10

Overfitting and cross-validation

For some practitioners of machine learning the most interesting aspect is devising new and exciting algorithms and words such as "evaluation" or "testing" is likely to be treated as an afterthought. However, testing and quantifying the performance of machine-learning methods is possibly the most important aspect of data modelling.

Suppose we are in a situation where we has S different models $\mathcal{M}_1, \ldots, \mathcal{M}_S$ that each tries to solve a particular supervised learning problem. Without an objective way of comparing the models we will not know which to choose. Sure, we might *feel* we should select model $\mathcal{M}_S$ which is the most complicated of the models, but that is not an objective justification. Worse yet, if we are working in a company, it will be impossible to quantify if progress is being made at solving the problem or if there is any benefits for the company to have a machine learning department at all.

Seen in this way quantification (and comparison) of model performance is something a machine-learning practitioner should be obsessively preoccupied with. In this chapter, we will discuss common issues with model performance evaluation and provide the industry standard, cross-validation, for estimating the generalization error which allow us to evaluate a given models performance and thereby select between different models. The chapter will finish with a discussion of how the generalization error provides more qualitative information about the goodness of a given model.

10.1 Cross-validation

The principal way of comparing and validating models is by cross-validation. In this chapter, we will introduce the reasoning behind cross-validation and discuss important applications of cross-validation. We will use the simple linear regression model as a running example.

10.1.1 A simple example, linear regression

To provide a concrete example, consider the simple regression problem in fig. 10.1 where the goal is to predict y from x and we have access to 9 data points collected in a training data set $\mathcal{D}^{\text{train}}$. The data consists of noisy observations of the black curve and and we wish to fit a regression model to the data. We assume we have access to three different models

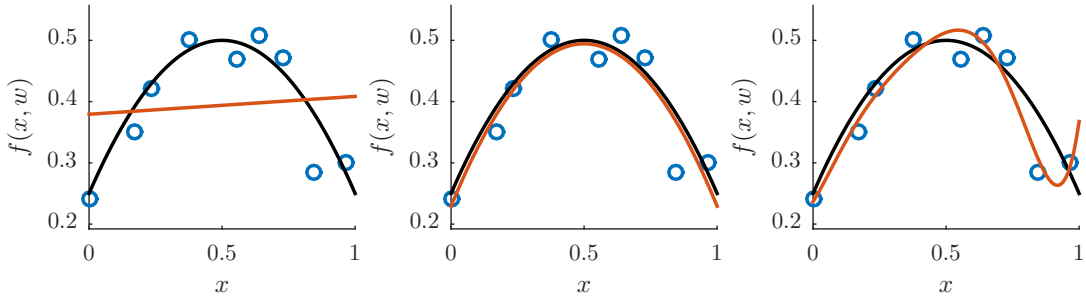

Fig. 10.1. A small dataset of nine observations generated from the true curve shown in the black line. The three red lines are three different linear regression models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ fitted to the dataset. Clearly the most complicated model $\mathcal{M}_3$ fits the dataset best, however, the second model $\mathcal{M}_2$ is better suited to account for the true black curve.

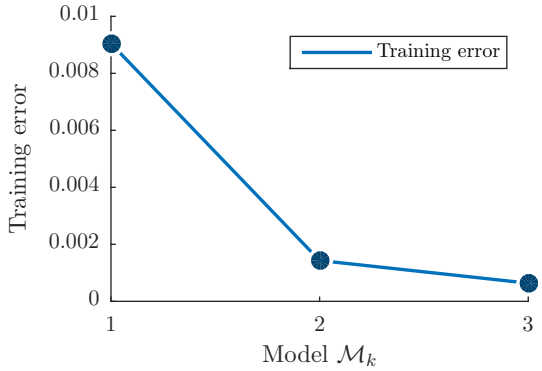

Fig. 10.2. Training error for each of the three models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ computed on the training data set. The most complicated model has the lowest training error.

$\mathcal{M}_1 = \{$1'st order polynomial, i.e. $\}f_{\mathcal{M}_1}(x, \boldsymbol{w}) = w_0 + w_1 x.$

$\mathcal{M}_2 = \{$2'nd order polynomial, i.e. $\}f_{\mathcal{M}_2}(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2.$

$\mathcal{M}_3 = \{$6'th order polynomial, i.e. $\}f_{\mathcal{M}_3}(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + w_6 x^6.$

The red line indicates the fitted polynomials. For each model we quantify how well the model fits the training data by the *training error* which for model $\mathcal{M}_s$ is

$$E_{\mathcal{M}_s}^{\text{train}} = \frac{1}{N^{\text{train}}} \sum_{i \in \mathcal{D}^{\text{train}}} (y_i - f_{\mathcal{M}_s}(x_i, \boldsymbol{w}))^2.$$

Here $f_{\mathcal{M}_s}$ is the model $\mathcal{M}_s$ fitted to the training data and $N^{\text{train}} = |\mathcal{D}^{\text{train}}|$ is the number of observations in the training data set. The training error of each of these three models is shown in fig. 10.2. Notice the "most correct" model, $\mathcal{M}_2$, fits the data better than the model $\mathcal{M}_1$, however, both models fits the data far worse than the complicated model $\mathcal{M}_3$.

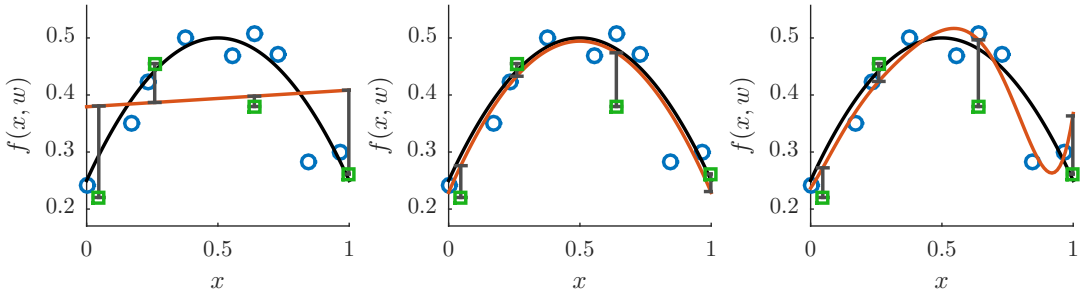

Fig. 10.3. The example from before with a test dataset of three new points. If the models are evaluated in terms of how they predict the *new* points $\mathcal{M}_2$ is preferred.

Nevertheless, we are not interested in a model like $\mathcal{M}_3$ as it clearly will not generalize well to new data. We say model $\mathcal{M}_3$ is *overfitting* the data. Thus, we can't tell the models apart by how well they fit the training data and in fact this can give us an entirely misleading picture of the models performance due to overfitting. This is such an important principle it is worth framing:

> Never, ever should you estimate how well a model performs by its predictions on data it was trained upon.

However, let's assume we obtain access to some new data, the test data $\mathcal{D}^{\text{test}}$, indicated by the green squares in fig. 10.3. Testing the models on this new test-dataset gives us the ability to estimate how well the models *generalize* to new data. We can define the test error as

$$E^{\text{test}}_{\mathcal{M}_s} = \frac{1}{N^{\text{test}}} \sum_{i \in \mathcal{D}^{\text{test}}} (y_i - f_{\mathcal{M}_s}(x_i, \boldsymbol{w}))^2.$$

Notice, $f_{\mathcal{M}_s}$ is the model that was fitted to the *training data*. The test error of each of these three models is shown in fig. 10.4. Notice, the test error allows us to select the correct model, i.e. the model that can be expected to generalize better to new data.

The problem is that as a rule nobody is going to turn up and give us a test dataset when we need it. The basic idea in cross-validation is to overcome this problem by taking our existing fixed data set $\mathcal{D}$ and manually divide it into a training set, $\mathcal{D}^{\text{train}}$, and a testing data set, $\mathcal{D}^{\text{test}}$, and then use these two to select the appropriate model.

10.1.2 The basic setup for cross-validation

The basic setup for cross-validation is as follows: We consider a supervised learning problem with a data set $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$. It is important to keep in mind the dataset is finite and *this is all the data we have*. As in the regression example, we consider different models for solving the problem, $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_S$ and for each of these models we have access to a *loss function* $L(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i)$ which quantifies the error of predicting $\hat{\boldsymbol{y}}_i$ when the true value is $\boldsymbol{y}_i$. In the regression example the loss function was the least square error $L(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|^2$, but in general the loss function will be defined according to the specific modeling purpose.

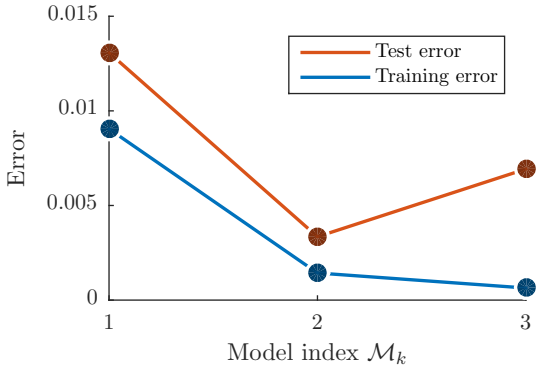

Fig. 10.4. Test error for each of the three models $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ computed on the test data set. The test error correctly singles out model $\mathcal{M}_2$ as the better model.

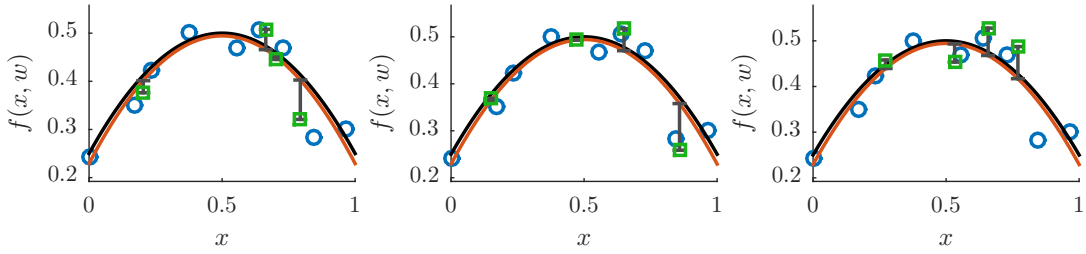

Fig. 10.5. The basic regression problem for model $\mathcal{M}_2$ and three random test sets. Since the test sets are random, the test error too will vary depending on the particulars of the test set. The generalization error overcomes this by averaging over all test sets.

Training and test error

If the data is divided into a training set and a test set $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{test}}$ and the model $\mathcal{M}$ is fitted on the training set to provide the prediction rule $\boldsymbol{f}_{\mathcal{M}}$ then we define the training and test errors as:

$$E_{\mathcal{M}}^{\text{train}} = \frac{1}{N^{\text{train}}} \sum_{i \in \mathcal{D}^{\text{train}}} L(\boldsymbol{y}_i, f_{\mathcal{M}}(\boldsymbol{x}_i)), \tag{10.1}$$

$$E_{\mathcal{M}}^{\text{test}} = \frac{1}{N^{\text{test}}} \sum_{i \in \mathcal{D}^{\text{test}}} L(\boldsymbol{y}_i, f_{\mathcal{M}}(\boldsymbol{x}_i)). \tag{10.2}$$

These definitions are similar except $\boldsymbol{f}_{\mathcal{M}}$ is fitted on the training set in both cases.

Generalization error

A problem with the test error is that it depends on the specific test set. Since we have to construct the test set ourselves, this makes the test error slightly random. This is illustrated in fig. 10.5 for the

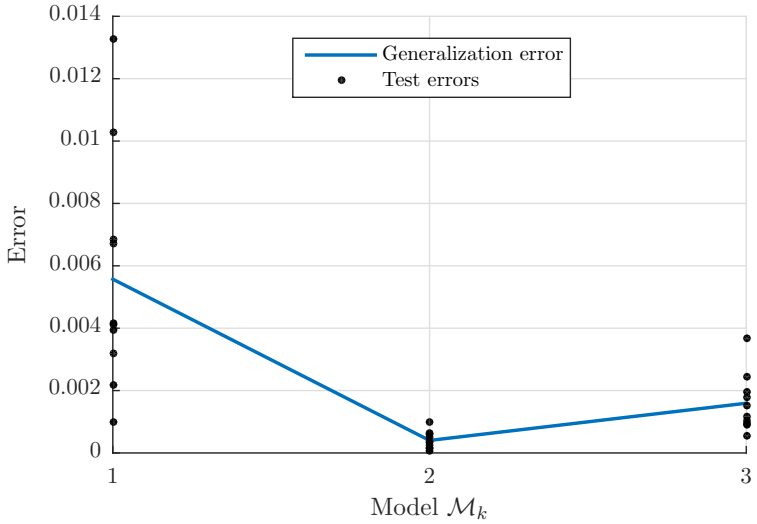

Fig. 10.6. Continuing the example, for the three models different test sets gives different test error as indicated by the black dots. The generalization error is simply the average over all possible test sets according to their probability of occurring.

test error for three different (random) test sets. To alleviate this problem we introduce a new, third error namely the *generalization error*. The generalization error is an idealized quantity indicating how well our model performs on average assuming we had an infinite amount of data to test it on, i.e. the average of the test errors as illustrated in fig. 10.6. *The generalization error is what we truly wish to estimate and the best model is the model with the lowest generalization error.* If we assume the test observations $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ come from a distribution $p(\boldsymbol{x}, \boldsymbol{y})$ then the generalization error is defined as follows:

- Train the model $\mathcal{M}$ on the full dataset available $\mathcal{D}$ to give a prediction rule $\boldsymbol{f}_{\mathcal{M}}$.
- The generalization error of model $\mathcal{M}$ is[1]

$$E_{\mathcal{M}}^{\text{gen}} = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}\left[L(\boldsymbol{y}, \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}))\right] \tag{10.3}$$

$$= \int L(\boldsymbol{y}, \boldsymbol{f}_{\mathcal{M}}(\boldsymbol{x}))p(\boldsymbol{x}, \boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y}. \tag{10.4}$$

The generalization error is the fairest estimate of how well our model can perform because it assumes we train our model on *all* data we have available and then computes the *average* loss on all future data.

[1] Another popular definition is to consider the training set random as well which we will later call the averaged generalization error. This is however notationally more cumbersome and leads to the same definition of the cross-validation algorithms.

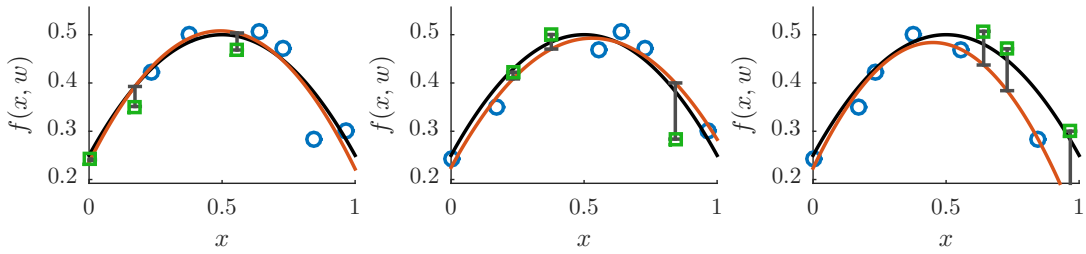

Fig. 10.7. Cross-validation applied to the model $\mathcal{M}_2$ using 3-fold cross-validation. Errors are estimated from the test data points and is indicated by the gray bars. Averaging all errors produce the estimate of the generalization error $\hat{E}^{\text{gen}}_{\mathcal{M}_2}$

10.1.3 Cross-validation for quantifying generalization

The obvious problem with the generalization error is that we cannot compute it since we don't know the true distribution of the data. Cross-validation, is thus a framework to estimate a model's generalization error typically based on one of the following three approaches:

Hold-out method

In the hold-out method, the full dataset $\mathcal{D}$ is split into a train and a testing set

$$\mathcal{D} = \mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{test}}.$$

Then, we train a model on $\mathcal{D}^{\text{train}}$ and compute the test error $E^{\text{test}}_{\mathcal{M}}$ using the test data set $\mathcal{D}^{\text{test}}$ and formula eq. (10.2) and simply use the approximation:

$$E^{\text{gen}}_{\mathcal{M}} \approx E^{\text{test}}_{\mathcal{M}}.$$

Why does this work? The test error is different in two ways from the generalization error. Firstly, we only train the model on a subset of the data $\mathcal{D}^{\text{train}}$ and not the full data set $\mathcal{D}$ and secondly we do not compute the true expectation but only the empirical average based on $\mathcal{D}^{\text{test}}$. However, if the training data set is large, we can expect (or rather, hope!) there will be little difference in using $\mathcal{D}^{\text{train}}$ instead of $\mathcal{D}$ and secondly, if we have a lot of test data in $\mathcal{D}^{\text{test}}$, and each element in the test data set is drawn from the true distribution $p(\boldsymbol{x}, \boldsymbol{y})$, we can expect the empirical average in eq. (10.2) to be quite close to the true average for the generalization error eq. (10.4). By recognizing these limitations we can provide two alternative methods which generally does better but are also computationally more demanding:

K-fold cross-validation

Ideally, we want each data point to be used in the test set, and one way to accomplish this is with K-fold cross-validation. In K-fold cross-validation the full data set is split into K pieces

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_K,$$

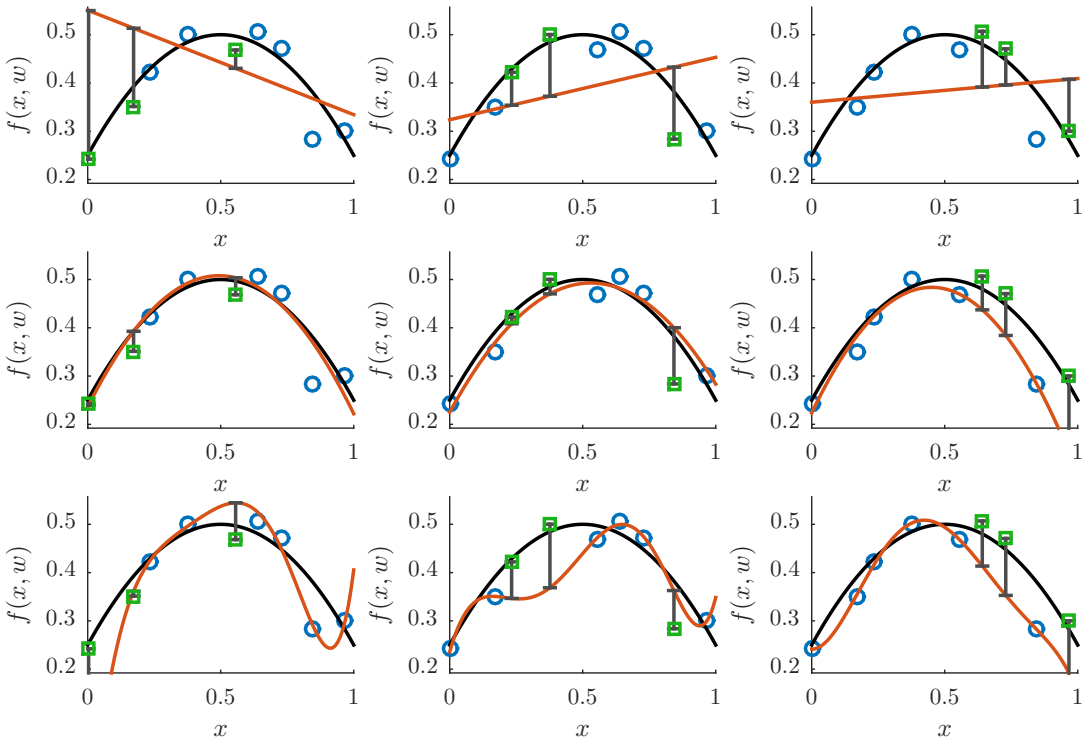

Fig. 10.8. Cross-validation applied to model-selection for the linear regression models. Each row corresponds to one of the three models, and each column to the estimate of the test error on that particular fold.

each containing $\frac{N}{K}$ observations. We then produce K splits into training and test sets by, for each k, treating $\mathcal{D}_k$ as the test set and the other $K-1$ pieces as the training set. Computing the test error on each of these K splits gives K estimates of the error $E^{\text{test}}_{\mathcal{M},1}, \ldots, E^{\text{test}}_{\mathcal{M},K}$ and we now approximate:

$$E^{\text{gen}}_{\mathcal{M}} \approx \sum_{k=1}^{K} \frac{N^{\text{test}}_k}{N} E^{\text{test}}_{\mathcal{M},k},$$

i.e., as the weighted average of the test errors, weighted by the number of test observations in each fold N^{test}_k relative to the total number of observations used for testing N. Since each data point is used once in the test set this method is generally more precise than the hold-out method, however, it requires K times more training and testing of models than the hold-out method.

Leave-one-out cross-validation

The final method is based on the intuition that we ideally want the training set $\mathcal{D}^{\text{train}}$ to be as close to $\mathcal{D}$ as possible. This can be accomplished by using K-fold cross-validation with $K = N$,

the total number of observations in the full data set. In this way, we train N models and each model is trained on the full data set except a single observation and then tested on that single observation. The benefit of this method is that it uses as much data as possible for training such that each trained model is less prone to overfitting than when larger parts of the data are taking out for testing at a time which is especially a concern when having very limited data. However, the drawback is that it requires N models to be trained which can be very wasteful. While leave-one-out cross-validation gives an almost unbiased estimate of the generalization error, in some cases it can have a larger variance compared with, say, 10-fold cross-validation. Overall, it is recommended to use 10-fold cross-validation [Kohavi, 1995].

In the following we will denote by $\hat{E}^{\text{gen}}_{\mathcal{M}}$ an estimate of the generalization error $E^{\text{gen}}_{\mathcal{M}}$ computed by any of the three above techniques. An illustration where 3-fold cross-validation is applied to the linear-regression example is given in fig. 10.7. Each figure corresponds to a fold and in each fold the test error is computed as the average of the error on the three datapoints that are left out. Notice, all nine data-points are part of the test set exactly once.

10.1.4 Cross-validation for model selection

We will accept that the generalization error eq. (10.4) is the optimal way to measure the performance of a model, and that cross-validation using any of the three techniques (hold-out, K-fold or leave-one-out) is a faithful estimate of the generalization error. Then, an obvious way to select between S models $\mathcal{M}_1, \ldots, \mathcal{M}_S$ is to estimate the generalization error of each model using cross-validation and select the model with the lowest cross-validation error. To summarize:

- For each model, compute the estimate of the generalization error $\hat{E}^{\text{gen}}_{\mathcal{M}_1}, \ldots, \hat{E}^{\text{gen}}_{\mathcal{M}_S}$ using cross-validation.
- Select the optimal model $\mathcal{M}_{s^*}$ as that with the lowest error:

$$s^* = \arg\min_s \hat{E}^{\text{gen}}_{\mathcal{M}_s}$$

There is nothing more to cross-validation for model selection than this! This technique is most often used in conjunction with K-fold cross-validation. In this case it is strongly recommended that the same data splits (i.e. choices of $\mathcal{D}_1, \ldots, \mathcal{D}_K$) is used for all models. Since the resulting method is so important it is provided as an explicit algorithm in algorithm 5.

An illustration of 3-fold cross-validation for model selection in the linear-regression example is given in fig. 10.8. Each of the columns corresponds to a fold and each of the rows to a model. In fig. 10.9 the estimated generalization and training errors (averaged over the cross-validation folds) is plotted. As can be seen the training error drops for the more complicated model, however, the cross-validation estimate of the generalization error allows us to select the right model $\mathcal{M}_2$.

10.1.5 Two-layer cross-validation

Let's turn to the following situation: We wish to select the optimal model $\mathcal{M}_{s^*}$ out of S models *and* estimate the generalization error for this optimal model $\mathcal{M}_{s^*}$. A tempting way to accomplish this is to apply K-fold cross-validation to estimate the generalization error for each model $\mathcal{M}_s$ using cross-validation and select the model with the lowest generalization error and then use the estimate of the generalization error as an estimate of how well the model performs. This is illustrated in

Algorithm 5: K-fold cross-validation for model selection

Require: K, the number of folds in the cross-validation loop
Require: $\mathcal{M}_1, \ldots, \mathcal{M}_S$. The S different models to select between
Ensure: $\mathcal{M}_{s^*}$ the optimal model suggested by cross-validation
  for $k = 1, \ldots, K$ splits **do**
    Let $\mathcal{D}_k^{\text{train}}$, $\mathcal{D}_k^{\text{test}}$ the k'th split of $\mathcal{D}$
    for $s = 1, \ldots, S$ models **do**
      Train model $\mathcal{M}_s$ on the data $\mathcal{D}_k^{\text{train}}$
      Let $E_{\mathcal{M}_s,k}^{\text{test}}$ be the *test error* of the model $\mathcal{M}_s$ when it is *tested* on $\mathcal{D}_k^{\text{test}}$
    end for
  end for
  For each s compute: $\hat{E}_{\mathcal{M}_s}^{\text{gen}} = \sum_{k=1}^{K} \frac{N_k^{\text{test}}}{N} E_{\mathcal{M}_s,k}^{\text{test}}$
  Select the optimal model: $s^* = \arg\min_s \hat{E}_{\mathcal{M}_s}^{\text{gen}}$
  $\mathcal{M}_{s^*}$ is now the optimal model suggested by cross-validation

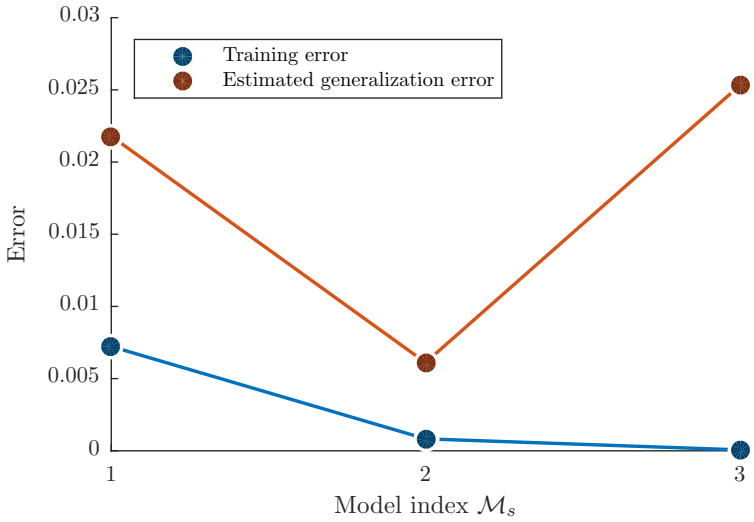

Fig. 10.9. Training error and the cross-validation estimate of the generalization error. The estimate of the generalization error is simply the averages of the errors in fig. 10.8 over all 3 folds as dictated by the cross-validation method for model selection. The model with the lowest estimated generalization error is $\mathcal{M}_2$ even though it does not have the lowest training error.

fig. 10.10 where we consider 14 different models and for each model the true generalization error is indicated as the black line and the estimated generalization error as the small red dots. The selected model is the model with the lowest (estimated) generalization error indicated with the red circle. The estimates of the generalization error is imprecise due to the randomness in the test set which is why they are not all on the black line.

There is however a problem with this approach. Suppose we had access to additional test sets and use these to estimate the generalization error (indicated in the 3 other panels of fig. 10.10). These too are estimates of the true generalization error (black line), but they are independent of

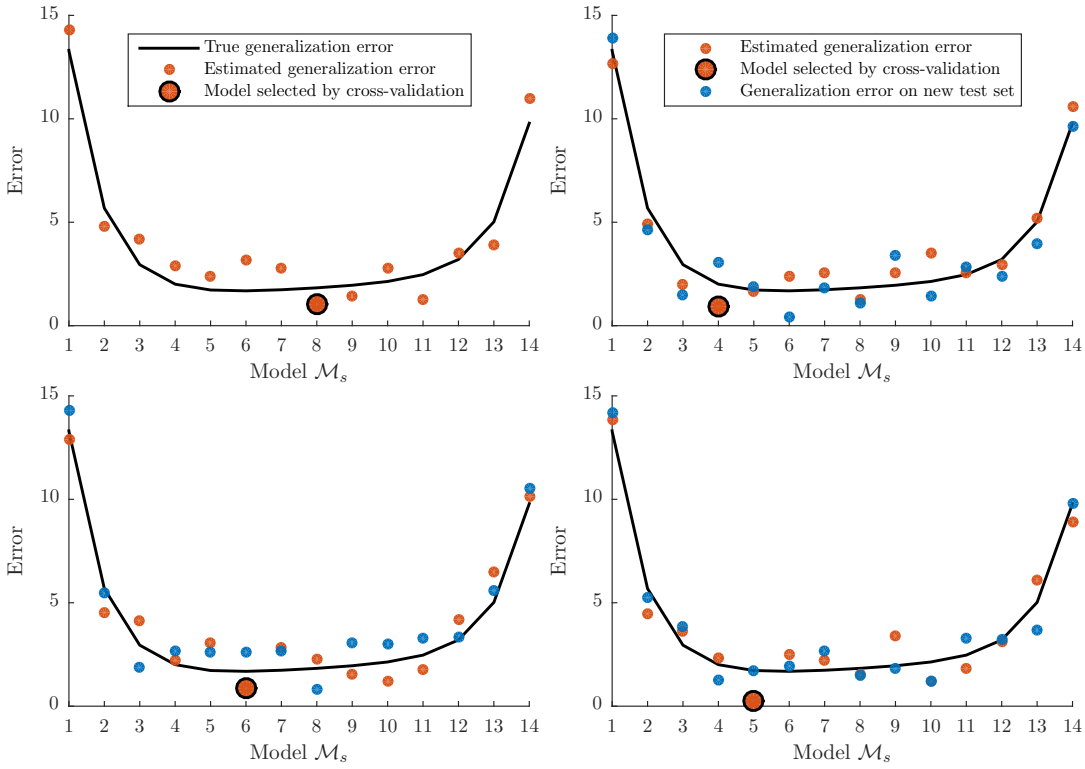

Fig. 10.10. Top right: The true generalization error of 14 models is indicated by the black line and estimates of the generalization error (computed using cross-validation) are indicated by red dots. Standard cross-validation then selects the model with the lowest (estimated) cross-validation error (indicated by red circle). However, this *estimate* of the generalization error is not in general a fair estimate of how the model will generalize to future data because it is selected as a minimum. In subplots 2-4 is shown the same procedure and as seen the estimated generalization error is too optimistic (below the black line) in all instances. A better estimate can be obtained by using a completely new test set, blue dots, which provides a fairer estimate of the generalization error for the selected values. This leads to two-layer cross-validation.

the red dots. However, since we always select the red dot with the *lowest* error, and the blue dots are *random*, we will in general be too optimistic with respect to our estimate of the generalization error. After all, there are many roughly equally good models to choose from, so when we select the *best* of these we will due to the randomness often do exceedingly well. In the figure, this is seen as the selected red point being far lower than the true generalization error in all instances. Obviously, this is cheating! To understand exactly what goes wrong we need to take a step back. By including the step where we select the optimal model $\mathcal{M}^* = M_{s^*}$ based on the data we have actually changed the underlying model being tested. The model the above method produces, $\mathcal{M}^*$, is now composed of two things:

- Use K_2-fold cross-validation to estimates $\hat{E}_s^{\text{gen}}$.
- Select $\mathcal{M}^*$ as the optimal model $\mathcal{M}_{s^*}$ where $s^* = \arg\min_s \hat{E}_s^{\text{gen}}$.

Algorithm 6: Two-level cross-validation

Require: K_1, K_2, folds in outer, and inner cross-validation loop respectively
Require: $\mathcal{M}_1, \ldots, \mathcal{M}_S$: The S different models to cross-validate
Ensure: $\hat{E}^{\text{gen}}$, the estimate of the generalization error
for $i = 1, \ldots, K_1$ **do**
Outer cross-validation loop. First make the outer split into K_1 folds
Let $\mathcal{D}_i^{\text{par}}$, $\mathcal{D}_i^{\text{test}}$ be the i'th split of $\mathcal{D}$
for $j = 1, \ldots, K_2$ **do**
Inner cross-validation loop. Use cross-validation to select optimal model
Let $\mathcal{D}_j^{\text{train}}$, $\mathcal{D}_j^{\text{val}}$ be the j'th split of $\mathcal{D}_i^{\text{par}}$
for $s = 1, \ldots, S$ **do**
Train $\mathcal{M}_s$ on $\mathcal{D}_j^{\text{train}}$
Let $E_{\mathcal{M}_s,j}^{\text{val}}$ be the *validation error* of the model $\mathcal{M}_s$ when it is *tested* on $\mathcal{D}_j^{\text{val}}$
end for
end for
For each s compute: $\hat{E}_s^{\text{gen}} = \sum_{j=1}^{K_2} \frac{|\mathcal{D}_j^{\text{val}}|}{|\mathcal{D}_i^{\text{par}}|} E_{\mathcal{M}_s,j}^{\text{val}}$
Select the optimal model $\mathcal{M}^* = \mathcal{M}_{s^*}$ where $s^* = \arg\min_s \hat{E}_s^{\text{gen}}$
Train $\mathcal{M}^*$ on $\mathcal{D}_i^{\text{par}}$
Let E_i^{test} be the *test error* of the model $\mathcal{M}^*$ when it is *tested* on $\mathcal{D}_i^{\text{test}}$
end for
Compute the estimate of the generalization error: $\hat{E}^{\text{gen}} = \sum_{i=1}^{K_1} \frac{|\mathcal{D}_i^{\text{test}}|}{N} E_i^{\text{test}}$

Thus, *estimating the generalization error requires estimating the generalization error of the model obtained through this two-step procedure*. Fortunately, we know how to estimate the generalization error of a model: Cross-validation. Since the method now makes use of two nested cross-validation procedures, one in selecting $\mathcal{M}_{s^*}$ as above and one for estimating performance, the resulting procedure is known as two-layer cross-validation. The method can be sketched as follows:

- For $i = 1, \ldots, K_1$ cross-validation iterations, split the data $\mathcal{D}$ into a training set $\mathcal{D}_i^{\text{par}}$ and a test set $\mathcal{D}_i^{\text{test}}$
- For each iteration, find the optimal value s^* using K_2-fold cross-validation on $\mathcal{D}_i^{\text{par}}$. (In the j^{th} inner fold $\mathcal{D}_i^{\text{par}}$ is split into a training set $\mathcal{D}_j^{\text{train}}$ and a test set (called a validation set) $\mathcal{D}_j^{\text{val}}$).
- Train the model $\mathcal{M}^*$ using the selected model structure $\mathcal{M}_{s^*}$ trained on the full outer fold training set $\mathcal{D}_i^{\text{par}}$
- Let $E_{\mathcal{M}^*,i}^{\text{test}}$ be the test error of $\mathcal{M}^*$ computed on the i'th test set $\mathcal{D}_i^{\text{test}}$
- Estimate the generalization error as $\hat{E}^{\text{gen}} = \sum_{i=1}^{K_1} \frac{|\mathcal{D}_i^{\text{test}}|}{N} E_{\mathcal{M}^*,i}^{\text{test}}$

Again, since this method is so important it is worth providing it in pseudo code as algorithm 6.

10.2 Sequential feature selection

Consider a dataset where observations $\boldsymbol{x}_i$ correspond to patients and we wish to predict a patient's survival time after an operation y_i using linear regression. Suppose for each patient we observe three attributes namely: x_1: *Age*, x_2: *The room number of the patient* and x_3: *Length of hospital stay*. Clearly, only the first and last attribute is relevant to our purpose, so rather than considering

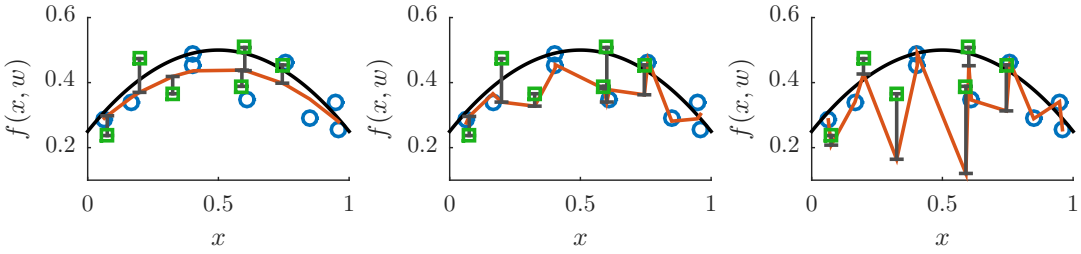

Fig. 10.11. A regularized linear regression model is fitted to the dataset of nine observations and six test observations. The different plots correspond to adding more "junk" attributes, i.e. attributes where the values are just random. As more random attributes are added, the model better fit the training set but does worse on the test set.

the attribute $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ we could just as well consider the smaller dataset: $\boldsymbol{x} = \begin{bmatrix} x_1 & x_3 \end{bmatrix}^T$. So does it matter that we include the room number x_2 in our dataset? Well in general irrelevant attributes matter for three reasons:

- If the number of attributes (in particular the irrelevant ones) is large compared to the total number of observations our model performance will degrade.
- Storing and manipulating irrelevant attributes takes space and makes our models slower.
- A hospital will often wish to know which attributes are important and which are irrelevant. A model with many irrelevant attributes will not tell them that directly.

Let's examine the first claim first. Suppose we have the simple, linear regression problem which can be fitted well with a second-order polynomial. That is, optimally we should consider:

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_1^2 \end{bmatrix} \tag{10.5}$$

However, we now add "junk" attributes to the dataset and considers

$$\boldsymbol{x} = \begin{bmatrix} x_1 & x_1^2 & x_3 & x_4 & \dots & x_{S+2} \end{bmatrix}$$

where S is the number of junk attributes added to the dataset. Thus $S = 0$ will correspond to eq. (10.5) and $S = 3$ will correspond to adding 3 junk attributes. The junk attributes are simply generated as random numbers in the unit interval.

Examples of the predictions on training and test set for $S = 0, 3, 6$ added junk attributes can be seen in fig. 10.11. As can be seen, when more junk attributes are added, the model will begin to overfit the training set. This is easily seen when plotting the training error against the test error as is done in fig. 10.12 for $S = 0, \dots, 7$ and the three specific values shown in fig. 10.11 are indicated by the circles. In a way, we already know how to solve this problem: Each selection of which features to use corresponds to a particular model, so in for instance the hospital example we can consider all eight possible models

$$\begin{array}{cccc} \mathcal{M}_{123} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} & \mathcal{M}_{12} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} & \mathcal{M}_{13} = \begin{bmatrix} x_1 & x_3 \end{bmatrix} & \mathcal{M}_{23} = \begin{bmatrix} x_2 & x_3 \end{bmatrix} \\ \mathcal{M}_1 = \begin{bmatrix} x_1 \end{bmatrix} & \mathcal{M}_2 = \begin{bmatrix} x_2 \end{bmatrix} & \mathcal{M}_3 = \begin{bmatrix} x_3 \end{bmatrix} & \mathcal{M}_\cdot = \begin{bmatrix} \bullet \end{bmatrix}, \end{array}$$

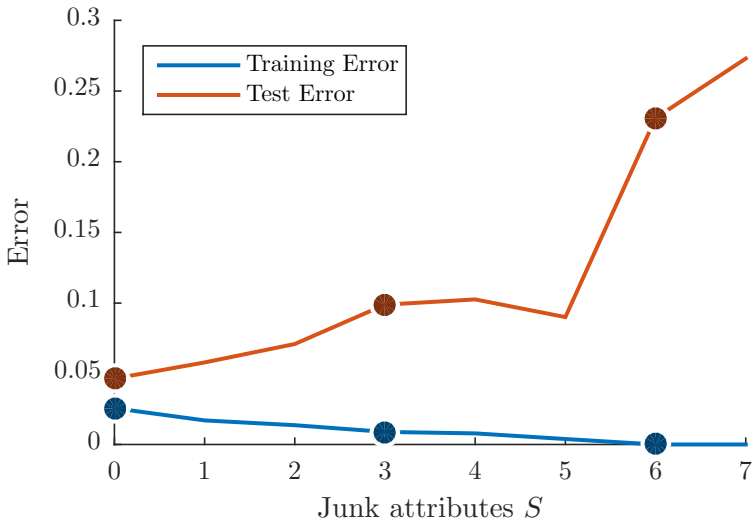

Fig. 10.12. The training and test error for different number of junk attributes in the example of fig. 10.11. The circles indicate the particular values shown in fig. 10.11.

and select the optimal model by the use of cross-validation for model selection as already described. In many ways this is the *best* we can do from a theoretical perspective, however, the problem is that this procedure quickly becomes very costly. In general, if we have M attributes to choose between, we must select between 2^M models, thus if $M = 20$ then this results in having to cross-validate more than a million different models. Clearly this won't do!

Sequential feature selection overcomes this problem by not considering *all* possible models but only a subset. Sequential feature selecting comes in two variation, forward and backward selection, but they are very similar.

10.2.1 Forward Selection

In *forward selection*, we first consider a model with no features

$$\mathcal{M}_\bullet = \left[\bullet\right]$$

That is, it predicts y_i as just being constant. Then it considers the models obtained by adding each attribute to the existing (empty) set of selected attributes thereby testing all the models:

$$\mathcal{M}_\bullet = \left[\bullet\right], \quad \mathcal{M}_1 = \left[x_1\right], \quad \mathcal{M}_2 = \left[x_2\right], \ldots, \mathcal{M}_K = \left[x_M\right].$$

Each of these $M + 1$ models are evaluated by cross-validation for model selection and the optimal model, say $\mathcal{M}_i$, is selected. If $M_\bullet$ is selected the process terminates. Else, this procedure is now repeated by evaluating the M models corresponding to

$$\mathcal{M}_i = \left[x_i\right], \quad \mathcal{M}_{1i} = \left[x_1\ x_i\right], \ldots \mathcal{M}_{i-1,i} = \left[x_{i-1}\ x_i\right] \ldots \mathcal{M}_{iM} = \left[x_i\ x_M\right]$$

Model	$\hat{E}^{\text{gen}}$
$\mathcal{M}_\bullet$	0.91
$\mathcal{M}_1$	0.86
$\mathcal{M}_2$	0.92
$\mathcal{M}_3$	0.88
$\mathcal{M}_4$	0.83
$\mathcal{M}_{12}$	0.78
$\mathcal{M}_{13}$	0.62
$\mathcal{M}_{14}$	0.78
$\mathcal{M}_{23}$	0.74
$\mathcal{M}_{24}$	0.72
$\mathcal{M}_{34}$	0.76
$\mathcal{M}_{123}$	0.64
$\mathcal{M}_{124}$	0.68
$\mathcal{M}_{134}$	0.73
$\mathcal{M}_{234}$	0.78
$\mathcal{M}_{1234}$	0.79

Table 10.1. The *estimated generalization error* $\hat{E}^{\text{gen}}$ as estimated by cross-validation for models trained on different subsets of the features x_1, x_2, x_3 and x_4.

Again, if $\mathcal{M}_i$ is the optimal model the process terminates, else an optimal model (say model $\mathcal{M}_{ij}$) is selected and then all $M-1$ models corresponding to $\mathcal{M}_{ij}$ and the $M-2$ models obtained by adding all other attributes than x_i, x_j to the set evaluated by cross-validation. If it is found that for instance $\mathcal{M}_{ij}$ is the optimal model, the process terminates, else it continues possibly terminating with the full model: $\mathcal{M}_{12\ldots M}$.

Example of forward selection

Let's illustrate this procedure with a concrete example. Suppose we have a dataset of $M = 4$ attributes giving 16 possible models with generalization errors (as estimated by cross-validation) shown in table 10.1. Forward selection now proceeds as follows

- Start with model $\mathcal{M}_\bullet$ with an error of 0.91.
- Compare models $\mathcal{M}_\bullet$ and $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$.
- Optimal model is $\mathcal{M}_4$ with error of 0.83.
- Compare models $\mathcal{M}_4$ and $\mathcal{M}_{14}, \mathcal{M}_{24}, \mathcal{M}_{34}$.
- Optimal model is $\mathcal{M}_{24}$ with error of 0.72.
- Compare models $\mathcal{M}_{24}$ and $\mathcal{M}_{124}, \mathcal{M}_{234}$.
- Optimal model is $\mathcal{M}_{124}$ with error of 0.68.
- Compare models $\mathcal{M}_{124}$ and $\mathcal{M}_{1234}$.
- Since $\mathcal{M}_{124}$ has lowest error, forward selection terminates and select features $1, 2, 4$.

Notice the procedure is completely mechanical, however, it is not guaranteed to select the model with the *lowest* overall generalization error. The benefit of forward selection is naturally that we don't have to compute all the generalization errors beforehand but can compute them as they are required.