Contents

1 General Linear Regression in Statistics

In general, the variables $X_1, \ldots, X_{p-1}$ in a regression model do not need to represent different predictor variables. We therefore define the general linear regression model, with normal error terms, simply in terms of X variables:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_{p-1} X_{i,p-1} + \varepsilon_i$$

where:

- $\beta_0, \beta_1, \ldots, \beta_{p-1}$ are parameters
- $X_{i1}, \ldots, X_{i,p-1}$ are known constants
- ε_i are independent $N\left(0, \sigma^2\right)$
- $i = 1, \ldots, n$

If we let $X_{i0} \equiv 1$, the regression model can be written as follows:

$$Y_i = \beta_0 X_{i0} + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_{p-1} X_{i,p-1} + \varepsilon_i$$

where $X_{i0} \equiv 1$, or:

$$Y_i = \sum_{k=0}^{p-1} \beta_k X_{ik} + \varepsilon_i \qquad \text{where } X_{i0} \equiv 1$$

The response function for the regression model, since $E\left\{\varepsilon_i\right\} = 0$, is as follows:

$$E\{Y\} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_{p-1} X_{p-1}$$

Thus, the general linear regression model with normal error terms implies that the observations Y_i are independent normal variables, with mean $E\left\{Y_i\right\}$ and with constant variance σ^2. In matrix notation, we have the following matrices:

$$\underset{n\times 1}{\mathbf{Y}} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \quad \underset{n\times p}{\mathbf{X}} = \begin{bmatrix} 1 & X_{11} & X_{12} & \cdots & X_{1,p-1} \\ 1 & X_{21} & X_{22} & \cdots & X_{2,p-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & X_{n1} & X_{n2} & \cdots & X_{n,p-1} \end{bmatrix} = \begin{bmatrix} X_{1,0} & X_{11} & X_{12} & \cdots & X_{1,p-1} \\ X_{2,0} & X_{21} & X_{22} & \cdots & X_{2,p-1} \\ \vdots & \vdots & \vdots & & \vdots \\ X_{n,0} & X_{n1} & X_{n2} & \cdots & X_{n,p-1} \end{bmatrix}$$

$$\underset{p\times 1}{\boldsymbol{\beta}} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{bmatrix} \quad \underset{n\times 1}{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

The $\mathbf{Y}$ and the $\boldsymbol{\varepsilon}$ are the response and error term vectors. The $\boldsymbol{\beta}$ vector contains additional regression parameters, and the X matrix contains a column of 1's as well as a column of the n observations for each of the $p-1$ X variables in the regression model. The row subscript for each element X_{ik} in the X matrix identifies the i^{th} trial or case, and the column subscript identifies the k^{th} X variable. In matrix terms, the general linear regression model is:

$$\underset{n\times 1}{\mathbf{Y}} = \underset{n\times p}{\mathbf{X}}\underset{p\times 1}{\boldsymbol{\beta}} + \underset{n\times 1}{\boldsymbol{\varepsilon}}$$

$$\underset{n\times 1}{\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}} = \underset{n\times p}{\begin{bmatrix} X_{1,0} & X_{11} & X_{12} & \cdots & X_{1,p-1} \\ X_{2,0} & X_{21} & X_{22} & \cdots & X_{2,p-1} \\ \vdots & \vdots & \vdots & & \vdots \\ X_{n,0} & X_{n1} & X_{n2} & \cdots & X_{n,p-1} \end{bmatrix}} \underset{p\times 1}{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{bmatrix}} + \underset{n\times 1}{\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}}$$

where:

- $\mathbf{Y}$ is a vector of responses
- $\boldsymbol{\beta}$ is a vector of parameters
- $\mathbf{X}$ is a matrix of constants
- $\boldsymbol{\varepsilon}$ is a vector of independent normal random variables with expectation $\mathbf{E}\{\boldsymbol{\varepsilon}\} = \mathbf{0}$ and variance-covariance matrix:

$$\underset{n\times n}{\sigma^2\{\varepsilon\}} = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma^2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma^2 \end{bmatrix} = \sigma^2\mathbf{I}$$

Consequently, the random vector $\mathbf{Y}$ has expectation:

$$\mathbf{E}\{\mathbf{Y}\} = \mathbf{E}\{\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}\}$$
$$\mathbf{E}\{\mathbf{Y}\} = \mathbf{E}\{\mathbf{X}\boldsymbol{\beta}\} + \mathbf{E}\{\boldsymbol{\varepsilon}\}$$
$$\mathbf{E}\{\mathbf{Y}\} = \mathbf{E}\{\mathbf{X}\boldsymbol{\beta}\} + \mathbf{0}$$

Since $\mathbf{X}\boldsymbol{\beta}$ is not random and $\mathbf{E}\{a\} = a$:

$$\underset{n\times 1}{\mathbf{E}\{\mathbf{Y}\}} = \mathbf{X}\boldsymbol{\beta}$$

and the variance-covariance matrix of $\mathbf{Y}$ is the same as that of $\boldsymbol{\varepsilon}$:

$$\sigma^2\{\mathbf{Y}\} = \sigma^2\{\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}\}$$
$$\sigma^2\{\mathbf{Y}\} = \sigma^2\{\boldsymbol{\varepsilon}\}$$
$$\underset{n\times n}{\sigma^2\{\mathbf{Y}\}} = \sigma^2\mathbf{I}$$

This follows from the special case of the variance operator $\sigma^2\{a + \mathbf{Y}\} = \sigma^2\{\mathbf{Y}\}$ where a is a constant.

1.1 Estimation of Regression Coefficients

The least squares criterion is generalized as follows for general linear regression model:

$$Q = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_{i1} - \cdots - \beta_{p-1} X_{i,p-1})^2$$

The least squares estimators are those values of $\beta_0, \beta_1, \ldots, \beta_{p-1}$ that minimize Q. Let us denote the vector of the least squares estimated regression coefficients $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_{p-1}$ as $\hat{\boldsymbol{\beta}}$:

$$\underset{p\times 1}{\hat{\boldsymbol{\beta}}} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_{p-1} \end{bmatrix}$$

The least squares normal equations for the general linear regression model are:

$$\underset{p\times n}{\mathbf{X}'} \underset{n\times p}{\mathbf{X}} \underset{p\times 1}{\hat{\boldsymbol{\beta}}} = \underset{p\times n}{\mathbf{X}'} \underset{n\times 1}{\mathbf{Y}}$$

and the least squares estimators are:

$$\underset{p\times 1}{\hat{\boldsymbol{\beta}}} = \left(\underset{p\times p}{\mathbf{X}'\mathbf{X}}\right)^{-1} \underset{p\times 1}{\mathbf{X}'\mathbf{Y}}$$

1.2 Fitted Values and Residuals

Let the vector of the fitted values $\hat{Y}_i$ be denoted by $\hat{\mathbf{Y}}$ and the vector of the residual terms $e_i = Y_i - \hat{Y}_i$ be denoted by $\mathbf{e}$:

$$\underset{n\times 1}{\hat{\mathbf{Y}}} = \begin{bmatrix} \hat{Y}_1 \\ \hat{Y}_2 \\ \vdots \\ \hat{Y}_n \end{bmatrix} \qquad \underset{n\times 1}{\mathbf{e}} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

The fitted values are represented by:

$$\underset{n\times 1}{\hat{\mathbf{Y}}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

$$\underset{n\times 1}{\begin{bmatrix} \hat{Y}_1 \\ \hat{Y}_2 \\ \vdots \\ \hat{Y}_n \end{bmatrix}} = \underset{n\times p}{\begin{bmatrix} X_{1,0} & X_{11} & X_{12} & \cdots & X_{1,p-1} \\ X_{2,0} & X_{21} & X_{22} & \cdots & X_{2,p-1} \\ \vdots & \vdots & \vdots & & \vdots \\ X_{n,0} & X_{n1} & X_{n2} & \cdots & X_{n,p-1} \end{bmatrix}} \underset{p\times 1}{\begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_{p-1} \end{bmatrix}}$$

where $X_{i0} \equiv 1$. The residual terms can be represented by:

$$\underset{n\times 1}{\mathbf{e}} = \mathbf{Y} - \hat{\mathbf{Y}} = \mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}$$

The vector of fitted values $\hat{\mathbf{Y}}$ can be expressed in terms of the hat matrix $\mathbf{H}$ as follows:

$$\begin{aligned} \underset{n\times 1}{\hat{\mathbf{Y}}} &= \mathbf{X}\hat{\boldsymbol{\beta}} \\ \underset{n\times 1}{\hat{\mathbf{Y}}} &= \mathbf{X}\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{Y} \\ \underset{n\times 1}{\hat{\mathbf{Y}}} &= \left(\underset{n\times p}{\mathbf{X}}\underset{p\times p}{\left(\mathbf{X}'\mathbf{X}\right)^{-1}}\underset{p\times n}{\mathbf{X}'}\right)\mathbf{Y} \\ \underset{n\times 1}{\hat{\mathbf{Y}}} &= \underset{n\times n}{\mathbf{H}}\,\mathbf{Y} \end{aligned}$$

where:

$$\underset{n\times n}{\mathbf{H}} = \mathbf{X}\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'$$

The rank of the hat matrix is p. Similarly, the vector of residuals can be expressed as follows:

$$\underset{n\times 1}{\mathbf{e}} = (\mathbf{I} - \mathbf{H})\mathbf{Y}$$

The variance-covariance matrix of the residuals is:

$$\underset{n\times n}{\boldsymbol{\sigma}}^2\{\mathbf{e}\} = \sigma^2(\mathbf{I} - \mathbf{H})$$

which is estimated by:

$$\underset{n\times n}{\mathbf{s}^2}\{\mathbf{e}\} = MSE(\mathbf{I} - \mathbf{H})$$

1.3 Sum of Squares

Finally, the sum of squares are given by the following matrix equations:

$$\text{SSTO} = \mathbf{Y}'\mathbf{Y} - \left(\frac{1}{n}\right)\mathbf{Y}'\mathbf{J}\mathbf{Y} = \mathbf{Y}'\left[\mathbf{I} - \left(\frac{1}{n}\right)\mathbf{J}\right]\mathbf{Y}$$
$$\text{SSE} = \mathbf{e}'\mathbf{e} = (\mathbf{Y} - \mathbf{X}\mathbf{b})'(\mathbf{Y} - \mathbf{X}\mathbf{b}) = \mathbf{Y}'\mathbf{Y} - \mathbf{b}'\mathbf{X}'\mathbf{Y} = \mathbf{Y}'(\mathbf{I} - \mathbf{H})\mathbf{Y}$$
$$\text{SSR} = \mathbf{b}'\mathbf{X}'\mathbf{Y} - \left(\frac{1}{n}\right)\mathbf{Y}'\mathbf{J}\mathbf{Y} = \mathbf{Y}'\left[\mathbf{H} - \left(\frac{1}{n}\right)\mathbf{J}\right]\mathbf{Y}$$

And the means squares are given as follows:

$$\text{MSR} = \frac{\text{SSR}}{p-1}$$
$$\text{MSE} = \frac{\text{SSE}}{n-p}$$

2 General Linear Regression in Machine Learning

In machine learning applications, the linear regression model is considered an algorithm. More generally, a linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the **bias term** (also called the intercept term). The machine learning notations are rather different compared to those employed in statistics.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

In this equation:

- $\hat{y}$ is the predicted (or fitted) value
- n is the number of features (compared to (p-1) predictors in statistics notations)
- x_i is the i^{th} feature value (which is denoted as the X_{ik} predictor among $X_{i1}, \cdots, X_{i,p-1}$ in statistics notations)
- θ_j is the j^{th} model parameter, including the bias term θ_0 and the feature weights or the estimated coefficients $(\theta_1, \theta_2, \cdots, \theta_n)$ Note: This is denoted as $(\hat{\beta}_0, \hat{\beta}_1, \cdots, \hat{\beta}_{p-1})$ in statistics notations

The estimated regression function above can be expressed in vector equation form as follows:

$$\begin{aligned}
\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) &= \boldsymbol{\theta} \cdot \mathbf{x} \\
&= \underset{(n+1)\times 1}{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}} \cdot \underset{(n+1)\times 1}{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}} \\
&= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\
&= \theta_0(1) + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\
&= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n
\end{aligned}$$

In this equation:

- $\boldsymbol{\theta}$ is the model's parameter vector, containing the bias term θ_0 and the feature weights θ_1 to θ_n
- $\mathbf{x}$ is the instance's feature vector, containing x_0 to x_n, with x_0 always equal to 1 .
- $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is of course equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$.
- $h_{\boldsymbol{\theta}}$ is the hypothesis function, using the model parameters $\boldsymbol{\theta}$.

In Machine Learning, vectors are often represented as column vectors, which are 2D arrays with a single column. If $\boldsymbol{\theta}$ and $\mathbf{x}$ are column vectors, then the prediction is

$$\begin{aligned}\hat{y} &= \boldsymbol{\theta}^\top \mathbf{x} \\ &= \underset{1\times(n+1)}{[\theta_0 \;\; \theta_1 \;\; \cdots \;\; \theta_n]} \underset{(n+1)\times 1}{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}}\end{aligned}$$

where $\boldsymbol{\theta}^\top$ is the transpose of $\boldsymbol{\theta}$ (a row vector instead of a column vector) and $\boldsymbol{\theta}^\top \mathbf{x}$ is the matrix multiplication of $\boldsymbol{\theta}^\top$ and $\mathbf{x}$. It is of course the same prediction, $\underset{1\times 1}{\hat{y}}$ except that it is now represented as a single-cell matrix rather than a scalar value.

2.1 Sum of Squares

The mean square error (MSE) is also denoted differently in machine learning than it is in statistics. In statistics, the mean square error is defined as follows:

$$\begin{aligned}\text{MSE}_{\text{predictor}} &= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_{p-1} X_{i,p-1})\right)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} - \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)^2\end{aligned}$$

The term mean squared error is also sometimes used to refer to the unbiased estimate of error variance: the residual sum of squares divided by the number of degrees of freedom.

$$
\begin{aligned}
\text{MSE}_{\text{Error Variance}} &= \frac{1}{n-p}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \\
&= \frac{1}{n-p}\sum_{i=1}^{n}\left(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_{p-1}X_{i,p-1})\right)^2 \\
&= \frac{1}{n-p}\sum_{i=1}^{n}\left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} - \cdots - \hat{\beta}_{p-1}X_{i,p-1}\right)^2
\end{aligned}
$$

This definition for a known, computed quantity differs from the above definition for the computed MSE of a predictor, in that a different denominator is used. The denominator is the sample size reduced by the number of model parameters estimated from the same data, (n-p) for p regressors. Although the MSE (as defined for a predictor) is not an unbiased estimator of the error variance, it is consistent, given the consistency of the predictor. In machine learning, the notation is different. Most notably, the residual is computed in a different order $\hat{\mathbf{Y}} - \mathbf{Y}$ instead of $\mathbf{Y} - \hat{\mathbf{Y}}$. The MSE cost function of a linear regression hypothesis $h_{\boldsymbol{\theta}}$ on a training set $\mathbf{X}$ is calculated as follows:

$$
\text{MSE}\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2
$$

where

- m is the number of instances in the dataset we are measuring the MSE on, which is denoted as n in statistics

- $\mathbf{x}^{(i)}$ is a vector of all the feature values (excluding the label) of the i^{th} instance in the dataset, meaning that we have $\begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$ where $1, 2, \cdots, i, \cdots, m$ Note: the i^{th} trial is denoted as a row subscript in statistics X_{ik} where $1, 2, \cdots, i, \cdots, n$

- $y^{(i)}$ is the label (the desired output value for the i^{th} instance, which is known as the response value of the i^{th} trial in statistics, Y_i

- $\mathbf{X}$ is a matrix containing all the feature values (excluding labels) of all instances in the dataset. There is one row per instance, and the i^{th} row is equal to the transpose of $\mathbf{x}^{(i)}$, noted $\left(\mathbf{x}^{(i)}\right)^{\text{T}}$:

$$
\underset{m\times n}{\mathbf{X}} = \begin{pmatrix} \left(\mathbf{x}^{(1)}\right)^\top \\ \left(\mathbf{x}^{(2)}\right)^\top \\ \vdots \\ \left(\mathbf{x}^{(m-1)}\right)^\top \\ \left(\mathbf{x}^{(m)}\right)^\top \end{pmatrix} = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^m & x_2^m & \cdots & x_n^m \end{pmatrix}
$$

Note that the $m \times n$ matrix $\mathbf{X}$ is not quite the design $n \times p$ matrix used in statistics. Specifically, it does not contain the column of $X_{i,0} \equiv 1$'s, and so it has one fewer column. We can think of the mapping between the two notations as follows: m (machine learning) $\longrightarrow$ n (statistics) and n (machine learning) $\longrightarrow$ $(p-1)$ (statistics). In statistics, the matrix above would have been represented as follows:

$$\underset{n\times(p-1)}{\mathbf{X}} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1,p-1} \\ X_{21} & X_{22} & \cdots & X_{2,p-1} \\ \vdots & \vdots & & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{n,p-1} \end{bmatrix}$$

Note that h is the system's prediction function, also called a hypothesis. When the system is given an instance's feature vector $\mathbf{x}^{(i)}$, it outputs a predicted value $\hat{y}^{(i)} = h(x^{(i)})$ for that instance ($\hat{y}$ is pronounced "y-hat"). The notation h_θ makes it clear that the model is parametrized by the vector $\boldsymbol{\theta}$. To simplify notations, the MSE cost function is often represented as MSE($\boldsymbol{\theta}$) instead of MSE$\left(\mathbf{X}, h_{\boldsymbol{\theta}}\right)$.

2.2 Computational Complexity

The Normal Equation computes the inverse of $X'X$, which is an (n + 1) $\times$ (n + 1) matrix (where n is the number of features). In statistics, this is a $\Big(\underset{p\times p}{\mathbf{X}'\mathbf{X}}\Big)^{-1}$ matrix where p is the number of *regressors*. The computational complexity of inverting such a matrix is typically about O($n^{2.4}$) to O(n^3), depending on the implementation. In other words, if we double the number of features or predictor variables, we essentially multiply the computation time by roughly $2^{2.4} = 5.3$ to $2^3 = 8$.

The SVD approach used by Scikit-Learn's LinearRegression class is about O(n^2). If we double the number of features, we multiply the computation time by roughly 4. Both the Normal Equation and the SVD approach get very slow when the number of features grows large (e.g., 100,000). On the positive side, both are linear with regard to the number of instances in the training set (they are O(m)), so they handle large training sets efficiently, provided they can fit in memory. Also, once we have trained the Linear Regression model (using the Normal Equation or any other algorithm), predictions are very fast: the computational complexity is linear with regard to both the number of instances we wish to make predictions on and the number of features. In other words, making predictions on twice as many instances (or twice as many features) will take roughly twice as much time.

The gradient descent is another approach to train a Linear Regression model, which is better suited for cases where there are a large number of features or too many training instances to fit in memory.

3 Gradient Descent

3.1 Directional Derivative

Suppose we want to compute the rate of change of a function $f(x, y)$ at the point $P = (a, b)$ in the direction of the unit vector $\vec{u} = u_1\vec{i} + u_2\vec{j}$. For distance $h > 0$, consider the point $Q = (a + hu_1, b + hu_2)$ whose displacement from P is $h\vec{u}$. (See Figure 14.29.) Since $\|\vec{u}\| = 1$, the distance from P to Q is h. Thus,

$$\text{Average rate of change in } f \text{ from P to Q} = \frac{\text{Change in } f}{\text{Distance from P to Q}} = \frac{f(a + hu_1, b + hu_2) - f(a, b)}{h}$$

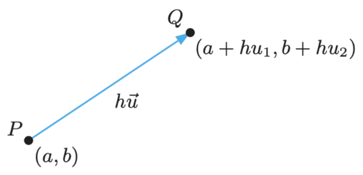

Figure 1: Displacement of $h\vec{u}$ from the point (a, b)

Taking the limit as $h \longrightarrow 0$ gives the instantaneous rate of change and the following definition:

Directional Derivative of f at (a, b) in the Direction of a Unit Vector $\vec{u}$

Definition 3.1. If $\vec{u} = u_1\vec{i} + u_2\vec{j}$ is a unit vector, we define the directional derivative, $f_{\vec{u}}$, by

$$f_{\vec{u}}(a, b) = \begin{array}{l}\text{Rate of change}\\ \text{of } f \text{ in direction}\\ \text{of } \vec{u} \text{ at } (a, b)\end{array} = \lim_{h \to 0} \frac{f(a + hu_1, b + hu_2) - f(a, b)}{h},$$

provided the limit exists.

Remark. Notice that if $\vec{u} = \vec{i}$, so $u_1 = 1, u_2 = 0$, then the directional derivative is the partial derivative f_x, since

$$f_{\vec{i}}(a, b) = \lim_{h \to 0} \frac{f(a + h, b) - f(a, b)}{h} = f_x(a, b)$$

Similarly, if $\vec{u} = \vec{j}$ then the directional derivative $f_{\vec{j}} = f_y$:

$$f_{\vec{j}}(a, b) = \lim_{h \to 0} \frac{f(a, b + h) - f(a, b)}{h} = f_y(a, b)$$

If f is differentiable, we can use local linearity to find a formula for the directional derivative which does not involve a limit. If $\vec{u}$ is a unit vector, the definition of $f_{\vec{u}}$ says

$$f_{\vec{u}}(a, b) = \lim_{h \to 0} \frac{f(a + hu_1, b + hu_2) - f(a, b)}{h} = \lim_{h \to 0} \frac{\Delta f}{h}$$

where $\Delta f = f(a+hu_1, b+hu_2) - f(a,b)$ is the change in f. We write Δx for the change in x, so $\Delta x = (a+hu_1) - a = hu_1$; similarly $\Delta y = (b+hu_2) - b = hu_2$. Using local linearity, we have

$$\begin{aligned} f(x,y) &\approx f(a,b) + f_x(a,b)(x-a) + f_y(a,b)(y-b) \\ f(x,y) &\approx f(a,b) + \Delta f \end{aligned}$$

Graphically, this linear approximation can be shown as follows:

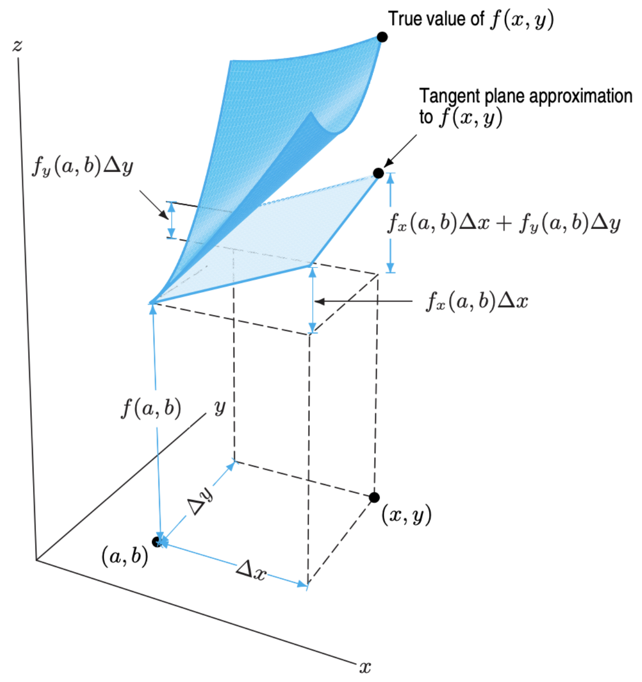

Figure 2: Local linearization: Approximating $f(x,y)$ by the z-value from the tangent plane

The expression in blue in the equation above can be used to approximate Δf:

$$\begin{aligned} \Delta f &\approx f_x(a,b)(x-a) + f_y(a,b)(y-b) \\ &\approx f_x(a,b)\Delta x + f_y(a,b)\Delta y = f_x(a,b)hu_1 + f_y(a,b)hu_2 \end{aligned}$$

Thus, dividing Δf by h gives

$$\begin{aligned} \frac{\Delta f}{h} &\approx \frac{f_x(a,b)hu_1 + f_y(a,b)hu_2}{h} \\ &\approx \frac{\cancel{h}\Big(f_x(a,b)u_1 + f_y(a,b)u_2\Big)}{\cancel{h}} = f_x(a,b)u_1 + f_y(a,b)u_2 \end{aligned}$$

This approximation becomes exact as $h \to 0$ or as we get closer and closer to $f(a,b)$, so we have the following formula:

$$f_{\vec{u}}(a,b) = f_x(a,b)u_1 + f_y(a,b)u_2$$

3.2 Gradient Vector

Notice that the expression for $f_{\vec{u}}(a,b)$ can be written as a dot product of $\vec{u}$ and a new vector:

$$\begin{aligned} f_{\vec{u}}(a,b) &= f_x(a,b)u_1 + f_y(a,b)u_2 \\ &= \left(f_x(a,b)\vec{i} + f_y(a,b)\vec{j}\right) \cdot \left(u_1\vec{i} + u_2\vec{j}\right) \\ &= \langle f_x(a,b), f_y(a,b)\rangle \cdot \langle u_1, u_2\rangle \\ &= \begin{bmatrix} \frac{\partial}{\partial x} f \\ \frac{\partial}{\partial y} f \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{aligned}$$

Thus, the **gradient Vector** of a differentiable function f at the point (a,b) is

$$\operatorname{grad} f(a,b) = f_x(a,b)\vec{i} + f_y(a,b)\vec{j}$$

$$\nabla f = \begin{bmatrix} \frac{\partial}{\partial x} f \\ \frac{\partial}{\partial y} f \end{bmatrix}$$

The fact that the directional derivative $f_{\vec{u}} = \operatorname{grad} f \cdot \vec{u}$ enables us to see what the gradient vector represents. Suppose θ is the angle between the vectors $\operatorname{grad} f$ and $\vec{u}$. At the point (a,b), using the geometric definition of the dot project, we have

$$f_{\vec{u}} = \operatorname{grad} f \cdot \vec{u} = \|\operatorname{grad} f\| \underbrace{\|\vec{u}\|}_{1} \cos\theta = \|\operatorname{grad} f\| \cos\theta$$

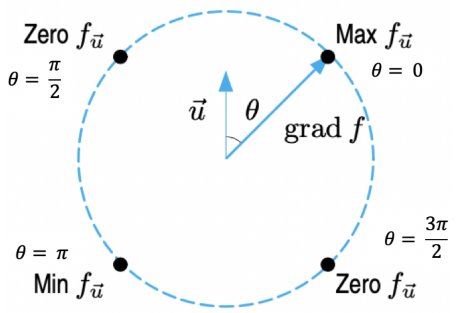

Figure 3: Values of the directional derivative $f_{\vec{u}}$ at different angles to the gradient

Imagine that $\operatorname{grad} f$ is fixed and that $\vec{u}$ can rotate. The maximum value of the directional derivative $f_{\vec{u}}$ occurs when $\cos\theta = 1$, where $\theta = 0$, and $\vec{u}$ is pointing in the direction of the gradient vectorgrad f. Then

$$\text{Maximum } f_{\vec{u}} = \|\operatorname{grad} f\| \cos 0 = \|\operatorname{grad} f\|$$

The minimum value of $f_{\vec{u}}$ occurs when $\cos\theta = -1$, so $\theta = \pi$ and $\vec{u}$ is pointing in the direction opposite to $\operatorname{grad} f$. Then

$$\text{Minimum } f_{\vec{u}} = \|\operatorname{grad} f\| \cos\pi = -\|\operatorname{grad} f\|$$

When $\theta = \frac{\pi}{2}$ or $\frac{3\pi}{2}$, so $\cos\theta = 0$, the directional derivative is zero. Thus, **the gradient vector points in the direction of the greatest rate of change at a point and the magnitude of the gradient vector is that rate of change.**

Geometric Properties of the Gradient Vector in the Plane
If f is a differentiable function at the point (a, b) and $\operatorname{grad} f(a, b) \neq \overrightarrow{0}$, then:

- The direction of $\operatorname{grad} f(a, b)$ is
  - Perpendicular to the contour of f through (a, b)
  - In the direction of the maximum rate of increase of f.
- The magnitude of the gradient vector, $\|\operatorname{grad} f\|$, is
  - The maximum rate of change of f at that point;
  - Large when the contours are close together and small when they are far apart.

The gradient vector is defined for the general case where f has any arbitrary number of variables. In vector calculus, the gradient of a scalar-valued differentiable function f of several variables is the vector field (or vector-valued function) ∇f whose value at a point p is the vector whose components are the **partial derivatives** of f at p. That is, for $f : \mathbb{R}^n \to \mathbb{R}$, its gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is defined at the point $p = (x_1, \ldots, x_n)$ in n-dimensional space as the vector:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

The nabla symbol ∇, written as an upside-down triangle and pronounced "del", denotes the vector differential operator.

3.3 Batch Gradient Descent

To find the gradient vector of of the cost function MSE, we need to compute the gradient of the cost function with respect to each estimated model parameter $\hat{\beta}_k$. In statistics notation, the derivation is as follows:

$$\begin{aligned} \text{MSE}_{\text{predictor}} &= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)^2 \\ \frac{\partial}{\partial \hat{\beta}_k}\text{MSE}_{\text{predictor}} &= \frac{\partial}{\partial \hat{\beta}_k}\left[\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)^2\right] \end{aligned}$$

The derivative of the sum of $1, \cdots, i, \cdots, n$ MSE functions is the same as the sum of the derivatives of each of those MSE functions. Therefore, the order of the summation operator and the partial derivative can be switched. Note: we also take out $\frac{1}{n}$ to the front since it is a constant (using the constant multiple rule of both the derivative and the summation operator):

$$\frac{\partial}{\partial \hat{\beta}_k}\text{MSE}_{\text{predictor}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial \hat{\beta}_k}\left[\left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)^2\right]$$

Applying the chain rule and the power rule

$$= \frac{1}{n}\sum_{i=1}^{n} 2\left(Y_i - \hat{\beta}_0 \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)\frac{\partial}{\partial \hat{\beta}_k}\left(Y_i - \hat{\beta}_0 \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)$$

Factor out the constant 2

$$= \frac{2}{n}\sum_{i=1}^{n} \left(Y_i - \hat{\beta}_0 \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)\frac{\partial}{\partial \hat{\beta}_k}\left(\overset{0}{\cancel{Y_i}} - \overset{0}{\cancel{\hat{\beta}_0}} \cdots - \overset{1}{\cancel{\hat{\beta}_k}} X_{ik} \cdots - \overset{0}{\cancel{\hat{\beta}_{p-1} X_{i,p-1}}}\right)$$

The partial derivative of $\frac{\partial}{\partial \hat{\beta}_k}(Y_i - \hat{Y}_i) = 1X_{ik} = X_{ik}$ since all other variables are fixed constants

$$= \frac{2}{n}\sum_{i=1}^{n} \left(Y_i - \hat{\beta}_0 \cdots - \hat{\beta}_k X_{ik} \cdots - \hat{\beta}_{p-1} X_{i,p-1}\right)\left(X_{ik}\right)$$
$$= \frac{2}{n}\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)\left(X_{ik}\right)$$

Finally, the **partial derivative of the cost function** with respect to estimated coefficient $\hat{\beta}_k$ can be expressed as follows:

$$\frac{\partial}{\partial \hat{\beta}_k}\text{MSE} = \frac{2}{n}\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)\left(X_{ik}\right)$$

The gradient vector is therefore:

$$\nabla \text{MSE} = \underset{p\times 1}{\begin{bmatrix} \frac{\partial}{\partial \hat{\beta}_0}\text{MSE} \\ \frac{\partial}{\partial \hat{\beta}_1}\text{MSE} \\ \vdots \\ \frac{\partial}{\partial \hat{\beta}_{p-1}}\text{MSE} \end{bmatrix}} = \underset{p\times 1}{\begin{bmatrix} \frac{2}{n}\sum_{i=1}^{n}\big(Y_i - \hat{Y}_i\big)\big(X_{i0}\big) \\ \frac{2}{n}\sum_{i=1}^{n}\big(Y_i - \hat{Y}_i\big)\big(X_{i1}\big) \\ \vdots \\ \frac{2}{n}\sum_{i=1}^{n}\big(Y_i - \hat{Y}_i\big)\big(X_{i,p-1}\big) \end{bmatrix}} = \frac{2}{n}\underset{p\times 1}{\begin{bmatrix} \sum_{i=1}^{n}\big(Y_i - \hat{Y}_i\big)\big(X_{i0}\big) \\ \sum_{i=1}^{n}\big(Y_i - \hat{Y}_i\big)\big(X_{i1}\big) \\ \vdots \\ \sum_{i=1}^{n}\big(Y_i - \hat{Y}_i\big)\big(X_{i,p-1}\big) \end{bmatrix}}$$

$$= \frac{2}{n}\begin{bmatrix} \underset{1\times n}{\begin{bmatrix} X_{1,0} & \cdots & X_{n,0}\end{bmatrix}} & \underset{n\times 1}{\begin{bmatrix} Y_1 - \hat{Y}_1 \\ \vdots \\ Y_n - \hat{Y}_n \end{bmatrix}} \\ \underset{1\times n}{\begin{bmatrix} X_{11} & \cdots & X_{n1}\end{bmatrix}} & \underset{n\times 1}{\begin{bmatrix} Y_1 - \hat{Y}_1 \\ \vdots \\ Y_n - \hat{Y}_n \end{bmatrix}} \\ \vdots & \vdots \\ \underset{1\times n}{\begin{bmatrix} X_{1,p-1} & \cdots & X_{n,p-1}\end{bmatrix}} & \underset{n\times 1}{\begin{bmatrix} Y_1 - \hat{Y}_1 \\ \vdots \\ Y_n - \hat{Y}_n \end{bmatrix}} \end{bmatrix}$$

$$= \frac{2}{n}\underset{p\times n}{\begin{bmatrix} X_{1,0} & \cdots & X_{n,0} \\ X_{11} & \cdots & X_{n1} \\ \vdots & & \\ X_{1,p-1} & \cdots & X_{n,p-1} \end{bmatrix}}\underset{n\times 1}{\begin{bmatrix} Y_1 - \hat{Y}_1 \\ \vdots \\ Y_n - \hat{Y}_n \end{bmatrix}}$$

$$= \frac{2}{n}\left[\underset{p\times n}{\mathbf{X}'}(\underset{n\times 1}{\mathbf{Y}} - \underset{n\times 1}{\hat{\mathbf{Y}}})\right]$$

$$= \frac{2}{n}\left[\underset{p\times n}{\mathbf{X}'}(\underset{n\times 1}{\mathbf{Y}} - \underset{n\times p}{\mathbf{X}}\underset{p\times 1}{\hat{\boldsymbol{\beta}}})\right]$$

Therefore, the gradient vector in matrix notation can be conveniently expressed as follows:

$$\underset{p\times 1}{\nabla \text{MSE}} = \frac{2}{n}\left[\underset{p\times n}{\mathbf{X}'}(\underset{n\times 1}{\mathbf{Y}} - \underset{n\times p}{\mathbf{X}}\underset{p\times 1}{\hat{\boldsymbol{\beta}}})\right]$$

Notice that this formula involves calculations over the full training set **X**, at each gradient descent step. This is why the algorithm is called **Batch Gradient Descent**: it uses the whole batch of training data at every step. As a result it is terribly slow on very large training sets. However, Gradient Descent scales well with the number of features; training a Linear Regression model when there are hundreds of thousands of features is much faster using Gradient Descent than using the Normal Equation or SVD decomposition.

Once we have obtained the gradient vector, which points in the direction of the greatest initial increase at the random initialization $(\hat{\beta}_0, \cdots, \hat{\beta}_{p-1})$, we need only proceed in the opposite direction to move towards the global minimum. This means subtracting $\nabla\,\text{MSE}$ from $\hat{\boldsymbol{\beta}}$. To determine the step size, we also involve a learning rate η (pronounced EE-Tuh), multiplying the gradient vector by η determines the amount by we decrease the vector $\hat{\boldsymbol{\beta}}$.

$$\underset{p\times 1}{\hat{\boldsymbol{\beta}}^{(\text{Next})}} = \underset{p\times 1}{\hat{\boldsymbol{\beta}}^{(\text{Current})}} - \eta\nabla \underset{p\times 1}{\text{MSE}}$$

3.4 Stochastic Gradient Descent

Benefits: The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large. At the opposite extreme, **Stochastic Gradient Descent** picks a random instance in the training set at every step and computes the gradients based only on that single instance or trial. Working on a single instance at a time makes the algorithm much faster because it has very little data to manipulate at every iteration. It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration.

Downside: On the other hand, due to its stochastic (i.e., random) nature, this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average. Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down. So once the algorithm stops, the final parameter values are good, but not optimal.

Solution: This randomness is good to escape from local optima, but bad because it means that the algorithm can never settle at the minimum. One solution to this dilemma is to gradually reduce the learning rate. The steps start out large (which helps make quick progress and escape local minima), then get smaller and smaller, allowing the algorithm to settle at the global minimum. The function that determines the learning rate at each iteration is called the **learning schedule**. If the learning rate is reduced too quickly, we may get stuck in a local minimum, or even end up frozen halfway to the minimum. If the learning rate is reduced too slowly, we may jump around the minimum for a long time and end up with a sub-optimal solution if we halt training too early.

For this algorithm, suppose the k^{th} instance or trial is randomly selected, the gradient vector is given as follows:

$$\nabla \text{MSE} = \underset{p\times 1}{\begin{bmatrix} \frac{\partial}{\partial \hat{\beta_0}}\text{MSE} \\ \frac{\partial}{\partial \hat{\beta_1}}\text{MSE} \\ \vdots \\ \frac{\partial}{\partial \hat{\beta_{p-1}}}\text{MSE} \end{bmatrix}}$$

$$= \frac{2}{n} \underset{p\times 1}{\begin{bmatrix} \big(Y_k - \hat{Y}_k\big)\big(X_{k0}\big) \\ \big(Y_k - \hat{Y}_k\big)\big(X_{k1}\big) \\ \vdots \\ \big(Y_k - \hat{Y}_k\big)\big(X_{k,p-1}\big) \end{bmatrix}}$$

$$= \frac{2}{n} \underset{p\times 1}{\begin{bmatrix} X_{k,0} \\ X_{k1} \\ \vdots \\ X_{k,p-1} \end{bmatrix}} \underset{1\times 1}{\begin{bmatrix} Y_k - \hat{Y}_k \end{bmatrix}}$$

$$= \frac{2}{n}\left[\underset{p\times 1}{\vec{X}_k'} (\underset{1\times 1}{\vec{Y}_k} - \underset{1\times 1}{\hat{Y}_k}) \right]$$

$$= \frac{2}{n}\left[\underset{p\times 1}{\vec{X}_k'} (\underset{1\times 1}{\vec{Y}_k} - \underset{1\times p}{\vec{X}_k} \underset{p\times 1}{\hat{\boldsymbol{\beta}}}) \right]$$

Therefore, the gradient vector in matrix notation can be conveniently expressed as follows for the stochastic gradient descent algorithm:

$$\underset{p\times 1}{\nabla \text{MSE}} = \frac{2}{n}\left[\underset{p\times 1}{\vec{X}_k'} (\underset{1\times 1}{\vec{Y}_k} - \underset{1\times p}{\vec{X}_k} \underset{p\times 1}{\hat{\boldsymbol{\beta}}}) \right]$$

3.5 Mini-batch Gradient Descent

The last Gradient Descent algorithm is called **Mini-batch Gradient Descent**. At each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch GD computes the gradients on small random sets of instances called mini-batches. The main advantage of Mini-batch GD over Stochastic GD is the performance boost from hardware optimization of matrix operations.

The algorithm's progress in the parameter space is less erratic than with Stochastic GD, especially with fairly large mini-batches. As a result, Mini-batch GD will end up walking around a bit closer to the minimum than Stochastic GD—but it may be harder for it to escape from local minima (in the case of problems that suffer from local minima, unlike Linear Regression). They all end up near the minimum, but Batch GD's path actually stops at the minimum, while both Stochastic GD and Mini-batch GD continue to walk around.

However, Batch GD takes a lot of time train the model, and Stochastic GD and Mini-batch GD would also reach the minimum if we use a good learning schedule.