10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Regularization

+

Perceptron

Perceptron Readings:
Murphy 8.5.4
Bishop 4.1.7
HTF --
Mitchell 4.4.0

Matt Gormley
Lecture 10
February 20, 2016

Reminders

- Homework 3: Linear / Logistic Regression
  - Release: Mon, Feb. 13
  - Due: Wed, Feb. 22 at 11:59pm
- Homework 4: Perceptron / Kernels / SVM
  - Release: Wed, Feb. 22
  - Due: Wed, Mar. 01 at 11:59pm
- Midterm Exam (Evening Exam)
  - Tue, Mar. 07 at 7:00pm – 9:30pm
  - See Piazza for details about location

1 week for HW4

Outline

- **Regularization**
  - Motivation: Overfitting
  - L2, L1, L0 Regularization
  - Relation between Regularization and MAP Estimation
- **Perceptron**
  - Online Learning
  - Margin Definitions
  - Perceptron Algorithm
  - Perceptron Mistake Bound
- **Generative vs. Discriminative Classifiers**

REGULARIZATION

Overfitting

Recall…

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:

- KNN (e.g. when k is small)
- Naïve Bayes (e.g. without a prior)
- Linear Regression (e.g. with basis function)
- Logistic Regression (e.g. with many rare features)

Motivation: Regularization

Example: Stock Prices

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

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

Motivation: Regularization

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

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

Regularization

Whiteboard

- L2, L1, L0 Regularization
- Example: Linear Regression
- Probabilistic Interpretation of Regularization

Regularization

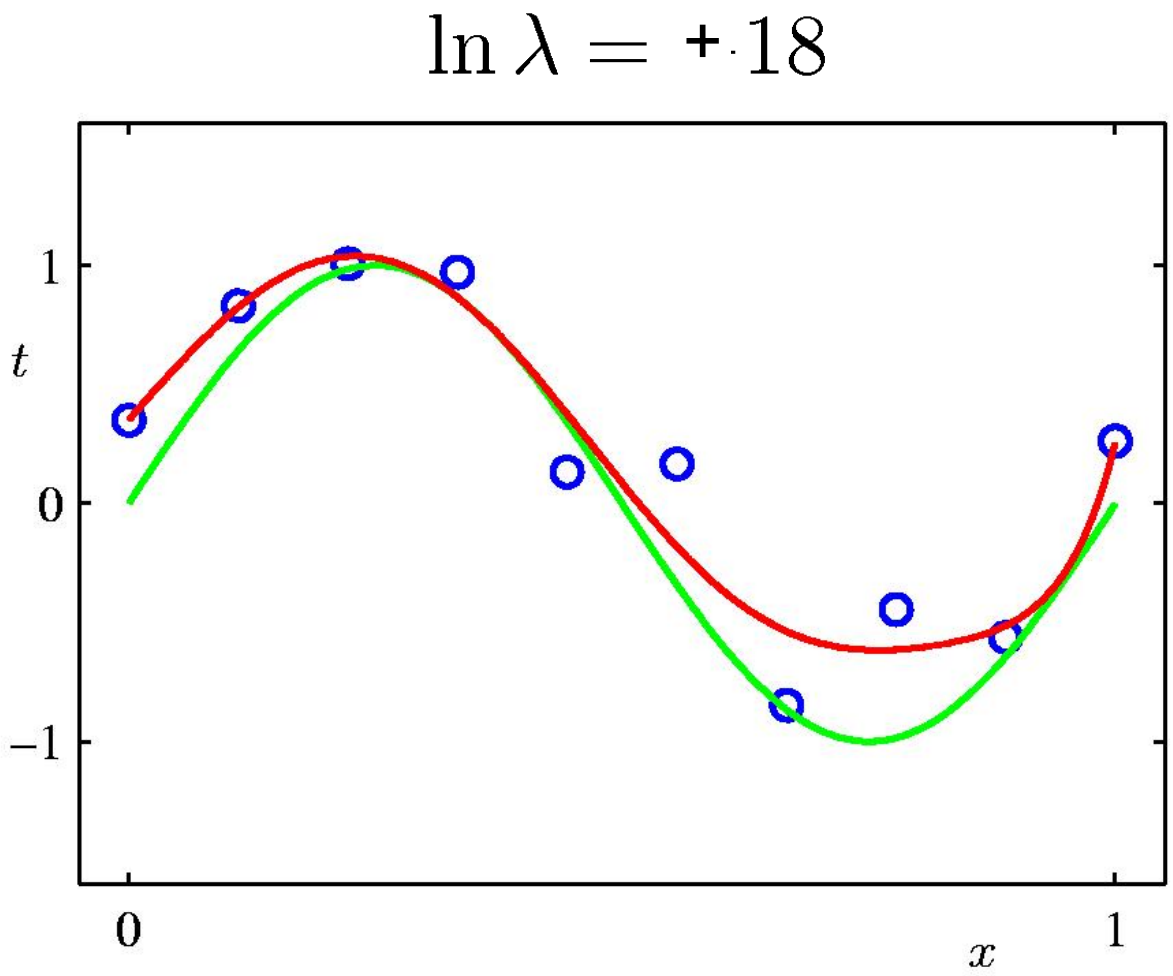
Don't Regularize the Bias (Intercept) Parameter!

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

Whitening Data

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

Regularization:



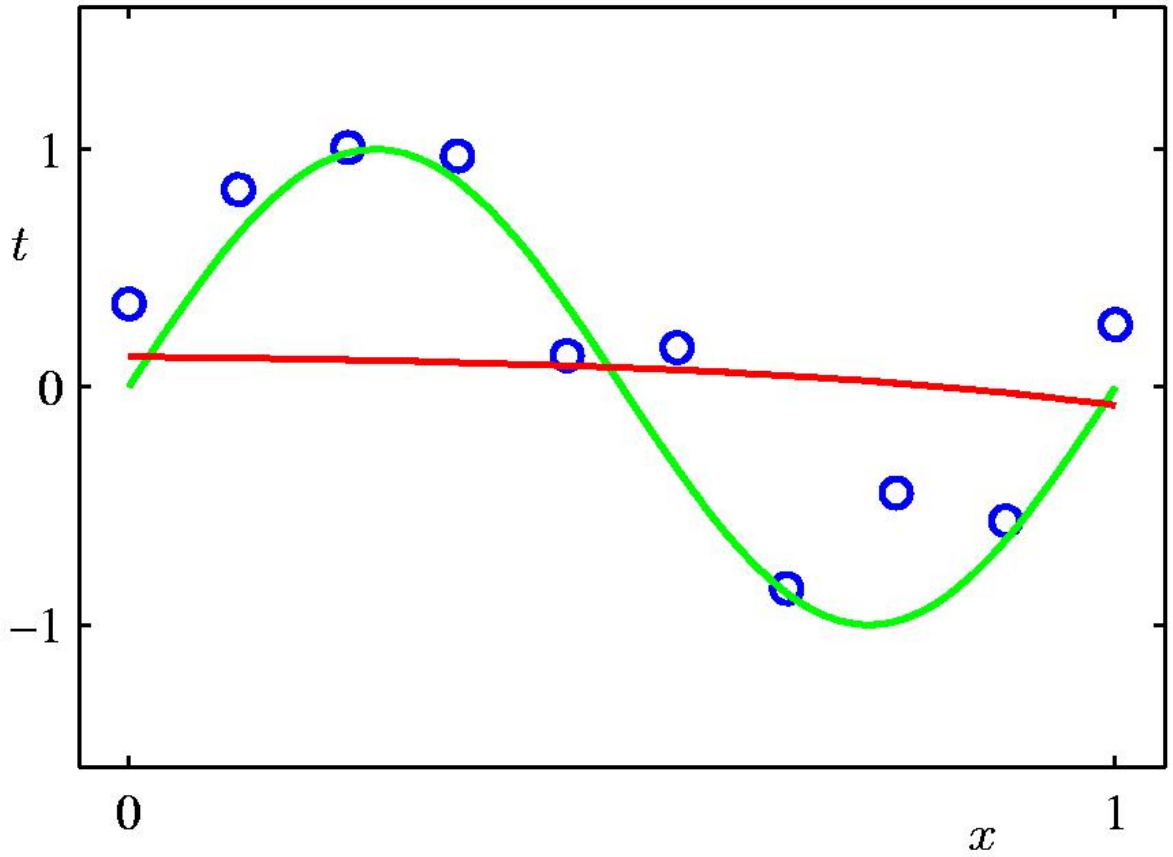
Slide courtesy of William Cohen

Polynomial Coefficients

	none	exp(18)	huge
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Slide courtesy of William Cohen

Over Regularization:

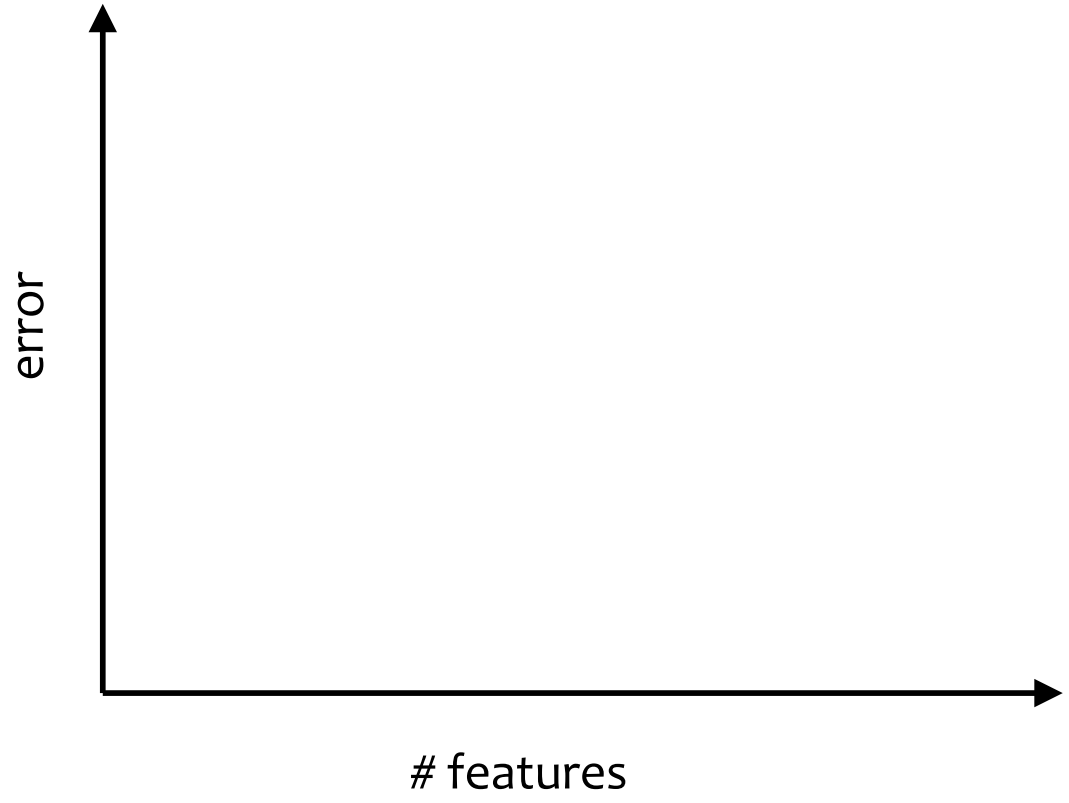


Slide courtesy of William Cohen

Regularization Exercise

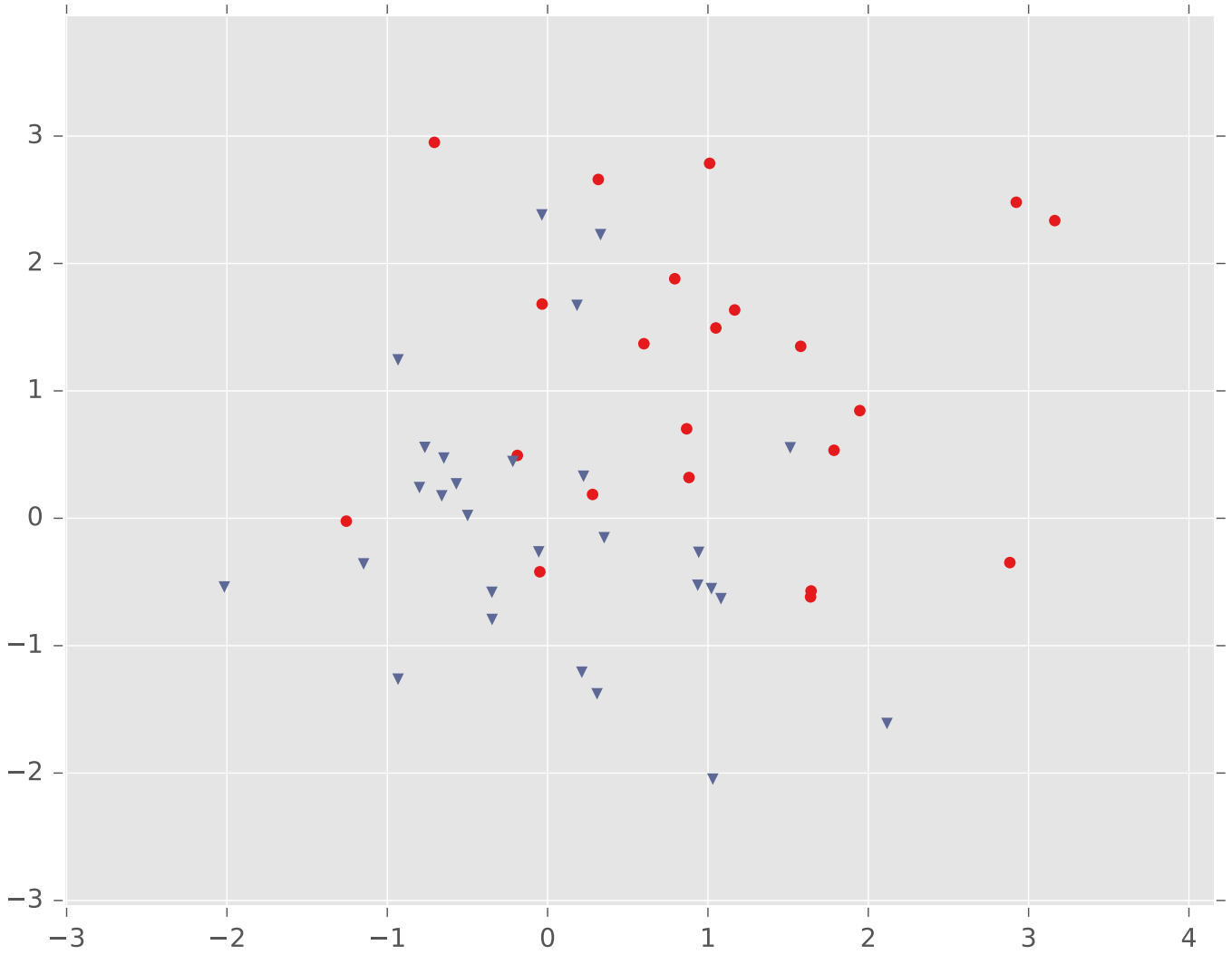
In-class Exercise

1. Plot train error vs. # features (cartoon)
2. Plot test error vs. # features (cartoon)



Example: Logistic Regression

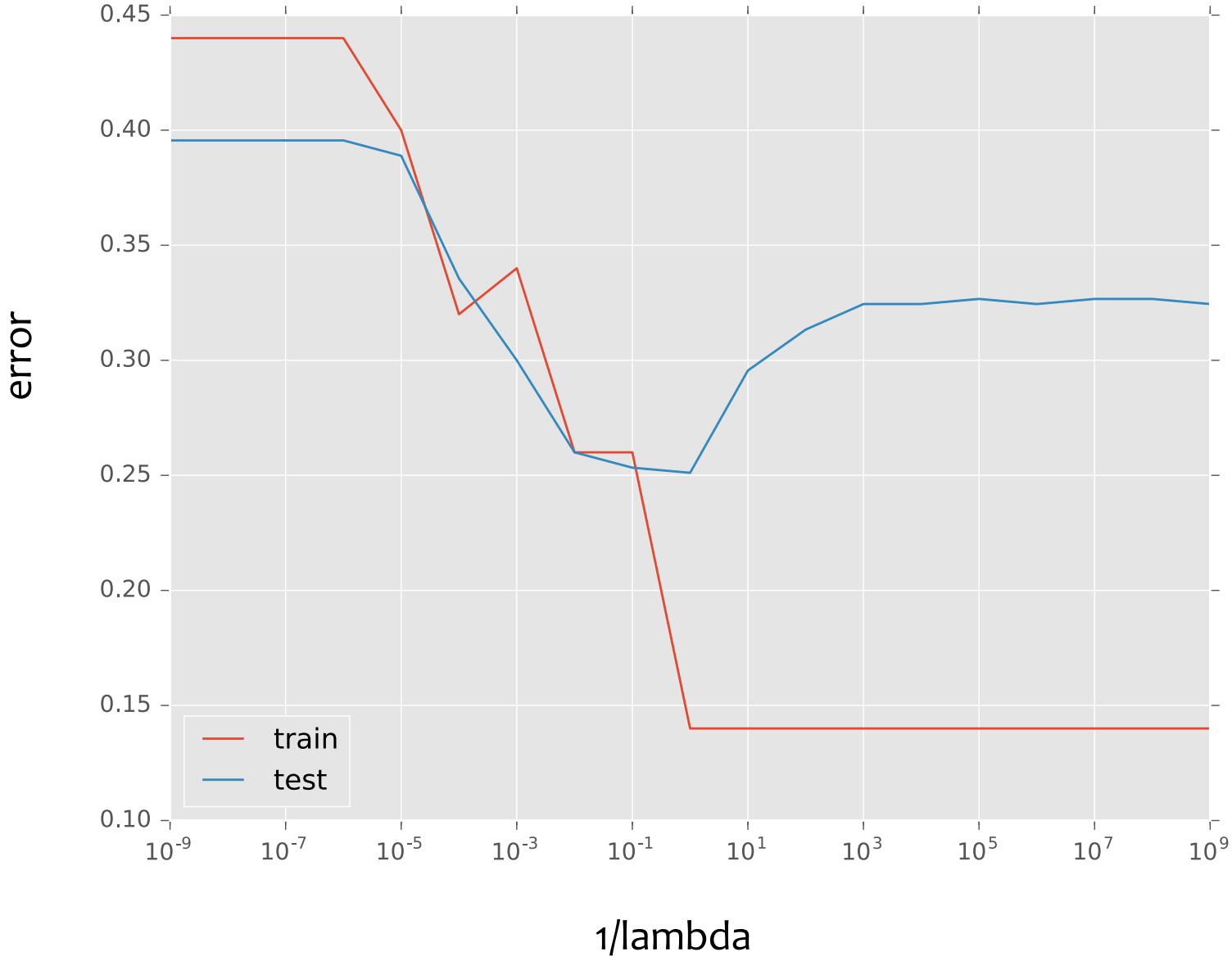
Training Data



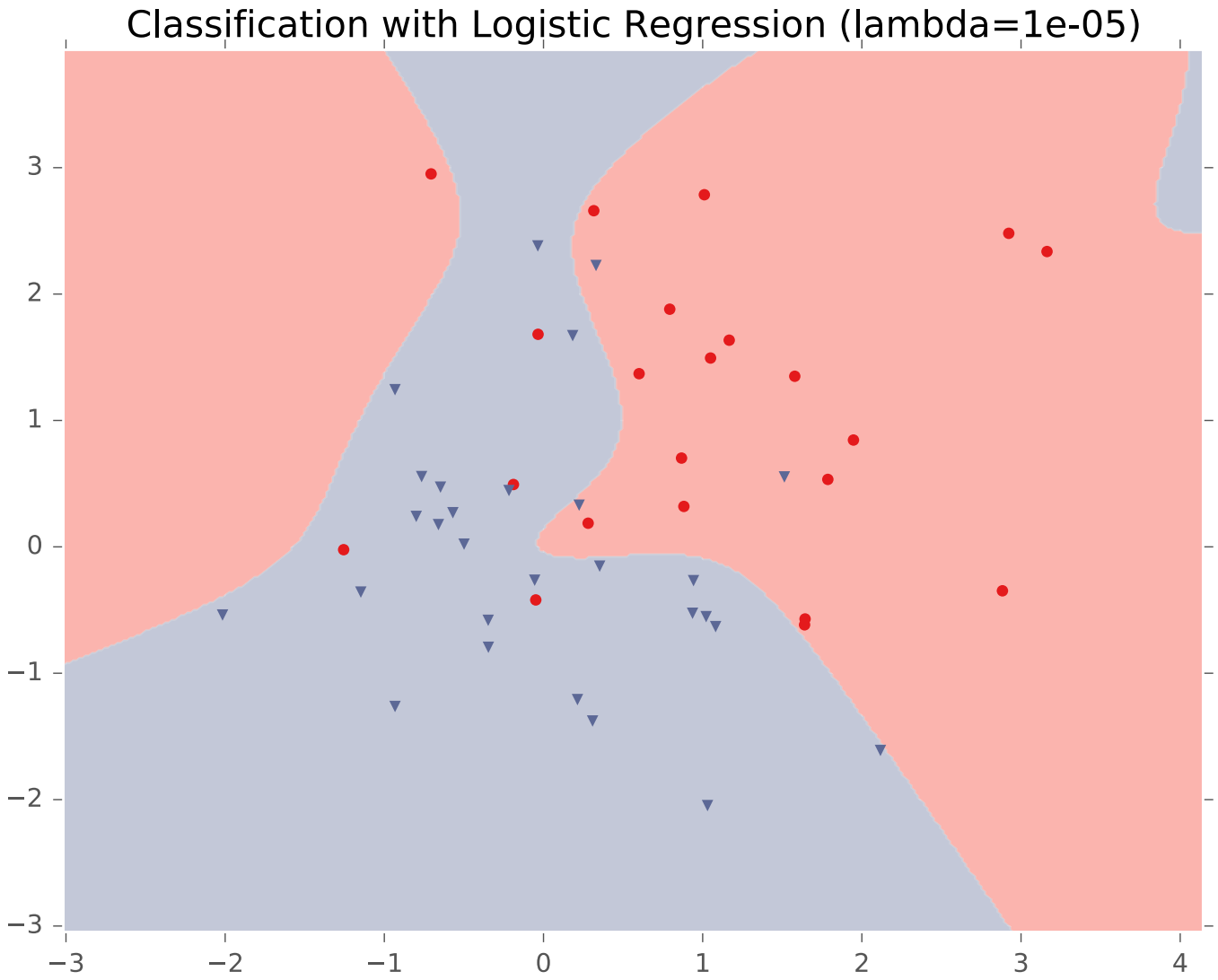
Example: Logistic Regression

Test Data

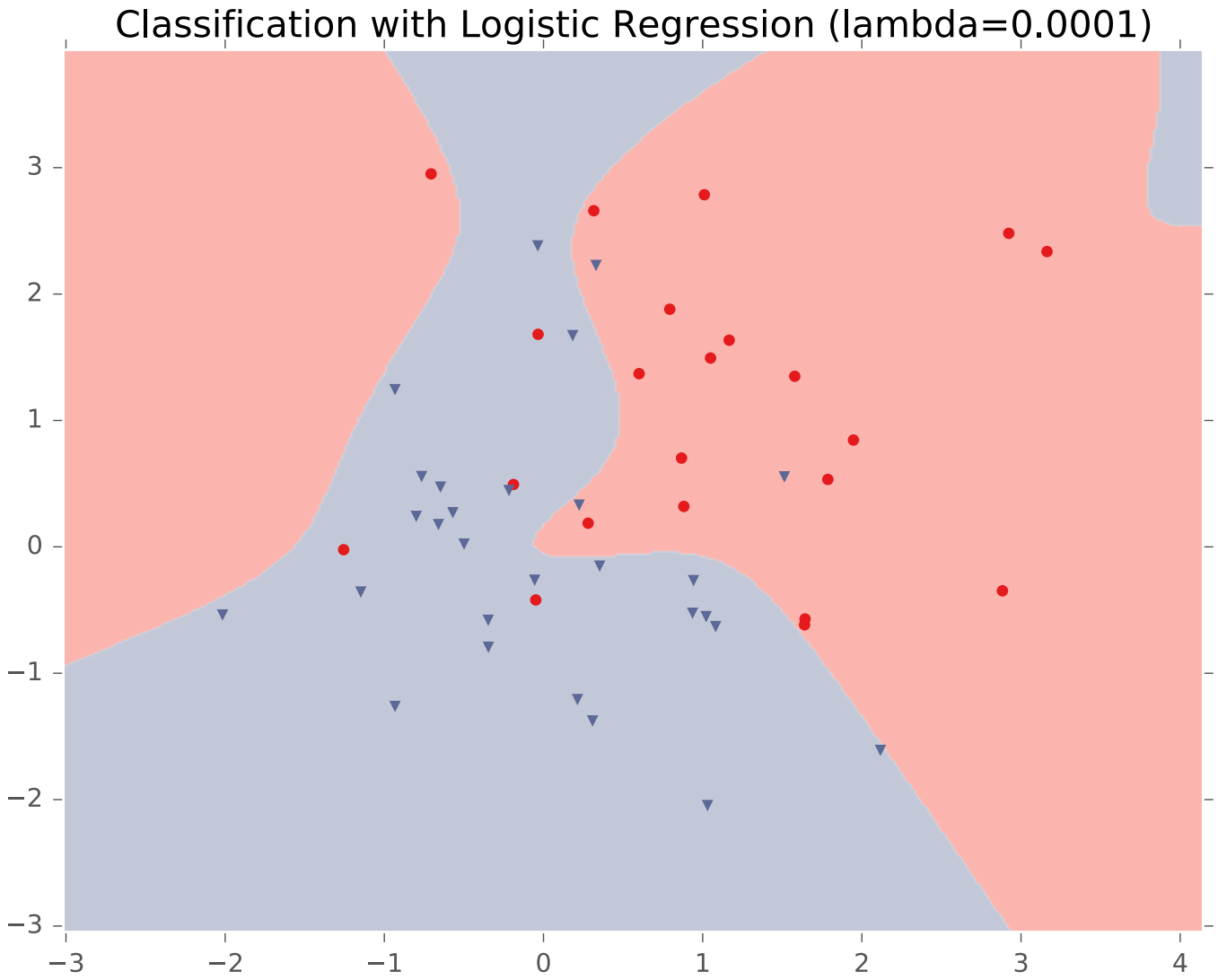
Example: Logistic Regression



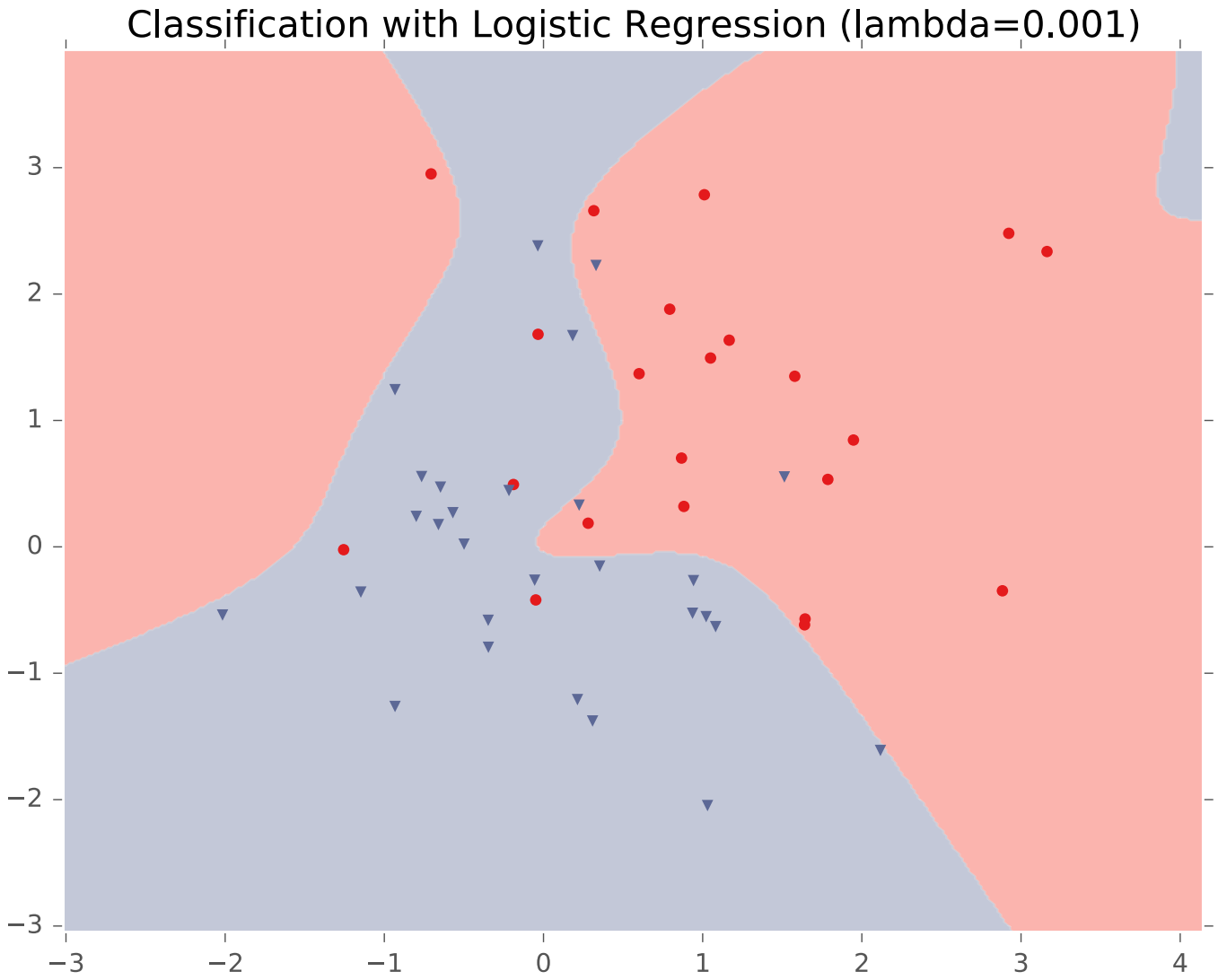
Example: Logistic Regression



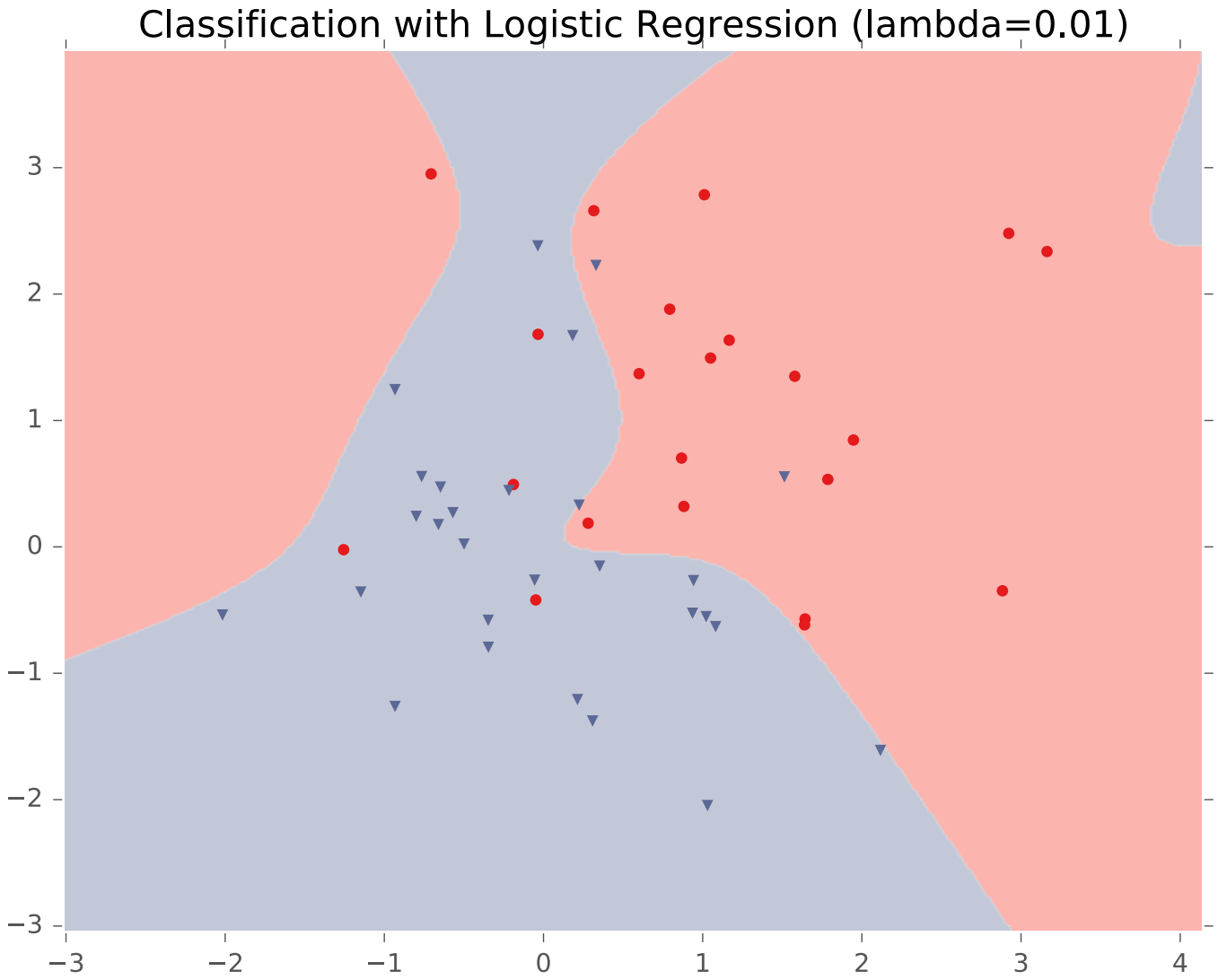
Example: Logistic Regression



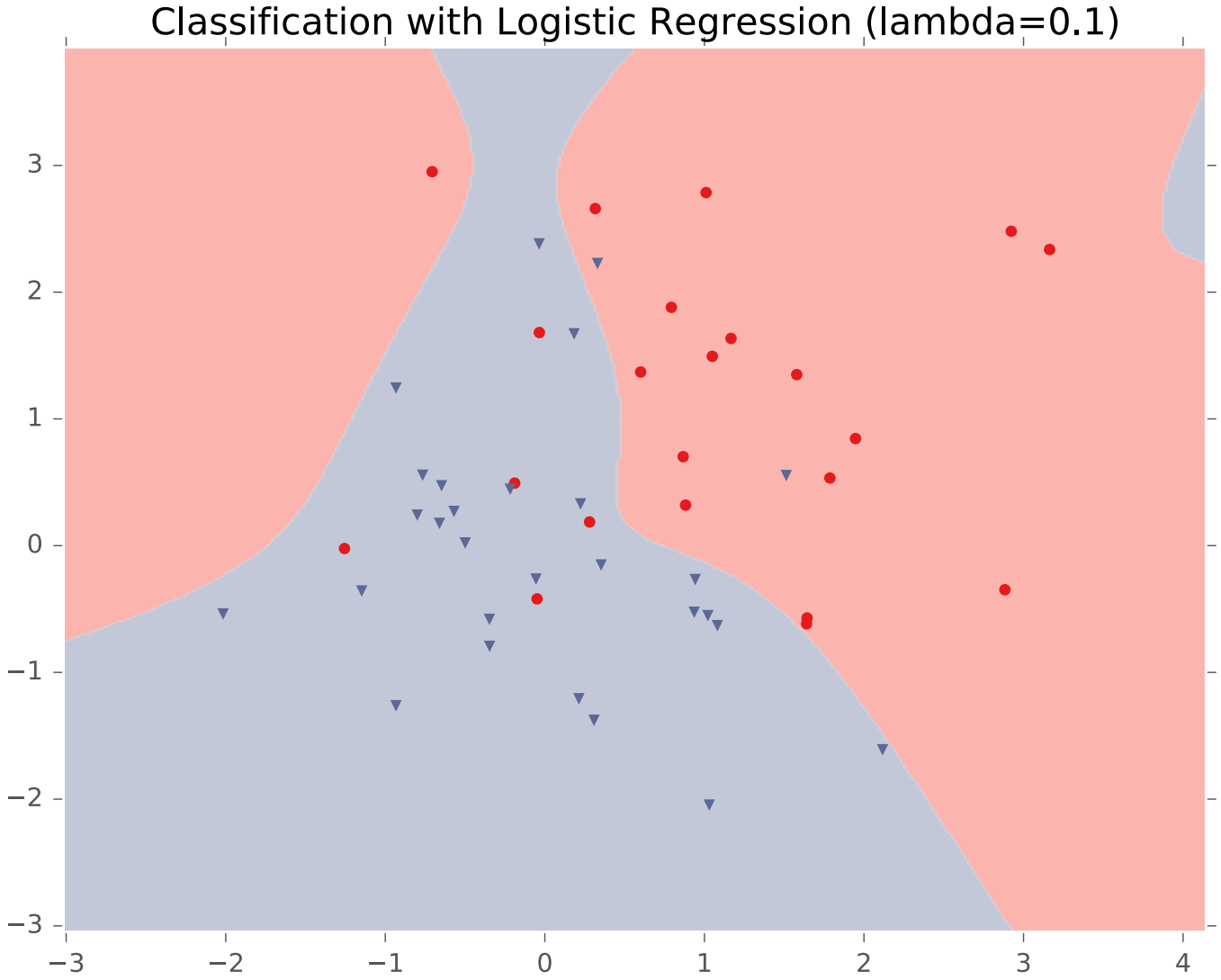
Example: Logistic Regression



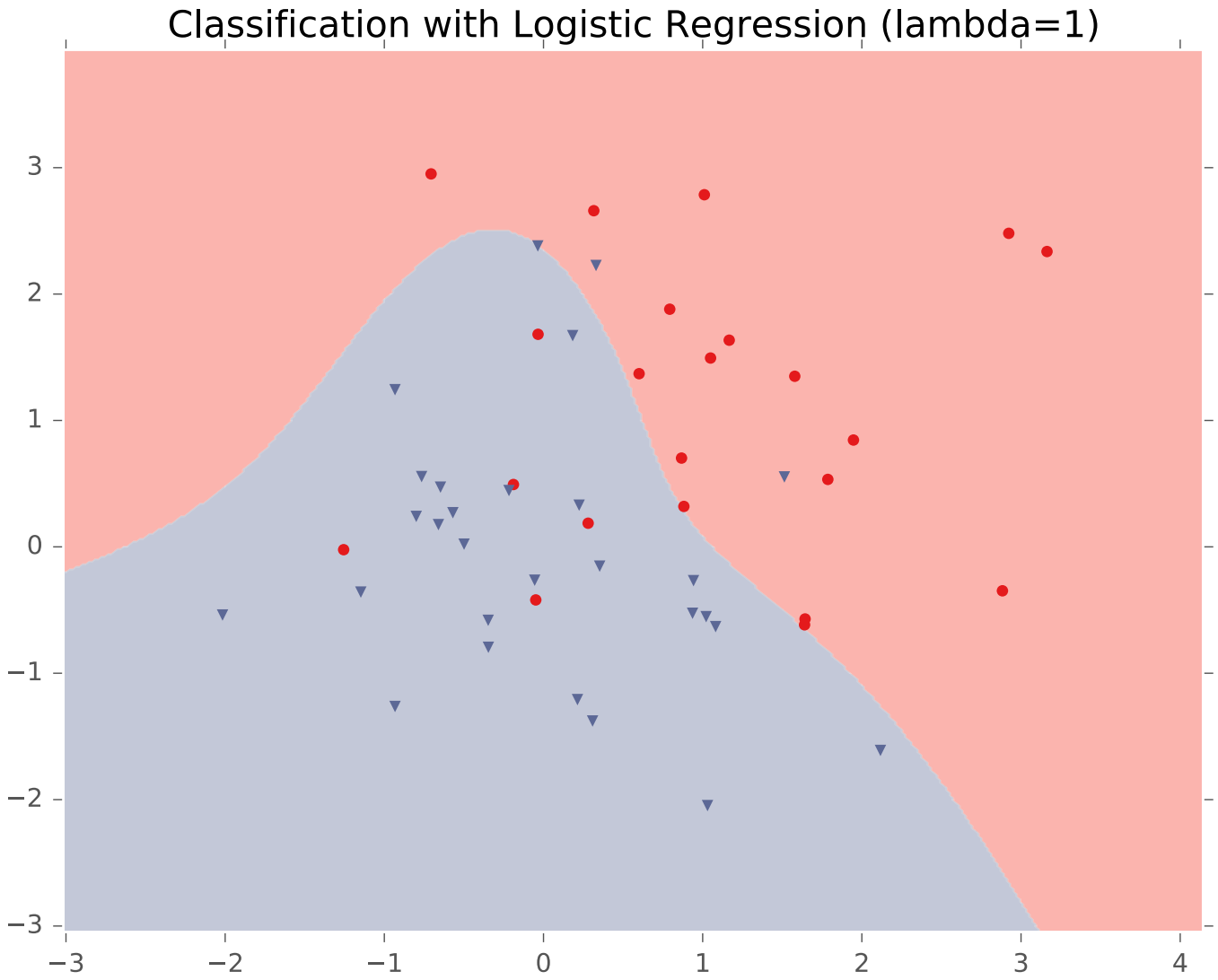
Example: Logistic Regression



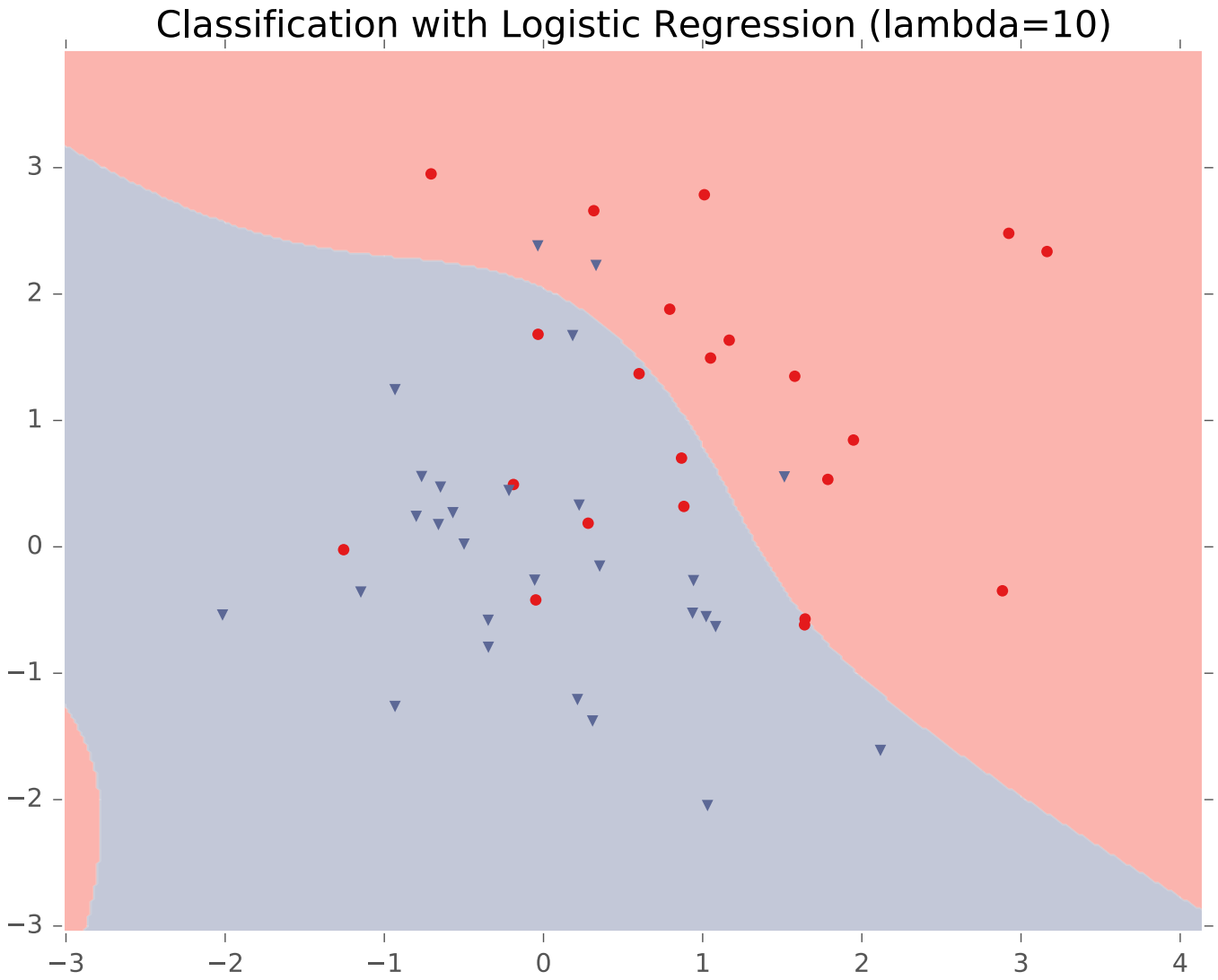
Example: Logistic Regression



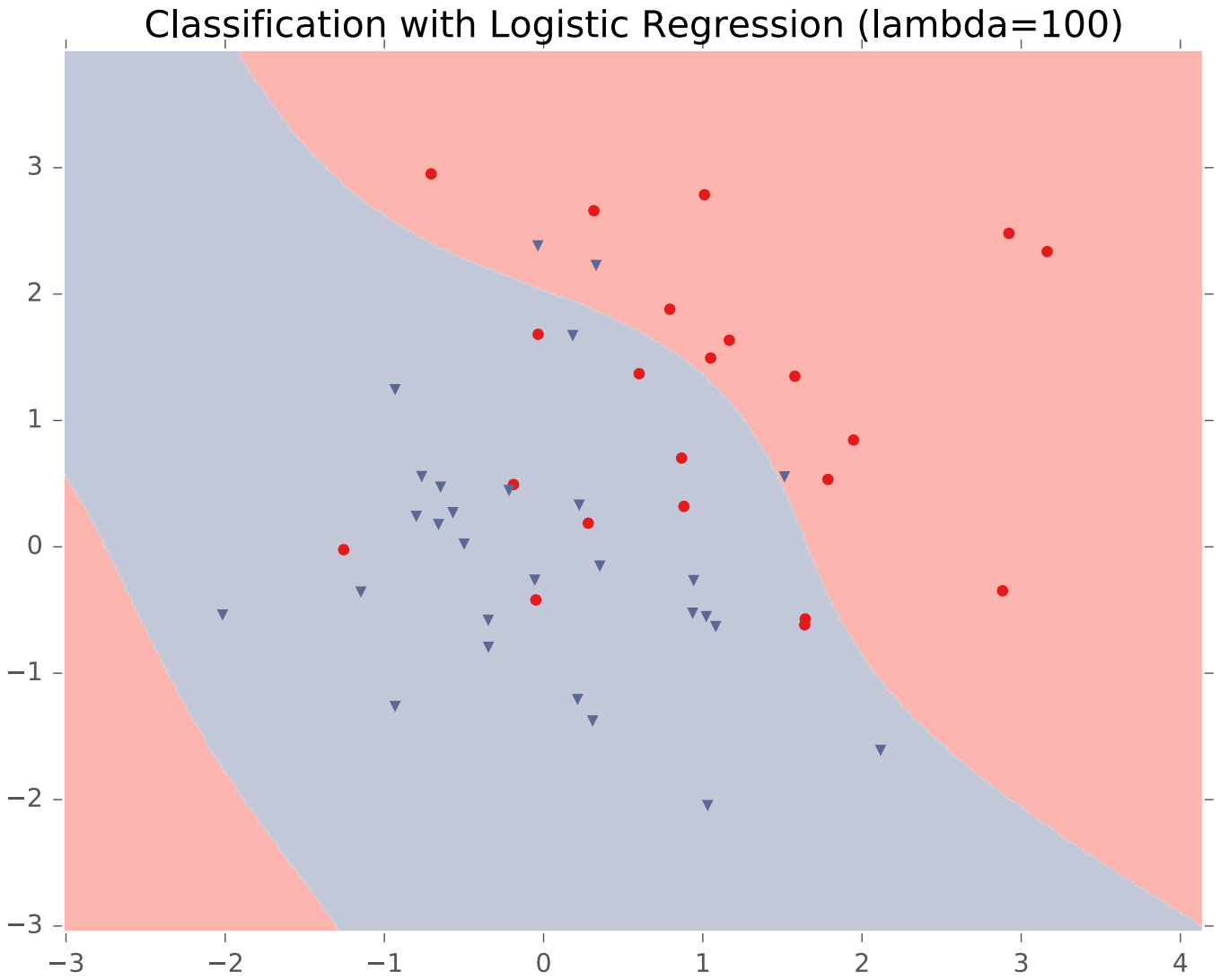
Example: Logistic Regression



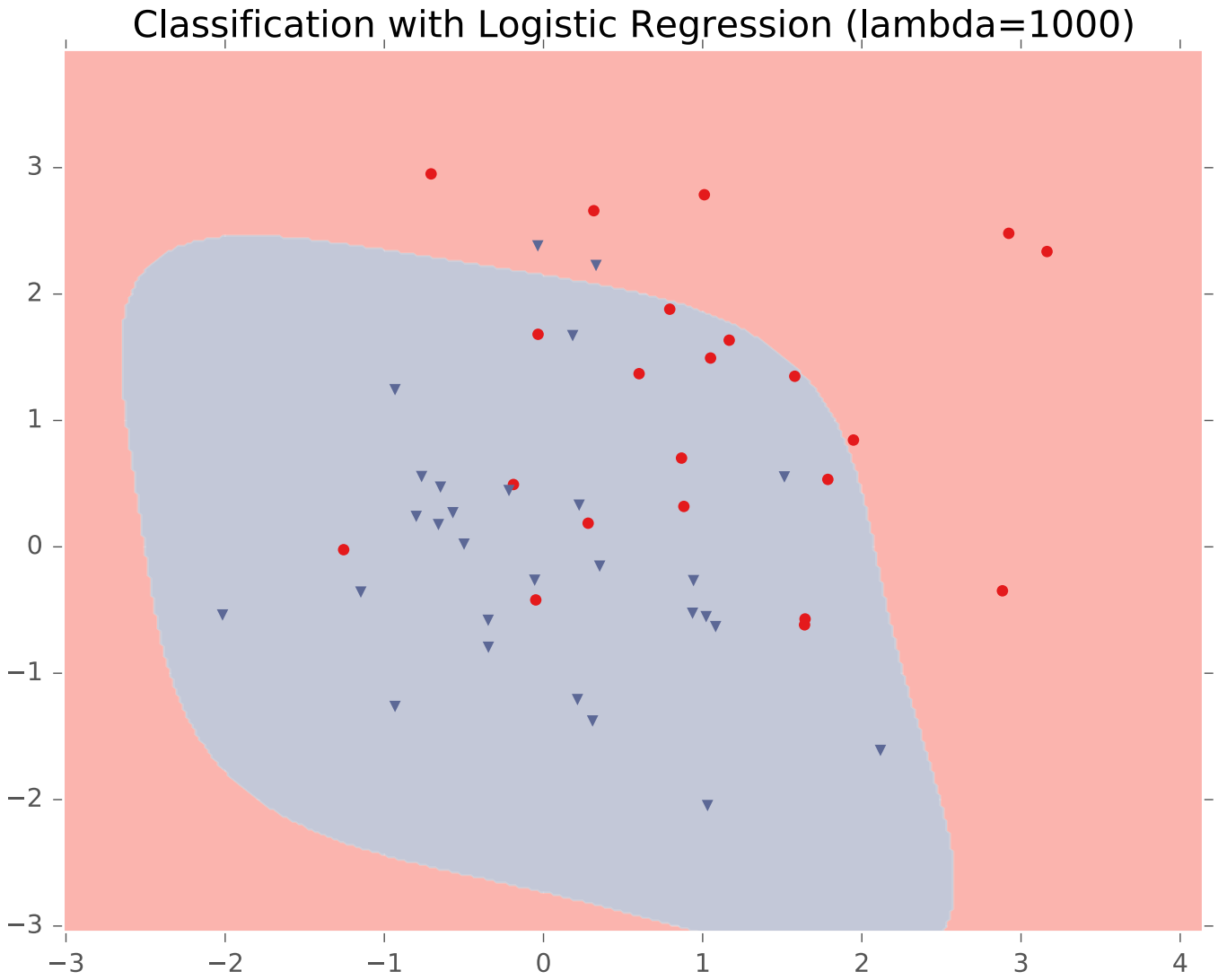
Example: Logistic Regression



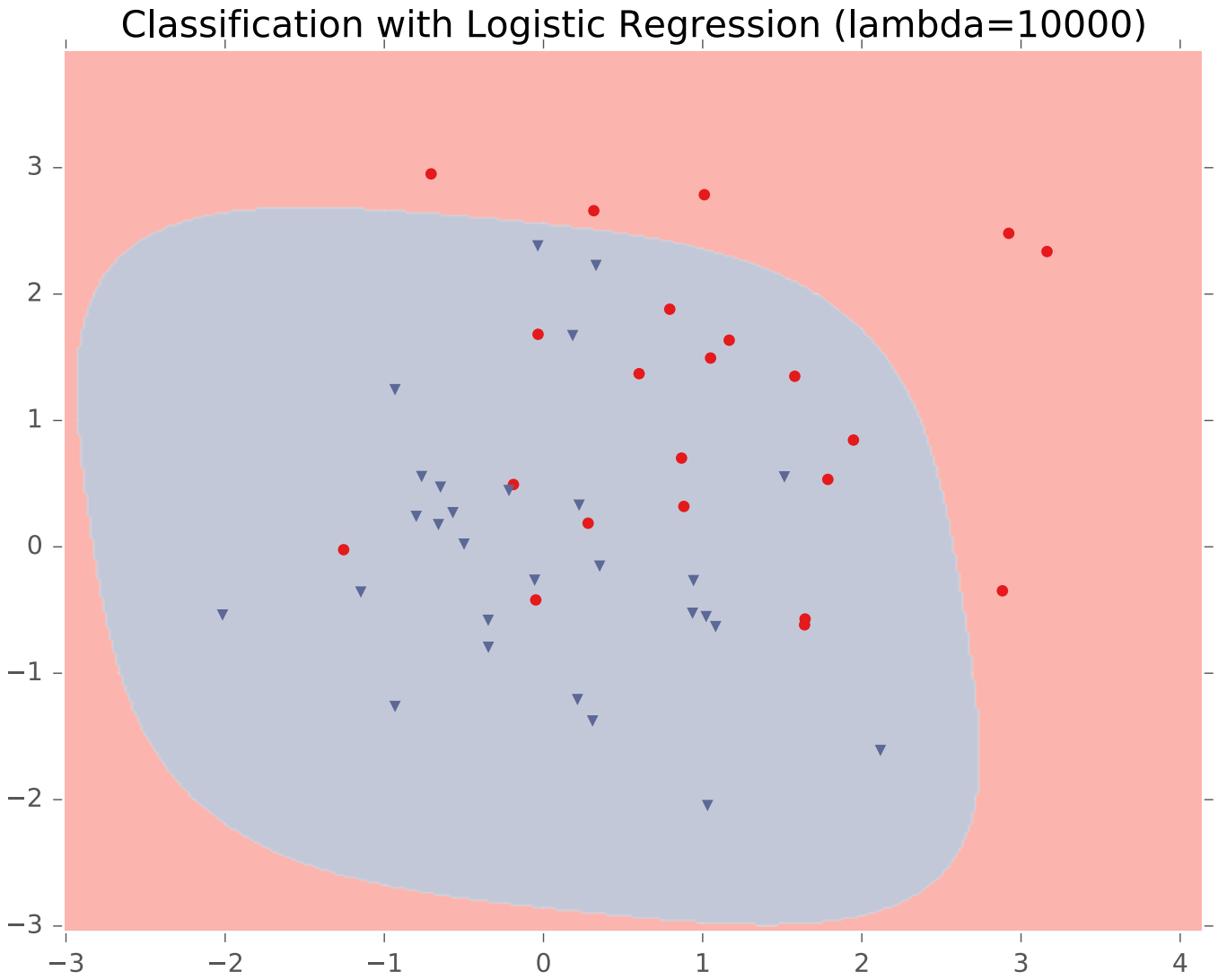
Example: Logistic Regression



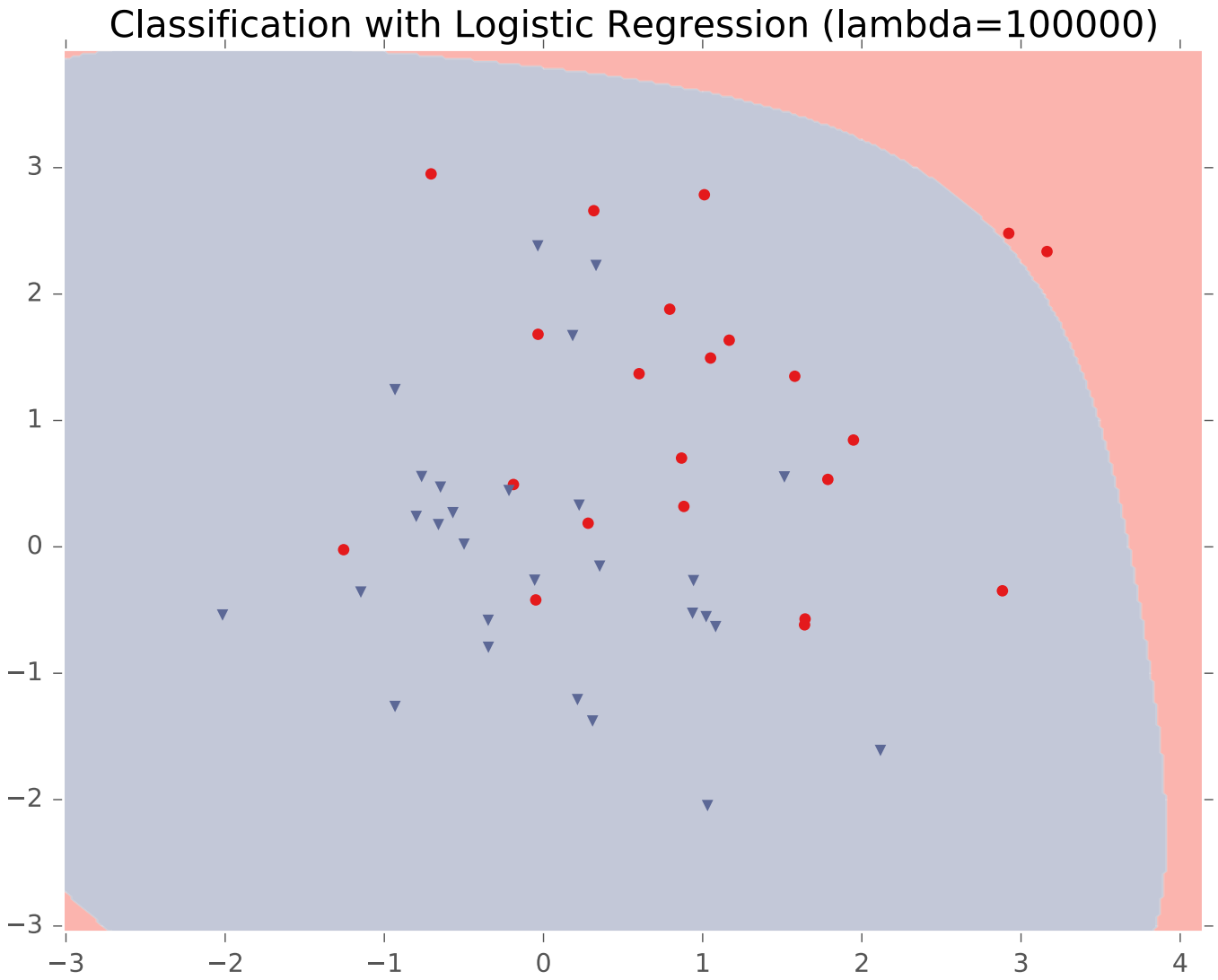
Example: Logistic Regression



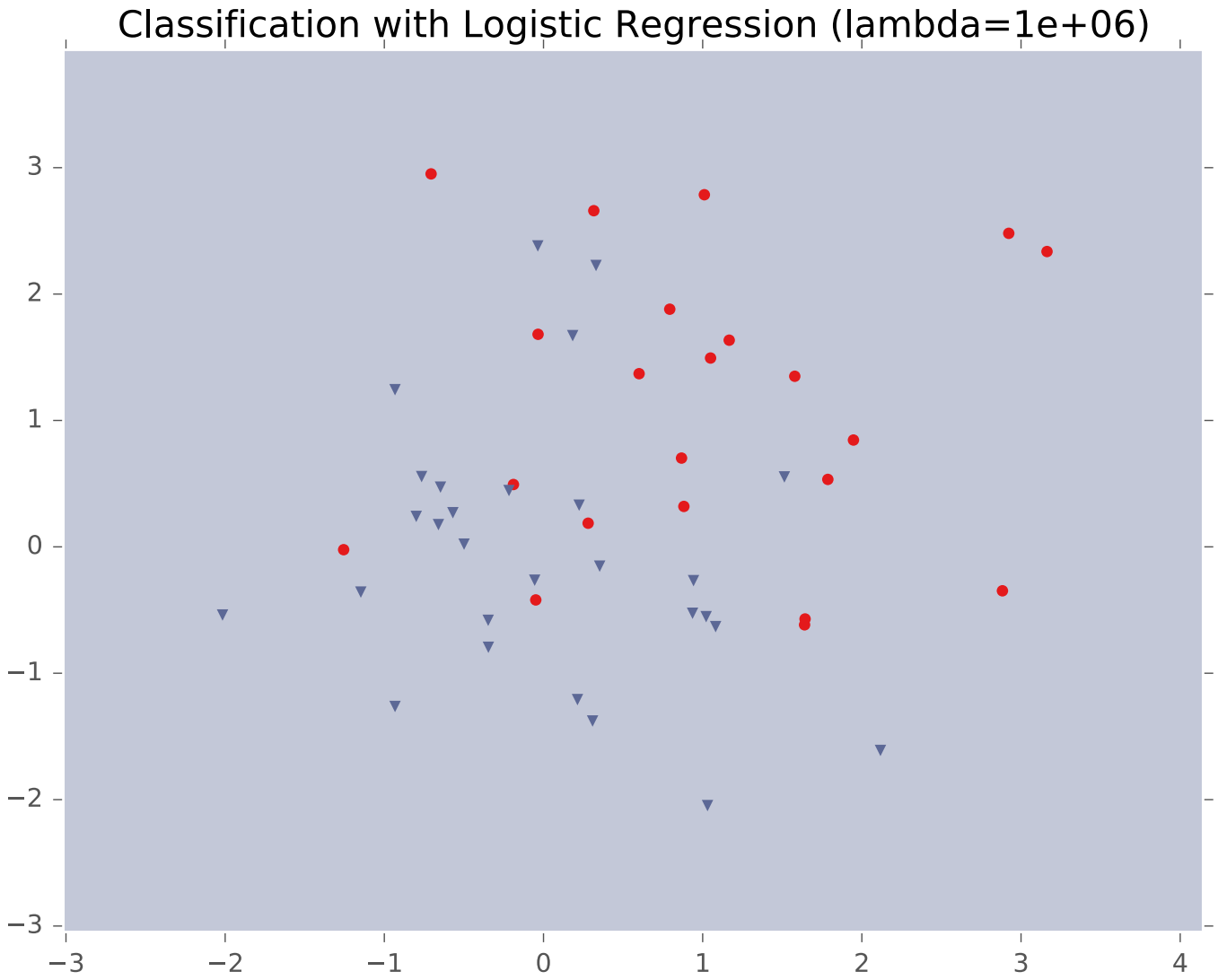
Example: Logistic Regression



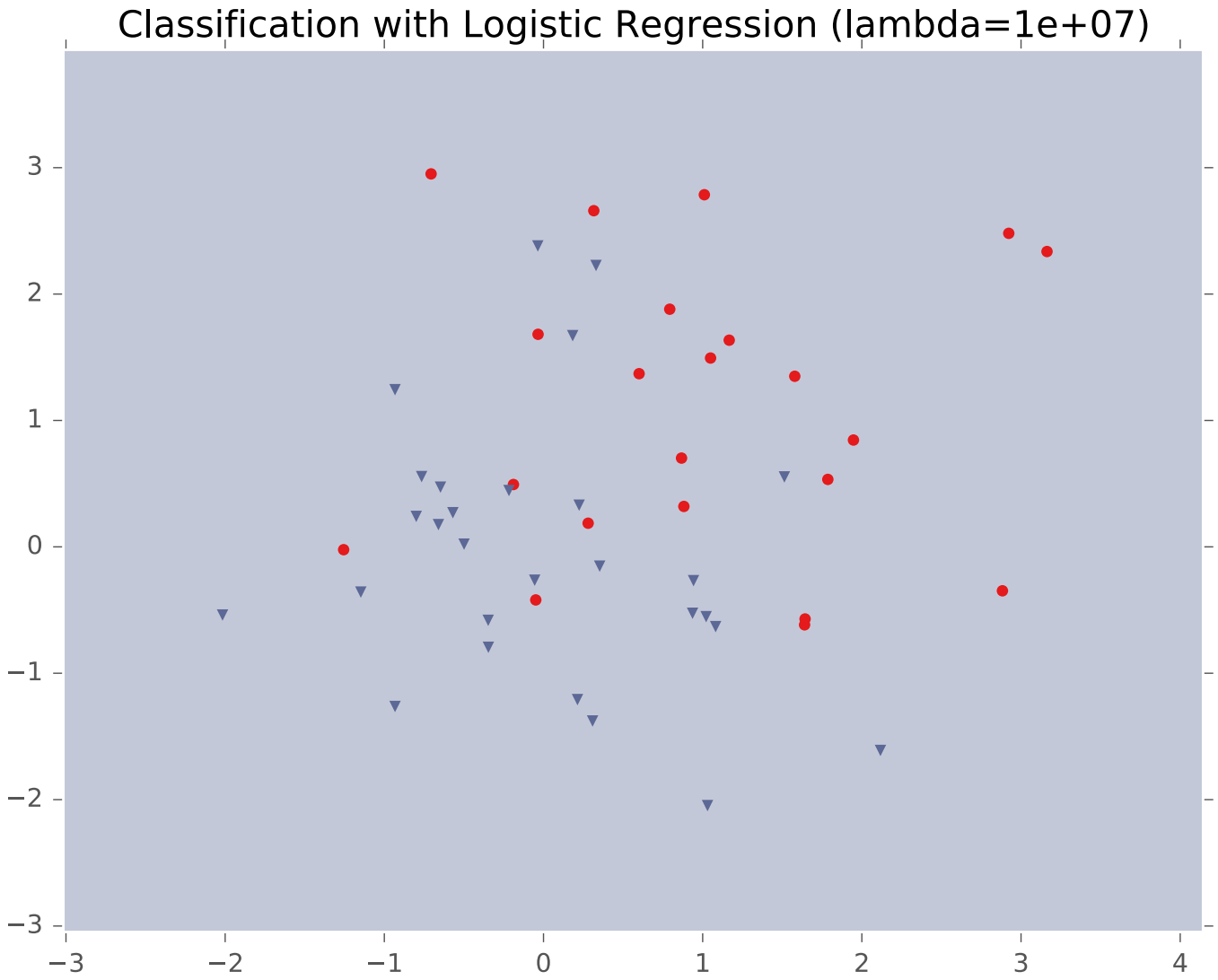
Example: Logistic Regression



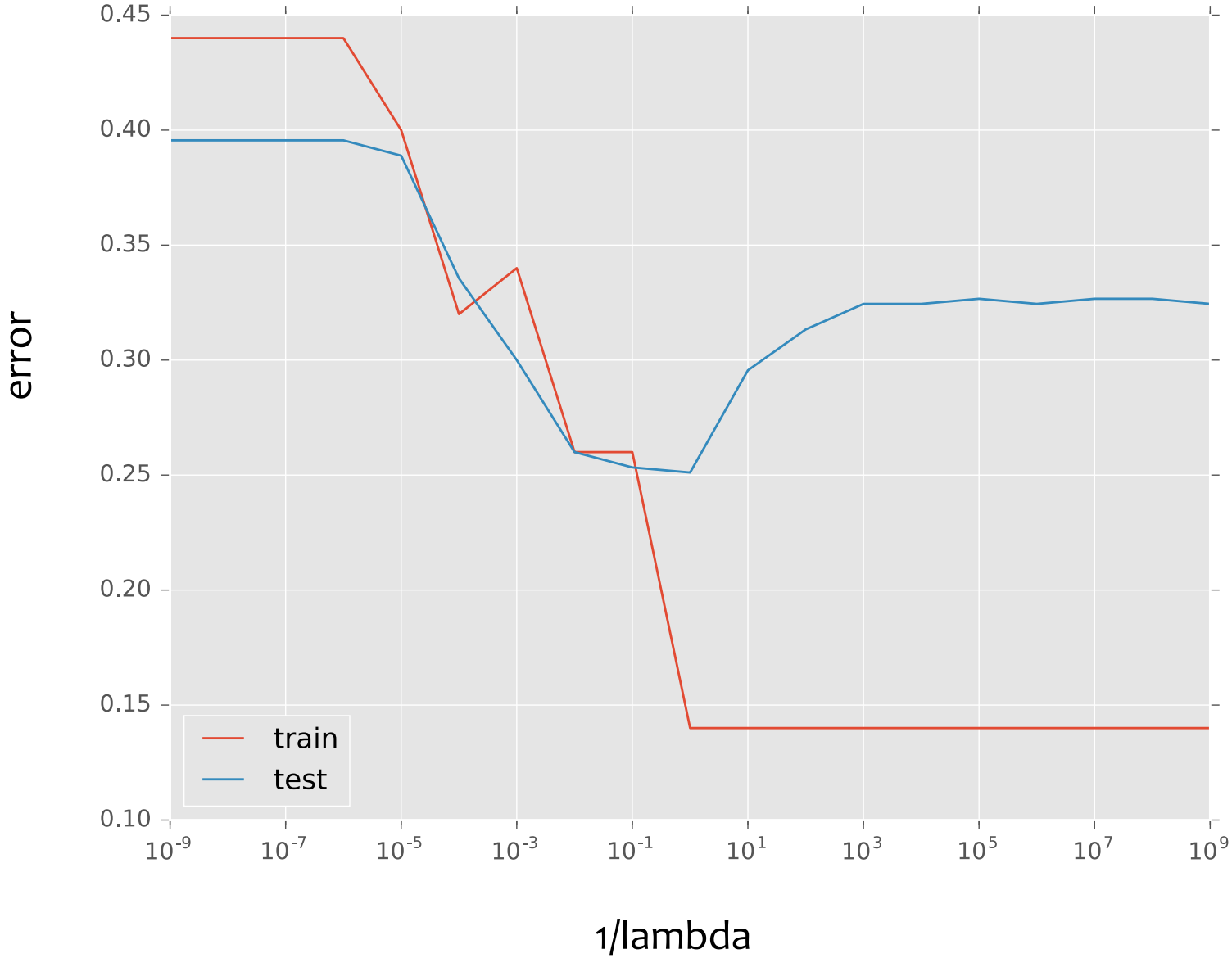
Example: Logistic Regression



Example: Logistic Regression



Example: Logistic Regression



Takeaways

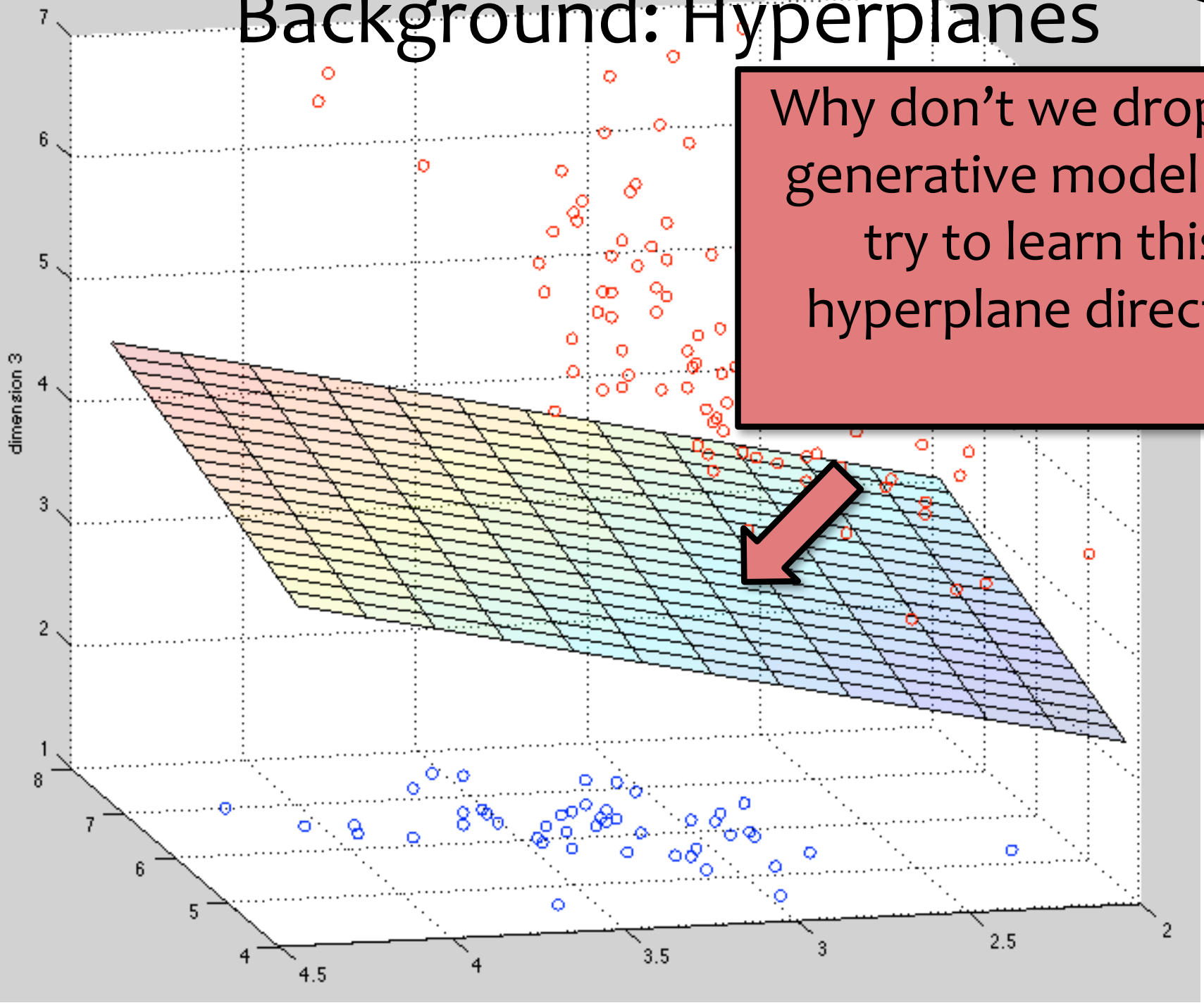
1. **Nonlinear basis functions** allow **linear models** (e.g. Linear Regression, Logistic Regression) to capture **nonlinear** aspects of the original input
2. Nonlinear features are **require no changes to the model** (i.e. just preprocessing)
3. **Regularization** helps to avoid **overfitting**
4.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

THE PERCEPTRON ALGORITHM

Background: Hyperplanes

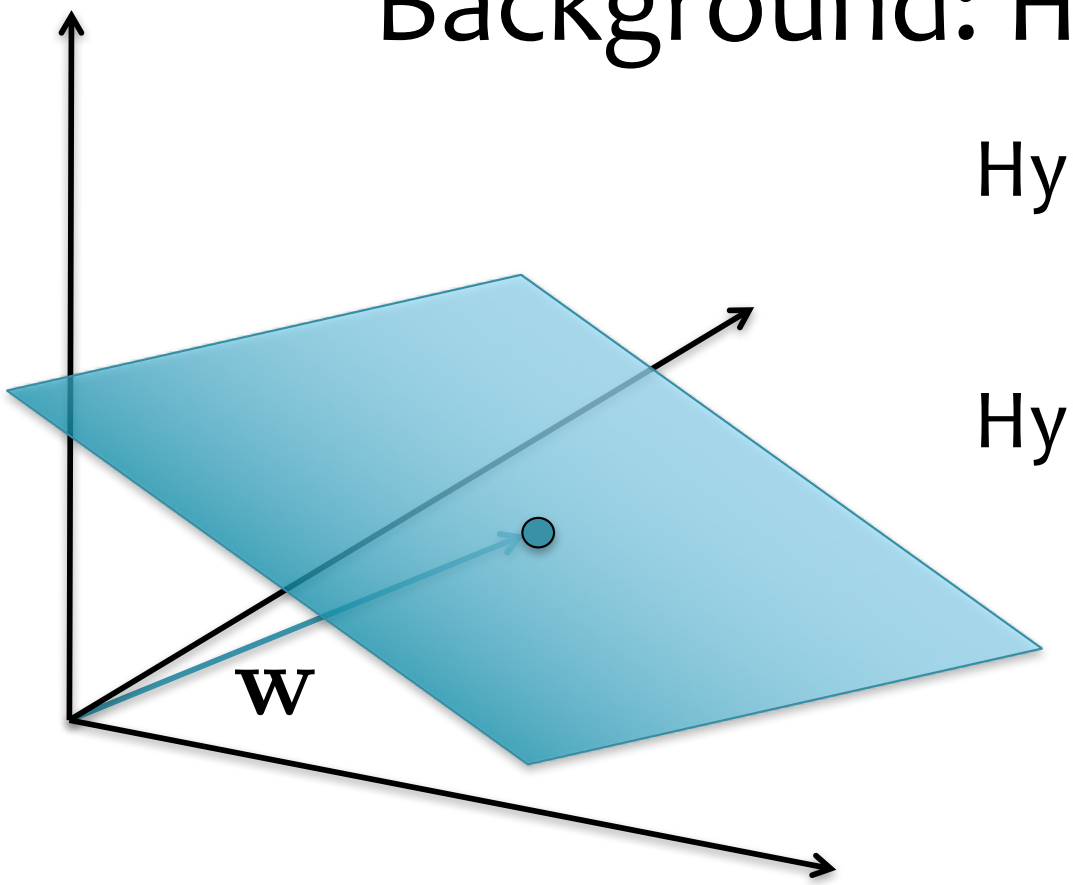
Recall…

Why don't we drop the generative model and try to learn this hyperplane directly?



Background: Hyperplanes

Recall…



Hyperplane (Definition 1):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = b\}$$

Hyperplane (Definition 2):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0 \text{ and } \mathrm{x}_0 = 1\}$$

Half-spaces:

$$\mathcal{H}^+ = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} > 0 \text{ and } \mathrm{x}_0 = 1\}$$

$$\mathcal{H}^- = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} < 0 \text{ and } \mathrm{x}_0 = 1\}$$

Recall…

d: Hyperplanes

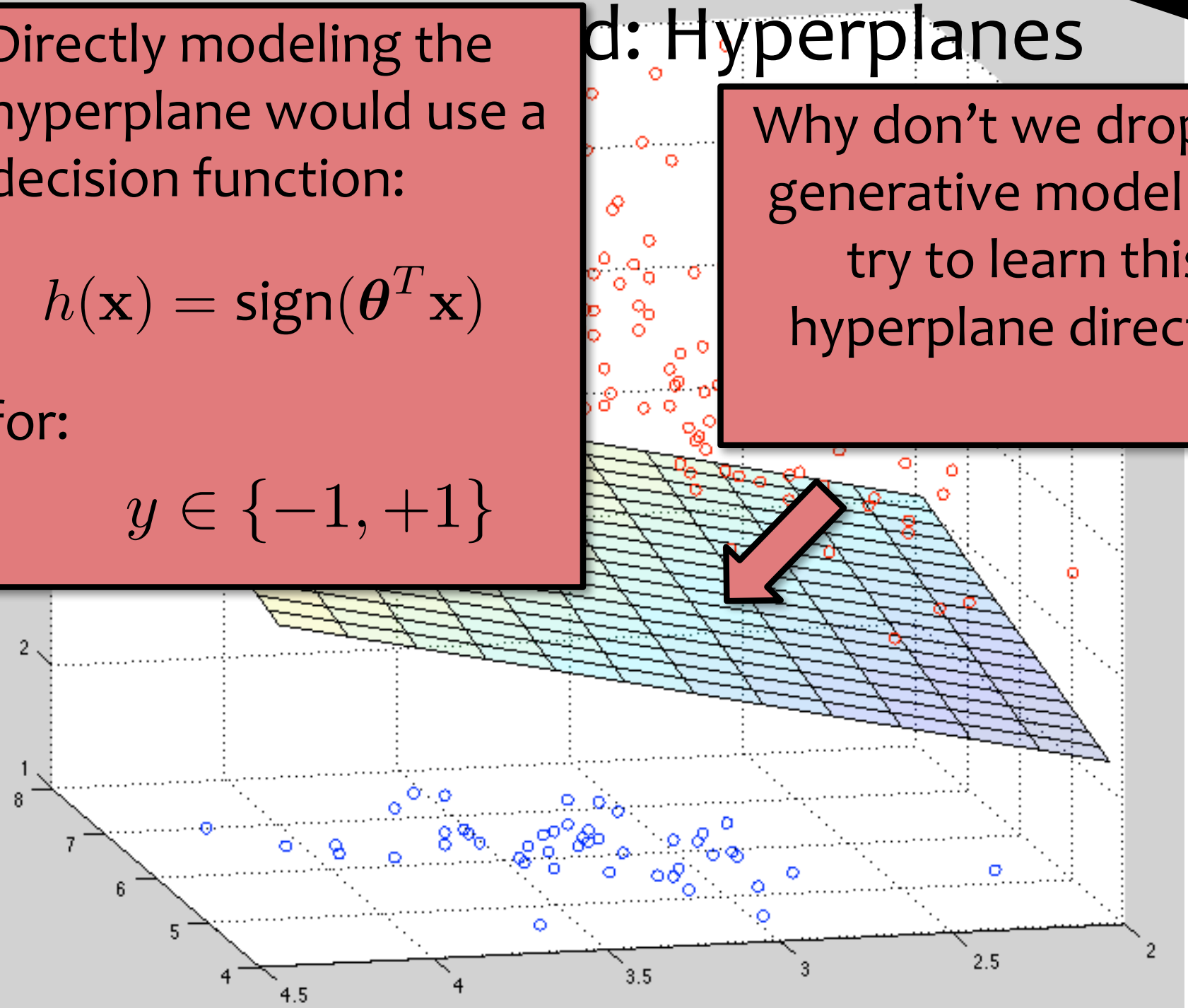
Directly modeling the hyperplane would use a decision function:

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

for:

$$y \in \{-1, +1\}$$

Why don't we drop the generative model and try to learn this hyperplane directly?



Online Learning

For i = 1, 2, 3, … **:**

- **Receive** an unlabeled instance $\mathbf{x}^{(i)}$
- **Predict** y' = h($\mathbf{x}^{(i)}$)
- **Receive** true label $y^{(i)}$
  Check for correctness (y' == $y^{(i)}$)

Goal:

- **Minimize** the number of **mistakes**

Online Learning: Motivation

Examples

1. Email classification (distribution of both spam and regular mail changes over time, but the target function stays fixed - last year's spam still looks like spam).
2. Recommendation systems. Recommending movies, etc.
3. Predicting whether a user will be interested in a new news article or not.
4. Ad placement in a new market.

Slide from Nina Balcan

Perceptron Algorithm

Data: Inputs are continuous vectors of length K. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^K$ and $y \in \{+1, -1\}$

Prediction: Output determined by hyperplane.

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

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Learning: Iterative procedure:

- **while** not converged
  - **receive** next example ($\mathbf{x}^{(i)}$, $y^{(i)}$)
  - **predict** y' = h($\mathbf{x}^{(i)}$)
  - **if** positive mistake: **add** $\mathbf{x}^{(i)}$ to parameters
  - **if** negative mistake: **subtract** $\mathbf{x}^{(i)}$ from parameters

Perceptron Algorithm

Data: Inputs are continuous vectors of length K. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^K$ and $y \in \{+1, -1\}$

Prediction: Output determined by hyperplane.

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

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Learning:

Algorithm 1 Perceptron Learning Algorithm (Online)

1: **procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots\}$)
2: $\quad \boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
3: $\quad$ **for** $i \in \{1, 2, \ldots\}$ **do** ▷ For each example
4: $\quad\quad \hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
5: $\quad\quad$ **if** $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
6: $\quad\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
7: $\quad$ **return** $\boldsymbol{\theta}$

Perceptron Algorithm: Example

Example: $(-1,2)$ − ✗

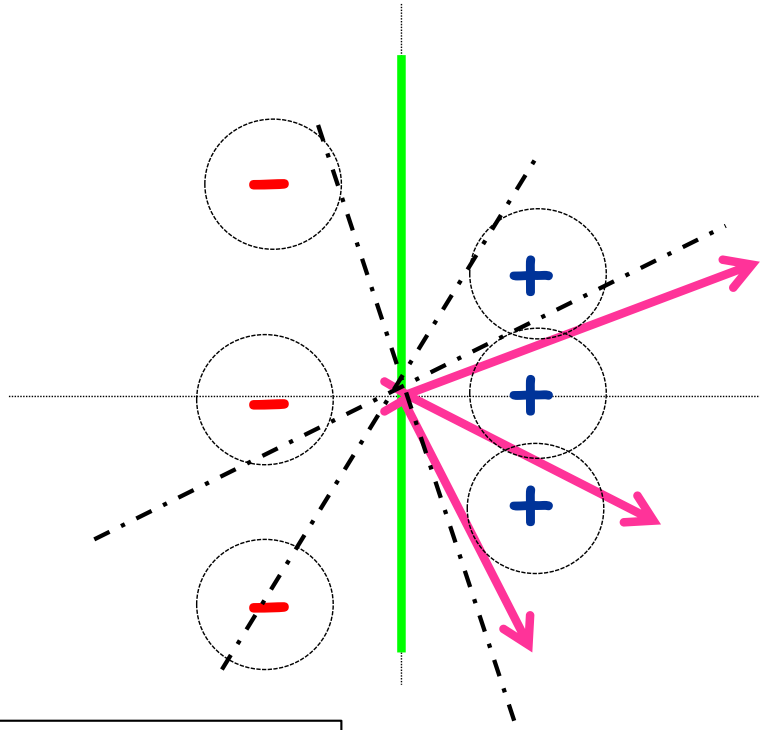
$(1,0)$ + ✓

$(1,1)$ + ✗

$(-1,0)$ − ✓

$(-1,-2)$ − ✗

$(1,-1)$ + ✓



Algorithm:

- Set t=1, start with all-zeroes weight vector w_1.
- Given example x, predict positive iff $\theta_t \cdot x \geq 0$.
- On a mistake, update as follows:
  - Mistake on positive, update $\theta_{t+1} \leftarrow \theta_t + x$
  - Mistake on negative, update $\theta_{t+1} \leftarrow \theta_t - x$

$\theta_1 = (0,0)$

$\theta_2 = \theta_1 - (-1,2) = (1,-2)$

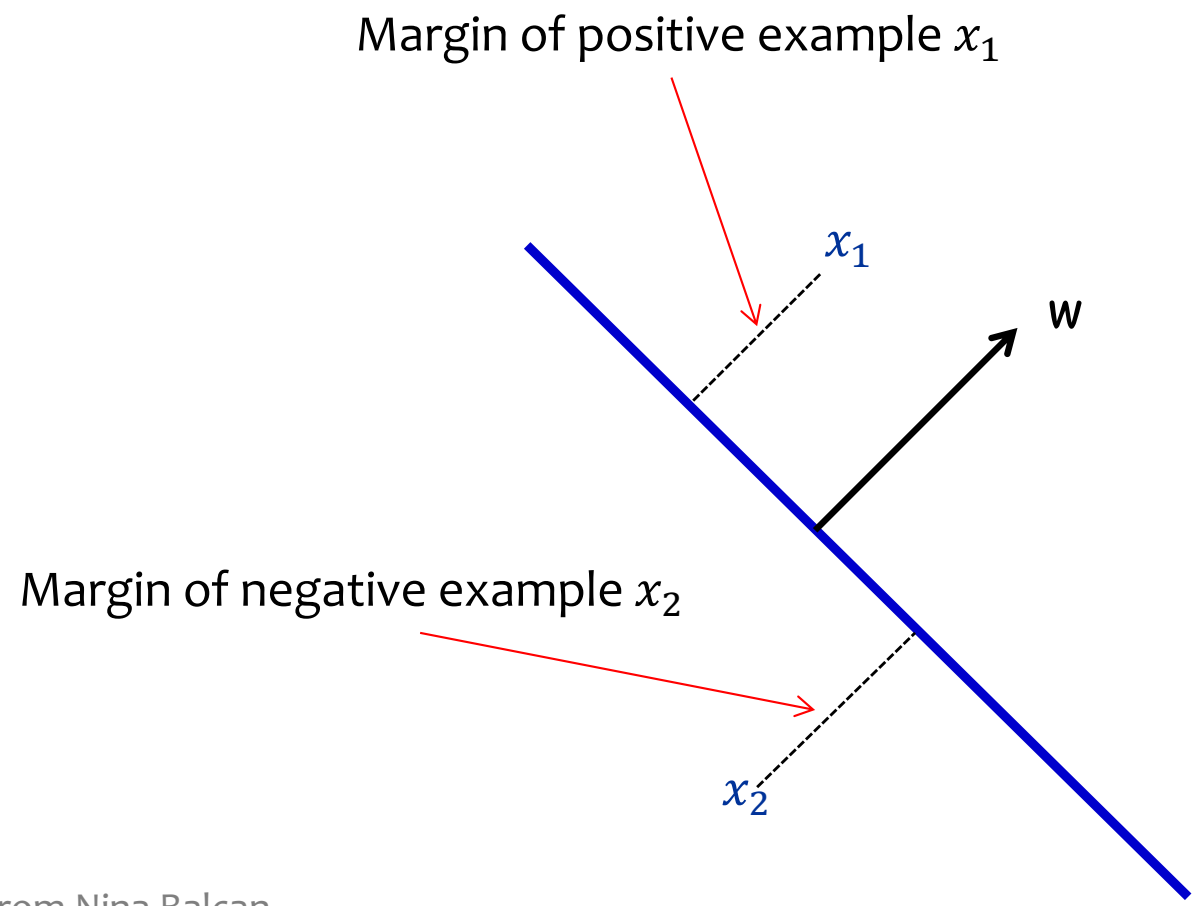
$\theta_3 = \theta_2 + (1,1) = (2,-1)$

$\theta_4 = \theta_3 - (-1,-2) = (3,1)$

Slide adapted from Nina Balcan

Geometric Margin

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

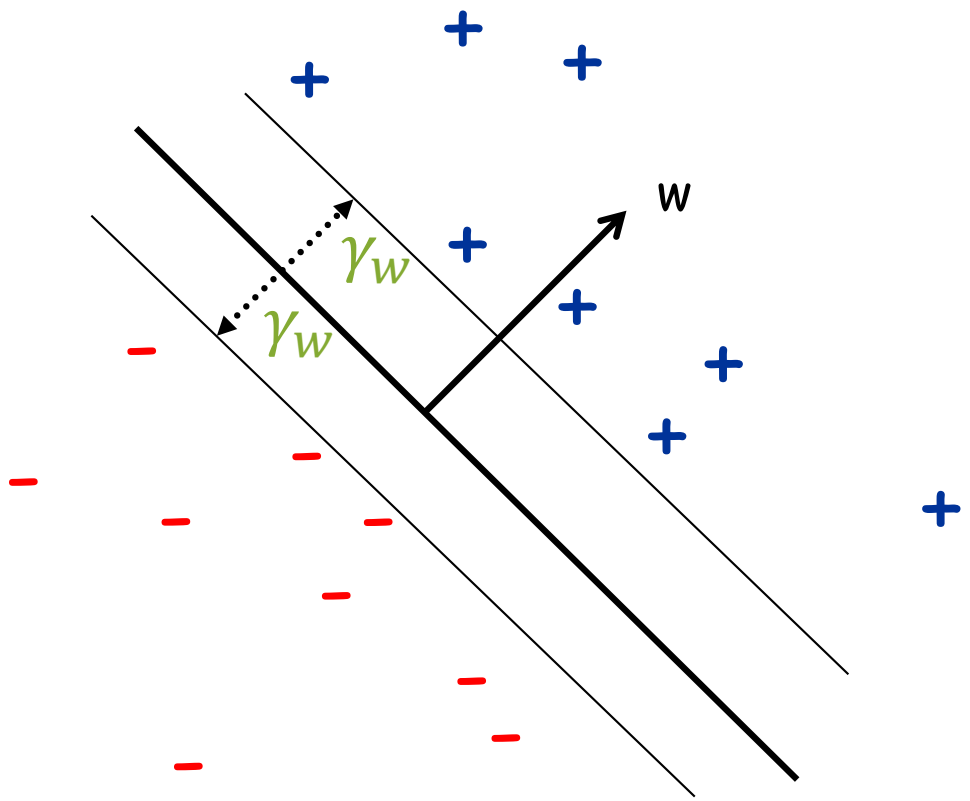


Slide from Nina Balcan

Geometric Margin

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.



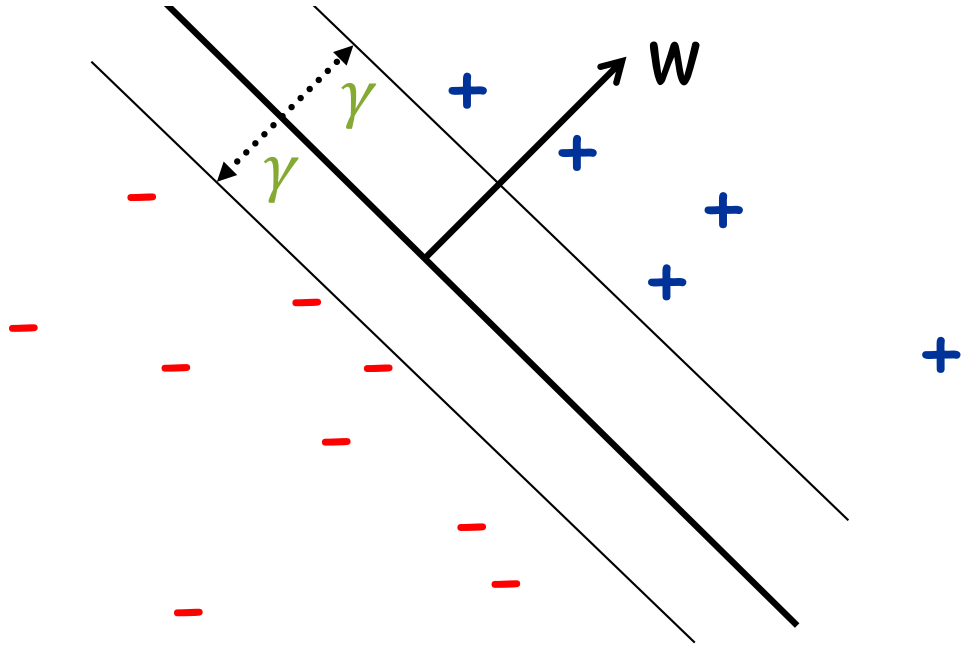
Slide from Nina Balcan

Geometric Margin

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.

Definition: The margin γ of a set of examples S is the maximum γ_w over all linear separators w.



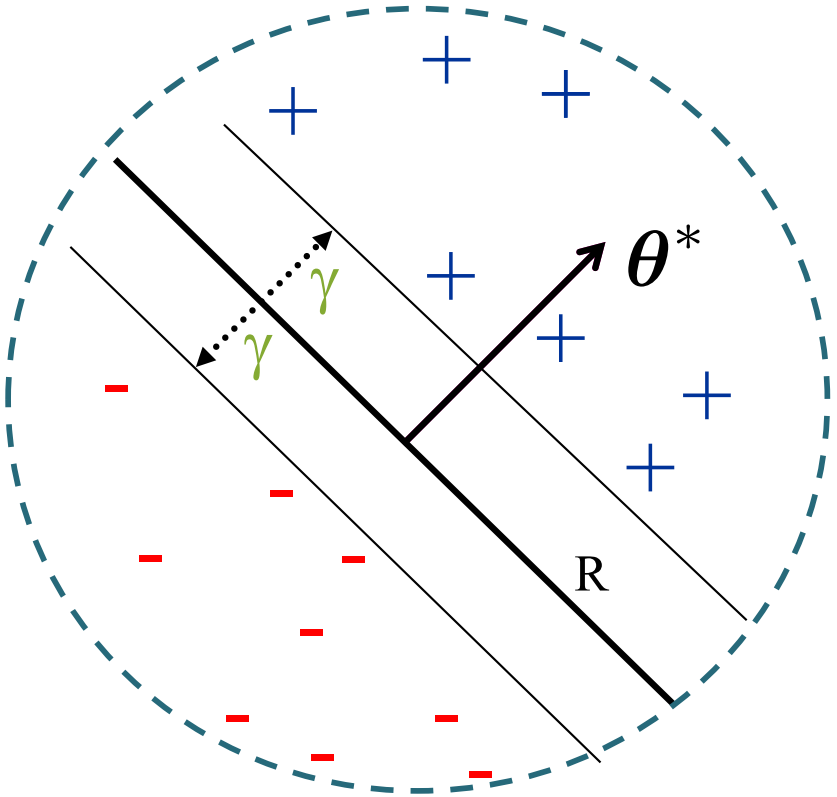
Slide from Nina Balcan

Analysis: Perceptron

Perceptron Mistake Bound

Guarantee: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



Slide adapted from Nina Balcan

Analysis: Perceptron

Perceptron Mistake Bound

Theorem 0.1 (Block (1962), Novikoff (1962)).

Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.

Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$

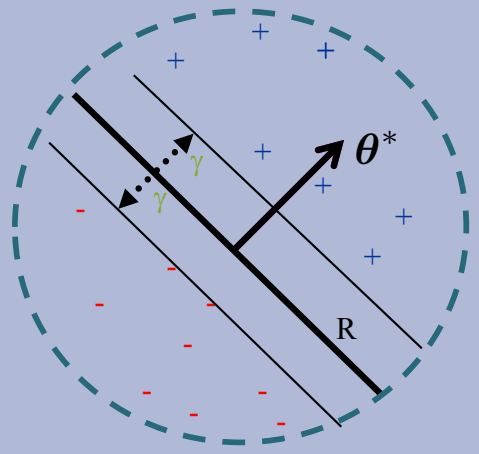


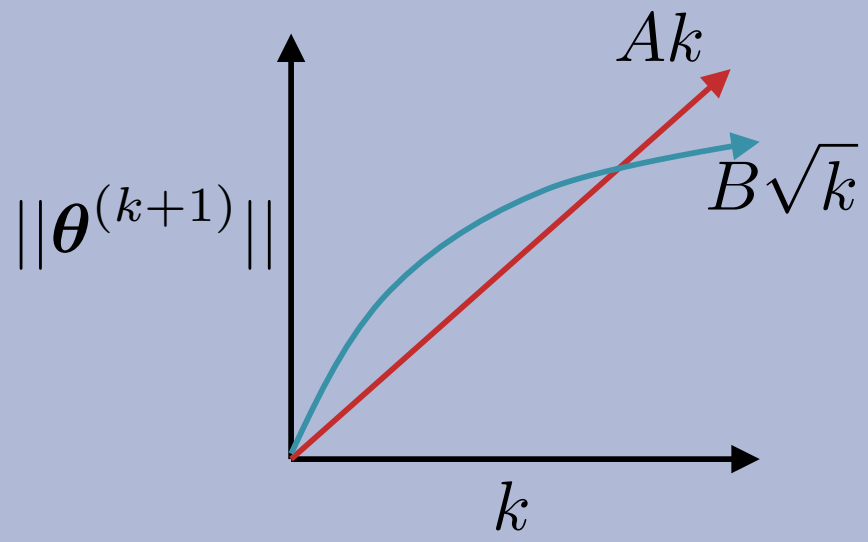
Figure from Nina Balcan

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

We will show that there exist constants A and B s.t.

$$Ak \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$$



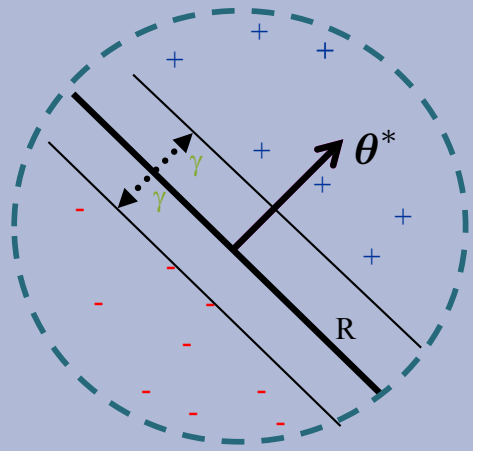
Analysis: Perceptron

Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$



Algorithm 1 Perceptron Learning Algorithm (Online)

procedure PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots\}$)
  $\boldsymbol{\theta} \leftarrow \mathbf{0}, k = 1$ ▷ Initialize parameters
  for $i \in \{1, 2, \ldots\}$ **do** ▷ For each example
    if $y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$ **then** ▷ If mistake
      $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}$ ▷ Update parameters
      $k \leftarrow k + 1$
  return $\boldsymbol{\theta}$

Analysis: Perceptron

Whiteboard:
Proof of Perceptron Mistake Bound

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 1: for some A, $Ak \leq ||\boldsymbol{\theta}^{(k+1)}||$

$$\boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* = (\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)})\boldsymbol{\theta}^*$$

by Perceptron algorithm update

$$= \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)})$$

$$\geq \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + \gamma$$

by assumption

$$\Rightarrow \boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* \geq k\gamma$$

by induction on k since $\theta^{(1)} = \mathbf{0}$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \geq k\gamma$$

since $||\mathbf{w}|| \times ||\mathbf{u}|| \geq \mathbf{w} \cdot \mathbf{u}$ and $||\theta^*|| = 1$

Cauchy-Schwartz inequality

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 2: for some B, $||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$

$$||\boldsymbol{\theta}^{(k+1)}||^2 = ||\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}||^2$$

by Perceptron algorithm update

$$= ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2 + 2y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)})$$

$$\leq ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2$$

since kth mistake $\Rightarrow y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$

$$= ||\boldsymbol{\theta}^{(k)}||^2 + R^2$$

since $(y^{(i)})^2||\mathbf{x}^{(i)}||^2 = ||\mathbf{x}^{(i)}||^2 = R^2$ by assumption and $(y^{(i)})^2 = 1$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}||^2 \leq kR^2$$

by induction on k since $(\theta^{(1)})^2 = 0$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 3: Combining the bounds finishes the proof.

$$k\gamma \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

$$\Rightarrow k \leq (R/\gamma)^2$$

The total number of mistakes must be less than this

(Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the "batch" setting in contrast to the "online" setting that we've discussed so far.

Algorithm 1 Perceptron Learning Algorithm (Batch)

procedure PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$)
  $\boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
  while not converged **do**
    for $i \in \{1, 2, \ldots, N\}$ **do** ▷ For each example
      $\hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
      if $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
  return $\boldsymbol{\theta}$

(Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the "batch" setting in contrast to the "online" setting that we've discussed so far.

Discussion:

The Batch Perceptron Algorithm can be derived in two ways.

1. By extending the online Perceptron algorithm to the batch setting (as mentioned above)
2. By applying **Stochastic Gradient Descent (SGD)** to minimize a so-called **Hinge Loss** on a linear separator

Extensions of Perceptron

- **Kernel Perceptron**
  - Choose a kernel K(x', x)
  - Apply the **kernel trick** to Perceptron
  - Resulting algorithm is **still very simple**
- **Structured Perceptron**
  - Basic idea can also be applied when **y** ranges over an exponentially large set
  - Mistake bound **does not** depend on the size of that set

Matching Game

Goal: Match the Algorithm to its Update Rule

1. SGD for Logistic Regression

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

2. Least Mean Squares

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

3. Perceptron

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

4.

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

5.

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

6.

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

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

Summary: Perceptron

- Perceptron is a **linear classifier**
- **Simple learning algorithm**: when a mistake is made, add / subtract the features
- For linearly separable and inseparable data, we can **bound the number of mistakes** (geometric argument)
- Extensions support nonlinear separators and structured prediction

DISCRIMINATIVE AND GENERATIVE CLASSIFIERS

Generative vs. Discriminative

- **Generative Classifiers:**
  - Example: Naïve Bayes
  - Define a joint model of the observations **x** and the labels y: $p(\boldsymbol{x}, y)$
  - Learning maximizes (joint) likelihood
  - Use Bayes' Rule to classify based on the posterior:

$$p(y|\mathbf{x}) = p(\mathbf{x}|y)p(y)/p(\mathbf{x})$$

- **Discriminative Classifiers:**
  - Example: Logistic Regression
  - Directly model the conditional: $p(y|\mathbf{x})$
  - Learning maximizes conditional likelihood

Generative vs. Discriminative

Whiteboard

– Contrast: To model p(x) or not to model p(x)?

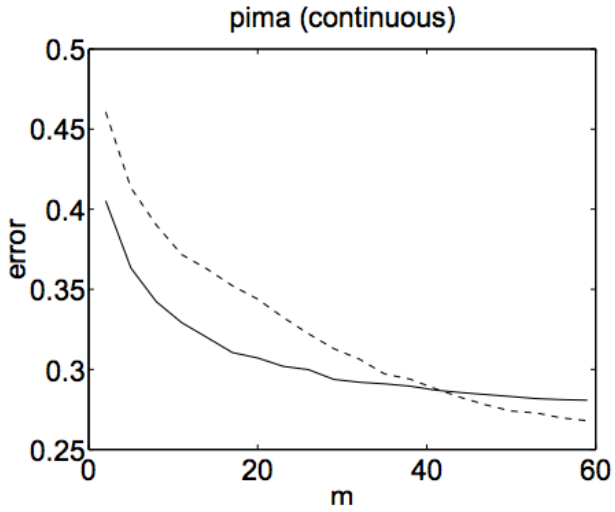
Generative vs. Discriminative

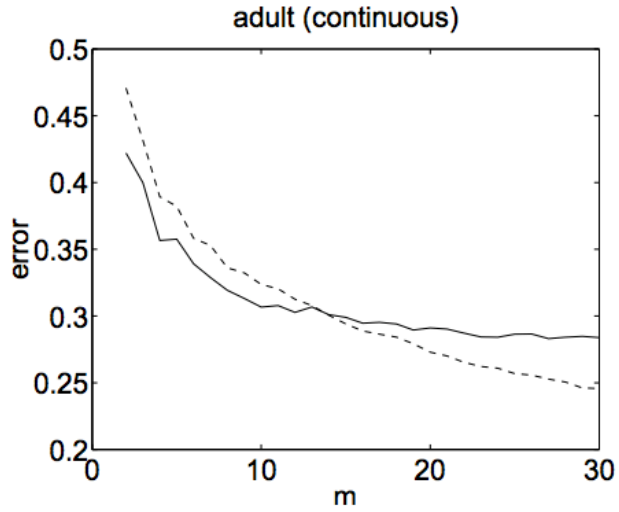
Finite Sample Analysis (Ng & Jordan, 2002)

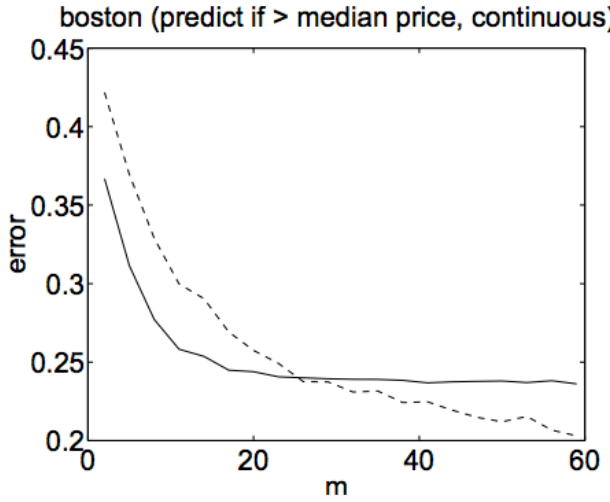
[Assume that we are learning from a finite training dataset]

If model assumptions are correct: Naive Bayes is a more efficient learner (requires fewer samples) than Logistic Regression

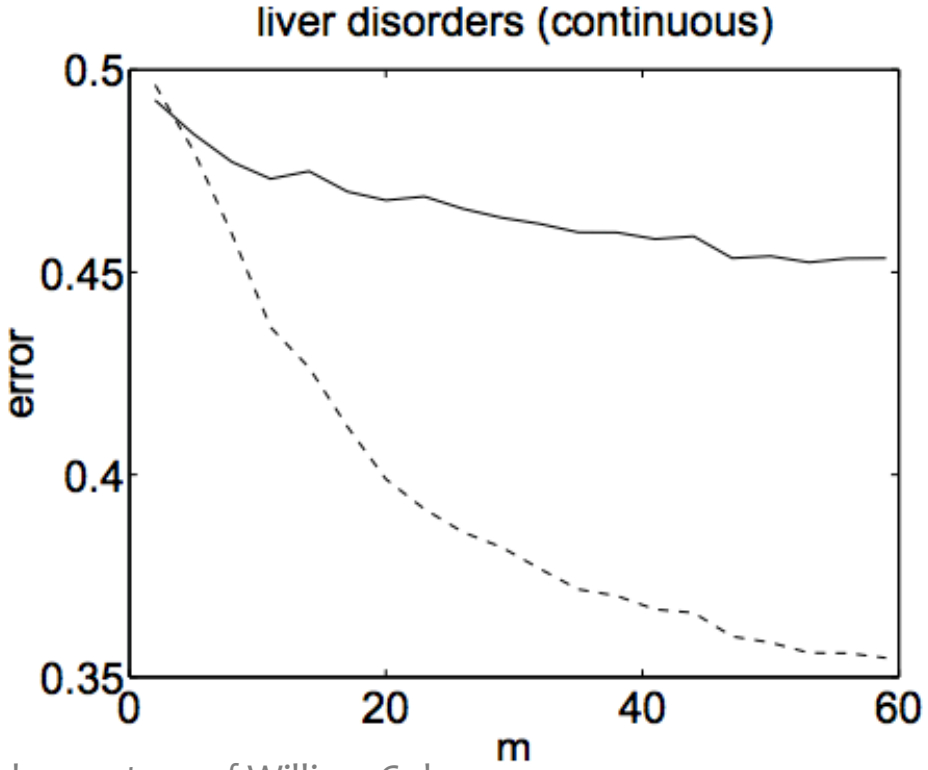
If model assumptions are incorrect: Logistic Regression has lower asymtotic error, and does better than Naïve Bayes

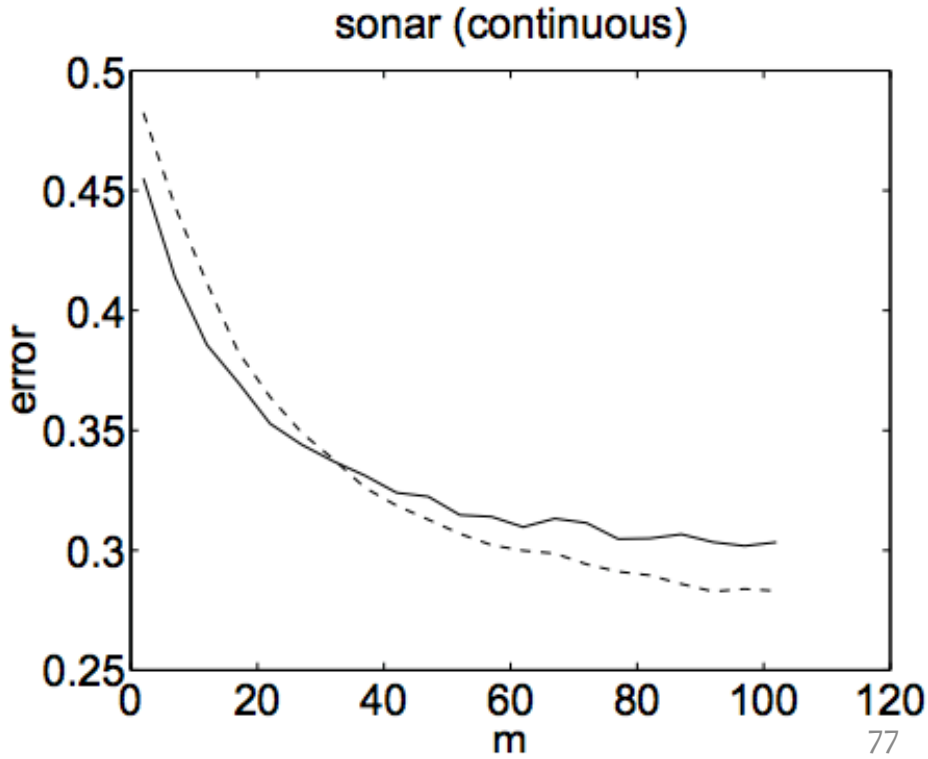






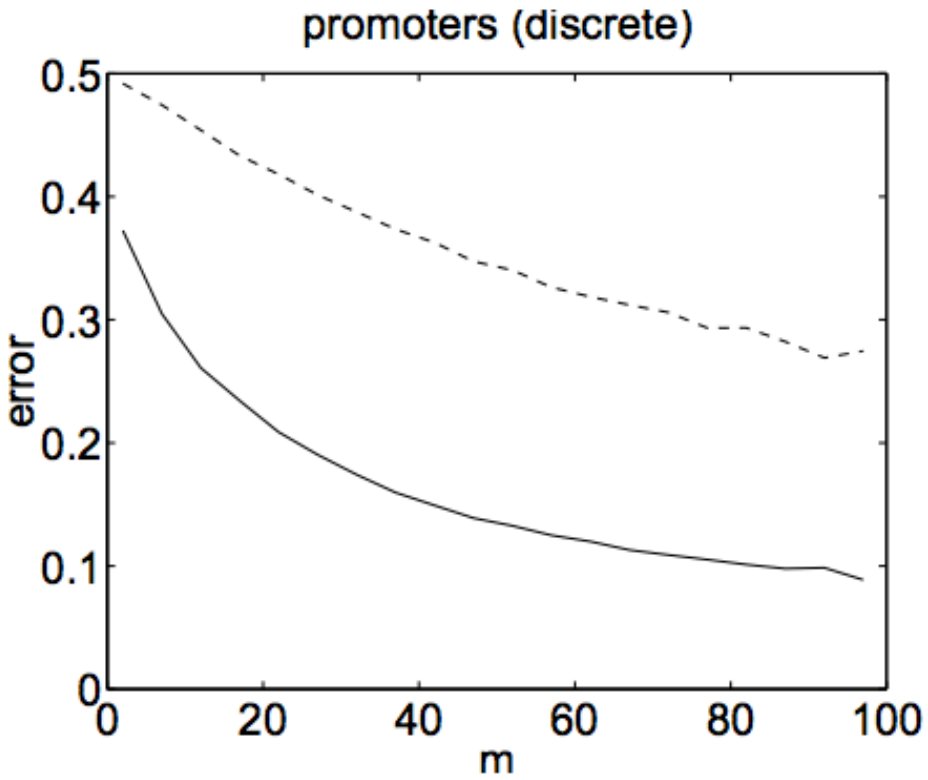
solid: NB dashed: LR

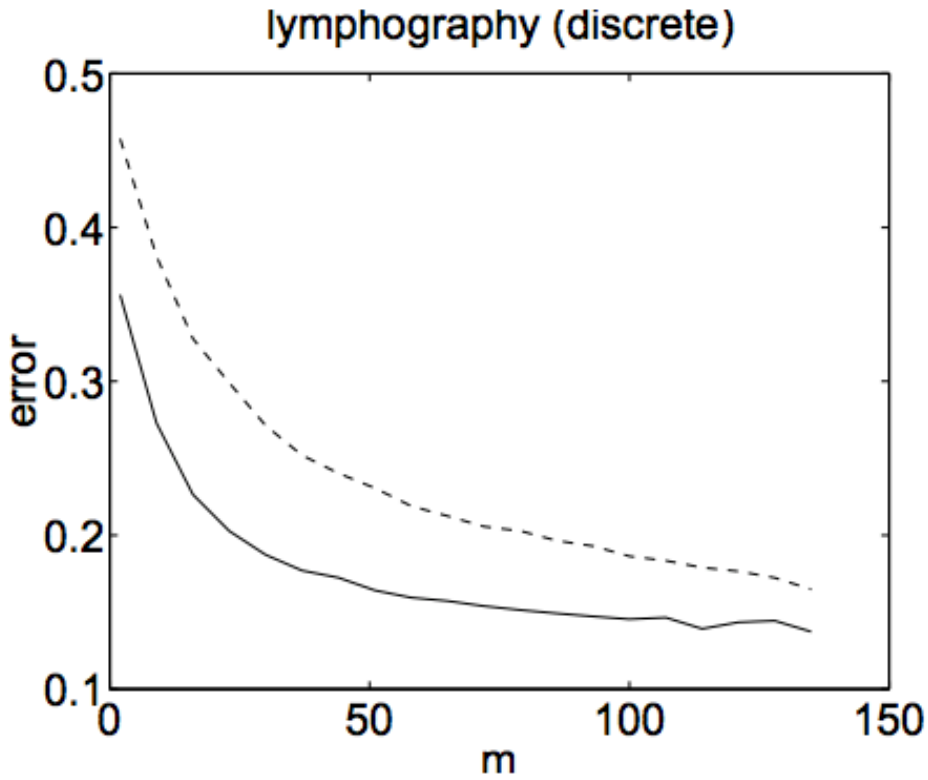




Slide courtesy of William Cohen

solid: NB dashed: LR





Naïve Bayes makes stronger assumptions about the data but needs fewer examples to estimate the parameters

"On Discriminative vs Generative Classifiers: …." Andrew Ng and Michael Jordan, NIPS 2001.

Slide courtesy of William Cohen

Generative vs. Discriminative

Learning (Parameter Estimation)

Naïve Bayes:
Parameters are decoupled → Closed form solution for MLE

Logistic Regression:
Parameters are coupled → No closed form solution – must use iterative optimization techniques instead

Naïve Bayes vs. Logistic Reg.

Learning (MAP Estimation of Parameters)

Bernoulli Naïve Bayes:
Parameters are probabilities → Beta prior (usually) pushes probabilities away from zero / one extremes

Logistic Regression:
Parameters are not probabilities → Gaussian prior encourages parameters to be close to zero

(effectively pushes the probabilities away from zero / one extremes)

Naïve Bayes vs. Logistic Reg.

Features

Naïve Bayes:
Features $\boldsymbol{x}$ are assumed to be conditionally independent given y. (i.e. Naïve Bayes Assumption)

Logistic Regression:
No assumptions are made about the form of the features $\boldsymbol{x}$. They can be dependent and correlated in any fashion.