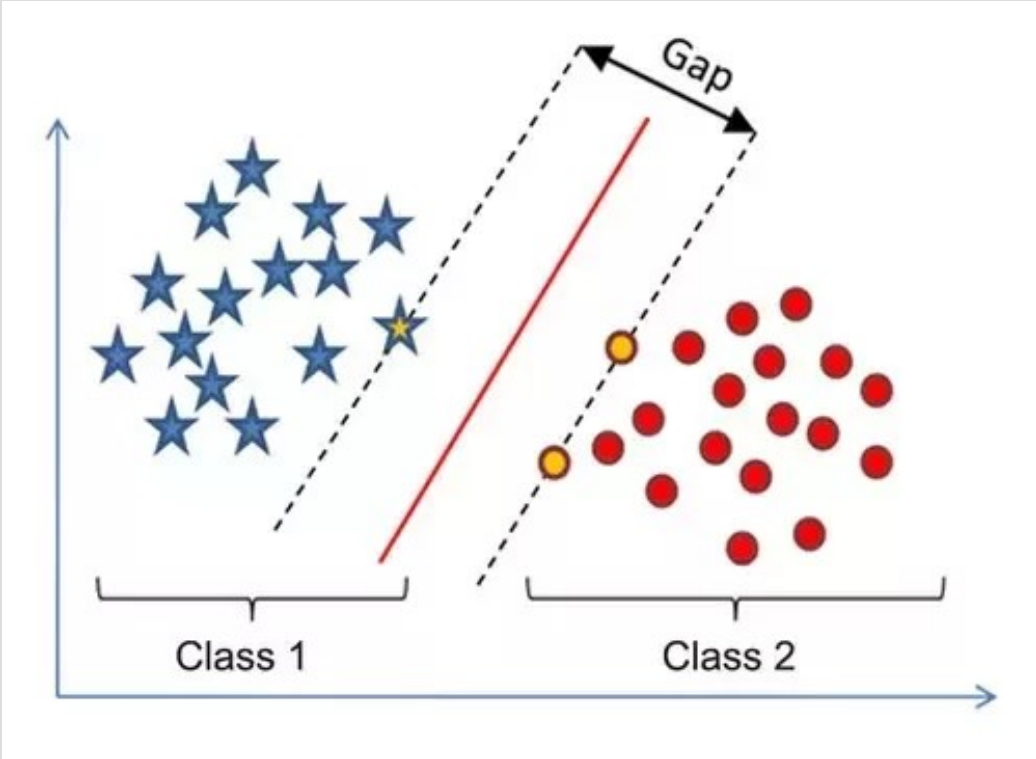
# Support Vector Machines (Ch. 18.9)

# SVM Basics

Support Vector Machines (SVMs) try to do our normal linear classification (last few lectures), but with a couple of twists

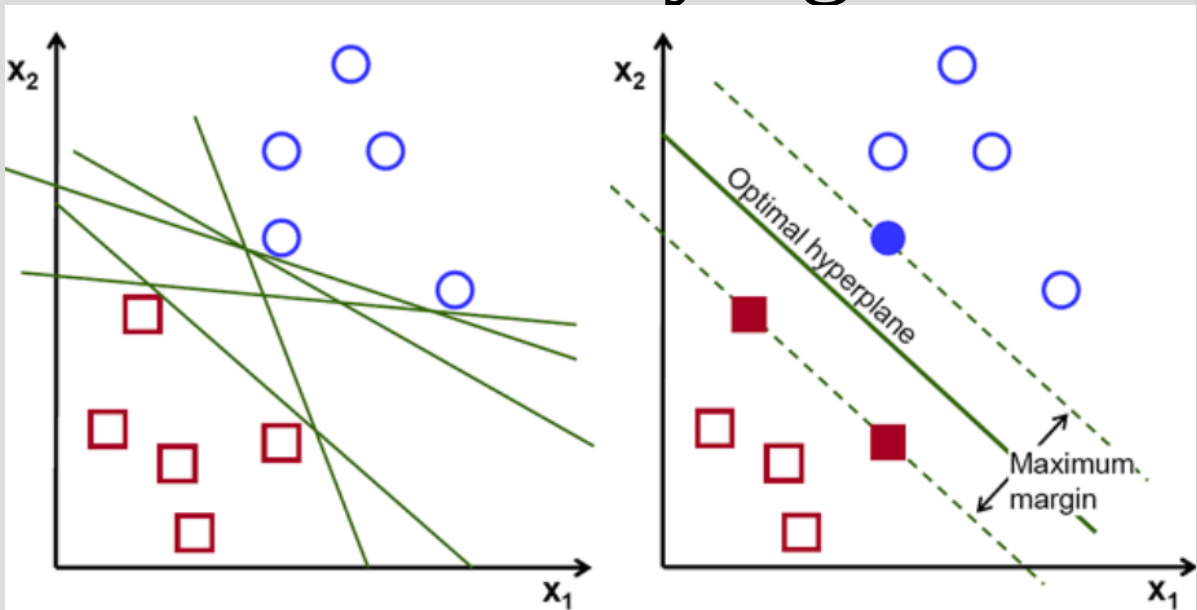
1. Find the line in the middle of points with the largest gap (called maximum margin separator)



# SVM Maximum Separation

The idea for having the largest gap/width is to avoid misclassification

If we drew the line close to a known example, we have a greater chance of classifying it the opposite type, despite being close



# SVM Maximum Separation

To define the separator, let's represent "w" as the normal vector to the plane (in 2D, a line)

To allow the (hyper-)plane to not pass through the origin, we will add an offset of "b"

Thus our separator is: $w \cdot x + b$

Now we need to find how to make the gap as big as possible in terms of "w" and "b"

# SVM Maximum Separation

Let's classify all the points above the line as +1 and all the points below the line as -1

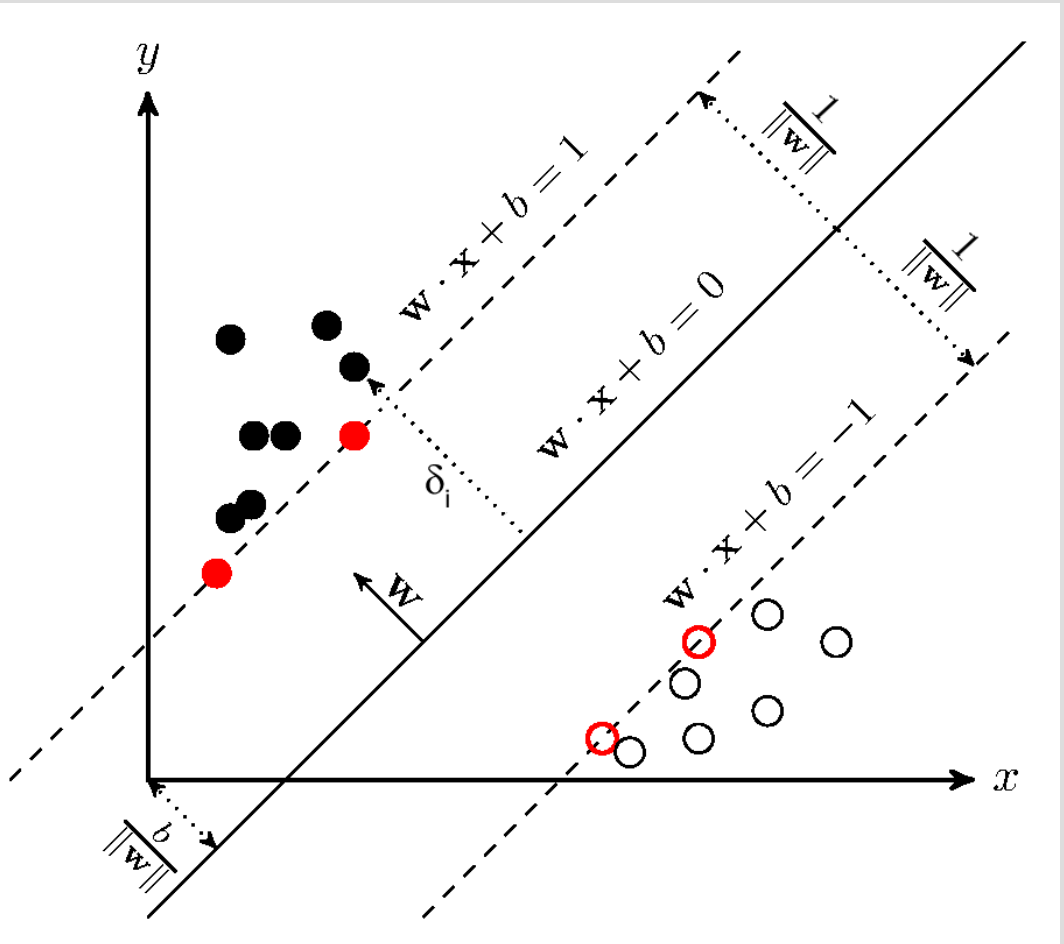
Then our separator needs:

if $w \cdot x + b \geq 1$

then y = +1

if $w \cdot x + b \leq -1$

then y = -1



# SVM Maximum Separation

We can combine these two conditions into:

$y(w \cdot x + b) \geq 1$ ... as condition for every point

Now that we have the requirements for our separator, need to represent “maximum gap”

The distance between a hyper-plane and a a point (a line in the case with just x,y):

$dist = \frac{|w \cdot x|}{|w|}$ (for higher dimension: $\frac{w_1 \cdot x_1 + w_2 \cdot x_2 + b}{\sqrt{w_1^2 + w_2^2}}$ )

# SVM Maximum Separation

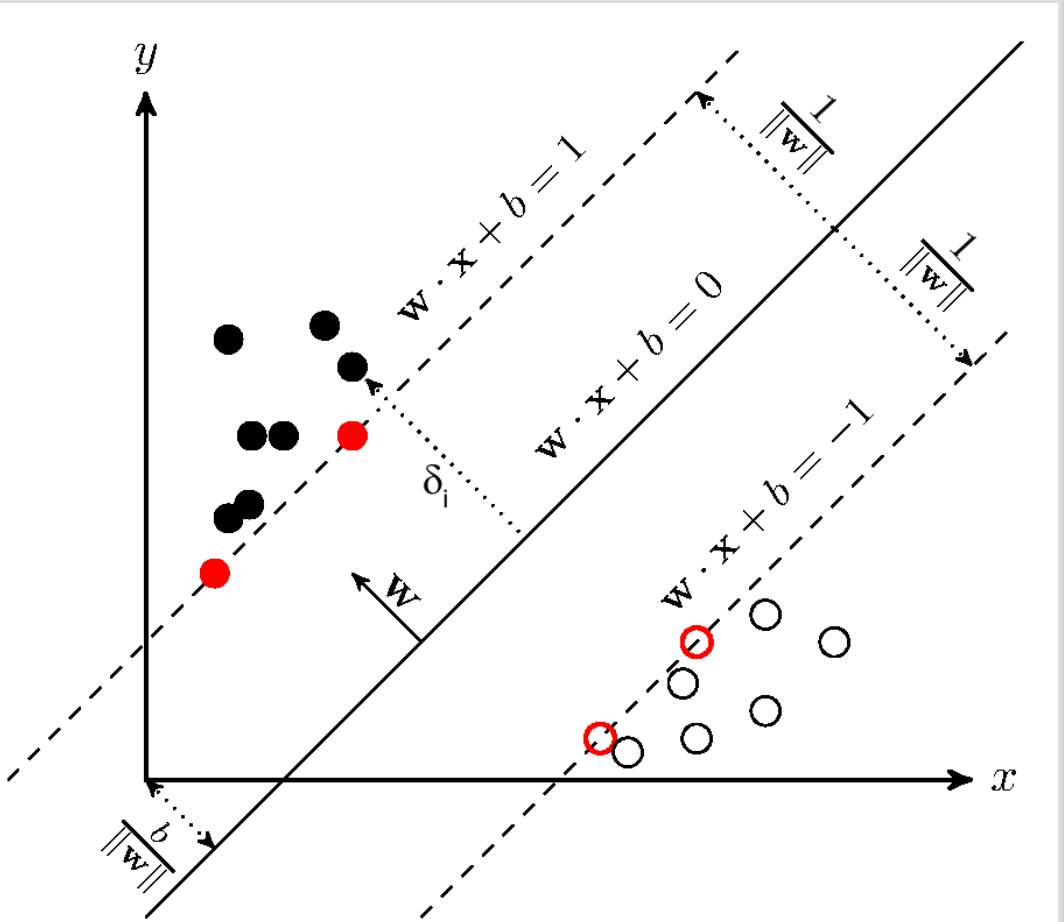
Since we want the closest points to be exactly

$$w \cdot x + b = \pm 1$$

The distance to these points and the line is just:

$$\frac{|w \cdot x|}{|w|} = \frac{|\pm 1|}{|w|} = \frac{1}{|w|}$$

So to maximize gap, we want min |w|



# SVM Maximum Separation

Thus we have an optimization problem:

minimize: $|w|$ (distance of vector)
constraints: $y_i(w \cdot x_i + b) \geq 1$, for all points $i$

At this point we could use our old friend gradient descent...

... but instead people tend to take a much more math-y option!

# Side note: Duality

Rather than solve that optimization directly, we will instead solve the dual problem (i.e. a different but equivalent problem)

If we were trying to “maximize profit” a dual could be framed as “minimizing loss”

Typically they are not exact opposites like this, and we have actually seen something similar in this class before

# Side note: Duality

In MDPs, we wanted to find the utility of each state/cell...

Doing this directly (with Bellman equations) is value iteration

The “dual” would be to realize finding the “correct” utilities is identical to finding the “correct” actions (policy iteration)

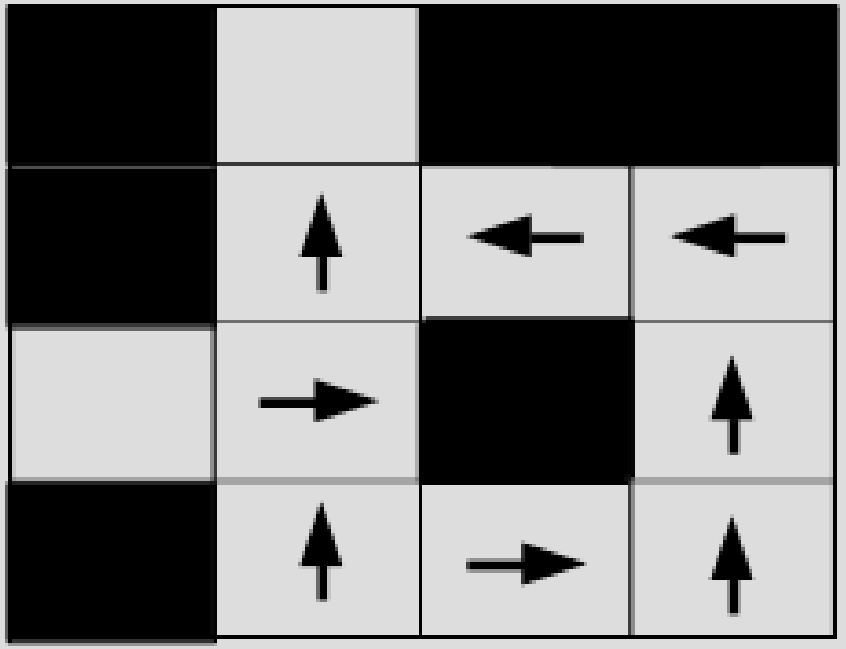
# Side note: Duality

So for MDPs we would have:

## Primal problem

| | | | |
|---|---|---|---|
| | 50 | | |
| | 48.59 | 47.34 | 45.93 |
| -50 | 37.93 | | 44.68 |
| | 37.28 | 42.03 | 43.28 |

## Dual problem



# SVM Maximum Separation

minimize: $|w|$ (distance of vector)
constraints: $y_i(w \cdot x_i + b) \geq 1$, for all points $i$

We can note that our optimization is quadratic (as min: $|w| = \sqrt{w_1^2 + w_2^2 + ...}$ same as min: $w_1^2 + w_2^2 + ...$)

change to min: $|w|^2$... or actually 0.5 $|w|^2$

So there will be a single unique point for the minimum, but we have a constraint so the global minimum might not be possible

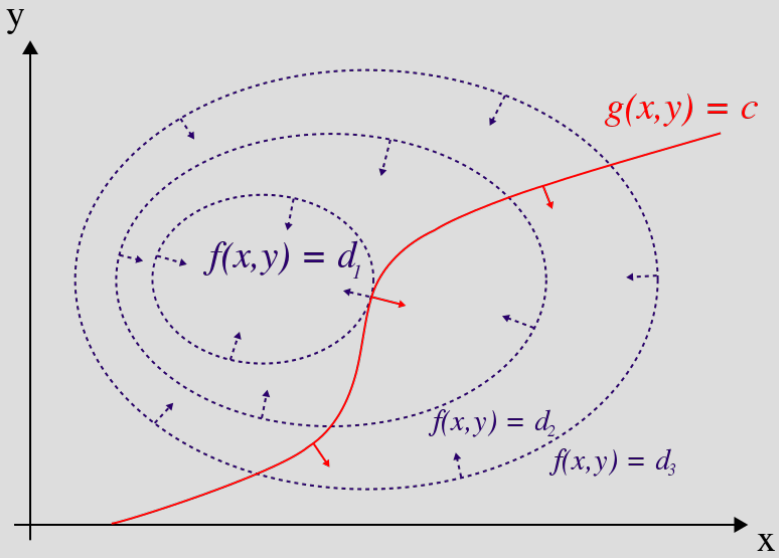
Let the minimum (with constraint) be “d”

# SVM Maximum Separation

minimize: $\frac{1}{2}|w|^2$ (distance of vector)
constraints: $y_i(w \cdot x_i + b) \geq 1$, for all points $i$

We can then say that the derivative with respect to the constraint is in the same/opposite direction as the derivative of |w| (min goal)

If they were not scalar multiples of each other, you could “head closer” than “d” to minimum



# SVM Maximum Separation

minimize: $\frac{1}{2}|w|^2$ (distance of vector)
constraints: $y_i(w \cdot x_i + b) \geq 1$, for all points $i$

This is called the Lagrangian dual (or function)
So if function "f" is our min/max goal
and "g" is our constraints:

$$\nabla f(x, y, ...) = \lambda \cdot \nabla g(x, y, ...)$$
... or: $\nabla f(x, y, ...) - \lambda \cdot \nabla g(x, y, ...) = 0$

The constraint is a bit annoying as it is an inequality... let's cheat and rewrite as:

$$y_i(w \cdot x_i + b) - 1 = 0$$

equality is only true for points directly on "gap"... more on this later

# SVM Maximum Separation

constraint for each point, so sum (math reasons)

Thus we have:

minimize: $\frac{1}{2}|w|^2 - \sum_i \lambda_i (y_i (w \cdot x_i + b) - 1)$

our book calls this α... doesn't matter, it's a scalar

... where the derivatives are zero (we get to control "w" and "b" for hyperplane)

partial wrt. w: $w - \sum_i \lambda_i y_i x_i = 0$

partial wrt. b: $\sum_i \lambda_i y_i = 0$

# SVM Maximum Separation

Plugging these back into equation:

minimize: $\frac{1}{2}|w|^2 - \sum_i \lambda_i y_i (w \cdot x_i) - b \sum_i \lambda_i y_i + \sum_i \lambda_i)$

$w = \sum_i \lambda_i y_i x_i$

FOIL

$\sum_i \lambda_i y_i = 0$

minimize: $\frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j x_i x_j - \sum_i \lambda_i y_i ((\sum_j \lambda_j y_j x_j) \cdot x_i) - 0 + \sum_i \lambda_i)$

... these are same...

minimize: $\sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$

actually a "maximize"
as like: c – 1/2 a $x^2$

... at this point, we can minimize λ (only var)

# SVM Maximum Separation

... erm, that was a lot

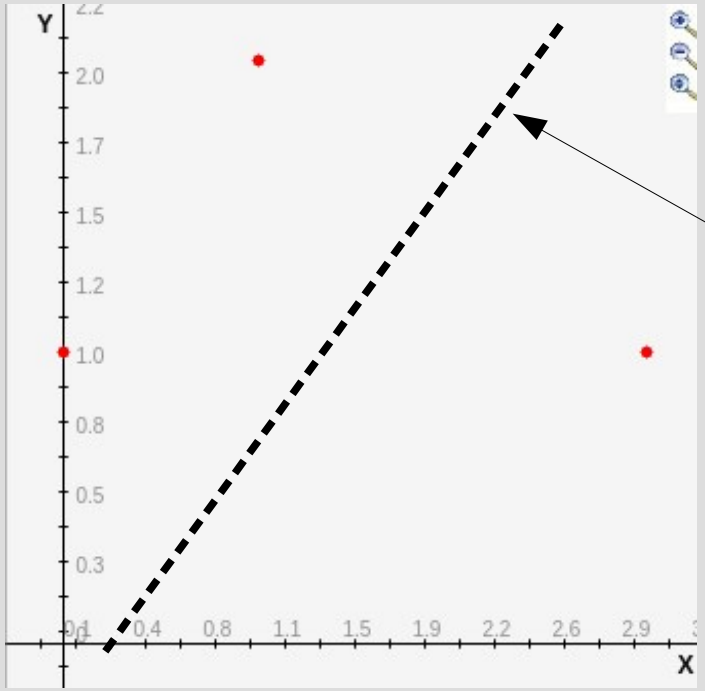
Let’s do an example!

Suppose we have 3 points, find the best line:

(0,1), y=+1

(1,2), y=+1

(3,1), y=-1




find

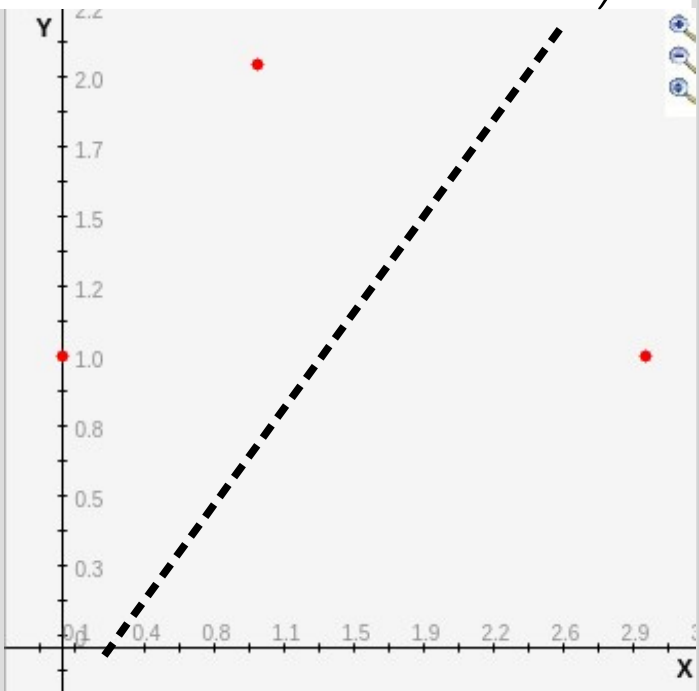
# SVM Maximum Separation

$$\text{maximize: } \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$$

$$\lambda_1 + \lambda_2 + \lambda_3 - \frac{1}{2}\Big( \lambda_1^2 1^2 (0,1)\cdot(0,1) + \lambda_1\lambda_2 1^2 (0,1)\cdot(1,2) + \lambda_1\lambda_3 1(-1)(0,1)\cdot(3,1)$$
$$+ \lambda_2\lambda_1 1^2 (1,2)\cdot(0,1) + \lambda_2^2 1^2 (1,2)\cdot(1,2) + \lambda_2\lambda_3 1(-1)(1,2)\cdot(3,1)$$
$$+ \lambda_3\lambda_1 (-1)1(3,1)\cdot(0,1) + \lambda_3\lambda_2(-1)1(3,1) + \cdot(1,2)\lambda_3^2(-1)(-1)(3,1)\cdot(3,1)\Big)$$

$$= \lambda_1 + \lambda_2 + \lambda_3 - \frac{1}{2}\Big( \lambda_1^2 + 2\lambda_1\lambda_2 - \lambda_1\lambda_3$$
$$+ 2\lambda_2\lambda_1 + 5\lambda_2^2 - 5\lambda_2\lambda_3$$
$$- \lambda_3\lambda_1 - 5\lambda_3\lambda_2 + 10\lambda_3^2\Big)$$

jam this into some optimizer



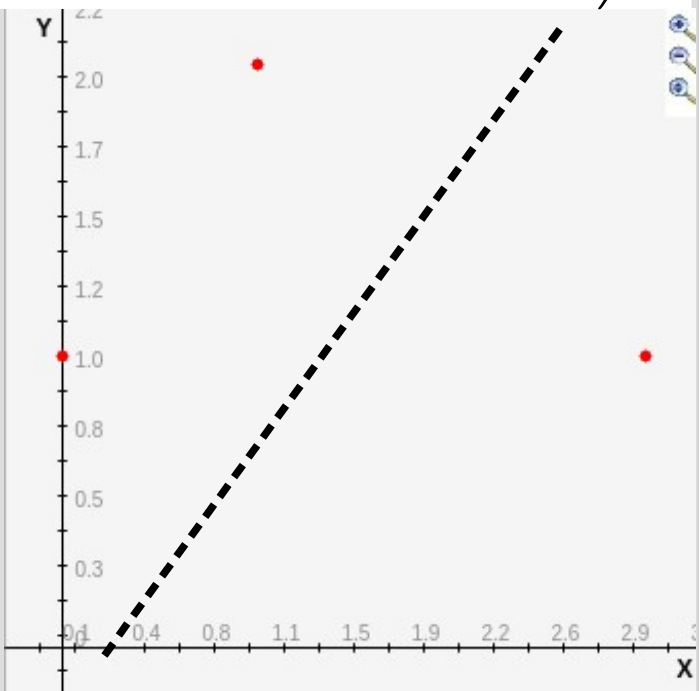
# SVM Maximum Separation

$$\text{maximize: } \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$$

$$\lambda_1 + \lambda_2 + \lambda_3 - \frac{1}{2}\Big( \lambda_1^2 1^2 (0,1)\cdot(0,1) + \lambda_1\lambda_2 1^2 (0,1)\cdot(1,2) + \lambda_1\lambda_3 1(-1)(0,1)\cdot(3,1)$$
$$+ \lambda_2\lambda_1 1^2 (1,2)\cdot(0,1) + \lambda_2^2 1^2 (1,2)\cdot(1,2) + \lambda_2\lambda_3 1(-1)(1,2)\cdot(3,1)$$
$$+ \lambda_3\lambda_1(-1)1(3,1)\cdot(0,1) + \lambda_3\lambda_2(-1)1(3,1) + \cdot(1,2)\lambda_3^2(-1)(-1)(3,1)\cdot(3,1)\Big)$$

$$f(x,y,z) = x + y + z - \frac{1}{2}\Big( x^2 + 2xy - xz$$
$$+ 2xy + 5y^2 - 5yz$$
$$- xz - 5yz + 10z^2 \Big)$$

jam this into some optimizer



# SVM Efficient storage

At this point, we solve for the $\lambda_i$ for each point

$\lambda_i$ will actually be zero for all points not on the gap (because we dropped the inequality)

This actually leads to the second useful fact of SVMs:
They only need to remember a few points (the ones on the gap)

# SVM Efficient storage

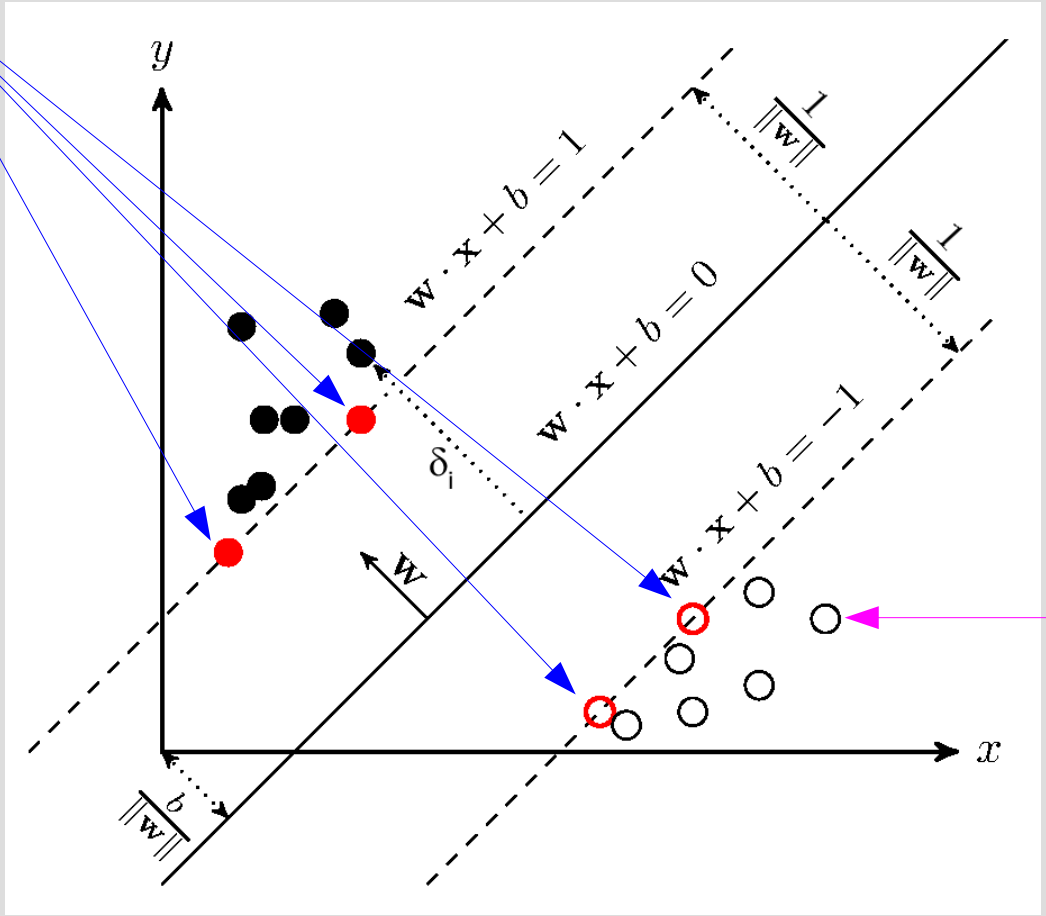
So regardless about the number of examples you learn on, you only need to store the ones closest to the separator

Thus the stored examples are proportional to the number of input/attributes (dimensions)

If you find a new example that is inside the gap, recompute separator... otherwise you don’t need to do anything

# SVM Efficient storage

So in this case, you only need to find $\lambda_i$ for these four point (they define “w” and “b”)




$\lambda_{this} = 0$

# SVM Dimensional Change

This third trick might seem a bit weird as we often say how higher dimensions cause issues

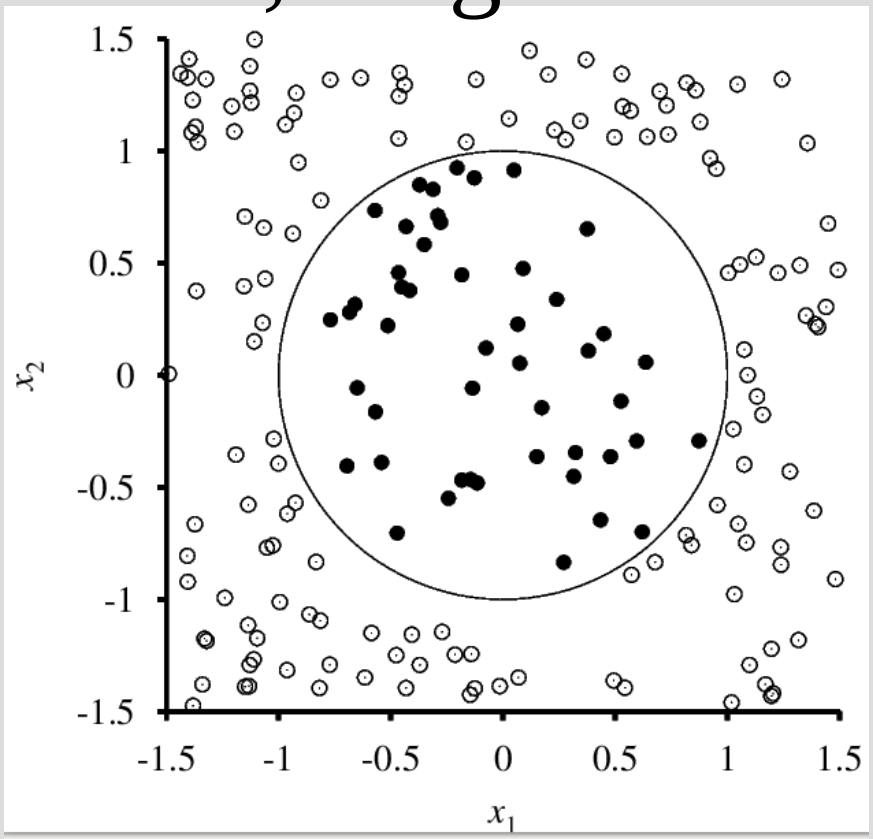
But it can actually be helpful as there is this useful fact:

You can (almost) always draw an N-1 dimensional (hyper)plane to perfectly separate N points
... what does “(almost)” mean?

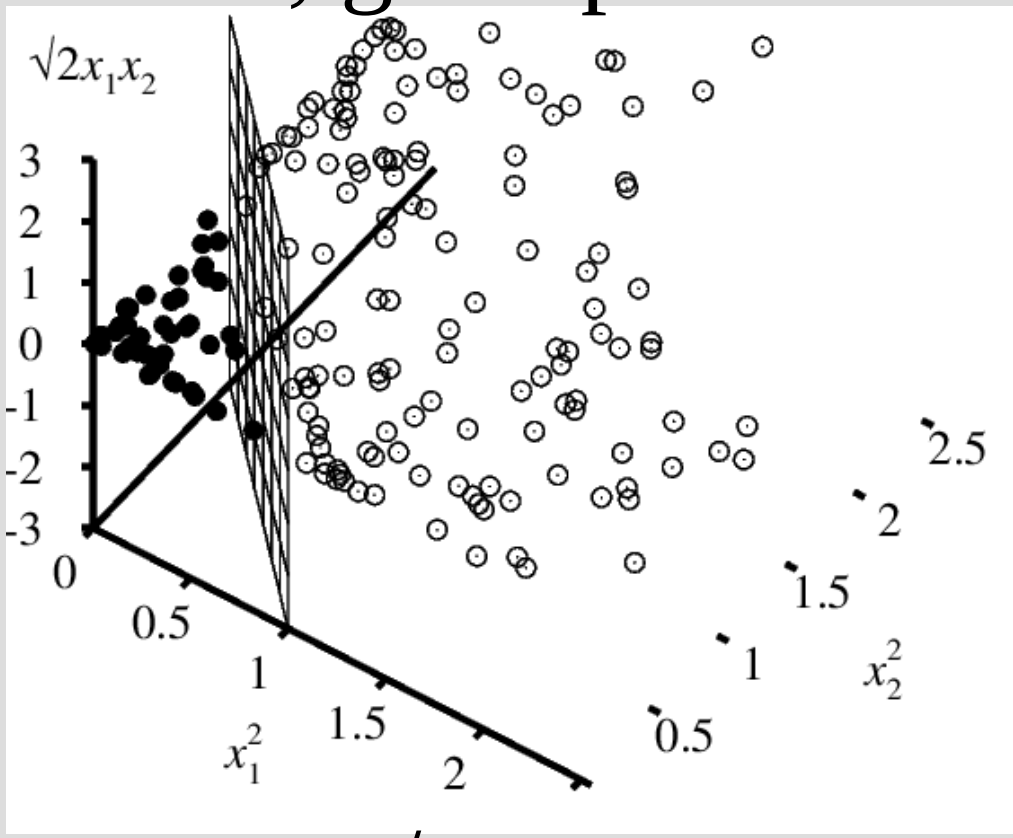
# SVM Dimensional Change

The book gives a good example of this:

2D, no good line




3D, good plane!




$(x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}\, x_1x_2, x_2^2)$

# SVM Dimensional Change

This change of dimension is called a kernel
(not to be confused with the other “kernels”)

Let’s review some equations before going deep

maximize: $\sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$

... we said you can use the above to find $\lambda_i$s,
once you have $\lambda_i$s, you can find “w” & “b”
to classify...

$$w = \sum_i \lambda_i y_i x_i$$

$$y_i(w \cdot x_i + b) - 1 = 0$$
(for points on gap)

# SVM Dimensional Change

However, if you have $\lambda_i$s, you actually don't need to go back to "w" and "b" (they represent the same thing)

Turns out you can classify directly as:

$$sign(\sum_j \lambda_j y_j (x_{new} \cdot x_j) - b)$$

if positive, $y_{new}$=+1
else (neg), $y_{new}$=-1

Also need to solve:

$$\text{maximize: } \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$$

... we need to be able to use both of these equations in the higher dimension as well

# SVM Dimensional Change

$$\text{classify} = sign(\sum_j \lambda_j y_j (x_{new} \cdot x_j) - b)$$

$$\text{maximize: } \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x_i \cdot x_j)$$

Both of these equations use the dot product of our X's (original domain)

So we want to use kernels/dim-change where:

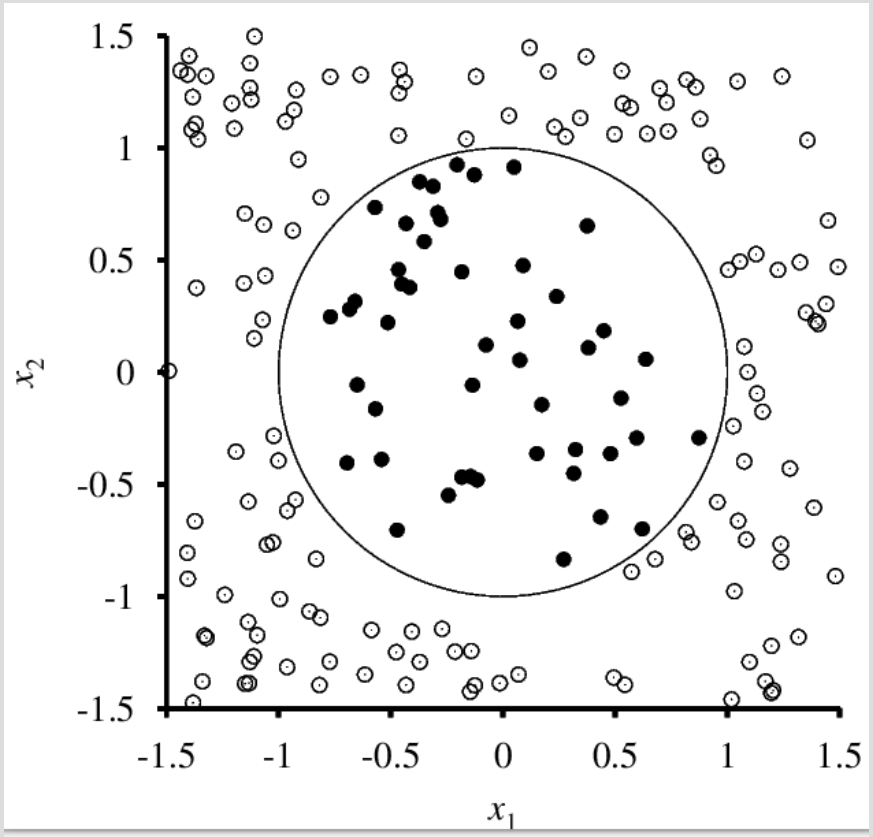
$$f(x_i) \cdot f(x_j) = K(x_i, x_j)$$

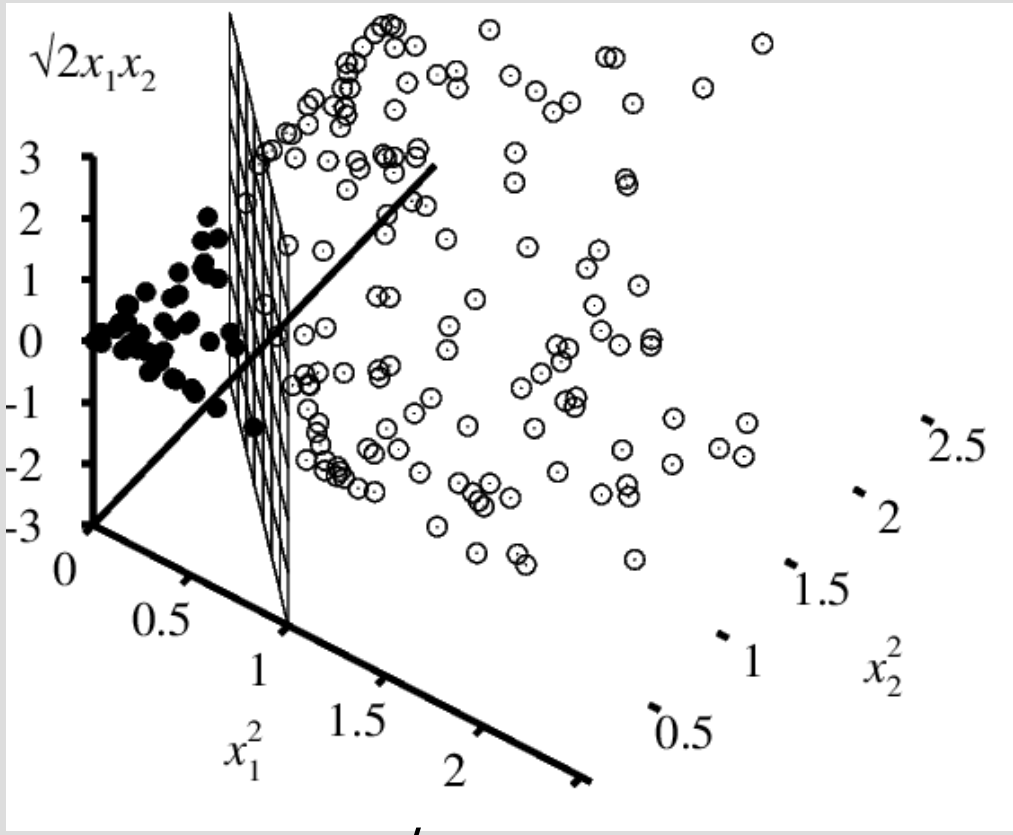
... then all of our equations are the same, we just need to change what "points" we are working with

# SVM Dimensional Change

This example indeed has: $f(x_i) \cdot f(x_j) = K(x_i, x_j)$

... where: $K(x, z) = (x \cdot z)^2$




$(x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$

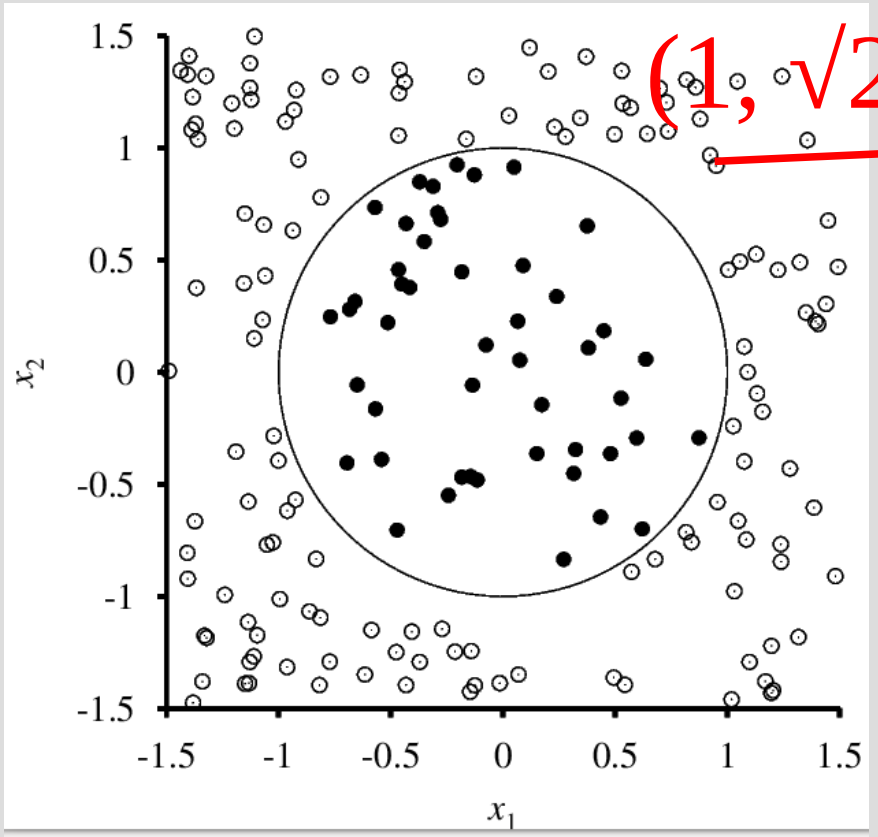
# SVM Dimensional Change

This example indeed has: $f(x_i) \cdot f(x_j) = K(x_i, x_j)$

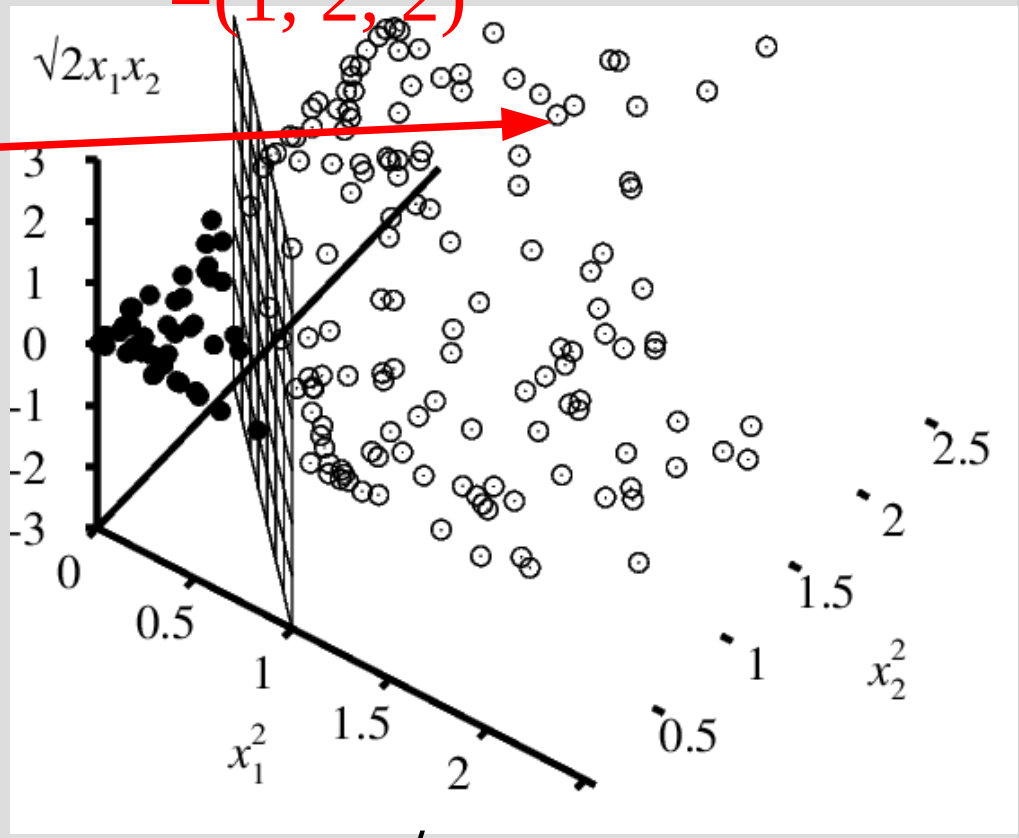
... where: $K(x, z) = (x \cdot z)^2$

$(1^2, \sqrt{2}(1)\sqrt{2}, \sqrt{2}\text{^}2)$

$=(1, 2, 2)$




$(1, \sqrt{2})$



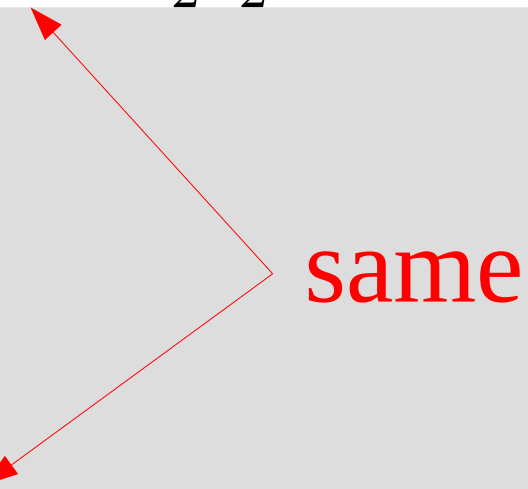

$(x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}\, x_1x_2, x_2^2)$

# SVM Dimensional Change

Proof: $(x_1, x_2) \Rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ mapping is $F(x, z) = (x \cdot z)^2$

$$map(x) \cdot map(z) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (z_1^2, \sqrt{2}z_1z_2, z_2^2)$$
$$= x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2$$

$$F(x \cdot z) = (x \cdot z)^2$$
$$= ((x_1, x_2) \cdot (z_1, z_2))^2$$
$$= (x_1z_1 + x_2z_2)^2$$
$$= x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2$$



# SVM Dimensional Change

There are a number of different dimension changing functions you could use

(mapping drops one point coordinate and square roots constant)

Common ones are:

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

RBF: $K(x, z) = e^{(-\gamma ||x - z||^2)}$

The polynomial one is especially nice as the number of terms in sum after FOIL = new dimension (grows very fast, like billions)

# SVM Miscellaneous

So far we have looked at the perfect classification only, but this can overfit

You can reuse the same complexity trade-off function we discussed in linear regression:

$$Cost(h_w) = Loss(h_w) + \lambda \cdot Complexity(h_w)$$

different λ constant

This is called “soft margin” where you trade accuracy for size of gap (|w|), but the overall approach is basically the same