

# Regularization and Model Selection

Prof. Alessandro Lucantonio

Aarhus University

31/10/2023

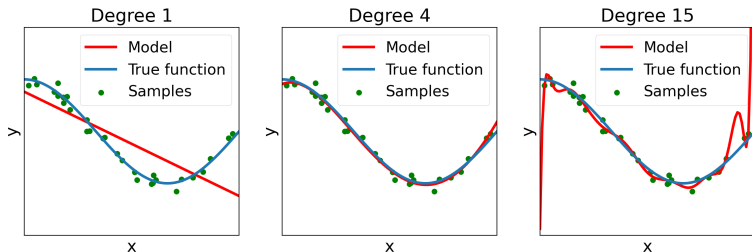
## Underfitting and overfitting - intuition

Imagine that you have to study for a written exam consisting of exercises.

Doing only a few exercises while studying leads to poor performance both on homework and at the exam. This is called *underfitting*: the performance at the exam will be bad because of poor training.

Moreover, memorize all the solutions to homework leads to the maximum score on homework (trivially) but probably a bad score on the exam exercises. This is called *overfitting*: you have a bad performance on the exam because you did not capture the true essence of the homework, rather you have memorized their peculiarities.

# Underfitting and overfitting - analytical example



**Figure:** Underfitting (polynomial degree 1), good fitting (polynomial degree 4), overfitting (polynomial degree 15).

Overfitting can be countered through:

- ▶ validation;
- ▶ regularization.

# Regularization

Overfitting is correlated with “complex” models: to avoid them, we penalize models with large weights. One way to do this is to consider the following cost function

$$E_r(\mathbf{w}) = E(\mathbf{w}) + R_\lambda(\mathbf{w})$$

where  $R_\lambda$  is the *regularization term*.

$\lambda > 0$  is a hyperparameter that must be chosen in the **model selection phase** (see later).

Most common regularization techniques:

- ▶ Tikhonov (or  $L^2$  or *Ridge*):  $R_\lambda(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 = \lambda \sum_i w_i^2$ . Tends to make *all* weights small.
- ▶ LASSO (or  $L^1$ ):  $R_\lambda(\mathbf{w}) = \lambda \|\mathbf{w}\|_1 = \lambda \sum_i |w_i|$ . Tends to make *some* weights 0 (feature selection).

# Linear Regression with Tikhonov regularization

- ▶ Cost function (without regularization):

$$E(\mathbf{w}) = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- ▶ Regularized cost function:

$$E_r(\mathbf{w}) = E(\mathbf{w}) + \lambda \|\mathbf{w}\|^2$$

- ▶ Gradient of the regularized cost function:

$$\nabla E_r(\mathbf{w}) = \nabla E(\mathbf{w}) + 2\lambda\mathbf{w} = 2 \left( \frac{1}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w} \right)$$

- ▶ Normal equation:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Note that in this case  $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}$  is *always* invertible (why?).

# Bias-Variance trade-off

**Bias:** expected discrepancy between targets and predictions given by the model.

**Variance:** measure of the deviation from the expected value given by the model that any particular sampling of the data (from the same underlying data-generating distribution) is likely to cause.

Some intuitions on the effect of regularization:

- ▶ High  $\lambda \rightsquigarrow$  simple model  $\rightsquigarrow$  high bias (underfitting).
- ▶ Low  $\lambda \rightsquigarrow$  complex model  $\rightsquigarrow$  high variance (overfitting).
- ▶ Intermediate  $\lambda$ : optimal solution (good bias-variance trade-off).

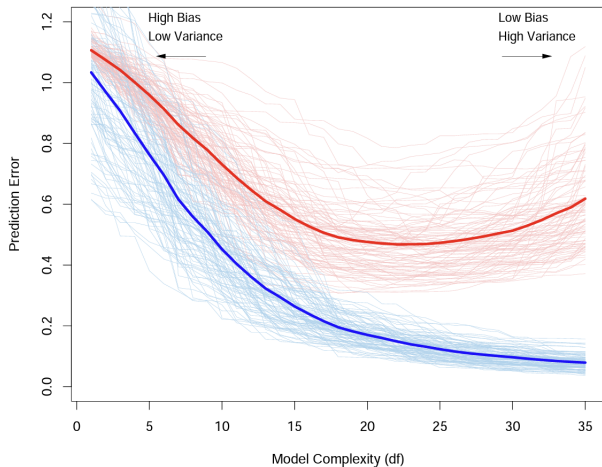
# Generalization

Central challenge in Machine Learning: **generalization!** Our algorithm must perform well on *new, previously unseen* inputs.

Recall that we have to find a balance between bias and variance. Even if we limit the complexity of our model using regularization, the training set does not provide a good estimate of the test (generalization) error.

In other words, generalization is compromised if we choose hyperparameters (including the regularization factor) only according to the training error.

# Generalization - Training vs test error



**Figure:** Training (blue) and test (red) errors as the model complexity varies.



# Model selection and model assessment

Do **not** use the test set to tune hyperparameters: introduce the **validation set**.

In general, we should distinguish between:

1. Model selection: estimate the performance of different models trained with different hyperparameters.
2. Model assessment: after choosing a final model we evaluate its performance on *new, previously unseen* (test) data.

Once we have selected a model using the validation set, we can assess its generalization error on the test set. Splitting the dataset into training and validation sets is called *hold-out*.

# Double Hold-out



- ▶ We split the entire dataset in three sets: training, validation and test (usually 60%-20%-20% or 70%-20%-10% of the total dataset).
- ▶ Training set is used to fit the model. Then, we evaluate its validation performance and adjust its hyperparameters to maximize it. We re-fit the model and keep adjusting the hyperparameters until we find a model with a good trade-off between training and validation performance.
- ▶ We assess the generalization capability of the best model by evaluating the performance on the test set.

# Cross-Validation

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Figure: 5-fold CV

- ▶ Split the data in  $k$  disjoint *folds*.
- ▶ Use  $k - 1$  folds as the training set and the remaining fold as the validation set. Repeat  $k$  times (see figure).
- ▶ The performance will be the mean  $\pm$  standard deviation computed across the  $k$  runs.

*Pro*: Not sensitive to a particular partition of the data. Mean filter the error.

*Cons*: Computationally expensive (but parallelizable).

## CV for selection and hold-out for assessment

1. Split dataset into two sets (hold-out): development  $D$  (model selection) and test  $T$  (model assessment).
2. Use the cross validation method on  $D$  to select the best model. *Optional*: retrain the best model using the entire dataset  $D$ .
3. Evaluate the generalization error on the test set  $T$ .

## Grid search

How to choose the best set of hyperparameters?

**Example:** two hyperparameters (learning rate  $\alpha$  and regularization parameter  $\lambda$ ). Consider sets of three values for each parameter, e.g.  $\alpha_{\text{vals}} = \{0.001, 0.01, 0.1\}$ ,  $\lambda_{\text{vals}} = \{0.0001, 0.001, 0.01\}$ . Evaluate the model with  $(\alpha, \lambda) = (i, j)$  for  $i \in \alpha_{\text{vals}}, j \in \lambda_{\text{vals}}$ .

		$\lambda$		
$\alpha$		(0.001, 0.0001)	(0.001, 0.001)	(0.001, 0.01)
		(0.01, 0.0001)	(0.01, 0.001)	(0.01, 0.01)
		(0.1, 0.0001)	(0.1, 0.001)	(0.1, 0.01)

Choose the pair of value corresponding to the *best performance on the validation set*.

**Refinement:** Suppose that the best is (0.1, 0.001). Then we can “zoom” near the best set doing another grid search with e.g.  $\alpha_{\text{vals}} = \{0.075, 0.1, 0.125\}$ ,  $\lambda_{\text{vals}} = \{0.00075, 0.001, 0.00125\}$ .