Machine Learning: Chenhao Tan

University of Colorado Boulder

LECTURE 18

Slides adapted from Jordan Boyd-Graber, Chris Ketelsen

Logistics

- Project proposal is due on Friday!

Overview

Hinge-loss view of soft-margin SVM

Kernels

Examples

Outline

Hinge-loss view of soft-margin SVM

Kernels

Examples

Recap: Karush-Kuhn-Tucker (KKT) conditions

Primal and dual feasibility

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0, C \geq \alpha_i \geq 0, \beta_i \geq 0$$

Stationarity

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i, \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i + \beta_i = C$$

Complementary slackness

$$\alpha_i[y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1 + \xi_i] = 0, \beta_i \xi_i = 0$$

Soft-margin SVM

$$\min_{\boldsymbol{w},b,\xi} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_i \xi_i$$

subject to

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, i \in [1, m]$$
$$\xi_i \geq 0, i \in [1, m]$$

What is ξ_i?

Soft-margin SVM

$$\min_{\boldsymbol{w}, b, \xi} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_i \xi_i$$

subject to

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, i \in [1, m]$$
$$\xi_i \geq 0, i \in [1, m]$$

What is ξ_i?

$$\xi_i = \begin{cases} 0, & \text{if } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \\ 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b), & \text{otherwise} \end{cases}$$

Soft-margin SVM

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_i \ell^{(\text{hin})}(y_i, \boldsymbol{w} \cdot \boldsymbol{x}_i + b)$$

You can solve this with gradient descent.

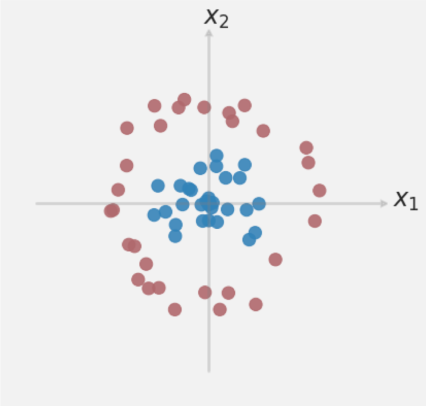
Outline

Hinge-loss view of soft-margin SVM

Kernels

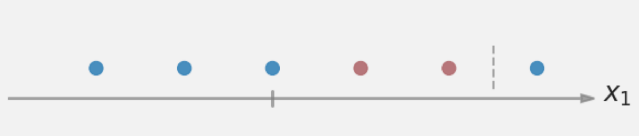
Examples

Can you solve this with linear separator?



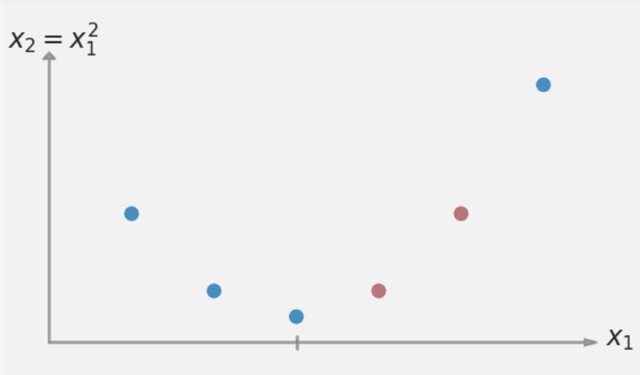
Can you solve this with linear separator?

What can we do if the data is clearly not linearly separable?



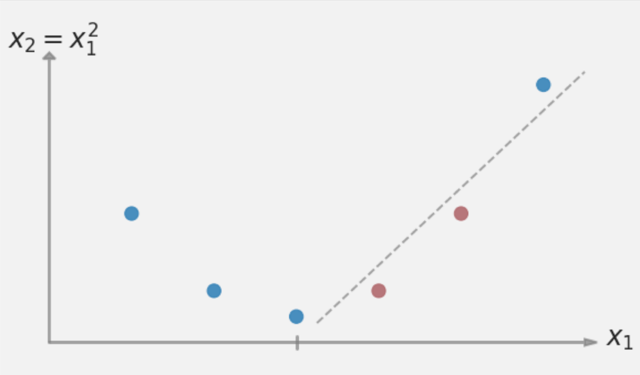
Can you solve this with linear separator?

Add a dimension.



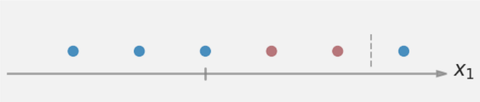
Can you solve this with linear separator?

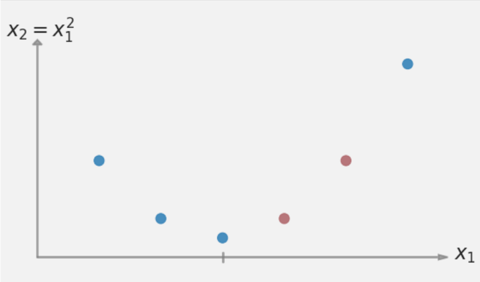
Add a dimension.



Derived features

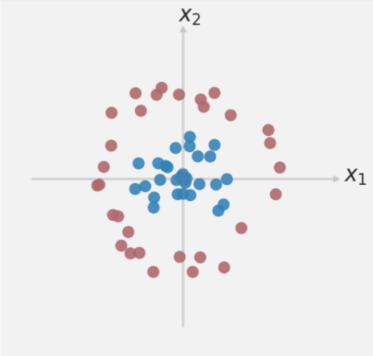
We started with the original feature vector, $\boldsymbol{x} = (x_1)$,
and we created a new derived feature vector, $\phi(\boldsymbol{x}) = (x_1, x_1^2)$.





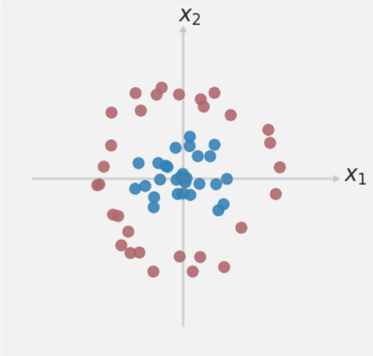
What about the previous problem?

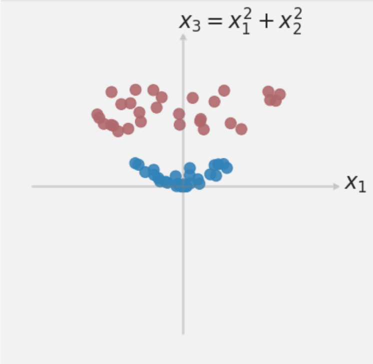
Definitely not separable in two dimensions.



What about the previous problem?

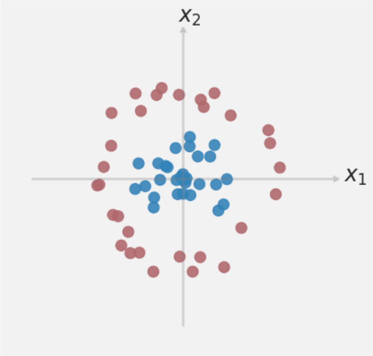
Definitely not separable in two dimensions.
But in three dimensions, it becomes easily separable.

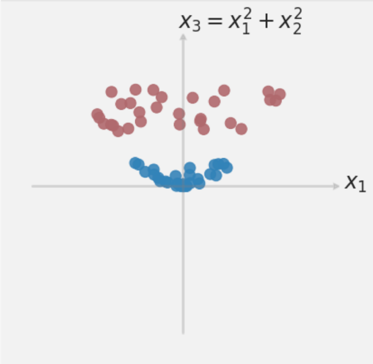




Derived features

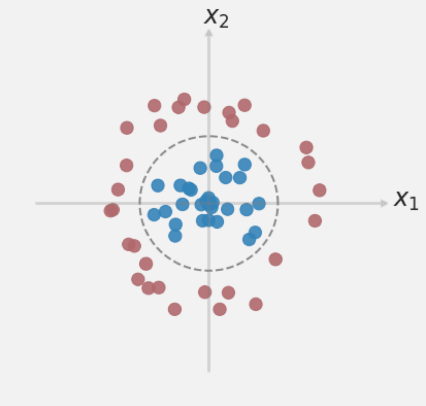
We started with the original feature vector, $\boldsymbol{x} = (x_1, x_2)$,
and we created a new derived feature vector, $\phi(\boldsymbol{x}) = (x_1, x_2, x_1^2 + x_2^2)$.





Derived features

We started with the original feature vector, $\boldsymbol{x} = (x_1, x_2)$,
and we created a new derived feature vector, $\phi(\boldsymbol{x}) = (x_1, x_2, x_1^2 + x_2^2)$.



What's special about SVMs?

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

What's special about SVMs?

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

- This dot product is basically just how much x_i looks like x_j. Can we generalize that?

What's special about SVMs?

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

- This dot product is basically just how much x_i looks like x_j. Can we generalize that?
- Kernels!

What's special about SVMs?

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}))$$

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i} \cdot \boldsymbol{x_j})$$

What does the kernel trick buy us?

Polynomial kernel:

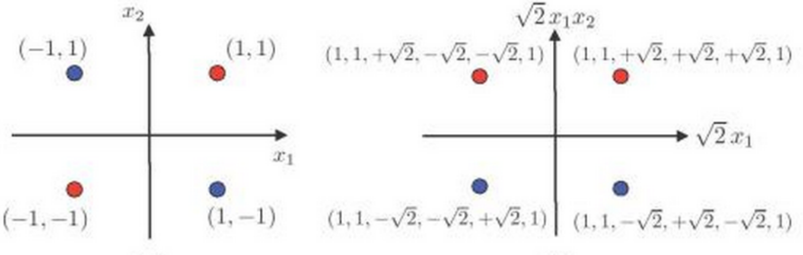
$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}' + c)^d$$

What does the kernel trick buy us?

Polynomial kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}' + c)^d$$

When $d = 2, c = 1$:



(a) (b)

What does the kernel trick buy us?

Polynomial kernel:

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}' + 1)^2$$

What is the corresponding $\phi(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^k$?
What is the complexity of storing $\phi(\boldsymbol{x})$ and computing $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$?
What about using the kernel function?

What's a kernel?

- A function $K : \mathcal{X} \times \mathcal{X} \mapsto \mathrm{R}$ is a kernel over $\mathcal{X}$.
- This is equivalent to taking the dot product $\langle \phi(x_1), \phi(x_2) \rangle$ for some mapping
- **Mercer's Theorem**: So long as the function is continuous and symmetric, then K admits an expansion of the form

$$K(x, x') = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(x')$$

What's a kernel?

- A function $K : \mathcal{X} \times \mathcal{X} \mapsto \mathrm{R}$ is a kernel over $\mathcal{X}$.
- This is equivalent to taking the dot product $\langle \phi(x_1), \phi(x_2) \rangle$ for some mapping
- **Mercer's Theorem**: So long as the function is continuous and symmetric, then K admits an expansion of the form

$$K(x, x') = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(x')$$

- The computational cost is just in computing the kernel

Kernel Matrix

The important property of the kernel matrix $\boldsymbol{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite.

Kernel Matrix

The important property of the kernel matrix $\boldsymbol{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite.

$$\boldsymbol{K}^T = \boldsymbol{K}$$

Kernel Matrix

The important property of the kernel matrix $\boldsymbol{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite.

$$\boldsymbol{K}^T = \boldsymbol{K}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$$

Kernel Matrix

The important property of the kernel matrix $\boldsymbol{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite.

$$\boldsymbol{K}^T = \boldsymbol{K}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$$

Also known as Gram matrix.

Gaussian Kernel

$$K(x, x') = \exp\left(-\frac{\|x' - x\|^2}{2\sigma^2}\right)$$

Gaussian Kernel

$$K(x, x') = \exp\left(-\frac{\left\|x' - x\right\|^2}{2\sigma^2}\right)$$

which can be rewritten as

$$K(x, x') = \sum_n \frac{(x \cdot x')^n}{\sigma^n n!}$$

(All polynomials!)

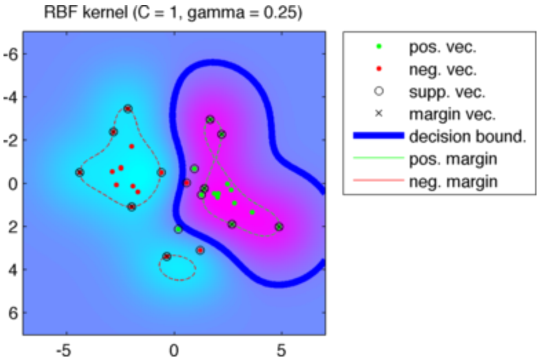
Gaussian Kernel

$$K(x,x') = \exp\left(-\frac{\|x'-x\|^2}{2\sigma^2}\right)$$

which can be rewritten as

$$K(x,x') = \sum_n \frac{(x \cdot x')^n}{\sigma^n n!}$$

(All polynomials!)



How does it affect optimization

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \qquad \max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

- Replace all dot product with kernel evaluations $K(x_1, x_2)$
- Makes computation more expensive, overall structure is the same

Outline

Hinge-loss view of soft-margin SVM

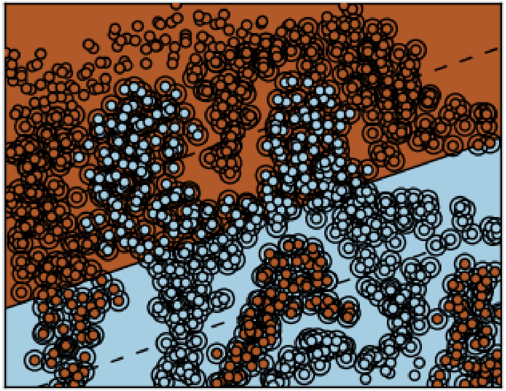
Kernels

Examples

Kernelized SVM

```
X, Y = read_data("ex8a.txt")
clf = svm.SVC(kernel=kk, degree=dd, gamma=gg)
clf.fit(X, Y)
```

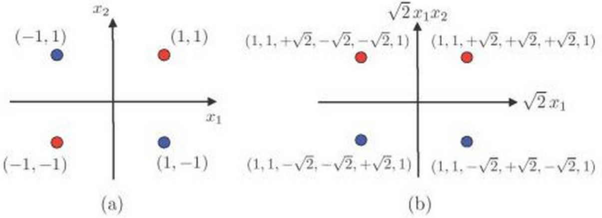
Linear Kernel Doesn't Work



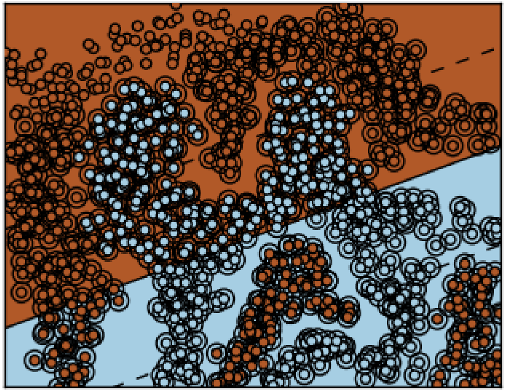
Polynomial Kernel

$$K(x, x') = (x \cdot x' + c)^d$$

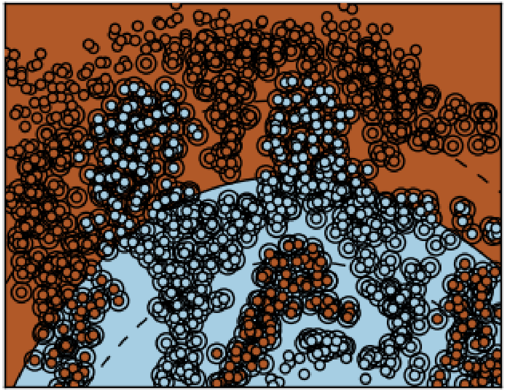
When $d = 2$:



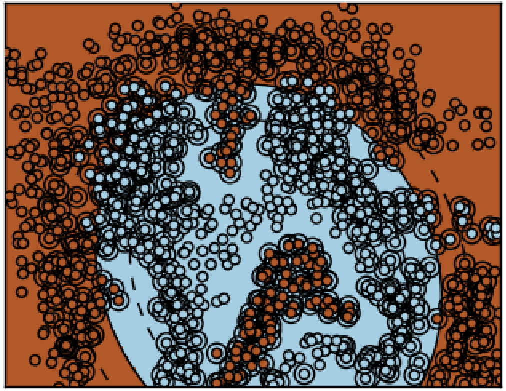
Polynomial Kernel $d = 1, c = 5$



Polynomial Kernel $d = 2, c = 5$

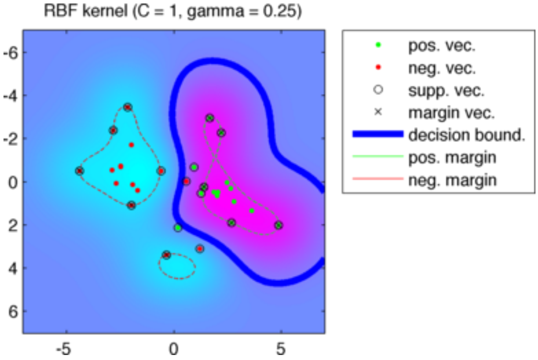


Polynomial Kernel $d = 3, c = 5$

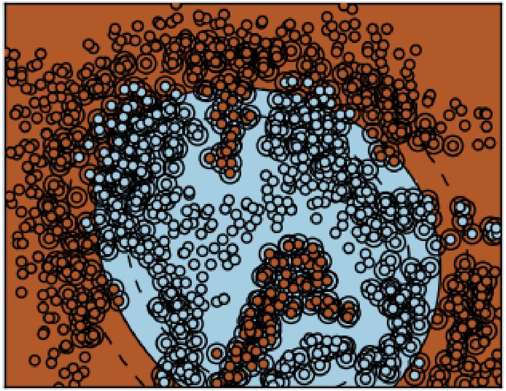


Gaussian Kernel

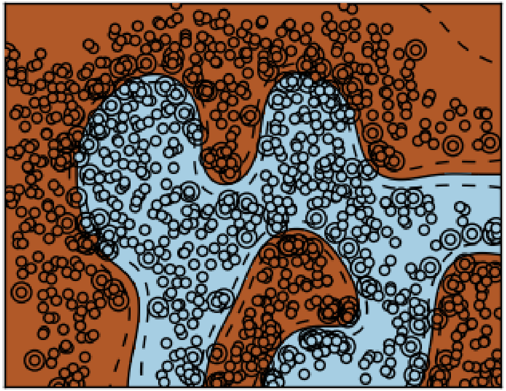
$$K(x, x') = \exp\left(-\gamma \left\|x' - x\right\|^2\right)$$



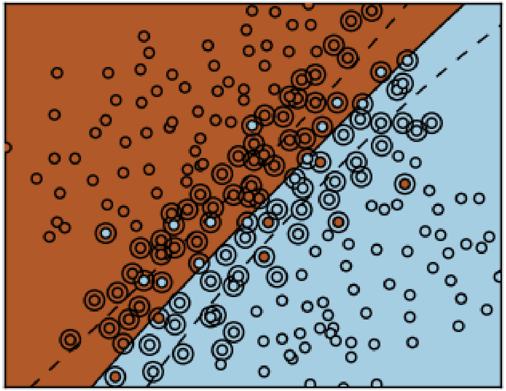
RBF Kernel $\gamma = 2$



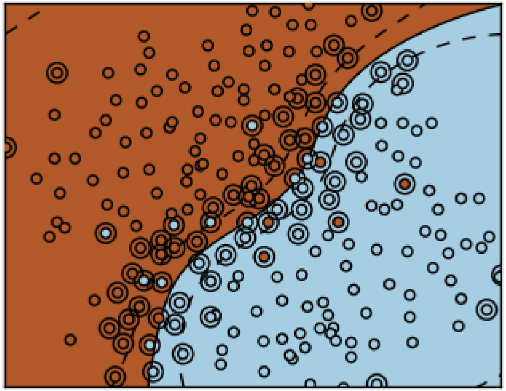
RBF Kernel $\gamma = 100$



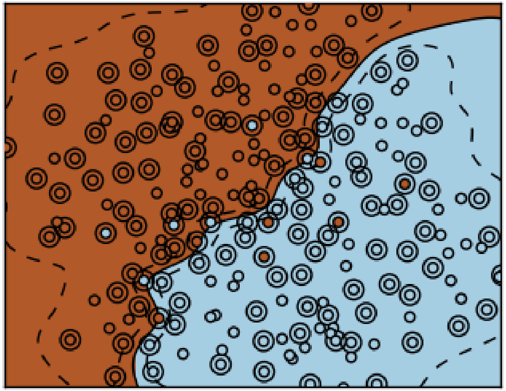
RBF Kernel $\gamma = 1$



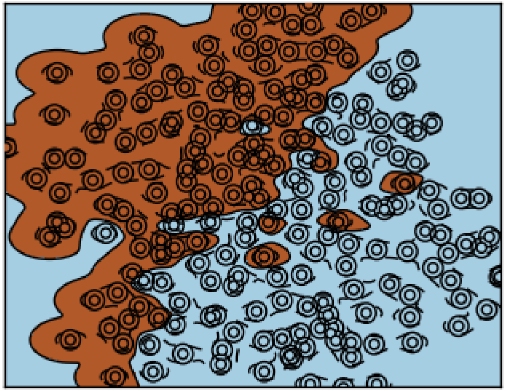
RBF Kernel $\gamma = 10$



RBF Kernel $\gamma = 100$



RBF Kernel $\gamma = 1000$



Be careful!

- Which has the lowest training error?
- Which one would generalize best?

Recap

- This completes our discussion of SVMs
- Workhorse method of machine learning
- Flexible, fast, effective

Recap

- This completes our discussion of SVMs
- Workhorse method of machine learning
- Flexible, fast, effective
- Kernels: applicable to wide range of data, inner product trick keeps method simple