Lecture 16: Modelling: Selection, fitting, and validation

We start this lecture by considering something that most of you are aware of: to build a model, we need three steps: a) selection, b) fitting, and c) validation. For the selection, we typically use our prior scientific knowledge to select a single or set of models that might work for the given process or experiment we are trying to model. Sometimes, we can also use various design methods we have discussed so far to obtain a rough understanding of which models work the best: for example linear vs non-linear; polynomial vs exponential, etc. We would then move on to finding a model within each class of selected models that approximates the process we are trying to model. This process is called fitting or training in the modern machine learning language. A model needs to be validated for its accuracy in both model selection and fitting. You may have used the perfect algorithm to fit your model but if it does not follow the same class as the process you are trying to model, it might not be useful at all. However, the availability of large computing resources combined with advances in data-driven modeling techniques in Machine Learning (ML), Statistics, and other approaches allows us to obtain what is referred to as universal approximators of continuous function. The caveat here is the no-free-lunch theorem that states every algorithm for modeling must make assumptions and that no ML algorithm works well in every setting. We will focus on the usage of ML models for experimental design: namely selection, fitting, and validation. Note that we need some data to perform the above three steps and we sometimes have the ability to obtain data on-demand which we can take advantage of. The last few lectures will focus on the ability to generate on-demand data and how can we make use of it in experimental design.

For now, let us consider the simplest case where we want to build a predictive model for the experiment we would like to study. In recent times, this is done via supervised learning methods in ML: assuming we have access to a set of data where we know the output values, we want to build a statistical model that predicts the output for unknown samples. The general setting is as follows: we are given a set of samples we call a dataset $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ where x captures a mathematical representation of our sample and y the output value. The output value can be a continuous value such as the temperature or a discrete label such as a crystalline phase. The model is typically learned by optimizing a *loss function* such that we find the best function within the set of model functions we selected as a *hypothesis*. The loss function is set up in a such way that the training process selects a model from the hypothesis class that makes the fewest mistakes on the data it has been provided. In other words, the loss function measures how well a model approximates our experiment within the given class and the data provided for training. Overfitting to training data is a common scenario in ML and it is recommended to fine-tune our model within the hypothesis class using a validation approach. For example, we specify a model with some learnable parameters such as the weights of a linear model which belongs to the hypothesis class of polynomial models. Then it is common to split our data into training and validation and use the validation set to select the right class. Test data is then used to measure the generalization of the model.

Let us think about how we can make these splits in the data because so far we have only discussed how to generate a set of data without considering modeling explicitly. One scenario is that we would run three different experimental design campaigns and collect three different data sets one each for training, validation, and testing. The problem, in this case, is that it can be quite expensive and can limit our ability to just a single set of training and validation for all the different hypotheses. An alternative to this approach is to generate data once and use the splitting approach commonly performed in ML. This is typically done via cross-validation approaches and care must be taken in how the splits are generated.

Another common issue in model fitting is underfitting where the training and validation loss both are high. This means that there is a 'sweet spot' among the hypotheses that have high enough training accuracy and low enough validation error.

We conclude this lecture by considering a very important notion of bias-variance trade off when training a model to approximate a given experiment. Instead of considering deterministic models, let us consider a probabilistic model that predicts a distribution of values for any given input. It can be proven that for any probabilistic model, the expected test error can be decomposed into three terms : variance, bias and noise. *Variance* captures how much your model changes with a change of training data thus signifying how "over specialized" is your data. *Bias* is the inherent error in your model that is due to a bias to a particular solution. Noise part measures the ambiguity due to data distribution or the quality of your mathematical representation of input. The following graphic in Figure 1 clearly illustrate the trade off between bias and variance :

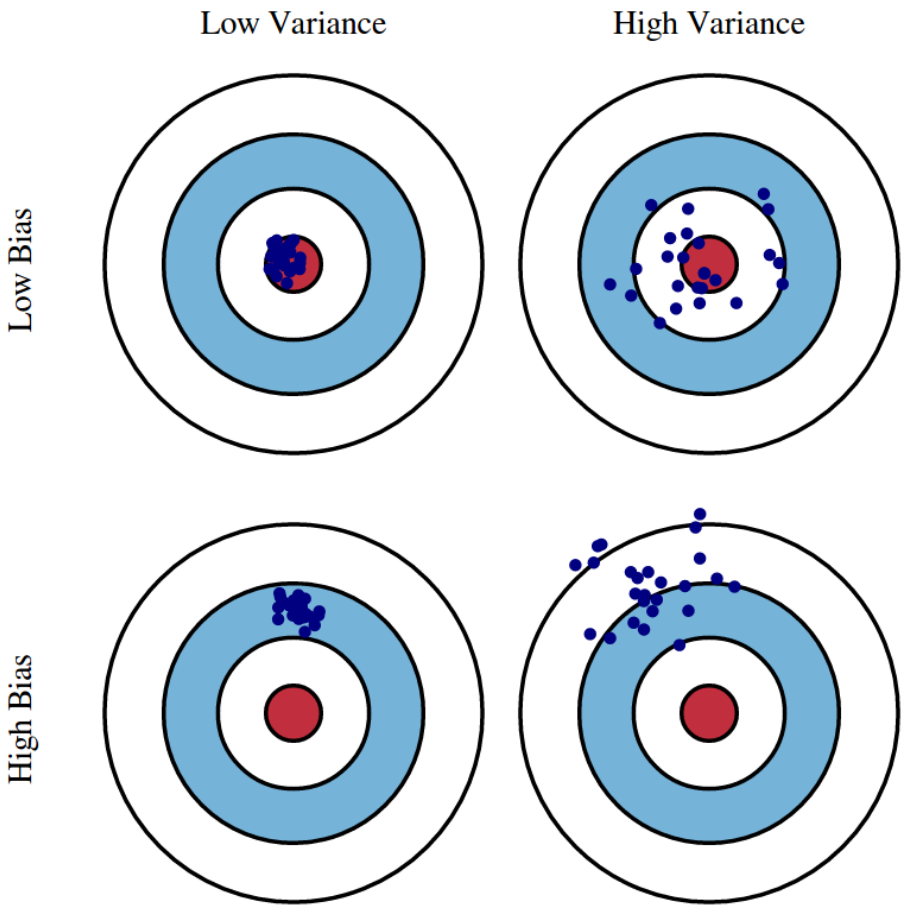

Figure 1: Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Many text books on ML and Statistics cover these topic in much more detail. I encourage interested readers to use the following link and look at lectures 1, 11, 12.