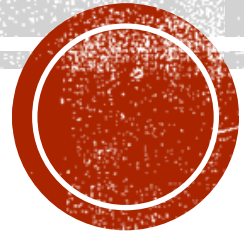
# SUPPORT VECTOR CLASSIFIER

Dr. Brian Mc Ginley



# GENERALISED LINEAR MODELS

- Recall

$$y = mx + c$$

- is the equation of a line.
- Basically, if that equation is satisfied for a point $(x, y)$ then the point falls on the line. We can also describe a line as

$$w_2x_2 + w_1x_1 + w_0 = 0$$

$$\mathbf{w}^T\mathbf{x} + w_0 = 0$$

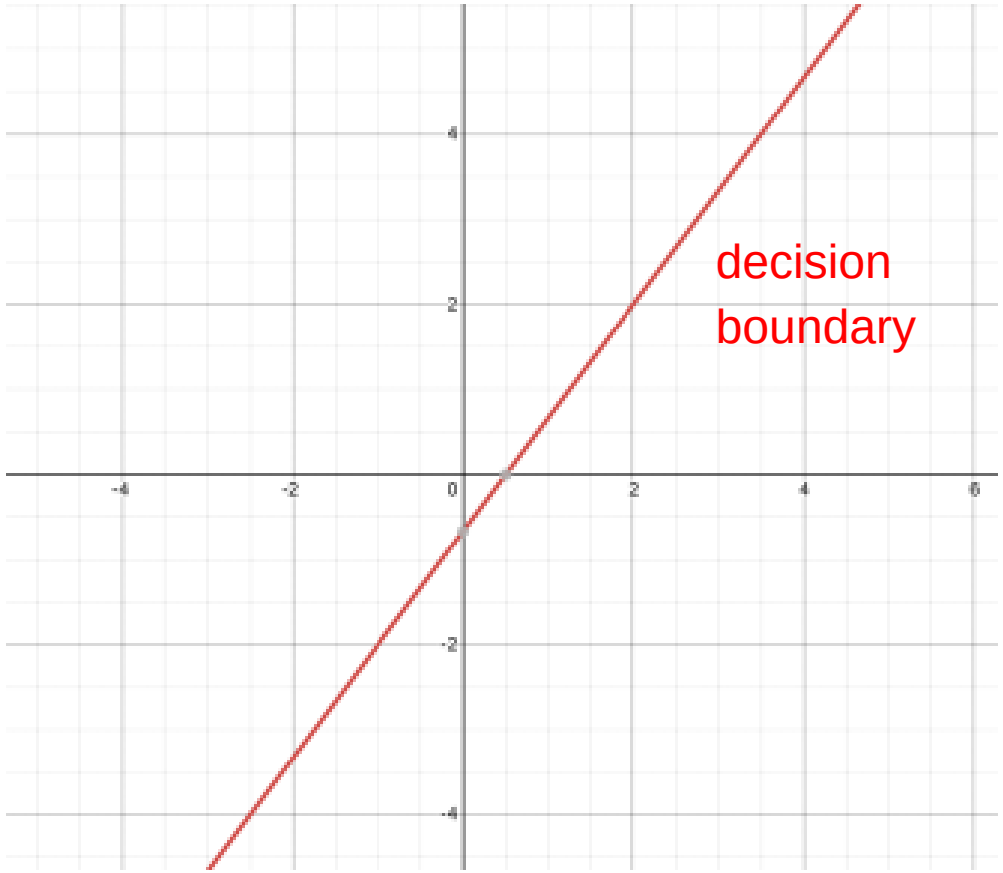
- is also an equation of a line, a vector equation of a line. Whatever way you want to think about it, it described a line - is linear.

if this was a decision boundary, which is what we're going to extend this to represent, what we can see is whether something falls on one side or the other side of the decision boundary and how far from the decision boundary it lies. That's what we're getting to with these support vector classifiers.

# JUNIOR CERT GEOMETRY

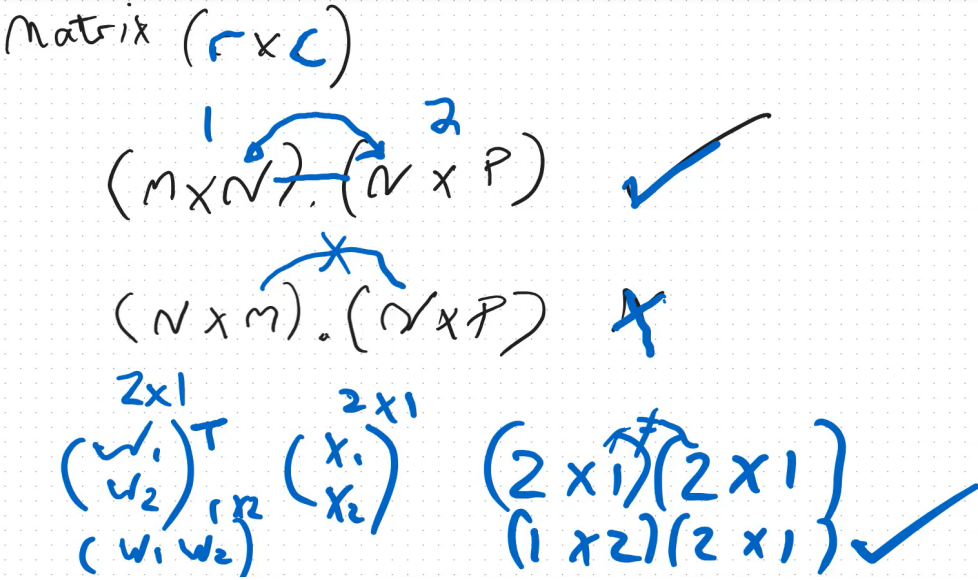
-ve +ve

- The following is the plot of the line $4x - 3y - 2 = 0$
- Clearly from the picture $(2, 2)$ falls on the line and $4(2) - 3(2) - 2 = 0$ so the equation is satisfied showing this.
- Look at the image, the point $(2, -2)$ is clearly to the right of the line. Try the equation and the result is $4(2) - 3(-2) - 2 = 12$. A positive number.
- Now, the point $(-4, 2)$ is clearly to the left of the line. Equation: $4(-4) - 3(2) - 2 = -24$. A negative number.
- So, whether the calculation of the line is positive or negative will tell us which side of the line it falls on.




magnitude of this result, 12 or 24 will tell us how far away from the line you are.

# LET'S EXTEND THIS




- Take $w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ $and$ $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

weight vector

x vector

vectors are expressed in rows by columns

- Put it into the equation of the line: $\boldsymbol{w}^T \boldsymbol{x} + w_0 = (w_1 \quad w_2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + w_0 = w_0 + w_1 x_1 + w_2 x_2$

Transpose

it's never just WX, it's always WTX.

- If $w = \begin{pmatrix} 4 \\ -3 \end{pmatrix}$ $and$ $w_0 = -2$
- Then we have $4x_1 - 3x_2 - 2$
- Then if we evaluate the sample $(2, -2)$, we find it is positive, so we classify as the positive class. $(-4, 2)$ classifies in the negative class. This expands easily to larger dimensions - we just can't visualise past 3!
- The result $\boldsymbol{w}^T \boldsymbol{x} + w_0$ describes a hyperplane in any dimension

in 3D

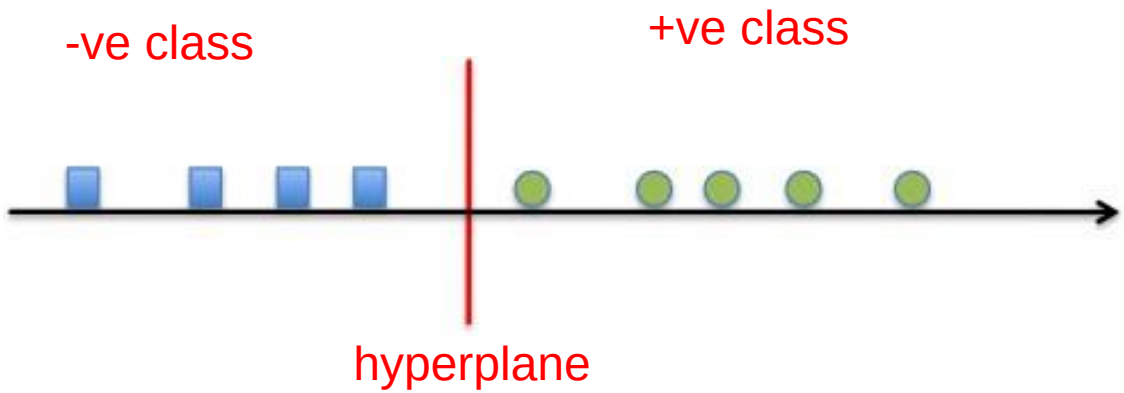
The result, WTX plus 0, describes a line in 2D, plane in 3D, and in higher dimensions than that, it describes a hyperplane. It's a hyperplane that divides 2 spaces.

# HYPERPLANE 1D

- Linear classifier has a linear boundary (hyperplane)

$$\boldsymbol{w}^T \boldsymbol{x} + w_0$$

- which separates the space into two "half-spaces". In 1D this is simply a threshold

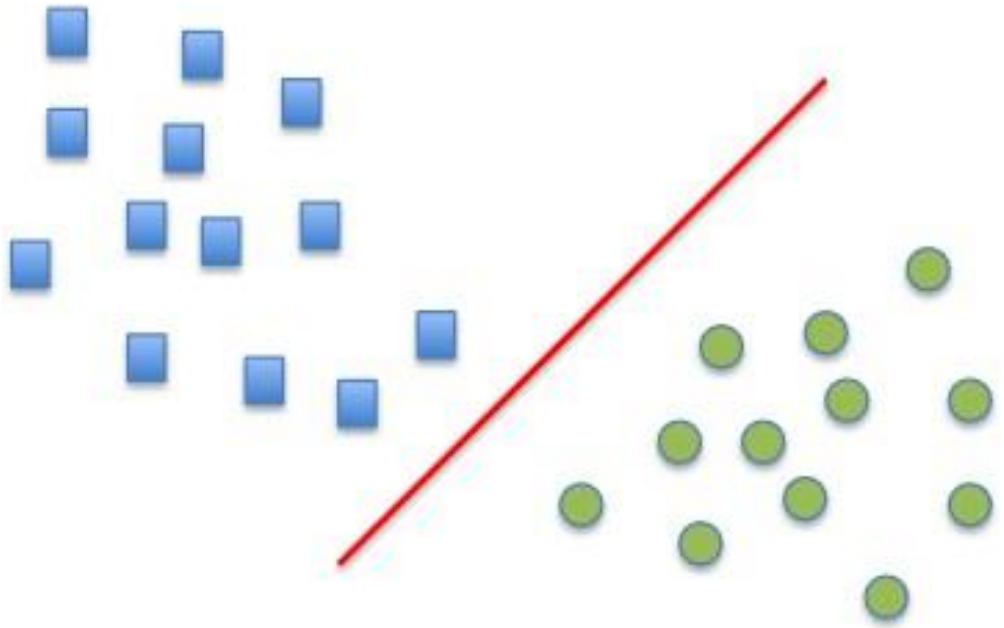


# HYPERPLANE 2D

- Linear classifier has a linear boundary (hyperplane)

$$\boldsymbol{w}^T \boldsymbol{x} + w_0$$

- which separates the space into two "half-spaces". In 2D this is a line

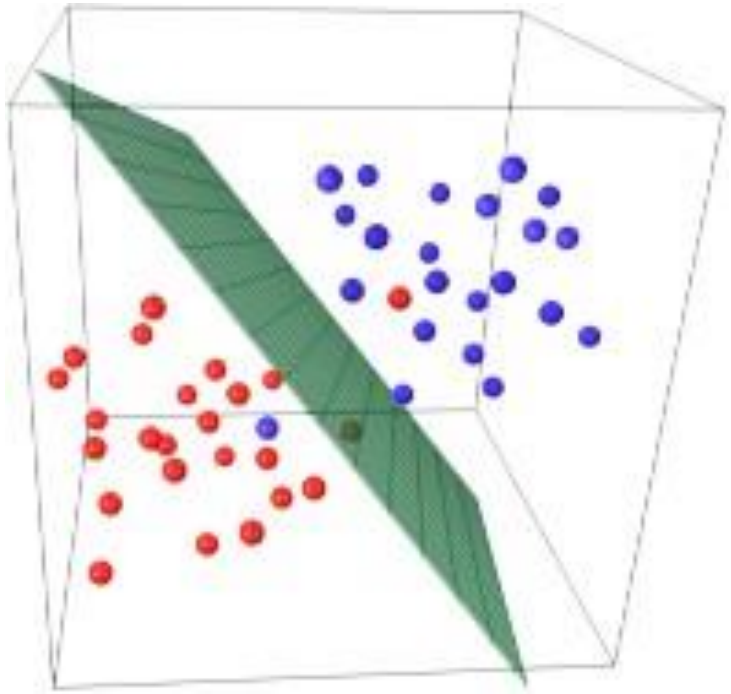


# HYPERPLANE 3D

- Linear classifier has a linear boundary (hyperplane)

$$\boldsymbol{w}^T \boldsymbol{x} + w_0$$

- which separates the space into two "half-spaces". In 3D this is a plane



# GEOMETRY

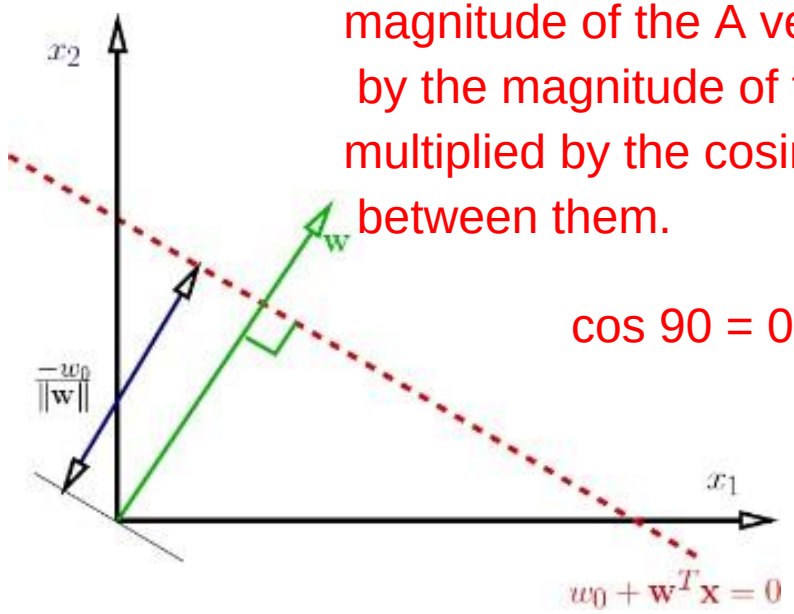
- $\boldsymbol{w}^T \boldsymbol{x} = 0$ is a line/hyperplane passing through the origin and is orthogonal to **w**
- The reason that **w** is orthogonal (perpendicular) to the hyperplane is that – the dot product of any 2 vectors can be 0 only if they're orthogonal (90 degrees)
- Dot product review:

$$\boldsymbol{a}.\boldsymbol{b} = a_1 b_1 + \cdots + a_n b_n \qquad \boldsymbol{a}.\boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| cos\theta$$

- $\boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$ shifts the hyperplane by $\mathrm{w}_0$

magnitude of the A vector multiplied by the magnitude of the B vector multiplied by the cosine of the angle between them.

cos 90 = 0

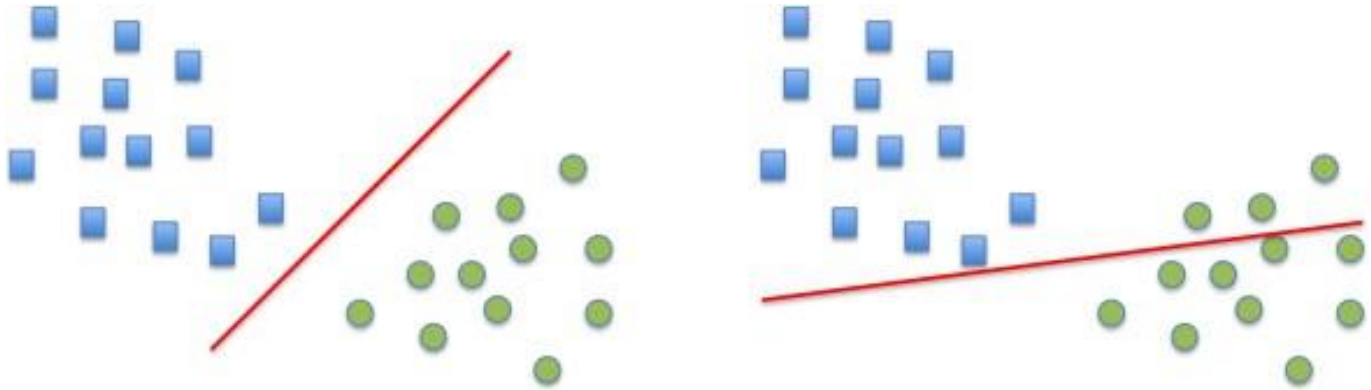



Recall $|\boldsymbol{a}|$ corresponds to the length (magnitude/modulus) of vector a.

# LEARNING LINEAR CLASSIFIERS

Mean squared error, you're trying to get the best fit by trying to minimise the distance between Y predicted and the actual Y,
While here is between the two different classes, we're trying to find a line that maximises the distance between these two things.

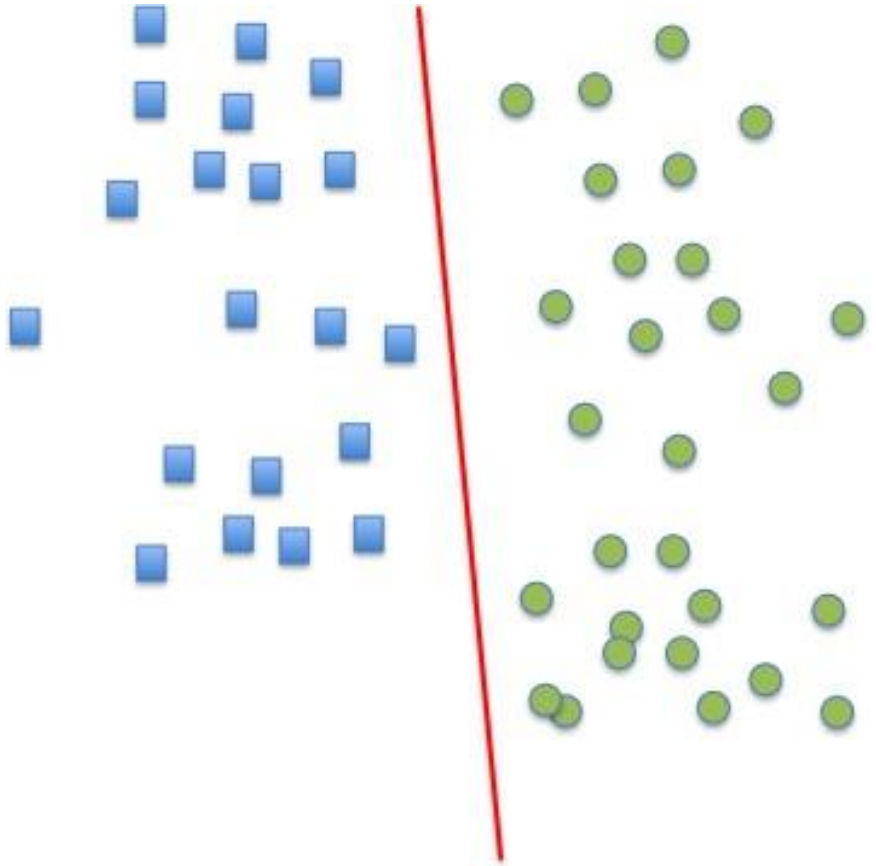
- Learning consists in estimating a "good" decision boundary
- We need to find $\boldsymbol{w}$ (direction) and $w_0$ (location) of the boundary.
- What does "good" mean?
- Is this boundary good? - We need a criteria that tell us how to select the parameters - use a loss function. (MSE)



# SEPARATING CLASSES

Hard margin classifiers only work when your data is perfectly linearly separable

- If we can separate the classes, the problem is **linearly separable**



# SEPARATING CLASSES

if your data say is linear and it's linearly separable, but you're not able to do it

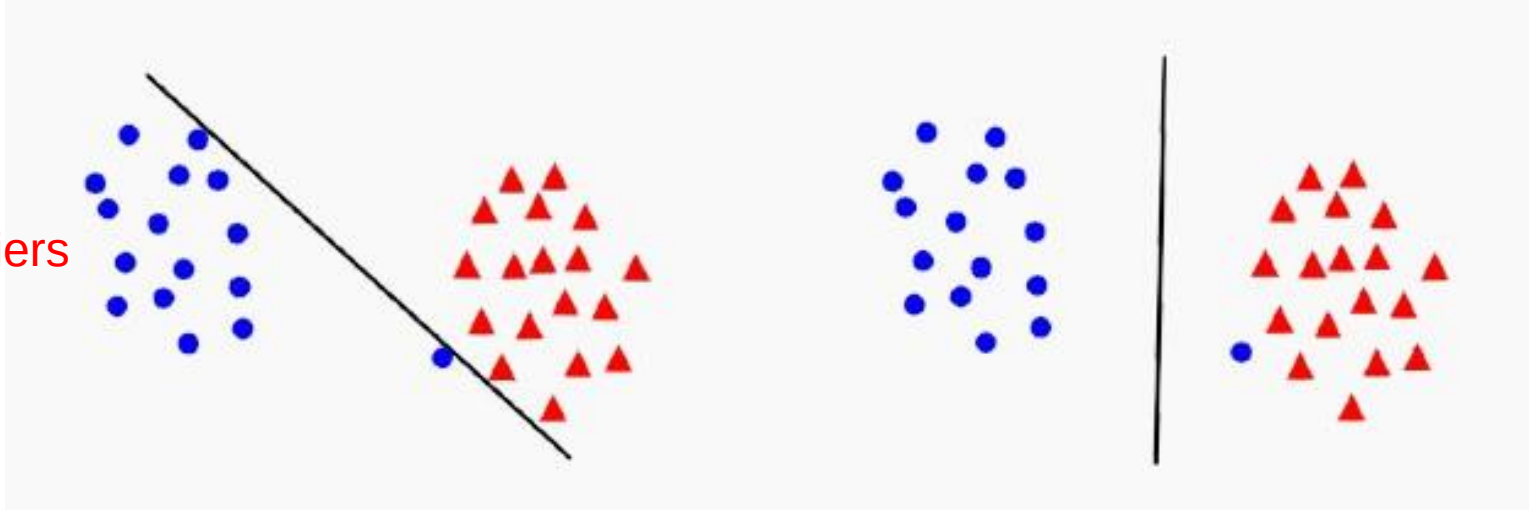
- Causes of non-perfect separation:
  - Model is too simple
  - Noise in the inputs (i.e., data attributes)
  - Simple features that do not account for all variations
  - Errors in data targets (mis-labellings)
- Should we make the model complex enough to have perfect separation in the training data? NO, you don't want to necessarily overtrain to the noise.You want to retain the model's generalisation capabilities

# DECISION BOUNDARIES

- For example, we may select a decision boundary that maximises the margin between both classes
  - Geometrically, this means choosing a boundary that maximizes the distance or margin between the boundary and both classes.
  - This is known as a Maximal Margin/Hard Margin Classifier
  - However, what if the data looks like this?
  - Maximal Margin /Hard Margin Classifiers are very sensitive to outliers and are prone to over-fitting
  - We can consider alternative/relaxed constraints that prevent overfitting.

Bad, blue is in the right decision boundry, but WRONG GROUP
overtrain, overfit to outliers

HARD MARGIN



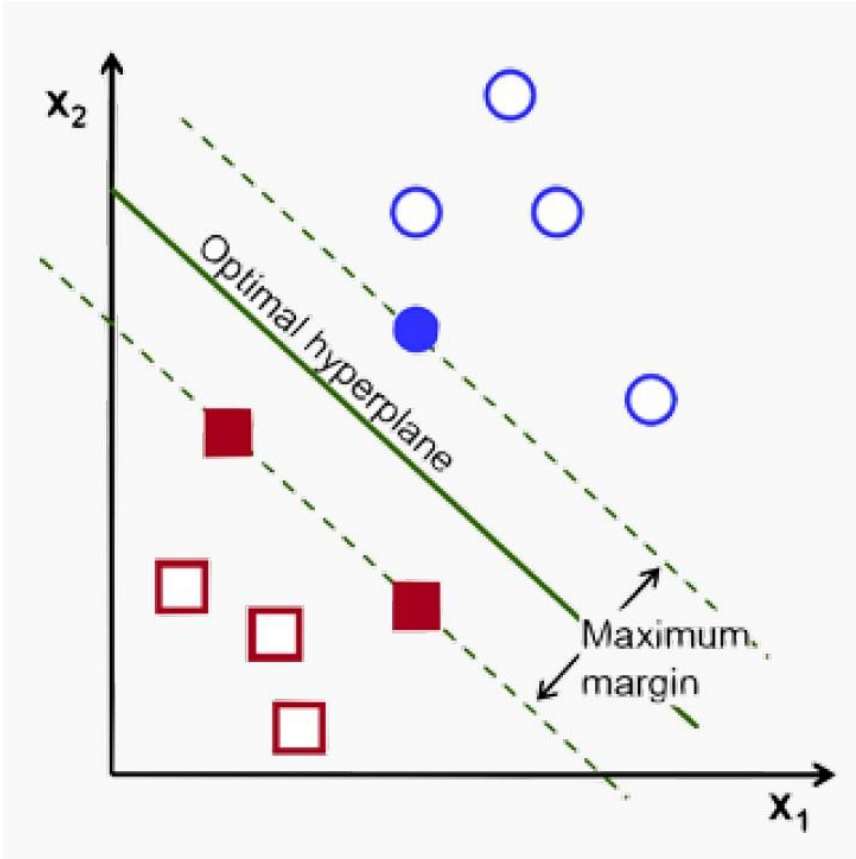
we've got a misclassification, but overall our generalisation performance is BETTER

SOFT MARGIN CLASSIFIER -

# MARGIN

- Definition: The shortest distance between the observations and the hyperplane is called the margin

This example is hard margin




two type of support vector classifiers, hard margin classifiers, which aren't really so often used, soft margin classifiers, which is what a support vector classifier is.

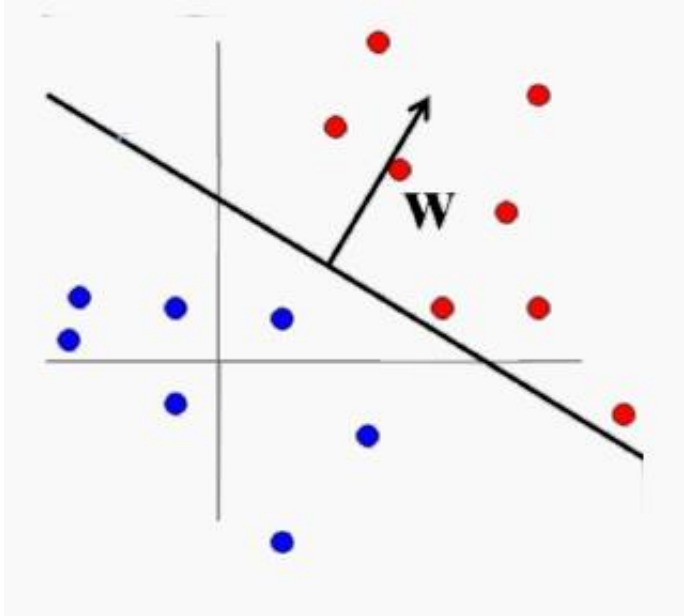
And then support vector classifiers have been extended to support vector machines

# GEOMETRY TO DECISION BOUNDARY

- Recall that the decision boundary is defined by some equation in terms of the predictors. A linear boundary is defined by

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$$

- The non-constant coefficients, **w**, represent a **normal vector**, pointing orthogonally away from the plane



# GEOMETRY TO DECISION BOUNDARY

- Now, using some geometry, we can compute the distance between any point to the decision boundary using $\boldsymbol{w}$ and $w_0$.
- The signed distance from a point $\boldsymbol{x} \in \mathrm{R}^n$ to the decision boundary is

a set of real numbers

distance is aboslute value

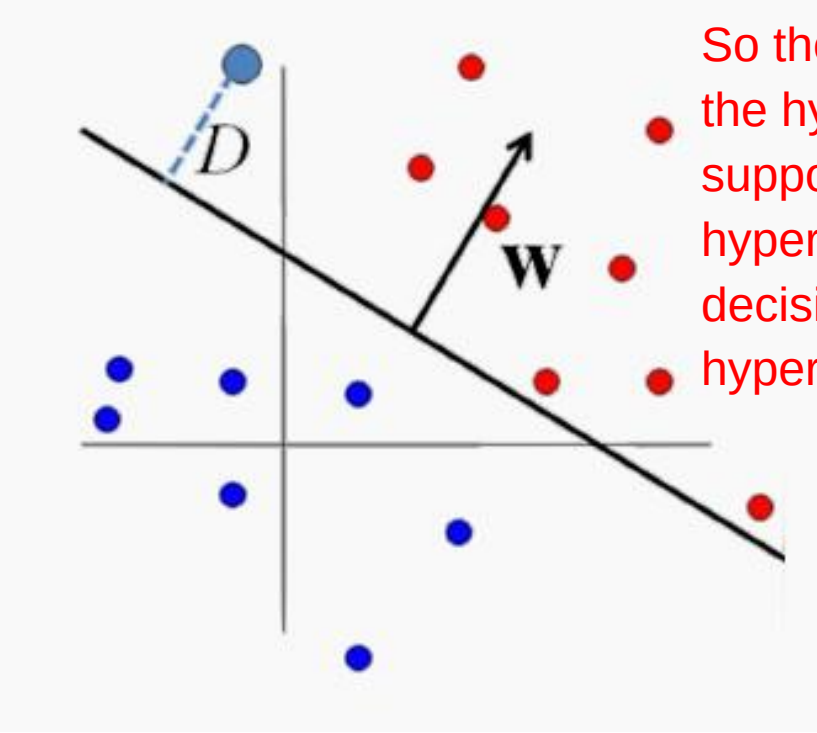
$$D(x) = \frac{\boldsymbol{w}^T \boldsymbol{x} + w_0}{\|\boldsymbol{w}\|}$$

if wtx+b = 1, (s17)

so 1/||w||

- Note: we need the signed distance because we care which side of the hyperplane the observation is on.
- E.g. in 2D (standard equation for distance from point to line):

$$D(x) = \frac{w_0 + w_1 x_1 + w_2 x_2}{\sqrt{w_1^2 + w_2^2}}$$




So the distance from the hyperplane, my support vector hyperplane to my decision boundary hyperplane is 1 / ||W||

# MAXIMISING MARGINS

- So our goal. Find a decision boundary that maximises the distance to both classes.
- A hard margin classifier doesn't maximise the distance of all points to the boundary. Instead, it only maximises the distance to the **closest** points.
- The points closest to the decision boundary are called support vectors.
- This means that only support vectors impact position of the hyperplane. Which training samples are used as support vectors is decided by cross-validation
- For any plane, we can always scale the equation

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = 0$$

  - so that the support vectors lie on the planes (depending on their classes)

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = \pm 1$$

Only support vectors determine the position of the hyperplane of this decision boundary,

# MAXIMISING MARGINS

setting up 2 other hyperplanes

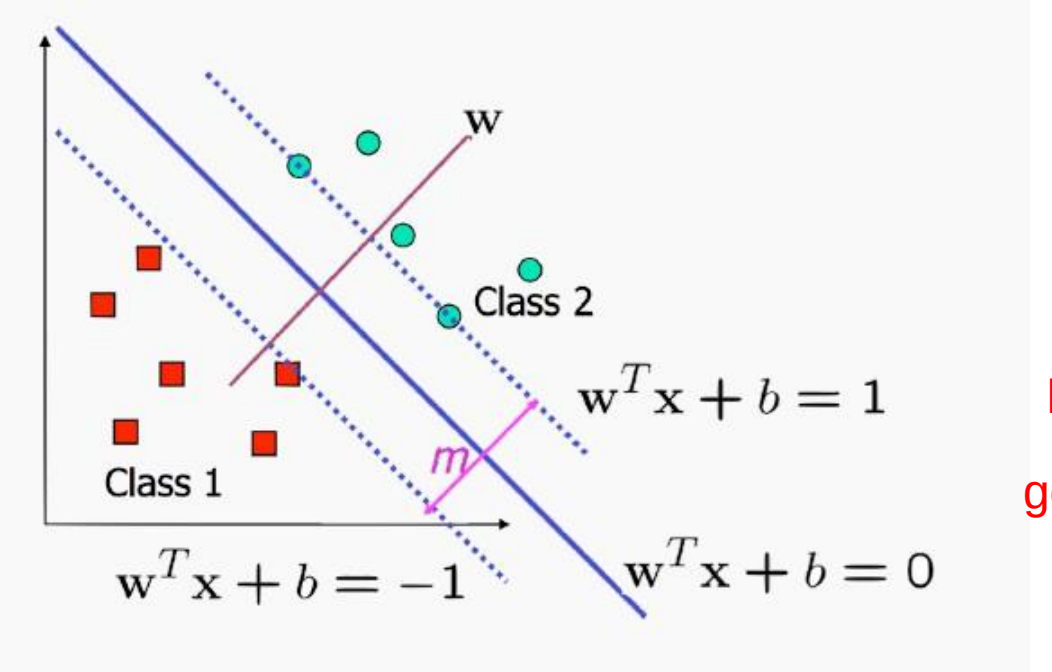
- For points on planes $\mathbf{w}^T\mathbf{x} + w_0 = \pm 1$, their distance to the decision boundary is $\pm \frac{1}{\|w\|}$

.

w modulus

- So we can define the **margin** of a decision boundary as the distance to its support vectors: $m = \frac{2}{\|w\|}$

these lines are going to be the edge of the margins where our support vectors lie on.




hyperplane +1, 2 SV sitting on it

go to formula s15

hyperplane -1, 1 SV sitting on it

because these constraints and conditions are so rigid, because this expression is so rigid, that's the reason that it's known as a hard margin classifier. It has to take accountant to every point and if the data is not linearly separable, meaning that you can't actually at all draw a straight line through it, this whole process will fall over.This will block it

# SUPPORT VECTOR CLASSIFIER: HARD MARGIN

If misclassification here, the whole thing just keels over and they'll give you an error.It says it'll just say cannot find a hard margin that separates the two classes.

- Finally, formulate our optimization problem: Find a decision boundary that maximises the distance to both classes – i.e. maximises the margin, M, while maintaining zero misclassifications

Where it won't turn out to be true is in the event of a misclassification where this is positive, but this predicted value comes out being negative on the wrong side of the decision boundary. +ve x -ve = -ve

$$\begin{cases} max_w \dfrac{2}{\|w\|} \\ such\ that\ y^{(i)}\left(w^T x^{(i)} + x_0\right) \geq 1, \qquad i = 1, \dots, N \end{cases}$$

minimize w so that value is bigger

yi = true y, wtx+w0=y predict

even if -ve, -ve x -ve will be +ve

- Maximising $\frac{2}{\|w\|}$ is the same as minimizing $\|w\|$. However L2 optimisations are more stable. Therefore:

minimise magnitude of the weight squared.

$$\begin{cases} min_w \|w\|^2 \\ such\ that\ y^{(i)}\left(w^T x^{(i)} + x_0\right) \geq 1, \qquad i = 1, \dots, N \end{cases}$$

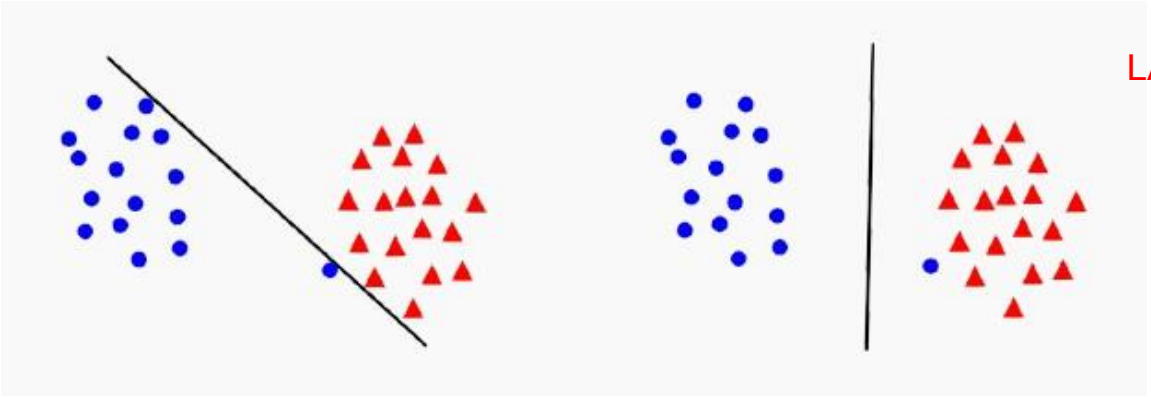
L2 optimizations where we're optimising the square of something, we get steeper slopes along the optimization.They're more stable than L1 optimizations where not squared

So what we do, because L2 optimizations are quicker, more stable, there's better algorithms for us.What we do is instead of trying to minimise W, we actually try and minimise W or the modulus W squared.

- This is a quadratic optimisation problem, has linear constraints and there is a unique solution.
- Calculus again! (Lagrange multipliers if you want to look up the maths)

# MARGIN ERROR/TRADE OFF

- Which one of these lines is a better generalisation?

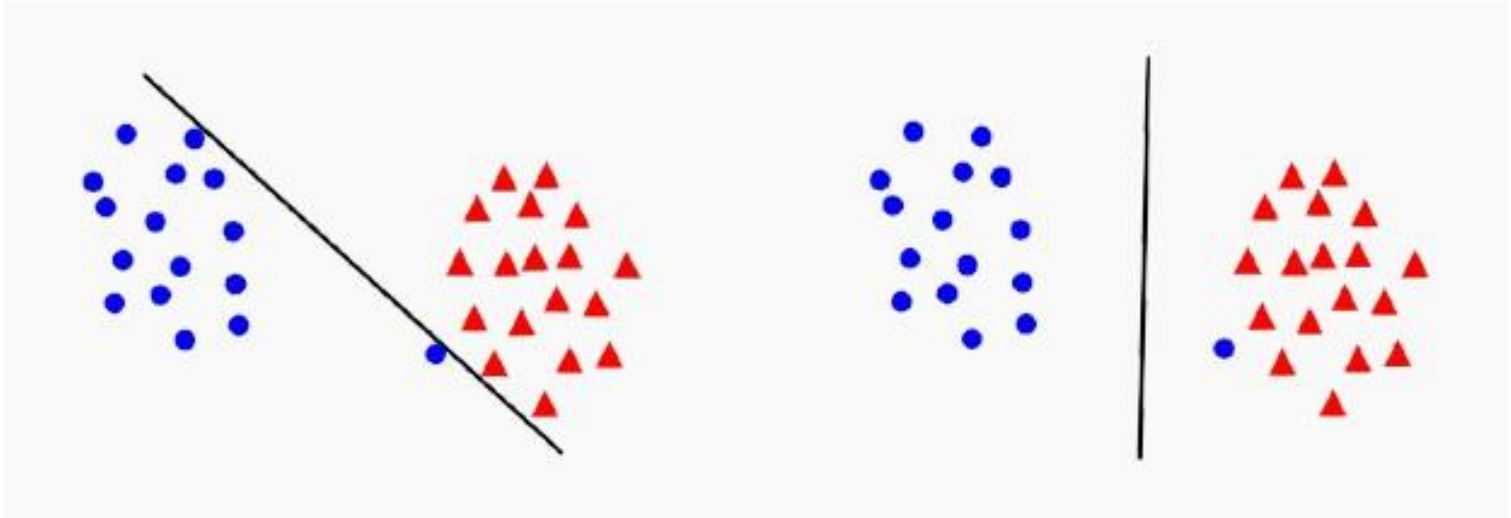


LARGE MARGIN IS BETTER

- In the first one the points can be linearly separated but there is a very narrow margin
- But possibly the large margin solution is better, even though one constraint is violated (this is known as a soft margin classifier)

# MARGIN ERROR/TRADE OFF

- Maximising the margin is a good idea as long as we know that the underlying classes are linear separable and that the data is noise free.
- If data is noisy, we might be sacrificing generalisation in order to minimise classification error with a very narrow margin
- With every decision boundary, there is a trade-off between maximising margin and minimising the error.



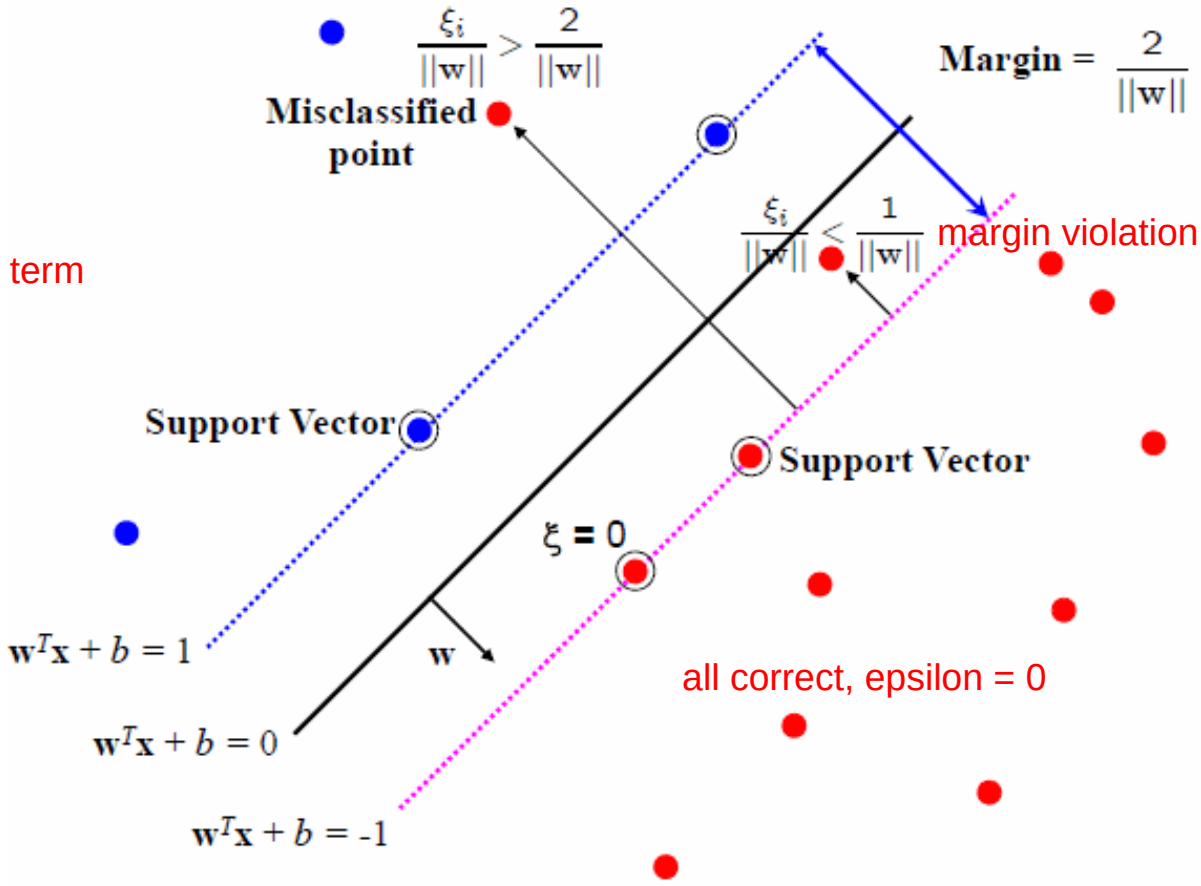
SOFT MARGIN CLASSIFIER

# SLACK VARIABLES

the bigger this penalty term is, the more wrong your classification is.

- We can add a variable $\xi_i \geq 0$ for each point/sample.
  - For $0 < \xi \leq 1$ point is between margin and correct side of hyperplane. This is called a margin violation
  - For $\xi \geq 1$ point is misclassified
  - For $\xi = 0$ point is the correct side of the margin.

Epsilon is a penalty term



# SOFT MARGIN SOLUTION

it's allowing us to to misclassify points in order to give better overall performance.

- To relax the constraints, our problem is re framed as

$$\begin{cases} min_w \|w\|^2 + C\sum_{i=1}^{N} \xi_i \\ such\ that\ y^{(i)}\left(w^T x^{(i)} + x_0\right) \geq 1 - \xi_i, \qquad i = 1, \ldots, N \end{cases}$$

C is regularisation for support vector, C scales how much that we can ignore the constraints

with a lambda is regularisation term which has a penalty

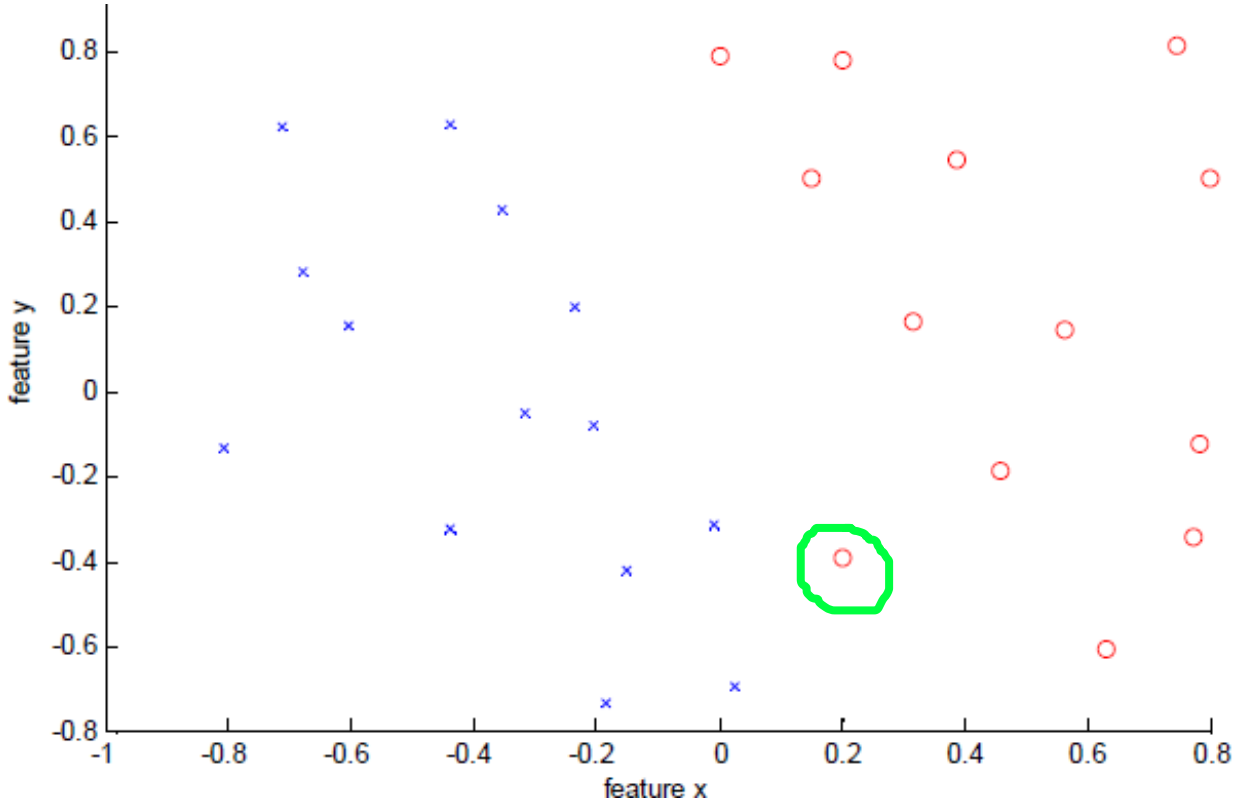
- C is a regularisation parameter: (some notes will use $\lambda$ instead of C , sklearn uses C)
  - Small C allows constraints to be easily ignored → large margin
  - Large C makes constraints hard to ignore → narrow margin
  - C → ∞ enforces all constraints: hard margin
- This is still a quadratic optimization problem and there is a unique minimum. Note: there is only one parameter, C (that you choose/cross-validation).
- In general, the best C parameter depends on the situation. Experiment (Cross-Validation). One note: larger C takes more computation to train.

choosing C is a bit more computationally expensive than it is in the case of where we saw regularisation

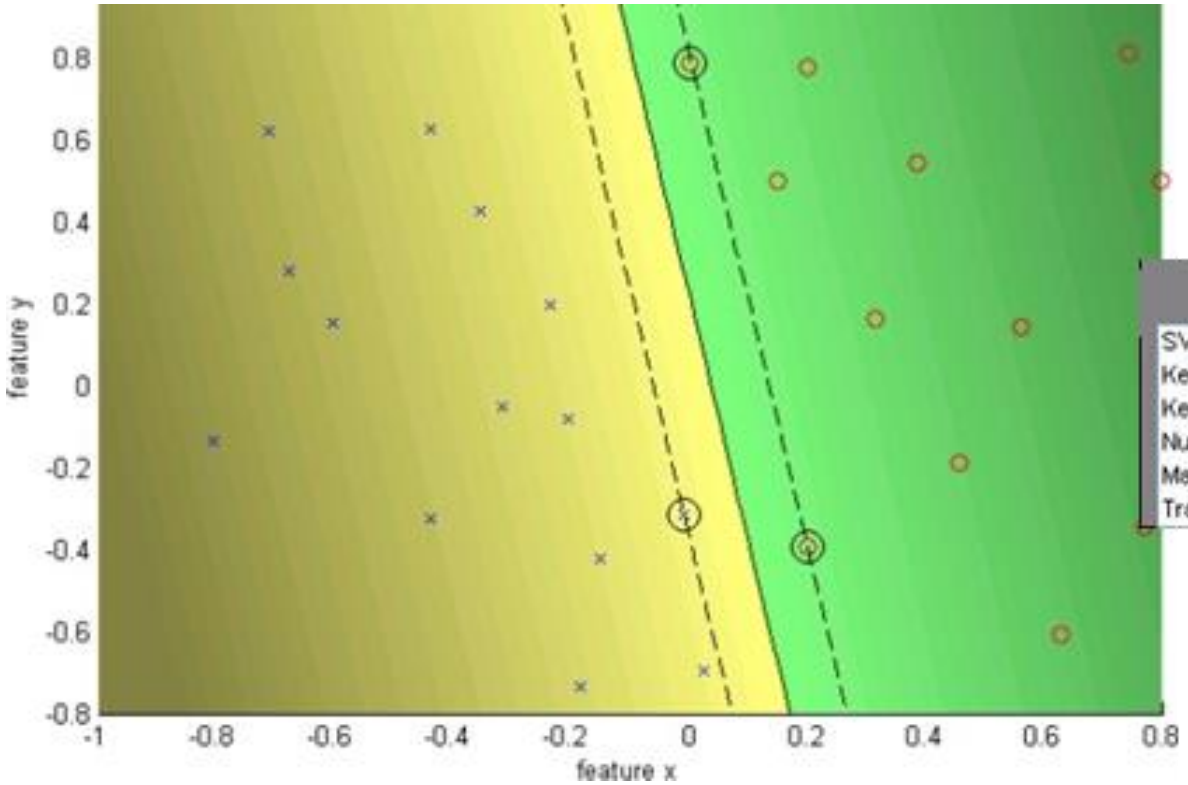
let's have a look at what a soft margin classifier kind of gives us in practise as we vary C.

# EXAMPLE:

- Data is linearly separable - but only with a narrow margin



# INFINITY C - HARD MARGIN



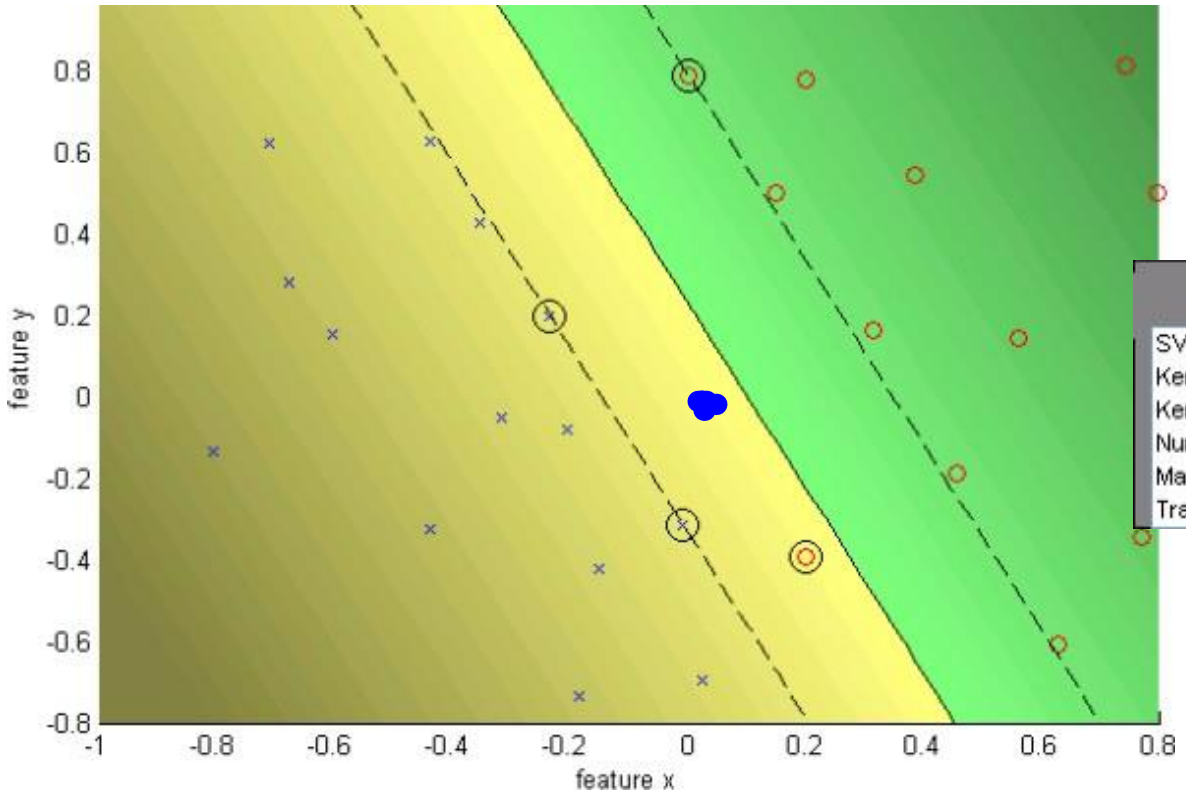

So if we set C to Infinity and we enforce hard margin, it's exactly that line and we can see our support vectors are highlighted, which sit on the margin.

Comment Window

```
SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: Inf
Kernel evaluations: 971
Number of Support Vectors: 3
Margin: 0.0966
Training error: 0.00%
```

# C = 10 - SOFT MARGIN

If we set C to 10 though, it relaxes our constraints a little bit and it gives us a softer margin, which means that our margin gets bigger. And this point, which it pointed out already, this red point which is out of the line, actually ends up on the incorrect side of the margin, which means it's been misclassified.
But even though it's been misclassified, overall we're going to get better generalisation performance that if a blue dot comes up here, you know, it's probably closer to the blue than it is to all the red, you know, if something a
So the net result is that even though we have, you know, we've got a misclassification for ourself, the overall generalisation improves.




# PREVIOUS PROBLEM - BREAST CANCER DATA SET

- SVM classifiers often do better on the ”hard” problems. If we return to the breast cancer data set:

```
model = make_pipeline(
    StandardScaler(),
    SVC(kernel='linear', C=2.0)
)
model.fit(X_train, y_train)
print(model.score(X_test, y_test))

0.993006993006993
```

SVC VS SVM
support vector classifier is when we use just the linear kernel.Technically, in my book anyways, it's not an SVM, it's not a support vector machine until we expand things into higher dimensionss, but where we can use different kernels, there's RBF and polynomial kernels and different types of kernels where we can project things into other dimensions. When we do that, we can when we do that, we can see or that then it becomes a support vector machine.
But this really what we've done today are support vector classifiers.
Now support vector machines in general often do better on the heart problems.
if we look at the breast cancer data set, we can see really good performance here, which is better than anything we've seen before with the Gaussian or K nearest neighbours classifier. Now it depends, on the value of C and and whatnot.

- That’s better than either GaussianNB or KNeighborsClassifier.