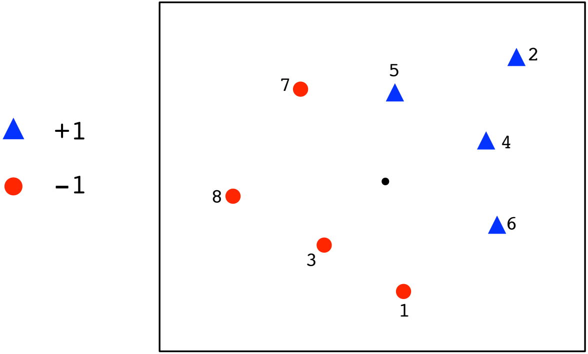
Kernels

DSE 220

Recall: Perceptron

Input space $\mathcal{X} = \mathbb{R}^p$, label space $\mathcal{Y} = \{-1, 1\}$

- $w = 0$
- while some (x, y) is misclassified:
  - $w = w + yx$

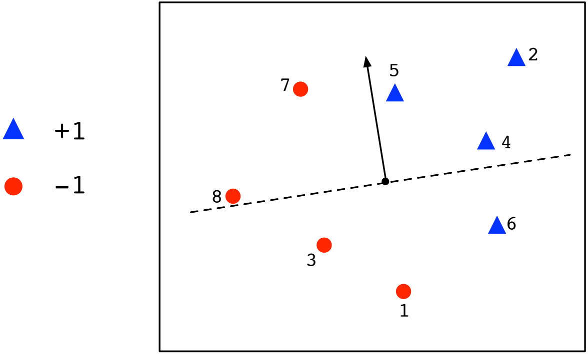


Separator: $w = 0$

Recall: Perceptron

Input space $\mathcal{X} = \mathbb{R}^p$, label space $\mathcal{Y} = \{-1, 1\}$

- $w = 0$
- while some (x, y) is misclassified:
    - $w = w + yx$

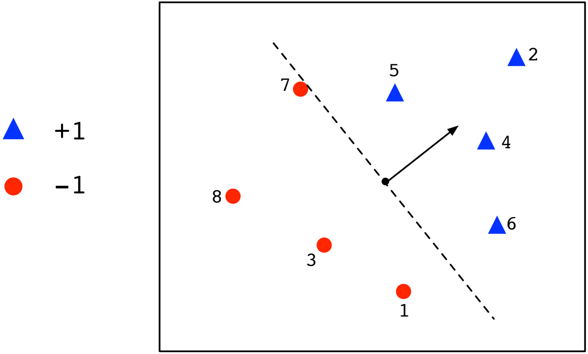


Separator: $w = -x^{(1)}$

Recall: Perceptron

Input space $\mathcal{X} = \mathbb{R}^p$, label space $\mathcal{Y} = \{-1, 1\}$

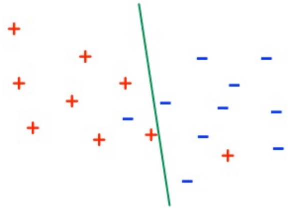
- $w = 0$
- while some (x, y) is misclassified:
  - $w = w + yx$



Separator: $w = -x^{(1)} + x^{(6)}$

Deviations from linear separability

Noise

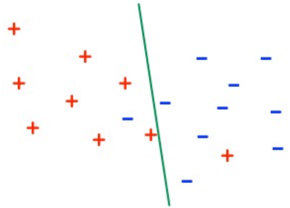


Find a separator that minimizes a convex loss function related to the number of mistakes.

e.g. SVM, logistic regression.

Deviations from linear separability

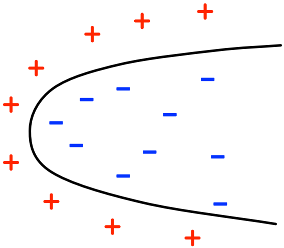
Noise



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

e.g. SVM, logistic regression.

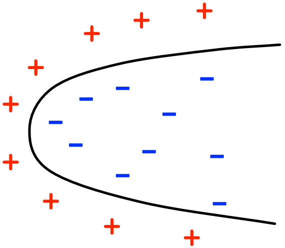
Systematic deviation



What to do with this?

Systematic inseparability

In this case, the actual boundary looks quadratic.



Quick fix: in addition to the regular features $x = (x_1, x_2, \ldots, x_p)$, add in extra features:

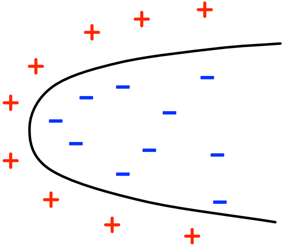
$$x_1^2,\ x_2^2,\ \ldots,\ x_p^2$$

$$x_1x_2,\ x_1x_3,\ \ldots,\ x_{p-1}x_p$$

The new, enhanced data vectors are of the form:

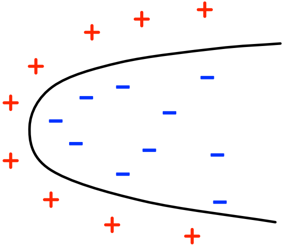
$$\Phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_p, x_1^2, \ldots, x_p^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{p-1}x_p).$$

Adding new features



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

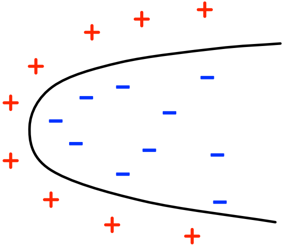
Adding new features



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

- This is quadratic in $x = (1, x_1, x_2)$
- But it is linear in $\Phi(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Adding new features



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

- This is quadratic in $x = (1, x_1, x_2)$
- But it is linear in $\Phi(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

By embedding the data in a higher-dimensional feature space, we can keep using a linear classifier!

Quick quiz

1. Suppose $x = (1, x_1, x_2, x_3)$. What is the dimension of $\Phi(x)$?

Quick quiz

1. Suppose $x = (1, x_1, x_2, x_3)$. What is the dimension of $\Phi(x)$?
2. What if $x = (1, x_1, \ldots, x_p)$?

Perceptron revisited

Learning in the higher-dimensional feature space:

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

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

Perceptron revisited

Learning in the higher-dimensional feature space:

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

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!

Perceptron revisited

Learning in the higher-dimensional feature space:

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

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** (Aizenman, Braverman, Rozonoer (1964)):

- The only time we ever access $\Phi(x)$ is 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 $\Phi(x) \cdot \Phi(x^{(i)})$.

Perceptron revisited

Learning in the higher-dimensional feature space:

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

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** (Aizenman, Braverman, Rozonoer (1964)):

- The only time we ever access $\Phi(x)$ is 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 $\Phi(x) \cdot \Phi(x^{(i)})$.
- Can we compute such dot products without writing out the $\Phi(x)$'s?

Computing dot products

In 2-d:

$$
\begin{aligned}
&\Phi(x) \cdot \Phi(z) \\
&= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1z_2) \\
&= 1 + 2x_1z_1 + 2x_2z_2 + x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\
&= (1 + x_1z_1 + x_2z_2)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}
$$

Computing dot products

In 2-d:

$$
\begin{aligned}
&\Phi(x) \cdot \Phi(z) \\
&= (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1z_2) \\
&= 1 + 2x_1z_1 + 2x_2z_2 + x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\
&= (1 + x_1z_1 + x_2z_2)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}
$$

In p dimensions:

$$
\begin{aligned}
&\Phi(x) \cdot \Phi(z) \\
&= (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_p, x_1^2, \ldots, x_p^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{p-1}x_p) \cdot \\
&\quad (1, \sqrt{2}z_1, \ldots, \sqrt{2}z_p, z_1^2, \ldots, z_p^2, \sqrt{2}z_1z_2, \ldots, \sqrt{2}z_{p-1}z_p) \\
&= 1 + 2\sum_i x_iz_i + \sum_i x_i^2z_i^2 + 2\sum_{i \neq j} x_ix_jz_iz_j \\
&= (1 + x_1z_1 + \cdots + x_pz_p)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}
$$

Computing dot products

In 2-d:

$$\begin{aligned}
&\Phi(x)\cdot\Phi(z)\\
&=(1,\sqrt{2}x_1,\sqrt{2}x_2,x_1^2,x_2^2,\sqrt{2}x_1x_2)\cdot(1,\sqrt{2}z_1,\sqrt{2}z_2,z_1^2,z_2^2,\sqrt{2}z_1z_2)\\
&=1+2x_1z_1+2x_2z_2+x_1^2z_1^2+x_2^2z_2^2+2x_1x_2z_1z_2\\
&=(1+x_1z_1+x_2z_2)^2 \quad = \quad (1+x\cdot z)^2
\end{aligned}$$

In p dimensions:

$$\begin{aligned}
&\Phi(x)\cdot\Phi(z)\\
&=(1,\sqrt{2}x_1,\ldots,\sqrt{2}x_p,x_1^2,\ldots,x_p^2,\sqrt{2}x_1x_2,\ldots,\sqrt{2}x_{p-1}x_p)\cdot\\
&\quad(1,\sqrt{2}z_1,\ldots,\sqrt{2}z_p,z_1^2,\ldots,z_p^2,\sqrt{2}z_1z_2,\ldots,\sqrt{2}z_{p-1}z_p)\\
&=1+2\sum_i x_iz_i+\sum_i x_i^2z_i^2+2\sum_{i\neq j}x_ix_jz_iz_j\\
&=(1+x_1z_1+\cdots+x_pz_p)^2 \quad = \quad (1+x\cdot z)^2
\end{aligned}$$

For MNIST: computing dot products in the 307720-dimensional quadratic feature space takes time proportional to just 784, the original dimension!

The kernel trick

Why does it work?

The kernel trick

Why does it work?

1. The only time we ever look at data – during training or subsequent classification – is to compute dot products $w \cdot \Phi(x)$.

The kernel trick

Why does it work?

1. The only time we ever look at data – during training or subsequent classification – 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 $[(1, \alpha_1), (22, \alpha_{22}), (37, \alpha_{37})]$.

The kernel trick

Why does it work?

1. The only time we ever look at data – during training or subsequent classification – 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, without ever writing 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:

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

Kernel Perceptron

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

Primal form:

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

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

- $\alpha = 0$
- while there is some i with $y^{(i)}\left(\sum_j \alpha_j y^{(j)} \underbrace{\Phi(x^{(j)}) \cdot \Phi(x^{(i)})}_{\text{efficient}} \right) < 0$:
  - $\alpha_i = \alpha_i + 1$

Kernel Perceptron

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

Primal form:

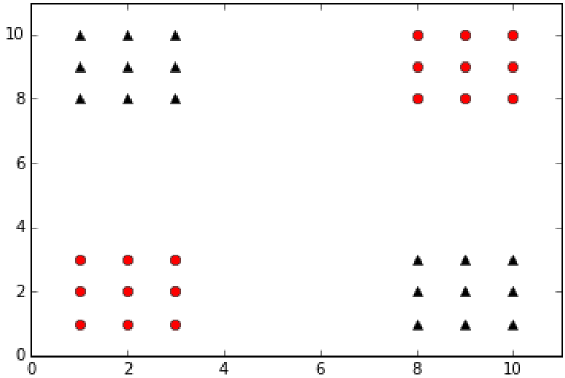
- $w = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)})) < 0$:
  - $w = w + y^{(i)}\,\Phi(x^{(i)})$

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

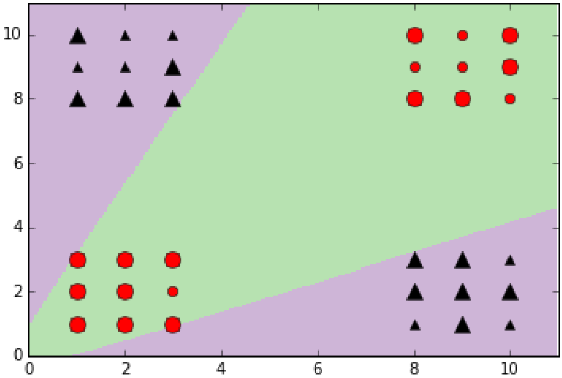
- $\alpha = 0$
- while there is some i with $y^{(i)}\left(\sum_j \alpha_j y^{(j)} \underbrace{\Phi(x^{(j)}) \cdot \Phi(x^{(i)})}_{\text{efficient}} \right) < 0$:
  - $\alpha_i = \alpha_i + 1$

To classify a new point x: Return sign $\left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x)\right)$.

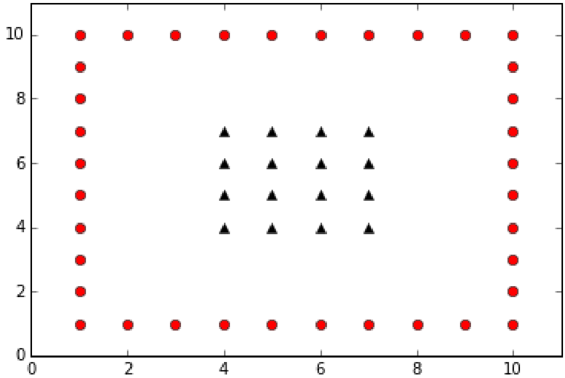
Kernel Perceptron: Examples



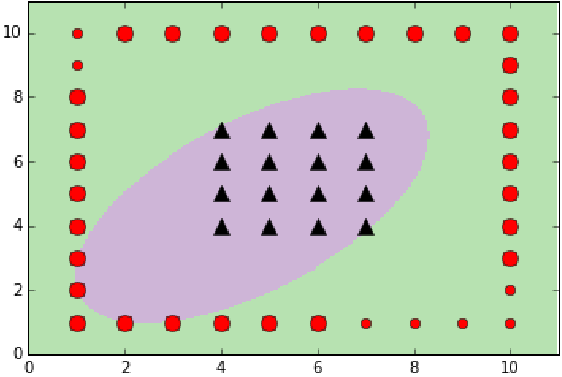
Kernel Perceptron: Examples



Kernel Perceptron: Examples



Kernel Perceptron: Examples



Quick quiz

Recall the kernel perceptron algorithm:

- $\alpha = 0$
- while there is some i with $y^{(i)}(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)})) < 0$:
  - $\alpha_i = \alpha_i + 1$

Suppose we run it on a data set and find that it converges after k updates.

1. How many dot product computations are needed to classify a new point?

Quick quiz

Recall the kernel perceptron algorithm:

- $\alpha = 0$
- while there is some i with $y^{(i)}(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)})) < 0$:
  - $\alpha_i = \alpha_i + 1$

Suppose we run it on a data set and find that it converges after k updates.

1. How many dot product computations are needed to classify a new point?
2. What is the total number of dot product computations during learning?

Does this work with SVMs?

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

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

$$\xi \geq 0$$

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

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

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

At optimality, $w = \sum_i \alpha_i y^{(i)} x^{(i)}$, with

$$0 < \alpha_i < C \quad \Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1$$

$$\alpha_i = C \quad \Rightarrow \quad y^{(i)}(w \cdot x^{(i)} + b) = 1 - \xi_i$$

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:

$$\boxed{\begin{gathered}\max_{\alpha\in\mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i\alpha_j y^{(i)} y^{(j)} (\underbrace{\Phi(x^{(i)}) \cdot \Phi(x^{(j)})}_{\text{efficient}}) \\ \text{s.t.:} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \\ 0 \le \alpha_i \le C\end{gathered}}$$

This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$.
Offset b is obtained from the complementary slackness conditions.

Kernel SVM

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

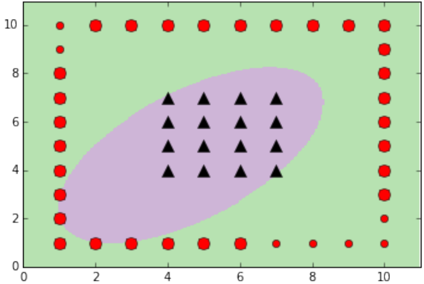
$$\boxed{\begin{gathered}\max_{\alpha\in\mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \underbrace{(\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))}_{\text{efficient}} \\ \text{s.t.:} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \\ 0 \leq \alpha_i \leq C\end{gathered}}$$

   This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$.
   Offset b is obtained from the complementary slackness conditions.
3. **Classification.** Given a new point x, classify as

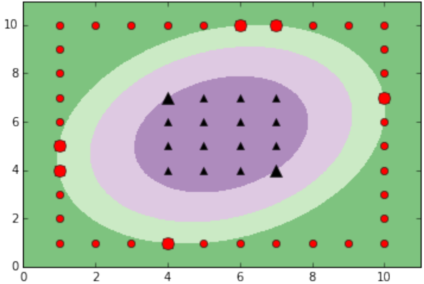
$$\operatorname{sign}\left(\sum_i \alpha_i y^{(i)} \underbrace{(\Phi(x^{(i)}) \cdot \Phi(x))}_{\text{again}} + b \right).$$

Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

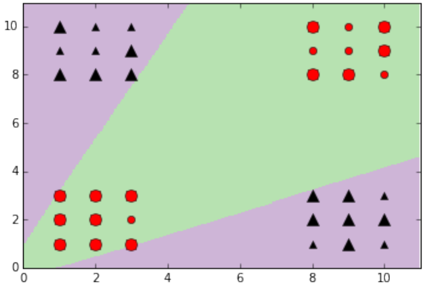


SVM:

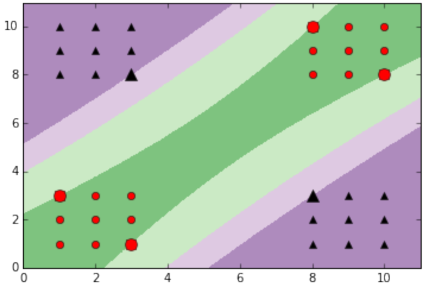


Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

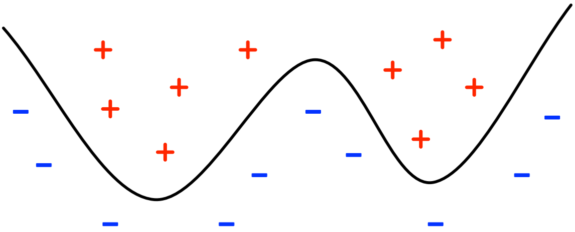


SVM:



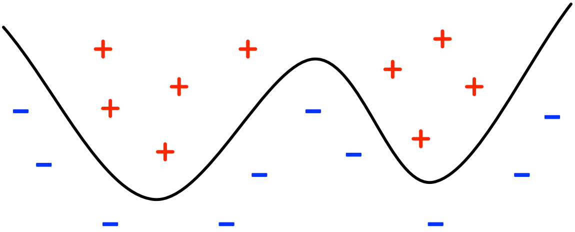
Polynomial decision boundaries

To get a decision surface which is an arbitrary polynomial of order d:



Polynomial decision boundaries

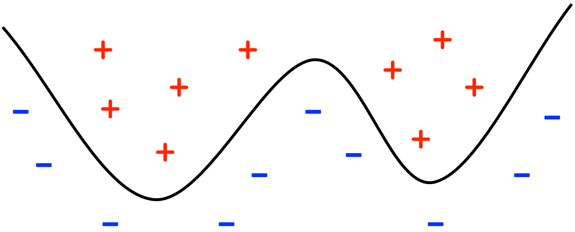
To get a decision surface which is an arbitrary polynomial of order d:



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

Polynomial decision boundaries

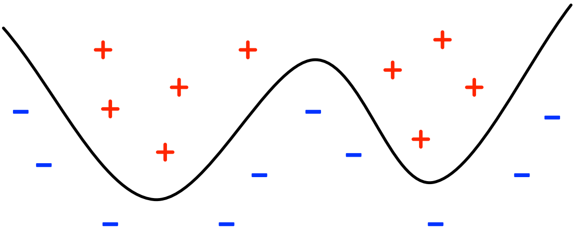
To get a decision surface which is an arbitrary polynomial of order d:



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

Polynomial decision boundaries

To get a decision surface which is an arbitrary polynomial of order d:



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

The **kernel function**, a measure of similarity:

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

String kernels

Sequence data:

- text documents
- speech signals
- protein sequences

String kernels

Sequence data:

- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

$$\mathcal{X} = \{A, C, G, T\}^*$$
$$\mathcal{X} = \{\text{English words}\}^*$$

String kernels

Sequence data:

- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

$$\mathcal{X} = \{A, C, G, T\}^*$$
$$\mathcal{X} = \{\text{English words}\}^*$$

What kind of embedding $\Phi(x)$ is suitable for variable-length sequences x?

String kernels

Sequence data:

- text documents
- speech signals
- protein sequences

Each data point is a sequence of arbitrary length. This yields input spaces like:

$$\mathcal{X} = \{A, C, G, T\}^*$$
$$\mathcal{X} = \{\text{English words}\}^*$$

What kind of embedding $\Phi(x)$ is suitable for variable-length sequences x?

We will use an infinite-dimensional embedding!

String kernels, cont'd

For each substring s, define *feature*:

$$\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each possible string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

String kernels, cont'd

For each substring s, define *feature*:

$$\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each possible string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

Example: the embedding of "aardvark" includes features

$$\Phi_{\text{ar}}(\text{aardvark}) = 2,\ \Phi_{\text{th}}(\text{aardvark}) = 0, \ldots$$

A linear classifier based on such features is potentially very powerful.

String kernels, cont'd

For each substring s, define *feature*:

$$\Phi_s(x) = \#\text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each possible string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

Example: the embedding of "aardvark" includes features

$$\Phi_{\text{ar}}(\text{aardvark}) = 2,\ \Phi_{\text{th}}(\text{aardvark}) = 0, \ldots$$

A linear classifier based on such features is potentially very powerful.

We can compute dot products fast! To compute $k(x, z) = \Phi(x) \cdot \Phi(z)$:

for each substring s of x: count how often s appears in z

String kernels, cont'd

For each substring s, define *feature*:

$$\Phi_s(x) = \# \text{ of times substring } s \text{ appears in } x$$

and let $\Phi(x)$ be a vector with one coordinate for each possible string:

$$\Phi(x) = (\Phi_s(x) : \text{all strings } s).$$

Example: the embedding of "aardvark" includes features

$$\Phi_{\text{ar}}(\text{aardvark}) = 2, \; \Phi_{\text{th}}(\text{aardvark}) = 0, \ldots$$

A linear classifier based on such features is potentially very powerful.

We can compute dot products fast! To compute $k(x, z) = \Phi(x) \cdot \Phi(z)$:

for each substring s of x: count how often s appears in z

Using dynamic programming, this takes time $O(|x| \cdot |z|)$.

The kernel function

Now shift attention:

- away from the embedding $\Phi(x)$, which we never explicitly construct,
- towards the thing we actually use: the **similarity measure** $k(x, z)$

Rewrite learning algorithm and final classifier in terms of k.

The kernel function

Now shift attention:

- away from the embedding $\Phi(x)$, which we never explicitly construct,
- towards the thing we actually use: 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

Now shift attention:

- away from the embedding $\Phi(x)$, which we never explicitly construct,
- towards the thing we actually use: 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 choose k to be any similarity function suitable for the application domain?

Quick quiz

For each of the following kernel functions, find a corresponding embedding $x \to \Phi(x)$.

1. $k(x, z) = \cos(\text{angle between } x \text{ and } z)$.

Quick quiz

For each of the following kernel functions, find a corresponding embedding $x \to \Phi(x)$.

1. $k(x, z) = \cos(\text{angle between } x \text{ and } z)$.
2. $k(x, z) = x^T Az$, where A is a symmetric positive semidefinite matrix.

Mercer's condition

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a valid kernel function if it corresponds to some embedding: that is, if there exists Φ defined on $\mathcal{X}$ such that

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

Mercer's condition

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a valid kernel function if it corresponds to some embedding: that is, if there exists Φ defined on $\mathcal{X}$ such that

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

Mercer (1909): This is equivalent to requiring that 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

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a valid kernel function if it corresponds to some embedding: that is, if there exists Φ defined on $\mathcal{X}$ such that

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

Mercer (1909): This is equivalent to requiring that 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 $k(x, z) = \Phi(x) \cdot \Phi(z)$?

Mercer's condition

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a valid kernel function if it corresponds to some embedding: that is, if there exists Φ defined on $\mathcal{X}$ such that

$$k(x, z) = \Phi(x) \cdot \Phi(z).$$

Mercer (1909): This is equivalent to requiring that 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.

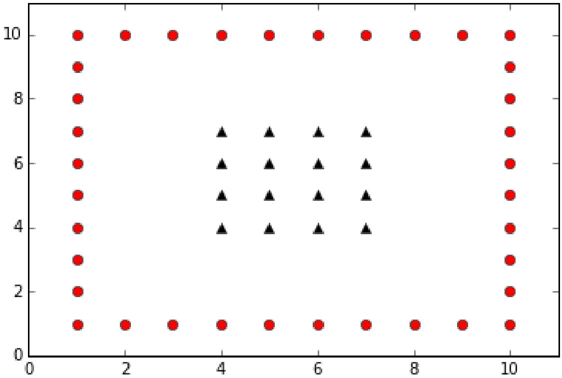
Do you see why this matrix is p.s.d. if $k(x, z) = \Phi(x) \cdot \Phi(z)$?

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

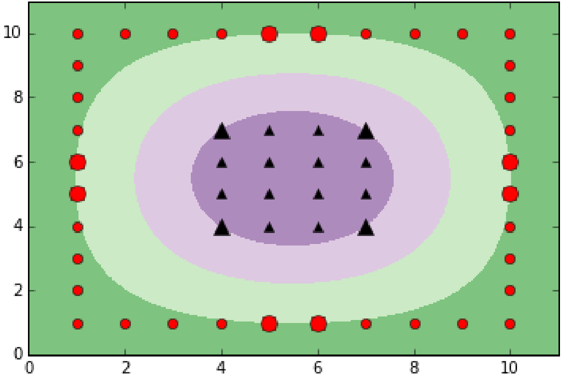
$$k(x, z) = e^{-\|x-z\|^2/2\sigma^2},$$

where σ is an adjustable scale parameter.

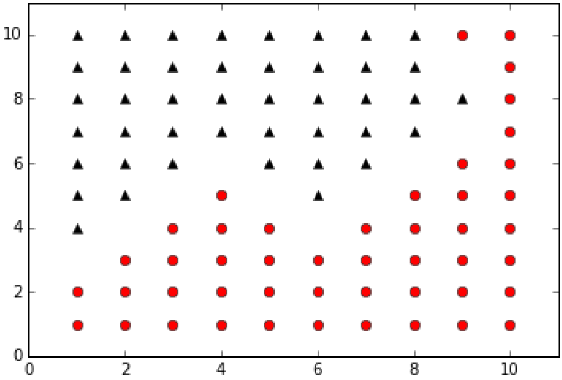
RBF kernel: examples



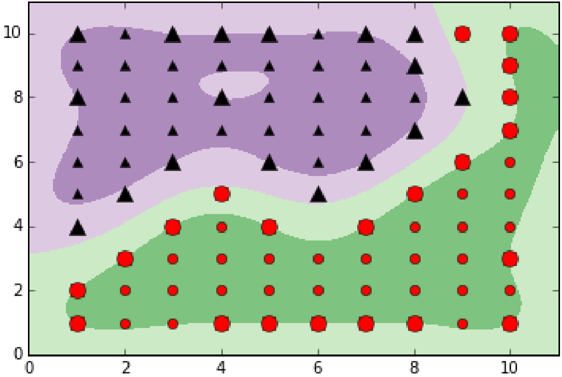
RBF kernel: examples



RBF kernel: examples



RBF kernel: examples



Quick quiz

Recall that the RBF kernel is given by $k(x, z) = e^{-\|x-z\|^2/2\sigma^2}$.

1. How does this function behave as $\sigma \uparrow \infty$?

Quick quiz

Recall that the RBF kernel is given by $k(x, z) = e^{-\|x-z\|^2/2\sigma^2}$.

1. How does this function behave as $\sigma \uparrow \infty$?
2. How does this function behave as $\sigma \downarrow 0$?

Quick quiz

Recall that the RBF kernel is given by $k(x,z) = e^{-\|x-z\|^2/2\sigma^2}$.

1. How does this function behave as $\sigma \uparrow \infty$?
2. How does this function behave as $\sigma \downarrow 0$?
3. As the amount of data increases, would it make sense to increase σ or to decrease it?

Kernels: postscript

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

Kernels: postscript

1. Customized kernels
   - For many different domains (NLP, biology, speech, ...)
   - Over many different structures (sequences, sets, trees, graphs, ...)
2. Learning the kernel function
   Given a set of plausible kernels, find a linear combination of them that works well.

Kernels: postscript

1. Customized kernels
   - For many different domains (NLP, biology, speech, ...)
   - Over many different structures (sequences, sets, 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.
   One idea:
   - Go back to the primal space!
   - Replace the embedding Φ by a low-dimensional mapping $\widetilde{\Phi}$ such that

   $$\widetilde{\Phi}(x) \cdot \widetilde{\Phi}(z) \approx \Phi(x) \cdot \Phi(z).$$

   This can be done, for instance, by writing Φ in the Fourier basis and then randomly sampling features.