10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Support Vector Machines

+

Kernels

Matt Gormley
Lecture 27
Nov. 22, 2019

Reminders

- **Homework 7: HMMs**
  - Out: Fri, Nov. 8
  - Due: Mon, Nov. 25 at 11:59pm
- **Homework 8: Learning Paradigms**
  - Out: Mon, Nov. 25
  - Due: Wed, Dec. 4 at 11:59pm
  - Can only be submitted up to 3 days late, so we can return grades before final exam

- **Today's In-Class Poll**
  - http://p27.mlcourse.org

CONSTRAINED OPTIMIZATION

Constrained Optimization

Unconstrained

$$\min_{\vec{\theta}} J(\vec{\theta})$$

Constrained

$$\min_{\vec{\theta}} J(\vec{\theta})$$

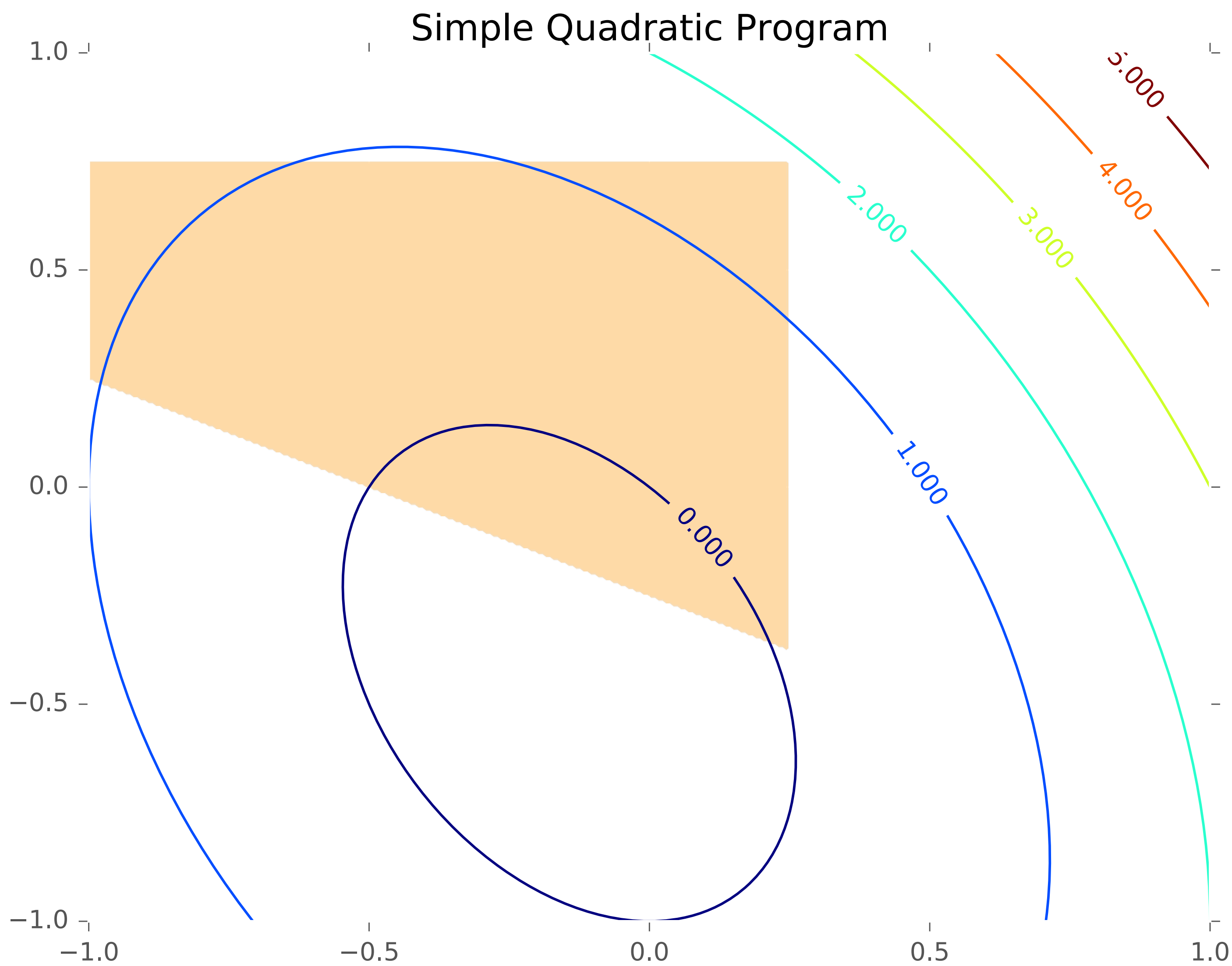
$$\text{s.t. } g(\vec{\theta}) \leq \vec{b}$$

SVM: Optimization Background

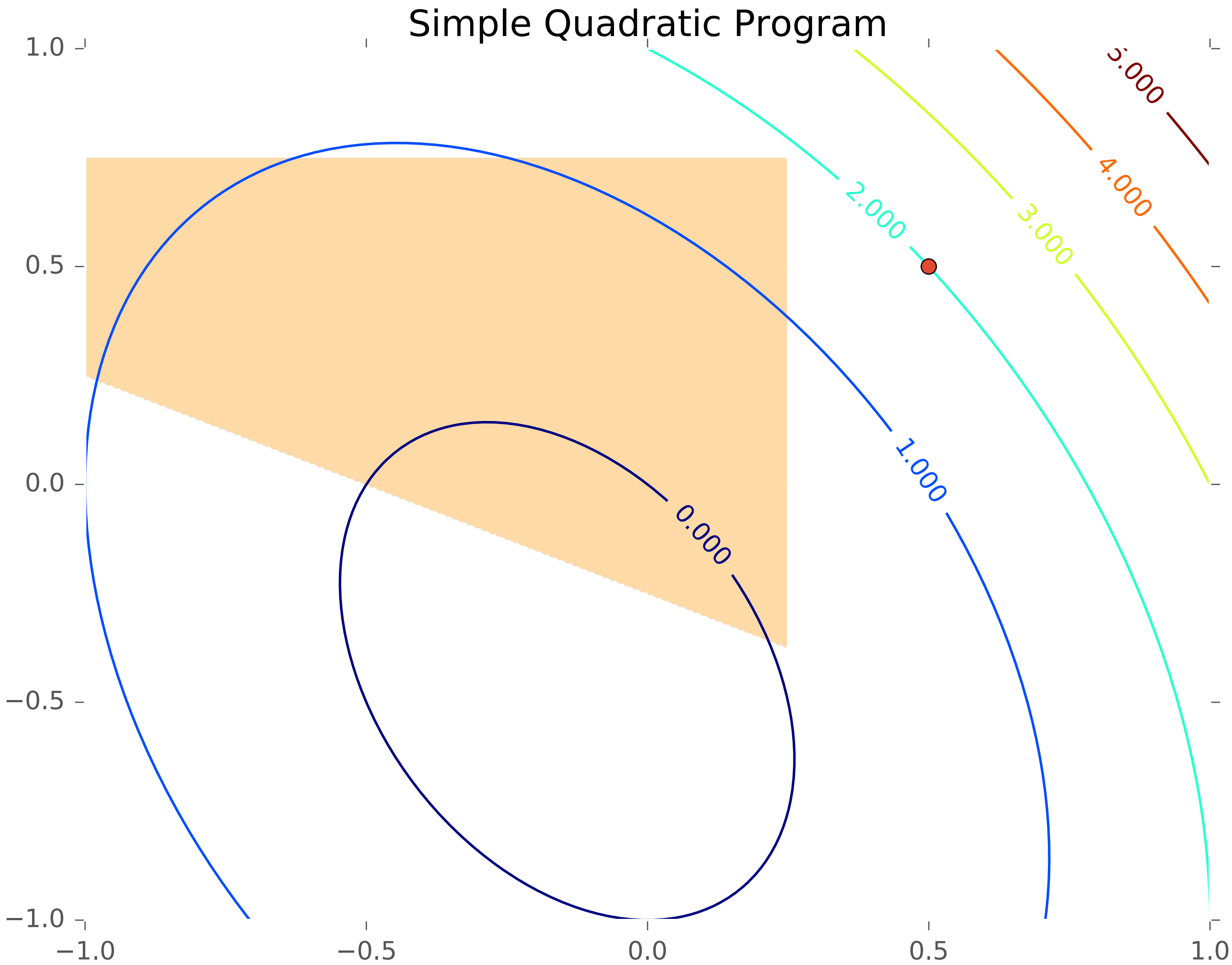
Whiteboard

- Constrained Optimization
- Linear programming
- Quadratic programming
- Example: 2D quadratic function with linear constraints

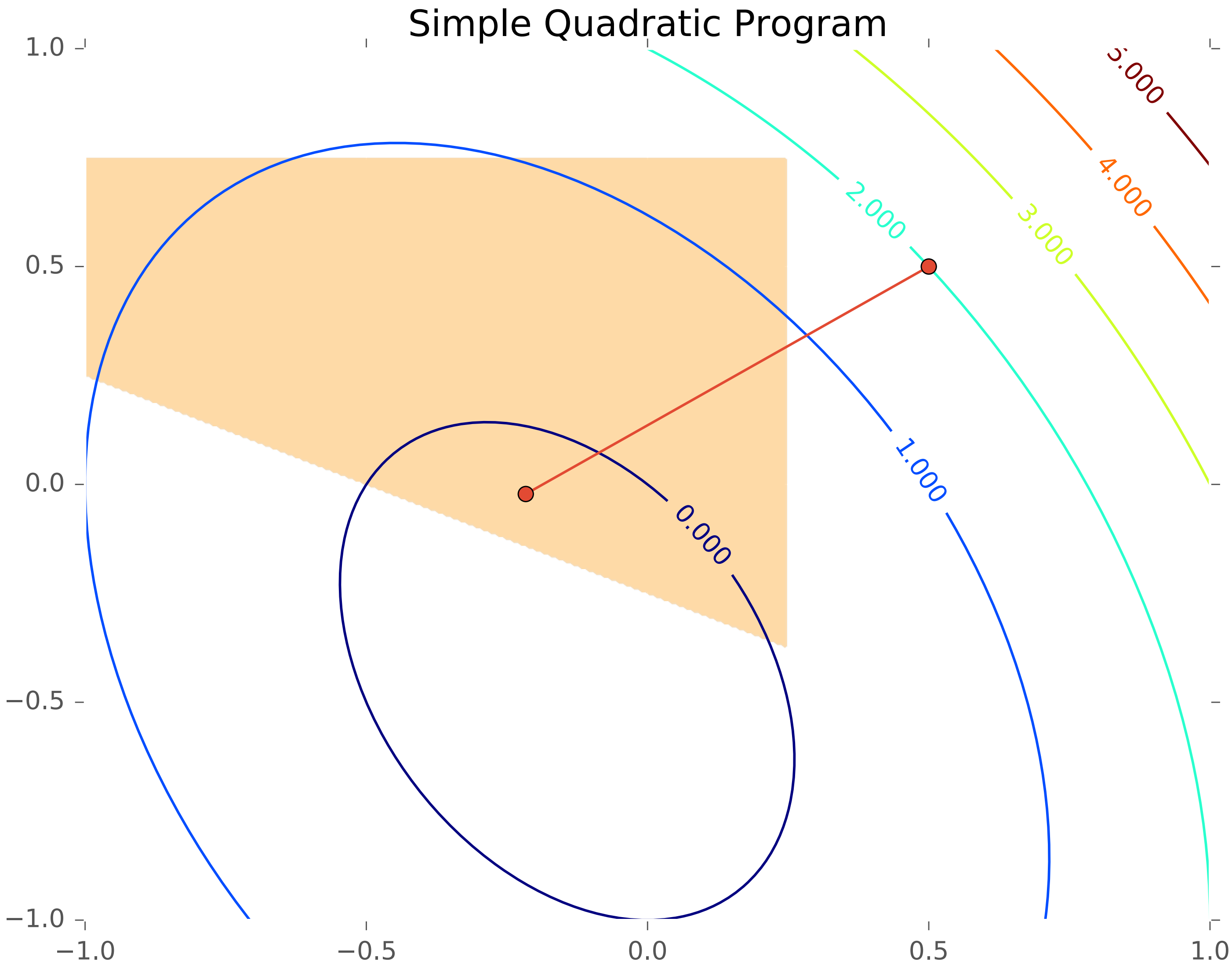
Quadratic Program



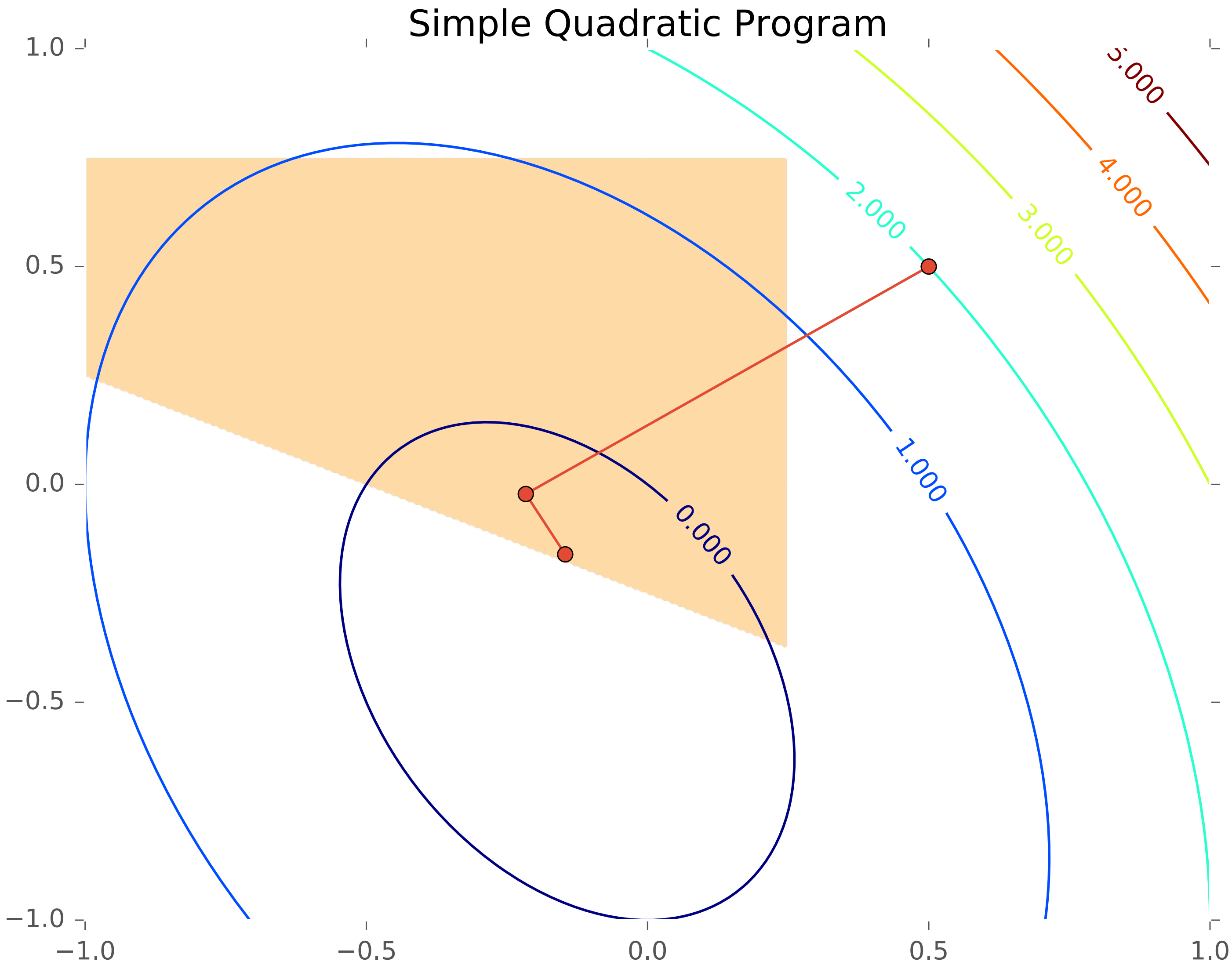
Quadratic Program



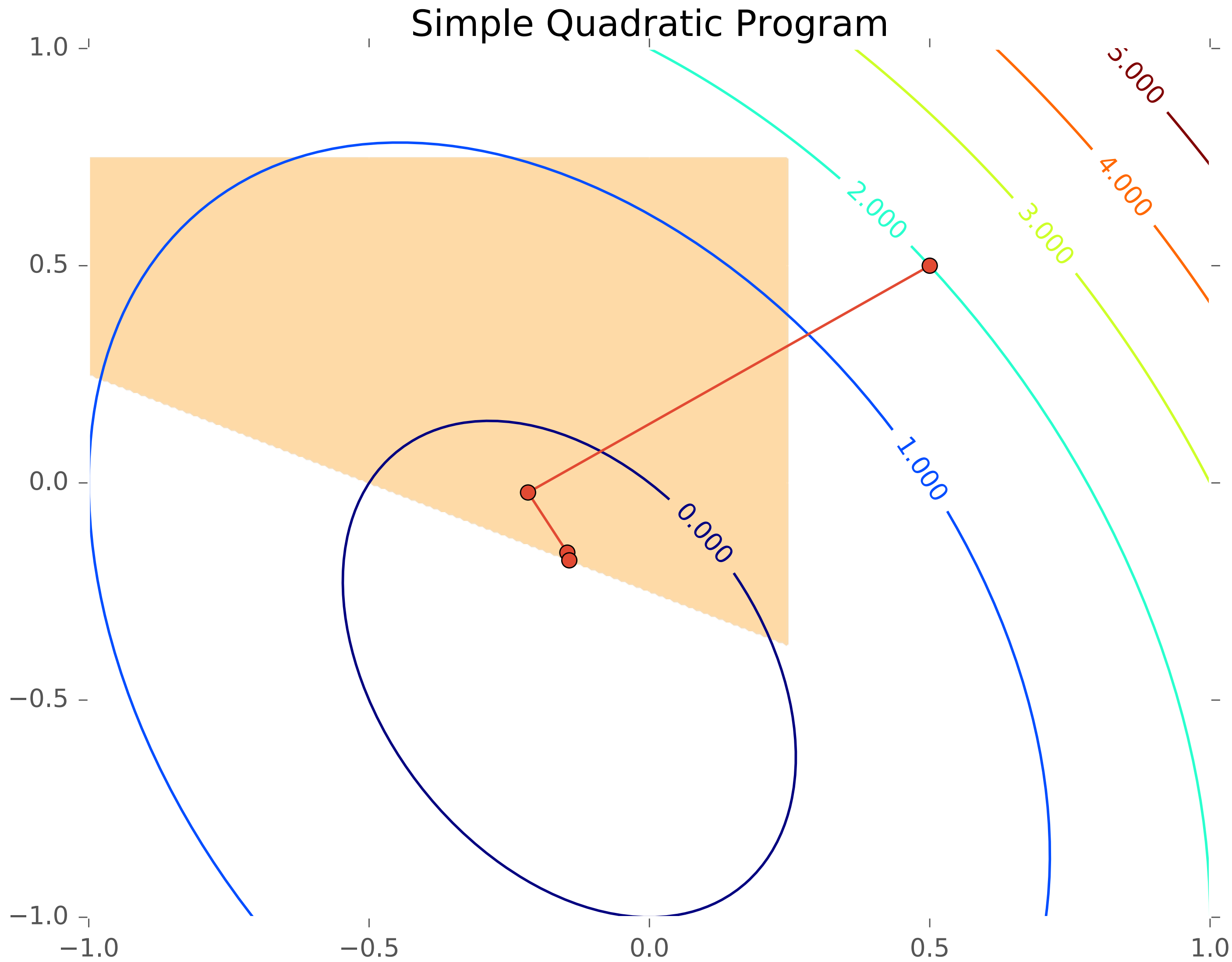
Quadratic Program



Quadratic Program



Quadratic Program



SUPPORT VECTOR MACHINE (SVM)

Example: Building a Canal

https://www.flickr.com/photos/hereistom/10438848375

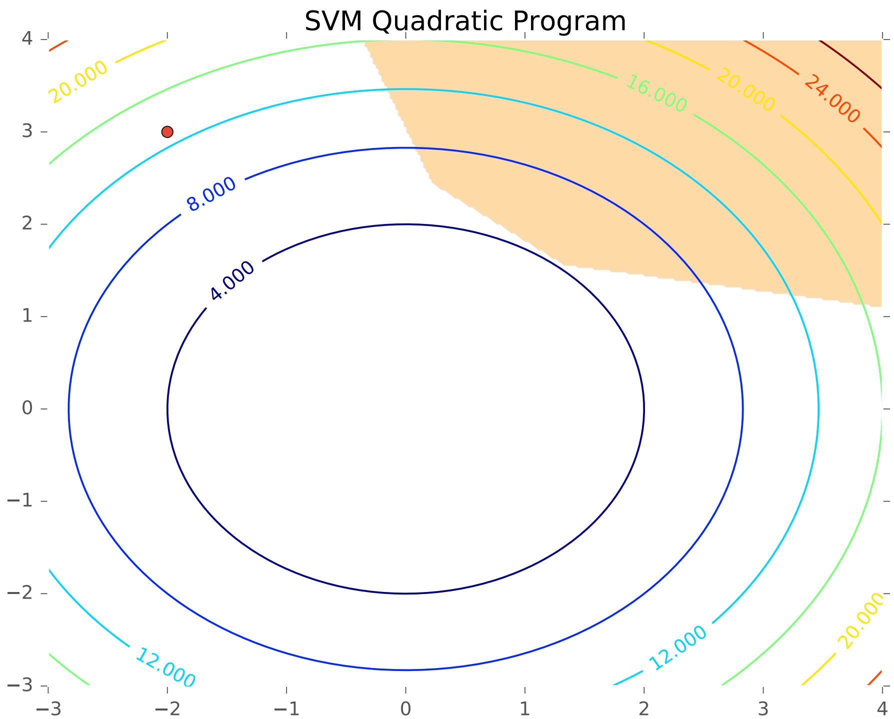
SVM

Whiteboard

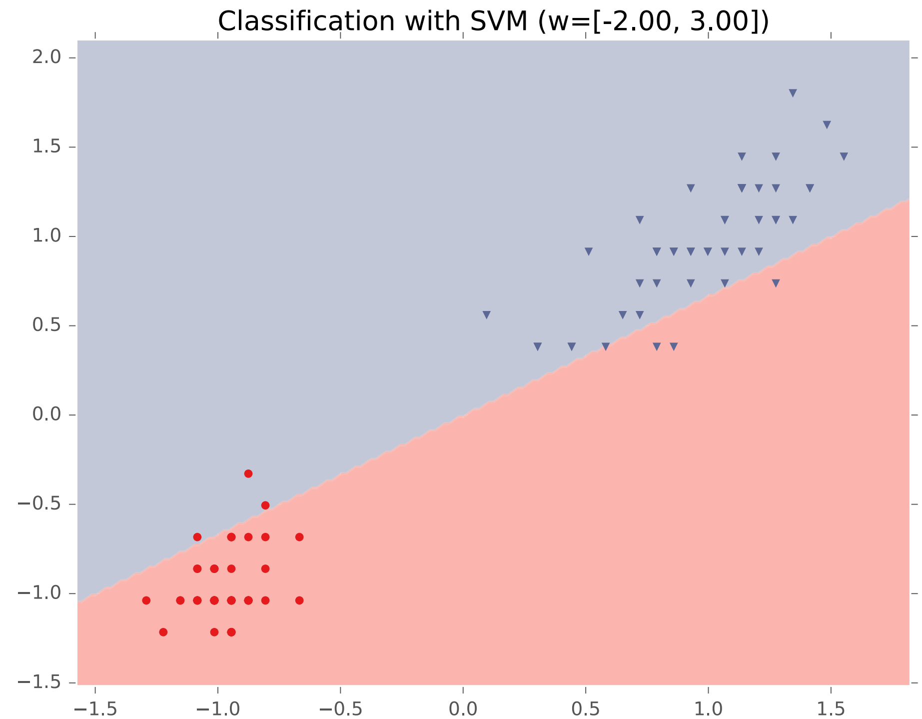
- SVM Primal (Linearly Separable Case)

This section borrows ideas from Nina Balcan's SVM lectures at CMU and Patrick Winston's "widest street" SVM lecture at MIT (https://www.youtube.com/watch?v=_PwhiWxHK8o).

SVM QP

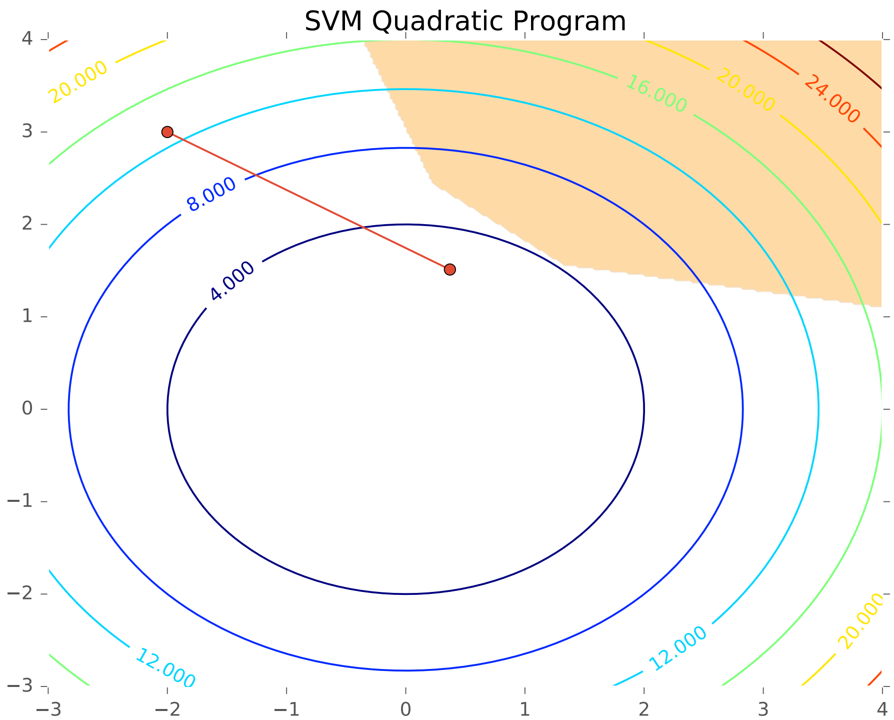


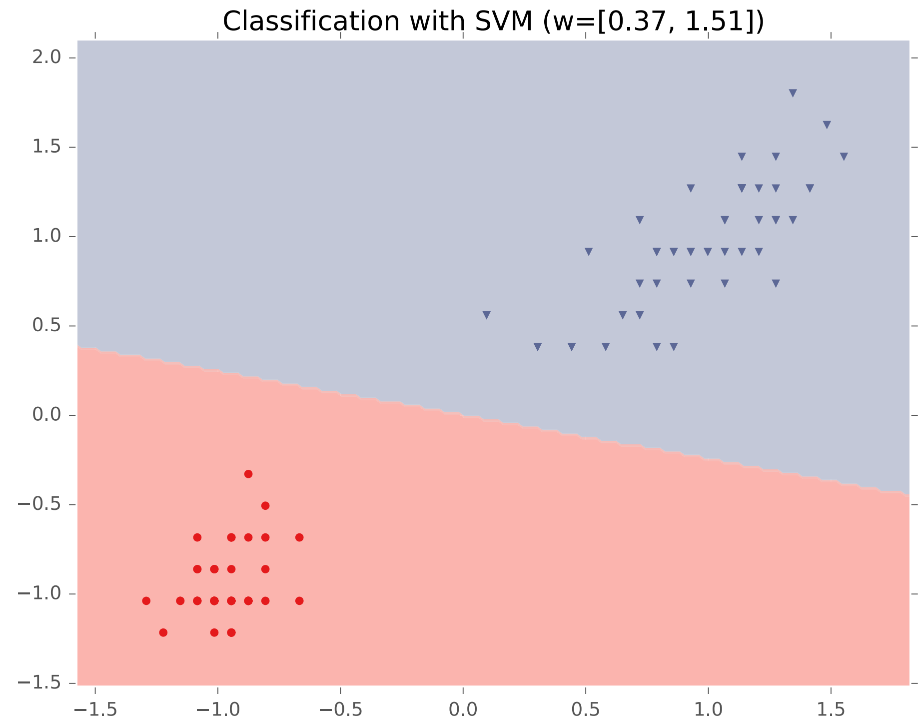
SVM Quadratic Program
20.000
16.000
24.000
8.000
4.000
12.000



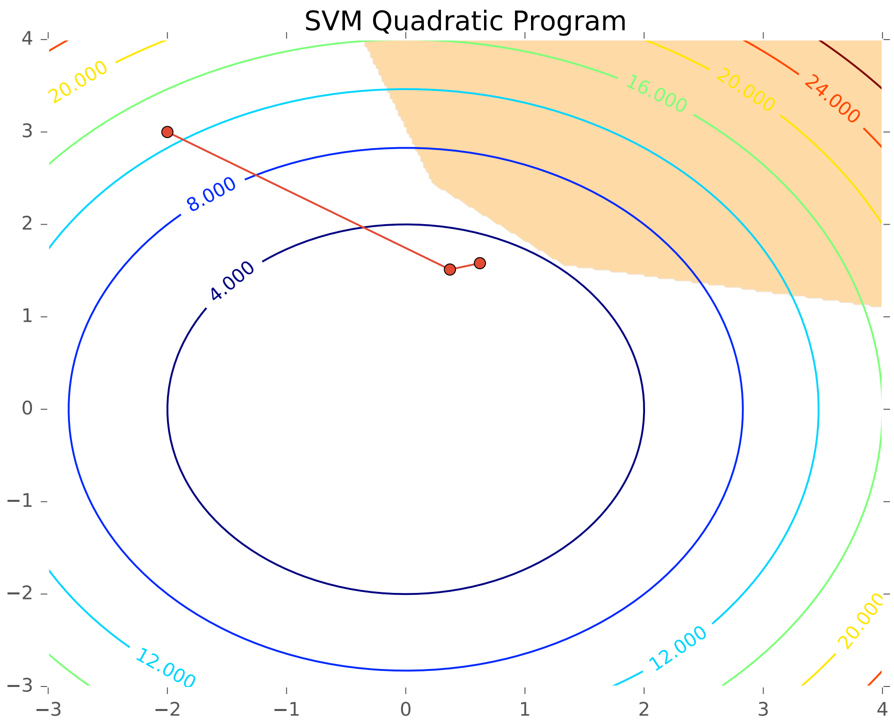
Classification with SVM (w=[-2.00, 3.00])

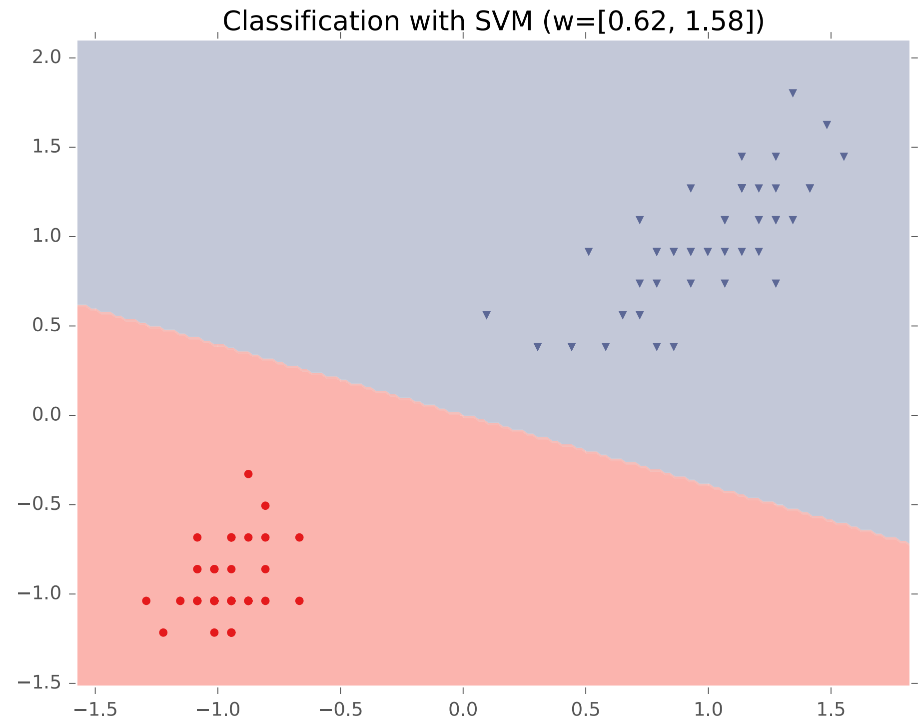
SVM QP


SVM Quadratic Program
20.000
16.000
20.000
24.000
8.000
4.000
12.000
12.000
20.000
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4

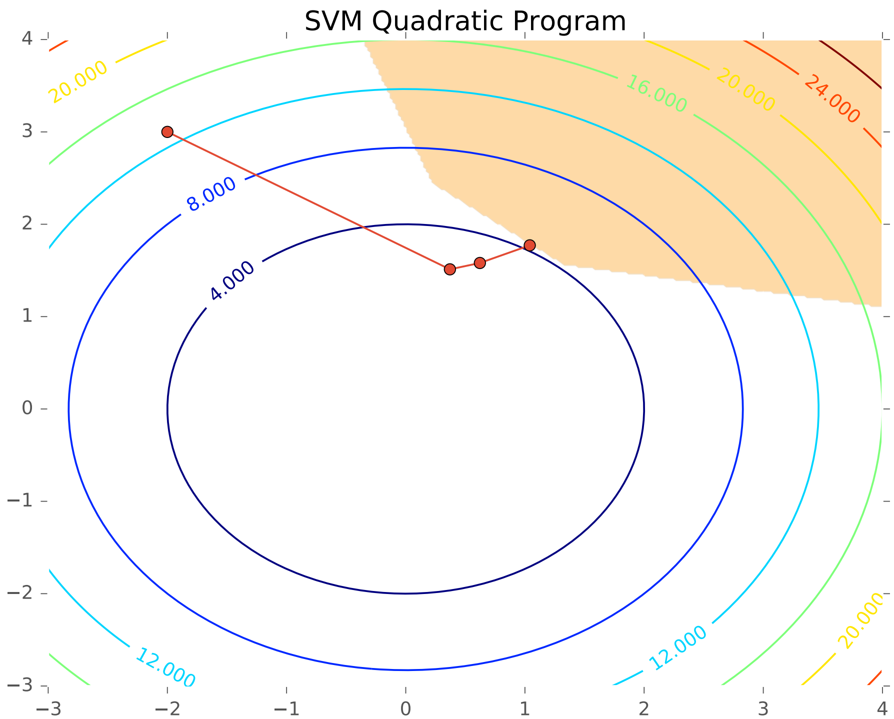

Classification with SVM (w=[0.37, 1.51])
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

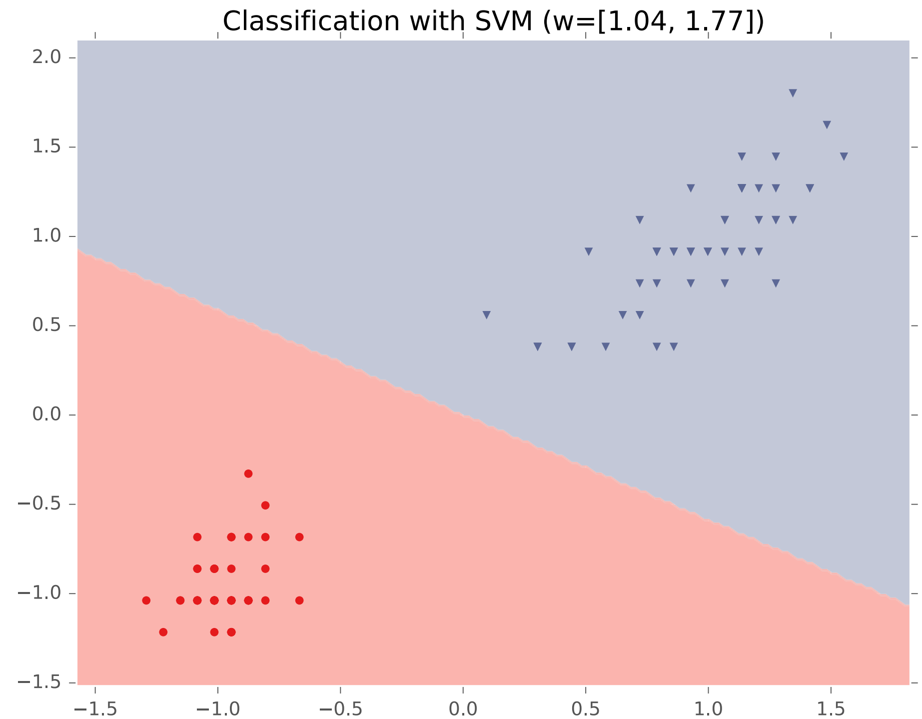
SVM QP


SVM Quadratic Program
20.000
16.000
20.000
24.000
8.000
4.000
12.000
12.000
20.000

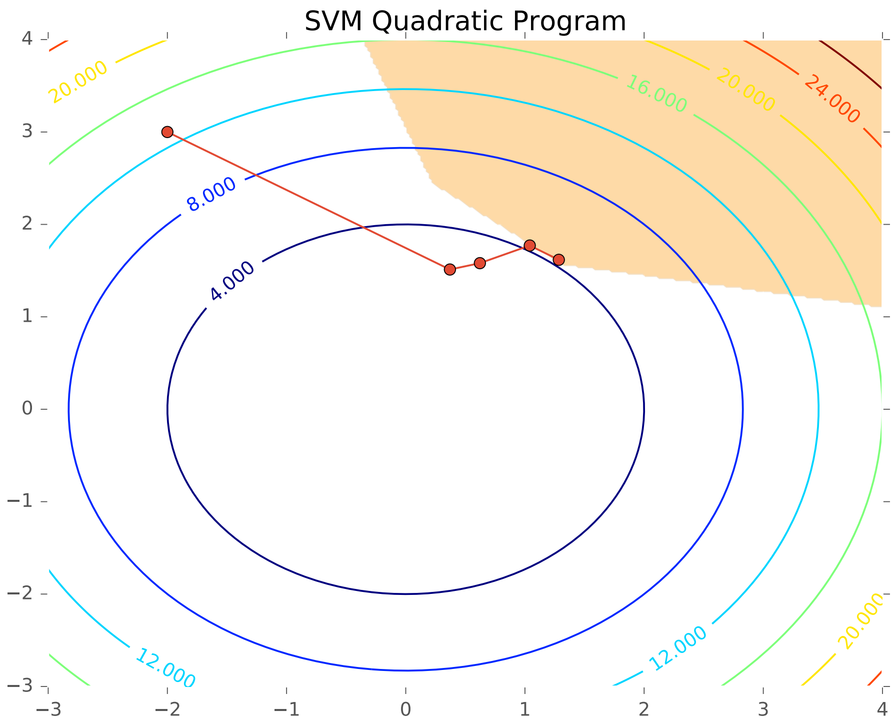

Classification with SVM (w=[0.62, 1.58])

SVM QP

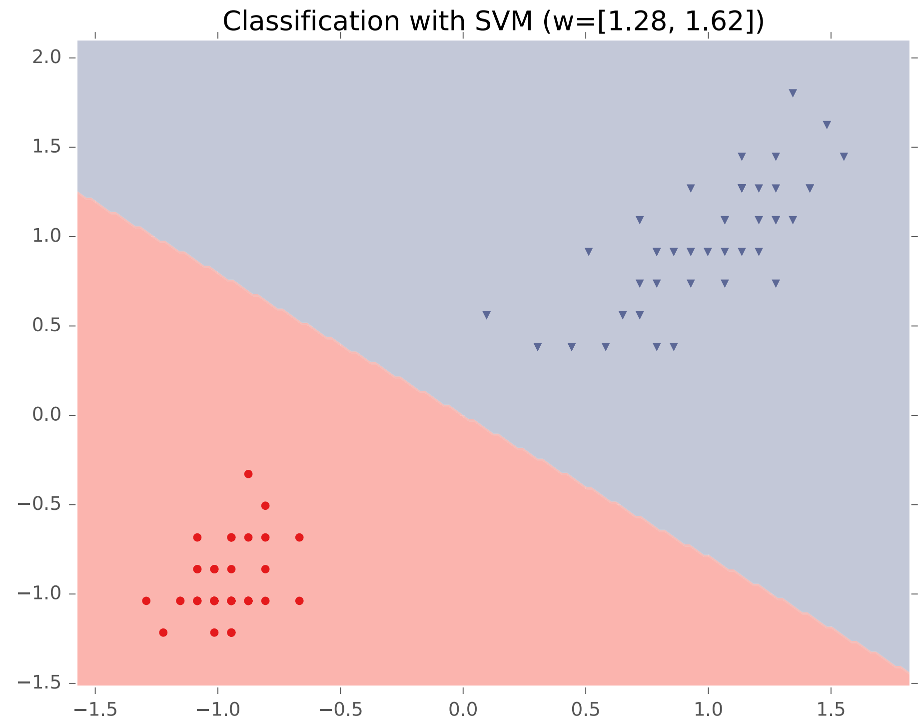

SVM Quadratic Program
20.000
16.000
20.000
24.000
8.000
4.000
12.000
12.000
20.000
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4


Classification with SVM (w=[1.04, 1.77])
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

SVM QP

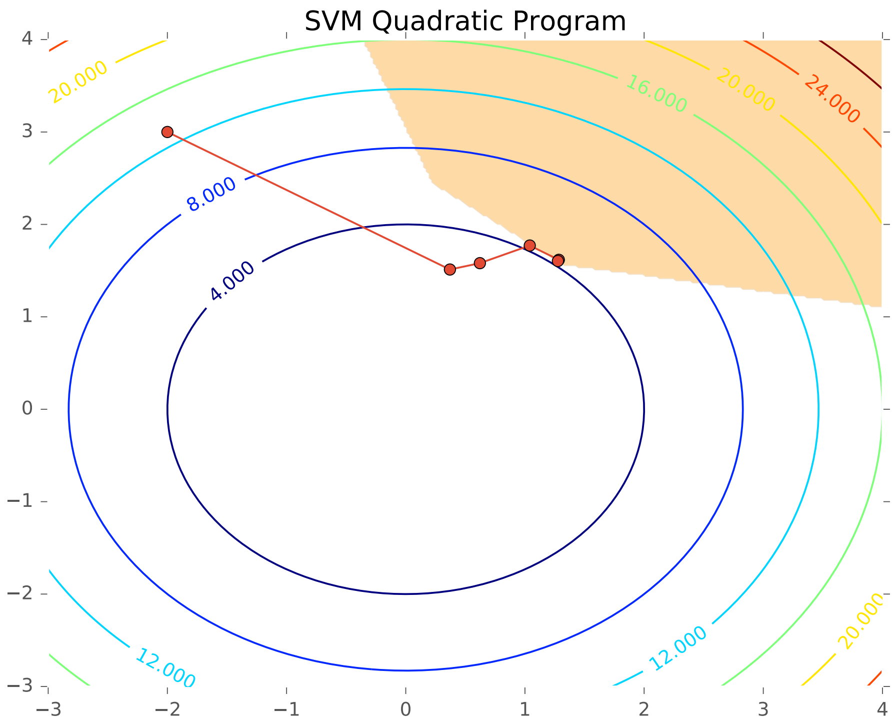


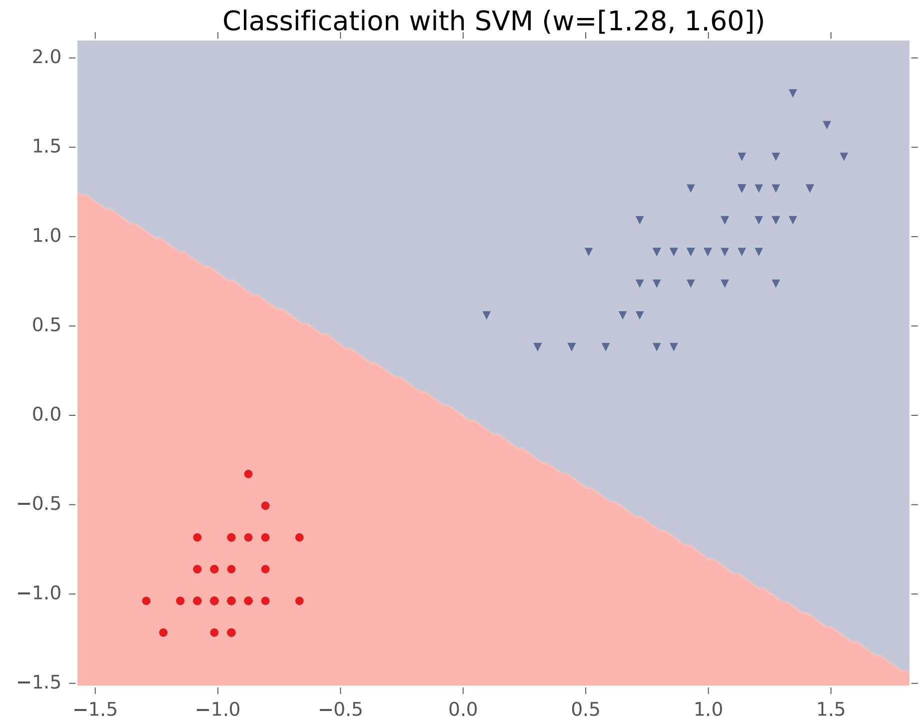
SVM Quadratic Program
20.000
16.000
24.000
12.000
8.000
4.000



Classification with SVM (w=[1.28, 1.62])

SVM QP


SVM Quadratic Program
4.000
8.000
12.000
16.000
20.000
24.000


Classification with SVM (w=[1.28, 1.60])

Support Vector Machines (SVMs)

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Hard-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$
$$\text{s.t. } \alpha_i \geq 0, \quad \forall i = 1, \ldots, N$$
$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

- Instead of minimizing the primal, we can maximize the dual problem
- For the SVM, these two problems give the same answer (i.e. the minimum of one is the maximum of the other)
- *Definition*: **support vectors** are those points $x^{(i)}$ for which $\alpha^{(i)} \neq 0$

METHOD OF LAGRANGE MULTIPLIERS

Method of Lagrange Multipliers

Method of Lagrange Multipliers (case w/inequalities)

Goal: $\min f(\vec{x})$ s.t. $g(\vec{x}) \leq c$

① Construct Lagrangian

$$L(\vec{x},\lambda) = f(\vec{x}) - \lambda\left(g(\vec{x}) - c\right)$$

② Solve

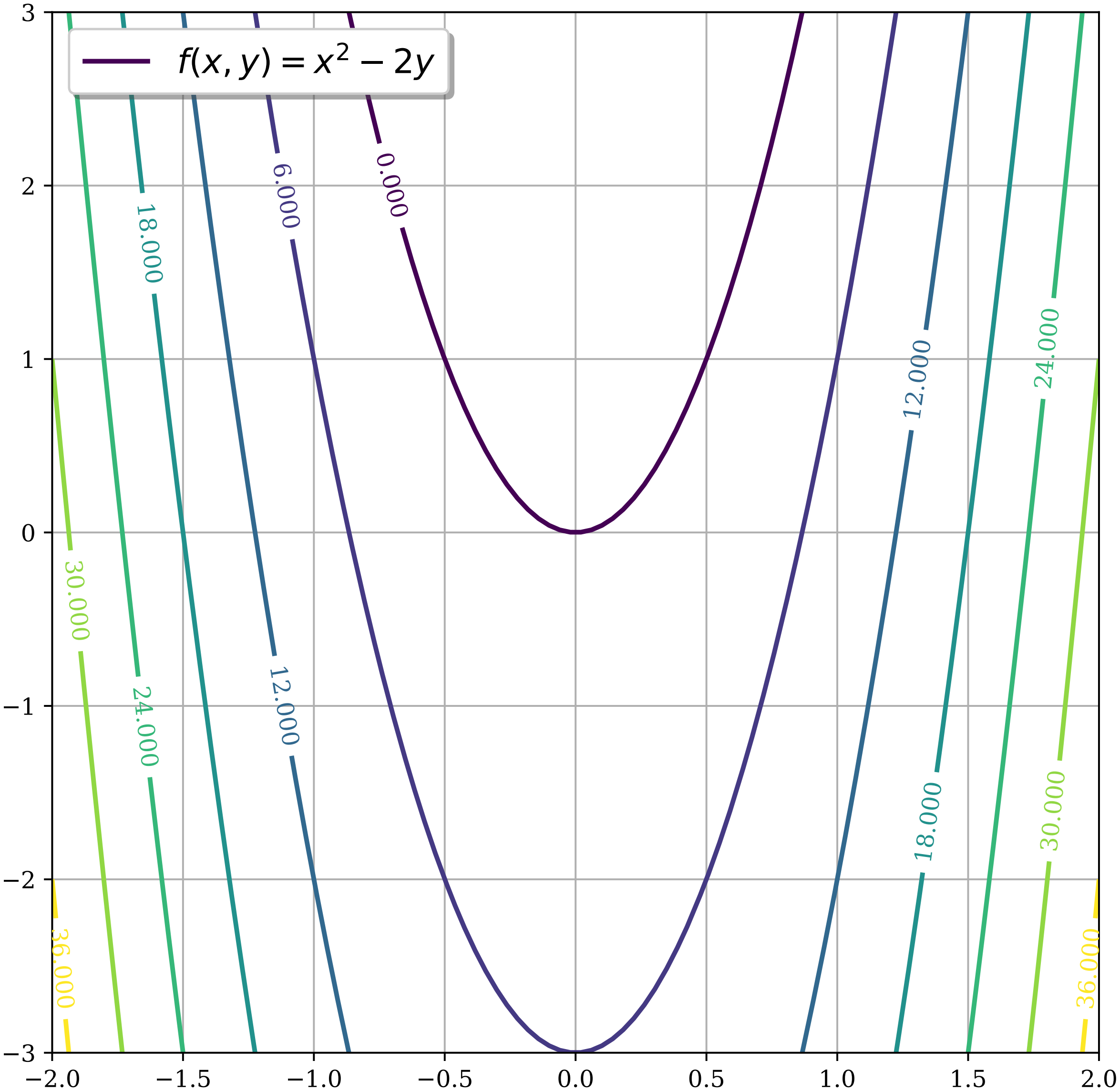
$$\min_{\vec{x}} \max_{\lambda} L(\vec{x},\lambda)$$

$$\nabla L(\vec{x},\lambda) = 0 \quad \text{s.t. } \lambda \geq 0,\ g(\vec{x}) \leq c$$

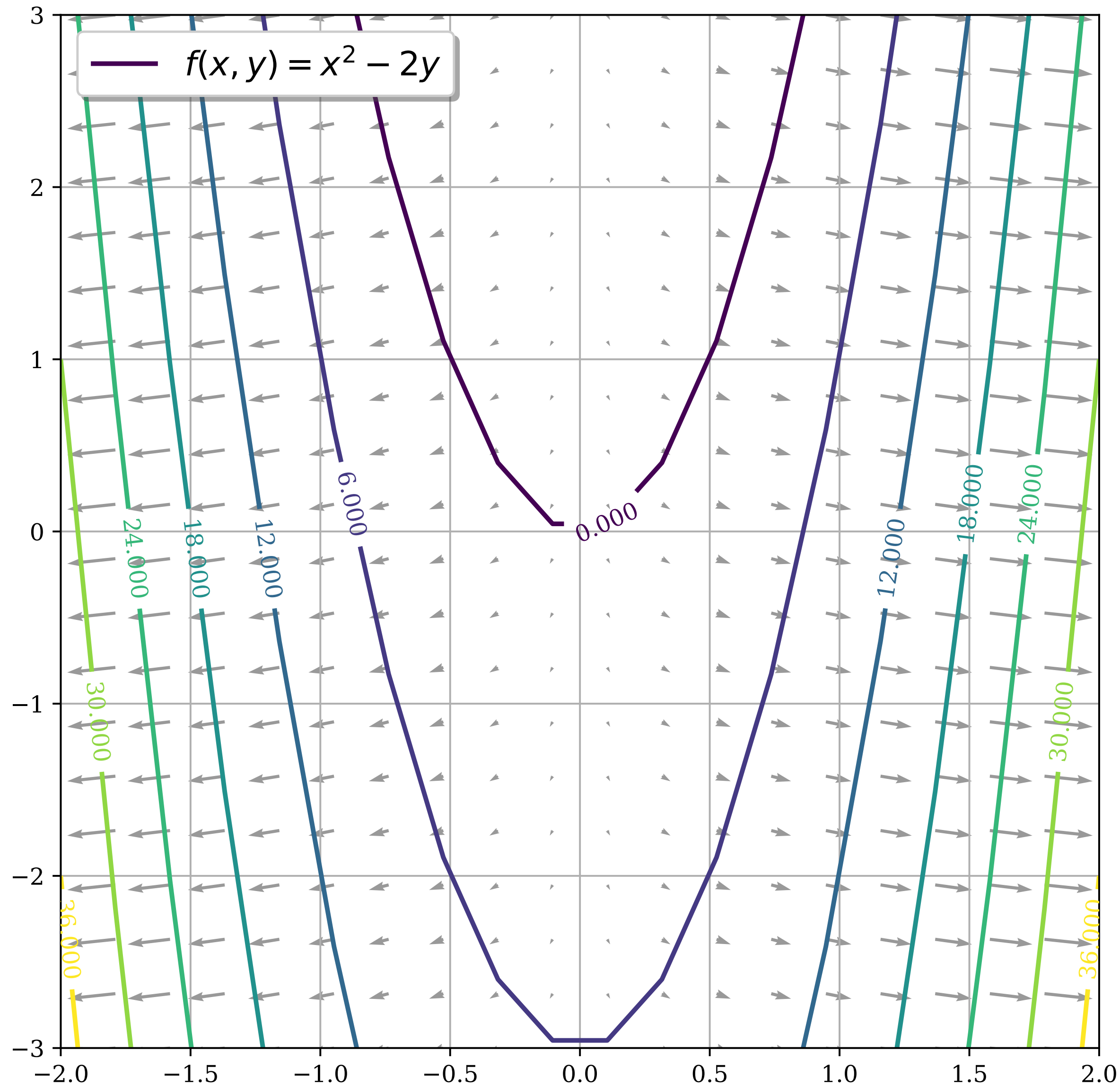
Equivalent to solving:

$$\nabla f(\vec{x}) = \lambda \nabla g(\vec{x}) \quad \text{s.t. } \lambda \geq 0,\ g(\vec{x}) \leq c$$

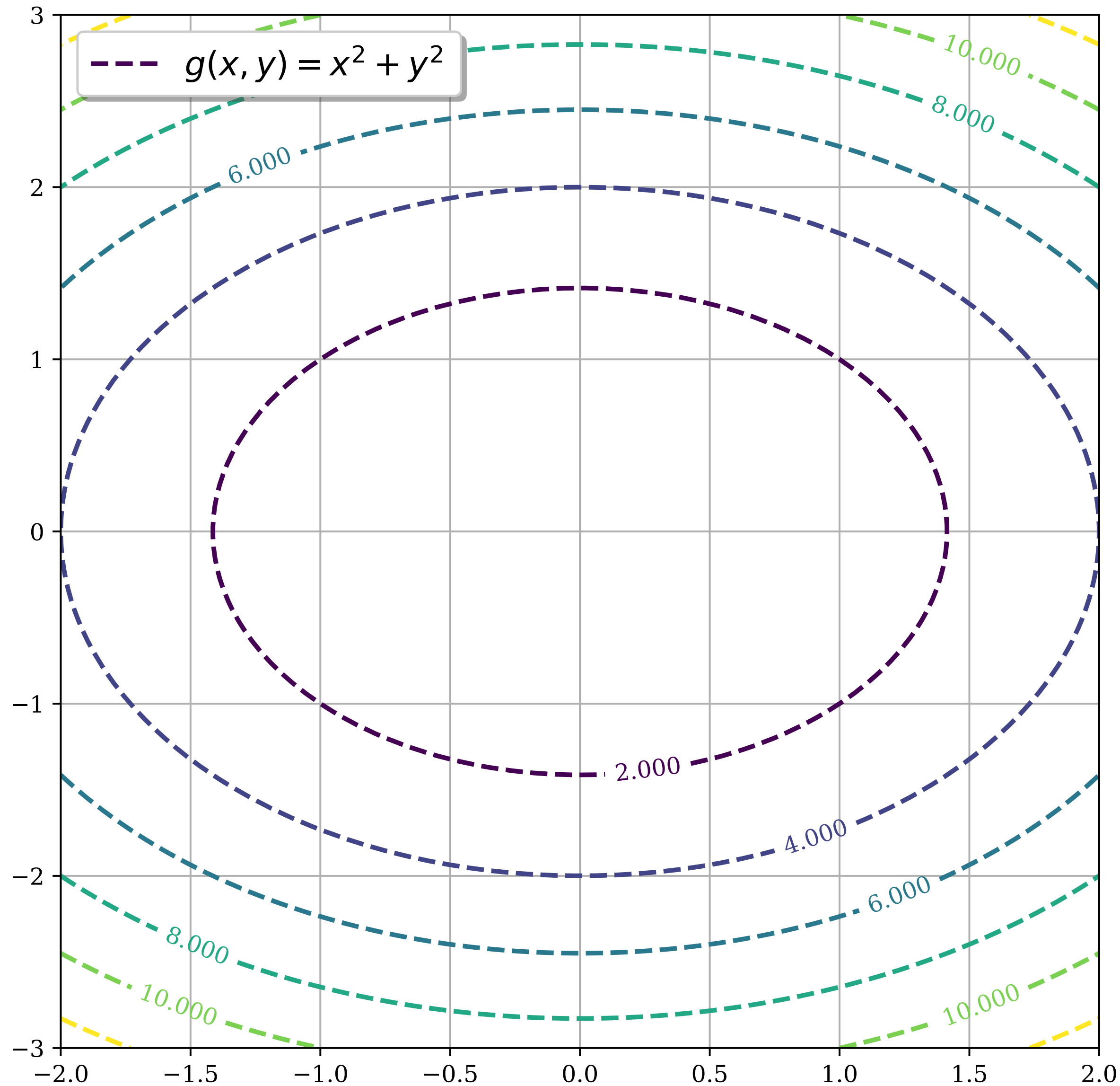
Method of Lagrange Multipliers



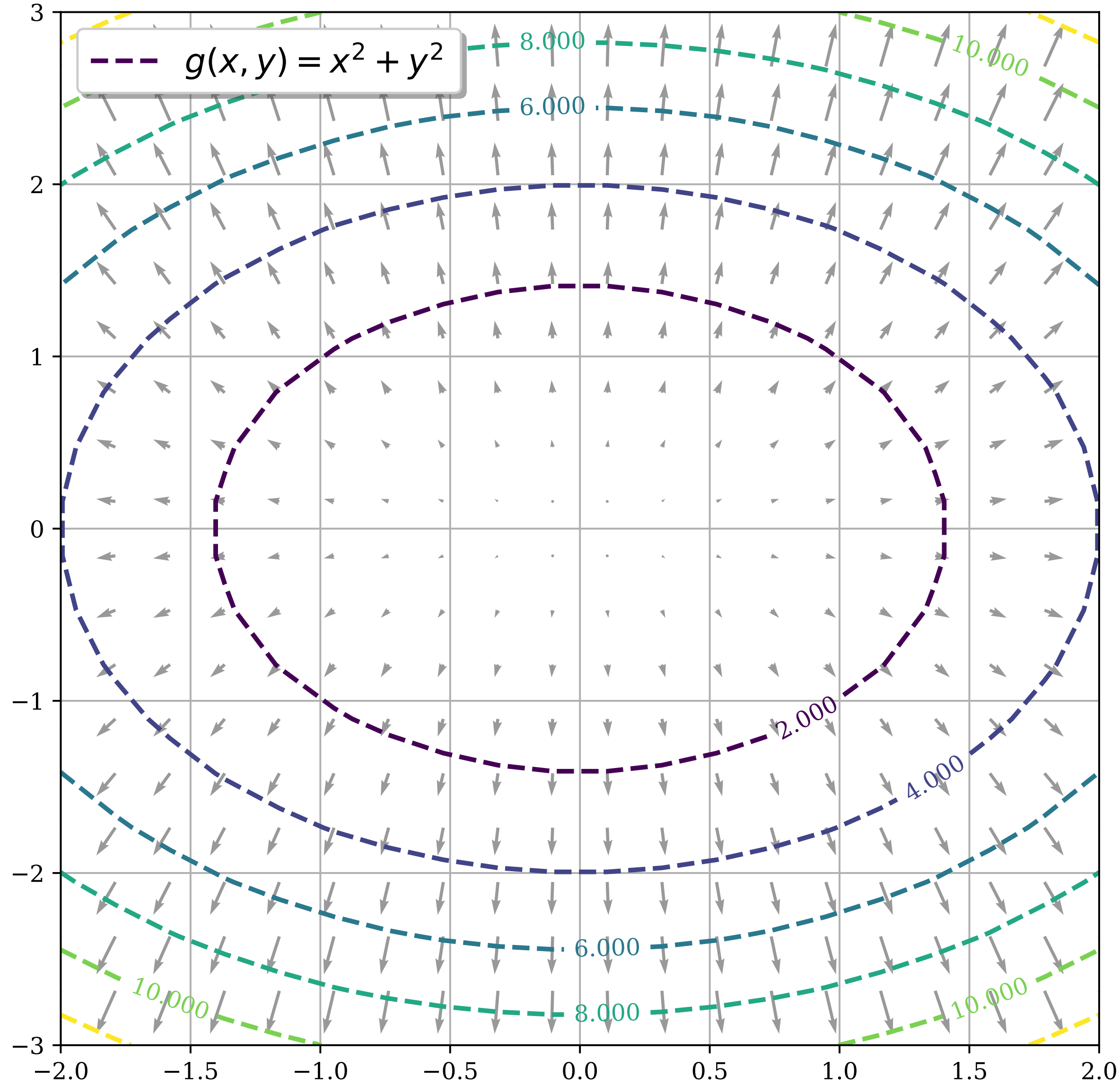
Method of Lagrange Multipliers



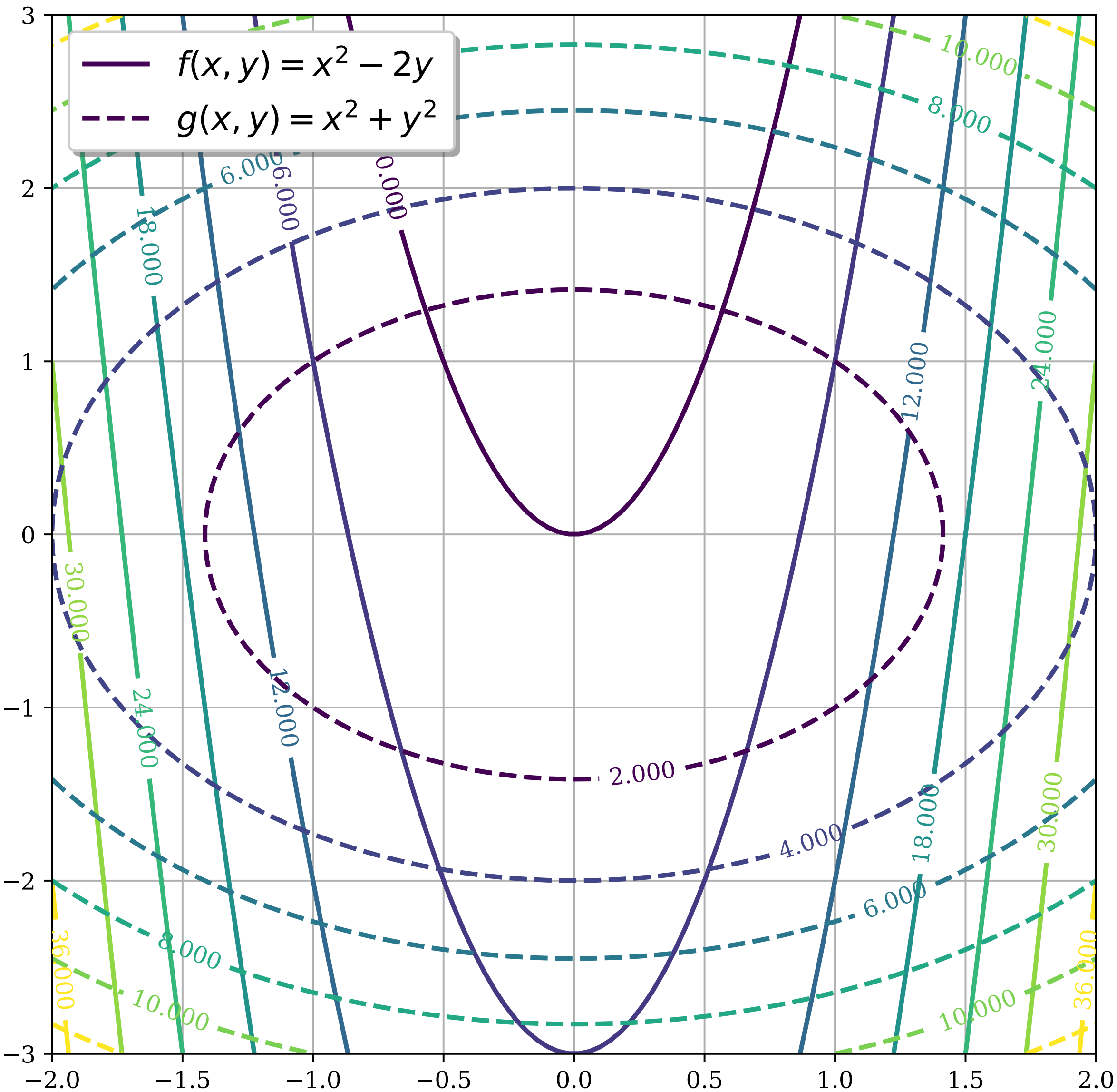
Method of Lagrange Multipliers



Method of Lagrange Multipliers



Method of Lagrange Multipliers



Method of Lagrange Multipliers

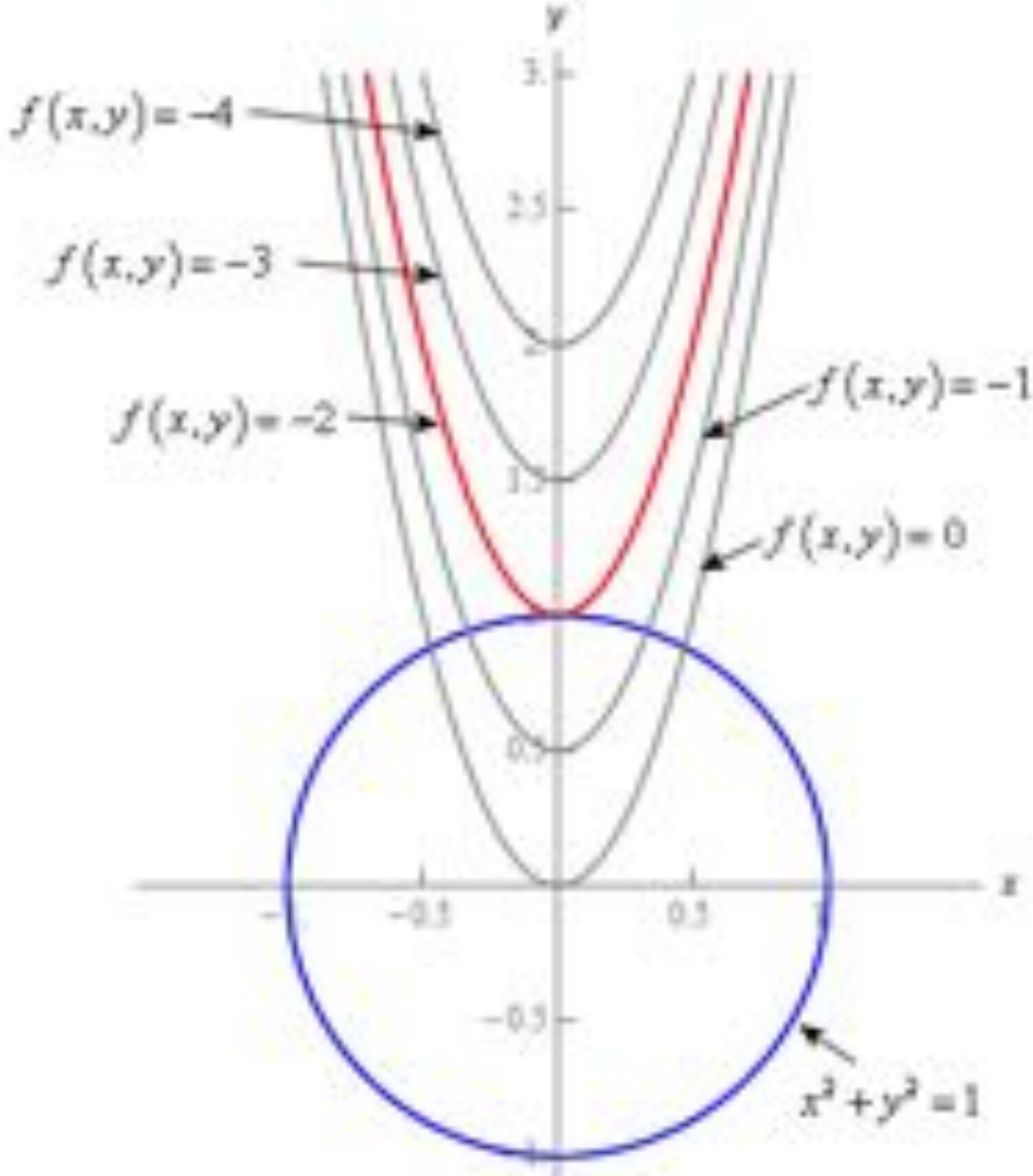


Figure from http://tutorial.math.lamar.edu/Classes/CalcIII/LagrangeMultipliers.aspx

Method of Lagrange Multipliers

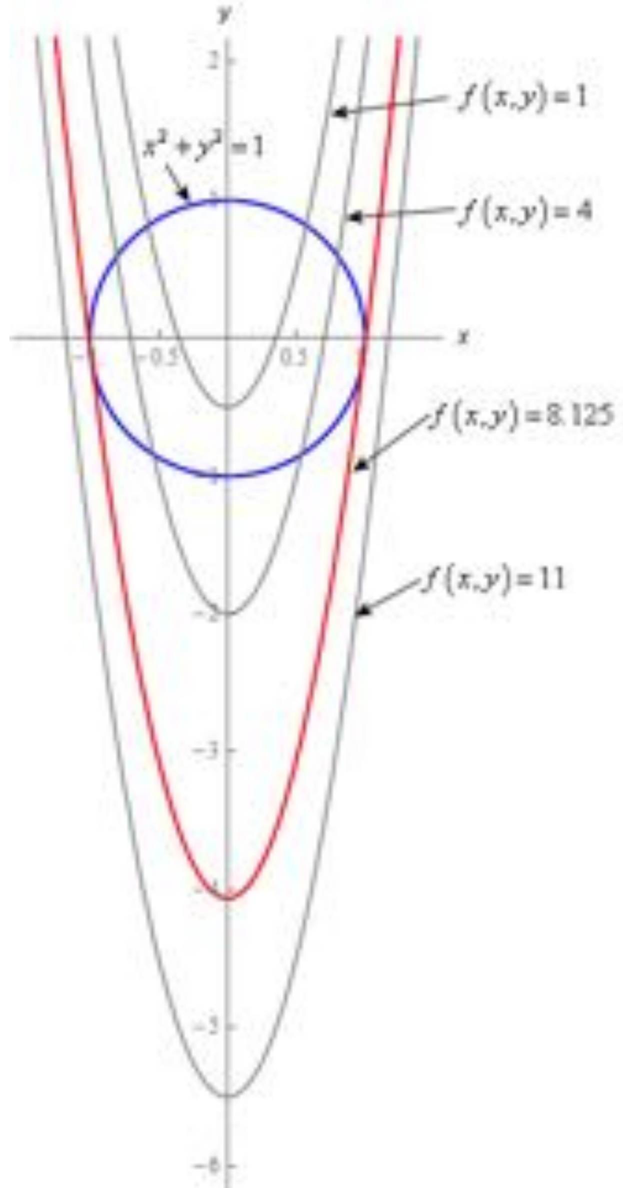


Figure from http://tutorial.math.lamar.edu/Classes/CalcIII/LagrangeMultipliers.aspx

SVM DUAL

Method of Lagrange Multipliers

Whiteboard

- Lagrangian Duality
- Example: SVM Dual

This section borrows ideas from Nina Balcan's SVM lectures at CMU and Patrick Winston's "widest street" SVM lecture at MIT (https://www.youtube.com/watch?v=_PwhiWxHK8o).

Support Vector Machines (SVMs)

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Hard-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(i)}y^{(j)}\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$

$$\text{s.t. } \alpha_i \geq 0, \quad \forall i = 1, \ldots, N$$

$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

- Instead of minimizing the primal, we can maximize the dual problem
- For the SVM, these two problems give the same answer (i.e. the minimum of one is the maximum of the other)
- *Definition*: **support vectors** are those points $x^{(i)}$ for which $\alpha^{(i)} \neq 0$

SVM EXTENSIONS

Soft-Margin SVM

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Soft-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\left(\sum_{i=1}^{N} e_i\right)$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - e_i, \quad \forall i = 1, \ldots, N$$
$$e_i \geq 0, \quad \forall i = 1, \ldots, N$$

- **Question:** If the dataset is not linearly separable, can we still use an SVM?
- **Answer:** Not the hard-margin version. It will never find a feasible solution.

  In the soft-margin version, we add **"slack variables"** that **allow some points to violate** the large-margin constraints.

  The constant C dictates **how large** we should allow the slack variables to be

Soft-Margin SVM

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Soft-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\left(\sum_{i=1}^{N} e_i\right)$$

$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - e_i, \quad \forall i = 1, \ldots, N$$

$$e_i \geq 0, \quad \forall i = 1, \ldots, N$$

Soft-Margin SVM

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i = 1, \ldots, N$$

Hard-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(i)}y^{(j)}\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$
$$\text{s.t. } \alpha_i \geq 0, \quad \forall i = 1, \ldots, N$$
$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

Soft-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\left(\sum_{i=1}^{N} e_i\right)$$
$$\text{s.t. } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - e_i, \quad \forall i = 1, \ldots, N$$
$$e_i \geq 0, \quad \forall i = 1, \ldots, N$$

Soft-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(i)}y^{(j)}\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$$
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, N$$
$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

We can also work with the dual of the soft-margin SVM

Multiclass SVMs

The SVM is **inherently** a **binary** classification method, but can be extended to handle K-class classification in many ways.

1. ***one-vs-rest*:**
   - build K binary classifiers
   - train the k^{th} classifier to predict whether an instance has label k or something else
   - predict the class with largest score
2. ***one-vs-one*:**
   - build (K choose 2) binary classifiers
   - train one classifier for distinguishing between each pair of labels
   - predict the class with the most "votes" from any given classifier

Learning Objectives

Support Vector Machines

You should be able to…

1. Motivate the learning of a decision boundary with large margin
2. Compare the decision boundary learned by SVM with that of Perceptron
3. Distinguish unconstrained and constrained optimization
4. Compare linear and quadratic mathematical programs
5. Derive the hard-margin SVM primal formulation
6. Derive the Lagrangian dual for a hard-margin SVM
7. Describe the mathematical properties of support vectors and provide an intuitive explanation of their role
8. Draw a picture of the weight vector, bias, decision boundary, training examples, support vectors, and margin of an SVM
9. Employ slack variables to obtain the soft-margin SVM
10. Implement an SVM learner using a black-box quadratic programming (QP) solver

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

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, …

SVM: Kernel Trick

Hard-margin SVM (Primal)

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

s.t. $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad \forall i$

- Suppose we do some feature engineering
- Our feature function is ϕ
- We apply ϕ to each input vector **x**

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2$$

s.t. $y^{(i)}(\mathbf{w}^T\phi\left(\mathbf{x}^{(i)}\right) + b) \geq 1, \quad \forall i$

Hard-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}$$

s.t. $\alpha_i \geq 0, \quad \forall i = 1, \dots, N$

$$\sum_{i=1}^{N}\alpha_i y^{(i)} = 0$$

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}\phi\left(\mathbf{x}^{(i)}\right)\cdot\phi\left(\mathbf{x}^{(j)}\right)$$

s.t. $\alpha_i \geq 0, \quad \forall i = 1, \dots, N$

$$\sum_{i=1}^{N}\alpha_i y^{(i)} = 0$$

SVM: Kernel Trick

Hard-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \phi\left(\mathbf{x}^{(i)}\right) \cdot \phi\left(\mathbf{x}^{(j)}\right)$$

s.t. $\alpha_i \geq 0, \quad \forall i = 1, \ldots, N$

$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

We could replace the dot product of the two feature vectors in the transformed space with a function k(x,z)
where $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi\left(\mathbf{x}^{(i)}\right) \cdot \phi\left(\mathbf{x}^{(j)}\right)$

SVM: Kernel Trick

Hard-margin SVM (Lagrangian Dual)

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

s.t. $\alpha_i \geq 0, \quad \forall i = 1, \ldots, N$

$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

We could replace the dot product of the two feature vectors in the transformed space with a function k(x,z)
where $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi\left(\mathbf{x}^{(i)}\right) \cdot \phi\left(\mathbf{x}^{(j)}\right)$

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:

  $k(x, z) = \varphi(x)^T \varphi(z)$ 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

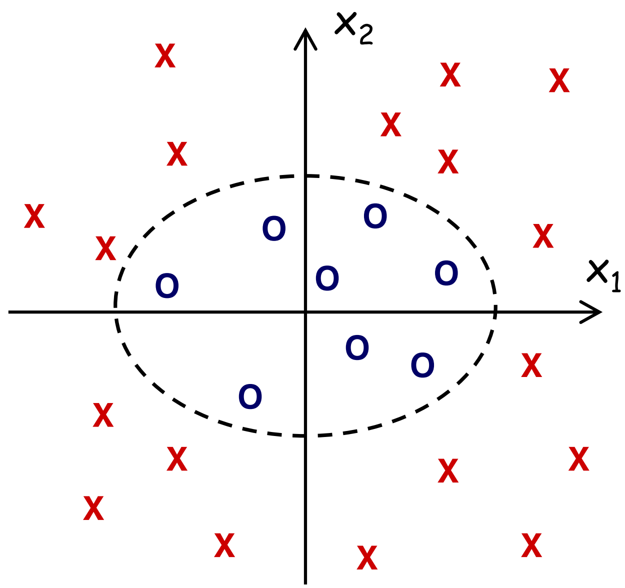
For n=2, d=2, the kernel $K(x,z) = (x \cdot z)^d$ corresponds to

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

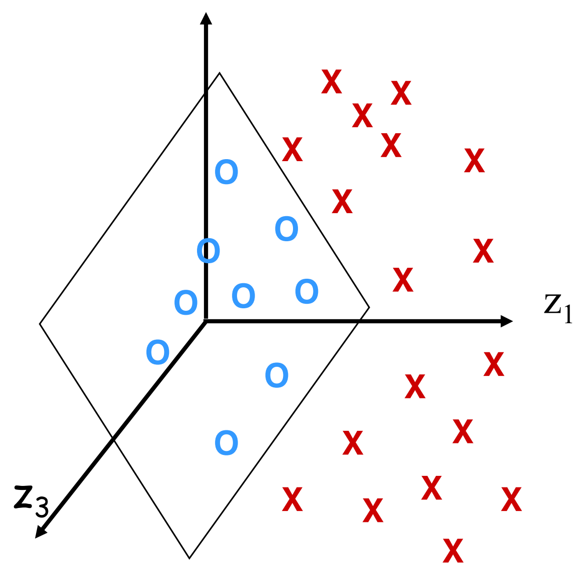
$$\phi(x) \cdot \phi(z) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2)$$

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

Original space



Φ-space

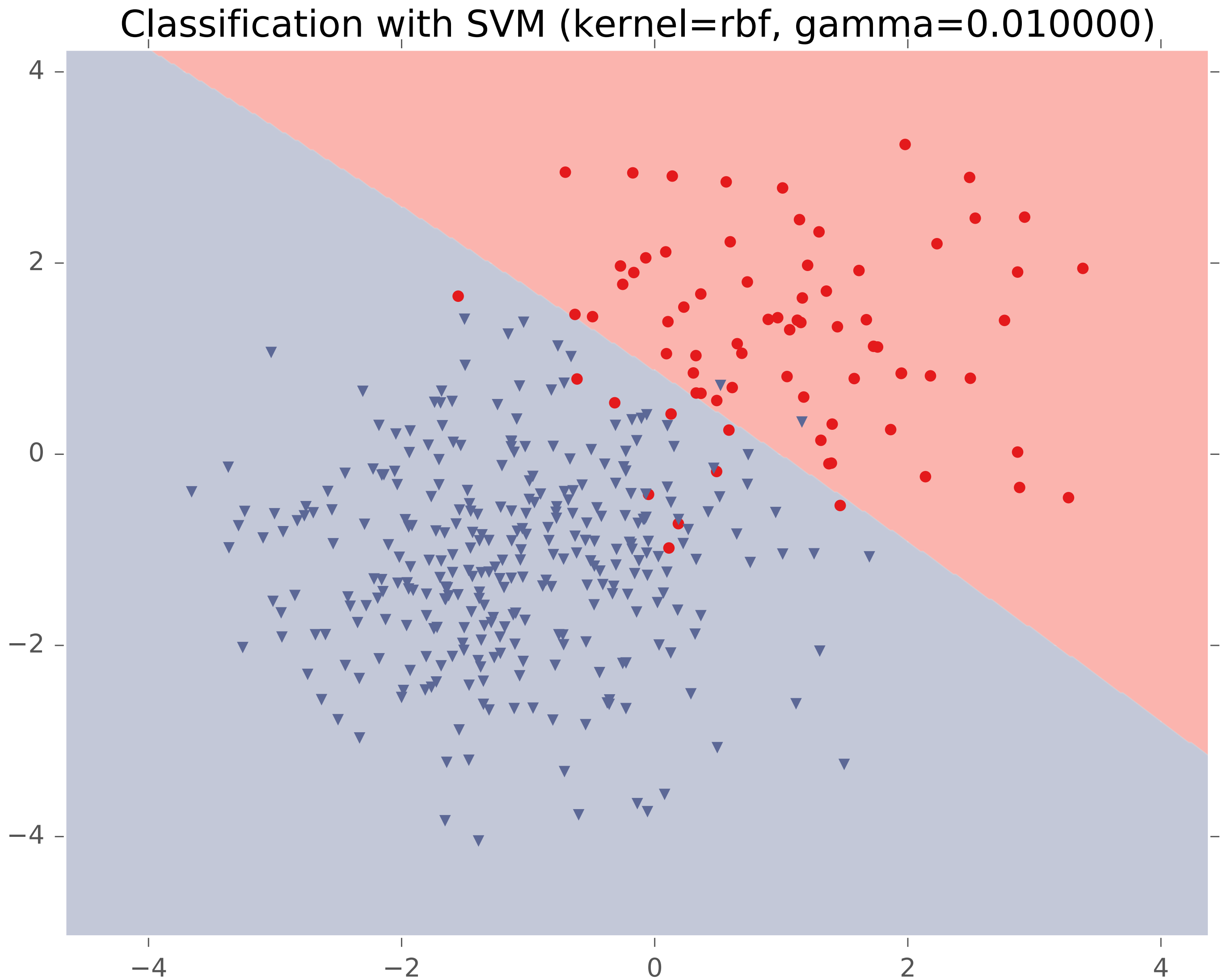


Slide from Nina Balcan

Kernel Examples

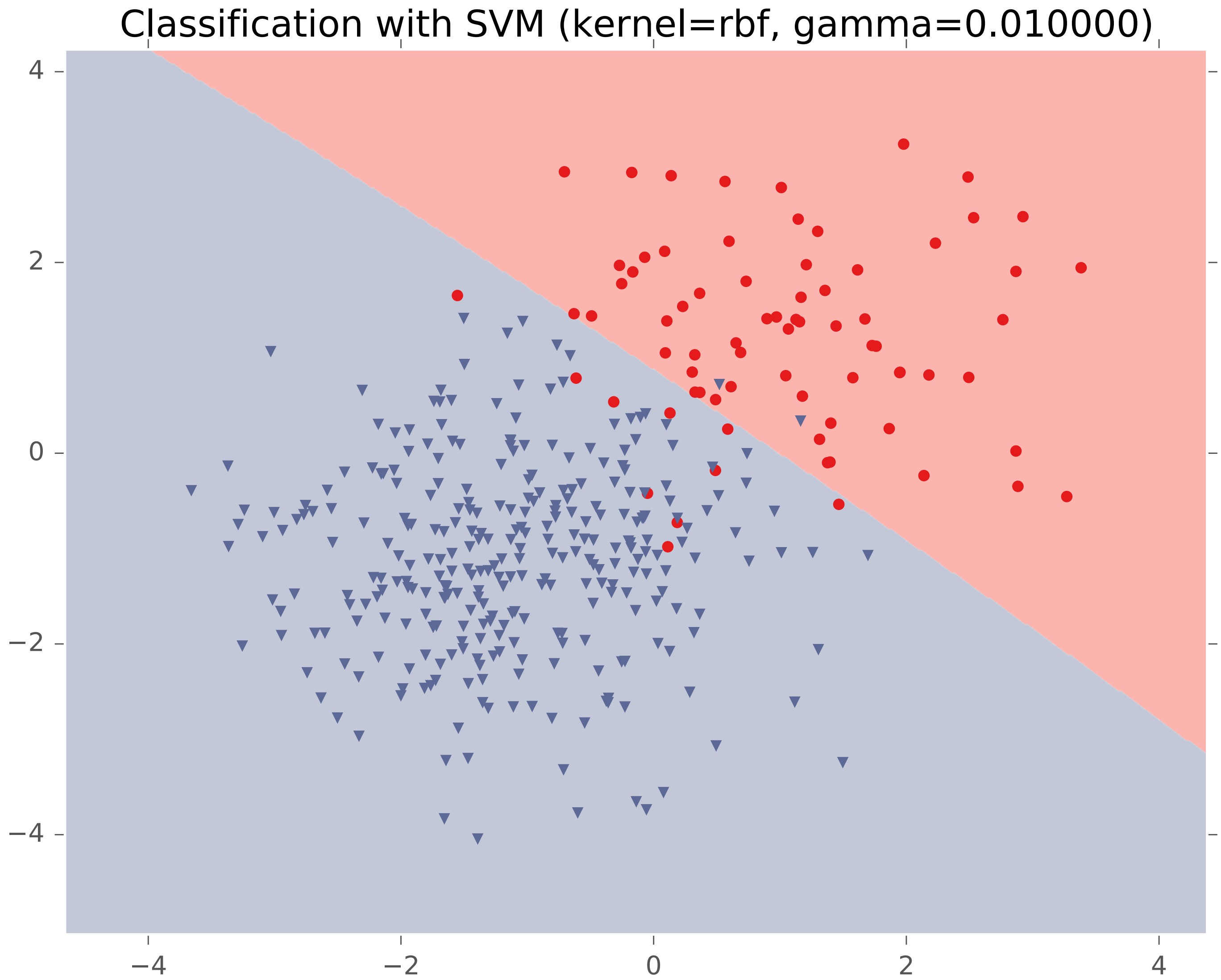
Name	Kernel Function (implicit dot product)	Feature Space (explicit dot product)
Linear	$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + 1)^d$	All polynomials **up to** degree d
Gaussian	$K(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|\|\mathbf{x} - \mathbf{z}\|\|_2^2}{2\sigma^2})$	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



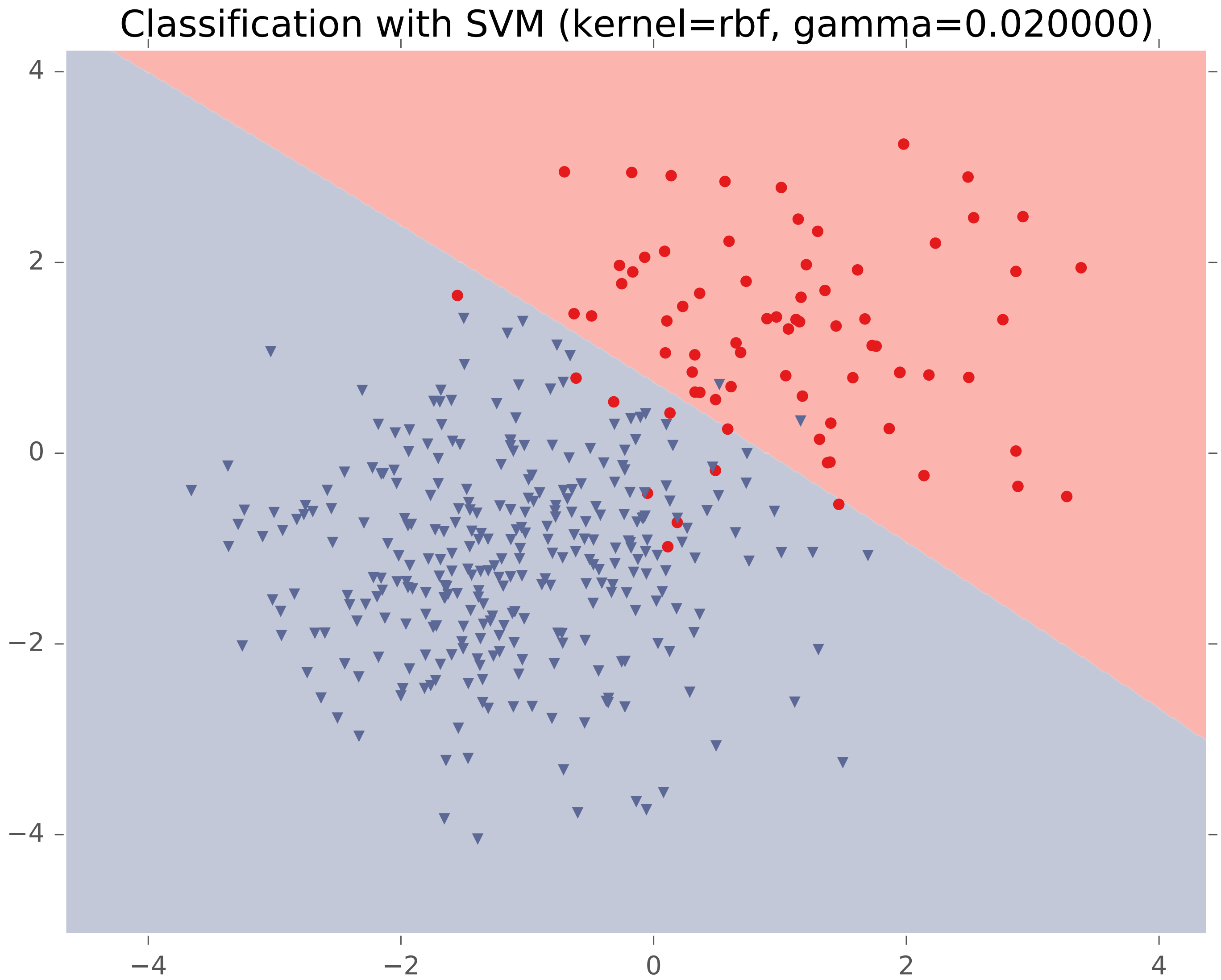
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



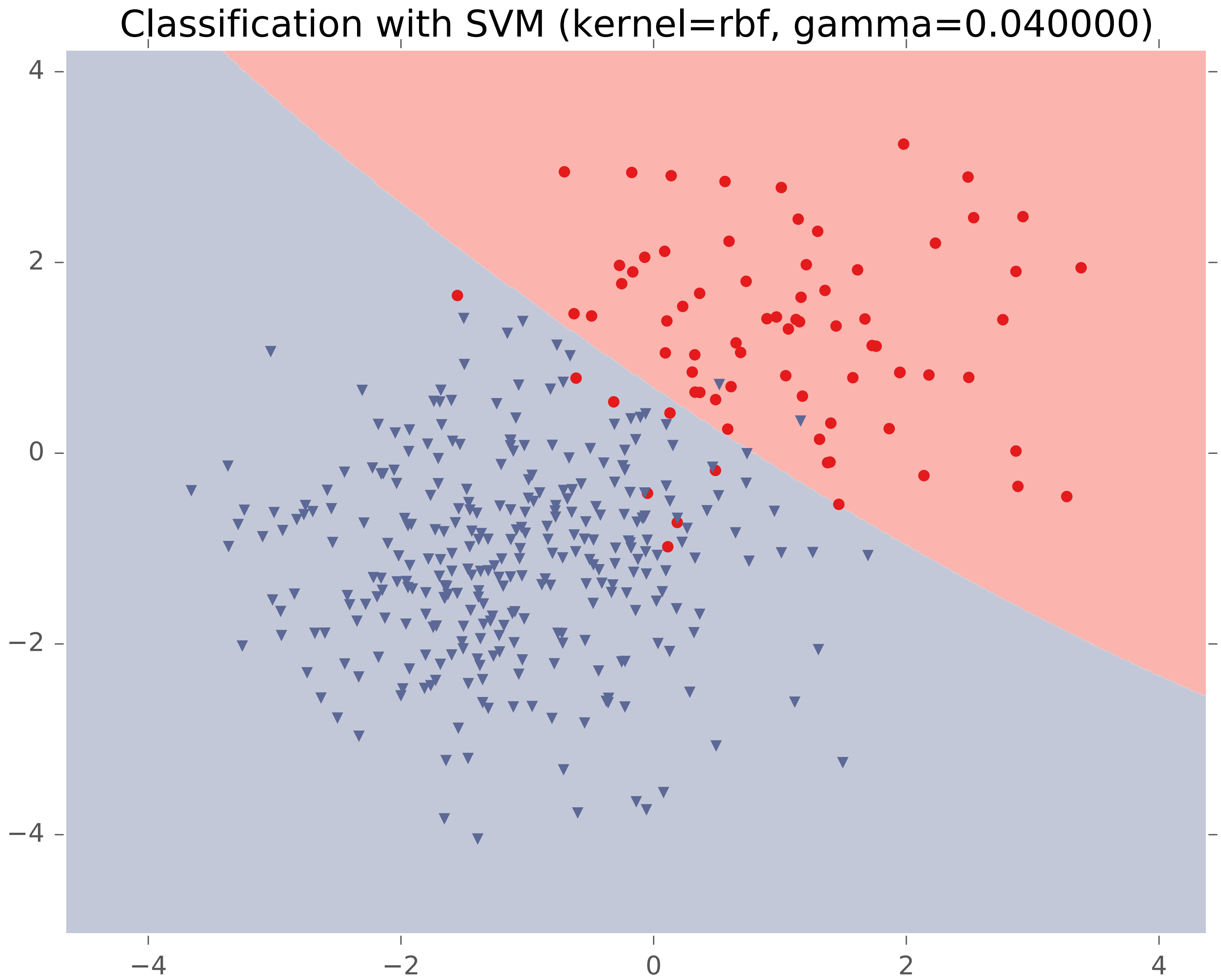
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



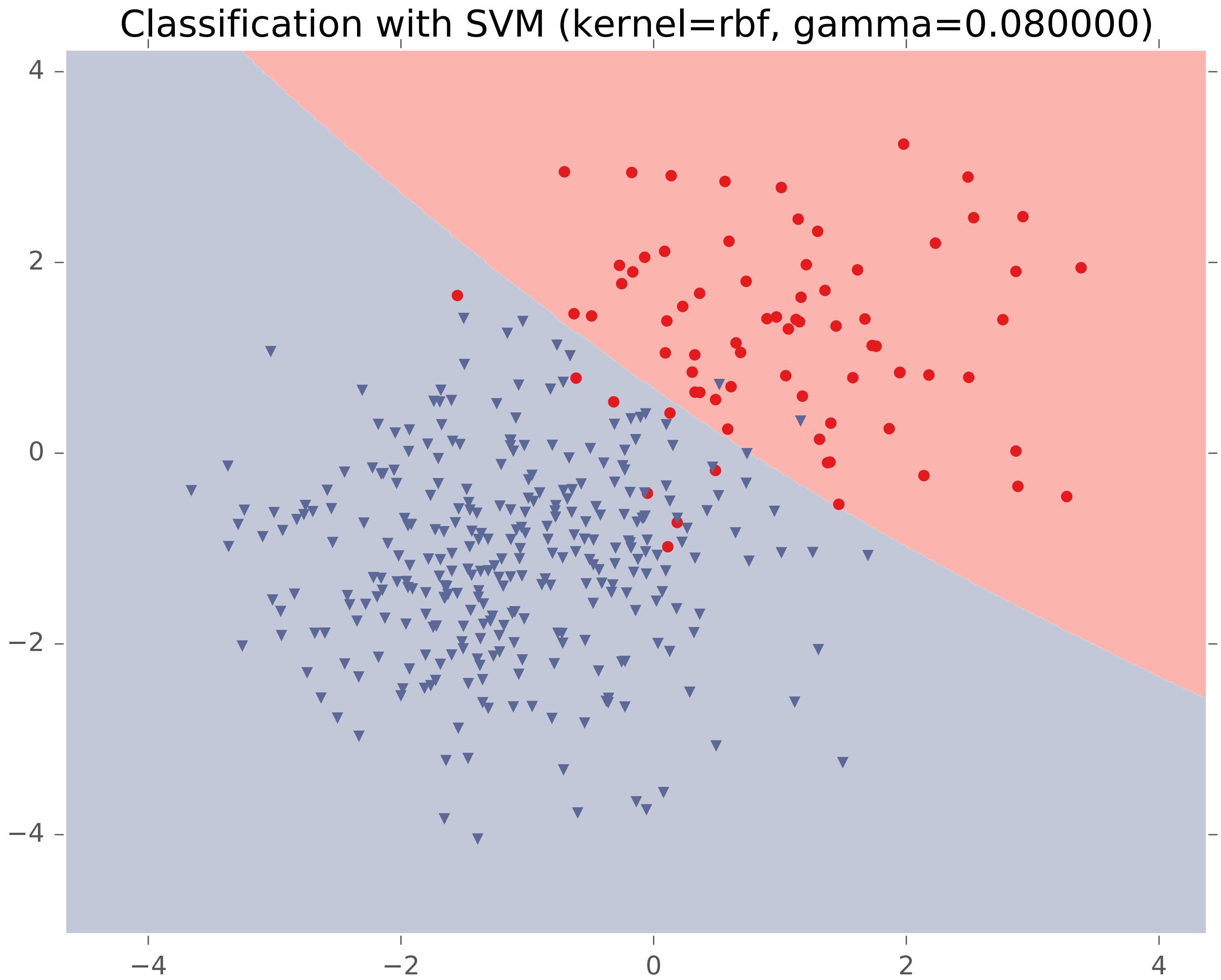
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



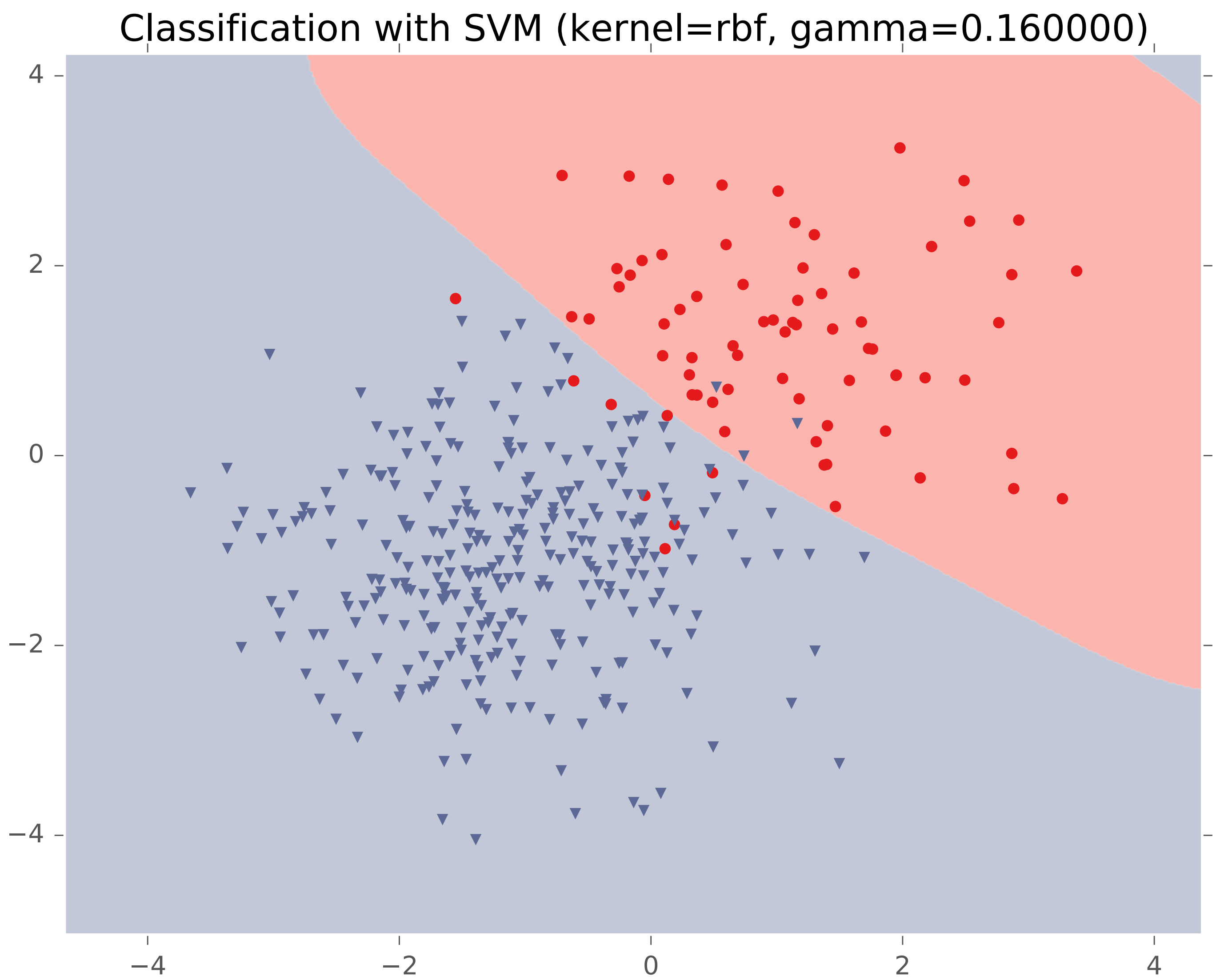
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



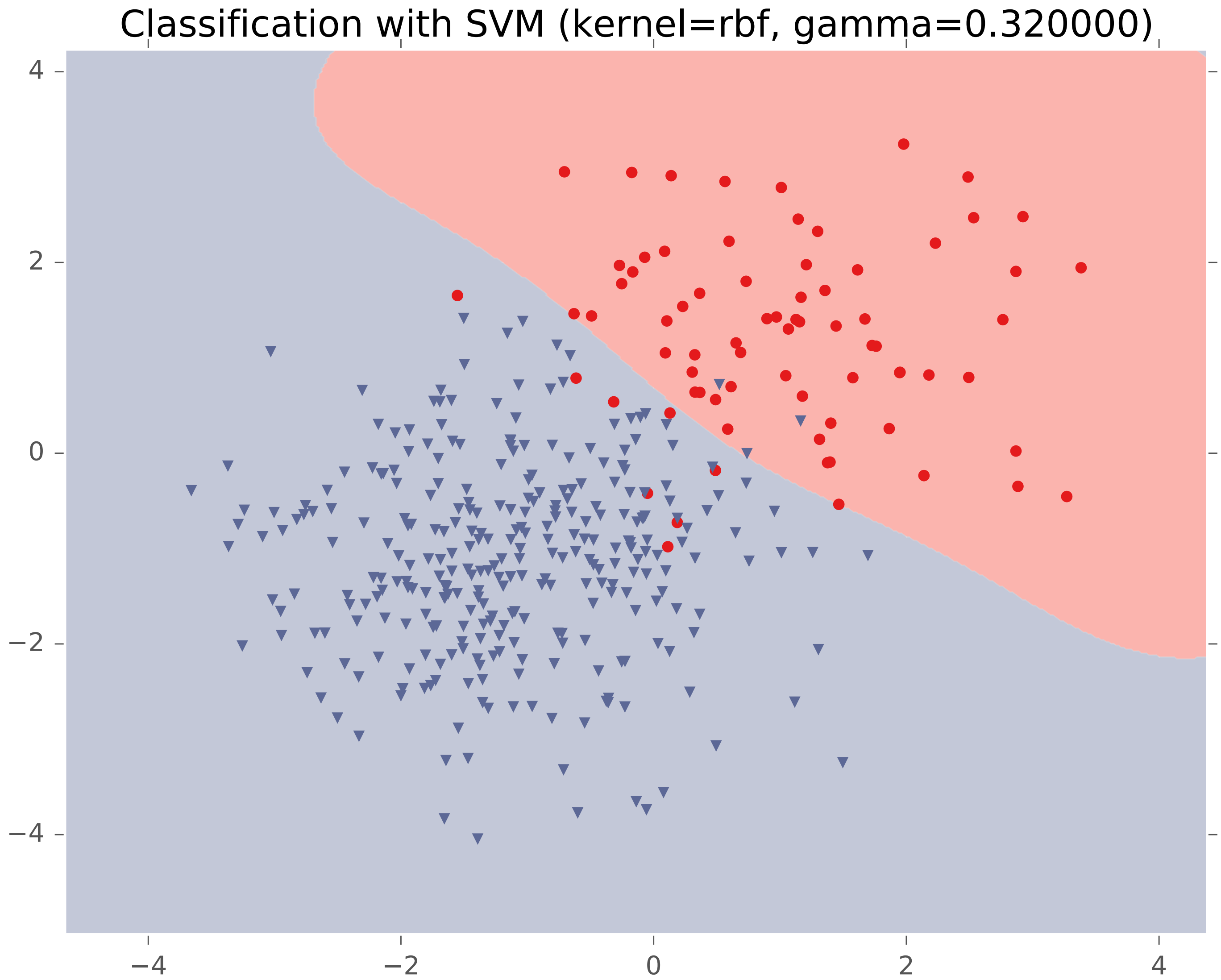
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



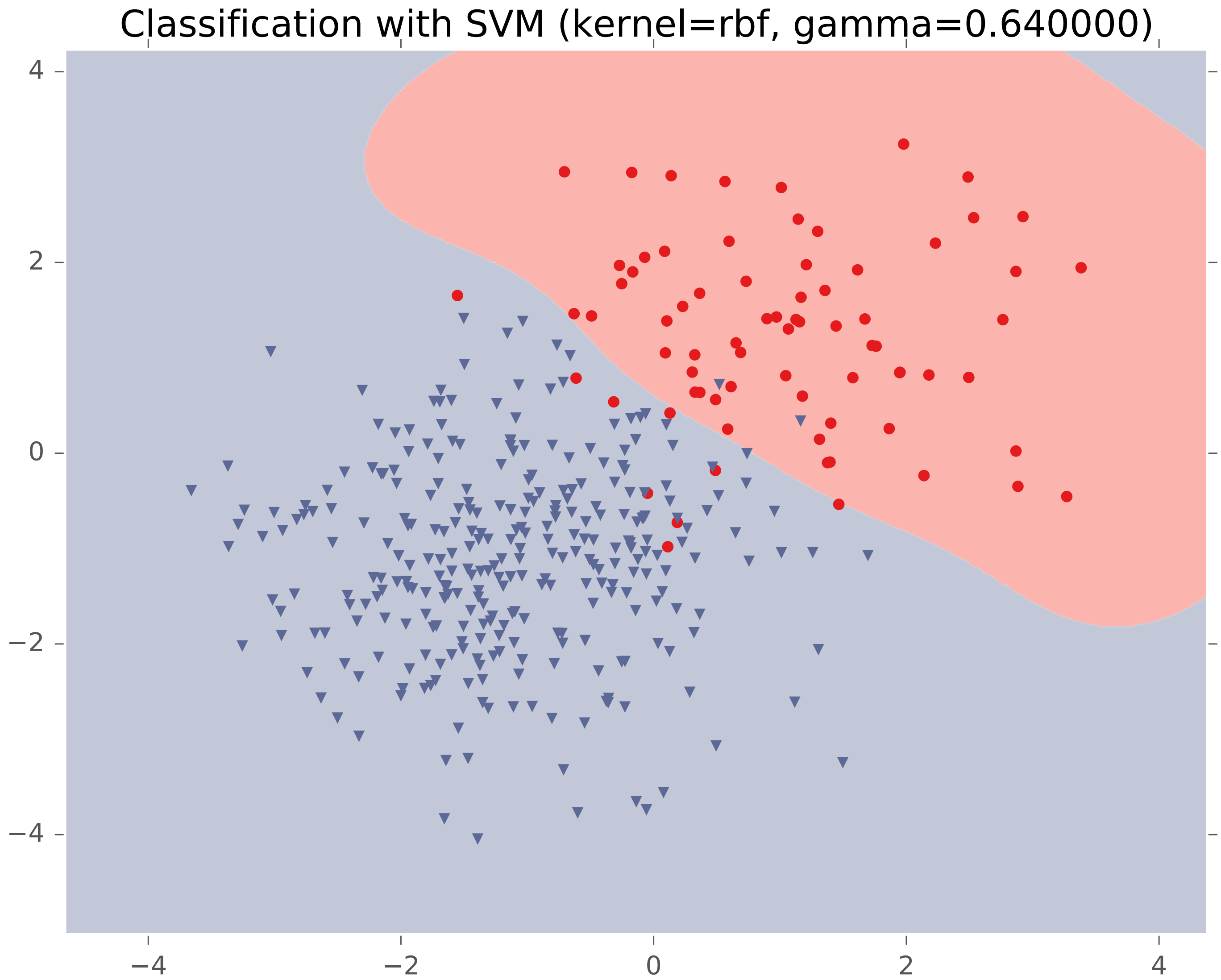
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



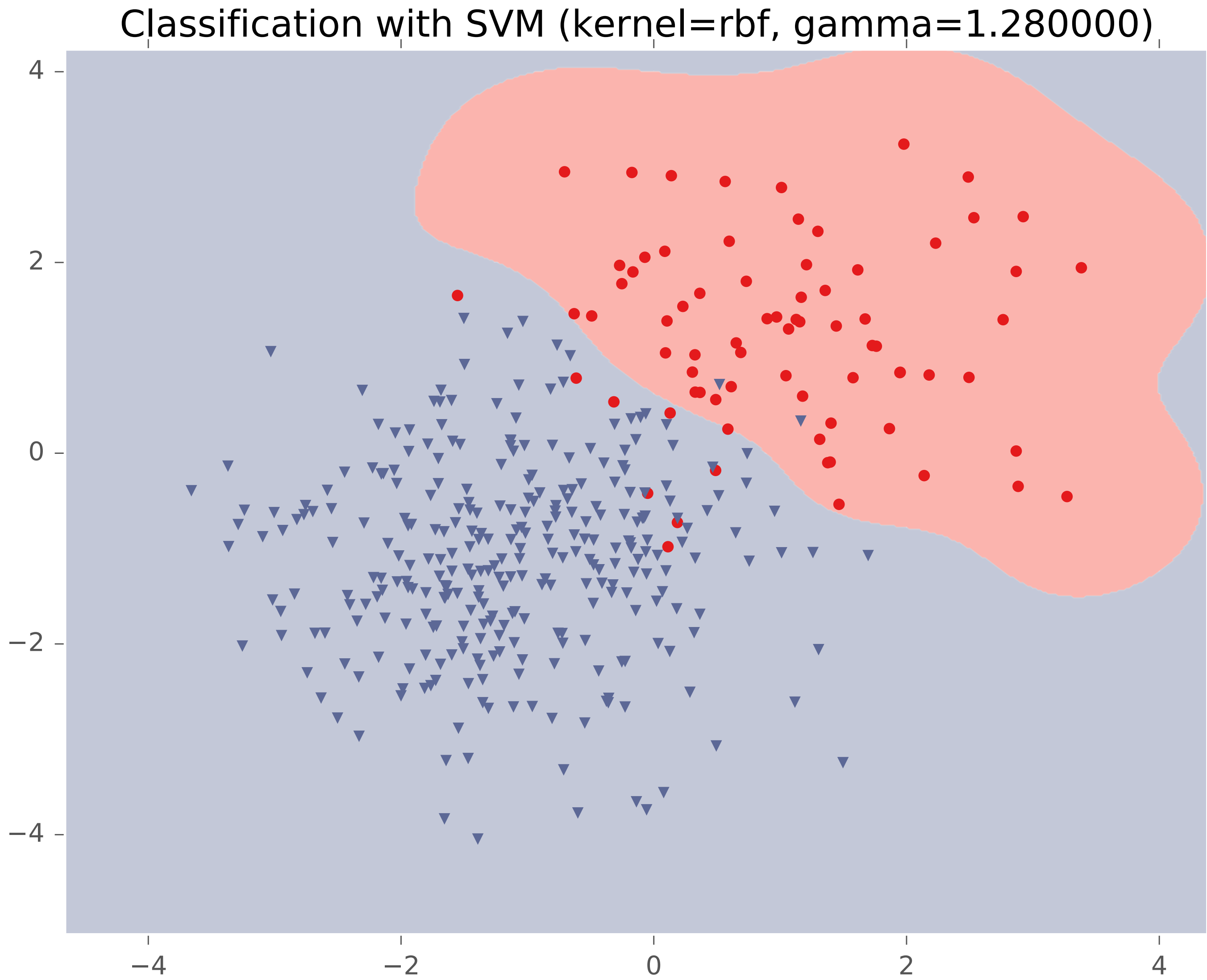
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



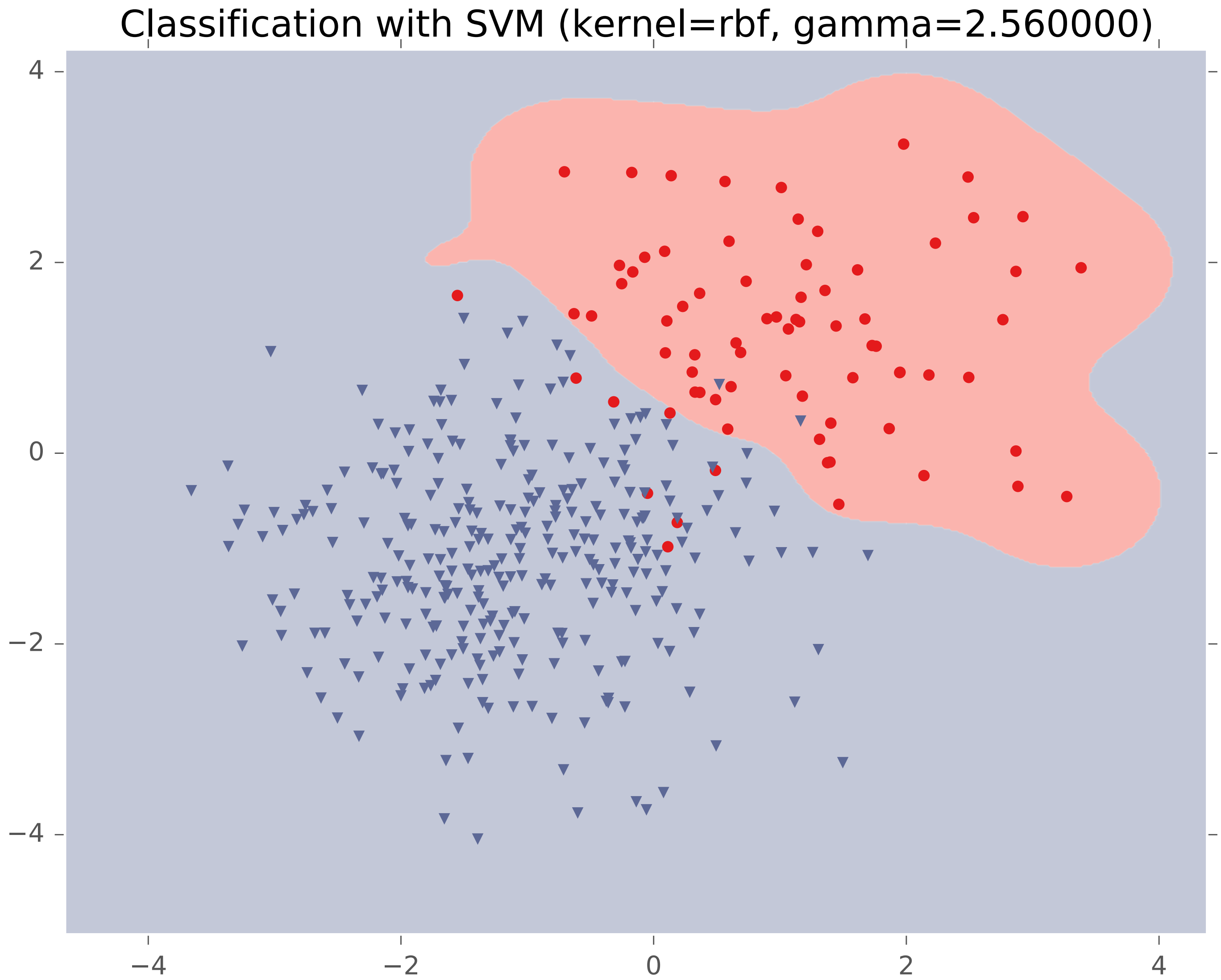
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



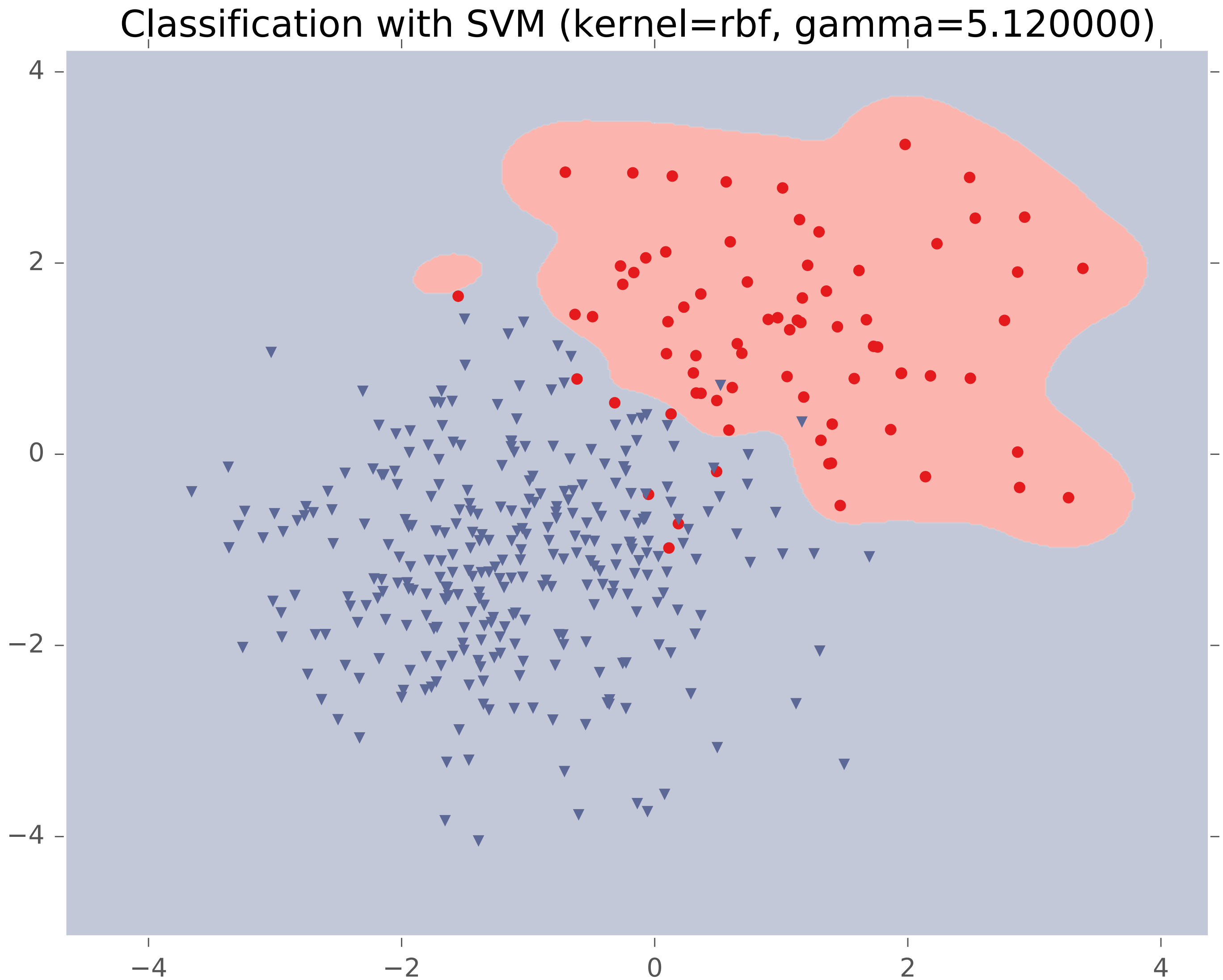
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



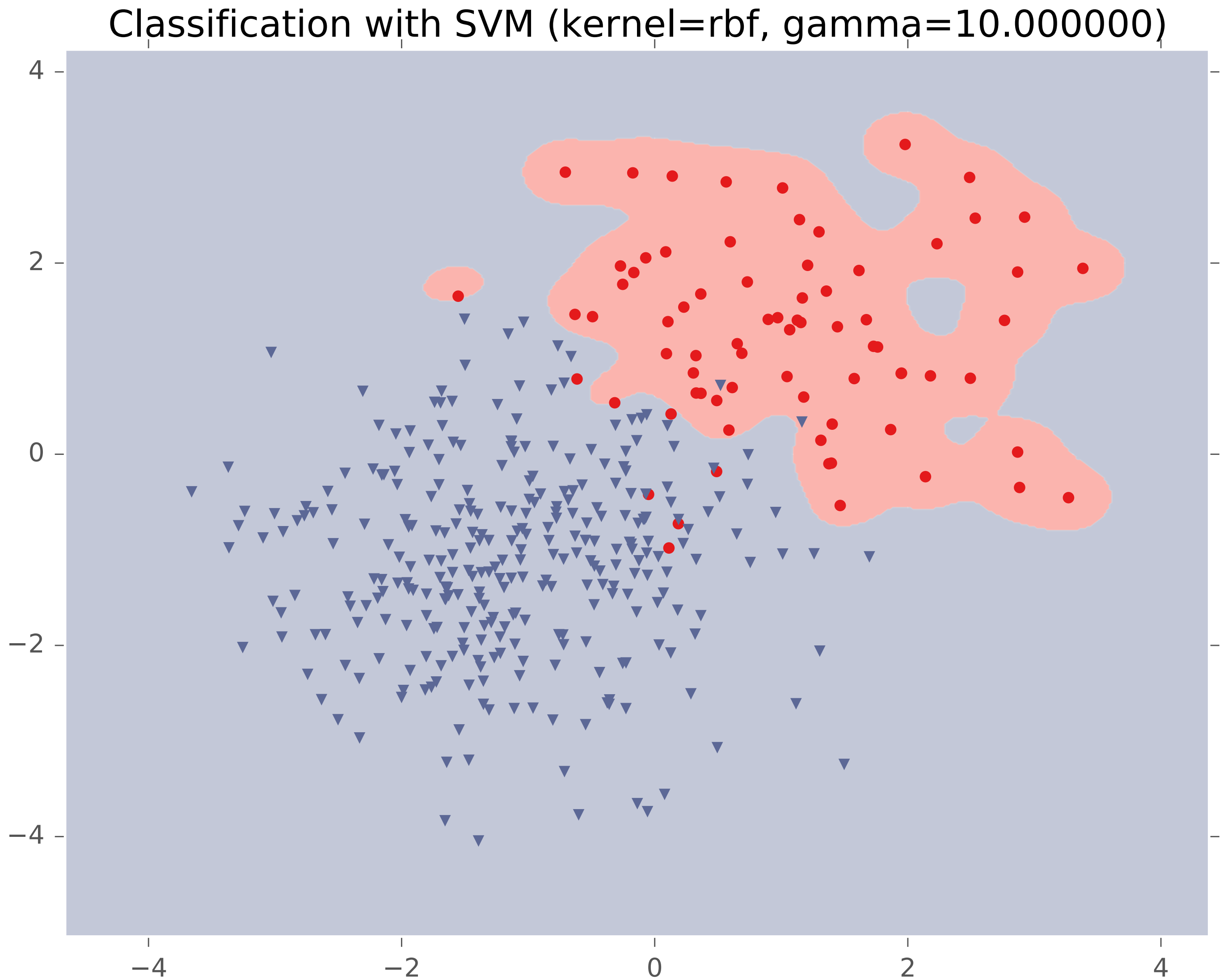
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



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

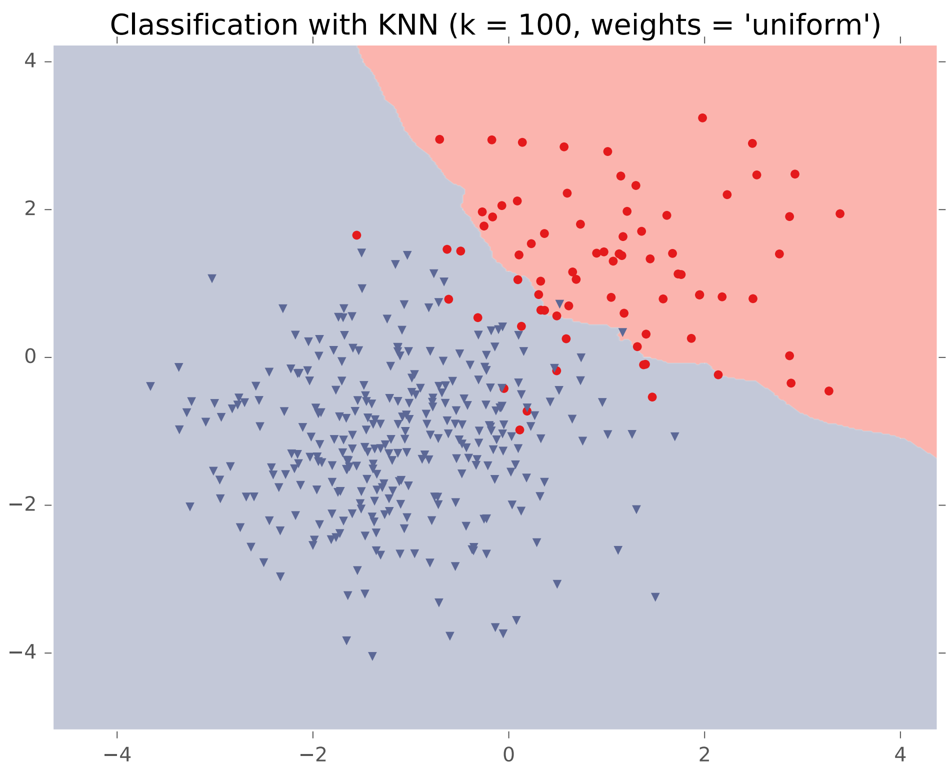
RBF Kernel Example

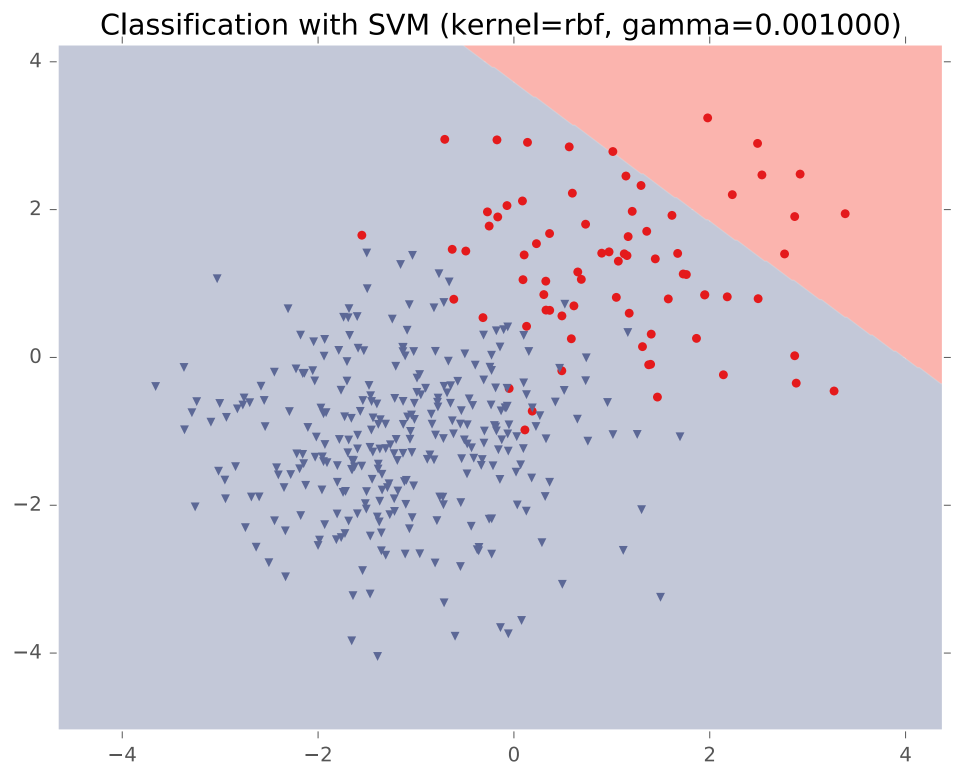


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM

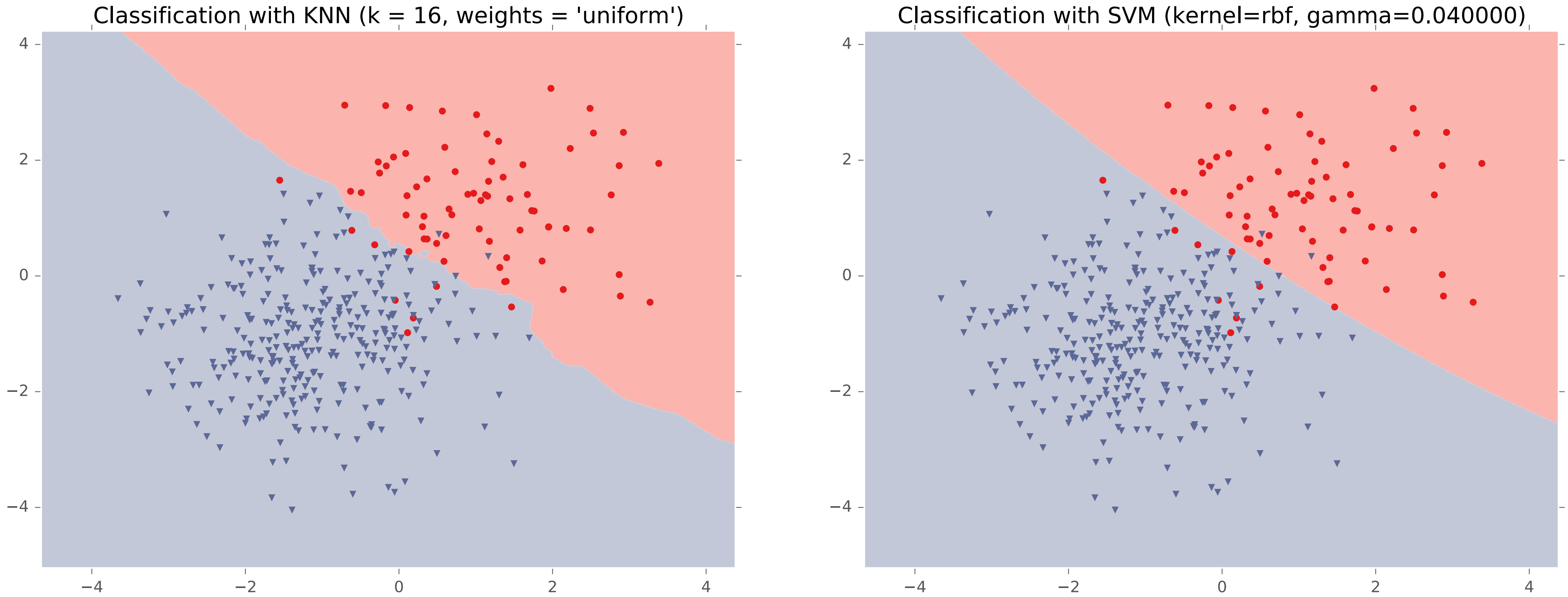




RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

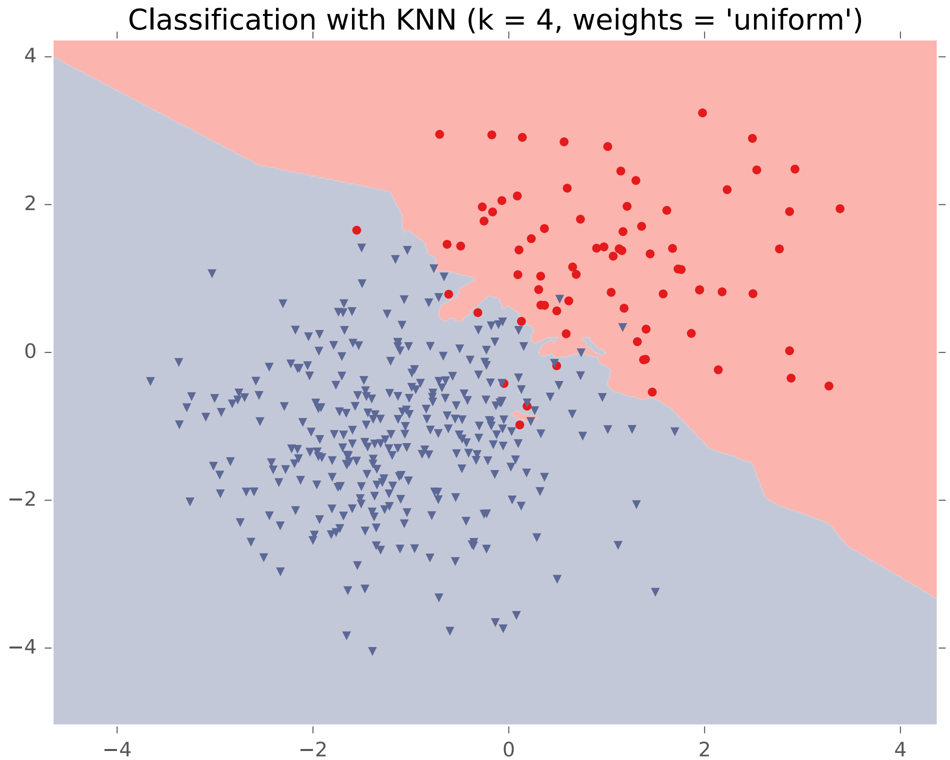
KNN vs. SVM

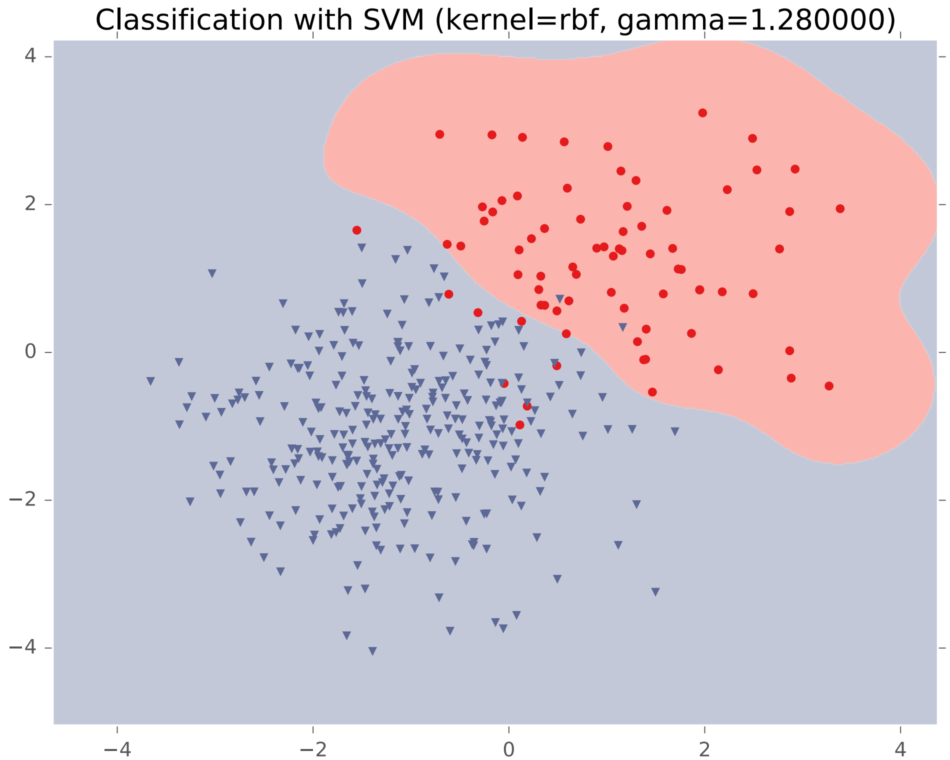


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM

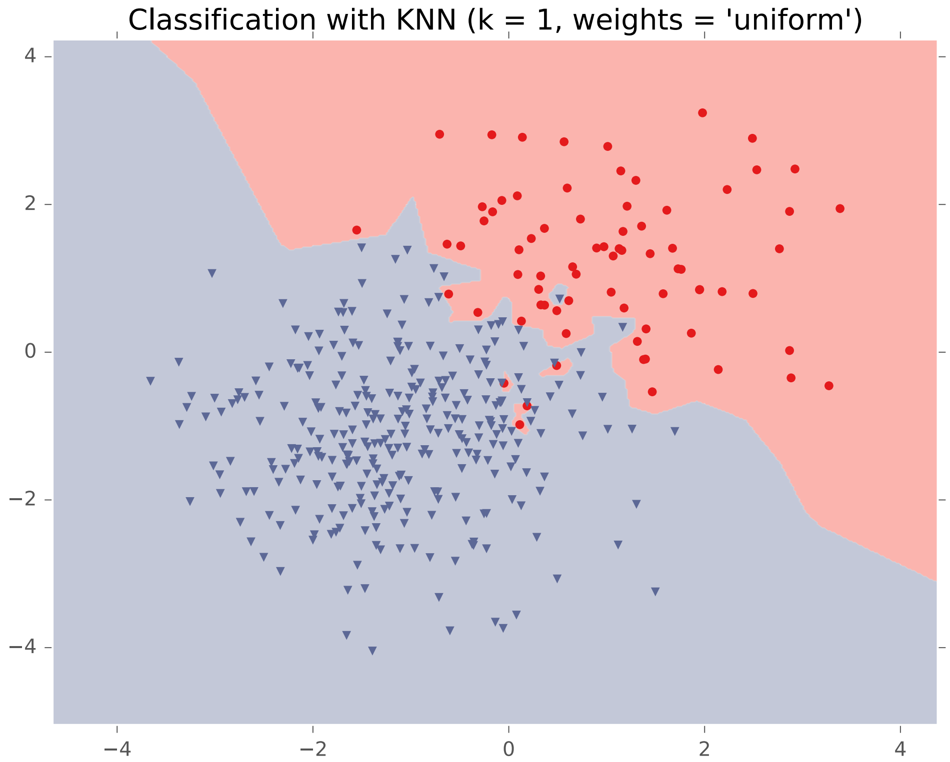


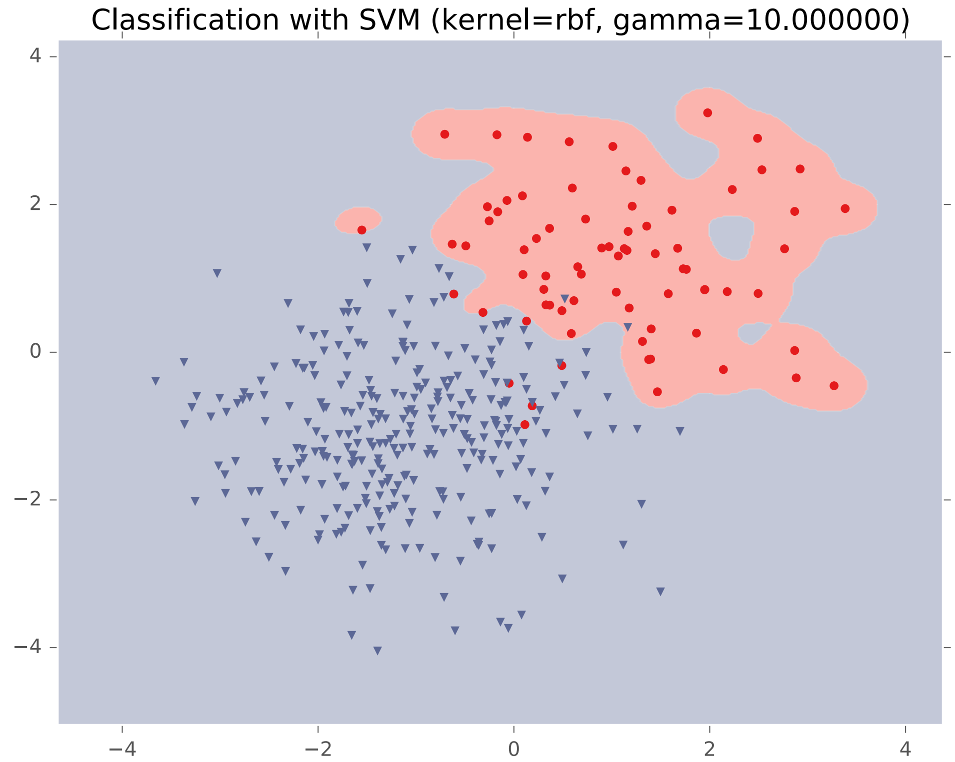


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM





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

Φ

Kernel Methods

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

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

  $k(x, z) = \varphi(x)^T\varphi(z)$ 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, …

SVM + Kernels: Takeaways

- Maximizing the margin of a linear separator is a **good training criteria**
- Support Vector Machines (SVMs) learn a **max-margin linear classifier**
- The SVM optimization problem can be solved with **black-box Quadratic Programming (QP) solvers**
- Learned decision boundary is defined by its **support vectors**
- Kernel methods allow us to work in a transformed feature space **without explicitly representing that space**
- The **kernel-trick** can be applied to **SVMs,** as well as many other algorithms

Learning Objectives

Kernels

You should be able to…

1. Employ the kernel trick in common learning algorithms
2. Explain why the use of a kernel produces only an implicit representation of the transformed feature space
3. Use the "kernel trick" to obtain a computational complexity advantage over explicit feature transformation
4. Sketch the decision boundaries of a linear classifier with an RBF kernel