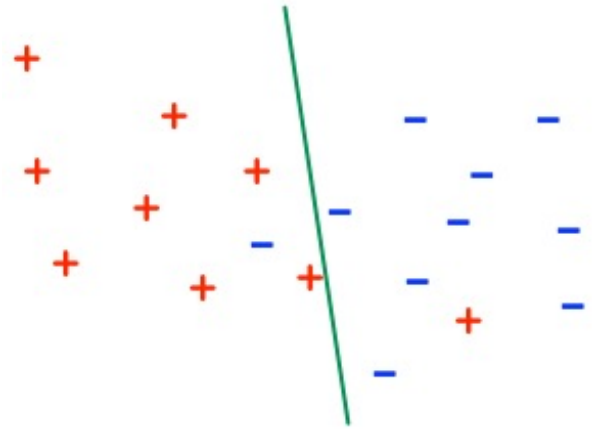
Kernel methods

Deviations from linear separability

Noise

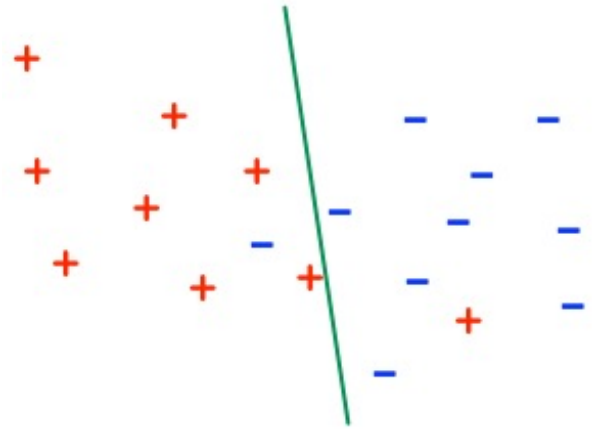


Find a separator that minimizes
a convex loss function related
to the number of mistakes.

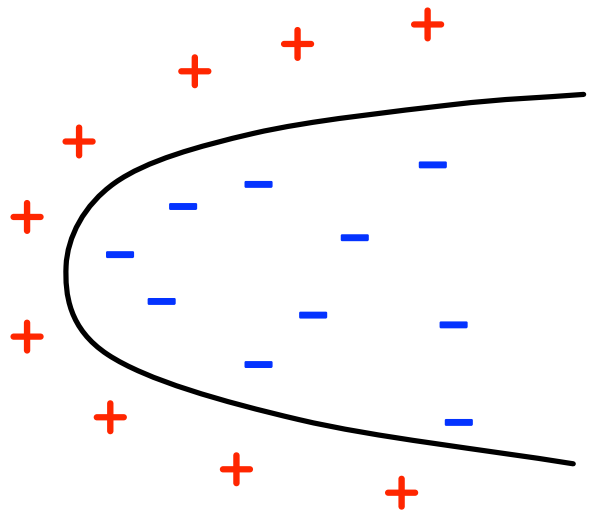
e.g. SVM, logistic regression.

Deviations from linear separability

Noise



Systematic deviation



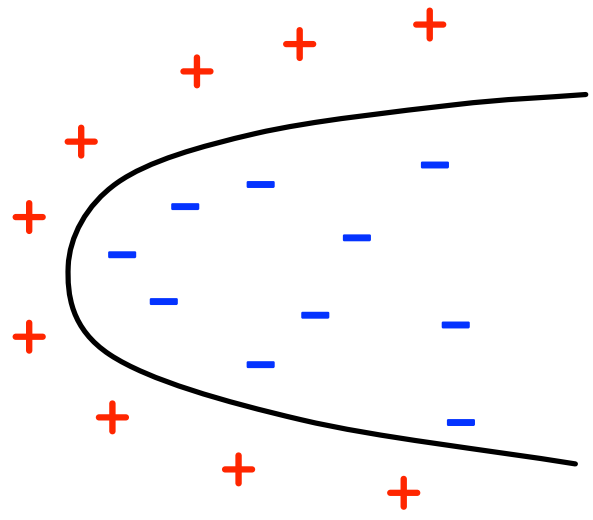
Find a separator that minimizes a convex loss function related to the number of mistakes.

e.g. SVM, logistic regression.

What to do with this?

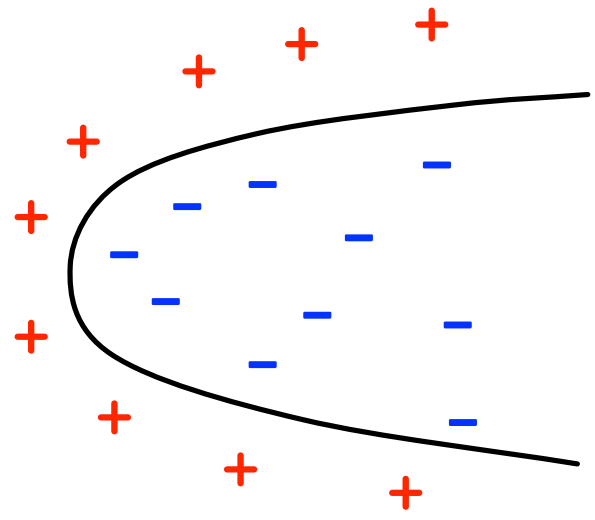
Systematic inseparability

Boundary looks **quadratic**:



Systematic inseparability

Boundary looks **quadratic**:



Quick fix: augment the regular features $x = (x_1, x_2, \ldots, x_d)$ with

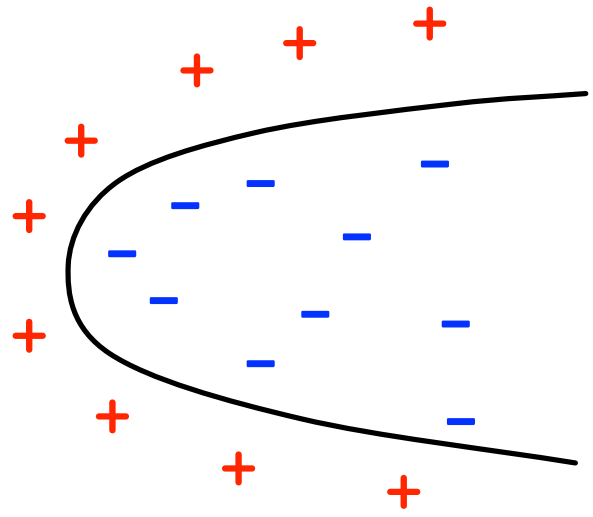
$$x_1^2,\ x_2^2,\ \ldots,\ x_d^2$$

$$x_1x_2,\ x_1x_3,\ \ldots,\ x_{d-1}x_d$$

New, enhanced data vectors of the form:

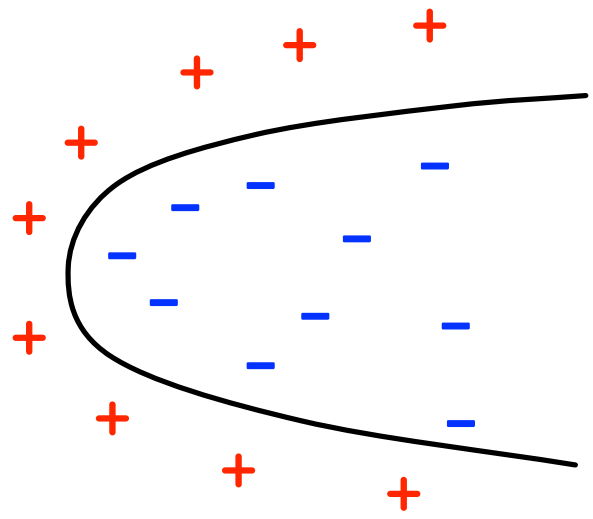
$$\Phi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1x_2, \ldots, x_{d-1}x_d)$$

Adding new features



Actual boundary is something like $x_1 = x_2^2 + 5$.

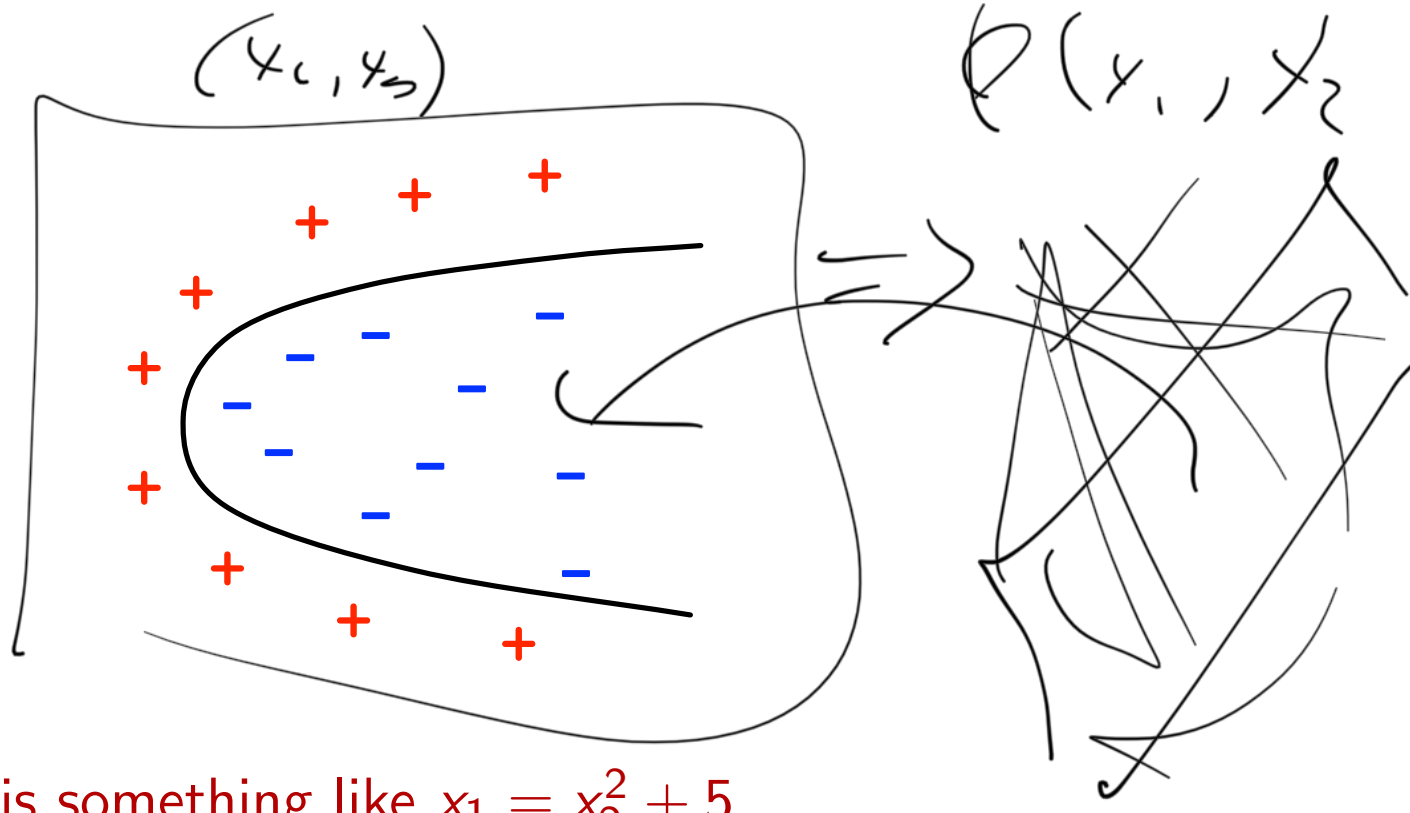
Adding new features



Actual boundary is something like $x_1 = x_2^2 + 5$.

- This is quadratic in $x = (x_1, x_2)$
- But it is linear in $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Adding new features



Actual boundary is something like $x_1 = x_2^2 + 5$.

- This is quadratic in $x = (x_1, x_2)$
- But it is linear in $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Basis expansion: embed the data in a higher-dimensional feature space. Then we can keep using a linear classifier!

Quick question

Suppose $x = (x_1, x_2, x_3)$. What is the dimension of $\Phi(x)$?

Suppose $x = (x_1, \ldots, x_d)$. What is the dimension of $\Phi(x)$?

Perceptron revisited

Learning in the higher-dimensional feature space:

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y\,\Phi(x)$
  - $b = b + y$

Final w is a weighted linear sum of various $\Phi(x)$.

Perceptron revisited

Learning in the higher-dimensional feature space:

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y\,\Phi(x)$
  - $b = b + y$

Final w is a weighted linear sum of various $\Phi(x)$.

Problem: number of features has now increased dramatically.
For MNIST, from 784 to 307720!

The kernel trick: idea

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y\,\Phi(x)$
  - $b = b + y$

Final w is a weighted linear sum of various $\Phi(x)$.

The **kernel trick**:

- We only access $\Phi(x)$ in order to compute $w \cdot \Phi(x)$. If, say,

$$w = a_1\Phi(x^{(1)}) + a_2\Phi(x^{(2)}) + a_3\Phi(x^{(3)}),$$

then $w \cdot \Phi(x)$ is a weighted sum of dot products $\Phi(x) \cdot \Phi(x^{(i)})$.

The kernel trick: idea

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y\,\Phi(x)$
  - $b = b + y$

Final w is a weighted linear sum of various $\Phi(x)$.

The **kernel trick**:

- We only access $\Phi(x)$ in order to compute $w \cdot \Phi(x)$. If, say,

$$w = a_1\Phi(x^{(1)}) + a_2\Phi(x^{(2)}) + a_3\Phi(x^{(3)}),$$

  then $w \cdot \Phi(x)$ is a weighted sum of dot products $\Phi(x) \cdot \Phi(x^{(i)})$.
- We can compute dot products $\Phi(x) \cdot \Phi(z)$ directly from x, z. No need to write out $\Phi(x)$ or $\Phi(z)$!

Computing dot products

First, in 2-d.
Suppose $x = (x_1, x_2)$ and $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

Actually, tweak a little: $\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$

Computing dot products

First, in 2-d.
Suppose $x = (x_1, x_2)$ and $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

Actually, tweak a little: $\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$

What is $\Phi(x) \cdot \Phi(z)$?

Computing dot products

Suppose $x = (x_1, x_2, \ldots, x_d)$ and

$$\Phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d)$$

$$\begin{aligned}
\Phi(x) \cdot \Phi(z) &= (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d) \cdot \\
&\quad (1, \sqrt{2}z_1, \ldots, \sqrt{2}z_d, z_1^2, \ldots, z_d^2, \sqrt{2}z_1z_2, \ldots, \sqrt{2}z_{d-1}z_d) \\
&= 1 + 2\sum_i x_i z_i + \sum_i x_i^2 z_i^2 + 2\sum_{i \neq j} x_i x_j z_i z_j \\
&= (1 + x_1z_1 + \cdots + x_dz_d)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}$$

Computing dot products

Suppose $x = (x_1, x_2, \ldots, x_d)$ and

$$\Phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d)$$

$$\begin{aligned}
\Phi(x) \cdot \Phi(z) &= (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d) \cdot \\
&\quad (1, \sqrt{2}z_1, \ldots, \sqrt{2}z_d, z_1^2, \ldots, z_d^2, \sqrt{2}z_1z_2, \ldots, \sqrt{2}z_{d-1}z_d) \\
&= 1 + 2\sum_i x_i z_i + \sum_i x_i^2 z_i^2 + 2\sum_{i \neq j} x_i x_j z_i z_j \\
&= (1 + x_1z_1 + \cdots + x_dz_d)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}$$

For MNIST:
We are computing dot products in 307720-dimensional space.
But it takes time proportional to 784, the original dimension!

The kernel trick

Why does it work?

1. The only time we ever look at the data – while training or using the classifier – is to compute dot products $w \cdot \Phi(x)$.

2. And w itself is a linear combination of $\Phi(x)$'s. If, say,

   $$w = \alpha_1 \Phi(x^{(1)}) + \alpha_{22} \Phi(x^{(22)}) + \alpha_{37} \Phi(x^{(37)}),$$

   we can store w as $[(1, \alpha_1), (22, \alpha_{22}), (37, \alpha_{37})]$.

3. Dot products $\Phi(x) \cdot \Phi(x')$ can be computed efficiently. Never need to write out the high-dimensional embedding $\Phi(\cdot)$.

Kernel Perceptron

Learning from data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}$

Primal form:

- $w = 0$ and $b = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$:
  - $w = w + y^{(i)}\, \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

Kernel Perceptron

Learning from data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}$

Primal form:

- $w = 0$ and $b = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$:
  - $w = w + y^{(i)}\, \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

Dual form: $w = \sum_j \alpha_j y^{(j)} \Phi(x^{(j)})$, where $\alpha \in \mathbb{R}^n$

- $\alpha = 0$ and $b = 0$
- while some i has $y^{(i)} \left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)}) + b \right) \leq 0$:
  - $\alpha_i = \alpha_i + 1$
  - $b = b + y^{(i)}$

Kernel Perceptron

Learning from data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}$

Primal form:

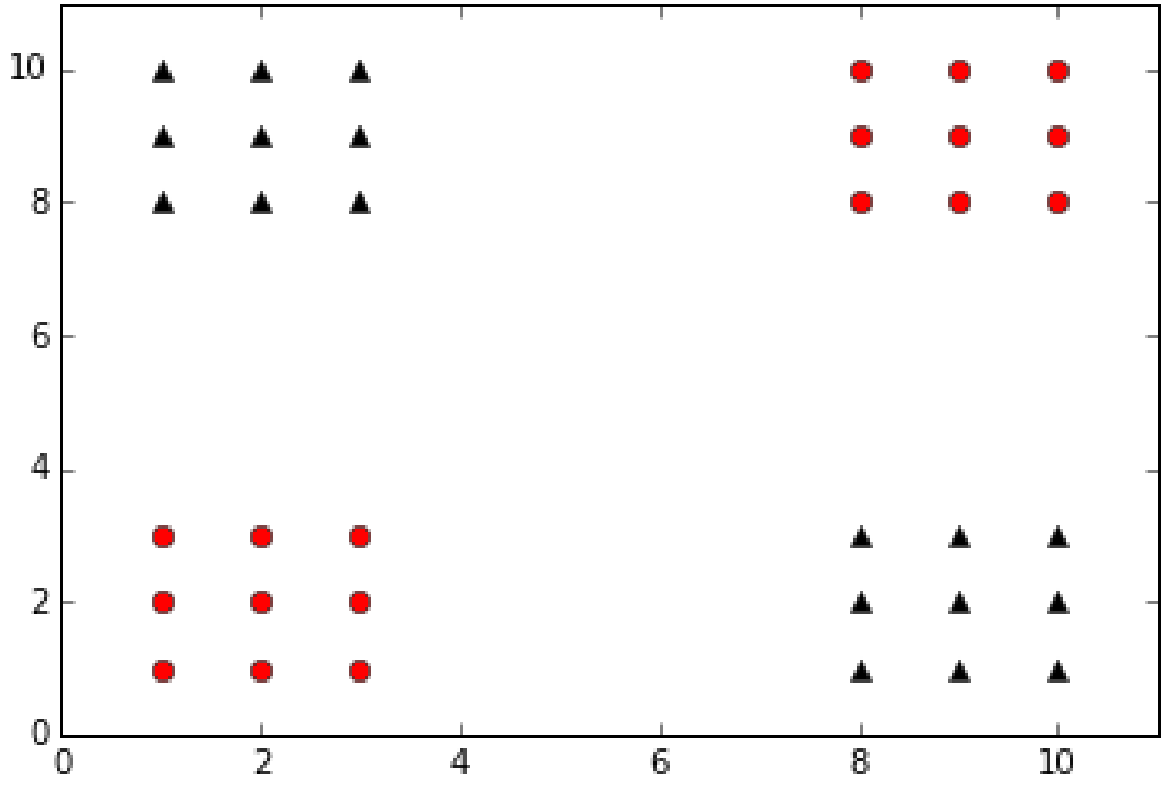
- $w = 0$ and $b = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$:
  - $w = w + y^{(i)}\, \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

Dual form: $w = \sum_j \alpha_j y^{(j)} \Phi(x^{(j)})$, where $\alpha \in \mathbb{R}^n$

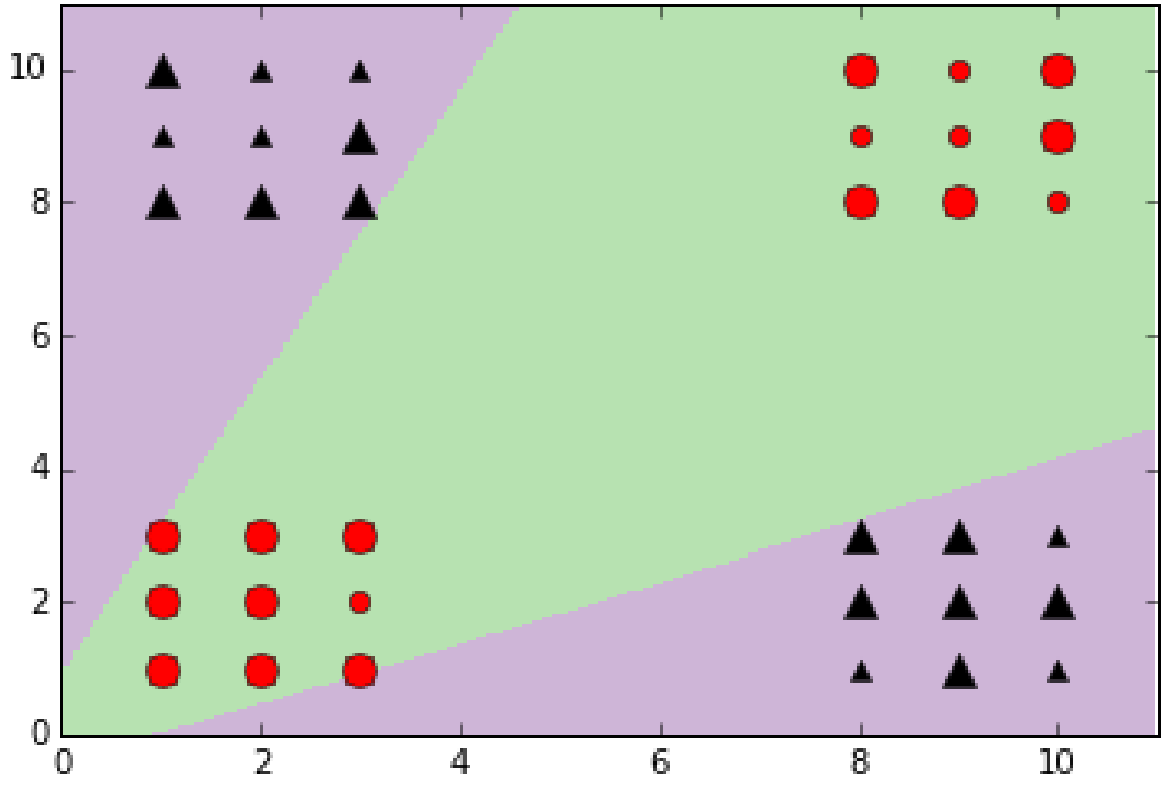
- $\alpha = 0$ and $b = 0$
- while some i has $y^{(i)} \left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)}) + b \right) \leq 0$:
  - $\alpha_i = \alpha_i + 1$
  - $b = b + y^{(i)}$

Classify a new point x: sign $\left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x) + b \right)$.

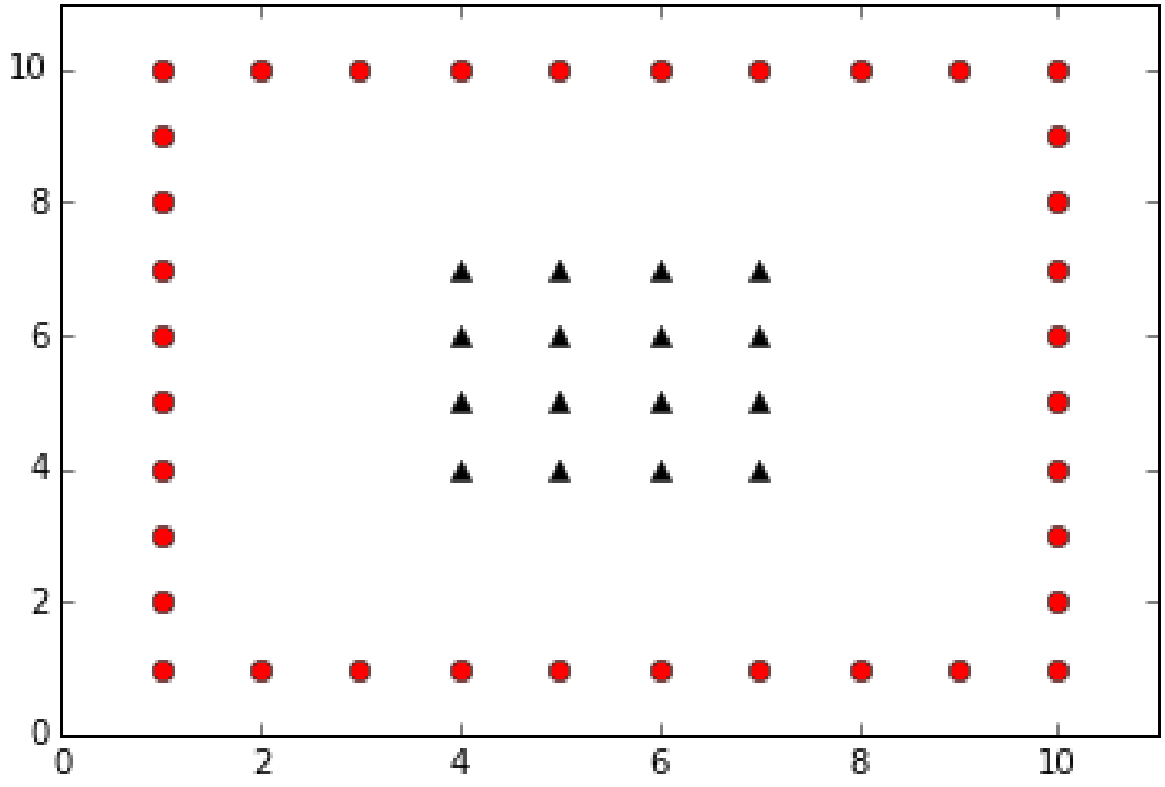
Kernel Perceptron: Examples



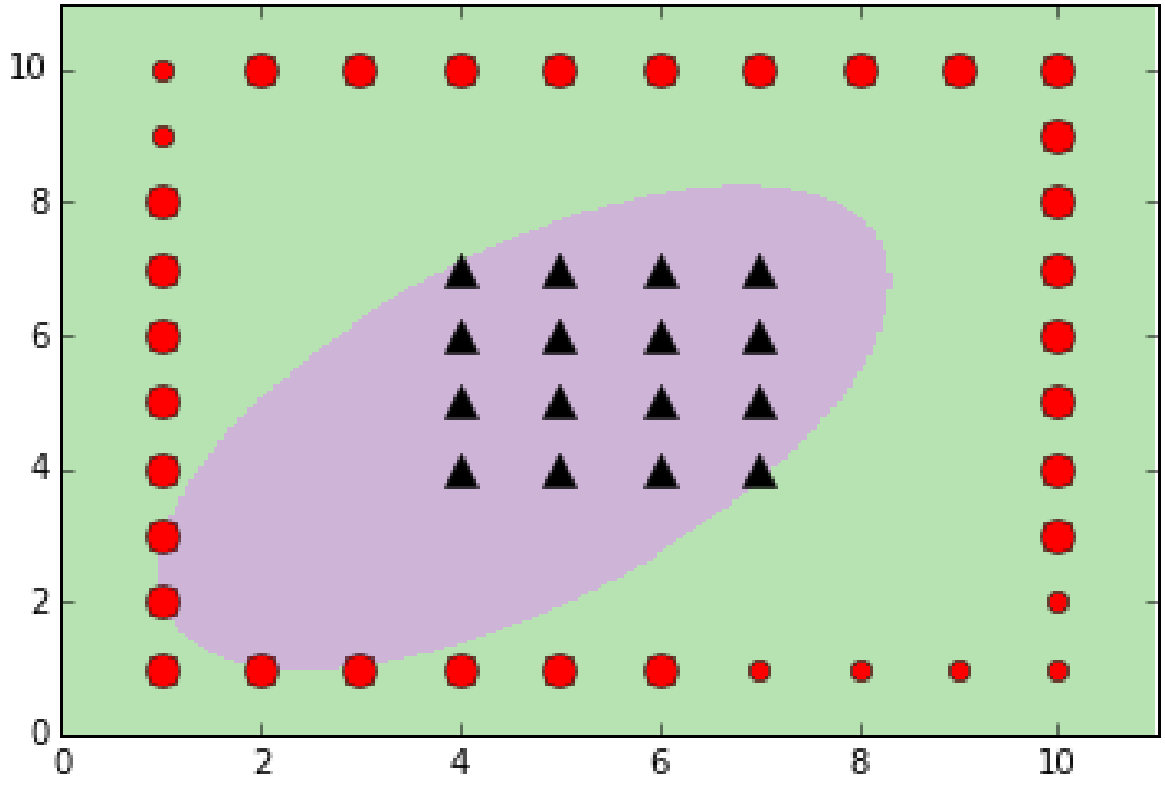
Kernel Perceptron: Examples



Kernel Perceptron: Examples



Kernel Perceptron: Examples



Does this work with SVMs?

$$\text{(PRIMAL)} \qquad \min_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.:} \quad y^{(i)}(w\cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n$$

$$\xi \geq 0$$

$$\text{(DUAL)} \qquad \max_{\alpha\in\mathbb{R}^n} \quad \sum_{i=1}^{n}\alpha_i - \sum_{i,j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)}\cdot x^{(j)})$$

$$\text{s.t.:} \quad \sum_{i=1}^{n}\alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

Solution: $w = \sum_i \alpha_i y^{(i)} x^{(i)}$.

Kernel SVM

1. **Embedding.** Pick a mapping $x \mapsto \Phi(x)$.

Kernel SVM

1. **Embedding.** Pick a mapping $x \mapsto \Phi(x)$.
2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))$$

$$\text{s.t.:} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$. Offset b also follows.

Kernel SVM

1. **Embedding.** Pick a mapping $x \mapsto \Phi(x)$.
2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))$$

$$\text{s.t.:} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

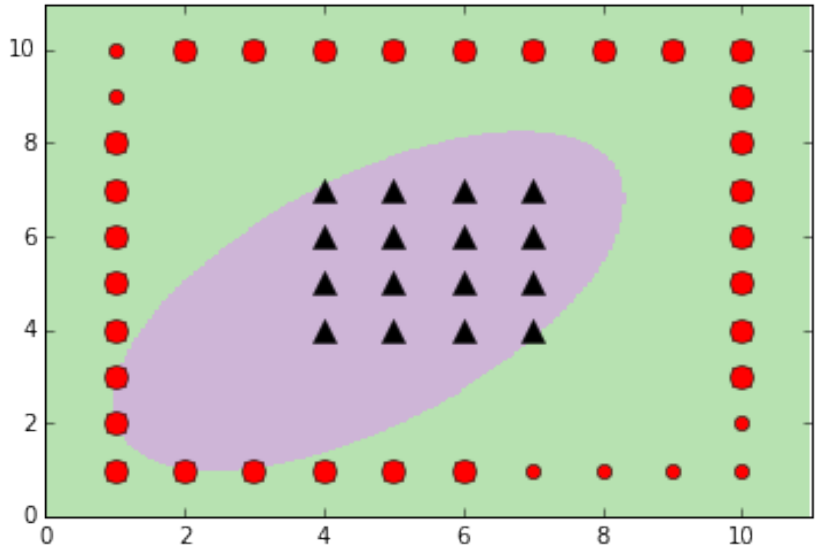
   This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$. Offset b also follows.

3. **Classification.** Given a new point x, classify as

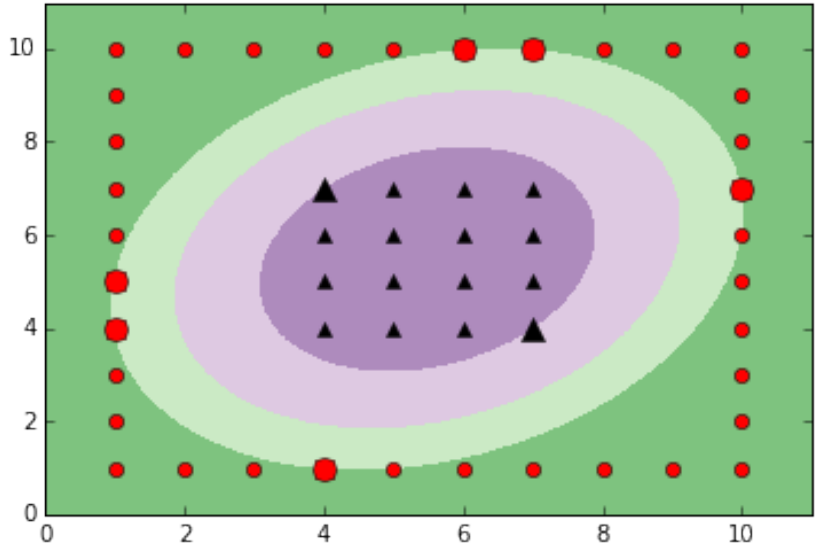
$$\text{sign} \left(\sum_i \alpha_i y^{(i)} (\Phi(x^{(i)}) \cdot \Phi(x)) + b \right).$$

Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

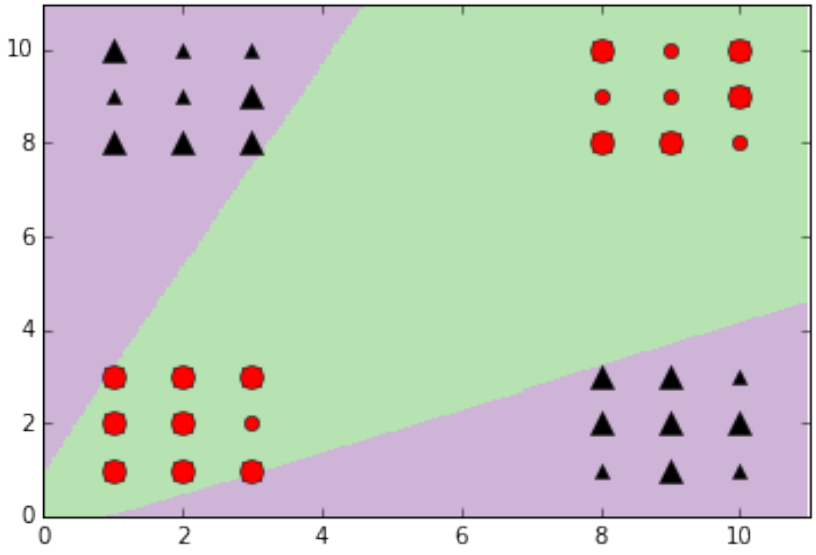


SVM:

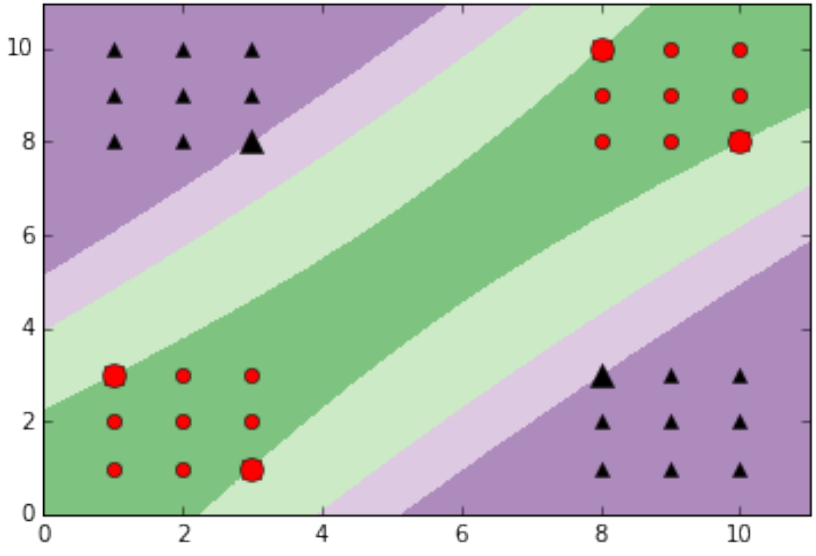


Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

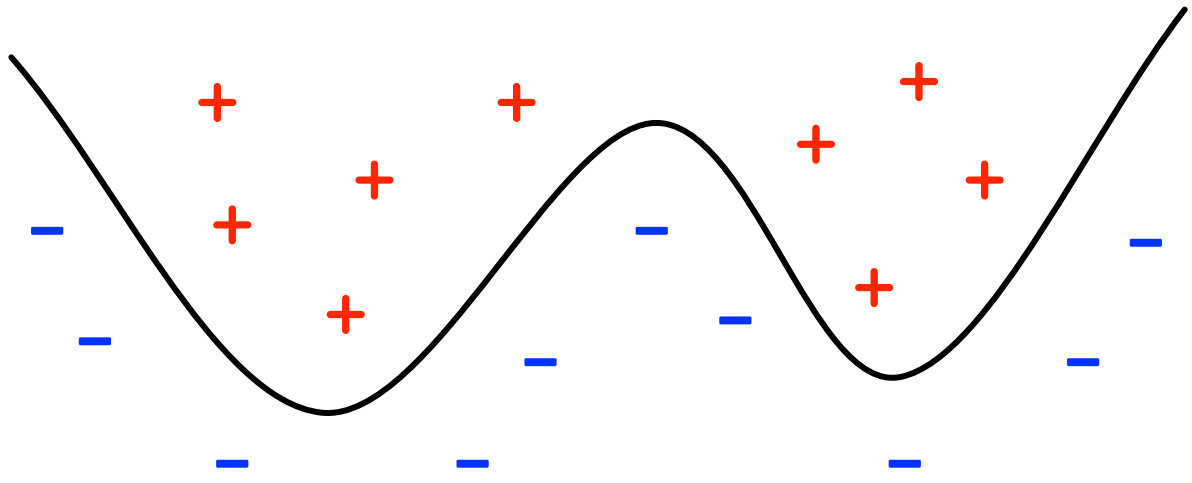


SVM:



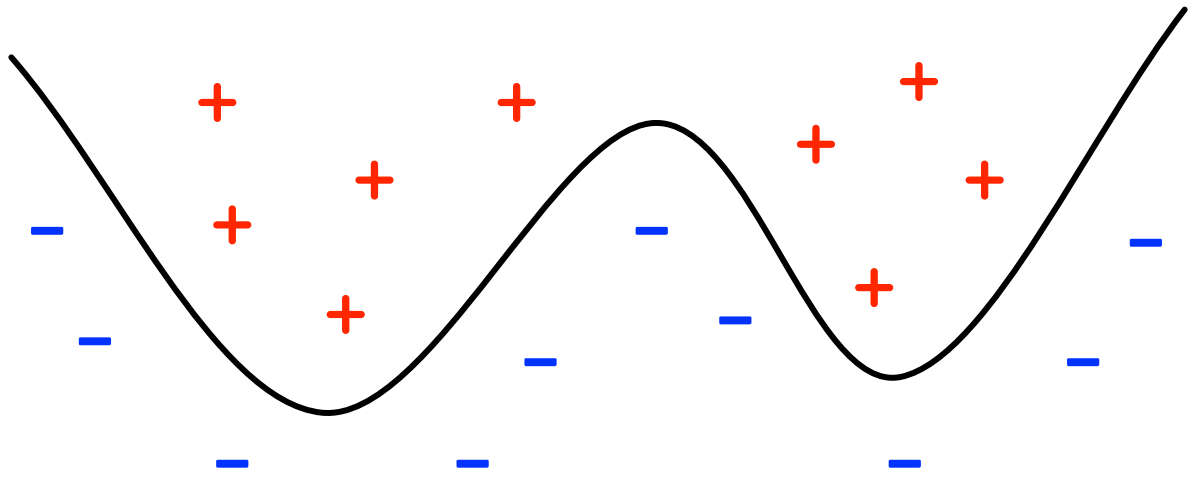
Polynomial decision boundaries

When the decision surface is a polynomial of order p:



Polynomial decision boundaries

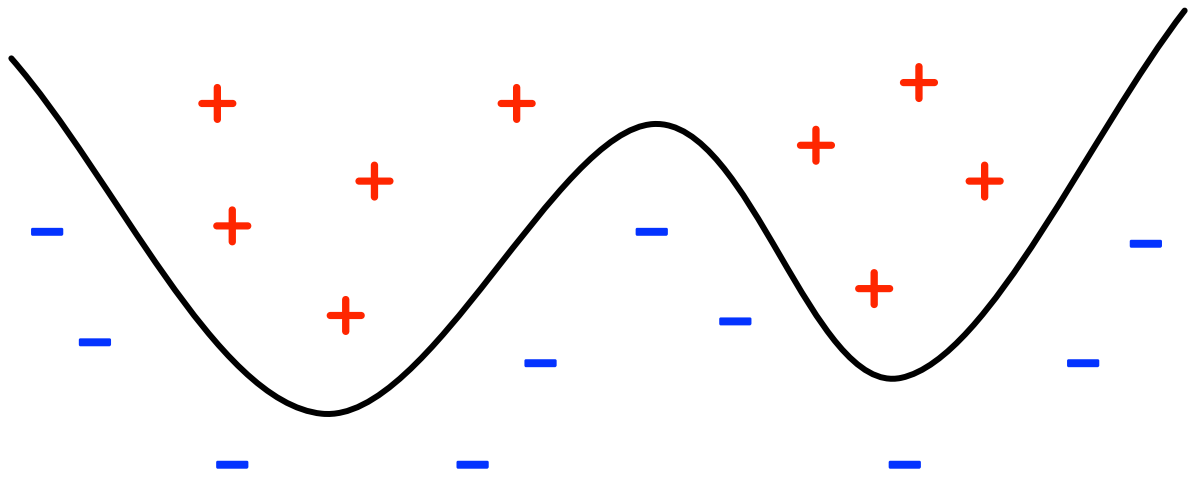
When the decision surface is a polynomial of order p:



- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1 x_2^2 x_3^{p-3}$. (How many such terms are there, roughly?)

Polynomial decision boundaries

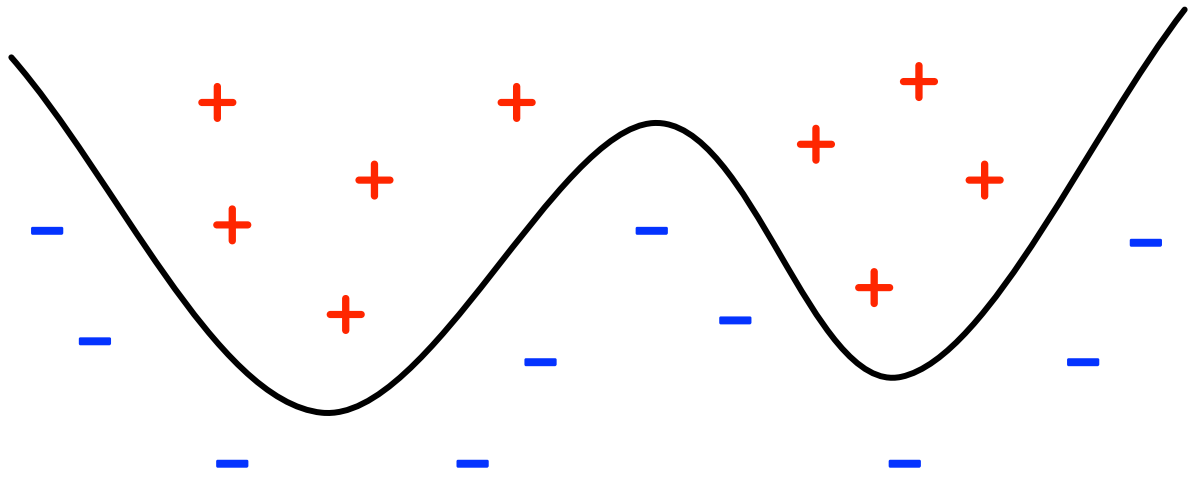
When the decision surface is a polynomial of order p:



- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1 x_2^2 x_3^{p-3}$. (How many such terms are there, roughly?)
- Same trick works: $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p$.

Polynomial decision boundaries

When the decision surface is a polynomial of order p:



- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1 x_2^2 x_3^{p-3}$. (How many such terms are there, roughly?)
- Same trick works: $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p$.

The **kernel function**, a similarity measure: $k(x, z) = \Phi(x) \cdot \Phi(z)$.

The kernel function

- We never explicitly construct the embedding $\Phi(x)$
- What we actually use is the **similarity measure** $k(x, z)$

Rewrite learning algorithm and final classifier in terms of k.

The kernel function

- We never explicitly construct the embedding $\Phi(x)$
- What we actually use is the **similarity measure** $k(x, z)$

Rewrite learning algorithm and final classifier in terms of k.

The classifier $w \cdot \Phi(x)$, for

$$w = \alpha_1 y^{(1)} \Phi(x^{(1)}) + \cdots + \alpha_n y^{(n)} \Phi(x^{(n)}),$$

becomes a **similarity-weighted vote**,

$$F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x).$$

The kernel function

- We never explicitly construct the embedding $\Phi(x)$
- What we actually use is the **similarity measure** $k(x, z)$

Rewrite learning algorithm and final classifier in terms of k.

The classifier $w \cdot \Phi(x)$, for

$$w = \alpha_1 y^{(1)} \Phi(x^{(1)}) + \cdots + \alpha_n y^{(n)} \Phi(x^{(n)}),$$

becomes a **similarity-weighted vote**,

$$F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x).$$

Can we set k to any similarity function?

Mercer's condition

When is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a valid kernel function?

- When there is an embedding Φ of $\mathcal{X}$ such that $k(x, z) = \Phi(x) \cdot \Phi(z)$.

Mercer's condition

When is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a valid kernel function?

- When there is an embedding Φ of $\mathcal{X}$ such that $k(x, z) = \Phi(x) \cdot \Phi(z)$.

- **Mercer's condition**: Equivalently, for any finite subset $\{x^{(1)}, \ldots, x^{(m)}\} \subset \mathcal{X}$, the $m \times m$ similarity matrix

$$K_{ij} = k(x^{(i)}, x^{(j)})$$

  is positive semidefinite.

Mercer's condition

When is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a valid kernel function?

- When there is an embedding Φ of $\mathcal{X}$ such that $k(x, z) = \Phi(x) \cdot \Phi(z)$.
- **Mercer's condition**: Equivalently, for any finite subset $\{x^{(1)}, \ldots, x^{(m)}\} \subset \mathcal{X}$, the $m \times m$ similarity matrix

$$K_{ij} = k(x^{(i)}, x^{(j)})$$

  is positive semidefinite.

Do you see why this matrix is p.s.d. if $k(x, z) = \Phi(x) \cdot \Phi(z)$?

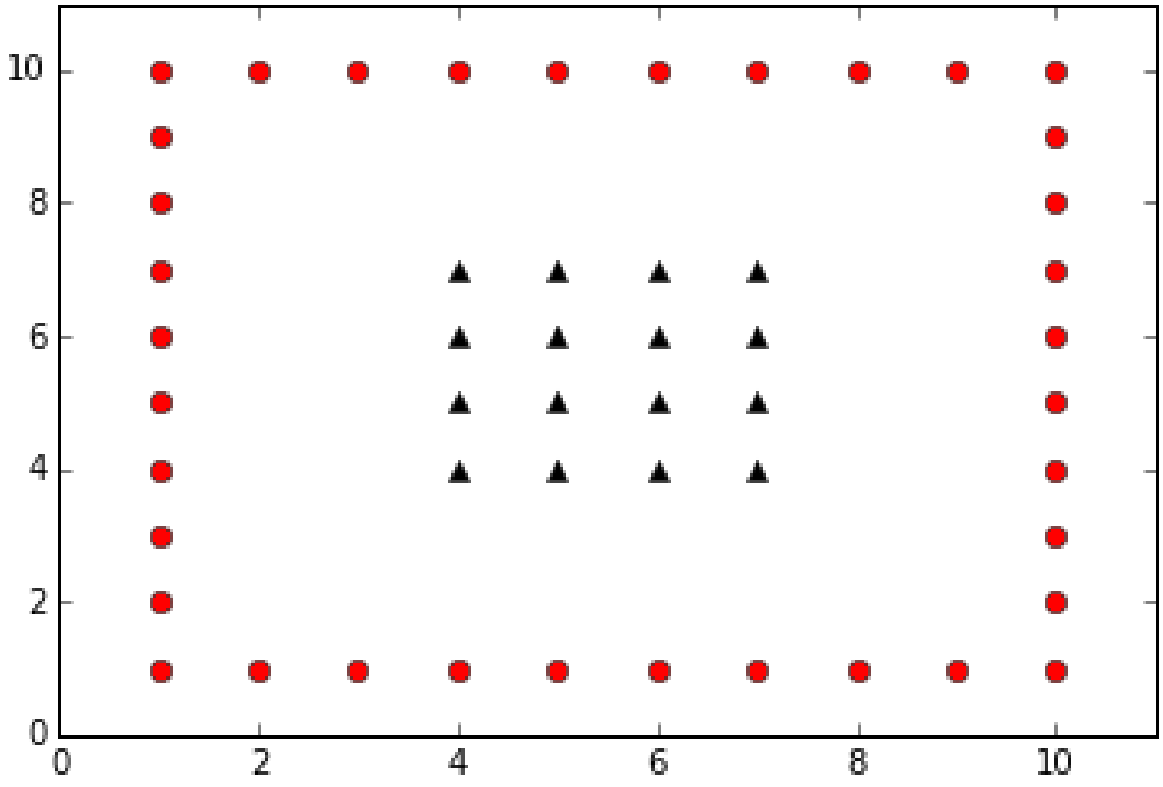
The RBF kernel

A popular similarity function: the **Gaussian kernel** or **RBF kernel**

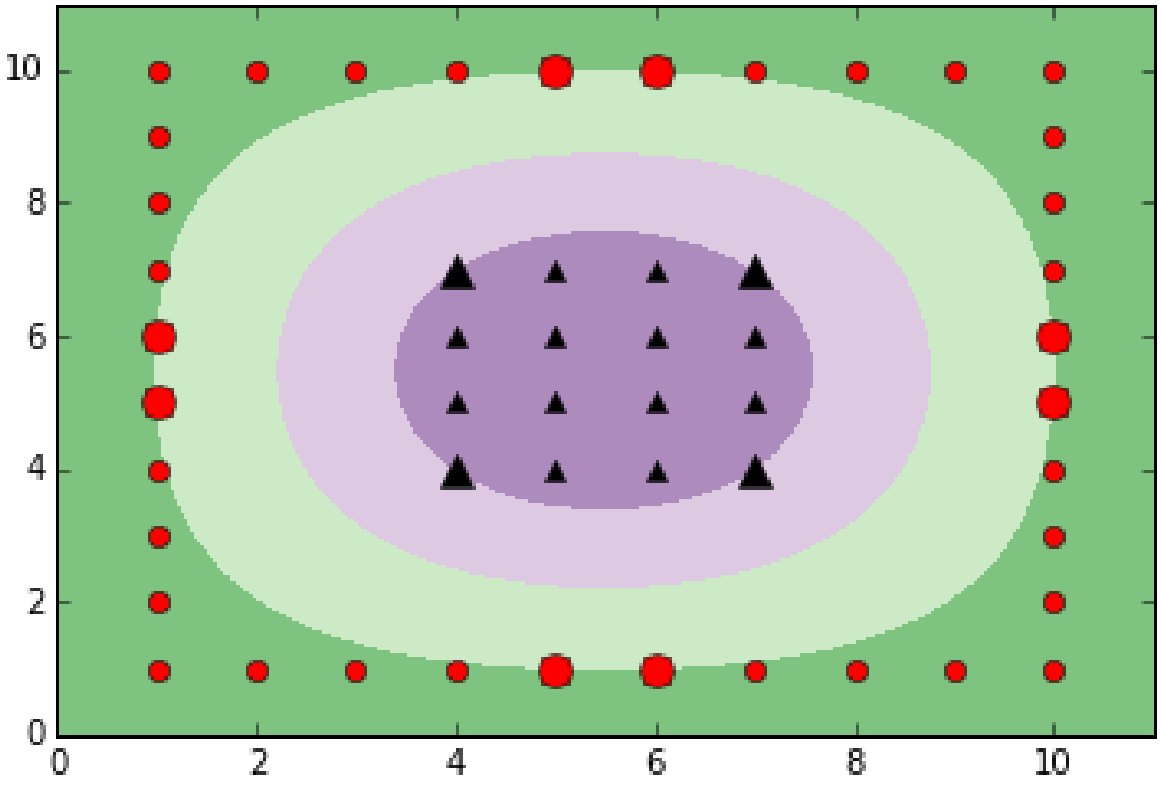
$$k(x, z) = e^{-\|x-z\|^2/2\sigma^2},$$

where σ is an adjustable scale parameter.

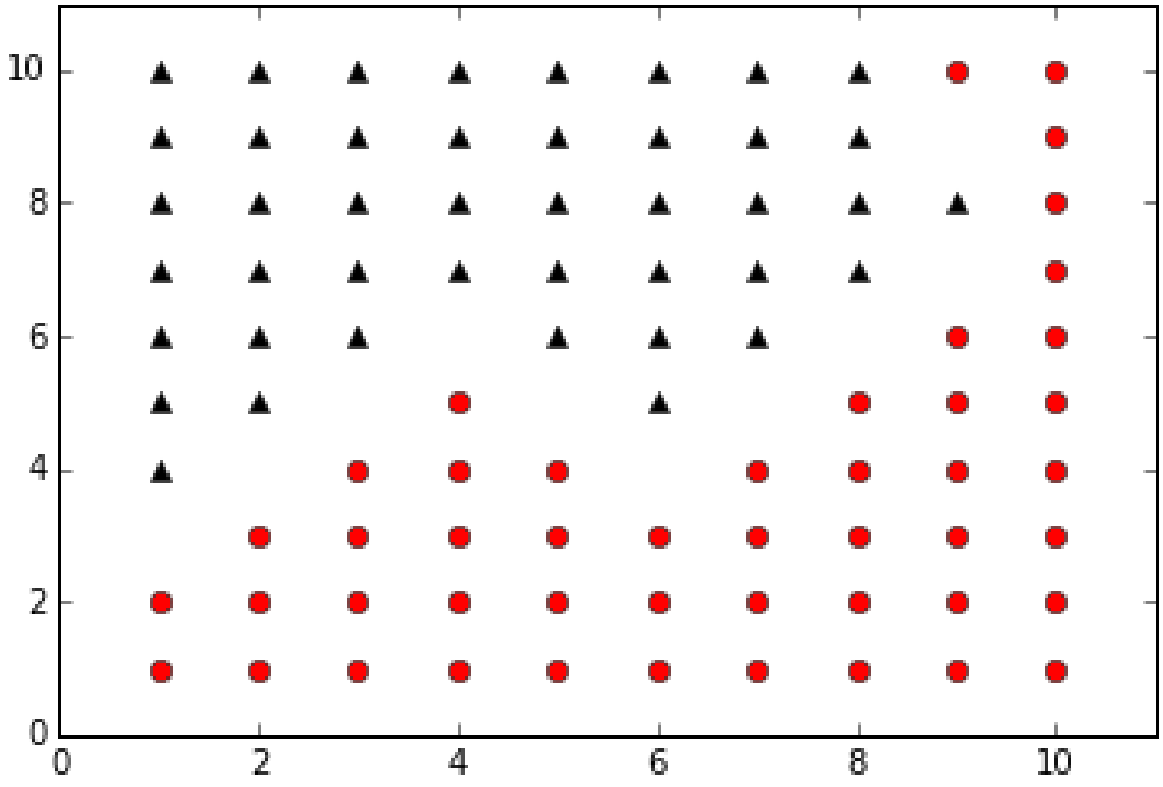
RBF kernel: examples



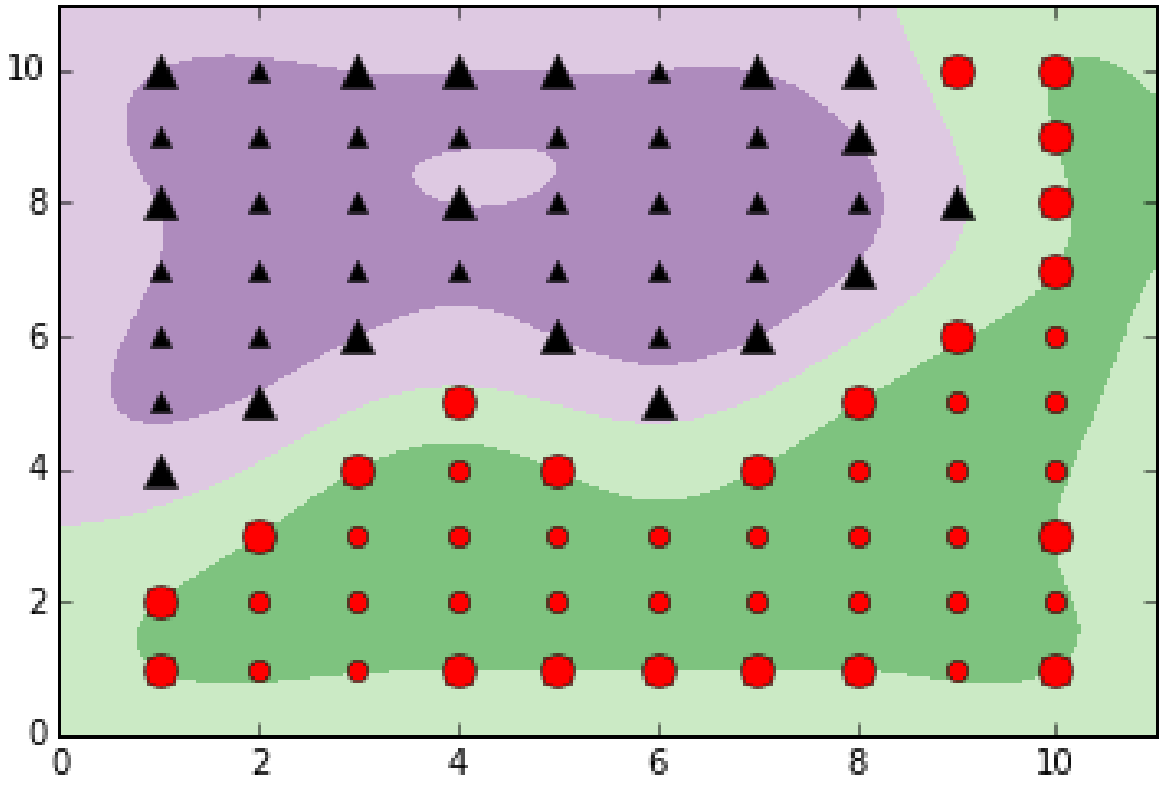
RBF kernel: examples



RBF kernel: examples



RBF kernel: examples



Question

RBF kernel: $k(x, z) = e^{-\|x-z\|^2/2\sigma^2}$.

1. How does this function behave as $\sigma \uparrow \infty$?
2. How does this function behave as $\sigma \downarrow 0$?
3. As we get more data, should we increase or decrease σ?

Kernels: postscript

1. Customized kernels
   - For different domains (NLP, biology, speech, ...)
   - And different structures (sequences, trees, graphs, ...)

2. Learning the kernel function
   Given a set of plausible kernels, find a linear combination of them that works well.

3. Speeding up learning and prediction
   The $n \times n$ kernel matrix $k(x^{(i)}, x^{(j)})$ is a bottleneck for large n. Various techniques for dealing with this.