10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Kernels

+

K-Means

Matt Gormley
Lecture 29
April 25, 2018

Reminders

- Homework 8: Reinforcement Learning
  - Out: Tue, Apr 17
  - Due: Fri, Apr 27 at 11:59pm
- Homework 9: Learning Paradigms
  - Out: Fri, Apr 27
  - Due: Fri, May 4 at 11:59pm

SVM

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 $\alpha^{(i)} \neq 0$

Not Covered

SVM EXTENSIONS

Soft-Margin SVM

Not Covered

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 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

Not Covered

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

Not Covered

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

Not Covered

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:

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

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

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

- Suppose we do some feature engineering
- Our feature function is ϕ
- We apply ϕ to each input vector $\mathbf{x}$

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

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

Hard-margin SVM (Lagrangian Dual)

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

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

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

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

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

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

SVM: Kernel Trick

Hard-margin SVM (Lagrangian Dual)

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

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

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

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)

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

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

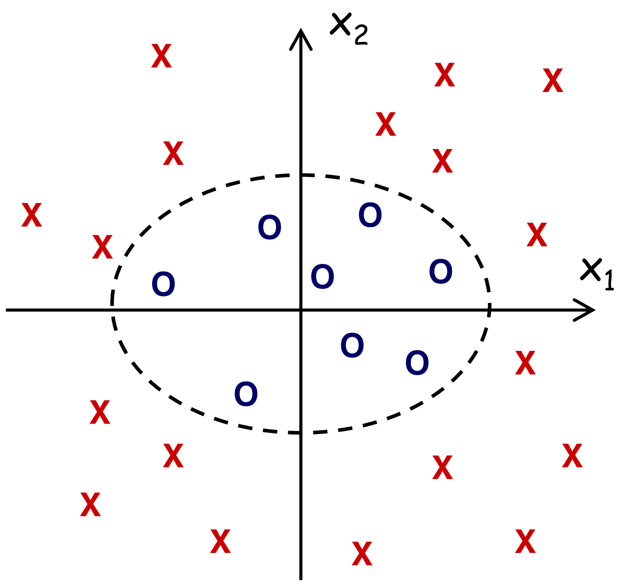
For n=2, d=2, the kernel $K(x, z) = (x \cdot z)^d$ corresponds to

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

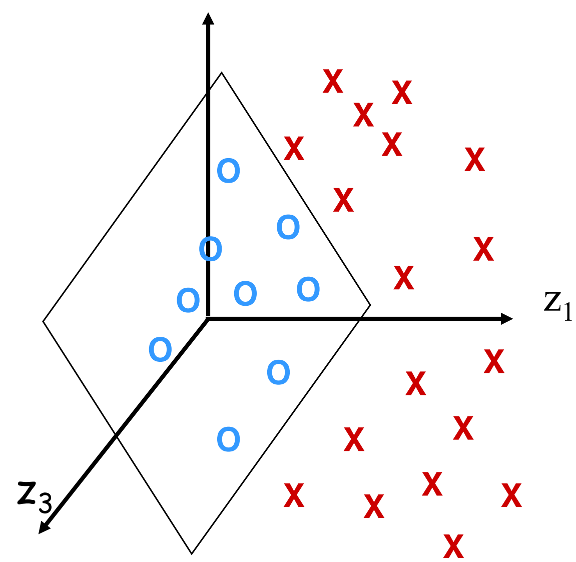
$$\phi(x) \cdot \phi(z) = \left(x_1^2, x_2^2, \sqrt{2} x_1 x_2\right) \cdot (z_1^2, z_2^2, \sqrt{2} z_1 z_2)$$

$$= (x_1 z_1 + x_2 z_2)^2 = (x \cdot z)^2 = K(x, z)$$

Original space



Φ-space

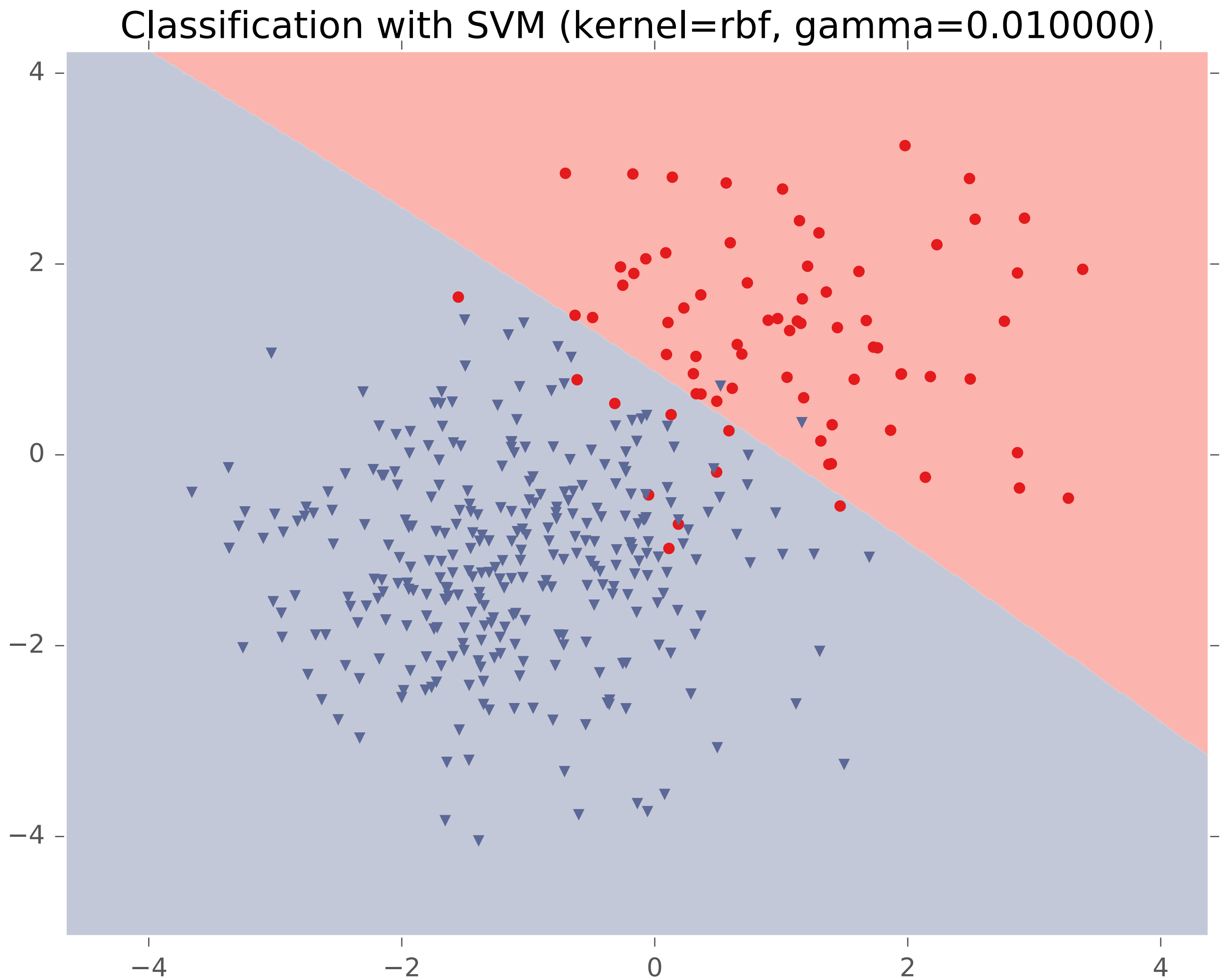


Slide from Nina Balcan

Kernel Examples

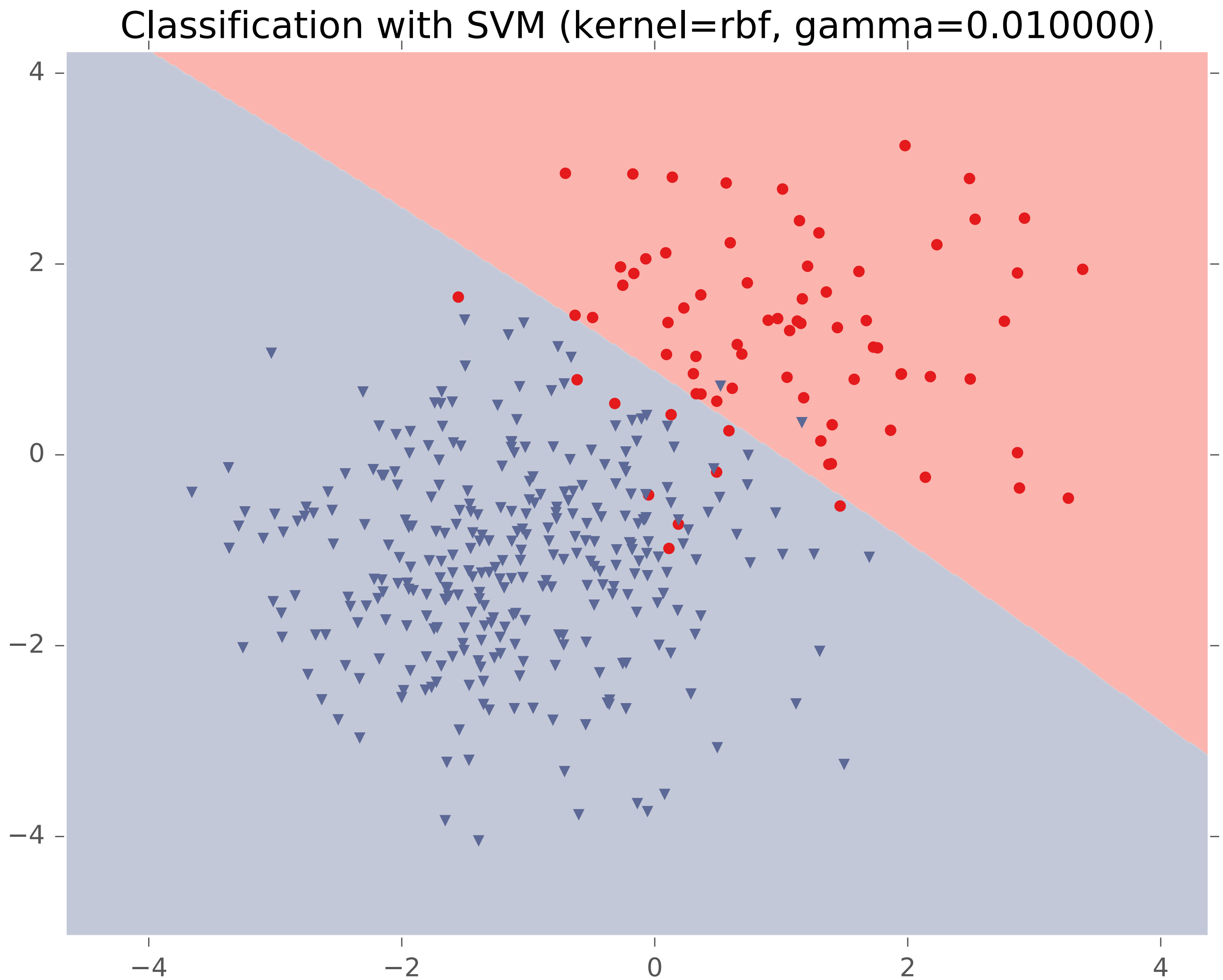
Name	Kernel Function (implicit dot product)	Feature Space (explicit dot product)
Linear	$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x},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



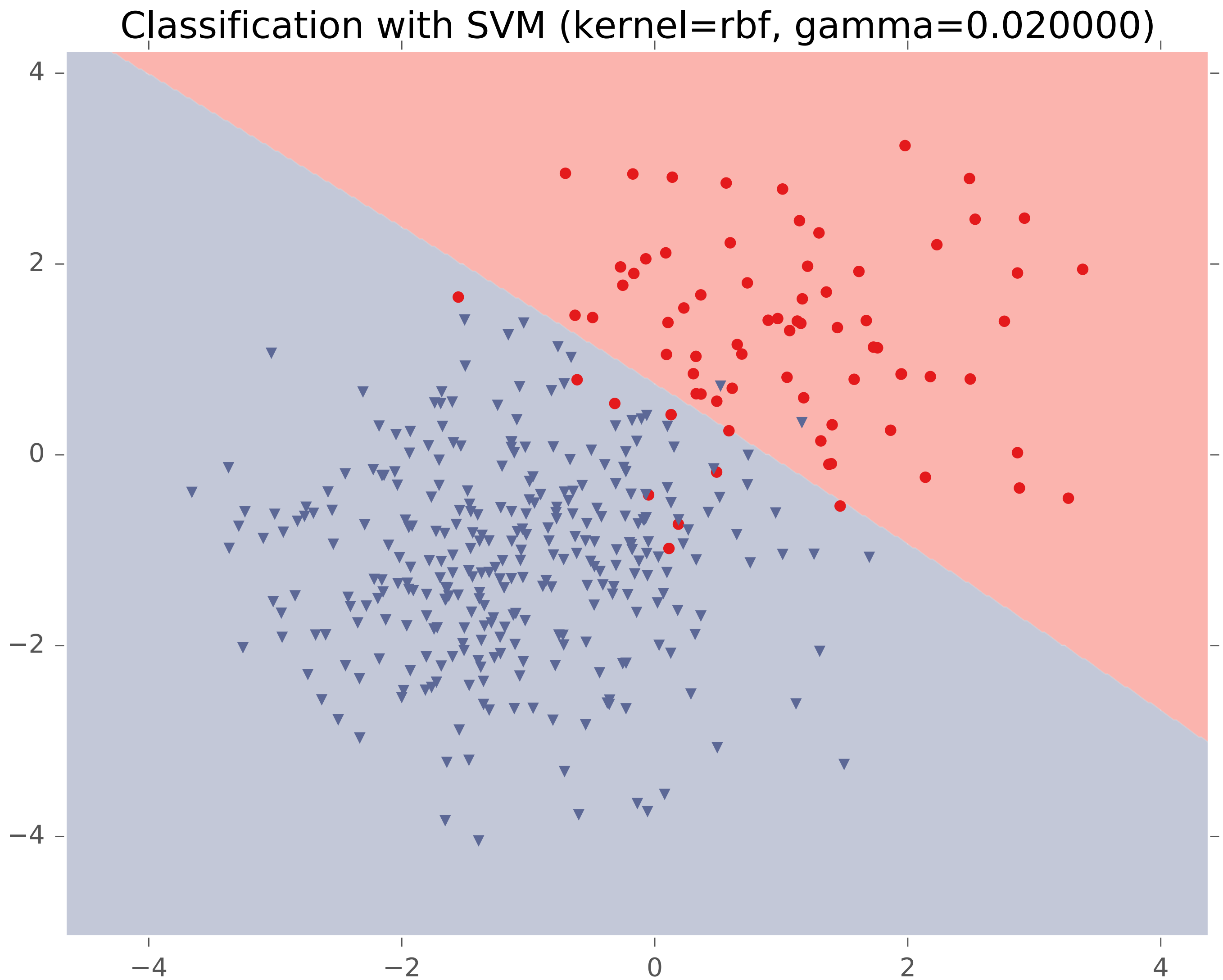
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



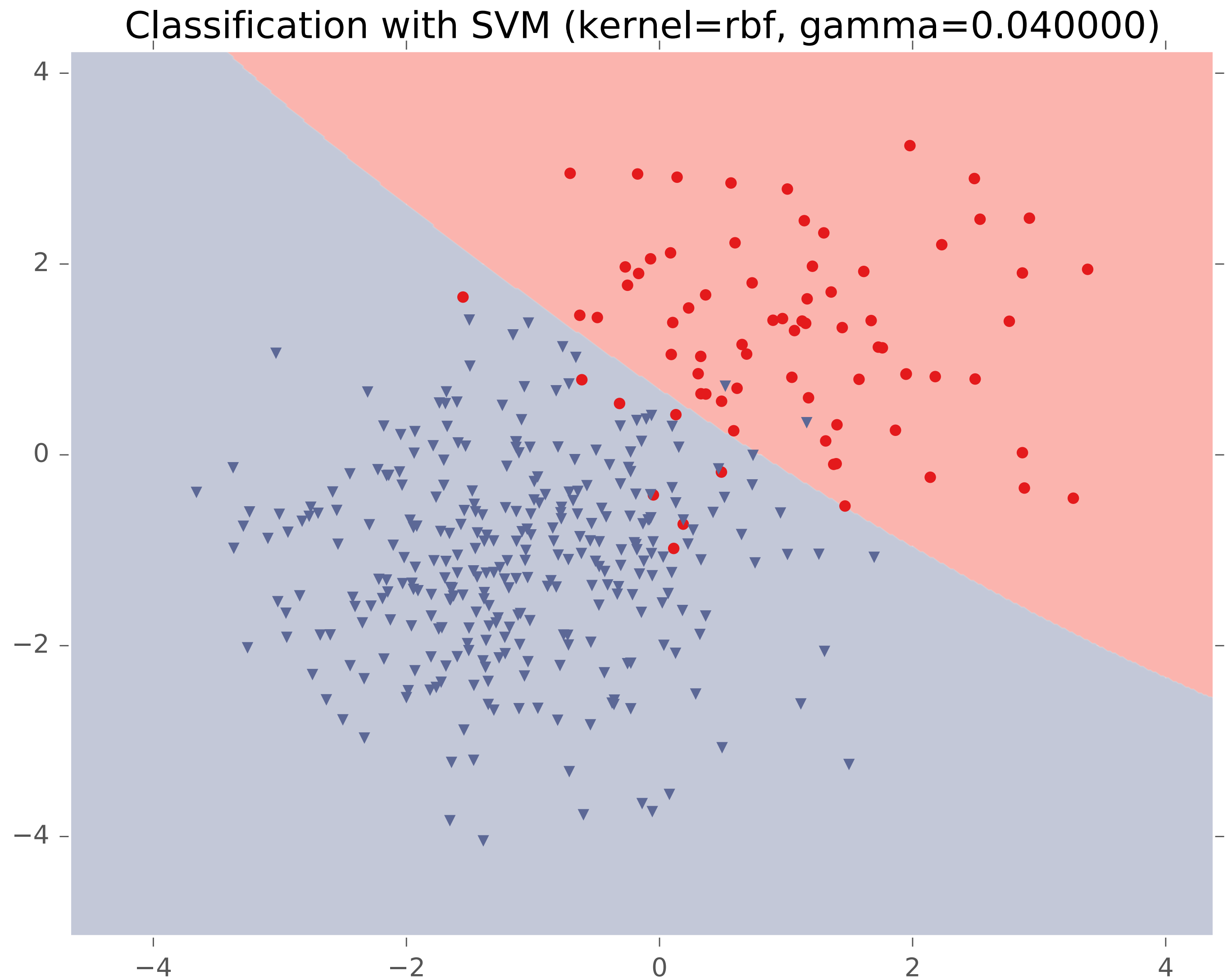
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



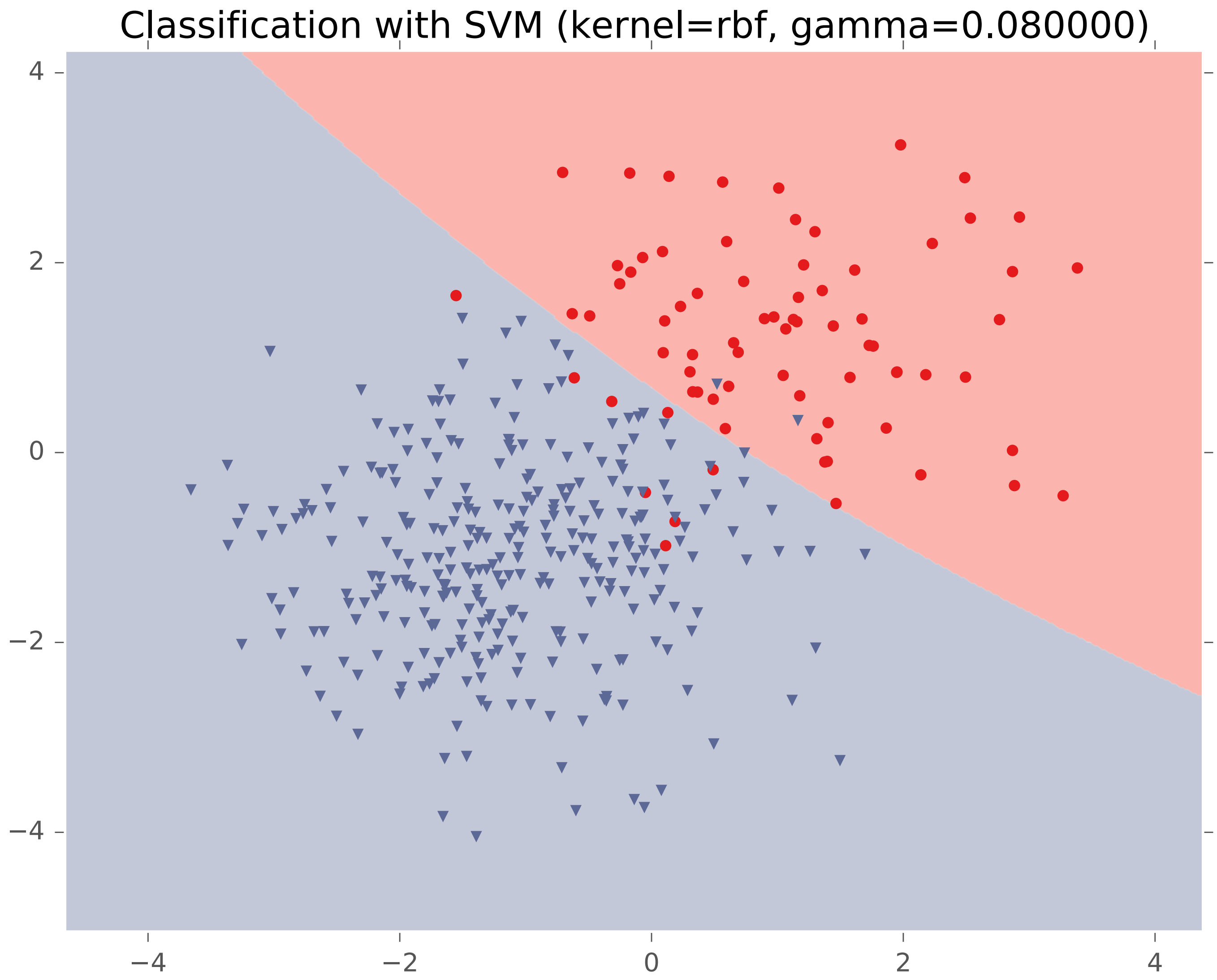
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



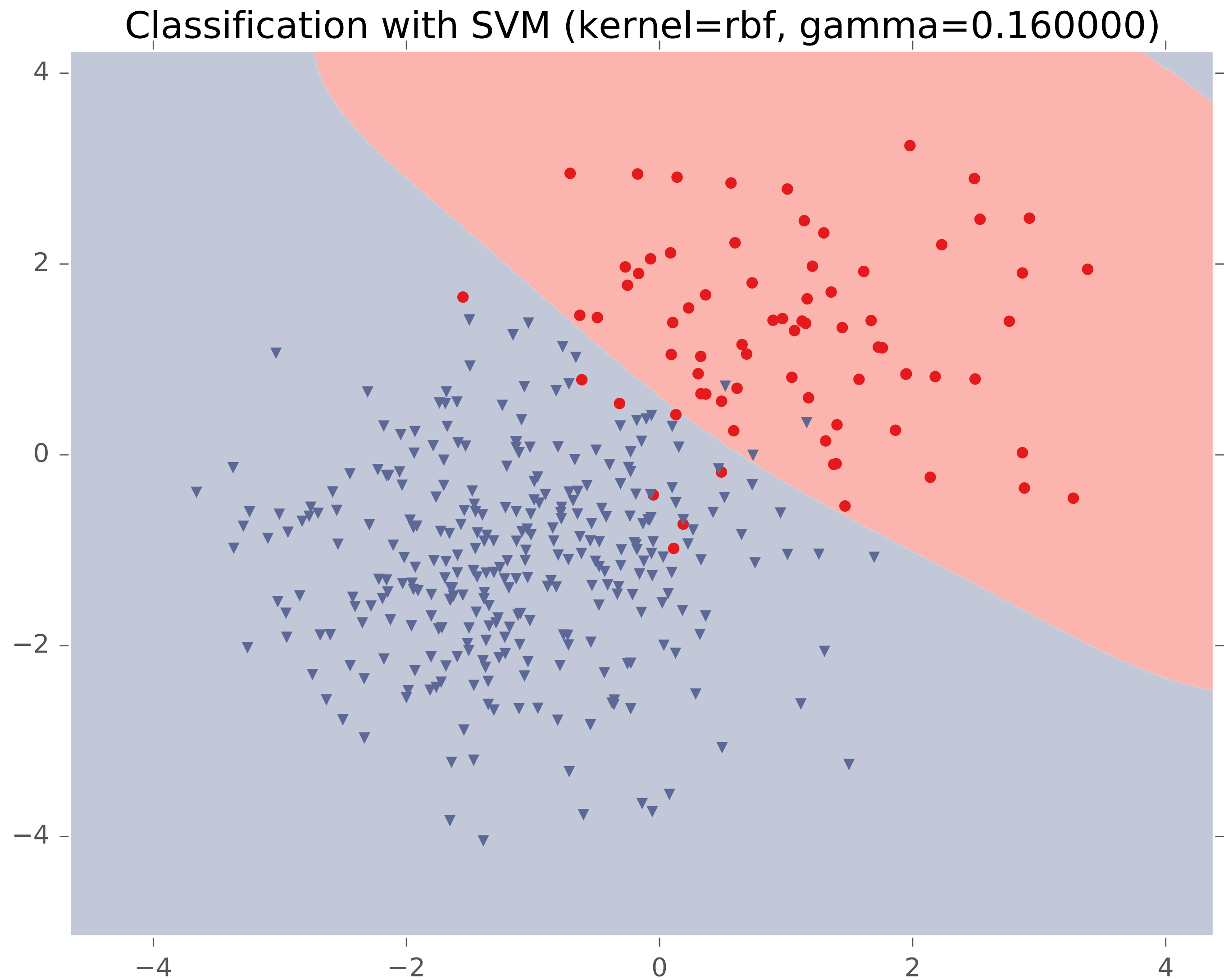
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



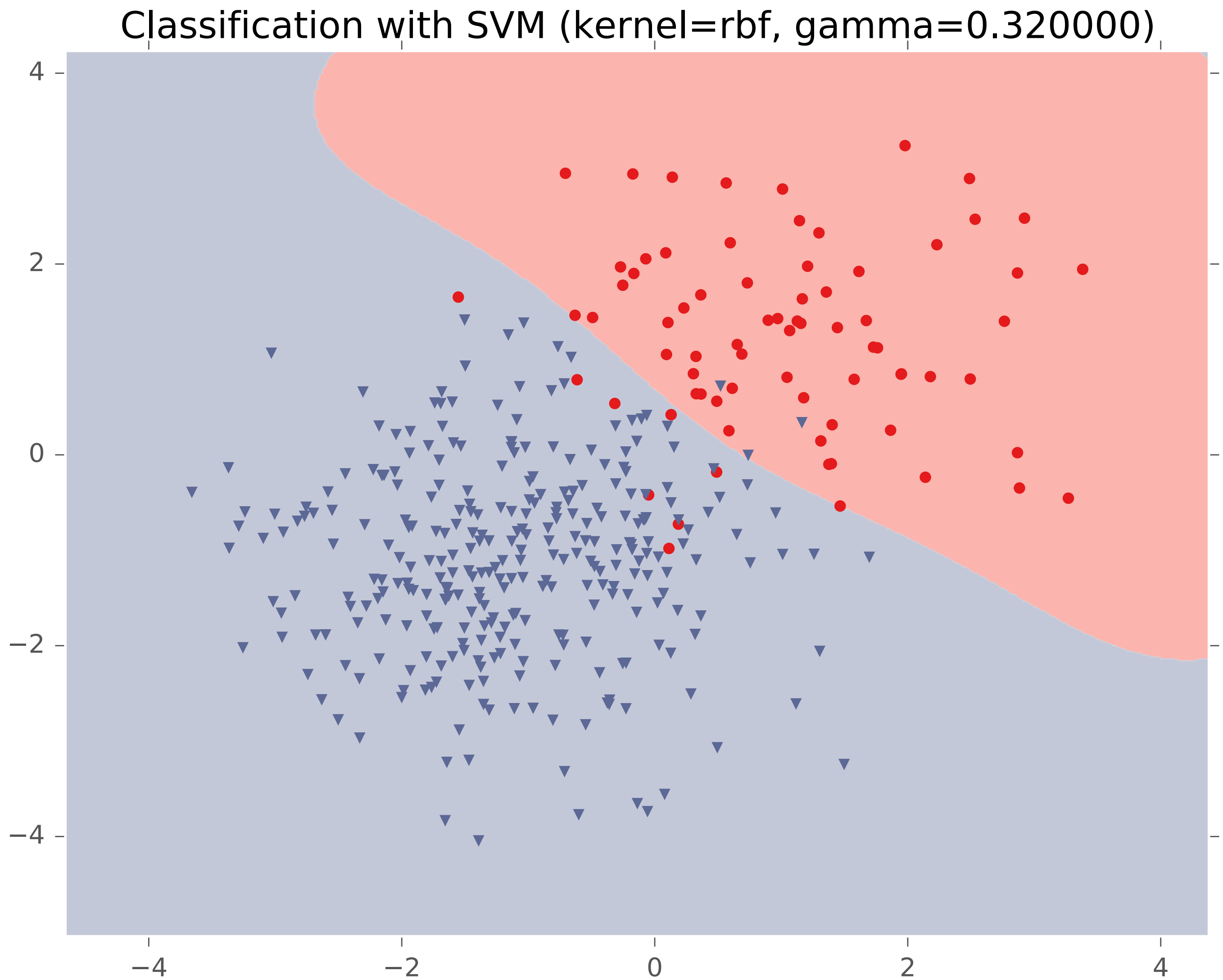
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



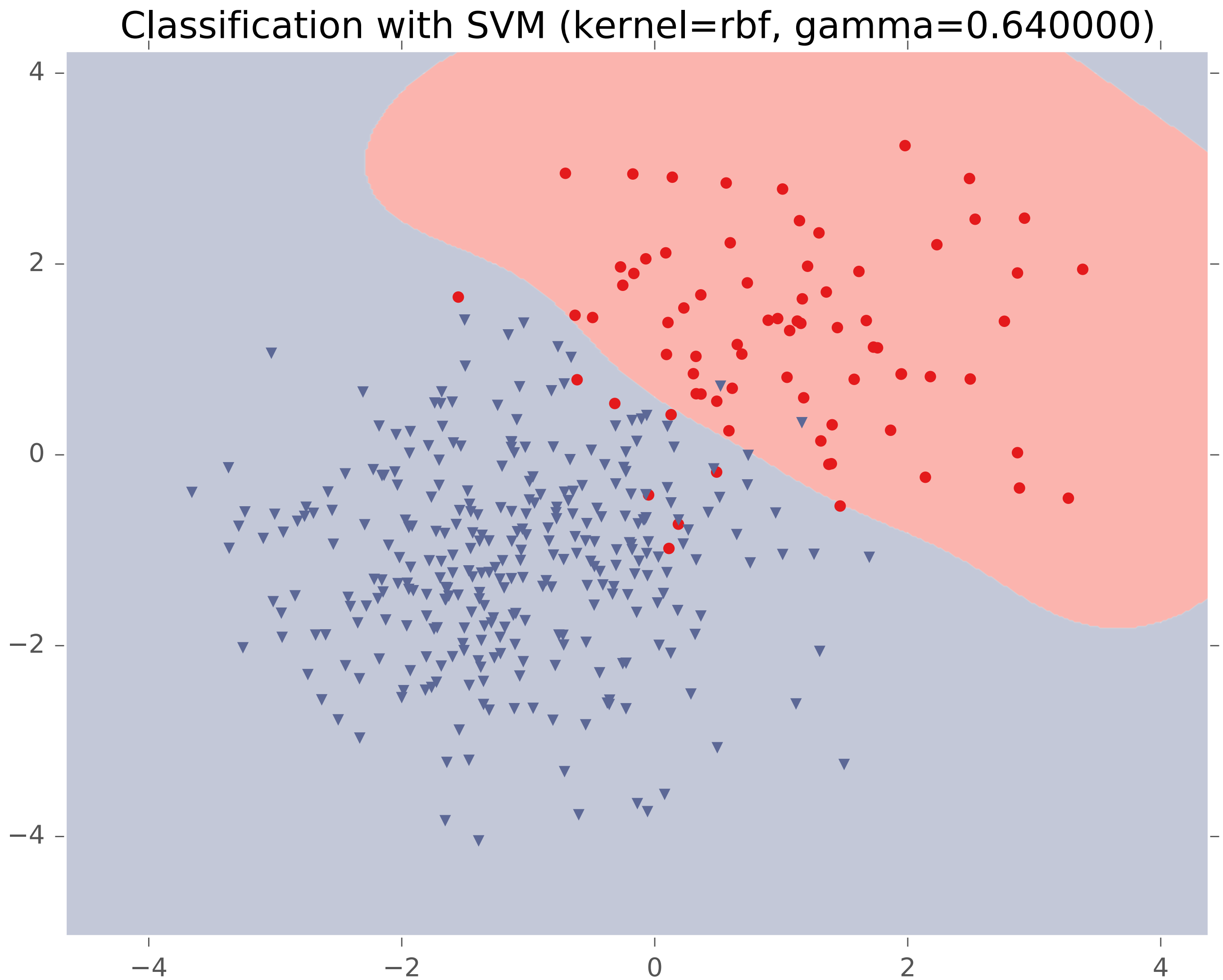
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



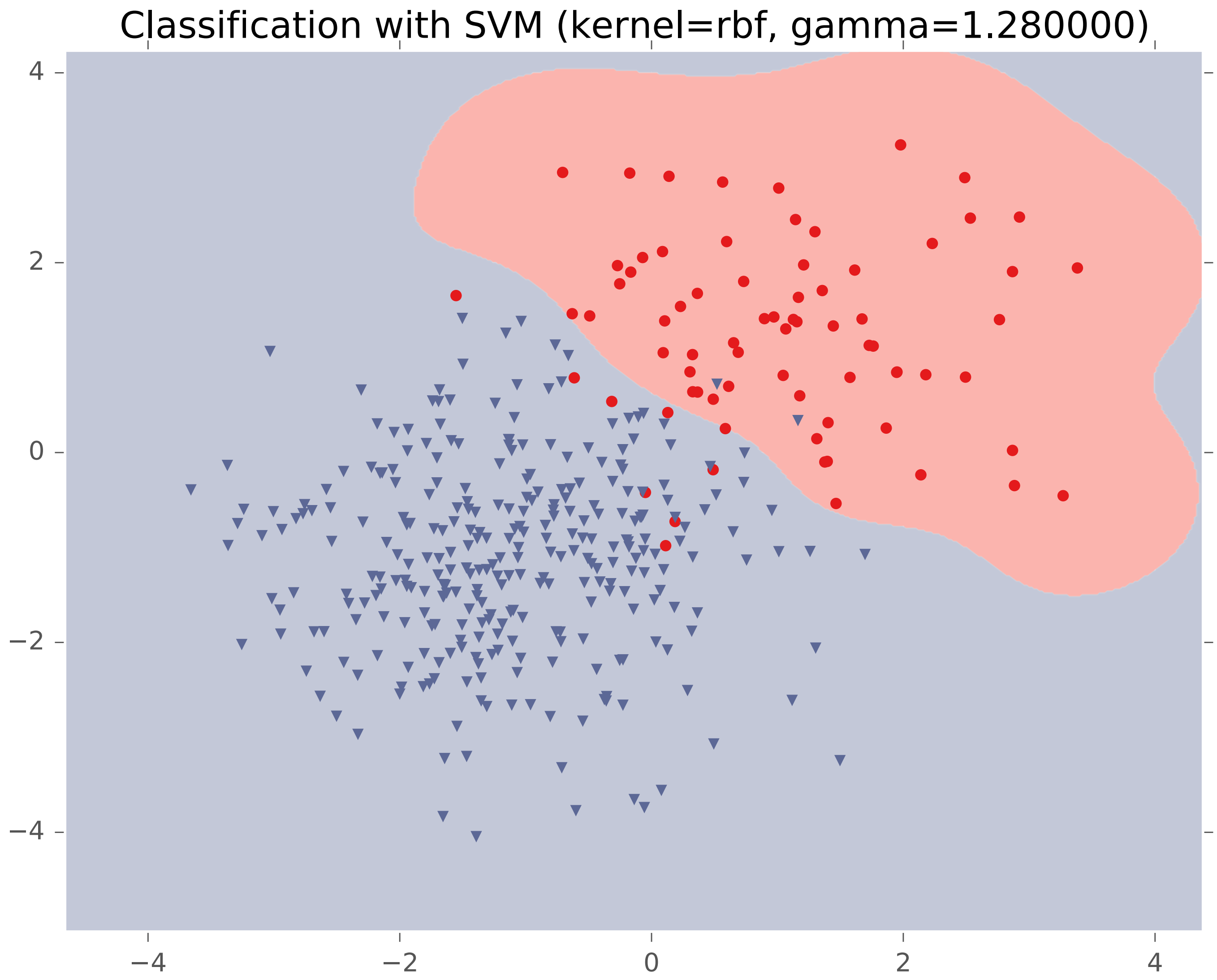
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



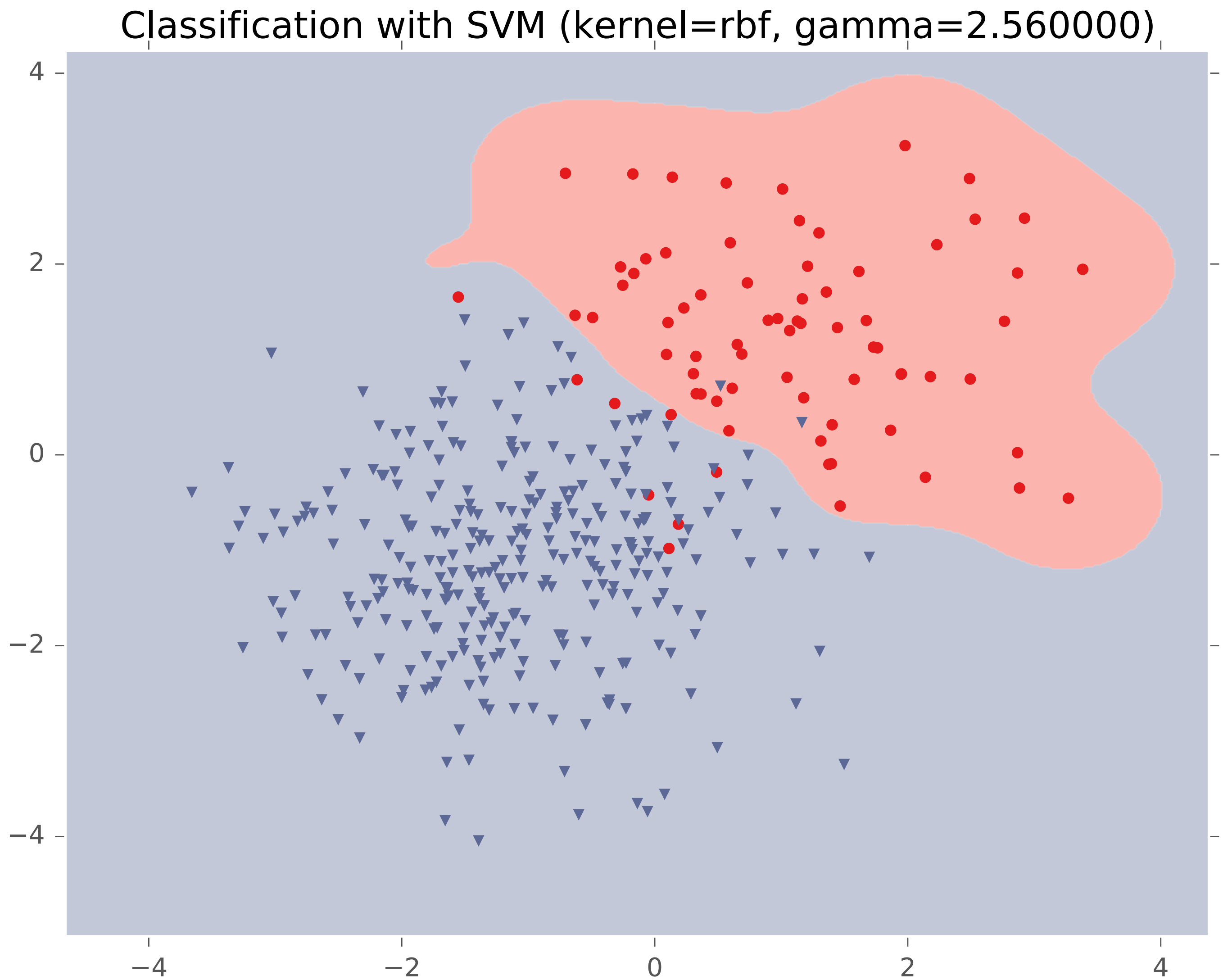
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



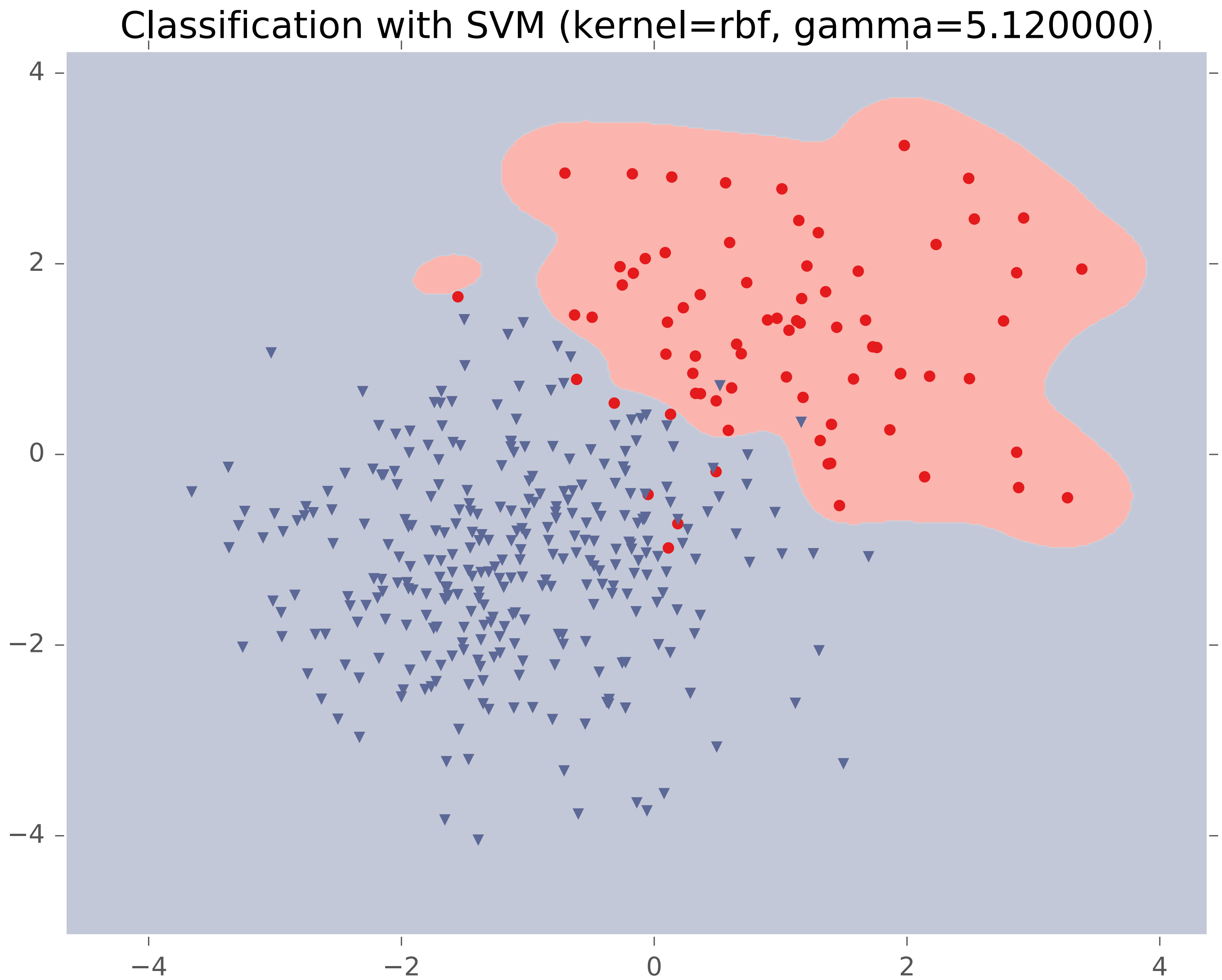
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



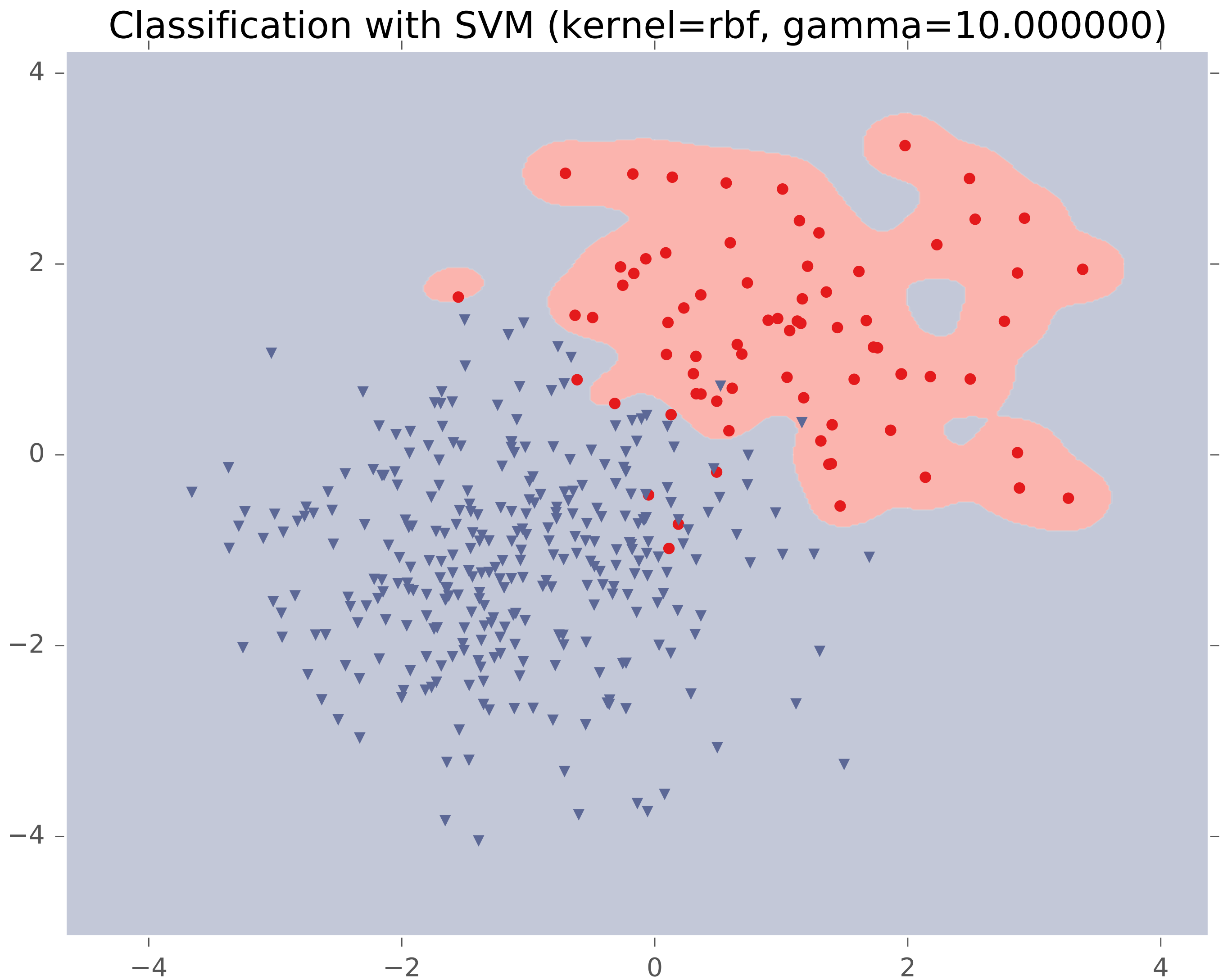
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



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

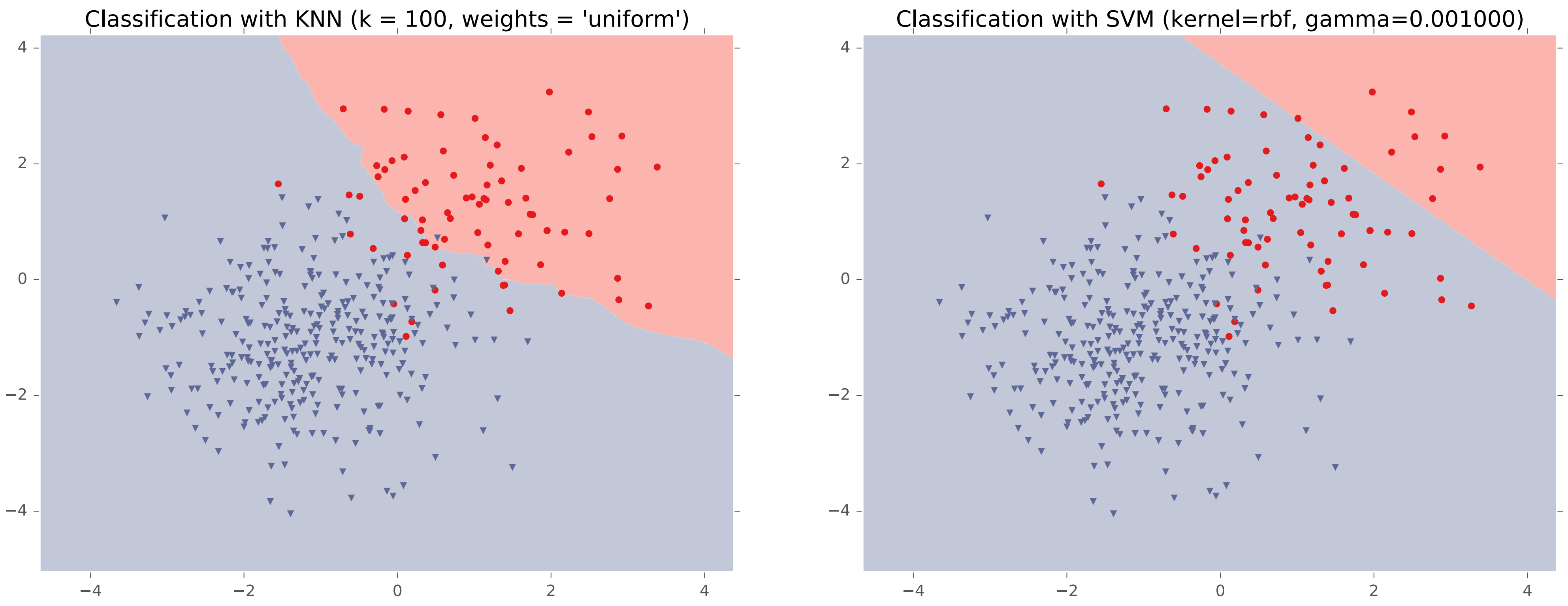
RBF Kernel Example



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

RBF Kernel Example

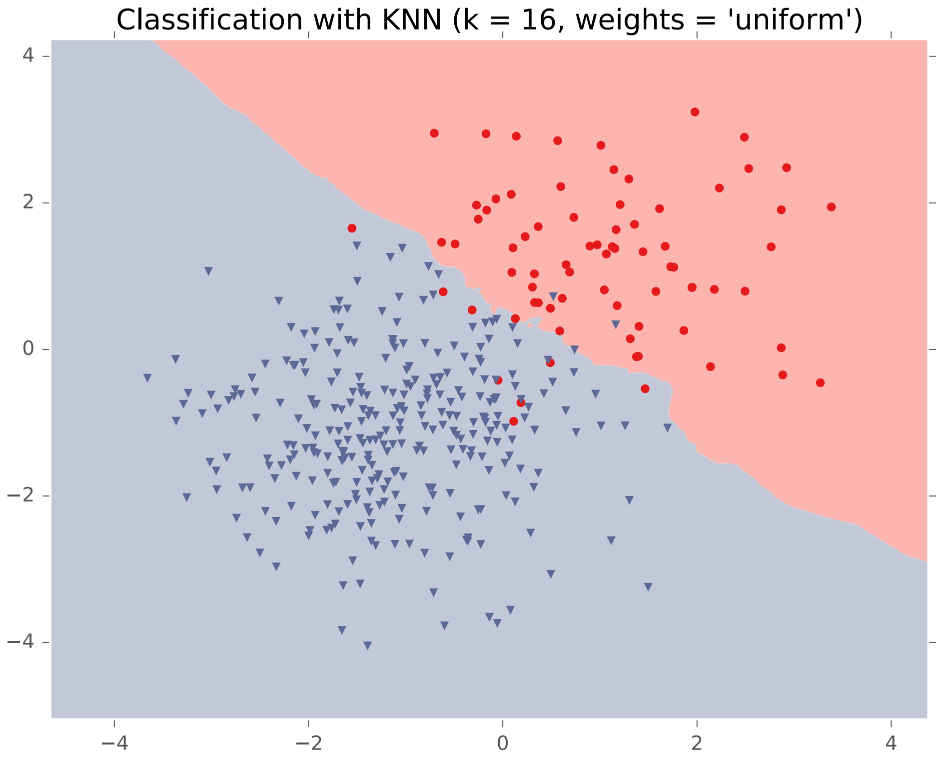
KNN vs. SVM

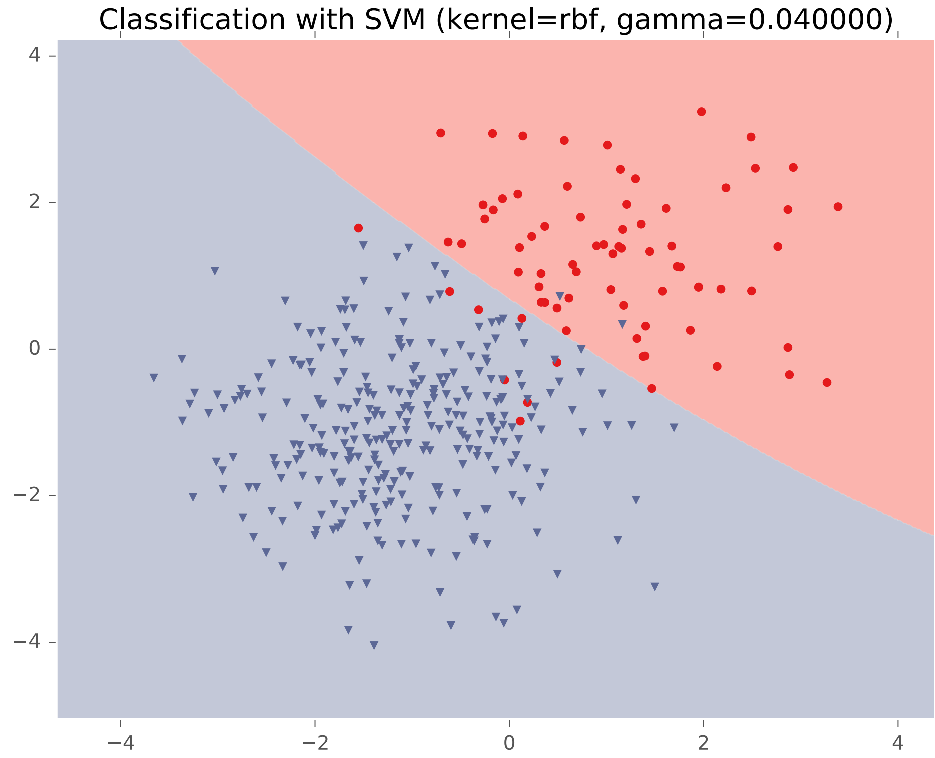


RBF Kernel: $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$$

RBF Kernel Example

KNN vs. SVM

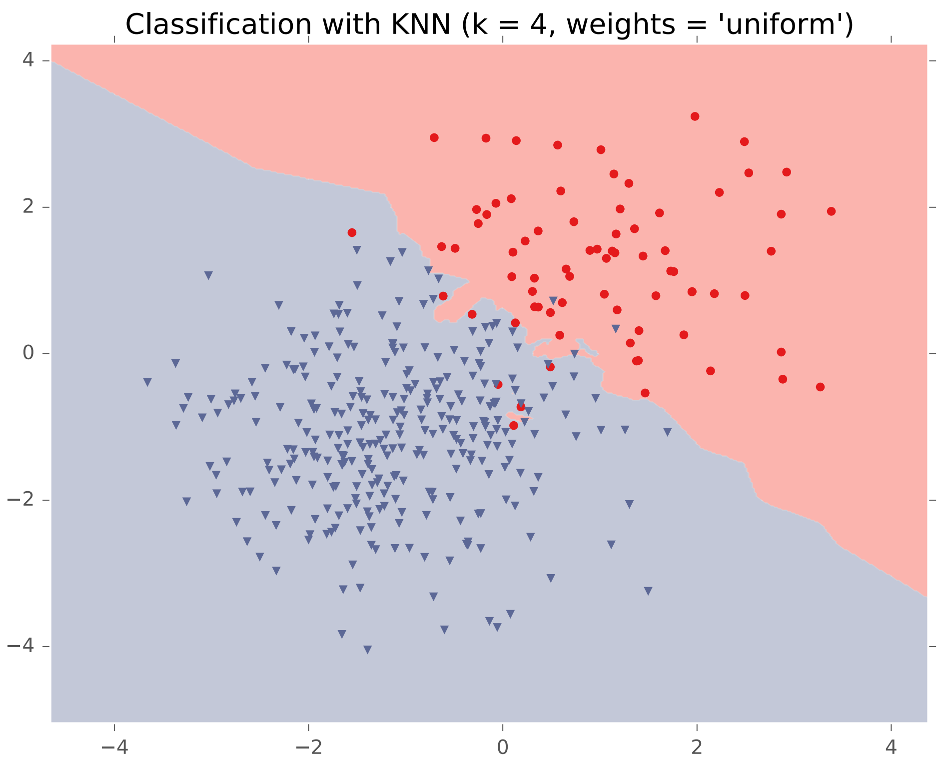


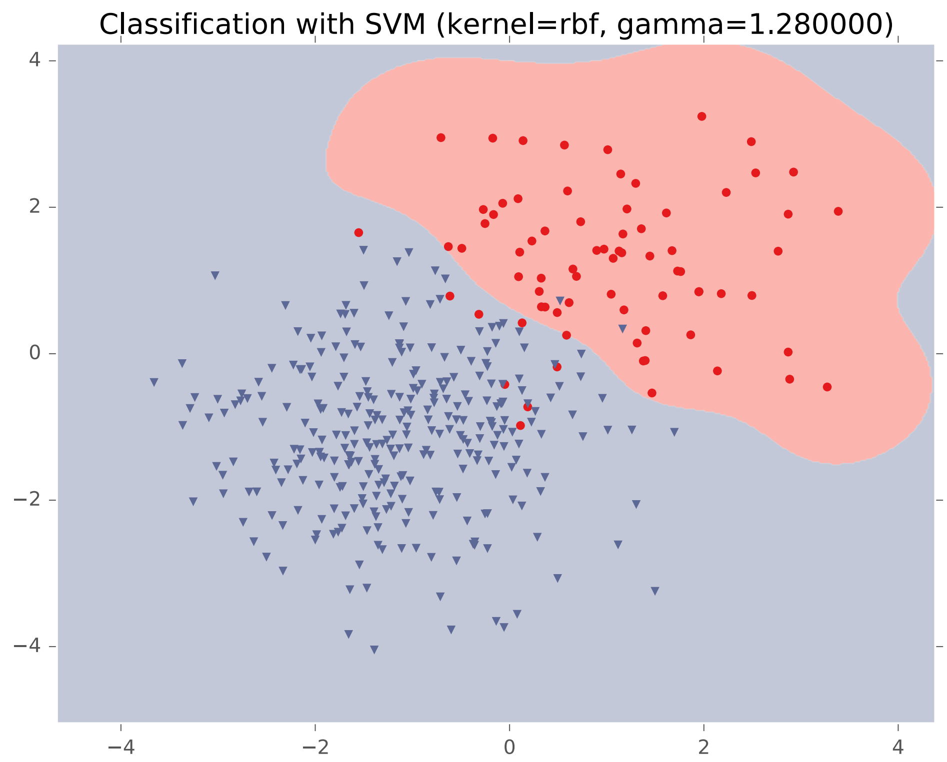


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM

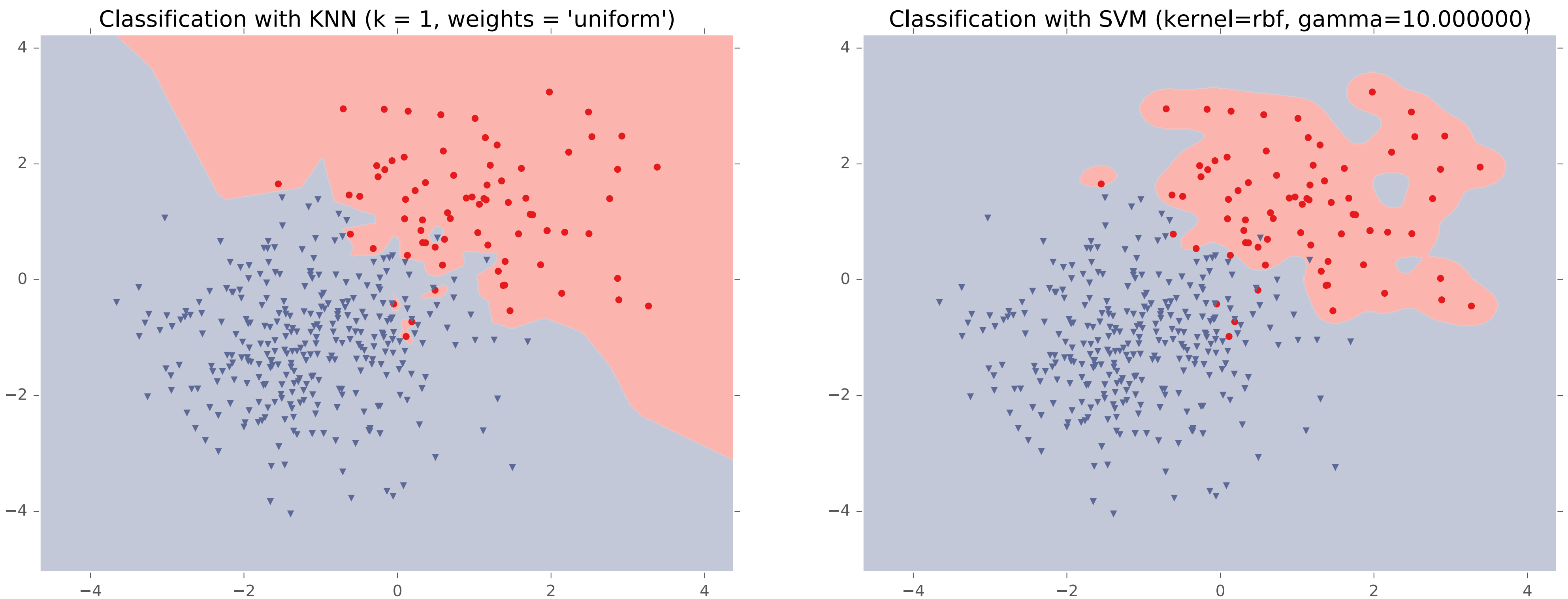




RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma ||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

KNN vs. SVM



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

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, …

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

K-MEANS

K-Means Outline

- **Clustering: Motivation / Applications**
- **Optimization Background**
  - Coordinate Descent
  - Block Coordinate Descent
- **Clustering**
  - Inputs and Outputs
  - Objective-based Clustering
- **K-Means**
  - K-Means Objective
  - Computational Complexity
  - K-Means Algorithm / Lloyd's Method
- **K-Means Initialization**
  - Random
  - Farthest Point
  - K-Means++

Clustering, Informal Goals

Goal: Automatically partition unlabeled data into groups of similar datapoints.

Question: When and why would we want to do this?

Useful for:

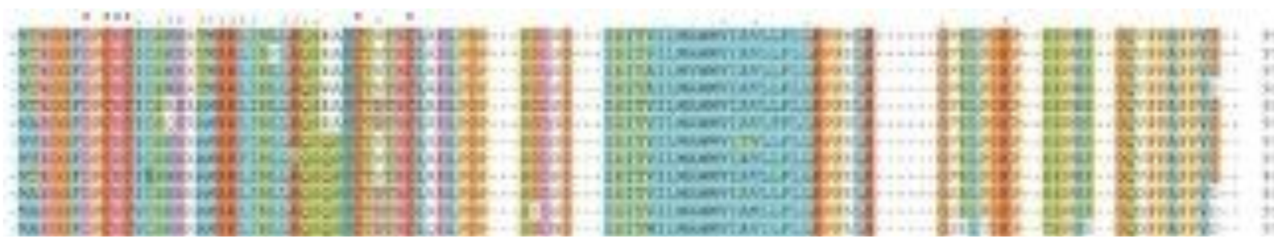
- Automatically organizing data.
- Understanding hidden structure in data.
- Preprocessing for further analysis.
  - Representing high-dimensional data in a low-dimensional space (e.g., for visualization purposes).

Applications (Clustering comes up everywhere…)

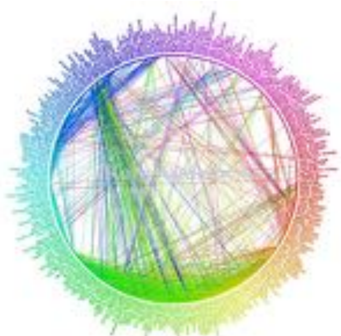
- Cluster news articles or web pages or search results by topic.

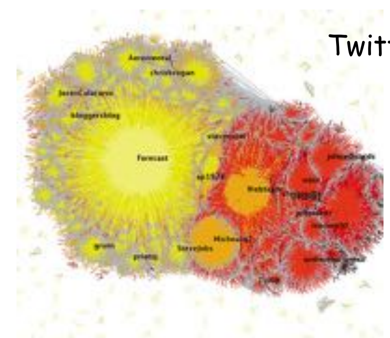


- Cluster protein sequences by function or genes according to expression profile.

- Cluster users of social networks by interest (community detection).


Facebook network


Twitter Network

Slide courtesy of Nina Balcan

Applications (Clustering comes up everywhere…)

- Cluster customers according to purchase history.

- Cluster galaxies or nearby stars (e.g. Sloan Digital Sky Survey)

- And many many more applications….

Slide courtesy of Nina Balcan

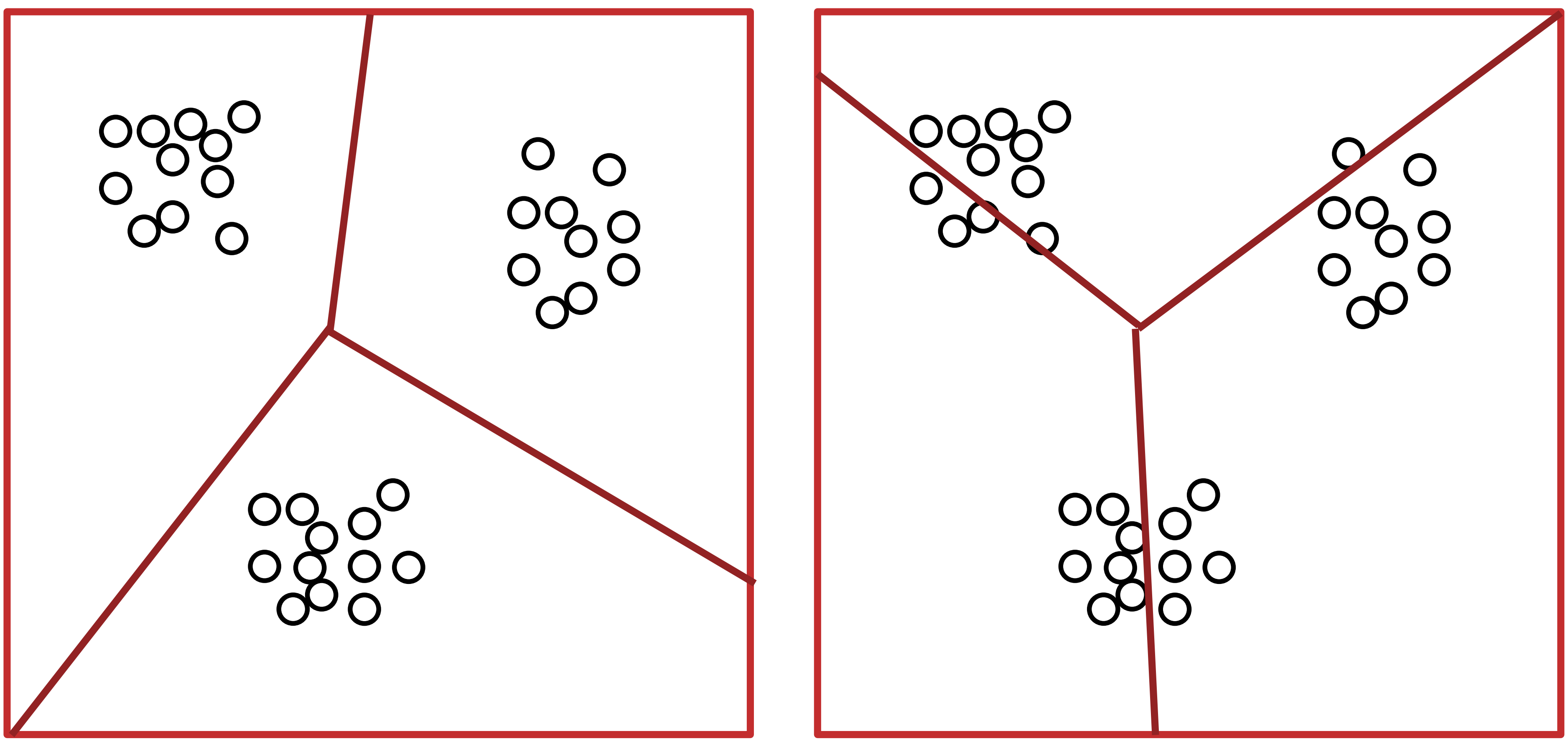
Optimization Background

Whiteboard:

- Coordinate Descent
- Block Coordinate Descent

Clustering

Question: Which of these partitions is "better"?



Clustering

Whiteboard:

- Inputs and Outputs
- Objective-based Clustering

K-Means

Whiteboard:

- K-Means Objective
- Computational Complexity
- K-Means Algorithm / Lloyd’s Method

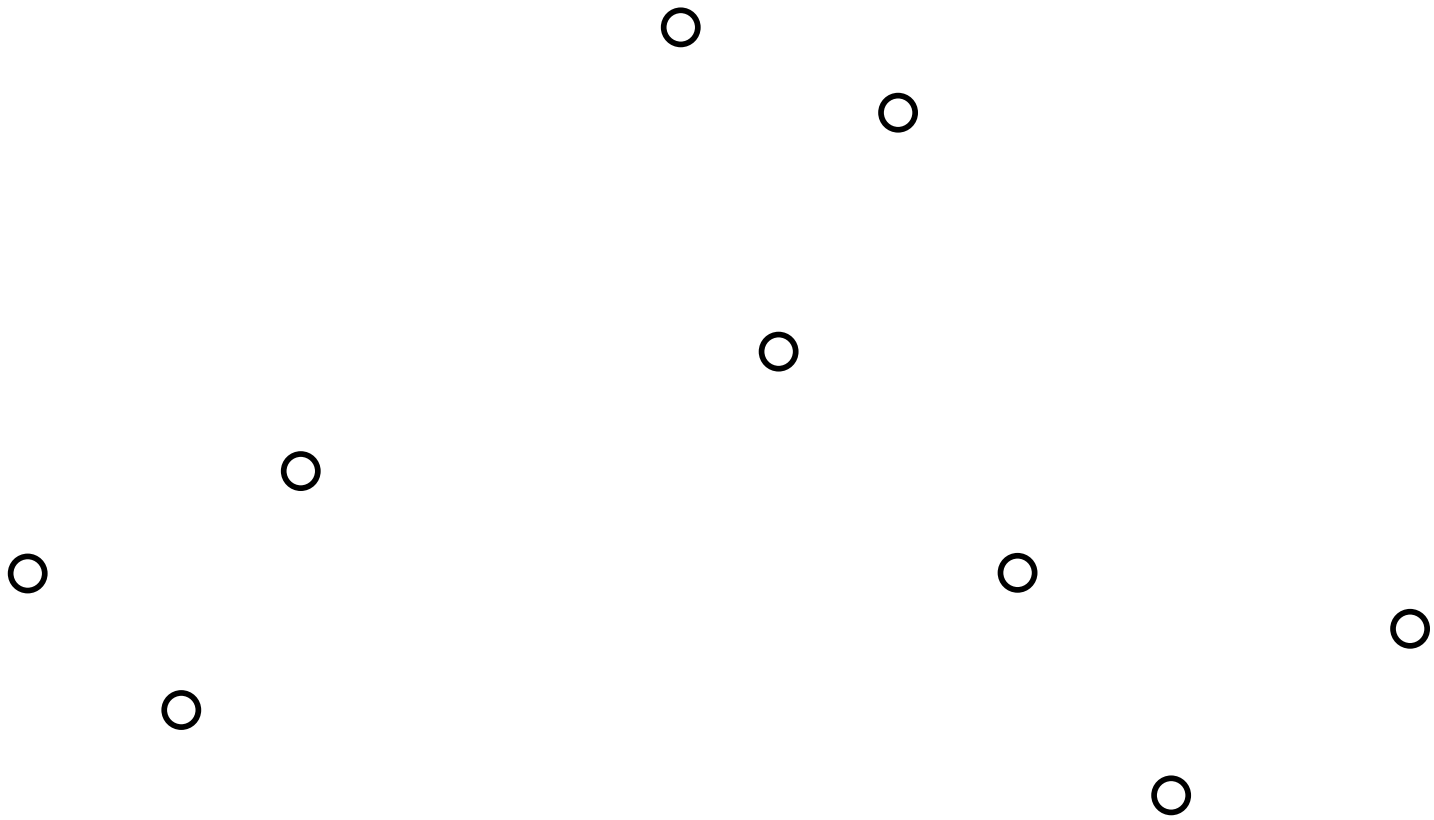
K-Means Initialization

Whiteboard:

- Random
- Furthest Traversal
- K-Means++

Lloyd's method: Random Initialization

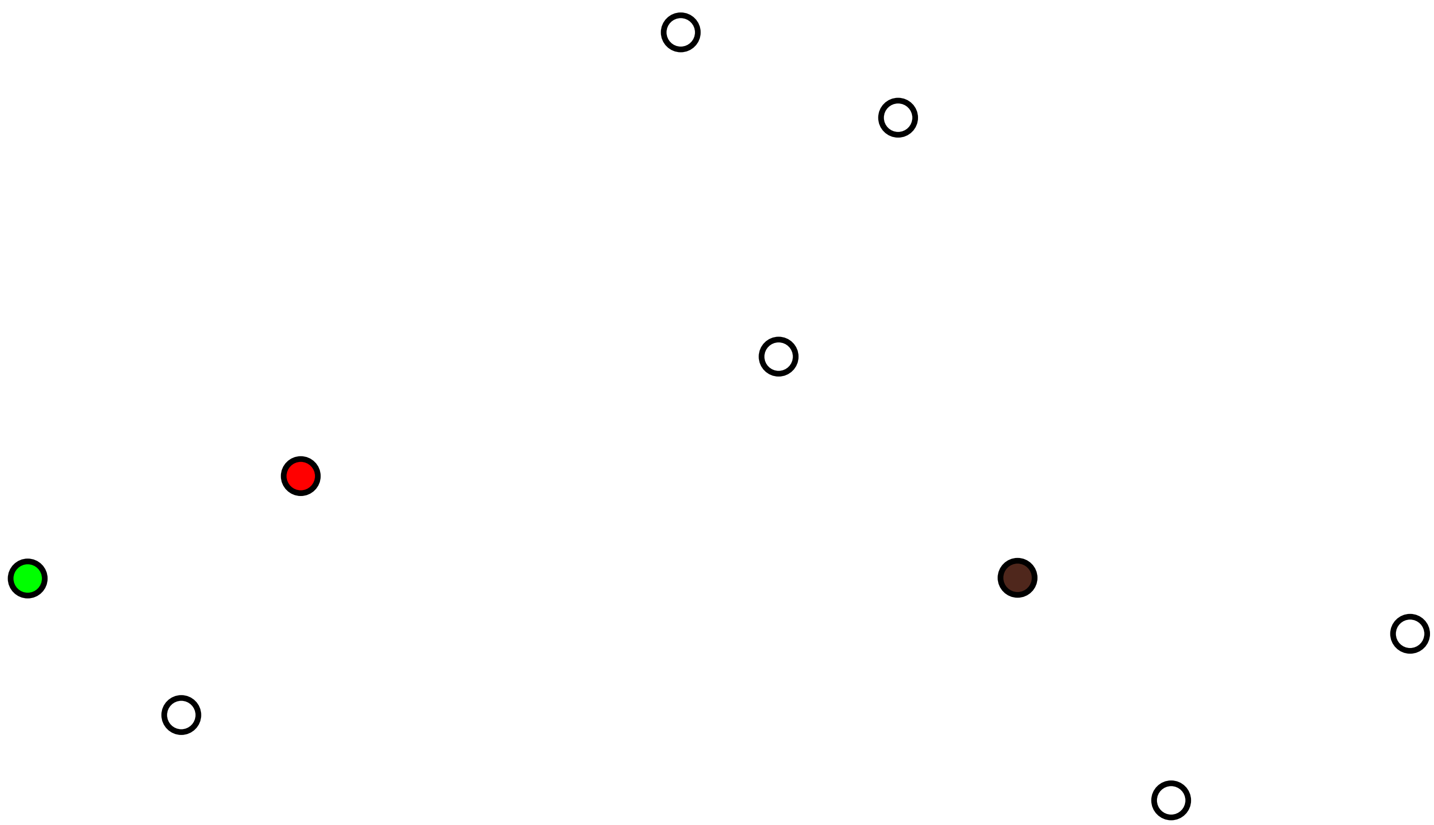
Example: Given a set of datapoints



Slide courtesy of Nina Balcan

Lloyd’s method: Random Initialization

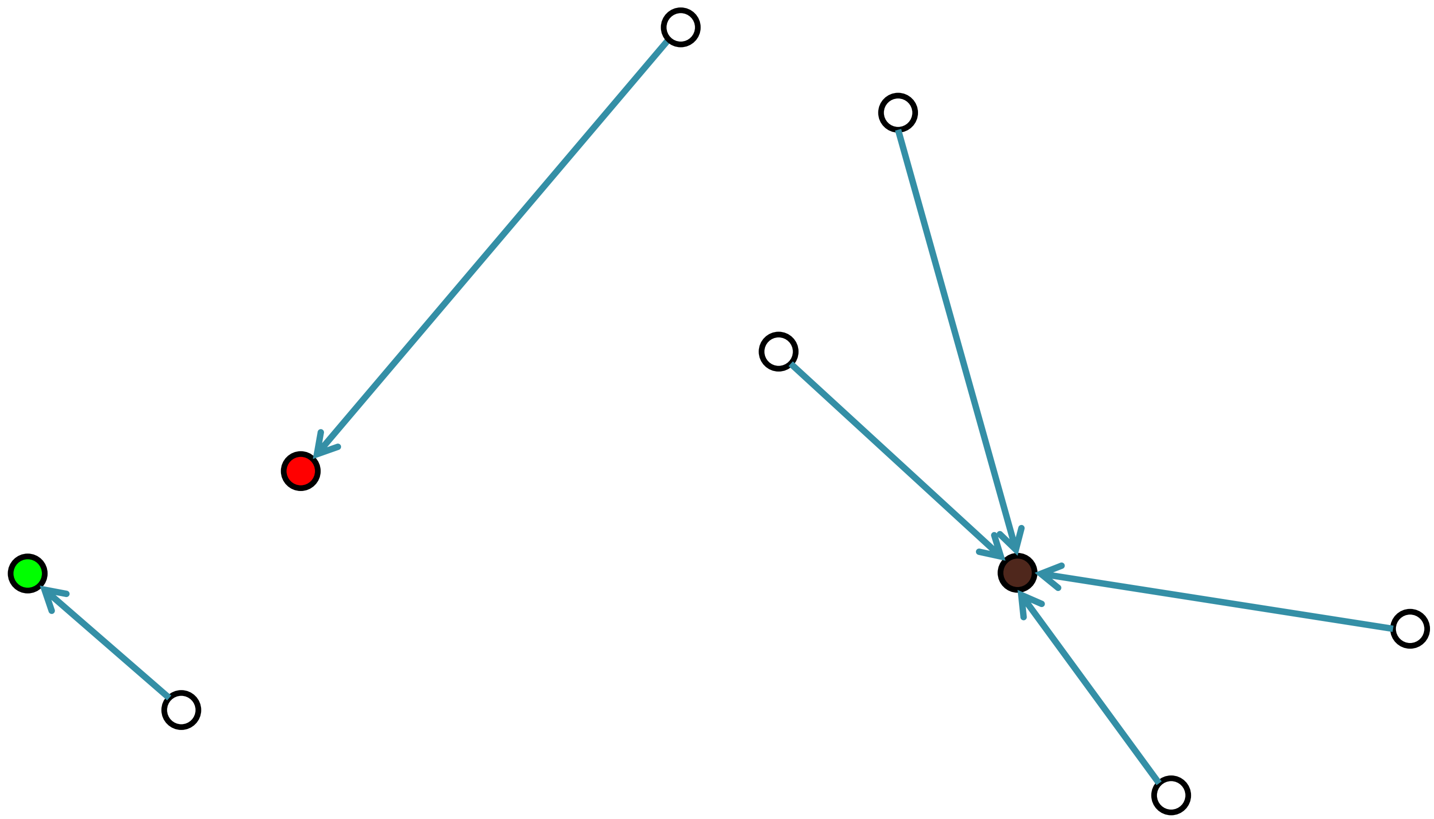
Select initial centers at random



Slide courtesy of Nina Balcan

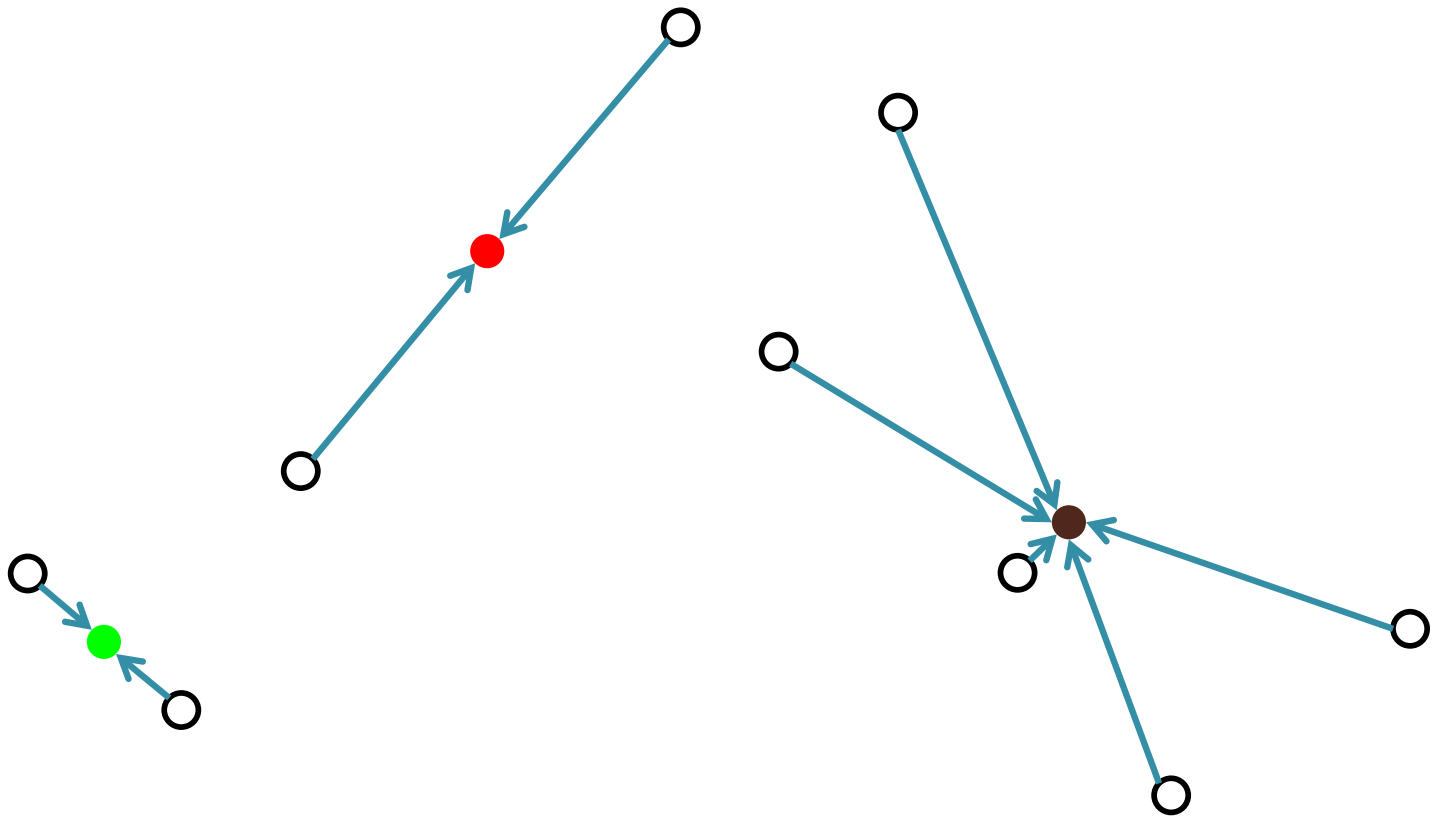
Lloyd's method: Random Initialization

Assign each point to its nearest center



Lloyd’s method: Random Initialization

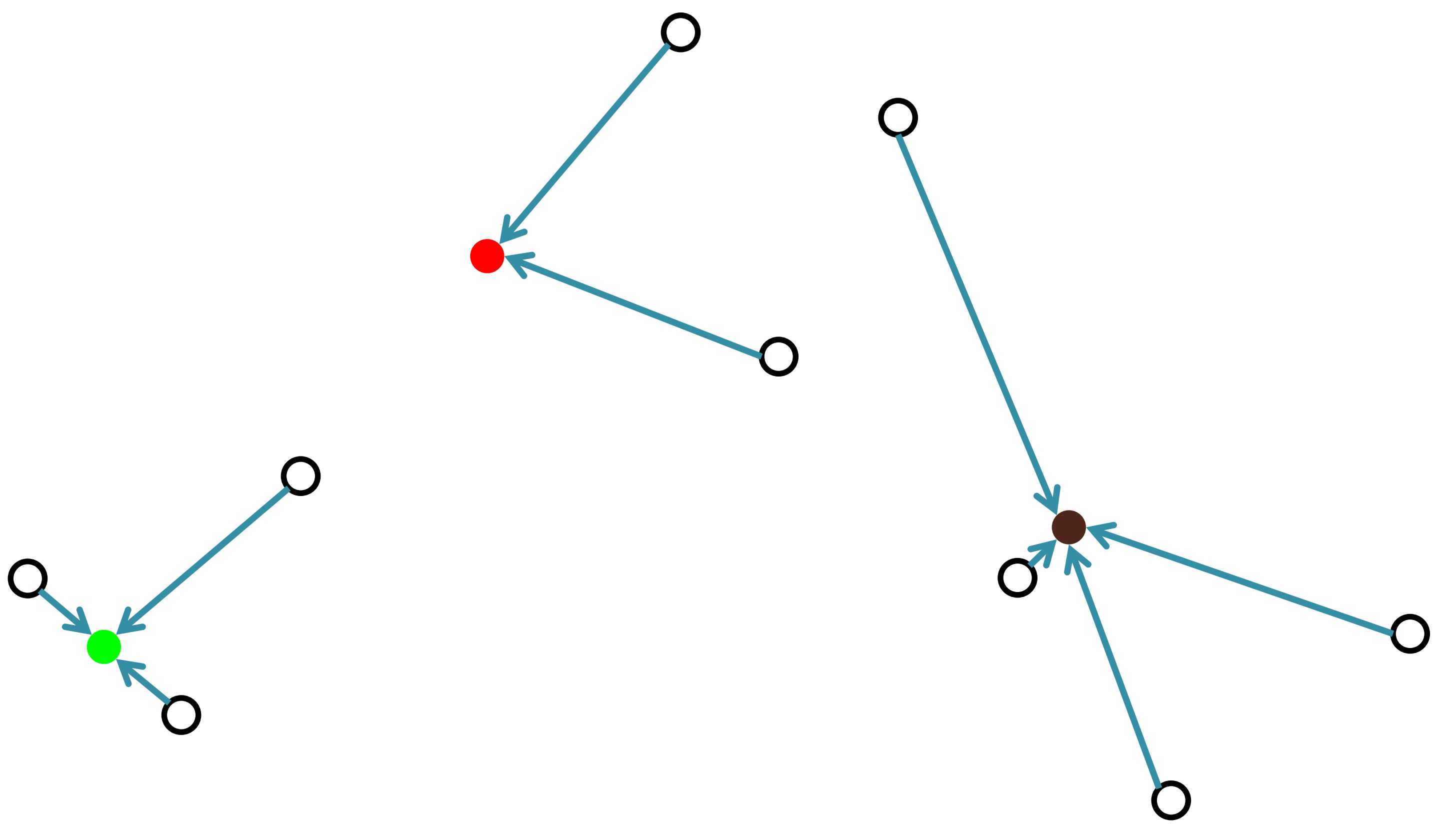
Recompute optimal centers given a fixed clustering



Slide courtesy of Nina Balcan

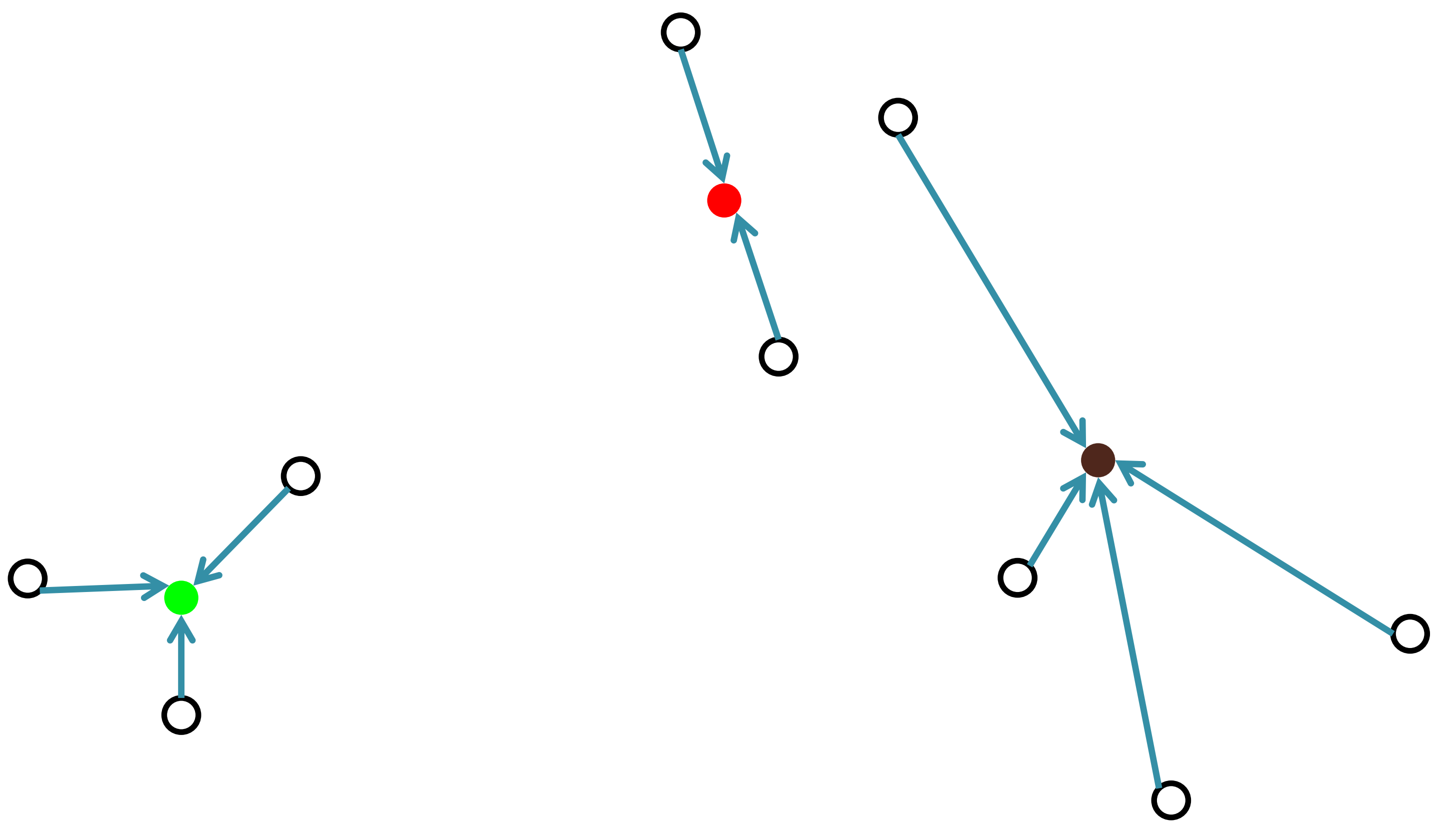
Lloyd's method: Random Initialization

Assign each point to its nearest center



Lloyd's method: Random Initialization

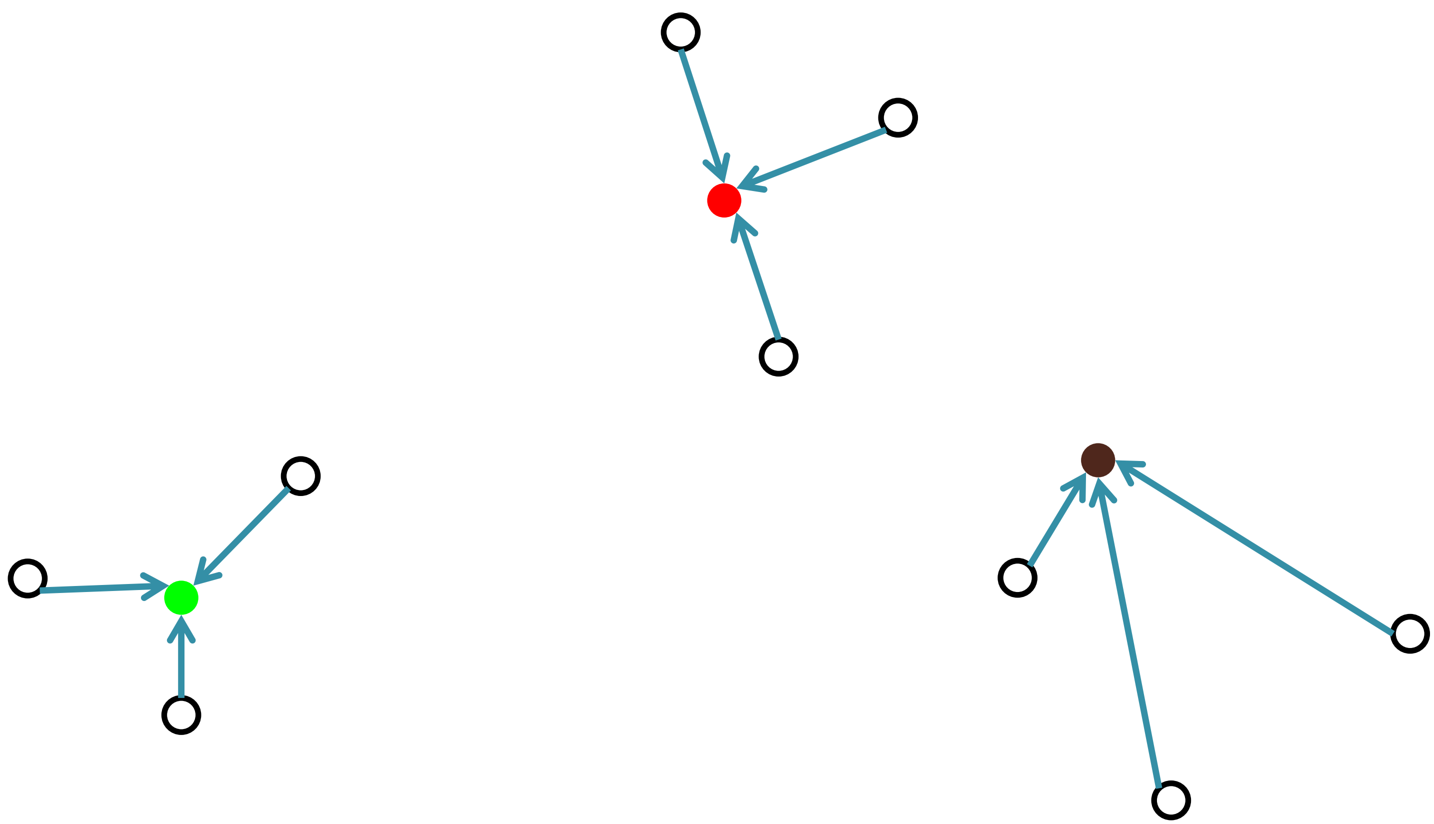
Recompute optimal centers given a fixed clustering



Slide courtesy of Nina Balcan

Lloyd's method: Random Initialization

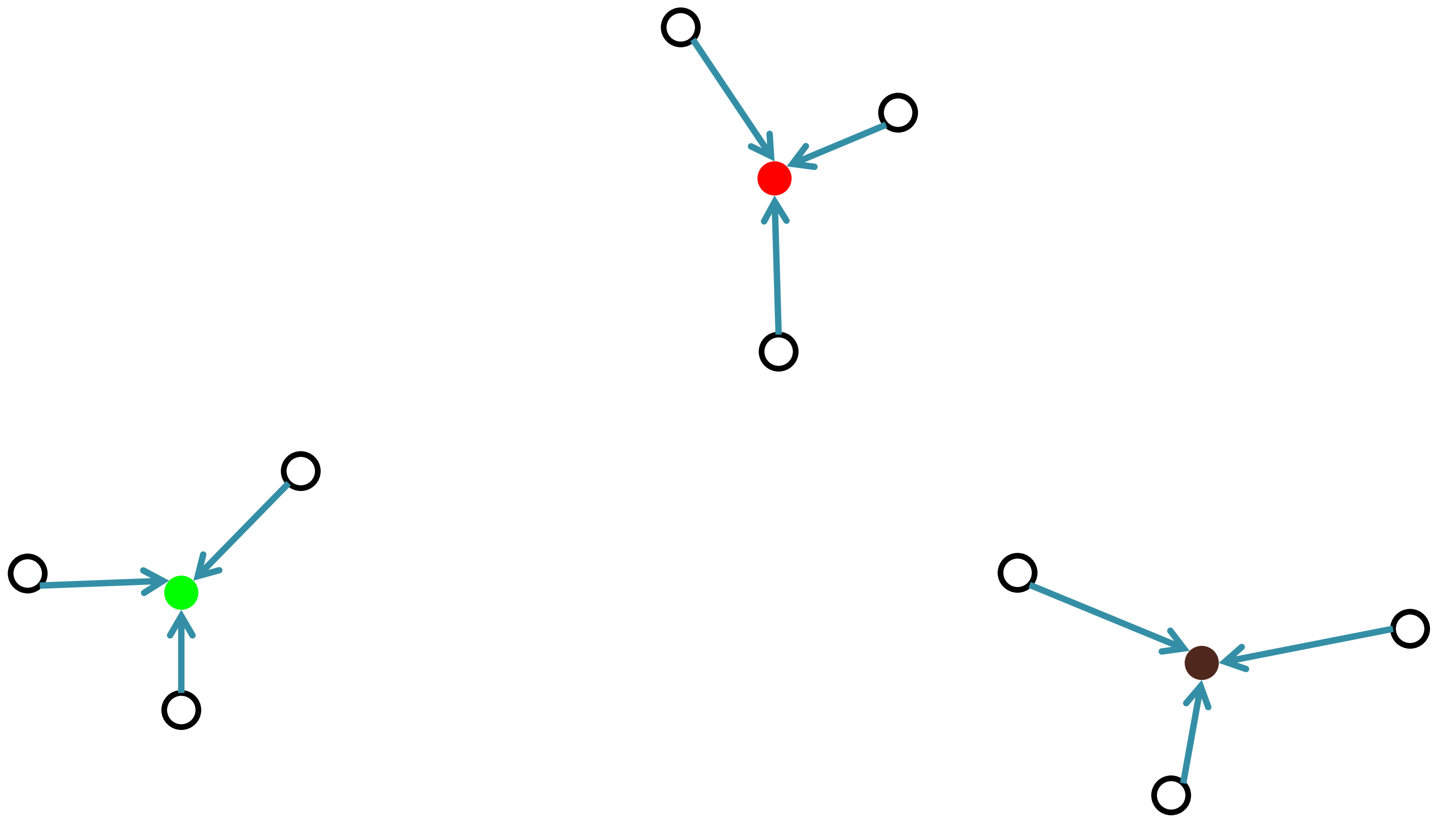
Assign each point to its nearest center



Slide courtesy of Nina Balcan

Lloyd’s method: Random Initialization

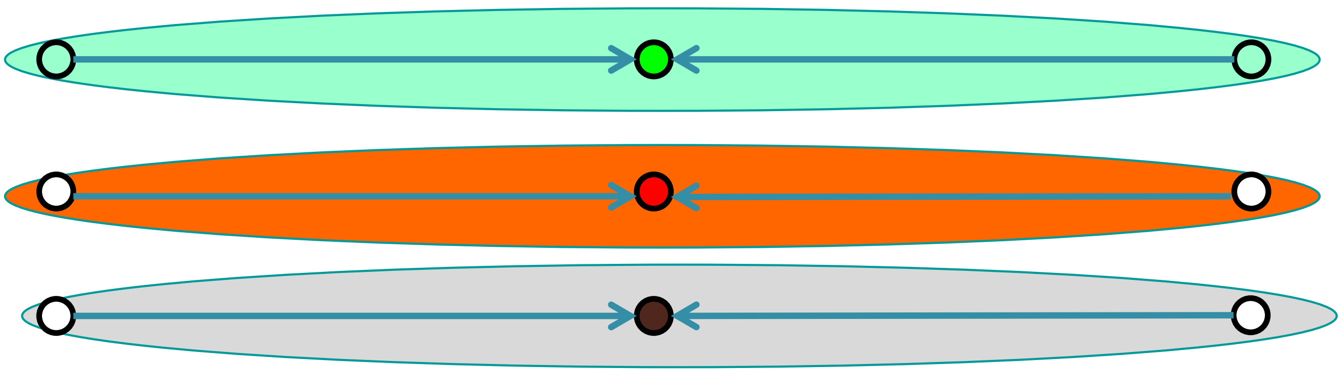
Recompute optimal centers given a fixed clustering



Get a good quality solution in this example.

Slide courtesy of Nina Balcan

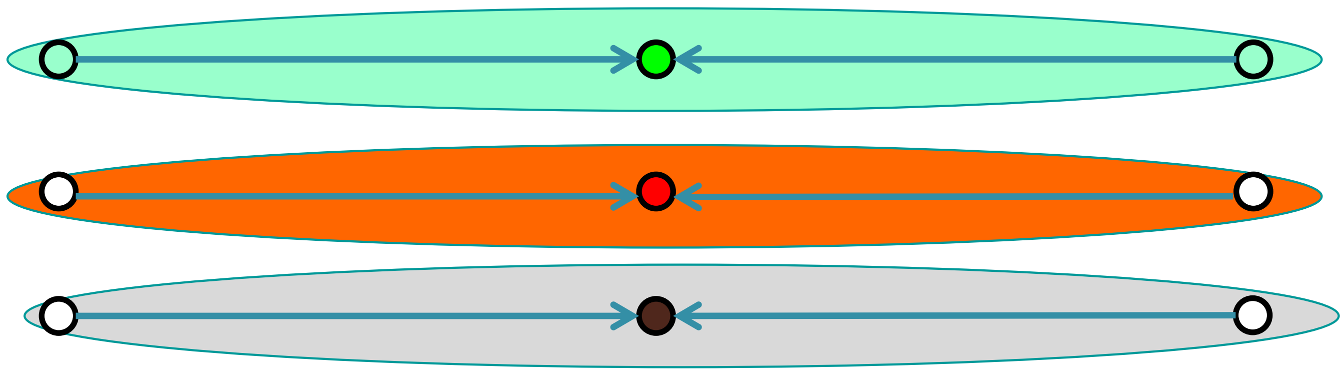
Lloyd's method: Performance



It always converges, but it may converge at a local optimum that is different from the global optimum, and in fact could be arbitrarily worse in terms of its score.

Slide courtesy of Nina Balcan

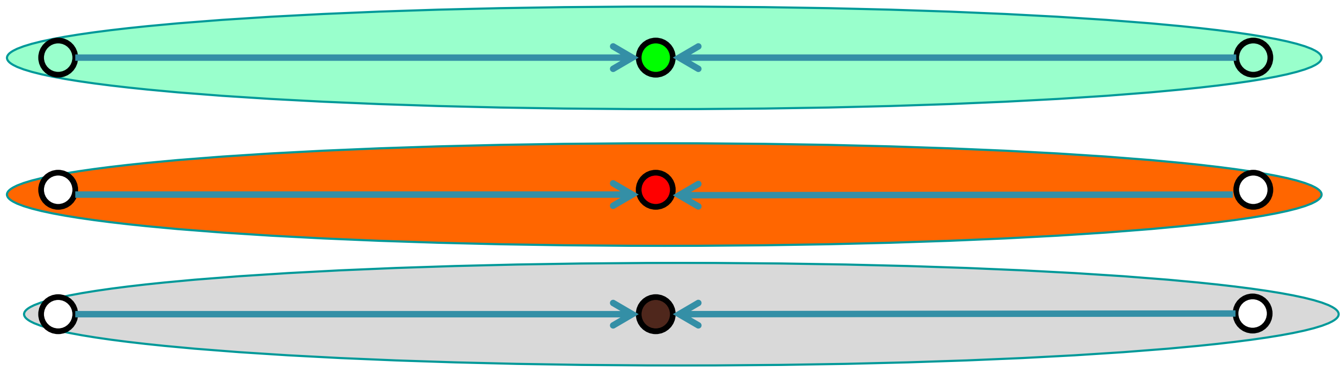
Lloyd's method: Performance



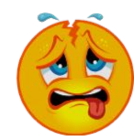
Local optimum: every point is assigned to its nearest center and every center is the mean value of its points.

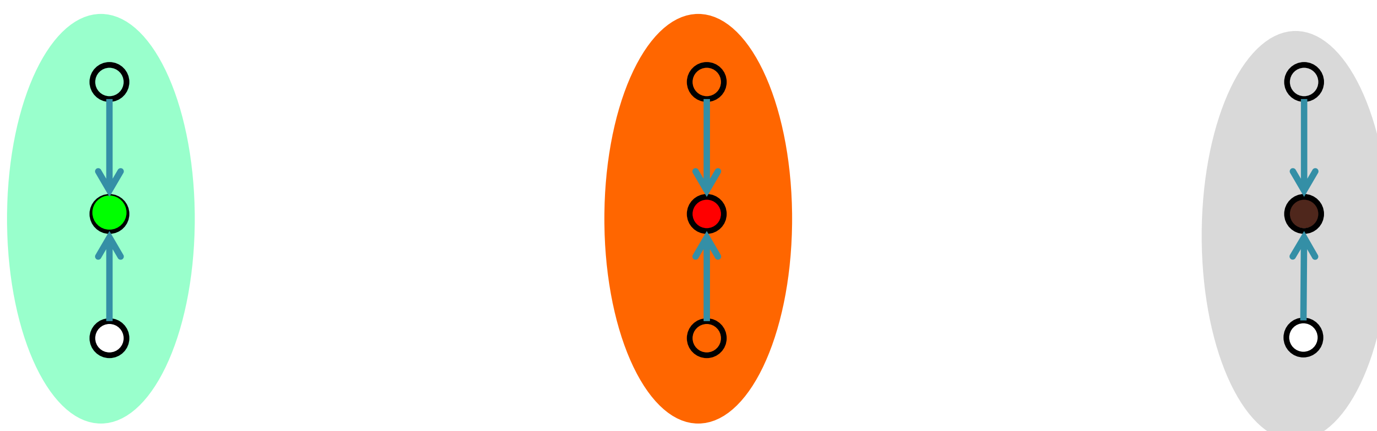
Slide courtesy of Nina Balcan

Lloyd's method: Performance



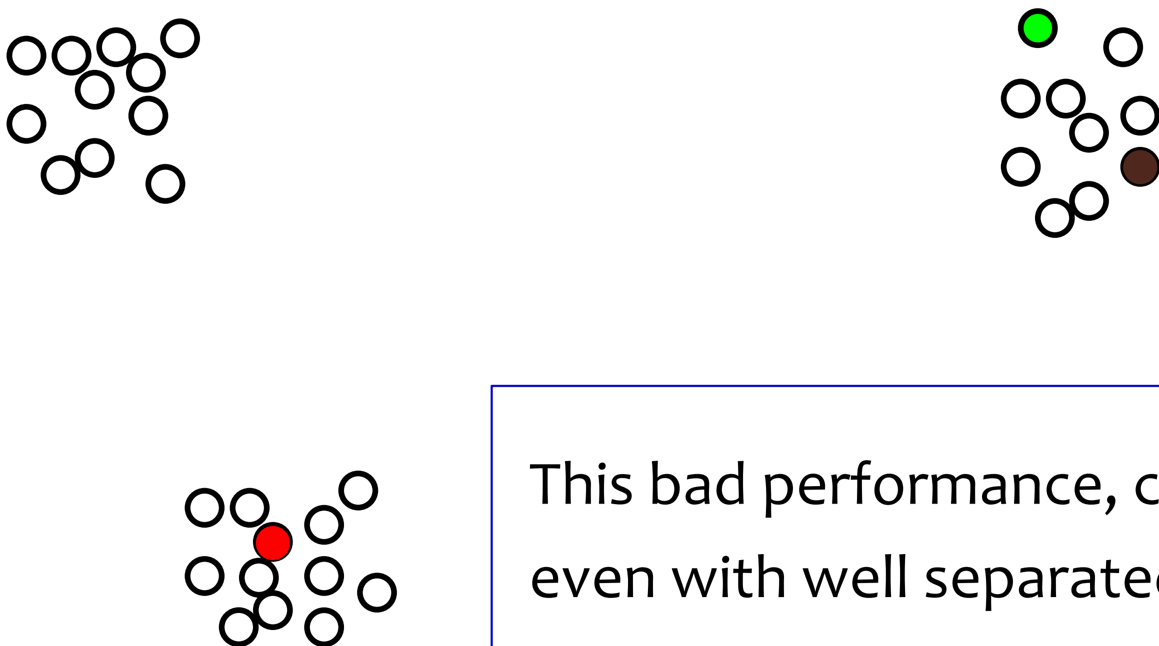
.It is arbitrarily worse than optimum solution….





Slide courtesy of Nina Balcan

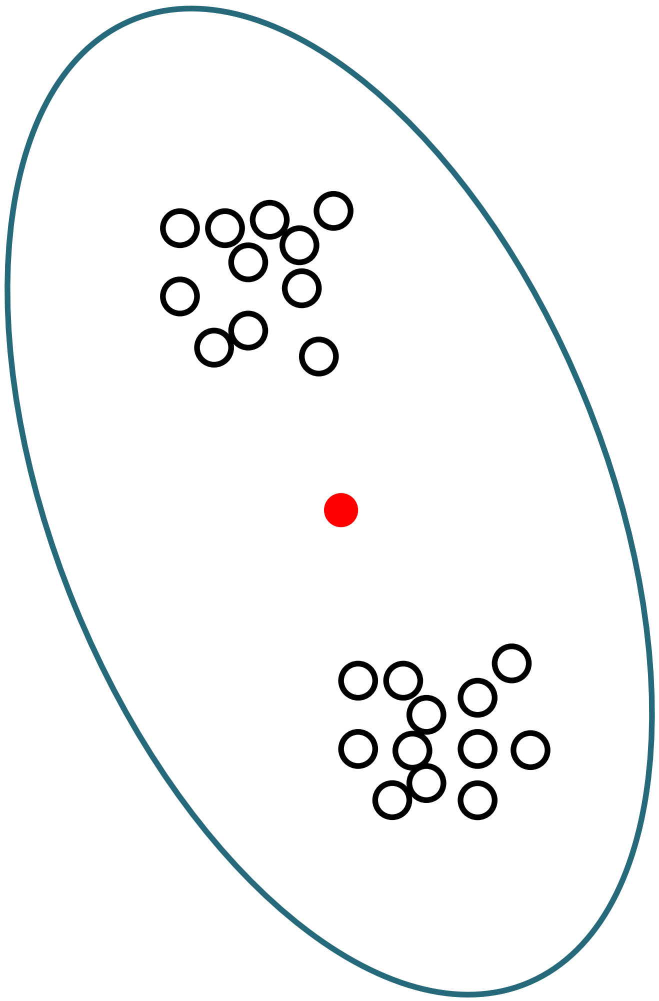
Lloyd's method: Performance

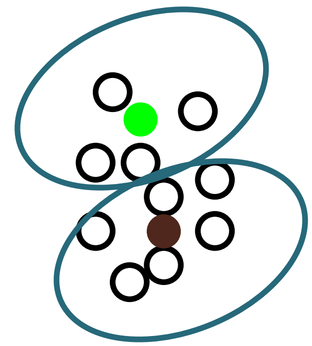


This bad performance, can happen even with well separated Gaussian clusters.

Slide courtesy of Nina Balcan

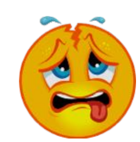
Lloyd's method: Performance





This bad performance, can happen even with well separated Gaussian clusters.

Some Gaussian are combined…..



Slide courtesy of Nina Balcan

Learning Objectives

K-Means

You should be able to…

1. Distinguish between coordinate descent and block coordinate descent
2. Define an objective function that gives rise to a "good" clustering
3. Apply block coordinate descent to an objective function preferring each point to be close to its nearest objective function to obtain the K-Means algorithm
4. Implement the K-Means algorithm
5. Connect the nonconvexity of the K-Means objective function with the (possibly) poor performance of random initialization