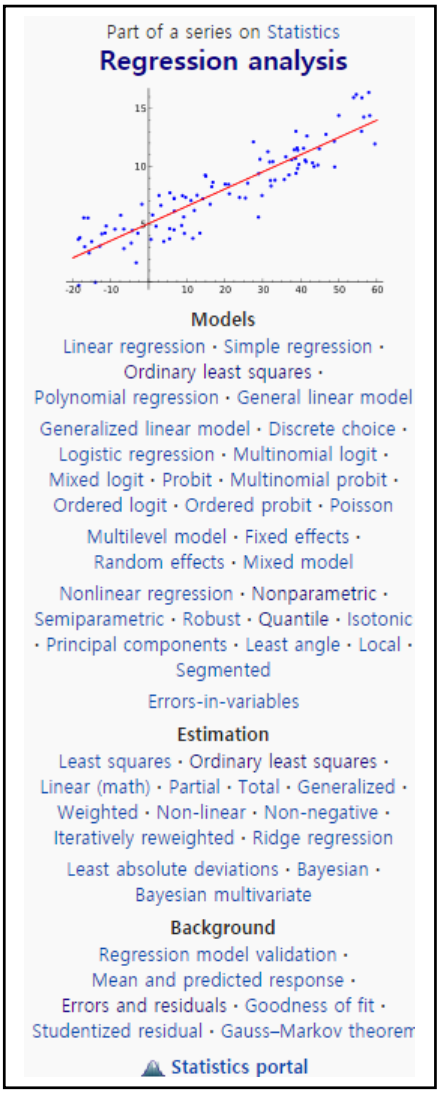
권도형

Regression :



the relationships among variables. the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors').

Regression models involve the following variables:

- The **unknown parameters**, denoted as **β**, which may represent a scalar or a vector.
- The **independent variables**, **X**.
- The **dependent variable**, *Y*.

A regression model relates *Y* to a function of **X** and **β**.

- $Y \approx f(\mathbf{X}, \boldsymbol{\beta})$

To carry out regression analysis, the form of the function *f* must be specified.

Assume now that the vector of unknown parameters **β** is of length *k*. In order to perform a regression analysis the user must provide information about the dependent variable *Y*:

- If *N* data points of the form (*Y*, **X**) are observed, where $N < k$, most classical approaches to regression analysis cannot be performed: since the system of equations defining the regression model is underdetermined, there are not enough data to recover **β**.
- If exactly $N = k$ data points are observed, and the function *f* is linear, the equations $Y = f(\mathbf{X}, \boldsymbol{\beta})$ can be solved exactly rather than approximately. This reduces to solving a set of *N* equations with *N* unknowns (the elements of **β**), which has a unique solution as long as the **X** are linearly independent. If *f* is nonlinear, a solution may not exist, or many solutions may exist.
- The most common situation is where $N > k$ data points are observed. In this case, there is enough information in the data to estimate a unique value for **β** that best fits the data in some sense, and the regression model when applied to the data can be viewed as an overdetermined system in **β**.

In the last case, the regression analysis provides the tools for:

1. Finding a solution for unknown parameters **β** that will, for example, minimize the distance between the measured and predicted values of the dependent variable *Y* (also known as method of least squares).
2. Under certain statistical assumptions, the regression analysis uses the surplus of information to provide statistical information about the unknown parameters **β** and predicted values of the dependent variable *Y*.

변수들 사이의 관계. 종속 변수와 하나 이상의 독립 변수 사이의 관계에 관심의 초점이 있다. 회귀 모델에는 다음의 변수들이 포함된다; "unknown parameters", 벡터 또는 스칼라를 나타내며, '베타'로 표현된다. "independent variables", 독립 변수, X "dependent variables", 종속 변수, Y

회귀 분석을 수행하기 위해서는 X와 베타로 이루어진 함수 f가 구체적으로 기술되어져야 한다.

권도형

벡터 베타가 k의 길이라고 가정한다. 회귀 분석을 수행하기 위해서는 Y에 대한 정보가 제공되어져야만 한다.

만일 관측된 데이터의 개수가 N일 때 N의 개수가 k보다 작다면, 대부분의 방법들은 적용되지 않는다. 데이터의 개수가 충분하지 않다.

Necessary number of independent measurements

Consider a regression model which has three unknown parameters, β_0, β_1, and β_2. Suppose an experimenter performs 10 measurements all at exactly the same value of independent variable vector **X** (which contains the independent variables X_1, X_2, and X_3). In this case, regression analysis fails to give a unique set of estimated values for the three unknown parameters; the experimenter did not provide enough information. The best one can do is to estimate the average value and the standard deviation of the dependent variable *Y*. Similarly, measuring at two different values of **X** would give enough data for a regression with two unknowns, but not for three or more unknowns.

If the experimenter had performed measurements at three different values of the independent variable vector **X**, then regression analysis would provide a unique set of estimates for the three unknown parameters in **β**.

In the case of general linear regression, the above statement is equivalent to the requirement that the matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is invertible.

Statistical assumptions

When the number of measurements, *N*, is larger than the number of unknown parameters, *k*, and the measurement errors ε_i are normally distributed then *the excess of information* contained in (*N* − *k*) measurements is used to make statistical predictions about the unknown parameters. This excess of information is referred to as the degrees of freedom of the regression.

Linear regression

In linear regression, the model specification is that the dependent variable, y_i is a linear combination of the *parameters* (but need not be linear in the *independent variables*). For example, in simple linear regression for modeling n data points there is one independent variable: x_i, and two parameters, β_0 and β_1:

straight line:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \dots, n.$$

In multiple linear regression, there are several independent variables or functions of independent variables.

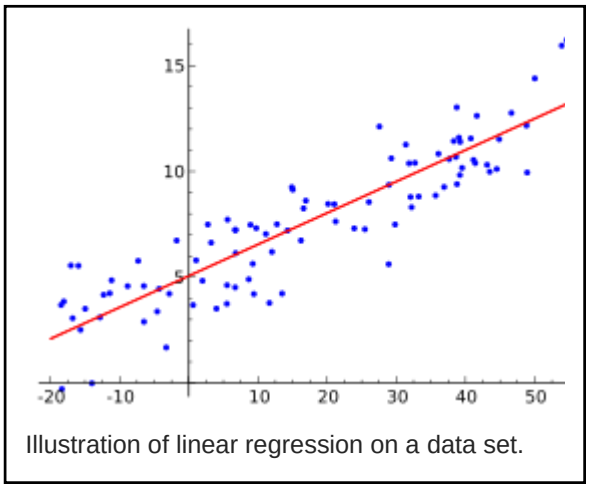


Illustration of linear regression on a data set.

Adding a term in x_i^2 to the preceding regression gives:

parabola: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, \ i = 1, \dots, n.$

권도형

This is still linear regression; although the expression on the right hand side is quadratic in the independent variable x_i, it is linear in the parameters β_0, β_1 and β_2.

In both cases, ε_i is an error term and the subscript i indexes a particular observation.

Returning our attention to the straight line case: Given a random sample from the population, we estimate the population parameters and obtain the sample linear regression model:

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

The residual, $e_i = y_i - \widehat{y}_i$, is the difference between the value of the dependent variable predicted by the model, $\widehat{y}_i$, and the true value of the dependent variable, y_i. One method of estimation is ordinary least squares. This method obtains parameter estimates that minimize the sum of squared residuals, SSE,[22][23] also sometimes denoted RSS:

$$SSE = \sum_{i=1}^{n} e_i^2.$$

Minimization of this function results in a set of normal equations, a set of simultaneous linear equations in the parameters, which are solved to yield the parameter estimators, $\widehat{\beta}_0, \widehat{\beta}_1$.

In the case of simple regression, the formulas for the least squares estimates are

$$\widehat{\beta}_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \text{ and } \hat{\beta}_0 = \bar{y} - \widehat{\beta}_1 \bar{x}$$

where $\bar{x}$ is the mean (average) of the x values and $\bar{y}$ is the mean of the y values.

Under the assumption that the population error term has a constant variance, the estimate of that variance is given by:

$$\hat{\sigma}_\varepsilon^2 = \frac{SSE}{n-2}.$$

This is called the mean square error (MSE) of the regression. The denominator is the sample size reduced by the number of model parameters estimated from the same data, (*n*-*p*) for *p* regressors or (*n*-*p*-1) if an intercept is used.[24] In this case, *p*=1 so the denominator is *n*-2.

The standard errors of the parameter estimates are given by

$$\hat{\sigma}_{\beta_0} = \hat{\sigma}_\varepsilon \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum(x_i - \bar{x})^2}}$$

$$\hat{\sigma}_{\beta_1} = \hat{\sigma}_\varepsilon \sqrt{\frac{1}{\sum(x_i - \bar{x})^2}}.$$

Under the further assumption that the population error term is normally distributed, the researcher can use these estimated standard errors to create confidence intervals and conduct hypothesis tests about the population parameters.

General linear model

In the more general multiple regression model, there are *p* independent variables:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i,$$

where x_{ij} is the i^{th} observation on the j^{th} independent variable, and where the first independent variable takes the value 1 for all i (so β_1 is the regression intercept).

The least squares parameter estimates are obtained from p normal equations. The residual can be written as

$$\varepsilon_i = y_i - \hat{\beta}_1 x_{i1} - \cdots - \hat{\beta}_p x_{ip}.$$

The **normal equations** are

$$\sum_{i=1}^{n} \sum_{k=1}^{p} X_{ij} X_{ik} \hat{\beta}_k = \sum_{i=1}^{n} X_{ij} y_i, \ j = 1, \ldots, p.$$

In matrix notation, the normal equations are written as

$$(\mathbf{X}^\top \mathbf{X}) \hat{\boldsymbol{\beta}} = \mathbf{X}^\top \mathbf{Y},$$

where the ij element of X is x_{ij}, the i element of the column vector Y is y_i, and the j element of $\hat{\beta}$ is $\hat{\beta}_j$. Thus X is $n \times p$, Y is $n \times 1$, and $\hat{\beta}$ is $p \times 1$. The solution is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}.$$

Diagnostics

Once a regression model has been constructed, it may be important to confirm the goodness of fit of the model and the statistical significance of the estimated parameters. Commonly used checks of goodness of fit include the R-squared, analyses of the pattern of residuals and hypothesis testing. Statistical significance can be checked by an F-test of the overall fit, followed by t-tests of individual parameters.

Nonlinear regression

When the model function is not linear in the parameters, the sum of squares must be minimized by an iterative procedure. This introduces many complications which are summarized in Differences between linear and non-linear least squares

Power and sample size calculations

There are no generally agreed methods for relating the number of observations versus the number of independent variables in the model. One rule of thumb suggested by Good and Hardin is $N = m^n$, where N is the sample size, n is the number of independent variables and m is the number of observations needed to reach the desired precision if the model had only one independent variable.[26] For example, a researcher is building a linear regression model using a dataset that contains 1000 patients (N). If the researcher decides that five observations are needed to precisely define a straight line (m), then the maximum number of independent variables the model can support is 4, because

$$\frac{\log 1000}{\log 5} = 4.29.$$

- Curve fitting

권도형

- Estimation Theory
- Forecasting
- Fraction of variance unexplained
- Function approximation
- Generalized linear models
- Kriging (a linear least squares estimation algorithm)
- Local regression
- Modifiable areal unit problem
- Multivariate adaptive regression splines
- Multivariate normal distribution
- Pearson product-moment correlation coefficient
- Prediction interval
- Regression validation
- Robust regression
- Segmented regression
- Signal processing
- Stepwise regression
- Trend estimation

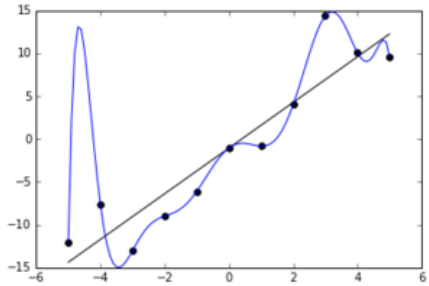
Curve fitting

Curve fitting[1][2] is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points. Fitted curves can be used to summarize the relationships among two or more variables.

Overfitting

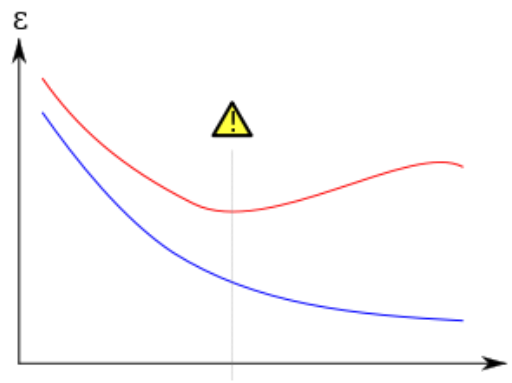
In statistics and machine learning, one of the most common tasks is to fit a "model" to a set of training data, so as to be able to make reliable predictions on general untrained data.

In **overfitting**, a statistical model describes random error or noise instead of the underlying relationship. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor



Noisy (roughly linear) data is fitted to both linear and polynomial functions. Although the polynomial function is a perfect fit, the linear version can be expected to generalize better. In other words, if the two functions were used to extrapolate the data beyond the fit data, the linear function would make better predictions.

권도형



Overfitting/overtraining in supervised learning (e.g., neural network). Training error is shown in blue, validation error in red, both as a function of the number of training cycles. If the validation error increases(positive slope) while the training error steadily decreases(negative slope) then a situation of overfitting may have occurred. The best predictive and fitted model would be where the validation error has its global minimum.

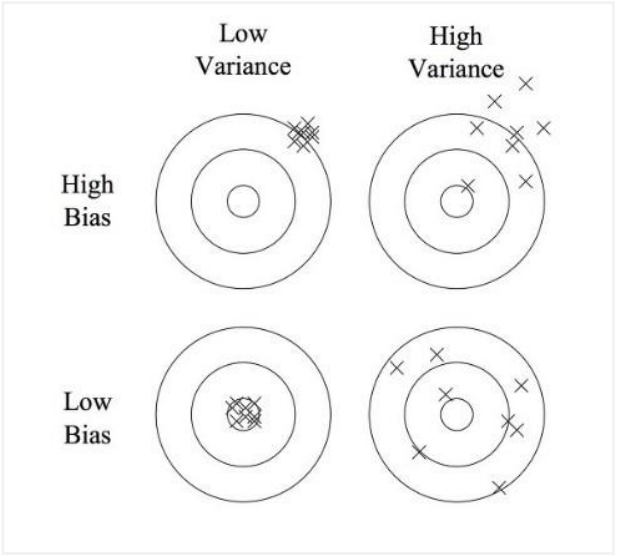
fluctuations in the training data. 과적합이 아니면서도 예측에 적합한 회귀선을 찾기 위해선 전체 dataset을 Training set과 Test set으로 나누어야 한다. Train set은 RSS를 줄이기 위한 파라미터를 찾게 하고 test set은 모델 평가에 쓰인다. 단, train set을 너무 크게하면 overfitting에 빠지게 된다. 가장 이상적인 train set과 test set의 비율은 7 : 3 이다.

권도형

Bias-Variance Trade-off

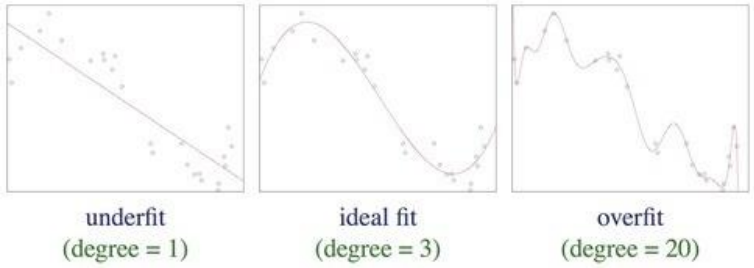
모든 학습 알고리즘은 세 가지 종류의 에러를 가지고 있다; Bias, Variance, Noise. noise는 데이터가 태생적으로 가지고 있는 한계치로서 irreducible error라고 불리며 bias와 variance는 모델에 따라 변하기 때문에 reducible error라고 불린다.

$$\mathbf{Error}(\mathbf{x}) = \boldsymbol{noise}(\boldsymbol{x}) + \boldsymbol{bias}(\boldsymbol{x}) + \boldsymbol{variance}(\boldsymbol{x})$$



Bias error는 왜 발생하는가? Bias는 모든 데이터를 고려하지 않음으로 인해 지속적으로 잘못된 것을 학습한 결과 잘못된 편향성을 갖게 된 것을 말한다. 제대로 fitting되지 않은 것이라고 할 수 있다.(지나친 보편성;under-fitting) 반대로 Variance error가 발생하는 이유는 데이터 내에 있는 noise까지 모델에 fitting 시킴으로써 실제 현상과는 관계 없는 것들까지 학습하여 결국 예측에 쓰기엔 적합하지 않을 정도로 정확도가 지나치게 높아진 것을 말한다.(과적합;over-fitting)

Bias와 Variance가 어떤 느낌인지 알 수 있다. 지나치게 편향성이 높은(High Bias)경우는 예측치들(x표)이 특정 영역에 편향되어져 있는 것을 볼 수 있다. 반면 편향성이 높지 않은 경우에는 편향성이 아닌 중립성을 지키는 것으로 보인다. Variance의 경우엔 이들 예측치들이 지나치게 한 경우에만 적합한 경우(High Variance) 마치 다트게임의 목표물을 상징하듯 한 곳에만 집중적으로 몰려져 있는 것을 볼 수 있다. 반면 Variance가 높을 경우 분산의 정도가 커보인다.



선형 모델 (degree=1) 은 under-fit 이다:

1) 이 모델은 데이터 내의 모든 정보를 고려하지 못하고 있다 (high bias). 하지만, 2) 새로운 데이터가 들어온다 하더라도 이 모델의 형태는 크게 변하지 않을 것이다 (low variance).

반면에, 고차 다항함수 모델 (degree=20) 은 over-fit 이다:

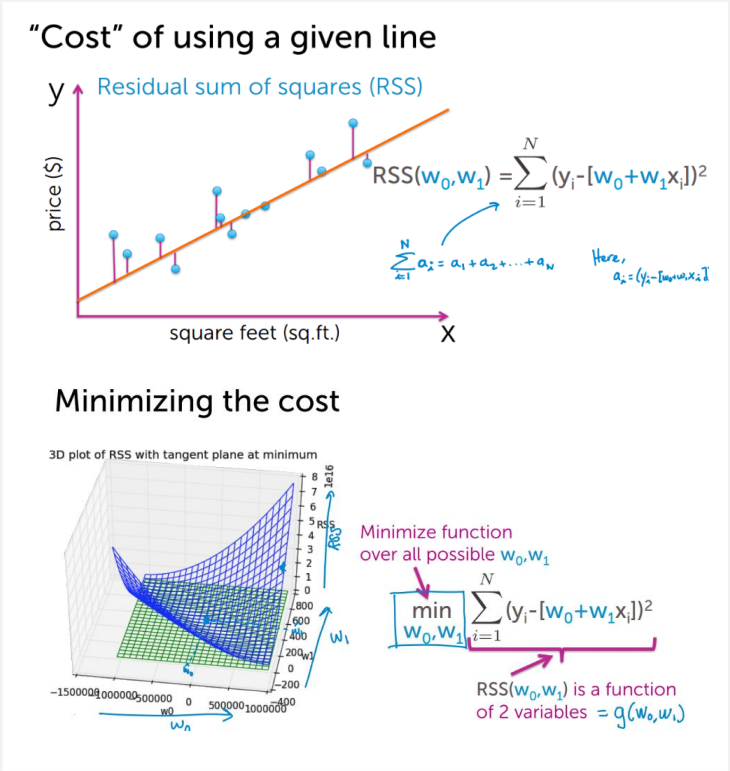
1) 이 곡선 모델은 주어진 데이터를 잘 설명하고 있다 (low bias). 하지만, 2) 이 함수는 새로운 데이터가 들어왔을 때 완전히 다른 형태로 변하게 되고, generality 를 잃게 될 것이다 (high variance).

즉, 이상적인 모델은 데이터의 규칙성을 잘 잡아내어 정확하면서도 다른 데이터가 들어왔을 때에도 잘 일반화할 수 있는 모델일 것이다 (degree=3). 하지만, 실제 상황에서는 두 가지를 동시에 만족하는 것은 거의 불가능하다. 따라서 트레이닝 데이터가 아닌 실제 데이터에서 좋은 성능을 내기 위해 이런 tradeoff는 반드시 생길 수 밖에 없으며 이는 bias-variance trade-off 라고 불린다. (A Few Useful Things to Know about Machine Learning(bias-variance tradeoff 에 관련된 논문) 참고)

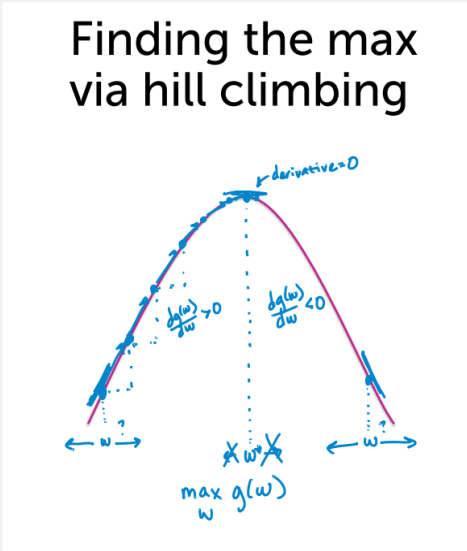
권도형

Cost function

In mathematical optimization, the loss function, a function to be minimized.

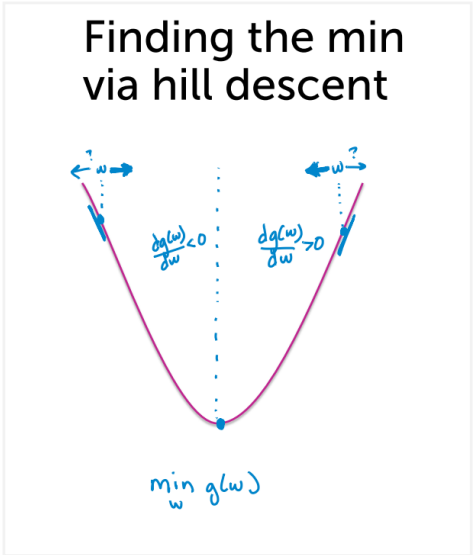


미분 원리(최대값, 최소값 찾기)



최대값을 찾기 위해 w값을 조금씩 optimum 에 가깝도록 이동시키고 싶다고 할 때, w를 증가시켜야 할까, 감소시켜야 할까? (w를 오른쪽으로 이동시켜야 할까, 왼쪽으로 이동시켜야 할까?)

그건 아마 w의 시작 지점과 미분의 부호에 의해 결정될 것이다. 최대값을 찾기 위해서는 미분의 결과가 positive하다면 w를 increase하는 방향으로(직관적으로 봤을 때 +와 increase이기 때문에 둘은 같은 방향), 미분의 결과가 negative하다면 w를 decrease하는 방향으로 이동시켜야 optimum max값에 가까워질 수 있다. 즉 미분한 결과가 이동 방향을 결정하는 것을 볼 수 있다. 이 때 얼마나 이동해야 할까? 이동할 양은 step size etha(η)에 의해 결정된다. 결국 미분은 이동할 방향을, η 는 이동할 양을 결정하는 것이다. $\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^{t} + \eta \nabla f_{\text{obj}}(\boldsymbol{\theta}^{t})$



같은 원리가 최소값 찾기에도 똑같이 적용된다. 감소의 경우엔, 미분의 결과가 negative할 때 w를 increase해야 하며, 미분의 결과가 positive할 땐 w를 decrease해야 optimum min값에 가까워진다(we want to move against the gradient. down toward the minimum.).

즉 미분에 의해 정해지는 w의 이동 방향이 w가 감소하는 방향이어야 한다.

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \gamma_i \nabla f(\mathbf{x}_i)$$

Step size

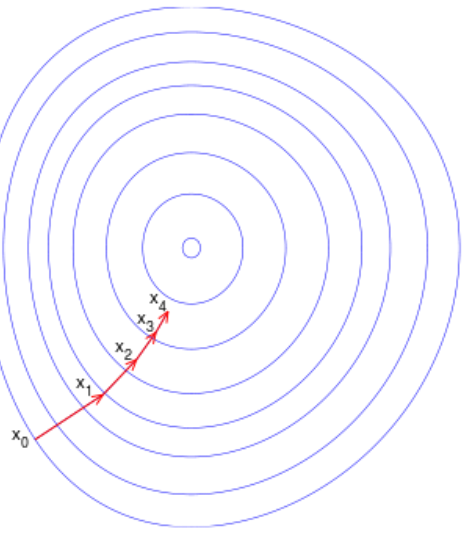
step size 를 정하는 방법은 두 가지다; Fixed step-size, Decreasing step-size.

Fixed step-size는 지그재그로 optimum에 접근한다. decreasing step-size 방법은 반복 횟수에 따라 step-size가 점차 감소하게 된다. 하지만 너무 느리다는 단점이 있다. 두 방법 중에 어느 것을 선택 하는가는 자유다.

권도형

Gradient Descent

To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point. Gradient descent methods aim to find a local minimum of a function by iteratively taking steps in the direction of the negative gradient of the function at the current point, i.e., the current parameter value. Gradient descent is also known as **steepest descent**, or the **method of steepest descent**. Gradient descent should not be confused with the method of steepest descent for approximating integrals.



A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient *descent* leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

one starts with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots$ such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$

so hopefully the sequence $(\mathbf{x}_n)$ converges to the desired local minimum. Note that the value of the *step size* γ is allowed to change at every iteration. With certain assumptions on the function F (for example, F convex and ∇F Lipschitz) and particular choices of γ (e.g., chosen via a line search that satisfies the Wolfe conditions), convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

Limitations

gradient descent is relatively slow close to the minimum. Technically, its asymptotic rate of convergence is inferior to many other methods. Non-descent methods, like subgradientprojection methods, may also be used.[1] These methods are typically slower than gradient descent. For solving linear equations, gradient descent is rarely used, with the conjugate gradient method being one of the most popular alternatives. The speed of convergence of gradient descent depends on the maximal and minimaleigenvalues of A, while the speed of convergence of conjugate gradients has a more complex dependence on the eigenvalues, and can benefit from preconditioning. Gradient descent also benefits from preconditioning, but this is not done as commonly.

Gradient descent can also be used to solve a system of nonlinear equations. Below is an example that shows how to use the gradient descent to solve for three unknown variables, x_1, x_2, and x_3. This example shows one iteration of the gradient descent.

Consider a nonlinear system of equations:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

권도형

suppose we have the function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

and the objective function

$$F(\mathbf{x}) = \tfrac{1}{2} G^{\mathrm{T}}(\mathbf{x}) G(\mathbf{x}) = \tfrac{1}{2} \left(\left(3x_1 - \cos(x_2 x_3) - \tfrac{3}{2}\right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1\right)^2 + \left(e\right.$$

With initial guess

$$\mathbf{x}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \nabla F(\mathbf{x}^{(0)})$$

where

$$\nabla F(\mathbf{x}^{(0)}) = J_G(\mathbf{x}^{(0)})^{\mathrm{T}} G(\mathbf{x}^{(0)})$$

The Jacobian matrix $J_G(\mathbf{x}^{(0)})$

$$J_G = \begin{bmatrix} 3 & \sin(x_2 x_3) x_3 & \sin(x_2 x_3) x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 20 \end{bmatrix}$$

Then evaluating these terms at $\mathbf{x}^{(0)}$

$$J_G\left(\mathbf{x}^{(0)}\right) = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 20 \end{bmatrix}, \qquad G(\mathbf{x}^{(0)}) = \begin{bmatrix} -2.5 \\ -1 \\ 10.472 \end{bmatrix}$$

So that

$$\mathbf{x}^{(1)} = 0 - \gamma_0 \begin{bmatrix} -7.5 \\ -2 \\ 209.44 \end{bmatrix}.$$

and

$$F\left(\mathbf{x}^{(0)}\right) = 0.5\left((-2.5)^2 + (-1)^2 + (10.472)^2\right) = 58.456$$

권도형

Now a suitable γ_0 must be found such that $F(\mathbf{x}^{(1)}) \leq F(\mathbf{x}^{(0)})$. This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001$ which gives

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{bmatrix}$$

evaluating at this value,

$$F\left(\mathbf{x}^{(1)}\right) = 0.5\left((-2.48)^2 + (-1.00)^2 + (6.28)^2\right) = 23.306$$

The decrease from $F(\mathbf{x}^{(0)}) = 58.456$ to the next step's value of $F(\mathbf{x}^{(1)}) = 23.306$ is a sizable decrease in the objective function. Further steps would reduce its value until a solution to the system was found.

Algorithms

Algorithm:

while not converged

$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla g(w^{(t)})$

$$\begin{bmatrix} \vdots \end{bmatrix} \leftarrow \begin{bmatrix} \vdots \end{bmatrix} - \eta \begin{bmatrix} \vdots \end{bmatrix}$$

Convergence: $\|\nabla g(w)\| < \epsilon$

$$\text{RSS}(w_0, w_1) = \sum_{i=1}^{N} (y_i - [w_0 + w_1 x_i])^2$$

Where $(y_i - [w_0 + w_1 x_i])^2 = g_i(w)$,

$$\begin{aligned} \frac{d}{dw} \sum_{i=1}^{N} g_i(w) &= \frac{d}{dw} \left(g_1(w) + g_2(w) + \dots g_N(w)\right) \\ &= \frac{d}{dw} g_1(w) + \frac{d}{dw} g_2(w) + \dots \frac{d}{dw} g_N(w) \\ &= \sum_{i=1}^{N} \frac{d}{dw} g_i(w) \end{aligned}$$

RSS가 최소가 되게 하는 파라미터를 찾기 위해서는 편미분을 해야 한다. 편미분의 결과로 벡터가 나오게 된다.)

Taking the derivative w.r.t. w_0

$$\sum_{i=1}^{N} 2\left(y_i - [w_0 + w_1 x_i]\right)^1 \cdot (-1)$$

$$= -2 \sum_{i=1}^{N} \left(y_i - [w_0 + w_1 x_i]\right)$$

Taking the derivative w.r.t. w_1

$$\sum_{i=1}^{N} 2\left(y_i - [w_0 + w_1 x_i]\right)^1 \cdot (-x_i)$$

$$= -2 \sum_{i=1}^{N} \left(y_i - [w_0 + w_1 x_i]\right) x_i$$

$$\nabla \text{RSS}(w_0, w_1) = \begin{bmatrix} -2 \sum_{i=1}^{N} [y_i - (w_0 + w_1 x_i)] \\ -2 \sum_{i=1}^{N} [y_i - (w_0 + w_1 x_i)] x_i \end{bmatrix} = \begin{bmatrix} -2 \sum_{i=1}^{N} [y_i - \hat{y}_i(w_0, w_1)] \\ -2 \sum_{i=1}^{N} [y_i - \hat{y}_i(w_0, w_1)] x_i \end{bmatrix}$$

$$\hat{w}_0 = \frac{\sum_{i=1}^{N} y_i}{N} - \hat{w}_1 \frac{\sum_{i=1}^{N} x_i}{N} = 0$$

권도형

$$w\hat{}_1 = \frac{\sum_{i=1}^{N} y_i x_i - \frac{\sum y_i x_i}{N}}{\sum x^2{}_i - \frac{\sum x_i \sum x_i}{N}} = 0$$

즉 , $\sum_{i=1}^{N} y_i$, $\sum_{i=1}^{N} x_i$, $\sum_{i=1}^{N} y_i x_i$, $\sum_{i=1}^{N} x^2{}_i$를 알아야 알아야 w_0와 w_1를 구할 수 있다.

w_1 유도과정

$$\sum y_i x_i - \hat{w_0} \sum x_i - \hat{w_1} \sum x_i^2 = 0$$

$$\sum y_i x_i - \left(\frac{\sum y_i}{N} - \hat{w_1}\frac{\sum x_i}{N}\right)\sum x_i - \hat{w_1} \sum x_i^2 = 0$$

$$\sum y_i x_i - \left(\frac{\sum y_i \sum x_i}{N} - \hat{w_1}\frac{\sum x_i \sum x_i}{N}\right) - \hat{w_1} \sum x_i^2 = 0$$

$$\sum y_i x_i - \frac{\sum y_i \sum x_i}{N} + \hat{w_1}\frac{\sum x_i \sum x_i}{N} - \hat{w_1} \sum x_i^2 = 0$$

$$\sum y_i x_i - \frac{\sum y_i \sum x_i}{N} + \hat{w_1}\left(\frac{\sum x_i \sum x_i}{N} - \sum x_i^2\right) = 0$$

$$\hat{w_1}\left(\frac{\sum x_i \sum x_i}{N} - \sum x_i^2\right) = \frac{\sum y_i \sum x_i}{N} - \sum y_i x_i$$

$$\hat{w_1} = \frac{\frac{\sum y_i \sum x_i}{N} - \sum y_i x_i}{\frac{\sum x_i \sum x_i}{N} - \sum x_i^2}$$

권도형

Example- Closed Form of Gradient Descent

X	Y
0	1
1	3
2	7
3	13
4	21

numerator = (sum of X*Y) - (1/N)*((sum of X) * (sum of Y))

denominator = (sum of X^2) - (1/N)*((sum of X) * (sum of X))

or

numerator = (mean of X * Y) - (mean of X)*(mean of Y)

denominator = (mean of X^2) - (mean of X)*(mean of X)

- N = 5
- The sum of the Ys = 45
- The sum of the Xs = 10
- The sum of the product of the Xs and the Ys = 140
- The sum of the Xs squared = 30
    - So that:

- numerator = [(140) - (1/5) * (45*10)] = 50
- denominator = [(30) - (1/5) * (10*10)] = 10

    - hence:

- slope = 50/10 = 5

slope = w1 = 5

w1을 구해야 w0를 구할 수 있으므로

- The mean of the Ys = 9
- The mean of the Xs = 2
- The mean of the product of the Xs and the Ys = 28
- The mean of the Xs squared = 6

So that

numerator = 28 - 9*2 = 10

권도형

denominator = 6 - 2*2 = 2

hence:

slope = 10 / 2 = 5

slope = 5, intercept = -1

Vector formulation

$$\nabla RSS(w_0,w_1) = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - (w_0+w_1x_i)] \\ -2\sum_{i=1}^{N}[y_i - (w_0+w_1x_i)]x_i \end{bmatrix} = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0,w_1)] \\ -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0,w_1)]x_i \end{bmatrix}$$

$$\begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \end{bmatrix} - \eta \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0^{(t)},w_1^{(t)})] \\ -2\sum_{i=1}^{N}[y_i - \hat{y}_i(w_0^{(t)},w_1^{(t)})]x_i \end{bmatrix}$$

while not converged $(-2)\cdot(-\eta)$

$$\begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \end{bmatrix} + 2\eta \begin{bmatrix} \sum_{i=1}^{N}[y_i - \hat{y}_i(w_0^{(t)},w_1^{(t)})] \\ \sum_{i=1}^{N}[y_i - \hat{y}_i(w_0^{(t)},w_1^{(t)})]x_i \end{bmatrix}$$

권도형

Example- Vector form of Gradient Descent

We will need a starting value for the slope and intercept, a step_size and a tolerance; initial_intercept = 0, initial_slope = 0, step_size = 0.05, tolerance = 0.01

The algorithm

In each step of the gradient descent we will do the following:
1. Compute the predicted values given the current slope and intercept
2. Compute the prediction errors (prediction - Y)
3. Update the intercept:

- compute the derivative: sum(errors)
- compute the adjustment as step_size times the derivative
- decrease the intercept by the adjustment
-

4. Update the slope:

- compute the derivative: sum(errors*input)
- compute the adjustment as step_size times the derivative
- decrease the slope by the adjustment

5. Compute the magnitude of the gradient
6. Check for convergence

First step: Intercept = 0, Slope = 0
1. predictions = [0, 0, 0, 0, 0]
2. errors = [-1, -3, -7, -13, -21]
3. update Intercept :: sum([-1, -3, -7, -13, -21]) = -45, adjustment = 0.05 * 45 = -2.25 , new_intercept = 0 - -2.25 = 2.25
4. update Slope :: sum([0, 1, 2, 3, 4] * [-1, -3, -7, -13, -21]) = -140, adjustment = 0.05 * 45 = -7, new_slope = 0 - -7 = 7
5. magnitude = sqrt((-45)^2 + (-140)^2) = 147.05
6. magnitude > tolerance: not converged

Second step: Intercept = 2.25 , Slope = 7
1. predictions = [2.25, 9.25, 16.25, 23.25, 30.25]
2. errors = [1.25, 6.35, 9.25, 10.25, 9.25]
3. update Intercept :: sum([1.25, 6.35, 9.25, 10.25, 9.25]) = 36.25, adjustment = 0.05 * 36.25 = 1.8125, new_intercept = 2.25-1.8125 = 0.4375
4. update Slope :: sum([0, 1, 2, 3, 4] * [1.25, 6.35, 9.25, 10.25, 9.25]) = 92.5, adjustment = 0.05 * 92.5 = 4.625, new_slope = 7 - 4.625 = 2.375
5. magnitude = sqrt((36.25)^2 + (92.5)^2) = 99.35
6. magnitude > tolerance: not converged

Third step: Intercept = 0.4375, Slope = 2.375
1. predictions = [0.4375, 2.8125, 5.1875, 7.5625, 9.9375]
2. errors = [-0.5625, -0.1875, -1.8125, -5.4375, -11.0625]

권도형

3. update Intercept :: sum([-0.5625, -0.1875, -1.8125, -5.4375, -11.0625]) = -19.0625, adjustment = 0.05 * = -0.953125, new_intercept = 0.4375 - -0.953125 = 1.390625
4. update Slope :: sum([0, 1, 2, 3, 4] * [-0.5625, -0.1875, -1.8125, -5.4375, -11.0625]) = -64.375, adjustment = 0.05 * -64.375= -3.21875, new_slope = 2.375 --3.21875 = 5.59375
5. magnitude = sqrt((-19.0625)^2 + (-64.375)^2) = 67.13806
6. magnitude > tolerance: not converged

Let's skip forward a few steps... after the 77th step we have gradient magnitude 0.0107.

78th Step:Intercept = -0.9937, Slope = 4.9978
1. predictions = [-0.99374, 4.00406, 9.00187, 13.99967, 18.99748]
2. errors = [-1.99374, 1.00406, 2.00187, 0.99967, -2.00252]
3. update Intercept :: sum([-1.99374, 1.00406, 2.00187, 0.99967, -2.00252]) = 0.009341224, adjustment = 0.05 * 0.009341224 = 0.0004670612, new_intercept = -0.9937 - 0.0004670612 = -0.994207
4. update Slope :: sum([0, 1, 2, 3, 4] * [-1.99374, 1.00406, 2.00187, 0.99967, -2.00252]) = -0.0032767, adjustment = 0.05 *-0.0032767 = -0.00016383, new_slope = 4.9978 --0.00016383 = 4.9979
5. magnitude = sqrt[()^2 + ()^2] = 0.0098992
6. magnitude < tolerance: converged!

Final slope: -0.994

Final Intercept: 4.998

If you continue you will get to (-1, 5) but at this point the change in RSS (our cost) is negligible.(무시해도 될 정도)

권도형

Concept Exercise

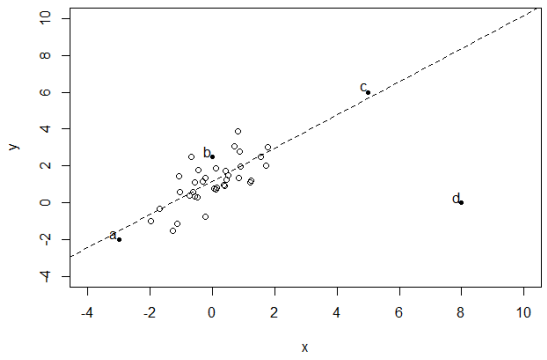
1. In a simple regression model, if you increase the input value by 1 then you expect the output to change by:

Also 1 The value of the *slope* parameter The value of the *intercept* parameter Impossible to tell

2. Two people present you with fits of their simple regression model for predicting house prices from square feet. You discover that the estimated intercept and slopes are exactly the same. This necessarily implies that these two people fit their models on *exactly* the same data set. True False

3. Your friend in the U.S. gives you a simple regression fit for predicting house prices from square feet. The estimated intercept is -44850 and the estimated slope is 280.76. You believe that your housing market behaves very similarly, but houses are measured in square meters. To make predictions for inputs in square meters, what **slope** must you use? Hint: there are 0.092903 square meters in 1 square foot.

4. Consider the following data set:



Which bold/labeled point, if removed, will have the largest effect on the fitted regression line (dashed)? a b c d

5. **Using the learned slope and intercept from the squarefeet model, what is the RSS for the simple linear regression using squarefeet to predict prices on TRAINING data?**

- Between 5e+12 and 5.2e+12
- Between 1.1e+14 and 1.3e+14
- Between 1.1e+15 and 1.3e+15
- Between 3.3e+15 and 3.5e+15

6. **Which of the two models (square feet or bedrooms) has lower RSS on TEST data?**

Model 1 (Square feet) Model 2 (Bedrooms)

Answer

1. ② | 2. False | 3. 3,022.0767898 | 4. d | 5. ③ | 6. ① | 7.

Multiple Regression Model

권도형

$$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \dots + w_D h_D(x_i) + \varepsilon_i = \sum_{j=0}^{D} w_j h_j(x_i) + \varepsilon_i$$

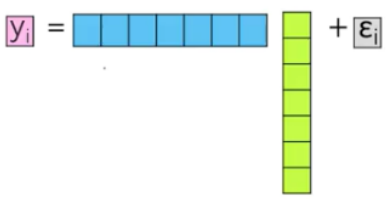
D-dimensional curve

Model:

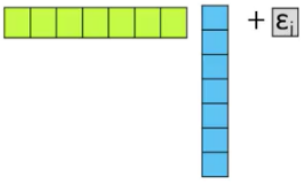
$$y_i = w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \dots + w_D h_D(\mathbf{x}_i) + \varepsilon_i = \sum_{j=0}^{D} w_j h_j(\mathbf{x}_i) + \varepsilon_i$$

feature 1 = $h_0(x)$...often 1 (constant)
feature 2 = $h_1(x)$... e.g., x
feature 3 = $h_2(x)$... e.g., x^2 or $\sin(2\pi x/12)$
...
feature D+1 = $h_D(x)$... e.g., x^p

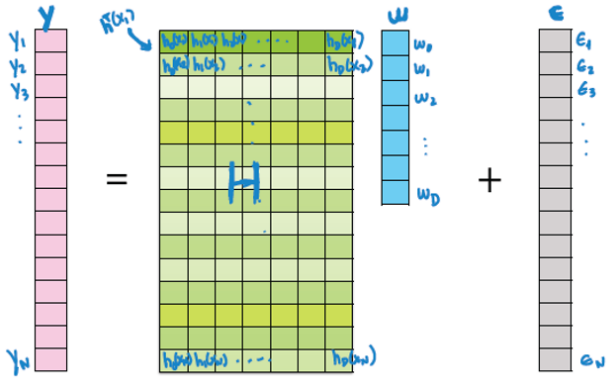
행렬을 이용하여 다시 표현하면



Or



Blue box represents parameters, and Green box represents features of observation i. When multiplying these two vectors, then it is transformed into the linear combination formation. That is, Scalar



Blue box represents parameters still, but Green box represents features of all the observations. It's a Matrix.

These are represented like this: $y = Hw + \epsilon$

Now we have to Compute the RSS.

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y_i - \quad)^2$$

In the Blank, what can be inserted? Predicted value of observation i.

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2 = (y - Hw)^T (y - Hw)$$

That is, $RSS(\mathbf{w}) = \sum_{i=1}^{N} (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2$. Its equivalent this equation.

The cause is following:

When the I th predicted value is called y hat, then the RSS formation is converted into

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2 = (y - \hat{y})^T (y - \hat{y})$$

$(y - \hat{y})$ = $\begin{bmatrix} \text{residual}_1 \\ \text{residual}_2 \\ \vdots \\ \text{residual}_N \end{bmatrix}$ ← residual$_i$ = $y_i - \hat{y}_i$

$(y - \hat{y})$, this means residual. So we can represent like this:

권도형

So $(y-Hw)$ is equivalent with

$residual_1$
$residual_2$
$residual_3$
...
$residual_N$

$(y-Hw)^T$ is equivalent with

$residual_1$	$residual_2$	$residual_3$	...	$residual_N$

$(\mathbf{y}-\mathbf{Hw})^T(\mathbf{y}-\mathbf{Hw})$ is equivalent with

$residual_1$	$residual_2$	$residual_3$	...	$residual_N$

$residual_1$
$residual_2$
$residual_3$
...
$residual_N$

Note that this representation is equivalent with the Linear combination of all the Residuals. 잔차제곱합(RSS)이다.

To minimize RSS, consider the Gradient of RSS.

$$\nabla RSS(\mathbf{w}) = \nabla[(\mathbf{y}-\mathbf{Hw})^T(\mathbf{y}-\mathbf{Hw})]$$
$$= -2\mathbf{H}^T(\mathbf{y}-\mathbf{Hw})$$

$\frac{d}{dw}(y-hw)(y-hw)$ (scalars)

$\frac{d}{dw}(y-hw)^2 = 2\cdot(y-hw)^1(-1)$
$= -2h(y-hw)$

Gradient를 구하는 방법은 Close form solution과 Vector form solution이 있다. closed form solution을 이용하면, feature의 개수가 많아질수록 computational complexity가 $O(D^3)$ (where n is the number of features.)로 증가하게 된다. As the number of features used in regression increases, the matrix operations required by the closed-form solution become computationaly expensive. 따라서 feature가 많은 경우엔 closed form solution이 아닌 vector form solution을 사용해야 한다. In cases with many features, an optimization algorithm is needed instead, and gradient descent is one of the most commonly used.

In Vector form solution,

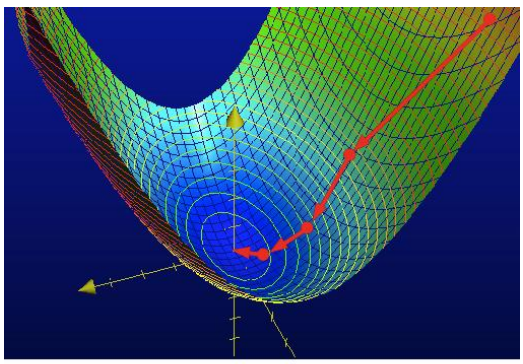
$$\nabla RSS(\mathbf{w}) = -2\mathbf{H}^T(\mathbf{y}-\mathbf{Hw}) = 0$$ or

$$\nabla RSS\left(\mathbf{w}^{(t)}\right) = \nabla\left[\left(\mathbf{y}-\mathbf{Hw}^{(t)}\right)^T\left(\mathbf{y}-\mathbf{Hw}^{(t)}\right)\right]$$
$$= -2\mathbf{H}^T\left(\mathbf{y}-\mathbf{Hw}^{(t)}\right)$$

$-2H^T(y-Hw) = -2H^Ty + 2H^THw = 0$

$-2H^Ty + 2H^THw = 0$
$H^TH\hat{w} = H^Ty$
$(H^TH)^{-1}H^THw = (H^TH)^{-1}H^Ty$
$\hat{w} = (H^TH)^{-1}H^Ty$

To get the value of parameters, then compute $\hat{\mathbf{w}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$



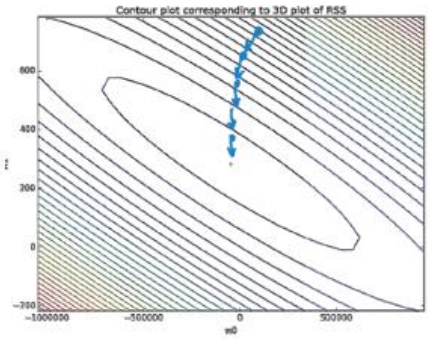
In gradient descent, estimations of coeffients of a model equation are iteratively updated based upon the current gradient of the function, descending a cost function until the gradient is near zero. For multiple linear regression the cost function is the residual sum of squares (RSS) of the model when applied to the test set.

With true outputs **y**, feature coefficient vector **w**(*t*) at iteration *t*, and learning rate η, the updated coeffiencents are given by:

권도형

w(t+1)=w(t)−η∇RSS(w(t))

Gradient descent



while not converged

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla RSS(w^{(t)})$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + 2\eta \left[\mathbf{H}^T \left(\mathbf{y} - \mathbf{H}\mathbf{w}^{(t)} \right) \right]$$

With data by feature matrix **H**, **∇RSS(w(t))=∇[(y-Hw(t))T(y-Hw(t))] =-2HT(y-Hw(t))**

This yields the final derivation: **w(t+1)=w(t)+2η[HT(y-Hw(t))]**

잔차와 feature의 선형결합(내적)

To compute w hat, 편미분을 해야 한다.

$$RSS(\mathbf{w}) = \sum_{i=1}^{N} (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2$$

$$= \sum_{i=1}^{N} (y_i - w_0 h_0(x_i) - w_1 h_1(x_i) - \dots - w_D h_D(x_i))^2$$

Partial with respect to w_j

$$\sum_{i=1}^{N} 2(y_i - w_0 h_0(x_i) - w_1 h_1(x_i) \dots - w_D h_D(x_i))^1 \cdot (-h_j(x_i))$$

$$= -2 \sum_{i=1}^{N} h_j(x_i)(y_i - h(x_i)^T w)$$

Observation i

Update to j^{th} feature weight:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta(-2 \sum_{i=1}^{N} h_j(x_i)(y_i - \underbrace{h^T(x_i) w^{(t)}}_{\hat{y}_i(w^{(t)})}))$$

All of the observation

Here is some pseudocode for the algorithm to translate from mathese into codespeak:

```
initialize coefficients

while gradient_magnitude >= tolerance:
    for each feature:
        updated_feature_coefficient = feature_coefficient - eta*feature_derivative
return coefficients
```

Now for the Python implementation:

First import libraries and create functions to make predictions based on the model coefficients and yeild the errors based upon those predictions.

In [1]:

```
import math
import random
import numpy as np
```

권도형

```
def predict_output(feature_matrix, coefficients):
    ''' Returns an array of predictions

    inputs -
        feature_matrix - 2-D array of dimensions data points by features
        coefficients - 1-D array of estimated feature coefficients

    output - 1-D array of predictions
    '''
    predictions = np.dot(feature_matrix, coefficients)
    return predictions
```

Next, a function to compute the partial derivative for each feature:

In [2]:

```
def feature_derivative(errors, feature):
    derivative = 2*np.dot(errors, feature)
    return(derivative)
```

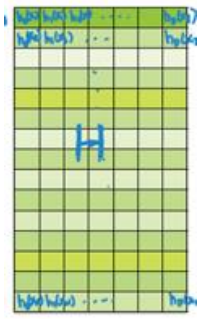
$w_0 h_0(x_i) - w_1 h_1(x_i) - \cdots - w_D h_D(x_i)$

잔차와 feature의 선형결합

Now we are ready to write the main function:

In [3]:

```
def gradient_descent_regression(H, y, initial_coefficients, eta, epsilon,
max_iterations=10000):
    ''' Returns coefficients for multiple linear regression.
    inputs -
        H - 2-D array of dimensions data points by features
        y - 1-D array of true output
        initial_coefficients - 1-D array of initial coefficients
        eta - float, the step size eta
        epsilon - float, the tolerance at which the algorithm will terminate
        max_iterations - int, tells the program when to terminate
    output - 1-D array of estimated coefficients
    '''
    converged = False
    w = initial_coefficients
    iteration = 0
    while not converged:
        if iteration > max_iterations:
            print 'Exceeded max iterations\nCoefficients: ', w
            return w
        pred = predict_output(H, w)
        residuals = pred-y
        gradient_sum_squares = 0
        for i in range(len(w)):
            partial = feature_derivative(residuals, H[:, i])
            gradient_sum_squares += partial**2
            w[i] = w[i] - eta*partial
        gradient_magnitude = math.sqrt(gradient_sum_squares)
        if gradient_magnitude < epsilon:
            converged = True
        iteration += 1
    print w
    return w
```



이 matrix가 H다.

H의 1열의 값은 전부 1이다.

$w(t+1)=w(t)+2\eta[H^T(y-Hw(t))]$

Let's test it out! I'll use a noisy sine function and try to fit a third degree polynomial.

In [4]:

```
%matplotlib inline
import matplotlib.pyplot as plt
random.seed(1)
n = 20
pts = [x/5. for x in xrange(n)]
X = np.array(pts)
```

권도형

```
y = np.array([math.sin(x) for x in pts]) + [random.gauss(0,1.0/3.0) for i in xrange(n)]

plt.plot(X, y, '*')
```

Out[4]:

```
[<matplotlib.lines.Line2D at 0x105b53bd0>]
```

Now to build the feature matrix with one column for each feature. The regression model will return coefficients **w** that minimize the RSS of the model:

y^i=w0+w1xi+w2x2i+w3x3i

In [5]:

```
feature_matrix = np.zeros(n*4)
feature_matrix.shape = (n, 4)
feature_matrix[:,0] = 1
feature_matrix[:,1] = X
feature_matrix[:,2] = X**2
feature_matrix[:,3] = X**3
```

Now to run the regression. Choosing η and ε is a bit of an art. Choosing a small learning rate will lead to a more precise answer, but if the learning rate is too small, the optimization may take a very long time. Conversely, a larger learning rate will converge faster, but if it's too large, the algorithm may overshoot the minimum and never converge, bouncing around the sides of the bowl into eternity. Setting a max number of iterations will avoid potential infinite loops.

A reasonable epsilon depends on how noisy the data is and how flexible the model is. Starting relatively high and stepping down will increase the precision of the model without running into too many iteration overflows.

For η, there are some interesting techniques for adjusting the parameter as the algorithm approaches the optimal solution. Check out Adagrad for an example.

Initializing the coefficients can also be done intelligently, but I'll just set them all to 0 here.

In [6]:

```
initial_coefficients = np.array([0., 0., 0., 0.])
coef = gradient_descent_regression(feature_matrix, y, initial_coefficients, 6e-5, 1)
```

```
[ 0.30764503  0.20667802  0.14696395 -0.07014237]
```

Let's see how we did.

In [7]:

```
model_prediction_X = np.array([x/10. for x in range(40)])
model_prediction_matrix = np.zeros(40*4)
model_prediction_matrix.shape = (40, 4)
model_prediction_matrix[:,0] = 1
model_prediction_matrix[:,1] = model_prediction_X
model_prediction_matrix[:,2] = model_prediction_X**2
model_prediction_matrix[:,3] = model_prediction_X**3
predictions = predict_output(model_prediction_matrix, coef)
plt.plot(X, y, '*', model_prediction_X, predictions, '-')
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x104013fd0>,
 <matplotlib.lines.Line2D at 0x105dc8950>]
```

Concept Exercise

1. Which of the following is NOT a linear regression model. *Hint: remember that a linear regression model is always linear in the parameters, but may use non-linear features.*

권도형

○ y = w_0 + w_1 * x ○ y = w_0 + w_1 * (x^2) ○ y = w_0 + w_1 * log(x) ○ y = w_0 * w_1 + log(w_1) * x

2. Your estimated model for predicting house prices has a large positive weight on 'square feet living'. This implies that if we remove the feature 'square feet living' and refit the model, the new predictive performance will be **worse** than before. ○ True ○ False

3. *Complete the following:* Your estimated model for predicting house prices has a positive weight on 'square feet living'. You then add 'lot size' to the model and re-estimate the feature weights. The new weight on 'square feet living' [_____] be positive. ○ will not ○ will definitely ○ might

4. If you double the value of a given feature (i.e. a specific column of the feature matrix), what happens to the least-squares estimated coefficients for every **other** feature? (assume you have no other feature that depends on the doubled feature i.e. no interaction terms).

○ They double ○ They halve ○ They stay the same ○ It is impossible to tell from the information provided

5. Gradient descent/ascent is...

○ A model for predicting a continuous variable ○ An algorithm for minimizing/maximizing a function
○ A theoretical statistical result ○ An approximation to simple linear regression ○ A modeling technique in machine learning

6. Gradient descent/ascent allows us to...

○ Predict a value based on a fitted function ○ Estimate model parameters from data
○ Assess performance of a model on test data

7. Which of the following statements about step-size in gradient descent is/are **TRUE** (select all that apply)

☐ It's important to choose a very small step-size
☐ The step-size doesn't matter
☐ If the step-size is too large gradient descent may not converge
☐ If the step size is too small (but not zero) gradient descent may take a very long time to converge

8. Let's analyze how many computations are required to fit a multiple linear regression model *using the closed-form solution* based on a data set with 50 observations and 10 features. In the videos, we said that computing the inverse of the 10x10 matrix (H^T)H was on the order of D^3 operations. Let's focus on forming this matrix **prior** to inversion. How many multiplications are required to form the matrix (H^T)H?

Please enter a number below.

9. More generally, if you have D features and N observations what is the total complexity of computing ((H^T)H)^(-1)?

○ O(D^3) ○ O(ND^3) ○ O(ND^2 + D^3) ○ O(ND^2) ○ O(N^2D + D^3) ○ O(N^2D)

Answer

1. ④ | 2. False | 3. ③ | 4. ③ | 5. ② | 6. ② | 7. 3,4 | 8. 2500 | 9. ③

권도형