Note Set 5: Predictive Modeling: Regression

Padhraic Smyth,
Department of Computer Science
University of California, Irvine
January 2024

1 Introduction

Prediction is the problem of predicting a variable Y taking values y, given values for an input vector $\underline{x}$. If y is real-valued the prediction problem is known as *regression* and if y is categorical (taking K values, $K \geq 2$) then the problem is known as classification. In this set of notes we will primarily focus on regression and discuss some of the basic concepts involved in regression modeling, including: functional forms of regression models, minimizing empirical risk and expected loss for fitting models to data, connections to probability (and likelihoods and priors), and classic results in bias-variance.

2 Notation

In the discussion below $\underline{x}$ will be a d-dimensional column vector with components $x_1, \ldots, x_d$. We will generally assume that one of the components is a constant $= 1$, which will allow us to have an intercept term in our models. It is also typical to assume that each of the x_j, $j = 1, \ldots, d$ are integer or real-valued. If we have a categorical variable in our input $\underline{x}$, taking K values, we can for example recode it as K binary variables.

For data we will use $D = \{(\underline{x}_i, y_i)\}, i = 1, \ldots, N$ where here the subscripts refer to data points. We can think of D as an $N \times (d + 1)$ data matrix if we wish, where each row corresponds to $(\underline{x}_i^T, y_i)$, i.e., the d component values of the transpose of $\underline{x}_i$ plus the corresponding target value y_i. In some contexts below we will refer to x_{ij}: this is the value of the jth component (the jth feature) for the ith data vector $\underline{x}_i$. The notation is not perfect but should be clear from the context.

We will refer to predictive models in the form of $f(\underline{x}; \theta)$: this is a function that takes a vector $\underline{x}$ as input and produces (unless stated otherwise) a scalar value that predicts y. This function is parametrized by a parameter vector $\underline{\theta}$ which we will treat as a column vector $\theta_1, \ldots, \theta_p$ with p parameters (sometimes also referred to as coefficients or weights). The goal of data-driven predictive modeling is to learn the parameters $\underline{\theta}$ from data D.

3 Examples of Predictive Models

Linear Models Let $f(\underline{x};\underline{\theta})$ be a model for predicting y, with functional form f and parameters $\underline{\theta}$. A *linear model* is one in which is linear in the parameters, e.g.,

$$f(\underline{x};\underline{\theta}) \;=\; \sum_{j=1}^{d} \theta_j x_j \;=\; \underline{\theta}^T \underline{x}$$

with parameter vector $\underline{\theta} = (\theta_1, \ldots, \theta_d)$. Note that we can define a linear model that is linear in the parameters θ and *non-linear* in the input variables $\underline{x}$. For example, if $\underline{x} = (x_1, x_2, 1)$ we could define a model such as

$$f(\underline{x};\underline{\theta}) \;=\; \theta_1 x_1 + \theta_2 x_2 + \theta_3 x^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2 + \theta_6$$

One way to think about this is that we have in effect replaced our original input $\underline{x}$ with an augmented input $\underline{x}' = (x_1, x_2, x_1^2, x_2^2, x_1 x_2, 1)$, and can then write our model as being a linear function of the parameters and the augmented input, i.e., as $\underline{\theta}^T \underline{x}'$.

Estimating the parameters $\underline{\theta}$ for a linear model is often quite straightforward. For example, if our loss function is squared-error and our model is linear then the empirical risk (see next section) will be convex, i.e., in general it has a single global minimum. And to find this minimum, if we take the derivative of the risk function with respect to parameters θ_j and set to 0, we get a set of simultaneous linear equations (referred to as the Normal equations in least-squares fitting in statistics), which can be solved straightforwardly by numerical (non-gradient) methods. This numerical approach is in effect equivalent to inverting a matrix of dimension $p \times p$, with complexity $O(p^3)$. As a result, this direct approach is fine unless we have a large number of parameters p (or if our system of equations is ill-conditioned), in which case gradient methods can be much more effective computationally than trying to directly solve the p simultaneous linear equations[1].

Generalized Linear Models (GLMs): GLMs are well-known and widely-used in statistics and take the form $f(\underline{x};\underline{\theta}) \;=\; m(\underline{\theta}^T \underline{x})$ where $m()$ is a scalar-valued function[2] (often non-linear) operating on the linear inner product $\underline{\theta}^T \underline{x}$. This is a relatively simple extension of a linear model, with one parameter per input dimension (i.e., $p = d$). The most well-known example of such a model in machine learning is the logistic regression model, where $m(z) \;=\; 1/(1 + \exp(-z))$, i.e., m is the logistic function and produces a value that is bounded between 0 and 1. The use of the logistic function here can be interpreted as mapping the real-valued inner product $\underline{\theta}^T \underline{x}$ (which could take values anywhere on the real line) to a value $f \in [0, 1]$, which can be very useful for example in classification problems (with binary-valued y) where we want f to represent the conditional probability of a class variable given an input $\underline{x}$ (we will discuss this in more detail in a later set of notes on classification). Other forms of GLMs can be used to represent other constraints

[1] We will not discuss optimization methods, such as gradient techniques, in any detail in these notes: students are encouraged to review basic concepts of continuous optimization for machine learning, e.g., Chapter 7 on Continuous Optimization in the *Mathematics for Machine Learning* text.

[2] For readers familiar with GLMs, in our notation we are using $m()$ to represent the inverse of the link function $g()$, i.e., $m = g^{-1}()$.

on the values we are predicting, such as the Poisson GLM, where $f(\underline{x};\underline{\theta}) = m(\underline{\theta}^t\underline{x}) = \exp(\underline{\theta}^t\underline{x})$, which constrains f to be non-negative. In general the fitting of GLM models will require iterative algorithms: second-order methods using the Hessian (the $p \times p$ matrix of second derivatives of the loss with respect to the parameters) are often used in statistics, but for higher-dimensional problems in machine learning we typically use first-order gradient methods when fitting such models since using the Hessian in optimization requires inverting the $p \times p$ matrix, which scales as $O(p^3)$ time for every update of the parameters.

Neural Networks: These models are of course widely-used in machine learning and there is a huge literature on the many different types of neural networks. For regression modeling (i.e., predicting a real-valued y given an input $\underline{x}$) we can loosely think of a neural network as operating in two stages. In the first stage, we learn some vector representation $\underline{z}_\phi(\underline{x})$ for the input $\underline{x}$; this is often referred to as **representation learning**. For example, if the inputs $\underline{x}$ are pixels then we can think of $\underline{z}_\phi(\underline{x})$ as a new low-dimensional feature representation of the pixels, a representation parametrized by ϕ and that (in principle) eliminates information in $\underline{x}$ that is not relevant to predicting y (e.g., $\underline{z}_\phi(\underline{x})$ might be a translation-invariant representation of pixels $\underline{x}$). There are a wide variety of "architectures" for creating effective representations $\underline{z}$ (convolutional models, recurrent models, transformer models, etc) and they often use multiple hierarchical (or "deep") layers of intermediate representations to generate $\underline{z}$ representations that are good for predicting y: we won't concern ourselves with the details of how the representation learning aspects of neural networks operate (there are many other sources for learning about these methods).

In the second (output) stage of a neural network we use (in effect) a generalized linear model to transform $\underline{z}_\phi = \underline{z}_\phi(\underline{x})$ into the type of output we want, i.e.,

$$f(\underline{x};\underline{\theta}) \;=\; m\big(\,\underline{\beta}^T\underline{z}_\phi\big)$$

where $\underline{\beta}$ is a vector of parameters of the same dimension as $\underline{z}_\phi(\underline{x})$ and the full set of parameters for the model is $\underline{\theta} = \{\phi, \underline{\beta}\}$. If our output y is unconstrained then $m()$ can just be the identity function, but if (for example) $y \in [0, 1]$ then $m()$ can be a logistic function (or "softmax" as it is referred to in the neural network literature).

A key point in neural networks is that both the $\underline{\beta}$ and ϕ are learned simultaneously—in particular the parameters ϕ of the non-linear transformation $\underline{z}_\phi(\underline{x})$ are directly learned to optimize the prediction of y, e.g., using gradient methods. This tends to be much more effective in practice to "two-stage learning" where feature representations for $\underline{x}$ are pre-defined based on human intuition, e.g., using different types of manually-defined spatial filters (which up until the early 2000's was typically how image classification models were typically built).

A challenge with neural network models, compared to the much simpler linear and GLM models, is that fitting neural networks models is much more complex since $\underline{z}_\phi(\underline{x})$ is usually a complicated non-linear function and the number of parameters in ϕ can be very large. Gradient methods, particularly stochastic gradient methods, have been found empirically to be the most effective method in practice for training neural networks, with a broad range of different variants and heuristics in use. The stochastic gradient

method, which uses approximate gradients computed on sequential small batches of randomly selected examples in the training data, is particularly useful for fast training of large neural network models.

The progression of models above, from linear to GLM to neural networks is intended to give you as the reader a sense of how these different models are related to each other, acknowledging that we are skipping over many many details in how these models can be specified, particularly for neural networks. There are of course many other classes of functional forms for f that can be very useful in different contexts, including non-parametric regression models (and their Bayesian counterparts, Gaussian processes), tree-structured regression models, generalized additive models (GAMs), and many more. In what follows below, in our discussion of regression, we will largely treat our functional form $f(\underline{x};\underline{\theta})$ as some general functional form without getting into details of its specification, and instead focus on what is happening when we try to fit these models to data.

4 An Optimization View of Prediction

A useful and common way to view learning of a predictive model is to see it as an optimization problem as follows. Given data $D = \{\underline{x}_i, y_i\}$ we define **empirical risk** as:

$$R(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \Delta\big(y_i, f(\underline{x}_i;\underline{\theta})\big) \tag{1}$$

There are three key aspects to this definition

1. **Model:** The functional form of f represents our choice of model (linear, etc) and $f_i = f(\underline{x}_i;\underline{\theta})$ is the prediction of our model, with input $\underline{x}_i$ and some fixed setting of the parameter vector $\underline{\theta}$.

2. **Loss:** The loss $\Delta(y_i, f_i) \geq 0$ defines the scalar-valued loss (or error) when we predict a true y value with a predicted value $f_i = f(\underline{x}_i;\underline{\theta})$. For real-valued y we could use for example $\Delta(y, f) = (y - f)^2$ the squared error loss—but there are a variety of other loss functions we can use too.

3. **Optimization:** Once we define the functional form f and the loss Δ, and given training data $D = \{(\underline{x}_i, y_i)\}$, then "all" that remains to do is to solve an optimization problem, i.e., minimize $R(\underline{\theta})$ as a function of $\underline{\theta}$. This optimization is referred to as **empirical risk minimization**. For complex models f in machine learning this optimization is typically carried out using gradient-based methods (assuming that the loss Δ and function f are such that we can define gradients with respect to the parameters $\underline{\theta}$).

There are of course many other aspects to predictive modeling. For example, it is common to add a regularization term on the parameters to $R(\underline{\theta})$ and, as we discussed earlier, there are many many different ways we can select the functional form f (linear models, generalized linear models, neural networks, tree-structured models, and so on). Nonetheless the 3 components above capture some of the key elements in empirical predictive modeling and will typically be present in some form for any predictive modeling methodology.

A natural question to ask is why no probabilities show up in the statement of the predictive modeling problem above. From the way we have stated it, via empirical risk minimization, the problem looks entirely like an optimization problem, and there appears to be no role for probabilistic models in this setup. However, as we will see below, probability is lurking beneath the surface in an implicit manner and plays a critical role in predictive modeling.

5 A Probabilistic View of Prediction

Implicit in the formulation of the optimization problem above (as empirical risk minimization) is the assumption that our data D consists of samples $(\underline{x}_i, y_i)$ drawn in an IID manner from some underlying unknown distribution $p(\underline{x}, y)$. This implicit assumption is present by virtue of the fact that we are training a model on training data with the expectation that this data is similar distributionally to future data that the model will see, i.e., that the model will generalize well to $(\underline{x}, y)$ pairs where y is unknown and we need to predict y given $\underline{x}$.

Given this, from a probabilistic perspective we will be interested in understanding how $p(\underline{x}, y)$ affects our optimization problem. We can factor $p(\underline{x}, y)$ as $p(y|\underline{x})p(\underline{x})$. Much of our focus naturally will be on the conditional $p(y|\underline{x})$ since this characterizes, in a stochastic manner, the predictive relationship from $\underline{x}$ to y that we are interested in. In addition, as we will see later, $p(\underline{x})$ also plays an important implicit role.

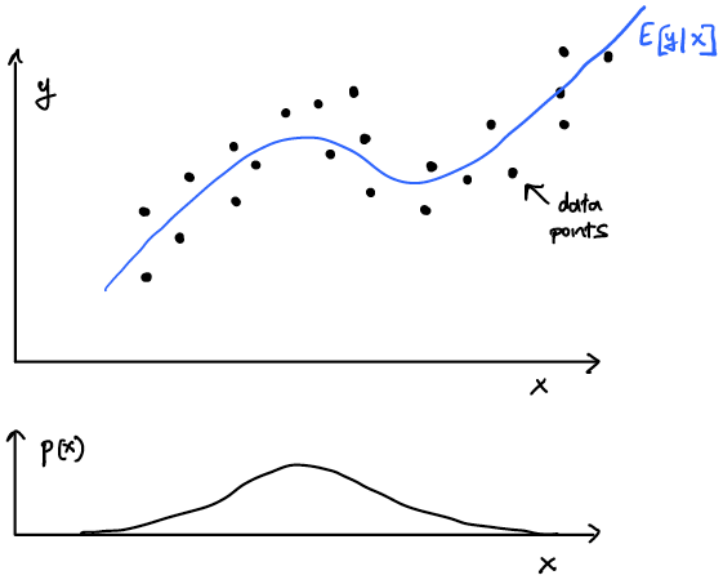

Figure 1: An illustration of regression: variation of data points y, distributed according to $p(y|x)$, centered around $E[y|x]$ at each x; and the distribution of x data points according to $p(x)$.

$p(y|\underline{x})$ reflects the stochastic dependence of y given a specific vector $\underline{x}$. You might ask "why don't we think of the relationship between y and $\underline{x}$ as being deterministic?". The reason is that in almost all real-world

prediction problems the $\underline{x}$ variables that we have access to (that we can measure and use as inputs to our model) are often not sufficient to deterministically determine the value of y. In other words, there will be factors that can affect y that will not be in our prediction model. Thus, as these unmeasured factors change, y's values will change (for a fixed $\underline{x}$) in a manner that will render y's dependence on $\underline{x}$ to appear stochastic rather than deterministic.

A simple example of this is when y is in the future, e.g,. y is the value of the stock market by the end of the week. No matter how many x variables we put into our model, there will be some uncertainty about y: there are so many possible factors that could influence y (e.g., social, political, natural events) that it is impossible to put them all in our prediction model. Another example is medical diagnosis where y is whether a particular patient has a specific disease or not: our $\underline{x}$ variables in our prediction model might be indirect measures such as test results and the only truly accurate way to determine y would be to do surgery. Yet another example is where $\underline{x}$ represents high-resolution pixel data and y is (for example) a binary variable indicating if there is a human face present or not. The relationship betwene $\underline{x}$ and y will be stochastic if the human face is too far away from the camera, or largely obscured, and so on. For these reasons it is natural to capture uncertainty in the relationship between y and $\underline{x}$ via $p(y|\underline{x})$.

5.1 Empirical Risk as Expected Loss

For simplicity of notation below we will replace vector notation $\underline{x}$ and $\underline{\theta}$ with just x and θ: but keep in mind that we are still considering x and θ to be vectors, e.g., $p(x)$ is referring to a density function over a vector x of dimension d, etc.

Let us consider what happens to the empirical risk as the size of our data set $D = (x_i, y_i), i = 1, \ldots, N$ becomes infinitely large, i.e., as $N \to \infty$.

$$\begin{aligned} \lim_{N\to\infty} R(\theta) &= \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \Delta\big(y_i, f(x_i;\theta)\big) \\ &= \int_x \int_y \Delta\big(y, f(x;\theta)\big)\; p(x,y) dy\; dx \end{aligned} \tag{2}$$

by the law of large numbers and by the fact that the empirical samples x_i, y_i are IID draws from $p(x, y)$. Thus,

$$\begin{aligned} \lim_{N\to\infty} R(\theta) &= \int_x \left(\int_y \Delta\big(y, f(x;\theta)\big)\; p(y|x) dy \right) p(x) dx \\ &= \int_x E_{p(y|x)}\big[\Delta\big(y, f(x;\theta)\big)\big]\; p(x) dx \end{aligned} \tag{3}$$

Thus, the true risk (or the true expected loss on new data), as the amount of data increases, approaches an integral where the innermost term $E_{p(y|x)}\big[\Delta\big(y, f(x;\theta)\big)\big]$ reflects the expected loss at a particular value of x, averaging over y, and the outer integral averages this expected pointwise loss over x weighted by $p(x)$. In other words, in the limit, a model that wants to minimize overall risk $R(\theta)$ should try to minimize the

pointwise risk at specific parts of the input space x, weighted by how likely those inputs are as determined by $p(x)$. For finite N, the empirical risk is an unbiased estimator of the true risk, as long as we randomly sample pairs (x_i, y_i) from the underlying joint density. But for small N, the empirical risk will have high variance—thus, empirically minimizing $R(\theta)$ as a function of θ may be minimizing a poor noisy estimate of the true risk (for small N) and the resulting model (as represented by the parameters that minimize this noisy empirical risk) might generalize poorly to new data.

5.2 Example: Squared Error Loss

Consider what happens if we define our loss function to be squared error, i.e., $\Delta\big(y, f(x;\theta)\big) = \text{MSE}_x = (y - f(x;\theta))^2$. Recall that our implicit assumption is that y is varying even when x is fixed, i.e., there is a stochastic relationship between x and y. But our deterministic predictor, $f(x;\theta)$ can only produce a deterministic prediction to predict y. So the question is "what value minimizes $(y - f(x;\theta))^2$ when y is stochastic?" Treating $f_x = f(x;\theta)$ as some unknown constant whose value we wish to optimally select, we seek to minimize

$$\begin{aligned}
\text{MSE}_x &= \int_y (y - f_x)^2 p(y|x) dy \\
&= \int_y (y^2 - 2f_x y + f_x^2) p(y|x) dy \\
&= \int_y y^2 p(y|x) dy - 2f_x \int_y y\, p(y|x) dy + f_x^2 \int_y p(y|x) dy
\end{aligned} \tag{4}$$

Noting that $\int_y p(y|x)dy = 1$, and denoting $\int_y yp(y|x)dy$ as $E[y|x]$, taking derivatives wrt f_x and setting to zero we arrive at the well-known result that the minimizing value is $f_x = E[y|x]$, i.e., to minimize the squared error when predicting a random variable (which here is y given x), predict the mean of the random variable.

Another way to look at this is to rewrite the mean-squared error at x as:

$$\begin{aligned}
\text{MSE}_x &= \int_y (y - f_x)^2 p(y|x) dy \\
&= \int_y (y - E[y|x] + E[y|x] - f_x)^2\, p(y|x) dy \\
&= \int_y (y - E[y|x])^2 p(y|x) dy + \int_y (E[y|x] - f_x)^2\, p(y|x) dy + \text{cross terms} \\
&= \int_y (y - E[y|x])^2 p(y|x) dy + \int_y (E[y|x] - f_x)^2\, p(y|x) dy
\end{aligned} \tag{5}$$

where the cross-terms cancel out (to verify this, note that both f_x and $E[y|x]$ can be treated as constants that can be taken outside the integral). Thus, defining $\sigma^2_{y|x} = \int_y (y - E[y|x])^2 p(y|x) dy$ as the conditional variance of y at x, we have

$$\begin{aligned}
\int_y (y - f_x)^2 p(y|x) dy &= \sigma^2_{y|x} + \int_y (E[y|x] - f_x)^2 p(y|x) dy \\
&= \sigma^2_{y|x} + (E[y|x] - f_x)^2
\end{aligned} \tag{6}$$

since, again, $E[y|x]$ and f_x can be taken outside the integral over y. From the last line of the equation above we see that the mean-squared error at x is composed of two terms:

1. $\sigma^2_{y|x}$: this is the intrinsic variance of y at x. We have no direct control over this, it effectively represents the variation in y that cannot be explained by x, and is a lower bound on the optimal achievable mean square error at x.

2. $(E[y|x] - f_x)^2$: this term can in principle be minimized to zero by setting our prediction f_x, at x, to the expected value of y at x, i.e., set $f_x = E[y|x]$.

Of course in practice we don't know what $E[y|x]$ is; instead we use samples from $p(x, y)$ to minimize the empirical risk (or empirical mean-squared error in this case).

Furthermore, we need to minimize the mean-squared error not just at some particular x, but we need to minimize the total expected squared error with respect to $p(x)$, i.e., minimize

$$\begin{aligned}
\text{MSE}_x &= E_{p(x)}\left[\int_y (y - f_x)^2 \, p(y|x)dy\right] \\
&= \int_x \int_y (y - f_x)^2 \, p(y|x)dy \, p(x)dx \\
&= \int_x \left(\int_y (y - E[y|x])^2 \, p(y|x)dy + (E[y|x] - f_x)^2\right) p(x)dx \\
&= \sigma^2_{y_x} + E_{p(x)}[(E[y|x] - f_x)^2] \qquad (7)
\end{aligned}$$

where each term is now averaged with respect to $p(x)$. The first term is defined as $\sigma^2_{y_x} = E_{p(x)}[\sigma^2_{y|x}]$, which is the expected irreducible variance in y relative to $E[y|x]$. The second term is the average error between the optimal prediction $E[y|x]$ at each x and the actual prediction f_x. As with the pointwise case, we can in theory minimize this by selecting $f_x = E[y|x]$ for all values of x, which will set the second term to 0. However, in practice our function f_x might not be representationally powerful enough to approximate $E[y|x]$ at all values of x. For example, imagine if the true (unknown) $E[y|x]$ is a 2nd order (or higher) polynomial in x and we have selected a linear function of x as our prediction model f_x. This type of error is known as **approximation error** or **bias**. What will tend to happen in this situation, is that to minimize $E_{p(x)}[E[y|x] - f_x)^2]$ the optimization will tend to focus on trying to match our predictor f_x to the unknown mean $E[y|x]$ as best it can, emphasizing regions of high input density in the x space.

To try to avoid approximation error, we could make our function f_x be much richer representationally (i.e., a more complex function) to try to match the potentially complex (but unknown) $E[y|x]$ function we are trying to learn. For example, we could use high-order polynomial functions of x (or polynomial functions of the components of x if it is a vector) or we could use a flexible model such as neural network. The potential problem we will run into by doing this is that, for a finite amount of data, as make our predictor f_x more complex, we will in general increase the **estimation error** or **variance**. For example, lets say that f_x is complex enough to model $E[y|x]$ exactly at all values of x. Even if this is the case, we still have to estimate $E[y|x]$ as a function of x from noisy observations sampled from $p(x, y)$, and in general this process will

result in non-zero estimation error—this estimation error can be quite large if we have relatively complex functions and relatively small amounts data to fit them with.

In the next section we will see that there is a fundamental tradeoff between approximation error (bias) and estimation error (variance).

6 The Bias-Variance Tradeoff in Regression

Let D be a data set of size N, i.e., $D = \{(x_i, y_i)\}, i = 1, \ldots, N$ where each x_i, y_i pair is an IID sample from some underlying unknown distribution $p(x, y)$. Let $p(D)$ be the distribution over all possible data sets of size N. For example if x and y were both discrete then $p(D)$ would be a distribution over a finite set of possible data sets—otherwise $P(D)$ will be a density function.

We will now analyze the average performance of fitting a model with respect to $p(D)$. This analysis is inherently theoretical in nature since in practice we are usually conditioning on a particular data set D when fitting a model—but we gain insight by analyzing what would happen on average if (say) many different individuals fit models using different data sets D with samples of size N drawn IID from the same underlying process $p(x, y)$.

Note that $f(x; \hat{\theta})$ is a random quantity w.r.t. $p(D)$, since $\hat{\theta}$ is an estimate of θ computed from D (e.g., $\hat{\theta}$ will be the result of some optimization procedure given D), Since D is random then $\hat{\theta}$, which depends on D, is random w.r.t $p(D)$. For different data sets D, we get different $\hat{\theta}$'s, which give different f's.

Recall that $\text{MSE}_x = \sigma^2_{y|x} + (E[y_x] - f(x; \hat{\theta}))^2$. Since the first part, $\sigma^2_{y|x}$ doesn't depend on the data D, we will analyze the expectation of the second part with respect to D: $E_{p(D)}[(E[y_x] - f(x; \hat{\theta}))^2]$. This is the component of our error at x that we can affect (by our choice of f and θ, here averaged over all possible data sets D of size N.

It will be convenient to define $\bar{f}_x = E_{p(D)}[f(x; \hat{\theta})]$. This can be thought of as the prediction we would make at x if we averaged over many different data sets D in estimating $f(x; \hat{\theta})$. We can also define the pointwise bias as $\text{Bias}_x = E[y_x] - \bar{f}_x$ which is the bias between the true $E[y|x]$ (our optimal prediction for squared error) and the average of our predictions.

Given these definitions we have:

$$\begin{aligned} E_{p(D)}[(E[y_x] - f(x; \hat{\theta}))^2] &= E_{p(D)}[(E[y_x] - \bar{f}_x + \bar{f}_x - f(x; \hat{\theta}))^2] \\ &= E_{p(D)}[(E[y_x] - \bar{f}_x)^2] + E_{p(D)}[(\bar{f}_x - f(x; \hat{\theta})^2)] \\ &= (E[y_x] - \bar{f}_x)^2 + E_{p(D)}[(\bar{f}_x - f(x; \hat{\theta})^2)] \end{aligned} \quad (8)$$

where in the 2nd last line the cross-terms cancel out again and in the last line we can drop the $E_{p(D)}$ (in the first term) since the terms $E[y_x]$ and $\bar{f}_x$ are constants with respect to the expectation over $p(D)$.

Consider both terms on the right-hand side above:

- **Bias:** $\text{Bias}_x^2 = (E[y_x] - \bar{f}_x)^2$. Note that bias is inherently a theoretical concept: in practice we won't be able to measure bias (except in simulated data) since we don't in practice know what the true $E[y_x]$ is.
- **Variance:** $\text{Variance}_x = E_{p(D)}[(\bar{f}_x - f(x;\hat{\theta})^2)]$. This term represents the variability in fitting f_x (i.e., the estimation error) due the variability in different data sets of size N. This could in principle be estimated in practice (e.g., via bootstrap methods).

We can average over $p(x)$ (in addition to averaging over $P(D)$) to define the total bias and variance as $\text{Bias}^2 = \int_x \text{Bias}_x^2\, p(x)dx$ and $\text{Variance} = \int_x \text{Variance}_x\, p(x)dx$. And finally, we can add back in the inherent variance in y (from earlier), to get the average MSE averaged over data sets D of size N for some functional form f as a predictive model:

$$\text{MSE} = \sigma_{y_x}^2 + \text{Bias}^2 + \text{Variance}$$

This is the fundamental bias-variance tradeoff for fitting predictive models. This is referred to as a tradeoff since bias and variance tend to act in opposite directions in practice:

- Simple models with few parameters can have high bias, but will tend to have low variance.
- Complex models with many parameters will have lower bias, but can have high variance when fit to relatively small amounts of data.

The bias-variance result we derived above is based on the squared-error loss. Somewhat analogous results can be obtained for other types of loss functions.

The classical bias-variance theory tends to suggest that as the complexity of a model increases (e.g., as the number of hidden units in a neural network increases) that bias decreases monotonically while variance increases monotonically. However, there has been some interesting recent work in deep learning that shows that (in certain cases) that variance at first increases with the complexity of a model but can then decrease. The precise reasons for this decrease are still not clear theoretically but it is conjectured that variance can in fact decrease once the complexity of a model reaches the over-parametrized regime (e.g., when a model is complex enough to completely interpolate a data set), possibly due to various regularization effects in training deep models. This is currently a hot research topic in machine learning; students interested in additional perspectives can refer to the chapter on Generalization in Hardt and Recht's text `https://mlstory.org/generalization.html`, as well as papers such as "Rethinking bias-variance trade-off for generalization of neural networks", ICML 2020 `http://proceedings.mlr.press/v119/yang20j/yang20j.pdf` and "Understanding double descent requires a fine-grained bias-variance decomposition", NeurIPS 2020 `https://proceedings.neurips.cc/paper/2020/hash/7d420e2b2939762031eed0447a` `html`.

7 Conditional Probabilistic Models for Regression

It is natural to think of how we might connect likelihood-based and Bayesian estimation ideas to regression. We begin by defining the **conditional likelihood** as $L(\underline{\theta}) = P(D_y|D_{\underline{x}}, \underline{\theta})$ where we have split the data pairs

in the data D into $D_y = \{y_i\}$ and $D_{\underline{x}} = \{\underline{x}_i\}$, $i = 1, \ldots, N$. We use a conditional likelihood since we are interested in modeling $p(y|x)$ rather than $p(x)$. Under an IID assumption, we have

$$L(\underline{\theta}) = P(D_y|D_{\underline{x}}, \underline{\theta}) = \prod_{i=1}^{N} p(y_i|\underline{x}_i, \underline{\theta}) \tag{9}$$

To model $p(y|\underline{x}, \underline{\theta})$ a natural approach is to model the mean of y as some function of $\underline{x}$ (see Figure 1) with some additive noise. For example, assuming Gaussian noise we can specify a conditional Gaussian model as

$$p(y_i|\underline{x}_i, \underline{\theta}) = N(f(\underline{x}; \underline{\theta}), \sigma_y^2) \propto \exp^{-\frac{1}{2\sigma_y^2}(y_i - f(\underline{x}_i; \underline{\theta}))^2} \tag{10}$$

where for convenience we will assume (i) that the noise is Gaussian, and (ii) that σ_y^2 is known and constant (e.g., does not vary with $\underline{x}$). Note that unlike in density estimation, the mean here is not a constant but instead varies as an unknown function of the inputs $\underline{x}$.

Writing out the log-likelihood and ignoring terms that don't depend on $\underline{\theta}$, we get

$$\log L(\underline{\theta}) = -\sum_{i=1}^{N} \Big(y_i - f(\underline{x}_i; \underline{\theta}) \Big)^2 \tag{11}$$

We note that this is (within a constant N) the same definition as the negative empirical risk with mean-squared error as the loss. Thus, *minimizing squared error in regression is equivalent to maximizing the conditional log-likelihood under an additive IID Gaussian noise assumption*. This is not surprising given the functional form of the Gaussian model.

This connection between empirical risk and log-likelihood opens up a range of modeling possibilities. In particular this allows us to design empirical risk and loss functions in a principled way, by defining an appropriate log-likelihoods. For example, we if we wanted our noise σ_y^2 to depend on $\underline{x}$, we could build this into the likelihood, e.g, by specifying $\sigma_y^2(\underline{x}) = h(\underline{x}; \gamma)$ where h is some functional form (e.g., linear) with unknown parameters γ, with a resulting likelihood $L(\underline{\theta}, \gamma)$. Or we could specify a model with non-Gaussian (e.g., asymmetric) noise if that were more appropriate for our problem rather than symmetric Gaussian noise. We also notice for example that the additive form of the standard empirical risk (with a sum over the N data points) in fact corresponds directly to an IID assumption in defining the likelihood. If we wanted to generalize to allow for dependence across the $\underline{x}_i, y_i$ pairs (e.g., if i were an index for time) then we can in principle write down a likelihood model that includes dependence for the data points (such as a Markov or autoregressive model) and take the log to obtain a correspond empirical risk function that we can minimize.

8 Bayesian Regression

The likelihood-based approach above allows us to make another very useful connection between regression and topics we discussed earlier in the course, i.e., we can formulate Bayesian approaches to regression. As an example, consider the conditional Gaussian likelihood discussed in the last section but where we now have a prior $p(\underline{\theta})$ on the parameters $\theta_1, \ldots, \theta_p$. For convenience we will assume that this is an independent

prior on each individual parameter, i.e., $p(\underline{\theta}) = \prod_{j=1}^{p} p(\theta_j)$. A common approach for example is to assume a relatively non-informative Gaussian prior for with $p(\theta_j) = N(0, \sigma_\theta^2)$ with zero mean. The role of σ_θ^2 essentially governs how strongly we believe that any of the parameters (e.g., weights or coefficients in a linear model) are different from 0. In a high-dimensional linear problem (where $p = d$ and d is large) we might believe that many of the inputs in the input vector $\underline{x}$ are not in fact relevant to predicting y and that their coefficients are likely to be 0. Of course we don't know a priori which ones should be zero and which should be non-zero, so we can let the data (via the likelihood) nudge us away from the prior which has its mode at 0.

Proceeding with a Bayesian analysis, and using the conditional Gaussian likelihood from the previous section, we can define the posterior on the unknown parameters $\underline{\theta}$ as

$$p(\underline{\theta}|D_y, D_x) \propto p(D_y|D_x, \underline{\theta})p(\underline{\theta}) = \prod_{i=1}^{N} N(f(\underline{x}; \underline{\theta}), \sigma_y^2) \prod_{j=1}^{p} N(0, \sigma_\theta^2) \tag{12}$$

where $p(\underline{\theta}|D_x) = p(\underline{\theta})$ since the priors on $\underline{\theta}$ don't depend on the data D_x. Taking logs to convert to the log-posterior, we get

$$\log p(\underline{\theta}|D_y, D_x) = -\frac{1}{2\sigma_y^2} \sum_{i=1}^{N} \Big(y_i - -f(\underline{x}_i; \underline{\theta}) \Big)^2 - \frac{1}{2\sigma_\theta^2} \sum_{i=1}^{N} \theta_j^2 \tag{13}$$

which, after negating and multiplying through by various constants (relative to $\underline{\theta}$) we can rewrite as

$$R'(\underline{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \Big(y_i - -f(\underline{x}_i; \underline{\theta}) \Big)^2 + \lambda \sum_{i=1}^{N} \theta_j^2 \tag{14}$$

with $\lambda = \frac{\sigma_y^2}{N\sigma_\theta^2}$. We recognize $R'(\underline{\theta})$ as a standard empirical risk function for squared error loss (the first term) but now with an additional second "L2-regularization" term that penalizes deviations of the θ_j parameters away from 0 where the penalty is in the form of squared error. In other words, what we have shown above is that *maximum a posteriori estimation for Bayesian regression, i.e., maximizing* $\log p(\underline{\theta}|D_y, D_x)$*, is equivalent to minimizing an empirical risk function with a regularization term where the regularization corresponds to a log prior on the parameters* $\underline{\theta}$.

This is an interesting and useful connection, allowing us to bridge the more empirically-oriented regularization world of predictive modeling with a Bayesian approach to regression. In some cases the two approaches are exactly equivalent (particularly for point estimates, as we will discuss below), just two different formalizations of the same problem. However, the ability to view regularization from a Bayesian perspective opens up potentially new ideas and approaches. As an example, if we wanted to impose additional structure in our regularization, we could do this by using joint priors with covariance structure on our parameters $\underline{\theta}$, e.g., where whole groups of parameters are either all zero or all non-zero. We could also look at non-Gaussian priors: for example the well-known Lasso technique for regularization, where the penalty term is of the form $\sum_{j=1}^{p} |\theta_p|$ corresponds to a Laplacian prior (two-sided exponential around 0), a prior that tends to decay faster than the Gaussian and that can lead to sparser solutions in the sense that the fitted

models (after minimization) will tend to have more parameters that are exactly 0 than if we had used a Gaussian prior.

We also note above that the Bayesian framework tells us what the value of λ should be, namely a ratio of data variance σ_y^2 divided by N to prior variance σ_θ^2. In practice λ is often determined empirically (e.g., by searching for the best value using a validation set), but for very small data sets where we have a priori knowledge about both the noise variance σ^2 in y and the prior uncertainty about our parameters θ_p, the ability to select $\lambda = \frac{\sigma_y^2}{N\sigma_\theta^2}$ could be useful.

The discussion above focuses just on MAP point estimation of parameters $\underline{\theta}$. To be fully Bayesian we would like to marginalize over parameter uncertainty and make predictions for a new y in this manner. Specifically, our distribution for y given some future $\underline{x}$ should be

$$\begin{aligned} p(y|\underline{x}, D_x, D_y) &= \int_{\underline{\theta}} p(y, \underline{\theta}|\underline{x}, D_x, D_y) d\underline{\theta} \\ &= \int_{\underline{\theta}} p(y|\underline{x}; \underline{\theta}) p(\underline{\theta}|D_x, D_y) d\underline{\theta} \\ &\propto \int_{\underline{\theta}} p(y|\underline{x}; \underline{\theta}) p(D_y|D_x, \underline{\theta}) p(\underline{\theta}) d\underline{\theta} \end{aligned} \tag{15}$$

From the second line above we see that the Bayesian formula for prediction in regression is to average over predictions with specific settings, $p(y|\underline{x}; \underline{\theta})$, weighted by the posterior on $\underline{\theta}$. If we don't want to compute a full predictive density for y, but just want to generate a point estimate for y, we could use the mean or mode of the predictive density $p(y|\underline{x}, D_x, D_y)$.

For relatively simple cases, such as linear models with Gaussian likelihoods and Gaussian priors, we can get closed-form solutions for both the posterior $p(\underline{\theta}|D_x, D_y)$ and the predictive density $p(y|\underline{x}, D_x, D_y)$ for regression. But generally (even for relatively simple models like GLMs) neither the posteriors nor the predictive densities are available in closed form, which necessitates the use of *approximate Bayesian inference methods*, typically either stochastic approaches (such as Markov Chain Monte Carlo (MCMC)) or optimization-based approaches that approximate the posterior with simpler forms (such as variational inference). Developing efficient and robust Bayesian methods for large-scale deep neural networks, that can contain potentially millions of parameters, is still considered an open research problem, e.g., see Wilson and Izmailov (2020), "Bayesian deep learning and a probabilistic perspective on generalization," `https://arxiv.org/pdf/2002.08791.pdf`.

Additional Reading on Regression

See the class Web page for links to the texts below, all are available online.

- **Empirical Risk Minimization**: Sections 8.1 and 8.2 in Mathematics for Machine Learning (MML) (2021).

- **Linear Regression**: Chapter 9 in MML (2021) and Chapter 11 in Murphy (2022).
- **GLMs**: Chapter 8 in Efron and Hastie, and Chapter 12 in Murphy (2022)
- **(Deep) Neural Networks**: Chapter 18 in Efron and Hastie for an informative brief review of neural network concepts from a statistical perspective, and Chapter 13 in Murphy (2022) for a more detailed treatment.
- **Bias-Variance:** a very nice blog post that provides great intuition: `http://scott.fortmann-roe.com/docs/BiasVariance.html`.
- **Bayesian Regression:** See in particular Section 9.3 in MML and Section 11.6 in Murphy for detailed treatments of Bayesian linear regression