ANCHORMEN
data activators

MACHINE LEARNING 2

BASIS EXPANSION & REGULARIZATION

Raoul Grouls

THE PROBLEM

LINEAR MODELS

We have a lot of models (both classification and regression) that work with linear models.

We prefer them in some cases because:

- Their simplicity helps us avoid overfitting
- They are fast and have optimal solutions (cf., we don’t have to fear about getting stuck in a local minimum)
- Their simplicity makes them ideal for a baseline model

LINEAR MODELS

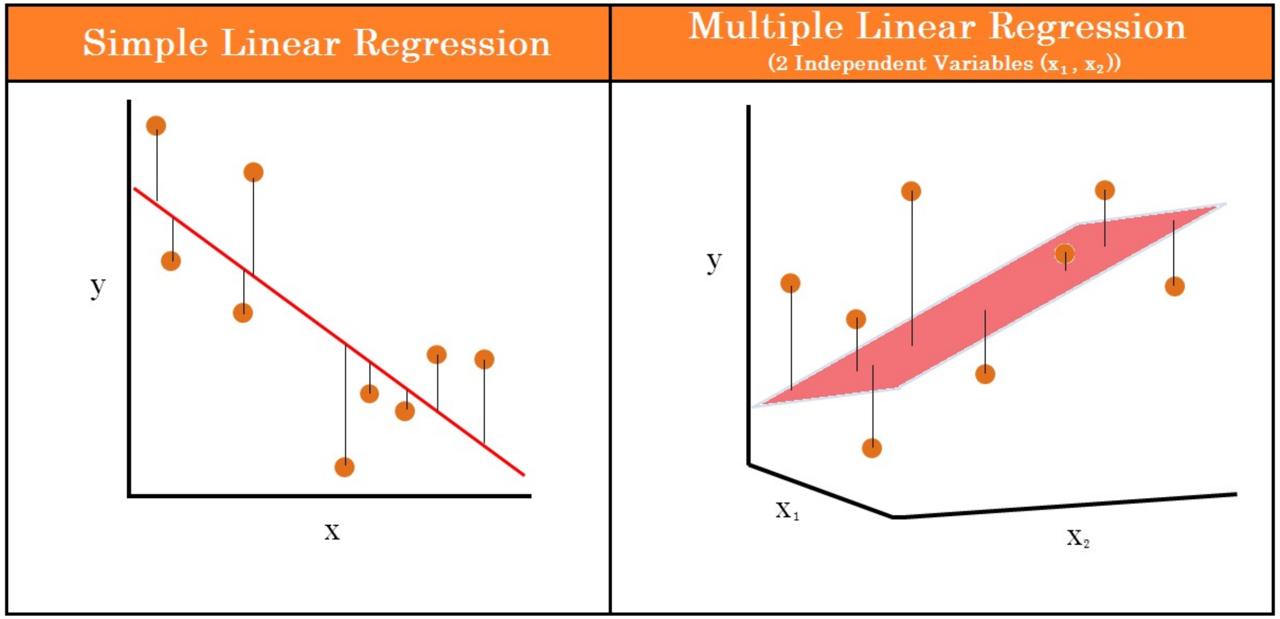
The basic mathematical shape of a linear model is

$$y = WX + b$$

- y_i is an observation
- $X_i = \{x_{i1}, \dots, x_{in}\}$ are n features
- W is a matrix with weights for every feature
- b is an additional bias weight.

We can write this out in full as:

$$y = w_1 x_{i1} + w_2 x_{i2} + \dots + w_n x_{in} + b$$


Simple Linear Regression
Multiple Linear Regression
(2 Independent Variables (x₁ , x₂))
y
x
y
x₁
x₂

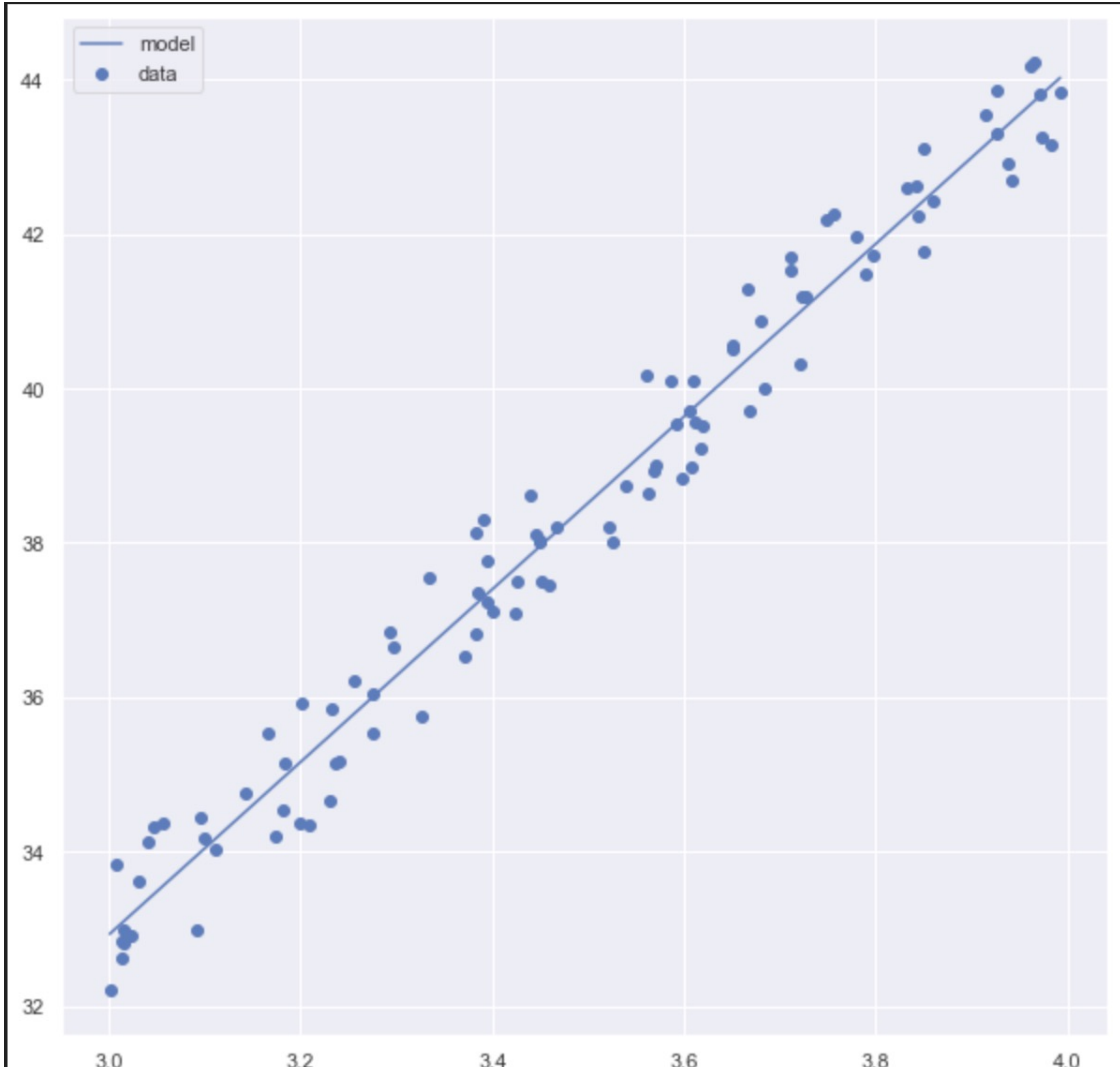
LINEAR DATA

We call these models a hyperplane: a subspace with a dimension, one less than the ambient space.

For 2D data like the blue points, a 1D linear model will make a good fit.

For a regression, we want to find a hyperplane that has as many observations as possible close to the hyperplane.

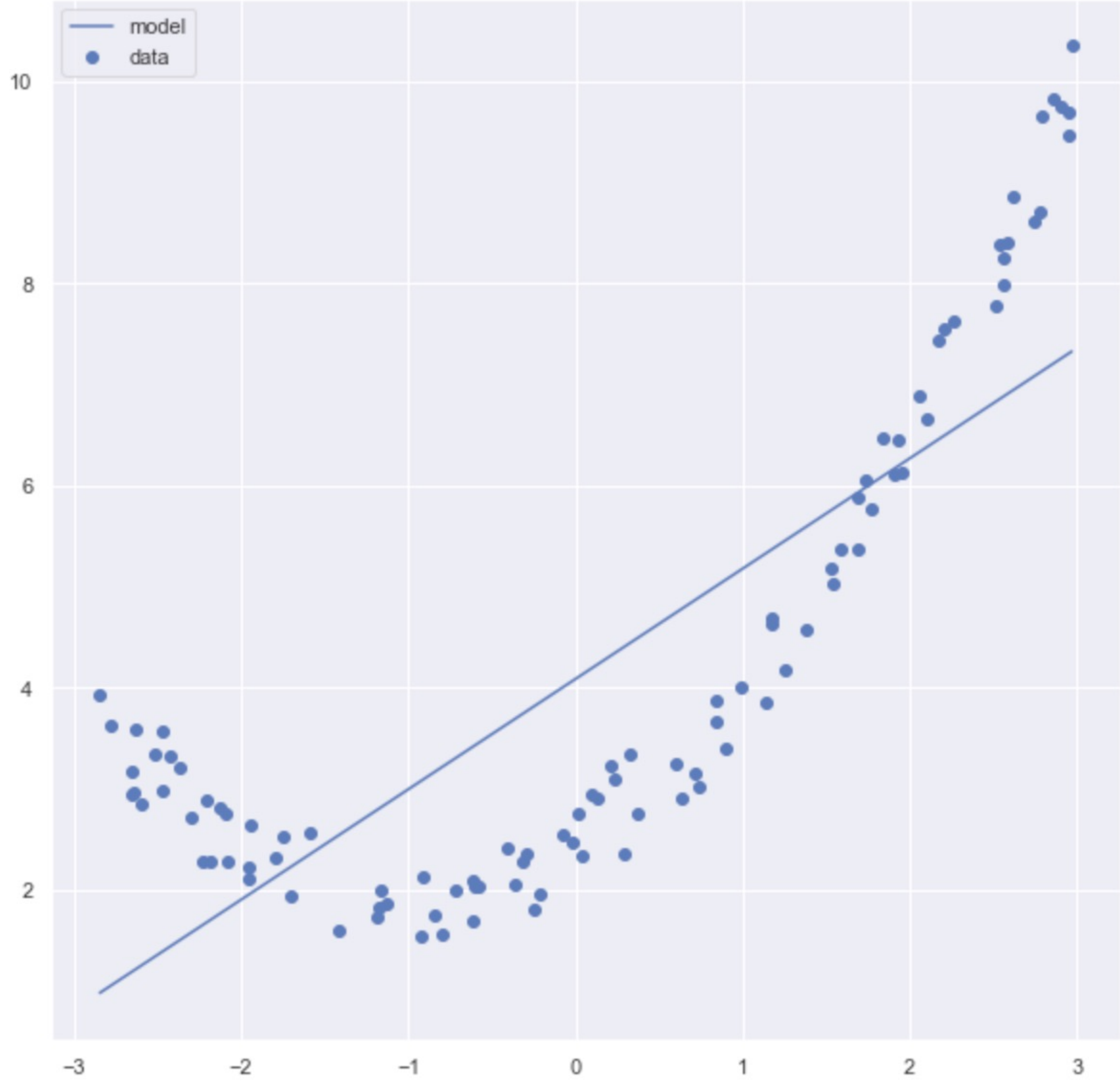
For a classification, we want to find a hyperplane that separates the two classes of observations.



LINEAR SEPARABILITY

But at the moment our data doesn’t follow a straight line, a linear model will never fit.

At least, not without tricks…

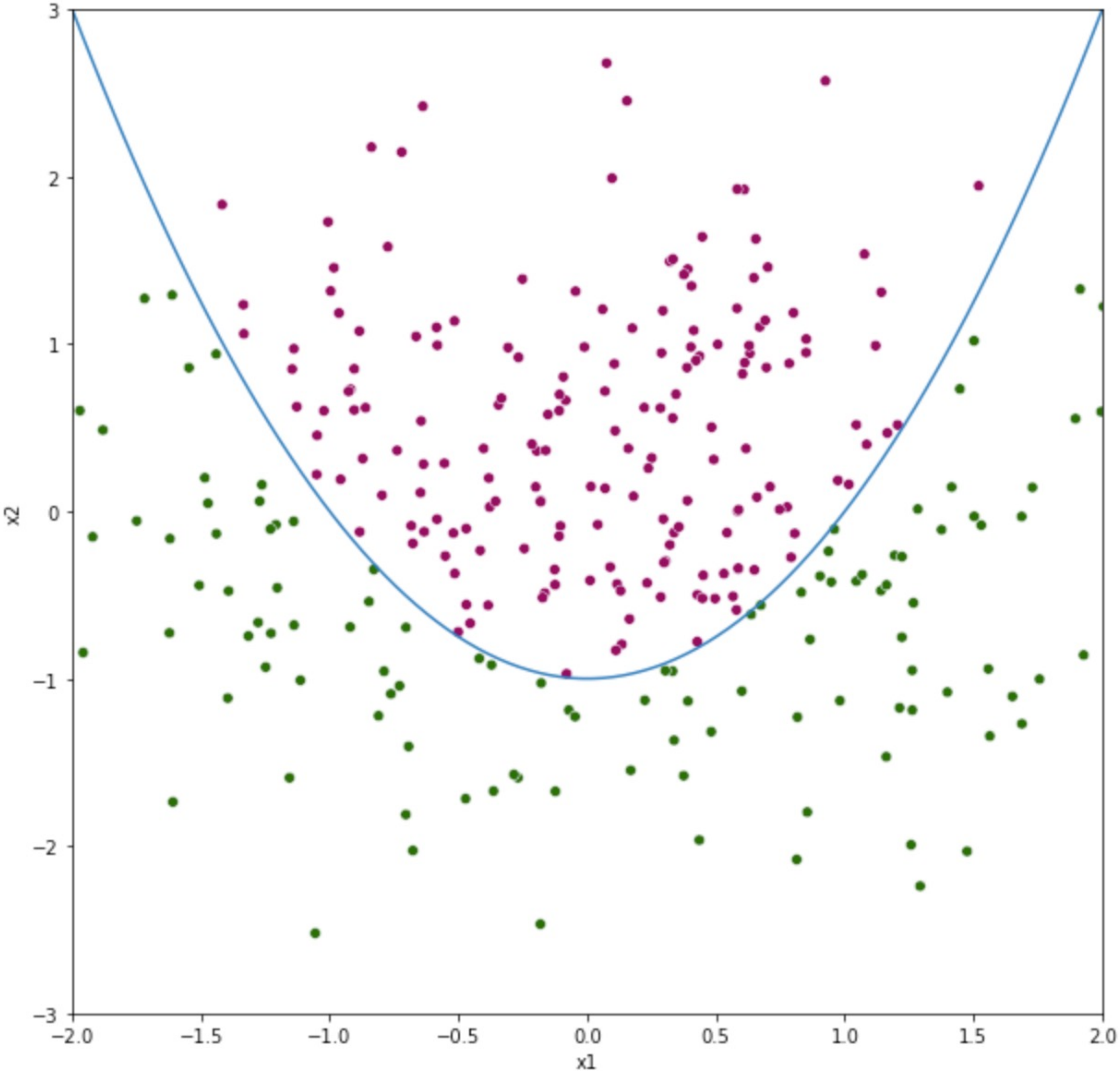


LINEAR SEPARABILITY

The same happens in the case of classification. At the moment the boundary between two classes is curved, we can’t separate the data with a linear model.

We will need a curve line, like the blue line to the right. But at that point, we have abandoned our linear model!

With that, we loose all advantages of the linear model...



THE SOLUTION

(At least, some of the time)

BASIS EXPANSION

The boundary in this example is:

$$x_2 = x_1^2 - 1$$

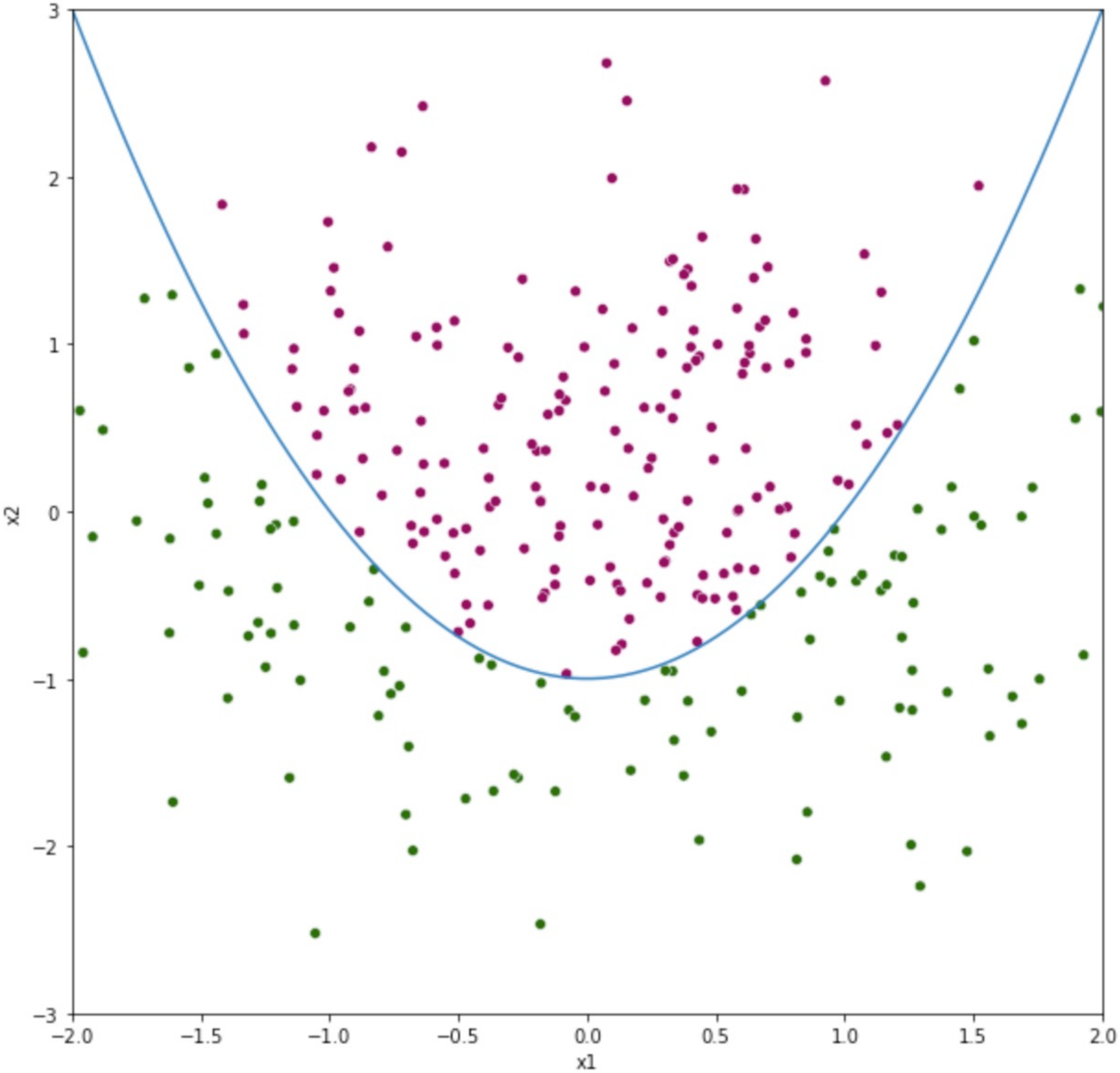
Or, similarly

$$x_2 - x_1^2 + 1 = 0$$

This is quadratic in $X = (x_1, x_2)$

But it is linear in:

$$\phi(X) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$



LINEAR SEPARABILITY

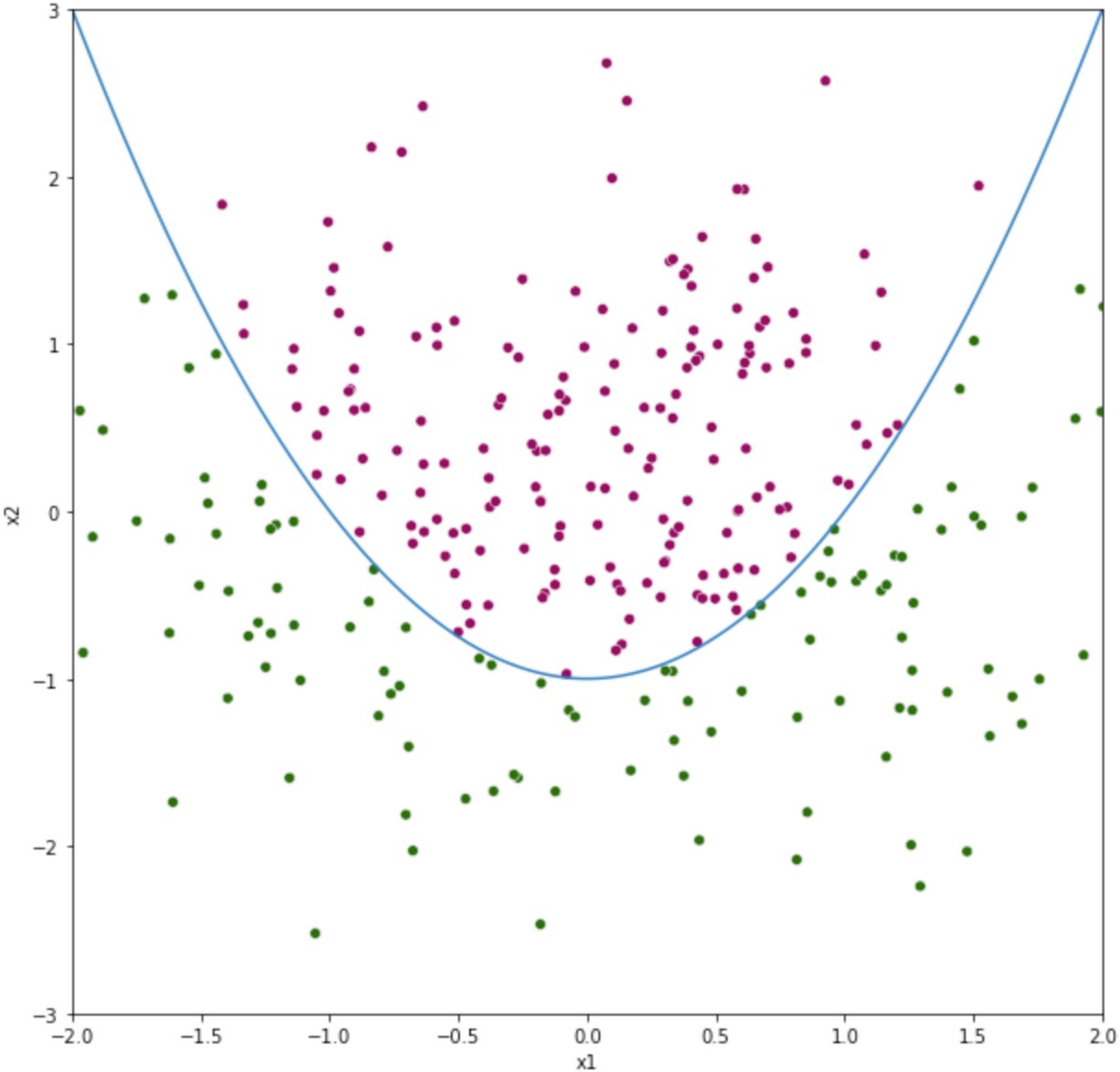
What did we just do?

We created new features. We went from 2-dimensional data, to 5-dimensional data:

$$\phi(X) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

We add the quadratic factors and the crossproduct.

We use the letter *phi* to denote this expansion.



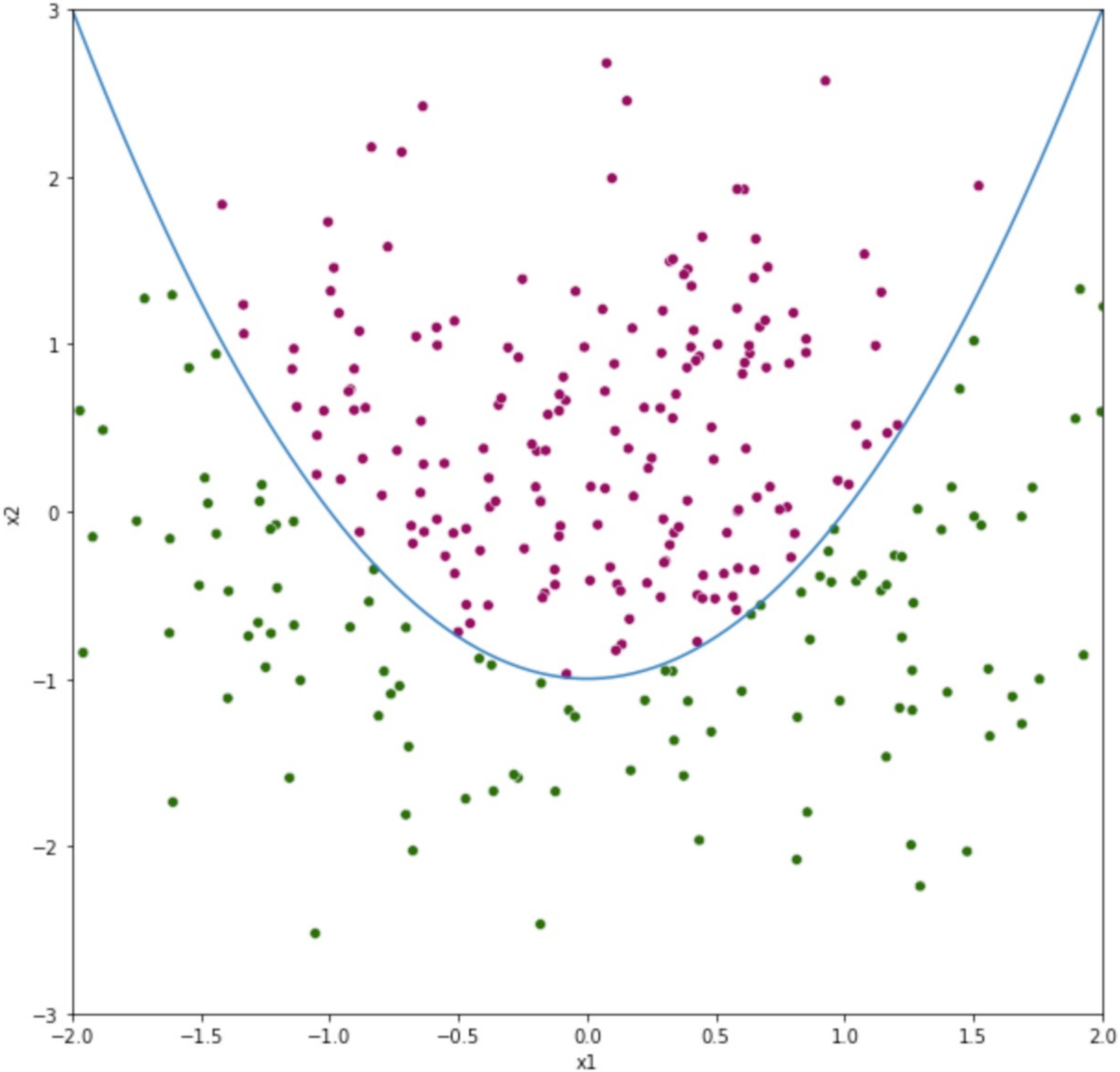
BASIS EXPANSION

This ‘trick’ is called Basis expansion**:**

We **<u>embed</u>** the data in a higher-dimensional space.

Suddenly, we **<u>can</u>** use a linear classifier on $\phi(X)$!

Let’s check that for ourselves…



BASIS EXPANSION

The boundary

$$x_2 - x_1^2 + 1 = 0$$

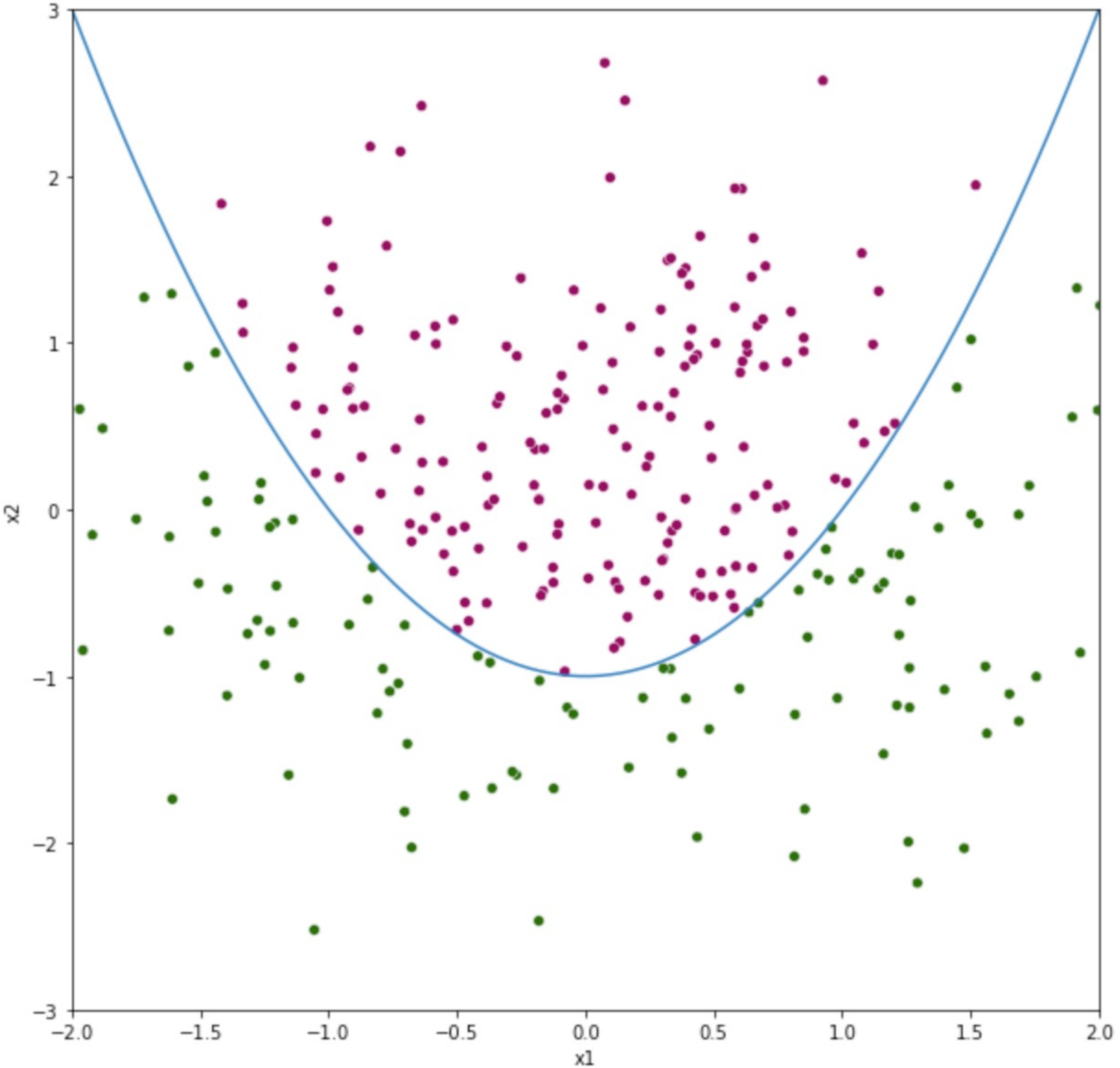
Can be learned as:

$$w \cdot \phi(X) + b = 0$$

When $\phi(X) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

What would the weights have to become to get the correct boundary?

Which weights should be set to 0? Which to either 1 or -1?



BASIS EXPANSION

The boundary

$$x_2 - x_1^2 + 1 = 0$$

Can be learned as:

$$w \cdot \phi(X) + b = 0$$

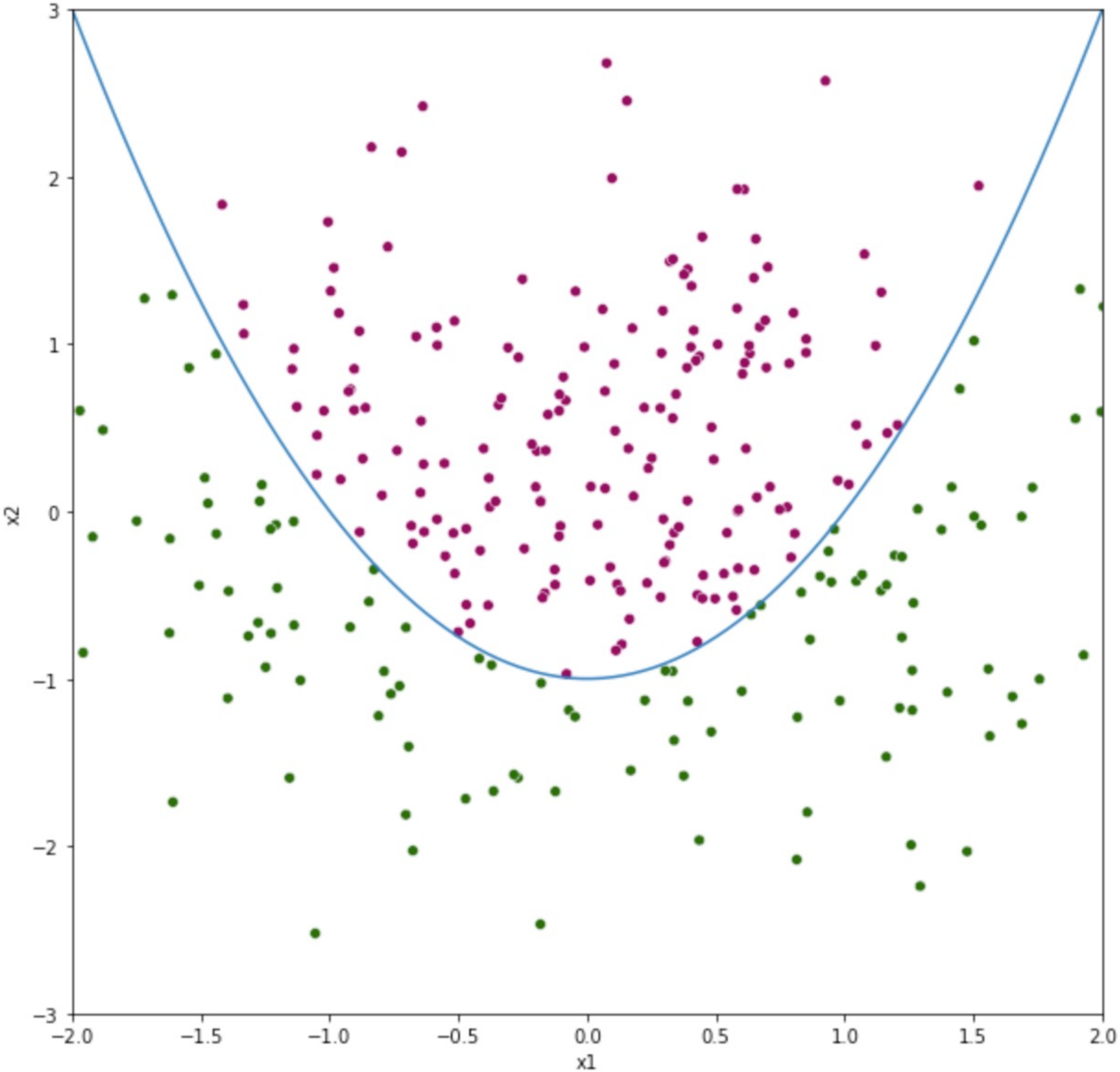
When $\phi(X) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

We will find that *w* and *b* are equal to:

$$w = (0, 1, -1, 0, 0)$$

$$b = 1$$

With these weights, we only pick the x_2 and x_1^2 features and set the rest to 0.



BASIS EXPANSION

This high-dimensional embedding can be generalised for *d* features:

For $\mathrm{X} = (x_1, x_2, \ldots, x_d)$ we obtain:

$\phi(X) = (x_1, \ldots, x_d,$ (the orginal features)

$x_1^2, \ldots, x_d^2,$ (the squares)

$x_1 x_2,\, x_1 x_3, \ldots, x_{d-1}\, x_d)$ (the cross-products)

Due to this, $\phi(X)$ scales kind of hard , roughly with $O(d^2)$ or to be exact with $2d + \frac{d(d-1)}{2}$

Note how this quadratic scaling could become a problem with large amounts of features. E.g. for the MNIST dataset, where we have 28x28=784 features, we will end up with 308503 features, just for d=2.

That is another problem we can solve with a kernel-trick.

THE KERNEL TRICK

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

What is $\phi(x) \cdot \phi(z)$?

$$\begin{aligned}\phi(x) \cdot \phi(z) = {} & \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \cdot \\ & \left(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1z_2\right)\end{aligned}$$

= $1 + 2\,x_1z_1 + 2\,x_2z_2 + x_1^2z_1^2 + x_2^2z_2^2 + 2\,x_1x_2z_1z_2$

= $(1 + \ x_1z_1 + x_2z_2)^2$

= $(1 + x \cdot z)^2$

Tadaaa! We only need to **take the dot product of the original vectors!** We can use 300.000+ features, but only have to calculate the dot product of 784.

This ‘trick’ is called a kernel: $K(x, z) = (1 + x \cdot z)^2$

This is a problem if $\phi(x)$ is really big!

This is what we want to avoid!

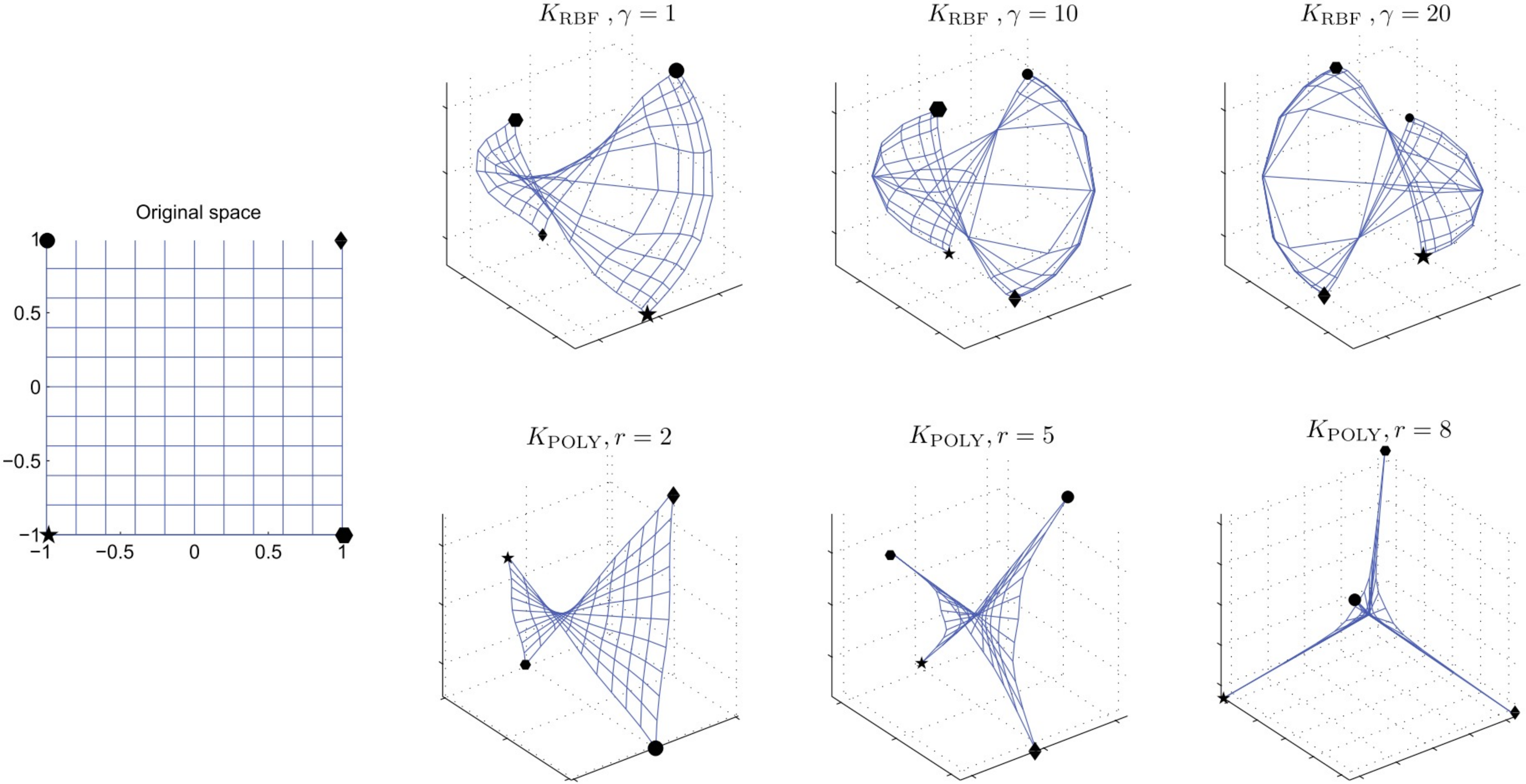
But we can reduce this

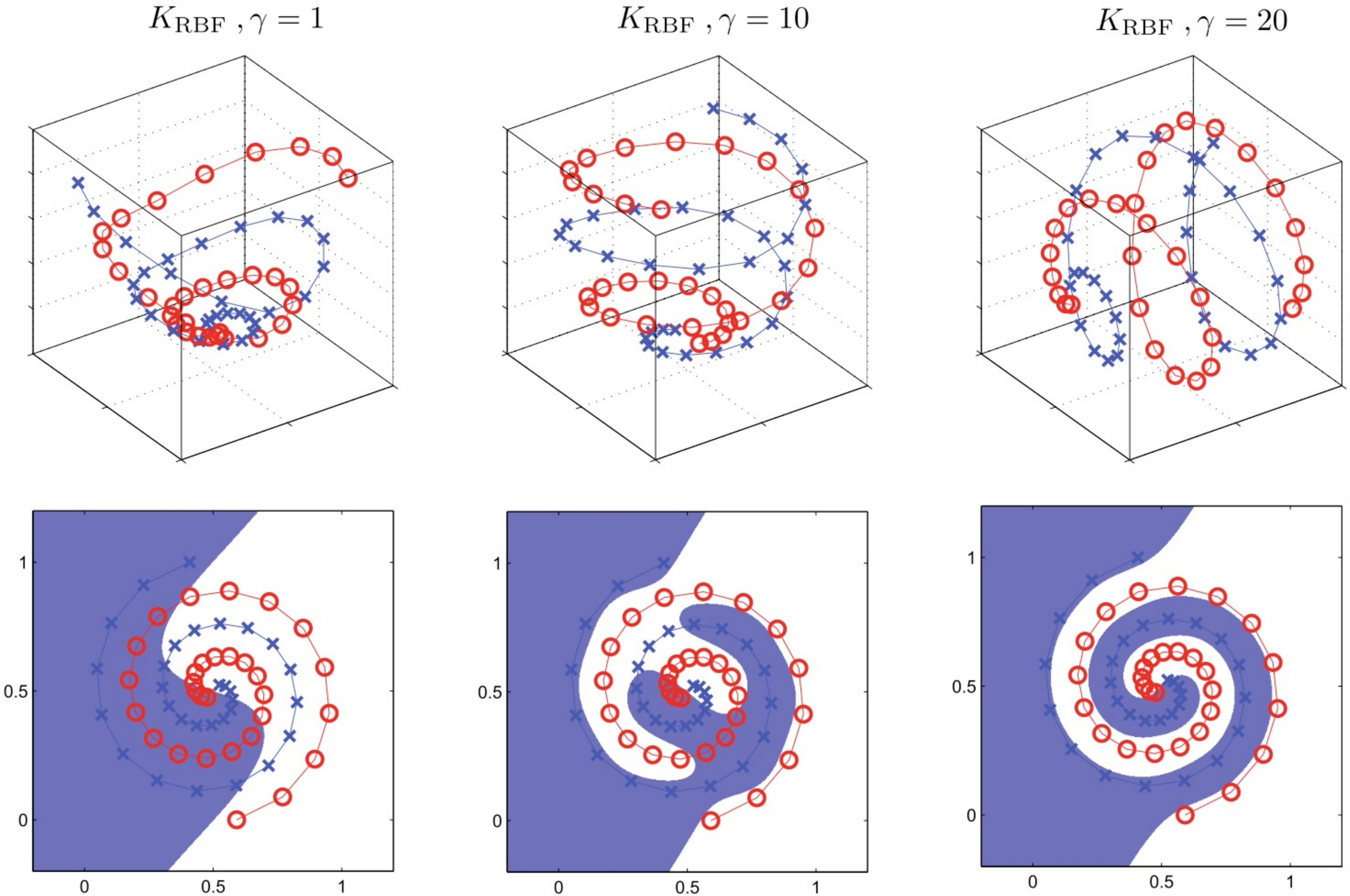
And this is much better

TYPES OF KERNELS

Linear:	$K(x,z) = x^T z$
Poly:	$K(x,z) = (\gamma x^T z + r)^d$
Gaussian:	$K(x,z) = exp(-\gamma \|x - z\|^2)$
Sigmoid:	$K(x,z) = tanh(\gamma x^T z + r)$

Kernels can have parameters. In sklearn:
γ is gamma
r is coef0


Original space
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1
$K_{\mathrm{RBF}}\ ,\gamma = 1$
$K_{\mathrm{RBF}}\ ,\gamma = 10$
$K_{\mathrm{RBF}}\ ,\gamma = 20$
$K_{\mathrm{POLY}}, r = 2$
$K_{\mathrm{POLY}}, r = 5$
$K_{\mathrm{POLY}}, r = 8$



$K_{\mathrm{RBF}}\ , \gamma = 1$

$K_{\mathrm{RBF}}\ , \gamma = 10$

$K_{\mathrm{RBF}}\ , \gamma = 20$

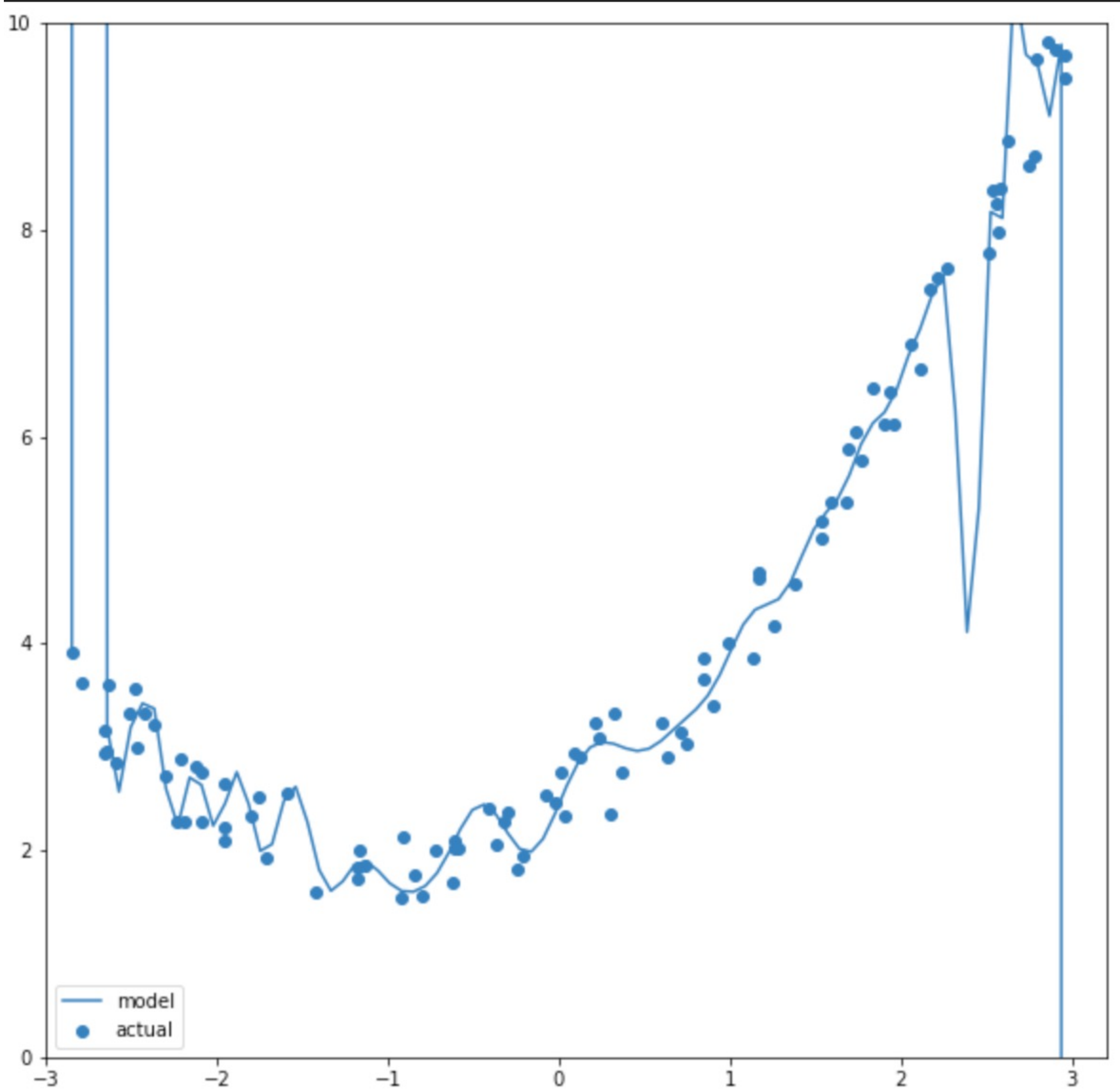
REGULARIZATION

REGULARIZATION

Sometimes we have a lot of features to start with, and sometimes we will need to create new features with basis expansion.

But, if we have a lot of features compared to the amount of data, we can get into problems.

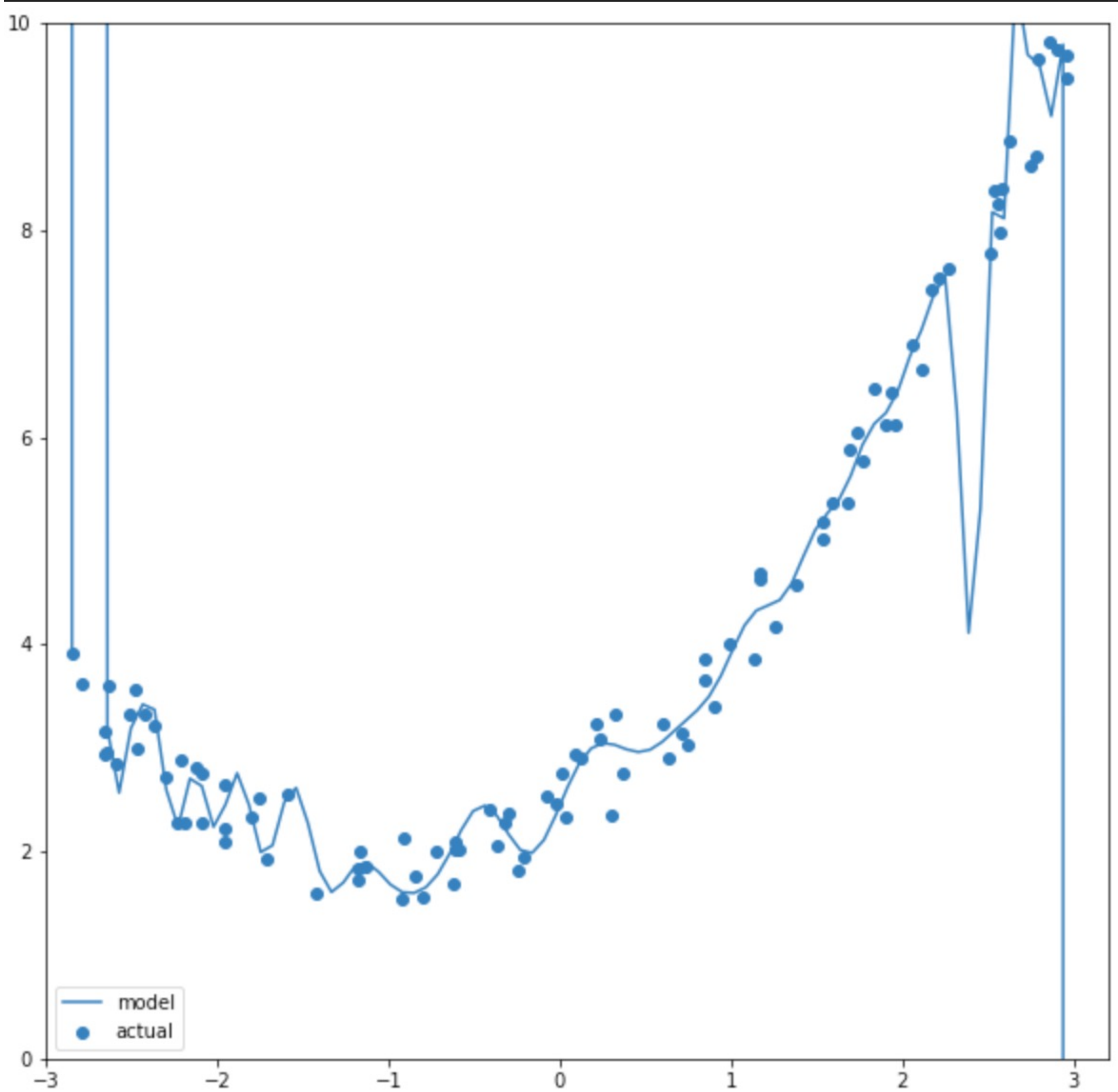
Our model becomes too complex and will overfit.



REGULARIZATION

The image illustrates a linear regression where the basis is expanded as polynomial features with a degree of 100.

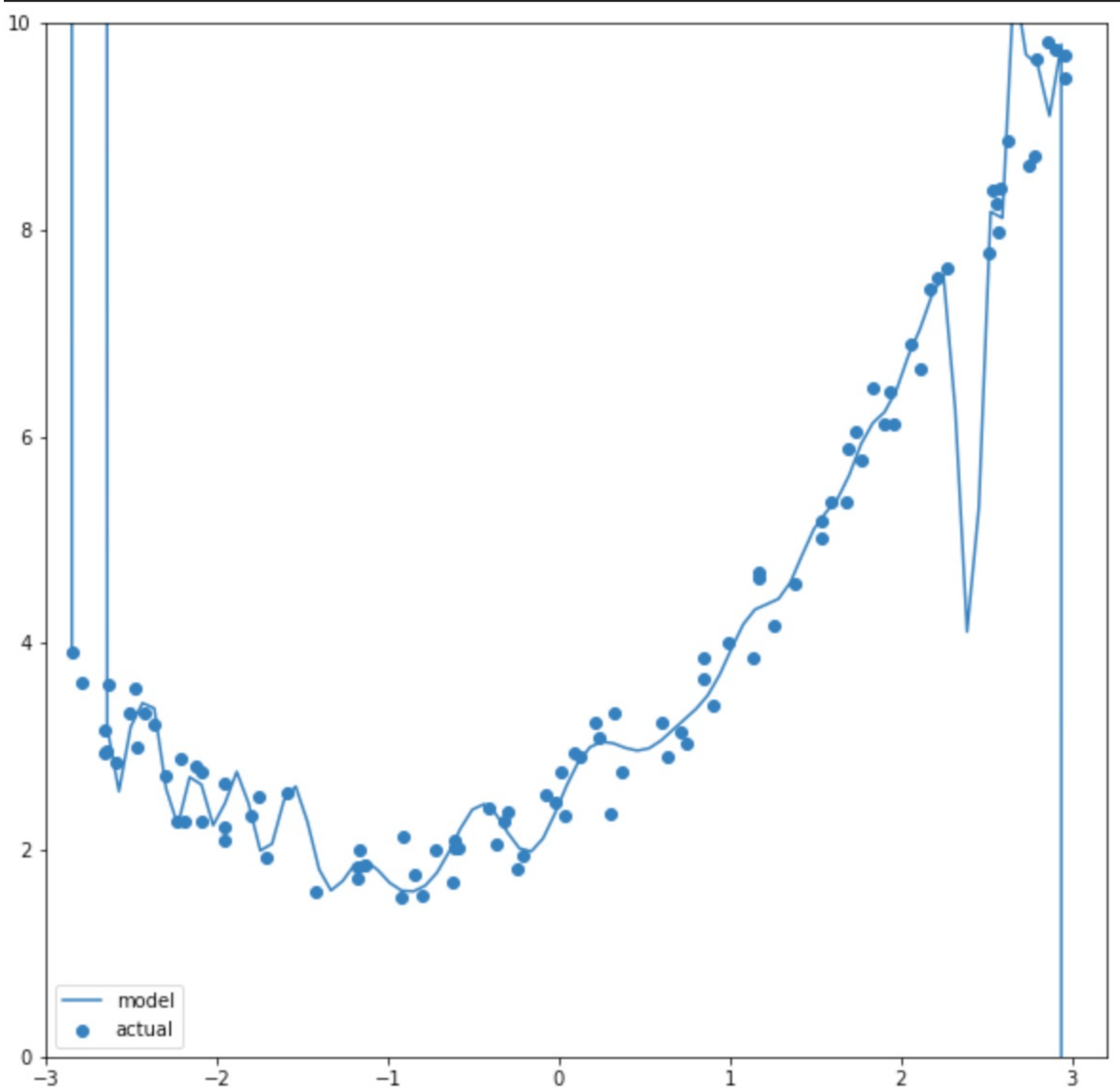
We will need a solution for this new problem we introduced with our last solution.



REGULARIZATION

One way to control this, is by using **regularization**.

With regularization we put a penalty on large weights and thus regain some control.



REGULARIZATION

Ridge Regression

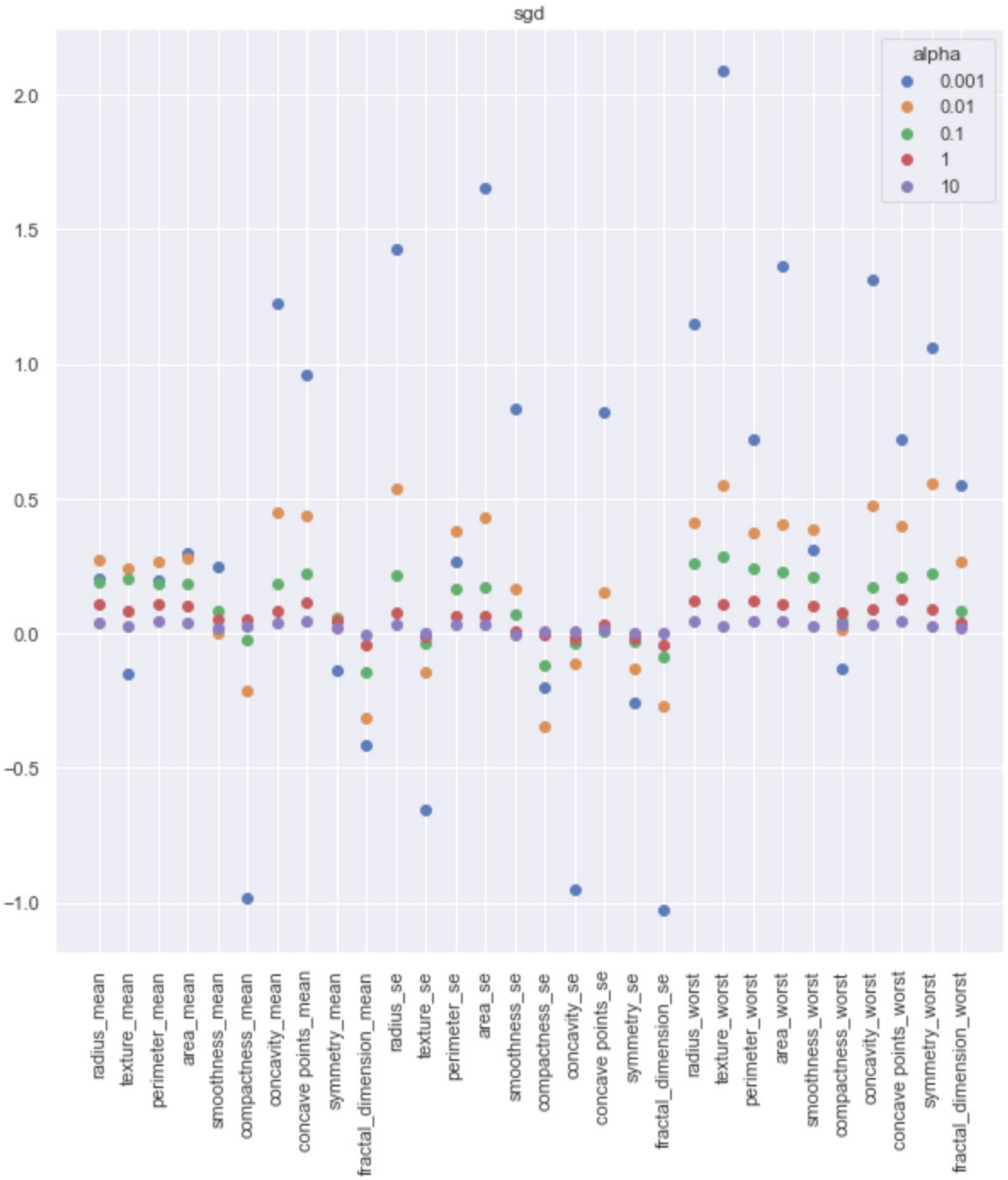
One way to regularize the weights (denoted as Θ) is to use Ridge Regression, also called l2. This is an extra term, added to the cost function $J(\Theta)$ (eg. Mean Square Error):

$$J(\Theta) = \mathrm{MSE}(\Theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \Theta_i^2$$

This extra term $\alpha \frac{1}{2} \sum_{i=1}^{n} \Theta_i^2$ is the sum of the square of all weights, multiplied with a parameter α.

We can see the impact of changing the value of α in the image.

With a large value of alpha, large weights get penalized and the model will seek a solution with smaller weights.



REGULARIZATION

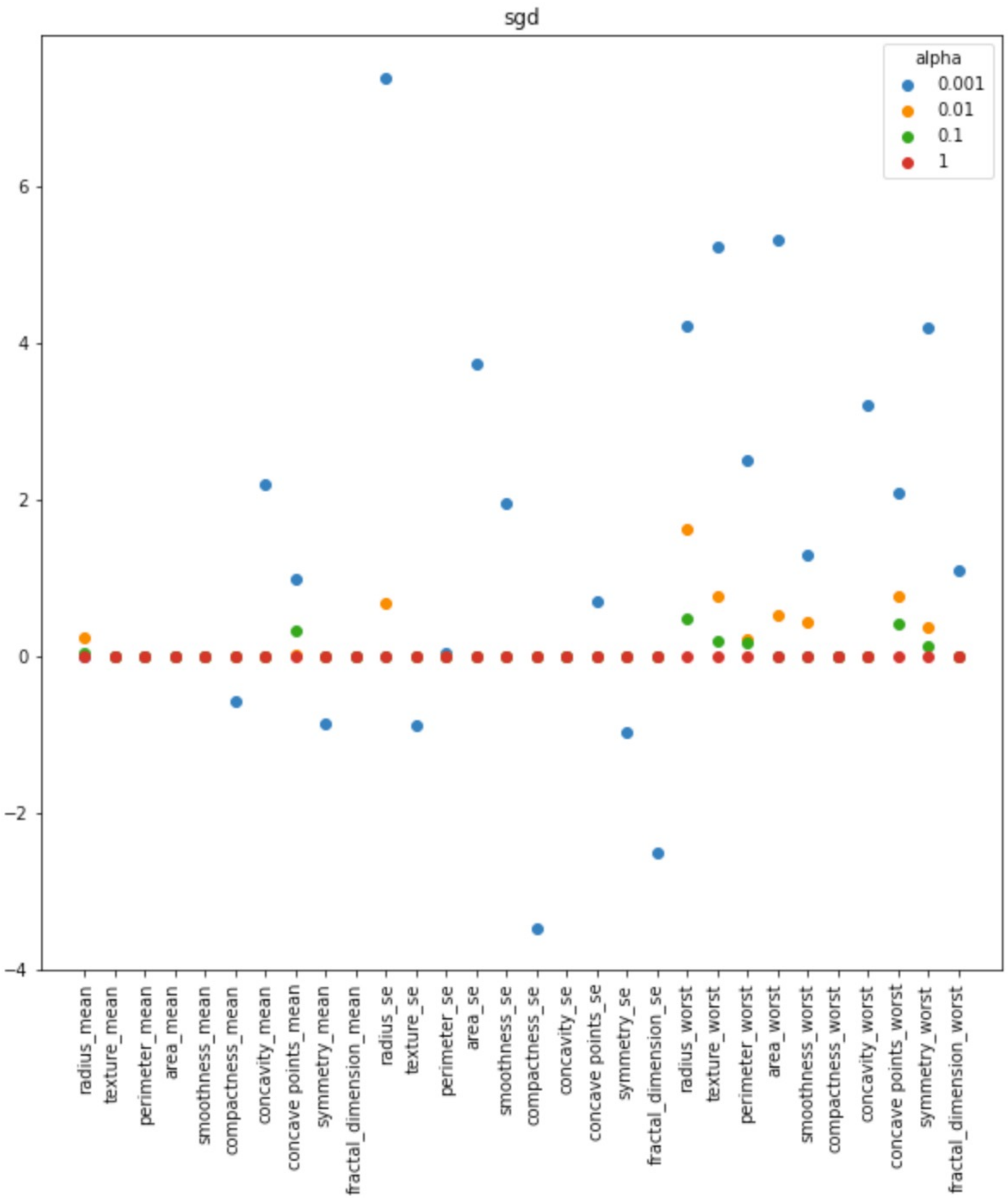
Lasso Regression

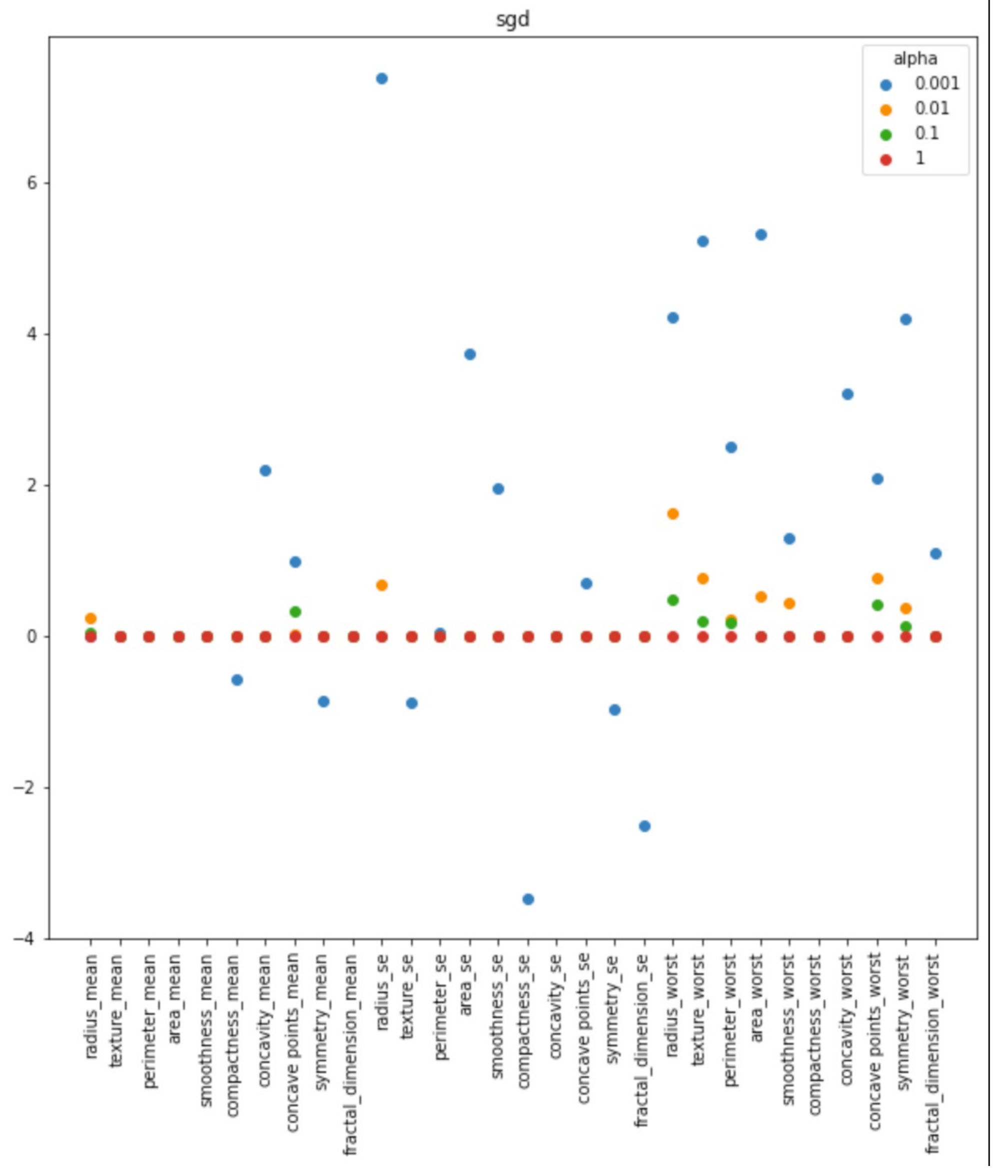
Another approach is called Lasso Regression, also named $\mathrm{l1}$. Here, to the cost function is added:

$$J(\Theta) = \mathrm{MSE}(\Theta) + \alpha \sum_{i=1}^{n} |\Theta_i|$$

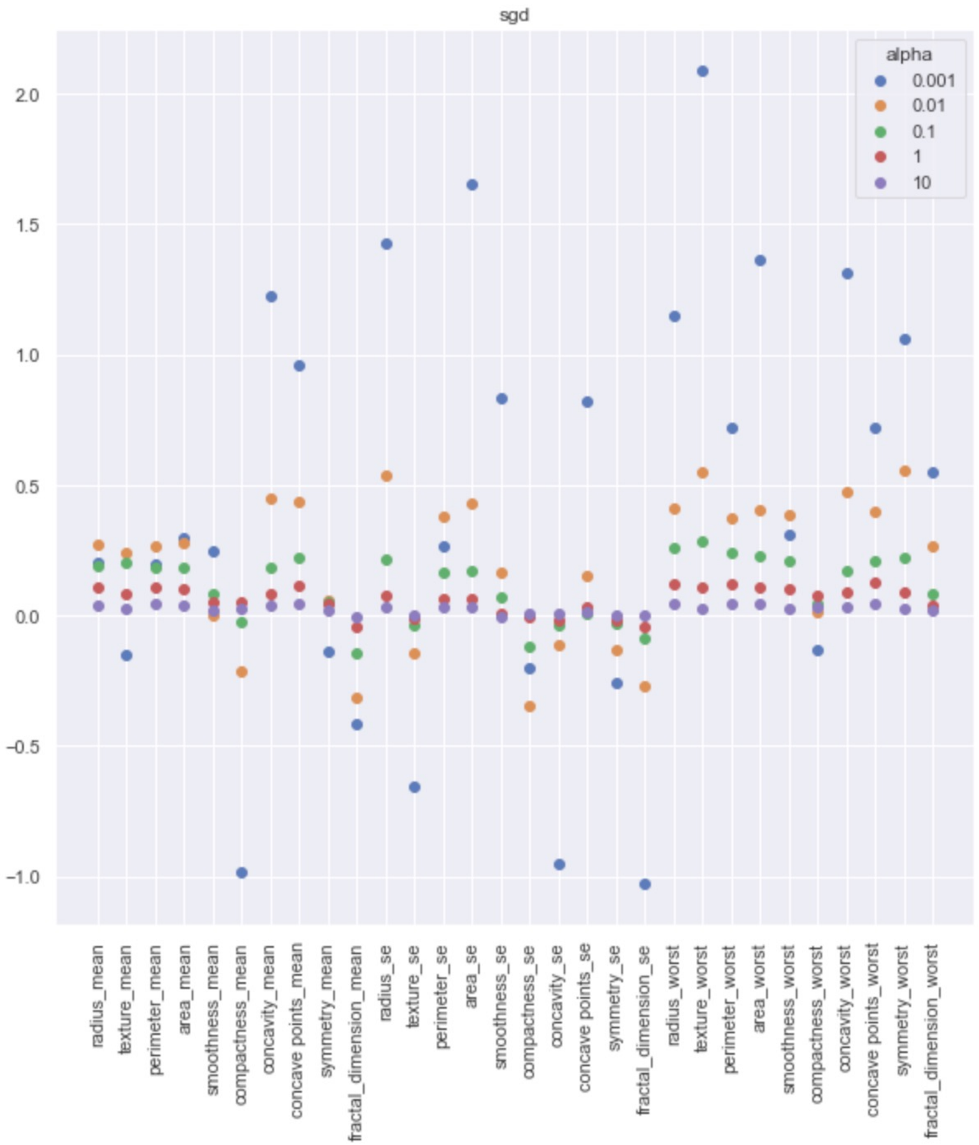
Now, the extra term is the sum of the absolute value of all weights, multiplied with a parameter α.

We can see the impact of changing the value of α in the image.





Lasso



Ridge

Spot the difference

REGULARIZATION

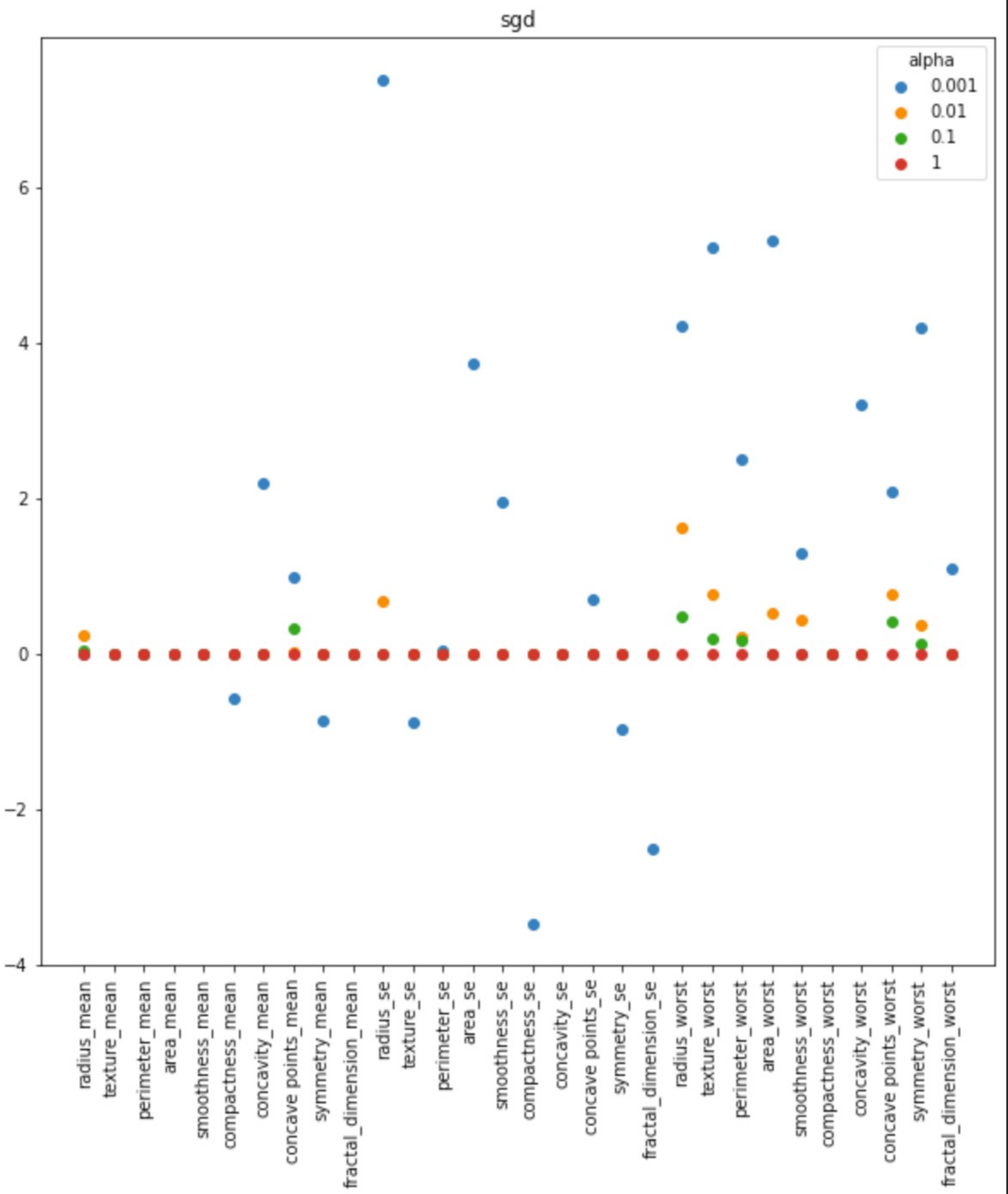
Elastic Net

Note that there is a difference between the two. Lasso tends to create much more sparse weights, were most weights drop to zero and only the most important weights are used.

Because it isn't always straight forward which of the two is better, we can add another factor to mix the two types of regression. This is called elastic net:

$$J(\Theta) = \text{MSE}(\Theta) + r\alpha \sum_{i=1}^{n} |\Theta_i| + \alpha \frac{1-r}{2} \sum_{i=1}^{n} \Theta_i^2$$

When the l1-ratio *r* = 1, this cost function is equal to Lasso. When r = 0, it is equal to Ridge. By picking a value in between 0 and 1, we mix the two.



USING REGULARIZATION FOR FEATURE SELECTION

While regularization is often used directly (often by default, there is also an indirect use of regularization as a way to do **feature selection** for more complex models:

Here, the regularization is part of a three step approach:

1. Do a gridsearch with **elasticnet** to find the optimal l1-ratio and alpha
2. Have a look at the weights. **Pick the k features** with the highest weights (and thus the most relevant)
3. Use this selection of features to **train a new model** that could be more complex than a linear model.

This approach is especially useful when you have much more features then observations and works better than using a correlation.