Machine Learning

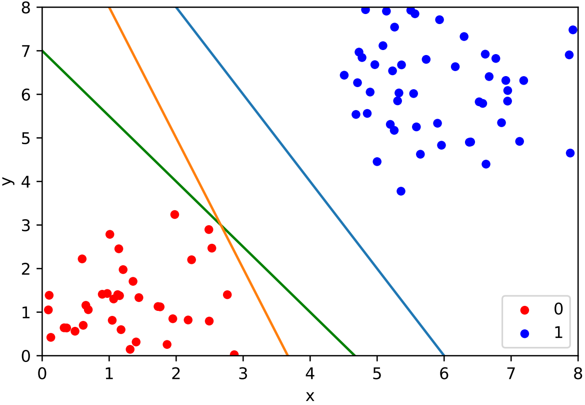
Lecture 5
Support Vector Machines

Justin Pearson[1]
mailto:it-1dl034@lists.uu.se

[1] http://user.it.uu.se/~justin/Hugo/courses/machinelearning/

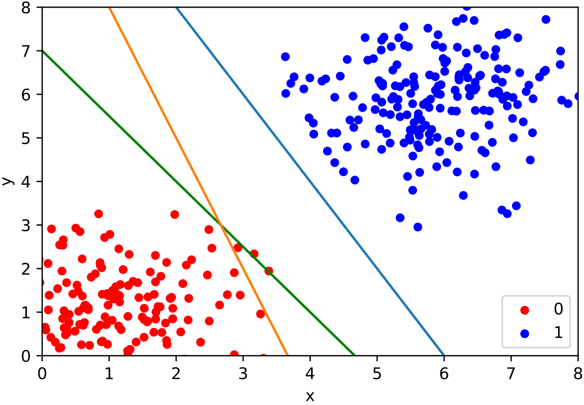
Separating Hyperplanes

Logistic regression (with linear features) finds a hyperplane that separates two classes. But which hyperplane is best?



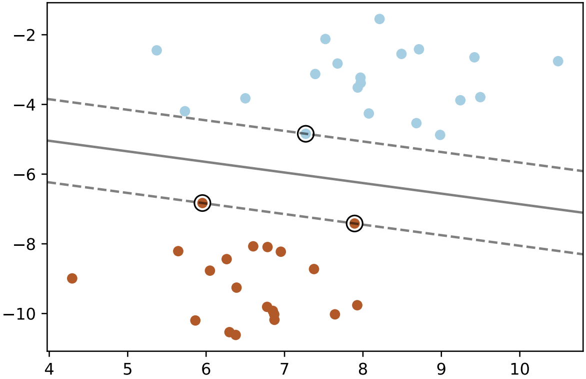
Separating Hyperplanes

It of course depends on how representative your training set is. With more points from the distribution our hyperplanes might look like:



Margin Classifiers

The intuition is that we find a hyperplane with a margin either side that maximises the space between the two clusters.



Support Vector Machine

- They have been in use since the 90s.
- More robust with outliers.
- Very good classifiers on certain problems such as image classification, handwritten digit classification.
- Non-linear models can be incorporated by the kernel trick.

Plan of action

- A motivation/modification of logistic regression.
- Finding margins as an optimisation problem.
- Different Kernels for non-linear classification.

Logistic Regression

Remember the error term for logistic regression

$$-y\log(\sigma(h_\theta(x))) - (1-y)\log(1-\sigma(h_\theta(x)))$$

Where

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Expanding the error term

$$-y\log(\sigma(h_\theta(x))) - (1-y)\log(1-\sigma(h_\theta(x)))$$

Equals

$$-y\log(\frac{1}{1+e^{h_\theta(x)}}) - (1-y)\log(1-\frac{1}{1+e^{h_\theta(x)}}))$$

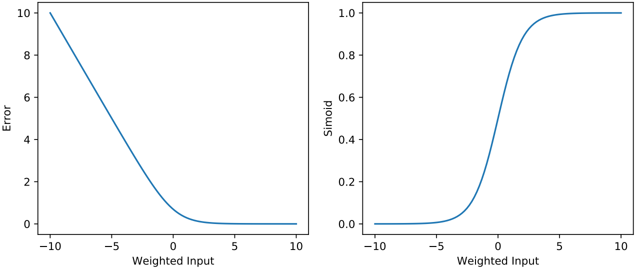
Remember that the two log terms are trying to force the model to learn 1 or 0.

Looking at the contribution

Just looking at

$$-y \log(\sigma(h_\theta(x)))$$

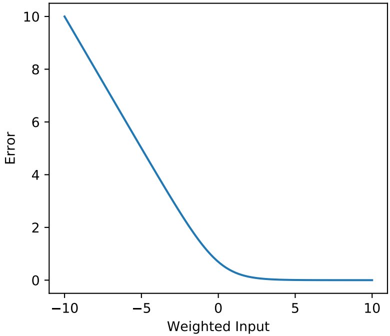
We are trying to force the term $\sigma(h_\theta(x))$ to be 1. The larger the value of θx the less the error.

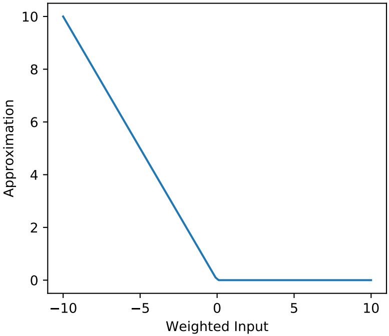


After 0 we do not really care we just want to force move the input over to the right

Approximating the error

Instead of using the logistic error we could approximate it with two linear functions.





After 0 we do not really care about the error.

Support Vector Machines

I am sorry, but to make the maths easier and to be consistent with the support vector machine literature but we are going to change our classification labels a bit.

We have data

$$x = x^{(1)}, \ldots, x^{(m)}$$

Where the data are points in some d dimensional space $\mathbb{R}^d$.

The labels for our classes will be -1 and 1 instead of 0 and 1.

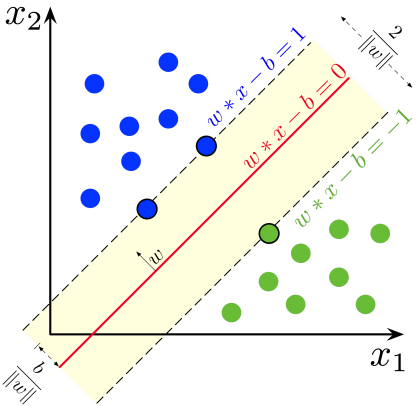
Linear Support Vector Machine

Linear SVMs are the easiest case and form the foundation for support vector machines. We want to find weights $\overline{w} \in \mathbb{R}^d$ and a constant b such that

$$\begin{cases} \overline{w} \cdot \overline{x} - b \geq 1 & \text{if } y = 1 \\ \overline{w} \cdot \overline{x} - b \leq -1 & \text{if } y = -1 \end{cases}$$

This is different from logistic regression or a single perceptron where you want to find a separating hyperplane.

SVM Margins in 2 dimensions[2]



If we push the two hyperplanes apart then we will eventually hit points in the two classes. The points that are on the two out hyperplanes are called the support vectors.
So the question is, how do we do this?

[2] Picture taken from wikipedia

Normal Vectors

- In two dimensions a vector normal to another vector is one at 90 degrees.

  In particular $\overline{w}$ and $\overline{x}$ are normal if the dot product equals 0.

$$\overline{w} \cdot \overline{x} = 0$$

- If you have a hyperplane defined by $\overline{w} \cdot \overline{x} = 0$ then the $\overline{w}$ is normal to that hyperplane.
- Similarly, if you have a hyperplane defined by $\overline{w} \cdot \overline{x} = b$ then the $\overline{w}$ is normal to that hyperplane.

Derivation

- The vector $\overline{w}$ is perpendicular to the hyperplanes. In particular the hyperplane $\overline{w}x - b = 0$.
- Given a two points $\overline{x_1}$ where $\overline{w} \cdot \overline{x_1} - b = -1$ and x_2 $\overline{w} \cdot \overline{x_2} - b = 1$.
- We want to know the distance between the two points. We can treat them as vectors and do the maths.

Assume that $\overline{x_2}$ and $\overline{x_1}$ are directly opposite each other.

Derivation

- $\overline{x_2} - \overline{x_1}$ is a vector, it has some length t and is in the direction $\overline{w}$. So $\overline{x_2} - \overline{x_1} = t\frac{\overline{w}}{||\overline{w}||}$.
- The vector $\frac{\overline{w}}{||\overline{w}||}$ is a unit vector.
- Now doing some rearranging

$$(\overline{w} \cdot \overline{x_2} - b) - (\overline{w} \cdot \overline{x_1} - b) = 1 - (-1) = 2$$

So

$$\overline{w} \cdot (\overline{x_2} - \overline{x_1}) = 2$$

Which gives

$$\overline{w} \cdot t\frac{\overline{w}}{||\overline{w}||} = 2$$

Derivation

Since $\overline{w} \cdot \overline{w} = ||w||^2$ we get that

$$\overline{w} \cdot t \frac{\overline{w}}{||\overline{w}||} = \frac{||w||^2}{||w||} = 2$$

So the distance between the two boundaries is

$$\frac{2}{||w||}$$

Maximising the margin

Thus to maximise the distance between the two hyperplanes $\overline{w}x - b = 1$ and $\overline{w}x - b = -1$ we want to maximise

$$t = \frac{2}{||\overline{w}||}$$

So we minimise $\frac{1}{2}||\overline{w}||$

SVMs optimisation problem for learning

Given a training set $x^{(1)}, \ldots, \ldots x^{(m)}$ We want to minimise $\frac{1}{2}||\overline{w}||$ such that for all data in the training set

$$\begin{cases} \overline{w} \cdot \overline{x^{(i)}} - b \geq 1 & \text{if } y^{(i)} = 1 \\ \overline{w} \cdot \overline{x^{(i)}} - b \leq -1 & \text{if } y^{(i)} = -1 \end{cases}$$

Since $y^{(i)}$ can only be -1 or $+1$ we can rewrite the constraint as

$$y^{(i)}(\overline{w} \cdot \overline{x^{(i)}} - b) \geq 1$$

and minimize

$$\frac{1}{2}\overline{w} \cdot \overline{w} = \frac{1}{2}||\overline{w}||^2$$

Minimising Subject to a Constraint

- So we have to minimise

  $$\frac{1}{2}\overline{w} \cdot \overline{w}$$

  subject to the constraint

  $$y^{(i)}(\overline{w} \cdot \overline{x^{(i)}} - b) \geq 1$$

To do this you have to use the technique of Lagrangian multipliers and dual problems. It is out of scope of the course, but it is possible.

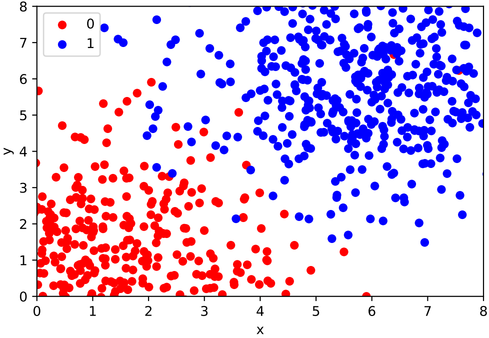
Quadratic programming

If you use Lagrangian multipliers you end up with a quadratic programming problem. Don't worry if you don't know what it is, but they are non-linear problems that are sometimes well behaved.

- Gradient descent will not work.
- Your optimisation problem includes lots of quadratic terms, but luckily the problem is convex.
- Quadratic programming solves this, and in the convex cases there are nice mathematical properties that give you bounds on the errors. How this all works is out of scope of the course.

Non-linearly separable sets

What happens if our clusters overlap? The quadratic programming model will not work so well.



Slack Variables

For each point in the training set introduce a slack variable η_i and rewrite the optimisation problem as

- Minimise $\frac{1}{2}||\overline{w}|| + C\sum_i \eta_i$ such that

$$y^{(i)}(\overline{w} \cdot \overline{x^{(i)}} - b) \geq 1 - \eta_i$$

An η_i greater than 0 allows the point to be miss-classified. Minimising $C\sum_i \eta_i$ for some constant C reduces the number of miss-classifications. The greater the constant C the more importance you give reducing the number of miss-classifications.

Kernels and Non-linear Classifiers

Warning what follows in the slides is not a complete description of what is going on with Kernels.
In particular I am not going to explain how the learning algorithm works.
To understand this, you need to know a bit about quadratic programming, Lagrange multipliers, dual bounds and functional analysis.
I am instead going to try to give you some intuition why kernels might work. This if often called the kernel trick. I will also try to give you some intuition how and what SVMs learn with Gaussian Kernels.

Making the non-linear linear

We already saw that with Linear regression we could learn non-linear functions. To learn a quadratic polynomial we take out data

$$\left(\begin{pmatrix}1\\x\end{pmatrix} \in \mathbb{R}^2\right) \mapsto \left(\begin{pmatrix}1\\x\\x^2\end{pmatrix} \in \mathbb{R}^3\right)$$

One way of thinking about this is that we add invent new features. When computing the gradients everything worked.

We turned a non-linear problem of trying to find a quadratic polynomial that minimises the error into a linear problem of trying to learn the coefficients. The problem with this is that it increases the number of dimensions that you have to work in.

The curse of dimensionality

Short story. The more dimensions you have the worse it gets.

- Often things become exponentially hard in the number of dimensions. Things such as complexity, convergence time, number of samples required etc.

So the less dimensions the better.

Non-linear to Linear

This is part of a general scheme.
We have a non-linear separation problem in low dimensions, we find a transformation that embeds our problem into a high-dimensional space.
One possibly misleading way of thinking about this, is that the more dimensions you have the more room you have, and so it is easier for the problem to be linear.

$$\mathbb{R}^d \xrightarrow{\Phi} \mathbb{H}$$

Where $\mathbb{H}$ is some higher[3] dimensional space.

[3] Actually $\mathbb{H}$ stands for Hilbert not higher, but do not worry about this.

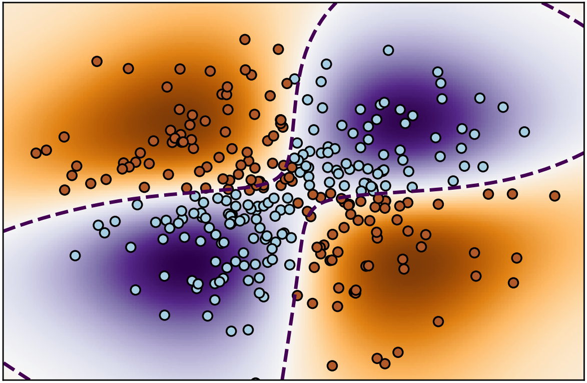
Non-Linear to Linear : Naive Scheme

$$\mathbb{R}^d \xrightarrow{\Phi} \mathbb{H}$$

- Find a good function Φ.
- Take your training data in $\mathbb{R}^d$ and transform it via Φ to the space $\mathbb{H}$.
- Train a linear classifier on the transformed data.
- Then given a data point $\overline{x} \in \mathbb{R}^d$ ask the trained linear classifier waht class $\Phi(\overline{x})$ belongs to.

Lots of problems with this approach. If $\mathbb{H}$ has a lot of dimensions of even infinite dimensional then training a linear classifier over the transformed training set is hard.

Example Decision Boundary



Predict the XOR of two inputs using a Gaussian kernel. Remember that XOR is not linearly separable.

Linear Hypotheses

A linear hypothesis $h_{\overline{w}}(\overline{x})$ has the form

$$h_{\overline{w}}(\overline{x}) = \sum_{i=1}^{d} w_i x_i + w_0 = \overline{w} \cdot \overline{x} + w_0$$

Where $\cdot$ is the inner (dot) products.
In our learning algorithms there are a lot of inner product calculations.

Non-linear to Linear

So to learn linear things in $\mathbb{H}$ we will need to do inner products.

$$\mathbb{R}^d \xrightarrow{\Phi} \mathbb{H}$$

We will need do lots of calculations on $\Phi(x_i) \cdot \Phi(x_j) \in \mathbb{H}$.

The Kernel Trick

Computing the inner products $\Phi(x_i) \cdot \Phi(x_j) \in \mathbb{H}$ can be computationally expensive, or even worse your space could be infinite dimensional[4]. For well behaved transformations Φ there exists a function $K(x, y) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that

$$K(x, y) = \Phi(x) \cdot \Phi(y)$$

Thus we can compute the inner product in the high-dimensional space by using a function on the lower dimensional vectors.

[4] Don't worry if your head hurts.

Some common Kernels

Instead of giving the higher-dimensional space you often just get the function K.

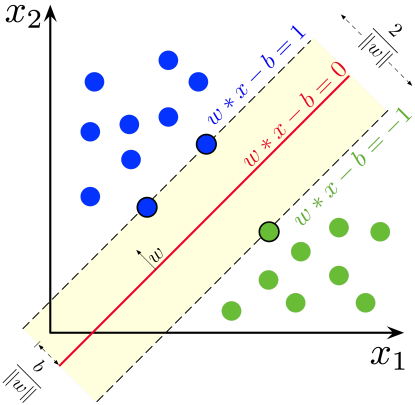
Radial basis or Gaussian $K(x, y) = \exp(-(x - y)^2/2\sigma^2)$

Polynomial $K(x, y) = (1 + x \cdot y)^d$

Sigmoid or Neural Network $K(x, y) = \tanh(\kappa_1 x \cdot y + \kappa_2)$

There are lots more, there are even kernels for text processing. If you are going to invent your own then you will need to understand the maths.

Support Vectors



I am not going to explain in any detail, but it is enough to remember the support vectors.

The dual version of the SVM learning

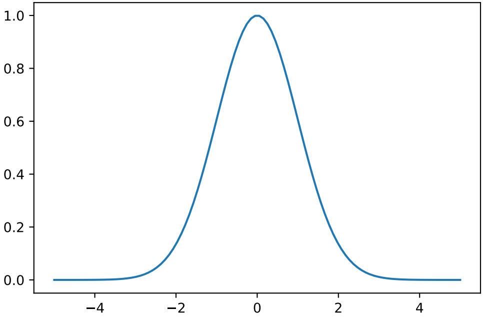
Learning with Kernels does the same thing, you find support vectors. The dual version of the algorithm learns some parameters α_i, y_i and b such that to decide if a point x belongs to a class you compute the sign of

$$\sum_{i=1}^{N_s} \alpha_i y_i K(s_i, x) + b$$

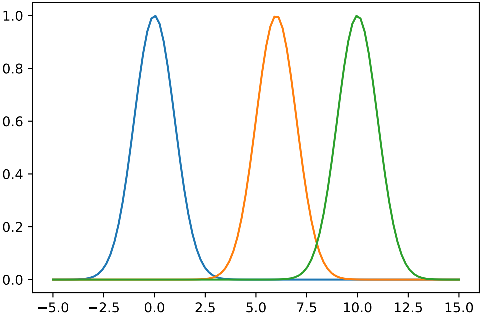
Where $s_1, \ldots, s_{N_s}$ are the support vectors.

Gaussian Kernels

This is a all a bit abstract. I'll try to explain what is going on with Gaussian Kernels. In one dimension for $\sigma = 1$ our Gaussian kernel $K(x, x') = \exp(-(x - x')^2/2)$ if we fix x' to be 0. We get the following graph

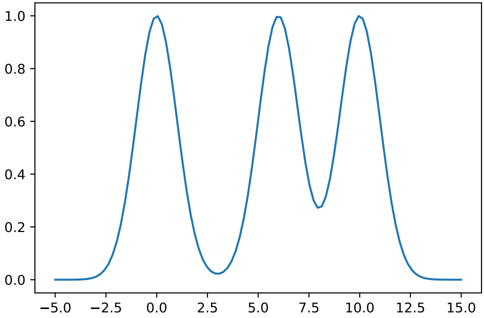


Gaussian Kernels - Multiple Support Vectors



Gaussian Kernels - Multiple Support Vectors

If we assume that all out weights are 1 and add them together. We get



The closer you value is to one of the peaks the more likely it is that you are in the class. You can think of each peak as a feature.

Support Vector Machines — What are the good for?

There are some nice theoretical guarantees that you can get with SVMs, but that is out of scope of this course.
Non-linear Kernels SVMs have successfully been used in many applications including

- Image recognition, bio-informatics, pattern recognition.
- Because the optimisation problem is convex there is only going to be one global minimum. So SVMs are easier to train than neural networks.
- Probably the most useful non-linear Kernel function is the Gaussian Kernel.
- For the moment, don't worry too much about how the SVM learns with non-linear Kernels, but just use an existing implementation.