Kernel Methods & SVM

Partially based on the ML lecture by Raymond J. Mooney
University of Texas at Austin

Back to Linear Classifier (Slightly Modified)

A linear classifier $h[\vec{w}]$ is determined by a vector of weights $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

Back to Linear Classifier (Slightly Modified)

A linear classifier $h[\vec{w}]$ is determined by a vector of weights $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

Given $\vec{x} = (x_1, \ldots, x_n) \in X \subseteq \mathbb{R}^n$,

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \\ -1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i < 0 \end{cases}$$

For convenience, we use values $\{-1, 1\}$ instead of $\{0, 1\}$. Note that this is not a principal modification, the linear classifier works exactly as the original one.

Back to Linear Classifier (Slightly Modified)

A linear classifier $h[\vec{w}]$ is determined by a vector of weights $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

Given $\vec{x} = (x_1, \ldots, x_n) \in X \subseteq \mathbb{R}^n$,

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \\ -1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i < 0 \end{cases}$$

For convenience, we use values $\{-1, 1\}$ instead of $\{0, 1\}$. Note that this is not a principal modification, the linear classifier works exactly as the original one.

Recall that given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the *augmented feature vector* is

$$\tilde{\mathbf{x}} = (x_0, x_1, \ldots, x_n) \quad \text{where } x_0 = 1$$

This makes the notation for the linear classifier more succinct:

$$h[\vec{w}](\vec{x}) = sig(\vec{w} \cdot \tilde{\mathbf{x}}) \text{ where } sig(y) = \begin{cases} 1 & y \geq 0 \\ -1 & y < 0 \end{cases}$$

Perceptron Learning Revisited

- Given a training set

$$D = \{(\vec{x}_1, y(\vec{x}_1)), (\vec{x}_2, y(\vec{x}_2)), \ldots, (\vec{x}_p, y(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y(\vec{x}_k) \in \{-1, 1\}$.

Perceptron Learning Revisited

- Given a training set

$$D = \{(\vec{x}_1, y(\vec{x}_1)), (\vec{x}_2, y(\vec{x}_2)), \dots, (\vec{x}_p, y(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \dots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y(\vec{x}_k) \in \{-1, 1\}$.

We write y_k instead of $y(\vec{x}_k)$.
Note that $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \dots, x_{kn})$ where $x_{k0} = 1$.

Perceptron Learning Revisited

- Given a training set

  $$D = \{(\vec{x}_1, y(\vec{x}_1)), (\vec{x}_2, y(\vec{x}_2)), \ldots, (\vec{x}_p, y(\vec{x}_p))\}$$

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y(\vec{x}_k) \in \{-1, 1\}$.

  We write y_k instead of $y(\vec{x}_k)$.
  Note that $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- A weight vector $\vec{w} \in \mathbb{R}^{n+1}$ is **consistent with** D if

  $$h[\vec{w}](\vec{x}_k) = sig(\vec{w} \cdot \tilde{\mathbf{x}}_k) = y_k \quad \text{for all } k = 1, \ldots, p$$

  D is **linearly separable** if there is a vector $\vec{w} \in \mathbb{R}^{n+1}$ which is consistent with D.

Perceptron Learning Revisited

Perceptron learning algorithm (slightly modified):

Consider training examples cyclically. Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

Perceptron Learning Revisited

Perceptron learning algorithm (slightly modified):

Consider training examples cyclically. Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.

  (This is a slight but harmless modification of the traditional algorithm.)

Perceptron Learning Revisited

Perceptron learning algorithm (slightly modified):

Consider training examples cyclically. Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
  (This is a slight but harmless modification of the traditional algorithm.)
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:
  - If $sig(\vec{w} \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $\vec{w}^{(t+1)} = \vec{w}^{(t)} + y_k \cdot \tilde{\mathbf{x}}_k$.
  - Otherwise, $\vec{w}^{(t+1)} = \vec{w}^{(t)}$.

  Here $k = (t \mod p) + 1$, i.e. the examples are considered cyclically.

(Note that this algorithm corresponds to the perceptron learning with the learning speed $\varepsilon = 1$.)

Perceptron Learning Revisited

Perceptron learning algorithm (slightly modified):

Consider training examples cyclically. Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.

  (This is a slight but harmless modification of the traditional algorithm.)
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:
  - If $sig(\vec{w} \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $\vec{w}^{(t+1)} = \vec{w}^{(t)} + y_k \cdot \tilde{\mathbf{x}}_k$.
  - Otherwise, $\vec{w}^{(t+1)} = \vec{w}^{(t)}$.

  Here $k = (t \mod p) + 1$, i.e. the examples are considered cyclically.

(Note that this algorithm corresponds to the perceptron learning with the learning speed $\varepsilon = 1$.)

We know: if D is linearly separable, then there is t^* such that $\vec{w}^{(t^*)}$ is consistent with D.

Perceptron Learning Revisited

Perceptron learning algorithm (slightly modified):

Consider training examples cyclically. Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.

  (This is a slight but harmless modification of the traditional algorithm.)
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:
  - If $sig(\vec{w} \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $\vec{w}^{(t+1)} = \vec{w}^{(t)} + y_k \cdot \tilde{\mathbf{x}}_k$.
  - Otherwise, $\vec{w}^{(t+1)} = \vec{w}^{(t)}$.

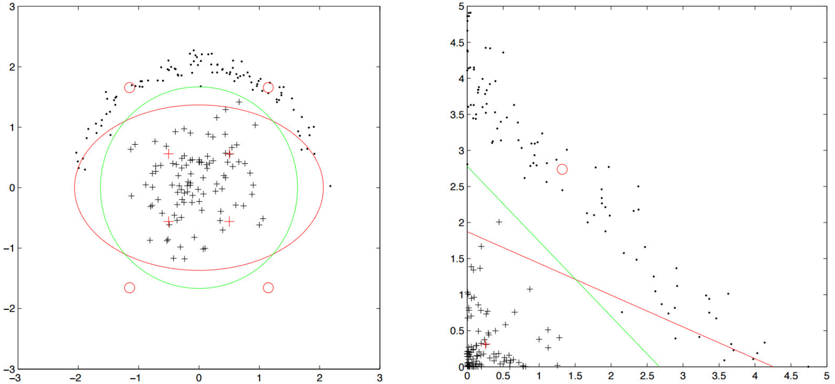
  Here $k = (t \mod p) + 1$, i.e. the examples are considered cyclically.

(Note that this algorithm corresponds to the perceptron learning with the learning speed $\varepsilon = 1$.)

We know: if D is linearly separable, then there is t^* such that $\vec{w}^{(t^*)}$ is consistent with D.

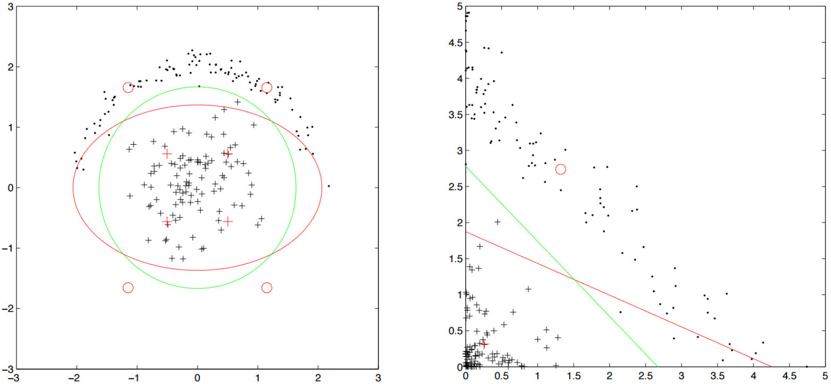
But what can we do if D is not linearly separable?

Quadratic Decision Boundary



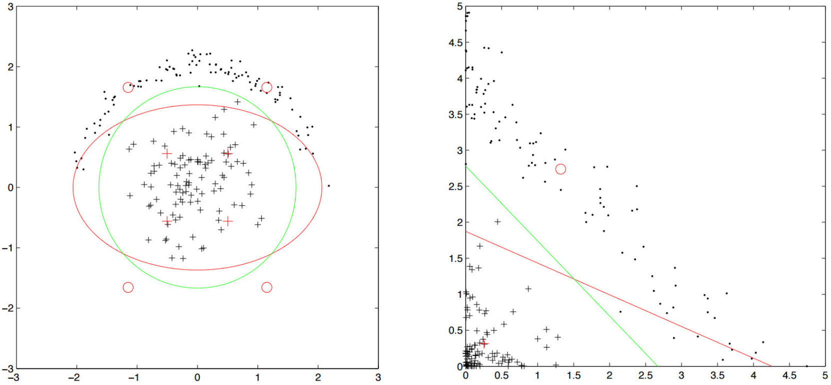
Left: The original set,

Quadratic Decision Boundary



Left: The original set, Right: Transformed using the square of features.

Quadratic Decision Boundary

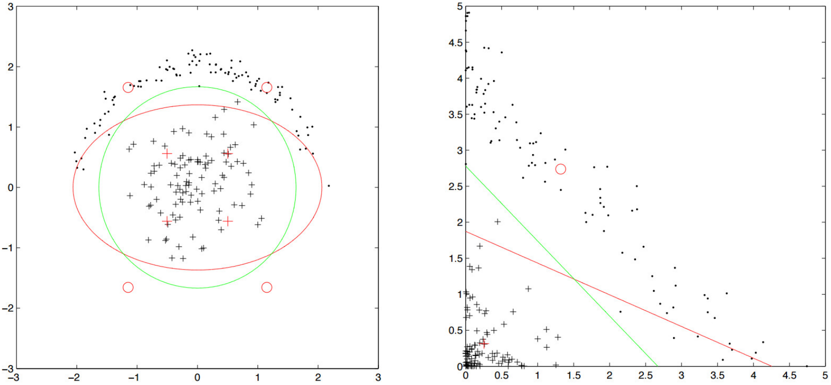


Left: The original set, Right: Transformed using the square of features.

Right: the green line is the decision boundary learned using the perceptron algorithm.

(The red boundary corresponds to another learning algorithm.)

Quadratic Decision Boundary



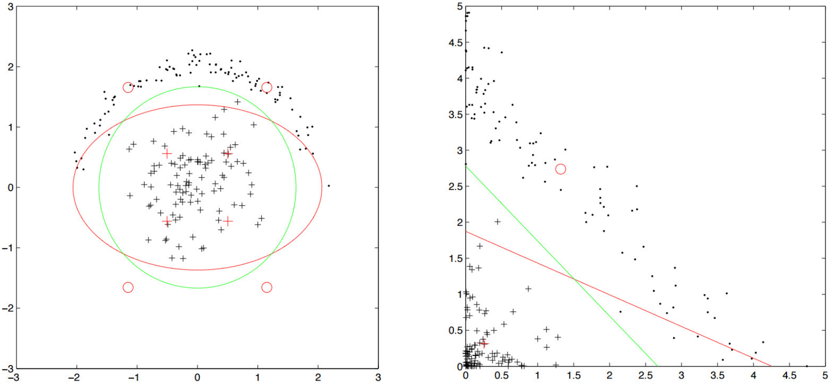
Left: The original set, Right: Transformed using the square of features.

Right: the green line is the decision boundary learned using the perceptron algorithm.

(The red boundary corresponds to another learning algorithm.)

Left: the green ellipse maps exactly to the green line.

Quadratic Decision Boundary



Left: The original set, Right: Transformed using the square of features.

Right: the green line is the decision boundary learned using the perceptron algorithm.

(The red boundary corresponds to another learning algorithm.)

Left: the green ellipse maps exactly to the green line.

How to classify (in the original space): First, transform a given feature vector by squaring the features, then use the linear classifier.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Sometimes its even beneficial to map to infinite-dimensional spaces.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Sometimes its even beneficial to map to infinite-dimensional spaces.

To avoid explicit construction of the higher dimensional feature space, we use so called *kernel trick*.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Sometimes its even beneficial to map to infinite-dimensional spaces.

To avoid explicit construction of the higher dimensional feature space, we use so called *kernel trick*.

But first we need to *dualize* our learning algorithm.

Perceptron Learning Revisited

Perceptron learning algorithm once more:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

Perceptron Learning Revisited

Perceptron learning algorithm once more:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.

Perceptron Learning Revisited

Perceptron learning algorithm once more:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:
  - If $sig(\vec{w} \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $\vec{w}^{(t+1)} = \vec{w}^{(t)} + y_k \cdot \tilde{\mathbf{x}}_k$.
  - Otherwise, $\vec{w}^{(t+1)} = \vec{w}^{(t)}$.

  Here $k = (t \mod p) + 1$, i.e. the examples are considered cyclically.

Perceptron Learning Revisited

Perceptron learning algorithm once more:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:
  - If $sig(\vec{w} \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $\vec{w}^{(t+1)} = \vec{w}^{(t)} + y_k \cdot \tilde{\mathbf{x}}_k$.
  - Otherwise, $\vec{w}^{(t+1)} = \vec{w}^{(t)}$.

  Here $k = (t \mod p) + 1$, i.e. the examples are considered cyclically.

Crucial observation:

Note that $\vec{w}^{(t)} = \sum_{\ell=1}^{p} n_\ell^{(t)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell$ for suitable $n_1^{(t)}, \ldots, n_p^{(t)} \in \mathbb{N}$.

Intuitively, $n_\ell^{(t)}$ counts how many times $\tilde{\mathbf{x}}_\ell$ was added to (if $y_\ell = 1$), or subtracted from (if $y_\ell = -1$) weights.

Dual Perceptron Learning

Dual Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

Dual Perceptron Learning

Dual Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

- $\vec{n}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.

Dual Perceptron Learning

Dual Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

- $\vec{n}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
- In $(t+1)$-th step, $(n_1^{(t+1)}, \ldots, n_p^{(t+1)})$ is computed as follows:
  - If $sig(\sum_{\ell=1}^{p} n_\ell^{(t)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $n_k^{(t+1)} := n_k^{(t)} + 1$, else, $n_k^{(t+1)} := n_k^{(t)}$.
  - $n_\ell^{(t+1)} := n_\ell^{(t)}$ for all $\ell \neq k$.

Here $k = (t \mod p) + 1$, the examples are considered cyclically.

Dual Perceptron Learning

Dual Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

- $\vec{n}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
- In $(t+1)$-th step, $(n_1^{(t+1)}, \ldots, n_p^{(t+1)})$ is computed as follows:
  - If $sig(\sum_{\ell=1}^{p} n_\ell^{(t)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_k) \neq y_k$, then $n_k^{(t+1)} := n_k^{(t)} + 1$, else, $n_k^{(t+1)} := n_k^{(t)}$.
  - $n_\ell^{(t+1)} := n_\ell^{(t)}$ for all $\ell \neq k$.

  Here $k = (t \mod p) + 1$, the examples are considered cyclically.

If D is linearly separable, there exists t^* such that $\sum_{\ell=1}^{p} n_\ell^{(t^*)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell$ is consistent with D. The algorithm stops at such t^* and returns $(n_1^{(t^*)}, \ldots, n_p^{(t^*)})$ so that $\sum_{\ell=1}^{p} n_\ell^{(t^*)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell$ is consistent with D.

Example

Training set:

$$D = \{((2, -1), 1), ((2, 1), 1), ((1, 3), -1)\}$$

Example

Training set:

$$D = \{((2, -1), 1), ((2, 1), 1), ((1, 3), -1)\}$$

That is

$$\begin{aligned} \vec{x}_1 &= (2, -1) & \quad \tilde{\mathbf{x}}_1 &= (1, 2, -1) \\ \vec{x}_2 &= (2, 1) & \quad \tilde{\mathbf{x}}_2 &= (1, 2, 1) \\ \vec{x}_3 &= (1, 3) & \quad \tilde{\mathbf{x}}_3 &= (1, 1, 3) \end{aligned}$$

Example

Training set:

$$D = \{((2,-1),1),((2,1),1),((1,3),-1)\}$$

That is

$$\begin{aligned} \vec{x}_1 &= (2,-1) \\ \vec{x}_2 &= (2,1) \\ \vec{x}_3 &= (1,3) \end{aligned} \qquad \begin{aligned} \tilde{\mathbf{x}}_1 &= (1,2,-1) \\ \tilde{\mathbf{x}}_2 &= (1,2,1) \\ \tilde{\mathbf{x}}_3 &= (1,1,3) \end{aligned}$$

$$\begin{aligned} y_1 &= 1 \\ y_2 &= 1 \\ y_3 &= -1 \end{aligned}$$

The initial values $n_1^{(0)} = n_2^{(0)} = n_3^{(0)} = 0$.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0, 0, 0)$.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0, 0, 0)$.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0,0,0)$.

-1
=

- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0,0,1)$.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0, 0, 1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = -1 \cdot (1, 1, 3) \cdot (1, 2, -1) = -1 \cdot 0 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(4)} = (0, 0, 1)$.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0, 0, 1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = -1 \cdot (1, 1, 3) \cdot (1, 2, -1) = -1 \cdot 0 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(4)} = (0, 0, 1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = -1 \cdot (1, 1, 3) \cdot (1, 2, 1) = -1 \cdot 6 = -6$, thus $sig(\sum_{\ell=1}^{p} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = -1 \neq y_2$. Hence, $\vec{n}^{(5)} = (0, 1, 1)$.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0,0,1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = -1 \cdot (1,1,3) \cdot (1,2,-1) = -1 \cdot 0 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(4)} = (0,0,1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = -1 \cdot (1,1,3) \cdot (1,2,1) = -1 \cdot 6 = -6$, thus $sig(\sum_{\ell=1}^{p} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = -1 \neq y_2$. Hence, $\vec{n}^{(5)} = (0,1,1)$.
- $\sum_{\ell=1}^{p} n_\ell^{(5)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_3 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_3 = -5$, thus $\vec{n}^{(6)} = (0,1,1)$. This is OK.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0,0,1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = -1 \cdot (1,1,3) \cdot (1,2,-1) = -1 \cdot 0 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(4)} = (0,0,1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = -1 \cdot (1,1,3) \cdot (1,2,1) = -1 \cdot 6 = -6$, thus $sig(\sum_{\ell=1}^{p} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = -1 \neq y_2$. Hence, $\vec{n}^{(5)} = (0,1,1)$.
- $\sum_{\ell=1}^{p} n_\ell^{(5)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_3 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_3 = -5$, thus $\vec{n}^{(6)} = (0,1,1)$. This is OK.
- $\sum_{\ell=1}^{p} n_\ell^{(6)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_1 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = 4$, thus $\vec{n}^{(7)} = (0,1,1)$. This is OK.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0, 0, 0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0, 0, 1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = -1 \cdot (1, 1, 3) \cdot (1, 2, -1) = -1 \cdot 0 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(4)} = (0, 0, 1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = -1 \cdot (1, 1, 3) \cdot (1, 2, 1) = -1 \cdot 6 = -6$, thus $sig(\sum_{\ell=1}^{p} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = -1 \neq y_2$. Hence, $\vec{n}^{(5)} = (0, 1, 1)$.
- $\sum_{\ell=1}^{p} n_\ell^{(5)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_3 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_3 = -5$, thus $\vec{n}^{(6)} = (0, 1, 1)$. This is OK.
- $\sum_{\ell=1}^{p} n_\ell^{(6)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_1 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = 4$, thus $\vec{n}^{(7)} = (0, 1, 1)$. This is OK.
- $\sum_{\ell=1}^{p} n_\ell^{(6)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_2 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = 0$, thus $\vec{n}^{(7)} = (0, 1, 1)$. This is OK.

- $\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(0)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(1)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(1)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = 1 = y_2$. Hence, $\vec{n}^{(2)} = (0,0,0)$.
- $\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(2)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3) = 1 \neq y_3$. Hence, $\vec{n}^{(3)} = (0,0,1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = -1 \cdot (1,1,3) \cdot (1,2,-1) = -1 \cdot 0 = 0$, thus $sig(\sum_{\ell=1}^{3} n_\ell^{(3)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1) = 1 = y_1$. Hence, $\vec{n}^{(4)} = (0,0,1)$.
- $\sum_{\ell=1}^{3} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = -1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = -1 \cdot (1,1,3) \cdot (1,2,1) = -1 \cdot 6 = -6$, thus $sig(\sum_{\ell=1}^{p} n_\ell^{(4)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2) = -1 \neq y_2$. Hence, $\vec{n}^{(5)} = (0,1,1)$.
- $\sum_{\ell=1}^{p} n_\ell^{(5)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_3 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_3 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_3 = -5$, thus $\vec{n}^{(6)} = (0,1,1)$. This is OK.
- $\sum_{\ell=1}^{p} n_\ell^{(6)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_1 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_1 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_1 = 4$, thus $\vec{n}^{(7)} = (0,1,1)$. This is OK.
- $\sum_{\ell=1}^{p} n_\ell^{(6)} \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}_2 = 1 \cdot \tilde{\mathbf{x}}_2 \cdot \tilde{\mathbf{x}}_2 - 1 \cdot \tilde{\mathbf{x}}_3 \cdot \tilde{\mathbf{x}}_2 = 0$, thus $\vec{n}^{(7)} = (0,1,1)$. This is OK.

The result: $\tilde{\mathbf{x}}_2 - \tilde{\mathbf{x}}_3$.

Dual Perceptron Learning – Output

$\vec{w}$ =

Let $\sum_{\ell=1}^{p} n_\ell \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell$ result from the dual perceptron learning algorithm.

I.e., each $n_\ell = n_\ell^{(t^*)} \in \mathbb{N}$ for suitable t^* in which the algorithm found a consistent vector.

This vector of weights determines a linear classifier that for a given $\vec{x} \in \mathbb{R}^n$ gives

$$h[\vec{w}](\vec{x}) = sig\left(\sum_{\ell=1}^{p} n_\ell \cdot y_\ell \cdot \tilde{\mathbf{x}}_\ell \cdot \tilde{\mathbf{x}}\right)$$

(Here $\tilde{\mathbf{x}}$ is the augmented feature vector obtained from $\vec{x}$.)

Crucial observation: The (augmented) feature vectors $\tilde{\mathbf{x}}_\ell$ and $\tilde{\mathbf{x}}$ occur *only* in scalar products!

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$
$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$
$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping ϕ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathbf{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$
$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping ϕ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathbf{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Then

$$\phi(\tilde{\mathbf{x}}_k) \cdot \phi(\tilde{\mathbf{x}}_\ell) = (\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathbf{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$

$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping ϕ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathbf{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Then

$$\phi(\tilde{\mathbf{x}}_k) \cdot \phi(\tilde{\mathbf{x}}_\ell) = (\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2$$

THE Idea: Define a *kernel* $\kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell) = (\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell)^2$ and replace $\tilde{\mathbf{x}}_k \cdot \tilde{\mathbf{x}}_\ell$ in the dual perceptron algorithm with $\kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell)$.

Kernel Perceptron Learning

Kernel Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

Kernel Perceptron Learning

Kernel Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

- $\vec{n}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.

Kernel Perceptron Learning

Kernel Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

- $\vec{n}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
- In $(t+1)$-th step, $(n_1^{(t+1)}, \ldots, n_p^{(t+1)})$ is computed as follows:
  - If $sig\left(\sum_{\ell=1}^{p} n_\ell^{(t)} \cdot y_\ell \cdot \kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell)\right) \neq y_k$, then $n_k^{(t+1)} := n_k^{(t)} + 1$,

    else, $n_k^{(t+1)} := n_k^{(t)}$.
  - $n_\ell^{(t+1)} := n_\ell^{(t)}$ for all $\ell \neq k$.

Here $k = (t \mod p) + 1$, the examples are considered cyclically.

$\tilde{x}_k \cdot \tilde{x}_\ell$

$\kappa(\bar{x}_k, \bar{x}_\ell) = (\bar{x}_k \cdot \bar{x}_\ell)^2$

Kernel Perceptron Learning

Kernel Perceptron learning algorithm :

Compute a sequence of vectors of numbers $\vec{n}^{(0)}, \vec{n}^{(1)}, \ldots$ where each $\vec{n}^{(t)} = (n_1^{(t)}, \ldots, n_p^{(t)}) \in \mathbb{N}^p$.

- $\vec{n}^{(0)}$ is initialized to $\vec{0} = (0, \ldots, 0)$.
- In $(t+1)$-th step, $(n_1^{(t+1)}, \ldots, n_p^{(t+1)})$ is computed as follows:
  - If $sig\left(\sum_{\ell=1}^{p} n_\ell^{(t)} \cdot y_\ell \cdot \kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell)\right) \neq y_k$, then $n_k^{(t+1)} := n_k^{(t)} + 1$, else, $n_k^{(t+1)} := n_k^{(t)}$.
  - $n_\ell^{(t+1)} := n_\ell^{(t)}$ for all $\ell \neq k$.

  Here $k = (t \mod p) + 1$, the examples are considered cyclically.

Intuition: The algorithm computes a linear classifier in $\mathbb{R}^6$ for training examples transformed using ϕ.
The trick is that the transformation ϕ itself *does not have to be explicitly computed!*

Dual Perceptron Learning

Let $\vec{n} = (n_1, \ldots, n_p)$ result from the kernel perceptron learning algorithm.
I.e., each $n_\ell = n_\ell^{(t^*)} \in \mathbb{N}$ for suitable t^* such that
$sig\left(\sum_{\ell=1}^{p} n_\ell^{(t^*)} \cdot y_\ell \cdot \kappa(\tilde{\mathbf{x}}_k, \tilde{\mathbf{x}}_\ell)\right) = y_k$ for all $k = 1, \ldots, p$.

We obtain a *non-linear classifier* that for a given $\vec{x} \in \mathbb{R}^n$ gives

$$h[\vec{w}](\vec{x}) = sig\left(\sum_{\ell=1}^{p} n_\ell \cdot y_\ell \cdot \kappa(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_\ell)\right)$$

(Here $\tilde{\mathbf{x}}$ is the augmented feature vector obtained from $\vec{x}$.)

Are there other kernels that correspond to the scalar product in higher dimensional spaces?

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Věta (Mercer's)

κ is a kernel if the corresponding Gram matrix K of the training set D, whose each ℓk-th element is $\kappa(\vec{x}_\ell, \vec{x}_k)$, is positive semi-definite for all possible choices of the training set D.

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Věta (Mercer's)

κ is a kernel if the corresponding Gram matrix K of the training set D, whose each ℓk-th element is $\kappa(\vec{x}_\ell, \vec{x}_k)$, is positive semi-definite for all possible choices of the training set D.

Kernels can be constructed from existing kernels by several operations

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Věta (Mercer's)

κ is a kernel if the corresponding Gram matrix K of the training set D, whose each ℓk-th element is $\kappa(\vec{x}_\ell, \vec{x}_k)$, is positive semi-definite for all possible choices of the training set D.

Kernels can be constructed from existing kernels by several operations

- linear combination (i.e. multiply by a constant, or sum),

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Věta (Mercer's)

κ is a kernel if the corresponding Gram matrix K of the training set D, whose each ℓk-th element is $\kappa(\vec{x}_\ell, \vec{x}_k)$, is positive semi-definite for all possible choices of the training set D.

Kernels can be constructed from existing kernels by several operations

- linear combination (i.e. multiply by a constant, or sum),
- multiplication,

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Věta (Mercer's)

κ is a kernel if the corresponding Gram matrix K of the training set D, whose each ℓk-th element is $\kappa(\vec{x}_\ell, \vec{x}_k)$, is positive semi-definite for all possible choices of the training set D.

Kernels can be constructed from existing kernels by several operations

- linear combination (i.e. multiply by a constant, or sum),
- multiplication,
- exponentiation,

Kernels

Given a (potential) kernel $\kappa(\vec{x}_\ell, \vec{x}_k)$ we need to check whether $\kappa(\vec{x}_\ell, \vec{x}_k) = \phi(\vec{x}_\ell) \cdot \phi(\vec{x}_k)$ for a function ϕ. This might be very difficult.

Věta (Mercer's)

κ is a kernel if the corresponding Gram matrix K of the training set D, whose each ℓk-th element is $\kappa(\vec{x}_\ell, \vec{x}_k)$, is positive semi-definite for all possible choices of the training set D.

Kernels can be constructed from existing kernels by several operations

- linear combination (i.e. multiply by a constant, or sum),
- multiplication,
- exponentiation,
- multiply by a polynomial with non-negative coefficients,
- ...

(see e.g. "Pattern Recognition and Machine Learning" by Bishop)

Examples of Kernels

- Linear: $\kappa(\vec{x}_\ell, \vec{x}_k) = \vec{x}_\ell \cdot \vec{x}_k$
  The corresponding mapping $\phi(\vec{x}) = \vec{x}$ is identity (no transformation).

Examples of Kernels

- Linear: $\kappa(\vec{x}_\ell, \vec{x}_k) = \vec{x}_\ell \cdot \vec{x}_k$
  The corresponding mapping $\phi(\vec{x}) = \vec{x}$ is identity (no transformation).
- Polynomial of power m: $\kappa(\vec{x}_\ell, \vec{x}_k) = (1 + \vec{x}_\ell \cdot \vec{x}_k)^m$
  The corresponding mapping assigns to $\vec{x} \in \mathbb{R}^n$ the vector $\phi(\vec{x})$ in $\mathbb{R}^{\binom{n+m}{m}}$.

Examples of Kernels

- Linear: $\kappa(\vec{x}_\ell, \vec{x}_k) = \vec{x}_\ell \cdot \vec{x}_k$
  The corresponding mapping $\phi(\vec{x}) = \vec{x}$ is identity (no transformation).
- Polynomial of power m: $\kappa(\vec{x}_\ell, \vec{x}_k) = (1 + \vec{x}_\ell \cdot \vec{x}_k)^m$
  The corresponding mapping assigns to $\vec{x} \in \mathbb{R}^n$ the vector $\phi(\vec{x})$ in $\mathbb{R}^{\binom{n+m}{m}}$.
- Gaussian (radial-basis function): $\kappa(\vec{x}_\ell, \vec{x}_k) = e^{-\frac{\|\vec{x}_\ell - \vec{x}_k\|^2}{2\sigma^2}}$
  The corresponding mapping ϕ maps $\vec{x}$ to an *infinite-dimensional* vector $\phi(\vec{x})$ which is, in fact, a Gaussian function; combination of such functions for support vectors is then the separating hypersurface.
- ...

$\mathcal{K}(\vec{w}, \vec{x})$

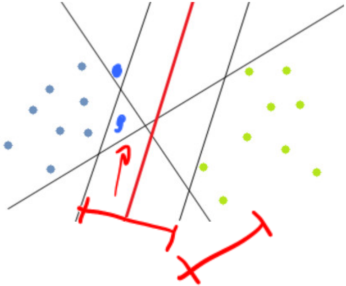
Examples of Kernels

- Linear: $\kappa(\vec{x}_\ell, \vec{x}_k) = \vec{x}_\ell \cdot \vec{x}_k$
  The corresponding mapping $\phi(\vec{x}) = \vec{x}$ is identity (no transformation).
- Polynomial of power m: $\kappa(\vec{x}_\ell, \vec{x}_k) = (1 + \vec{x}_\ell \cdot \vec{x}_k)^m$
  The corresponding mapping assigns to $\vec{x} \in \mathbb{R}^n$ the vector $\phi(\vec{x})$ in $\mathbb{R}^{\binom{n+m}{m}}$.
- Gaussian (radial-basis function): $\kappa(\vec{x}_\ell, \vec{x}_k) = e^{-\frac{\|\vec{x}_\ell - \vec{x}_k\|^2}{2\sigma^2}}$
  The corresponding mapping ϕ maps $\vec{x}$ to an *infinite-dimensional* vector $\phi(\vec{x})$ which is, in fact, a Gaussian function; combination of such functions for support vectors is then the separating hypersurface.
- ...

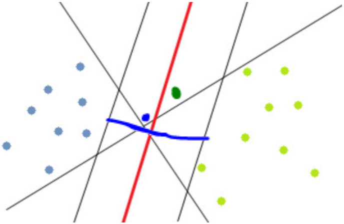
Choosing kernels remains to be black magic of kernel methods. They are usually chosen based on trial and error (of course, experience and additional insight into data helps).

Now let's go on to the main area where kernel methods are used: to enhance support vector machines.

SVM Idea – Which Linear Classifier is the Best?



SVM Idea – Which Linear Classifier is the Best?



Benefits of maximum margin:

- Intuitively, maximum margin is good w.r.t. generalization.
- Only the *support vectors* (those on the magin) matter, others can, in principle, be ignored.

Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,

Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,
- $\underline{\vec{w}} = (w_1, \ldots, w_n)$ a vector of all weights except w_0,

Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,
- $\underline{\vec{w}} = (w_1, \ldots, w_n)$ a vector of all weights except w_0,
- $\vec{x} = (x_1, \ldots, x_n)$ a (generic) feature vector.

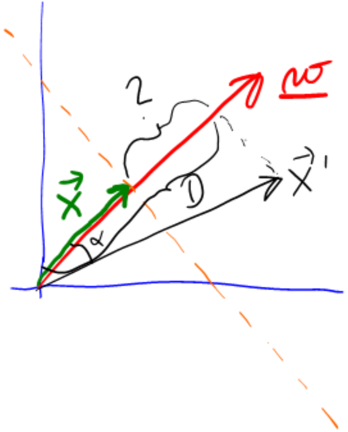
Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,
- $\underline{\vec{w}} = (w_1, \ldots, w_n)$ a vector of all weights except w_0,
- $\vec{x} = (x_1, \ldots, x_n)$ a (generic) feature vector.

Consider a linear classifier:

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = w_0 + \underline{\vec{w}} \cdot \vec{x} \geq 0 \\ -1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = w_0 + \underline{\vec{w}} \cdot \vec{x} < 0 \end{cases}$$



$$\underline{w} \cdot \vec{x} = \|\underline{w}\| \cdot \|\vec{x}\|$$

$$w_0 + \underline{w}\,\vec{x} = 0 \iff$$

$$w_0 + \|\underline{w}\| \cdot \|\vec{x}\| = 0 \iff$$

$$\|\vec{x}\| = \frac{-w_0}{\|\underline{w}\|}$$

$$\underline{w} \cdot \vec{x}\,' = \|\underline{w}\| \cdot \|\vec{x}\,'\| \cdot \cos\alpha = \|\underline{w}\| \cdot D$$

$$\iff D = \frac{\underline{w}\,\vec{x}\,'}{\|\underline{w}\|}$$

$$D - \|\vec{x}\| = \frac{\underline{w}\,\vec{x}\,'}{\|\underline{w}\|} - \left(\frac{-w_0}{\|\underline{w}\|}\right) = \frac{w_0 + \underline{w}\,\vec{x}\,'}{\|\underline{w}\|}$$

Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,
- $\underline{\vec{w}} = (w_1, \ldots, w_n)$ a vector of all weights except w_0,
- $\vec{x} = (x_1, \ldots, x_n)$ a (generic) feature vector.

Consider a linear classifier:

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = w_0 + \underline{\vec{w}} \cdot \vec{x} \geq 0 \\ -1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = w_0 + \underline{\vec{w}} \cdot \vec{x} < 0 \end{cases}$$

The *signed distance* of $\vec{x}$ from the decision boundary determined by $\vec{w}$ is

$$d[\vec{w}](\vec{x}) = \frac{w_0 + \underline{\vec{w}} \cdot \vec{x}_k}{\|\underline{\vec{w}}\|}$$

Here $\|\underline{\vec{w}}\| = \sqrt{\sum_{i=1}^{n} w_i^2}$ is the Euclidean norm of $\underline{\vec{w}}$.

$\vec{x} \in \mathbb{R}^1 \quad : \quad w_0 + w_1 x$

$$w_0 + w_1 x_0 = 0 \iff x_0 = -\frac{w_0}{w_1}$$

$$x - x_0 = x - \left(-\frac{w_0}{w_1}\right) = \frac{w_0 + w_1 x}{w_1}$$

Support Vector Machines (SVM)

Notation:

- $\vec{w} = (w_0, w_1, \ldots, w_n)$ a vector of weights,
- $\underline{\vec{w}} = (w_1, \ldots, w_n)$ a vector of all weights except w_0,
- $\vec{x} = (x_1, \ldots, x_n)$ a (generic) feature vector.

Consider a linear classifier:

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = w_0 + \underline{\vec{w}} \cdot \vec{x} \geq 0 \\ -1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i = w_0 + \underline{\vec{w}} \cdot \vec{x} < 0 \end{cases}$$

The *signed distance* of $\vec{x}$ from the decision boundary determined by $\vec{w}$ is

$$d[\vec{w}](\vec{x}) = \frac{w_0 + \underline{\vec{w}} \cdot \vec{x}_k}{\|\underline{\vec{w}}\|}$$

Here $\|\underline{\vec{w}}\| = \sqrt{\sum_{i=1}^{n} w_i^2}$ is the Euclidean norm of $\underline{\vec{w}}$.

$|d[\vec{w}](\vec{x})|$ is the distance of $\vec{x}$ from the decision boundary.
$d[\vec{w}](\vec{x})$ is positive for $\vec{x}$ on the side to which $\underline{\vec{w}}$ points and negative on the opposite side.

Support Vectors & Margin

- Given a training set

$$D = \{(\vec{x}_1, y(\vec{x}_1)), (\vec{x}_2, y(\vec{x}_2)), \ldots, (\vec{x}_p, y(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y(\vec{x}_k) \in \{-1, 1\}$.

Support Vectors & Margin

- Given a training set

  $$D = \{(\vec{x}_1, y(\vec{x}_1)), (\vec{x}_2, y(\vec{x}_2)), \ldots, (\vec{x}_p, y(\vec{x}_p))\}$$

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y(\vec{x}_k) \in \{-1, 1\}$.

  We write y_k instead of $y(\vec{x}_k)$.

- Assume that D is linearly separable, let $\vec{w}$ *be consistent with D so that the distance of the decision boundary from the nearest examples on both sides is the same* (if not, it suffices to adjust w_0).

Support Vectors & Margin

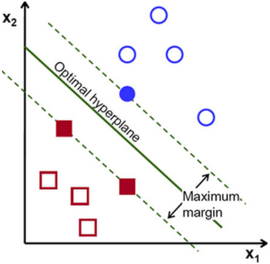
- Given a training set

$$D = \{(\vec{x}_1, y(\vec{x}_1)), (\vec{x}_2, y(\vec{x}_2)), \ldots, (\vec{x}_p, y(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in X \subseteq \mathbb{R}^n$ and $y(\vec{x}_k) \in \{-1, 1\}$.

We write y_k instead of $y(\vec{x}_k)$.

- Assume that D is linearly separable, let $\vec{w}$ *be consistent with D so that the distance of the decision boundary from the nearest examples on both sides is the same* (if not, it suffices to adjust w_0).



- *Support vectors* are those $\vec{x}_k$ that minimize $|d[\vec{w}](\vec{x}_k)|$.
- *Margin* ρ of $\vec{w}$ is twice the distance between support vectors and the decision boundary.

Our goal is to find a classifier that maximizes the margin.

Maximizing the Margin

For $\vec{w}$ consistent with D (such that no $\vec{x}_k$ lies on the decision boundary) we have

$$\varrho = 2 \cdot \frac{|w_0 + \underline{\vec{w}} \cdot \vec{x}_k|}{||\underline{\vec{w}}||} = 2 \cdot \frac{y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k)}{||\underline{\vec{w}}||} > 0$$

where $\vec{x}_k$ is a support vector.

Maximizing the Margin

For $\vec{w}$ consistent with D (such that no $\vec{x}_k$ lies on the decision boundary) we have

$$\varrho = 2 \cdot \frac{|w_0 + \underline{\vec{w}} \cdot \vec{x}_k|}{||\underline{\vec{w}}||} = 2 \cdot \frac{y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k)}{||\underline{\vec{w}}||} > 0$$

where $\vec{x}_k$ is a support vector.

We may safely consider only $\vec{w}$ such that $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$ for the support vectors.

Just adjust the length of $\vec{w}$ so that $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$, the denominator $||\underline{\vec{w}}||$ will compensate.

Maximizing the Margin

For $\vec{w}$ consistent with D (such that no $\vec{x}_k$ lies on the decision boundary) we have

$$\varrho = 2 \cdot \frac{|w_0 + \underline{\vec{w}} \cdot \vec{x}_k|}{\|\underline{\vec{w}}\|} = 2 \cdot \frac{y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k)}{\|\underline{\vec{w}}\|} > 0$$

where $\vec{x}_k$ is a support vector.

We may safely consider only $\vec{w}$ such that $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$ for the support vectors.

Just adjust the length of $\vec{w}$ so that $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$, the denominator $\|\underline{\vec{w}}\|$ will compensate.

Then maximizing ϱ is equivalent to maximizing $2/\|\underline{\vec{w}}\|$.

(In what follows we use a bit looser constraint:

$$y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) \geq 1 \text{ for all } \vec{x}_k$$

However, the result is the same since even with this looser condition, the support vectors always satisfy $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$ whenever $2/\|\underline{w}\|$ is maximal.)

SVM – Optimization

Margin maximization can be formulated as a *quadratic optimization problem:*

Find $\vec{w} = (w_0, \ldots, w_n)$ such that

$$\rho = \frac{2}{\|\underline{\vec{w}}\|} \text{ is maximized}$$

and for all $(\vec{x}_k, y_k) \in D$ we have $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) \geq 1$.

SVM – Optimization

Margin maximization can be formulated as a *quadratic optimization problem:*

Find $\vec{w} = (w_0, \ldots, w_n)$ such that

$$\rho = \frac{2}{\|\underline{\vec{w}}\|} \text{ is maximized}$$

and for all $(\vec{x}_k, y_k) \in D$ we have $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) \geq 1$.

$\|\vec{w}\| = \sqrt{\vec{w} \cdot \vec{w}}$

$\Leftrightarrow \min \frac{\|\vec{w}\|}{2} \Leftrightarrow \min \frac{\|\vec{w}\|^2}{2} \Leftrightarrow \min \frac{\vec{w} \cdot \vec{w}}{2}$

which can be reformulated as:

Find $\vec{w}$ such that

$$\Phi(\vec{w}) = \|\underline{\vec{w}}\|^2 = \underline{\vec{w}} \cdot \underline{\vec{w}} \text{ is minimized}$$

and for all $(\vec{x}_k, y_k) \in D$ we have $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) \geq 1$.

SVM – Optimization

- Need to optimize a quadratic function subject to linear constraints.

SVM – Optimization

- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which efficient methods (and tools) exist.

SVM – Optimization

- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which efficient methods (and tools) exist.
- The solution usually involves construction of a *dual problem* where *Lagrange multipliers* α_i are associated with every inequality (constraint) in the original problem:

SVM – Optimization

- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which efficient methods (and tools) exist.
- The solution usually involves construction of a *dual problem* where *Lagrange multipliers* α_i are associated with every inequality (constraint) in the original problem:

Find $\alpha = (\alpha_1, \ldots, \alpha_p)$ such that

$$\Psi(\alpha) = \sum_{\ell=1}^{p} \alpha_\ell - \frac{1}{2} \sum_{\ell=1}^{p} \sum_{k=1}^{p} \alpha_\ell \cdot \alpha_k \cdot y_\ell \cdot y_k \cdot \vec{x}_\ell \cdot \vec{x}_k \text{ is maximized}$$

so that the following constraints are satisfied:

- $\sum_{\ell=1}^{p} \alpha_\ell y_\ell = 0$
- $\alpha_\ell \geq 0$ for all $1 \leq \ell \leq p$

The Optimization Problem Solution

- Given a solution $\alpha_1, \ldots, \alpha_n$ to the dual problem, solution $\vec{w} = (w_0, w_1, \ldots, w_n)$ to the original one is:

$$\underline{\vec{w}} = (w_1, \ldots, w_n) = \sum_{\ell=1}^{p} \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell$$

$$w_0 = y_k - \sum_{\ell=1}^{p} \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell \cdot \vec{x}_k \text{ for an arbitrary } \alpha_k > 0$$

The Optimization Problem Solution

- Given a solution $\alpha_1, \ldots, \alpha_n$ to the dual problem, solution $\vec{w} = (w_0, w_1, \ldots, w_n)$ to the original one is:

$$\underline{\vec{w}} = (w_1, \ldots, w_n) = \sum_{\ell=1}^{p} \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell$$

$$w_0 = y_k - \sum_{\ell=1}^{p} \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell \cdot \vec{x}_k \text{ for an arbitrary } \alpha_k > 0$$

Note that $\alpha_k > 0$ iff $\vec{x}_k$ is a support vector. Hence it does not matter which $\alpha_k > 0$ is chosen in the above definition of w_0.

The Optimization Problem Solution

- Given a solution $\alpha_1, \ldots, \alpha_n$ to the dual problem, solution $\vec{w} = (w_0, w_1, \ldots, w_n)$ to the original one is:

$$\underline{\vec{w}} = (w_1, \ldots, w_n) = \sum_{\ell=1}^{p} \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell$$

$$w_0 = y_k - \sum_{\ell=1}^{p} \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell \cdot \vec{x}_k \text{ for an arbitrary } \alpha_k > 0$$

  Note that $\alpha_k > 0$ iff $\vec{x}_k$ is a support vector. Hence it does not matter which $\alpha_k > 0$ is chosen in the above definition of w_0.

- The classifier is then

$$\begin{aligned} h(\vec{x}) &= sig(w_0 + \underline{\vec{w}} \cdot \vec{x}) \\ &= sig\left(y_k - \textstyle\sum_\ell \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell \cdot \vec{x}_k + \sum_\ell \alpha_\ell \cdot y_\ell \cdot \vec{x}_\ell \cdot \vec{x}\right) \end{aligned}$$

Note that both the dual optimization problem as well as the classifier contain training feature vectors only in the scalar product! We may apply the kernel trick!

Kernel SVM

- Choose your favourite kernel κ.

Kernel SVM

- Choose your favourite kernel κ.
- Solve the *dual problem* with kernel κ:

Find $\alpha = (\alpha_1, \ldots, \alpha_p)$ such that

$$\Psi(\alpha) = \sum_{\ell=1}^{p} \alpha_\ell - \frac{1}{2} \sum_{\ell=1}^{p} \sum_{k=1}^{p} \alpha_\ell \cdot \alpha_k \cdot y_\ell \cdot y_k \cdot \kappa(\vec{x}_\ell, \vec{x}_k) \text{ is maximized}$$

so that the following constraints are satisfied:

- $\sum_\ell \alpha_\ell y_\ell = 0$
- $\alpha_\ell \geq 0$ for all $1 \leq \ell \leq p$

Kernel SVM

- Choose your favourite kernel κ.
- Solve the *dual problem* with kernel κ:

Find $\alpha = (\alpha_1, \ldots, \alpha_p)$ such that

$$\Psi(\alpha) = \sum_{\ell=1}^{p} \alpha_\ell - \frac{1}{2} \sum_{\ell=1}^{p} \sum_{k=1}^{p} \alpha_\ell \cdot \alpha_k \cdot y_\ell \cdot y_k \cdot \kappa(\vec{x}_\ell, \vec{x}_k) \text{ is maximized}$$

so that the following constraints are satisfied:

  - $\sum_\ell \alpha_\ell y_\ell = 0$
  - $\alpha_\ell \geq 0$ for all $1 \leq \ell \leq p$

- Then use the classifier:

$$h(\vec{x}) = sig\left(y_k - \sum_\ell \alpha_\ell \cdot y_\ell \cdot \kappa(\vec{x}_\ell, \vec{x}_k) + \sum_\ell \alpha_\ell \cdot y_\ell \cdot \kappa(\vec{x}_\ell, \vec{x})\right)$$

Kernel SVM

- Choose your favourite kernel κ.
- Solve the *dual problem* with kernel κ:

Find $\alpha = (\alpha_1, \ldots, \alpha_p)$ such that

$$\Psi(\alpha) = \sum_{\ell=1}^{p} \alpha_\ell - \frac{1}{2} \sum_{\ell=1}^{p} \sum_{k=1}^{p} \alpha_\ell \cdot \alpha_k \cdot y_\ell \cdot y_k \cdot \kappa(\vec{x}_\ell, \vec{x}_k) \text{ is maximized}$$

so that the following constraints are satisfied:

  - $\sum_\ell \alpha_\ell y_\ell = 0$
  - $\alpha_\ell \geq 0$ for all $1 \leq \ell \leq p$

- Then use the classifier:

$$h(\vec{x}) = sig\left(y_k - \sum_\ell \alpha_\ell \cdot y_\ell \cdot \kappa(\vec{x}_\ell, \vec{x}_k) + \sum_\ell \alpha_\ell \cdot y_\ell \cdot \kappa(\vec{x}_\ell, \vec{x})\right)$$

- Note that the optimization techniques remain the same as for the linear SVM without kernels!

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.
- For large problems this is usually not possible, many methods avoiding direct solution have been devised.

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.
- For large problems this is usually not possible, many methods avoiding direct solution have been devised.
- These methods usually decompose the optimization problem into a sequence of smaller ones. Intuitively,

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.
- For large problems this is usually not possible, many methods avoiding direct solution have been devised.
- These methods usually decompose the optimization problem into a sequence of smaller ones. Intuitively,
  - start with a (smaller) subset of training examples.

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.
- For large problems this is usually not possible, many methods avoiding direct solution have been devised.
- These methods usually decompose the optimization problem into a sequence of smaller ones. Intuitively,
  - start with a (smaller) subset of training examples.
  - Find an optimal solution using any solver.

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.
- For large problems this is usually not possible, many methods avoiding direct solution have been devised.
- These methods usually decompose the optimization problem into a sequence of smaller ones. Intuitively,
  - start with a (smaller) subset of training examples.
  - Find an optimal solution using any solver.
  - Afterwards, only support vectors matter in the solution! Leave only them in the training set, and add new training examples.

Comments on Algorithms

- The main bottleneck of SVM's is in complexity of quadratic programming (QP). A naive QP solver has cubic complexity.
- For small problems any general purpose optimization algorithm can be used.
- For large problems this is usually not possible, many methods avoiding direct solution have been devised.
- These methods usually decompose the optimization problem into a sequence of smaller ones. Intuitively,
  - start with a (smaller) subset of training examples.
  - Find an optimal solution using any solver.
  - Afterwards, only support vectors matter in the solution! Leave only them in the training set, and add new training examples.
  - This iterative procedure decreases the (general) cost function.

SVM in Applications (Mooney's lecture)

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.

SVM in Applications (Mooney's lecture)

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.

SVM in Applications (Mooney's lecture)

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.

SVM in Applications (Mooney's lecture)

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik et al. '97], principal component analysis [Schölkopf et al. '99], etc.

SVM in Applications (Mooney's lecture)

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik et al. '97], principal component analysis [Schölkopf et al. '99], etc.
- Most popular optimization algorithms for SVMs use decomposition to hillclimb over a subset of α_i's at a time, e.g. SMO [Platt '99] and [Joachims '99]

SVM in Applications (Mooney's lecture)

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik et al. '97], principal component analysis [Schölkopf et al. '99], etc.
- Most popular optimization algorithms for SVMs use decomposition to hillclimb over a subset of α_i's at a time, e.g. SMO [Platt '99] and [Joachims '99]
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.