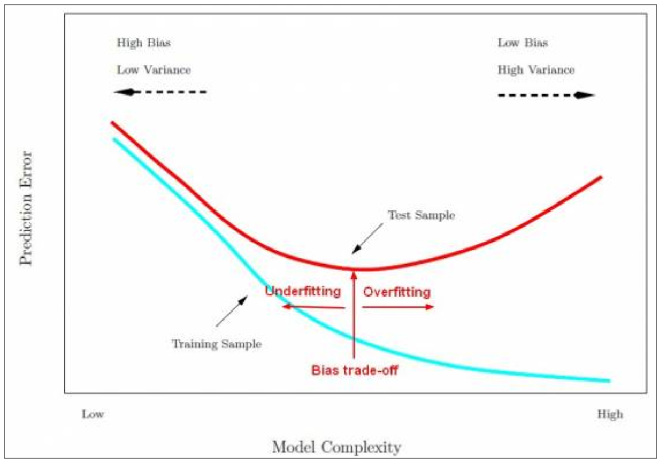
Contents

Chapter 1

Alberi di decisione

1.1 Overfitting

We're trying to solve a problem using data that isn't necessarily usefull (too noisy, wrong features). When training we try to find the function that best approximates the link between training data. But if the training data doesn't mirror the actual properties of the problem, it's possible to end up **overfitting** to the training data and loose generalisation.



Using decision trees, the model complexity corresponds to the depth of the tree. In a neural network it's the number of nodes.

Definition 1.1.1: Overfitting

The problem is we don't know $\mathcal{D}$, so we take two disjointed sets:

- **training set**:
- **validation set**:

There are two ways to avoid overfitting:

- **early stopping**: The top-down construction of the tree is stopped when the benefit to the precision of the model isn't relevant (based on the entropy gained)
- **post pruning**: the tree is first fully developped, then it's retroactivly 'pruned'

1.1.1 Reduce-error post-pruning

Uses the validation set to calculate the accuracy loss when removing subtrees. Notice how the model is still biased to the validation set, so a third set should be used to correctly assess the model's accuracy.

1.2 Random Forests

Instead of just one decision tree, why don't we use an ensamble of trees whose output is the weighted mean of all the outputs. Seen as the construction of a tree is completly determined by the set of training data and the features analyzed, we can use two different methods to differentiate trees in a random forest:

- **Bagging**:
- **Feature randomness**:

Chapter 2

Probabilistic approach

Instead of trying to find the funciton:

$$f : X \to Y$$

we can compute the probability:

$$p : P(Y|X)$$

this is because we just want to approximate the function
va be ripassino di probabilita'