# *Lecture 3: Empirical Methods for Model Selection*

*Iain Styles*

*18 October 2019*

## *Selecting and Evaluating Models*

So far, we have seen how, given a set of data and a model (in our case, a linear sum of polynomials), we can compute the model coefficients such that the model matches the data. We have also seen how the choice of model can matter, and that models will typically have a low bias (fit the data well) or a low variance (robust to different data), but not both. For most problems we will have to choose which model is most appropriate, requiring us to formally compare and evaluate different models. How can we do this?

When we perform a regression task there are typically two cases of particular interest:

1. The underlying model is known and we wish to find its parameters.

2. The underlying model is unknown and we wish to learn to make predictions on unseen data.

An example of the first situation is the trajectory of a moving object that is known to be moving according to the well-known equation $s = s_0 + ut + \frac{1}{2}at^2$. If we acquire a dataset $\mathcal{D} = \{(t_i, s_i)\}_{i=1}^N$ of measurements of position $s$ at time $t$ and then fit a quadratic model $y(t) = w_0 + w_1 t + w_2 t^2$ then we can deduce the initial position $s_0 = w_0$, the initial velocity $u = w_1$, and the acceleration $a = w_2$. In this case we use our prior knowledge of the problem to select the appropriate model. Notice that in this case we do not risk overfitting because the model is chosen to model the underlying trend and does not have the capacity to model noise in the measurements.

The second situation is much more interesting and difficult. When we have no first-principles way of determining what the model should be, we have to turn to empirical means, by which we mean "experiments".

In the small experiments we have done so far, we have treated our dataset as a single object and looked to see how well out model can describe it. We have also argued, on intuitive grounds, that when the model is too complex, it can overfit the data. We will now investigate this claim. There are two situation in which we will be interested:

- Cases where we have a large quantity of data

- Cases where we have a small quantity of data

*The Large Data Regime: Train–Validate–Test Split*

Consider a situation in which we have a large dataset $\mathcal{D}$ that is far larger than is needed to estimate the underlying model. The basic idea behind a *holdout* scheme is that we use some of the data to train the model, some of the data to evaluate whether that model is any good, and some of the data to test what we think is the best model. Let us define a partitioning of the data as follows:

- A training set $\mathcal{T}$, randomly sampled from $\mathcal{D}$.
- An validation set $\mathcal{V}$, randomly sampled from $\mathcal{D} - \mathcal{T}$.
- A test, or evaluation set $\mathcal{E} = \mathcal{D} - \mathcal{T} - \mathcal{V}$.

Our starting position is that we have a set of models $\{\mathcal{M}_i\}_{k=1}^{K}$ that we wish to evaluate (using a loss function $\mathcal{L}$) to determine which has the most predictive ability. We need to distinguish between two distinct aspects of our models: their learnable parameters $\mathbf{w}$, and their *hyperparameters*, which are those features of the model that we do not learn directly from the data: the choice of basis, the regularisation scheme etc. The basic idea is then that we use the training set to learn the model parameters, and then we use the validation set to select the choice of hyperparameters that best allows the model learned on $\mathcal{T}$ to generalise to $\mathcal{V}$, in other words, for *model selection*. Finally, we perform an evaluation of the best model on $\mathcal{E}$ to assess how well the model performs on unseen data, which is the ultimate test of its ability to generalise, and allows us to guard against overfitting of the hyperparameters to the validation set. This process is described by Algorithm 1.

**Data**: Set of models $\{\mathcal{M}_i\}_i$
**Data**: Dataset $\mathcal{D}$ split into training ($\mathcal{T}$), validation ($\mathcal{V}$), and test/evaluation ($\mathcal{E}$) sets.
**Result**: Identification of model $\mathcal{M}*$ with best predictive power.
**for** *each model* $\mathcal{M}_i$ **do**
  Train $\mathcal{M}_i$ on $\mathcal{T}$;
  Compute model loss $\mathcal{L}_{\mathcal{T}}$ on training set $\mathcal{T}$;
  Compute model loss $\mathcal{L}_{\mathcal{V}}$ on evaluation set $\mathcal{V}$;
**end**
Select model $\mathcal{M}*$ with best overall performance on training and validation sets.;
Compute loss on test set $\mathcal{E}$ to determine final model performance;

**Algorithm 1**: Model Selection and Evalulation using a Train–Evaluate–Test Procedure.

There are some important features of this scheme that are worth some consideration. Firstly, the dataset $\mathcal{D}$ must be large enough to enable the split to be performed whilst ensuring that the training set $\mathcal{T}$ is large enough to enable the model to learn its structure. Secondly, the dataset partitioning must be carefully performed to

make sure that each of the sub-groups ($\mathcal{T}$, $\mathcal{V}$, $\mathcal{E}$) is representative of the full dataset. For example, in our toy problem of data generated by $y = \sin(2\pi x) + \epsilon$, the partioning needs to be done such that each subset contains sample from across the data's domain $x$. This is particularly important to bear in mind when working with data that has a natural order to it. Similarly, we would also need to be careful to make sure that the full range of $y$ values is represented and the data subsets are not just sampling (for example) the positive values of $y$. This is very difficult to do when working with small datasets and so an alternative schemes is necessary.

Let us implement this on our example. We generate twenty data points and split them into a training set $\mathcal{T}$ of ten points, a validation set $\mathcal{V}$ of five points, and a test set $\mathcal{E}$ of 5 points. Although these are quite small sets, for this simple example, this is sufficient.

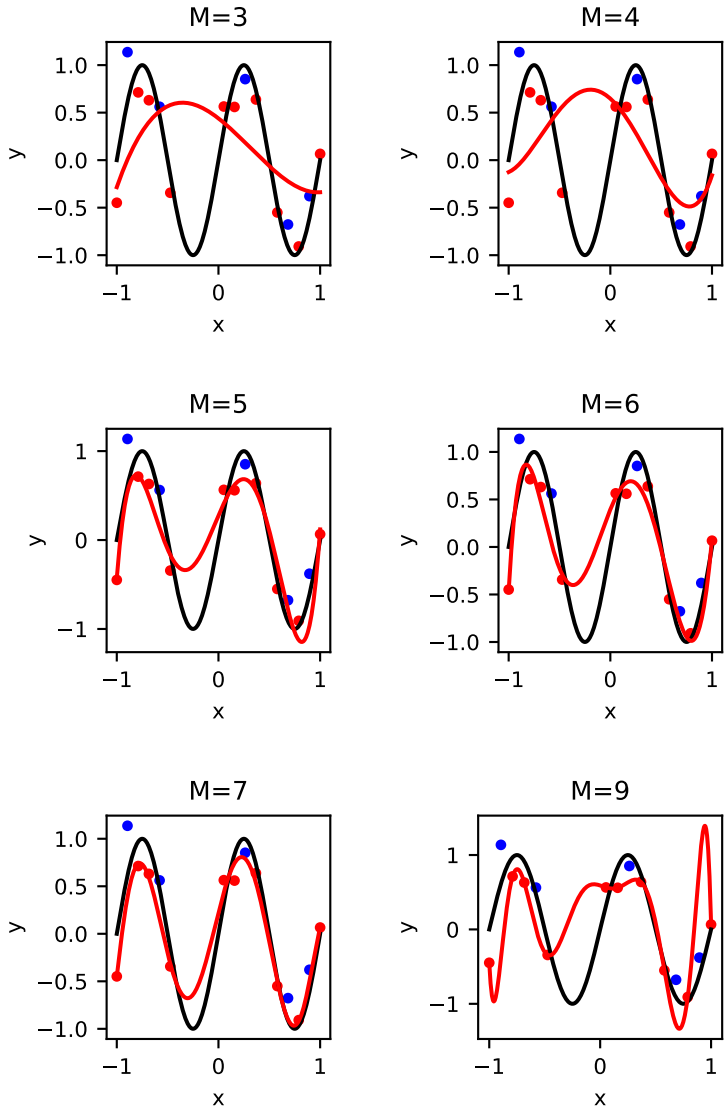


Figure 1: Evaluation of polynomial regression to $y = \sin(2\pi x)$ using a train-validate-test split of (10, 5, 5) samples. Training data is shown as red dots, validation data as blue dots. The true trend is shown as a solid black line. The polynomial fit is shown as a red line.

Visually, we see from the fits, that, as we saw before, the low-

order polynomials are not able to explain the data very well. At degree $M = 5$ we see that this changes visually. With reference to the error shown in Figure 2, we see that the training error decreases significant at this point, and the validation error also shows a significant decrease. It is interesting to see that the validation error tracks the training error quite closely for low-degree polynomials. This is to be expected of a model where the training and validation datasets are representative of the full dataset. If the validation error does not track the training error, this is an indication that at least one of the two sets is not representative of the full dataset.

At very high order fits, (remembering that we are training on ten points), we see that whislt the training error continues to improve, the validation error suddenly gets dramatically worse. This is the characteristic sign of overfitting: the model is able to fit the training data very well, but has learned both the trend in the data and the noise distribution of those datapoints: it has effectively memorised the training data and is unable to generalise its results to the validation set. On this data, fits of order $M = 5$ or $M = 6$ seem like they ought to be optimal, and a useful maxim to apply here is *Occam's Razor* which roughly states that one should choose the simplest hypothesis (model) that is supported by the data, i.e we should choose $M = 5$. On the test set, this gives an RMS error of 0.68 which is comparable with the errors on the validation set and suggests that we have learned the underlying trend in the data reasonably well.

For the data we have considered here, a full training-validation-testing split seems to work, and is consistent with our intuitions. However, this is a very small dataset and an alternative technique – cross-validation – would be more appropriate.

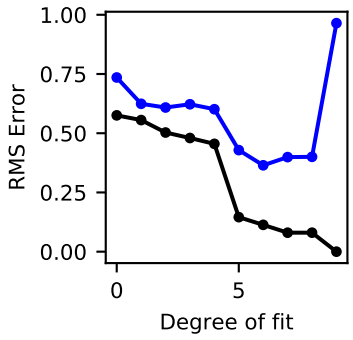


Figure 2: Training and validation errors as a function of polynomial degree for the data shown in Figure 1.

### *The Small Data Regime: Cross-Validation*

If we only have a small number of data points (ten, in our toy problem), it is clear that a three-way split of the training data is problematic. Any significant reduction in the number of data points used to train the model will inevitably lead to difficulties: loss of the trend in the data, overfitting, loss of generalisation etc. In this circumstance, a more data-efficient method is necessary: *cross-validation*.

In a cross-validation scheme, the dataset is typically divided into a test/evaluation set $\mathcal{E}$, and a cross-validation set $\mathcal{C}$. The test set serves the same role as it did before: a fraction of the data is held out in order to evaluate the best model on unseen data. The cross-validation set is used somewhat differently. It is used to both *train* and *validate* the model and therefore to both learn the model parameters and the hyperparameters; that is, both training and validation data is drawn from $\mathcal{C}$. The way that this is done is by splitting $\mathcal{C}$ into $K$ *folds*. $K - 1$ of the folds are used to train the data, and then the remaining fold is used as the validation set. This process is repeated $K$ times with each fold being used in turn as the

**Data**: Set of models $\{\mathcal{M}_i\}_i$
**Data**: Dataset $\mathcal{D}$ split into cross-validation ($\mathcal{V}$), and test/evaluation ($\mathcal{E}$) sets.
**Data**: Number of folds, $K$
**Result**: Identification of model $\mathcal{M}*$ with best predictive power.
Divide $\mathcal{C}$ into $K$ folds $\{c_k\}_{k=1}^{K}$ such that $\mathcal{C} = \bigcup_{k=1}^{K} c_k$;
**for** *each model* $\mathcal{M}_i$ **do**
  **for** $k = 1 \rightarrow K$ **do**
    Train $\mathcal{M}_i$ on training set $\mathcal{C} - c_k$;
    Compute model loss $\mathcal{L}_\mathcal{T}$ on training set $\mathcal{C} - c_k$;
    Compute model loss $\mathcal{L}_\mathcal{V}$ on evaluation fold $c_k$;
  **end**
**end**
Select model $\mathcal{M}*$ with best overall performance on training and validation sets;
Compute loss on test set $\mathcal{E}$ to determine final model performance;

**Algorithm 2**: Model Selection and Evaluation using Cross-Validation.

validation set. The best performing model over the $K$ repeats, typically judged on the average performance over all of the split, is selected as the final model which can then be evaluated against the test set. Common approaches to cross validation are 10-fold cross-validation (train on 90%, validate on 10%), and hold-one-out, which is equivalent to $N$-fold cross validation, for $N$ data points (train on $N - 1$, validate on 1). This is a common strategy when the data is very small indeed.

Let's look at how to do this. Using the same set of data as we studied in the previous example. The process runs as follows:

1. Divide the twenty data points into five "folds" of four data points each – five-fold cross-validation, taking care to randomise the order of the points before doing this.
2. For each model (in this case, polynomial order):
   (a) For each fold:
      i. Validation set ← current fold
      ii. Training set ← all other folds combined
      iii. Learn model weight from training set
      iv. Evaluate model on validation set
   (b) Compute mean training and validation errors across the folds

The

The results should not be compare directly to the full training-validation-test split. We note the following points.

- The training set sizes used in cross-validation can be larger that in a single split

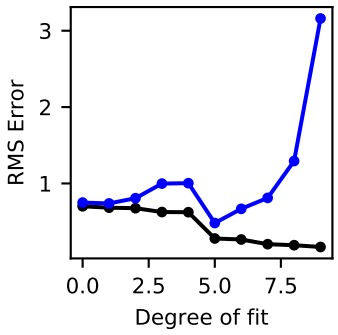


Figure 3: Training and validation errors as a function of polynomial degree for the data shown in Figure 1 using five-fold cross-validation.

- The validation set is typically slightly smaller
- These results are averaged over all folds and this will remove some fluctuations
- Similar conclusions can be drawn: models of order 5 perform well on both the training and validation sets
- We are unlikely to overfit this problems – 16 data points, order 10 model.

*Reading*

Sections 1.3 of Bishop, Pattern Recognition and Machine Learning gives empirical selection a cursory and cautious treatment.