Overfitting and regularisation

Herman Kamper

Overfitting example

Suppose we want to fit a simple regression model (scalar input) using basis functions to a training set with $N = 10$ items.

Our goal:

$$\mathbf{y} \approx \mathbf{\Phi} \mathbf{w}$$

If we use two basis functions, the shapes will be:

$$\mathbf{y} \approx \mathbf{\Phi} \mathbf{w}$$

If instead we use ten basis functions, the shapes will be:

$$\mathbf{y} \approx \mathbf{\Phi} \mathbf{w}$$

But this is solvable exactly! We have ten equations with ten unknowns (the ten weights). So we can solve this exactly:

$$\mathbf{w} = \mathbf{\Phi}^{-1} \mathbf{y}$$

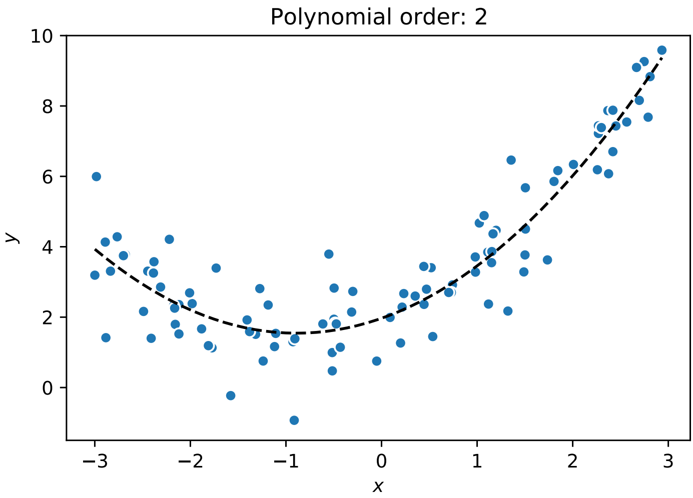
Questions

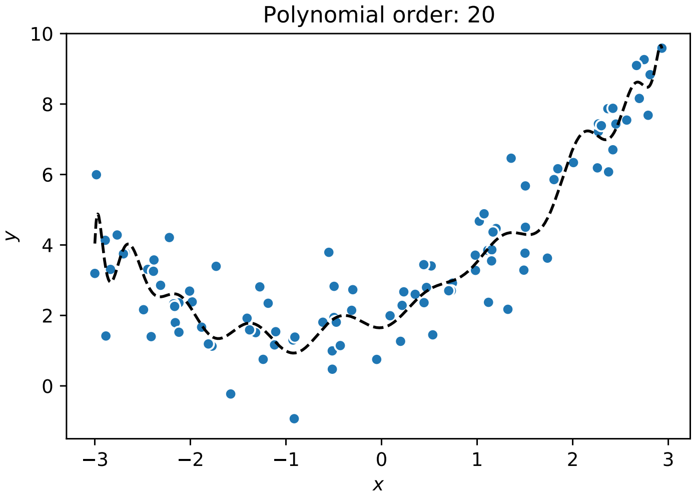
- What would the value of the loss J be?
- Would this be a good fit? Would this model make good future predictions?

Let us look at a few examples to develop an intuition for what happens when the "complexity" of our model is similar to the number of data points on which we train (or higher).

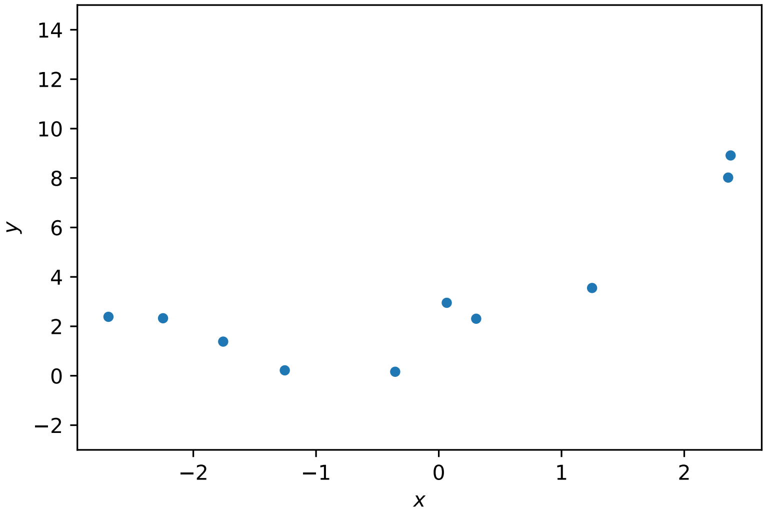
Polynomial regression examples

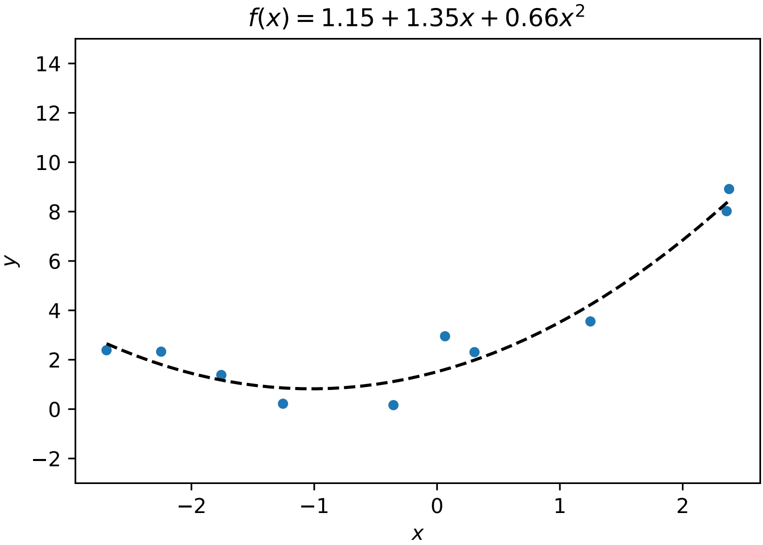
Quadratic data

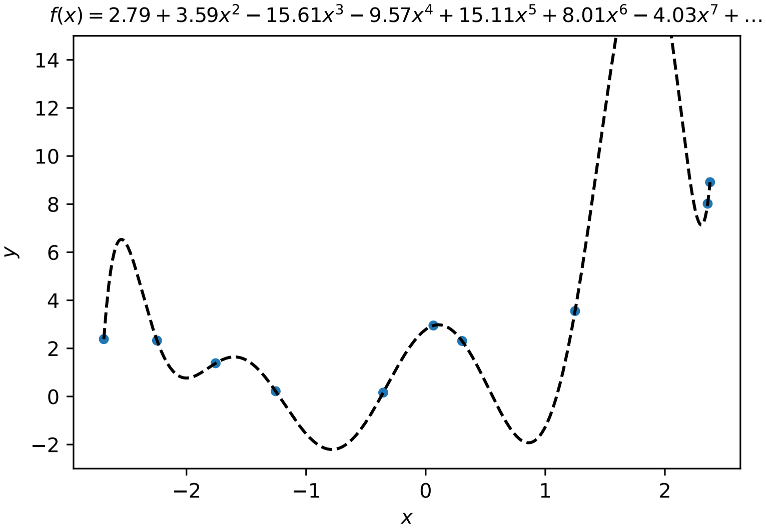




With less training items

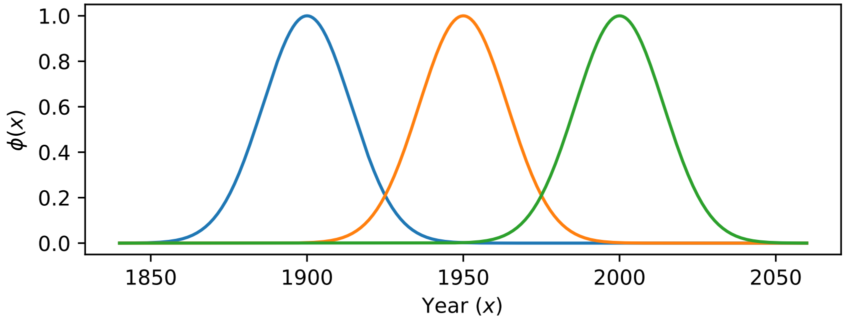



$f(x) = 1.15 + 1.35x + 0.66x^2$
14
12
10
8
6
4
2
0
−2
y
−2
−1
0
1
2
x

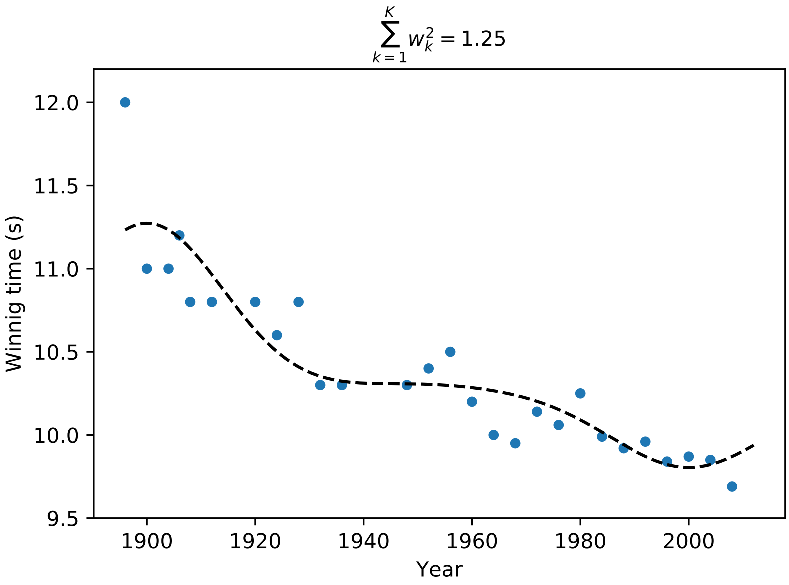

$f(x) = 2.79 + 3.59x^2 - 15.61x^3 - 9.57x^4 + 15.11x^5 + 8.01x^6 - 4.03x^7 + \ldots$
y
14
12
10
8
6
4
2
0
−2
−2
−1
0
1
2
x

Radial basis function examples

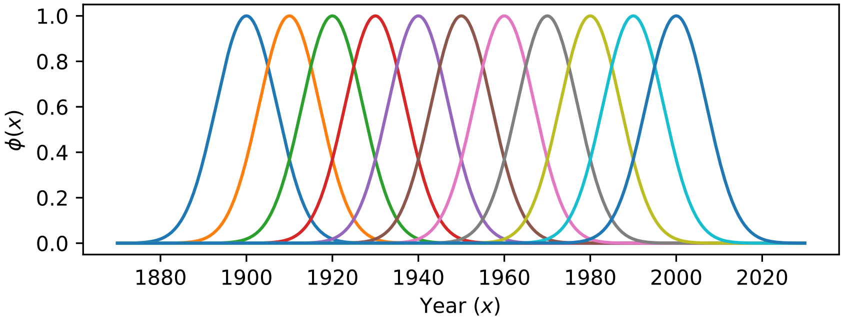
Basis functions:



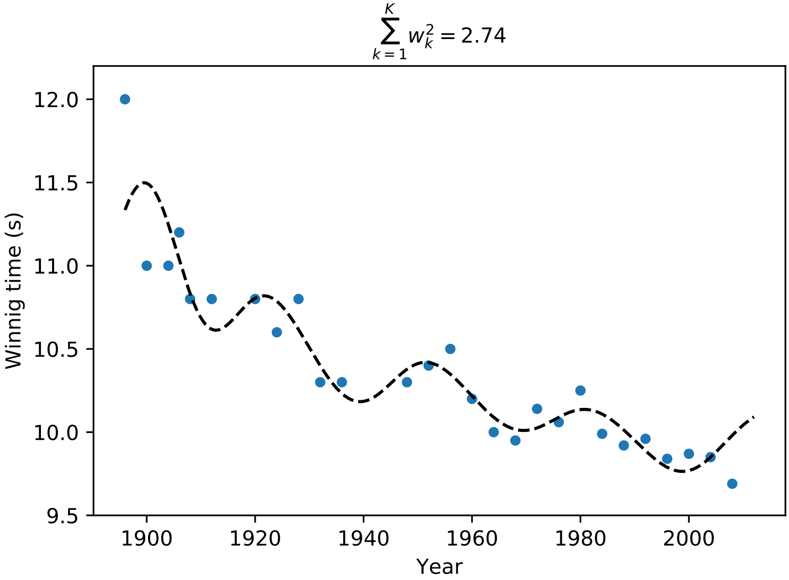
RBF with $c = [1900, 1950, 2000]$ and $h = 20$:



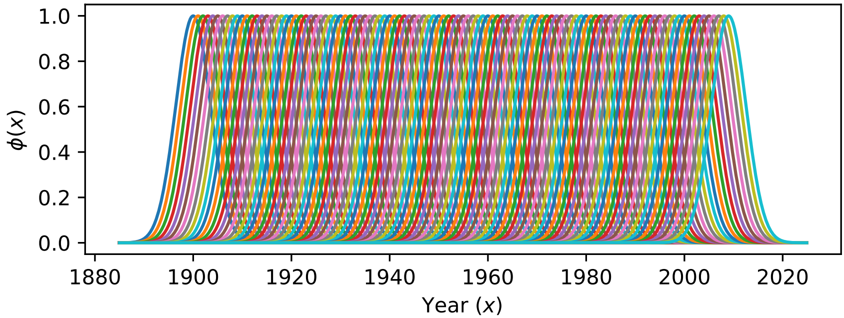
Basis functions:



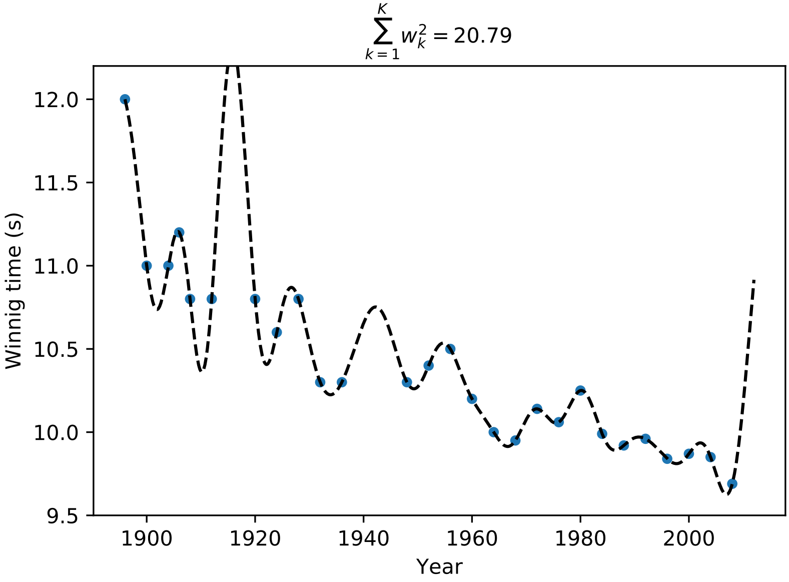
RBF with $c = [1900, 1910, \ldots, 2000]$ and $h = 10$:



Basis functions:



RBF with $c = [1900, 1901, \ldots, 2000]$ and $h = 1$:



Regularisation

We might want to fit higher-order models, but want a handle to control their “complexity” in some way.

Idea:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \{J(\mathbf{w}) + \text{penalty}(\mathbf{w})\}$$

Penalty functions that constrain $\mathbf{w}$ to be small are sometimes called *shrinkage* methods.

We consider two regularisation approaches:

- Ridge (L_2) regularisation
- Lasso (L_1) regularisation

Ridge (L_2) regularisation

$$J_\lambda(\mathbf{w}) = \sum_{n=1}^{N} \left(y^{(n)} - f(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 +$$

We normally don't regularise w_0. Why not?

An easy hack if you don't want to deal with w_0 is to zero-mean your data beforehand, i.e. the columns of $\mathbf{X}$ (or $\mathbf{\Phi}$) are normalised to have a mean of $\mathbf{0}$ and the target vector $\mathbf{y}$ are normalised to have a mean of 0.

We can then write the regulariser in vector form (good for vectorised implementations):

$$J_\lambda(\mathbf{w}) = \sum_{n=1}^{N} \left(y^{(n)} - f(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 + \lambda \mathbf{w}^\top \mathbf{w}$$

$$=$$

We can find a closed-form solution exactly as we did before:

$$\hat{\mathbf{w}} = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I} \right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Lasso (L_1) regularisation

$$J_\lambda = \sum_{n=1}^{N} \left(y^{(n)} - f(\mathbf{x}^{(n)}; \mathbf{w}) \right)^2 +$$

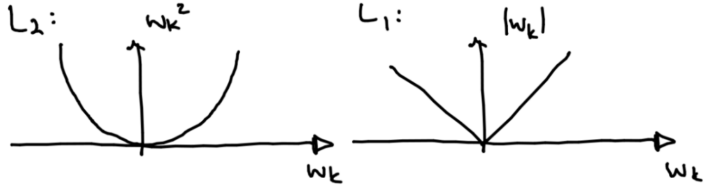
This loss function is still convex (unique minimum) but not “smooth” (differentiable in all places) so we can’t find a closed-form solution.

But other methods can be used to optimise it (e.g. gradient descent).

L_1 regularisation has the effect of pushing weights to absolute 0. This can be useful for interpreting data or a model (but be careful!).

Why does L_1 push weights to zero but not L_2 regularisation?

Just intuitively from the loss functions:

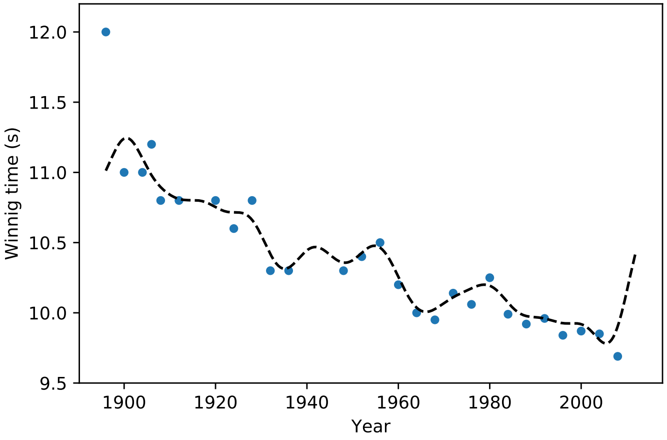


ISLR 6.2 gives a more formal explanation (non-examinable).

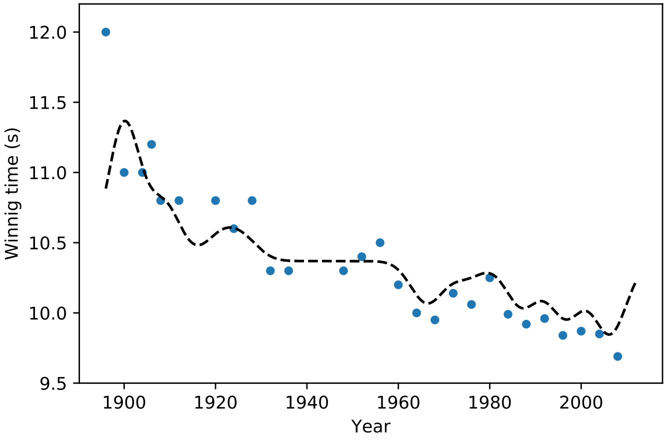
Regularisation examples

RBF with $c = [1900, 1901, \ldots, 2000]$ and $h = 1$:

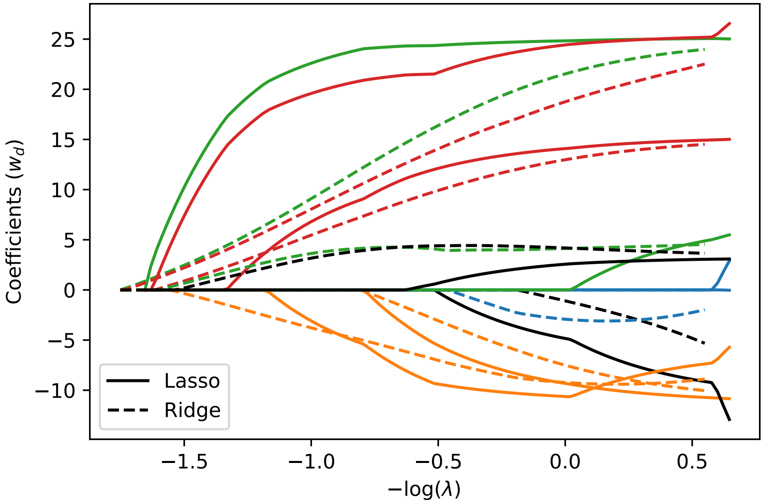
Ridge regression with $\lambda = 1$: $\sum_{k=1}^{K} w_k^2 = 0.68$



Lasso regression with $\lambda = 0.01$: $\sum_{k=1}^{K} w_k^2 = 1.52$



Lasso and ridge regression on diabetes data:[1]

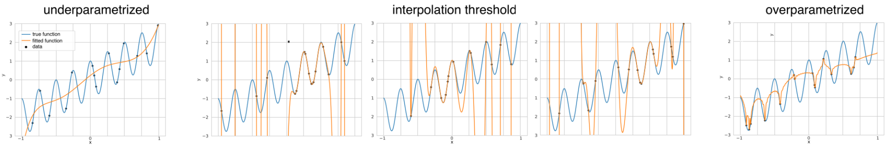


[1] Example from scikit-learn.

Advanced: Double descent

We have seen that when we have exactly the same number of polynomial features K as data points N, we get a zero training error but a very large validation error. But there is an interesting observation that, if we continue to increase the number of features K, we start doing better again on held-out validation data (Schaeffer et al. 2023).

For this over-parametrised case, we cannot use the the ordinary least squares equation any more becomes it becomes ill-posed. We therefore need to constrain the optimisation problem in some way. It seems that, if you do this, and you continue to grow K, you implicitly get even more regularisation, which makes the model fit quite well. Here is the figure from (Schaeffer et al. 2023):



There are still some arguments about this, but what is interesting is that I often have found this in practice with neural networks as well: As you continue to make them bigger (which pushes the training error to zero at some point), you continue to get better validation performance (even if nothing happens to the training error). Apart from size, I have also seen this when you keep the size fixed but continue to train: at some point the training error doesn't decrease any more (it just fluctuates around a stable point), but your validation error continues to go down.

Videos covered in this note

- Linear regression 4: Overfitting (10 min)
- Linear regression 5: Regularisation (15 min)

Reading

- ISLR 6.2
- ISLR 6.2.1
- ISLR 6.2.2