10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

(Linear Models)
+ Feature Engineering
+ Regularization

Matt Gormley
Lecture 10
Sep. 30, 2022

Q&A

Q: I think you graded seven different questions incorrectly. Should I put them all in on Gradescope regrade request and submit that?

A: Please, no. I'd encourage you to watch this tutorial video from Gradescope so you know how to use this important tool.

https://help.gradescope.com/article/8hchz9h8wh-student-regrade-request

Reminders

- Practice Problems: Exam 1
- Exam 1
  – Tue, Oct 4, 6:30pm – 8:30pm
  – see Piazza for details
- Homework 4: Logistic Regression
  – Out: Tue, Oct 4
  – Due: Thu, Oct 13 at 11:59pm

Linear Models

PERCEPTRON, LINEAR REGRESSION, AND LOGISTIC REGRESSION

Why is it not "Logistic Classification"?

Whiteboard

- Conceptual Change: 2D classification in 3D
- Why is it called Logistic *Regression* and not Logistic *Classification*?

Matching Game

Q1

Question:

Match the Algorithm to its Update Rule

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

½ learning rate

Answer:

5% I=toxic

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

SGD for Logistic Regression

Q2

Question:

Which of the following is a correct description of SGD for Logistic Regression?

Answer:

At each step (i.e. iteration) of SGD for Logistic Regression we…

A. (1) compute the gradient of the log-likelihood for all examples (2) update all the parameters using the gradient

B. ~~(1) ask Matt for a description of SGD for Logistic Regression, (2) write it down, (3) report that answer~~ toxic

C. (1) compute the gradient of the log-likelihood for all examples (2) randomly pick an example (3) update only the parameters for that example

D. (1) randomly pick a parameter, (2) compute the partial derivative of the log-likelihood with respect to that parameter, (3) update that parameter for all examples

70% E. (1) randomly pick an example, (2) compute the gradient of the log-likelihood for that example, (3) update all the parameters using that gradient

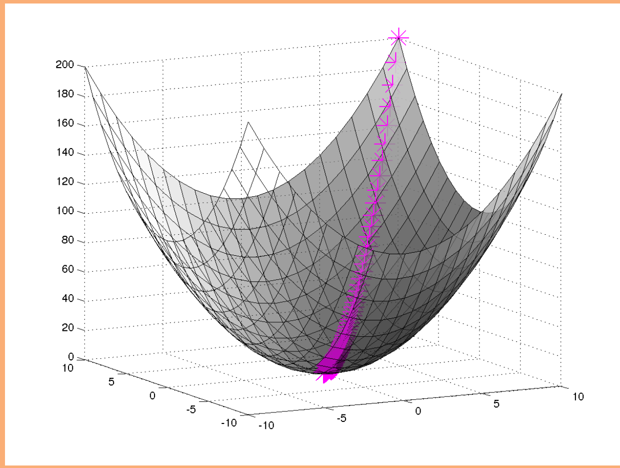
F. (1) randomly pick a parameter and an example, (2) compute the gradient of the log-likelihood for that example with respect to that parameter, (3) update that parameter using that gradient

Recall...

Gradient Descent

Algorithm 1 Gradient Descent

1: **procedure** $\text{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)})$
2: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$ **while** not converged **do**
4: $\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: $\quad$ **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).

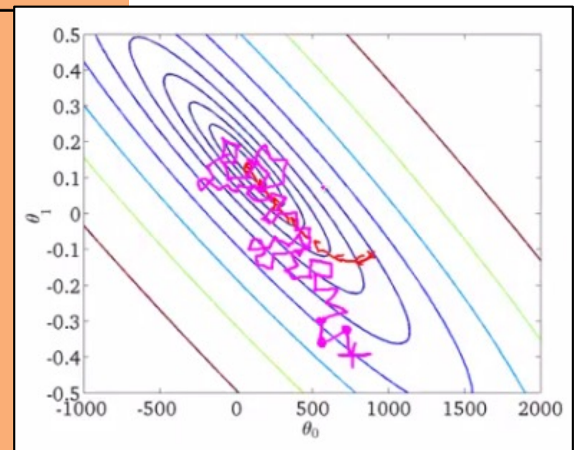
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M} J(\boldsymbol{\theta}) \end{bmatrix}$$

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

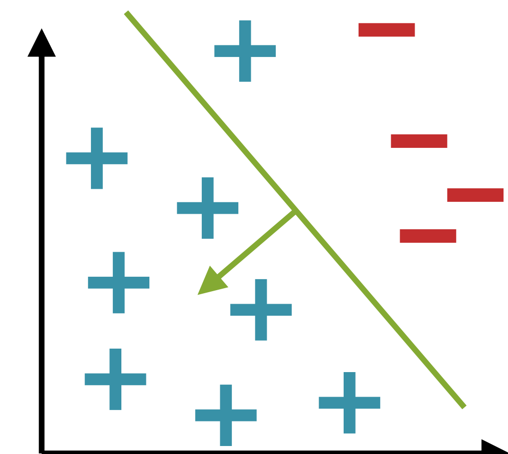
Q3

A = toxic   B = True
25%

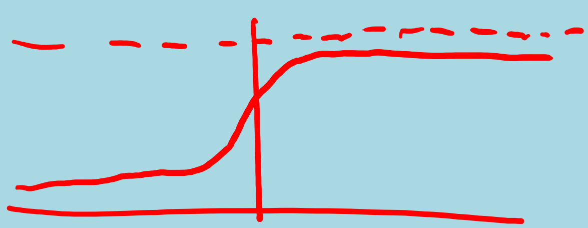
C = False
75%

Question:

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:



BAYES OPTIMAL CLASSIFIER

Bayes Optimal Classifier

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? $\in \{0,1\}$

Answer:

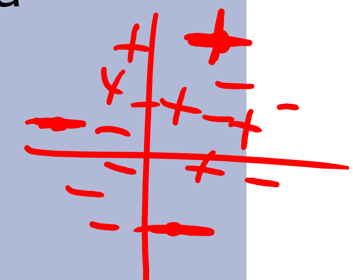
You'd use the *Bayes optimal classifier*!

ur

approximates $c^*(\mathbf{x})$

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

Whiteboard

- Bayes Optimal Classifier
- Reducible / irreducible error
- Ex: Bayes Optimal Classifier for 0/1 Loss

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 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 $\forall s$

Logistic Regression Objectives

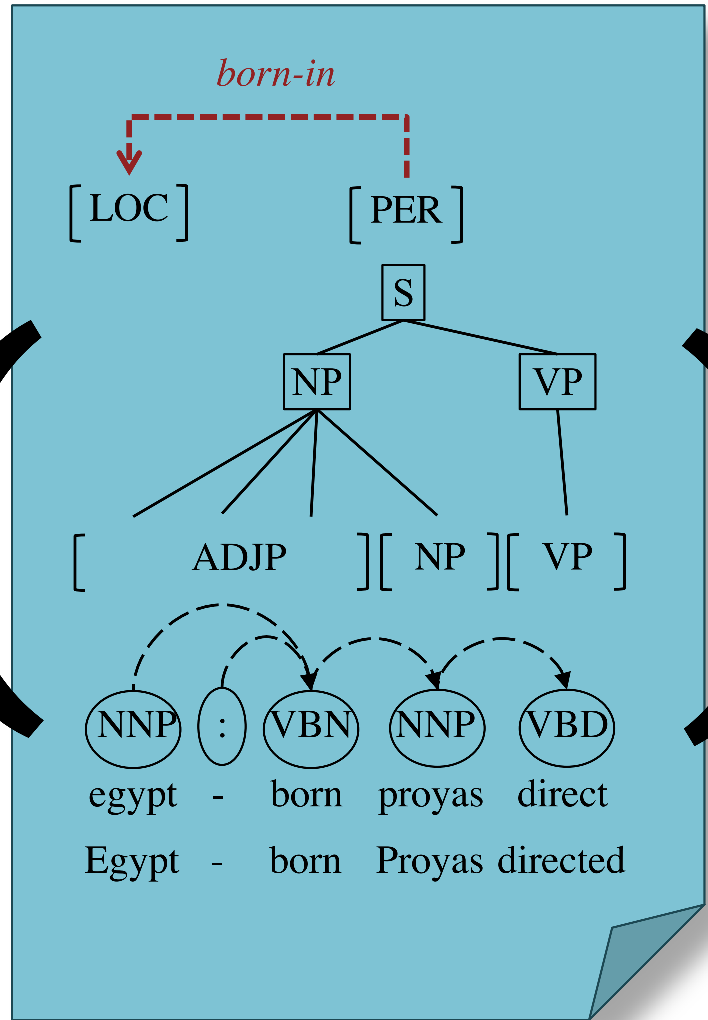
You should be able to…

- 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 or multiclass classification
- Prove that the decision boundary of binary logistic regression is linear
- For linear regression, show that the parameters which minimize squared error are equivalent to those that maximize conditional likelihood

FEATURE ENGINEERING

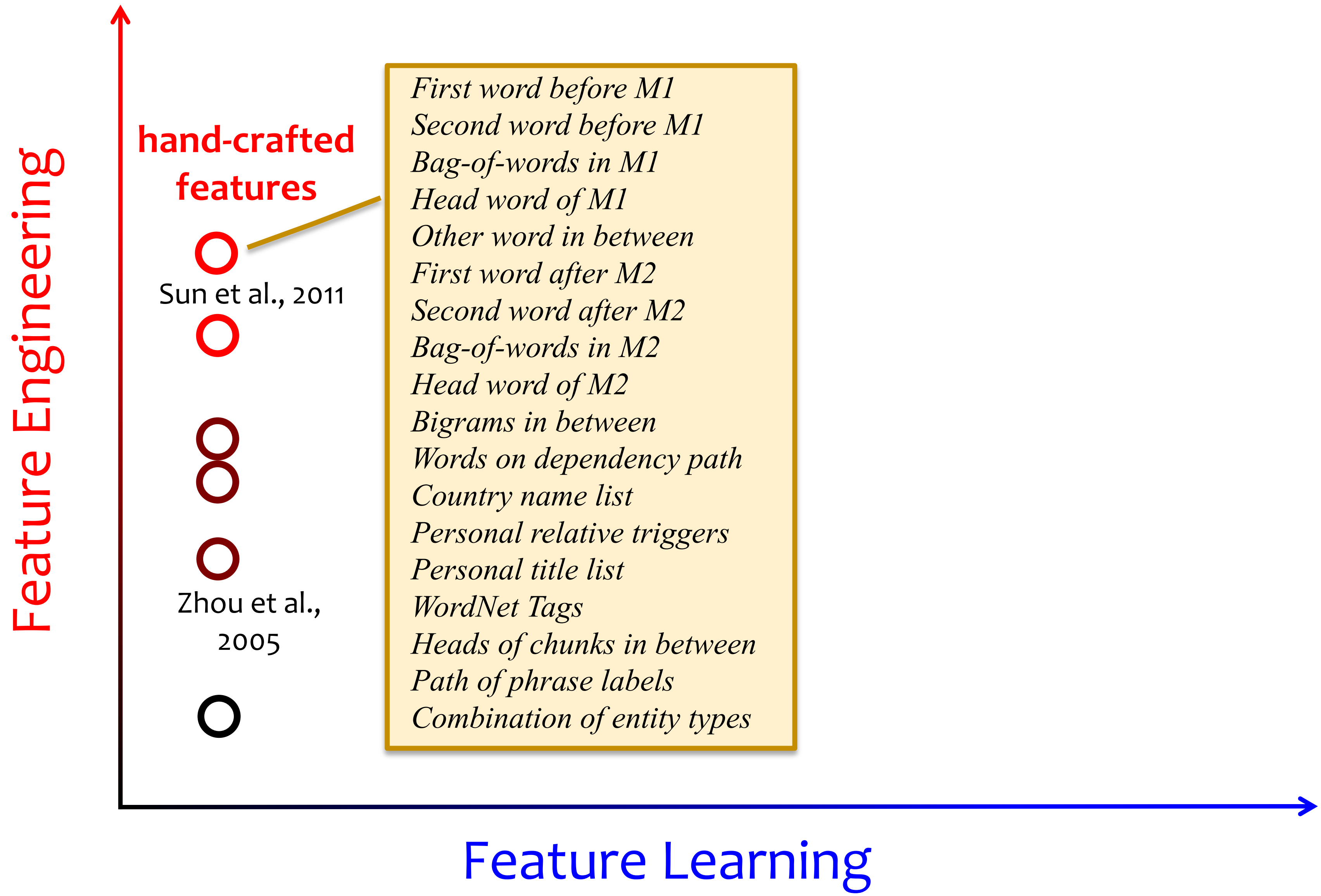
Handcrafted Features

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

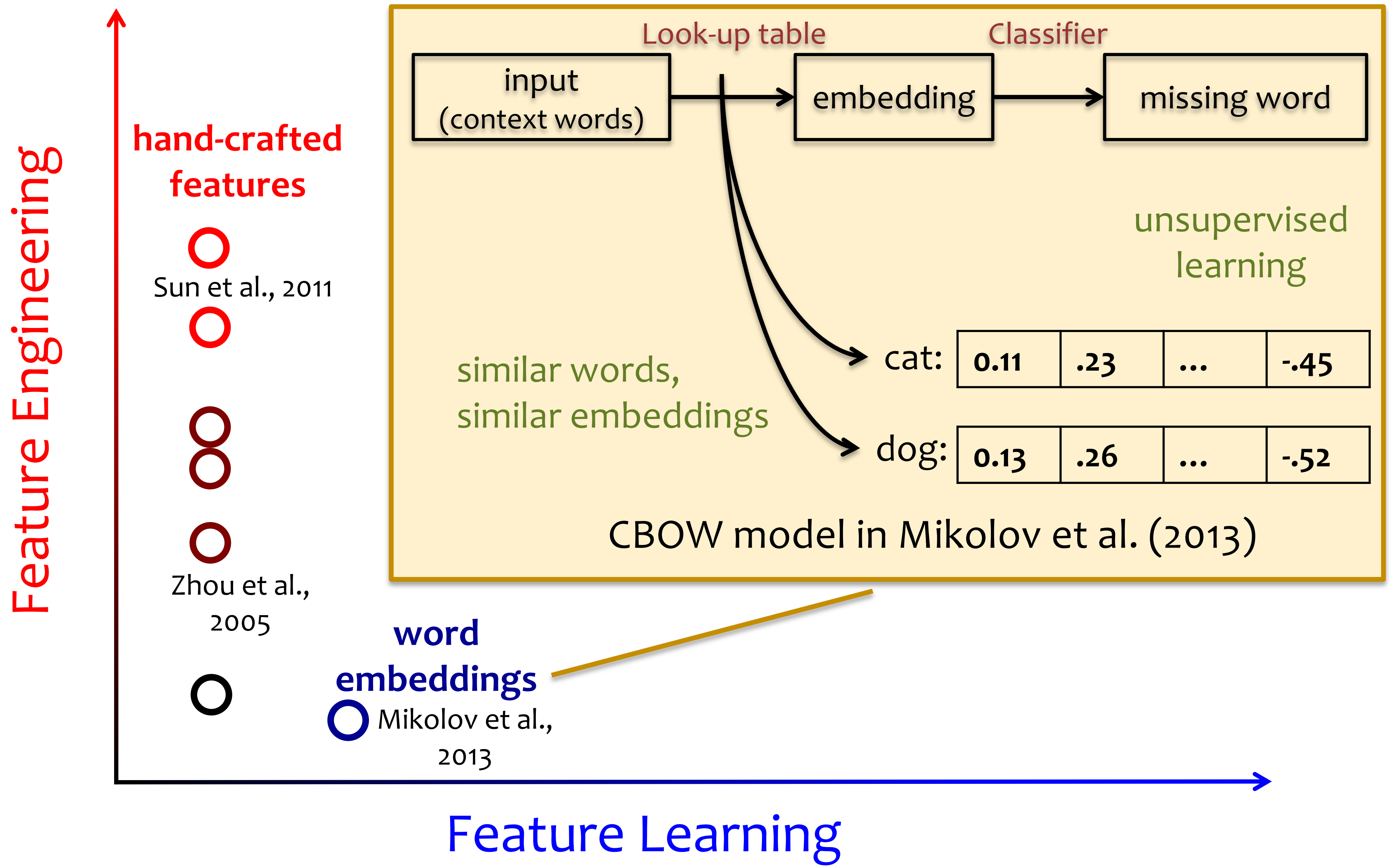


$\Big))$

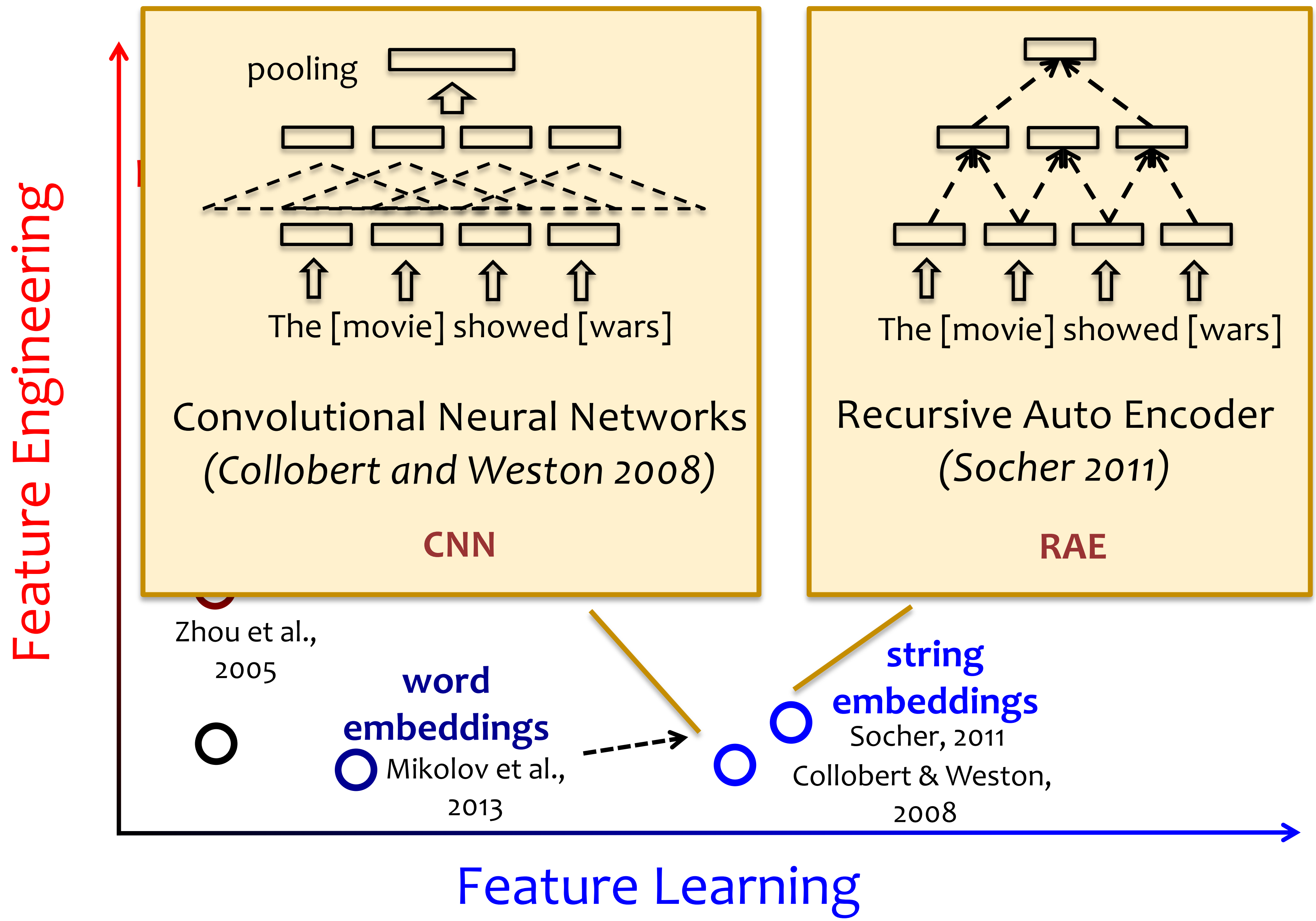
Where do features come from?



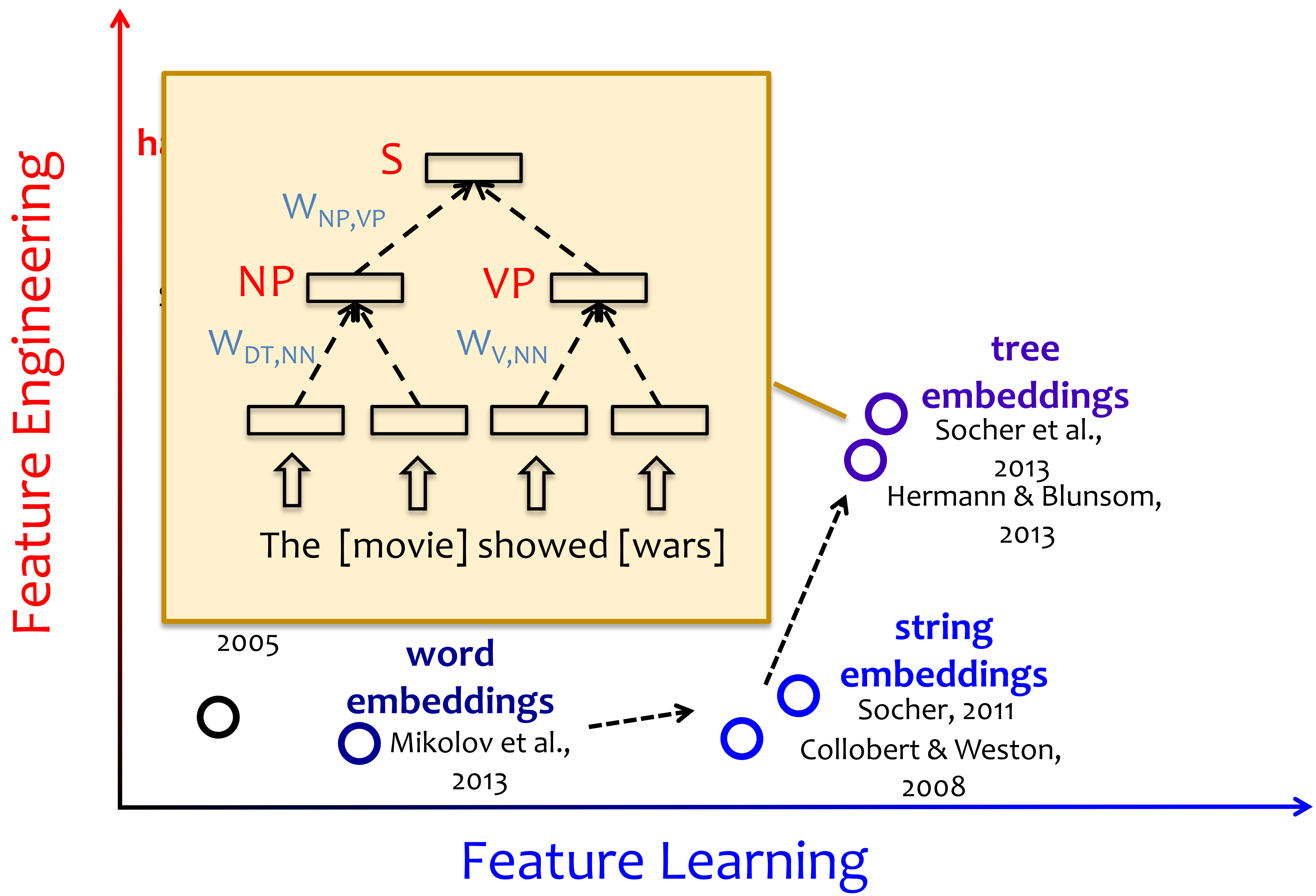
Where do features come from?



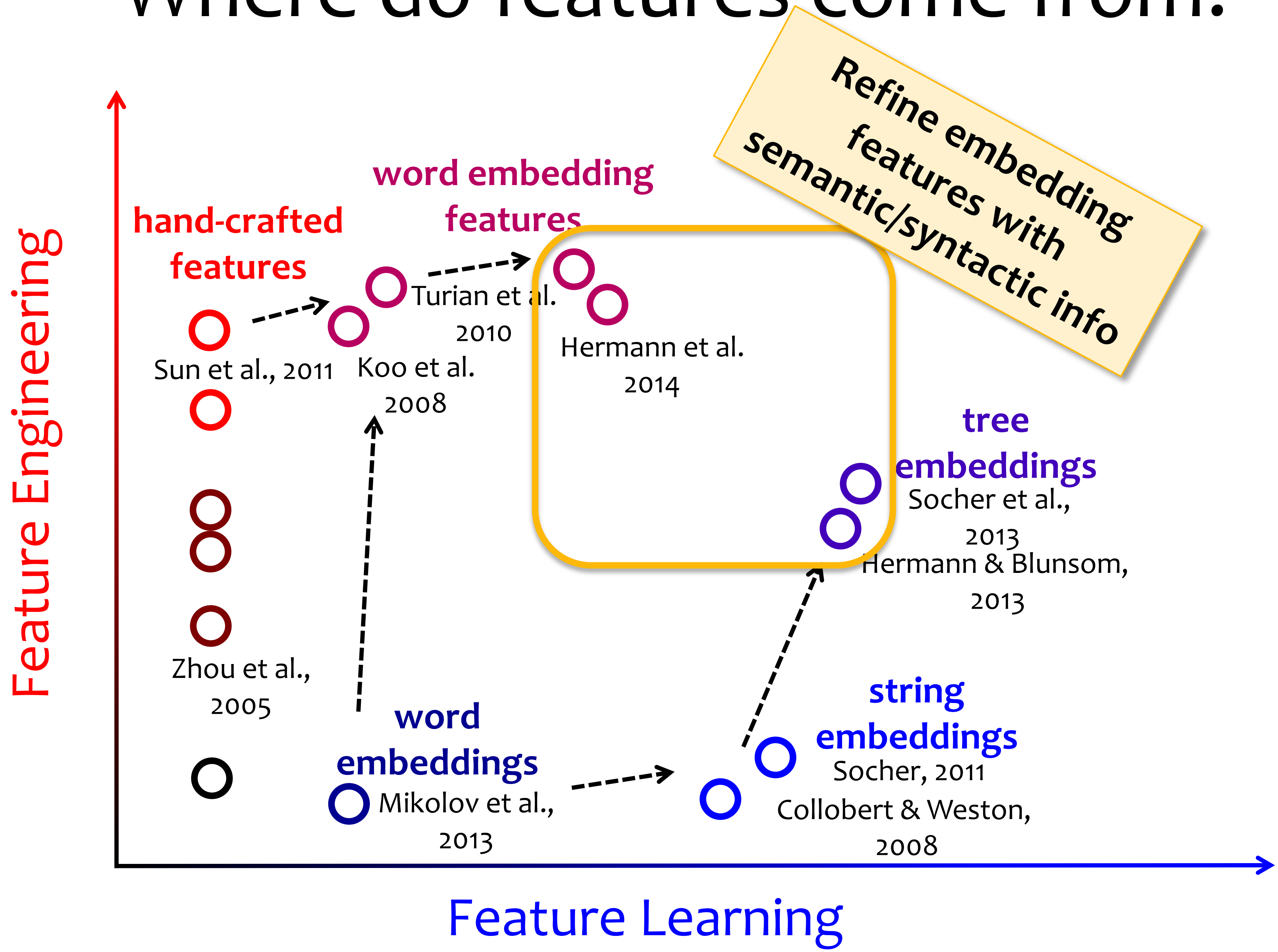
Where do features come from?



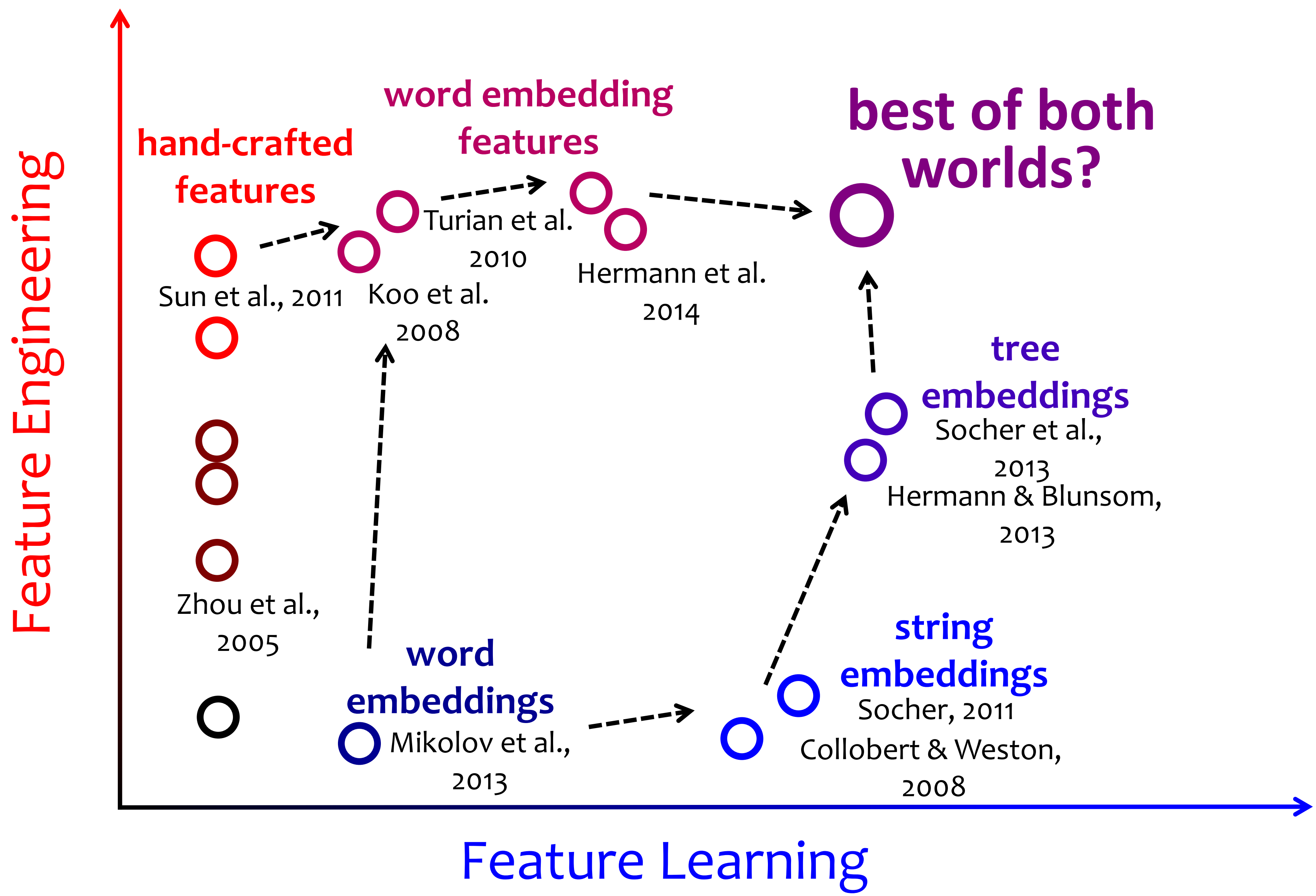
Where do features come from?



Where do features come from?



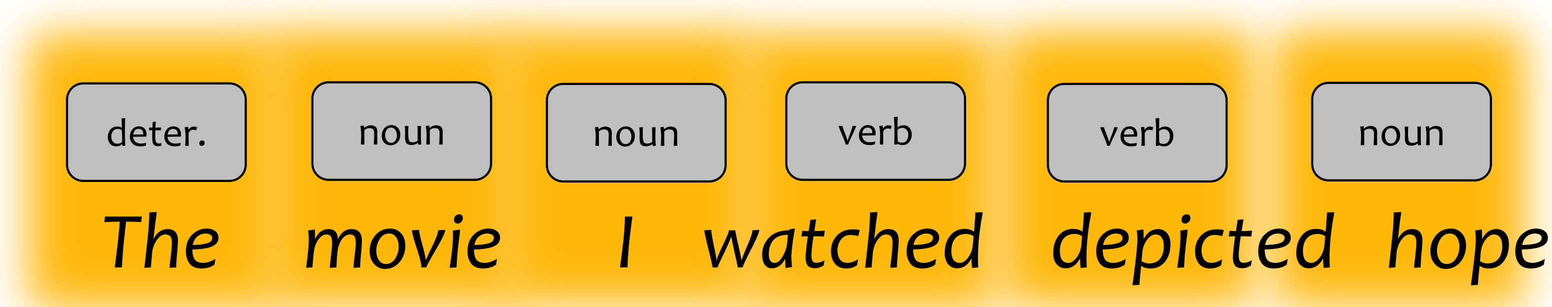
Where do features come from?



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(wi)`	1	0	1	0	0	0
`endswith(wi,"e")`	1	1	0	0	0	1
`endswith(wi,"d")`	0	0	0	1	1	0
`endswith(wi,"ed")`	0	0	0	1	1	0
`wi == "aardvark"`	0	0	0	0	0	0
`wi == "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_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.

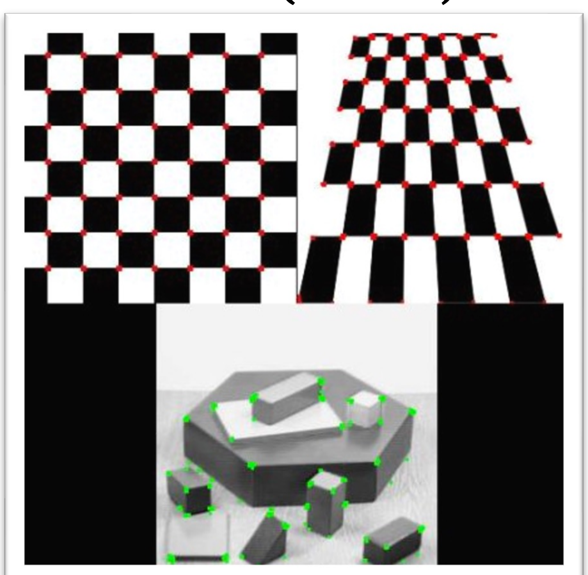
Model	Feature Templates	# Feats	Sent. Acc.	Token Acc.	Unk. Acc.
3GramMemm	See text	248,798	52.07%	96.92%	88.99%
NAACL 2003	See text and [1]	460,552	55.31%	97.15%	88.61%
Replication	See text and [1]	460,551	55.62%	97.18%	88.92%
Replication′	+rareFeatureThresh = 5	482,364	55.67%	97.19%	88.96%
5w	$+\langle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

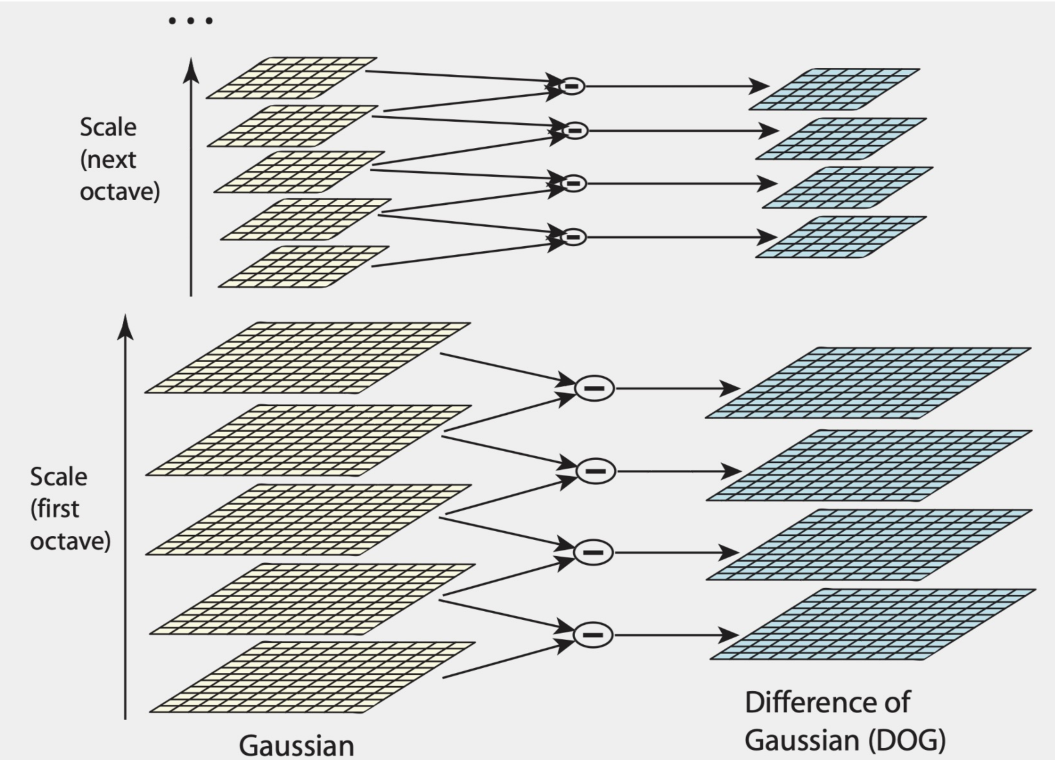


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled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

Figure from Lowe (1999) and Lowe (2004)

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: $\mathbf{x} \in \mathbb{R}^M$ where $M' > M$ (usually)
  - new input: $\mathbf{x}' \in \mathbb{R}^{M'}$
  - 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:

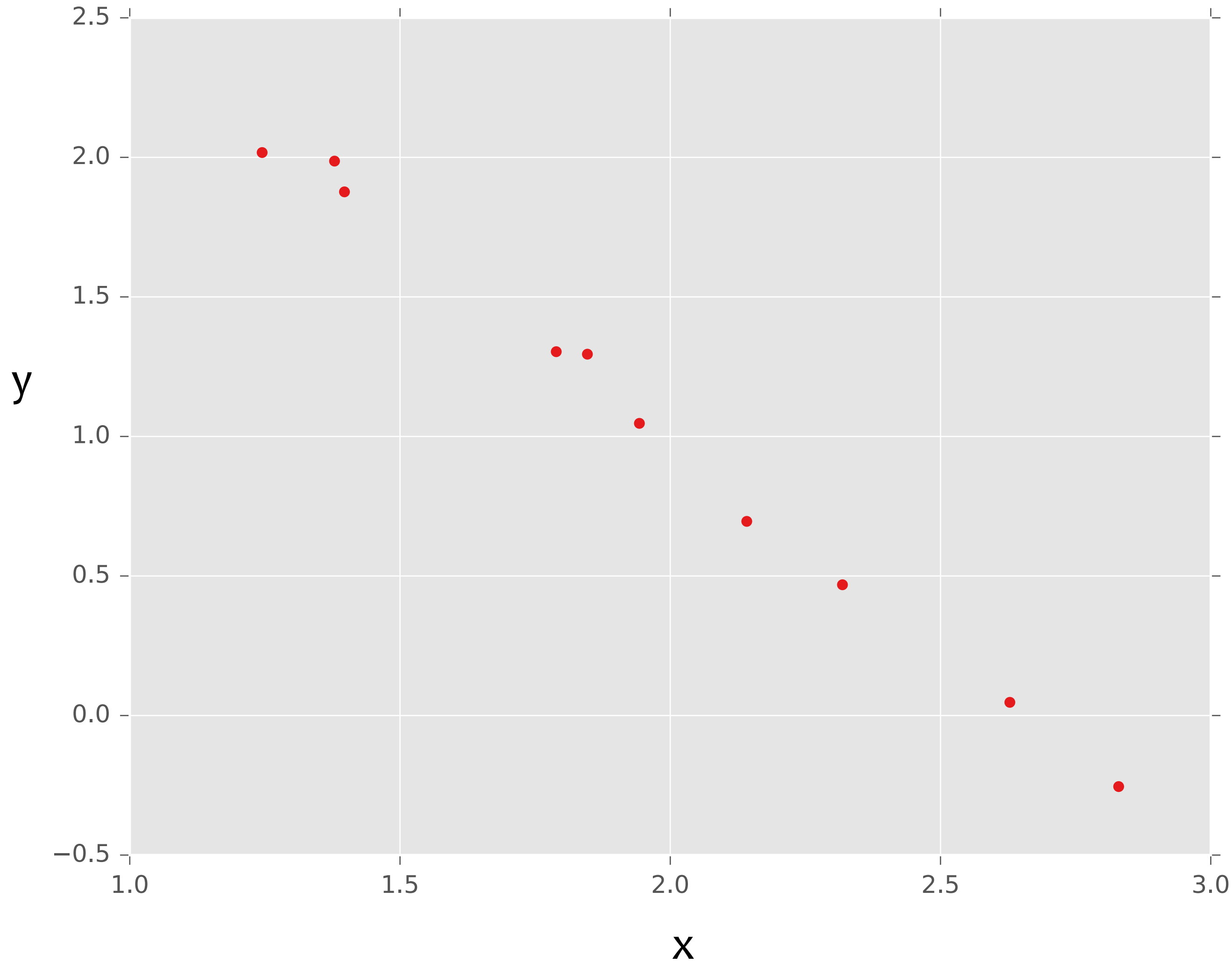
- Perceptron
- Linear regression
- Logistic regression

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

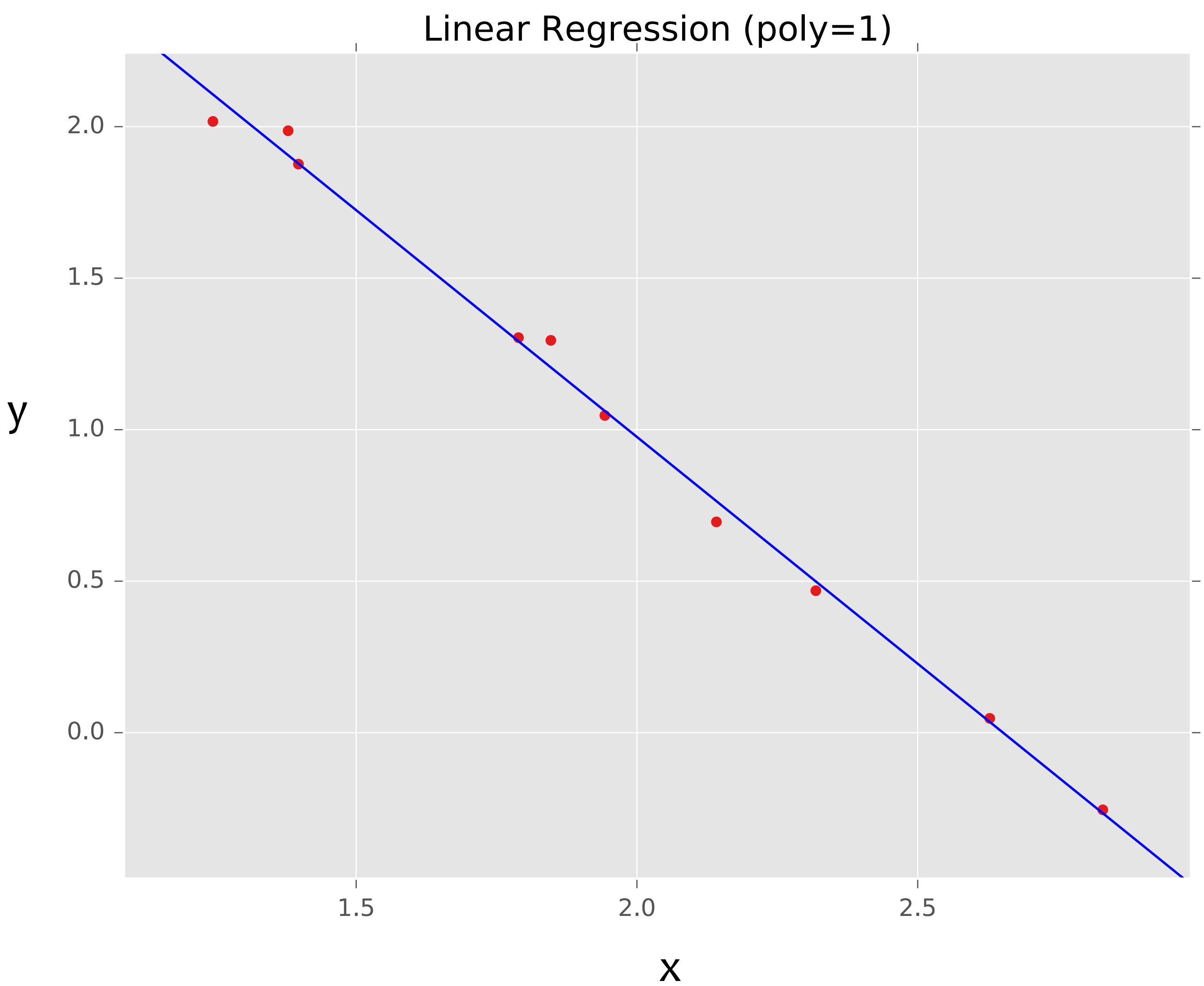


Example: Linear Regression

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

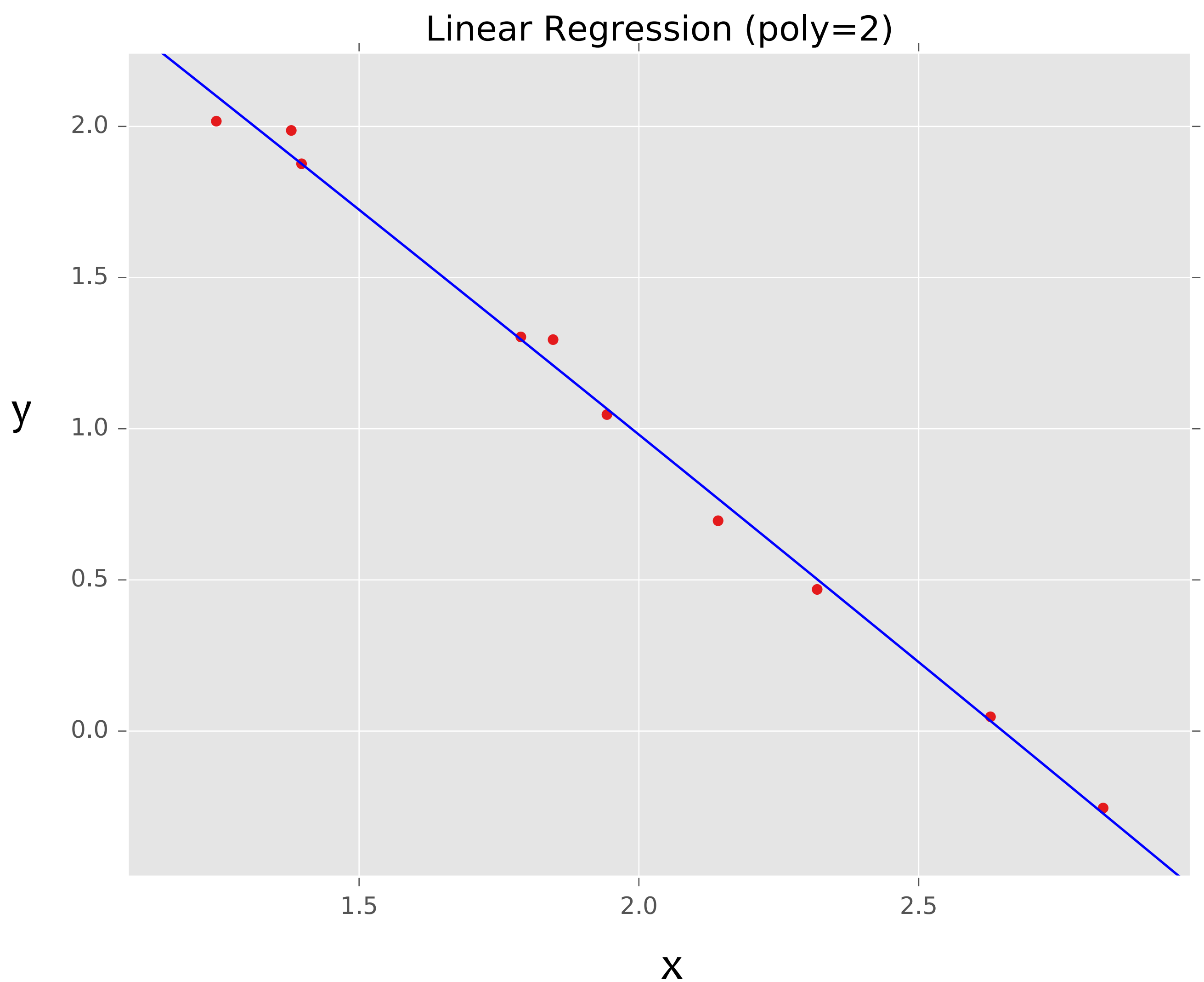


Example: Linear Regression

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

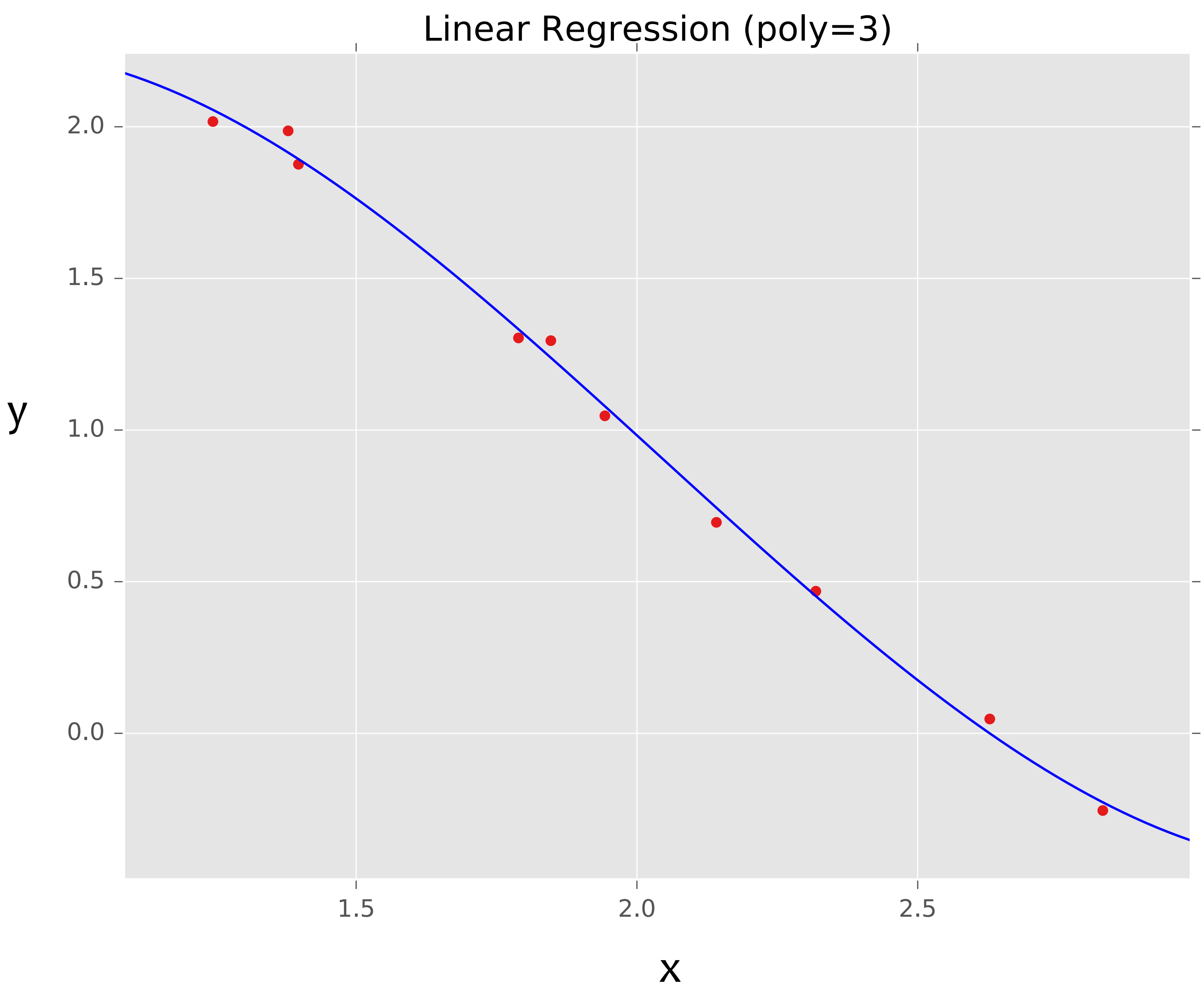


Example: Linear Regression

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

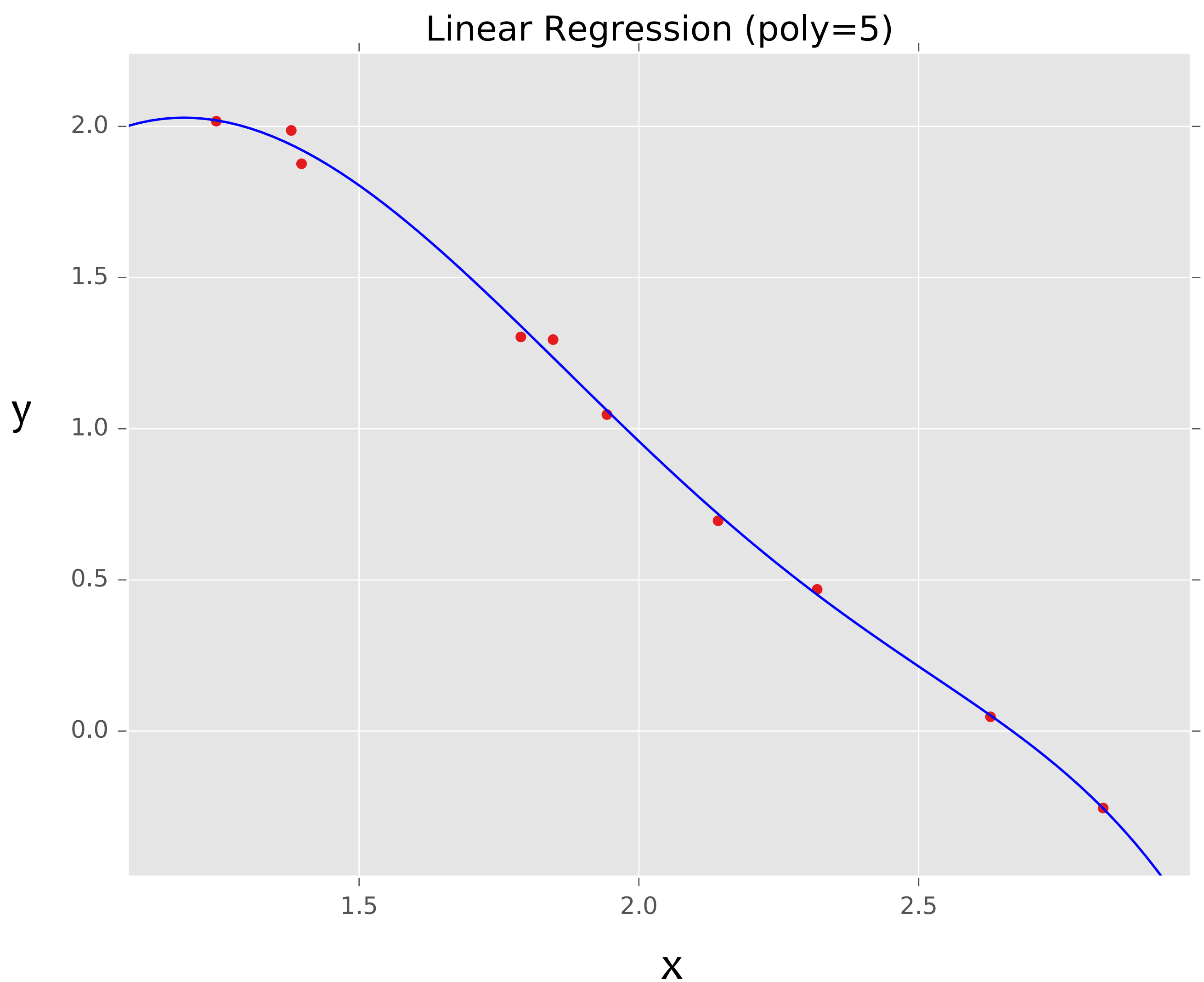


Example: Linear Regression

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

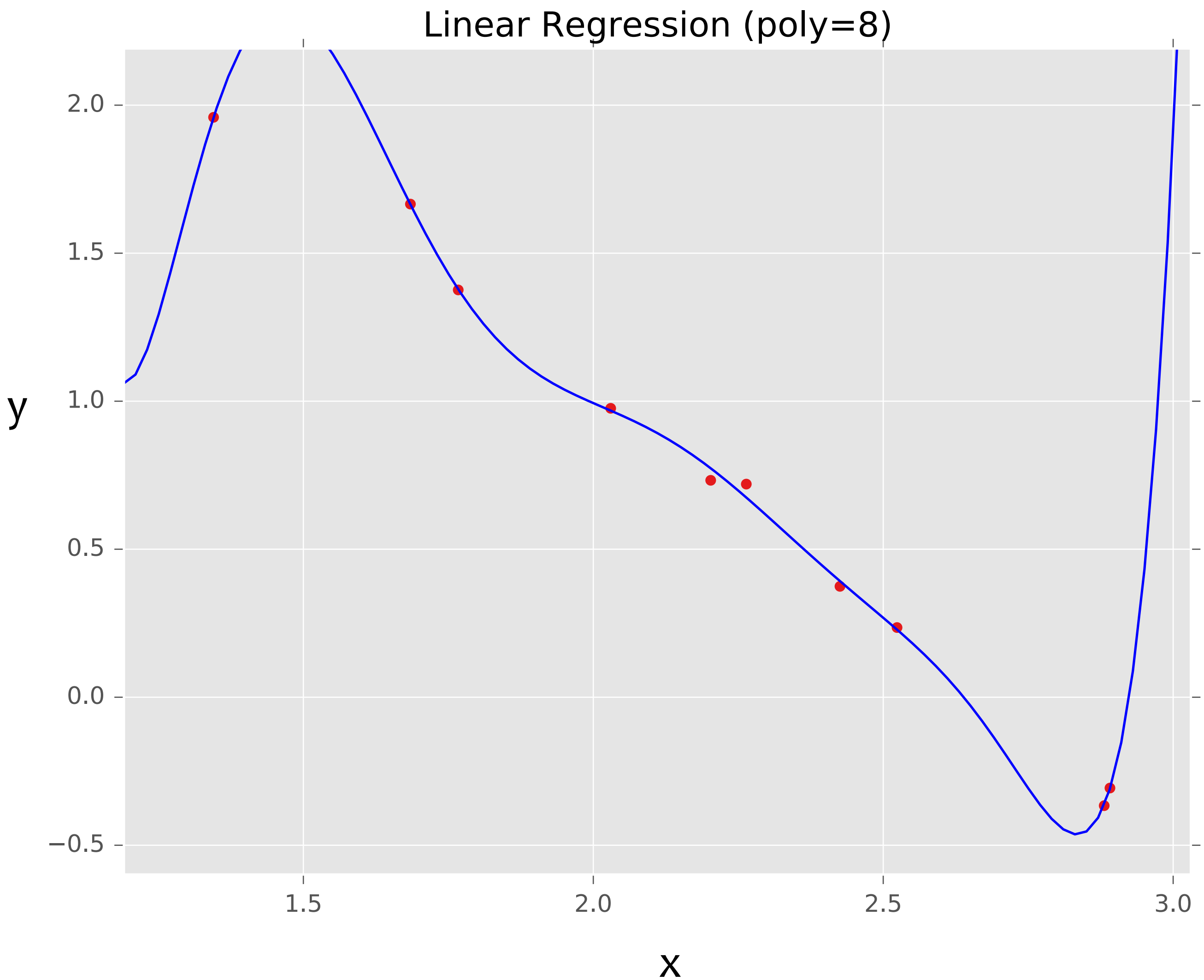


Example: Linear Regression

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

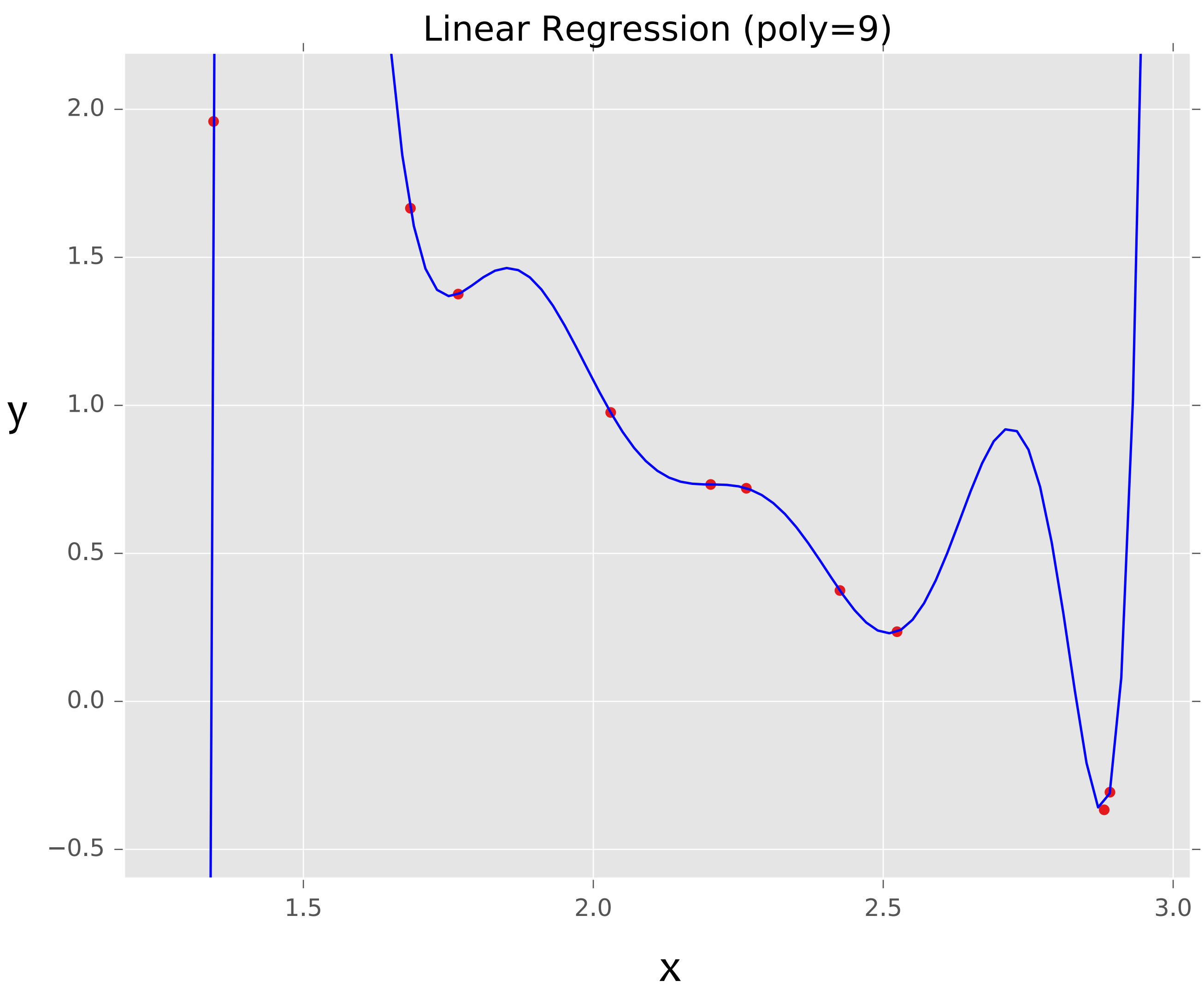


Example: Linear Regression

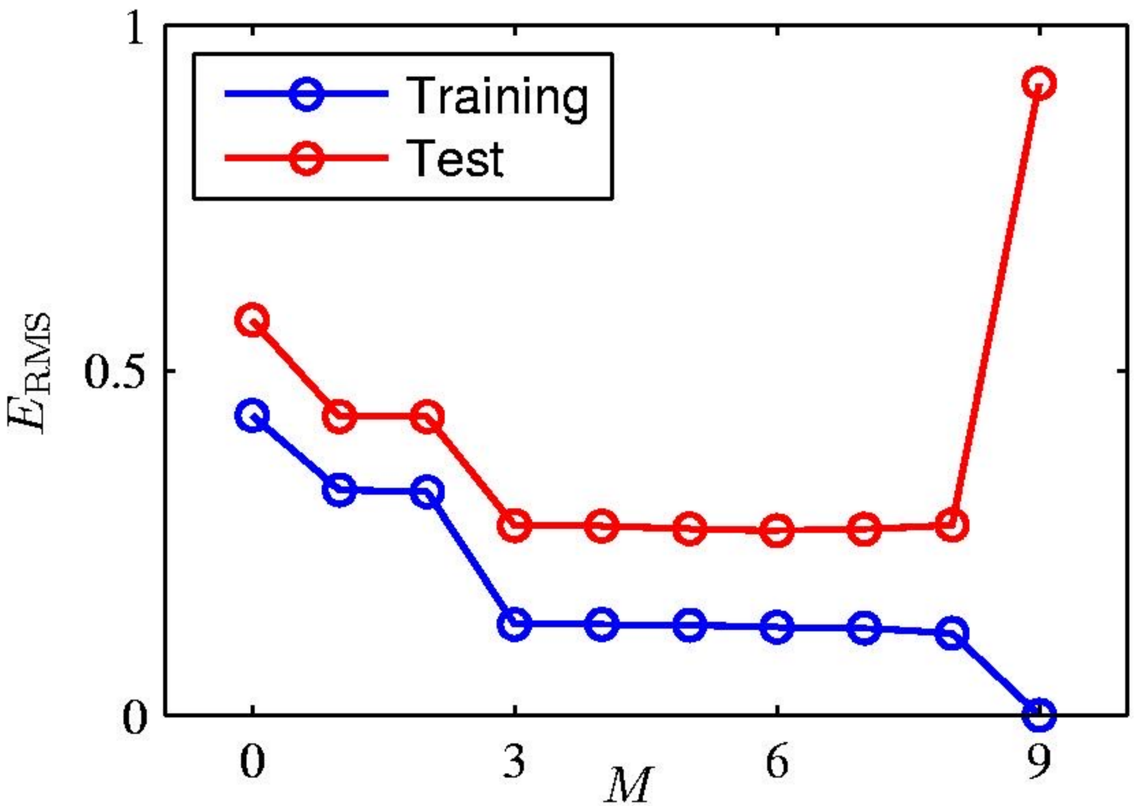
Goal: Learn y = $\mathbf{w}^T$ f($\mathbf{x}$) + b
where f(.) is a polynomial basis function

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

Slide courtesy of William Cohen

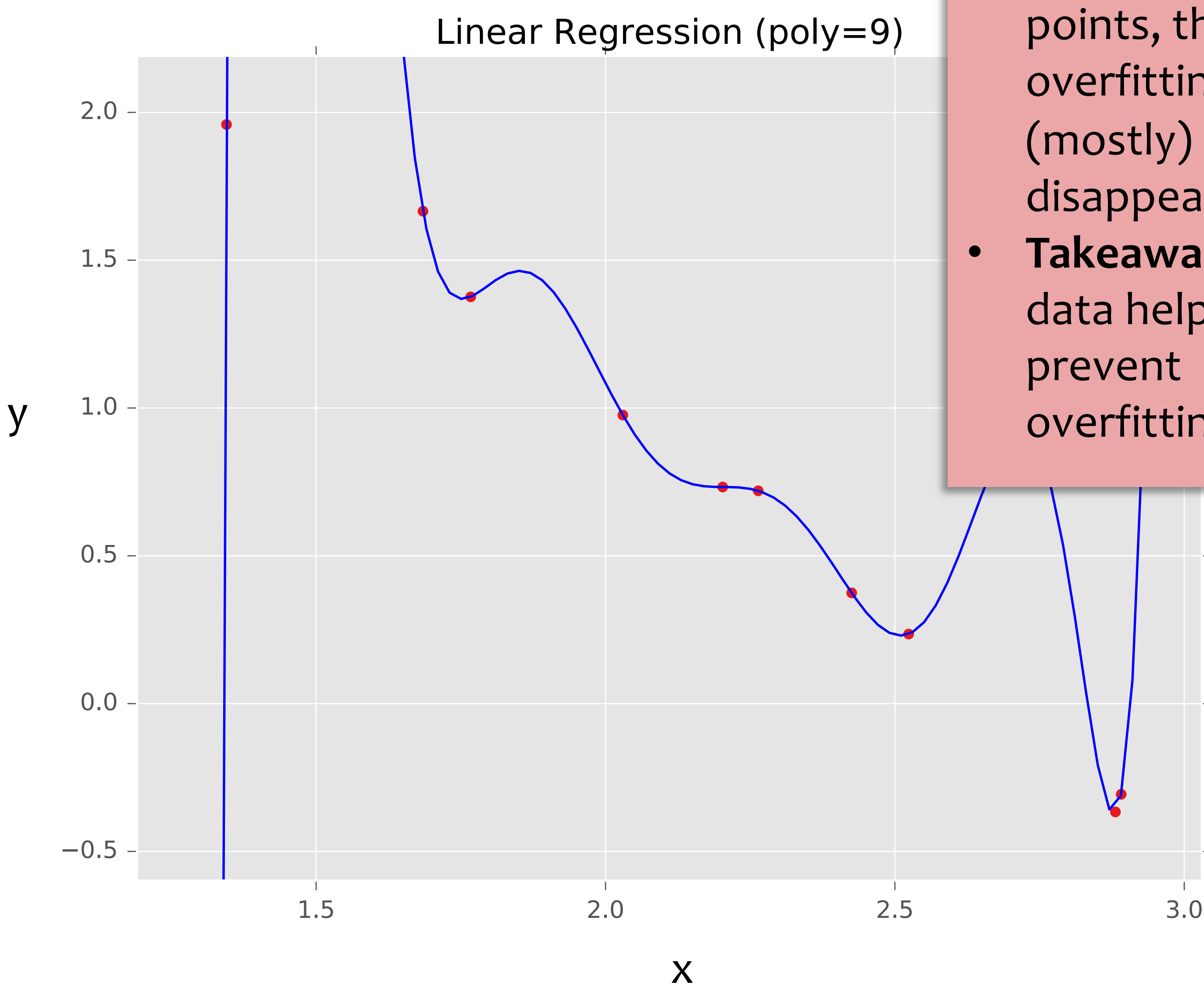
Polynomial Coefficients

	$M=0$	$M=1$	$M=3$		$M=9$
θ_0	0.19	0.82	0.31	x^0	0.35
θ_1		-1.27	7.99	x^1	232.37
θ_2		intercept	-25.43	x^2	-5321.83
θ_3			17.37	x^3	48568.31
θ_4		slope			-231639.30
θ_5					640042.26
θ_6					-1061800.52
θ_7					1042400.18
θ_8					-557682.99
θ_9					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	…	$(1.9)^9$

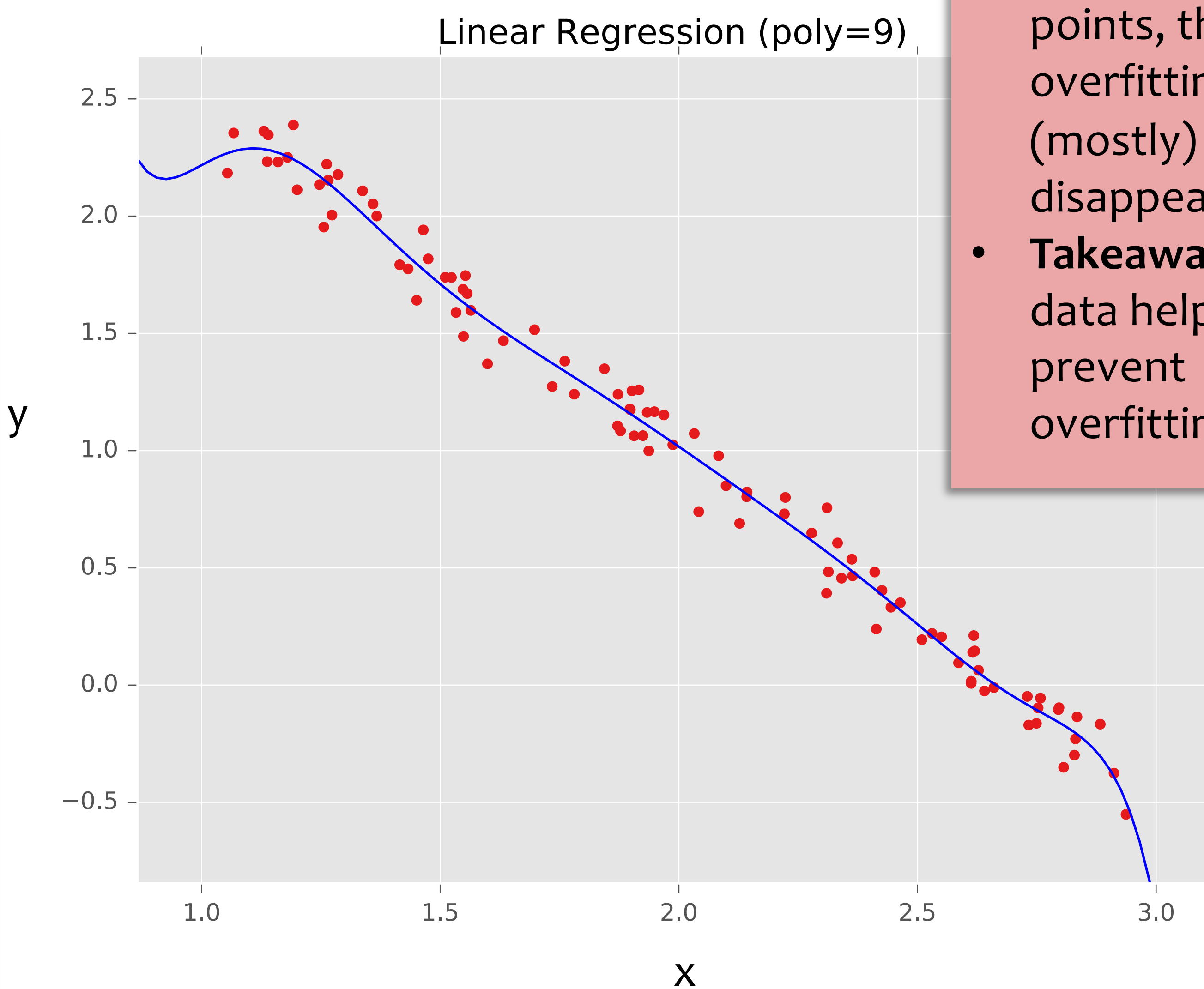


- With just N = 10 points we overfit!
- But with N = 100 points, the overfitting (mostly) disappears
- **Takeaway:** more data helps prevent overfitting

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**)

$$\vec{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = \begin{bmatrix} 23 \\ 7 \\ 0.000001 \\ -4 \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 4 \end{bmatrix} \qquad \vec{\theta}^T \vec{x} = \sum_{m=1}^{4} \theta_m x_m$$

Regularization

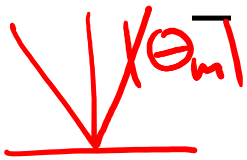
MSE, log-likelihood (fit data)

regularizer (simple)

- **Given** objective function: $J(\theta)$
- **Goal** is to find: $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \underbrace{J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})}$

trades off btwn.

- **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