10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Perceptron

+

Kernels

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

Kernel Readings:
Murphy 14.1 – 14.2.4
Bishop 6.1 – 6.2
HTF --
Mitchell --

Matt Gormley
Lecture 11
February 22, 2016

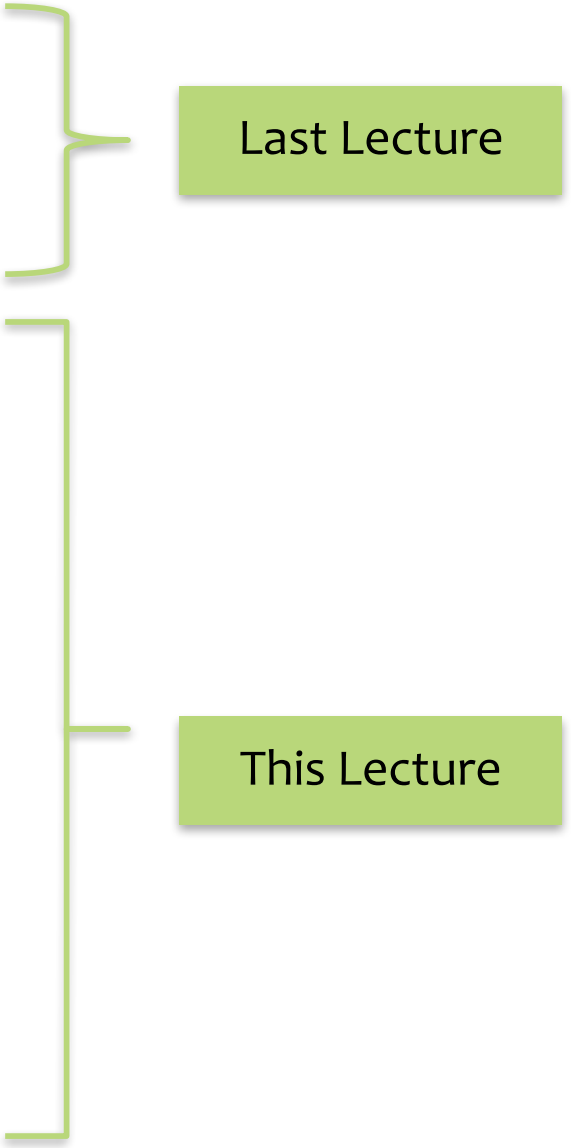
Reminders

- Homework 3: Linear / Logistic Regression
  - Release: Mon, Feb. 13
  - Due: Wed, Feb. 22 at 11:59pm
- Course Survey [5 pts]
  - due Fri, Feb 24 at 11:59pm
- Homework 4: Perceptron / Kernels / SVM
  - Release: Wed, Feb. 22
  - Due: Fri, Mar. 03 at 11:59pm

New due date
(9 days for HW4)

Outline

- **Perceptron**
  - Online Learning
  - Perceptron Algorithm
  - Margin Definitions
  - Perceptron Mistake Bound
- **Kernels**
  - Kernel Perceptron
  - Kernel as a dot product
  - Gram matrix
  - Examples: Polynomial, RBF
- **Support Vector Machine (SVM)**

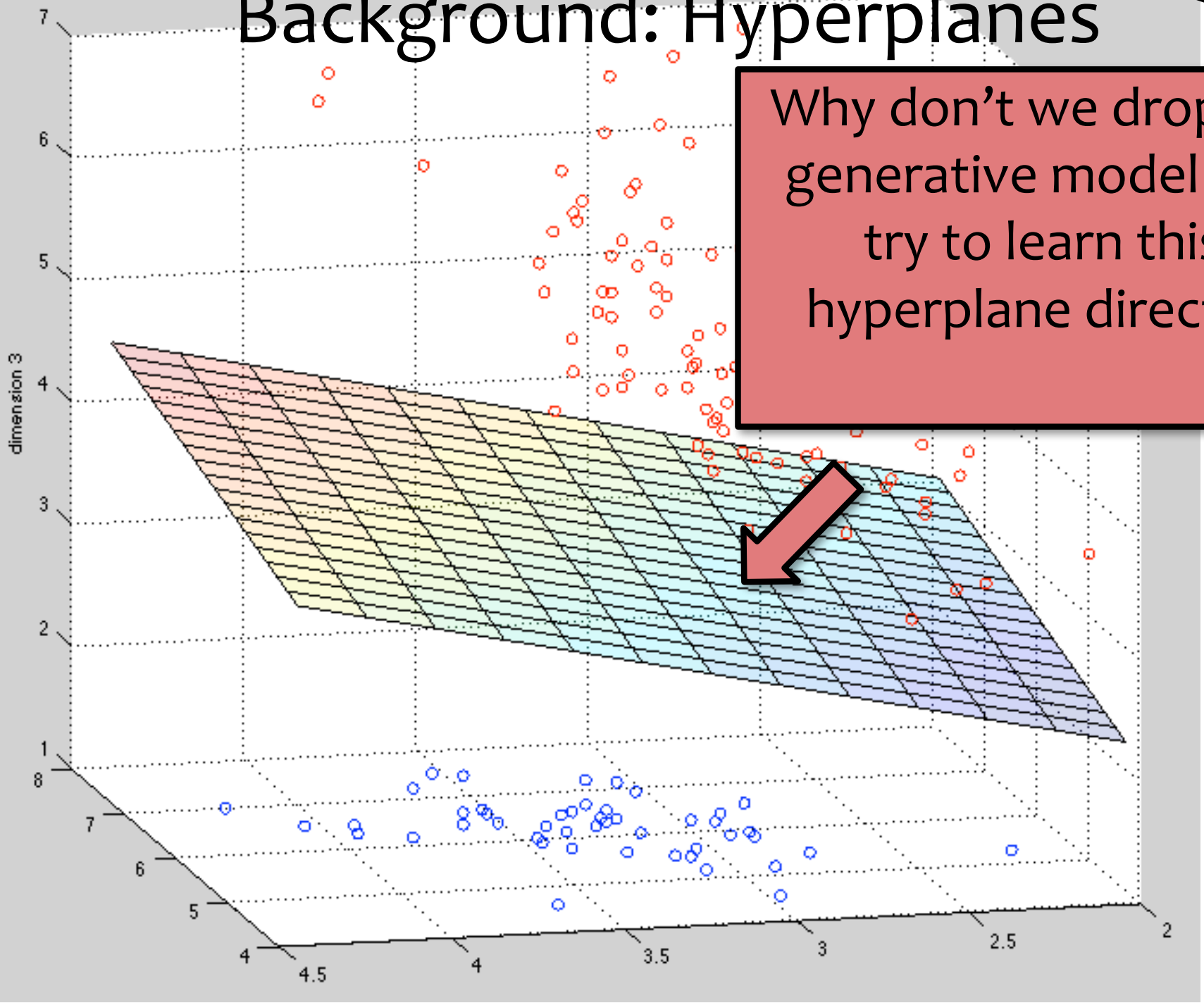


PERCEPTRON

Background: Hyperplanes

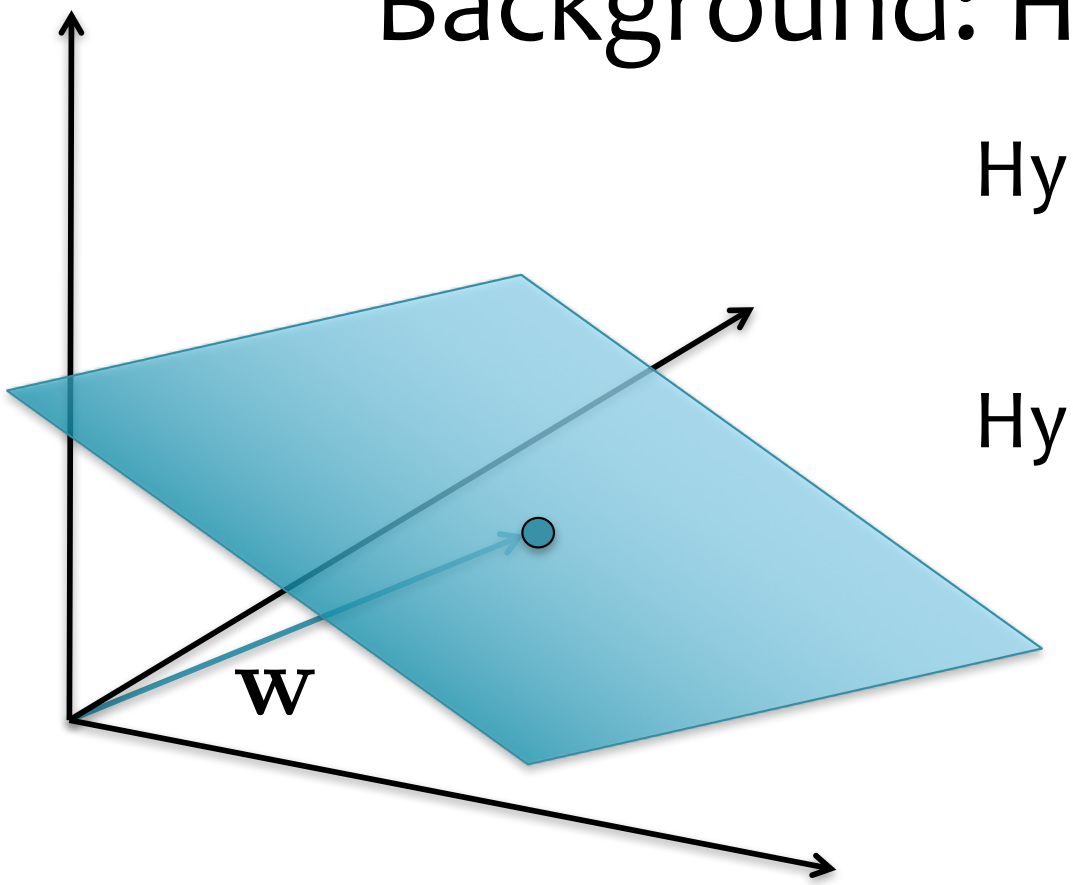
Recall…

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

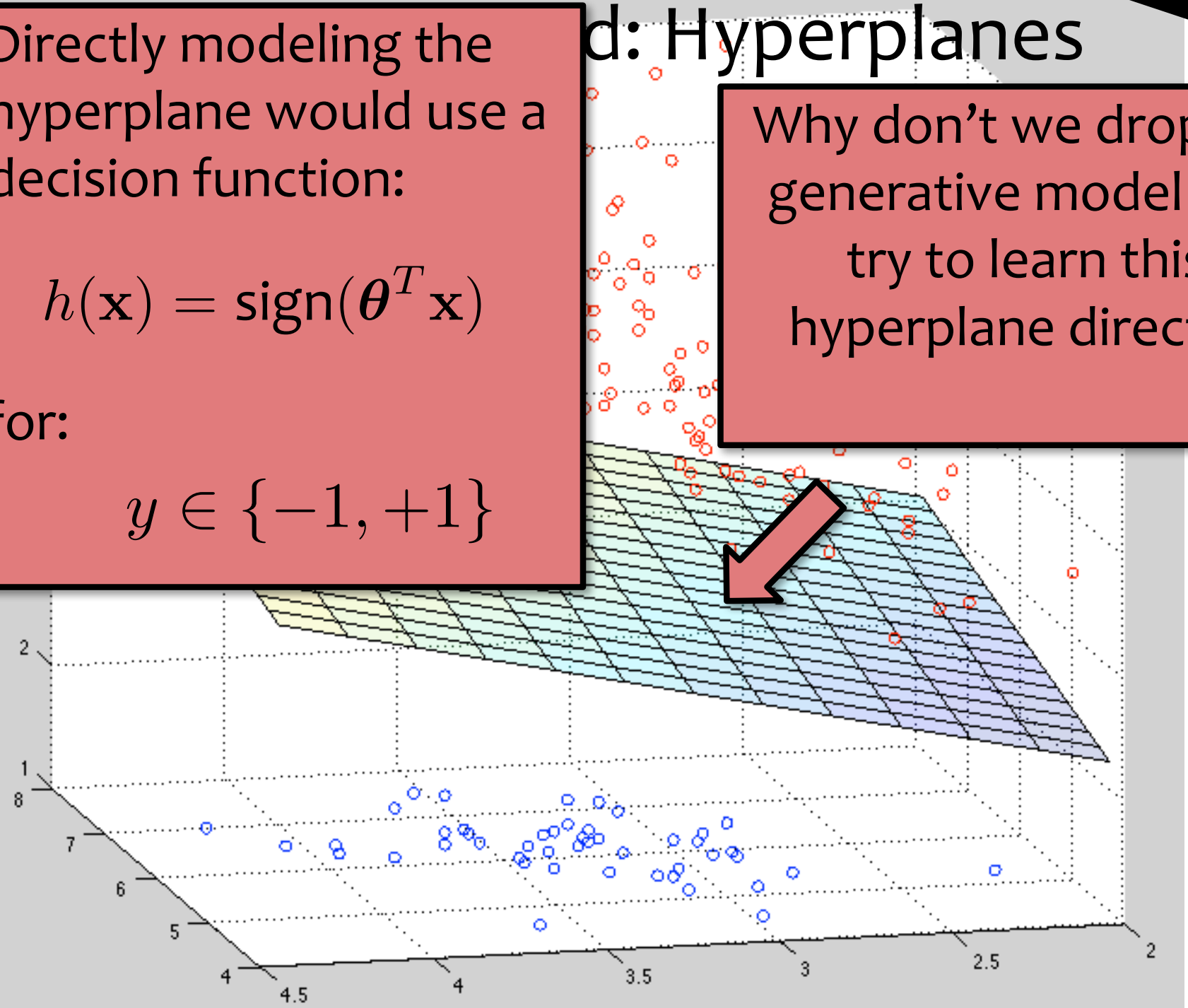
Directly modeling the hyperplane would use a decision function:

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

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: $(-1,2)$ − ✗

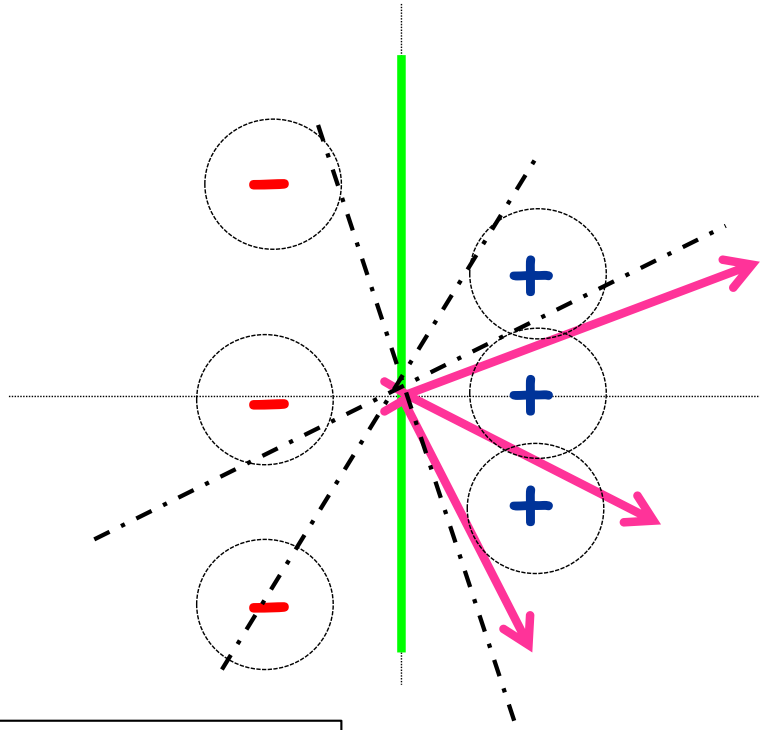
$(1,0)$ + ✓

$(1,1)$ + ✗

$(-1,0)$ − ✓

$(-1,-2)$ − ✗

$(1,-1)$ + ✓



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 $\theta_{t+1} \leftarrow \theta_t - x$

$\theta_1 = (0,0)$

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

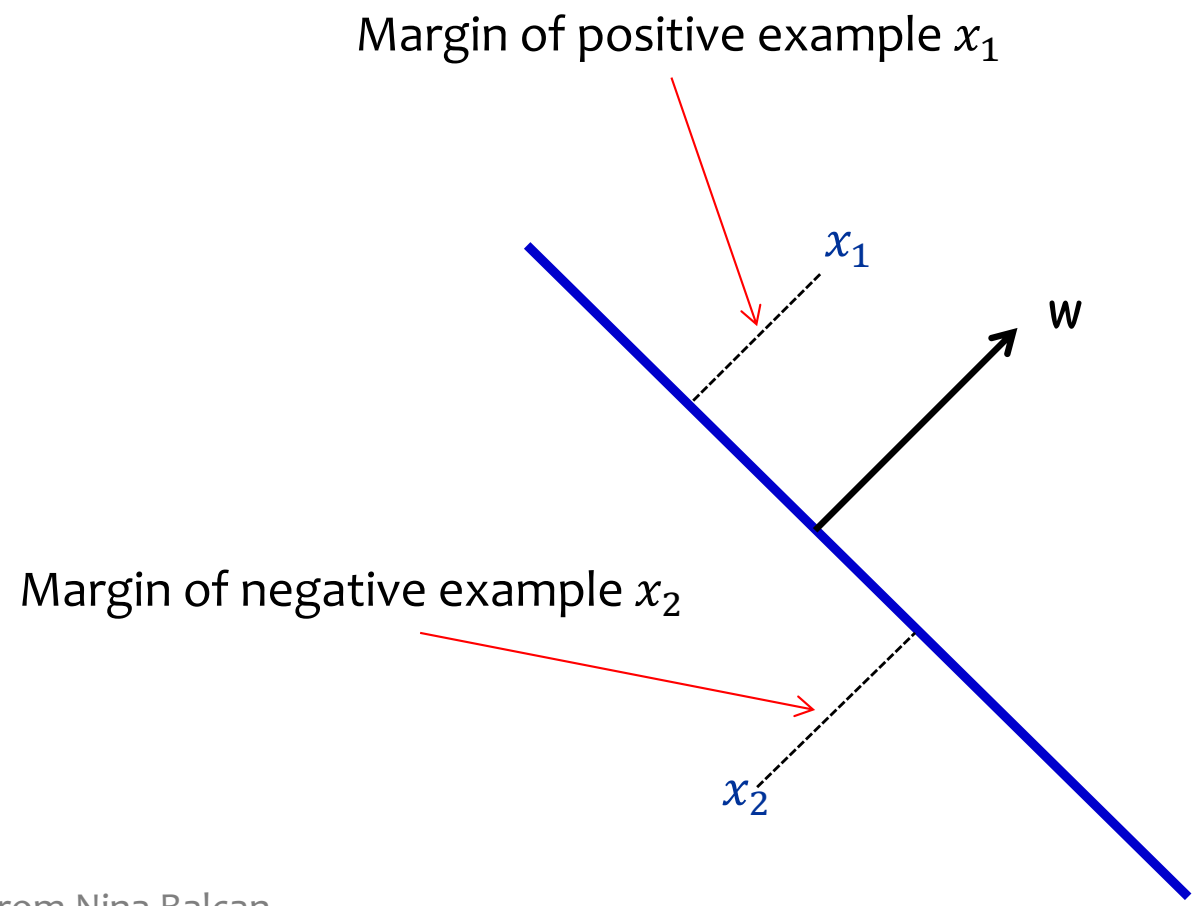
$\theta_3 = \theta_2 + (1,1) = (2,-1)$

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

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 $\theta \cdot x = 0$ (or the negative if on wrong side)

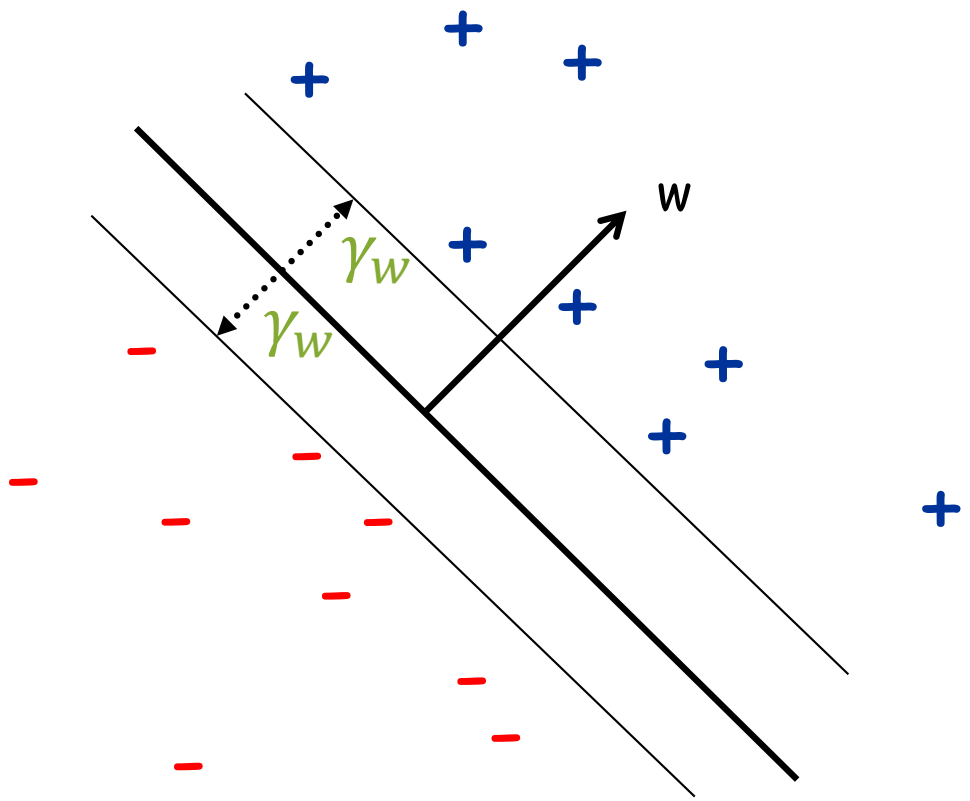


Slide from Nina Balcan

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 $x \in S$.



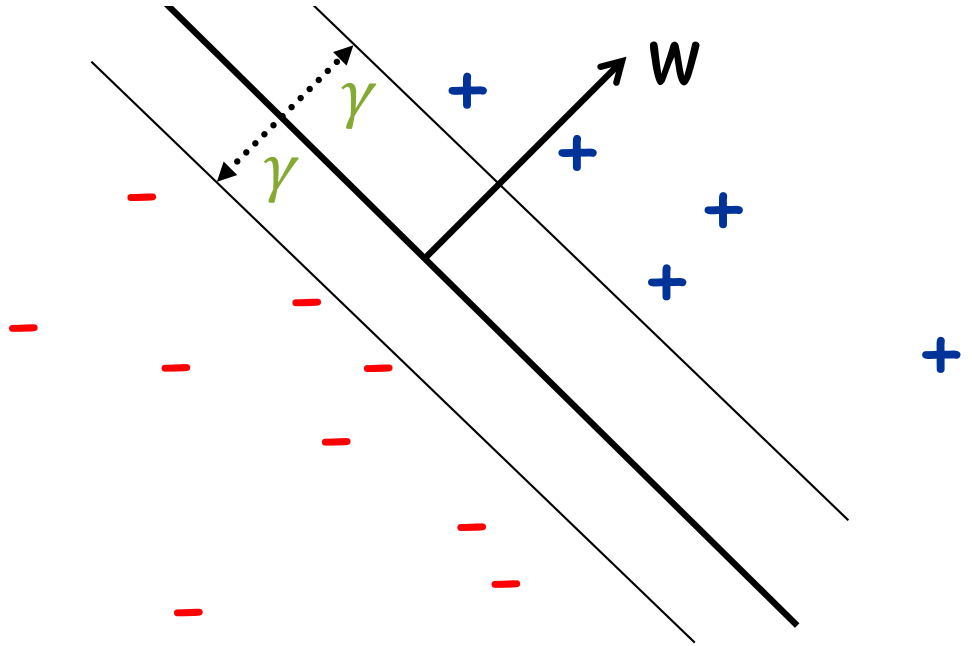
Slide from Nina Balcan

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.



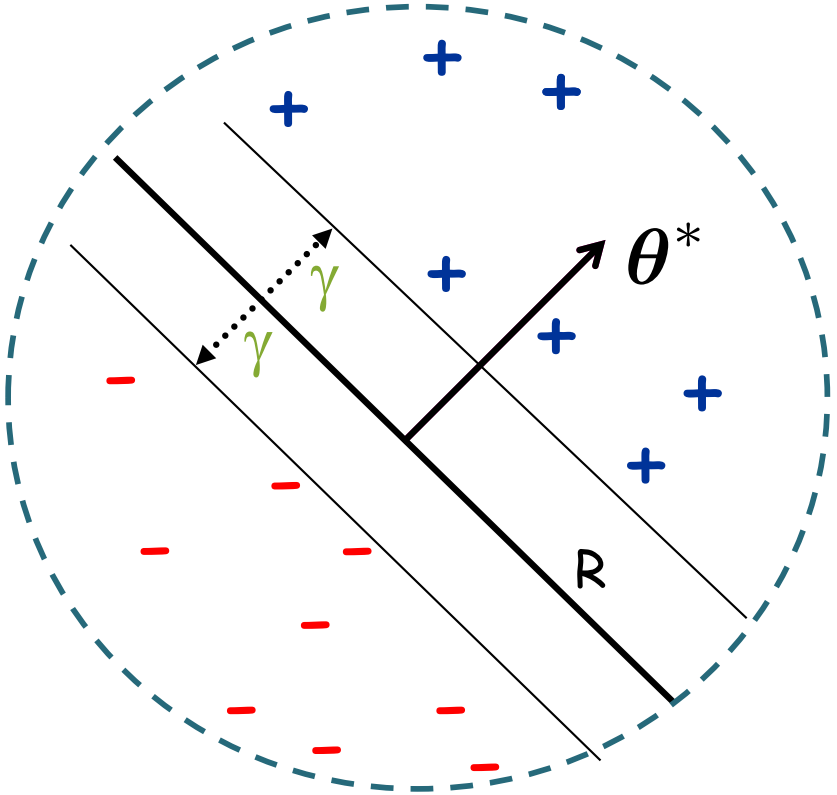
Slide from Nina Balcan

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

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

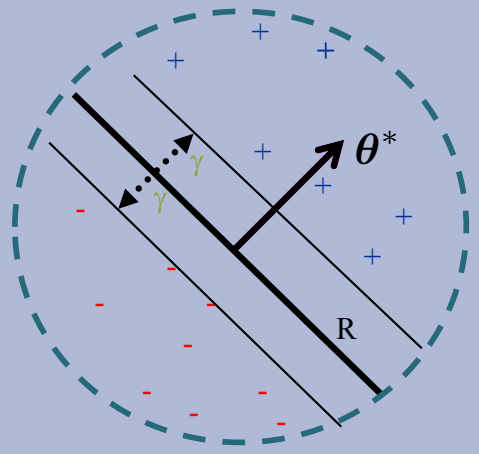


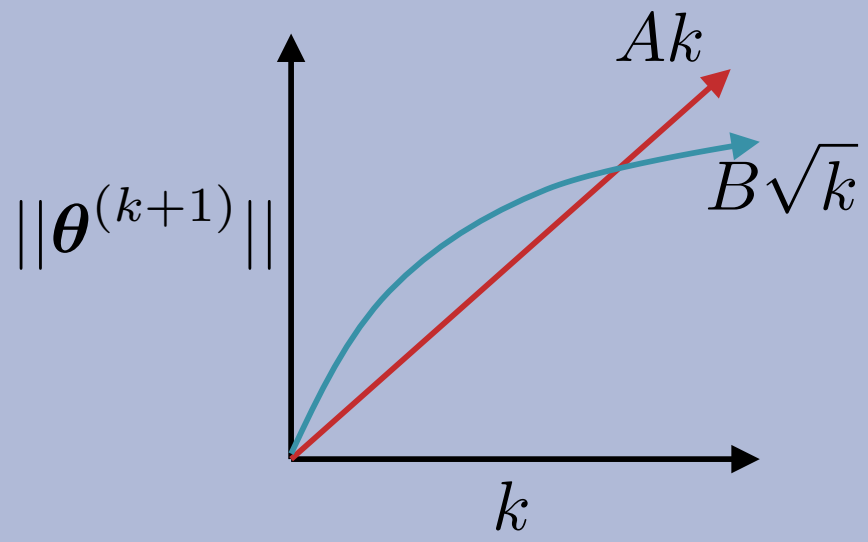
Figure from Nina Balcan

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

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

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



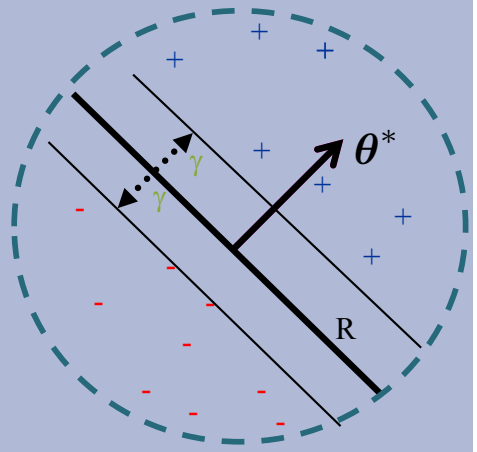
Analysis: Perceptron

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 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

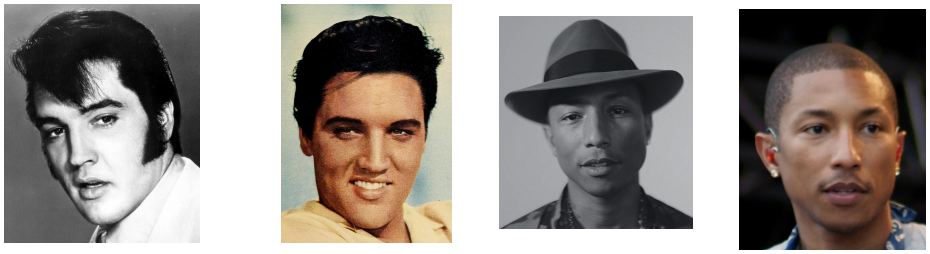
- Perceptron is a **linear classifier**
- **Simple learning algorithm**: when a mistake is made, add / subtract the features
- For linearly separable and inseparable data, we can **bound the number of mistakes** (geometric argument)
- Extensions support nonlinear separators and structured prediction

KERNELS

Kernels: Motivation

Most real-world problems exhibit data that is not linearly separable.

Example: pixel representation for Facial Recognition:



Q: When your data is **not linearly separable**, how can you still use a linear classifier?

A: Preprocess the data to produce **nonlinear features**

Kernels: Motivation

- Motivation #1: Inefficient Features
  - Non-linearly separable data requires **high dimensional** representation
  - Might be **prohibitively expensive** to compute or store
- Motivation #2: Memory-based Methods
  - k-Nearest Neighbors (KNN) for facial recognition allows a **distance metric** between images -- no need to worry about linearity restriction at all

Kernels

Whiteboard

- Kernel Perceptron
- Kernel as a dot product
- Gram matrix
- Examples: RBF kernel, string kernel

Φ

Kernel Methods

- **Key idea:**
  1. Rewrite the algorithm so that we only work with **dot products** $x^T z$ of feature vectors
  2. Replace the **dot products** $x^T z$ with a **kernel function** $k(x, z)$

- The kernel $k(x,z)$ can be **any** legal definition of a dot product:

$$k(x, z) = \varphi(x)^T \varphi(z) \text{ for any function } \varphi: \mathcal{X} \rightarrow \mathbf{R}^D$$

So we only compute the φ dot product **implicitly**

- This **"kernel trick"** can be applied to many algorithms:
  - classification: perceptron, SVM, …
  - regression: ridge regression, …
  - clustering: k-means, …

Kernel Methods

Q: These are just non-linear features, right?

A: Yes, but…

Q: Can't we just compute the feature transformation φ explicitly?

A: That depends…

Q: So, why all the hype about the kernel trick?

A: Because the **explicit features** might either be **prohibitively expensive** to compute or **infinite length** vectors

Example: Polynomial Kernel

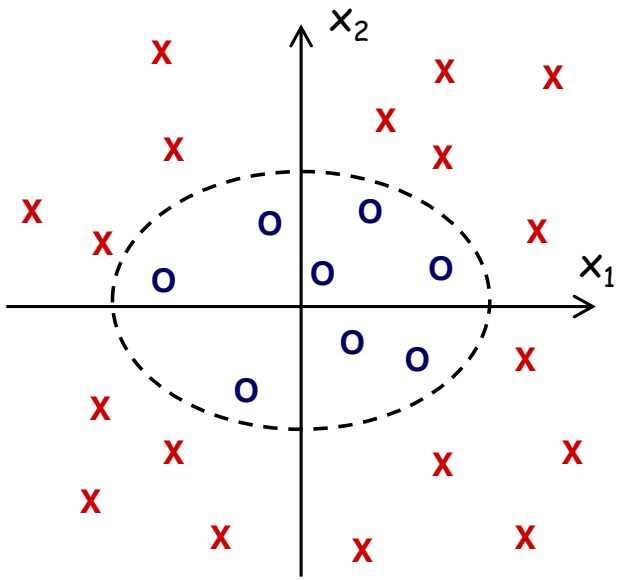
For n=2, d=2, the kernel $K(x, z) = (x \cdot z)^d$ corresponds to

$$\phi: R^2 \to R^3, (x_1, x_2) \to \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

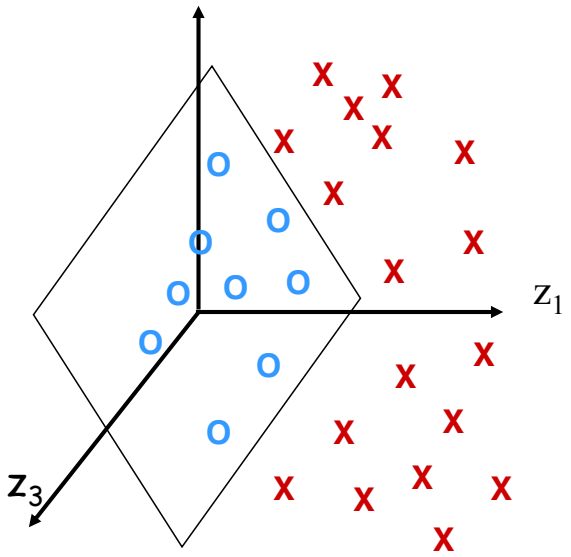
$$\phi(x) \cdot \phi(z) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2)$$

$$= (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)$$

Original space



Φ-space



Example: Polynomial Kernel

Feature space can grow really large and really quickly….

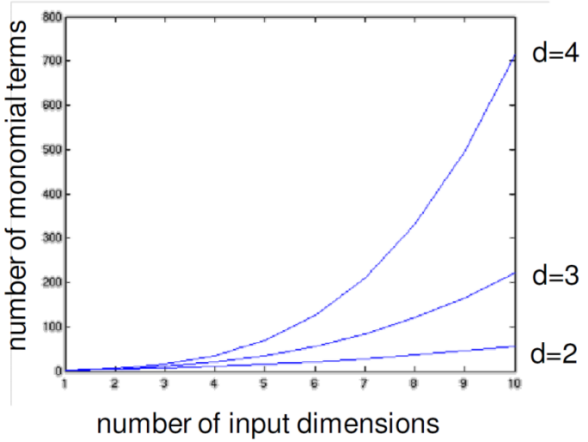
Crucial to think of ϕ as implicit, not explicit!!!!

Polynomial kernel degreee d, $k(x,z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$

- $x_1^d, x_1 x_2 \ldots x_d, x_1^2 x_2 \ldots x_{d-1}$
- Total number of such feature is

$$\binom{d+n-1}{d} = \frac{(d+n-1)!}{d!\,(n-1)!}$$

- $d = 6, n = 100$, there are 1.6 billion terms



$O(n)$ *computation*!

$$k(x,z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$$

Kernel Examples

Side Note: The feature space might not be unique!

Explicit representation #1:

$$\phi: \mathrm{R}^2 \to \mathrm{R}^3, (\mathrm{x}_1, \mathrm{x}_2) \to \Phi(\mathrm{x}) = (\mathrm{x}_1^2, \mathrm{x}_2^2, \sqrt{2}\mathrm{x}_1\mathrm{x}_2)$$

$$\phi(\mathrm{x}) \cdot \phi(z) = \left(\mathrm{x}_1^2, \mathrm{x}_2^2, \sqrt{2}\mathrm{x}_1\mathrm{x}_2\right) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2)$$

$$= (\mathrm{x}_1z_1 + \mathrm{x}_2z_2)^2 = (\mathrm{x} \cdot z)^2 = \mathrm{K}(\mathrm{x}, \mathrm{z})$$

Explicit representation #2:

$$\phi: \mathrm{R}^2 \to \mathrm{R}^4, (\mathrm{x}_1, \mathrm{x}_2) \to \Phi(\mathrm{x}) = (\mathrm{x}_1^2, \mathrm{x}_2^2, \mathrm{x}_1\mathrm{x}_2, \mathrm{x}_2\mathrm{x}_1)$$

$$\phi(\mathrm{x}) \cdot \phi(z) = (\mathrm{x}_1^2, \mathrm{x}_2^2, \mathrm{x}_1\mathrm{x}_2, \mathrm{x}_2\mathrm{x}_1) \cdot (\mathrm{z}_1^2, \mathrm{z}_2^2, \mathrm{z}_1\mathrm{z}_2, \mathrm{z}_2\mathrm{z}_1)$$

$$= (\mathrm{x} \cdot z)^2 = \mathrm{K}(\mathrm{x}, \mathrm{z})$$

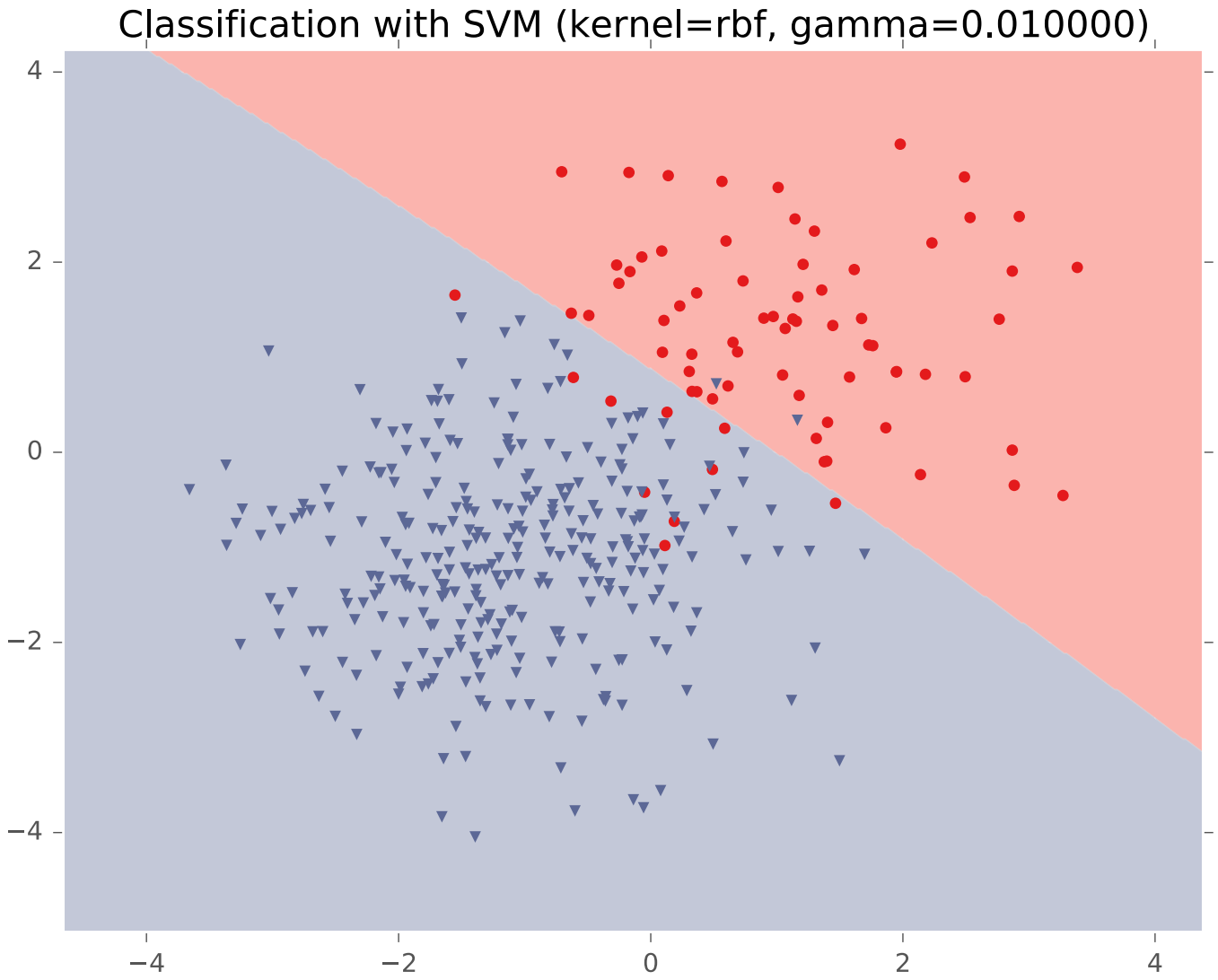
These two different feature representations correspond to the same kernel function!

Slide from Nina Balcan

Kernel Examples

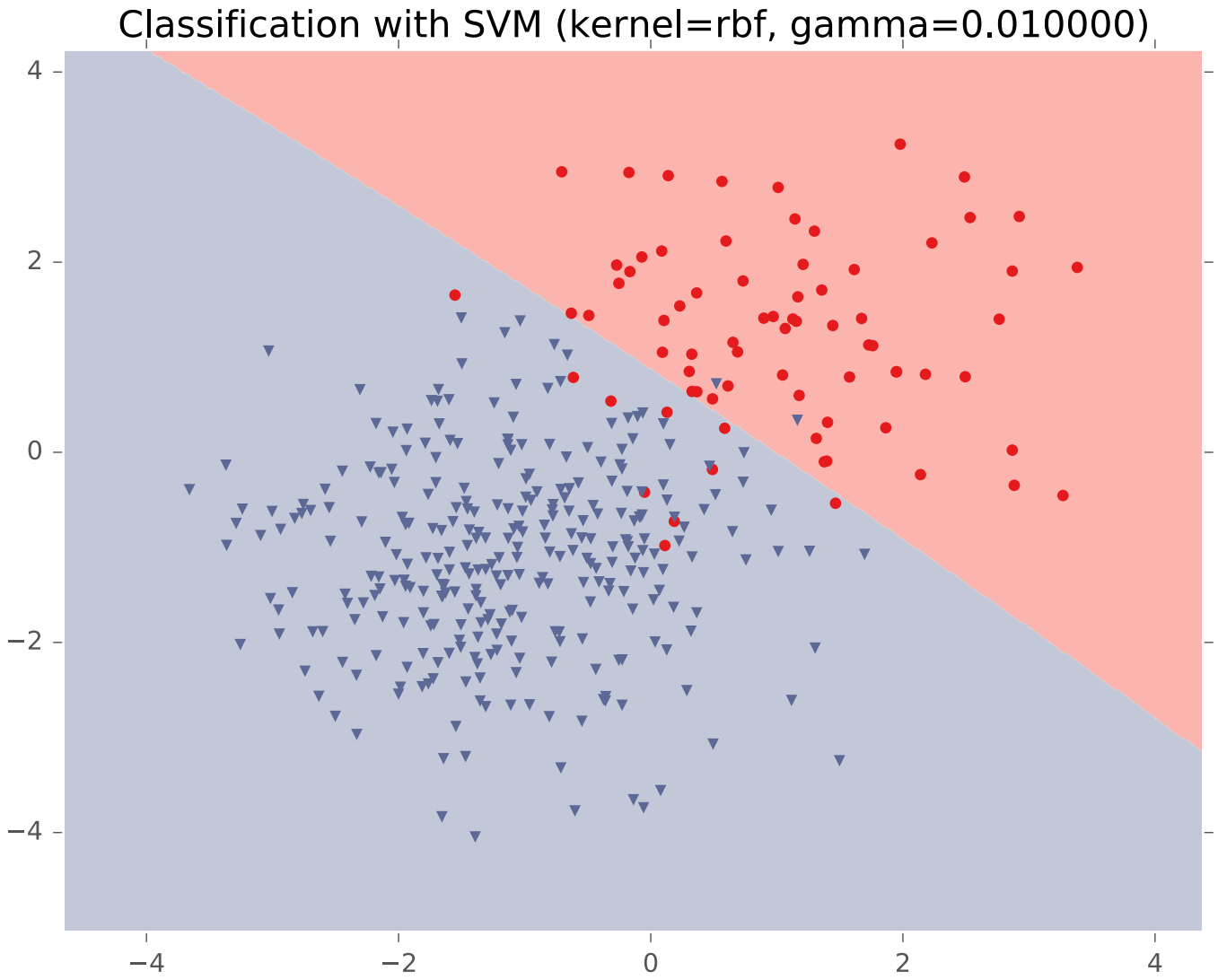
Name	Kernel Function (implicit dot product)	Feature Space (explicit dot product)
Linear	$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + 1)^d$	All polynomials **up to** degree d
Gaussian	$K(\mathbf{x}, \mathbf{z}) = \exp(-\fr\|\|\mathbf{x} - \mathbf{z}\|\|_2^2}{2\sigma^2})$	Infinite dimensional space
Hyperbolic Tangent (Sigmoid) Kernel	$K(\mathbf{x}, \mathbf{z}) = \tanh(\alpha\mathbf{x}^T\mathbf{z} + c)$	(With SVM, this is equivalent to a 2-layer neural network)

RBF Kernel Example



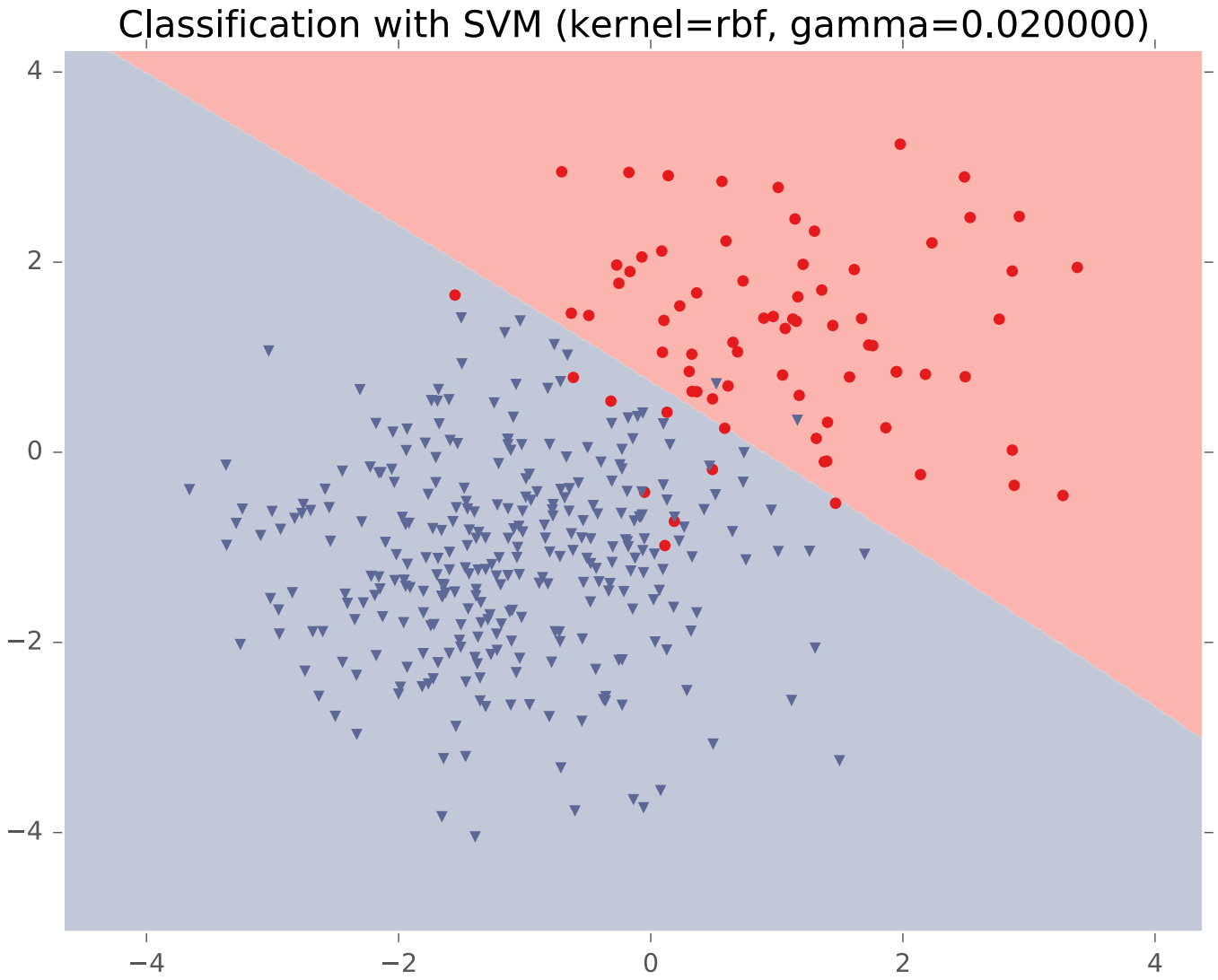
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



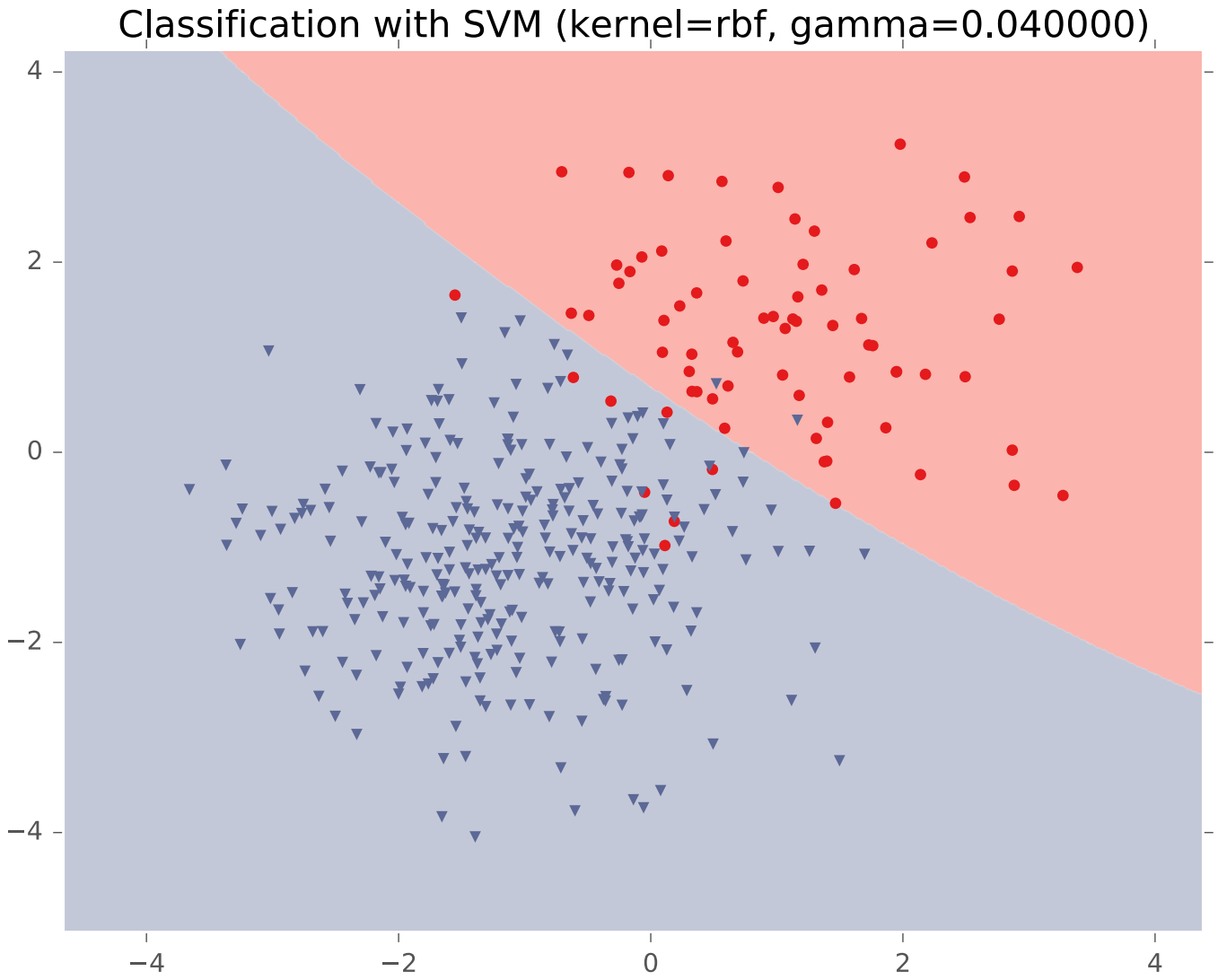
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



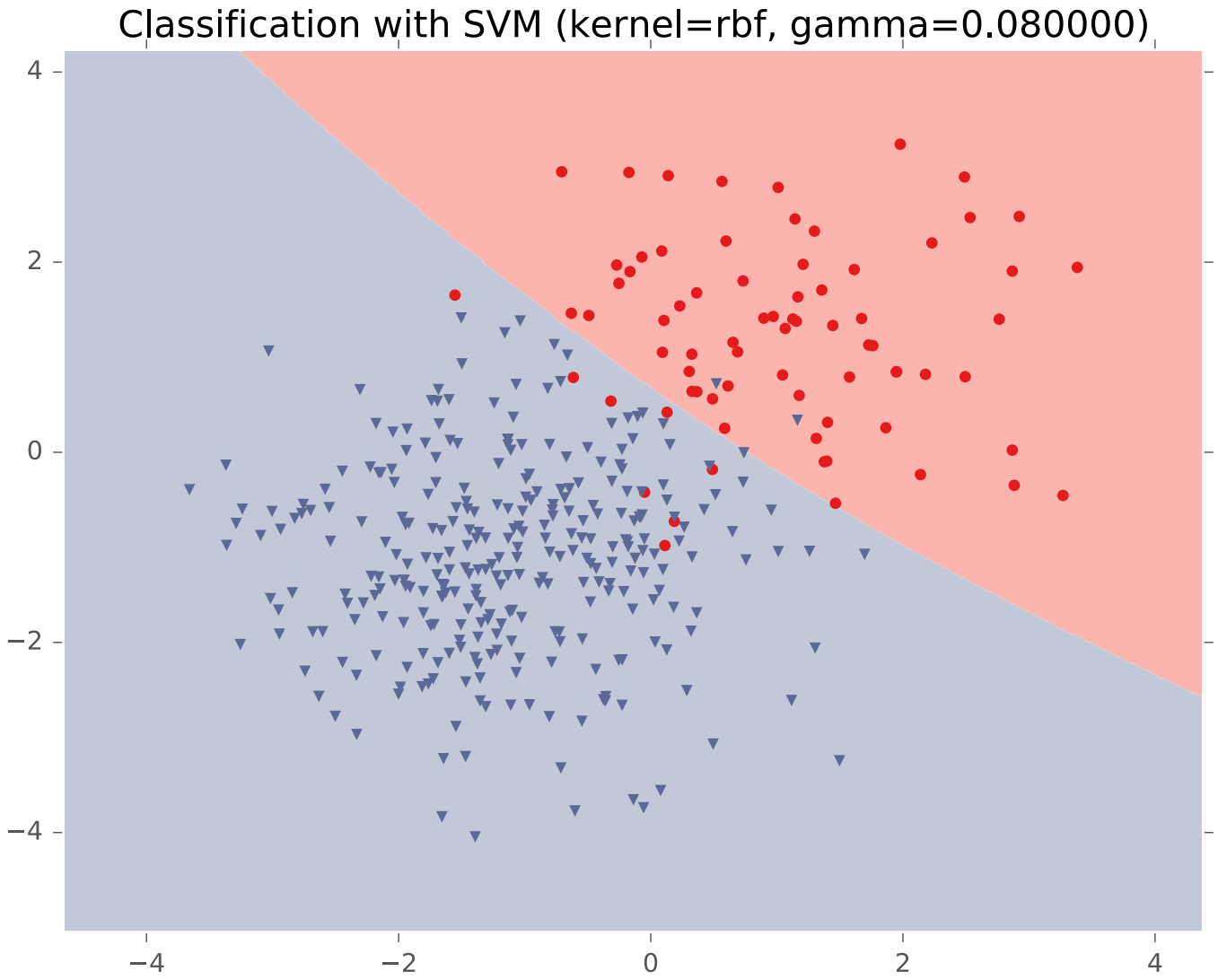
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



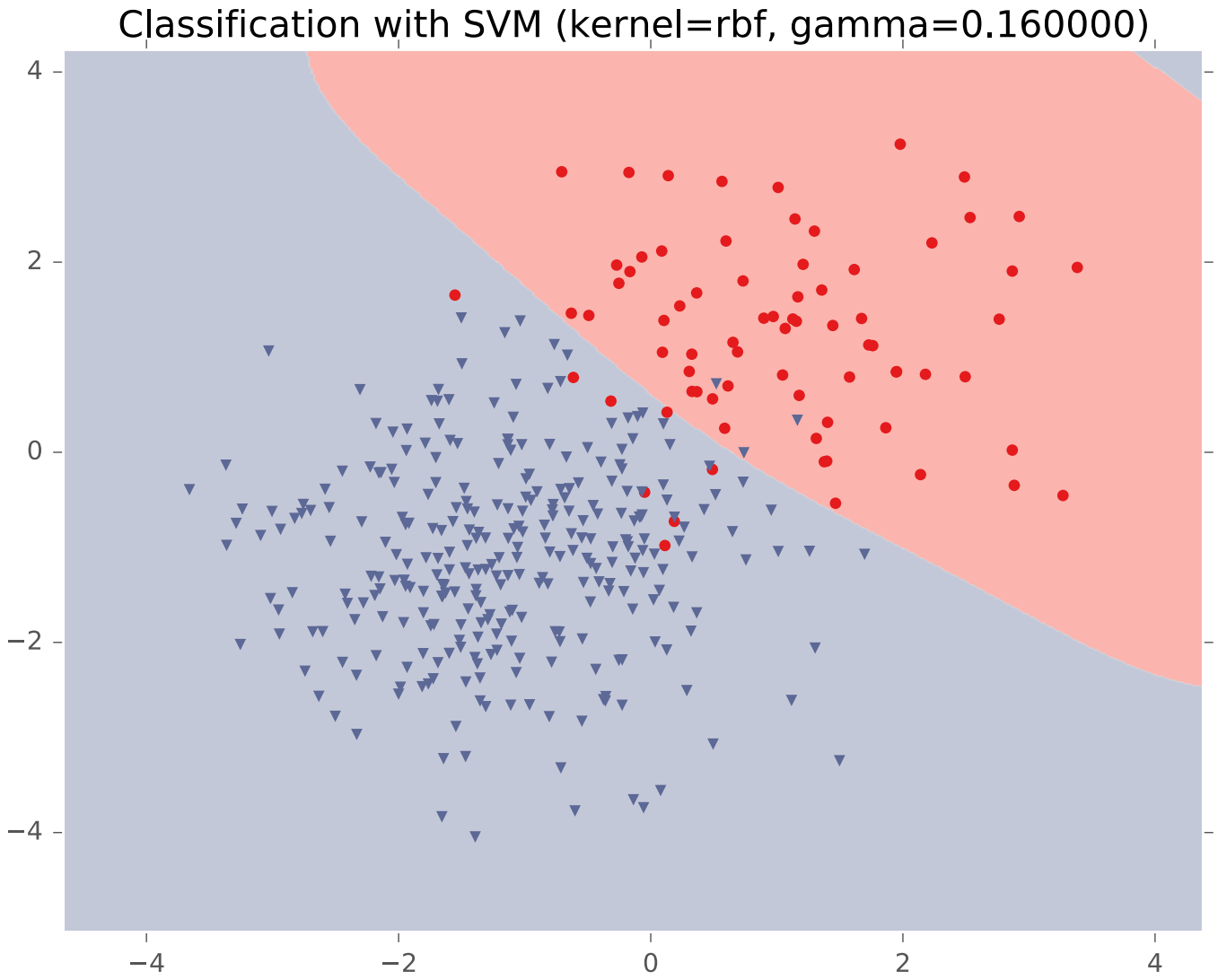
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



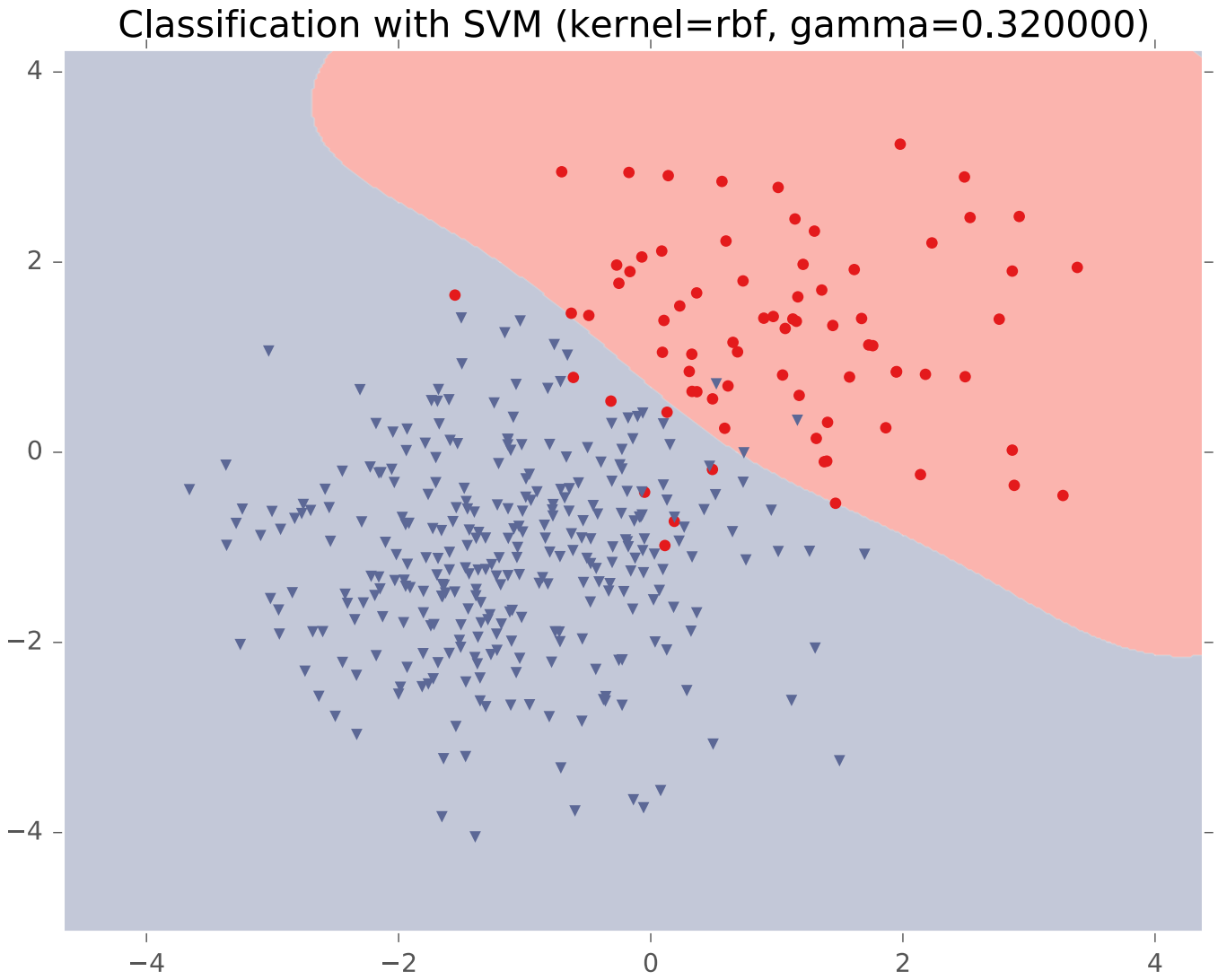
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



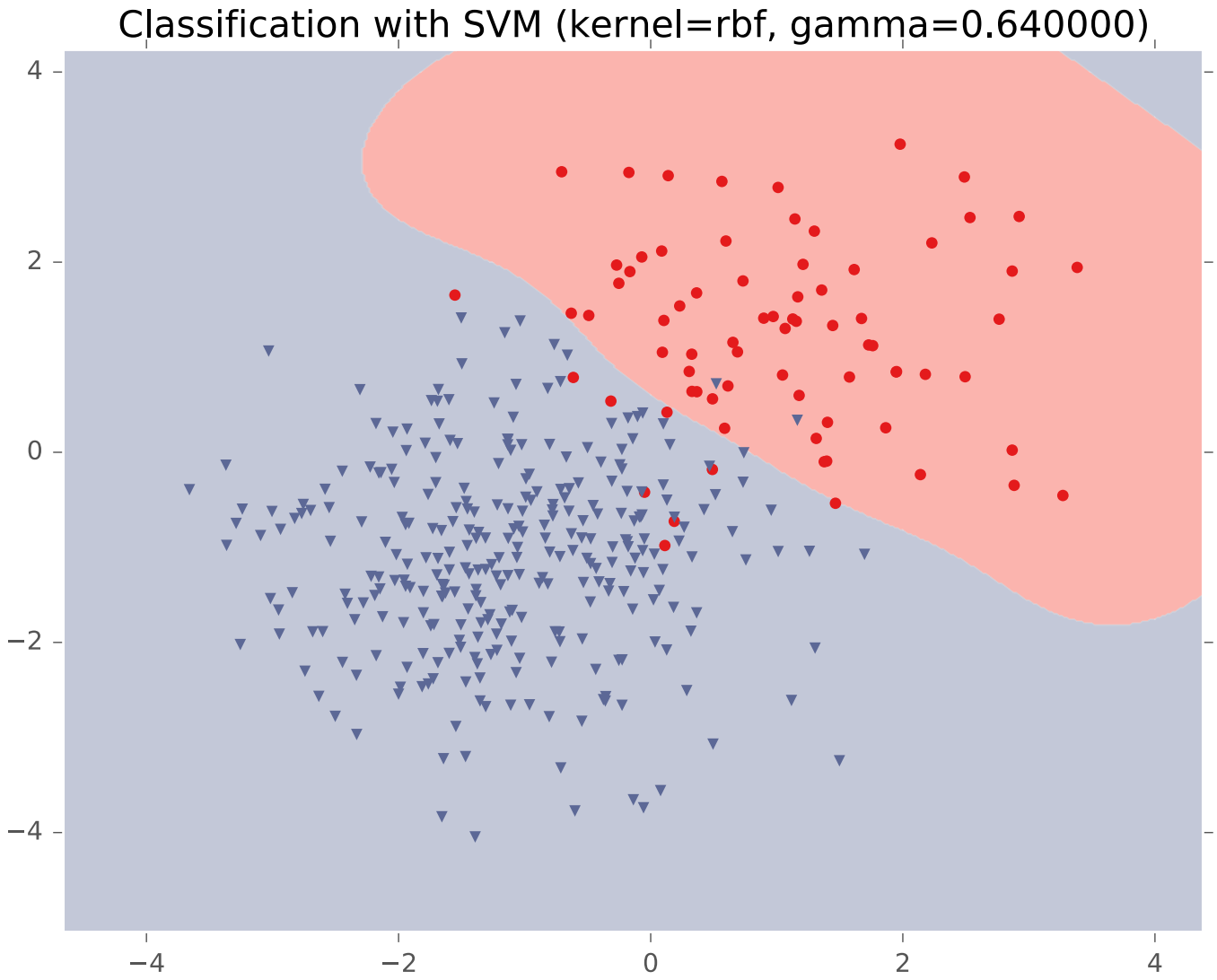
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



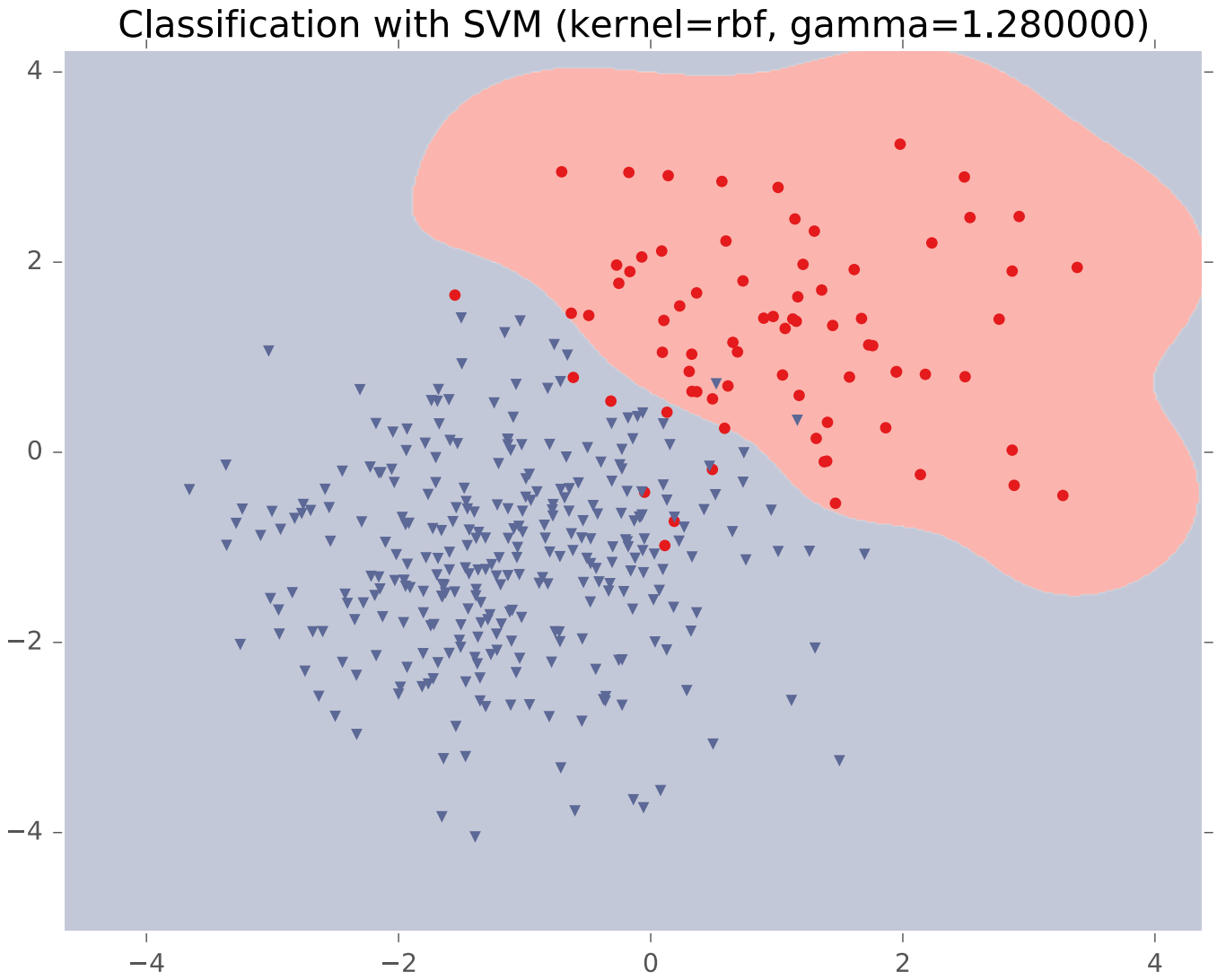
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



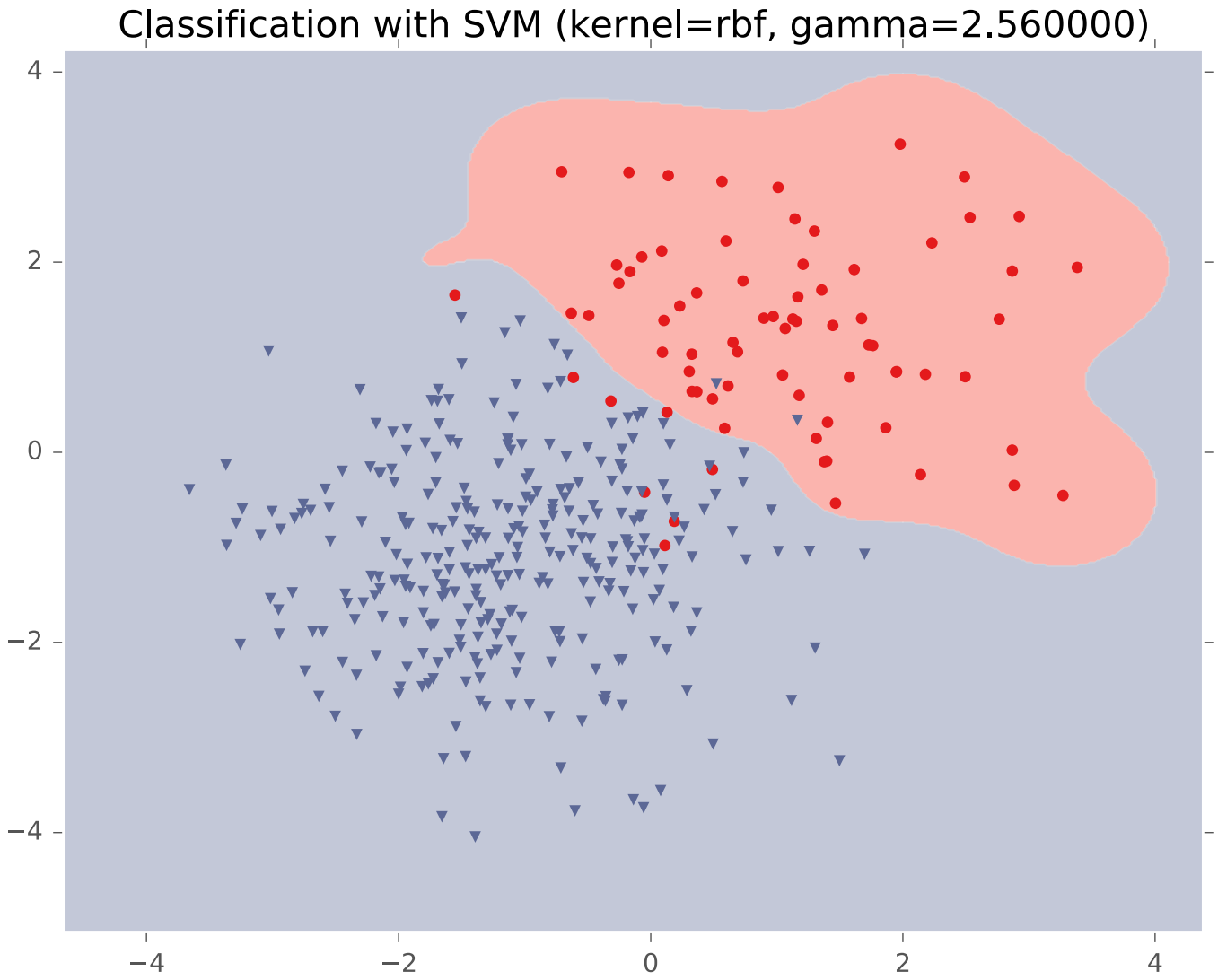
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



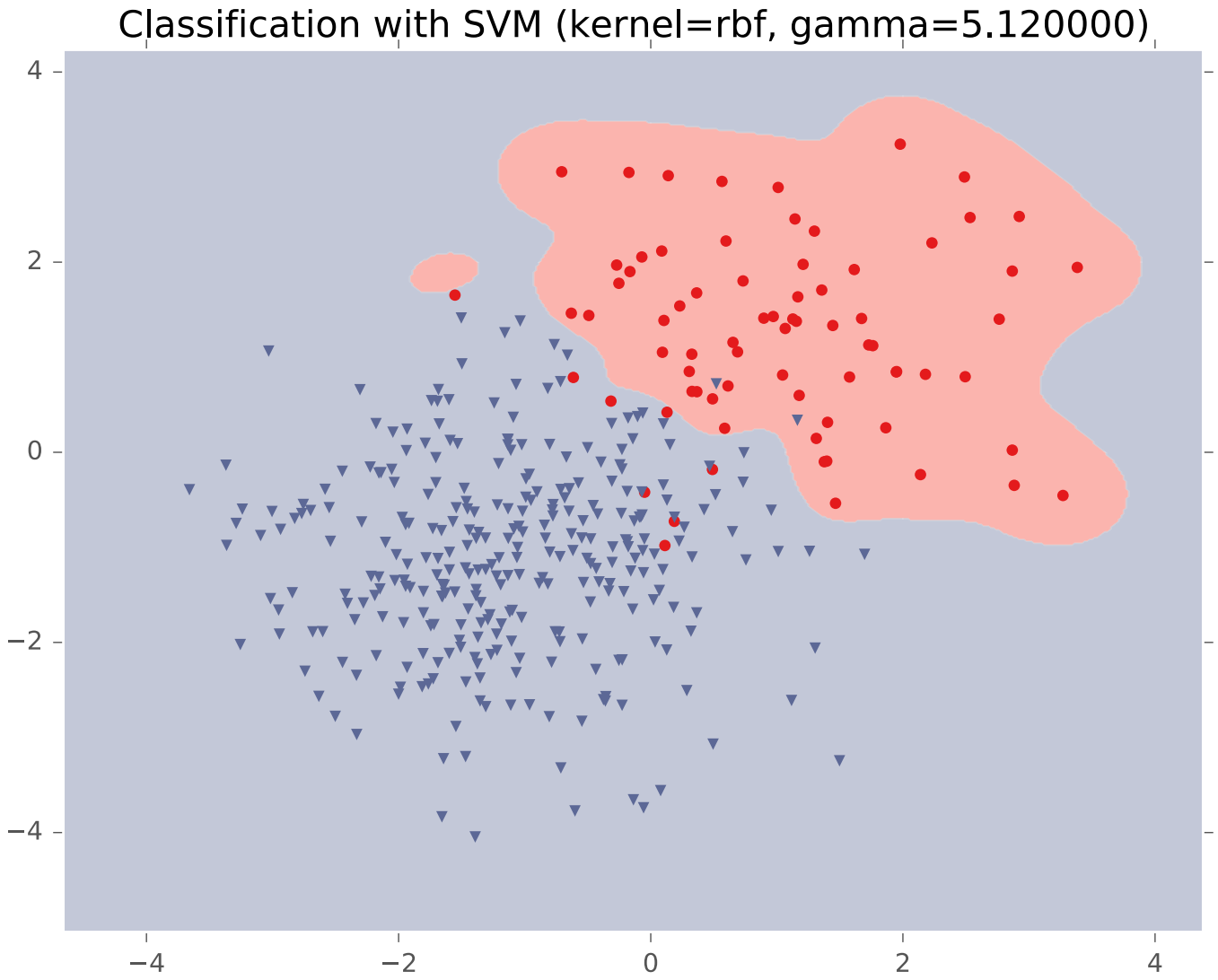
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



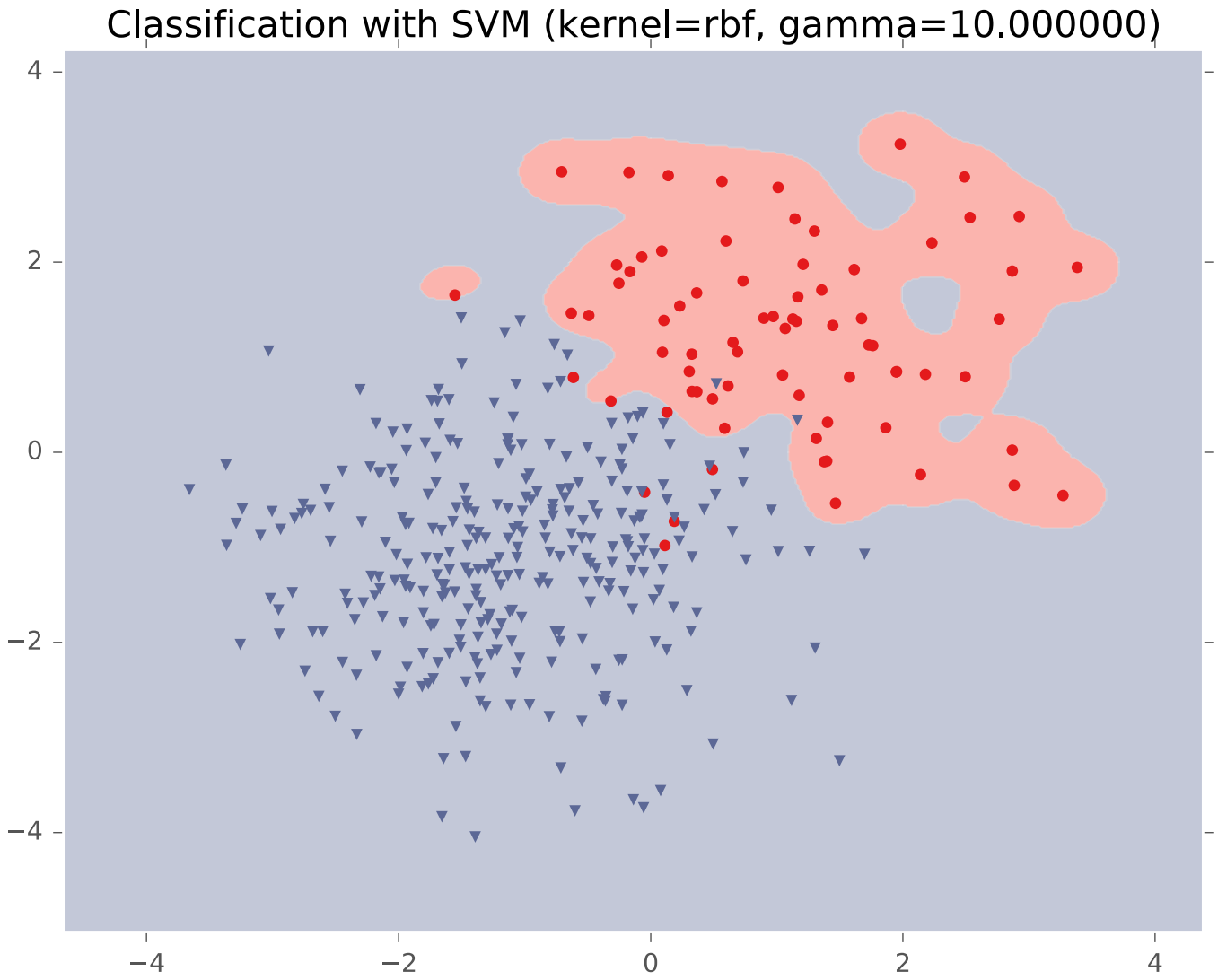
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

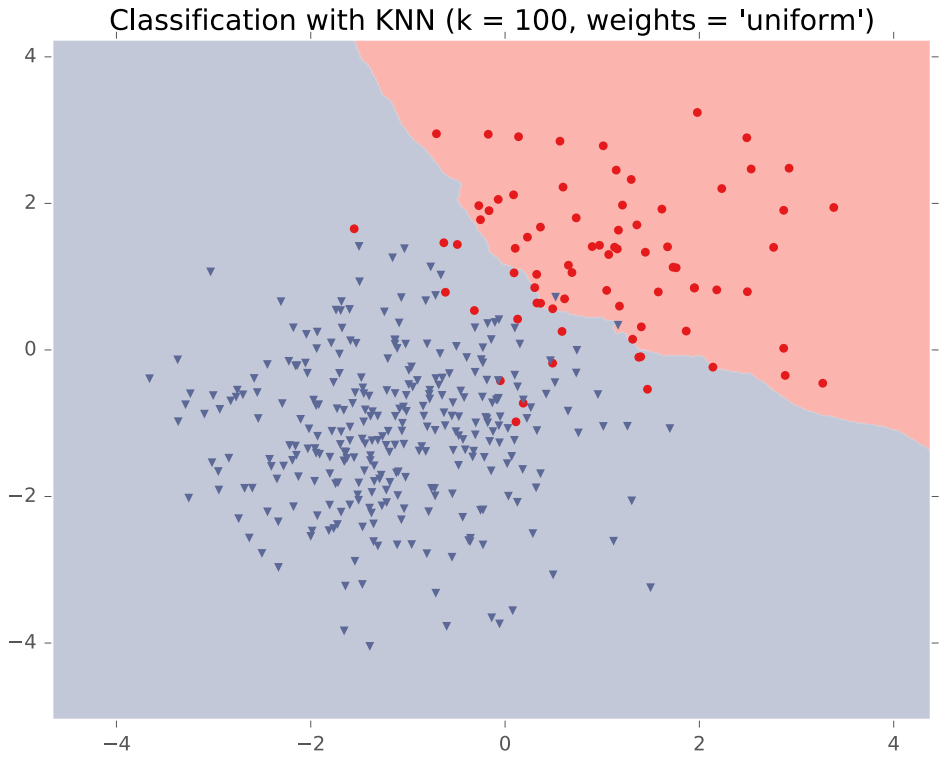
RBF Kernel Example

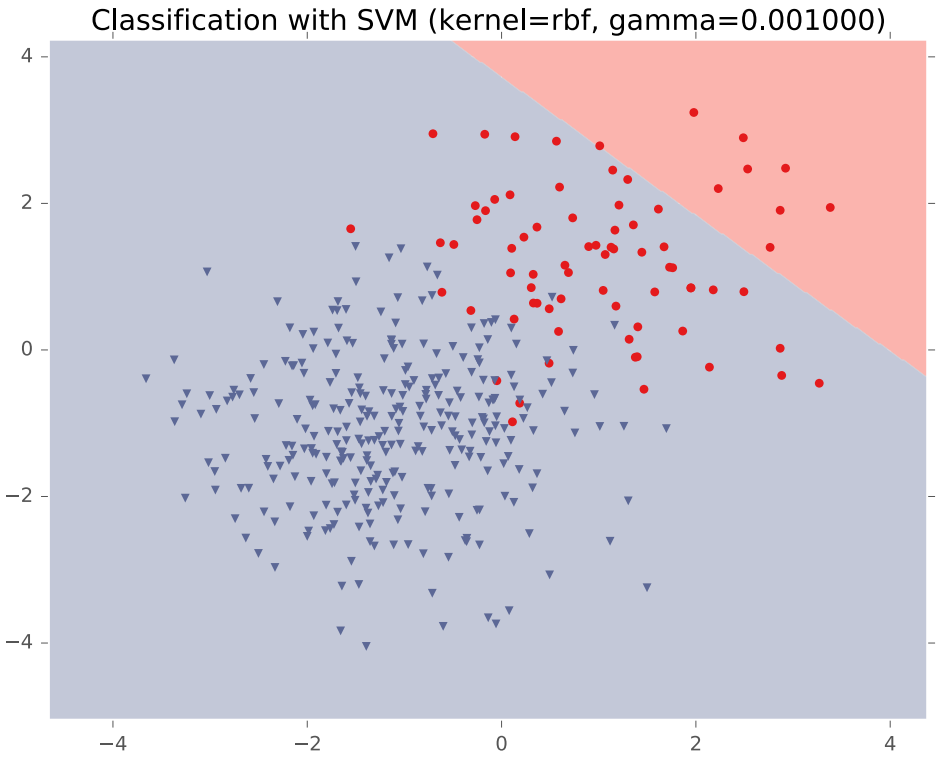


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM

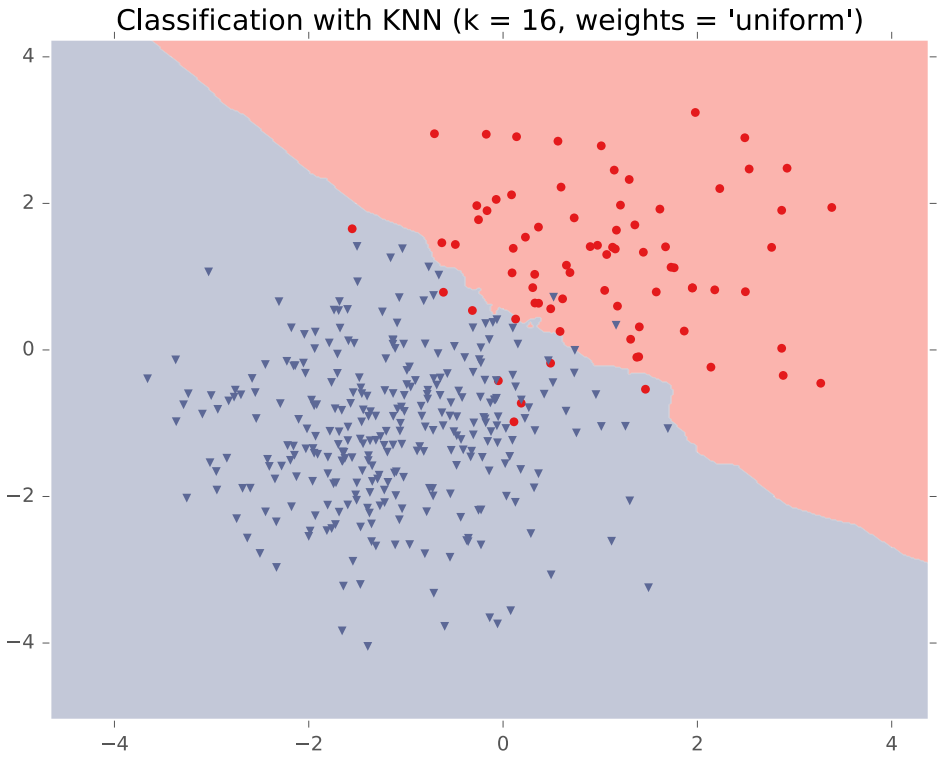


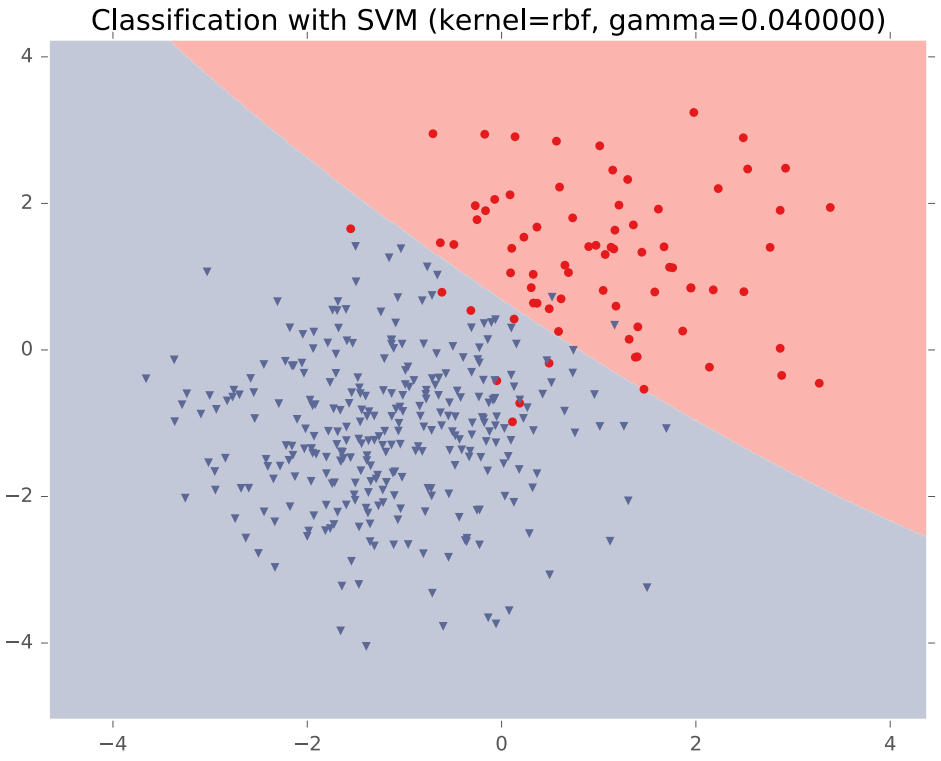


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM

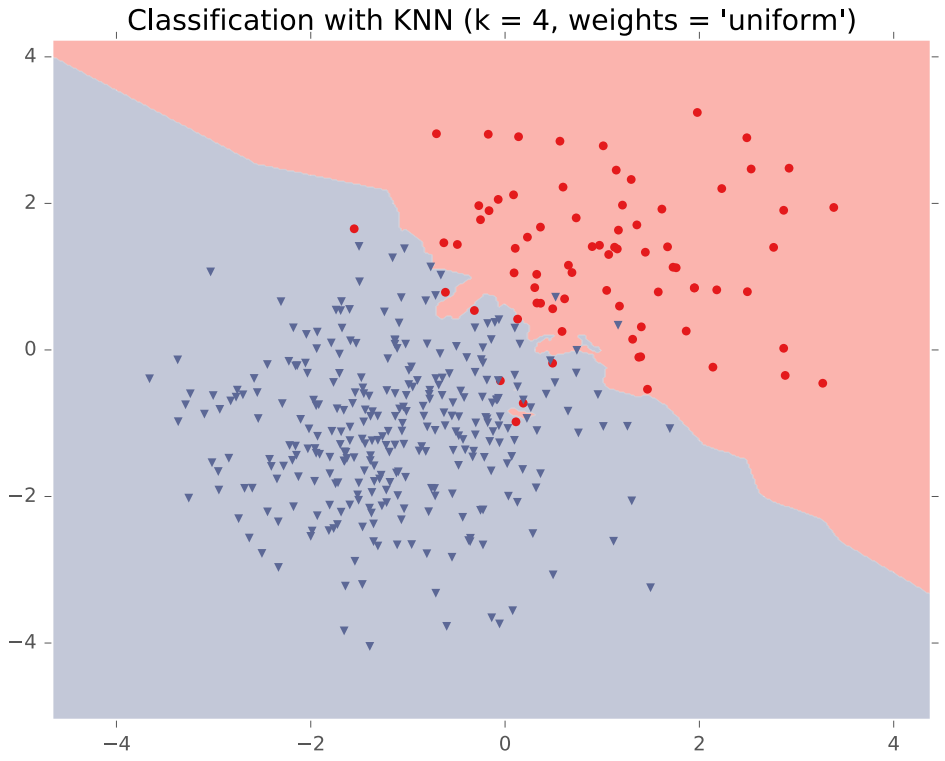


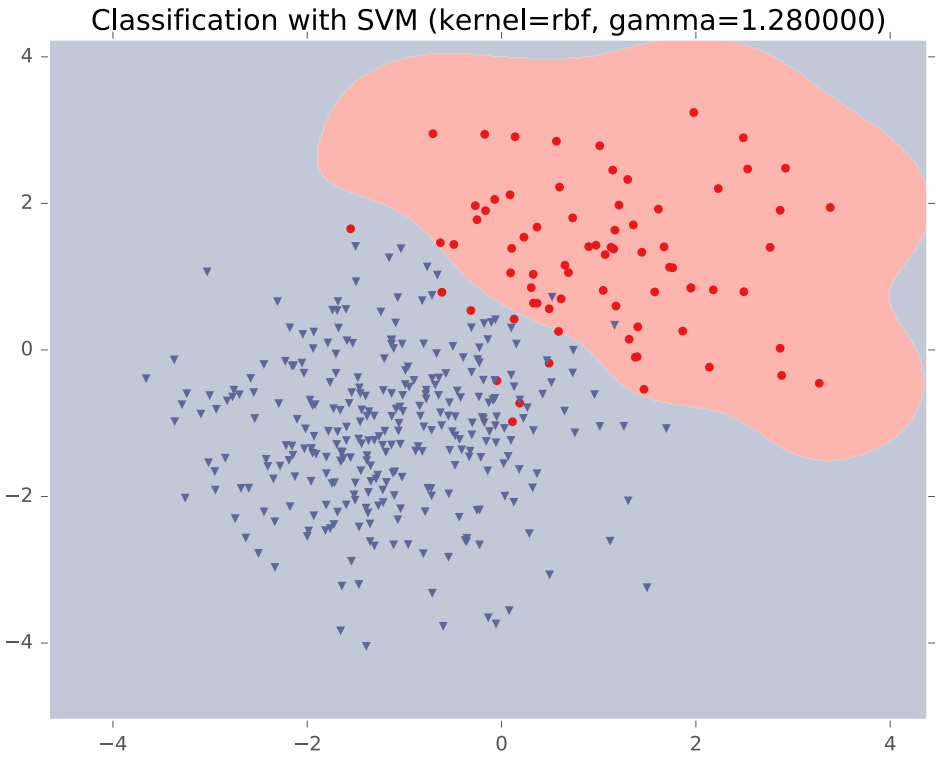


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM

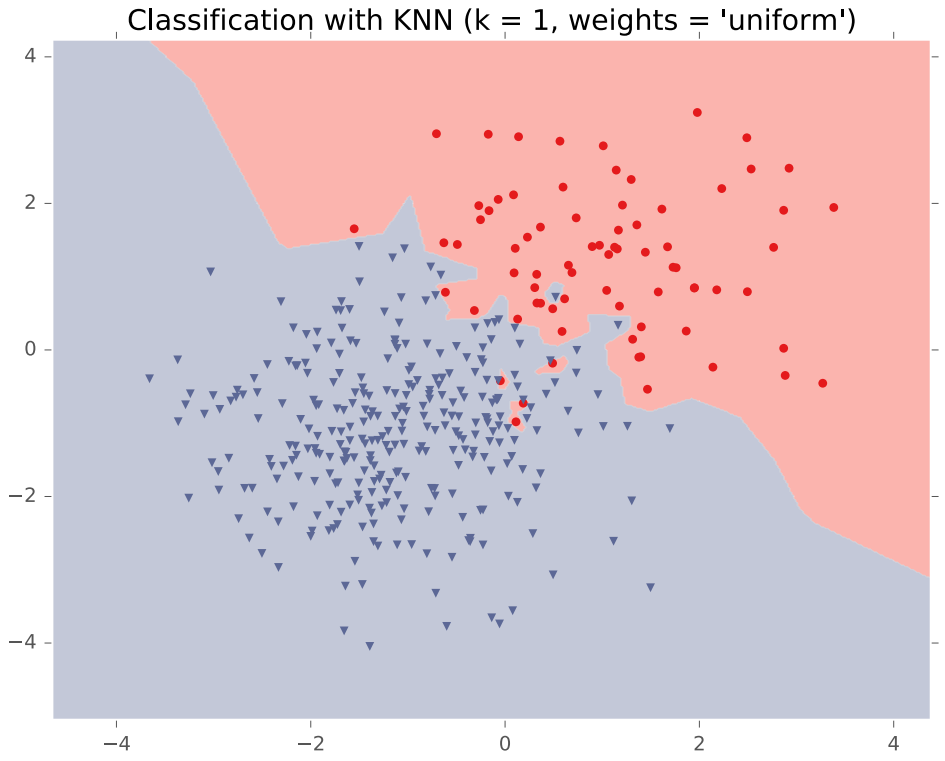


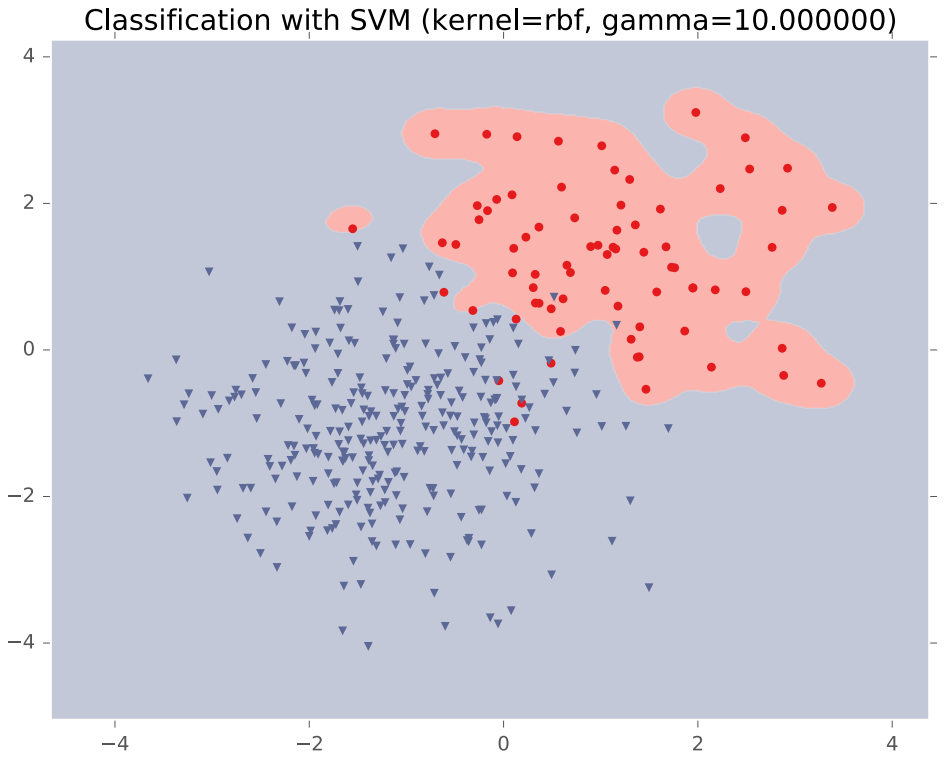


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM





RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

Kernels: Discussion

- If all computations involving instances are in terms of inner products then:
  - Conceptually, work in a very high diml space and the alg's performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each $x \cdot z$ with a $K(x, z)$.

How to choose a kernel:

- Kernels often encode domain knowledge (e.g., string kernels)
- Use Cross-Validation to choose the parameters, e.g., σ for Gaussian Kernel $K(x, z) = \exp\left[-\frac{||x-z||^2}{2\sigma^2}\right]$
- Learn a good kernel; e.g., [Lanckriet-Cristianini-Bartlett-El Ghaoui-Jordan'04]

Slide from Nina Balcan