Least Squares Data Fitting (VMLS Ch. 13)

We will introduce one of the most important applications of least squares methods: fitting a mathematical model of some relation given some observed data.

A typical data fitting problem takes the following form. There is some underlying feature vector or independent variable $x \in \mathbb{R}^n$, and a scalar outcome or response variable that we believe are (approximately) related by some function $f: \mathbb{R}^n \to \mathbb{R}$, such that

$$y \approx f(x). \quad (M)$$

Data: Our goal is to fit (or "learn") a model f given some data:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)}).$$

These data pairs $(x^{(i)}, y^{(i)})$ are sometimes also called observations, examples, samples, or measurements depending on context.

NOTE: The superscript $^{(i)}$ denotes the i^{th} data point. For example, $x^{(i)} \in \mathbb{R}^n$ is the i^{th} independent variable, and the number $x^{(i)}_j$ is the value of the j^{th} feature for example i.

Model parameterization: Our goal is to choose a model $\hat{f}: \mathbb{R}^n \to \mathbb{R}$ that approximates the model (M) well, i.e., such that $y \approx \hat{f}(x)$. The hat notation is traditionally used to highlight that $\hat{f}$ is an approximation to f. Similarly, we will write $\hat{y} = \hat{f}(x)$ to highlight that $\hat{y}$ is an approximate prediction of the outcome y.

In order to efficiently search over candidate model functions $\hat{f}$, we need to parameterize a model class $\mathcal{F}$ that is easy to work with. A powerful and commonly used model class is the set of linear in the parameters models of the form

$$\hat{f}(x) = \theta_1 f_1(x) + \theta_2 f_2(x) + \cdots + \theta_p f_p(x). \quad (LP)$$

In (LP), the functions $f_i: \mathbb{R}^n \to \mathbb{R}$ are basis functions or features that we choose before hand. Note that the term basis here is related to, but different from, our previous use of the term. When we solve the data fitting problem, we will look for the parameters θ_i that, among other things, make the model prediction $\hat{y}^{(i)} = \hat{f}(x^{(i)})$ consistent with the observed data, i.e., we want $\hat{y}^{(i)} \approx y^{(i)}$.

Data fitting: For the i^{th} observation $y^{(i)}$ and the i^{th} prediction $\hat{y}^{(i)}$, we define the prediction error or residual $r^{(i)} = \hat{y}^{(i)} - y^{(i)}$.

The least squares data fitting problem chooses the model parameters θ_i that minimize the (average of the) sum of the square of the prediction errors on the data set:

$$\frac{(r^{(1)})^2 + \cdots + (r^{(N)})^2}{N}.$$

Next we'll show that this problem can be cast as a least squares problem over the model parameters θ_i. Before doing that though, we pause to highlight the conceptual shift we are making.

DATA DRIVEN: Rather than hand crafting our function $\hat{f}$ from scratch, we solve an optimization problem to identify the parameters θ_i that best explain the data, i.e., we learn the model from the data. Of course, if we know something about the model structure, we should encode this in our choice of feature functions f_i. We'll see examples of such feature engineering later.

Data fitting as least squares

We start by stacking the outcomes $y^{(i)}$, predictions $\hat{y}^{(i)}$, and residuals $r^{(i)}$ as vectors in $\mathbb{R}^N$:

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix}, \quad \hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(N)} \end{bmatrix}, \quad r = \begin{bmatrix} r^{(1)} \\ r^{(2)} \\ \vdots \\ r^{(N)} \end{bmatrix} = \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \vdots \\ \hat{y}^{(N)} - y^{(N)} \end{bmatrix}$$

Then we can compactly write the squared prediction error as $\|r\|^2$. Next, we compile our model parameters into a vector $\theta \in \mathbb{R}^p$, and build our feature matrix or measurement matrix $A \in \mathbb{R}^{N \times p}$ by setting

$$A_{ij} = f_j(x^{(i)}), \quad i = 1, \ldots, N, \quad j = 1, \ldots, p.$$

The j^{th} column of the matrix A is composed of the j^{th} basis function evaluated on each of the data points $x^{(1)}, \ldots, x^{(N)}$:

$$f_1(x) = \begin{bmatrix} f_1(x^{(1)}) \\ f_1(x^{(2)}) \\ \vdots \\ f_1(x^{(N)}) \end{bmatrix}, \quad \cdots, \quad f_p(x) = \begin{bmatrix} f_p(x^{(1)}) \\ f_p(x^{(2)}) \\ \vdots \\ f_p(x^{(N)}) \end{bmatrix}$$

and $A = [f_1(x) \cdots f_p(x)]$. In matrix-vector notation, we then have

$$\hat{y} = A\theta = \theta_1 f_1(x) + \cdots + \theta_p f_p(x).$$

The least squares data fitting problem then becomes to

$$\text{minimize } \|\underline{\Lambda}\|^2 \quad (\Rightarrow \text{minimize } \|\underline{y} - A\underline{\theta}\|^2$$

over the model parameters $\underline{\theta}$, which we recognize as a least squares problem! Assuming we have chosen basis functions f_j such that the columns of A are linearly independent (what would it mean if this weren't true?), we have that the least squares solution is

$$\hat{\underline{\theta}} = (A^T A)^{-1} A^T \underline{y}.$$

The resulting average least squares error $\frac{\|A\hat{\theta} - \underline{y}\|^2}{N}$ is called the Minimum Mean-Square Error (MMSE).

Warm up: Fitting a Constant Model:

We start with the simplest possible model and set the number of features $p = 1$ and $f_1(x) = 1$, so that our (admittedly boring) model becomes $\hat{f}(x) = \theta_1$.

First, we construct $A \in \mathbb{R}^{N \times 1}$ by setting $A_{i1} = f_1(x^{(i)}) = 1$. Therefore A is the N-dimensional all ones vector $\underline{1}_N$. We plug this into our formula for $\hat{\underline{\theta}}$:

$$\hat{\underline{\theta}} = \hat{\theta}_1 = (\underline{1}^T \underline{1})^{-1} \underline{1}^T \underline{y} = \frac{1}{N} \sum_{i=1}^{N} y^{(i)} = \text{average}(\underline{y}).$$

We have just shown that the mean or average of the outcomes $y^{(1)}, \ldots, y^{(N)}$ is the best least squares fit of a constant model. In this case, the MMSE is

$$\frac{1}{N} \sum_{i=1}^{N} (\text{average}(\underline{y}) - y^{(i)})^2,$$

which is called the variance of $\underline{y}$, and measures how "wiggly" $\underline{y}$ is.

Univariate Functions: Straight Line Fit

We start by considering the univariate function setting where our feature vector $\underline{x} = x \in \mathbb{R}$ is a scalar, and hence we are looking to approximate a function $f: \mathbb{R} \to \mathbb{R}$. This is a nice way to get intuition because it is easy to plot the data $(x^{(i)}, y^{(i)})$ and the model function $\hat{y} = \hat{f}(x)$.

We'll start with a straight line fit model: we set $p = 2$, with $f_1(x) = 1$ and $f_2(x) = x$. In this case our model class is composed of models of the form

$$\hat{f}(x) = \theta_1 + \theta_2 x.$$

Here, we can easily interpret θ_1 as the y-intercept and θ_2 as the slope of the straight line model we are searching for.

In this case, the matrix $A \in \mathbb{R}^{N \times 2}$, and takes the form

$$A = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \vdots & \vdots \\ 1 & x^{(N)} \end{bmatrix}$$

Although we can work out formulas for $\hat{\theta}_1$ and $\hat{\theta}_2$, they are not particularly interesting or informative. Instead, we'll focus on some examples of how to use these ideas.

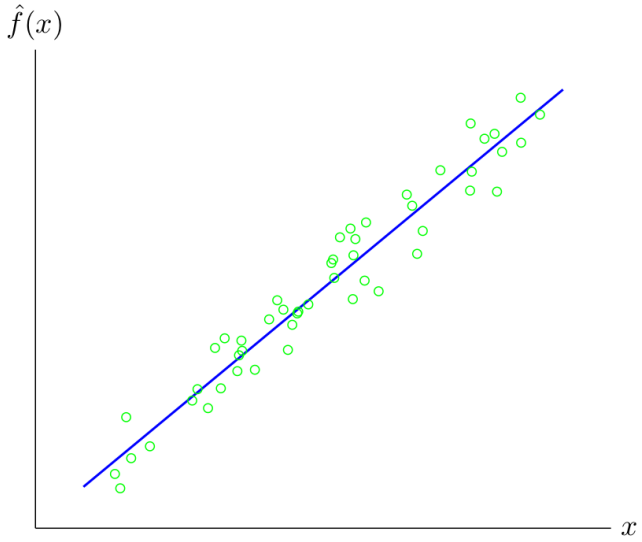

Figure 13.2 Straight-line fit to 50 points $(x^{(i)}, y^{(i)})$ in a plane.

Example: Time Series Trend

In this setting, $y^{(i)}$ is the value of a quantity of interest at time $x^{(i)} = i$. The straight line model $\hat{y}^{(i)} = \hat{\theta}_1 + \hat{\theta}_2 i$ is called a trend line, and $y - \hat{y}$ is called the de-trended time series, and $\hat{\theta}_2$ is the trend coefficient.

When the de-trended time series is positive, it means the time series lies above the straight-line fit; when it is negative, it is below the straight-line fit. In the figures below, we apply this idea to world petroleum consumption. Can you identify when major geopolitical events occurred based on the de-trended line?

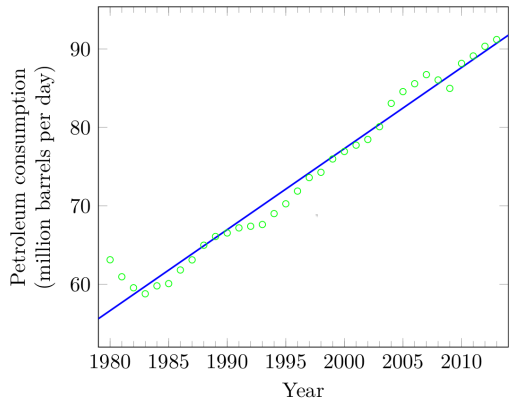

Figure 13.3 World petroleum consumption between 1980 and 2013 (dots) and least squares straight-line fit (data from www.eia.gov).

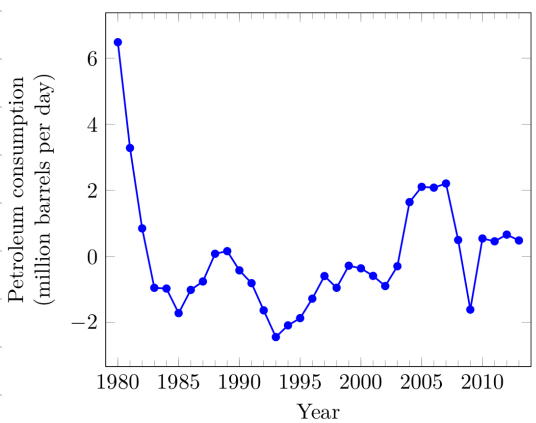

Figure 13.4 De-trended world petroleum consumption.

Univariate Functions: Polynomial Fit

A simple extension beyond the straight-line fit is a polynomial fit where we set the j^{th} feature to be

$$f_j(x) = x^{j-1}$$

for $j = 1, \ldots, p$. This leads to a model class composed of polynomials of at most degree $p-1$:

$$\hat{f}(x) = \theta_1 + \theta_2 x + \theta_3 x^2 + \cdots + \theta_p x^{p-1}.$$

CAUTION: Here x^i means a generic scalar raised to the i^{th} power; $x^{(i)}$ means the i^{th} observed scalar data value.

In this case, our matrix $A \in \mathbb{R}^{N \times p}$ and takes the form

$$A = \begin{bmatrix} 1 & x^{(1)} & \cdots & (x^{(1)})^{p-1} \\ 1 & x^{(2)} & \cdots & (x^{(2)})^{p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x^{(N)} & \cdots & (x^{(N)})^{p-1} \end{bmatrix},$$

which you might recognize as a Vandermonde Matrix, which we encountered earlier in the class when considering polynomial interpolation. An important property of such matrices is that their columns are linearly independent provided that the numbers $x^{(1)}, \ldots, x^{(N)}$ include at least p different values. The figures below show examples of least squares fits of polynomials of degree 2, 6, 10, and 15 to a set of 100 data points.

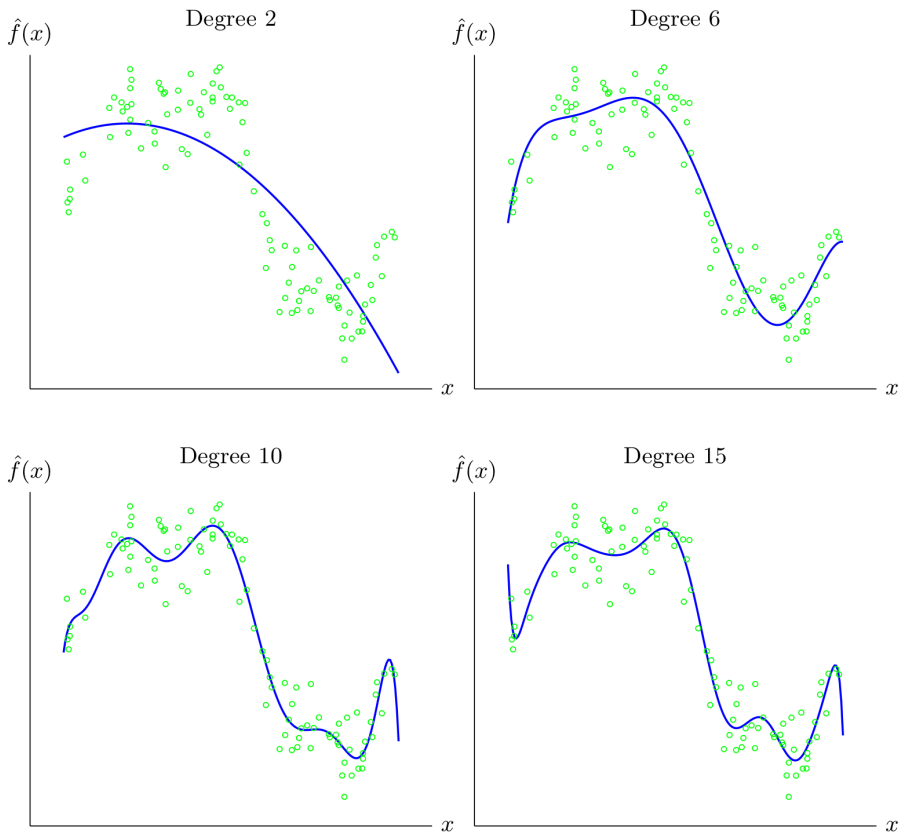

Figure 13.6 Least squares polynomial fits of degree 2, 6, 10, and 15 to 100 points.

An important observation is that since any polynomial of degree less than r is also one of degree less than s if $r \le s$, it follows that the MMSE will decrease as we make the polynomial degree larger. This suggests that we should use the largest degree polynomial possible so as to get a model with the smallest MMSE possible. We'll see later that this is NOT TRUE, and you'll explore methods for model selection in recitation and the homework.

Regression Models

We now consider the setting of vector-valued independent variables $\underline{x} \in \mathbb{R}^n$. The analog of a straight-line fit here is a linear regression model of the form

$$\hat{y} = \hat{f}(\underline{x}) = \underline{\beta}^T \underline{x} + v,$$

where $\underline{\beta} \in \mathbb{R}^n$ and $v \in \mathbb{R}$. If we set $\underline{\theta} = \begin{bmatrix} v \\ \underline{\beta} \end{bmatrix}$, then the model becomes

$$\hat{y} = \theta_1 + \theta_2 x + \cdots + \theta_{n+1} x_n.$$

We can view this as fitting within our general linear in the parameters model by setting $f_1(\underline{x}) = 1$ and $f_i(\underline{x}) = x_{i-1}$ for $i = 2, \ldots, n+1$, so that $p = n+2$.

We are of course not obliged to use these features. Instead, suppose that we have $p-1$ features $f_2(\underline{x}), \ldots, f_p(\underline{x})$, and assume we have set $f_1(\underline{x}) = 1$, as is commonly done. If we define

$$\underline{\tilde{x}} = \begin{bmatrix} f_2(\underline{x}) \\ \vdots \\ f_p(\underline{x}) \end{bmatrix} \in \mathbb{R}^{p-1}$$

we can write a linear regression model in the new feature vector $\underline{\tilde{x}}$:

$$\hat{y} = \theta_1 f_1(\underline{x}) + \cdots + \theta_p f_p(\underline{x}) = \underline{\beta}^T \underline{\tilde{x}} + v$$

where:
- $\underline{\tilde{x}} = (f_2(\underline{x}), \ldots, f_p(\underline{x}))$ are the transformed features
- $v = \theta_1$ is called the affine term
- $\underline{\beta} = (\theta_2, \theta_3, \ldots, \theta_p)$ is the linear term

Application: Auto-Regressive Time Series Modeling

Here is a very widely used application of the above ideas in the context of time-series forecasting. Our goal here is to fit a model that predicts elements of a time-series $z_1, z_2, \ldots$, where $z_t \in \mathbb{R}$ is a scalar quantity of interest.

A standard approach is to use an auto-regressive (AR) prediction model:

$$\hat{z}_{t+1} = \theta_1 z_t + \cdots + \theta_M z_{t-M+1}, \quad t = M, M+1, \ldots. \qquad (AR)$$

In equation (AR), the parameter M is the memory of the model, and $\hat{z}_{t+1}$ is the prediction of the next value based on the previous M observations. We will choose $\underline{\theta} \in \mathbb{R}^M$ to minimize the sum of squares of prediction errors:

$$(\hat{z}_{M+1} - z_{M+1})^2 + \cdots + (\hat{z}_T - z_T)^2$$

We can fit this within our regression model framework by setting $\underline{\beta} = \underline{\theta}$, and

$$y^{(i)} = z_{M+i}, \quad \underline{x}^{(i)} = \begin{bmatrix} z_{M+i-1} \\ z_{M+i-2} \\ \vdots \\ z_i \end{bmatrix} \in \mathbb{R}^M, \quad i = 1, \ldots, T-M.$$

A little bit of bookkeeping allows us to conclude that we have $N = T-M$ examples and $p = M$ features.

The online notes will show you an example of applying these ideas to predicting temperatures at LAX.

Model Selection, Generalization, and Validation

This section is entirely practical: these are standard "tricks of the trade" that you will revisit in more detail in more advanced classes on statistics and machine learning.

Our starting point is a philosophical question: what is the goal of a learned model? Perhaps surprisingly, it is NOT TO PREDICT OUTCOMES FOR THE GIVEN DATA; after all, we already have this data! Instead, we want to predict the outcome on new, unseen data.

If a model makes reasonable predictions on new unseen data, it is said to generalize. On the other hand, a model that makes poor predictions on new unseen data, but predicts the given data well, is said to be over-fit.

A simple but effective method to guess if a model will generalize is called validation. The idea is to split your original data into a training set and a test set. Typical splits used in practice are 80% / 20% and 90% / 10%.

Then, we only use the training data to fit (or "train") our model, and then evaluate the model's performance on the test set. If the prediction errors on the training and test sets are similar, then we guess the model will generalize. This is rarely guaranteed, but such a comparison is often predictive of a model's generalization properties.

Validation is often used for **Model Selection**, i.e., to choose among different candidate models. For example, by comparing train/test errors, we can select between:

- Polynomial models of different degrees.
- Regression models with different sets of features
- AR models with different memories.

Example: Models are fit using a training set of 100 points, and plots show test set of 100 (green) points.

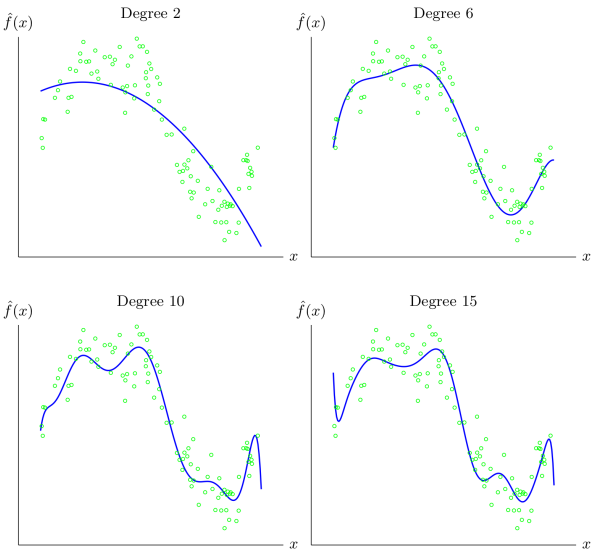

Figure 13.10 The polynomial fits of figure 13.6 evaluated on a test set of 100 points.

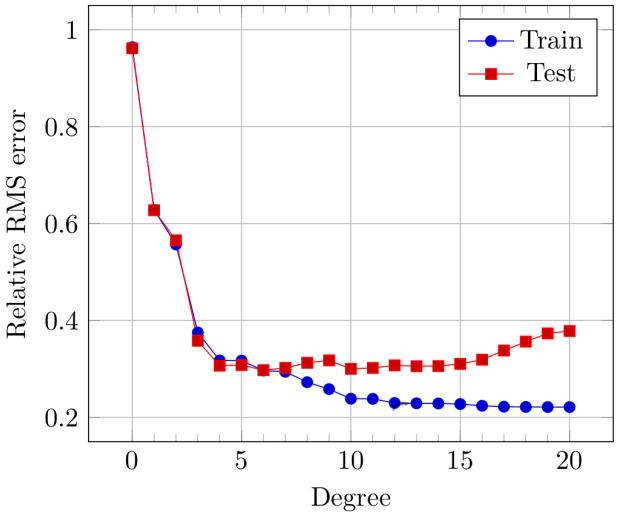

Figure 13.11 RMS error versus polynomial degree for the fitting example in figures 13.6 and 13.10. Circles indicate RMS errors on the training set. Squares show RMS errors on the test set.

The plot on the right shows train and test error (RMS standards for root mean square, and is the square root of MMSE) vs. the degree of the polynomial being fit. Notice that despite train error decreasing monotonically, test error goes down and then increases as we start to over-fit. This plot suggests that polynomials of degree 4, 5, or 6 will generalize well while achieving a small error.

For more about validation and feature engineering, refer to UMLS Ch. 13.2 and 13.3. These are essential components of modern data science and machine learning.