(Linear Models)
+ Feature Engineering
+ Regularization
Reminders

• Homework 4: Logistic Regression
  – Out: Fri, Feb 17
  – Due: Sun, Feb. 26 at 11:59pm
LOGISTIC REGRESSION ON GAUSSIAN DATA
$p(y=1 \mid x) = \sigma(\theta^T x) = \sigma(w^T x + b)$

Logistic Regression

Logistic Regression Distribution

$\sigma(\theta^T x) = 0.5$
Logistic Regression

Classification with Logistic Regression
LEARNING LOGISTIC REGRESSION
**Learning:** finds the parameters that minimize some objective function.

\[ \theta^* = \arg\min_{\theta} J(\theta) \]

We minimize the *negative* log conditional likelihood:

\[ J(\theta) = -\log \prod_{i=1}^{N} p_{\theta}(y^{(i)} | x^{(i)}) \]

Why?

1. We can’t maximize likelihood (as in Naïve Bayes) because we don’t have a joint model \( p(x, y) \)
2. It worked well for Linear Regression (least squares is actually MCLE! more on this later... )
Maximum Conditional Likelihood Estimation

Learning: Four approaches to solving \[ \theta^* = \arg\min_{\theta} J(\theta) \]

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

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

- **Approach 3:** Newton’s Method
  (use second derivatives to better follow curvature)

- **Approach 4:** Closed Form???
  (set derivatives equal to zero and solve for parameters)
Maximum Conditional Likelihood Estimation

**Learning:** Four approaches to solving \( \theta^* = \arg\min_{\theta} J(\theta) \)

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

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

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

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

Logistic Regression does not have a closed form solution for MLE parameters.
PERCEPTRON, LINEAR REGRESSION, AND LOGISTIC REGRESSION
Matching Game

### Question: Q1
Match the Algorithm to its Update Rule

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Update Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. SGD for Logistic Regression</td>
<td>[ \theta_k \leftarrow \theta_k + \left( h_\theta(x^{(i)}) - y^{(i)} \right) ]</td>
</tr>
<tr>
<td>2. Least Mean Squares</td>
<td>[ \theta_k \leftarrow \theta_k + \frac{1}{1 + \exp \lambda(h_\theta(x^{(i)}) - y^{(i)})} ]</td>
</tr>
<tr>
<td>3. Perceptron</td>
<td>[ \theta_k \leftarrow \theta_k + \lambda(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}_k ]</td>
</tr>
</tbody>
</table>

### Answer:
- 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
- **I. None of the above**
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
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
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
In order to apply GD to Logistic Regression all we need is the gradient of the objective function (i.e. vector of partial derivatives).
Stochastic Gradient Descent (SGD)

We need a per-example objective:

Let \( J(\theta) = \sum_{i=1}^{N} J^{(i)}(\theta) \)

where \( J^{(i)}(\theta) = -\log p_{\theta}(y^i|x^i) \).

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

**Algorithm 1 Stochastic Gradient Descent (SGD)**

1. \textbf{procedure} \( \text{SGD}(\mathcal{D}, \theta^{(0)}) \)
2. \( \theta \leftarrow \theta^{(0)} \)
3. \textbf{while} not converged \textbf{do}
4. \hspace{1em} \textbf{for} \( i \in \text{shuffle}([1, 2, \ldots, N]) \) \textbf{do}
5. \hspace{2em} \( \theta \leftarrow \theta - \gamma \nabla_{\theta} J^{(i)}(\theta) \)
6. \textbf{return} \( \theta \)
**Logistic Regression vs. Perceptron**

**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 $p^*(y \mid x)$ 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:

$$
\begin{align*}
x^{(i)} &\sim p^*(\cdot) \\
y^{(i)} &\sim p^*(\cdot \mid x^{(i)})
\end{align*}
$$

Our goal is to learn a probability distribution $p(y \mid x)$ that best approximates $p^*(y \mid x)$.
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 knew a \( p^*(y|x) \) and picked a 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 knew a \( p^*(y|x) \) and picked a 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 randomly chosen examples
Mini-Batch SGD

While not converged: $\theta \leftarrow \theta - \gamma g$

Three variants of first-order optimization:

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

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

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

Good use of GPU memory

$\{i_1, i_2, i_3, i_4\} = \{7, 2, 3, 56, 100\}$

$S = 4$
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
FEATURE ENGINEERING
Handcrafted Features

\[
p(y|x) \propto \exp(\Theta_y \cdot f)
\]
Where do features come from?

- Hand-crafted features:
  - Sun et al., 2011
  - Zhou et al., 2005

- Feature engineering:
  - First word before M1
  - Second word before M1
  - Bag-of-words in M1
  - Head word of M1
  - Other word in between
  - First word after M2
  - Second word after M2
  - Bag-of-words in M2
  - Head word of M2
  - Bigrams in between
  - Words on dependency path
  - Country name list
  - Personal relative triggers
  - Personal title list
  - WordNet Tags
  - Heads of chunks in between
  - Path of phrase labels
  - Combination of entity types

- Feature learning
Where do features come from?

- **Feature Engineering**
  - hand-crafted features
    - Sun et al., 2011
    - Zhou et al., 2005
  - word embeddings
    - Mikolov et al., 2013

- **Feature Learning**
  - **Look-up table**
    - input (context words)
    - embedding
    - missing word
  - **Classifier**
    - similar words, similar embeddings
    - CBOB model in Mikolov et al. (2013)
  - unsupervised learning

- Similar words, similar embeddings:
  - cat: 0.11 0.23 ...
  - dog: 0.13 0.26 ...
  - unsupervised learning
Where do features come from?

**Feature Engineering**

1. **hand-crafted features**
   - Sun et al., 2011
   - Zhou et al., 2005

2. **word embeddings**
   - Mikolov et al., 2013

**Feature Learning**

1. **Convolutional Neural Networks**
   - Collobert and Weston, 2008
   - The [movie] showed [wars]

2. **Recursive Auto Encoder**
   - Socher, 2011
   - string embeddings
   - Collobert & Weston, 2008
Where do features come from?

- **Hand-crafted features**
  - Sun et al., 2011
  - Zhou et al., 2005

- **Word embeddings**
  - Mikolov et al., 2013
  - Socher, 2011

- **String embeddings**
  - Collobert & Weston, 2008

- **Tree embeddings**
  - Socher et al., 2013
  - Hermann & Blunsom, 2013

The [movie] showed [wars]
Where do features come from?

Feature Engineering

hand-crafted features
- Sun et al., 2011
- Zhou et al., 2005

word embedding features
- Turian et al., 2010
- Koo et al., 2008
- Mikolov et al., 2013

Feature Learning

- Socher et al., 2013
- Hermann & Blunsom, 2013
- string embeddings
  - Socher, 2011
- Hermann et al., 2014
- tree embeddings
  - Collobert & Weston, 2008

Refine embedding features with semantic/syntactic info
Where do features come from?

- **Hand-crafted features**
  - Sun et al., 2011
  - Zhou et al., 2005

- **Word embedding features**
  - Turian et al., 2010
  - Koo et al., 2008
  - Hermann et al., 2014

- **Best of both worlds?**
  - Mikolov et al., 2013
  - Collobert & Weston, 2008

- **String embeddings**
  - Socher, 2011

- **Tree embeddings**
  - Socher et al., 2013
  - Hermann & Blunsom, 2013

- **Feature Engineering**

- **Feature Learning**
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:

- is-capital($w_i$)
- endswith($w_i$, “e”)
- endswith($w_i$, “d”)
- endswith($w_i$, “ed”)
- $w_i$ == “aardvark”
- $w_i$ == “hope”

<table>
<thead>
<tr>
<th>Feature</th>
<th>$x^{(1)}$</th>
<th>$x^{(2)}$</th>
<th>$x^{(3)}$</th>
<th>$x^{(4)}$</th>
<th>$x^{(5)}$</th>
<th>$x^{(6)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-capital($w_i$)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>endswith($w_i$, “e”)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>endswith($w_i$, “d”)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>endswith($w_i$, “ed”)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$w_i$ == “aardvark”</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$w_i$ == “hope”</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The movie I watched depicted hope
Feature Engineering for NLP

Context Features:

<table>
<thead>
<tr>
<th>wi</th>
<th>wi+1</th>
<th>wi-1</th>
<th>wi+2</th>
<th>wi-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The movie I watched depicted hope.
Feature Engineering for NLP

Context Features:

<table>
<thead>
<tr>
<th></th>
<th>$x^{(1)}$</th>
<th>$x^{(2)}$</th>
<th>$x^{(3)}$</th>
<th>$x^{(4)}$</th>
<th>$x^{(5)}$</th>
<th>$x^{(6)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$w_i$ = “I”</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$w_{i+1}$ = “I”</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$w_{i-1}$ = “I”</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$w_{i+2}$ = “I”</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$w_{i-2}$ = “I”</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The movie I watched depicted hope
# Feature Engineering for NLP

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3GRAMMEMM</td>
<td>See text</td>
<td>248,798</td>
<td>52.07%</td>
<td>96.92%</td>
<td>88.99%</td>
</tr>
<tr>
<td>NAACL 2003</td>
<td>See text and [1]</td>
<td>460,552</td>
<td>55.31%</td>
<td>97.15%</td>
<td>88.61%</td>
</tr>
<tr>
<td>Replication</td>
<td>See text and [1]</td>
<td>460,551</td>
<td>55.62%</td>
<td>97.18%</td>
<td>88.92%</td>
</tr>
<tr>
<td>Replication’</td>
<td>+rareFeatureThresh = 5</td>
<td>482,364</td>
<td>55.67%</td>
<td>97.19%</td>
<td>88.96%</td>
</tr>
<tr>
<td>5W</td>
<td>+⟨t₀, w₋₂⟩, ⟨t₀, w₂⟩</td>
<td>730,178</td>
<td>56.23%</td>
<td>97.20%</td>
<td>89.03%</td>
</tr>
<tr>
<td>5WSHAPES</td>
<td>+⟨t₀, s₋₁⟩, ⟨t₀, s₀⟩, ⟨t₀, s₊₁⟩</td>
<td>731,661</td>
<td>56.52%</td>
<td>97.25%</td>
<td>89.81%</td>
</tr>
<tr>
<td>5WSHAPESDS</td>
<td>+ distributional similarity</td>
<td>737,955</td>
<td>56.79%</td>
<td>97.28%</td>
<td>90.46%</td>
</tr>
</tbody>
</table>

The movie I watched depicted hope
Feature Engineering for CV

Edge detection (Canny)

Corner Detection (Harris)

Figures from http://opencv.org
Scale Invariant Feature Transform (SIFT)

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:
Suppose you are building a linear regression model to predict the construction cost of all houses built in Pittsburgh in the 1930s – 1940s (in 30/40s dollars).

What features would you use?

Answer:
1. Soil type
2. Asbestos
3. Steel?
4. # bedrooms
5. # baths
6. sq ft. (livable)
7. Grade
8. (sq ft)^2
9. log(sq ft)
10. exp(sq ft)
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 \( \mathbf{x} \)
  
  – 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  
  \[
  b_j(x) = \exp \left( \frac{-(x - \mu_j)^2}{2\sigma^2_j} \right)
  \]

  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 \( b(\mathbf{x}) \) even though a nonlinear function of \( \mathbf{x} \)

**Examples:**

- Perceptron
- Linear regression
- Logistic regression
Example: Linear Regression

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
</tr>
</tbody>
</table>

true “unknown” target function is linear with negative slope and gaussian noise
Example: Linear Regression

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
</tr>
</tbody>
</table>

ture “unknown” target function is linear with negative slope and gaussian noise
**Example: Linear Regression**

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>( x^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>(1.2)^2</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>(1.7)^2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
<td>(1.9)^2</td>
</tr>
</tbody>
</table>

true “unknown” target function is linear with negative slope and gaussian noise
Example: Linear Regression

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>x^2</th>
<th>x^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>(1.2)^2</td>
<td>(1.2)^3</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>(1.7)^2</td>
<td>(1.7)^3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
<td>(1.9)^2</td>
<td>(1.9)^3</td>
</tr>
</tbody>
</table>

true “unknown” target function is linear with negative slope and gaussian noise
Goal: Learn $$y = \mathbf{w}^T f(x) + b$$ where $$f(.)$$ is a polynomial basis function

table:

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>...</th>
<th>$$x^5$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>...</td>
<td>(1.2)^5</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>...</td>
<td>(1.7)^5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
<td>...</td>
<td>(1.9)^5</td>
</tr>
</tbody>
</table>

text:

true “unknown” target function is linear with negative slope and gaussian noise
**Example: Linear Regression**

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>...</th>
<th>( x^8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>...</td>
<td>( (1.2)^8 )</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>...</td>
<td>( (1.7)^8 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
<td>...</td>
<td>( (1.9)^8 )</td>
</tr>
</tbody>
</table>

true “unknown” target function is linear with negative slope and gaussian noise.
Example: Linear Regression

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>...</th>
<th>( x^9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>...</td>
<td>((1.2)^9)</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>...</td>
<td>((1.7)^9)</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
<td>...</td>
<td>((1.9)^9)</td>
</tr>
</tbody>
</table>

true “unknown” target function is linear with negative slope and gaussian noise.
Over-fitting

Root-Mean-Square (RMS) Error: \[ E_{\text{RMS}} = \sqrt{\frac{2E(w^*)}{N}} \]
## Polynomial Coefficients

<table>
<thead>
<tr>
<th></th>
<th>$M = 0$</th>
<th>$M = 1$</th>
<th>$M = 3$</th>
<th>$M = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_0$</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
<td></td>
</tr>
<tr>
<td>$\theta_2$</td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta_3$</td>
<td>17.37</td>
<td>48568.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta_4$</td>
<td></td>
<td>-231639.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta_5$</td>
<td></td>
<td></td>
<td>640042.26</td>
<td></td>
</tr>
<tr>
<td>$\theta_6$</td>
<td></td>
<td></td>
<td></td>
<td>-1061800.52</td>
</tr>
<tr>
<td>$\theta_7$</td>
<td></td>
<td></td>
<td></td>
<td>1042400.18</td>
</tr>
<tr>
<td>$\theta_8$</td>
<td></td>
<td></td>
<td></td>
<td>-557682.99</td>
</tr>
<tr>
<td>$\theta_9$</td>
<td></td>
<td></td>
<td></td>
<td>125201.43</td>
</tr>
</tbody>
</table>
**Example: Linear Regression**

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>...</th>
<th>$x^9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>...</td>
<td>$(1.2)^9$</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>...</td>
<td>$(1.7)^9$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>1.1</td>
<td>1.9</td>
<td>...</td>
<td>$(1.9)^9$</td>
</tr>
</tbody>
</table>

- 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 $y = w^T f(x) + b$ where $f(.)$ is a polynomial basis function

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

<table>
<thead>
<tr>
<th>i</th>
<th>y</th>
<th>x</th>
<th>...</th>
<th>$x^9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.0</td>
<td>1.2</td>
<td>...</td>
<td>$(1.2)^9$</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.7</td>
<td>...</td>
<td>$(1.7)^9$</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>2.7</td>
<td>...</td>
<td>$(2.7)^9$</td>
</tr>
<tr>
<td>4</td>
<td>1.1</td>
<td>1.9</td>
<td>...</td>
<td>$(1.9)^9$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>98</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>99</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>0.9</td>
<td>1.5</td>
<td>...</td>
<td>$(1.5)^9$</td>
</tr>
</tbody>
</table>
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

- **Given** objective function: \( J(\theta) \)
- **Goal** is to find: \( \hat{\theta} = \arg\min_\theta J(\theta) + \lambda r(\theta) \)

- **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): \( \|\theta\|_q = \left( \sum_{m=1}^{M} |\theta_m|^q \right)^{\frac{1}{q}} \)

<table>
<thead>
<tr>
<th>( q )</th>
<th>( r(\theta) )</th>
<th>yields parameters that are...</th>
<th>name</th>
<th>optimization notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( |\theta|_0 = \sum \mathbb{1}(\theta_m \neq 0) )</td>
<td>zero values</td>
<td>L0 reg.</td>
<td>no good computational solutions</td>
</tr>
<tr>
<td>1</td>
<td>( |\theta|_1 = \sum</td>
<td>\theta_m</td>
<td>)</td>
<td>zero values</td>
</tr>
<tr>
<td>2</td>
<td>( (|\theta|_2)^2 = \sum \theta_m^2 )</td>
<td>small values</td>
<td>L2 reg.</td>
<td>differentiable</td>
</tr>
</tbody>
</table>
Regularization Examples

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

\[
J_{RR}(\theta) = J(\theta) + \lambda \| \theta \|^2_2
\]

\[
= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \sum_{m=1}^{M} \theta_m^2
\]

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

\[
J_{\text{LASSO}}(\theta) = J(\theta) + \lambda \| \theta \|_1
\]

\[
= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \sum_{m=1}^{M} |\theta_m|
\]
Regularization Examples

Add an **L2 regularizer** to Logistic Regression

\[ J'(\theta) = J(\theta) + \lambda \|\theta\|_2^2 \]

\[ = \frac{1}{N} \sum_{i=1}^{N} -\log p(y^{(i)} | x^{(i)}, \theta) + \lambda \sum_{m=1}^{M} \theta_m^2 \]

Add an **L1 regularizer** to Logistic Regression

\[ J'(\theta) = J(\theta) + \lambda \|\theta\|_1 \]

\[ = \frac{1}{N} \sum_{i=1}^{N} -\log p(y^{(i)} | x^{(i)}, \theta) + \lambda \sum_{m=1}^{M} |\theta_m| \]
Question:
Suppose we are minimizing $J'(\theta)$ where

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

As $\lambda$ increases, the minimum of $J'(\theta)$ will...

A. ...move towards the midpoint between $J(\theta)$ and $r(\theta)$
B. ...move towards the minimum of $J(\theta)$
C. ...move towards the minimum of $r(\theta)$
D. ...move towards a theta vector of positive infinities
E. ...move towards a theta vector of negative infinities
F. ...stay the same