Overfitting

In statistics, overfitting is “the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably”. An overfitted model is a statistical model that contains more parameters than can be justified by the data. The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e. the noise) as if that variation represented underlying model structure.

Let’s say we have 10 dots on a graph. Let’s fit these 10 dots:

- with a line
- with a 2nd order polynomial
- with a 5th order polynomial
- with a 9th order polynomial

Here, we can see a simplified illustration for this example:

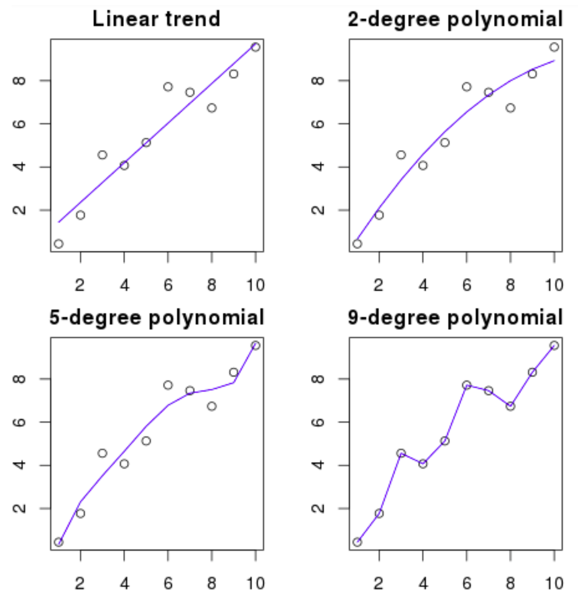

Figure 1

The higher the polynomial order, the better it will fit the existing dots.
However, the high order polynomials, despite looking like to be better models for the dots, are actually overfitting them. It models the noise rather than the true data distribution.
As a consequence, if you add a new dot to the graph with your perfectly fitting curve, it’ll probably be further away from the curve than if you used a simpler low order polynomial.

Overfitting isn't necessarily 'overthinking'. It happens any time you read too much into your notions (whether right or wrong) to optimize something in your future.

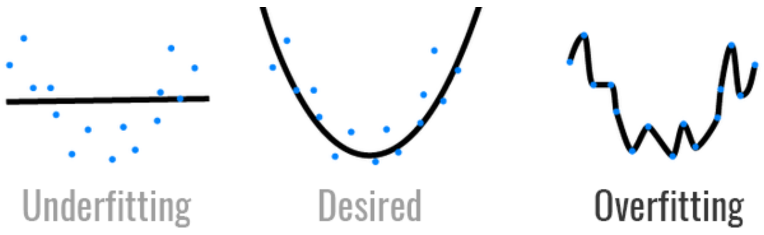

Figure 2

How Overfitting works ?

- Every dataset that you will encounter in the real world contains some form of noise: essentially, patterns that are not pertinent to the task at hand.
- At the onset, a supervised-learner tries to reduce the error between its predictions and targets in the training data.
- This process is beneficial at first, the model keeps getting better and better at grasping useful trends.
- After a certain point however, the learner starts gathering spurious patterns that help with the training data, but don't really generalize well. This, is called overfitting.

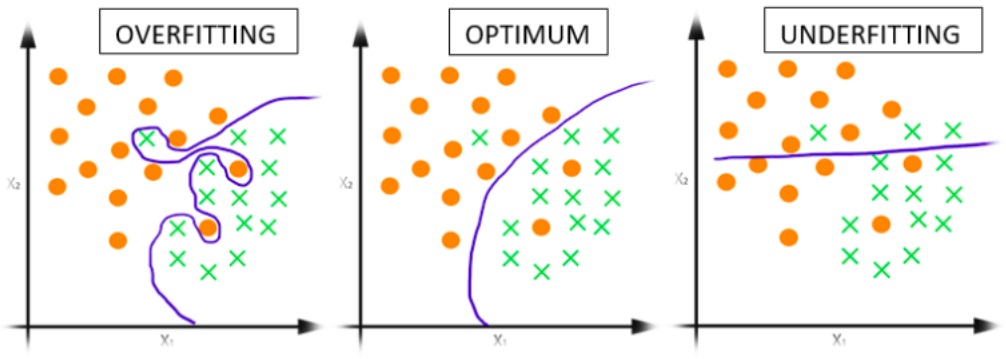

Figure 3

How to detect overfitting ?

A key challenge with overfitting, and with machine learning in general, is that we can't know how well our model will perform on new data until we actually test it.

- To address this, we can split our initial dataset into separate training and test subsets.
- This method can estimate how well our model will perform on new data.
- If our model does much better on the training set than on the test set, then we're likely overfitting.

For example, it would be a big red flag if our model saw 99% accuracy on the training set but only 55% accuracy on the test set. Another tip is to start with a very simple model to serve as a benchmark. Then, as you try more complex algorithms, you'll have a reference point to see if the additional complexity is worth it.

How to reduce overfitting ?

Cross-validation: Cross-validation is a powerful preventative measure against overfitting.

The idea is clever: Use your initial training data to generate multiple mini train-test splits. Use these splits to tune your model.

In standard k-fold cross-validation, we partition the data into k subsets, called folds. Then, we iteratively train the algorithm on k-1 folds while using the remaining fold as the test set (called the "holdout fold").

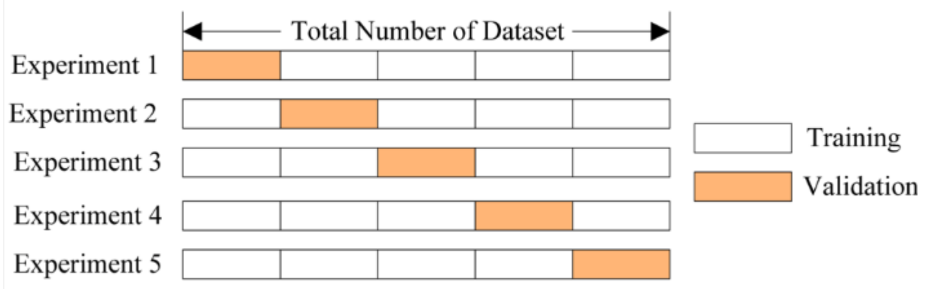

Figure 4

Cross-validation allows you to tune hyperparameters with only your original training set. This allows you to keep your test set as a truly unseen dataset for selecting your final model.

Train with more data: It won't work everytime, but training with more data can help algorithms detect the signal better. Of course, that's not always the case. If we just add more noisy data, this technique won't help. That's why you should always ensure your data is clean and relevant.

Remove features: Some algorithms have built-in feature selection. For those that don't, you can manually improve their generalization by removing irrelevant input features. An interesting way to do so is to tell a story about how each feature fits into the model. This is like the data scientist's spin on software engineer's rubber duck debugging technique (rubber duck debugging or rubber ducking is a method of debugging code. The name is a reference to a story in the book The Pragmatic Programmer in which a programmer would carry around a rubber duck and debug their code by forcing themselves to explain it, line-by-line, to the duck), where they debug their code by explaining it, line-by-line, to a rubber duck. If anything doesn't make sense, or if it's hard to justify certain features, this is a good way to identify them.

Early stopping: When you're training a learning algorithm iteratively, you can measure how well each iteration of the model performs. Up until a certain number of iterations, new iterations improve the model. After that point, however, the model's ability to generalize can weaken as it begins to overfit the training data.Early stopping refers stopping the training process before the learner passes that point.

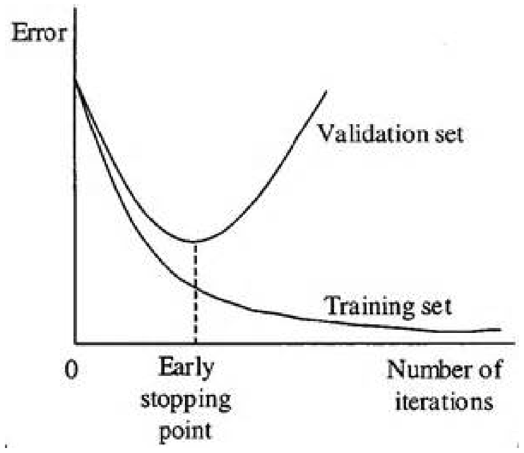

Figure 5

Regularization: Regularization refers to a broad range of techniques for artificially forcing your model to be simpler. The method will depend on the type of learner you're using. For example, you could prune a decision tree, use dropout on a neural network, or add a penalty parameter to the cost function in regression. Oftentimes, the regularization method is a hyperparameter as well, which means it can be tuned through cross-validation.

Ensembling: Ensembles are machine learning methods for combining predictions from multiple separate models. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

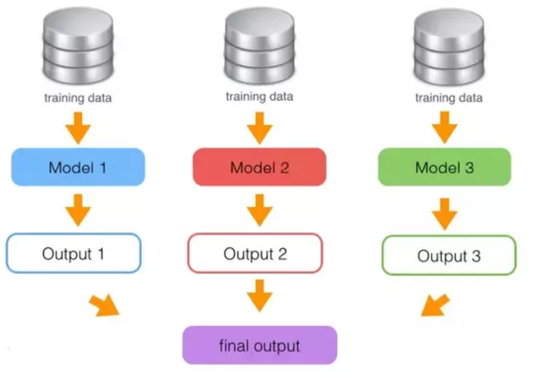

Figure 6

References:

For more details can refer below links:
Regression
`https://en.wikipedia.org/wiki/Overfitting#Regression`
Cross validation
`http://pillowlab.princeton.edu/teaching/mathtools16/slides/lec20_CrossValidation.pdf`
How to detect overfitting
`http://statisticsbyjim.com/regression/overfitting-regression-models/`