COMP30027 Machine Learning
Support Vector Machines

Semester 1, 2019

Jeremy Nicholson & Tim Baldwin & Karin Verspoor

Lecture Outline

Nearest Prototype Classification

- A parametric variant of nearest-neighbour classification is the **nearest prototype**, whereby we calculate the centroid of each class, and classify each test instance according to the class of the centroid it is nearest to
- The centroid is calculated simply by averaging the numeric values along each axis:

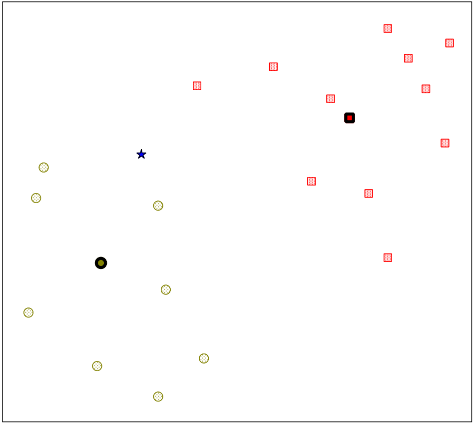
$$\textit{for a class } C_j = \{x_i : \langle a_{i,1}, a_{i,2}, \ldots, a_{i,D} \rangle\},$$

$$\textit{the prototype } P_j = \langle a_1^*, a_2^*, \ldots, a_D^* \rangle$$

$$\textit{where each } a_k^* = \sum_{i=1}^{M} \frac{a_{i,k}}{M}$$

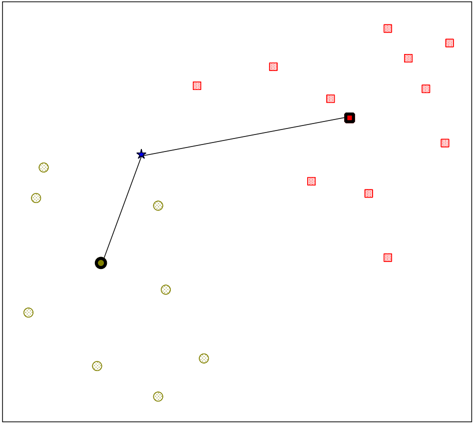
Nearest Prototype Classification

- Classification is then based on simple Euclidean distance:



Nearest Prototype Classification

- Classification is then based on simple Euclidean distance:



Lecture Outline

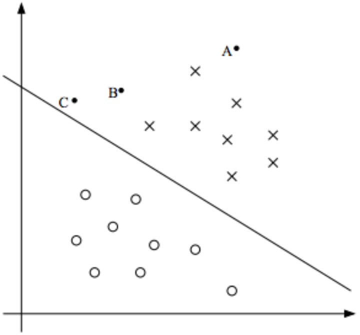
What is a Support Vector Machine?

A support vector machine is a non-probabilistic binary linear classifier.

- A (linear) hyperplane-based classifier for a two-class classification problem
- The particular hyperplane it selects is the *maximum margin* hyperplane
- *Soft margins* allow some data points to violate the separating hyperplane
- A *kernel function* can be used to allow the SVM to find a non-linear separating boundary between two classes

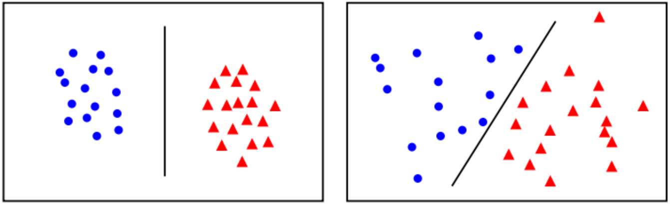
What is a Support Vector Machine?

The goal is to find a hyperplane that separates two classes.

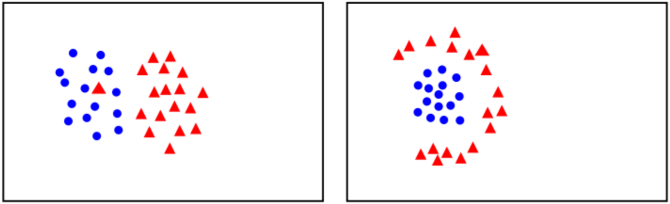


Linear separability

linearly
separable



not
linearly
separable



Linear classifiers I

A separating hyperplane in D dimensions can be defined by a **normal** $\boldsymbol{w}$ and an **intercept** b

In 3-D (a plane): $cx + dy + ez + b = 0, \boldsymbol{w} = \langle c, d, e \rangle$

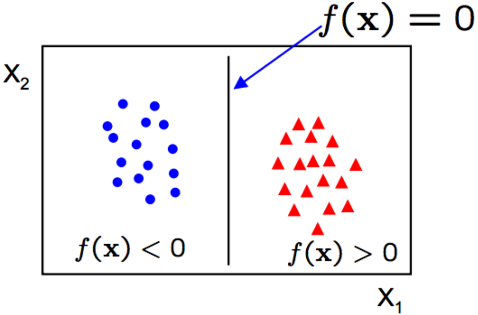
More generally, $\boldsymbol{w} = \langle w_1, w_2, ...w_m \rangle$
And a point $\boldsymbol{x} = \langle x_1, x_2, ...x_m \rangle$
So, the hyperplane equation is:

$$\begin{aligned} w_1 x_1 + w_2 x_2 ... w_m x_m + b &= 0 \\ \boldsymbol{w} \cdot \boldsymbol{x} + b &= 0 \end{aligned}$$

(b is occasionally called w_0, in which case it is also known as a **bias**)
(These are column vectors, by convention, and the dot product is written $\boldsymbol{w}^T \boldsymbol{x} = w_1 x_1 + w_2 x_2 + \ldots + w_m x_m = \boldsymbol{w} \cdot \boldsymbol{x}$)

Linear classifiers II

- A linear classifier takes the form $f(x) = \boldsymbol{w}^T\boldsymbol{x} + b$
- In 2D, this is a line:

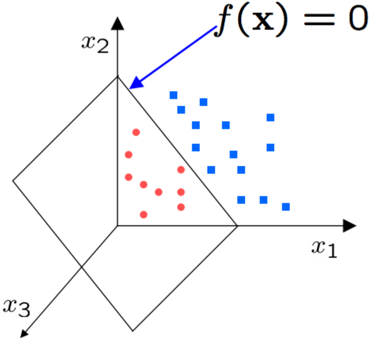


Linear classifiers III

- A linear classifier takes the form $f(x) = \boldsymbol{w}^T\boldsymbol{x} + b$
- In 3D, this is a plane:

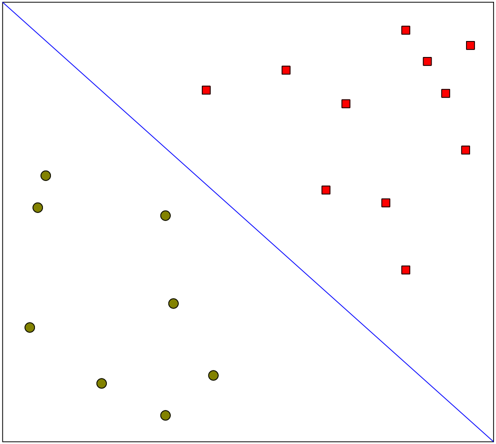
For a k-NN classifier it was necessary to 'carry' the training data.

For a linear classifier, the training data is used to learn $\boldsymbol{w}$ (the "weight vector") and then (mostly) discarded.



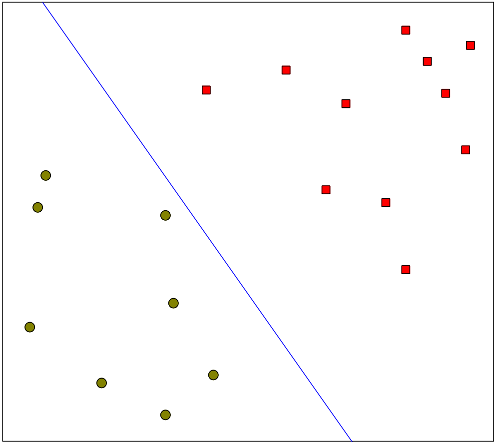
SVMs: Maximum Margin

- One solution:



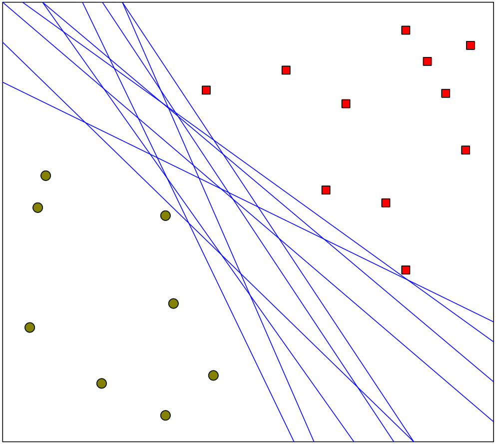
SVMs: Maximum Margin

- Another solution:



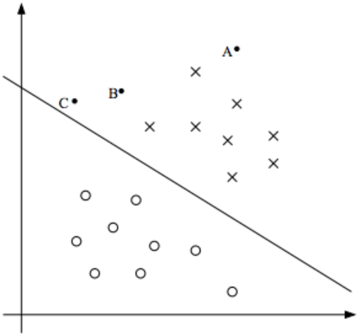
SVMs: Maximum Margin

- Lots more solutions:



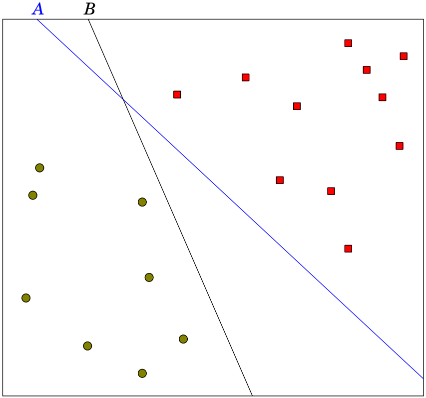
Margins

- For point A, we should be quite confident about the prediction of its class.
- For point C, a small change to the decision boundary might change our decision to change; we are less confident in the prediction.



SVMs: Maximum Margin

- How can we rate the different decision boundaries to work out which is "best" (e.g. is A "better" than B)?



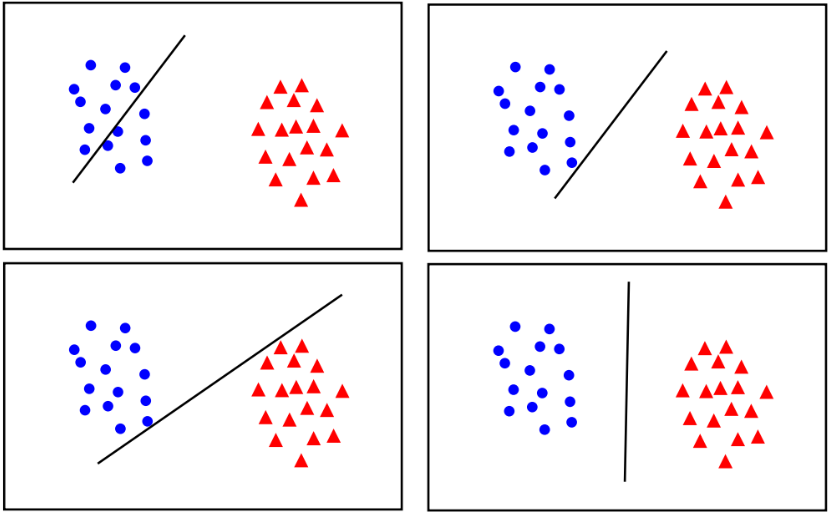
Optimal solution

For a given training set, we would like to find a decision boundary that allows us to make all correct and confident (far from the decision boundary) predictions on the training examples.

Some methods find a separating hyperplane, but not the optimal one. SVM finds an optimal solution.

- Maximizes the distance between the hyperplane and the "difficult points" close to decision boundary
- *Intuition:* if there are no points near the decision surface, then there are no very uncertain classification decisions

What is the best hyperplane?

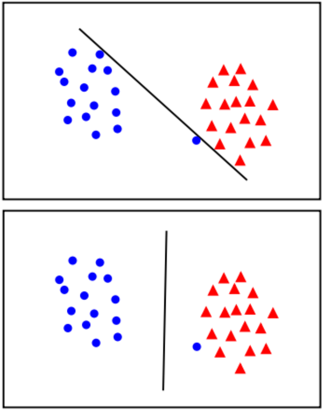


Maximum margin solution: most stable under perturbations of the inputs

What is the best hyperplane? Soft margins

Possibly large margin solution is better even though one constraint is violated

Trade-off between the margin and the number of mistakes on the training data



SVM-based classification

- Associate one class as positive (+1), and one as negative (-1)
- Find the best hyperplane $\boldsymbol{w}$ and b, which maximises the margin between the positive and negative training instances (the **model**)
- To make a prediction for a test instance $\boldsymbol{t} = t_1, t_2, \ldots t_n$:
  - Find the sign of $f(t) = \boldsymbol{w}^T \boldsymbol{t} + b$
  - Sometimes we assign "?" to instances within the margin
  - The value of $f(t)$ can be transformed into a "probability", with some extra work

Learning the SVM

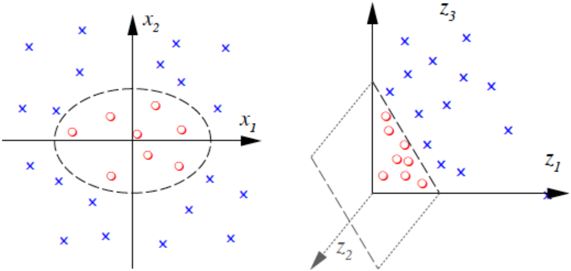
For small training sets, we can use a naive training method:

- Pick a plane $\boldsymbol{w}$ and b
- Find the worst classified sample y_i
  (Note: This step is computationally expensive for large data sets)
- Move plane $\boldsymbol{w}$ and/or b to improve the classification of y_i
- Repeat steps 2-3 until the algorithm converges

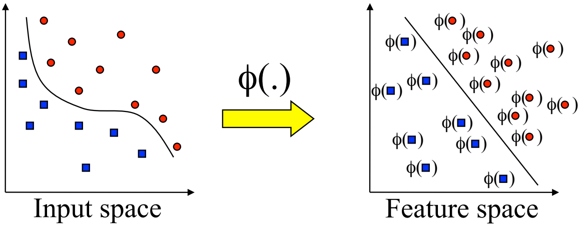
If the data isn't linearly separable

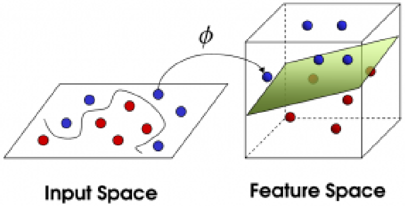
To obtain a non-linear classifier, we can transform our data by applying a mapping function, and then apply a linear classifier to the new feature vectors.

$$\Phi : R^2 \rightarrow R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2) x_1 x_2, x_2^2)$$



Kernel function





- Make non-separable problem separable.
- Map data into better representational space.

Lecture Outline

Formal specification of SVM

Let the input be a set of N training vectors $\{\boldsymbol{x_k}\}_{k=1}^{N}$ and corresponding class labels $\{y_k\}_{k=1}^{N}$, where $\boldsymbol{x_k} \in \mathbb{R}^D$ and $y_k \in \{-1, 1\}$. Initially we assume that the two classes are linearly separable. The hyperplane separating the two classes can be represented as:

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0,$$

such that:

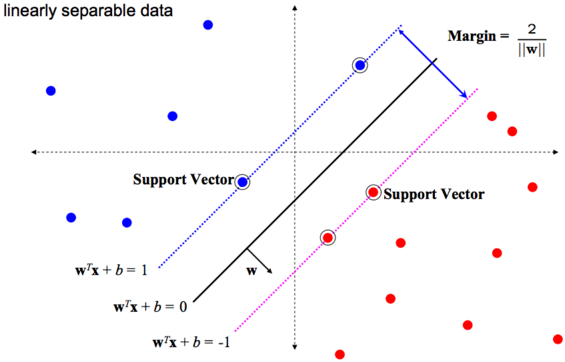
$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}_k + b \geq 1 &\quad \text{for} \quad y_k = +1, \\ \boldsymbol{w}^T\boldsymbol{x}_k + b \leq -1 &\quad \text{for} \quad y_k = -1. \end{aligned}$$

n.b.:

$$y_k(\boldsymbol{w}^T\boldsymbol{x}_k + b) - 1 \geq 0$$

"Support Vectors"

- Objective is to find the data points that act as the boundaries of the two classes.
- These are referred to as the "support vectors".
- They constrain the margin between the two classes.



Optimisation: Maximizing the margin I

- We want to choose $\boldsymbol{w}$ so that the margin $\frac{2}{||\boldsymbol{w}||}$ is maximised, given that all points are on the correct side of the separating hyperplane $y_k(\boldsymbol{w}^T\boldsymbol{x}_k + b) - 1 \geq 0$
- It turns out that maximising $\frac{2}{||\boldsymbol{w}||}$ is inconvenient (the partial derivatives are ugly)
- So we instead minimise $\frac{1}{2}||\boldsymbol{w}||^2 = \frac{1}{2}(w_1^2 + w_2^2 + \ldots + w_n^2)$ (note nicer derivatives)

Optimisation: Maximizing the margin II

- Given the relationship between the margin, and the normalisation factor of the weight vector, maximizing the margin corresponds to minimizing $\|\boldsymbol{w}\|$.
- Determination of model parameters corresponds to a convex quadratic optimisation problem. Any local solution is also a global optimum.

"Slack" — allow soft margins

Now, let's consider the case when the two classes are not (completely) linearly separable. We introduce slack variables $\{\xi_k\}_{k=1}^N$ and allow few points to be on the wrong side of the hyperplane at some cost. The modified objective function:

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{k=1}^{N}\xi_k \\ \text{s.t.} \quad & y_k(\boldsymbol{w}^T\boldsymbol{x_k} + b) + \xi_k - 1 \geq 0, \\ & \xi_k \geq 0, \quad \forall k \in \{1..N\} \end{aligned}$$

The parameter C must be tuned.

Solving the optimisation problem I

- Current state-of-the-art for solving constrained optimisation problems uses the method of Lagrange multipliers, where we introduce a value α_k for each constraint.
- In this case, that means a Lagrange multiplier α_k for every instance in the training set.

Solving the optimisation problem II

Solving the optimisation problem III

The classification function eventually becomes:

$$f(\boldsymbol{t}) = \sum_i \alpha_i y_i \boldsymbol{x}_i^T \boldsymbol{t} + b$$

$$b = y_j(1 - \xi_j) - \sum_i \alpha_i y_i \boldsymbol{x_i}^T \boldsymbol{x_j}$$

- Most α_k are 0; the non-zero values correspond to **support vectors**.
- If we wish to recover $\boldsymbol{w}$ and b, we can do so by only considering the instances with non-zero α_k.
- Effectively, we can ignore every training instance not on the decision boundary at this point.

Solving the optimisation problem IV

If we need a non-linear SVM, we replace our dot product with the corresponding kernel function:

$$f(\boldsymbol{t}) = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{t}) + b$$

$$b = y_j(1 - \xi_j) - \sum_i \alpha_i y_i K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

Everything else is the same!

Lecture Outline

Extending SVM to multiple classes

SVMs are inherently two-class classifiers.
Most common approaches to extending to multiple classes:

- one-versus-all (or one-versus-rest) classification
  choose class which classifies test data point with greatest margin
- one-versus-one classification (one classifier per pair of classes)
  choose class selected by most classifiers

Training time becomes a serious issue, because we need to build *many* SVMs...

Summary and Resources

- SVMs is a high-accuracy *margin classifier*
- Learning a model means finding the best separating hyperplane.
- Classification is built on projection of a point onto a hyperplane normal.
- SVMs have lots of parameters that need to be optimised (slow?).
- SVMs can be applied to non-linearly-separable data with an appropriate kernel function.

http://nlp.stanford.edu/IR-book/pdf/15svm.pdf
Mathematical Formulation:
https://www.youtube.com/watch?v=_PwhiWxHK8o
http://research.microsoft.com/pubs/67119/svmtutorial.pdf

Lecture Outline

1. Nearest Prototype Classification
2. Introduction to SVMs
   - Hyperplane
   - Margins
   - Classification
   - Non-linear SVMs
3. Terrifying Maths
4. Multi-class SVMs
5. Appendix: More Terrifying Maths

Common kernel functions

- Linear kernel

$$K(x_i, x_j) = {x_i}^T x_j$$

- Polynomial kernel

$$K(x_i, x_j) = ({x_i}^T x_j + \theta)^d$$

- Radial basis kernel

$$K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

A kernel function K must be continuous, symmetric, and have a positive definite gram matrix.

Watch a polynomial kernel in action:
https://www.youtube.com/watch?v=3liCbRZPrZA

Why a kernel function, instead of a transformation? I

We could explicity transform our dataset into a higher-order representation. For example, the polynomial kernel of order 2 ϕ_{P2} transforms a vector of m dimensions into a vector of $C(m,2) + 2m + 1 = \frac{m^2}{2} + \frac{3m}{2} + 1$ dimensions:

$$\begin{aligned}
\boldsymbol{x} &: \langle x_1, x_2, \ldots x_m \rangle \rightarrow \\
\phi_{P2}(\boldsymbol{x}) &: \langle 1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_m, x_1^2, x_2^2, \ldots x_m^2, \\
&\quad \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots \sqrt{2}x_{m-1}x_m \rangle
\end{aligned}$$

Why a kernel function, instead of a transformation? II

- In training, we need to find the dot product between all pairs of training instances.
- This is *a lot* of calculations ($\mathcal{O}(DN^2)$, for D attributes and N training instances)
- We have now increased our number of attributes to $\mathcal{O}(D^2)$
-) – :

Why a kernel function, instead of a transformation? III

- A kernel function acts on the un-transformed vectors, but calculates the dot product of the **transformed** vectors
- For example, given 2D vectors $\boldsymbol{x}_i = [x_{i1}, x_{i2}]$ and $\boldsymbol{x}_j = [x_{j1}, x_{j2}]$:

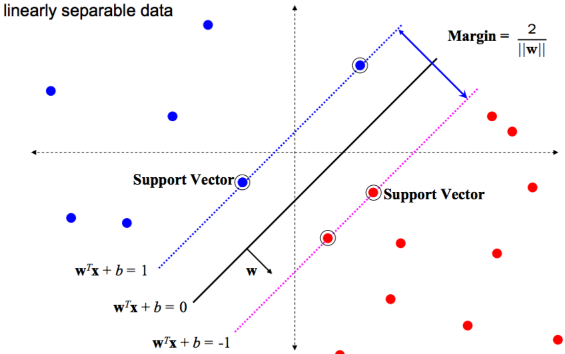
$$
\begin{aligned}
&K_{P2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^T \boldsymbol{x}_j)^2 \\
&= 1 + x_{i1}^2 x_{i2}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} \\
&= [1, x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}]^T [1, x_{j1}^2, \sqrt{2}x_{j1}x_{j2}, x_{j2}^2, \sqrt{2}x_{j1}, \sqrt{2}x_{j1} \\
&= \phi_{P2}(\boldsymbol{x}_i)^T \phi_P 2(\boldsymbol{x}_j)
\end{aligned}
$$

Why a kernel function, instead of a transformation? IV

- Using the polynomial kernel function, we need:
  - The dot product between the two vectors (which we needed to calculate anyway)
  - One extra addition
  - One extra exponentiation
- And we get the dot product between the higher-order vectors
- So, we effectively skip the cost of transformation step, plus all of the (many) extra calculations!

Why is the margin $2/||w||$?

- Since $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ and $c(\boldsymbol{w}^T\boldsymbol{x} + b) = 0$ define the same plane, we can choose the normalisation of $\boldsymbol{w}$
- Choose normalisation such that $\boldsymbol{w}^T\boldsymbol{x}_+ + b = +1$ and $\boldsymbol{w}^T\boldsymbol{x}_- + b = -1$ for positive / negative support vectors, respectively
- Then the margin is given by $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot (x_+ - x_-) = \frac{\boldsymbol{w}^T(x_+ - x_-)}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$



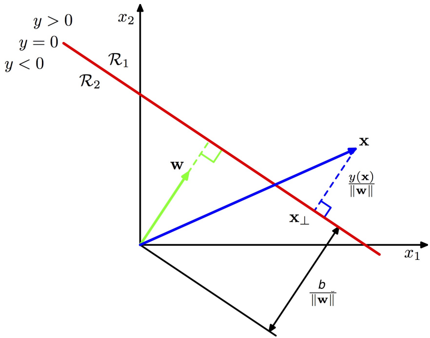
Another look at normalisation and the margin

$y(x) = \boldsymbol{w}^T \boldsymbol{x} + b$

Decision surface (red) is perpendicular to $\boldsymbol{w}$; its displacement from origin is controlled by b

The signed orthogonal distance of a point $\boldsymbol{x}$ from the decision surface is $y(\boldsymbol{x})/\|w\|$

We push the margin in/out by rescaling $\boldsymbol{w}$.
The margin moves out with $\frac{1}{\|\boldsymbol{w}\|}$.



Constrained optimisation I

Given the constrained optimisation problem:

$$
\begin{aligned}
\min_{\boldsymbol{w}} \quad & \frac{1}{2}||\boldsymbol{w}||^2 \\
\text{s.t.} \quad & y_k(\boldsymbol{w}^T\boldsymbol{x_k} + b) - 1 \geq 0 \\
& \forall k \in \{1..N\}
\end{aligned}
$$

Constrained optimisation II

We construct a Lagrangian (called the "primal") that we need to minimise:

$$\mathcal{L} \quad : \quad \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{N} \alpha_i y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) + \sum_{i=1}^{N} \alpha_i$$

All of the partial derivatives must equal 0 to find a minimum:
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$, $\frac{\partial \mathcal{L}}{\partial b}$, $\frac{\partial \mathcal{L}}{\partial \alpha_k}$ (for all $k \in \{1..N\}$
From the first two partial derivatives, we can immediately observe $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$ and $\sum_i \alpha_i y_i = 0$; now we have a *new* set of constraints!

Constrained optimisation III

We construct an equivalent (the "Wolfe dual") formulation through substitution, except now we need to maximise:

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$

Constrained optimisation IV

If we allow soft margins, we have:

$$
\begin{aligned}
\min_{\boldsymbol{w}} \quad & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{k=1}^{N}\xi_k \\
\text{s.t.} \quad & y_k(\boldsymbol{w}^T\boldsymbol{x_k} + b) + \xi_k - 1 \geq 0, \\
& \xi_k \geq 0, \quad \forall k \in \{1..N\}
\end{aligned}
$$

Constrained optimisation V

We can again construct two (equivalent) Lagrangians, now with more multipliers (μ) for the extra conditions over the slack variables (ξ). The primal:

$$\frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)-1+\xi_i] - \sum_{i=1}^{N}\mu_i\xi_i$$

And the dual, where we modify the conditions $\alpha_k \geq 0$ to $0 \leq \alpha_i \leq C$ (but note that the ξ terms are nicely absent):

$$\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j$$

Constrained optimisation VI

And if we have a non-linear SVM? The dot product is replaced by the kernel function:

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Constrained optimisation VII

The most popular solver is Sequential Minimal Optimisation, which attempts to solve the above (dual) formulation numerically by breaking the problem down:

- Choose a Lagrange multiplier which violates the Karush-Kuhn-Tucker conditions (KKT) (remember that a Lagrange multiplier corresponds to an instance)
- Choose a second Lagrange multiplier whose value is neither 0 nor C
- Optimise for just these two multipliers
- Iterate until all multipliers pass the KKT conditions (within tolerance ϵ)

John Platt (1998) Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical Report.