CAP 5625: Computational Foundations for Artificial Intelligence

Linear regression

Our first statistical (machine) learning method

In this lecture, we will introduce linear regression.

Notation

Input X: X is often multidimensional. Each dimension of X is referred to as a feature, predictor, or independent variable

Output Y: The response, or dependent variable

Categorization

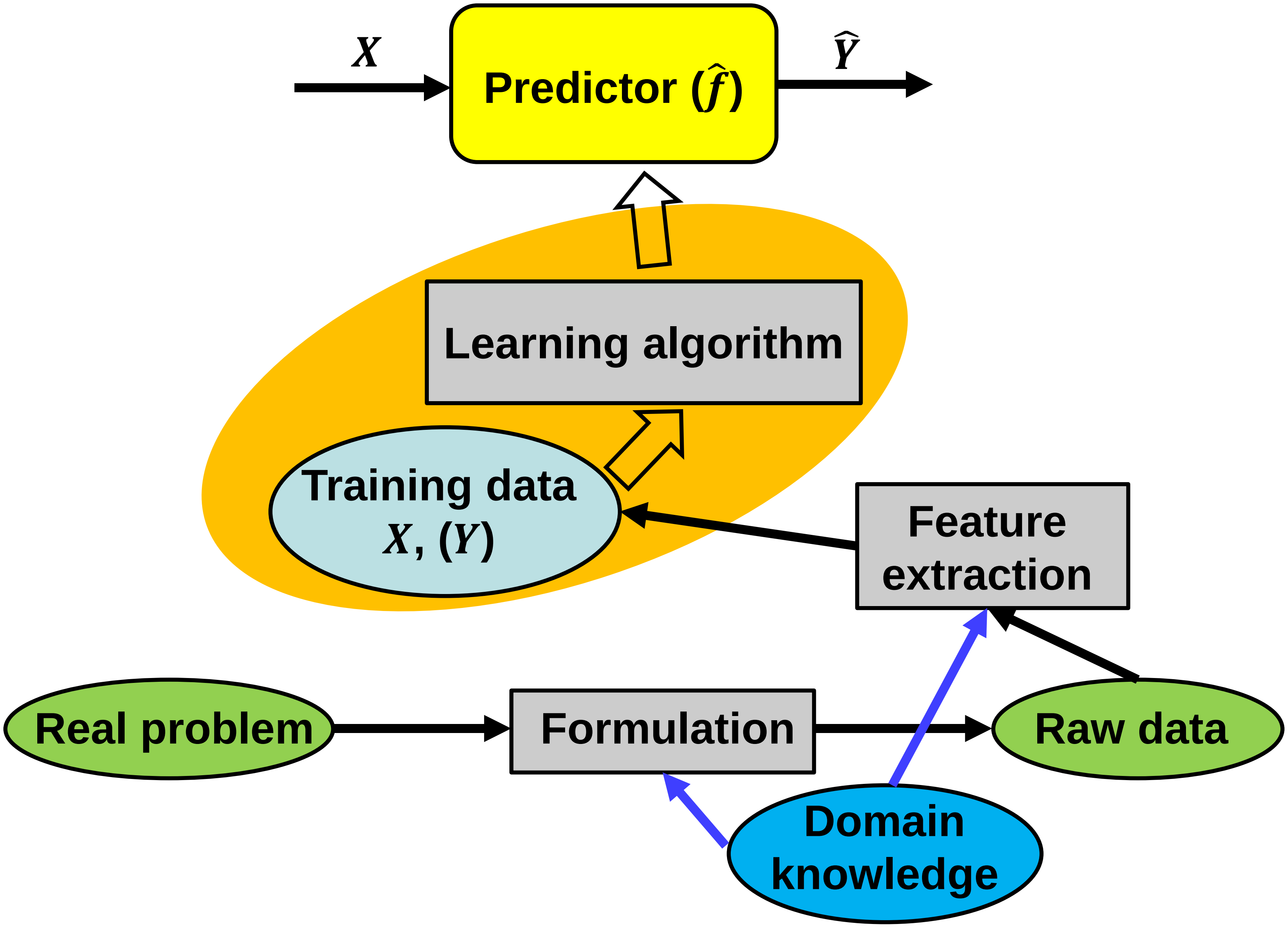
Supervised learning vs. unsupervised learning

Is Y available in the training data?

Regression vs. classification

Is Y quantitative or qualitative?

Statistical learning is simply function (f) estimation



Estimating the function f

To estimate the function f used to predict response Y from features X, we employ **training data**.

Training data are a set of N training observations, such that each observation i, $i = 1,2,\ldots,N$, has a measured feature value x_i and a measured response value y_i.

This data forms a set of tuples (x_i, y_i) for $i = 1,2,\ldots,N$.

We apply a statistical (machine) learning method to the set of training data to estimate the function f, and denote the estimate by $\hat{f}$.

That is, wish to find function $\hat{f}$ such that $Y \approx \hat{f}(X)$ for any observation (X, Y).

Two classes of supervised learning methods

Two key classes of supervised learning methods are parametric and non-parametric approaches.

Parametric methods use an explicit model that makes assumptions about the shape of the underlying function f being estimated.

Non-parametric methods do not make such assumptions about the shape of f, but instead attempt to find an estimate that is close to the training data with an assumed level of smoothness (complexity).

In general, parametric give better predictions than non-parametric methods if their model assumptions hold, and tend to perform worse when their modeling assumptions are violated.

Linear regression

Linear regression is a parametric (model-based) supervised learning method that assumes a response Y can be estimated well by a linear combination of input features $X = [X_1, X_2, \dots, X_p]$.

That is, linear regression assumes $Y = f(X_1, X_2, \dots, X_p)$ has the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

where $\beta = [\beta_0, \beta_1, \dots, \beta_p]$ is the collection of $p + 1$ model **parameters**, which are also known as **regression coefficients**.

The goal is that, given some training data, we hope to be able to learn a linear relationship between Y and input features $X = [X_1, X_2, \dots, X_p]$ so that given measurements of these features on a new observation, we may be able to accurately predict its response Y.

Learning f from parametric method

Step 1: Make an assumption about the shape of f.

For linear regression, f is linear function of $X = [X_1, X_2, \ldots, X_p]$ of the form

$$Y = f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

Step 2: Use training data to **fit** (**train**) the model (*i.e.*, estimate model parameters).

For linear regression, we estimate parameters $\beta = [\beta_0, \beta_1, \ldots, \beta_p]$ such that

$$Y \approx \hat{f}(X) = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \cdots + \hat{\beta}_p X_p$$

where $\hat{\beta}_j$ is an estimate of parameter β_j, $j = 0,1, \ldots, p$.

Learning f from non-parametric methods

Non-parametric methods do not impose constraint on shape of underlying function f, leading to many diverse ways of estimating f.

Such methods in general produce an estimate of f that is as close as possible to the observed training data with a restriction on how smooth (complex) the underlying function should be.

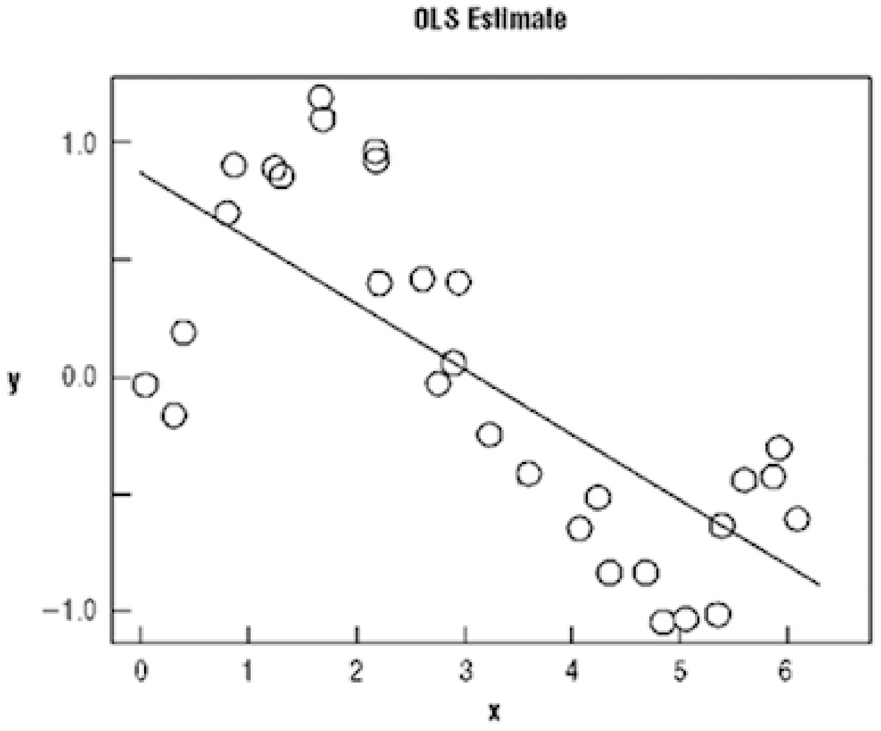
Too complex functions will **overfit** the training data, and so it is important to determine the appropriate level of **smoothness** of f.

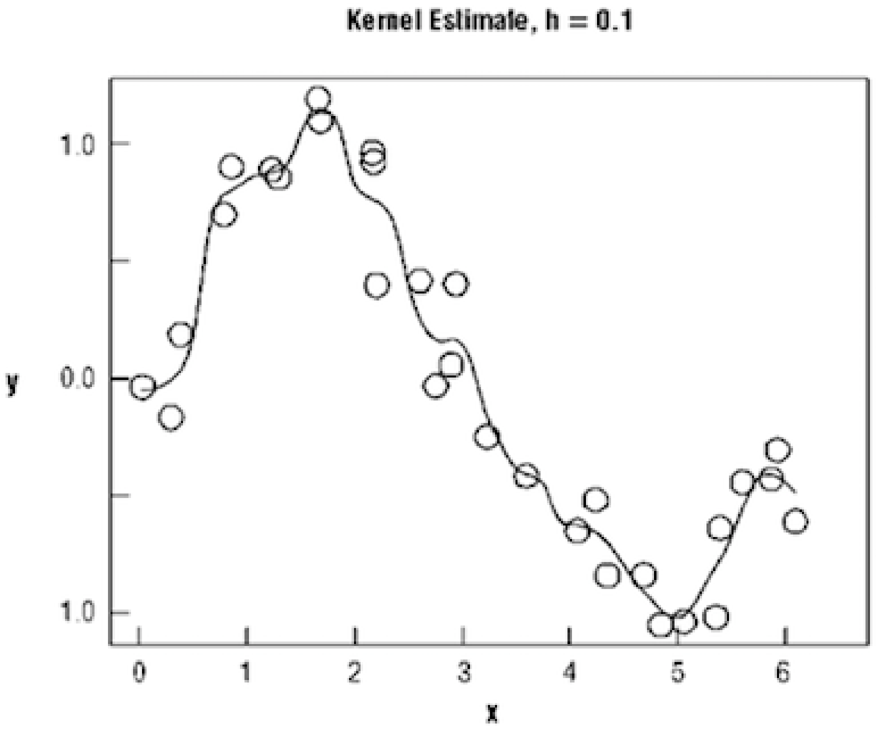
We will see a number of examples in this course on how to prevent overfitting.

Key advantages and disadvantages of method type

Advantages of parametric methods are disadvantages of non-parametric, and vice versa.

Below is an application of linear regression (parametric) compared to kernel estimation (non-parametric) on identical training data (dots).



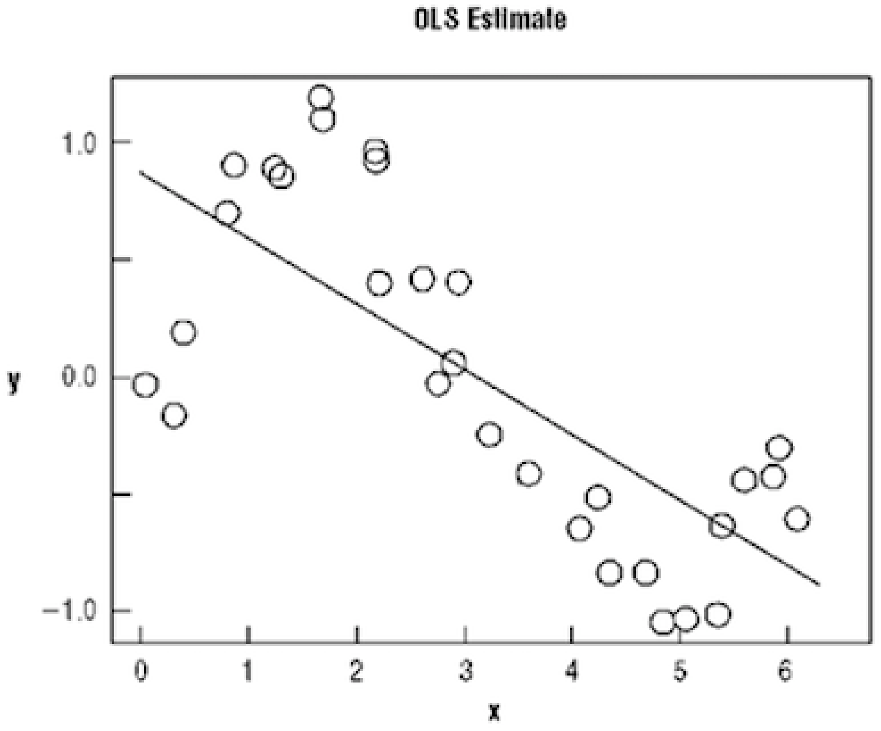


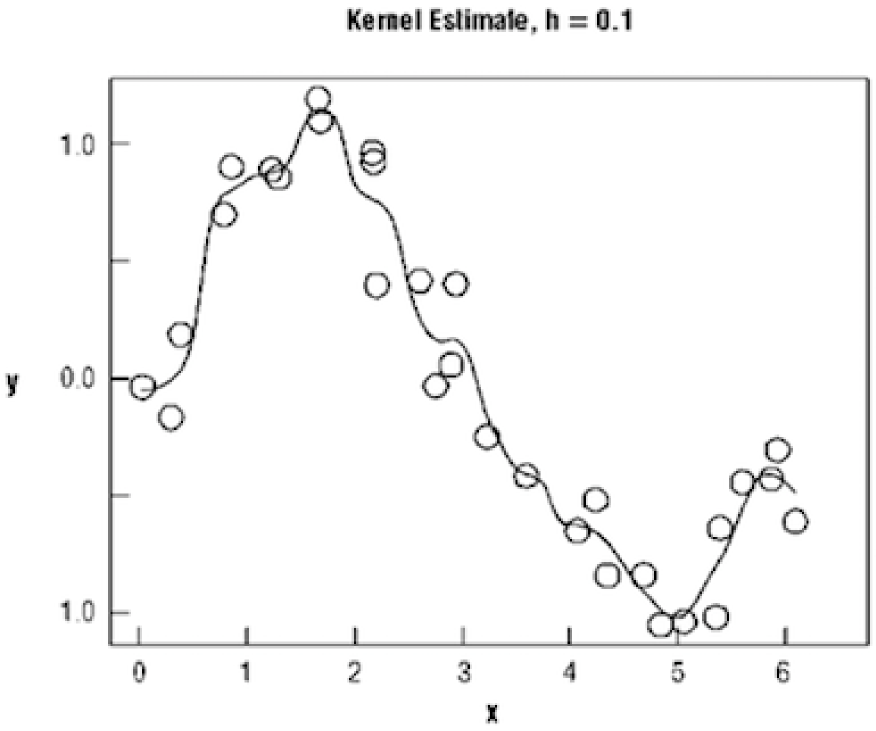
Easier to estimate this parametric model

The model fitting is easier for the parametric method, as it is easier to estimate the two parameters β_0 and β_1 that form the linear model

$$Y = f(X) = \beta_0 + \beta_1 X$$

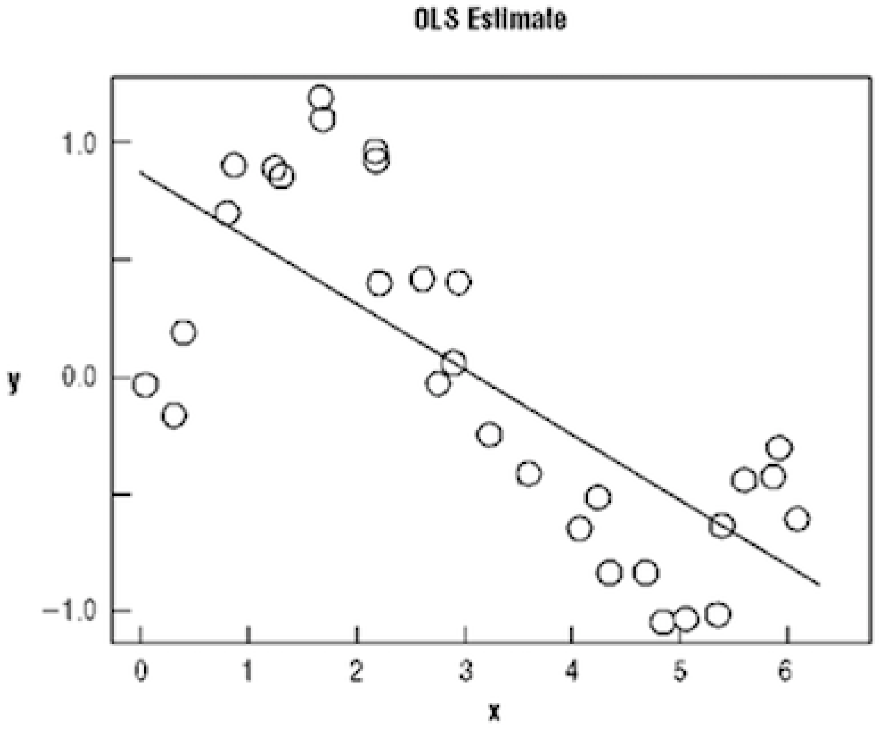
than it is to estimate the arbitrary function f for the non-parametric model.

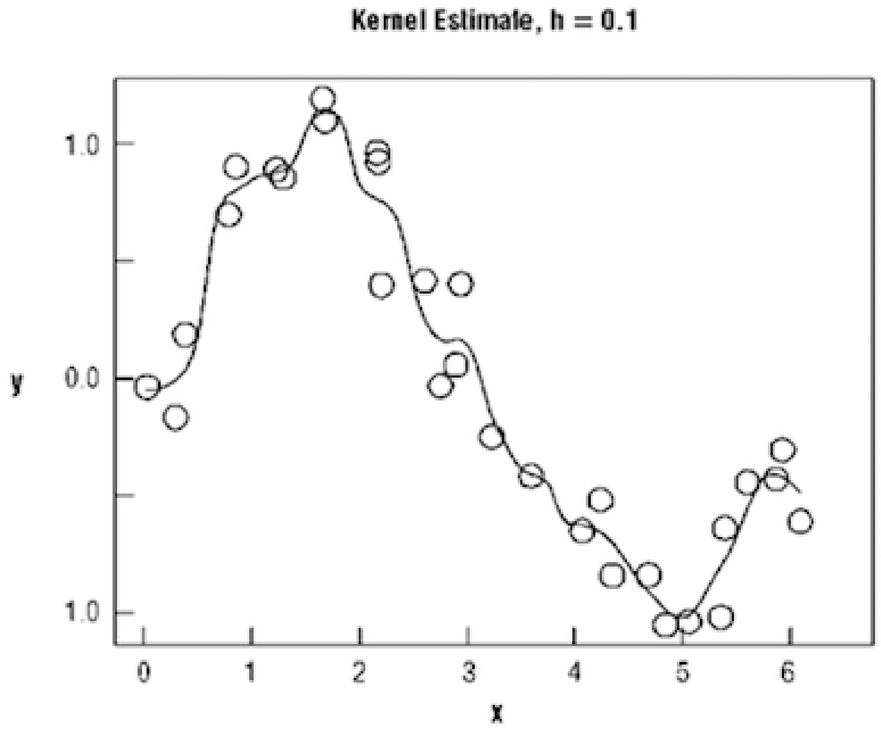




Easier to interpret this parametric model

The trained model (*i.e.*, fit function f) is more interpretable than the function learned by the non-parametric method, as its shape is explicit (linear), making the relationship between the response Y and the features (X) easy to understand.

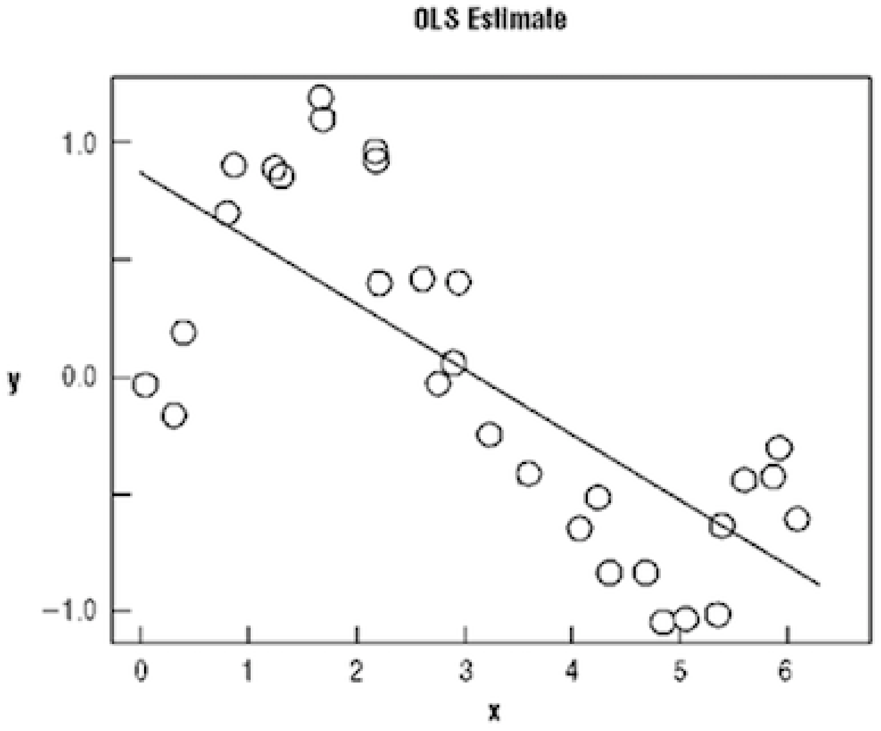


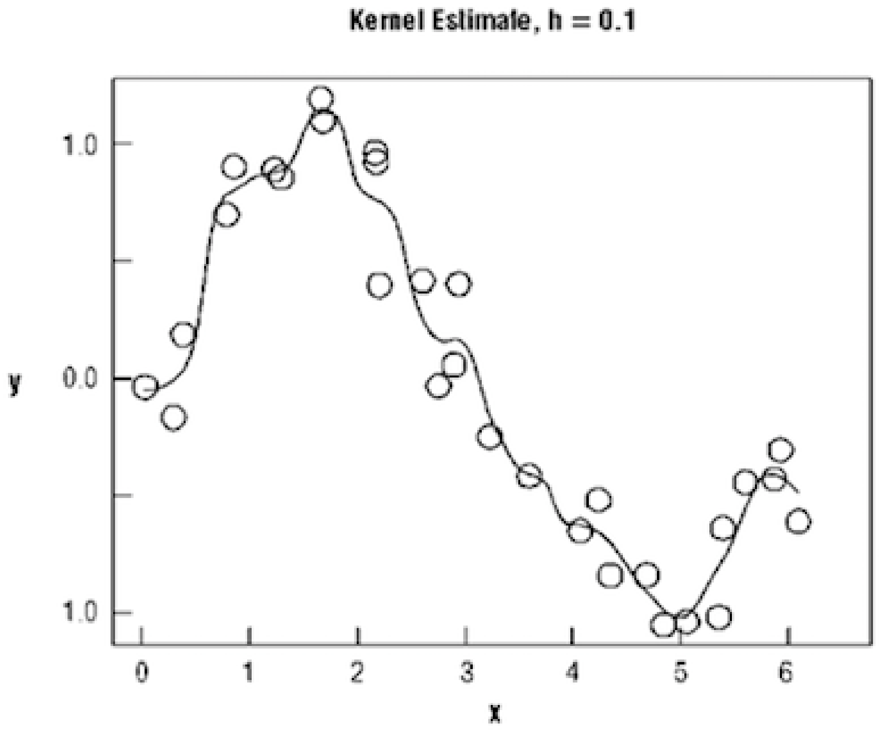


Easier to match unknown shape for non-parametric

In this example, the parametric model makes an assumption that the shape of f is linear in X.

However, if we do not know the true underlying form of f, then it is easier to match this form with a non-parametric method.

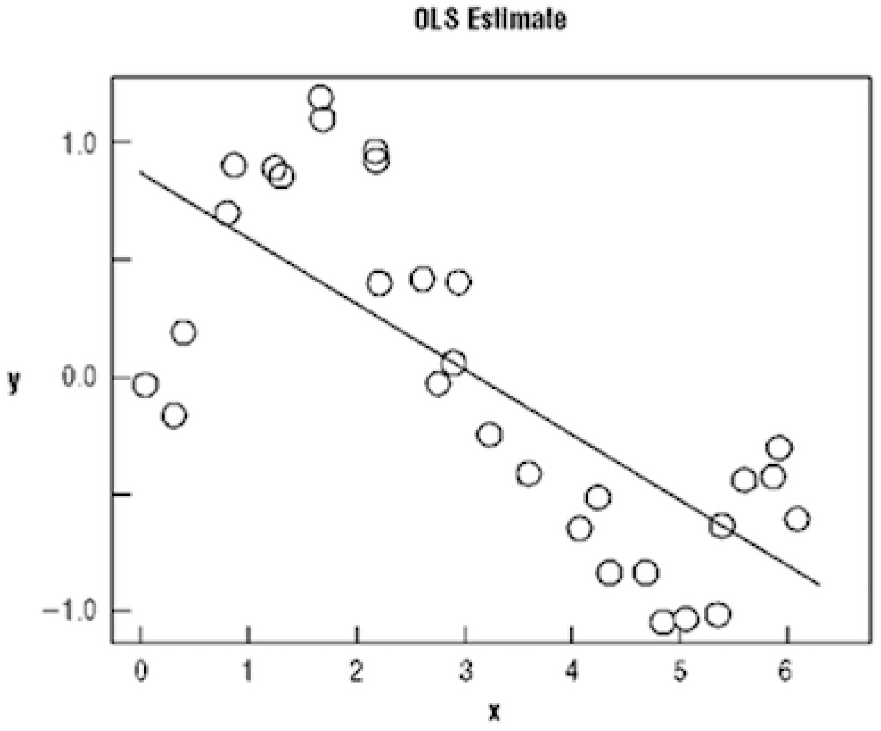


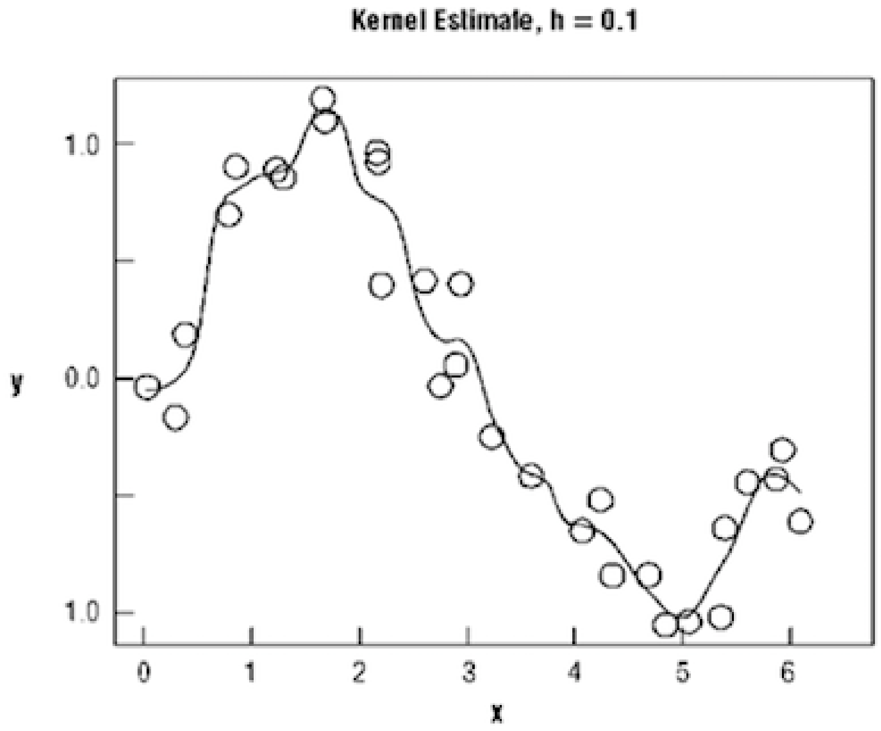


Greater flexibility of non-parametric

This relatively unconstrained non-parametric model in this example has greater flexibility than the constrained linear model.

However, this greater flexibility requires caution, as it can lead to the model overfitting the training data (*i.e.*, poor predictions on new data).

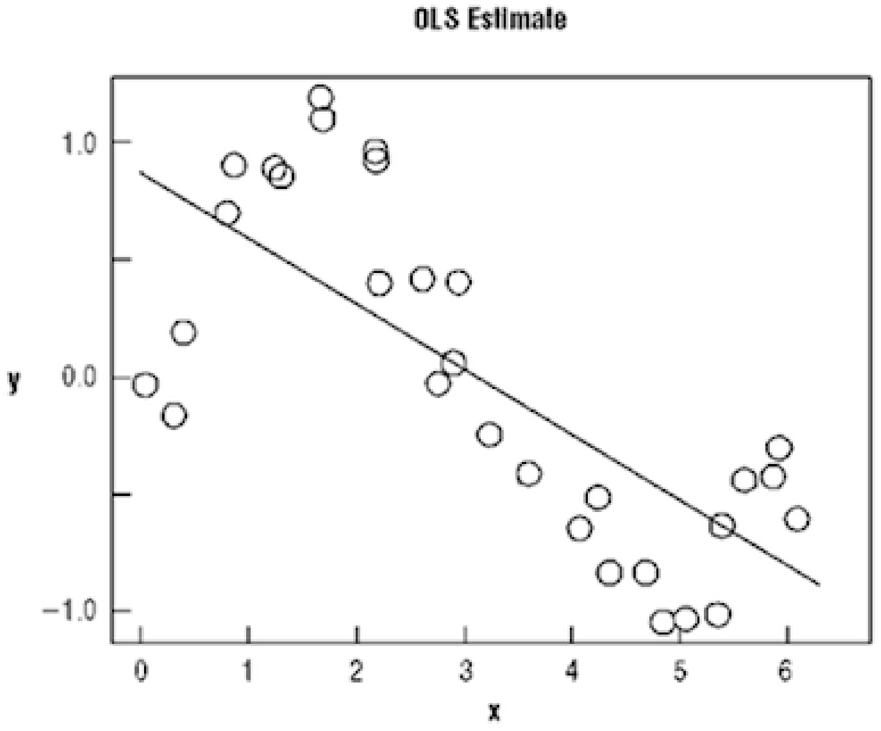


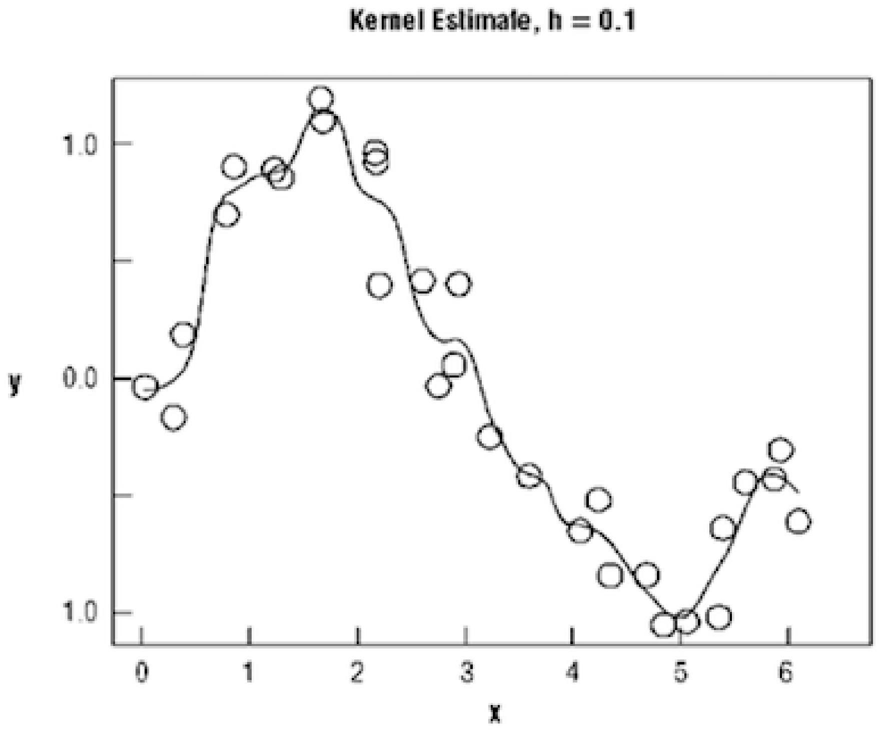


There is a balance of complexity

The left plot assumes too simple of a model, and so the predictions will be poor due to a phenomenon called **high bias**.

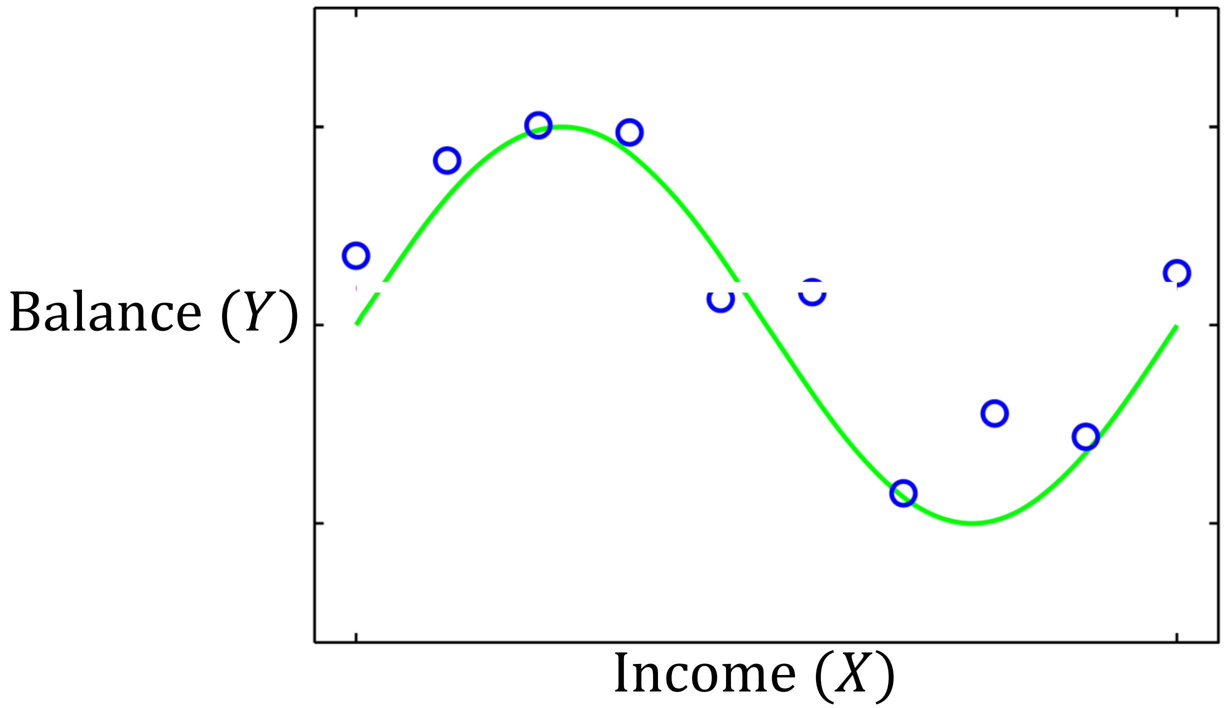
The right plot fits many details of the training data, which could lead to poor predictions due to **high variance**.





Modeling non-linear relationships

Suppose the relationship between credit balance Y and income X is nonlinear, as illustrated below as a green curve $Y = \sin(2\pi X)$.



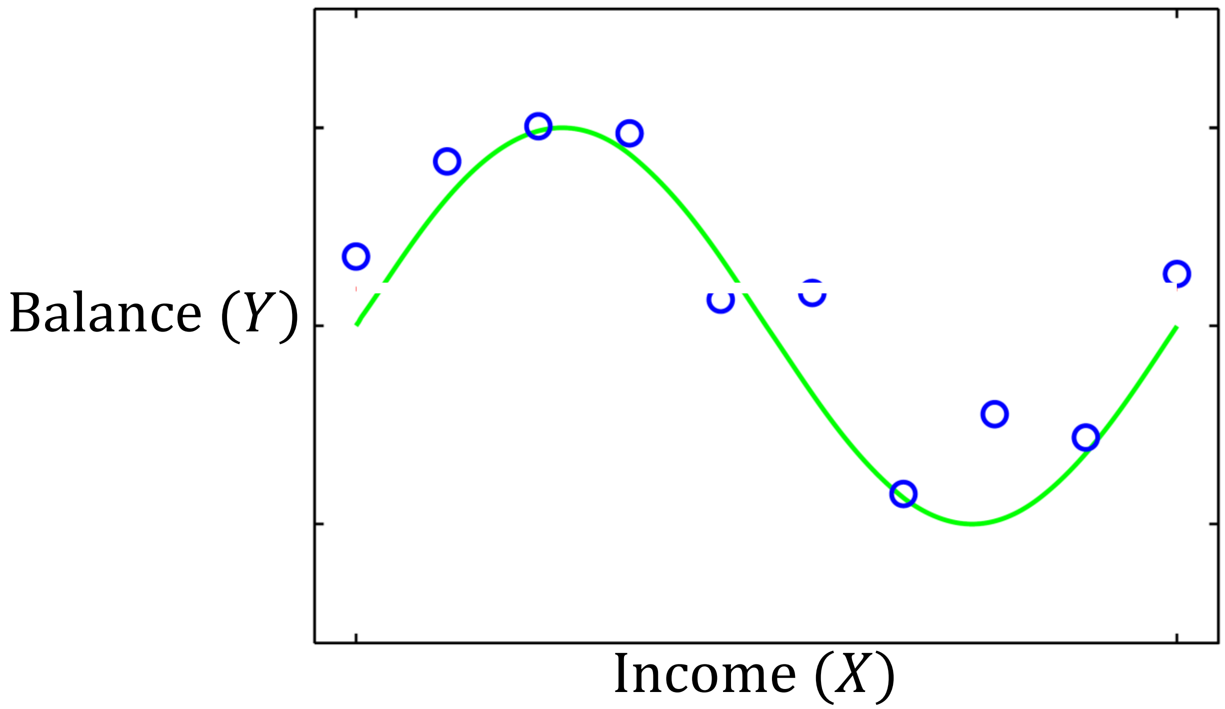
How might we model such a relationship? We could try letting balance Y be a polynomial function of income X.

Modified from Bishop (2006) *Pattern Recognition and Machine Learning*. Springer.

Modeling output as a polynomial function of input

Specifically, we could let Y be written as a polynomial function of X of degree M, $M = 0,1,\ldots,$ which is represented as

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_M X^M$$

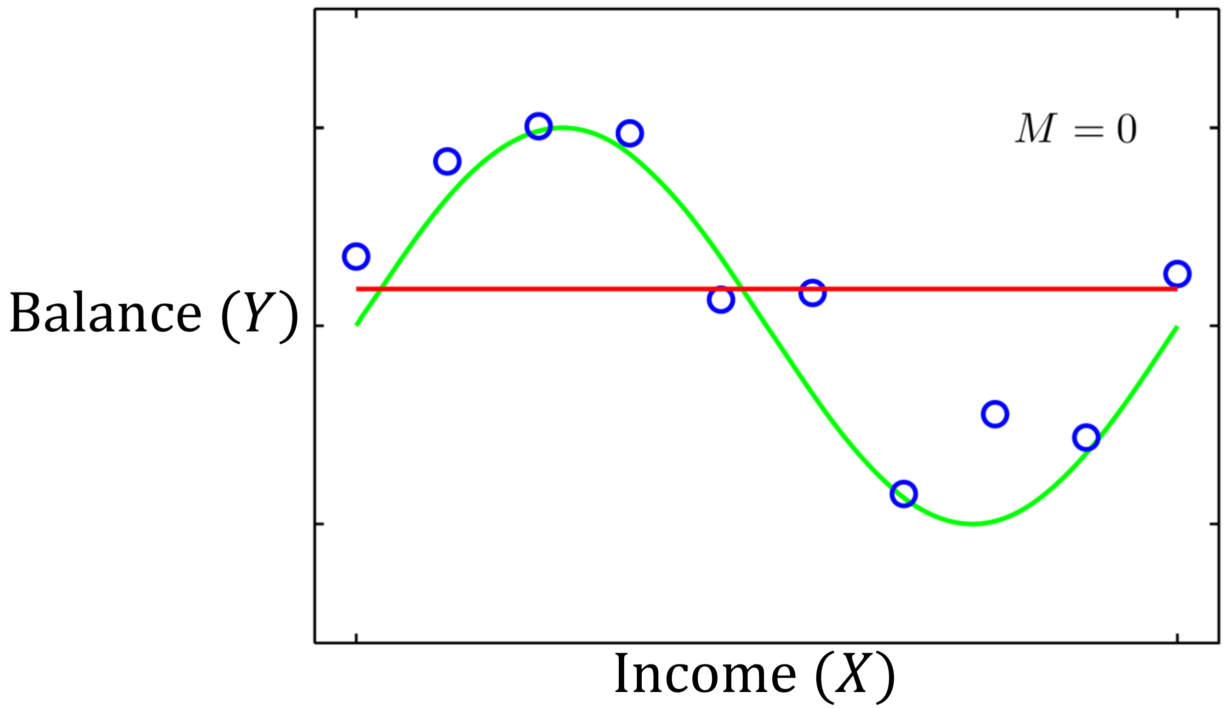


Modified from Bishop (2006) *Pattern Recognition and Machine Learning*. Springer.

Poor fit with a constant function (zero degree)

Letting the degree be $M = 0$, we fit a horizontal line (red), which demonstrates a pretty poor fit to the data (blue circles)

$$Y = \beta_0$$

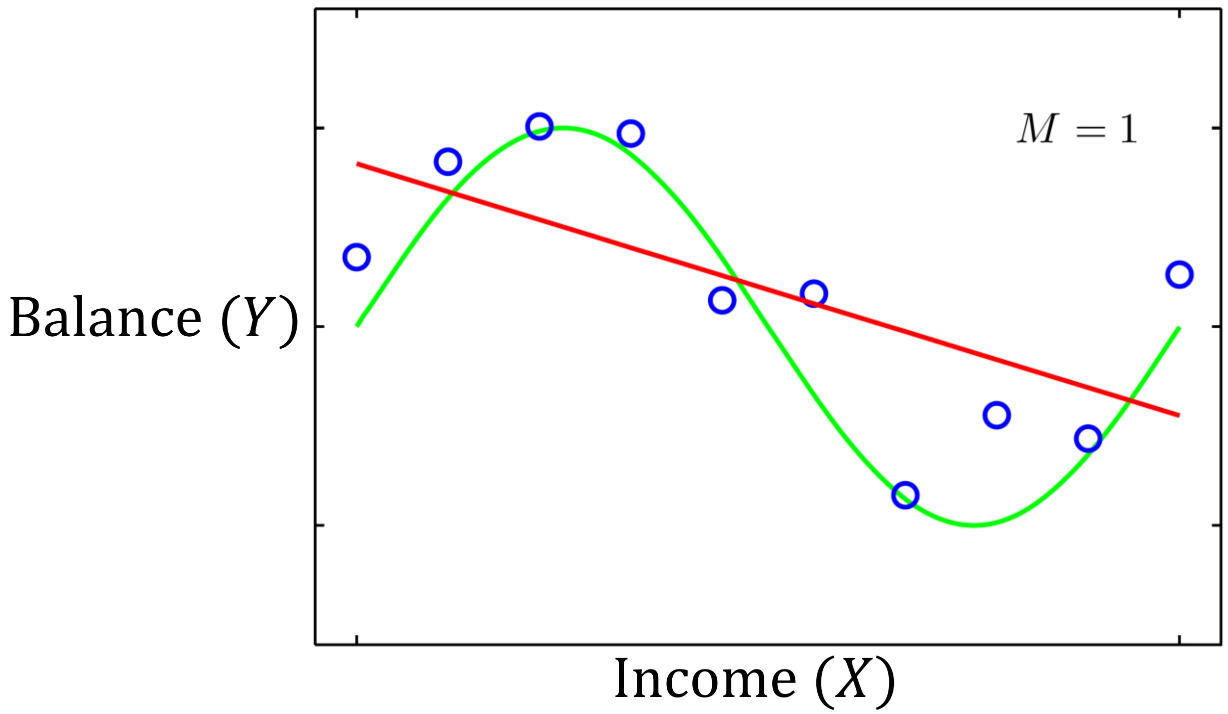


Modified from Bishop (2006) *Pattern Recognition and Machine Learning*. Springer.

Poor fit with a linear function (first degree)

Letting the degree be $M = 1$, we fit a line with a non-zero slope (red), which also demonstrates a pretty poor fit to the data, as the Y is nonlinear with respect to X

$$Y = \beta_0 + \beta_1 X$$

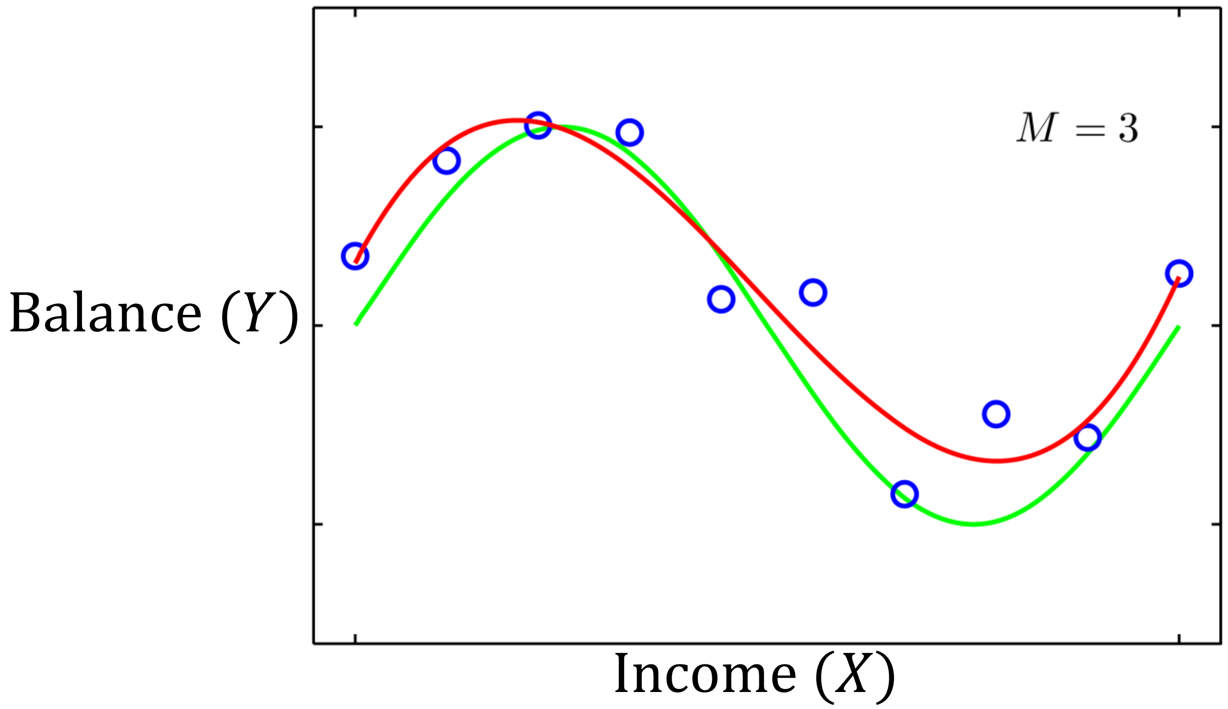


Modified from Bishop (2006) *Pattern Recognition and Machine Learning*. Springer.

Good fit with a third-degree polynomial

Letting the degree be $M = 3$, we fit a curve (red), which demonstrates a fairly good fit to the data, and models the oscillations of the function that generated the data

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$$

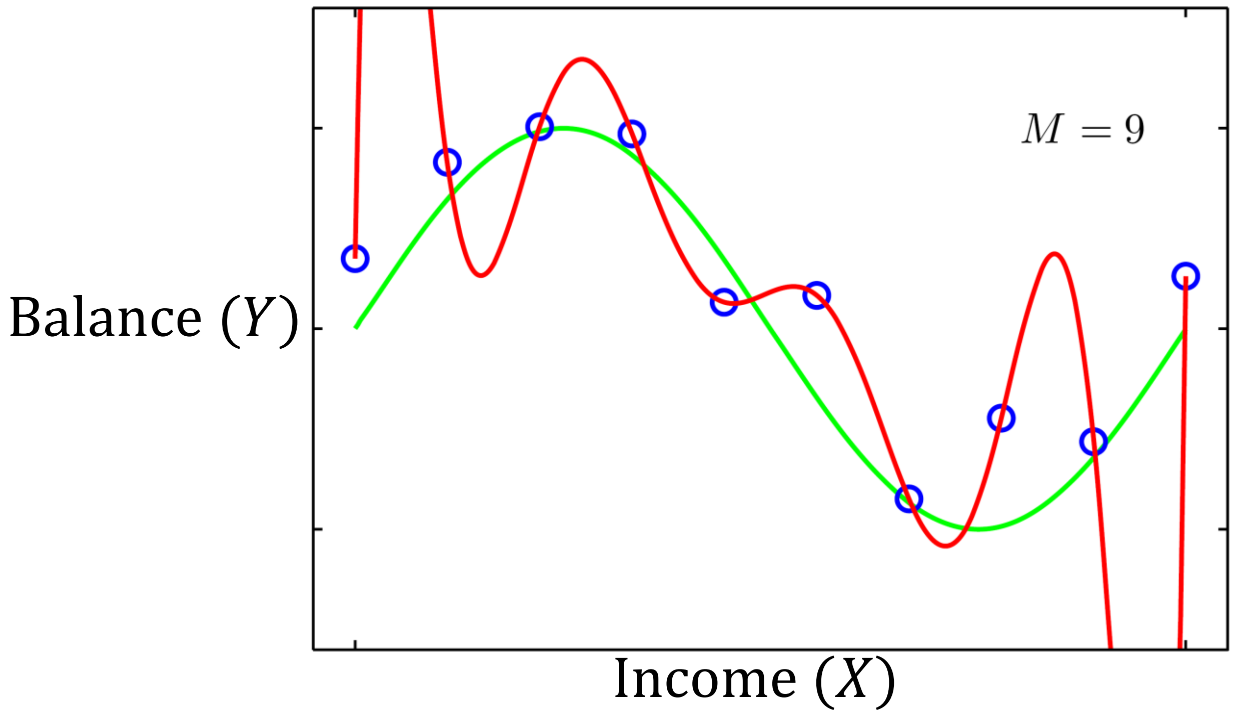


Modified from Bishop (2006) *Pattern Recognition and Machine Learning*. Springer.

Poor fit with a ninth-degree polynomial

Letting the degree be $M = 9$, we fit a curve (red), which demonstrates a poor fit to the data, as we are now **overfitting** by modeling every training example exactly

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \cdots + \beta_9 X^9$$



Modified from Bishop (2006) *Pattern Recognition and Machine Learning*. Springer.

Simple linear regression

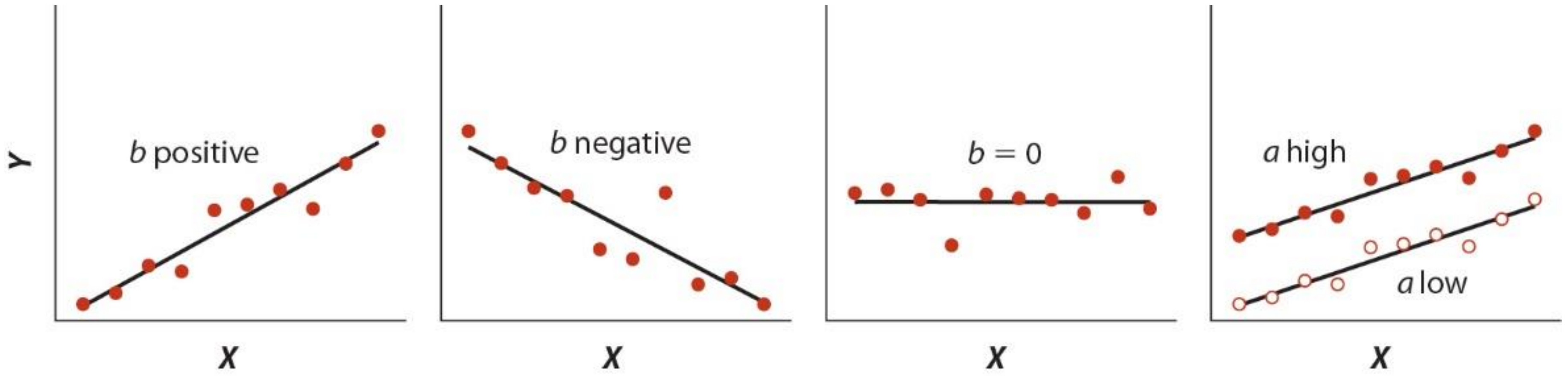
Simple linear regression predicts response Y from one feature X as

$$Y = \beta_0 + \beta_1 X$$

which is just an equation for a line with intercept β_0 and slope β_1.

Here, β_0 is the value of Y when $X = 0$ and β_1 describes the rate of change in Y as a function of X.

The β_0 and β_1 parameters are unknown quantities from a **population** that we estimated from our training data **sample**.



Simple linear regression

We first fit the linear regression model

$$Y = \beta_0 + \beta_1 X$$

to the sampled training data (x_i, y_i), $i = 1,2,\ldots,N$ to estimate the population parameters β_0 and β_1.

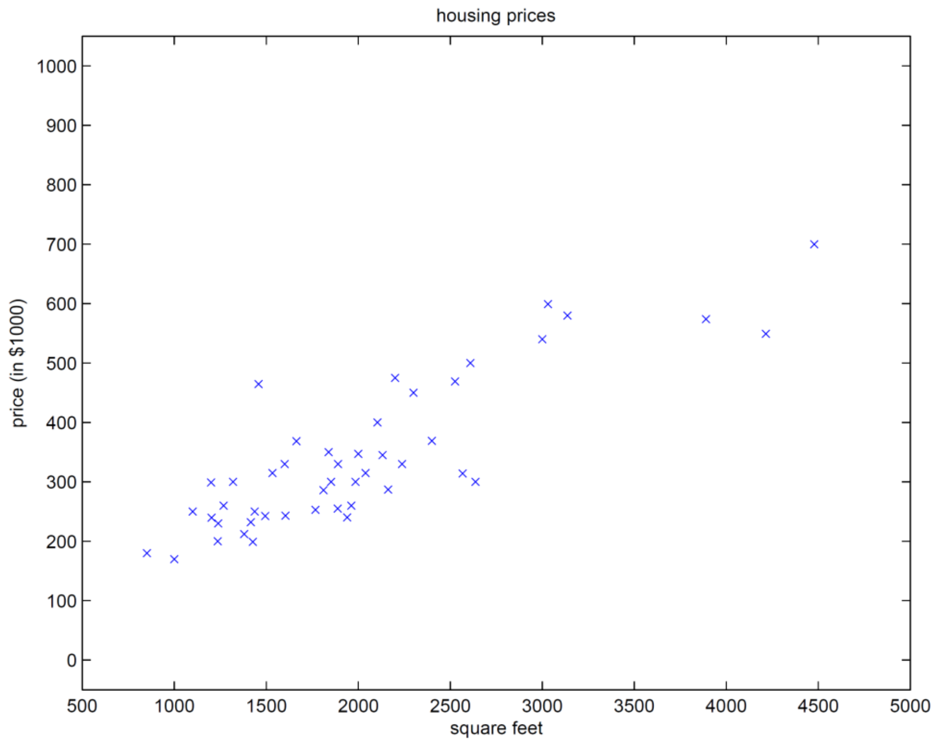
The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ of the parameters provides us with a relationship between Y and X that we can use to make predictions.

Given a new observation X, the hope is that our model $\hat{f}(X)$ can accurately predict the response Y with a model of the form

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

Example on estimating housing prices

Consider the following set of plotted datapoints, demonstrating the relationship between price of a house (y-axis) and square footage of the home (x-axis).



Price appears linearly related with square footage, and we wish to uncover what this linear relationship is.

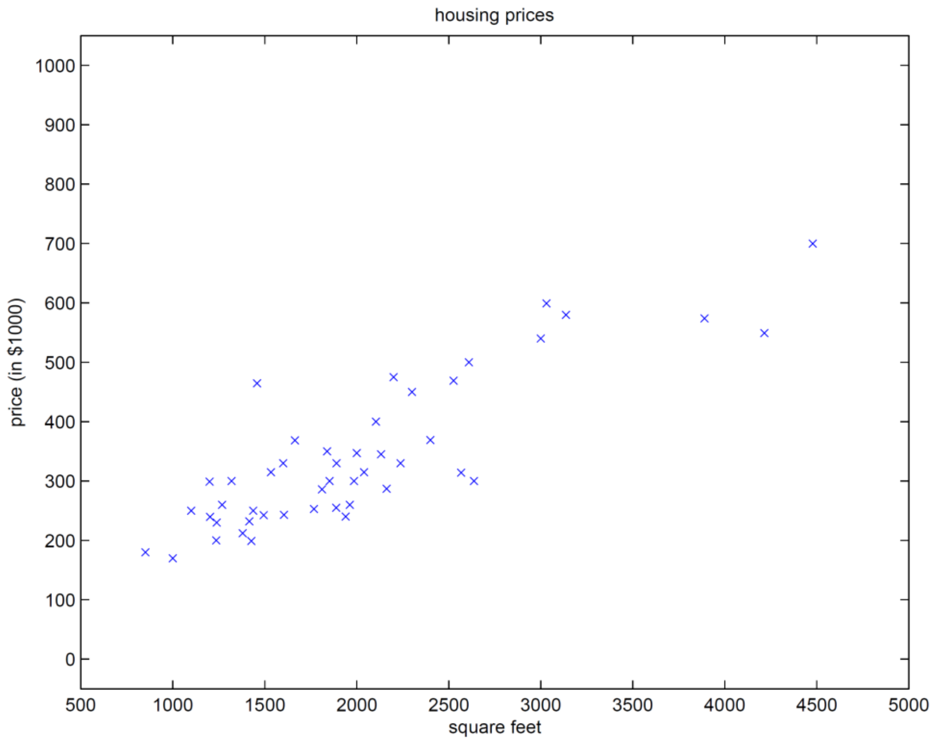
http://cs229.stanford.edu/notes/cs229-notes1.pdf

Example on estimating housing prices

A linear model relating housing price (Y) to square footage (X) would take the following form

$$Y = \beta_0 + \beta_1 X$$

Our goal is to estimate β_0 and β_1 that best fit the data.

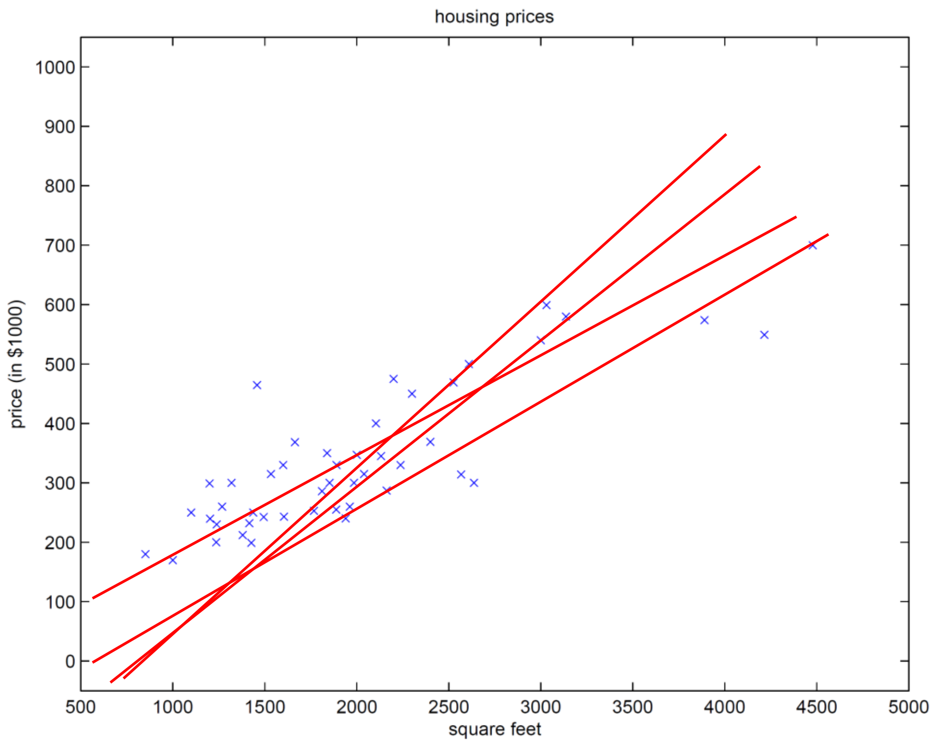


http://cs229.stanford.edu/notes/cs229-notes1.pdf

Example on estimating housing prices

There are an infinite number of lines that fit our data.

We want to find an estimate of the intercept $\hat{\beta}_0$ and slope $\hat{\beta}_1$ that provides the best fit (line close to the set of training data points).



http://cs229.stanford.edu/notes/cs229-notes1.pdf

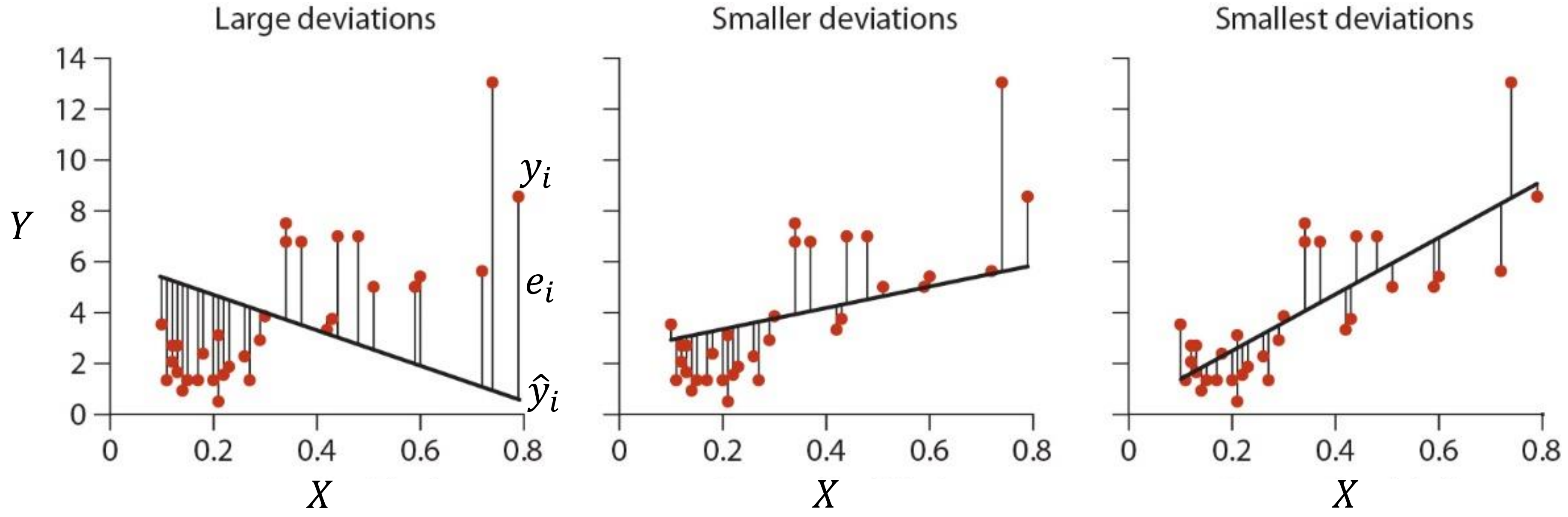
Estimating the best-fit regression line

To estimate the parameters β_0 and β_1, we need to minimize the deviations (differences) between observed and predicted responses.

That is, for training observation (x_i, y_i), $i = 1,2,\ldots,N$, let the prediction of its response y_i based on input features x_i be

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

with the **residual** as $e_i = y_i - \hat{y}_i$.



The cost function $J(\beta)$

The residual sum of squares (RSS) quantifies the sum of squared residuals as

$$\mathrm{RSS} = e_1^2 + e_2^2 + \cdots + e_N^2$$

and is therefore directly related to the reducible error (recall the bias-variance tradeoff).

Therefore, the best fit line is the one that minimizes RSS, which is referred to as the **least squares** regression line.

In statistical (machine) learning, a **cost function** quantifies the amount of error between predictions and observations, and we therefore seek to minimize the least squares regression cost function

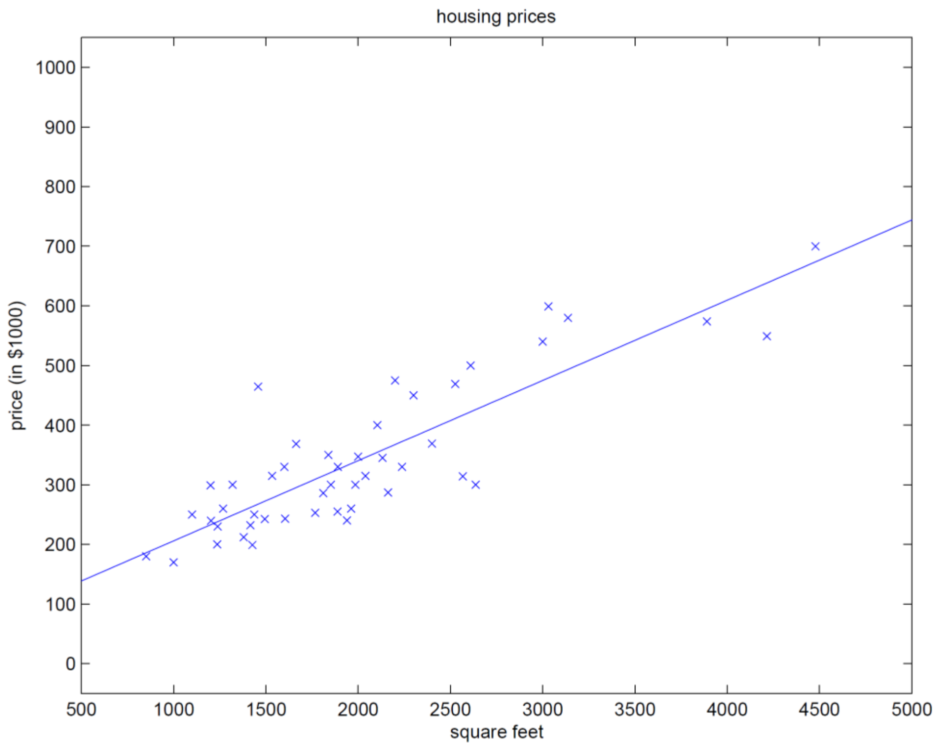
$$J(\beta) = \mathrm{RSS} = e_1^2 + e_2^2 + \cdots + e_N^2$$

Example on estimating housing prices

The best-fit parameters are $\hat{\beta}_0 = 71.27$ and $\hat{\beta}_1 = 0.1345$, yielding

$$Y = 71.27 + 0.1345X$$

This best-fit model is shown as the blue line in the center of the datapoints in the graph below.

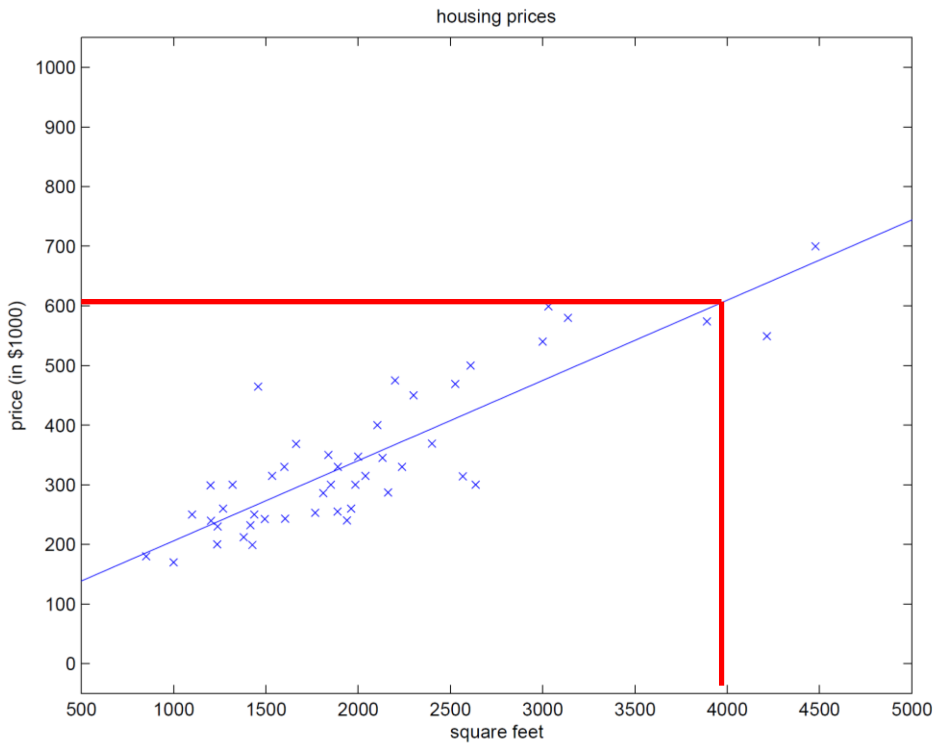


http://cs229.stanford.edu/notes/cs229-notes1.pdf

Example on estimating housing prices

We can now predict Y from a new observation ($e.g.$, $X = 4000$) as

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$
$$= 71.27 + 0.1345(4000)$$
$$= 609.27$$



http://cs229.stanford.edu/notes/cs229-notes1.pdf

Extending the housing price example

This housing price example was overly simplistic, as it is likely that we will have access to measurements on other characteristics of a home that could lead to better prediction.

As an example, online real-estate companies that provide predictions of home prices may use a number of additional features such as

- Location
- Numbers of bedrooms and bathrooms
- Price of comparable recently-sold homes
- Etc.

We want to ultimately be able to build models that incorporate as many features as possible, and that will automatically put greater emphasis on those that are important for prediction.

Multiple linear regression models

In general, for linear regression we assume that a response Y can be estimated by a linear combination of p features $X = [X_1, X_2, \dots, X_p]$ as

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

We seek to find the $p + 1$ model parameters $\beta = [\beta_0, \beta_1, \dots, \beta_p]$ that minimize the cost function $J(\beta) = \mathrm{RSS}$ to learn the relationship between Y and $X = [X_1, X_2, \dots, X_p]$.

Our goal is that given measured features $X = [X_1, X_2, \dots, X_p]$ on a new observation, we wish to accurately predict the response Y for that observation.

Some notation for this course

Consider a set of N observations, for which we measure p features.

Let x_{ij} represent the value of the jth feature for the ith observation, for $i \in \{1,2,\dots,N\}$ and $j \in \{1,2,\dots,p\}$.

Let $\mathbf{X}$ denote an $N \times p$ **design matrix** whose element in the ith row and jth column is x_{ij}, and that is represented as

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{bmatrix}$$

Rows of $\mathbf{X}$ as column vectors

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{bmatrix}$$

If we are interested in rows of $\mathbf{X}$, then we write $x_1, x_2, \dots, x_N$, with

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{bmatrix}$$

which is a column vector (all will be column vectors by default). We can therefore represent $\mathbf{X}$ by

$$\mathbf{X} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix}$$

Columns of $\mathbf{X}$ as column vectors

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{bmatrix}$$

If we are interested in columns of $\mathbf{X}$, then we write $\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_p$, with

$$\mathrm{x}_j = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{Nj} \end{bmatrix}$$

We can therefore represent $\mathbf{X}$ by

$$\mathbf{X} = [\mathrm{x}_1 \quad \mathrm{x}_2 \quad \cdots \quad \mathrm{x}_p]$$

Building a linear regression model

To begin we will need to first collect data to build a model. This data is known as **training data**.

Suppose we have available a set of N training observations, such that every training observation $i \in \{1,2,\ldots,N\}$ has both input measurements x_i and output measurements y_i.

That is, our training data is the set of tuples (x_i, y_i) for $i = 1,2,\ldots,N$.

We want to train a linear regression model such that for a new input X, we can predict the output Y.

Prediction under a linear model

We begin by developing a simple parametric approach for predicting output Y from given input $X^T = \left[X_1, X_2, \dots, X_p\right] \in \mathbb{R}^p$, where the vector of inputs X is based upon p features.

Assuming that Y is linearly related to X, we can estimate (predict) the output Y assuming the linear model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

where $\hat{\beta}_0$ is the intercept of our trained model, and $\hat{\beta}_j$ represents the rate of change of the output as a function of changes in feature j.

Prediction under a linear model

It is often convenient to write the linear model in vector notation.

Consider redefining X as $X^T = [1, X_1, X_2, \dots, X_p]$, and define the vector $\beta^T = [\beta_0, \beta_1, \dots, \beta_p]$.

Let $\hat{\beta}^T = [\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p]$ be an estimate of the coefficient vector β.

Using these vectors, we can see that $\hat{Y} = X^T \hat{\beta}$ because

$$\begin{bmatrix} 1 & X_1 & \cdots & X_p \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{bmatrix} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

Extension to vector-valued output

So far our output Y has been a scalar value. However, this framework can extend to vector-valued output Y.

Suppose that output $Y^T = [Y_1, Y_2, \dots, Y_K] \in \mathbb{R}^K$ is K-dimensional.

To accommodate these K outputs per observation, we can assume a separate regression coefficient β_{jk} for feature $j \in \{0,1,\dots,p\}$ of output dimension $k \in \{1,2,\dots,K\}$.

Therefore, the regression coefficients would be represented by the $(p+1) \times K$-dimensional matrix $\mathbf{B} \in \mathbb{R}^{(p+1)\times K}$, and would predict Y as

$$\hat{Y}^T = X^T \hat{\mathbf{B}}$$

Extension to vector-valued output

Specifically, we would have

$$\mathbf{B} = \begin{bmatrix} \beta_{01} & \beta_{02} & \cdots & \beta_{0K} \\ \beta_{11} & \beta_{12} & \cdots & \beta_{1K} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{p1} & \beta_{p2} & \cdots & \beta_{pK} \end{bmatrix}$$

such that the prediction of Y from estimate $\widehat{\mathbf{B}}$ of $\mathbf{B}$ is

$$\begin{bmatrix} \hat{Y}_1 \\ \hat{Y}_2 \\ \vdots \\ \hat{Y}_K \end{bmatrix}^T = \begin{bmatrix} 1 & X_1 & \cdots & X_p \end{bmatrix} \begin{bmatrix} \hat{\beta}_{01} & \hat{\beta}_{02} & \cdots & \hat{\beta}_{0K} \\ \hat{\beta}_{11} & \hat{\beta}_{12} & \cdots & \hat{\beta}_{1K} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\beta}_{p1} & \hat{\beta}_{p2} & \cdots & \hat{\beta}_{pK} \end{bmatrix} = \begin{bmatrix} \hat{\beta}_{01} + \sum_{j=1}^{p} X_j \hat{\beta}_{j1} \\ \hat{\beta}_{02} + \sum_{j=1}^{p} X_j \hat{\beta}_{j2} \\ \vdots \\ \hat{\beta}_{0K} + \sum_{j=1}^{p} X_j \hat{\beta}_{jK} \end{bmatrix}^T$$

How do we estimate the regression coefficients β?

To estimate the regression coefficients, we will fit a linear model to a set of training data (x_i, y_i), $i = 1,2, \dots, N$, using **least squares**.

That is, we will identify the coefficients β that minimize the residual sum of squares

$$J(\beta) = \mathrm{RSS}(\beta) = \sum_{i=1}^{N} \left(y_i - x_i^T \beta\right)^2$$

which represents the sum of squared differences between the linear function $x_i^T \beta$ used to predict the output and the true output y_i. Here, $J(\beta)$ is known as our **cost function**.

$J(\beta)$ is a convex function, and so an identified local minimum will also be the global minimum.

Writing the residual sum of squares in matrix notation

We construct the matrix of inputs $\mathbf{X} \in \mathbb{R}^{N\times(p+1)}$ with the ith row being the input of the ith training observation, and the vector $\mathbf{y} \in \mathbb{R}^N$ with the ith element being the output of the ith training observation. That is

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Note that $\mathbf{y} - \mathbf{X}\beta$ has value

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} y_1 - x_1^T\beta \\ y_2 - x_2^T\beta \\ \vdots \\ y_N - x_N^T\beta \end{bmatrix}$$

We can rewrite this function in matrix notation as

$$J(\beta) = \mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

Finding β (normal equations) that minimizes $J(\beta)$

We begin by writing $J(\beta)$ in an expanded form

$$J(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 = \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

Taking the partial derivative with respect to β_0, we have

$$\frac{\partial J(\beta)}{\partial \beta_0} = -\sum_{i=1}^{N} 2 \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right) = -2 \sum_{i=1}^{N} (y_i - x_i^T \beta)$$

which can be rewritten as

$$\frac{\partial J(\beta)}{\partial \beta_0} = -2 \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} y_1 - x_1^T \beta \\ y_2 - x_2^T \beta \\ \vdots \\ y_N - x_N^T \beta \end{bmatrix} = -2 \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} (\mathbf{y} - \mathbf{X}\beta)$$

where $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$ is an N-dimensional row vector.

Finding β (normal equations) that minimizes $J(\beta)$

Taking the partial derivative with respect to β_k, $k > 0$, we have

$$\frac{\partial J(\beta)}{\partial \beta_k} = -\sum_{i=1}^{N} 2x_{ik}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right) = -2\sum_{i=1}^{N} x_{ik}(y_i - x_i^T\beta)$$

which can be rewritten as

$$\frac{\partial J(\beta)}{\partial \beta_k} = -2\begin{bmatrix} x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix}\begin{bmatrix} y_1 - x_1^T\beta \\ y_2 - x_2^T\beta \\ \vdots \\ y_N - x_N^T\beta \end{bmatrix}$$

$$= -2\begin{bmatrix} x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix}(\mathbf{y} - \mathbf{X}\beta)$$

where $\begin{bmatrix} x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix}$ is an N-dimensional row vector.

Finding β (normal equations) that minimizes $J(\beta)$

Putting this together, to identify the minimum, we take the gradient with respect to β, which is a $(p+1)$-dimensional vector, with dimension k the partial derivative of $J(\beta)$ with respect to β_k, $k \in \{0,1,\dots,p\}$. The gradient is therefore

$$\frac{\partial J(\beta)}{\partial \beta} = -2\begin{bmatrix} [1 \quad 1 \quad \cdots \quad 1](\mathbf{y}-\mathbf{X}\beta) \\ [x_{11} \quad x_{21} \quad \cdots \quad x_{N1}](\mathbf{y}-\mathbf{X}\beta) \\ \vdots \\ [x_{1p} \quad x_{2p} \quad \cdots \quad x_{Np}](\mathbf{y}-\mathbf{X}\beta) \end{bmatrix}$$

$$= -2\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{21} & \cdots & x_{N1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{Np} \end{bmatrix}(\mathbf{y}-\mathbf{X}\beta)$$

$$= -2\mathbf{X}^T(\mathbf{y}-\mathbf{X}\beta)$$

Finding β (normal equations) that minimizes $J(\beta)$

Setting the gradient to the 0 vector, gives

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

which yields

$$\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\beta = 0$$

or

$$\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y}$$

If $\mathbf{X}^T\mathbf{X}$ is invertible, the estimated coefficients (**normal equations**) are

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

and is the vector $\beta \in \mathbb{R}^{p+1}$ that minimizes

$$J(\beta) = \|\mathbf{y} - \mathbf{X}\beta\|_2^2$$

with $\|v\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots v_n^2}$ for $v = [v_1, v_2, \ldots, v_n]^T \in \mathbb{R}^n$, and is known as the L_2 norm.

How do we predict output given input data?

We have now shown that $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ is the least squares estimate of the regression coefficients given training observations (x_i, y_i), $i = 1,2,\dots,N$.

The fitted value vector (estimate of the output of the original training data), is given by

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Define the hat matrix denoted by $\mathbf{H}$ as

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$$

such that $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$. $\mathbf{H}$ is called the hat matrix, because it puts the "hat" on $\mathbf{y}$.

Geometric interpretation of least squares estimate $\hat{\mathbf{y}}$

Each column of $\mathbf{X}$ is a vector in N-dimensional space, which we can denote as (recall our notation for the columns of $\mathbf{X}$)

$$\mathbf{X} = \begin{bmatrix} \mathrm{x}_0 & \mathrm{x}_1 & \cdots & \mathrm{x}_p \end{bmatrix}$$

with $\mathrm{x}_0 = 1$, the N-dimensional vector of ones.

The predicted output vector $\hat{\mathbf{y}}$ is a linear combination of the column vectors. That is

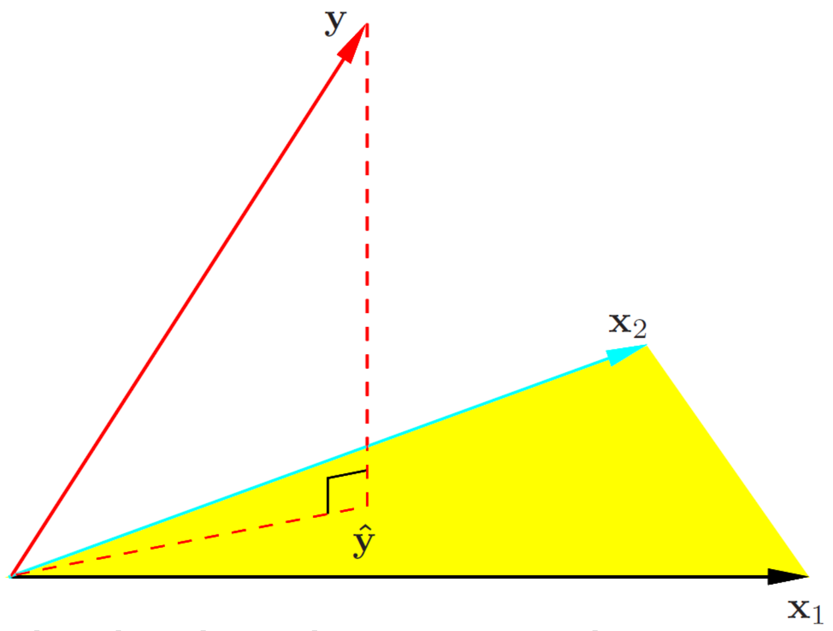
$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta} = \begin{bmatrix} \mathrm{x}_0 & \mathrm{x}_1 & \cdots & \mathrm{x}_p \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{bmatrix} = \sum_{j=0}^{p} \hat{\beta}_j \mathrm{x}_j$$

Therefore, $\hat{\mathbf{y}}$ lies in the subspace spanned by $\{\mathrm{x}_0, \mathrm{x}_1, \dots, \mathrm{x}_p\}$.

Geometric interpretation of least squares estimate $\hat{\mathbf{y}}$

We want to identify the vector lying within the subspace spanned by the column vectors of $\mathbf{X}$ that is closest to $\mathbf{y}$.

The projection of $\mathbf{y}$ on the subspace spanned by the column vectors of $\mathbf{X}$ is this vector of interest.



The space $\mathbb{R}^N$ can be broken into two subspaces, one spanned by the column vectors of $\mathbf{X}$, and one comprised of all vectors orthogonal to vectors in the subspace spanned by column vectors of $\mathbf{X}$.

Hastie *et al* (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd Ed.

Concrete example

Consider the set of $N = 3$ training observations at $p = 1$ inputs.

$$\{(x_1, y_1) = (1,1), (x_2, y_2) = (2,4), (x_3, y_3) = (3,4)\}$$

Use least squares to fit the best-fit line to these training data. Based on the fit model, predict the output Y for input $X = 4$.

The input matrix $\mathbf{X}$ and output vector $\mathbf{y}$ of the training data are

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 4 \\ 4 \end{bmatrix}$$

We want to find the vector $\beta = [\beta_0, \beta_1]^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. We start by computing the transpose of the design matrix

$$\mathbf{X}^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

Concrete example

We then compute $\mathbf{X}^T\mathbf{X}$ as

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

Taking the inverse, we obtain

$$(\mathbf{X}^T\mathbf{X})^{-1} = \begin{bmatrix} 7/3 & -1 \\ -1 & 1/2 \end{bmatrix}$$

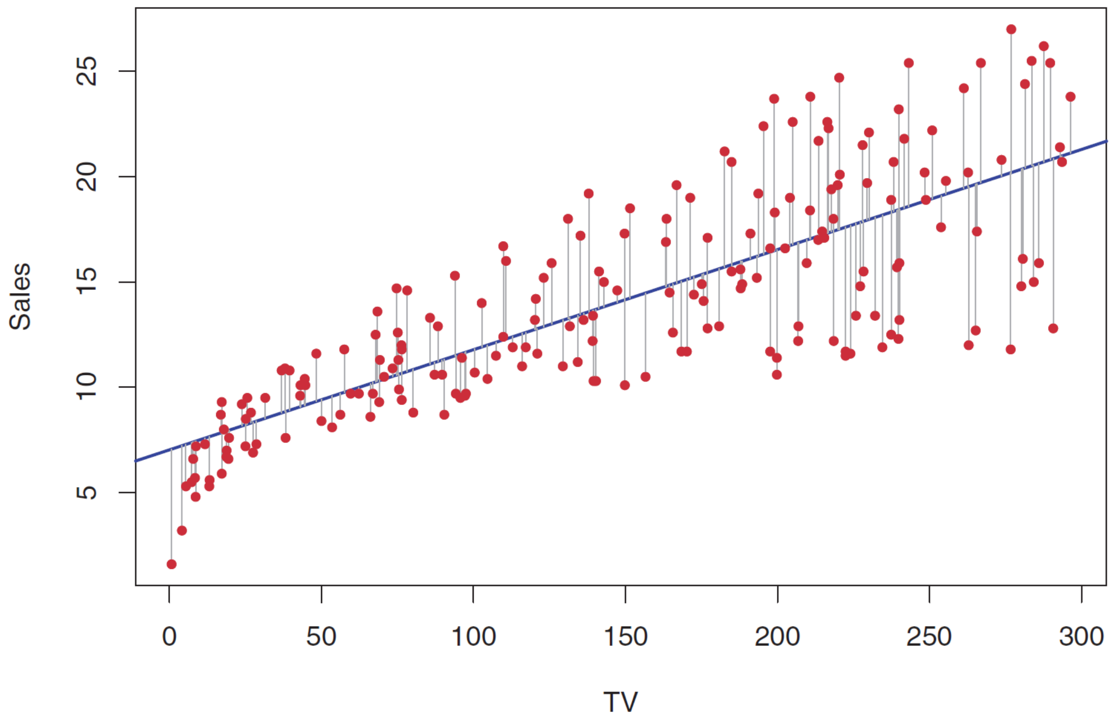
An estimate of β is

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \begin{bmatrix} 7/3 & -1 \\ -1 & 1/2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3/2 \end{bmatrix}$$

yielding $\hat{\beta}_0 = 0$ and $\hat{\beta}_1 = 3/2$. Therefore, the estimate of the output is

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X = (3/2)X = (3/2) \cdot 4 = 6$$

Advertising data example

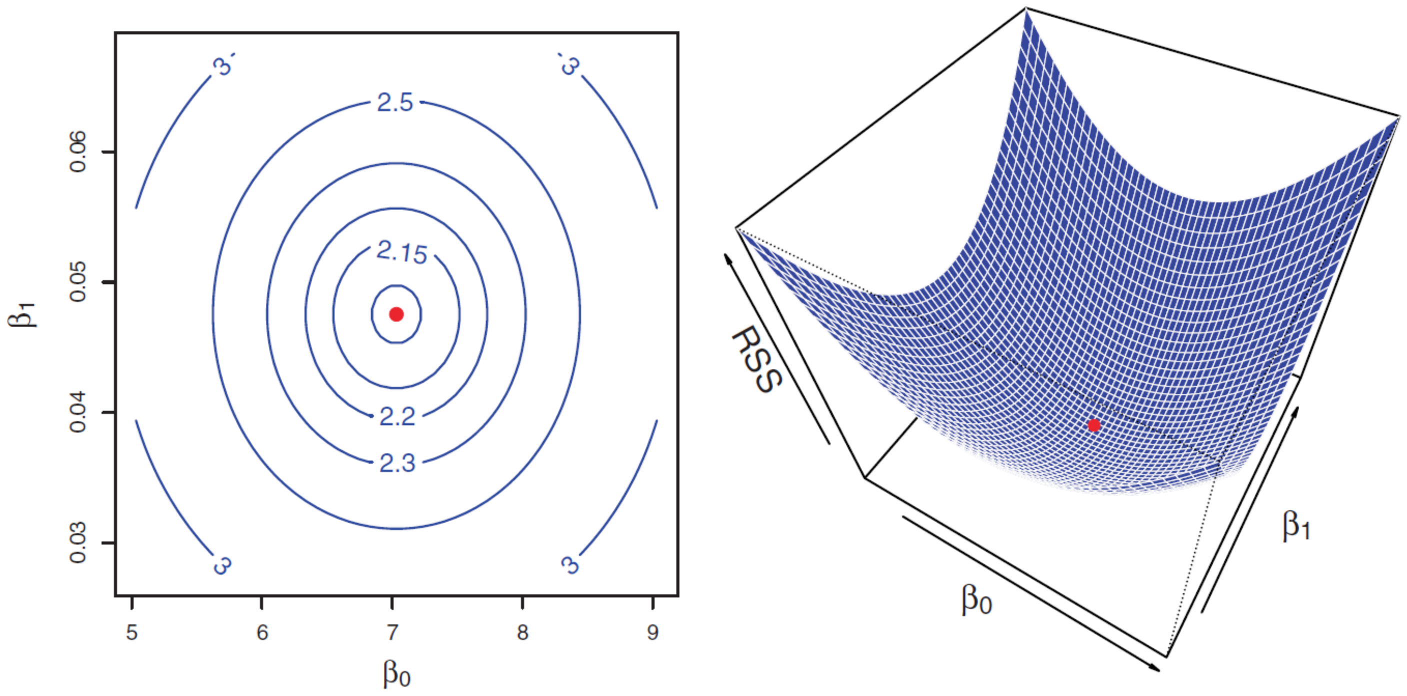


Least squares fit for the regression of sales onto money spent on TV advertising.

Gray segments are residuals (error), with model fit a compromise based on the average (or sum) of the squares of these residuals.

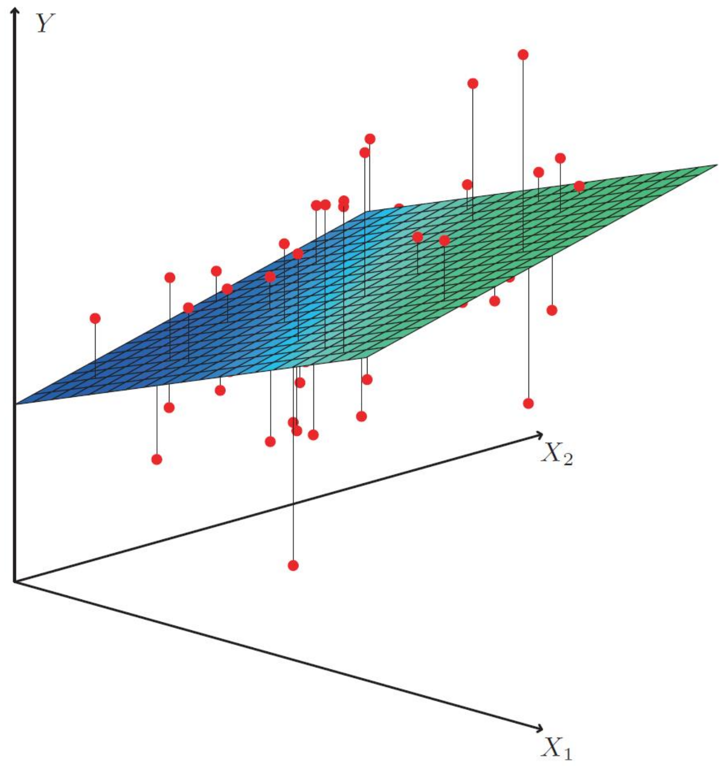
James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Cost function surface



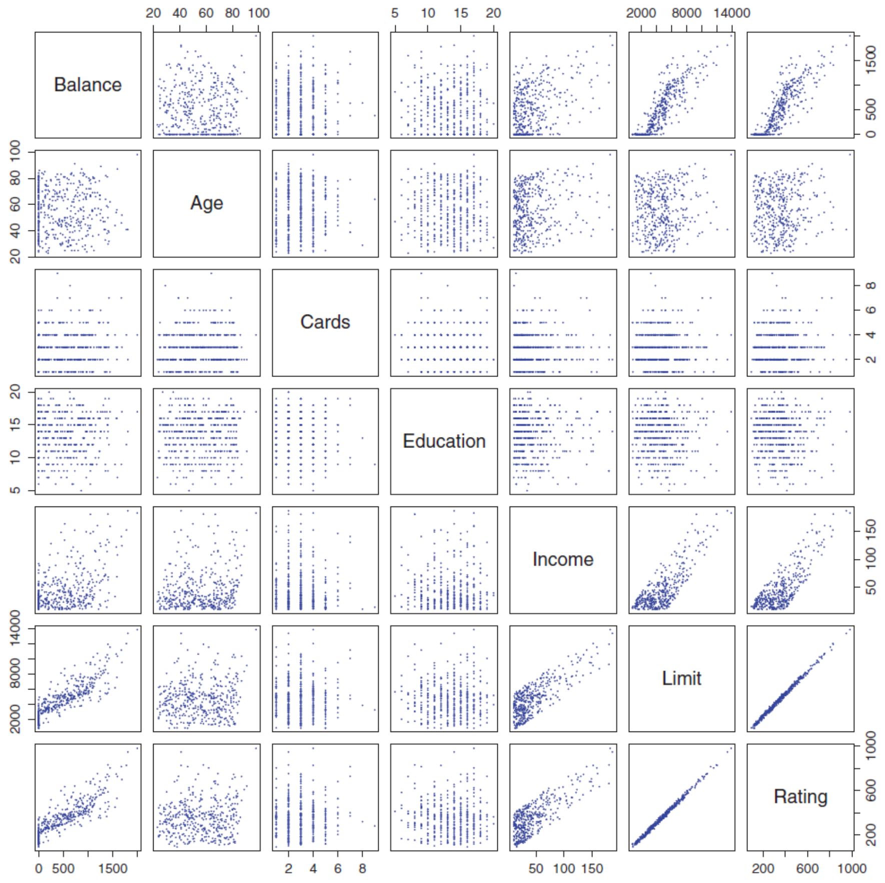
Contour and three-dimensional surface of $J(\beta)$ for the fit of sales as a function of money spent on TV advertising.

Multiple linear regression fit as a function of two inputs



Response Y as a function of two features X_1 and X_2. Rather than a line, the least square solution is a plane, and this plane minimizes the RSS (vertical line segments) of the observations and the plane.

What about qualitative features?



Consider the credit data above, which provides information on credit card balance (or debt) for individuals together with quantitative features, such as age, number of cards, income, etc.

James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Using qualitative features in linear regression

In this credit setting, how can we account for other qualitative (non-quantitative) information that we may have access to?

As an example, what if we had information on student status, gender, or their ethnicity of individuals?

There are straight-forward mechanisms for incorporating such data into our linear regression framework, without making any changes to the setting that we have derived.

Qualitative features with only two levels (binary)

Suppose we had data on $p-1$ quantitative features $\{X_1, X_2, \ldots, X_{p-1}\}$ about an individual's credit, and we want to predict their balance Y.

Moreover, suppose we have information on each individual's student status, and we code this as a **dummy variable** (binary feature) X_p.

Let us assume that we assign the following code

$$X_p = \begin{cases} 1 & \text{student} \\ 0 & \text{non−student} \end{cases}$$

We could have instead coded students as 0 and non-students as 1, and there would be no change in our ability to predict the balance Y.

Example regression conditioning on student status

Suppose we wanted to predict credit balance Y from an individual's income X_1, conditioning on their student status X_2, with

$$X_2 = \begin{cases} 1 & \text{student} \\ 0 & \text{non}-\text{student} \end{cases}$$

The resulting regression model would then be

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 = \begin{cases} (\beta_0 + \beta_2) + \beta_1 X_1 & \text{student} \\ \beta_0 + \beta_1 X_1 & \text{non}-\text{student} \end{cases}$$

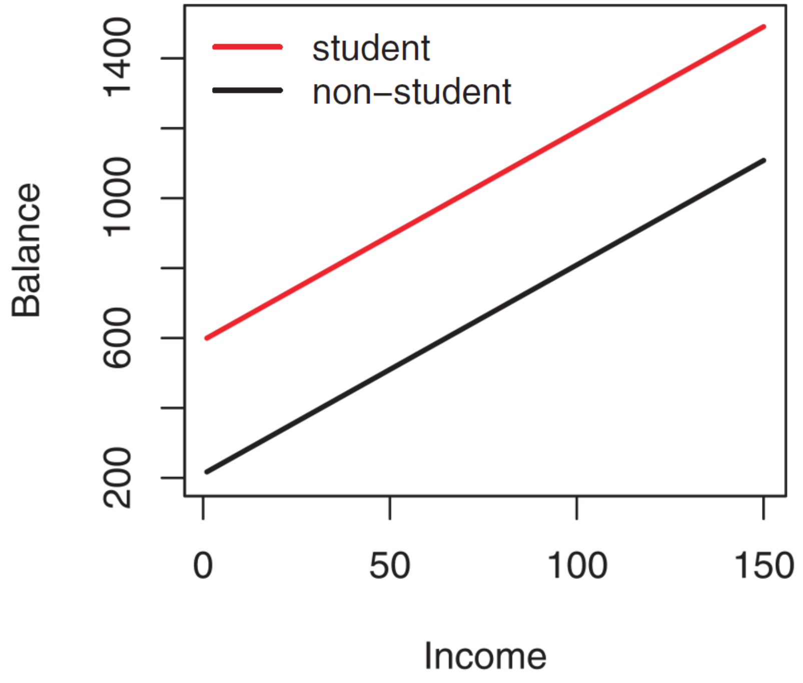
Whether an individual is a student, we are modeling that their balance will change at the same rate as a function of income. However, students and non-students will start at different baselines (intercepts).

The fact that the effect of income (X_1) on balance (Y) is the same regardless of student status could be a severe limitation, as income may have a different influence depending on whether an individual is a student or not.

Example regression conditioning on student status

Fit of regression model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 = \begin{cases} (\beta_0 + \beta_2) + \beta_1 X_1 & \text{student} \\ \beta_0 + \beta_1 X_1 & \text{non-student} \end{cases}$$



The model fits two parallel lines, each with a different intercept.

Permitting different slopes based on student status

In the previous model, the effect of income (X_1) on balance (Y) was the same regardless of student status.

We can therefore modify our regression model to account for this limitation by instead defining the model

$$Y = \beta_0 + \beta_1 X_1 + (\beta_2 + \beta_3 X_1) X_2 = \begin{cases} (\beta_0 + \beta_2) + (\beta_1 + \beta_3) X_1 & \text{student} \\ \beta_0 + \beta_1 X_1 & \text{non-student} \end{cases}$$

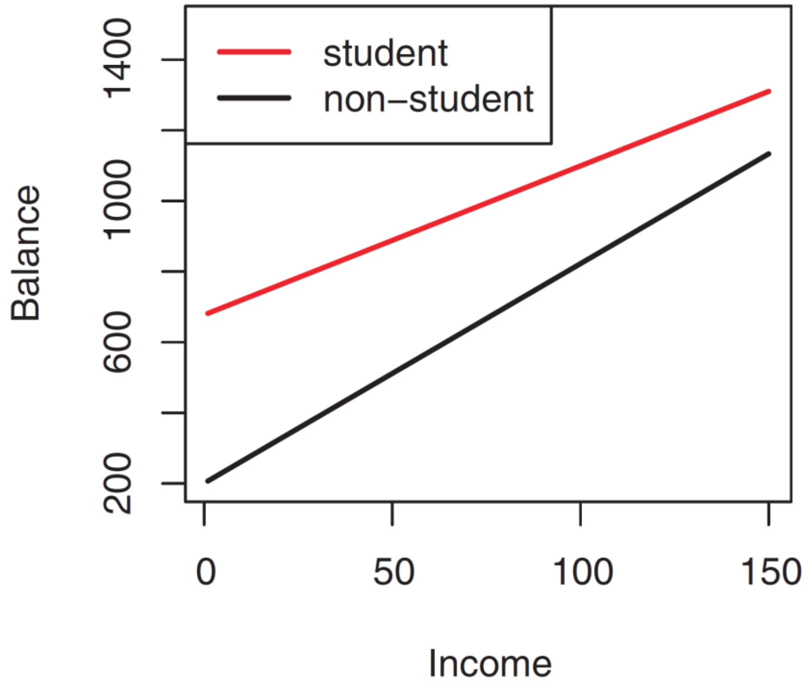
Here student status yields a different intercept or baseline. That is $\beta_0 + \beta_2$ compared with β_0.

More importantly, the effect of income X_1 on balance Y is different depending on student status. That is $\beta_1 + \beta_3$ compared with β_1.

Example with different slope based on student status

Fit of regression model

$$Y = \beta_0 + \beta_1 X_1 + (\beta_2 + \beta_3 X_1) X_2 = \begin{cases} (\beta_0 + \beta_2) + (\beta_1 + \beta_3) X_1 & \text{student} \\ \beta_0 + \beta_1 X_1 & \text{non-student} \end{cases}$$



The model fits two lines, each with a different intercept and slope.

Qualitative features with more than two levels

Suppose in our credit example, we had data on $p-2$ quantitative features $\{X_1, X_2, \ldots, X_{p-2}\}$ to predict credit balance Y.

Moreover, suppose we have information on each individual's environment, which can, say, take values urban, suburban, or rural.

We can encode this information using two dummy variables (features) X_{p-1} and X_p.

Let us assume that we assign the following codes

$$X_{p-1} = \begin{cases} 1 & \text{urban} \\ 0 & \text{not urban} \end{cases} \qquad X_p = \begin{cases} 1 & \text{rural} \\ 0 & \text{not rural} \end{cases}$$

Example regression conditioning on environment

Suppose we wanted to predict credit balance Y from an individual's income X_1, conditioning on their environment, which we code as dummy variables X_2 and X_3, with

$$X_2 = \begin{cases} 1 & \text{urban} \\ 0 & \text{not urban} \end{cases} \qquad X_3 = \begin{cases} 1 & \text{rural} \\ 0 & \text{not rural} \end{cases}$$

The resulting regression model would then be

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 = \begin{cases} (\beta_0 + \beta_2) + \beta_1 X_1 & \text{urban} \\ (\beta_0 + \beta_3) + \beta_1 X_1 & \text{rural} \\ \beta_0 + \beta_1 X_1 & \text{suburban} \end{cases}$$

The model fits three parallel lines, each with a different intercept.

Are normal equations best method to estimate β?

Recall that the normal equations estimate β as

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

A clear merit of this approach is that it is an analytical formula, and does not require approximation or iterative algorithms to compute.

However, consider the computation of $\mathbf{X}^T\mathbf{X}$.

Because $\mathbf{X} \in \mathbb{R}^{N\times(p+1)}$, we have that $\mathbf{X}^T \in \mathbb{R}^{(p+1)\times N}$.

Therefore, $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{(p+1)\times(p+1)}$.

Because many modern applications of machine learning methods are applied to observations of a massive number of features p, then using the normal equations would require a matrix inversion of a huge matrix, which is computationally expensive.

We may instead be able to estimate β with other types of algorithms that will be more computationally feasible on datasets with a large number of features.

Estimating β using an iterative algorithm

We want to identify β to minimize $J(\beta)$.

We will begin with an initial guess $\beta^{(0)}$ of our model parameters (regression coefficients).

We will then estimate a new β, $\beta^{(1)}$, from the previous estimate

$$\beta^{(1)} := f(\beta^{(0)})$$

More generally, from iteration k, we will estimate $\beta^{(k)}$ from $\beta^{(k-1)}$.

$$\beta^{(k)} := f(\beta^{(k-1)})$$

We iterate until convergence, and estimate β as

$$\hat{\beta} = \beta^{(\text{iteration at convergence})}$$

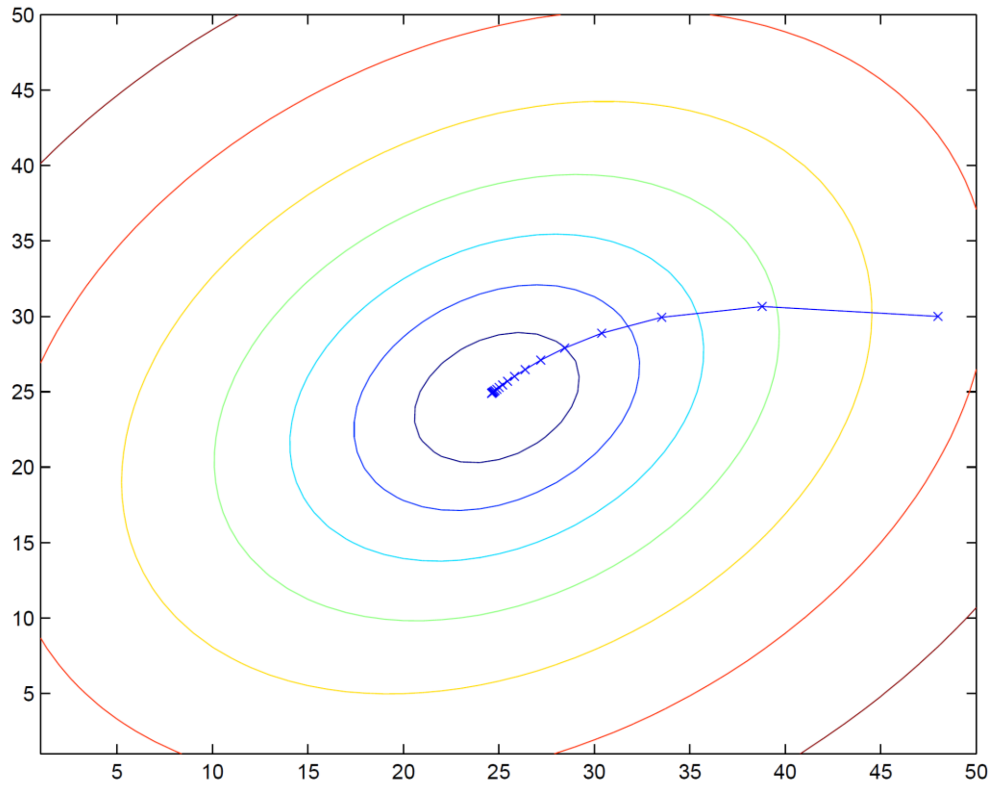
An algorithm that finds a local minimum of our $J(\beta)$ will find the global minimum, as the function is convex.

Example illustration using gradient descent

The iterative algorithm we will consider is **gradient descent**.

Below is a contour plot, of the $J(\beta)$ with two paramaters.

After the initial guess for β, the algorithm iteratively calls itself until it finds the minimum (in the center ellipse.)



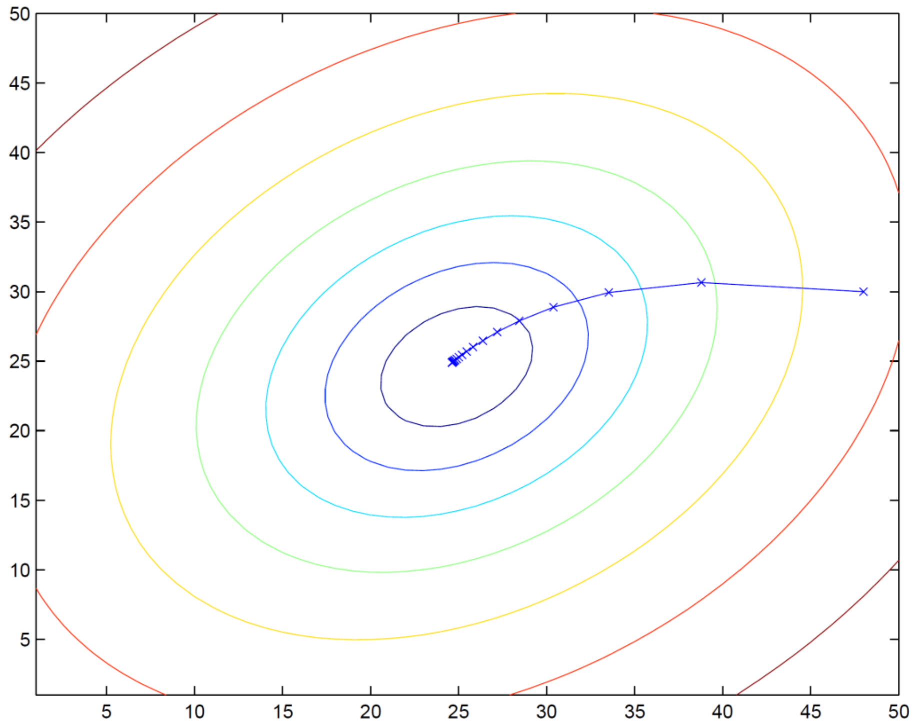
http://cs229.stanford.edu/notes/cs229-notes1.pdf

Finding β that minimizes $J(\beta)$ with gradient descent

To apply gradient descent, we need to compute the gradient vector

$$\frac{\partial J(\beta)}{\partial \beta}$$

The gradient at a point has direction of greatest (steepest) increase of the function at that point.



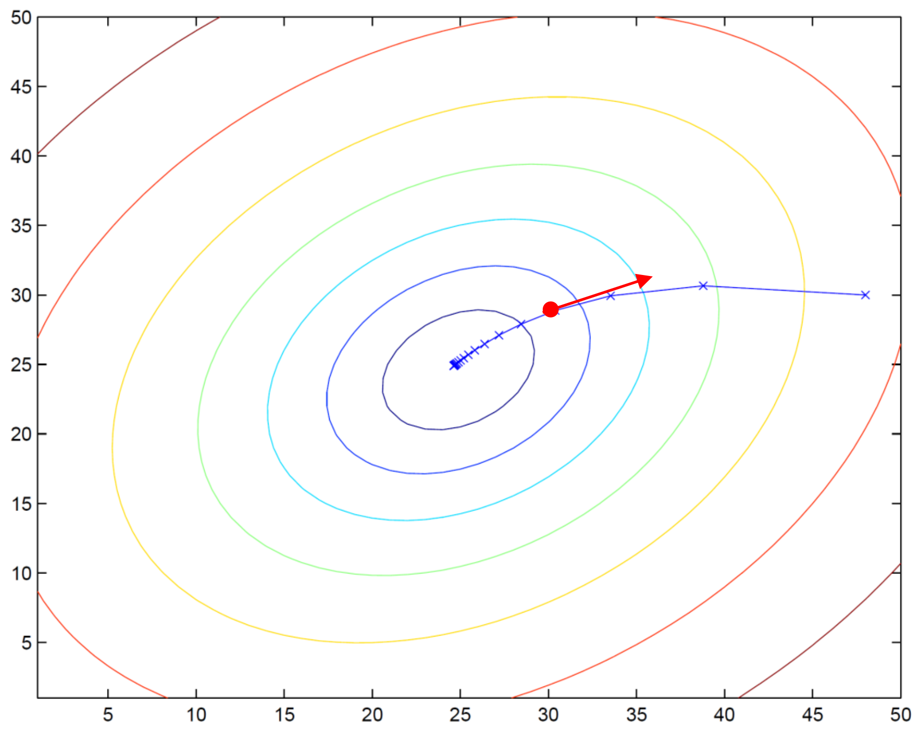
http://cs229.stanford.edu/notes/cs229-notes1.pdf

Finding β that minimizes $J(\beta)$ with gradient descent

To apply gradient descent, we need to compute the gradient vector

$$\frac{\partial J(\beta)}{\partial \beta}$$

The gradient at a point has direction of greatest (steepest) increase of the function at that point.



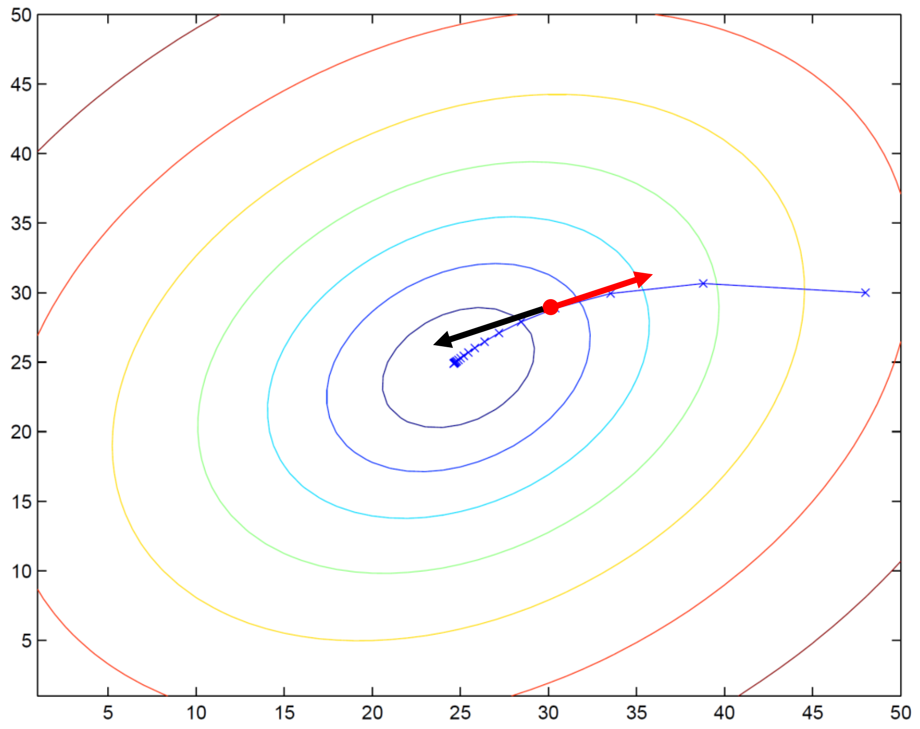
http://cs229.stanford.edu/notes/cs229-notes1.pdf

Finding β that minimizes $J(\beta)$ with gradient descent

To apply gradient descent, we need to compute the gradient vector

$$\frac{\partial J(\beta)}{\partial \beta}$$

The gradient at a point has direction of greatest (steepest) increase of the function at that point.



Move in the **opposite direction** of **gradient** for minimum

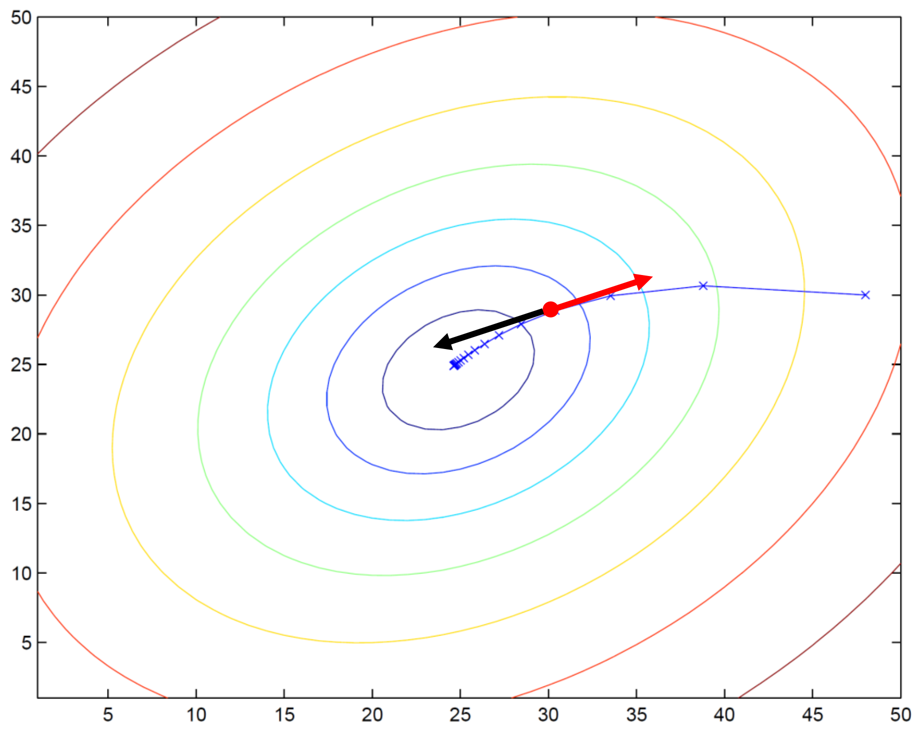
http://cs229.stanford.edu/notes/cs229-notes1.pdf

Finding β that minimizes $J(\beta)$ with gradient descent

How far in the opposite direction of the gradient do we need to move?

This parameter is called the **learning rate**, and we will denote it by α.

Too large α may miss the minimum at iterations close to the minimum, and too small α may take too long to converge.



http://cs229.Stanford.edu/notes/cs229-notes1.pdf

Finding β that minimizes $J(\beta)$ with gradient descent

We formulate the gradient descent update for β_j, $j \in \{0,1,\ldots,p\}$ as

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$

with this update performed for all j simultaneously and where

$$\frac{\partial}{\partial \beta_j} J(\beta) = \begin{cases} -2\sum_{i=1}^{N}(y_i - x_i^T\beta) & j = 0 \\ -2\sum_{i=1}^{N} x_{ij}(y_i - x_i^T\beta) & j = 1,2,\ldots,p \end{cases}$$

which we derived earlier.

Finding β that minimizes $J(\beta)$ with gradient descent

Putting it together, for each iteration, we get as updates

$$\beta_j := \begin{cases} \beta_0 + 2\alpha \sum_{i=1}^{N} (y_i - x_i^T \beta) & j = 0 \\ \beta_j + 2\alpha \sum_{i=1}^{N} x_{ij}(y_i - x_i^T \beta) & j = 1,2,\dots,p \end{cases}$$

and we compute these updates for all parameters j at each iteration.

Whereas the normal equations would be computationally difficult for millions of parameters, this simple procedure can be followed regardless of the number of parameters.

Hence, such iterative algorithms are used in modern machine learning applications where the number of parameters is enormous.

Issues with gradient descent

There are a couple pitfalls associated with gradient descent.

The first is that the **batch gradient descent** algorithm (update equations below) require that data from all observed data points be used simultaneously for the update.

$$\beta_j := \begin{cases} \beta_0 + 2\alpha \sum_{i=1}^{N} (y_i - x_i^T \beta) & j = 0 \\ \beta_j + 2\alpha \sum_{i=1}^{N} x_{ij}(y_i - x_i^T \beta) & j = 1,2,\ldots,p \end{cases}$$

This simultaneous (batch) update is problematic if there are many observed data points, and would require the model to be refitted if new data points become available (also known as **online learning** and is common in many modern machine learning applications).

We can address this issue with **stochastic gradient descent**, approximates the gradient of the cost function $J(\beta)$

Finding β with stochastic gradient descent

1. Randomly initialize β.
2. Randomly permute (shuffle) the order of the N training observations.
3. For observation i in the permuted list of training observations, update each of the p parameters as

$$\beta_j := \begin{cases} \beta_0 + 2\alpha(y_i - x_i^T\beta) & j = 0 \\ \beta_j + 2\alpha x_{ij}(y_i - x_i^T\beta) & j = 1,2,\ldots,p \end{cases}$$

4. Repeat steps 2 and 3 until convergence.

Finding β with mini-batch gradient descent

Each observation makes a small update to the training parameters.

Stochastic gradient descent will often get close to the minimum of $J(\beta)$ substantially faster than batch gradient descent.

One pitfall is that stochastic gradient descent many not converge to a solution, with estimates oscillating around the minimum.

A single training observation may not approximate the gradient well, yet employing the entire training set can be computationally prohibitive for large N.

Mini-batch gradient descent strikes a compromise between stochastic and batch gradient descent, by approximating the gradient with a smaller subset n of the full set of N training observations.

At each iteration this subset is chosen uniformly at random from the larger set.

Finding β with mini-batch gradient descent

1. Randomly initialize β.
2. Choose a subset $\mathcal{S}$ of size n of the N training observations uniformly at random.
3. Approximate the gradient with these n observations and update each of the p parameters as

$$\beta_j := \begin{cases} \beta_0 + 2\alpha \sum_{(x_i,y_i)\in\mathcal{S}} (y_i - x_i^T\beta) & j = 0 \\ \beta_j + 2\alpha \sum_{(x_i,y_i)\in\mathcal{S}} x_{ij}(y_i - x_i^T\beta) & j = 1,2,\ldots,p \end{cases}$$

4. Repeat steps 2 and 3 until convergence.

Choosing the learning rate α

As stated earlier, too large α may miss the minimum at iterations close to the minimum, and too small α may take too long to converge.

A solution is to have a **learning rate schedule**, such that α decreases with increasing iterations.

These learning rate schedules allow for large steps in the beginning to get close to the minimum, and small steps later on to fine-tune the parameter estimates.

Such a learning schedule can be used to help stochastic gradient descent converge to the minimum in situations where it oscillates.

Why is least squares linear regression reasonable?

A natural question is why is least squares linear regression a reasonable model?

To try to answer this question, we will begin by assuming that for each training observation $i \in \{1,2,\ldots,N\}$

$$y_i = x_i^T \beta + \epsilon_i$$

where ϵ_i is an error term associated with observation i.

The quantity ϵ_i accounts for random noise as well as effects on y_i that are not modeled.

Modeling the error term ϵ_i

To model ϵ_i, we will make some assumptions about its distribution.

We will assume that the ϵ_i are independent across the training datapoints, and that they are identically distributed (IID), with distribution following a Gaussian (normal) distribution with mean 0 and variance σ^2.

We write this as $\epsilon_i \sim N(0, \sigma^2)$.

Because each error term is distributed $N(0, \sigma^2)$, their probability densities are

$$f(\epsilon_i) = f(\epsilon_i; 0, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right)$$

Computing the likelihood of β

Because $\epsilon_i = y_i - x_i^T\beta$, we can write

$$f(y_i|x_i;\beta) = f(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\left(y_i - x_i^T\beta\right)^2}{2\sigma^2}\right)$$

which represents the density of the output y_i, given the input values x_i, and which is parameterized by β.

Given input matrix $\mathbf{X}$ and associated output vector $\mathbf{y}$ from the training data, the likelihood of the parameters β (also the probability density of the output $\mathbf{y}$ given data $\mathbf{X}$ and parameters β) is

$$L(\beta) = L(\beta;\mathbf{X},\mathbf{y}) = f(\mathbf{y}|\mathbf{X};\beta)$$

Maximum likelihood estimate of the parameters

Using the fact that ϵ_i are IID, we have

$$L(\beta) = \prod_{i=1}^{N} f(y_i | x_i; \beta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - x_i^T\beta\right)^2}{2\sigma^2}\right)$$

We can then estimate β by maximizing the likelihood function $L(\beta)$, which is equivalent to maximizing the log likelihood

$$\ell(\beta) = \log L(\beta) = \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - x_i^T\beta\right)^2}{2\sigma^2}\right)$$

$$= N \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(y_i - x_i^T\beta\right)^2$$

$$= N \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \mathrm{RSS}(\beta)$$

which is maximized when $\mathrm{RSS}(\beta)$ is minimized.