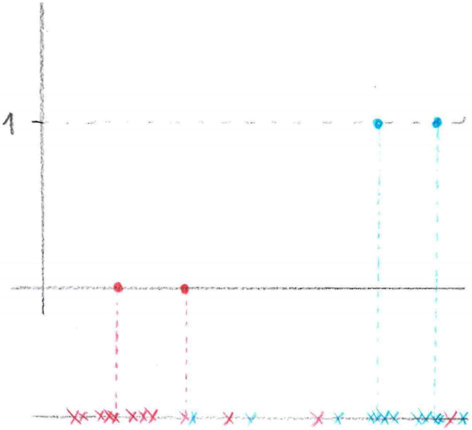
Logistic Regression & SVM

What about classification using regression?

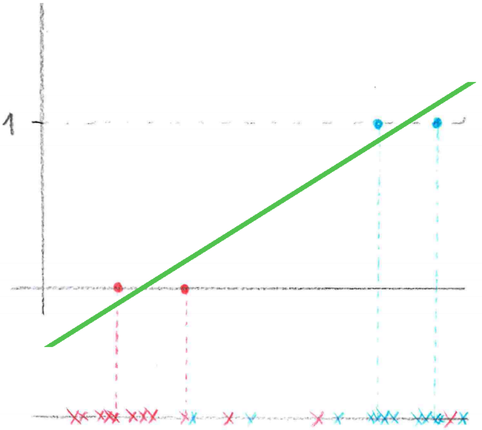
Binary classification: Desired outputs 0 and 1

... we want to capture the probability distribution of the classes



What about classification using regression?

Binary classification: Desired outputs 0 and 1
... we want to capture the probability distribution of the classes

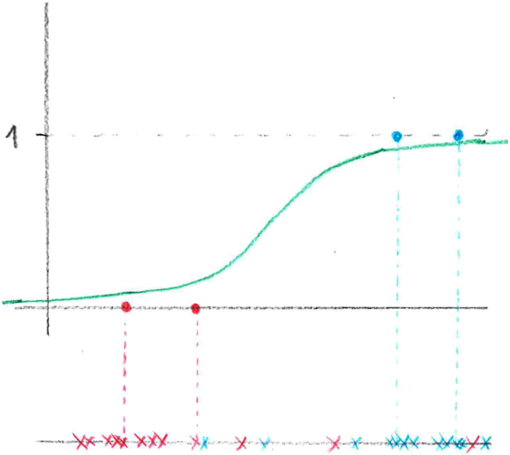


... does not capture the probability well (it is not a probability at all)

What about classification using regression?

Binary classification: Desired outputs 0 and 1

... we want to capture the probability distribution of the classes



... logistic sigmoid $\frac{1}{1+e^{-(\vec{w}\cdot\tilde{x})}}$ is much better!

Logistic Regression

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

Logistic Regression

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

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

$$h[\vec{w}](\vec{x}) := \frac{1}{1 + e^{-\left(w_0 + \sum_{k=1}^{n} w_k x_k\right)}} = \frac{1}{1 + e^{-(\vec{w} \cdot \tilde{\mathrm{x}})}}$$

Here

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

is the *augmented feature vector*.

But what is the meaning of the sigmoid?

The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$.
But why do we model such a probability using $1/(1 + e^{-\vec{w}\cdot\tilde{\mathrm{x}}})$??

But what is the meaning of the sigmoid?

The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$. But why do we model such a probability using $1/(1+e^{-\vec{w}\cdot\tilde{\mathrm{x}}})$??

Denote by $\hat{h}$ the probability $P(Y = 1 \mid X = \vec{x})$, i.e., the "true" probability of the class 1 given the features $\vec{x}$.

The probability $\hat{h}$ cannot be easily modeled using a linear function (the probabilities are between 0 and 1).

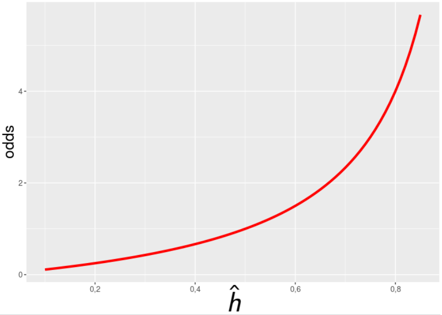
But what is the meaning of the sigmoid?

The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$. But why do we model such a probability using $1/(1+e^{-\vec{w}\cdot\tilde{x}})$??

Denote by $\hat{h}$ the probability $P(Y=1 \mid X=\vec{x})$, i.e., the "true" probability of the class 1 given the features $\vec{x}$.

What about odds of the class 1?

$$odds(\hat{h}) = \hat{h}/(1-\hat{h})$$



Better, at least it is unbounded on one side ...

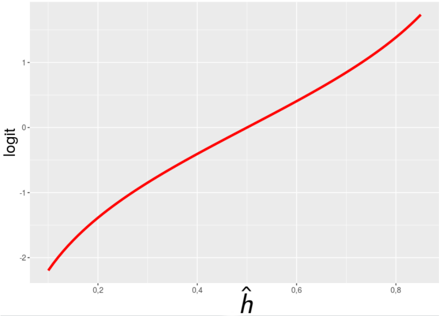
But what is the meaning of the sigmoid?

The model gives probability $h[\vec{w}](\vec{x})$ of the class 1 given an input $\vec{x}$. But why do we model such a probability using $1/(1+e^{-\vec{w}\cdot\tilde{x}})$??

Denote by $\hat{h}$ the probability $P(Y=1 \mid X=\vec{x})$, i.e., the "true" probability of the class 1 given the features $\vec{x}$.

What about log odds (aka logit) of the class 1?

$$logit(\hat{h}) = \log(\hat{h}/(1-\hat{h}))$$



Looks almost linear, at least for probabilities not too close to 0 or 1

But what is the meaning of the sigmoid?

Assume that $\hat{h}$ is the true probability of the class 1 for an "object" with features $\vec{x} \in \mathbb{R}^n$. Put

$$\log(\hat{h}/(1-\hat{h})) = \vec{w} \cdot \tilde{\mathrm{x}}$$

But what is the meaning of the sigmoid?

Assume that $\hat{h}$ is the true probability of the class 1 for an "object" with features $\vec{x} \in \mathbb{R}^n$. Put

$$\log(\hat{h}/(1-\hat{h})) = \vec{w} \cdot \tilde{\mathrm{x}}$$

Then

$$\log((1-\hat{h})/\hat{h})) = -\vec{w} \cdot \tilde{\mathrm{x}}$$

But what is the meaning of the sigmoid?

Assume that $\hat{h}$ is the true probability of the class 1 for an "object" with features $\vec{x} \in \mathbb{R}^n$. Put

$$\log(\hat{h}/(1-\hat{h})) = \vec{w} \cdot \tilde{\mathrm{x}}$$

Then

$$\log((1-\hat{h})/\hat{h})) = -\vec{w} \cdot \tilde{\mathrm{x}}$$

and

$$(1-\hat{h})/\hat{h} = e^{-\vec{w} \cdot \tilde{\mathrm{x}}}$$

But what is the meaning of the sigmoid?

Assume that $\hat{h}$ is the true probability of the class 1 for an "object" with features $\vec{x} \in \mathbb{R}^n$. Put

$$\log(\hat{h}/(1-\hat{h})) = \vec{w} \cdot \tilde{\mathrm{x}}$$

Then

$$\log((1-\hat{h})/\hat{h}) = -\vec{w} \cdot \tilde{\mathrm{x}}$$

and

$$(1-\hat{h})/\hat{h} = e^{-\vec{w} \cdot \tilde{\mathrm{x}}}$$

and

$$\hat{h} = \frac{1}{1 + e^{-\vec{w} \cdot \tilde{\mathrm{x}}}} = h[\vec{w}](\vec{x})$$

That is, if we model log odds using a linear function, the probability is obtained by applying the logistic sigmoid on the result of the linear function.

Logistic Regression

- Given a set D of training samples:

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

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

In what follows we use c_k to denote $c(\vec{x}_k)$.

Logistic Regression

- Given a set D of training samples:

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

Here $\vec{x}_k = (x_{k1} \dots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

In what follows we use c_k to denote $c(\vec{x}_k)$.

Recall that $h[\vec{w}](\vec{x}_k) = 1 \,/\, \left(1 + e^{-\vec{w}\cdot\tilde{x}_k}\right)$ where $\tilde{x}_k = (x_{k0}, x_{k1} \dots, x_{kn})$, here $x_{k0} = 1$

Our goal: Find $\vec{w}$ such that for every $k = 1, \dots, p$ we have that $h[\vec{w}](\vec{x}_k) \approx c_k$

Logistic Regression

- Given a set D of training samples:

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

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

  In what follows we use c_k to denote $c(\vec{x}_k)$.

  Recall that $h[\vec{w}](\vec{x}_k) = 1 \,/\, \left(1 + e^{-\vec{w} \cdot \tilde{\mathrm{x}}_k}\right)$ where
  $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$, here $x_{k0} = 1$
  Our goal: Find $\vec{w}$ such that for every $k = 1, \ldots, p$ we have that $h[\vec{w}](\vec{x}_k) \approx c_k$

- **Binary Cross-entropy:**

$$E(\vec{w}) = -\sum_{k=1}^{p} c_k \log(h[\vec{w}](\vec{x}_k)) + (1 - c_k) \log(1 - h[\vec{w}](\vec{x}_k))$$

Gradient of the Error Function

Consider the **gradient** of the error function:

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w})\right) = \sum_{k=1}^{p} \left(h[\vec{w}](\vec{x}_k) - c_k\right) \cdot \tilde{\mathrm{x}}_k$$

Fakt

If $\nabla E(\vec{w}) = \vec{0} = (0, \ldots, 0)$*, then* $\vec{w}$ *is a global minimum of* E*.*

This follows from the fact that E is convex.

Note that using the squared error with the logistic sigmoid would lead to a non-convex error with several minima!

Logistic Regression – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.

Logistic Regression – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)})$$

Logistic Regression – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathrm{x}}_k\end{aligned}$$

  Here $0 < \varepsilon \leq 1$ is the learning rate.

Note that the algorithm is almost similar to the batch perceptron algorithm!

Logistic Regression – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathrm{x}}_k\end{aligned}$$

Here $0 < \varepsilon \leq 1$ is the learning rate.

Note that the algorithm is almost similar to the batch perceptron algorithm!

Tvrzení

For sufficiently small $\varepsilon > 0$ the sequence $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ converges (component-wisely) to the global minimum of E.

Maximum Likelihood vs Cross-entropy (Dim 1)

Fix a training set $D = \{(x_1, c_1), (x_2, c_2), \ldots, (x_p, c_p)\}$
Generate a sequence $c'_1, \ldots, c'_p \in \{0, 1\}^p$ where each c'_k has been generated independently by the Bernoulli trial generating 1 with probability

$$h[w_0, w_1](x_k) = \frac{1}{1 + e^{-(\mathbf{w_0} + \mathbf{w_1} \cdot x_k)}}$$

and 0 otherwise.

Here w_0, w_1 are **unknown weights**.

How "probable" is it to generate the correct classes $c_1, \ldots, c_p$?

Maximum Likelihood vs Cross-entropy (Dim 1)

Fix a training set $D = \{(x_1, c_1), (x_2, c_2), \ldots, (x_p, c_p)\}$
Generate a sequence $c'_1, \ldots, c'_p \in \{0, 1\}^p$ where each c'_k has been generated independently by the Bernoulli trial generating 1 with probability

$$h[w_0, w_1](x_k) = \frac{1}{1 + e^{-(\mathbf{w_0} + \mathbf{w_1} \cdot x_k)}}$$

and 0 otherwise.

Here w_0, w_1 are **unknown weights**.

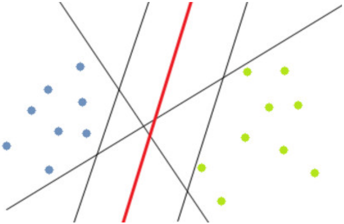
How "probable" is it to generate the correct classes $c_1, \ldots, c_p$?

The following conditions are equivalent:

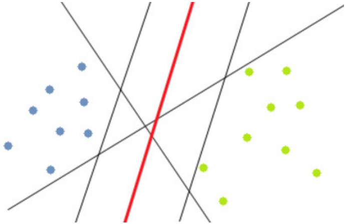
- w_0, w_1 minimize the binary cross-entropy E
- w_0, w_1 maximize the likelihood (i.e., the "probability") of generating the correct values $c_1, \ldots, c_p$ using the above described Bernoulli trials (i.e., that $c'_k = c_k$ for all $k = 1, \ldots, p$)

Note that the above equivalence is a property of the cross-entropy and is not dependent on the "implementation" of $h[w_0, w_1](x_k)$ using the logistic sigmoid.

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:

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

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 $\underline{\vec{w}}$.

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

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

  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 $y(\vec{x}_k)$.

- Assume that D is linearly separable, let $\vec{w}$ *be consistent with D.*

Support Vectors & Margin

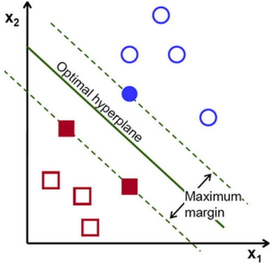
- Given a training set

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

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.



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

Our goal is to find $\vec{w}$ that maximizes the margin $\rho[\vec{w}]$.

Maximizing the Margin

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

$$\rho[\vec{w}] = 2 \cdot \frac{|w_0 + \underline{\vec{w}} \cdot \vec{x}_k|}{\|\underline{\vec{w}}\|} = 2 \cdot \frac{{\color{red}y_k} \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k)}{\|\underline{\vec{w}}\|} > 0$$

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

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

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

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

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 $\rho[\vec{w}]$ 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 $y_k \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) \geq 1$.

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.

But why the SVM have been so successful?

... the improvement by finding the maximum margin classifier does not seem to be so strong ... right?

SVM – Optimization

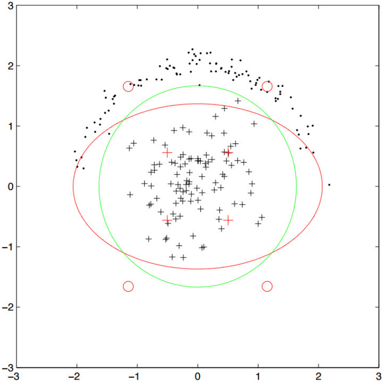
- Need to optimize a quadratic function subject to linear constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which efficient methods (and tools) exist.

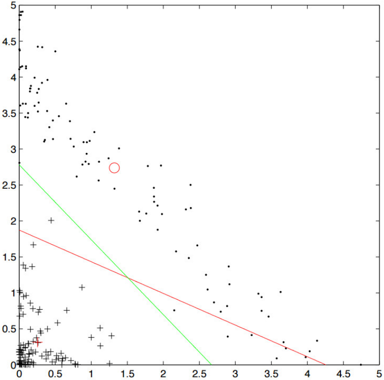
But why the SVM have been so successful?

... the improvement by finding the maximum margin classifier does not seem to be so strong ... right?

The answer lies in their ability to deal with non-linearly separable sets in an efficient way using so called *kernel trick*.

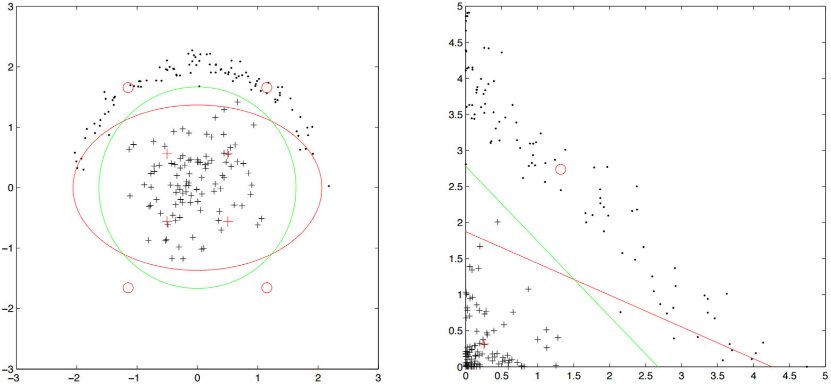
Quadratic Decision Boundary





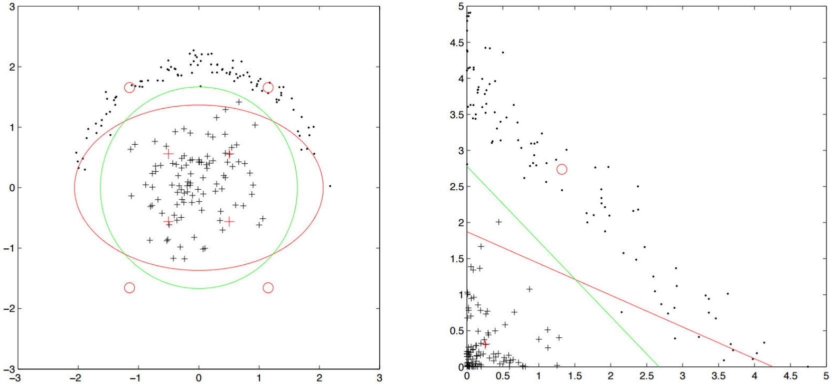
Left: The original set,

Quadratic Decision Boundary



Left: The original set, **Right:** Squared features: $(x_1, x_2) \mapsto (x_1^2, x_2^2)$

Quadratic Decision Boundary

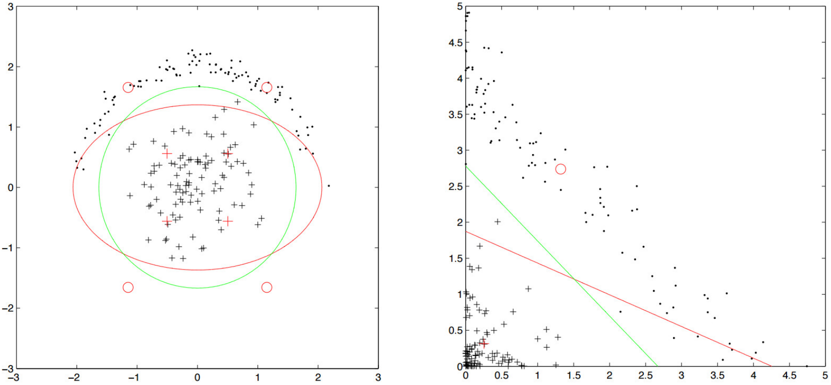


Left: The original set, **Right:** Squared features: $(x_1, x_2) \mapsto (x_1^2, x_2^2)$

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

(The red boundary corresponds to another learning algorithm.)

Quadratic Decision Boundary



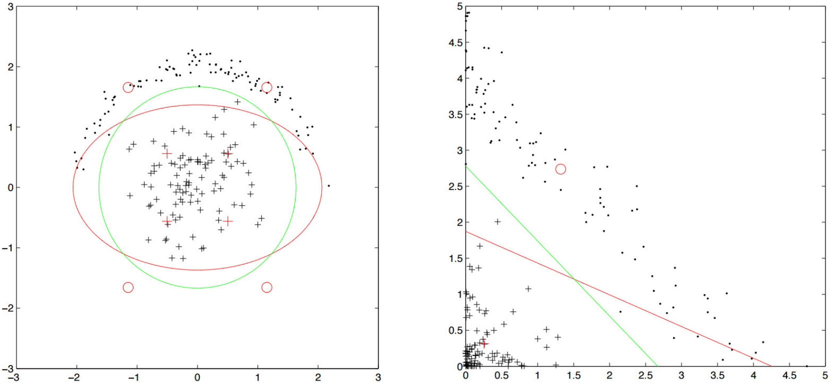
Left: The original set, **Right:** Squared features: $(x_1, x_2) \mapsto (x_1^2, x_2^2)$

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:** Squared features: $(x_1, x_2) \mapsto (x_1^2, x_2^2)$

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 the 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 the 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 the dimension.

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

To avoid explicit construction of the higher dimensional feature space, we use the 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 the dimension.

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

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

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

Dual SVM

The original SVM optimization:

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

Dual SVM

The original SVM optimization:

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

The dual problem (here p is the number of training samples):

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:

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

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

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:

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

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

Note that $\alpha_k > 0$ iff $\vec{x}_k$ is a support vector iff $y_1 \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$. 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:

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

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

  Note that $\alpha_k > 0$ iff $\vec{x}_k$ is a support vector iff $y_1 \cdot (w_0 + \underline{\vec{w}} \cdot \vec{x}_k) = 1$. Hence it does not matter which $\alpha_k > 0$ is chosen in the above definition of w_0.

- The classifier is then

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

Dual SVM

Find $\alpha = (\alpha_1, \dots, \alpha_p)$ 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:

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

The classifier:

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

Dual SVM after projection

Consider your favorite projection φ to another space (where possibly the linear classification works)

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

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

so that the following constraints are satisfied:

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

The classifier:

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

Dual SVM after projection

Consider your favorite projection φ to another space (where possibly the linear classification works)

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

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

so that the following constraints are satisfied:

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

The classifier:

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

... wait a second ... do we really need to *compute* the values of $\varphi(\vec{x}_\ell)$ etc. to obtain the scalar products??

Dual SVM after projection

Consider your favorite projection φ to another space (where possibly the linear classification works)

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

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

so that the following constraints are satisfied:

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

The classifier:

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

... wait a second ... do we really need to *compute* the values of $\varphi(\vec{x}_\ell)$ etc. to obtain the scalar products??NO!

Kernel Dual SVM

Introduce a function $\kappa(\vec{u}, \vec{v}) = \varphi(\vec{u}) \cdot \varphi(\vec{v})$ which computes the scalar product in the space transformed by φ.

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 $1 \leq \ell \leq p$

The classifier:

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

... but now we no longer care what the φ is, right? We just need to know that it exists.

Examples of Kernels

- Linear: $\kappa(\vec{u}, \vec{v}) = \vec{u} \cdot \vec{v}$
  The corresponding mapping $\phi(\vec{u}) = \vec{u}$ is identity (no transformation).

Examples of Kernels

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

Examples of Kernels

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

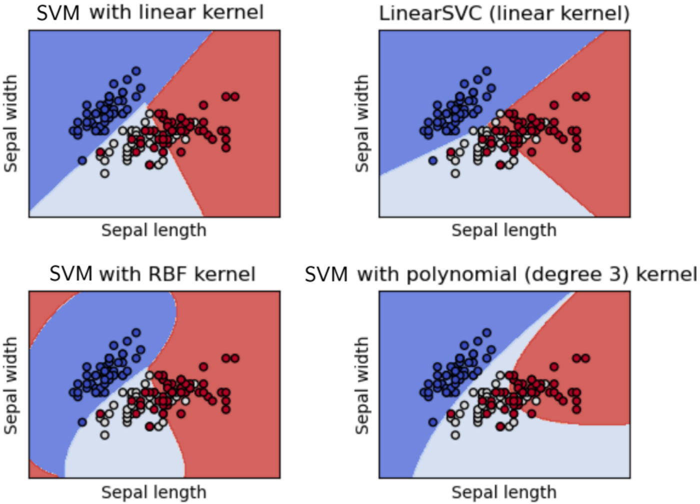
Examples of Kernels

- Linear: $\kappa(\vec{u}, \vec{v}) = \vec{u} \cdot \vec{v}$
  The corresponding mapping $\phi(\vec{u}) = \vec{u}$ is identity (no transformation).
- Polynomial of power m: $\kappa(\vec{u}, \vec{v}) = (1 + \vec{u} \cdot \vec{v})^m$
  The corresponding mapping assigns to $\vec{u} \in \mathbb{R}^n$ the vector $\phi(\vec{v})$ in $\mathbb{R}^{\binom{n+m}{m}}$.
- Gaussian (radial-basis function): $\kappa(\vec{u}, \vec{v}) = e^{-\frac{\|\vec{u}-\vec{v}\|^2}{2\sigma^2}}$
  The corresponding mapping ϕ maps $\vec{u}$ to an *infinite-dimensional* vector $\phi(\vec{u})$ 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.

Kernel SVM examples



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.

Comments on SVM

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

Comments on SVM

- 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.

Comments on SVM

- 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.

Comments on SVM

- 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.

Comments on SVM

- 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]

Comments on SVM

- 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.