UNIVERSITY OF LEEDS

Class: Machine Learning

Support Vector Machines

Instructor: Matteo Leonetti

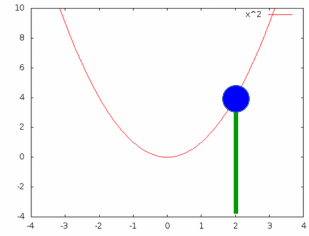
Learning outcomes

UNIVERSITY OF LEEDS

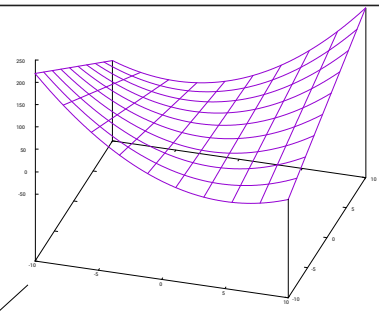
- Derive the dual formulation of Support Vector Machine
- Explain the kernel trick
- Apply dual SVMs and the kernel trick to datasets.

The Dual Problem

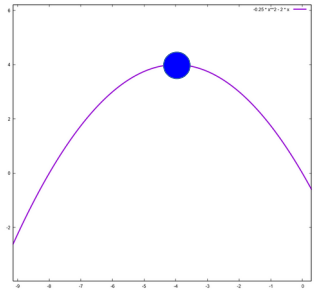
UNIVERSITY OF LEEDS



$$\min f(x)=x^2 \quad \text{s.t.} \quad x=2$$



$$L(x,\lambda)=x^2+\lambda(x-2)$$



$$\max q(\lambda)=-\frac{1}{4}\lambda^2-2\lambda$$

$$f(2)=q(-4)=4$$

We were looking at Lagrangian and duality.

KKT Conditions

UNIVERSITY OF LEEDS

$\min\ f(\boldsymbol{x})$

$\min\ f(\boldsymbol{x})$

Subject to

$h_i(\boldsymbol{x})=0\ \ \forall i=1,\ldots,m$

$\min\ f(\boldsymbol{x})$

Subject to

$h_i(\boldsymbol{x})\leq 0\ \ \forall i=1,\ldots,m$

Corresponding system of equations

$\nabla_x f(\boldsymbol{x})=0$

$\nabla_x L(\boldsymbol{x},\boldsymbol{\lambda})=0$

$\nabla_\lambda L(\boldsymbol{x},\boldsymbol{\lambda})=0$

$\nabla_x L(\boldsymbol{x},\boldsymbol{\lambda})=0$

$\lambda_i g_i(\boldsymbol{x})=0\ \ \forall i=1,\ldots,n$

$\lambda_i\geq 0\ \ \forall i=1,\ldots,n$

We derived the KKT conditions for inequality constraints (see previous slide deck).

Duality and SVM

UNIVERSITY OF LEEDS

What is the dual formulation of this?

minimise: $\frac{1}{2}\|\boldsymbol{w}\|^2$

Subject to the constraints: $t_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) \geq 1$

Follow the Duality Recipe

UNIVERSITY OF LEEDS

1. compile constraints into the Lagrangian

$$\min\ f(x)=x^2$$
$$\text{s.t.}\ -x-3\le 0$$
$$x+2\le 0$$

$$\downarrow$$

$$L(x,\boldsymbol{\lambda})=x^2+\lambda_1(-x-3)+\lambda_2(x+2)$$

$$\min\ \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{s.t.:}\ t_i(\boldsymbol{w}^T\boldsymbol{x_i}+w_0)\ge 1$$

$$\downarrow$$

$$\min\ \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{s.t.:}\ 1-t_i(\boldsymbol{w}^T\boldsymbol{x_i}+w_0)\le 0$$

$$\downarrow$$

$$L(\boldsymbol{w},w_0,\boldsymbol{\lambda})=\frac{1}{2}\|\boldsymbol{w}\|^2+\sum_{n=1}^{N}\lambda_n(1-t_n(\boldsymbol{w}^T\boldsymbol{x}_n+w_0))$$

Now we want to do the same dual formulation with support vector machines. What we did with the example of the parabola is on the left, just for reference.

The first step is to put the constraints in canonical form, and then construct the Lagrangian.

Follow the Duality Recipe

UNIVERSITY OF LEEDS

2. solve for the optimal primal variables

$$L(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \lambda_n (1 - t_n(\boldsymbol{w}^T \boldsymbol{x_n} + w_0))$$

$$\nabla_x L(x, \boldsymbol{\lambda}) = 2x - \lambda_1 + \lambda_2 = 0$$

$$x = \frac{\lambda_1 - \lambda_2}{2}$$

$$\nabla_{\boldsymbol{w}} L = \boldsymbol{w} - \sum_{n=1}^{N} \lambda_n t_n \boldsymbol{x_n} = 0 \qquad \boldsymbol{w}^* = \sum_{n=1}^{N} \lambda_n t_n \boldsymbol{x_n}$$

$$\frac{\partial L}{\partial w_0} = -\sum_{n=1}^{N} \lambda_n t_n = 0$$

The second step is to solve for the primal variables (x in the example, w in the SVM) using the derivative with respect to them.

This gives us an expression of the vector of weights in terms of the Lagrange multipliers. When we solve for w_0, however, it disappears from the equation, and we are left with an equation that we can't use to substitute the value of w_0. This equation still has to be satisfied by our solution, so we carry it forward as an additional constraint emerged from the formation of the dual problem.

Follow the Duality Recipe

UNIVERSITY OF LEEDS

3. substitute the solution for x

$$L(x,\boldsymbol{\lambda}) = x^2 + \lambda_1(-x-3) + \lambda_2(x+2)$$

$$x = \frac{\lambda_1 - \lambda_2}{2}$$

$$q(\boldsymbol{\lambda}) = -\frac{1}{4}\lambda_1^2 - \frac{1}{4}\lambda_2^2 - 3\lambda_1 + 2\lambda_2 + \frac{1}{2}\lambda_1\lambda_2$$

Lastly, we need to substitute the value for x (w in the case of SVM) we just found into the Lagrangian.

This was easy for the parabola, while it is a little more involved with the SVM...

Follow the Duality Recipe

UNIVERSITY OF LEEDS

3. substitute the solution for w and w_0

$$L(\boldsymbol{w}, w_0, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{n=1}^{N} \lambda_n (1 - t_n(\boldsymbol{w}^T \boldsymbol{x_n} + w_0))$$

$$\boldsymbol{w}^* = \sum_{n=1}^{N} \lambda_n t_n \boldsymbol{x}_n \qquad \sum_{n=1}^{N} \lambda_n t_n = 0$$

$$L(\boldsymbol{\lambda}) = \frac{1}{2}\|\sum_n \lambda_n t_n x_n\|^2 + \sum_n \lambda_n (1 - t_n((\sum_k \lambda_k t_k x_k) x_n + w_0))$$

$$= \frac{1}{2}\|\sum_n \lambda_n t_n x_n\|^2 + \sum_n \lambda_n - \sum_n \lambda_n t_n w_0 - \sum_n \lambda_n t_n (\sum_k \lambda_k t_k x_k) x_n$$

$$= \frac{1}{2}\|\sum_n \lambda_n t_n x_n\|^2 + \sum_n \lambda_n - \underbrace{\sum_n \lambda_n t_n w_0}_{=0} - (\sum_n \lambda_n t_n x_n)(\sum_k \lambda_k t_k x_k)$$

This is the substitution of w, followed by some simplifications and rearrangements of the function.

This is never on any book, and the dual formulation just appears out of nowhere. I worked out all the necessary steps in this slide.

The first red arrow indicates that the term that is cancelled out is zero because of the constraint we found before and decided to carry forward.

The second red arrow indicates that the circled term on the right is just the same as the square of the norm that appears on the left (you can expand the square of the norm if you want to convince yourself…), and therefore they can be summed.

Formulations

UNIVERSITY OF LEEDS

Min	Max
$\frac{1}{2}\|\boldsymbol{w}\|^2$	$\sum_{n=1}^{N}\lambda_n-\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m t_n t_m \boldsymbol{x}_n^T \boldsymbol{x}_m$
Subject to	Subject to
$t_i(\boldsymbol{w}^T x_i+w_0)\geq 1$	$\lambda_n\geq 0$
	$\sum_{n=1}^{N}\lambda_n t_n=0$
$\boldsymbol{w}^*=\sum_{n=1}^{N}\lambda_n t_n \boldsymbol{x}_n$	
$w_0=\frac{1}{N_s}\sum_{j\in\text{support vectors}}(t_j-\boldsymbol{w}^t \boldsymbol{x}_j)$	$w_0=\frac{1}{N_s}\sum_{j\in\text{support vectors}}(t_j-\sum_{i=1}^{N}\lambda_i t_i \boldsymbol{x}_i^T \boldsymbol{x}_j)$
$y(\boldsymbol{x})=\boldsymbol{w}^T \boldsymbol{x}+w_0$	$y(\boldsymbol{x})=\sum_{n=1}^{N}\lambda_n t_n \boldsymbol{x}^T \boldsymbol{x}_n+w_0$

And so we get to the dual formulation of SVMs!

Here we see the primal formulation on the left and the dual on the right.

Note how the optimal vector w^* is entirely determined by the points in the dataset for which the corresponding λ is non-zero, that is, the support vectors!

One can create the dual, solve it and then obtain the weights to do the classification, or do the classification in the dual form directly (last row of the slide).

The vector of weights can be obtained from the substitution that we computed from the Lagrangian. However, we could not find an expression for w_0.

This is not a problem though, because we know that every constraint that represents a support vector is verified at the equality. Therefore, it is an equation whose only unknown is w_0 (since we have w from the substitution), and can be solved for w_0.

This is true in principle, while in practice it is subject to numerical errors, especially if the points are affected by noise. Therefore, it is common, in practice, to use all the equations of the support vectors to solve for w_0 and then compute the average.

The primal formulation of SVM is *parametric* method, because the number of parameters in the vector w is predetermined (it's the number of features). However, the dual formulation si *non-parametric*, because the number of λs is not pre-determined, it's the number of support vectors in the dataset, and its a property of the dataset. For comparison, this of the other non-parametric method we say: nearest neighbour. That one also uses the dates, but it need to memorize the entire dataset, while the dual SVM only needs the support vectors!

Dual problem

UNIVERSITY OF LEEDS

Max $$L(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m t_n t_m \boldsymbol{x}_n^T \boldsymbol{x}_m$$

$$\lambda_n \geq 0$$

$$\sum_{n=1}^{N} \lambda_n t_n = 0$$

The input vectors only appear **multiplied**

To classify:

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} \lambda_n t_n \boldsymbol{x}^T \boldsymbol{x}_n + w_0$$

The dual form has an important characteristic: the input variables only appear multiplied.

We are about to see how to take advantage of this, which will justify our journey through the Lagrangian to obtain the dual formulation of SVMs.

Remember how we expanded the dimensions...

UNIVERSITY OF LEEDS

The original dataset has 1 variable:

$$\langle x_{1,} t_1 \rangle, \langle x_{2,} t_2 \rangle, \ldots, \langle x_N, t_N \rangle$$

But we want a higher dimensional space...

Let's use polynomial features: $\Phi_i(x) = x^i$

Our points become:

$$\langle 1, x_{1,} x_{1,}^2 x_{1,}^3 \ldots, x_{1,}^d t_1 \rangle, \langle 1, x_{2,} x_{2,}^2 x_{2,}^3 \ldots, x_{2,}^d t_2 \rangle, \ldots, \langle 1, x_N, x_N^2, x_N^3, \ldots, x_N^d, t_N \rangle$$

We discussed before how to expand the number of dimensions of a data point through a set of *basis* functions. For instance, we took a 1-dimensional dataset, and transformed it into a data set in as many dimensions as we want by using polynomials as basis functions.

We saw that given enough many dimensions a dataset can be made linearly separable.

Features

UNIVERSITY OF LEEDS

Substitute with features

$$L(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m t_n t_m \boldsymbol{\Phi}(x_n)^T \boldsymbol{\Phi}(x_m)$$

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} \lambda_n t_n \boldsymbol{\Phi}(x)^T \boldsymbol{\Phi}(x_n) + w_0$$

So, if we wanted to use our extended dimensions for SVMs, we would need to compute the dot product of the extended vectors of features.

Here is where we are going to do a magic trick.

Experience the power of kernels

UNIVERSITY OF LEEDS

Given the two 2 dimensional points:

$$x = \langle 1, -1 \rangle, y = \langle -1, 2 \rangle$$

Compute the order 2 features:

$$\Phi(x) = \langle 1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2 \rangle$$

Compute the dot product:

$$\Phi(x)^T \Phi(y) = ?$$

Evaluate:

$$(1 + \boldsymbol{x}^T \boldsymbol{y})^2 = ?$$

Let's consider the feature vector Φ given above, with a particular set of basis functions.

We have two 2-dimensional points, and can expand them into two 7-dimensional points.

Compute the dot product of the two extended vectors.

Now, use the formula in the last row, in which you compute the dot product of the two, original, 2-dimensional points. You will see that the result is the same, even though you had to do much fewer multiplications for this one.

This is not a coincidence!

Example: Polynomial features

UNIVERSITY OF LEEDS

$$\begin{bmatrix}1 & x_1 & x_1^2\end{bmatrix}\cdot\begin{bmatrix}1\\ x_2\\ x_2^2\end{bmatrix}=1+x_1x_2+x_1^2x_2^2$$

Note that: $(1+x_1x_2)^2=1+2x_1x_2+x_1^2x_2^2$

Which is close!
If it wasn't for that factor of 2...

But wait, the features can be whatever we want...

$$\begin{bmatrix}1 & \sqrt{2}x_1 & x_1^2\end{bmatrix}\cdot\begin{bmatrix}1\\ \sqrt{2}x_2\\ x_2^2\end{bmatrix}=1+2x_1x_2+x_1^2x_2^2=(1+x_1x_2)^2$$

The set of basis functions that we just used has been specifically constructed so that its dot product can be computed efficiently from the original points.

Kernels

UNIVERSITY OF LEEDS

$$k(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{\Phi}(\boldsymbol{x})^T \boldsymbol{\Phi}(\boldsymbol{y})$$

$k(\boldsymbol{x},\boldsymbol{x}) = (1+\boldsymbol{x}^T\boldsymbol{y})^s$ — Polynomials

$k(\boldsymbol{x},\boldsymbol{y}) = \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma}\right) = \exp(-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|^2)$ — "Gaussian" (RBF)

$k(\boldsymbol{x},\boldsymbol{y}) = \tanh(\kappa\,\boldsymbol{x}^T\boldsymbol{y}-\delta)$ — Sigmoid

These functions that compute the dot product of two vectors in the extended space by using the vectors in the original space are called *kernels*.

Kernels have been created for several sets of basis functions, here we see three common kernels, that correspond to an extended vector made of polynomials, gaussians, and sigmoids respectively.

The Gaussian kernel is particularly interesting because it is easy to compute, and corresponds to a feature space with infinite dimensions! With the Gaussian kernel we are guaranteed to be able to find a linear separation in feature space for any dataset. Beware of the risk of overfitting though...

Constructing kernels

UNIVERSITY OF LEEDS

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

The interesting thing is that once we have a few kernel functions, they can be composed using the rules above to create even more kernel functions.

At this point we only know that the result of the kernel function is the dot product of some extended vector of features… but we may not be able to tell what that vector looks like!

However, we never need to work in the extended vector space. We ever only need the dot product of two vectors in such a space, which is what the kernel function does.

So, computing $k(x_1, x_2)$ is the same as expanding x_1 and x_2 to more dimensions (we may not even know exactly how many…) and compute the dot product there. However, we can do this without ever really expanding the dimensions explicitly.

This is called the ***kernel trick***.

Kernels

UNIVERSITY OF LEEDS

Substitute with kernels!

$$L(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m t_n t_m k(x_n, x_m)$$

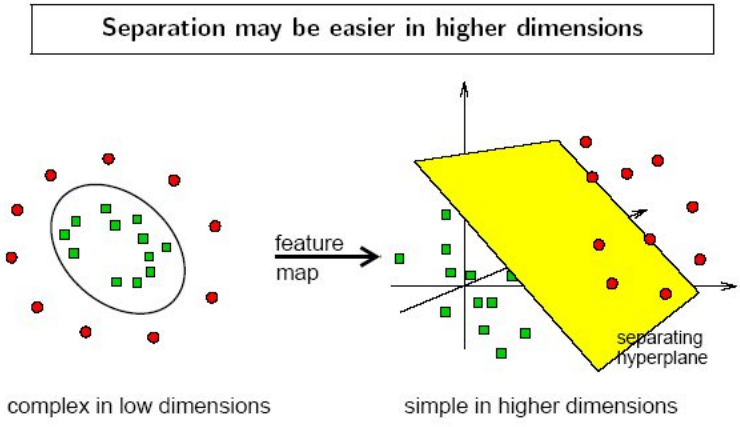
$$y(\boldsymbol{x}) = \sum_{n=1}^{N} \lambda_n t_n k(x, x_n) + w_0$$

We can use the kernel trick on SVMs, effectively expanding the data set into as many dimensions as we want, without ever computing the expanded vectors!

The support vectors, now, exist in the expanded space. So if we look at the corresponding “unexpanded” points they may not look like they are the closest to the decision boundary. However, they are the closest to the extended boundary in the extended space! I know, this is a bit mind bending.

Spaces

UNIVERSITY OF LEEDS



This is a graphical representation of the idea.

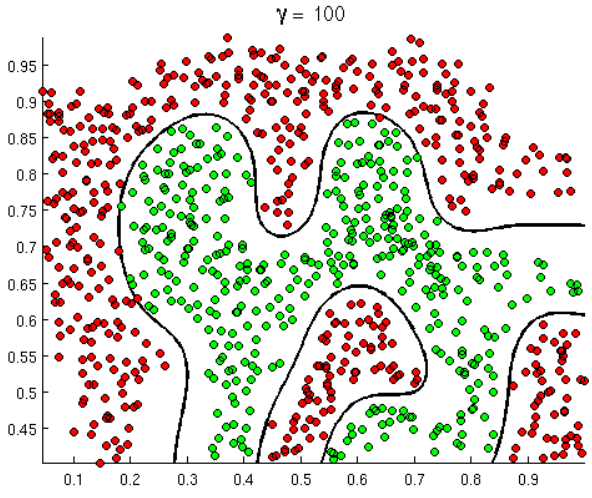
The dataset is not linearly separable in the original dimensions.

Through the kernels, the support vectors are identified in a much higher-dimensional space. In this space, we only need to compute the dot products of the points, and we can do so without ever expanding the vectors of the points. We may even use a space with infinite dimensions! This would of course be impossible if we had to expand each point into an infinite vector…

Note how the hyperplane is in the high-dimensional space, and therefore the support vectors are the closest point to **that** hyperplane. You never get to "see" the hyperplane though. Therefore, if you see the support vectors in their original space, they won't be the points that are closest to the boundary. Their **projection** is closest to the boundary in the high-dimensional space.

Example

UNIVERSITY OF LEEDS



Gaussian kernel

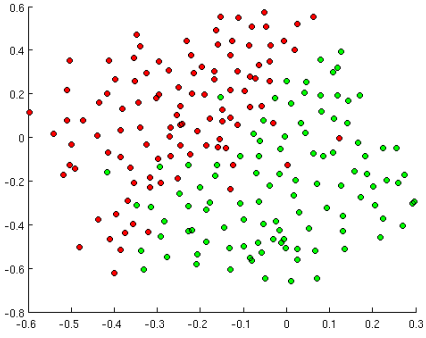
[from Andrew Ng's ML class]

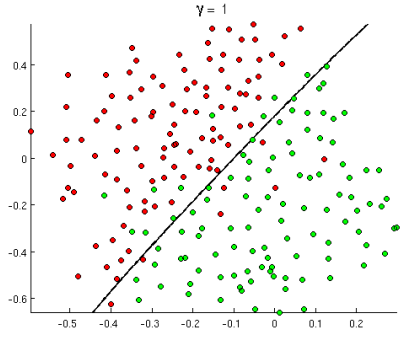
These are example of what you can achieve with a Gaussian kernel, which expands into infinite dimensions.

If you are curious about the infinite dimensions, which we are not going to see in class, a beautiful explanation is here https://www.youtube.com/watch?v=XUj5JbQihlU starting at 22:43.

Example 2

UNIVERSITY OF LEEDS





Gaussian kernel

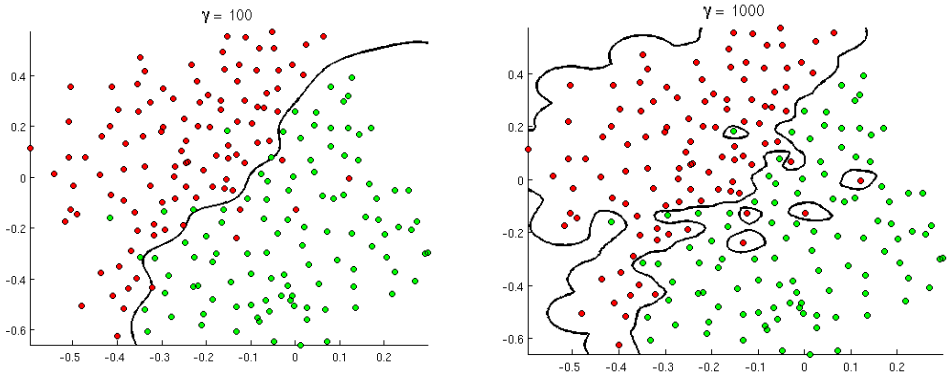
[from Andrew Ng's ML class]

Through the parameter γ you can regulate the area affected by the support vectors, and from that the complexity of the solution and its generalisation.

For low values of γ the decision boundary is almost a straight line (with slack variables, otherwise this wouldn't be a solution…).

Example 2

UNIVERSITY OF LEEDS



Gaussian kernel

[from Andrew Ng's ML class]

For high values of gamma you can get a much more complex function, up to overfitting.

History

UNIVERSITY OF LEEDS

- 1963 - Vladimir Vapnik, Alexey Chervonenkis

- 1992 - Isabelle Guyon

  Proposed the dual formulation with the kernel trick

- 1995 – Corinna Cortes (now head of Google Research)

  Proposed the soft-margin SVM

(They all worked together in the 90s at Bell Labs)

Read this interesting article by Guyon herself:
https://www.kdnuggets.com/2016/07/guyon-data-mining-history-svm-support-vector-machines.html

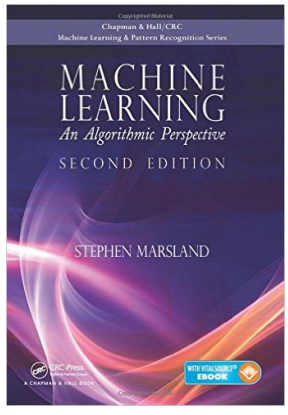
Conclusion

Learning outcomes

UNIVERSITY OF LEEDS

- Derive the dual formulation of Support Vector Machine
- Explain the kernel trick
- Apply dual SVMs and the kernel trick to datasets.

UNIVERSITY OF LEEDS



Chapter 8.2