7 A closer look at the bias-variance dilemma

In this short section we will give a formal treatment of the bias-variance theme. First we will see how the relatively young *statistical learning theory*[1] (SLT) addresses this problem. Then we will take a more traditional stance and see how the generalization error of any learning method is made of two components, the *approximation error* (squared "bias") and an *estimation error* ("variance"). The approximation error captures how close the best possible model in some model class comes to the target function; the variance measures how strongly the learnt models adapt themselves to the random variations of individual training data sets. I use material from Section 9.1 of the Bishop book and from the texts indicated in the footnotes.

We consider the following situation of learning a regression model. We are given a probability space (Ω, F, P), a random variable X with values $\mathbf{x}$ in $\mathbb{R}^k$ and a random variable Y with values y in $\mathbb{R}$. Pairs $(X(\omega), Y(\omega)) = (\mathbf{x}, y)$ are argument-value pairs of some stochastic function which we want to learn from such pairs. From an eagle's perspective, the task of statistical learning is to estimate a function $\hat{f}$ (the *model* that we learn) which provides the smallest possible value of the average error R that we make when we use functions f to predict y from $\mathbf{x}$,

$$(7.1) \qquad R(f) = \int_{\Omega} (f(X(\omega)) - Y(\omega))^2 \, dP(\omega) = \int_{\mathbb{R}^k \times \mathbb{R}} (f(\mathbf{x}) - y)^2 \, dP_{X \times Y}(\mathbf{x}, y)$$

$R(f)$ is called the *risk* in statistical learning theory. The risk can be interpreted as a generalization error or expected test error. It is not possible to use (7.1) for minimizing the risk because the joint distribution $P_{X \times Y}$ is unknown. All that is available for learning is a finite sample of N instances of data pairs $D = ((\mathbf{x}_i, y_i))_{i = 1,\ldots,N}$ – well known to us as training data. A straightforward way to estimate $\hat{f}$ is to minimize the training error $R_{\text{emp}}(f)$, which is called *empirical risk* in statistical learning theory:

$$(7.2) \qquad R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 .$$

We know from all our experience that a brute-force minimization of $R_{\text{emp}}(f)$ is likely to succeed perfectly, yielding a model $\hat{f}$ that has zero empirical risk – but generalizes poorly because we just (over)fitted the training data. SLT is a rigorous mathematical account of this situation. One main result gives bounds on the risk that connect the sample size N to the model complexity of f (for the time being, think of model complexity as the number of parameters available to tune f).

SLT assumes that models f are selected (by learning) from some family – for instance, the family of linear neural networks, the family of linear neural networks with k neurons, or whatever. Formally, such a family of potential models is written as $(f_\alpha)_{\alpha \in \Lambda}$. The risk of a particular function f_α from such a family is also written as $R(\alpha)$, the empirical risk as $R_{\text{emp}}(\alpha)$.

[1] Recommended textbook: Vladimir N. Vapnik, The Nature of Statistical Learning Theory (Second Edition). Springer Verlag 1999[2]. Recommended introductory paper (online via IRC): V. N. Vapnik, An Overview of Statistical Learning Theory. IEEE Transactions on Neural Networks 10(5), 1999

A fundamental idea in SLT is to characterise such a family of candidate functions by their "expressiveness", giving a single characteristic quantity that captures how "rich" the family $(f_\alpha)_{\alpha\in\Lambda}$ is. There are several such characteristics, but the most widely used is the *Vapnik-Chervonenkis(VC)-dimension*. Here is the definition of the VC dimension for a family of functions, for two cases: when the functions are used for binary classification tasks and when they are used for regression tasks.

Definition 7.1 (VC dimension for indicator functions). Let $(f_\alpha)_{\alpha\in\Lambda}$ be a family of indicator functions (that is, 0-1-valued functions) on some vector space V. Then the VC-dimension of $(f_\alpha)_{\alpha\in\Lambda}$ is the maximal number h of vectors $z_1, ..., z_h$ that can be *shattered* by functions from $(f_\alpha)_{\alpha\in\Lambda}$, that is, for each of the 2^h possible ways of assigning the vectors to two classes C_1 and C_2, there exists one function from the family that assigns a value of 0 to the vectors in C_1 and a value of 1 to the vectors in C_2. If any number of vectors can be shattered, $h = \infty$.

Definition 7.2 (VC dimension for real-valued functions). Let $(f_\alpha)_{\alpha\in\Lambda}$ be a family of real-valued functions on some vector space V. Consider the family of indicator functions $(I_{\alpha,\beta})_{\alpha\in\Lambda,\,\beta\in\mathbb{R}}$ obtained from $(f_\alpha)_{\alpha\in\Lambda}$ by putting $I_{\alpha,\beta}(\mathbf{x}) = 0$ if $f_\alpha(\mathbf{x}) - \beta < 0$, else 1. Then the VC-dimension h of $(f_\alpha)_{\alpha\in\Lambda}$ is the VC-dimension of $(I_{\alpha,\beta})_{\alpha\in\Lambda,\,\beta\in\mathbb{R}}$.

Example 1. If $(f_\alpha)_{\alpha\in\Lambda}$ is the family of lines in the plane [more precisely, the family of indicator functions that are 0 on one side of a line], then $h = 3$, because 3 points in the plane can be separated into all possible two-class partitions by lines (Fig. 7.1 left) whereas this is not possible for 4 points (Fig. 7.1 right).

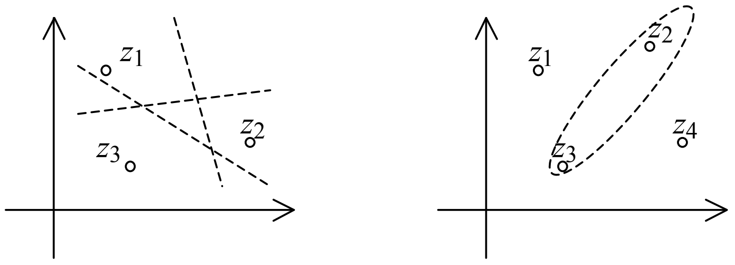

Figure 7.1: Three points z_1, z_2, z_3 can be shattered in the plane by lines, but with four points there are always two (here: z_2 and z_3) that cannot be separated from the other two by lines.

Example 2. Let H be the Heaviside step function (recall: $H(x) = 0$ if $x < 0$, else 1). Consider the set of *linear indicator functions*

$$(7.3) \qquad f_\alpha((x_1,...,x_k)) = H\left[\sum_{i=1}^{k} \alpha_i x_i + \alpha_0\right]$$

on $\mathbb{R}^k$. This set of functions has VC dimension $h = k+1$.

Example 3. The VC dimension of the set of linear functions

$$(7.4) \qquad f_\alpha((x_1,...,x_k)) = \sum_{i=1}^{k} \alpha_i x_i + \alpha_0$$

on $\mathbb{R}^k$ also has VC dimension $h = k+1$.

Note that in these examples, the number of free parameters $\alpha_0,..., \alpha_k$ equals the VC-dimension, in accordance with our old intuition that the complexity, or expressiveness, of a class of models scales with the number of tuneable parameters. In general, however, this correspondence need not hold:

Example 3. Let Λ be the set $1\{0,1\}^*$ of all binary strings starting with a 1, interpreted as integer numbers written in base 2. For $\alpha \in \Lambda$, consider the function f_α: $\mathbb{R} \to \{0,1\}$ defined by

$$(7.5) \qquad f_\alpha(x) = \begin{cases} 0, & \text{if the remainder of } \alpha / x \text{ (in base 2) starts with a 0} \\ 1, & \text{if the remainder of } \alpha / x \text{ starts with a 1} \end{cases}$$

With this set of indicator functions, we can shatter k points $z_1 = 10$, $z_2 = 100$, ..., $z_k = 2^k$, (in binary representation), for any k. To see why, consider an example where $k = 4$ and we want to find an indicator function that assigns z_1, z_2 and z_4 to class 2, z_3 to class 1. We choose $\alpha = 1011$ and find $\alpha/z_1 = 101.1$, $\alpha/z_2 = 10.11$, $\alpha/z_3 = 1.011$, $\alpha/z_4 = 0.1011$, that is, $f_\alpha(z_1) = f_\alpha(z_2)$ $f_\alpha(z_4) = 1$ and $f_\alpha(z_3) = 0$. It becomes clear from this example how we just exploit a binary shift operation to code arbitrary class memberships. Because this works for any k, the VC dimension of this family is infinite – although we only have a single free parameter, α.

The VC dimension is called called the *capacity* of a family of models. An important contribution of SLT is that by the VC dimension / capacity it has found a rigorous and productive method to quantify what we have earlier in the lecture called the complexity (or expressiveness) of a class of models, and what we intuitively related to the number of tuneable parameters. One lesson of SLT is that the sheer number of free parameters in a model family is not always an appropriate measure of the family's modelling capacity.

Equipped with the capacity h, we can now state a fundamental result of SLT, which gives an upper bound on the total risk $R(\alpha)$:

Theorem 7.1 (structural risk minimization principle[2]). The total risk $R(\alpha)$ is bounded by

$$(7.6) \qquad R(\alpha) \leq R_{emp}(\alpha) + \phi\left(\frac{h}{N}, \frac{\log(\eta)}{N}\right)$$

with a probability of at least $1-\eta$, where the *confidence term* ϕ is defined by

$$(7.7) \qquad \phi\left(\frac{h}{N}, \frac{\log(\eta)}{N}\right) = \sqrt{\frac{h\left(\log\frac{2N}{h} + 1\right) - \log(\eta/4)}{N}}.$$

Here, N is the size of the training data set and h is the VC-dimension of the family $(f_\alpha)_{\alpha\in\Lambda}$.

[2] given here as presented in B. Schölkopf, Support Vector Learning, GMD-Bericht Nr. 287, R. Oldenbourg Verlag 1997

The bound in (7.6) deserves some comment. It is intended as a guide in situations where we have small training samples, where "small" means that the ratio N/h is small, say, $N/h < 20$. In this condition, the confidence term increases with h. If we wish to control the risk we can adjust two quantities, the empirical risk and the confidence term. The empirical risk becomes smaller when we fit our training data better – which we can achieve by increasing h, that is, use more complex models. However, by increasing h at the same time we increase the confidence term. Thus we fare best at some compromise value of h.

Concretely, SLT proposes to do the following. Instead of considering a single family $(f_\alpha)_{\alpha\in\Lambda}$, a sequence of families $(f^n{}_\alpha)_{\alpha\in\Lambda_n}$ is considered, such that the corresponding capacities h_n form an increasing sequence $h_1 \le h_2 \le \ldots \le h_n \le \ldots$. This could, for instance, be achieved by considering families of neural networks with increasing numbers of neurons. The capacity is used as a control parameter to optimize the final risk, that is, to minimize the generalization error. One considers the sum of the empirical risk and the confidence term, according to (7.6), and selects that h_i which makes this minimal. Figure 7.2 shows the error curves.

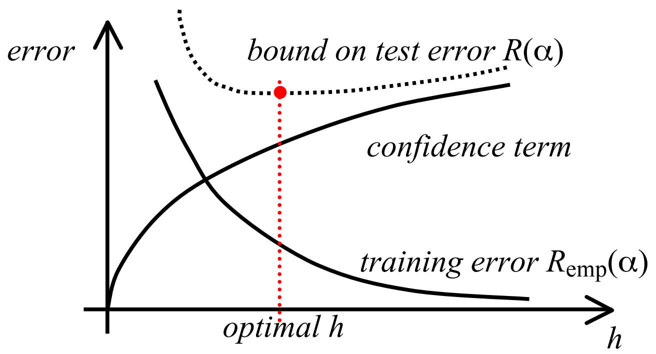

Figure 7.2 Optimal model capacity as a compromise between conflicting demands of small training error vs. small confidence term.

This principle for defining and finding an optimal tradeoff between small training error and a small confidence term is called the *principle of structural risk minimization* in SLT. It can be regarded as one way to deal with the bias-variance dilemma. SLT offers numerous concrete techniques for various types of learning problems to implement the principle of structural risk minimization. The most conspicuous contribution of SLT is, however, that it gives a theoretical foundation for *support vector machines*, a relatively recent technique for learning classification functions. From a SLT perspective, support vector machines are distinguished by the fact that they have a huge number of tuneable parameters but a small VC dimension (just the contrary of our binary shift example 3 from above!). The huge number of tuneable parameters brings with it a small training error, which is however not bought at the expense of a bad generalization error, because the confidence term can be kept small due to the small h.

After this glimpse on SLT we re-address the bias-variance dilemma from a more traditional angle. We will finally explain the origin of the term "bias-variance"!

Without proof we note the following, intuitively plausible fact. Among all functions f that we may consider, the risk $R(f) = \int_{\mathbb{R}^k \times \mathbb{R}} (f(\mathbf{x}) - y)^2 \, dP_{X\times Y}(\mathbf{x}, y)$ is minimized by the function

$$(7.8) \qquad f_{\text{minrisk}}(\mathbf{x}) = E_P[Y \mid X = \mathbf{x}] = \int_{\mathbb{R}} y \; dP_{Y|X=\mathbf{x}}(y),$$

that is, the expected value of y under condition $X = \mathbf{x}$. Here $P_{Y|X=\mathbf{x}}$ is the conditional distribution of y under hypothesis $\mathbf{x}$. We will use shorthand $\langle y \mid \mathbf{x} \rangle$ for $E_P\,[Y \mid X = \mathbf{x}]$. Note that $\langle y \mid \mathbf{x} \rangle$ is just a function of $\mathbf{x}$. We use subscript P in E_P to indicate that the expectation is computed w.r.t. a conditional probability measure that is derived from the probability space (Ω, F, P).

Let us analyse the learning situation. Statistically, a learning algorithm is an estimator that gets data $D = ((\mathbf{x}_i, y_i))_{i=1,\dots,N}$ as input and returns an estimate $\hat{f}$. We can consider this estimator as a random variable from a probability space (Ω', F', P'), whose elements ω' are events of drawing a sample $((\mathbf{x}_i, y_i))_{i=1,\dots,N}$, so we should correctly write $((\mathbf{x}_i, y_i))_{i=1,\dots,N}\,(\omega')$ or $((\mathbf{x}_i(\omega'), y_i(\omega')))_{i=1,\dots,N}$. The estimates $\hat{f}$ are also random variables over this probability space, and we should correctly write $\hat{f}\,(\omega')$ to denote the function obtained from learning, and $\hat{f}\,(\omega')(\mathbf{x})$ to denote the value on argument $\mathbf{x}$ of this function.

Now let us fix some $\mathbf{x}$ and ask by how much $\hat{f}\,(\mathbf{x})$ deviates, on average and in the squared error sense, from the theoretical optimal function $\langle y \mid \mathbf{x} \rangle$. This expected error is

$$(7.9) \qquad E_{P'}[(\hat{f}\,(\mathbf{x}) - \boxed{\langle y \mid \mathbf{x} \rangle})^2] = \int_{\Omega'} (\hat{f}(\omega)(\mathbf{x}) - \langle y \mid \mathbf{x} \rangle)^2 \, dP'.$$

We can learn more about this error if we re-write $(\hat{f}\,(\mathbf{x}) - \langle y \mid \mathbf{x} \rangle)^2$ as follows:

$$(7.10) \qquad \begin{aligned} (\hat{f}(\mathbf{x}) - \langle y \mid \mathbf{x} \rangle)^2 &= (\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})] + E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle)^2 \\ &= (\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})])^2 + (E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle)^2 \\ &\quad + 2(\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})])(E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle). \end{aligned}$$

If we now take the expectation $E_{P'}$ on both sides, we see that the third term on the r.h.s. vanishes and we get

$$(7.11) \qquad E_{P'}[(\hat{f}(\mathbf{x}) - \langle d \mid \mathbf{x} \rangle)^2] = (E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle)^2 + E_{P'}[(\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})])^2].$$

The two components of this error are conventionally named the (squared) *bias* $(E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle)^2$ and the *variance* $E_{P'}[(\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})])^2]$ contribution to the error $E_{P'}[(\hat{f}(\mathbf{x}) - \langle y \mid \mathbf{x} \rangle)^2]$. The bias measures how on average the learning result $\hat{f}\,(\mathbf{x})$ differs from the optimal value $\langle y \mid \mathbf{x} \rangle$. The bias measures how strongly the average learning result deviates from the optimal value; thus is indicates a systematic error component. The variance measures how strongly the learning results $\hat{f}\,(\mathbf{x})$ vary around their mean $E_{P'}[(\hat{f}\,(\mathbf{x})]$; thus this is an indication of how strongly the particular training data sets induce variations on the learning result. Note that the bias and variance shown in (7.11) are functions of $\mathbf{x}$. By integrating over $\mathbf{x}$, one can obtain the average values for the bias and variance:

(7.12)
$$\text{(bias)}^2 = E_{P'}[(E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle)^2] = \int_{\mathbb{R}^k} (E_{P'}[\hat{f}(\mathbf{x})] - \langle y \mid \mathbf{x} \rangle)^2 \, dP_X$$
$$\text{variance} = E_{P'}[E_{P'}[(\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})])^2]] = \int_{\mathbb{R}^k} E_{P'}[(\hat{f}(\mathbf{x}) - E_{P'}[\hat{f}(\mathbf{x})])^2] \, dP_X .$$

For an elementary demonstration of the bias-variance theme, consider a toy situation where we want to estimate a point-argument function f: $\{1\} \to \mathbb{R}$, $f(1) = 1$. The training sample is a set of pairs $(\mathbf{x}_i, y_i)_{i=1,...,N}$, where for all i we have $\mathbf{x}_i = 1$ and $y_i = f(\mathbf{x}_i) + \nu = 1 + \nu$, whith ν being a "noise term" (technically, a random variable $(\nu(\omega_i))$!). Now consider three machine learning algorithms (a statistician would call them "estimators") S, T, U, which take such training samples $\mathbf{x}_i, y_i)_{i=1,...,N}$ and return the following three estimates $\hat{f}(1)$:

$$S : ((1,y_1),...,(1,y_N)) \mapsto \hat{f}(1) = (y_1 + y_2)/2,$$
$$T : ((1,y_1),...,(1,y_N)) \mapsto \hat{f}(1) = (y_1 + y_2 + ... + y_N)/N,$$
$$U : ((1,y_1),...,(1,y_N)) \mapsto \hat{f}(1) = \frac{1}{2}\left(1.1 + (y_1 + y_2 + ... + y_N)/N\right),$$

where 1.1 is an informed guess about the true (but not precisely known) true value $f(1) = 1$. Figure 7.3 shows typical outcomes $\hat{f}(1)$ of applying these estimators S, T, U to samples $(\mathbf{x}_i, y_i)_{i=1,...,N} = (X_i(\omega), Y_i(\omega))_{i=1,...,N}$.

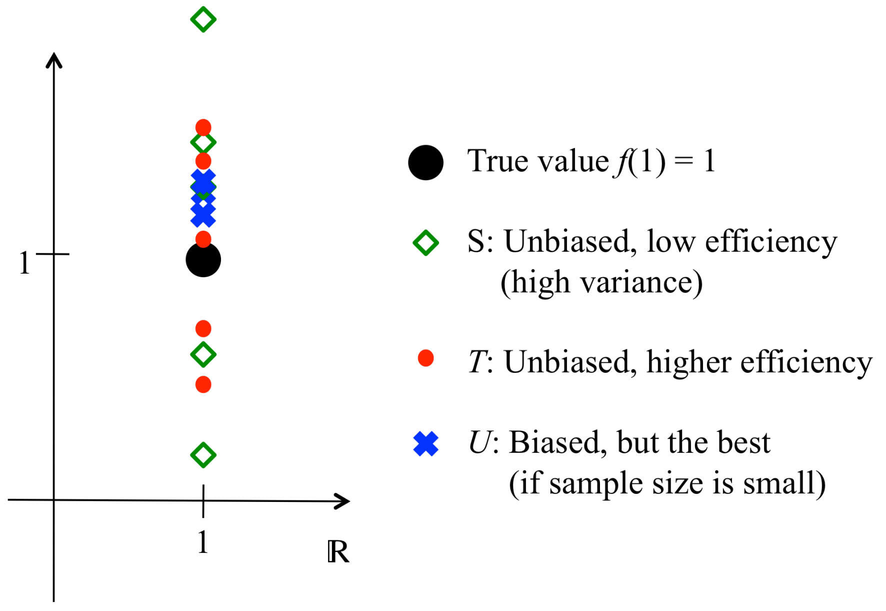

Figure 7.3 Estimator visualization for three simplistic estimators with different bias/variance characteristics. Each plotted point depicts a learning outcome $\hat{f}(\omega)(1)$, where $\hat{f}$ was estimated with S, T or U.