10-301/10-601 Introduction to Machine Learning
Machine Learning Department
School of Computer Science
Carnegie Mellon University

Logistic Regression
+ Feature Engineering
+ Regularization

Geoff Gordon
Lecture 10
w/thanks to Matt Gormley & Henry Chai

LOGISTIC REGRESSION

Logistic Regression

1. Model

$$p(y \mid x, \boldsymbol{\theta}) = \begin{cases} \sigma(\boldsymbol{\theta}^\top \mathbf{x}) & \text{if } y = 1 \\ 1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x}) & \text{if } y = 0 \end{cases}$$

↑
Bernoulli
distribution
(coin flip)

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

2. Objective

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

$$J(\boldsymbol{\theta}) = -\frac{1}{N}\ell(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{-\log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})}_{J^{(i)}(\boldsymbol{\theta})}$$

Logistic Regression

3A. Derivatives

$$
\begin{aligned}
\frac{\partial J^{(i)}}{\partial \theta_m} &= \frac{\partial}{\partial \theta_m}(-\log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})) \\
&= \begin{cases} \dfrac{\partial}{\partial \theta_m}\left[-\log \sigma(\boldsymbol{\theta}^\top \mathbf{x})\right] & \text{if } y^{(i)} = 1 \\ \dfrac{\partial}{\partial \theta_m}\left[-\log(1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x}))\right] & \text{if } y^{(i)} = 0 \end{cases} \\
&= \dots \\
&= -\,(y^{(i)} - \sigma(\boldsymbol{\theta}^\top \mathbf{x}^{(i)}))\mathbf{x}_m^{(i)}
\end{aligned}
$$

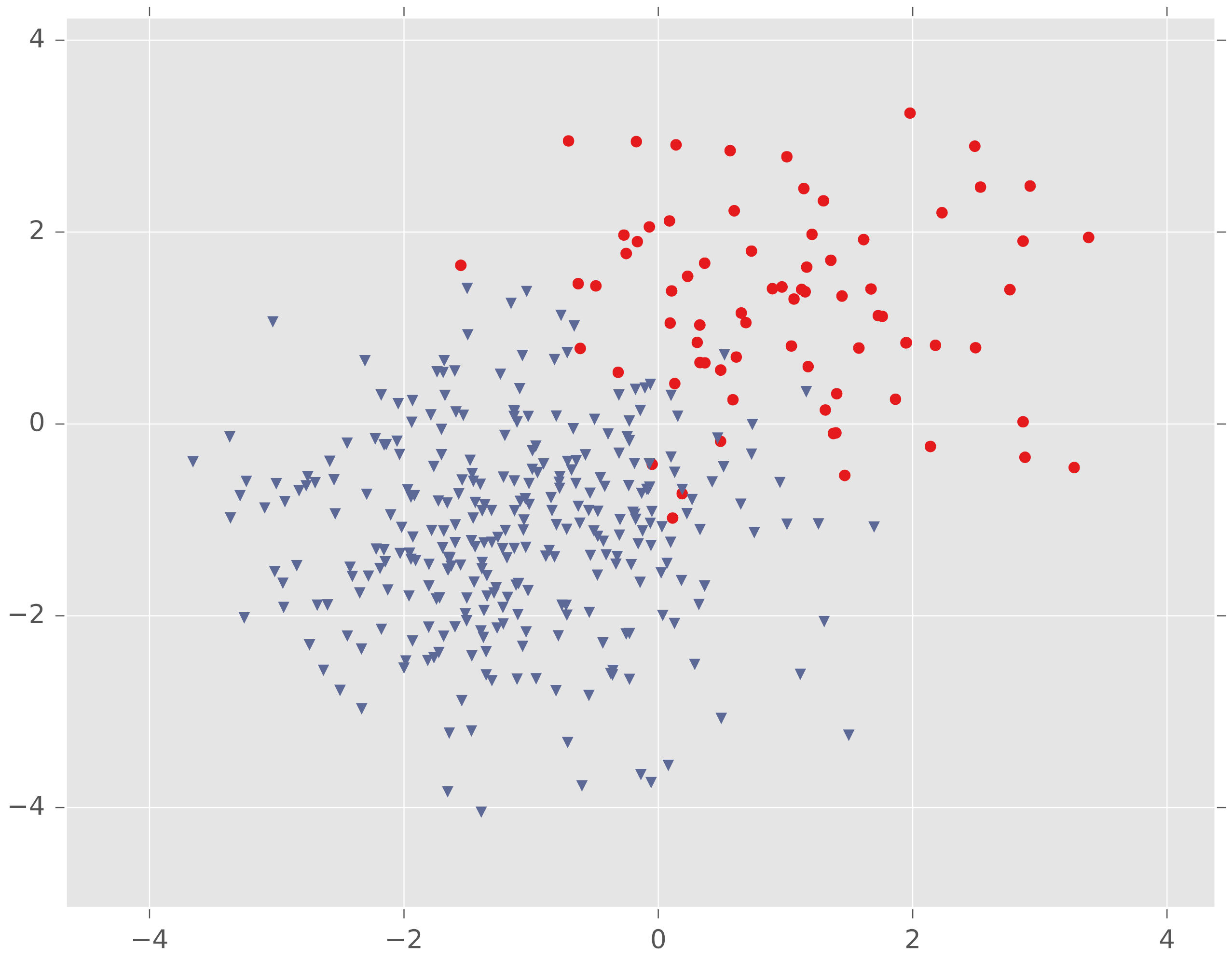
3B. Gradients

Logistic Regression

4. Optimization

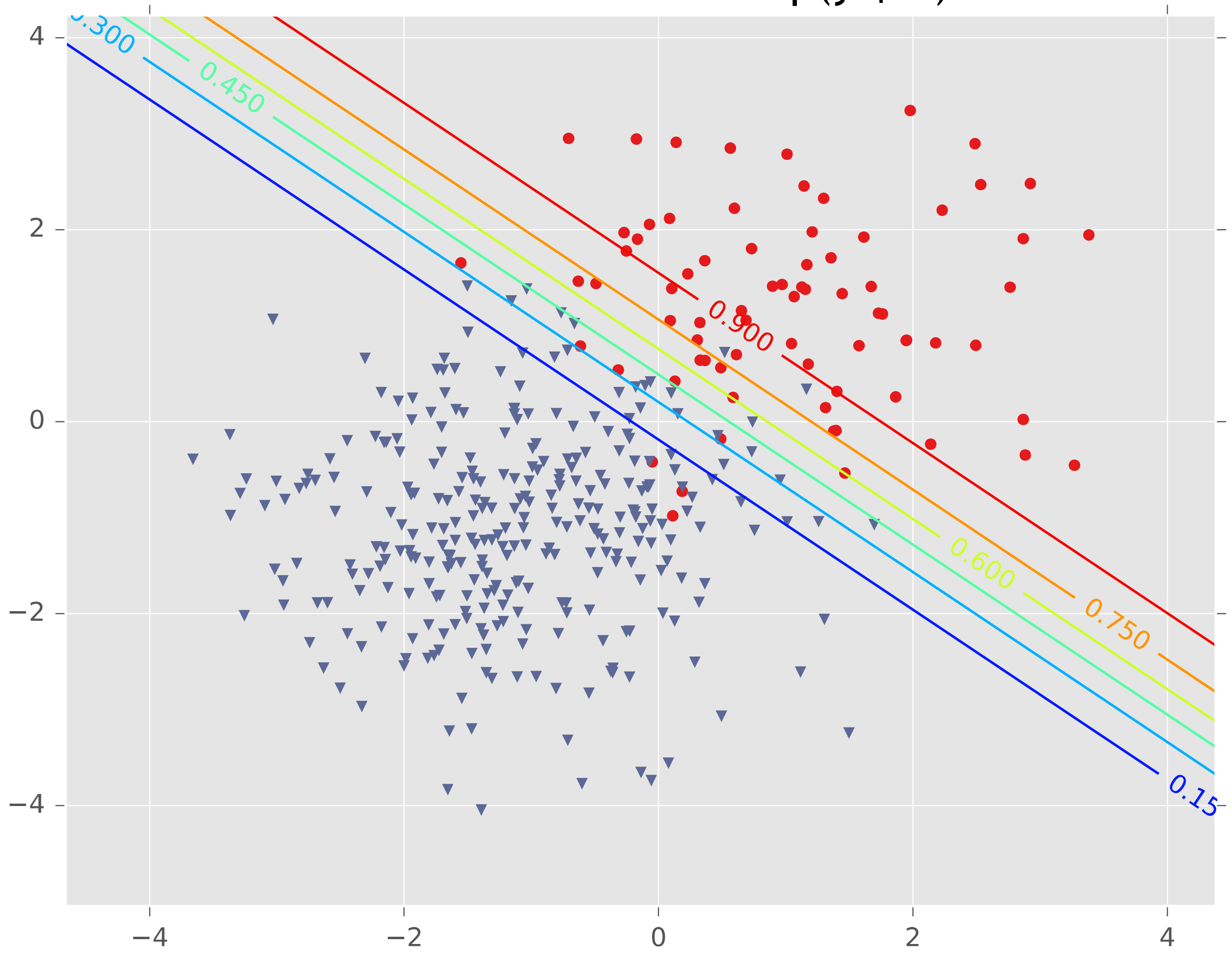
5. Prediction

Logistic Regression

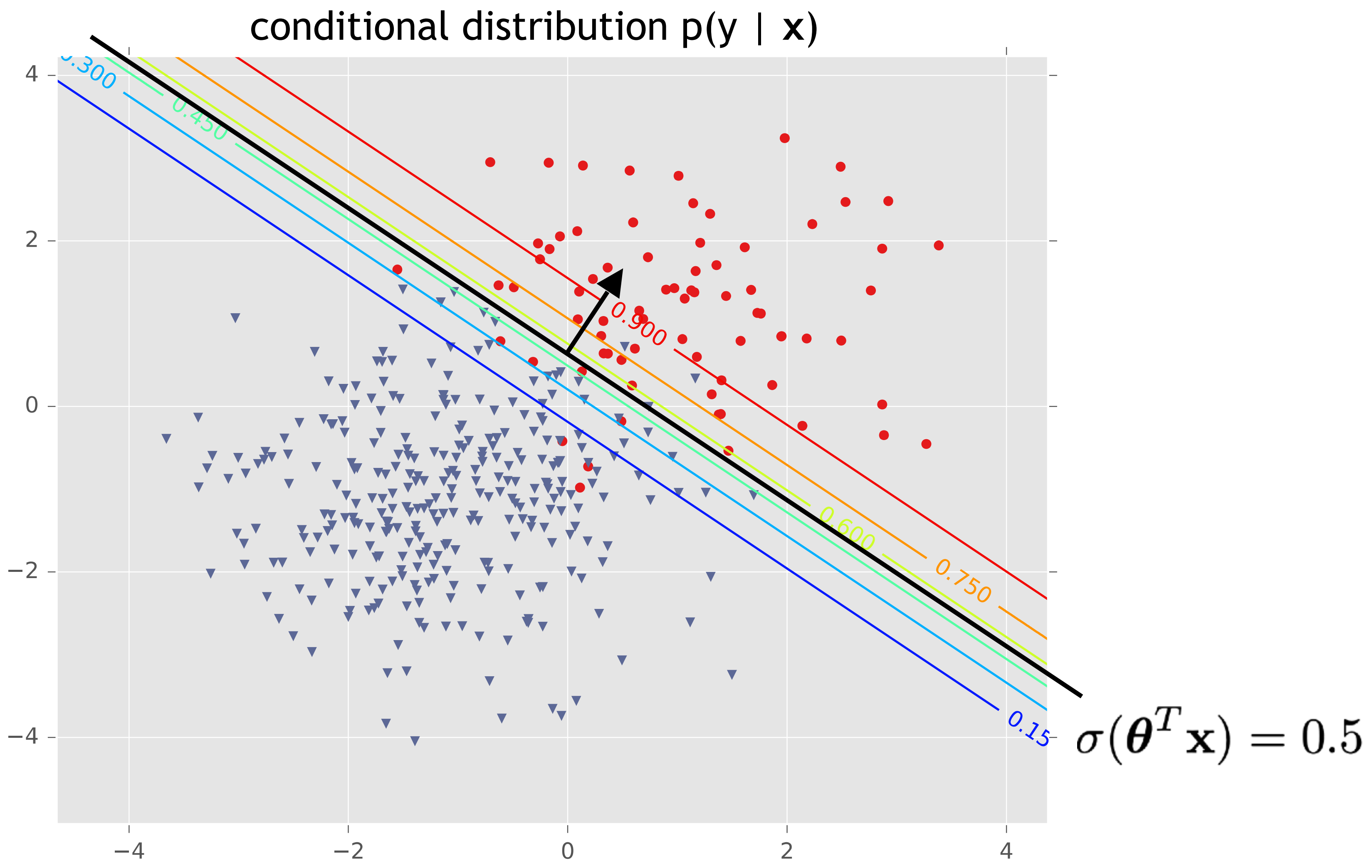


Logistic Regression

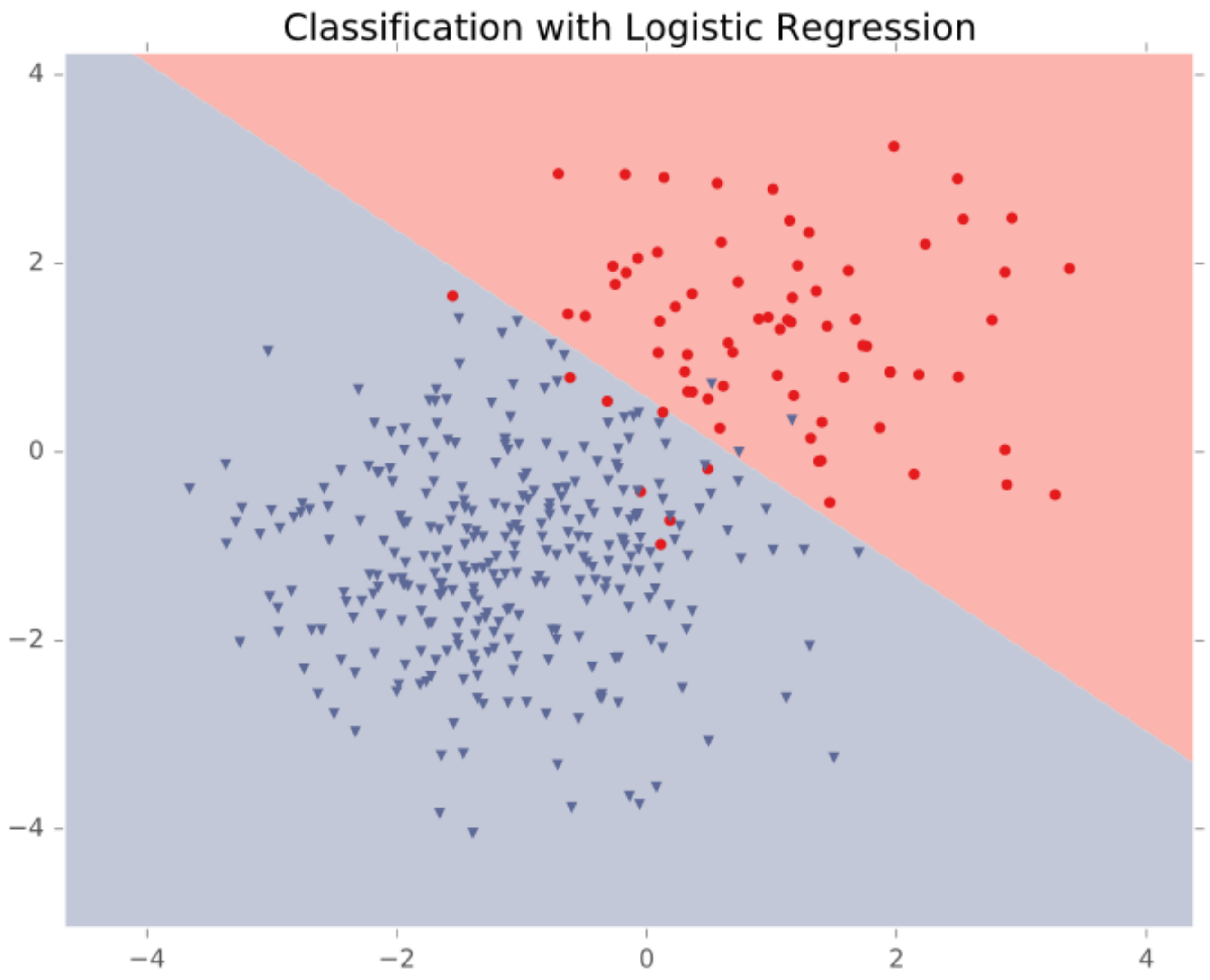
conditional distribution p(y | **x**)



Logistic Regression



Logistic Regression



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/

IM GENET

SEARCH

14,197,122 images, 21841 synsets indexed

Home | Explore
About | Download

Not logged in. Login I Signup

Bird

Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings

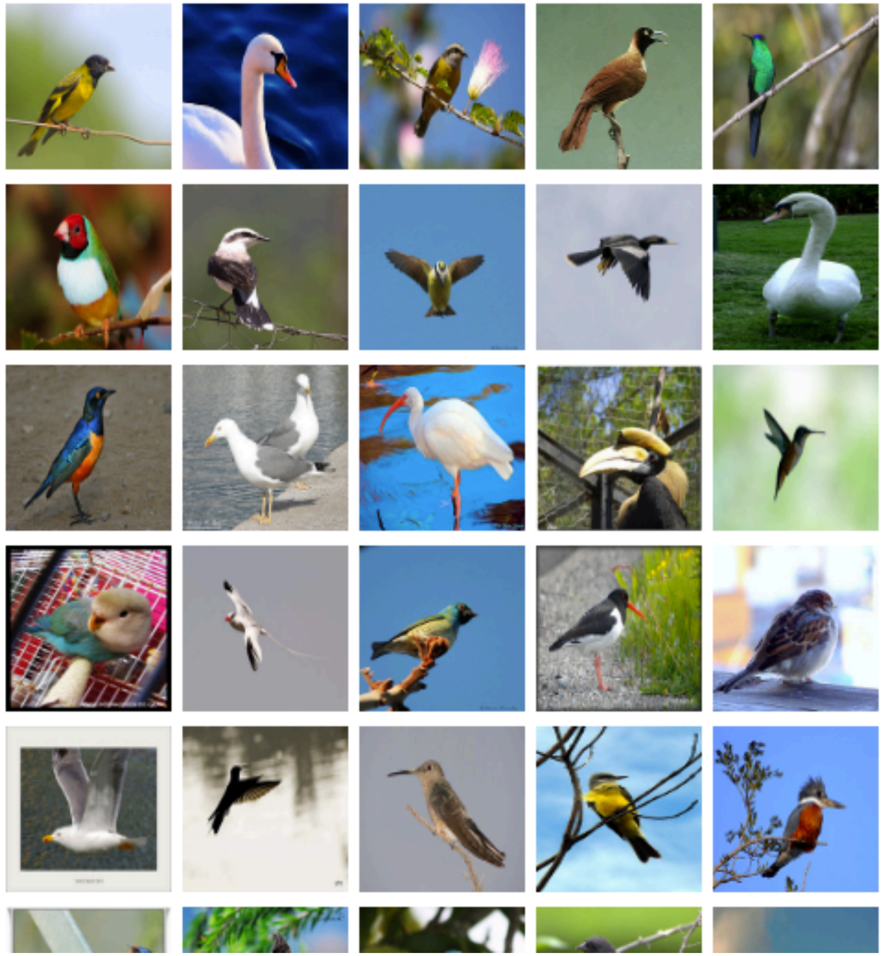
2126 pictures

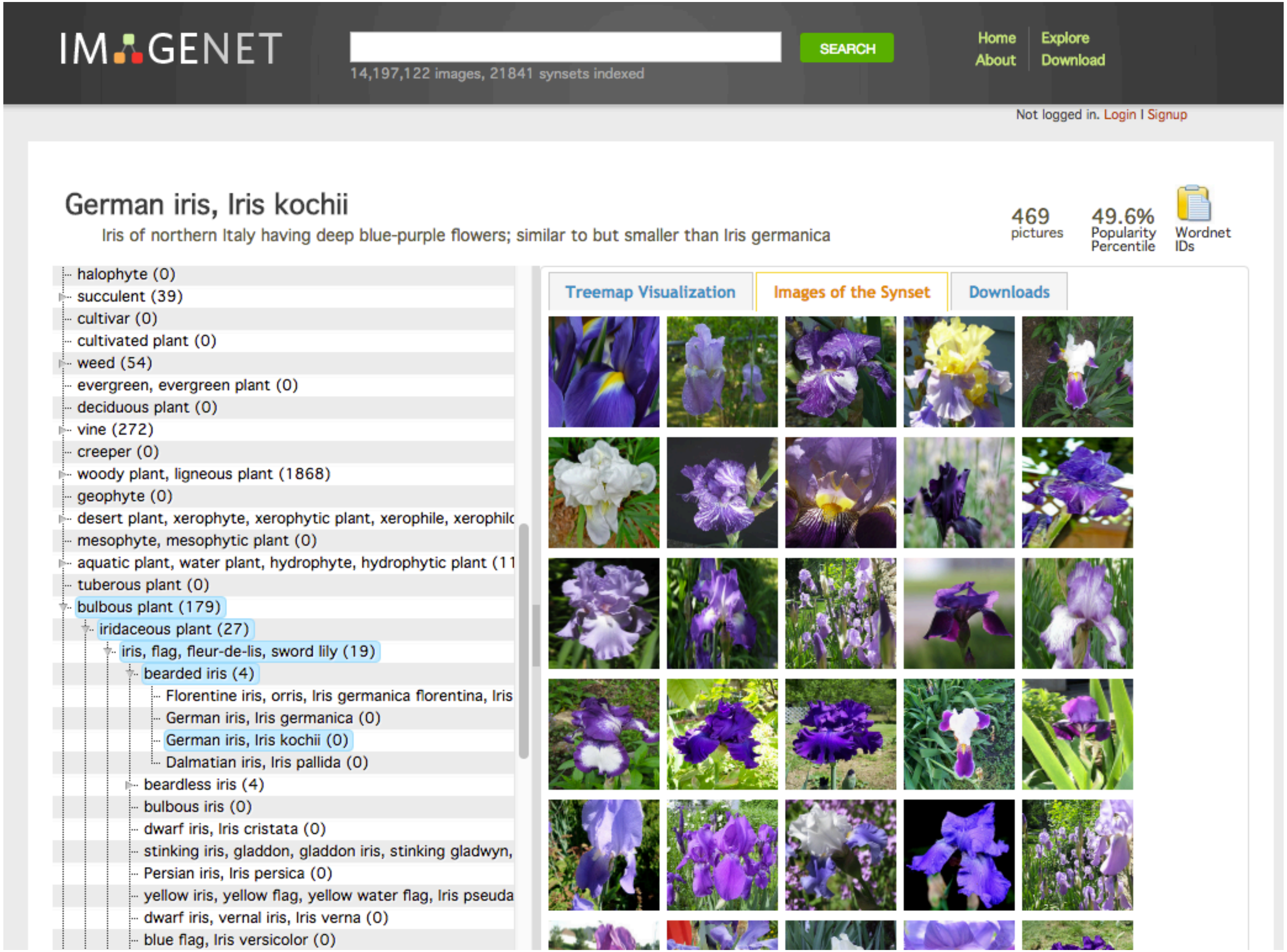
92.85% Popularity Percentile

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 I 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 | Explore
About | Download

Not logged in. Login I Signup

Court, courtyard

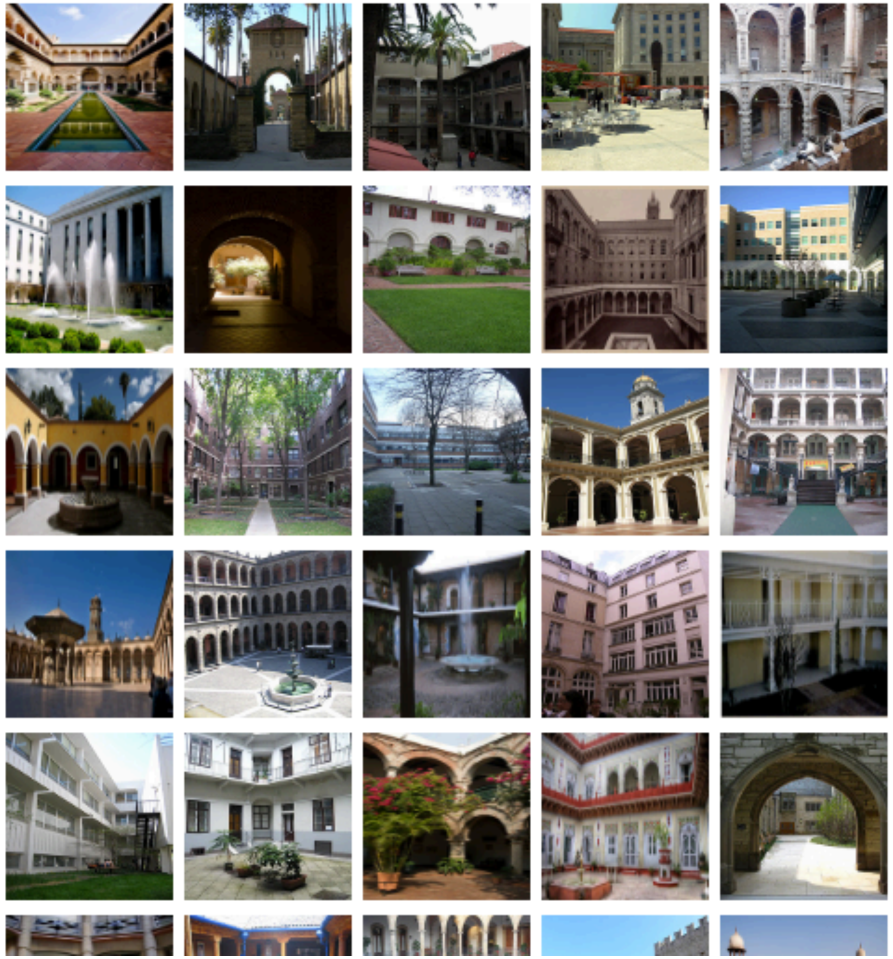
An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"

165 pictures | 92.61% Popularity Percentile | Wordnet IDs

Numbers in brackets: (the number of synsets in the subtree).

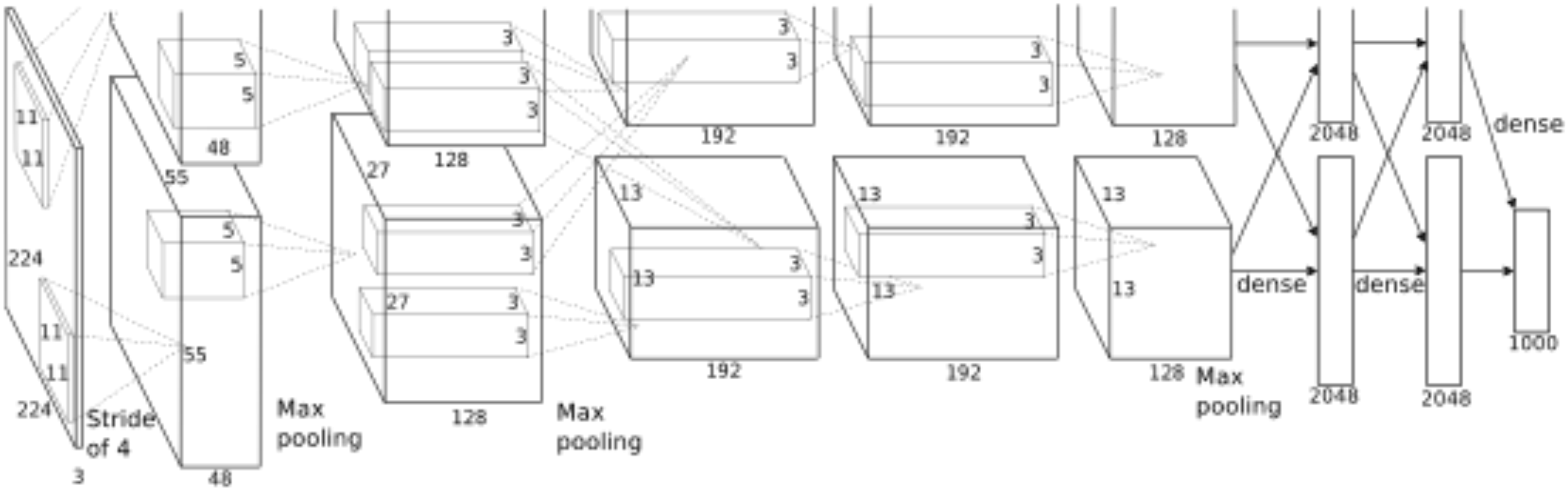
- ImageNet 2011 Fall Release (32326)
  - plant, flora, plant life (4486)
  - geological formation, formation (175)
  - natural object (1112)
  - sport, athletics (176)
  - artifact, artefact (10504)
    - instrumentality, instrumentation (5494)
    - structure, construction (1405)
      - airdock, hangar, 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

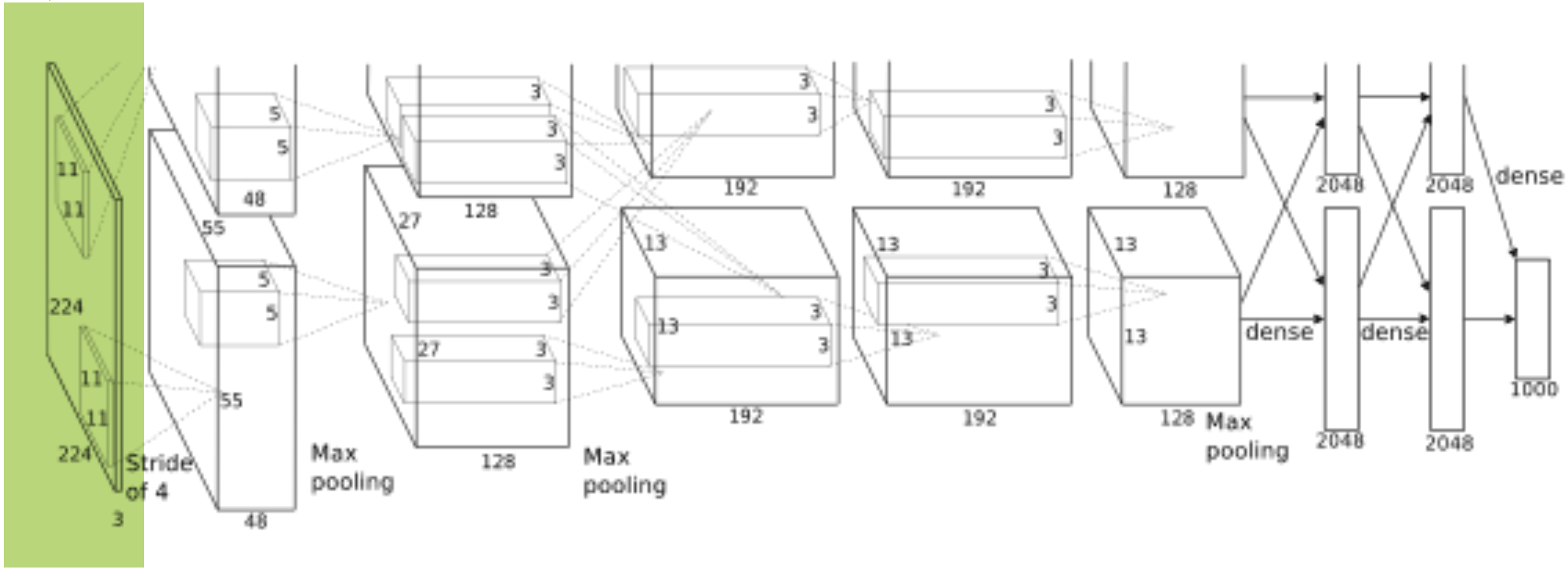
CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2011)
17.5% error on ImageNet LSVRC-2010 contest



Example: Image Classification

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

Input image (pixels)

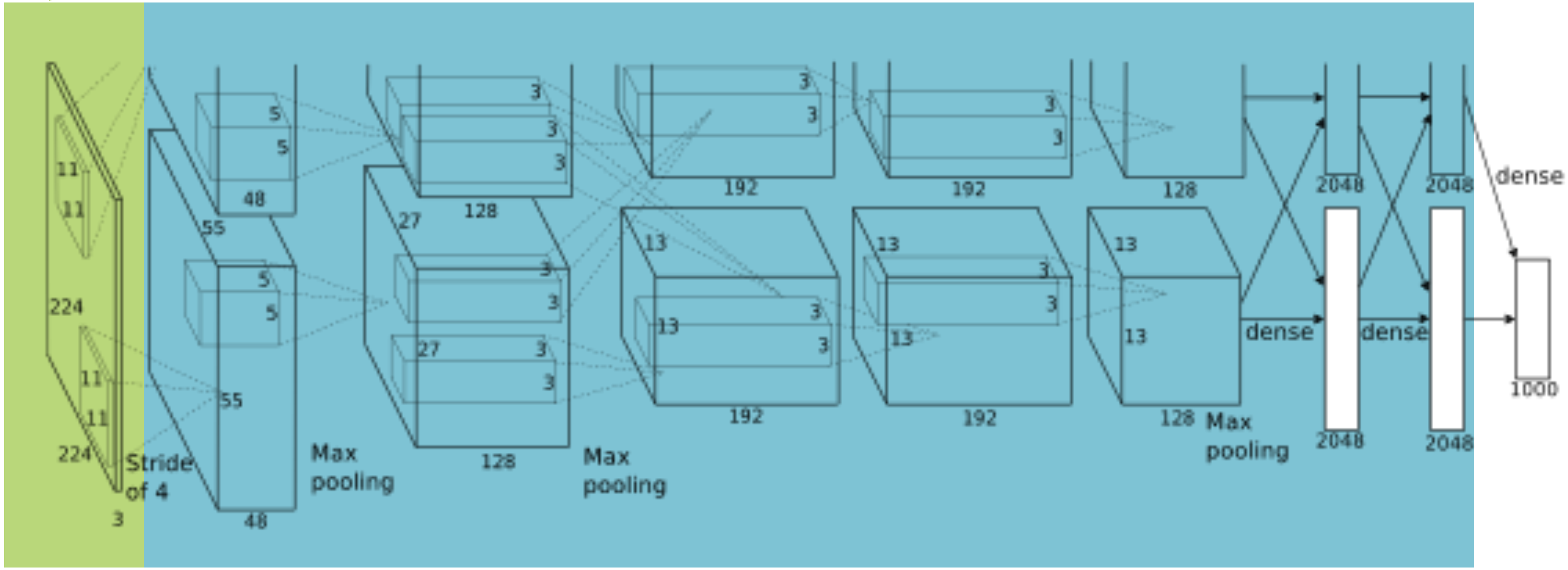


Example: Image Classification

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



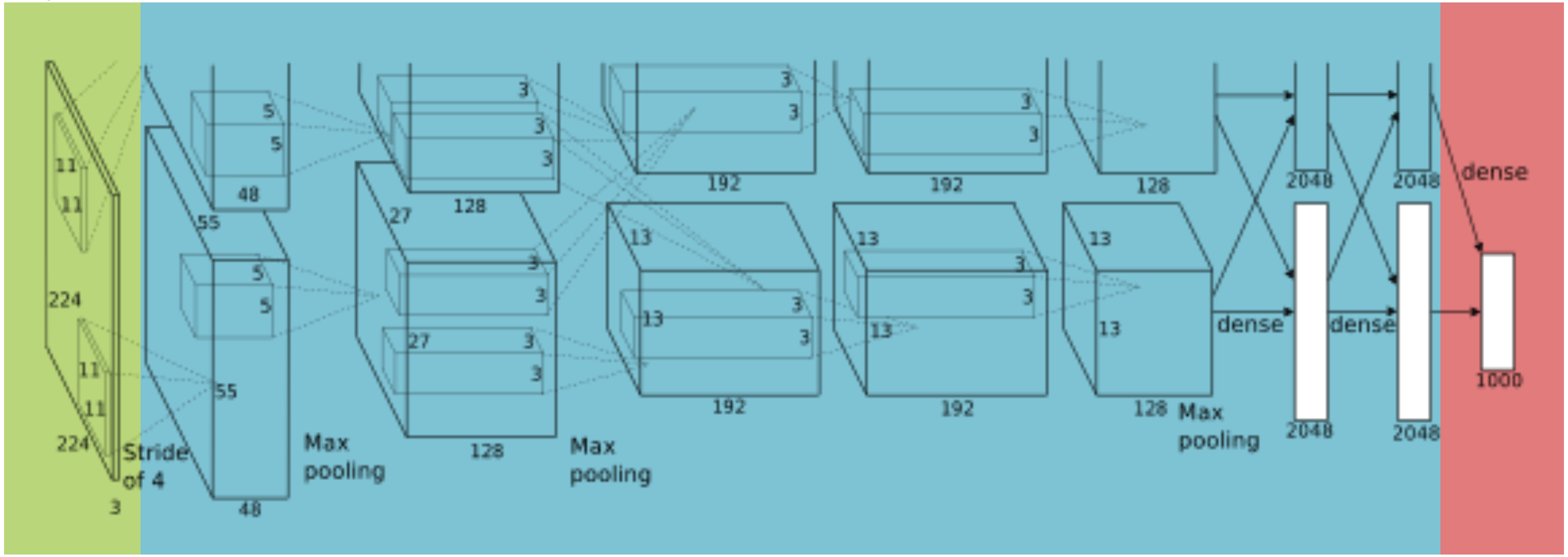
Example: Image Classification

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



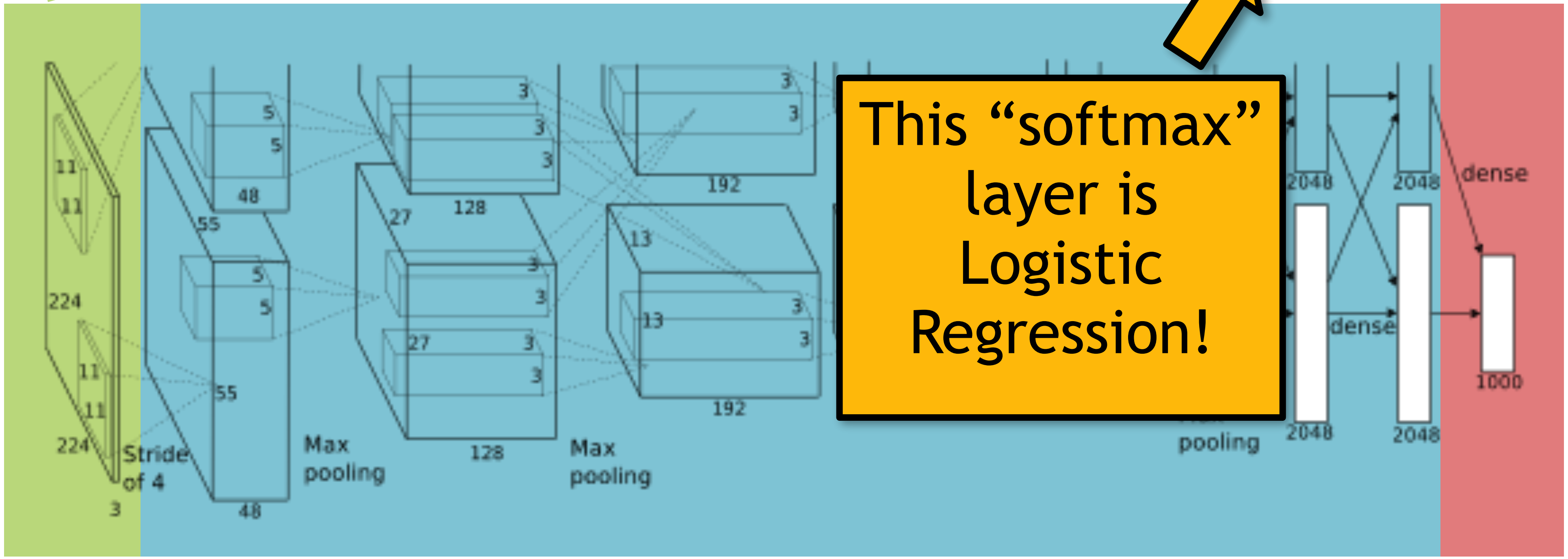
Example: Image Classification

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



This "softmax" layer is Logistic Regression!

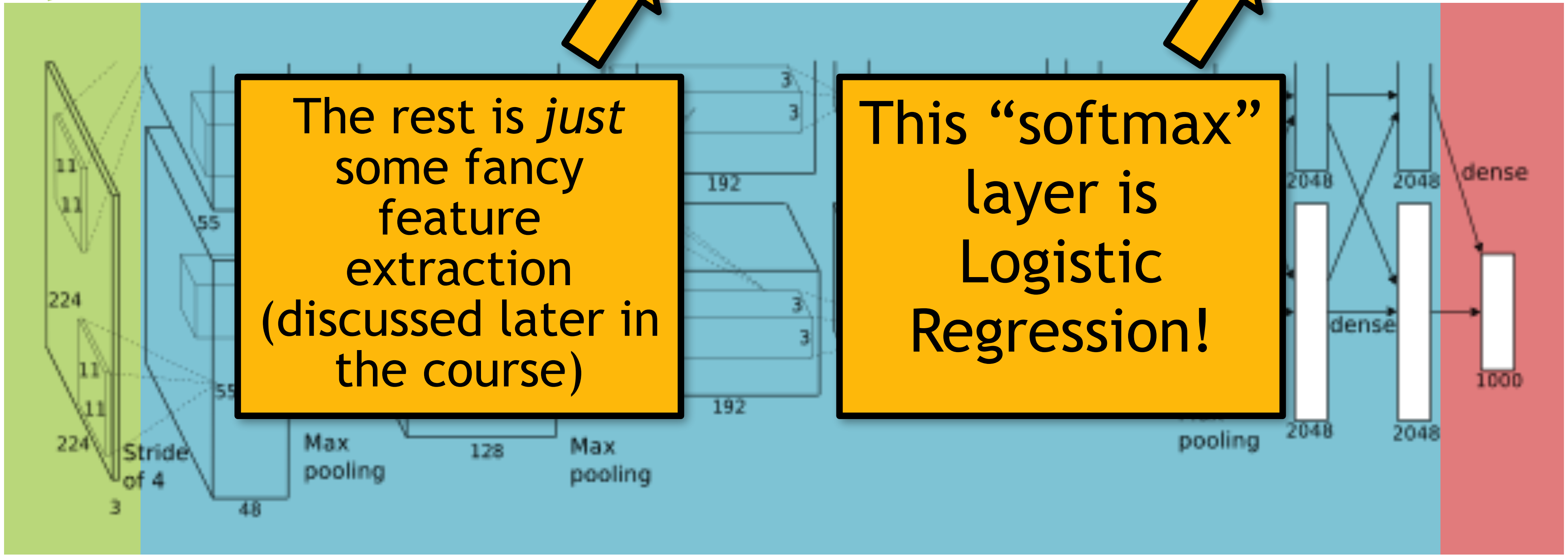
Example: Image Classification

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



The rest is *just* some fancy feature extraction (discussed later in the course)

This “softmax” layer is Logistic Regression!

Logistic Regression 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 classification
- Prove that the decision boundary of binary logistic regression is linear

BAYES OPTIMAL CLASSIFIER

Bayes Optimal Classifier

Functi

Previou
was gen
target f

Our goa
that be

Suppose you knew the distribution $p^*(y \mid \mathbf{x})$ or had a good approximation to it.

Question:
How would you design a function $y = h(\mathbf{x})$ 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 $p(y|\mathbf{x})$ that best approximates $p^*(y|\mathbf{x})$

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.

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 B randomly chosen examples

Minibatch SGD

procedure minibatch_sgd($\mathscr{D}, \boldsymbol{\theta}^{(0)}$)

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

while not converged **do**

$I \leftarrow$ sample$(B, \{1\ldots N\})$

$\mathbf{g} \leftarrow \frac{1}{B}\sum_{i \in I} \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$

return $\boldsymbol{\theta}$

Minibatch SGD

procedure minibatch_sgd($\mathscr{D}, \boldsymbol{\theta}^{(0)}$)

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

while not converged **do**

$I \leftarrow$ sample$(B, \{1 \ldots N\})$ $\leftarrow$ typically w/o replacement

$\mathbf{g} \leftarrow \frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$

return $\boldsymbol{\theta}$

Minibatch SGD

procedure minibatch_sgd($\mathscr{D}, \boldsymbol{\theta}^{(0)}$)

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

while not converged **do**

$I \leftarrow$ sample$(B, \{1 \ldots N\})$ ← typically w/o replacement

$\mathbf{g} \leftarrow \frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$

return $\boldsymbol{\theta}$

how should we choose B?

Comparing the variants

procedure minibatch_sgd($\mathscr{D}, \boldsymbol{\theta}^{(0)}$)

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

while not converged **do**

$I \leftarrow$ sample$(B, \{1 \ldots N\})$ $\triangleright$ w/o replacement

$\mathbf{g} \leftarrow \frac{1}{B} \sum_{i \in I} \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$

return $\boldsymbol{\theta}$

Gradient descent: $B = N$

SGD: $B = 1$

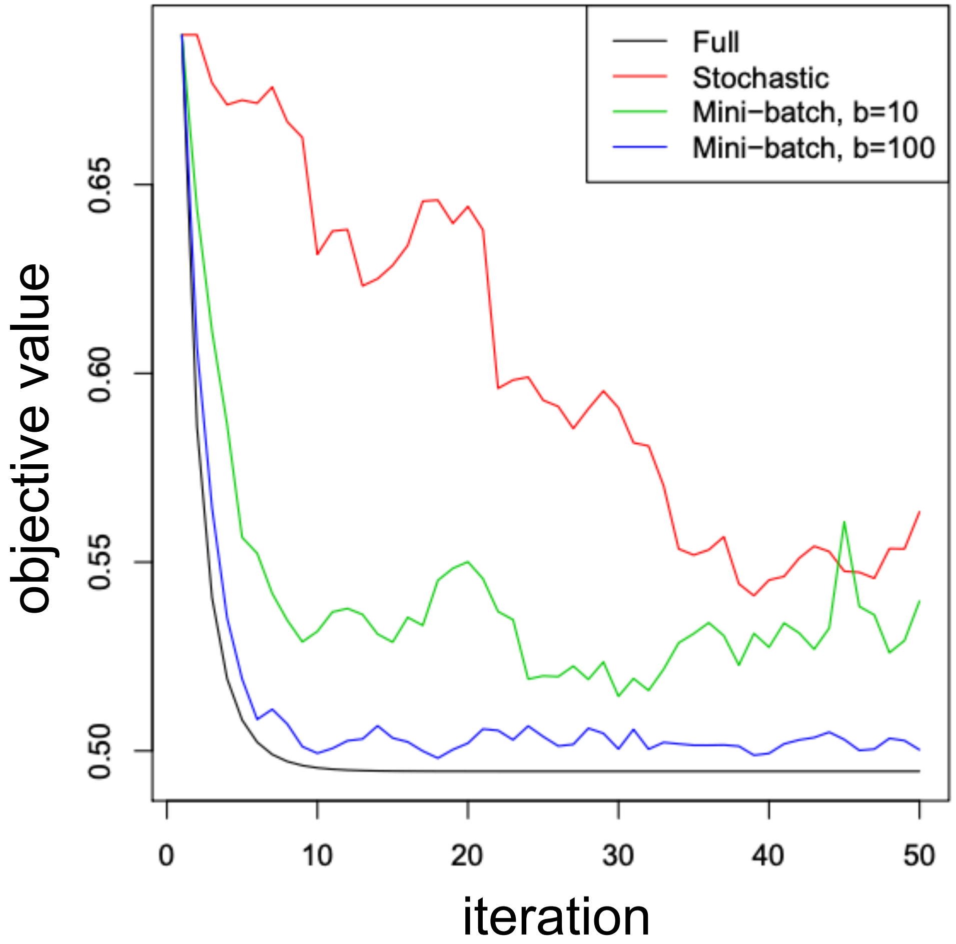
Minibatch SGD: $1 < B \ll N$

SGD vs. minbatch-SGD vs. GD

Compare number of iterations vs. objective value

- Logistic regression with 10k examples, 20 features
- Fixed step size

→ *Larger batches improve convergence speed (in iterations), make convergence more stable*



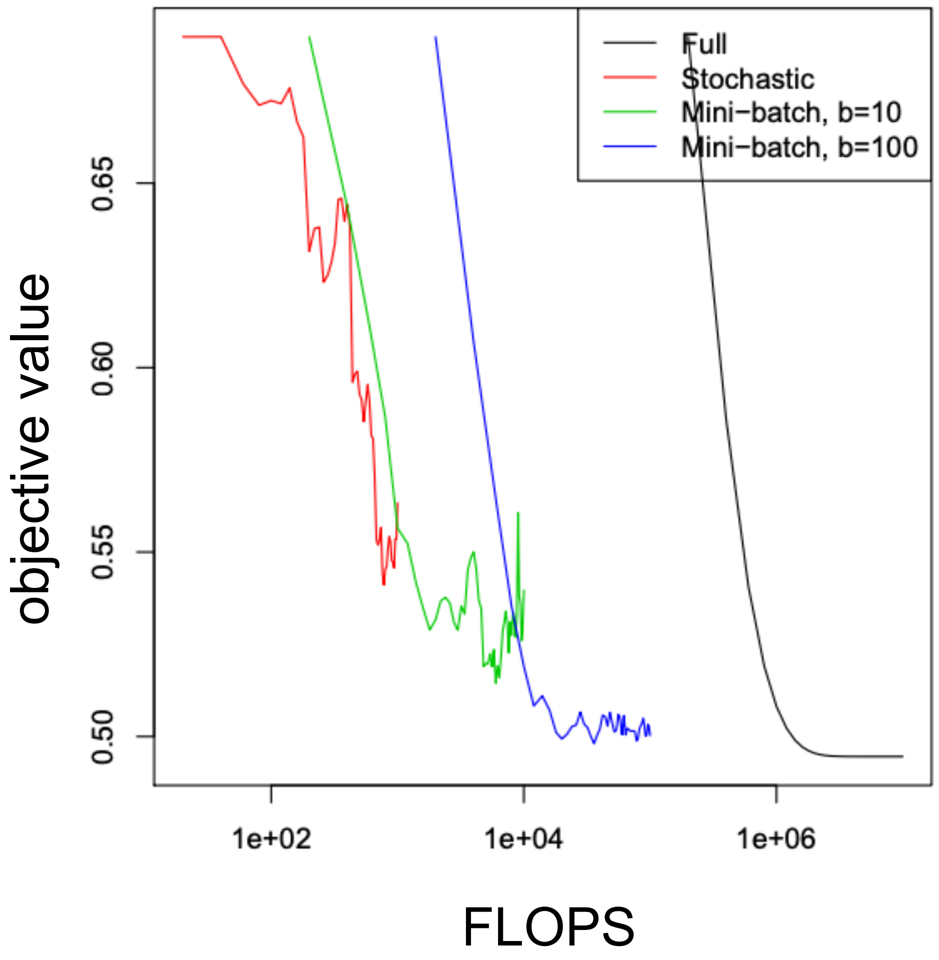
Credit: Ryan Tibshirani

SGD vs. minibatch-SGD vs. GD

Compare FLOPs vs. objective value

- FLOPs = floating point operations (compute cost)

→ *Although minibatch-SGD converges more slowly than GD terms of iterations, it may converge more quickly in terms of FLOPs*



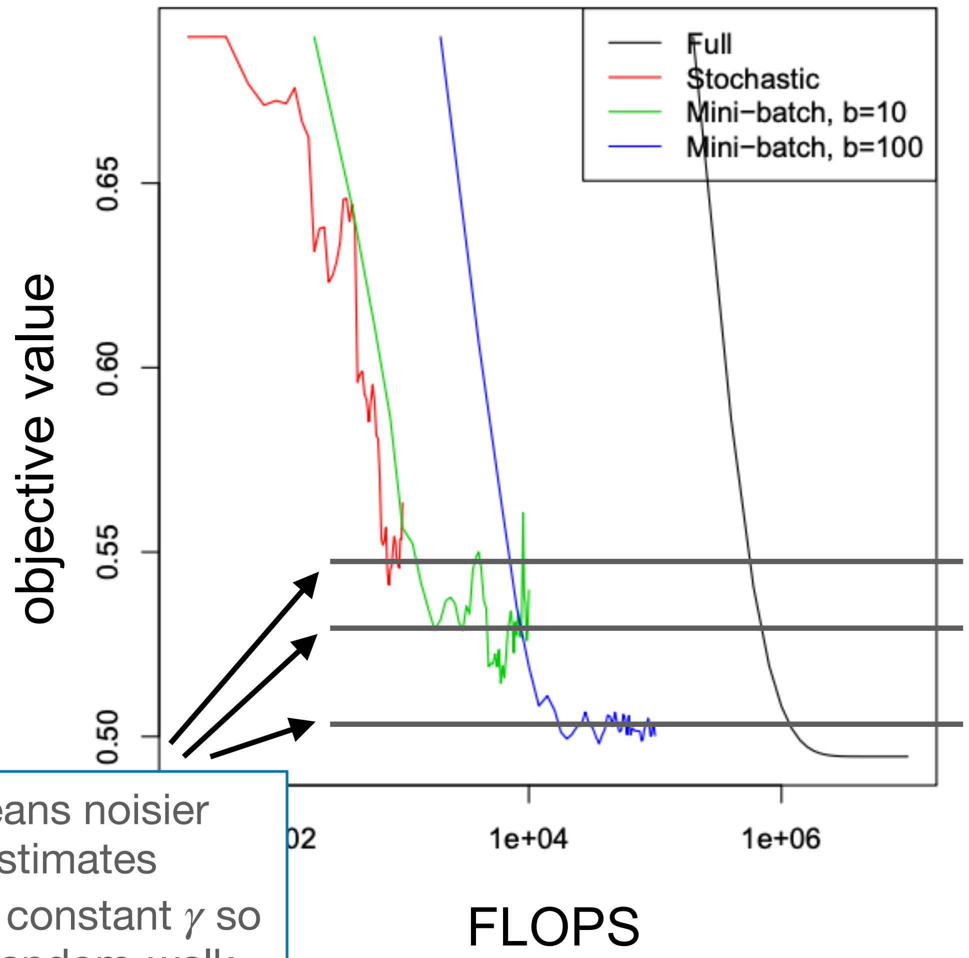
Credit: Ryan Tibshirani

SGD vs. minibatch-SGD vs. GD

Compare FLOPs vs. objective value

- FLOPs = floating point operations (compute cost)

→ *Although minibatch-SGD converges more slowly than GD terms of iterations, it may converge more quickly in terms of FLOPs*



Smaller b means noisier gradient estimates

We are keeping constant γ so we reach the random-walk stage at higher error

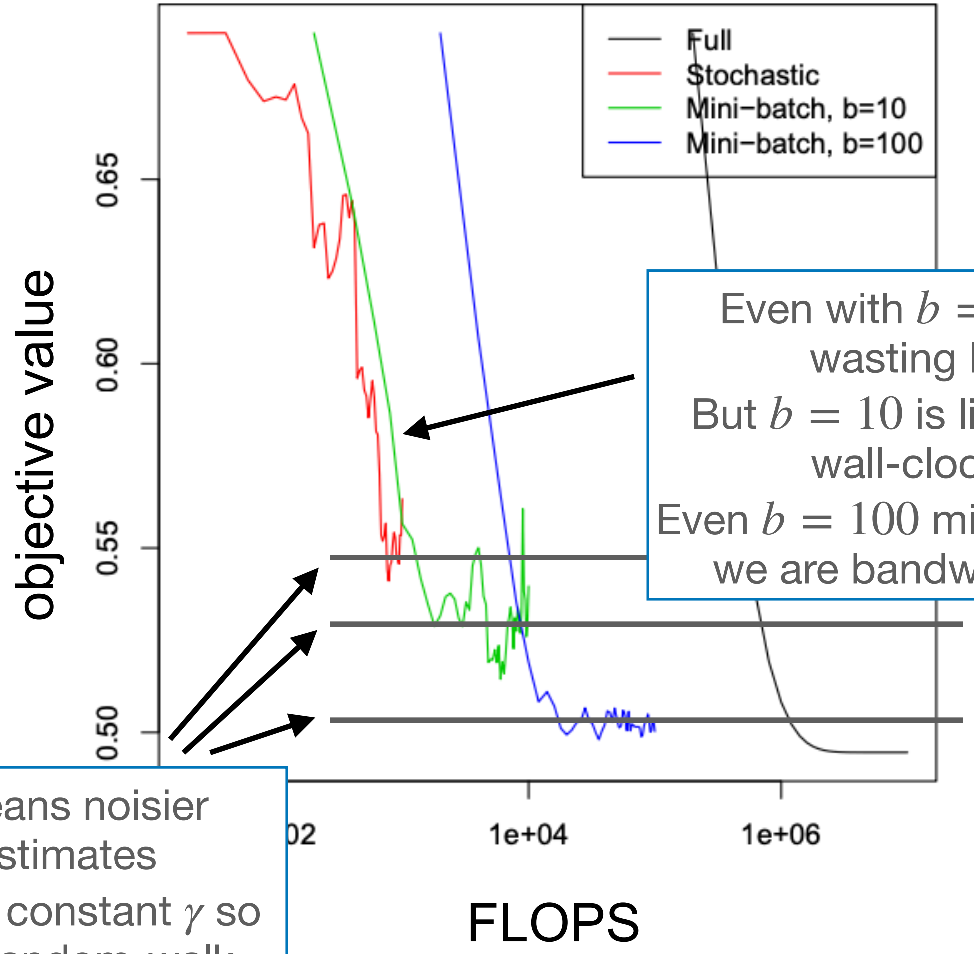
Credit: Ryan Tibshirani

SGD vs. minibatch-SGD vs. GD

Compare FLOPs vs. objective value

- FLOPs = floating point operations (compute cost)

→ *Although minibatch-SGD converges more slowly than GD terms of iterations, it may converge more quickly in terms of FLOPs*



Even with $b = 10$ we are wasting FLOPs

But $b = 10$ is likely faster in wall-clock time

Even $b = 100$ might be faster if we are bandwidth-limited

Smaller b means noisier gradient estimates

We are keeping constant γ so we reach the random-walk stage at higher error

Credit: Ryan Tibshirani

FEATURE ENGINEERING

Handcrafted Features

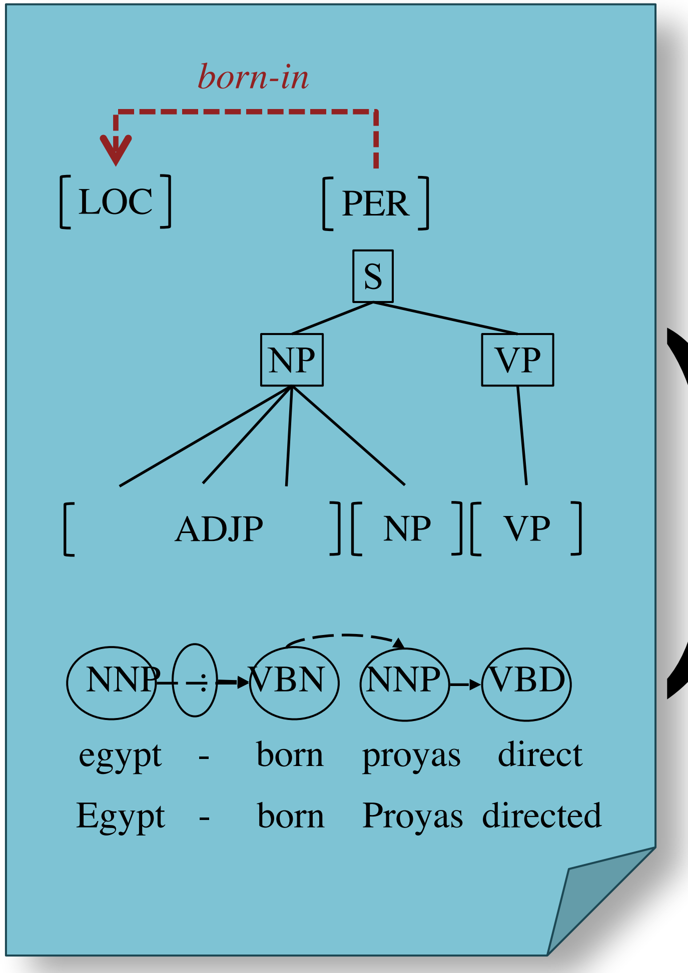
$$p(y|x) \propto \exp(\Theta_y \cdot f(\text{"Egypt-born Proyas directed..."}))$$

↑ "feature transform"

↑ "raw input"

Handcrafted Features

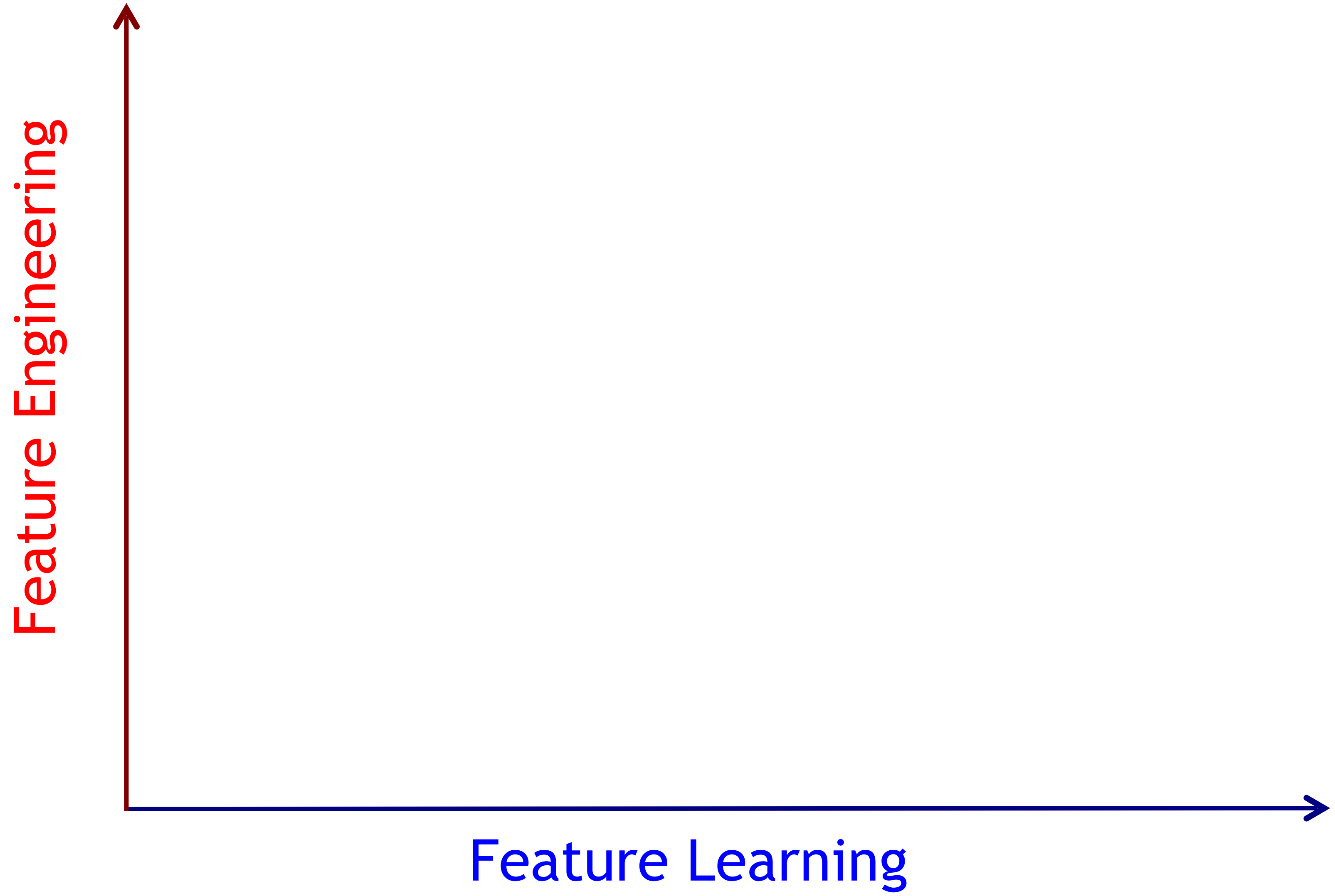
$$p(y|x) \propto \exp(\Theta_y \cdot f($$



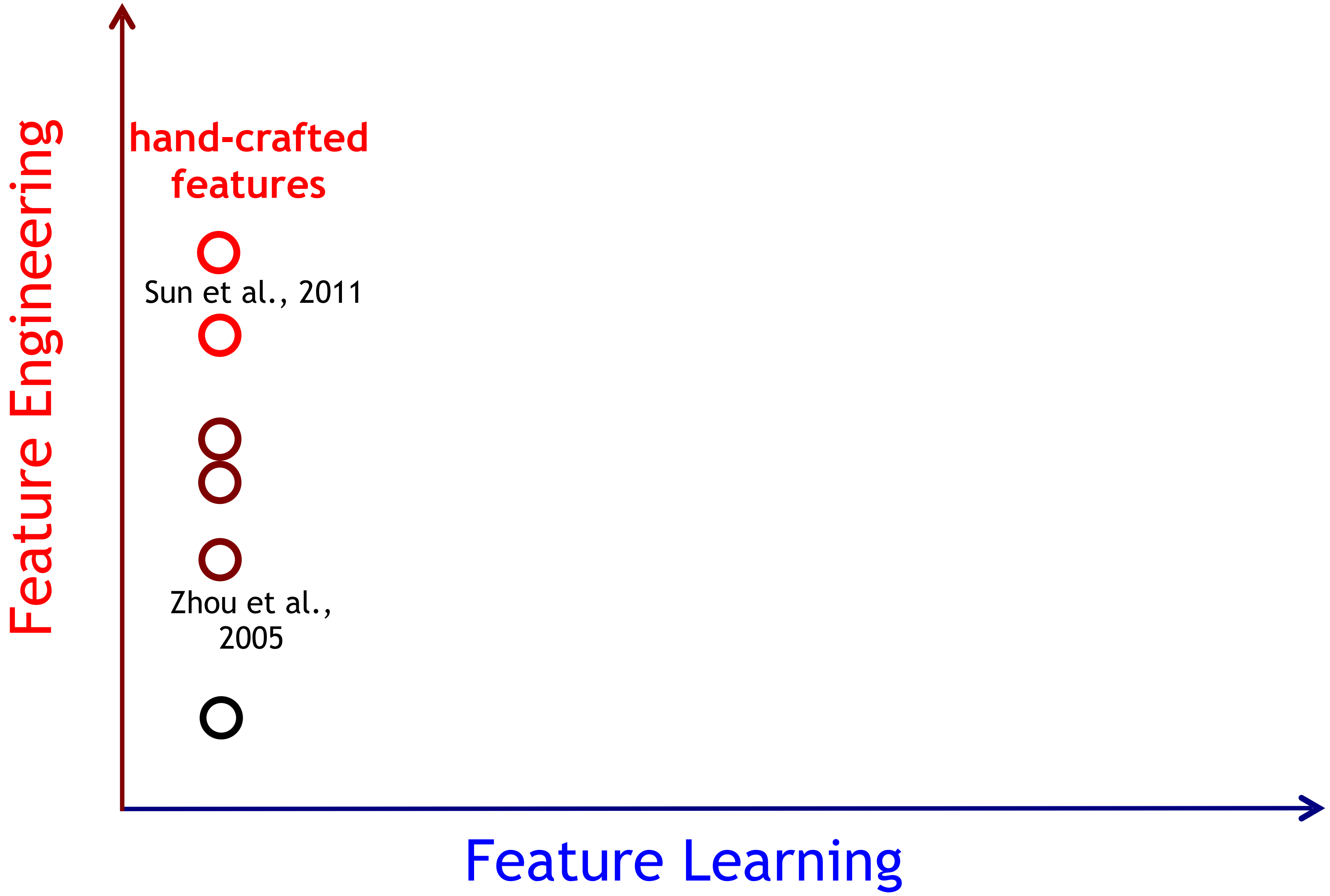
$$))$$

Feature transform can be complex code, often multiple stages

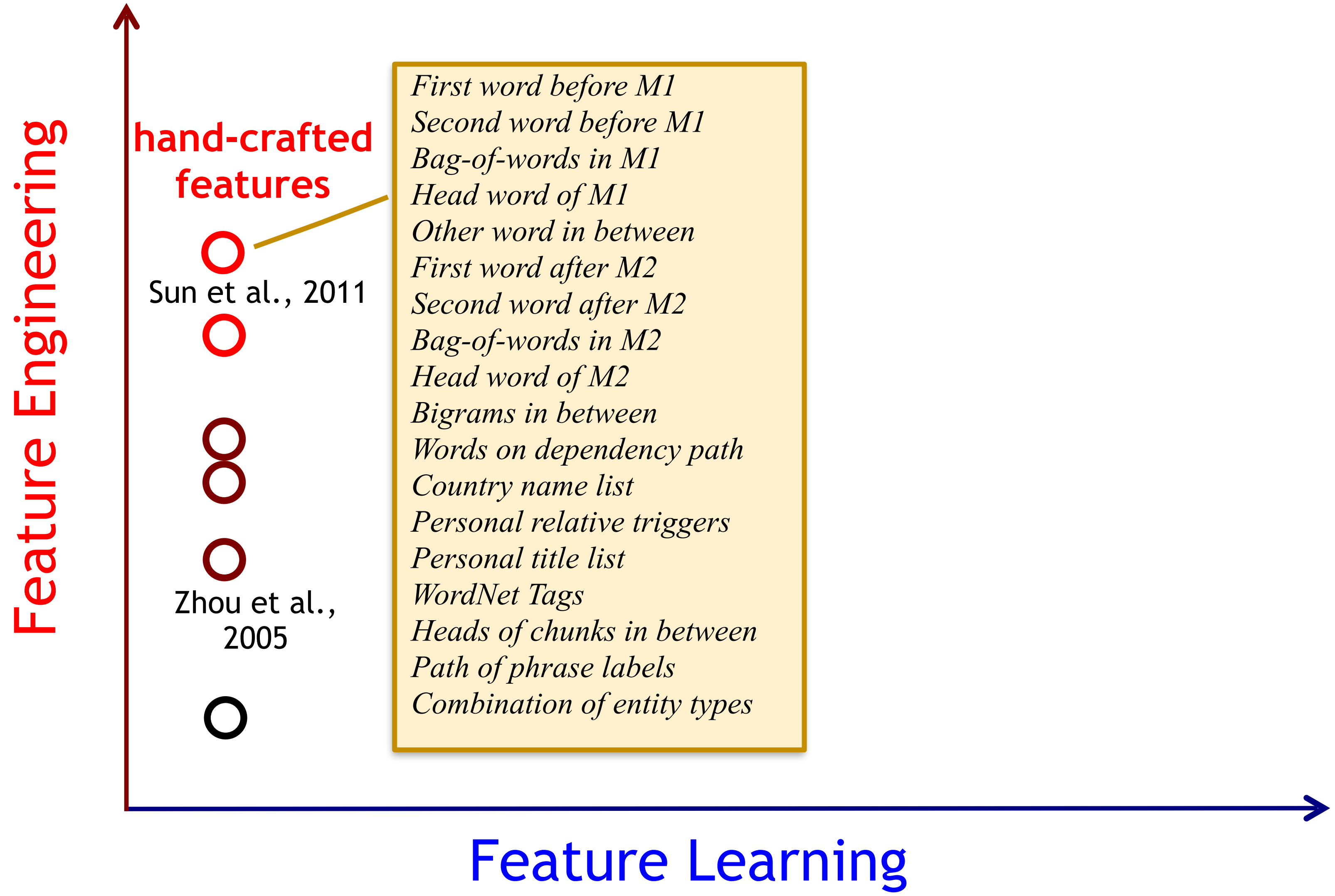
Where do features come from?



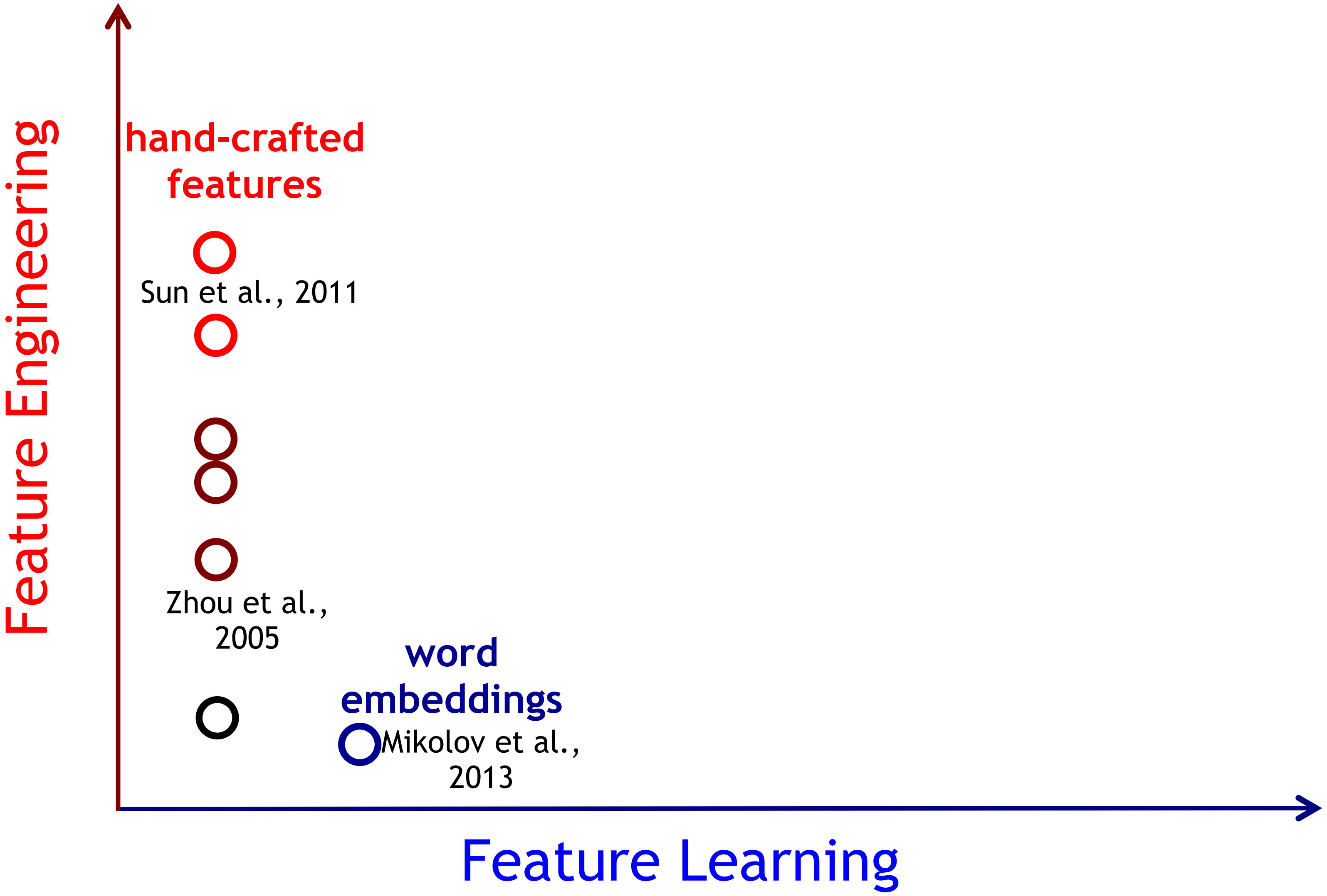
Where do features come from?



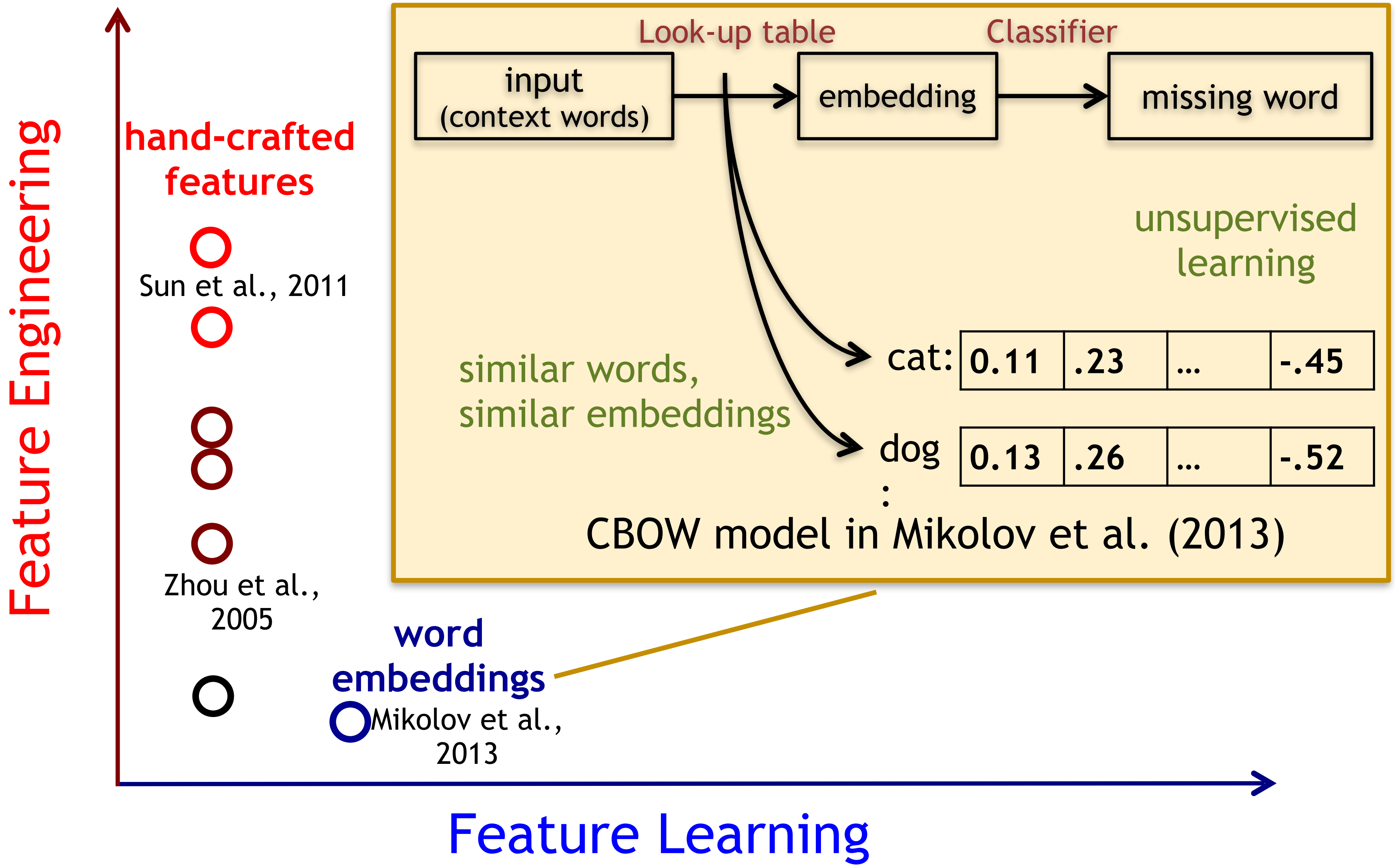
Where do features come from?



Where do features come from?



Where do features come from?



Word Embeddings

Cf: one-hot vectors

- Each coordinate: is this word “cat”?
- Vectors for related words share nothing in common

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

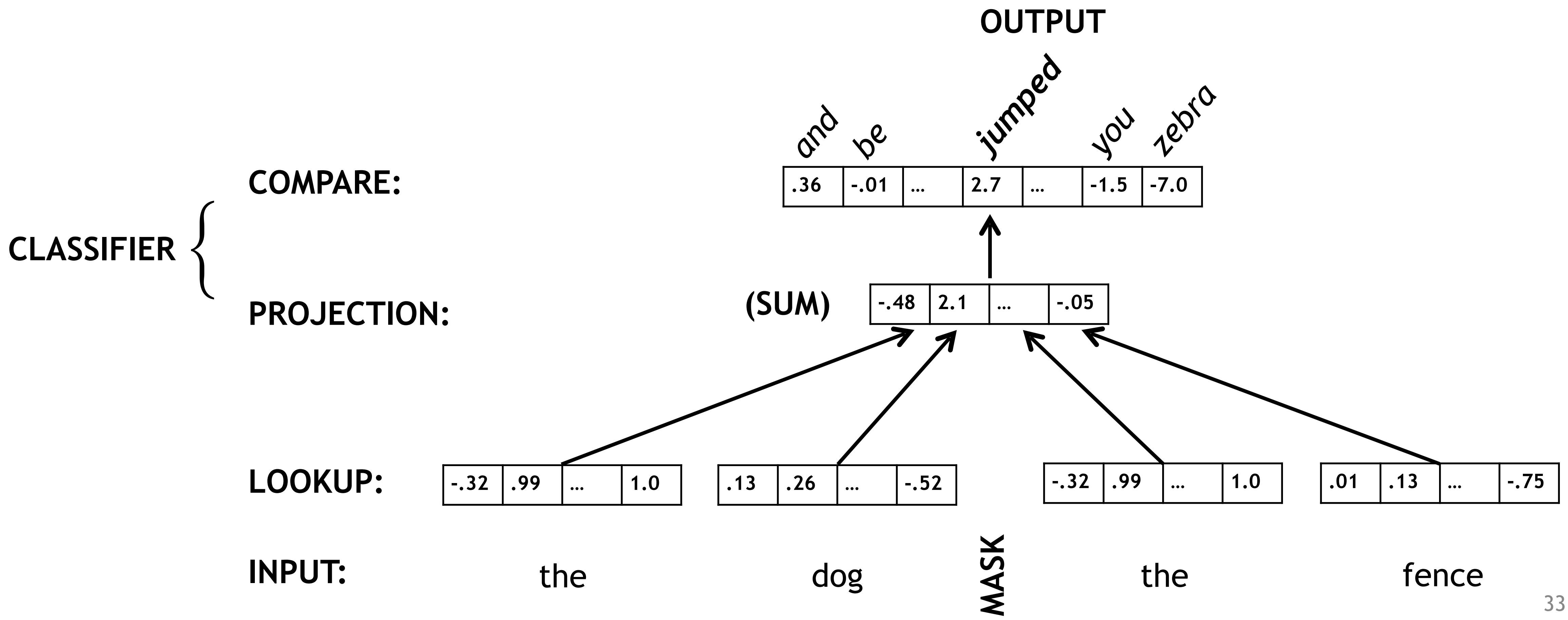
Word embeddings

- Real-valued vector representation of a word in *M* dimensions
- Typically ***learned***: linear model, neural net, …
- 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

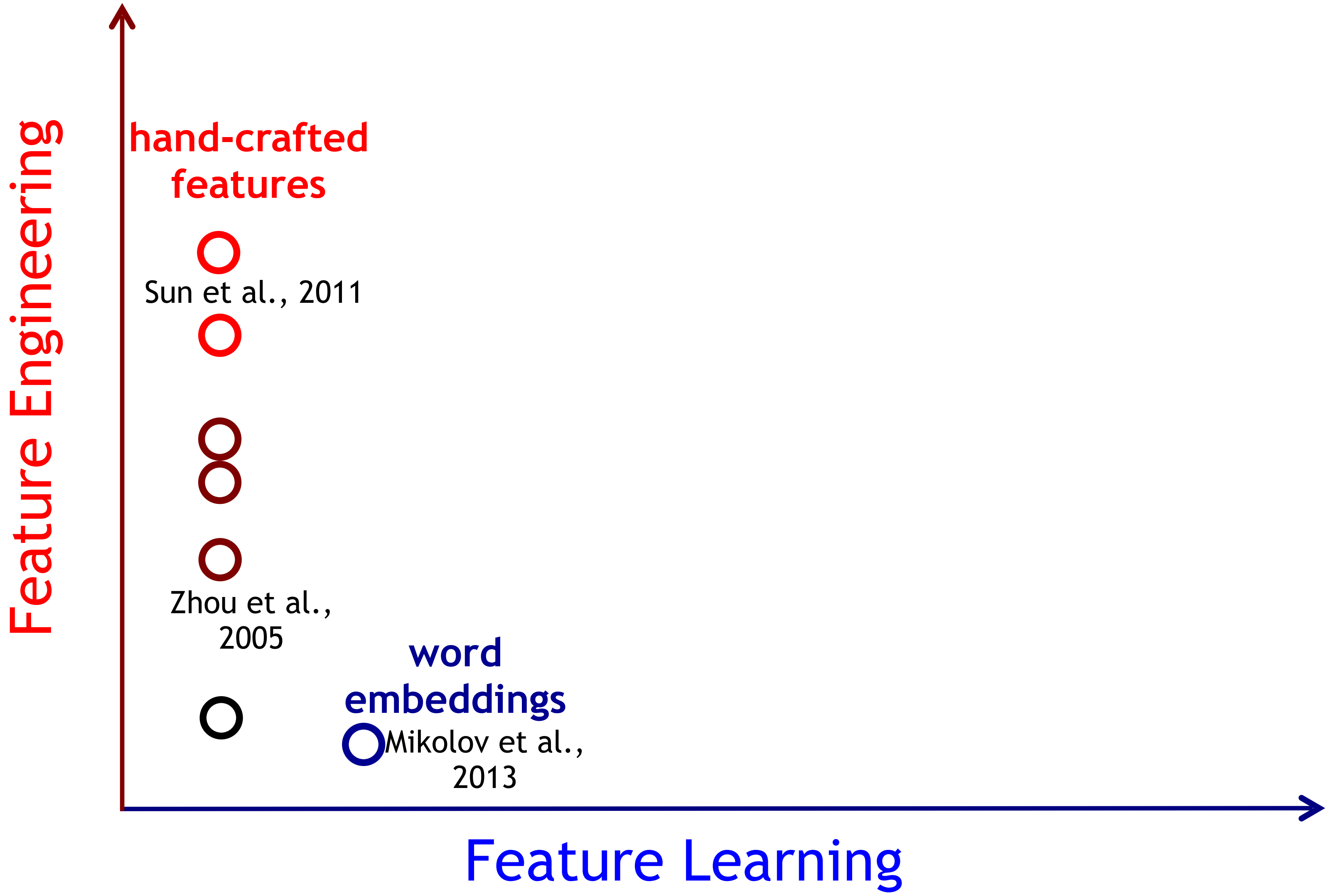
cat:	**0.13**	**.26**	…	**-.52**
dog:	**0.11**	**.23**	…	**-.45**

CBOW

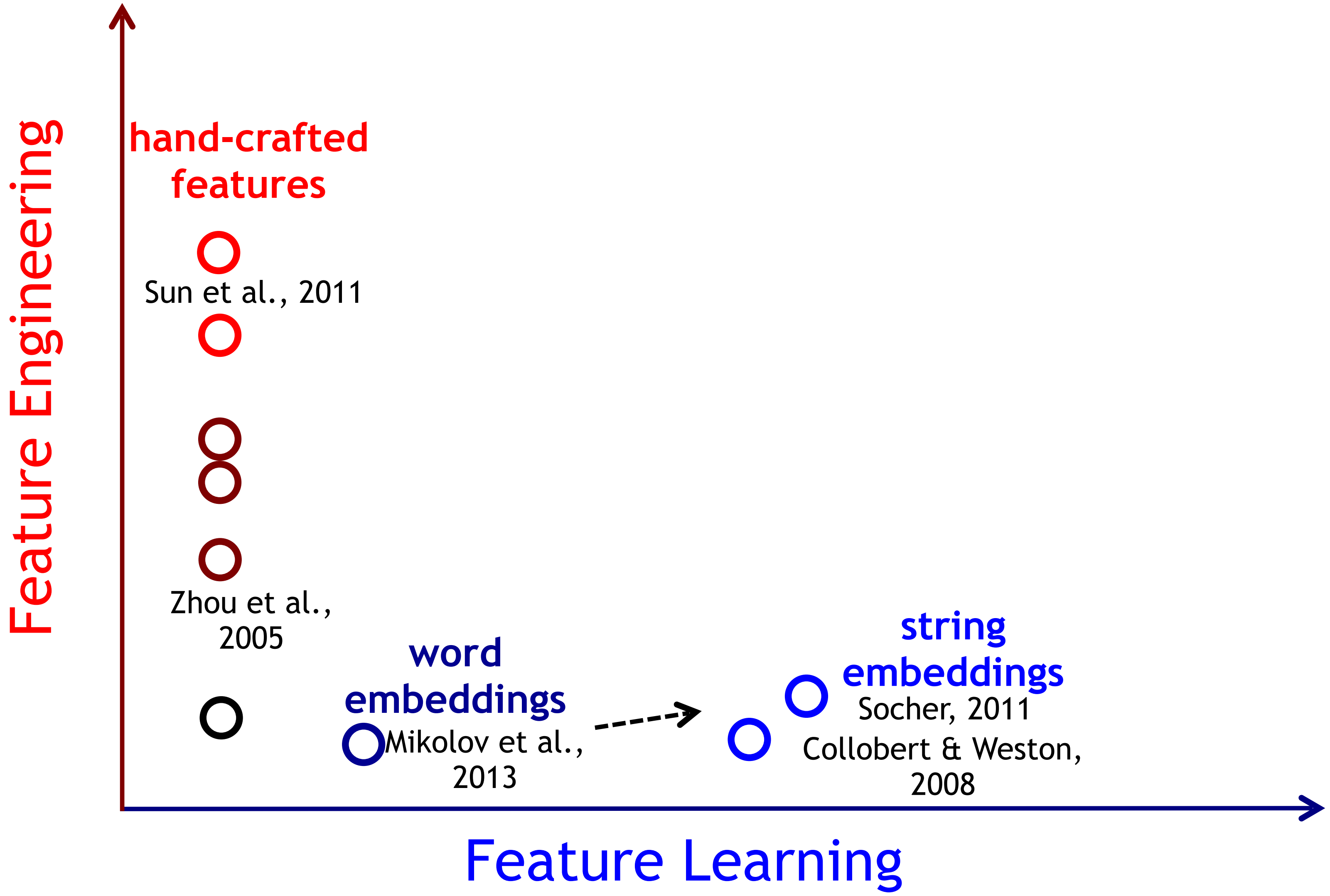
- Masked language modeling: cover up some words, predict them from context
- The Continuous Bag-of-words Model (CBOW) (Mikolov et al., 2013) trains a dictionary of embeddings to maximize the likelihood of a word given a short unordered context window around it



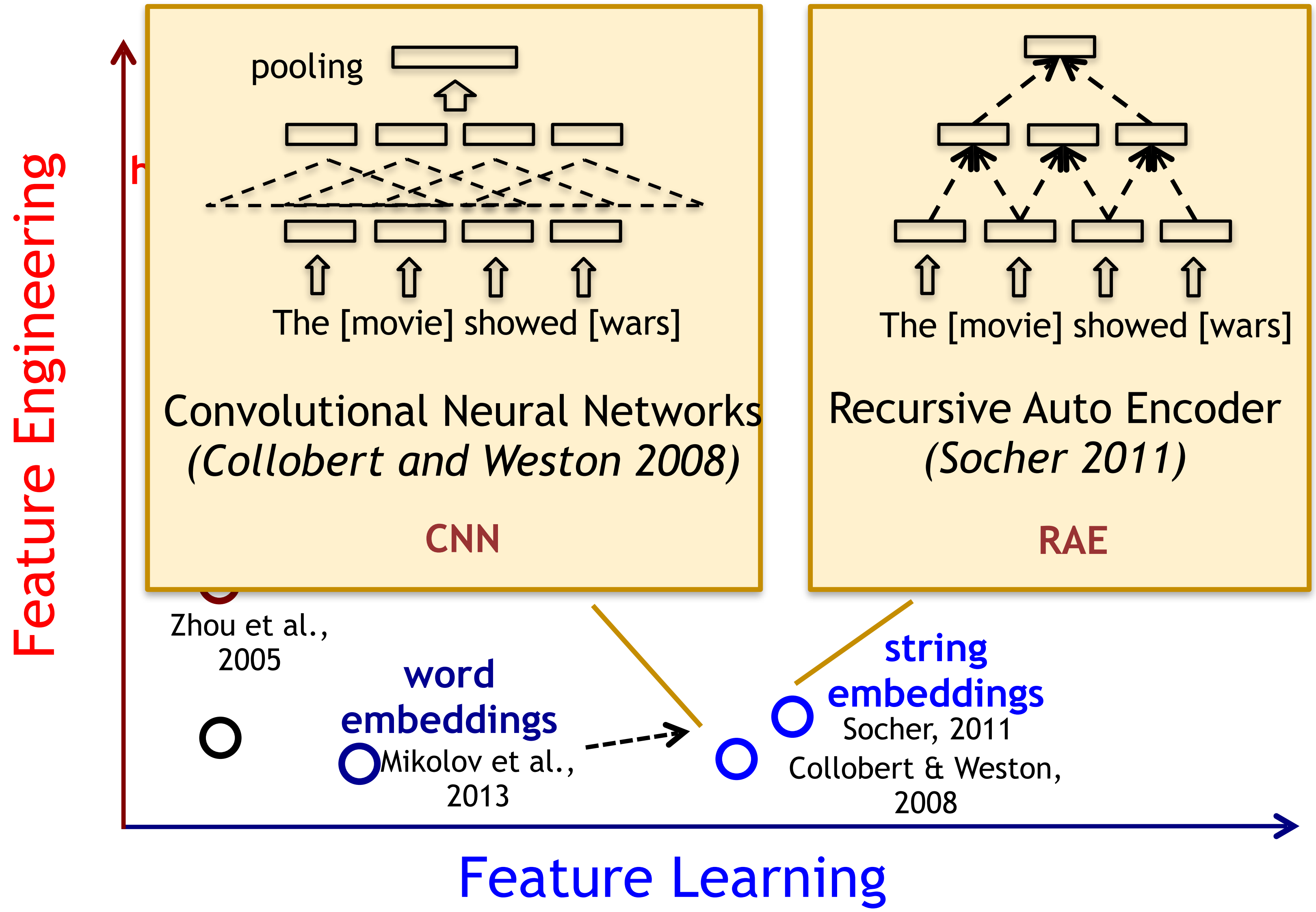
Where do features come from?



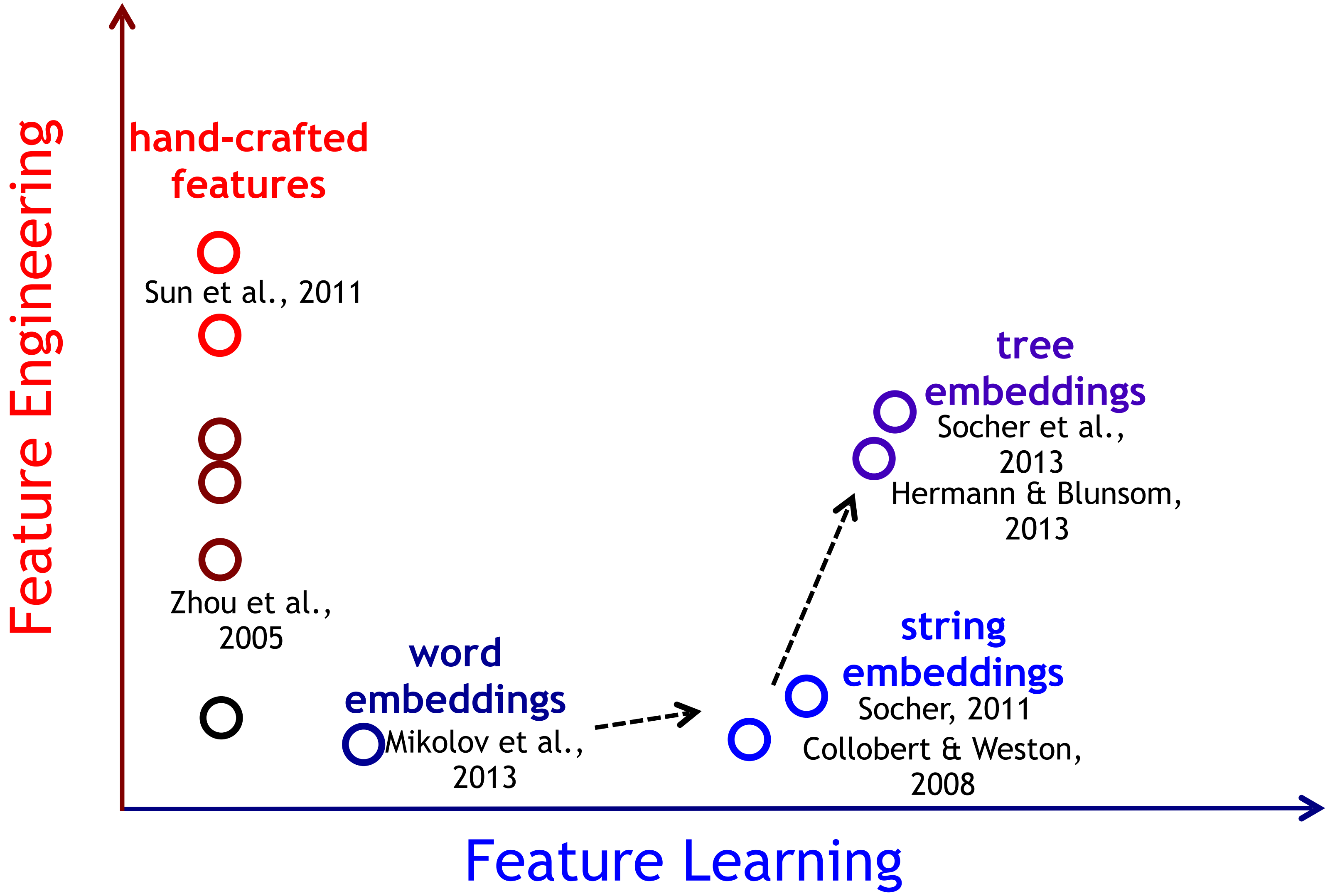
Where do features come from?



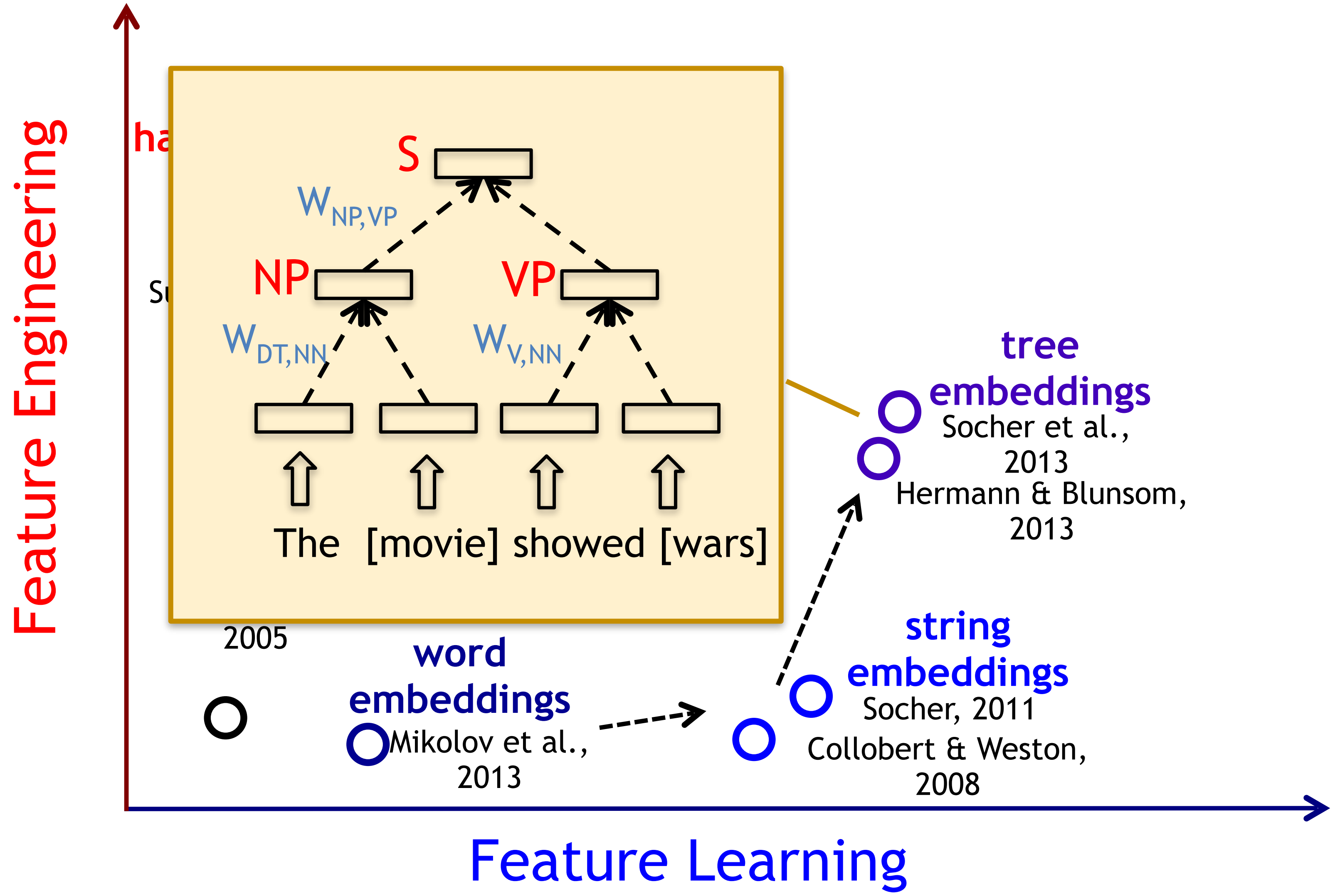
Where do features come from?



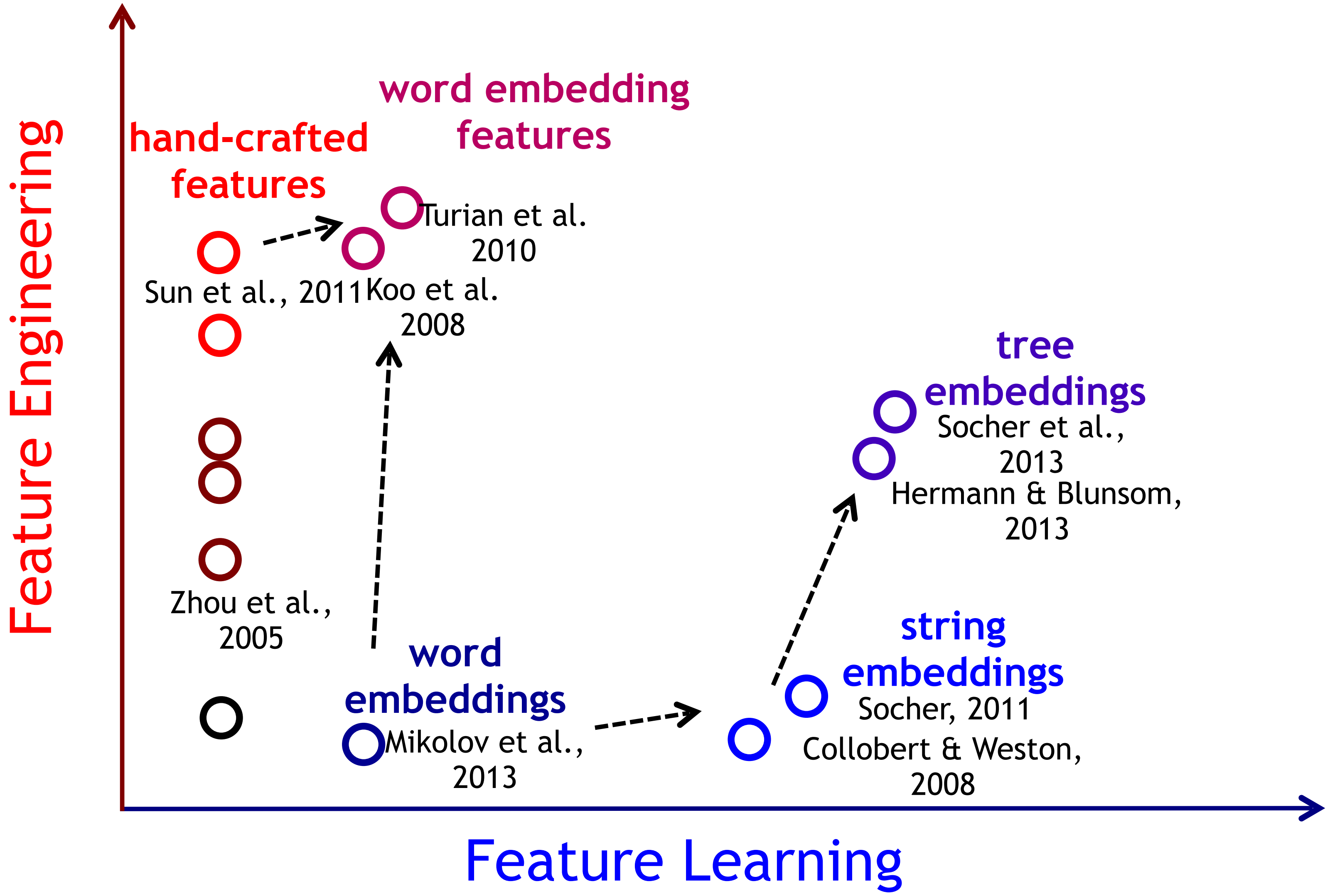
Where do features come from?



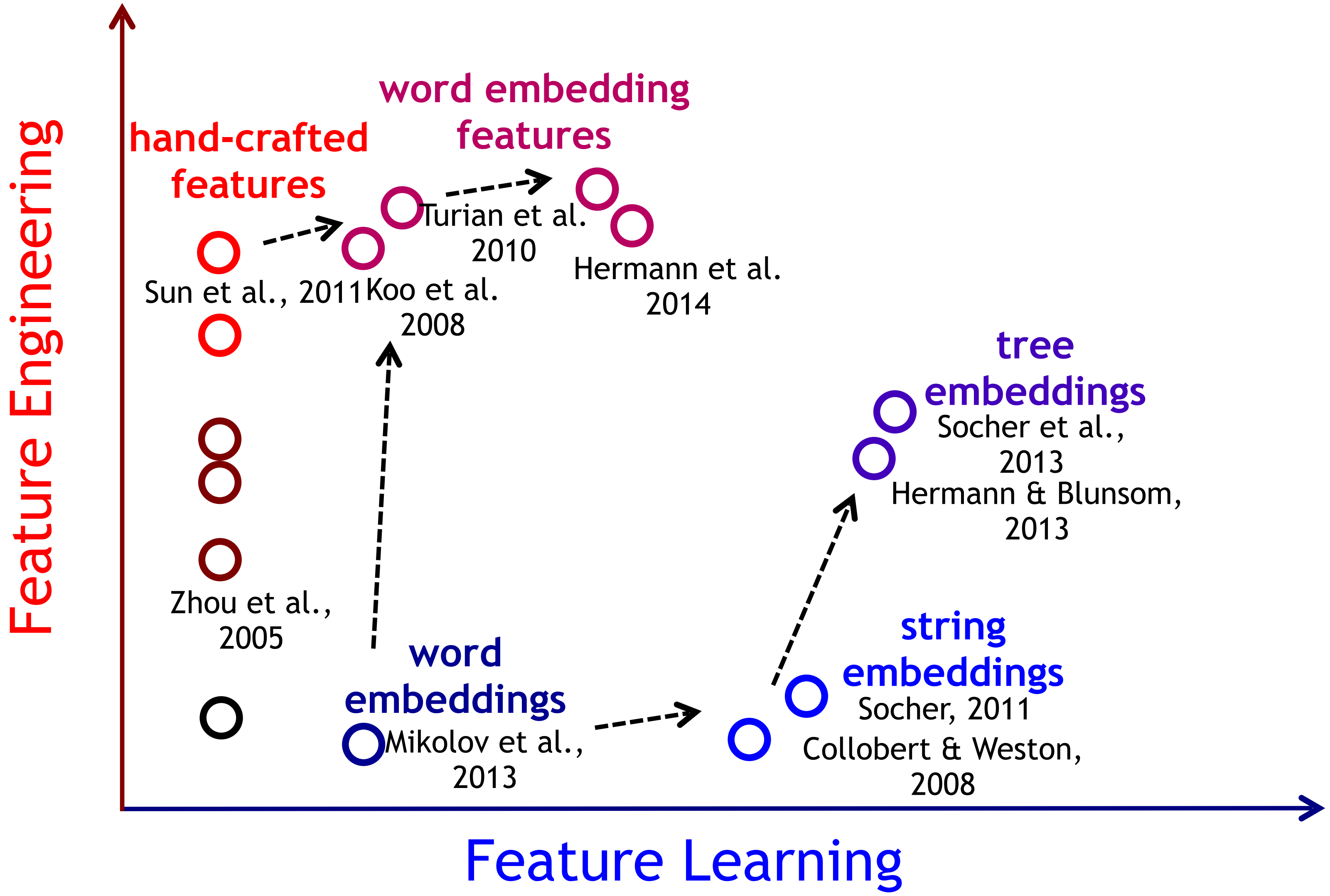
Where do features come from?



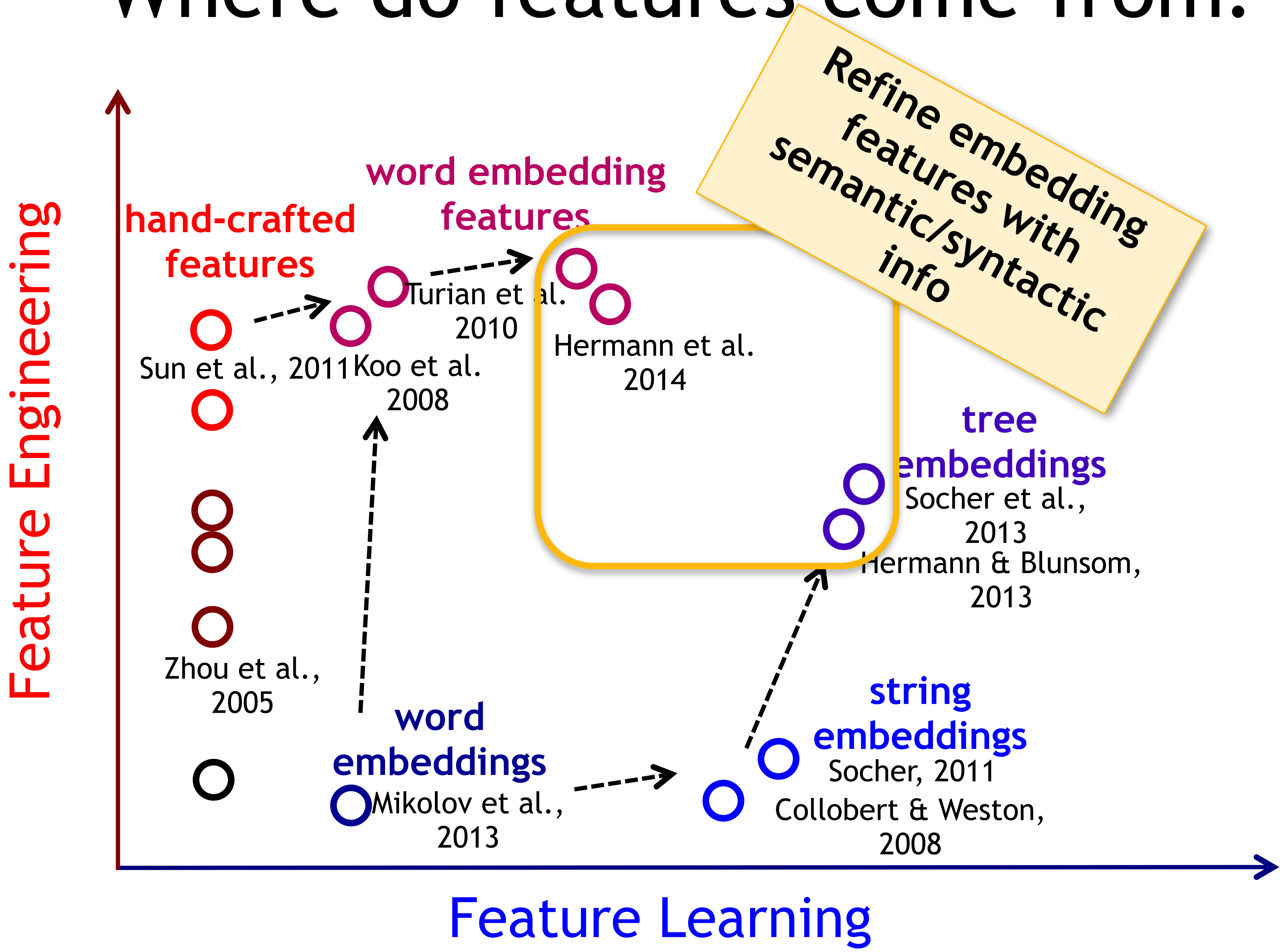
Where do features come from?



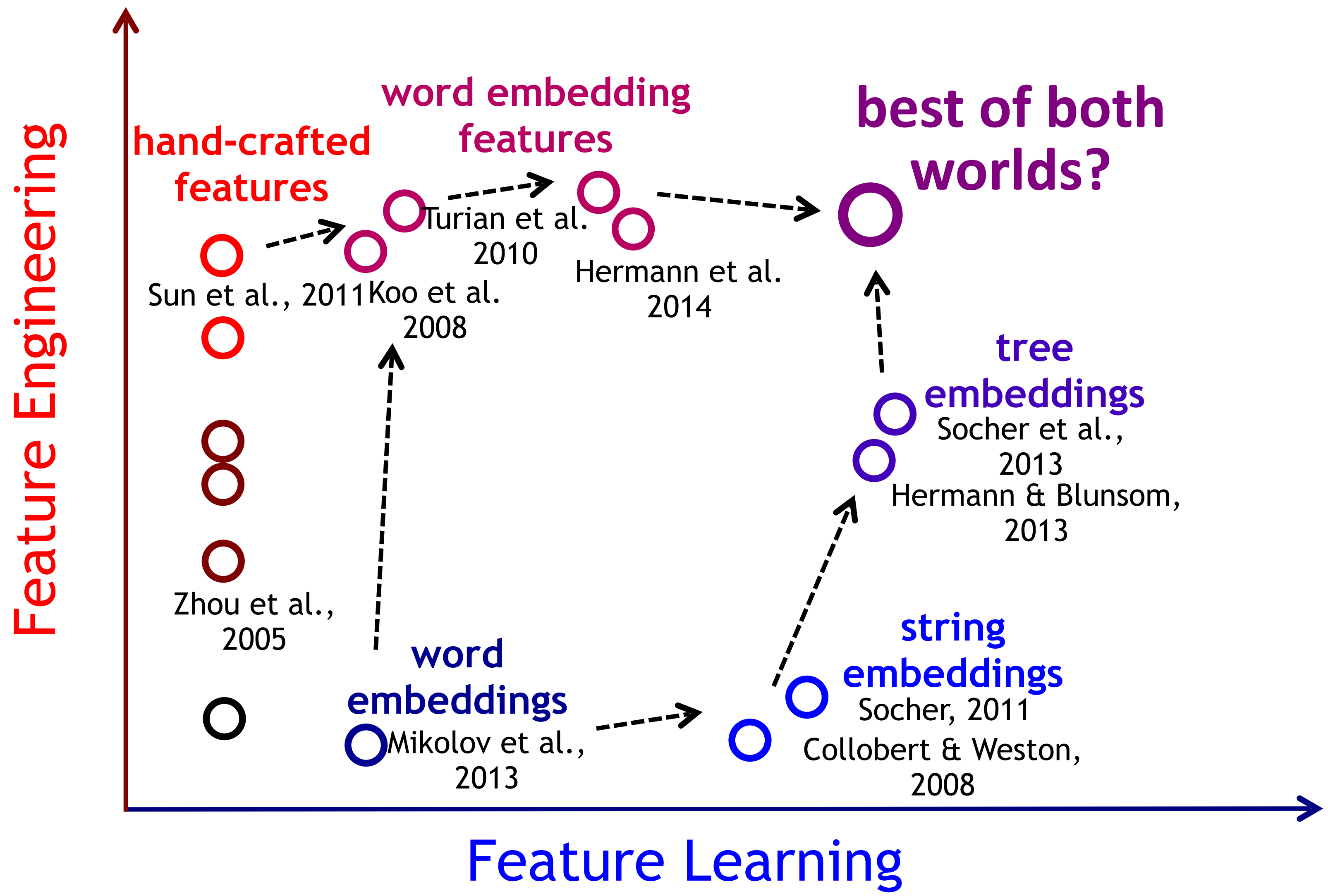
Where do features come from?



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?

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

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?

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

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?

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

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.

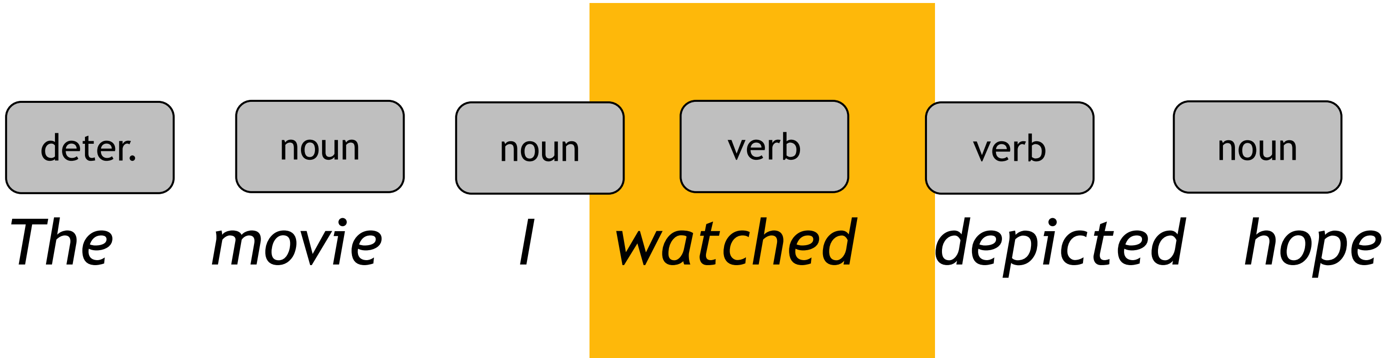
What features should you use?

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

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.

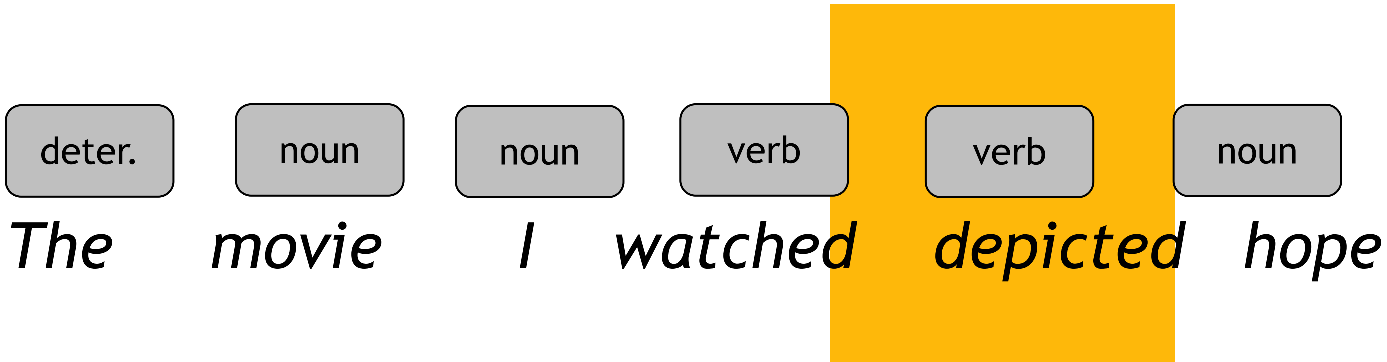
What features should you use?



Feature Engineering for NLP

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

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?

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

Feature Engineering for NLP

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

Feature Engineering for NLP

Per-word Features:

is-capital(w_i)

endswith(w_i,“e”)

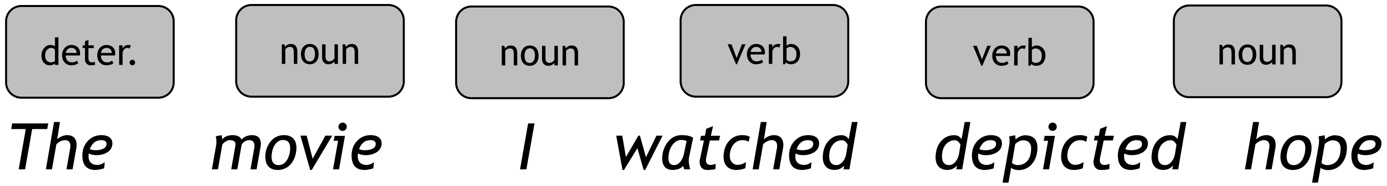
endswith(w_i,“d”)

endswith(w_i,“ed”)

w_i == “aardvark”

w_i == “hope”

…



Feature Engineering for NLP

Per-word Features:

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

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

Feature Engineering for NLP

Per-word Features:

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

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

Feature Engineering for NLP

Per-word Features:

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

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

Feature Engineering for NLP

Per-word Features:

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

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

Feature Engineering for NLP

Per-word Features:

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

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

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

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

Feature Engineering for NLP

Context Features:

…

w_i == "watched"

w_{i+1} == "watched"

w_{i-1} == "watched"

w_{i+2} == "watched"

w_{i-2} == "watched"

…

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

Feature Engineering for NLP

Context Features:

	$x^{(1)}$
...	...
w_i == "watched"	0
w_{i+1} == "watched"	0
w_{i-1} == "watched"	0
w_{i+2} == "watched"	0
w_{i-2} == "watched"	0
...	...

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

Feature Engineering for NLP

Context Features:

	$x^{(1)}$	$x^{(2)}$
…	…	…
w_i == “watched”	0	0
w_{i+1} == “watched”	0	0
w_{i-1} == “watched”	0	0
w_{i+2} == “watched”	0	1
w_{i-2} == “watched”	0	0
…	…	…

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

Feature Engineering for NLP

Context Features:

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

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

Feature Engineering for NLP

Context Features:

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

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

Feature Engineering for NLP

Context Features:

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

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

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

Feature Engineering for NLP

Context Features:

…

w_i == “I”

w_{i+1} == “I”

w_{i-1} == “I”

w_{i+2} == “I”

w_{i-2} == “I”

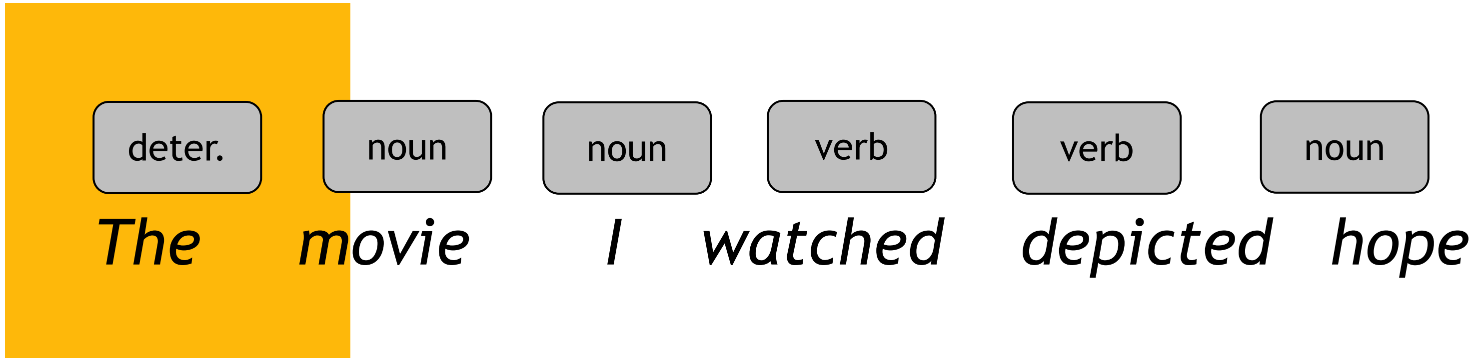
…

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

Feature Engineering for NLP

Context Features:

	$x^{(1)}$
…	…
w_i == "I"	0
w_{i+1} == "I"	0
w_{i-1} == "I"	0
w_{i+2} == "I"	1
w_{i-2} == "I"	0
…	…



Feature Engineering for NLP

Context Features:

	$x^{(1)}$	$x^{(2)}$
…	…	…
w_i == "I"	0	0
w_{i+1} == "I"	0	1
w_{i-1} == "I"	0	0
w_{i+2} == "I"	1	0
w_{i-2} == "I"	0	0
…	…	…

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

Feature Engineering for NLP

Context Features:

	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$
…	…	…	…
w_i == "I"	0	0	1
w_{i+1} == "I"	0	1	0
w_{i-1} == "I"	0	0	0
w_{i+2} == "I"	1	0	0
w_{i-2} == "I"	0	0	0
…	…	…	…

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

Feature Engineering for NLP

Context Features:

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

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

Feature Engineering for NLP

Context Features:

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

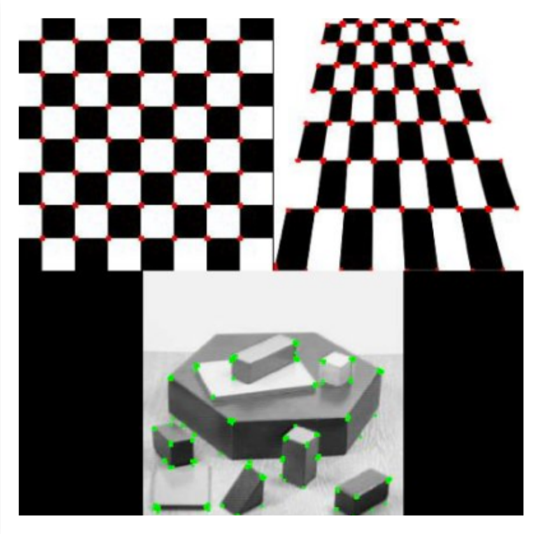
Model	Feature Templates	# Feats	Sent. Acc.	Token Acc.	Unk. Acc.
3GRAMMEMM	See text	248,798	52.07%	96.92%	88.99%
NAACL 2003	See text and [1]	460,552	55.31%	97.15%	88.61%
Replication	See text and [1]	460,551	55.62%	97.18%	88.92%
Replication′	+rareFeatureThresh = 5	482,364	55.67%	97.19%	88.96%
5W	$+\langle t_0, w_{-2}\rangle, \langle t_0, w_2\rangle$	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%

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

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.

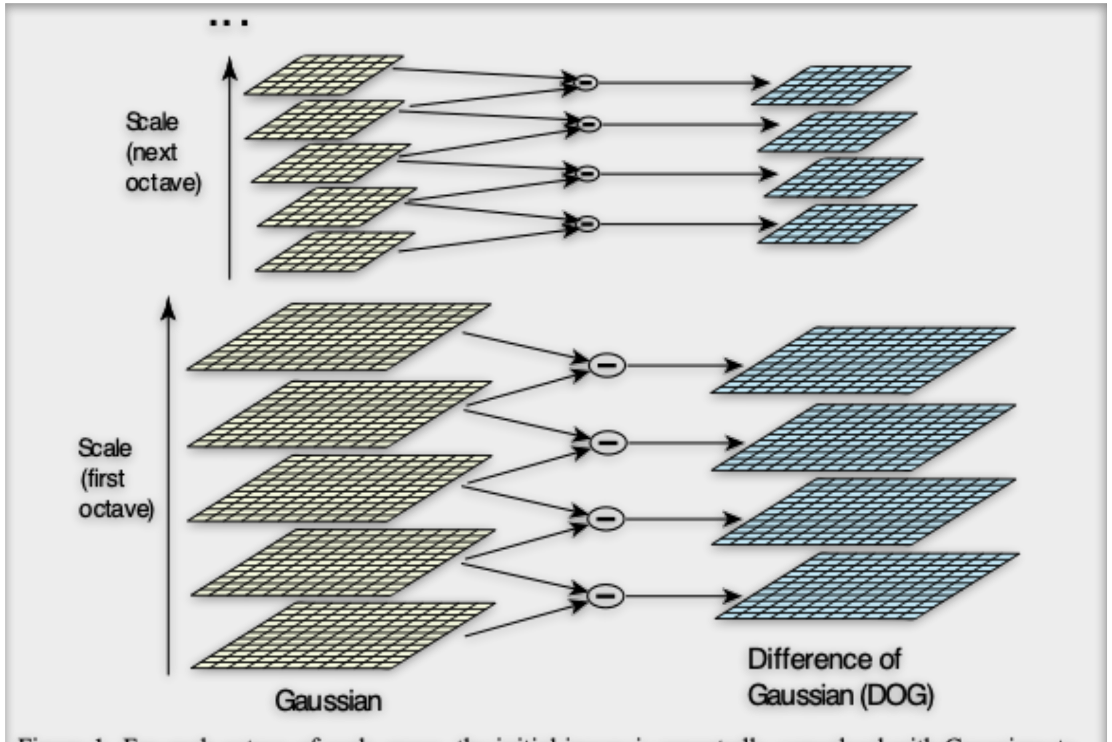


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled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

Figure from Lowe (1999) and Lowe (2004)

NONLINEAR BASIS FUNCTIONS

What if raw input is a real vector?

$$p(y|x) \propto \exp(\Theta_y \cdot f(\text{“Egypt-born Proyas directed...”}))$$

What if raw input is a real vector?

$$p(y|x) \propto \exp(\Theta_y \cdot f((3.7,\ -0.2, 1.7,\ \ldots,\ -4.2)^T))$$

Continuous nonlinear feature transform

- aka. "nonlinear basis functions"
- **Key Idea**: pick a family of functions of **x**
  - original input: $\mathbf{x} \in \mathbb{R}^M$
  - new transformed input: $\mathbf{x}' \in \mathbb{R}^{M'}$ where $M' > M$ (usually)
  - define

$$\mathbf{x}' = f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots f_{M'}(\mathbf{x}))$$

each $f_i : \mathbb{R}^M \to \mathbb{R}$ is arbitrary but fixed

- **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	$b_j(x) = \frac{1}{1+\exp(-\omega_j x)}$

Continuous nonlinear feature transform

- aka. “nonlinear basis functions”
- **Key Idea:** pick a family of functions of **x**
  - original input: $\mathbf{x} \in \mathbb{R}^M$
  - new transformed input: $\mathbf{x}' \in \mathbb{R}^{M'}$ where $M' > M$ (usually)
  - define

$$\mathbf{x}' = f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots f_{M'}(\mathbf{x}))$$

each $f_i : \mathbb{R}^M \rightarrow \mathbb{R}$ is arbitrary but fixed

- **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	$b_j(x) = \frac{1}{1 + \exp(-\omega_j x)}$

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

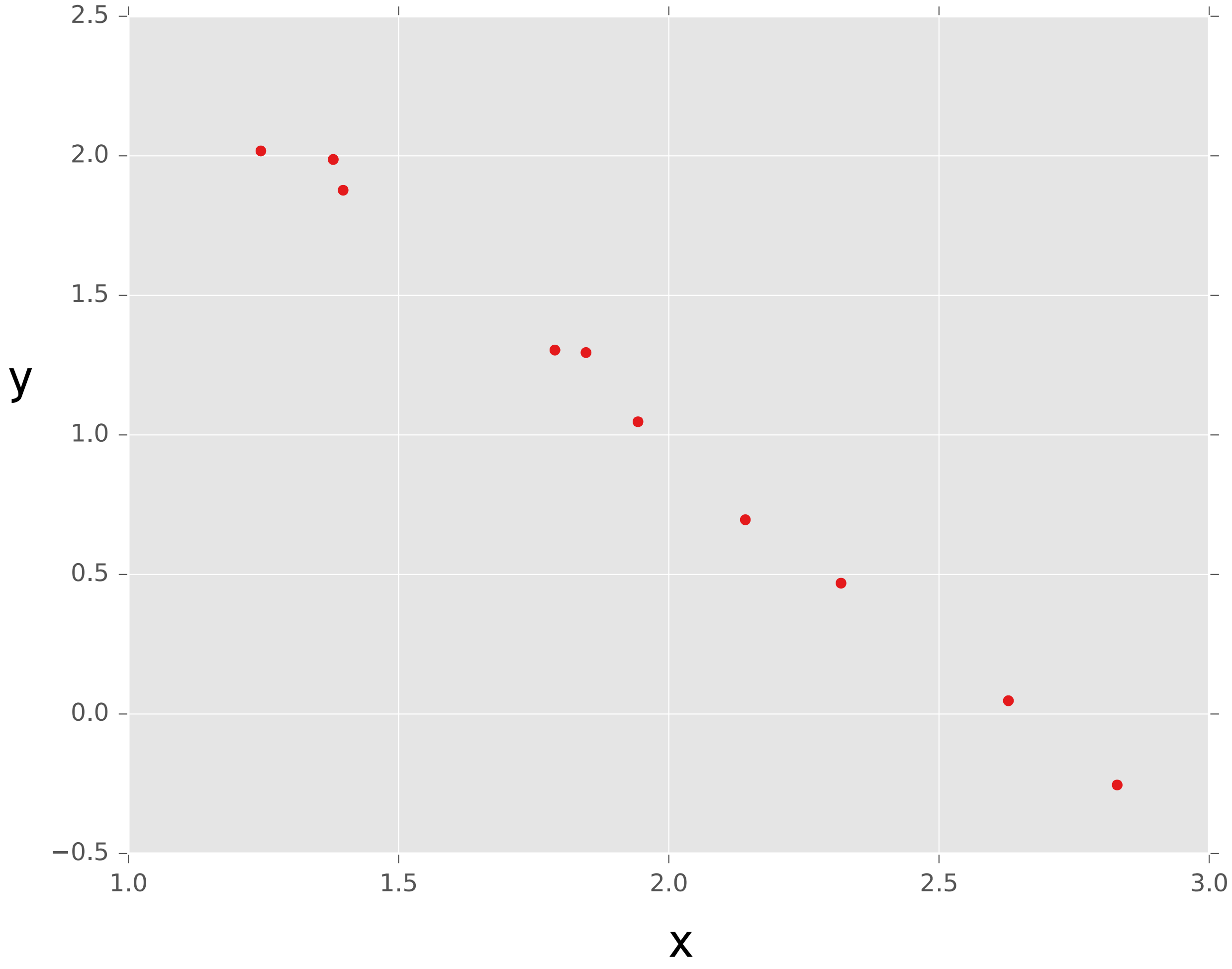
Example uses:
- Perceptron
- Linear regression
- Logistic regression

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

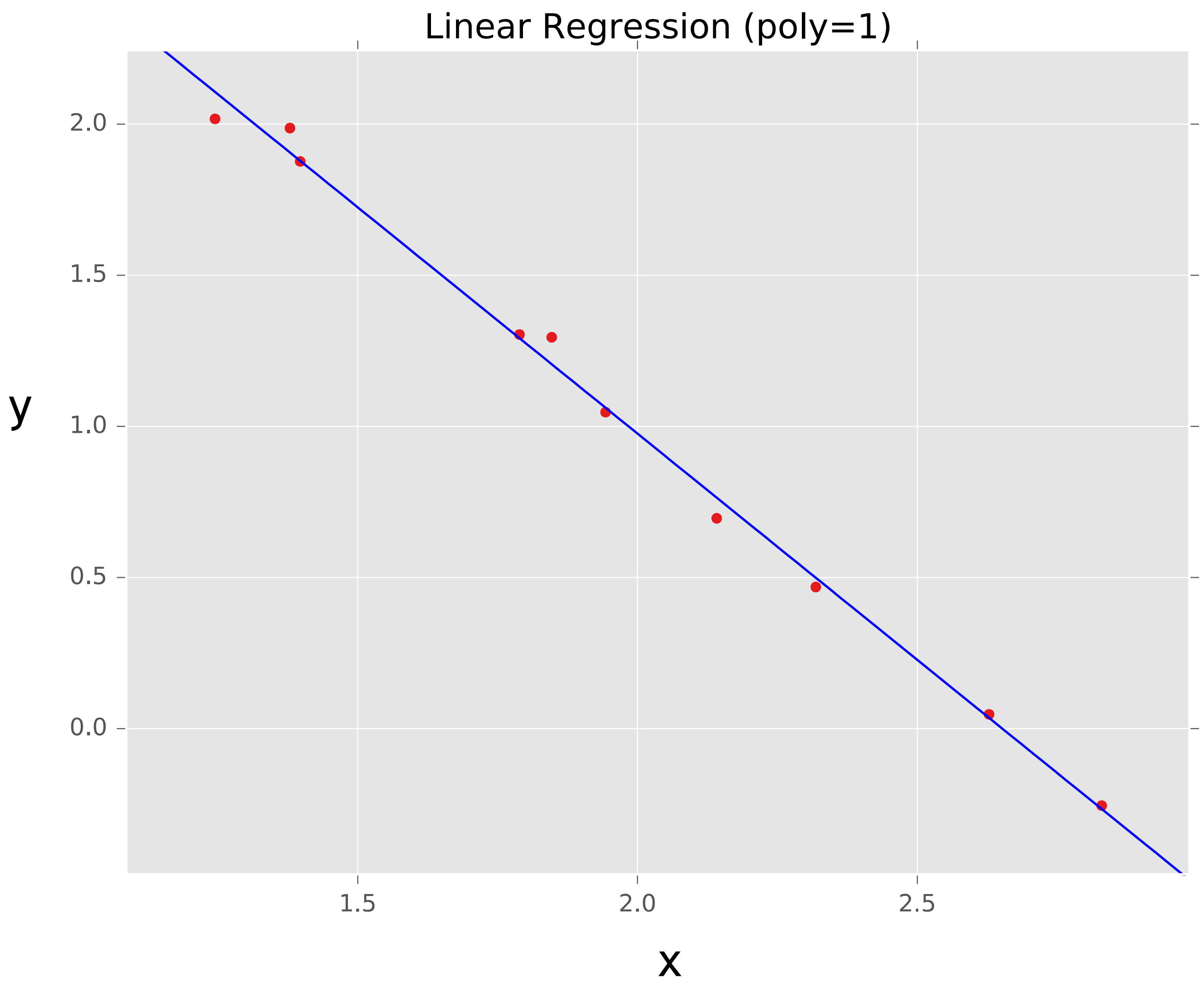


Example: Linear Regression

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

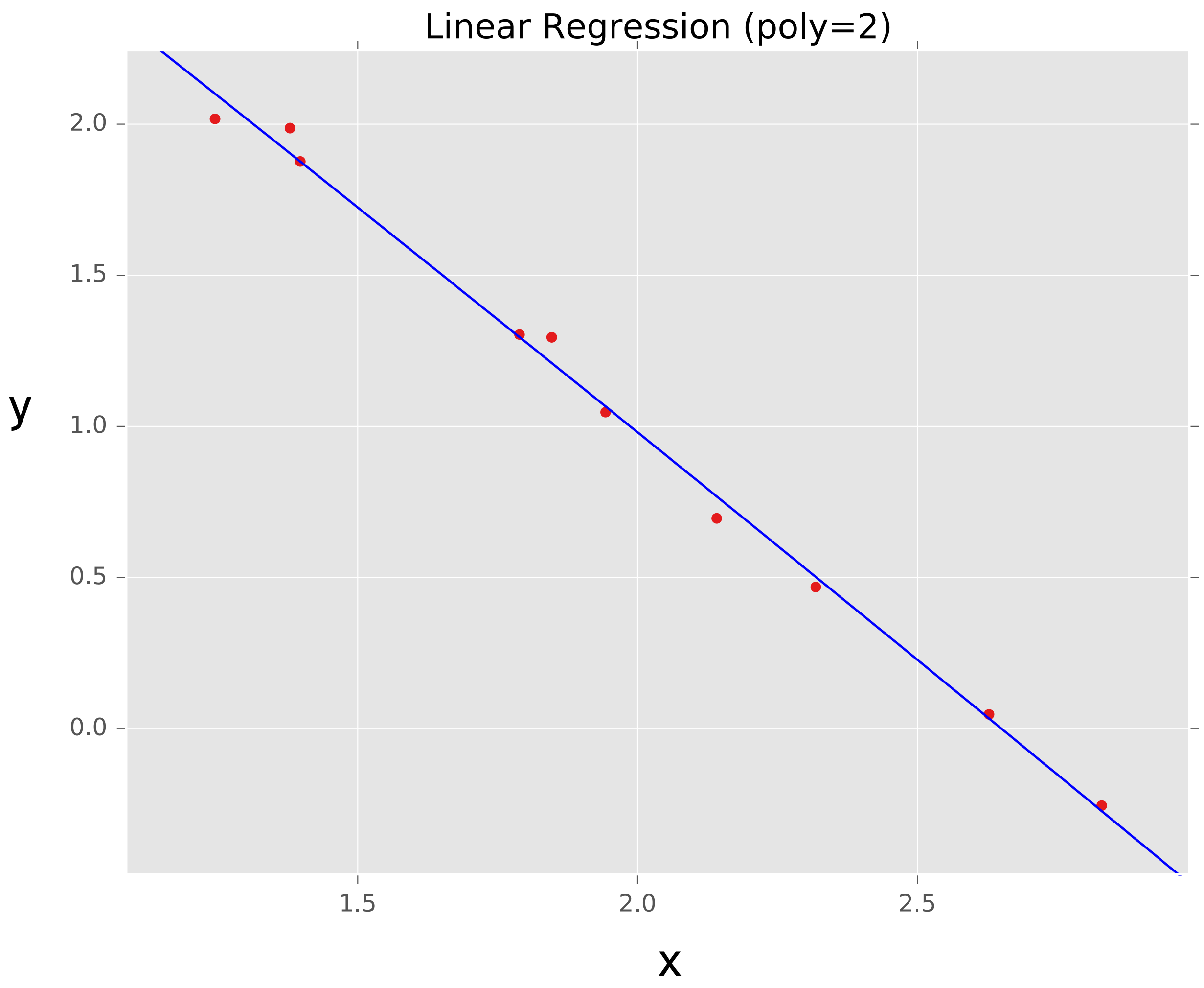


Example: Linear Regression

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

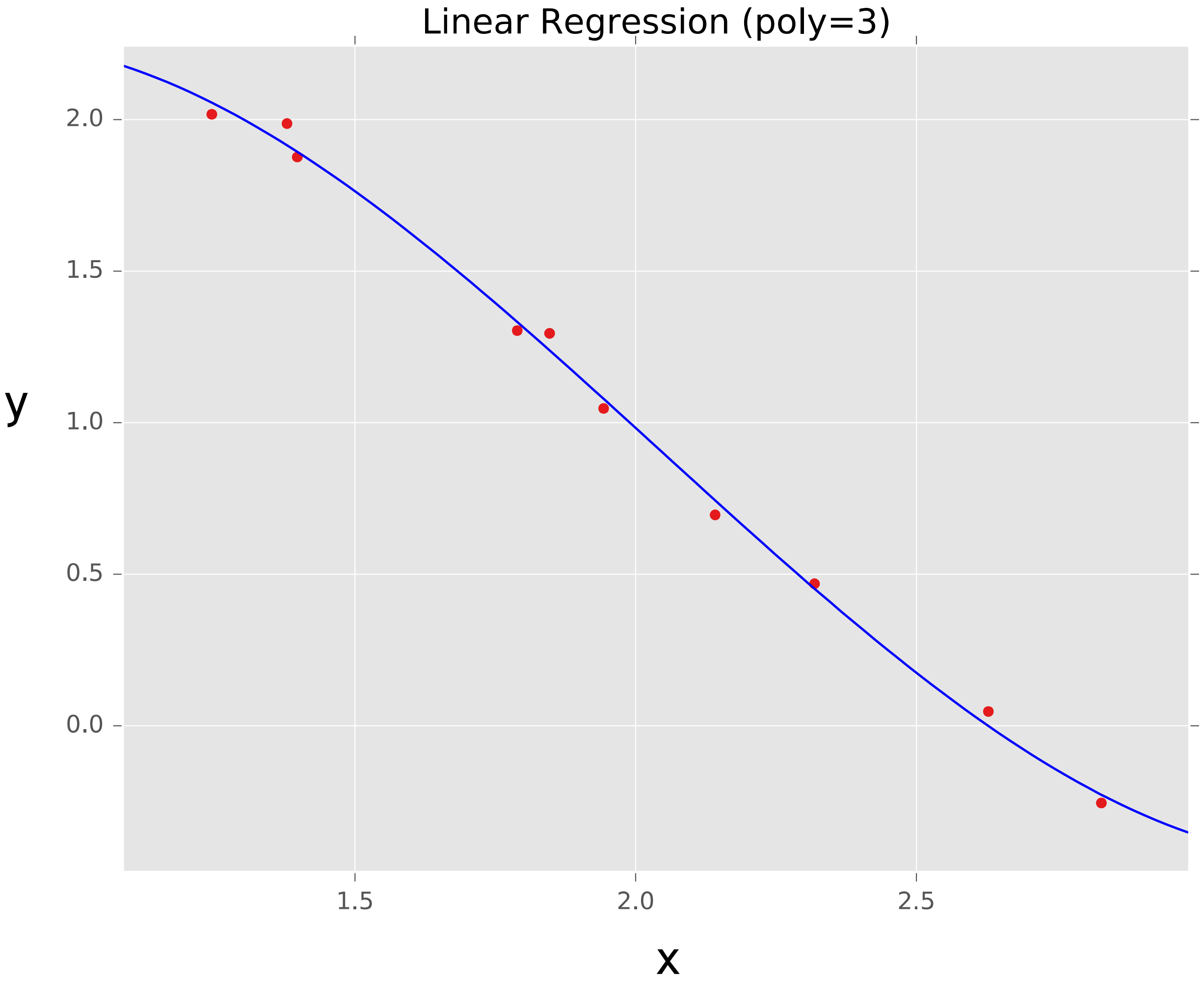


Example: Linear Regression

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

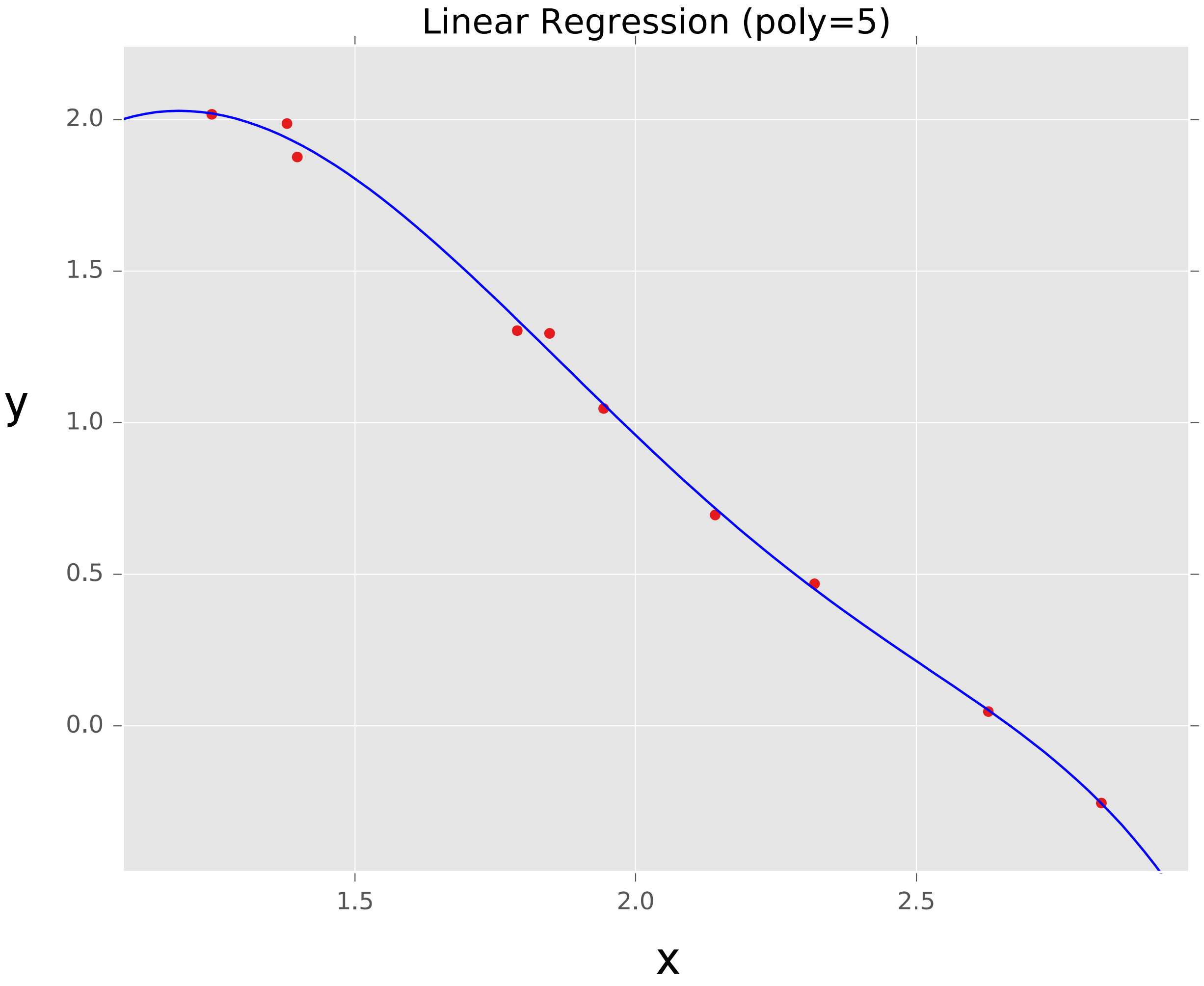


Example: Linear Regression

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

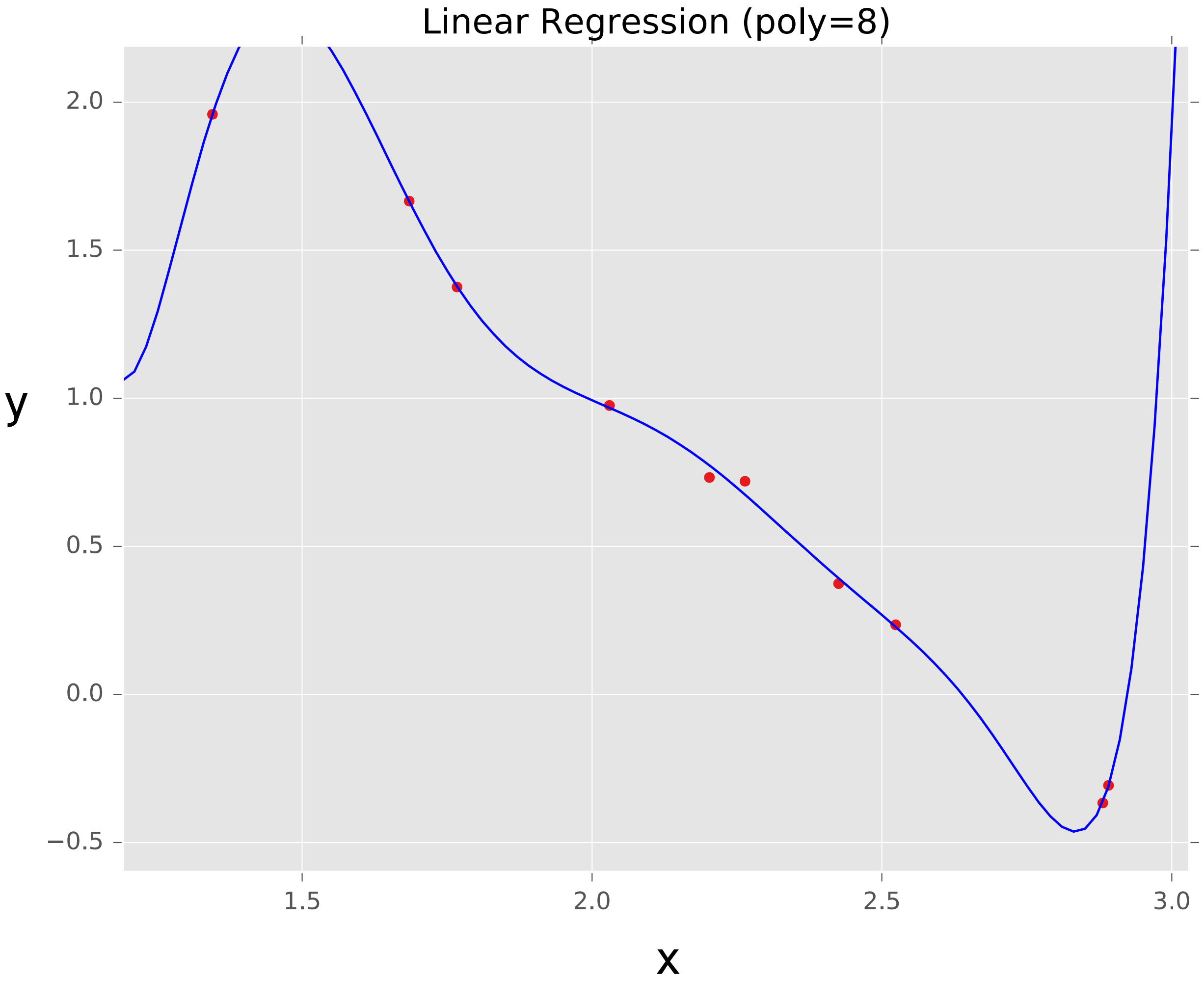


Example: Linear Regression

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

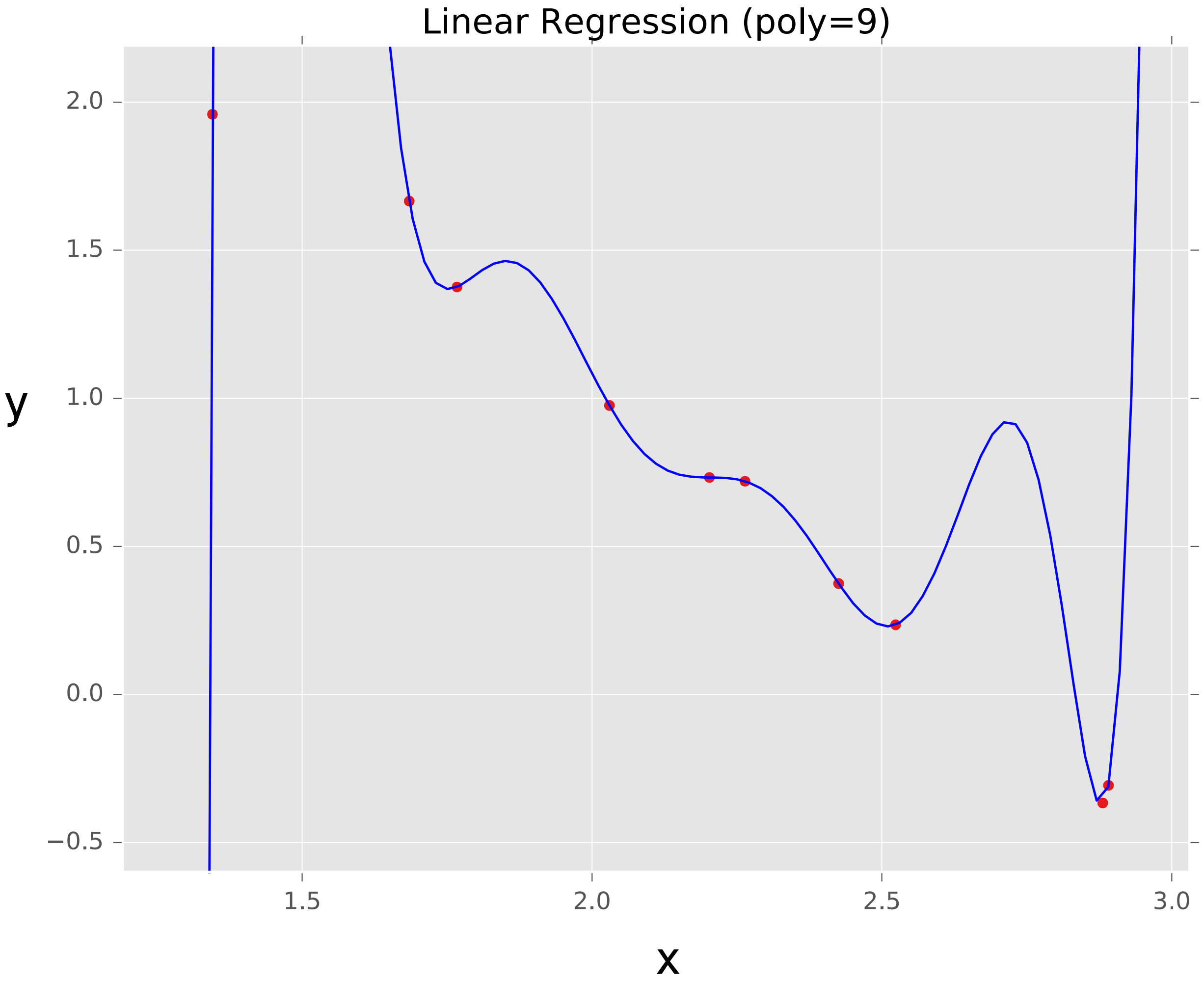


Example: Linear Regression

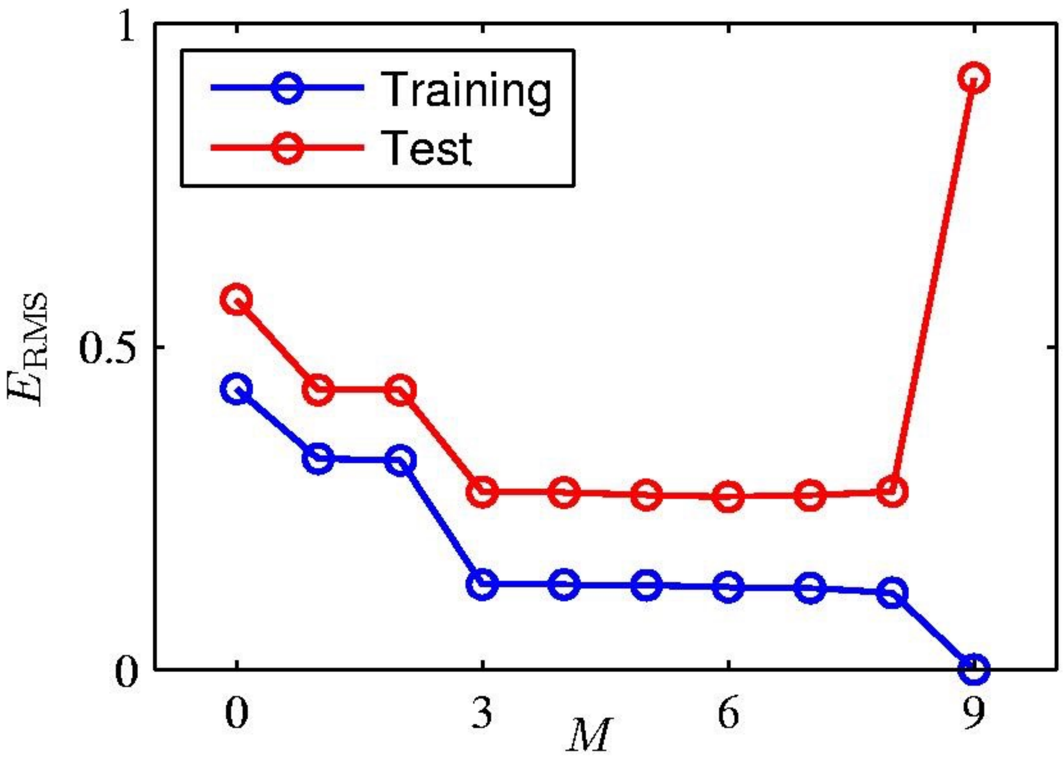
Goal: Learn $y = \mathbf{w}^T f(\mathbf{x}) + b$
where f(.) is a polynomial
basis function

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

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

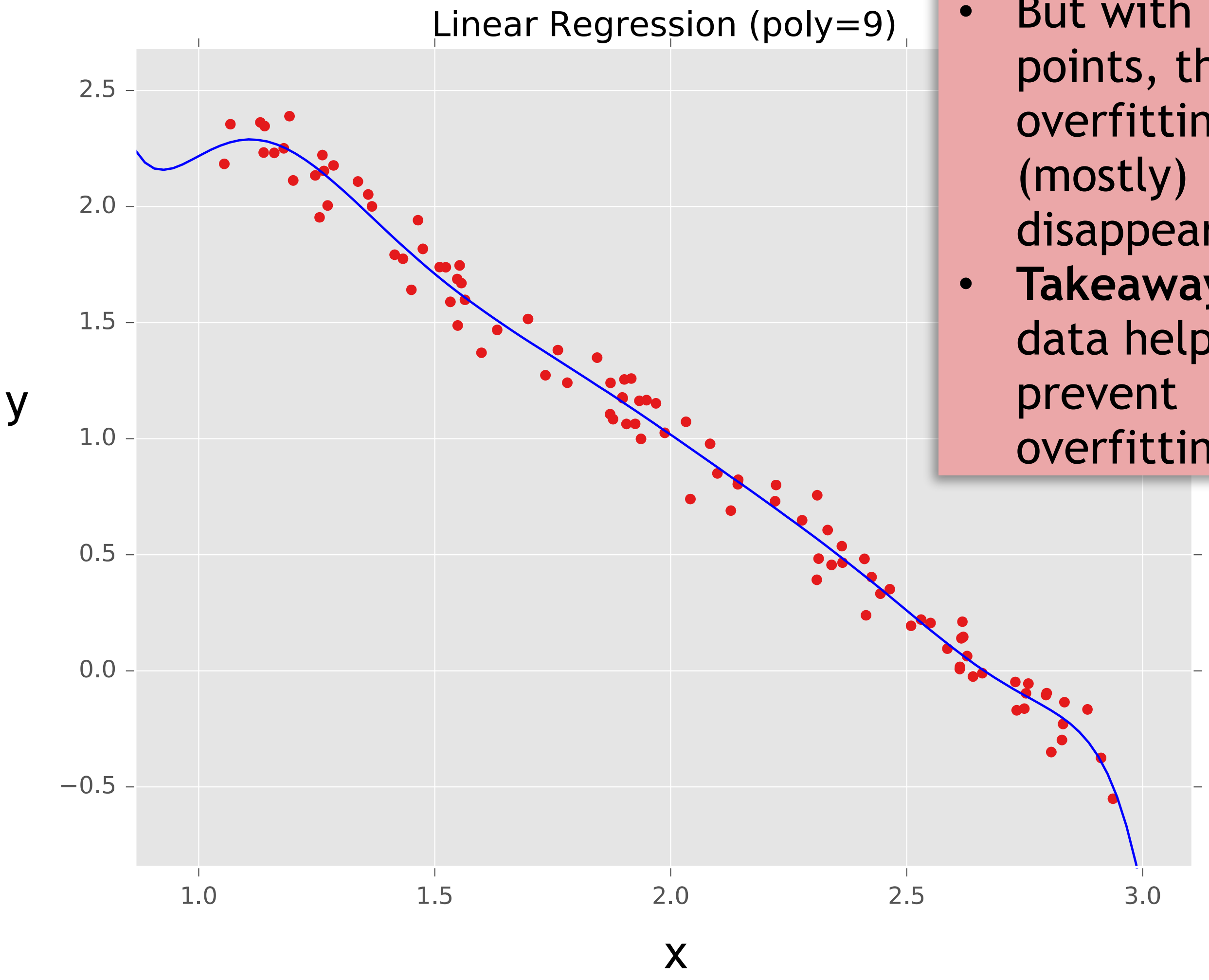
i	y	x	…	x^9
1	2.0	1.2	…	$(1.2)^9$
2	1.3	1.7	…	$(1.7)^9$
…	…	…	…	…
10	1.1	1.9	…	$(1.9)^9$

- 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

Recall: 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 too many features for # examples)
- Linear Regression (e.g. if we add nonlinear features)
- Logistic Regression (e.g. with many rare features)

Motivation: Regularization

- **Occam’s Razor:** prefer the simplest hypothesis that fits the data

- We know how to measure **fits the data**
- What is a **simple** hypothesis (or model)?
  1. small number of features (**model selection**)
  2. small number of “important” features (**shrinkage**)

Regularization

- **Given** original objective function: $J(\theta)$
- **New goal** is to find:

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$$

- **Key idea**: Define regularizer $r(\boldsymbol{\theta})$ that measures **simplicity**
- Now we trade off between fitting the data and keeping the model simple
- **Choose form of regularizer:**
  - Common choices:

L_1 $\qquad$ L_2^2

Regularization Examples

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

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

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

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

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

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

Regularization Examples

Add an **L2 regularizer** to Logistic Regression

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

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

Add an **L1 regularizer** to Logistic Regression

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

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

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

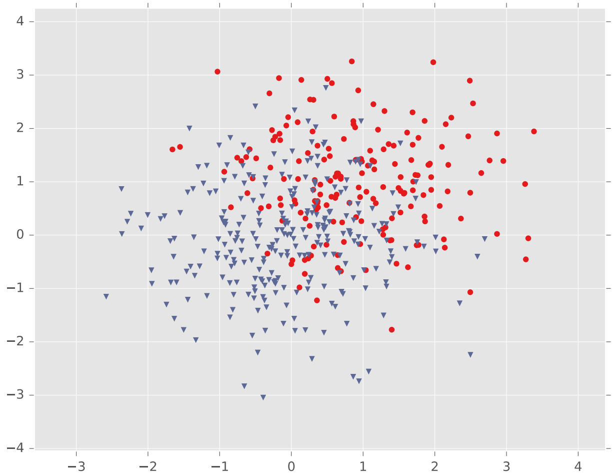
- It’s common to *standardize* each feature by subtracting its mean and dividing by its standard deviation
- For regularization, this helps all the features be penalized in the same units
(e.g. convert both centimeters and kilometers to z-scores)

REGULARIZATION EXAMPLE: LOGISTIC REGRESSION

Example: Logistic Regression

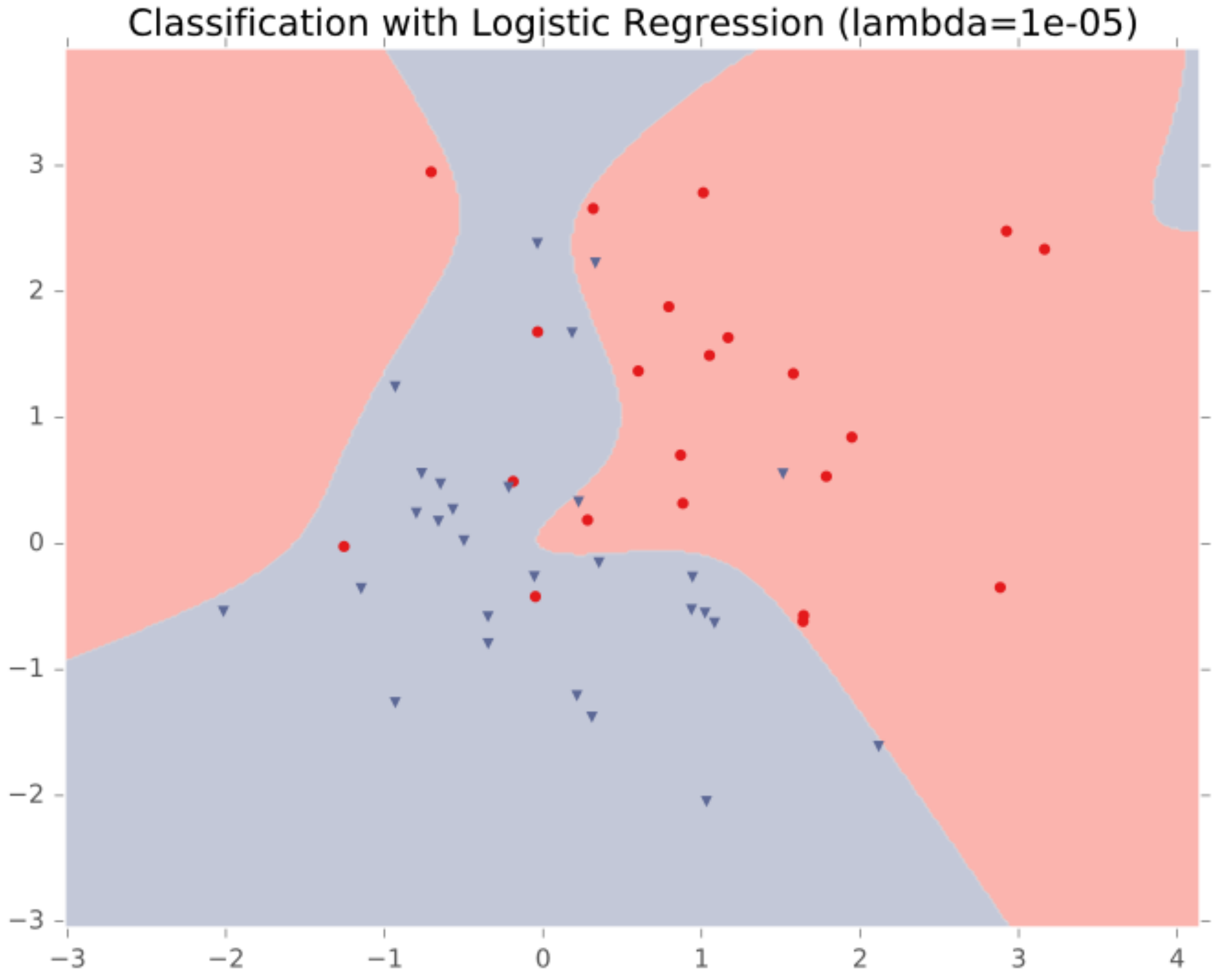
Training Data

Test Data

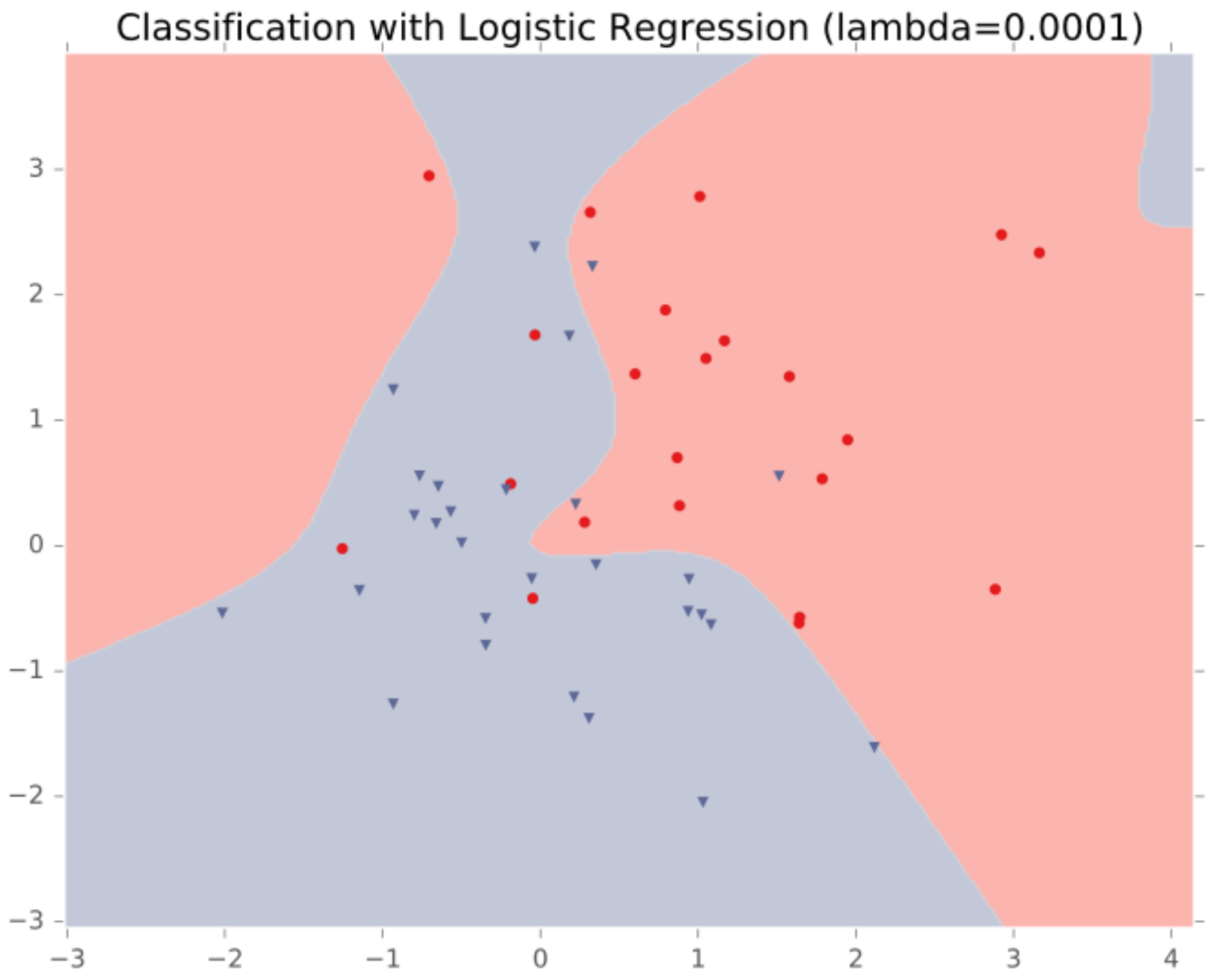


- For this example, we construct **nonlinear features** (i.e. feature engineering)
- Specifically, we add **polynomials up to order 9** of the two original features x_1 and x_2
- Thus our classifier is **linear** in the **high-dimensional feature space**, but the decision boundary is **nonlinear** when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

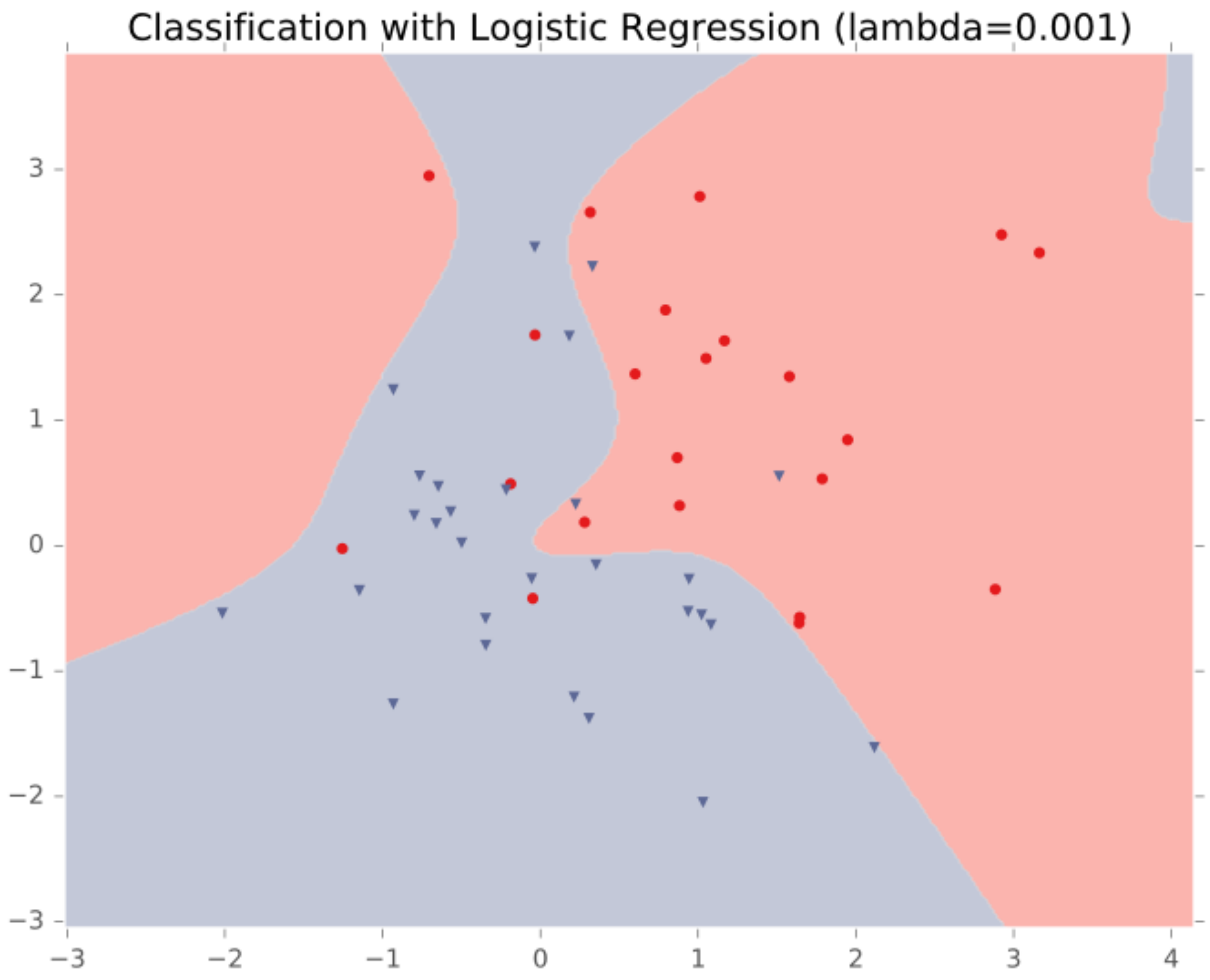
Example: Logistic Regression



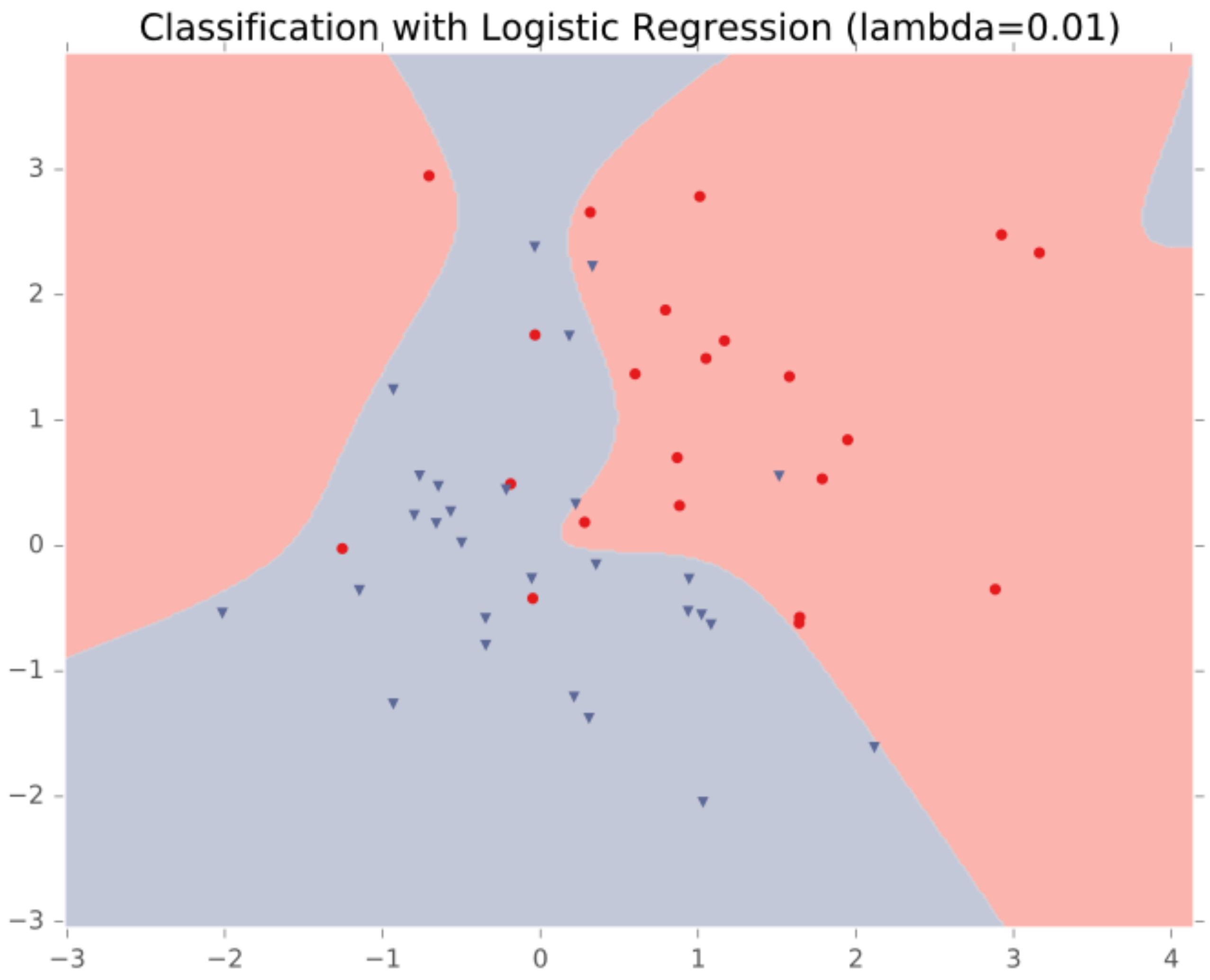
Example: Logistic Regression



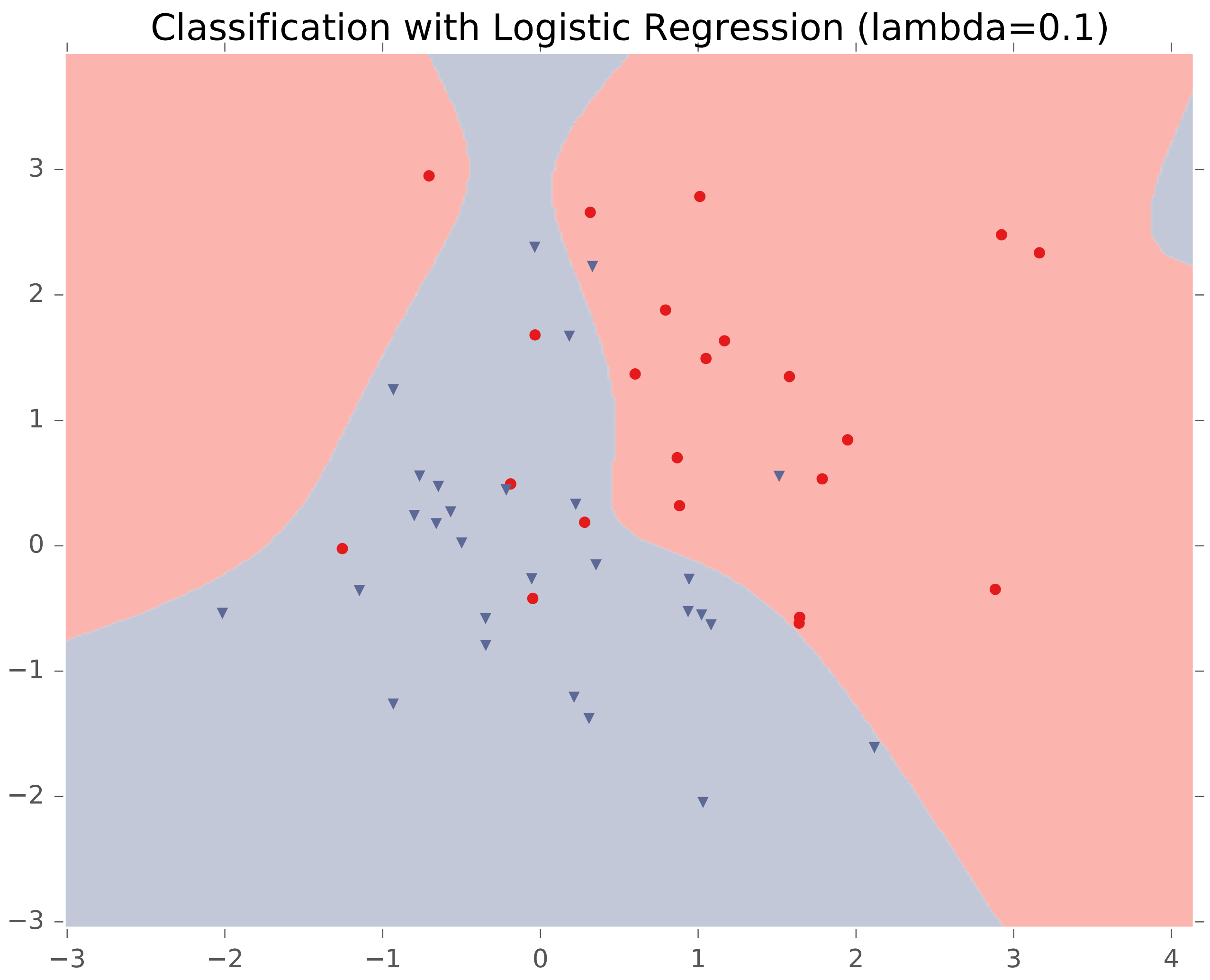
Example: Logistic Regression



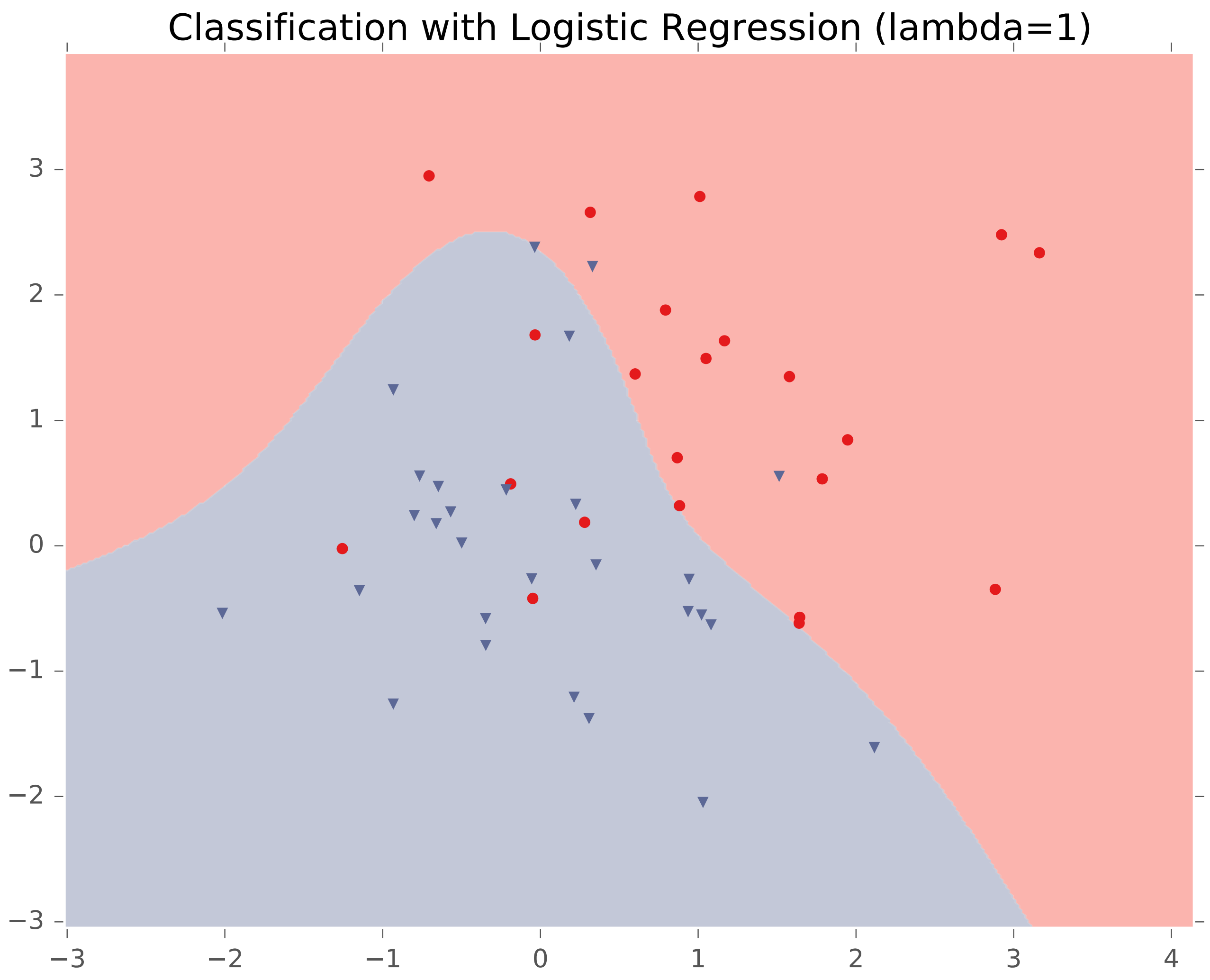
Example: Logistic Regression



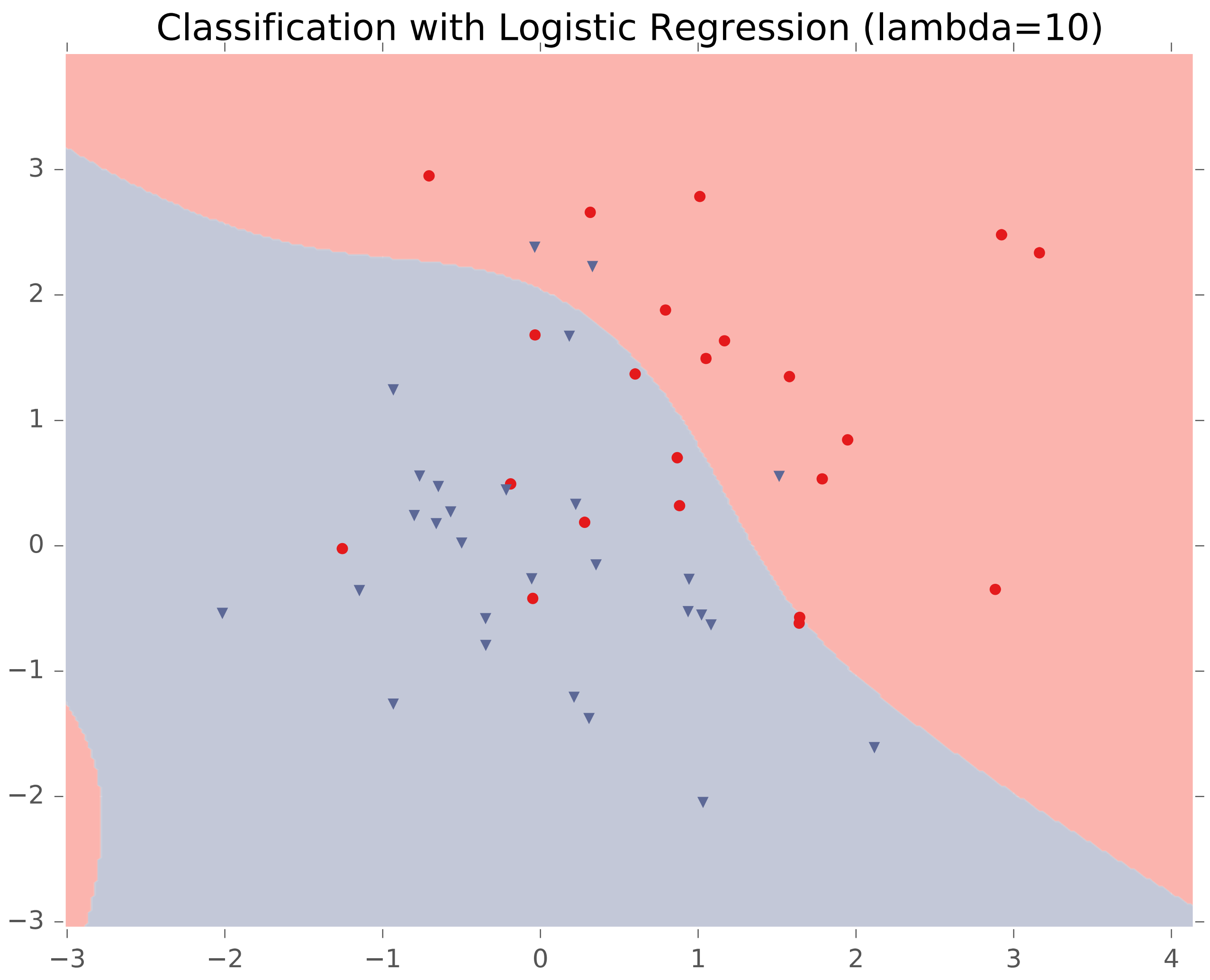
Example: Logistic Regression



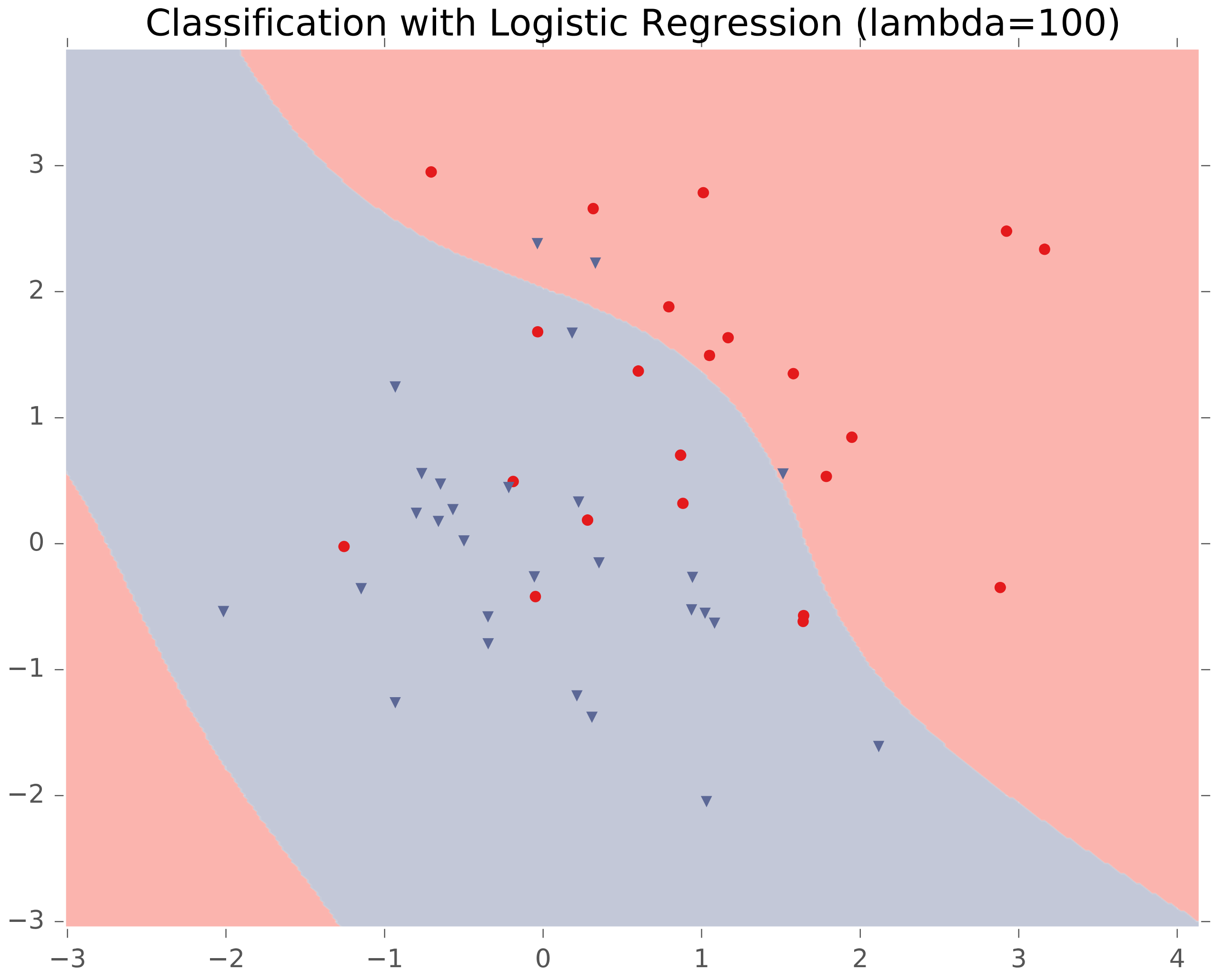
Example: Logistic Regression



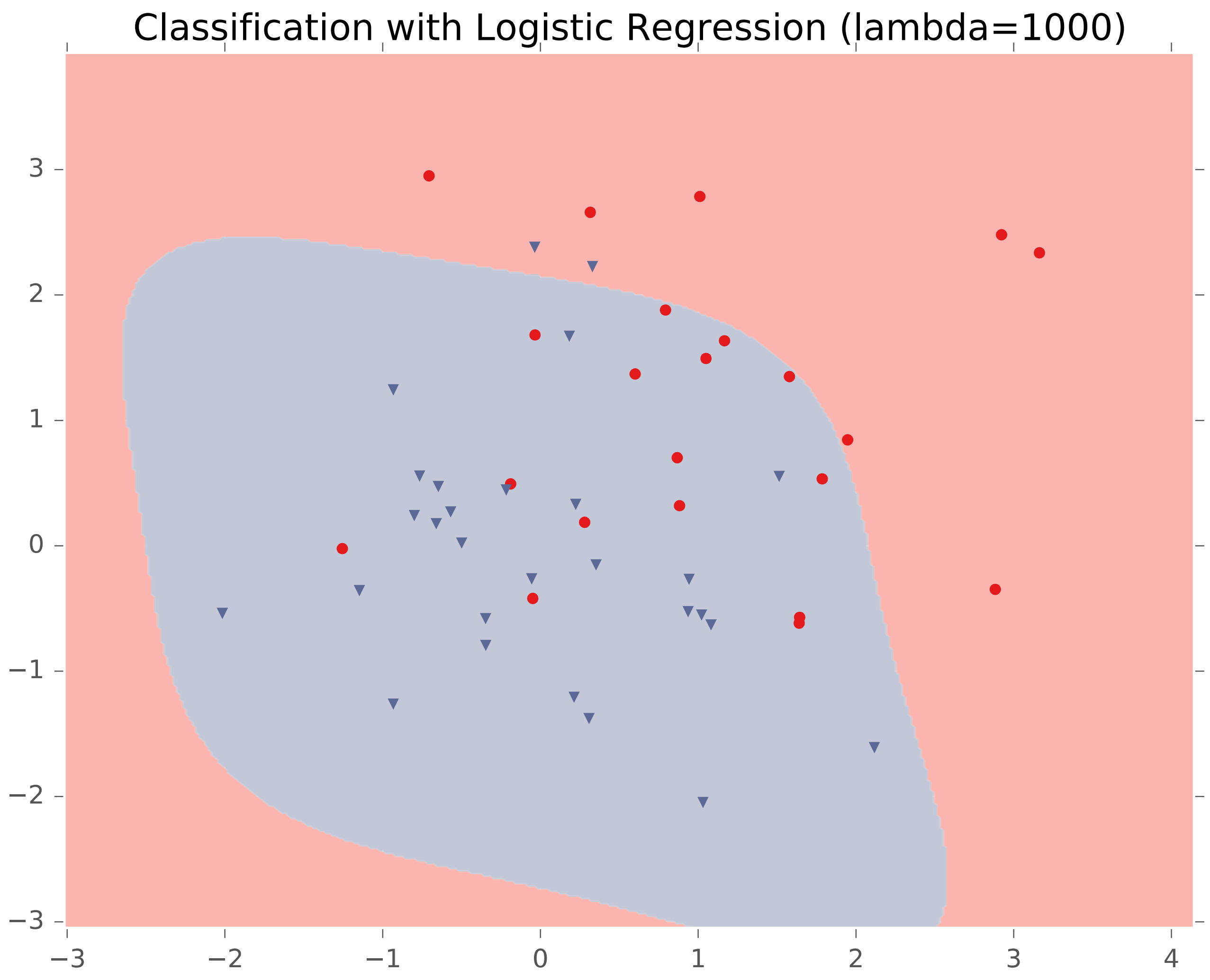
Example: Logistic Regression



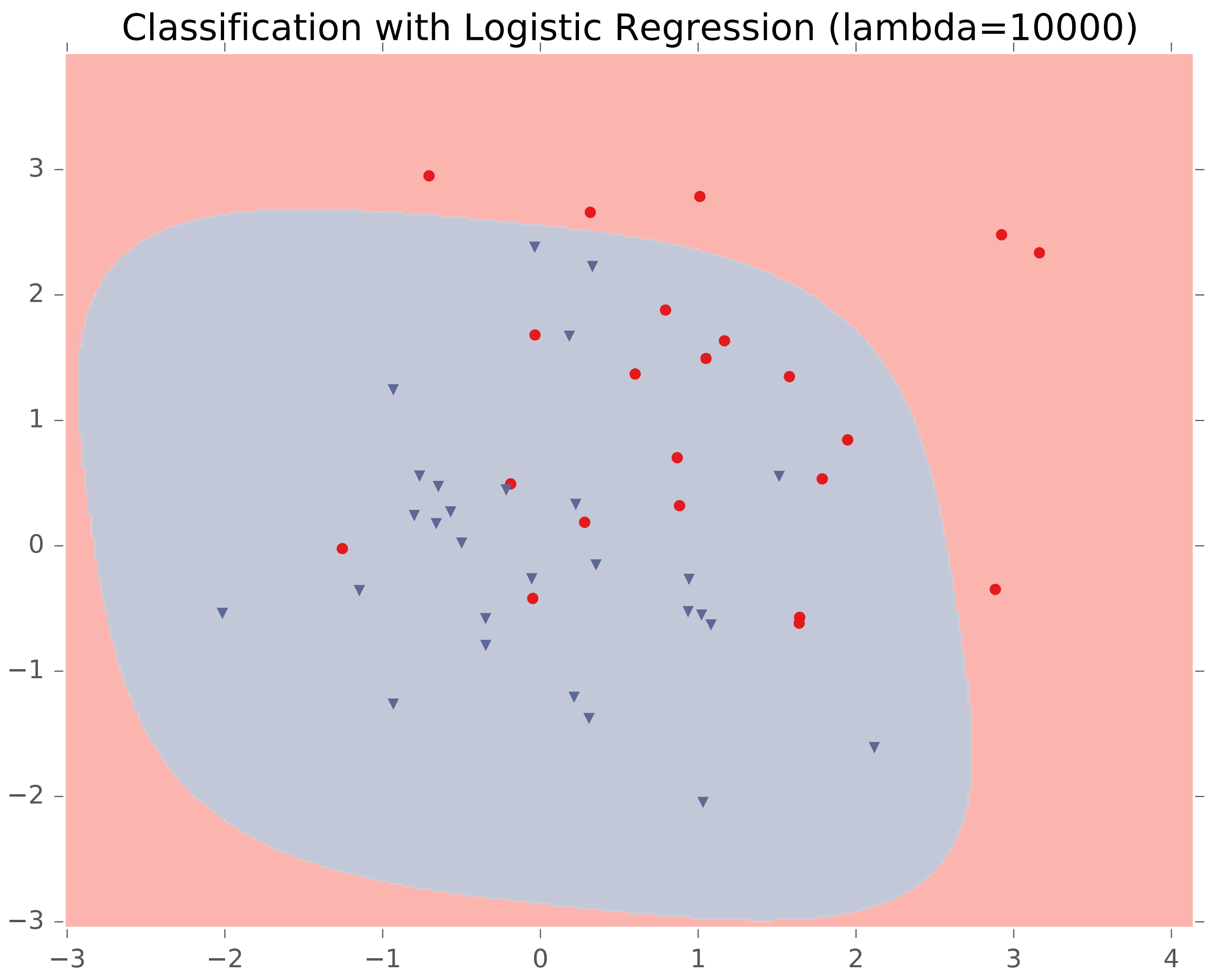
Example: Logistic Regression



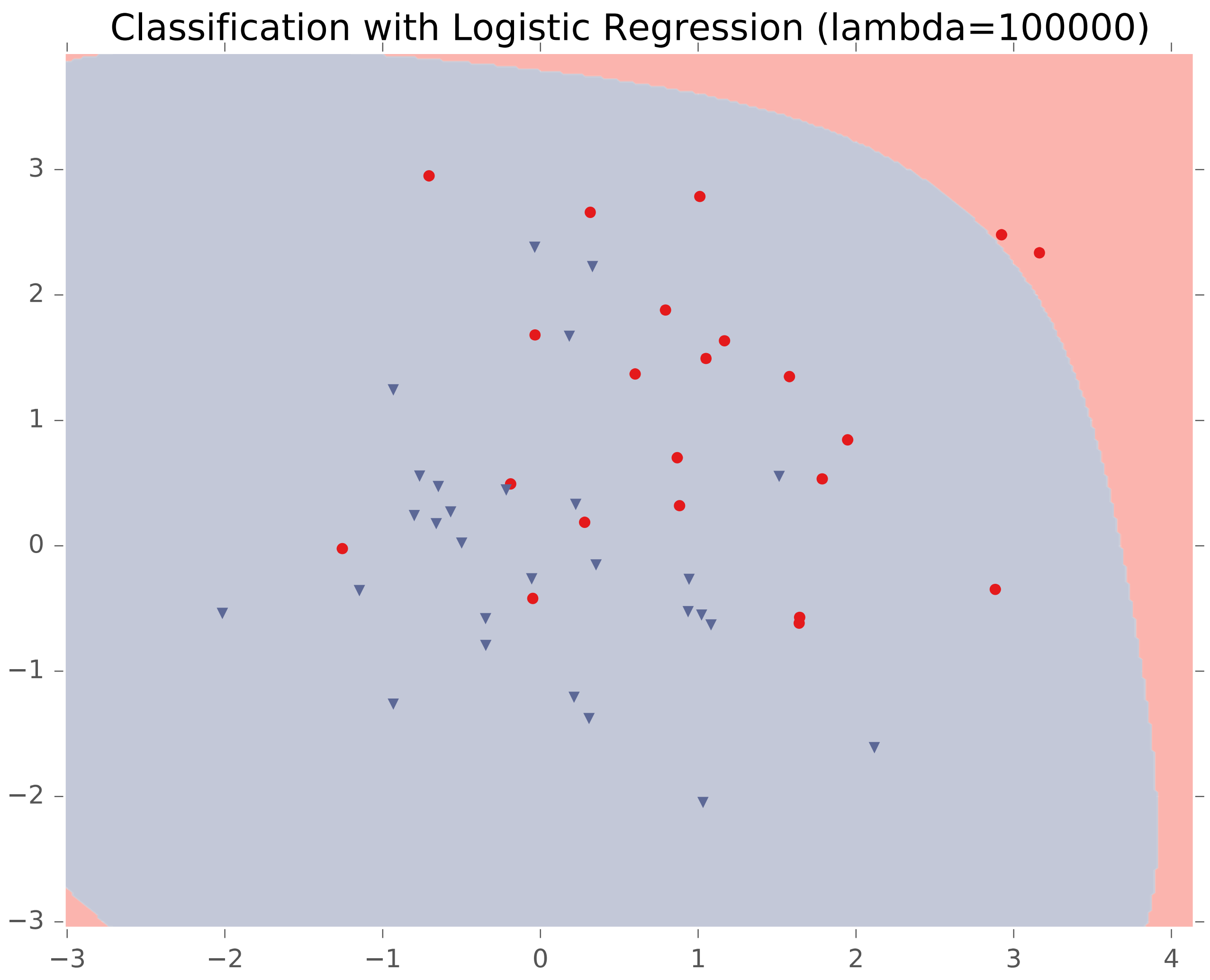
Example: Logistic Regression



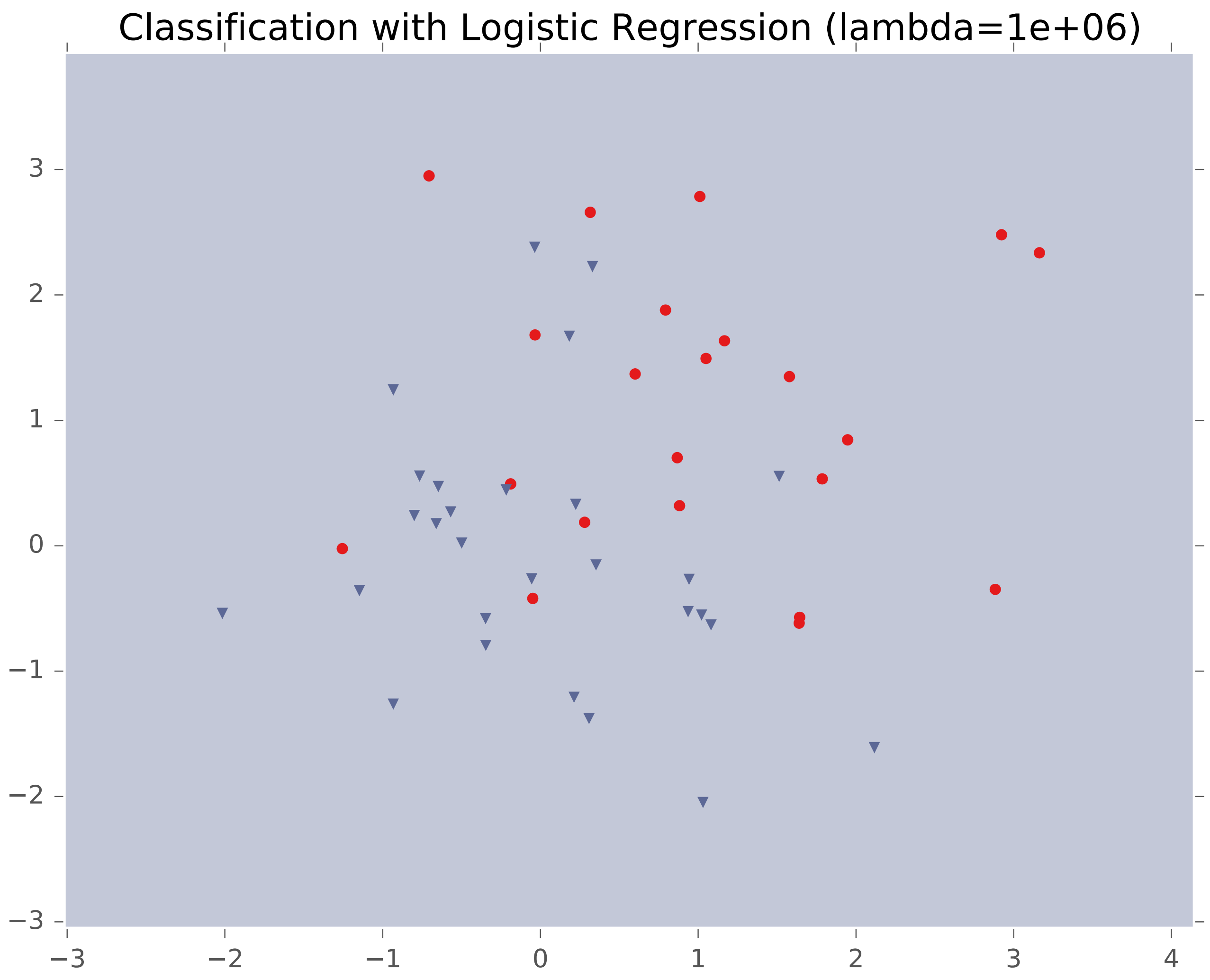
Example: Logistic Regression



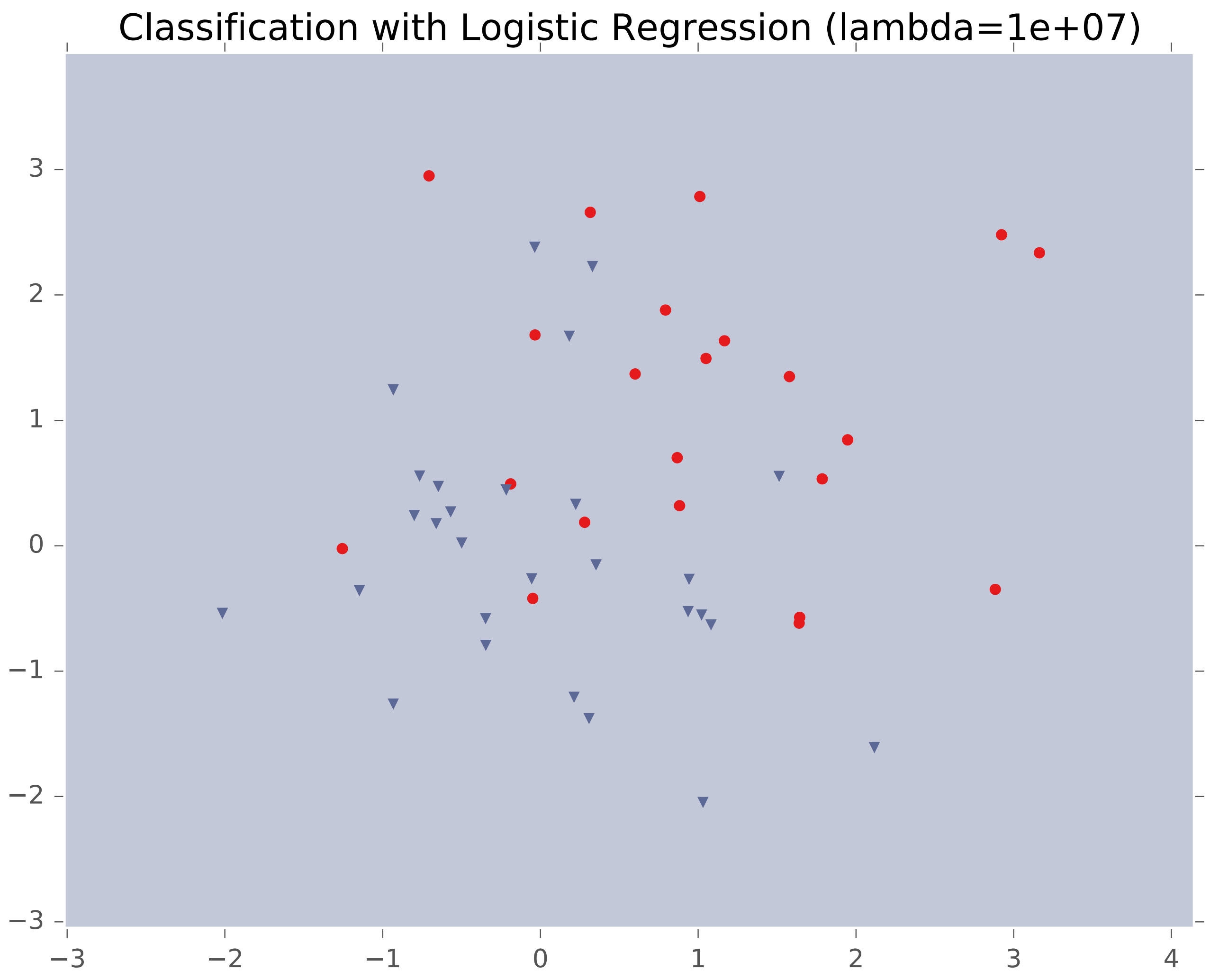
Example: Logistic Regression



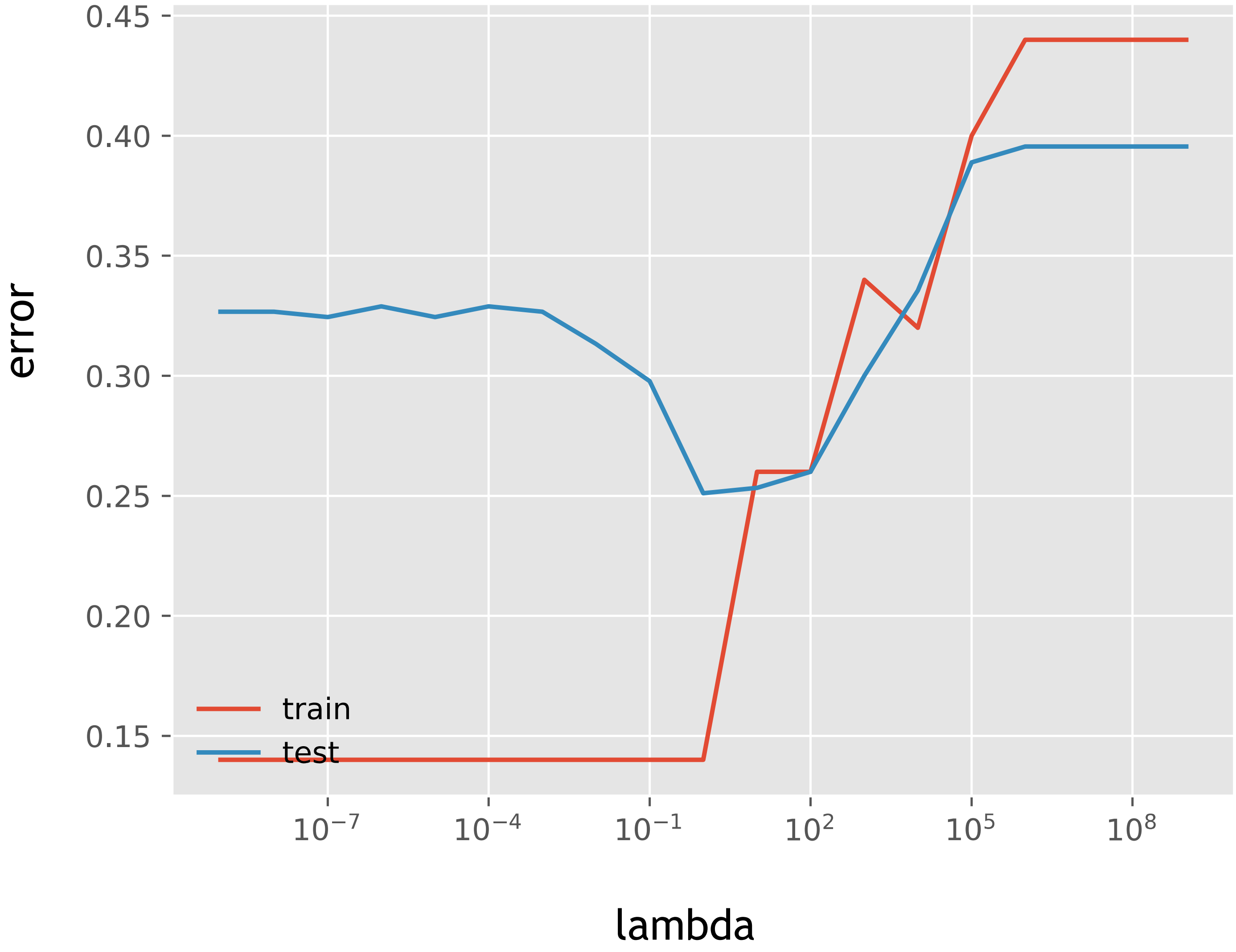
Example: Logistic Regression



Example: Logistic Regression



Example: Logistic Regression



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