EVALUATING MODEL PREDICTIONS AND MACHINE LEARNING

Contents

1. Learning objectives

- Cross validation and how to use it to perform model selection.
- Bias-variance decomposition of mean-squared error, and why it explains the U-shaped test error curve.
- Fourier series as linear regression models. Orthogonality and why it is important.

2. Cross validation

- We saw from the previous example that R^2 is not very helpful if we want to decide how complex we'd like to make our model, since it is possible to explain all the variation in the Y values with a model which is too complex. In order to address this, we take the approach of **cross validation**. The basic idea of cross validation is to break our data up into two subsets: **training set**, which we use to fit the model, and a **testing set**, which we compare to the predictions of the fitted model.
- **Some Notation**
  - I will use D to refer to our entire data set:

$$D = (Y, X) = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}$$

  - $D^{\text{train}} = (Y^{\text{train}}, X^{\text{train}})$ and $D^{\text{test}} = (Y^{\text{test}}, X^{\text{test}})$ will represent the subsets of the data to be used to training (that is, fitting) the model and testing the fit respectively. We will assume that

$$D = D^{\text{train}} \cup D^{\text{test}}$$

    Let N_{train} and N_{test} denote the number of points in each group.
  - Let $\hat{y}(x, D)$ the prediction of $E[Y|X = x]$ using the fitted coefficients based on a data set D; that is

$$\hat{y}(x, D) = \sum_{i=1}^{K} \hat{\beta}_i \phi_i(x)$$

    where $\hat{\beta}$ are the (usually least squares) fitted coefficients using the data in D.
- Now let us define the **training error**

$$\epsilon^2_{\text{train}} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} (\hat{y}(X_i^{\text{train}}, D^{\text{train}}) - Y_i^{\text{train}})^2$$

  where the average is taken over different replicates of our data and $\hat{Y}_i$ is our prediction of $E[Y|X]$. Not that the training error tells us about how well our model does at predicting the same data we used to fit it, and therefore not surpisingly, it is closely related to R^2:

$$R^2 \approx 1 - \frac{\epsilon^2_{\text{train}}}{\text{var}(Y_{\text{train},i})}.$$

  In order to see how well our model does at predicting the points we did NOT use to fit it, we introduce the test error:

$$\epsilon^2_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (\hat{y}(X_i^{\text{test}}, D^{\text{train}}) - Y_i^{\text{test}})^2$$

  Note that the fitted coefficients used to compute $\hat{y}_i(X^{\text{test}})$ come from fitting the model to the training data, even though we are evaluating $\hat{y}$ at the test points.

Example 1 (Cross validation on polynomial model)**.** *Consider example 6 from the previous weeks notes.*

Question: Plot the training and test error as a function of the number of parameters.

Solution: See colab notebook.

3. Bias, variance and overfitting

- In order to understand U-shaped curve seen in Example 1, we first need to introduce some more general terminology and notation for talking about estimators (of which $\hat{y}$ is an example). To this end, we introduce the **mean-squared error** of an estimator $\hat{\theta}$ of some quantity θ. θ could be a parameter, or it could be a value of a function, such as $f(x)$, that we would like to predict.

$$\text{MSE}_{\hat{\theta}} = E\left[(\hat{\theta} - \theta)^2\right] \tag{1}$$

  For now, let's just think of $\hat{\theta}$ as any estimator.
- The following theorem is the key result which will allow us to understand the U-shaped curve.

  Theorem 1 (Bias variance decomposition)**.**

$$\text{MSE}_{\hat{\theta}} = \text{var}(\hat{\theta}) + E\left[\hat{\theta} - \theta\right]^2 \tag{2}$$

  Proof. Using the definition of variance

$$\text{var}\left(\hat{\theta} - \theta\right) = E\left[(\hat{\theta} - \theta)^2\right] - E\left[\hat{\theta} - \theta\right]^2.$$

  Since θ is a constant, $\text{var}\left(\hat{\theta} - \theta\right) = \text{var}\left(\hat{\theta}\right)$, so rearranging terms yields the result. □

  Now let's break this result down a bit.
  - The first term in Equation 2 is simply the variance, which in the case of an estimator of a parameter we can recognize as the squared standard error. This is tell us how much variation there is in our estimate from replicate of replicate. You should recognize that a consistent estimator is exactly one for which this quantity vanishes when N is very large.
  - The second term is what we will define as the squared of the bias:

$$\text{Bias}_{\hat{\theta}} = E[\hat{\theta} - \theta].$$

    This tell us whether our estimate will on average give us the correct value of θ. You should recognize that an unbiased estimator is exactly one for which this quantity is zero!
- Now let's return to working in the context of a model of the form

$$Y|X \sim \text{Normal}(f(X), \sigma^2)$$

  where $f(X)$ can be written as a linear combination of features

$$f(X) = \sum_{i=1}^{N} \beta_i \phi_i(X).$$

  Letting $\hat{\theta} = \hat{y}(x, D)$, applying Equation 1 yields

$$\text{MSE}_{\hat{y}(x,D)} = E\left[(\hat{y}(x, D) - f(x))^2\right]$$

  where the average is taken over replicates of our data D. This is a measurement of our ability to predict $f(x)$, which is exactly what the test error seeks to measure.
- To more closely relate the bias-variance tradeoff to the test error, we will use a slightly different definition of MSE, which is

$$\widetilde{\text{MSE}}_{\hat{y}(x,D)} = E\left[(\hat{y}(x, D) - Y(x))^2\right] \tag{3}$$

  where Y is a sample from $Y|(X = x)$. You can show (see Exercise) that

$$\widetilde{\text{MSE}}_{\hat{y}(x,D)} = \sigma^2 + \text{var}(\hat{y}(x, D)) + E\left[\hat{y}(x, D) - f(x)\right]^2$$

  The new term, σ^2, captures the fact that, unlike the deterministic term f, we can NEVER predict the *random* variable Y exactly. You should be able to justify that

$$\epsilon^2_{\text{test}} \approx E\left[\widetilde{\text{MSE}}_{\hat{y}(X,D^{\text{train}})}\right]$$

- The important thing to recognize is that a complicated model will tend to have a higher variance, since it will be able to change more in response to data. Meanwhile, a very simple model will tend to have a higher bias.

  Example 2 (Plotting Bias and variance)**.** *Let's continue with Example 1 and try to illustrate the bias variance tradeoff by computing these (or really approximations to them) separately.*

  Solution: See colab notebook.

- We now summarize some observations we've made thus far:
  - In a regression model, as we add more coefficients, eventually the model becomes more and more "flexible", in the sense that is can describe more different types of data sets. Here, by "describe", we mean that it can be fit to those data sets with a small value of R^2 or training error. With enough features, a model can perfectly interpolate between the data points, meaning $\hat{y}(X_i, D) = Y_i$ for each data point (X_i, Y_i) when the model is fit on all our data points.
  - Unlike the training error and R^2, the test error, ϵ^2_{test} does not simply increase as we make the model more complex, rather it has a U-shape. Thus, there is a optimal model size at which our model's predictions of new data points – that is, data points outside the set we used to fit it – is best.
  - The U-shape can be understood in terms of bias and variance. We know this because the mean-squared error, which is the "math world" version of ϵ^2_{test}, can be decomposed into a bias and variance term. The variance term tells us how variable the predictions of our model will be when we fit it to different training sets, while the bias tells us how much its predictions. will differ, on average, from real data.

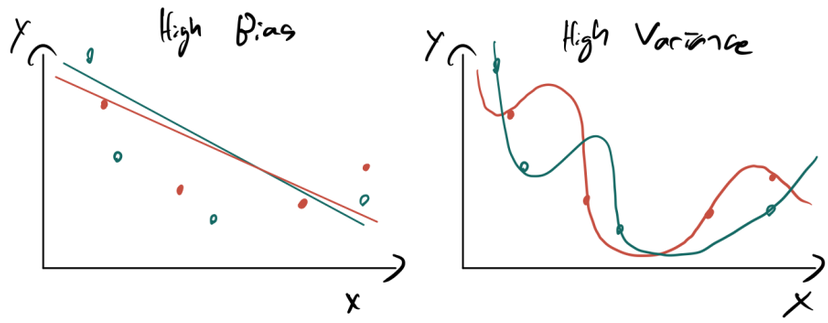

Figure 1. Bias and variance illustrated with fits to two different datasets (blue and red) drawn the from the same distribution

4. Orthogonality and Fourier analysis

- Sometimes we have a particularly hypothesis about what features my contribute to an signal we are analyzing – for example, the CO2 data from last week we suspected there was a linear trend and yearly trend superimpose together. This was reasonable because we know that whether tends to follow a yearly cycle. But in general, if we don't have such knowledge how should we select the functions ϕ? It is often to our advantage to select basis function ϕ_i so that our predictors are not correlated. That is, if $\phi_i(X)$ and $\phi_j(X)$ are thought of as random variables (the randomness comes from X) then we want them to be uncorrelated.
- Throughout our discussion, we will assume that $E[\phi_i(X)] = 0$. We can easily make this so by subtracting the means of our features. Therefore, we would like to find ϕ_i such that

$$E[\phi_i(X)\phi_j(X)] = 0 \quad i \neq j. \tag{4}$$

We this holds for some distribution of X, we say that ϕ_i and ϕ_j are **orthogonal** with respect to this distribution.

Example 3 (Sin series)**.** *Let us now work with the particular example where*

$$X \sim \text{Uniform}(-L, L). \tag{5}$$

Even if our X points are not random, but are say evenly spaced on the interval $[-L, L]$, a random sample from the uniform distribution will be statistical similar to some randomly selected evenly spaced X points. Thus, we can think of the uniform distribution as approximating the spread of our X data with a random variable.

Now consider the basis functions

$$\phi_j(x) = \sin\left(\frac{\pi x j}{L}\right).$$

Question: Show using simulations that ϕ_j are orthogonal with respect to the uniform distribution on $[-L, L]$ (Equation 5).

Solution: See colab notebook.

- Notice that if ϕ_j are orthogonal, then we can express the regression coefficients as

$$\beta_j = \frac{\text{cov}(Y, \phi_j(X))}{\text{var}(\phi_j(X))} = \frac{E[Y\phi_j(X)]}{E[\phi_j(X)^2]}$$

and hence our fitted coefficients from N data points can be approximated as

$$\hat{\beta}_j \approx \frac{\sum_{i=1}^N \phi_j(X_i)Y_i}{\sum_{i=1}^N \phi_j(X_i)^2} \tag{6}$$

This suggests $\hat{\beta}_j$ should depend only weakly how many of the features ϕ we have included in our model! This is consequence of orthogonality. In contrast, in the earlier example where we used polynomial features adding a new predictor, say X^4, would dramatically change the fitted value of β_2.

- Now let's return to Example 3. A limitation of this model is that it only permits us to model "odd functions" that is, functions for which $f(x) = -f(x)$, as each ϕ_j has this property. It is therefore desirable to add addition features which permit this, but we'd like them to still be orthogonal. To this end, we introduce a new set of features to the model given by the cosine functions:

$$\cos\left(\frac{\pi x j}{L}\right).$$

Our new model, called a **Fourier series**, is given by

$$f(x) = \beta_0 + \sum_{i=1}^K \beta_i \sin\left(\frac{\pi i x}{L}\right) + \alpha_i \cos\left(\frac{\pi i x}{L}\right)$$

where we are using α_i to represent the coefficients of the cosine terms. Fourier series are on of the most important models in data science and engineering. It can be proved that as $K \to \infty$ we can approximate essentially ANY function with a series of this form.

Example 4 (Working with Fourier series in python)**.** *Suppose we have data from the model with*

$$f(x) = \sin(3\pi x) + \sin(10\pi x)$$

where $X_i = (i-1)/N$ (meaning the X points are evenly spaced in $[0,1]$.

Question: For different values of σ^2, generate data from this model and fit it to a Fourier series with $K = 100$ terms.

Solution: See colab notebook.

- The process of computing the coefficients $\hat{\beta}_j$ and $\hat{\alpha}_j$ for the Fourier series is called a **Discrete Fourier Transform**. Usually DFT refers to the cases where the X_i are equally spaced. In this case, the orthogonality condition (Equation 4) is true is "data world", not just "math word" – what I mean by this is that for equally spaced data points, say $X_i = L(i-1)/N$, we have

$$\frac{1}{N}\sum_{i=1}^{N} \sin\left(\frac{2\pi j X_i}{L}\right)\sin\left(\frac{2\pi k X_i}{L}\right) = 0 \quad k \neq j$$

- Often we want to summarize how different frequencies are represented in our data, but we don't particularly care about whether they come form the sin or cos terms. To achieve this, one uses the **power spectrum density**, also known as the **peridogram**,

$$P_j = \beta_j^2 + \alpha_j^2.$$

The power spectrum density is a fundamental object in signal processing, and it essentially tells us how "wobbly" a signal is.

Example 5 (Periodogram)**.** *Question: Compute the periodogram of the data generated in Example* **??** *and confirm the same periodogram can be generated with the* `periodogram` *function from the* `scipy.signal` *library. This is neat and not often made point, that this fundamental structure from signal processing is in fact coming from fitting a linear regression model with least square!!*

Solution: See colab notebook.