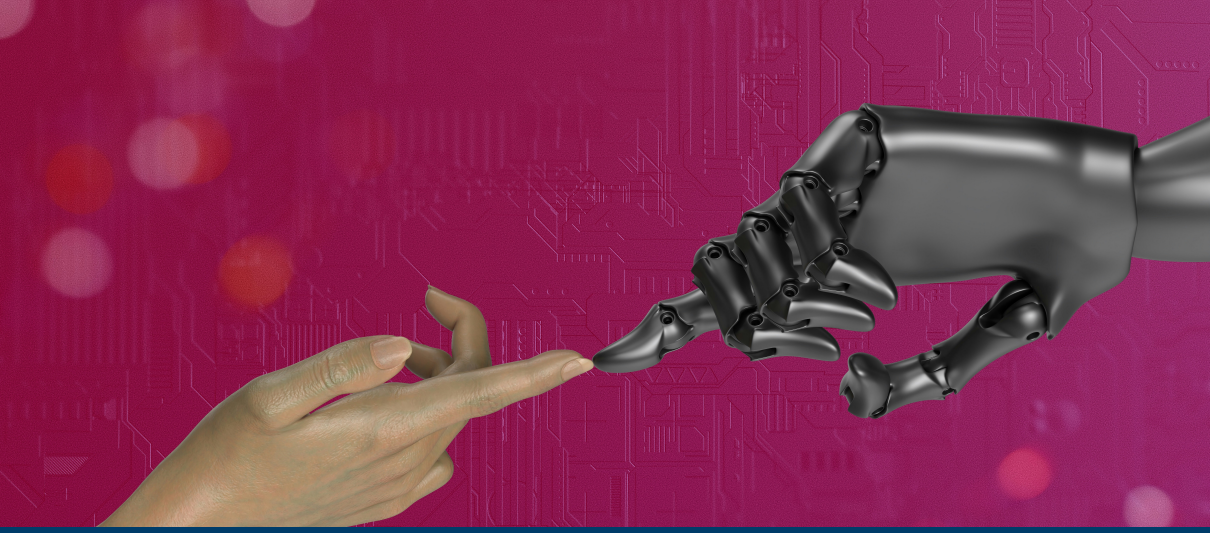


Hands-On Machine Learning

Dr. Derek Bridge | 2026/2027

UCC Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

School of Computer Science and Information Technology
Scoil na Ríomheolaíochta agus na Teicneolaíochta Faisnéise

11. Model Complexity

In this lecture, we use several of the datasets that we used previously.

Non-Linear Models

- Suppose we have three features, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$.
- A linear model will learn values for parameters $b, \boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3$:

$$y = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_3\boldsymbol{x}_3$$

- But what if the true relationship between the features and the target values is non-linear?
- Then you can use non-linear approaches such as a Decision Trees, kNN or even neural networks.

Polynomial Models

- But there's a really neat trick for using a *linear model* to get some of the same effect!
- We systematically add extra features to our dataset.
- Some of the new features will be *powers* of the original features, e.g. a new feature $\boldsymbol{x}_4 = \boldsymbol{x}_1^2$.
- Others will be *products* of the original features, e.g. a new feature $\boldsymbol{x}_7 = \boldsymbol{x}_1\boldsymbol{x}_2$. (These are often called *interaction features*).
- (This should be reminiscent of *feature engineering* — where we derived new features from existing features.)
- Then learn a linear model on the new dataset.

Example: Quadratic Model

- We want to learn models of this form:

$$y = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_3\boldsymbol{x}_3 + \boldsymbol{w}_4\boldsymbol{x}_1^2 + \boldsymbol{w}_5\boldsymbol{x}_2^2 + \boldsymbol{w}_6\boldsymbol{x}_3^2 + \boldsymbol{w}_7\boldsymbol{x}_1\boldsymbol{x}_2 + \boldsymbol{w}_8\boldsymbol{x}_1\boldsymbol{x}_3 + \boldsymbol{w}_9\boldsymbol{x}_2\boldsymbol{x}_3$$

- But instead we learn linear models of this form:

$$y = b + \boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{w}_2\boldsymbol{x}_2 + \boldsymbol{w}_3\boldsymbol{x}_3 + \boldsymbol{w}_4\boldsymbol{x}_4 + \boldsymbol{w}_5\boldsymbol{x}_5 + \boldsymbol{w}_6\boldsymbol{x}_6 + \boldsymbol{w}_7\boldsymbol{x}_7 + \boldsymbol{w}_8\boldsymbol{x}_8 + \boldsymbol{w}_9\boldsymbol{x}_9$$

  but where

$$\boldsymbol{x}_4 = \boldsymbol{x}_1^2 \quad \boldsymbol{x}_7 = \boldsymbol{x}_1\boldsymbol{x}_2$$

$$\boldsymbol{x}_5 = \boldsymbol{x}_2^2 \quad \boldsymbol{x}_8 = \boldsymbol{x}_1\boldsymbol{x}_3$$

$$\boldsymbol{x}_6 = \boldsymbol{x}_3^2 \quad \boldsymbol{x}_9 = \boldsymbol{x}_2\boldsymbol{x}_3$$

- Question: What would a Cubic Model look like?

Warning

- Polynomial Regression of degree d transforms a dataset that had n features into one that has $\frac{(n+d)!}{d!n!}$ features.

Model Complexity

- The **complexity of a model** is its capacity to capture the underlying patterns in the data.
- Do not confuse with **time complexity**, **space complexity**, **sample complexity**, ...

Examples

- Decision trees: complexity increases with depth.
- kNN: complexity decreases with k. (Why?)
- In parametric learning: complexity increases with the number of parameters.
- Polynomial models: complexity increases with the degree. (Why is this saying the same as the previous bullet point?)

Underfitting and Overfitting

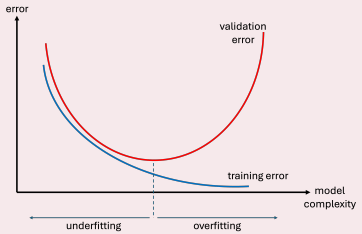
- **Underfitting**: the model is not complex enough.
- **Overfitting**: the model is too complex.

Recap

- We train on the training set.
- But we can evaluate on the training set, validation set or test set.
- This gives us the **training error**, **validation error** or **test error**.

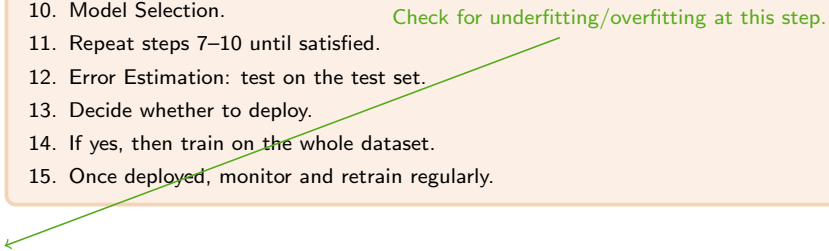
Detecting underfitting/overfitting

- Underfitting: the validation error is high and even the training error is high.
- Overfitting: the training error is low but the validation error is high.



Machine Learning Projects (from previous lecture)

1. Understand the business problem.
2. Select performance measures.
3. Acquire a dataset.
4. Take a cheeky look.
5. Cleanup anything that is simply invalid.
6. Split into training and test sets using holdout.
7. Exploratory Data Analysis.
8. Feature Engineering.
9. Preprocess the data.
10. Model Selection.
11. Repeat steps 7–10 until satisfied.
12. Error Estimation: test on the test set.
13. Decide whether to deploy.
14. If yes, then train on the whole dataset.
15. Once deployed, monitor and retrain regularly.



Remedies for underfitting

- Gathering more training examples **will not help**.

- Change to a more complex model:
  - Polynomial regression: increase the degree.
  - Decision Trees: increase the depth.
  - kNN: reduce k.
- Collect data for additional features that you hope will be more predictive.
- Feature engineering: create new features which you hope will be predictive.

Remedies for overfitting

- Gathering more training examples may help. (Why?)
- If we cannot get more genuine examples, we may synthesize additional examples (see **data augmentation** in a later lecture).

- Change to a less complex model:
  - Polynomial regression: reduce the degree.
  - Decision Trees: reduce the depth.
  - kNN: increase k.
- Clean the training examples to remove noise.
- Dimensionality reduction/feature selection: to reduce the number of features.
- Parametric models: use **Regularization**.
- Decision Trees: use post-pruning.
- Neural Networks: use dropout; use batch-normalisation (see later lectures).
- Gradient Descent: use early-stopping.
- Use an ensemble (although the relationship between ensembles and overfitting is not well-understood).

Regularization for Parametric Models

- **Regularization** constrains the range of values that the parameters can take.
- We 'encourage' the learning algorithm to only choose small values (close to zero).
- This makes the distribution of the values of the parameters more regular.
- We do this by modifying the loss function, e.g.

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2 + \text{penalty}$$

- This modified loss function prefers a hypothesis that (a) fits the data, while at the same time (b) uses small parameter values.

Lasso Regression

- In **Lasso Regression**, we penalize by the l_1-norm of the weights, which is simply the sum of their absolute values, i.e.:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2 + \lambda \sum_{j=1}^{n} |\boldsymbol{w}_j|$$

(Minor point: we don't penalize b.)

The Regularization Parameter

- λ is called the **regularization parameter**.
- It controls how much penalization we want and this determines the balance between the two parts of the modified loss function: fitting the data versus shrinking the parameters.
- When $\lambda = 0$, there is no regularization; but as λ increases, the penalties will get ever greater.
- As λ grows, some of the $\boldsymbol{w}_j$ will be driven to zero. This means that the model learns to treat some features as irrelevant. Hence, Lasso Regression performs some **feature selection** too. (Compare this with Ridge Regression.)
- λ isn't actually a 'parameter'. What should we call it and how could we choose its value?

Ridge Regression

- In **Ridge Regression**, we penalize by the l_2-norm, which is simply the sum of the squares of the weights, i.e.:

$$J(\boldsymbol{X}, \boldsymbol{y}, h) = \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{X}, y \in \boldsymbol{y}} (h(\boldsymbol{x}) - y)^2 + \lambda \sum_{j=1}^{n} \boldsymbol{w}_j^2$$

(Strictly speaking, the l_2-norm would be the square root of the sum of squares.)

- Both Lasso and Ridge Regression shrink the values of the weights
- But we saw Lasso Regression may additionally result in coefficients being set to zero.
- This does not happen with Ridge Regression.
- Optionally, consult section 3.4.3 of The Elements of Statistical Learning by Hastie, Friedman & Tibshirani (available online) for an explanation.
- One observation from the book is that, roughly speaking, Lasso Regression shrinks the coefficients by approximately the same constant amount (unless they are so small that they get shrunk to zero), whereas, again roughly speaking, Ridge Regression shrinks the coefficients by approximately the same proportion.