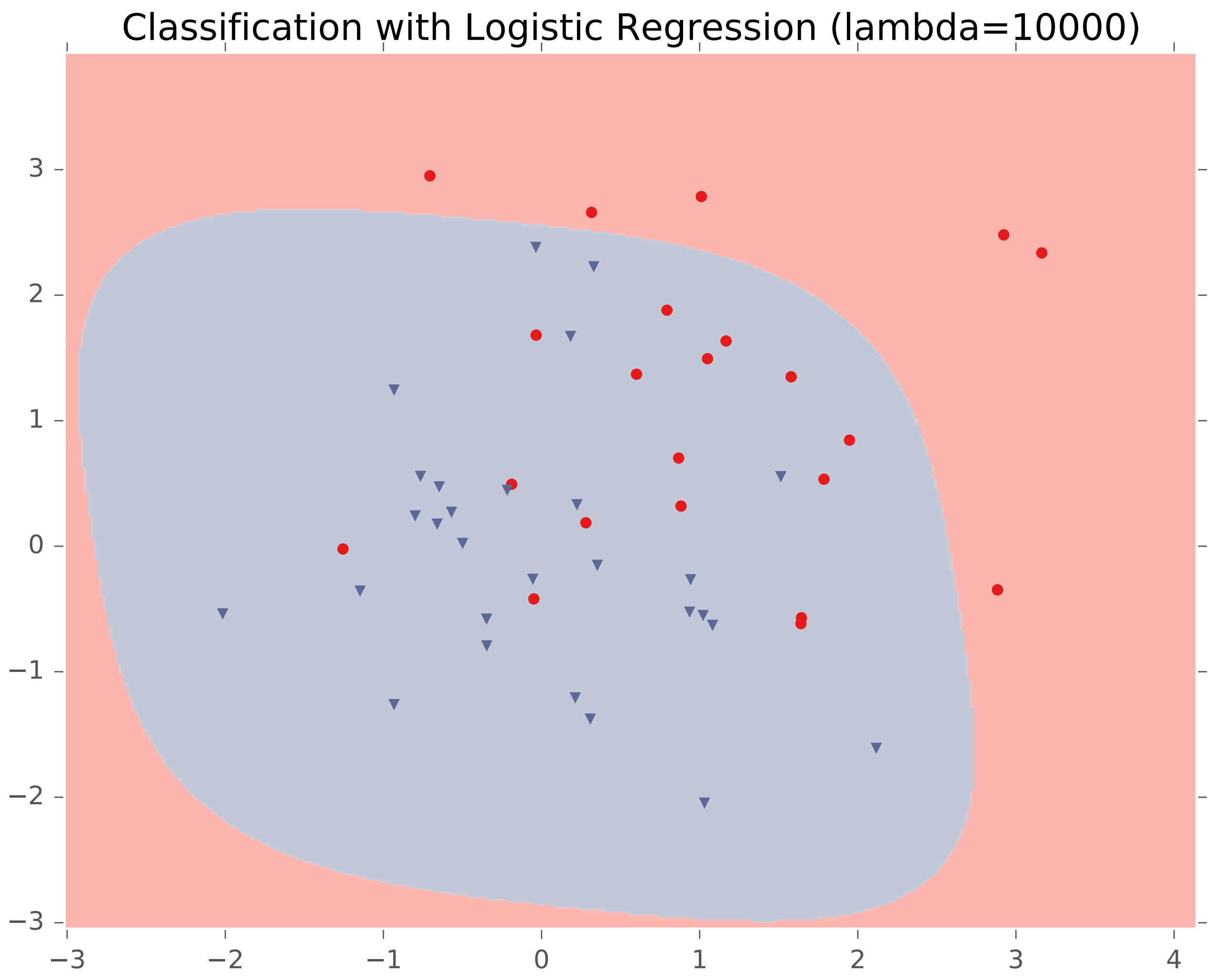
Example: Logistic Regression



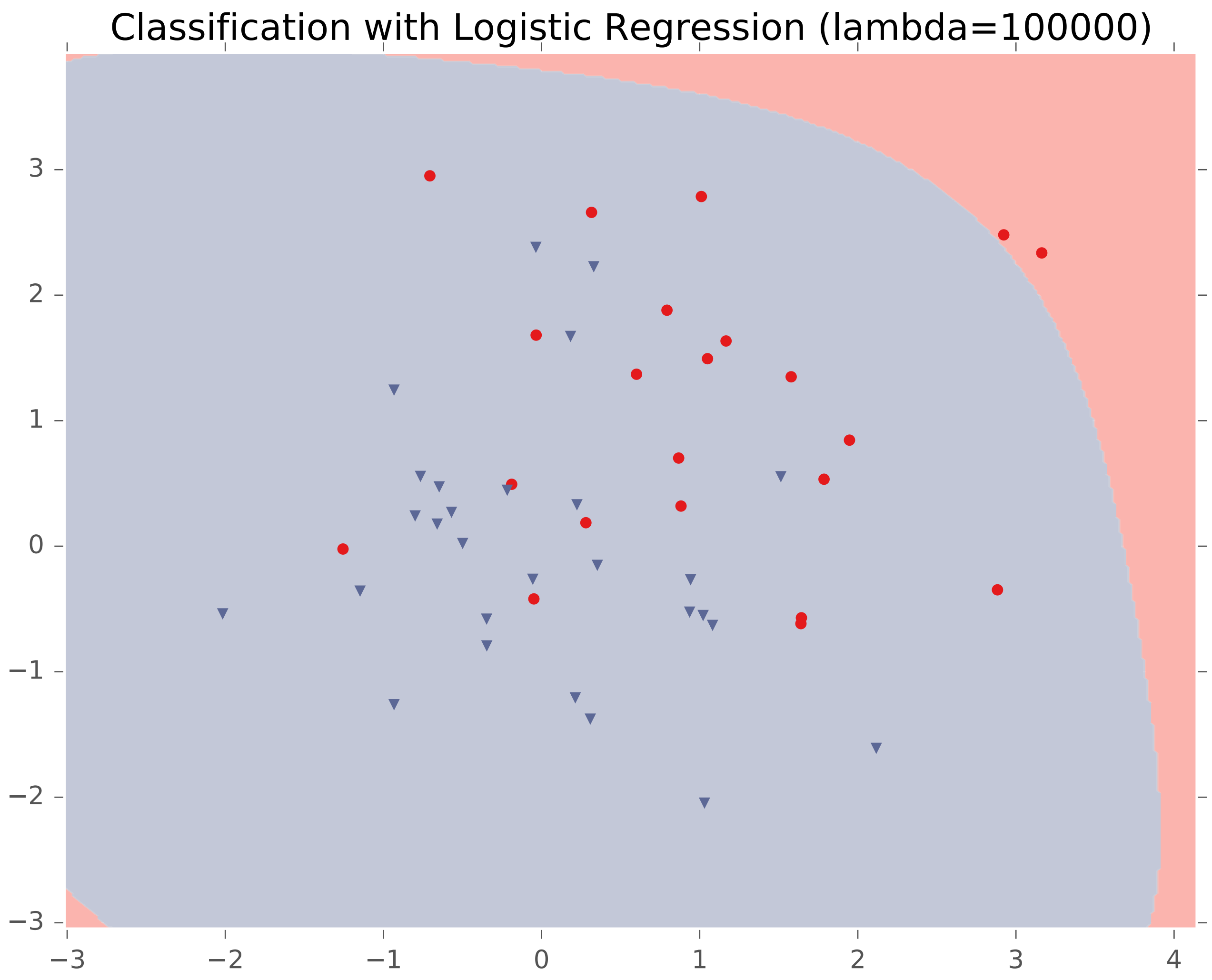
Example: Logistic Regression



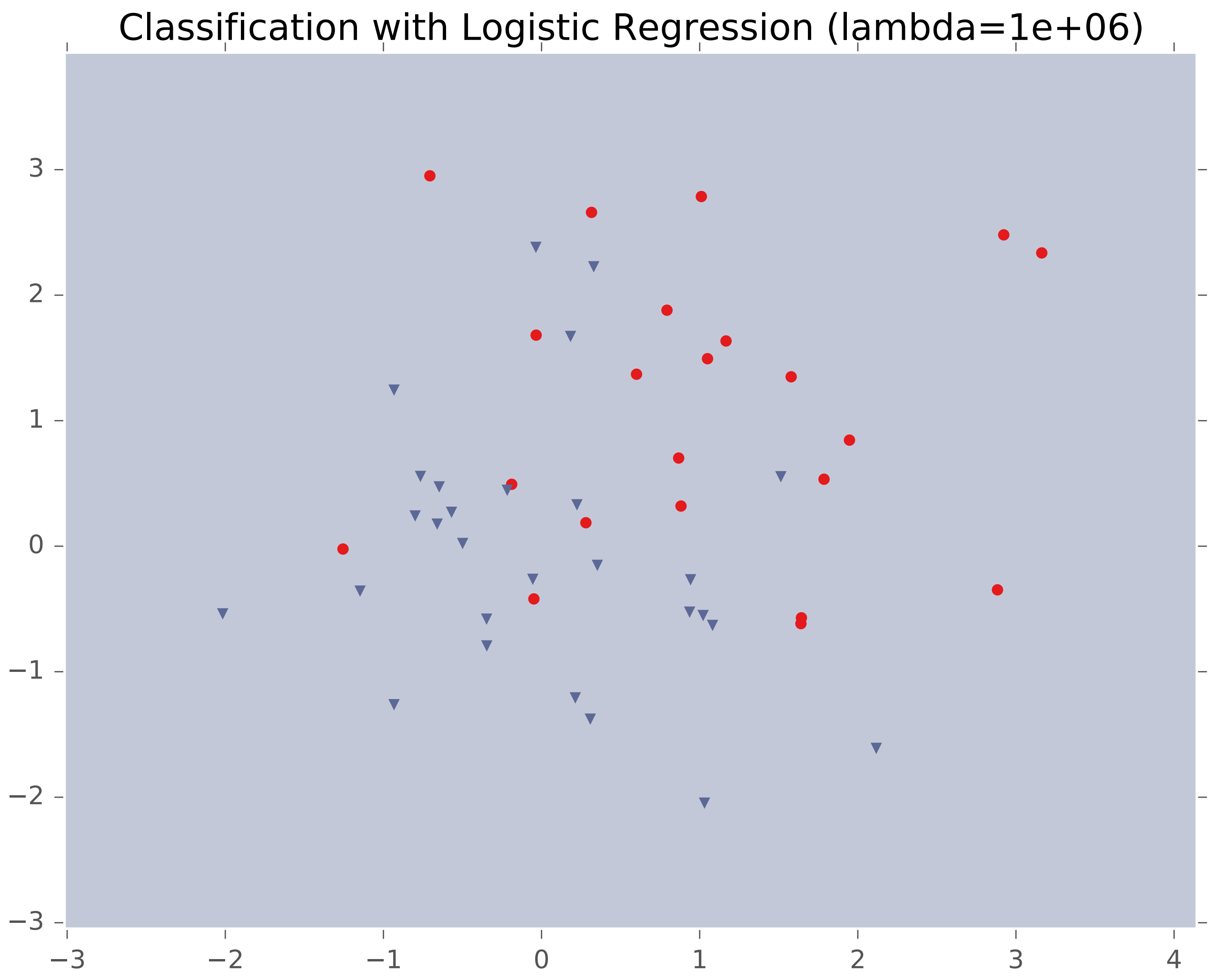
Example: Logistic Regression



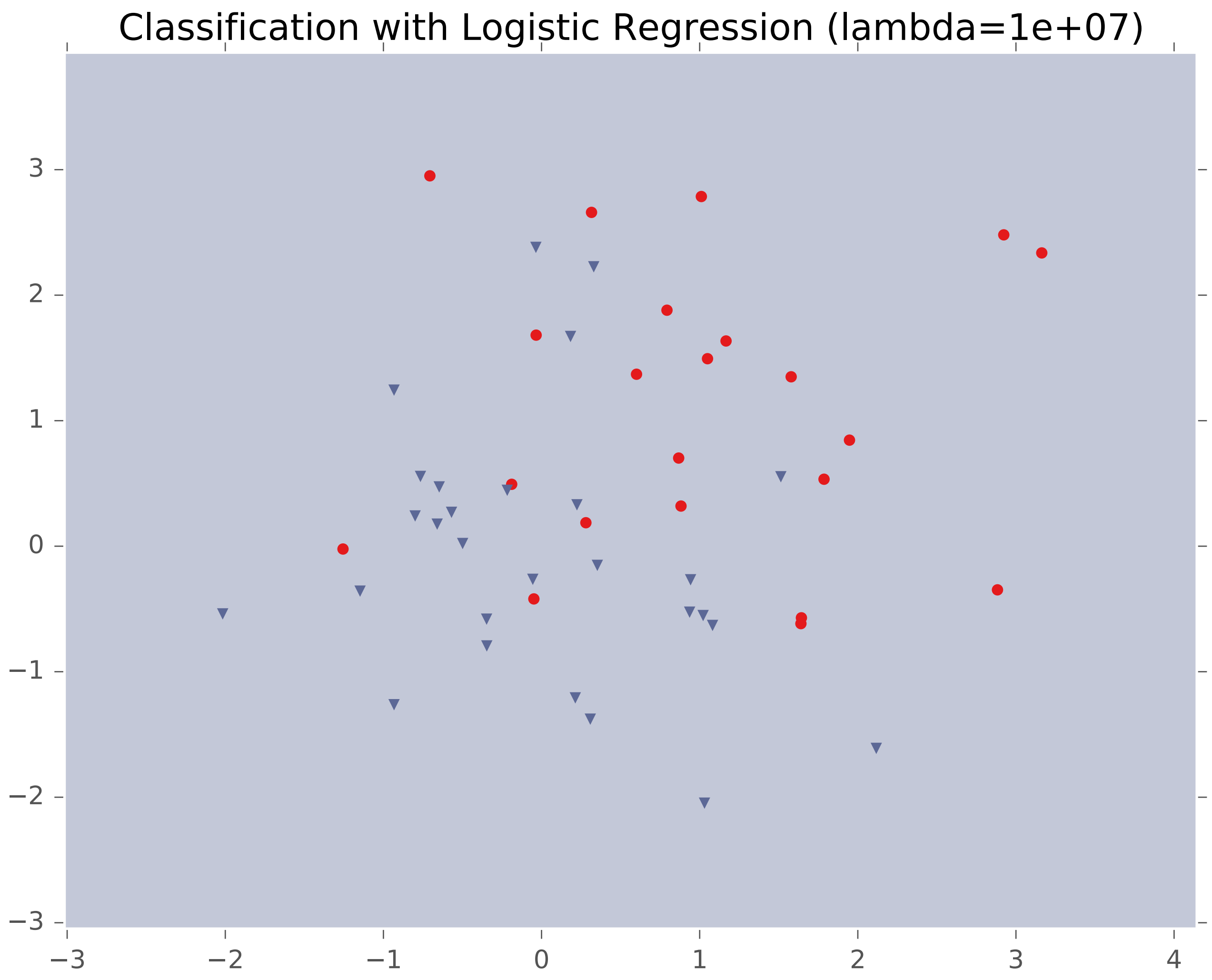
Example: Logistic Regression



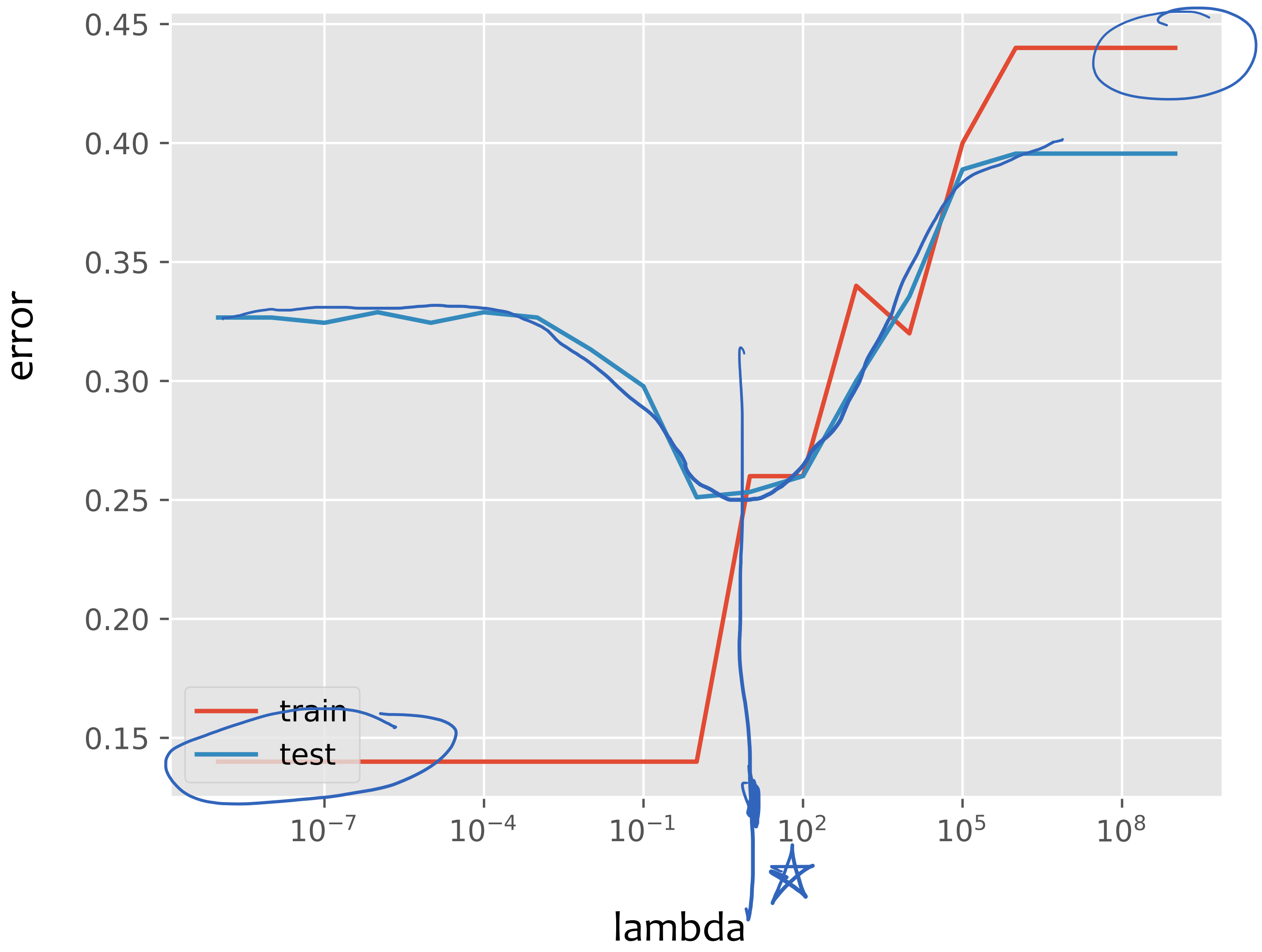
Example: Logistic Regression



Example: Logistic Regression



Example: Logistic Regression



OPTIMIZATION FOR L1 REGULARIZATION

Optimization for L1 Regularization

Can we apply SGD to the LASSO learning problem?

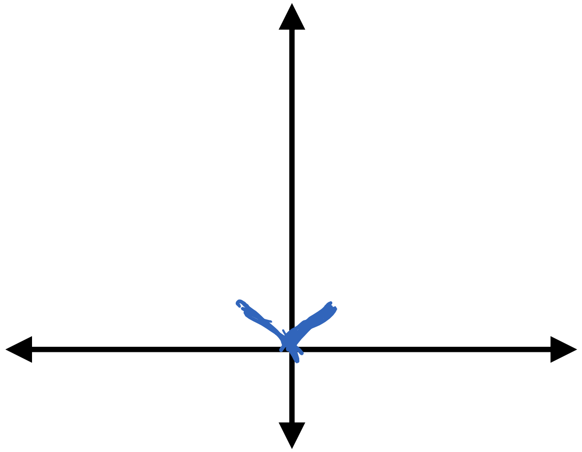
$$\underset{\boldsymbol{\theta}}{\text{argmin}} \; J_{\text{LASSO}}(\boldsymbol{\theta})$$

$$\begin{aligned} J_{\text{LASSO}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_1} \\ &= \frac{1}{2} \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{k=1}^{K} |\theta_k|} \end{aligned}$$

Optimization for L1 Regularization

- Consider the absolute value function:

$$r(\boldsymbol{\theta}) = \lambda \sum_{k=1}^{K} |\theta_k|$$



- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)

Def: A vector $g \in \mathbb{R}^M$ is called a **subgradient** of a function $f(\mathbf{x}) : \mathbb{R}^M \rightarrow \mathbb{R}$ at the point $\mathbf{x}$ if, for all $\mathbf{x}' \in \mathbb{R}^M$, we have:

$$f(\mathbf{x}') \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{x}' - \mathbf{x})$$

Optimization for L1 Regularization

- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)
- An array of optimization algorithms exist to handle this issue:
  - Subgradient descent
  - Stochastic subgradient descent
  - Coordinate Descent
  - Othant-Wise Limited memory Quasi-Newton (OWL-QN) (Andrew & Gao, 2007) and provably convergent variants
  - Block coordinate Descent (Tseng & Yun, 2009)
  - Sparse Reconstruction by Separable Approximation (SpaRSA) (Wright et al., 2009)
  - Fast Iterative Shrinkage Thresholding Algorithm (FISTA) (Beck & Teboulle, 2009)

Basically the same as GD and SGD, but you use one of the subgradients when necessary

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