Deep learning

2.2. Over and under fitting

François Fleuret

https://fleuret.org/dlc/

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

Would you feel confident if you interviewed one candidate and they make a perfect score?

What about interviewing ten candidates and picking the best? What about interviewing one thousand?

Here the candidates are our models and the questions are the training examples used to pick the best one.

With

$$Q_k^n \sim \mathscr{B}(0.5),\ \ n = 1, \ldots, 1000,\ \ k = 1, \ldots, 10,$$

independent standing for "candidate n answers question k correctly", we have

$$\forall n,\ \ P(\forall k, Q_k^n = 1) = \frac{1}{1024}$$

and

$$P(\exists n, \forall k, Q_k^n = 1) \simeq 0.62.$$

So there is 62% chance that among $1,000$ candidates answering completely at random, at least one will score perfectly.

Selecting a candidate based on a statistical estimator biases the said estimator for that candidate. And you need a greater number of "competence checks" if you have a larger pool of candidates.

Notes

Here, we put the problem of selecting a candidate with 10 binary questions in a mathematical way: We consider 1,000 candidates answering the 10 questions. The probability of a candidate answering perfectly all the questions is $1/2^{10}$, because the total number of possible answer is $2^{10} = 1024$, and only one of them corresponds to all good answers, and consequently the probability to respond incorrectly to at least one question is

$$1 - 1/2^{10}.$$

Since the responses of candidates are independent events, the probability that they all respond incorrectly is,

$$\left(1 - 1/2^{10}\right)^{1000} = 0.376423\ldots.$$

Over and under-fitting, capacity. K-nearest-neighbors

A simple classification procedure is the "K-nearest neighbors."

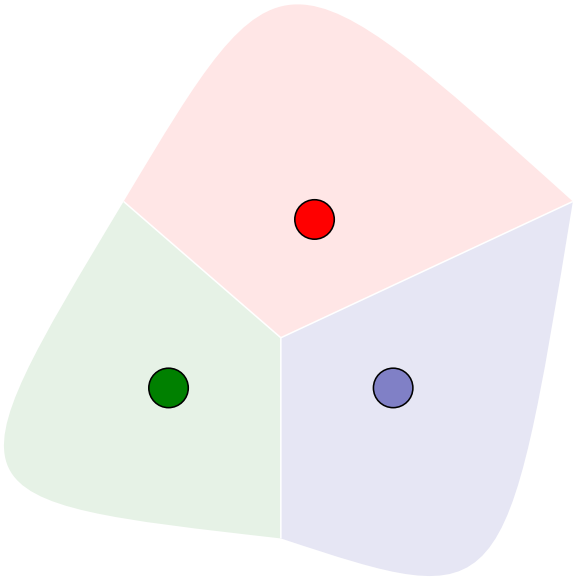
Given

$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \ldots, C\}, \;\; n = 1, \ldots, N$$

to predict the y associated to a new x, take the y_n of the closest x_n:

$$\begin{aligned} n^*(x) &= \operatorname*{argmin}_n \|x_n - x\| \\ f^*(x) &= y_{n^*(x)}. \end{aligned}$$

This recipe corresponds to $K = 1$, and makes the empirical training error zero.



$K = 1$

Notes

The three disks correspond to three "training points" in $\mathbb{R}^2$ of three different classes.
The colored areas show the prediction of a 1-NN that classifies a point as being of the class of the closest training points. For instance all the points closest to the red dots than to the blue and green are colored red.
In the case of an Euclidean space as here, given $x_n, \quad n = 1, \ldots, N$, the set of points closer to x_n than to any of the $x_k, \quad k \neq n$ is called the Voronoi cell of x_n.

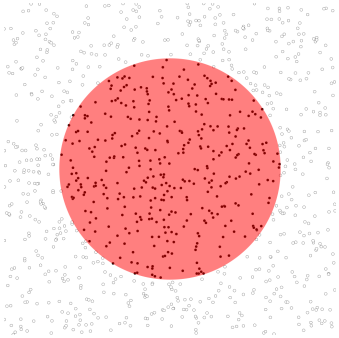
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \to \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

It can be made more stable by looking at the $K > 1$ closest training points, and taking the majority vote.

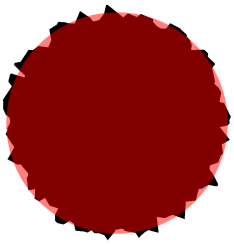
If we let also $K \to \infty$ "not too fast", the error rate is the (optimal!) Bayes' Error rate.

Notes

The K-nearest neighbor rule assigns to a sample the label of the K closest training samples following a majority vote: the most frequent class among the K closest training samples "wins".
It can be shown that when $N \to \infty$ and when K grows at roughly the square root of N (i.e. grows slower than N), the asymptotic error rate reaches the optimal Bayes' error, because we look at more and more samples, but they are more and more geometrically localized.



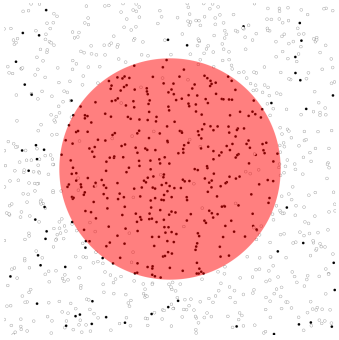
Training set



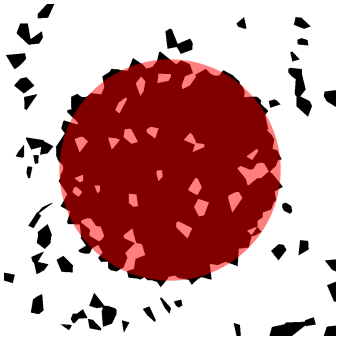
Prediction (K=1)

Notes

We consider a 2d toy example. Training points are sampled uniformly in a square. The true class is illustrated with the red disk: training samples within the circle have label 1 and are depicted as black dots, and training samples outside have label 0 and are depicted as white dots.
The prediction of the 1-nearest-neighbor is shown on the image below. The black area is the set of points which are classified as being of class 1, and the rest is the set of points which are classified as being of class 0.
We can see that the classification is good inside and outside the circle. But it is noisy at the boundary, especially when the density of training points is low.



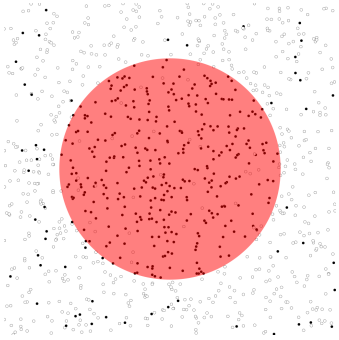
Training set



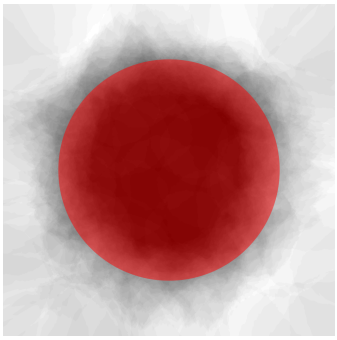
Prediction (K=1)

Notes

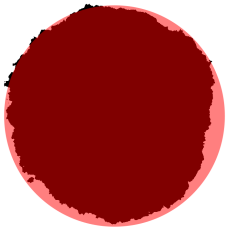
In this situation, we add noise to the training labels and flip 10% of them at random, keeping everything else the same.
The 1-nearest-neighbor results in noisy predictions, since any flipped label flips the predictions for all the point in the corresponding Voronoi cell.
This can be mitigated by increasing K so that the prediction remains unchanged in spite of some labels being incorrect.



Training set



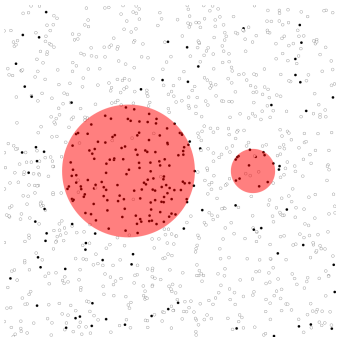
Votes (K=51)



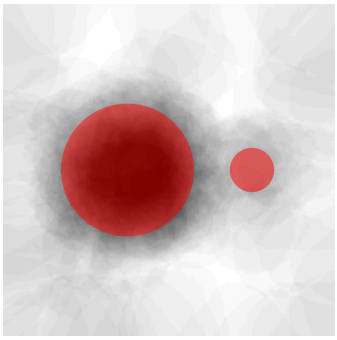
Prediction (K=51)

Notes

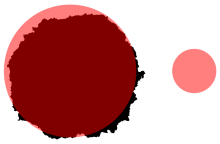
Finding the best K can be done using another set of samples which are generated the same way as the training set.
Optimizing K consists in computing the accuracy on the other set for different values of K and keeping the one which achieves the lowest error. This results here in picking $K = 51$.
The "Votes" picture depicts the number of votes that a test point gets. Black corresponds to the K closest training point being all of class 1 and white to them being all of class 0. Gray levels stand for intermediate counts.
The "Prediction" picture shows the majority decision, which is much smoother than with $K = 1$: there are no longer the artifacts due to the noise in the training data.



Training set



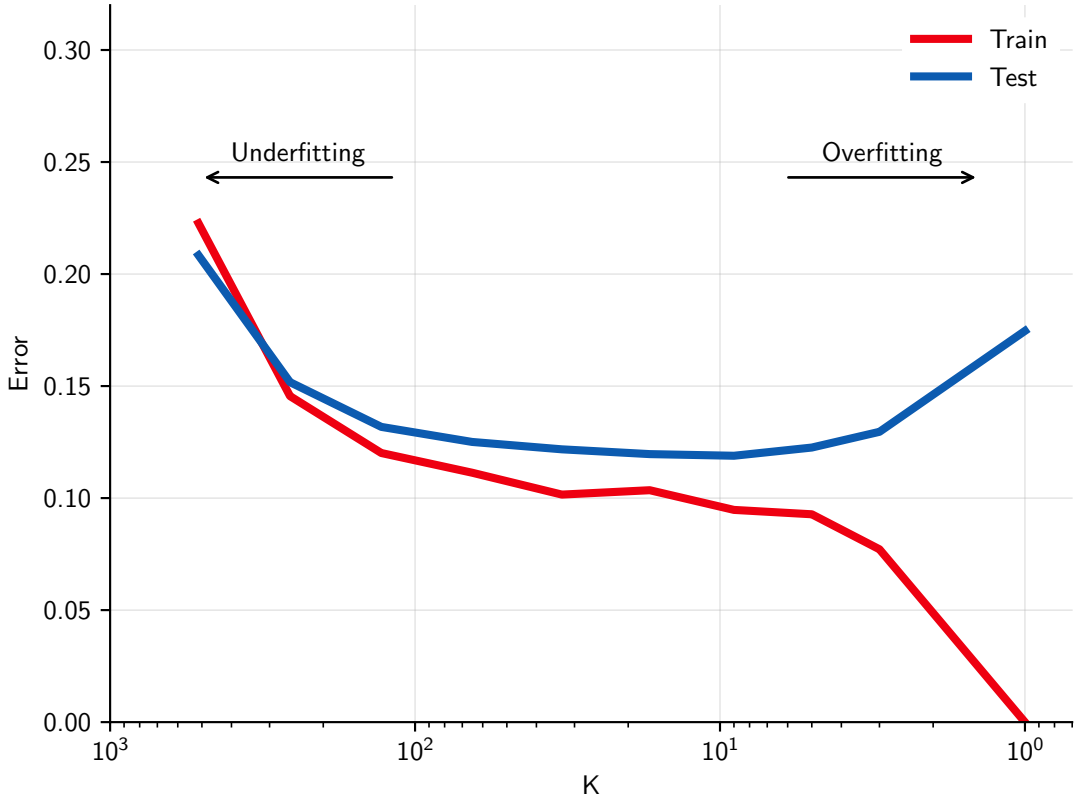
Votes (K=51)



Prediction (K=51)

Notes

Unfortunately, this recipe does not work on a slightly different data set.
The new data we are considering here correspond to a ground truth composed two disks of different radii inside which points have label 1.
The value $K = 51$ was appropriate when there was only one disk, but is no longer here: the small disk is completely erased in the final prediction.
This example is interesting because this raises a philosophical issue: at which point should we consider that structure is noise. If we look at the training image without the red disks, the training data of class 1 around the small disk could perfectly be random noise.



Notes

This graph shows the error rate of the prediction estimated on the training examples (aka "train error") in red, and the error rate estimated on another set of samples generated with the same procedure, but that differs due to the randomness of the process (aka "test error") in blue, for different values of K. This is the setup with noise in the training labels.

Regarding the train error, for $K = 1$, the train error is 0 since each training point is predicted as being of its own label. When K gets larger (going from right to left), votes are counted in a larger neighborhood, and the train error increases since "noisy labels" are not predicted correctly. At the limit K is the size of the training set and the predictor has a constant response, equal to the dominant class among training examples.

Regarding the test error, for $K = 1$, there is no robustness to flipped training labels, which contrary to what happen on the train data results in a high error rate. When K increases, at first the regularization helps and the error goes down, until it get too strong and completely hide the structure of the data. Contrary to the behavior on the train data, the error hence is not monotonic with K and there is an optimal value.

"Over-fitting" occurs when the model fits the training data too well and models the noise of the data (here small K).

"Under-fitting" occurs when the model is not flexible enough to model the actual structure of the data (here large K).

Over and under-fitting, capacity, polynomials

Given a polynomial model

$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R},\ f(x;\alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \ldots, N$, the quadratic loss is

$$\begin{aligned}
\mathscr{L}(\alpha) &= \sum_n (f(x_n;\alpha) - y_n)^2 \\
&= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n \right)^2 \\
&= \left\| \begin{pmatrix} x_1^0 & \ldots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \ldots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2.
\end{aligned}$$

Hence, minimizing this loss is a standard quadratic problem, for which we have efficient algorithms.

Notes

A polynomial of degree D is parameterized by $D+1$ coefficients.
Given N training points (x_n, y_n), polynomial regression consists of finding the coefficients of the polynomial that minimize the quadratic loss between the actual training points and the polynomial prediction. The goal is to get each $f(x_n;\alpha)$ as close as possible [in a quadratic sense] to y_n for all n.
For the nth training point, the computation of $f(x_n;\alpha)$ can be written as a dot product in a matrix form: we build the vector $[1, x_n, x_n^2, \ldots, x_n^D]$ and we have

$$\begin{aligned}
f(x_n;\alpha) &= \sum_{d=0}^{D} \alpha_d x_n^d \\
&= \begin{bmatrix} 1 & x_n & x_n^2 & \ldots & x_n^D \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{D-1} \\ \alpha_D \end{bmatrix}
\end{aligned}$$

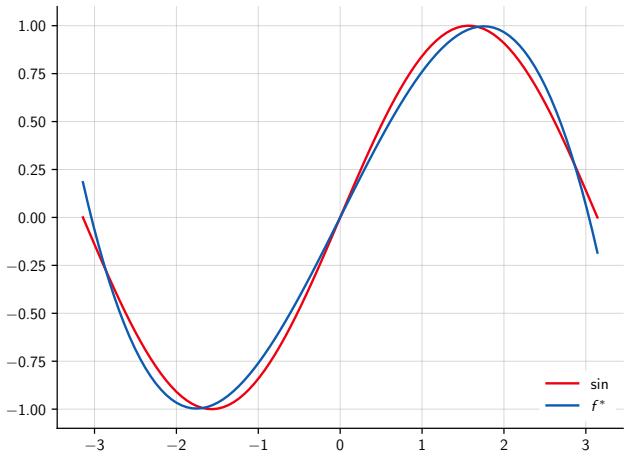
$$\underset{\alpha}{\text{argmin}} \left\| \begin{pmatrix} x_1^0 & \dots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \dots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2$$

```
def fit_polynomial(D, x, y):
    # Broadcasting magic
    X = x[:, None] ** torch.arange(0, D + 1)[None]

    # Least square solution
    return torch.linalg.lstsq(X, y).solution
```


Notes

Function `fit_polynomial` returns the coefficients of the polynomial after optimization.
The first step is to build the matrix containing the power of the x_n.
The second step builds a tensor of the y_n values for calling `torch.linalg.lstsq`. We ignore the second returned value

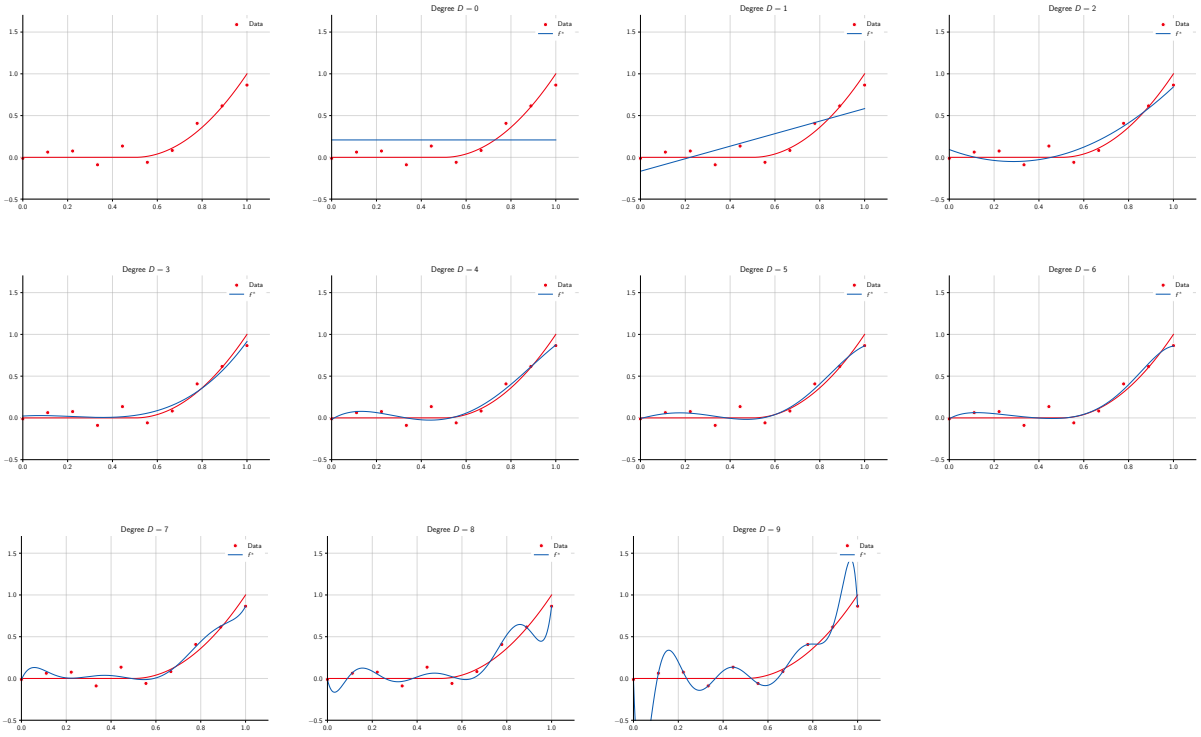
```
D, N = 4, 100
x = torch.linspace(-math.pi, math.pi, N)
y = x.sin()
alpha = fit_polynomial(D, x, y)

X = x[:, None] ** torch.arange(0, D + 1)[None]

y_hat = X @ alpha

for k in range(N):
    print(x[k].item(), y[k].item(), y_hat[k].item())
```



Notes

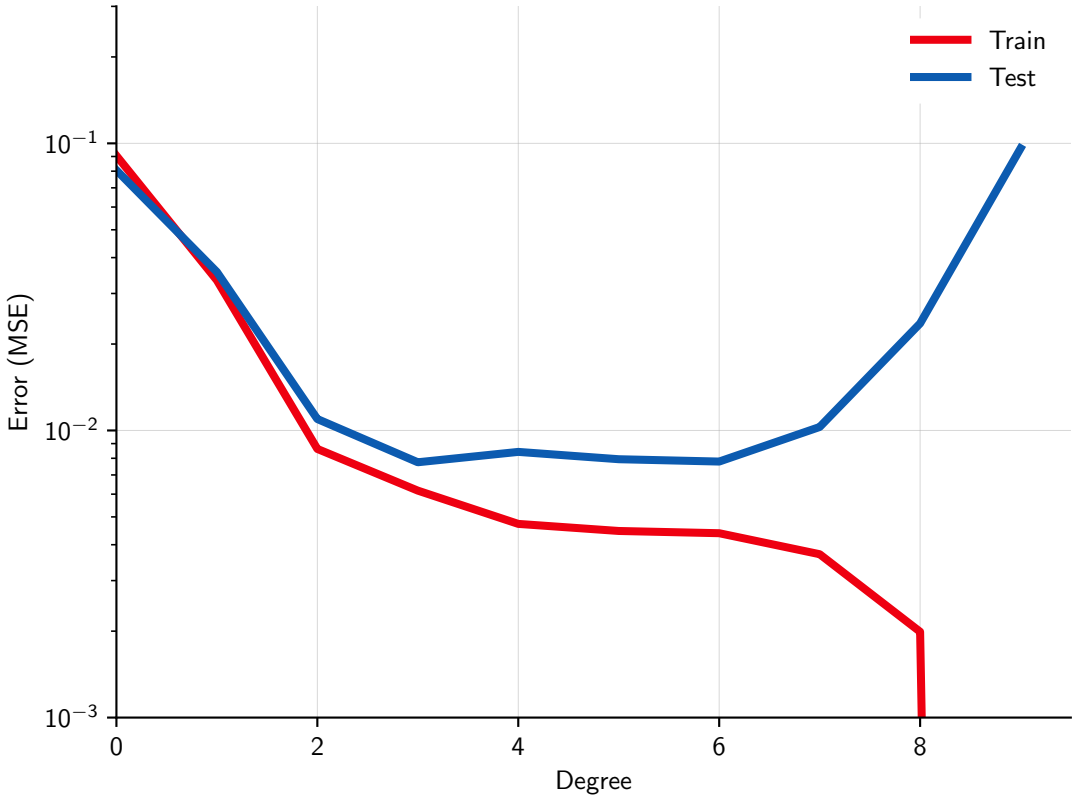
Here, we fit a polynomial of degree 4 to the sin function on $[-\pi, \pi]$. We build a vector `x` with values uniformly spaced in $[-\pi, \pi]$, and `y` as the sin of `x`.
On the plot, the red curve is the sin function and the blue one is the fitted polynomial of degree 4.

We can use this model to illustrate how the prediction changes when we increase the degree or the regularization.



Notes

We try here to approximate the "true functional" depicted as a red curve, which is constant equal to zero on $[0, 0.5]$ and quadratic on $[0.5, 1]$.
The red points are the training examples obtained by taking $x_1, \ldots, x_n$ regularly spaced in $[0, 1]$ and computing the corresponding y_n with an added Gaussian noise.
The graphs show the fitting of polynomials of increasing certain degrees. As expected, the resulting mapping better fits the training points for higher degrees (i.e. the blue curve gets closer to the red dots).
For $D = 4$, the fitting starts going wrong, and an oscillation appears. When the degree continues to increase, we see that the fitting gets closer to the data points, but starts diverging from the true underlying structure of the data. Note that with degree 9, the polynomial has as many coefficients as there are training points, and consequently can perfectly fit the training set.



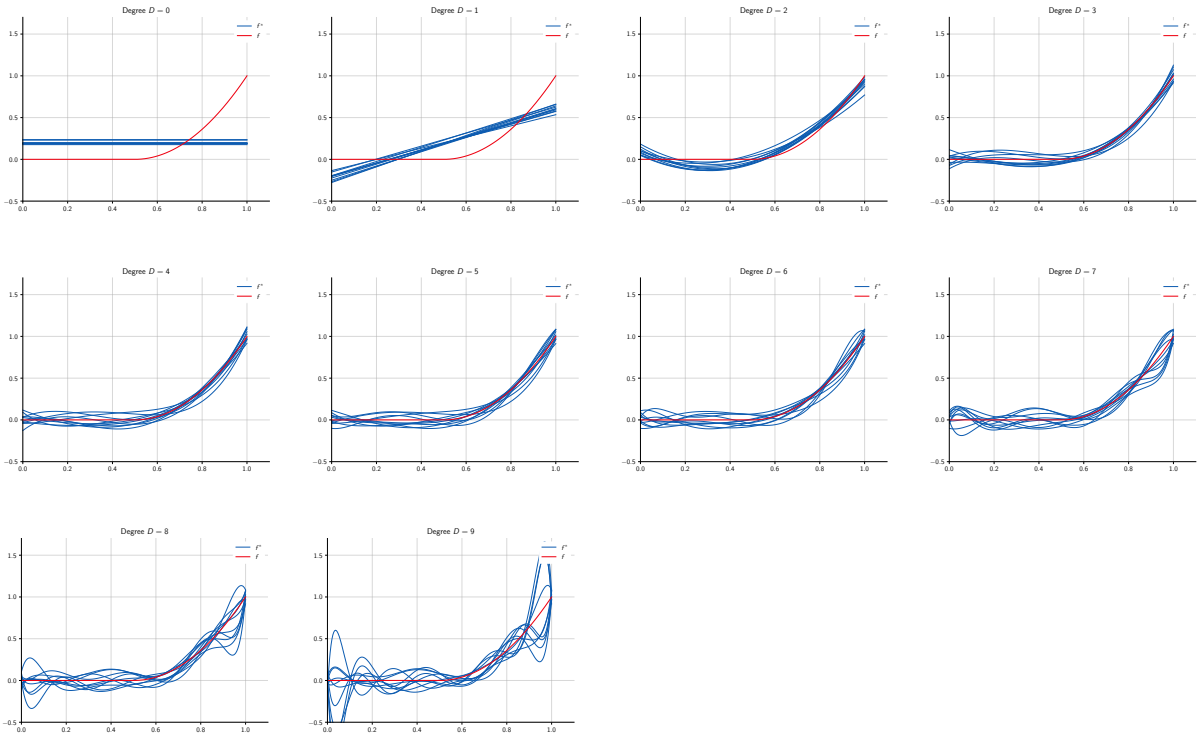
Notes

As for the K-NN, this graph shows the error rate of the prediction estimated on the training examples (aka "train error") in red, and the error rate estimated on another set of samples generated with the same procedure, but that differs due to the randomness of the process (aka "test error") in blue, for different values of D.
When the degree is small, there is no flexibility at all in the model and both training and test errors are high. This corresponds to under-fitting.
As the degree increases, the polynomial better fits the training data, which results in a lower train error. The error reaches 0 for degree 9, when the polynomial passes through all the training points.
On the test set the error decreases until degree 3, and start increasing again, which corresponds to over-fitting

We can visualize the influence of the noise by generating multiple training sets $\mathscr{D}_1, \ldots, \mathscr{D}_M$ with different noise, and training one model on each.

Notes

All the training sets are generated with the same noise parameters, but differs due to the randomness of the sampling process.



Notes

We fit one polynomial on each training set $\mathscr{D}_1, \ldots, \mathscr{D}_M$, and draw them as blue curves.
When the degree is small ($D \leq 3$), the polynomials are concentrated and close to each other. As the degree increases, they tend to diverge from one another. When the degree is large ($D \geq 8$), the flexibility of the models can be seen through the strong variations between the fitted polynomials: different training sets lead to very different predictors.
The variations between all these models is problematic as they serve the should be model of the same underling "true" functional.

We can reformulate this control of the degree with a penalty

$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2 + \sum_d l_d(\alpha_d)$$

where

$$l_d(\alpha) = \begin{cases} 0 & \text{if } d \leq D \text{ or } \alpha = 0 \\ +\infty & \text{otherwise.} \end{cases}$$

Such a penalty kills any term of degree $> D$.

This suggests more subtle variants. For instance, to keep all this quadratic

$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2 + \rho \sum_d \alpha_d^2.$$

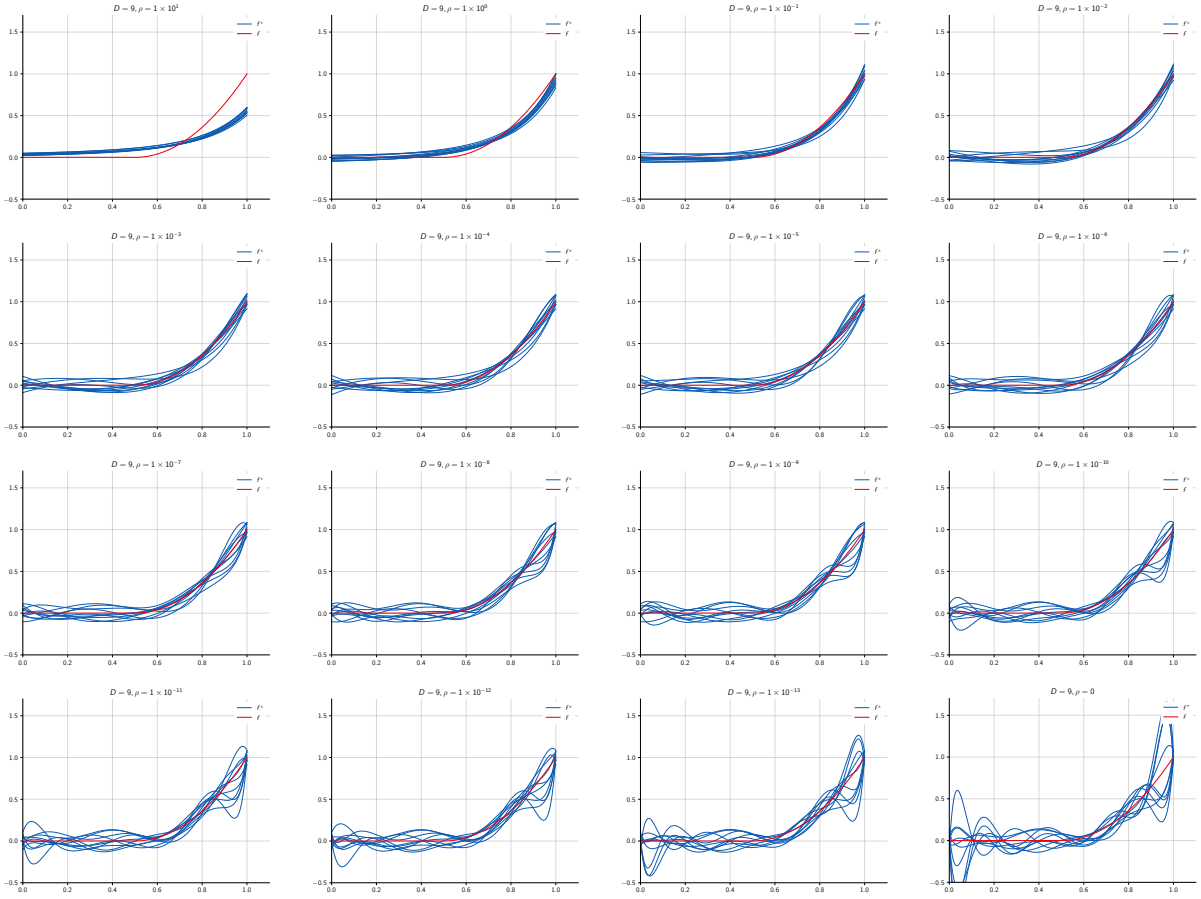
Notes

The control of the degree can be formulated with a penalty over the functional itself, to kill any monomial whose coefficient is greater than D: instead of manually look for a polynomial of degree D, we can incorporate this constrain in the loss itself.
This can also be formulated in a more gentle manner by using a penalty term which is the sum of the squares of the coefficients: instead of preventing any monomial of degree greater than D, it will tend to select polynomials whose parameters are all close to zero.
The resulting least square problem can be formulated as:

$$\sum_n (f(x_n;\alpha) - y_n)^2 + \rho \sum_d \alpha_d^2$$

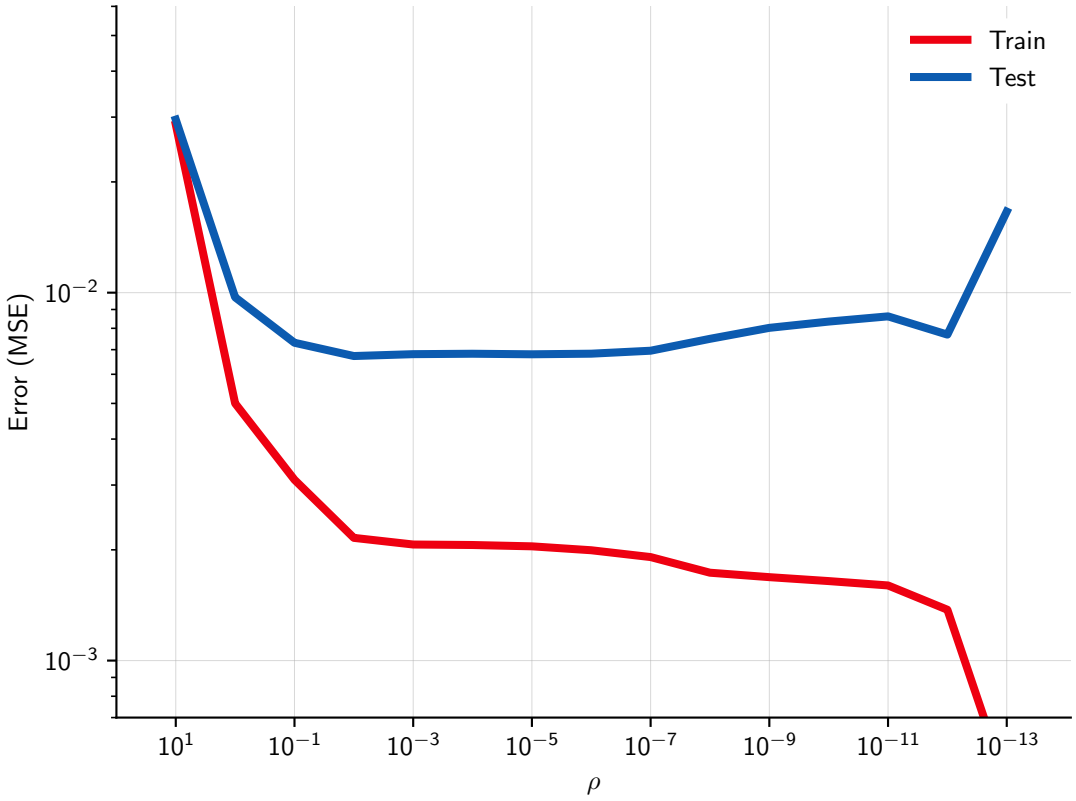
$$= \left\| \begin{bmatrix} x_1^0 & \cdots & x_n^D \\ \vdots & & \vdots \\ x_N^0 & \cdots & x_N^D \\ \sqrt{\rho} & & \\ & \ddots & \\ & & \sqrt{\rho} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{bmatrix} - \begin{bmatrix} y_0 \\ \vdots \\ y_N \\ 0 \\ \vdots \\ 0 \end{bmatrix} \right\|^2$$

and solved as before with `torch.linalg.lstsq`.



Notes

The following plots show the fitted polynomials of degree 9 for decreasing values of ρ, the weight of the regularization penalty.
At first, when ρ is large, high degrees are more penalized, and the polynomials themselves are constrained. When ρ gets smaller, we start observing the same behavior as before with no penalization, which is the situation where monomials with high degrees can have a higher coefficients. The resulted fitted polynomials are less similar because each one models the noise in their respective training data set. When $\rho = 0$, this corresponds to no penalization at all.



Notes

This graph shows the training and the test errors as a function of the regularization parameter ρ.
When ρ is large (left part of the graph), the learning is pushed toward polynomials with small coefficients, so the training data matters less, and both the training and test errors are large: this is under-fitting.
When ρ is small (right part of the graph), the polynomial is able to fit the training data very well, which leads to a low train error, but a high test error: this is over-fitting.

We define the **capacity** of a set of predictors as its ability to model an arbitrary functional. This is a vague definition, difficult to make formal.

A mathematically precise notion is the Vapnik–Chervonenkis dimension of a set of functions, which, in the Binary classification case, is the cardinality of the largest set that can be labeled arbitrarily (Vapnik, 1995).

It is a very powerful concept, but is poorly adapted to neural networks. We will not say more about it in this course.

Notes

In the example of polynomial fitting, a polynomial of large degree has more capacity than a polynomial with a smaller degree. As we saw, with degree 9, the polynomial could fit all the training points. This does not mean that this is a better model.

Although the capacity is hard to define precisely, it is quite clear in practice how to modulate it for a given class of models.

In particular one can control over-fitting either by

- Reducing the space $\mathscr{F}$ (less functionals, constrained or degraded optimization), or
- Making the choice of f^* less dependent on data (penalty on coefficients, margin maximization, ensemble methods).

Notes

Many practical methods allow to reduce the space $\mathscr{F}$.
In deep learning, it can be done by choosing a simpler neural network with fewer modules, or fewer layers.
A criterion used for certain methods pushes the functional away from the training points in a metric sense (e.g. support vector machines).
Ensemble methods consists in training multiple predictors, and at test time, averaging over all the predictions to make the final decision.

References

V. N. Vapnik. The Nature of Statistical Learning Theory. Springer-Verlag, New York, 1995.