Linear Model

Ying Nian Wu and Quanshi Zhang

March 22, 2020

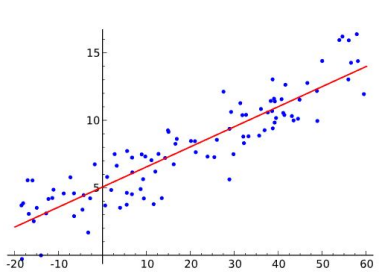
Contents

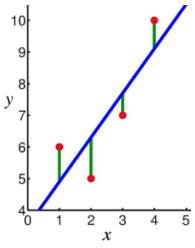
Note on Deep Learning and Backgrounds for M232A at University of California, Los Angeles (UCLA). All the figures are taken from the web.

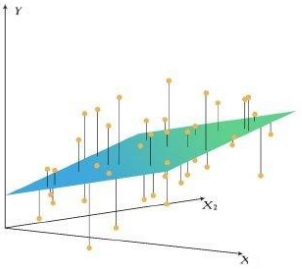
1 Basics

1.1 Linear regression

	input	output
1	$x_{11}, x_{12}, ..., x_{1p}$	y_1
2	$x_{21}, x_{22}, ..., x_{2p}$	y_2
...		
n	$x_{n1}, x_{n2}, ..., x_{np}$	y_n







$$X = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Figure 1: Linear regression. `https://en.wikipedia.org//wiki//Linear_regression`

The dataset of linear regression consists of an $n \times (p+1)$ matrix $\mathbf{X}^\top = [\mathbf{X}_1^\top, \mathbf{X}_2^\top, \ldots \mathbf{X}_n^\top]$, and a $n \times 1$ vector $\mathbf{Y}^\top = [y_1, y_2, \ldots, y_n]$, where $\mathbf{X}_i^\top = [1, x_{i1}, x_{i2}, \ldots, x_{ip}]$. The model is of the following form:

$$\begin{aligned} y_i &= \beta_0 1 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots + \beta_p x_{ip} + \varepsilon_i \\ &= \sum_{j=0}^{p} x_{ij}\beta_j + \varepsilon_i, \qquad \text{because } x_{i0} = 1 \\ y &= \mathbf{X}^\top \beta + \varepsilon \end{aligned}$$

for $i = 1, ..., n$, where $\varepsilon_i \sim \mathrm{N}(0, \sigma^2)$ independently. Here we are deliberately ambiguous about the intercept term. We absorb the intercept term to β, if we set $x_{i0} = 1$ for all i (*i.e.* we can design an additional dimension for x_i, $x_i^{new} \leftarrow [1, x_{i1}, x_{i2}, \ldots, x_{ip}]^\top$). y_i is called response variable, outcome, dependent variable. x_{ij} is called predictor, regressor, covariate, independent variable, or simply variable. In the experimental design setting, $\mathbf{X}$ is called the design matrix.

ε is a vector of values ε_i. This part of the model is called the error term, disturbance term, or sometimes noise (in contrast with the "signal" provided by the rest of the model). This variable captures all other

factors which influence the dependent variable y other than the regressors x. The relationship between the error term and the regressors, for example their correlation, is a crucial consideration in formulating a linear regression model, as it will determine the appropriate estimation method.

The process of estimating β is called learning from the training data. The purpose is two-fold.

(1) Explanation: understanding the relationship between y_i and $(x_{ij}, j = 1, ..., p)$.

(2) Prediction: learn to predict y_i based on $(x_{ij}, j = 1, ..., p)$, so that in the testing stage, if we are given the predictor variables, we should be able to predict the outcome.

	input	output
1	$X_1^\top$	y_1
2	$X_2^\top$	y_2
...		
n	$X_n^\top$	y_n

We can arrange the data in terms of $X_i^\top = (x_{ij}, j = 1, ..., p)$, where $X_i^\top$ is the i-th row of $\mathbf{X}$. We can write the model as

$$y_i = X_i^\top \beta + \varepsilon_i,$$

where $\beta = (\beta_j, j = 1, ..., p)^\top$.

We can also write the model as follows:

$$\text{Generic notation:} \qquad y = X^\top \beta + \varepsilon = \sum_{j=0}^{p} x_j \beta_j + \varepsilon$$

$$\text{Prediction:} \qquad y = X^\top \beta = \sum_{j=0}^{p} x_j \beta_j$$

where we drop the subscript i.

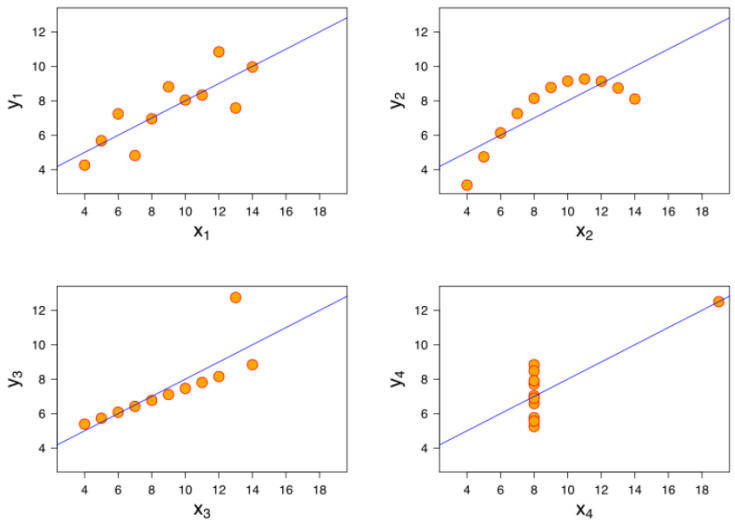


Figure 2: Data that are suitable and unsuitable for linear regression. `https://en.wikipedia.org//wiki//Linear_regression`

1.2 Logistic regression

Consider a dataset with n training examples, where $\mathbf{X}_i^\top = [x_{i1}, \cdots, x_{ip}]$ consists of p predictors, $y_i \in \{0,1\}$ is the boolean label.

- We assume all dimensions x_{ij} in x_i are conditionally independent given y_i.
- $\Pr(x_{ij}|y_i) \sim Gauss(\mu_j, \sigma_j^2)$.
- $\Pr(y_i|X_i) = \text{Bernoulli}(p_i)$, *i.e.* $\Pr(y_i = 1|X_i) = p_i$, $\Pr(y_i = 0|X_i) = 1 - p_i$

Let p_i denote the probability of $\Pr(y_i = 1|X_i) = p_i$, which is estimated by the regression model, *i.e.* $y_i \sim \text{Bernoulli}(p_i)$. We assume

$$\text{logit}(p_i) = \log \frac{p_i}{1-p_i} = X_i^\top \beta.$$

Then

$$p_i = \text{sigmoid}(X_i^\top \beta) = \frac{e^{X_i^\top \beta}}{1+e^{X_i^\top \beta}} = \frac{1}{1+e^{-X_i^\top \beta}},$$

where the sigmoid function $\text{sigmoid}(a) = \frac{e^a}{1+e^a} = \frac{1}{1+e^{-a}}$ is the inverse of the logit function.

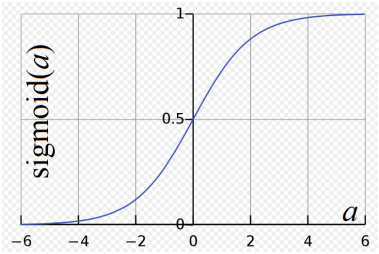


Figure 3: Sigmoid function. `https://en.wikipedia.org/wiki/Sigmoid_function`

Let $\{(X_i, y_i^*)\}_{i=1,\ldots,n}$ denote a set of training samples, where y_i^* is the ground-truth label. The probability for correct estimation is

$$\Pr(y_i = y_i^*|\mathbf{X}_i) = p_i^{y_i^*}(1-p_i)^{1-y_i^*} = \left(\frac{e^{X_i^\top \beta}}{1+e^{X_i^\top \beta}}\right)^{y_i^*}\left(1 - \frac{e^{X_i^\top \beta}}{1+e^{X_i^\top \beta}}\right)^{1-y_i^*} = \frac{e^{y_i^* X_i^\top \beta}}{1+e^{X_i^\top \beta}}$$

The likelihood of labels of all samples being correctly estimated is given as

$$\Pr(\beta) = \prod_{i=1}^n p_i^{y_i^*}(1-p_i)^{1-y_i^*} = \prod_{i=1}^n \frac{e^{y_i^* X_i^\top \beta}}{1+e^{X_i^\top \beta}}.$$

The corresponding log-likelihood is

$$\log \Pr(\beta) = \sum_{i=1}^n \left[y_i^* X_i^\top \beta - \log(1+\exp X_i^\top \beta)\right].$$

The maximum likelihood is to find the most plausible explanation to the observed data.

$$\underset{\beta}{\text{argmax}} \Pr(\beta) \equiv \underset{\beta}{\text{argmax}} \log \Pr(\beta)$$

An example

credit: `https://en.wikipedia.org/wiki/Logistic_regression`

The table shows the number of hours each student spent studying, and whether they passed (1) or failed (0).

Hours x_i	0.50	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50
Pass y_i^*	0	0	0	0	0	0	1	0	1	0
Hours x_i	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass y_i^*	1	0	1	0	1	1	1	1	1	1

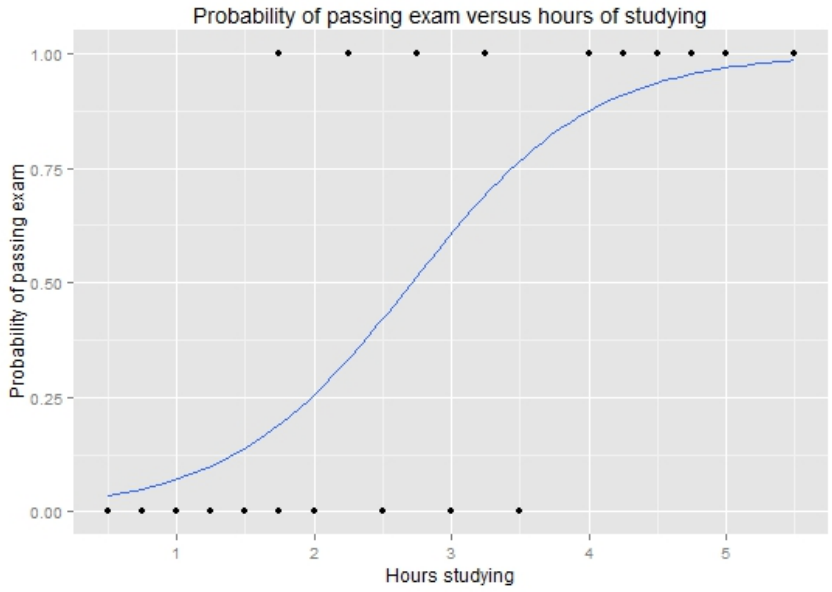


Figure 4: Logistic regression learned using the training data.

Let a student study for 3.50 hours, then $x_i = [1, 3.50]$ (where 1 is for the intercept term). The probability of him/her passing examination is

$$p_i = \frac{1}{1 + e^{-[1,3.50]\beta}}.$$

1.3 Classification

For the logistic regression, we want to learn β either for the purpose of explanation or understanding, or for the purpose of classification or prediction. In the context of classification, we usually let $y_i \in \{+1, -1\}$ instead of $y_i \in \{1, 0\}$. Those X_i with $y_i = +1$ are called positive examples, and those X_i with $y_i = -1$ are called negative examples.

We may call β a classifier. $X_i^\top \beta = \langle X_i, \beta \rangle$ is the projection of X_i on the vector β, so the vector β is the direction that reveals the difference between positive X_i and negative X_i. Thus β should be aligned with positive X_i and negatively aligned with negative X_i, i.e., β should point from the negative examples to the positive examples.

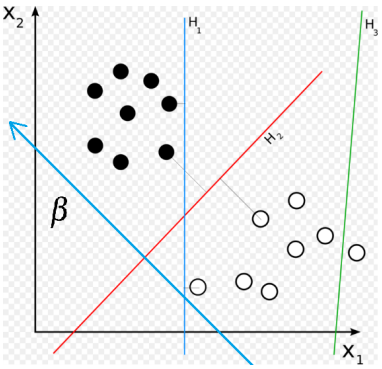


Figure 5: Linear classification. `https://en.wikipedia.org/wiki/Linear_classifier`

According to the previous subsection,

$$\Pr(y_i = +1) = \frac{1}{1+\exp(-X_i^\top \beta)},$$

and

$$\Pr(y_i = -1) = \frac{1}{1+\exp(X_i^\top \beta)}.$$

Combining them, we have

$$p(y_i) = \frac{1}{1+\exp(-y_i X_i^\top \beta)}.$$

1.4 Perceptron

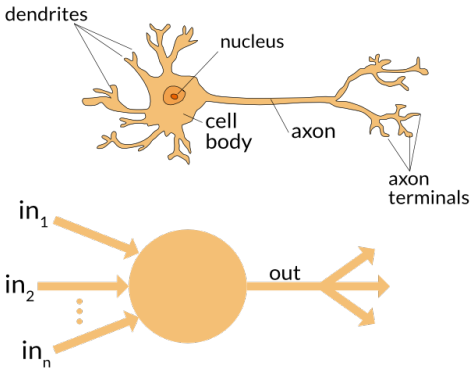


Figure 6: A neuron takes inputs, and generates outputs.

A deterministic version of the logistic regression is the perceptron model

$$y_i = \text{sign}(X_i^\top \beta),$$

where

$$\text{sign}(a) = \begin{cases} +1, & a > 0 \\ 0, & a = 0 \\ -1, & a < 0 \end{cases}$$

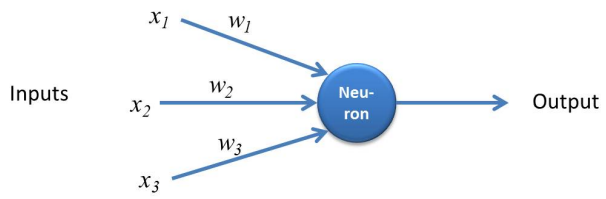


Figure 7: A perceptron is a simple model of a neuron. It computes a weighted sum of the inputs (plus a bias term), and outputs the sign of the weighted sum.

The perceptron model is inspired by neural science. See Figure 6. It can be considered an over-simplified model of a neuron, which takes input X_i, and emits output y_i. See Figure 7.

The perceptron model can be generalized to neural networks, support vector machine, as well as adaboost, which are three big tools for classification.

Notationally, in machine learning literature, the perceptron is often written as

$$y_i = \text{sign}\left(\sum_{j=1}^{p} w_j x_{ij} + b\right) = \text{sign}(X_i^\top w + b),$$

where $w = (w_j, j = 1, ..., p)^\top$ are the connection weights, and b is the bias term. (w, b) corresponds to β.

x_{i1}	x_{i2}	y_i
0	0	0
0	1	1
1	0	1
1	1	0

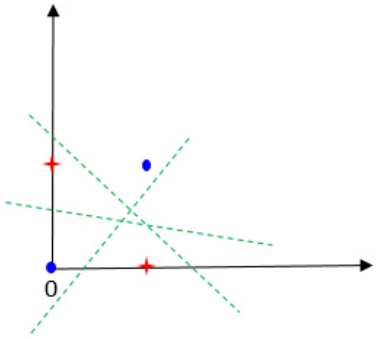


Figure 8: A perceptron is a linear model, which cannot solve the XOR problem. `https://blog.csdn.net/yangfeisc/article/details/45486067`

1.5 Three models

credit: `https://en.wikipedia.org/wiki/Generative_model`

Generative models

In statistical classification, two main approaches are called the generative approach and the discriminative approach. These compute classifiers by different approaches, differing in the degree of statistical modelling.

Given an observable variable X and a target variable Y, a generative model is a statistical model of the joint probability distribution on $X \times Y$, $p_\theta(X, Y)$;

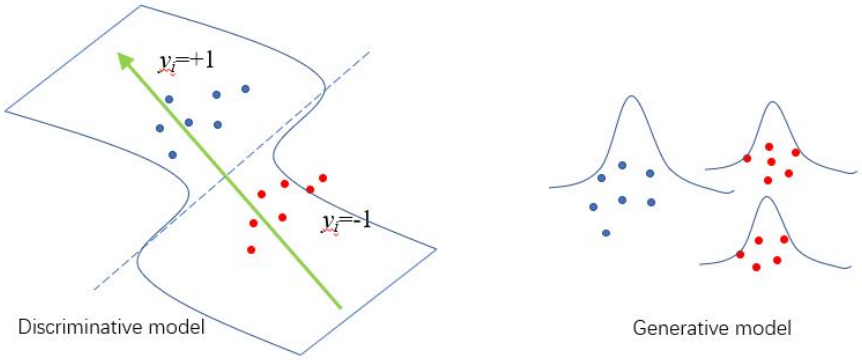


Figure 9: Discriminative models vs. generative models

Do the conditional inference.

$$p_\theta(Y|X) = \frac{p_\theta(X,Y)}{p_\theta(X)} = \frac{p_\theta(X,Y)}{\sum_{Y'} p_\theta(X,Y')}$$

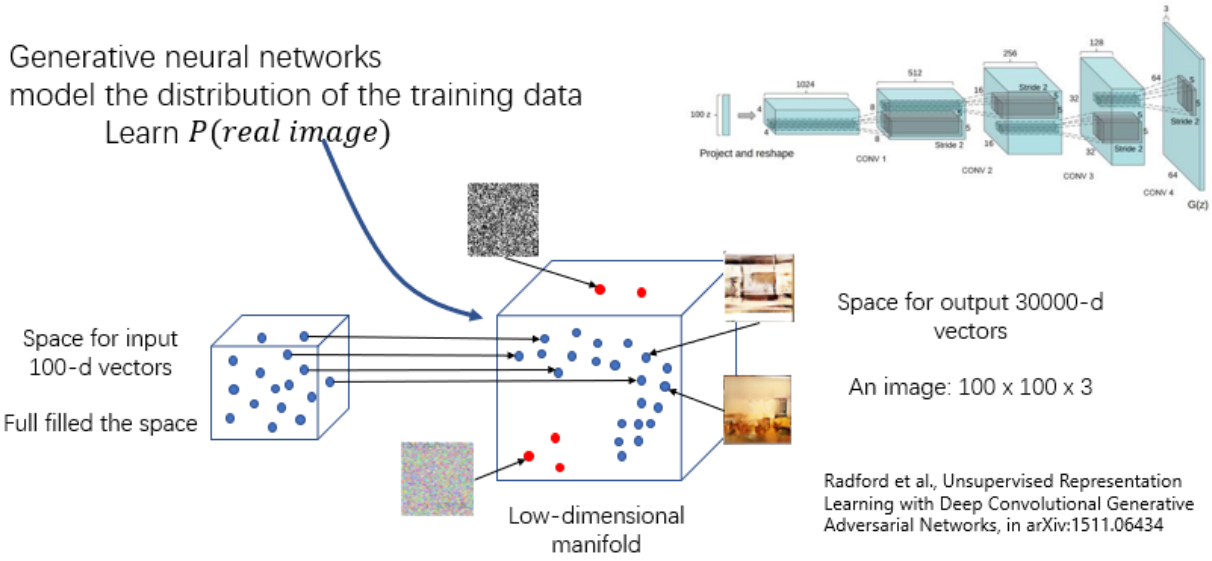


Figure 10: A generative neural network.

In deep learning, the generative neural network g_θ is usually given as

$$h \sim p(h) \sim \mathrm{N}(0,I),\ X = g_\theta(h) + \varepsilon,$$

where h is the input of the generative neural network, which is a low-dimensional vector. X is the network output, *e.g.* an generated image.

$$p_\theta(X) = \int p(h) p_\theta(X|h) dh.$$

$$X - g_\theta(h) = \varepsilon \sim \mathrm{N}(0, \sigma^2 \mathbf{I}) \quad \Rightarrow \quad X|h \sim \mathrm{N}(g_\theta(h), \sigma^2 \mathbf{I})$$

$$\log p_\theta(X|h) = -\frac{1}{2}(X - g_\theta(h))^T \Sigma^{-1}(X - g_\theta(h)) + \text{constant} = -\frac{\|X - g_\theta(h)\|^2}{2\sigma^2} + \text{constant},$$

where $\Sigma = \sigma^2 \mathbf{I}$ is the covariance matrix of ε.

$$\log p(h) = -\frac{1}{2}\|h\|^2 + \text{constant}.$$

Definition of the covariance matrix: $\Sigma = (c_{ij})_{p\times p}$.
Let X is a $n \times p$ matrix. x_{ki} denotes the i-th dimension of the feature vector of the k-th sample.

$$c_{ij} = \mathbb{E}_k\Big[(x_{ki} - \mathbb{E}_{i'}[x_{ki'}])(x_{kj} - \mathbb{E}_{j'}[x_{kj'}])\Big]$$

A more general case: for $(X_i, Y_i)_{i=1,\dots,n}$, the covariance matrix is

$$cov(X,Y) = (c_{ij})_{p\times p}$$

$$c_{ij} = \mathbb{E}_k\Big[(x_{ki} - \mathbb{E}_{i'}[x_{ki'}])(y_{kj} - \mathbb{E}_{j'}[y_{kj'}])\Big]$$

- $cov(X,Y) = cov(Y,X)^\top$
- $cov(AX + b, Y) = Acov(X,Y)$, where A is a matrix, and b is a vector.
- $cov(X + Y, Z) = cov(X,Z) + cov(Y,Z)$

Discriminative models

A discriminative model is a model of the conditional probability of the target Y, given an observation X, symbolically, $P(Y|X)$. Classifiers computed without using a probability model are also referred to loosely as "discriminative."

- Probabilistic: $p_\theta(y|x)$
- Deterministic: $y = g_\theta(x)$ Sometimes, deterministic models can be represented as $y = \text{argmax}_y w^\top \phi(x,y)$. In this case, $p_\theta(y|x) \propto e^{w^\top \phi(x,y)}$.

In deep learning, a discriminative neural network with the softmax output layer is

$$p_\theta(y = k|X) = p_k = \frac{1}{Z(\theta)} \exp(f_\theta^{(k)}(X)),$$

where $f_\theta^{(k)}(X)$ is a K-dimensional network output with respect to the input X before the softmax layer. $f_\theta^{(k)}(X)$ indicates the k-th output dimension.
for $k = 1, \dots, K$, and

$$Z(\theta) = \sum_k \exp(f_\theta^{(k)}(X)).$$

1.6 Descriptive model

The descriptive model aims to estimate the data distribution $p_\theta(X)$, which can be given as

$$p_\theta(X) = \frac{1}{Z(\theta)} \exp(f_\theta(X)),$$

where

$$Z(\theta) = \int \exp(f_\theta(x))dx.$$

1.7 Loss functions

In order to learn β from the training data $\{(X_i, y_i)\}_{i=1,\dots,n}$, we can minimize the loss function

$$\mathscr{L}(\beta) = \sum_{i=1}^{n} L(y_i, X_i^\top \beta),$$

where $L(y_i, X_i^\top \beta)$ is the loss for each training example (X_i, y_i). We need to define $L(y_i, X_i^\top \beta)$.

Loss function for least squares regression

For linear regression, we usually use the least squares loss,

$$L(y_i, X_i^\top \beta) = \left(y_i - X_i^\top \beta\right)^2.$$

The least squares loss for linear regression can also be derived from the log-likelihood if we assume the errors follow a normal distribution.

$$\begin{aligned}
& y_i - X_i^\top \beta = \varepsilon_i \sim \mathrm{N}(0,1) \\
\Rightarrow\quad & \Pr(y_i|X_i) = \Pr(\varepsilon_i) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\varepsilon_i^2}{2}} = \frac{1}{\sqrt{2\pi}} e^{-\frac{(y_i - X_i^\top \beta)^2}{2}} \\
\Rightarrow\quad & \log \Pr(y_i|X_i) = -\frac{(y_i - X_i^\top \beta)^2}{2} + \text{constant} \\
\Rightarrow\quad & \mathscr{L}(\beta) = \sum_{i=1}^{n} L(y_i, X_i^\top \beta) = -2 \sum_{i=1}^{n} [\log \Pr(y_i|X_i) - \text{constant}] = \sum_{i=1}^{n} \left(y_i - X_i^\top \beta\right)^2, \qquad \text{Maximum likelihood estimation}
\end{aligned}$$

Loss function for robust linear regression

We may also use the mean absolute value loss,

$$L(y_i, X_i^\top \beta) = \left|y_i - X_i^\top \beta\right|,$$

which penalizes big difference between y_i and $X_i^\top \beta$ to a less degree than the least squares loss, so that the estimated β is less affected by the outliers.

A combination of least squares loss and mean absolute value loss is the Huber loss[2],

$$L(y_i, X_i^\top \beta) = \begin{cases} \frac{1}{2}\left(y_i - X_i^\top \beta\right)^2, & \text{if } |y_i - X_i^\top \beta| \le \delta \\ \delta |y_i - X_i^\top \beta| - \frac{\delta^2}{2}, & \text{otherwise} \end{cases}$$

where δ is a cut-off value that is heuristically chosen, so that beyond δ we penalize the deviation by absolute value.

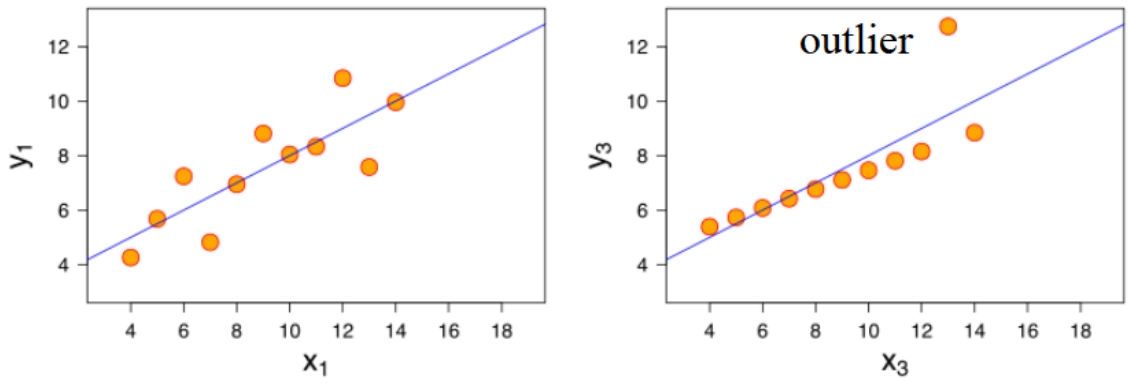


Figure 11: Linear regression with and without outliers.

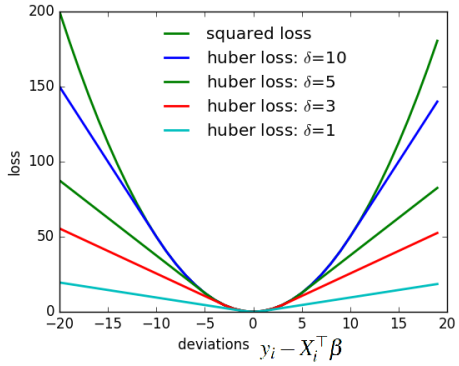


Figure 12: Huber loss. `https://blog.csdn.net/lanchunhui/article/details/50427055`

Loss function for logistic regression with 0/1 responses

For logistic regression, we usually maximize the likelihood function, which is

$$\prod_{i=1}^{n} \Pr(y_i|X_i,\beta)$$

That is, we want to find β to maximize the probability of the observed $(y_i, i = 1,...,n)$ given $(X_i, i = 1,...,n)$. The maximum likelihood estimate gives the most plausible explanation to the observed data.

For $y_i \in \{0,1\}$,

$$\Pr(y_i = 1|X_i,\beta) = \text{sigmoid}(X_i^\top \beta) = \frac{\exp(X_i^T \beta)}{1+\exp(X_i^\top \beta)},$$

$$\Pr(y_i = 0|X_i,\beta) = 1 - \Pr(y_i = 1|X_i,\beta) = \frac{1}{1+\exp(X_i^\top \beta)},$$

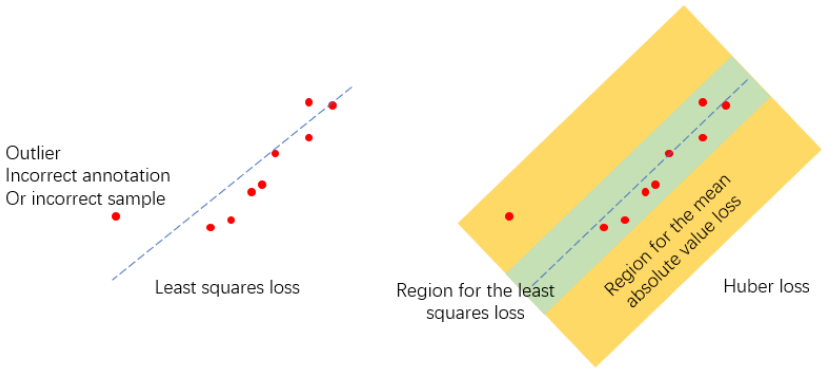


Figure 13: Huber loss.

We can combine the above two equations by

$$\begin{aligned}\Pr(y_i|X_i,\beta) &= \Pr(y_i=1|X_i,\beta)^{y_i}\Pr(y_i=0|X_i,\beta)^{1-y_i} \\ &= \left[\frac{\exp(X_i^T\beta)}{1+\exp(X_i^\top\beta)}\right]^{y_i}\left[\frac{1}{1+\exp(X_i^\top\beta)}\right]^{1-y_i} \\ &= \frac{\exp(y_iX_i^T\beta)}{1+\exp(X_i^\top\beta)}.\end{aligned}$$

The log-likelihood is

$$\sum_{i=1}^{n}\log\Pr(y_i|X_i,\beta)=\sum_{i=1}^{n}\left[y_iX_i^\top\beta-\log(1+\exp(X_i^\top\beta))\right].$$

We can define the loss function as the negative log-likelihood

$$L(y_i,X_i^\top\beta)=-\left[y_iX_i^\top\beta-\log(1+\exp(X_i^\top\beta))\right].$$

Loss function for logistic regression with $\pm$ responses

If $y_i\in\{+1,-1\}$, we have

$$\Pr(y_i=+1)=\frac{1}{1+\exp(-X_i^\top\beta)},$$

and

$$\Pr(y_i=-1)=\frac{1}{1+\exp(X_i^\top\beta)}.$$

Combining them, we have

$$p(y_i)=\frac{1}{1+\exp(-y_iX_i^\top\beta)}.$$

The log-likelihood is

$$\sum_{i=1}^{n}\log\Pr(y_i|X_i,\beta)=-\sum_{i=1}^{n}\log\left(1+\exp(-y_iX_i^\top\beta)\right).$$

We define the loss function as the negative log-likelihood. Thus

$$L(y_i, X_i^\top \beta) = \log\left[1 + \exp(-y_i X_i^\top \beta)\right].$$

This loss is called the **logistic loss**.

Loss functions for classification

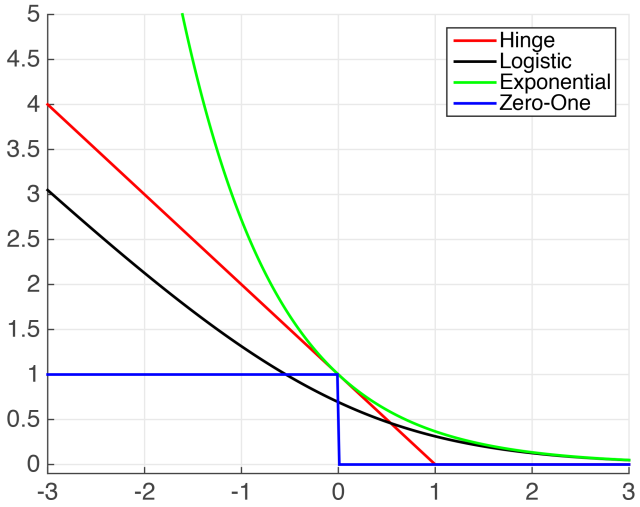


Figure 14: Loss functions for classification. The horizontal axis is $m_i = y_i X_i^\top \beta$. The vertical axis is $L(y_i, X_i^\top \beta)$. The exponential loss and the hinge loss can be considered approximations to the logistic loss. These loss functions penalize negative m_i. The more negative m_i is, the bigger the loss. The loss functions also penalize small positive m_i, e.g., those $m_i < 1$. Such loss functions encourage correct and confident classifications.

The following summarizes possible choices for the loss function $L(y_i, X_i^\top \beta)$ for classification. See Figure 14.

$$\begin{aligned}
&\text{Logistic loss} = \log\left(1 + \exp(-y_i X_i^\top \beta)\right),\\
&\text{Exponential loss} = \exp\left(-y_i X_i^\top \beta\right),\\
&\text{Hinge loss} = \max\left(0, 1 - y_i X_i^\top \beta\right),\\
&\text{Zero-one loss} = 1\left(y_i X_i^\top \beta < 0\right)
\end{aligned}$$

Both the exponential and hinge losses can be considered approximations to the logistic loss. The logistic loss is used by logistic regression. The exponential loss is used by adaboost. The hinge loss is used by support vector machine. The zero-one loss is to count the number of mistakes. It is not differentiable and is not used for training.

All the above loss functions are based on $m_i = y_i X_i^\top \beta$. We call m_i the margin for example (y_i, X_i). We want y_i and $X_i^\top \beta$ to be of the same sign for correct classification. If $y_i = +1$, we want $X_i^\top \beta$ to be very positive. If $y_i = -1$, we want $X_i^\top \beta$ to be very negative. We want the margin m_i to be as large as possible for confident classification.

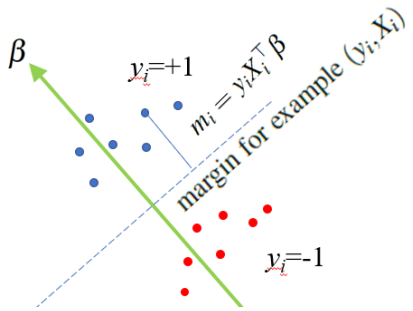


Figure 15: $m_i = y_i X_i^\top \beta$, the margin for example (y_i, X_i).

1.8 Least Squares

One of the key concepts in statistical learning is overfitting. In this section and the next, we'll study this concept in detail, in the least-squares and classification settings. First, consider a linear regression model $Y = X\beta + \varepsilon$, where X is $n \times p$, with $n > p$, and $\varepsilon \sim N(0, \sigma^2 I_n)$.

A special case: we will assume X is orthonormal ($X^T X = I_p$). Note that this is without loss of generality: if X is not orthonormal, one can employ a QR decomposition of X to obtain a corresponding matrix that is itself orthonormal.

Let's look for the least square estimate $\hat{\beta}$:

$$\hat{\beta} = \arg\min_{\beta} |Y - X\beta|^2$$

The first order condition is:

$$2X^T(Y - X\beta) = 0$$

Which yields the well-known solution:

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

A special case: If X is orthonormal ($X^T X = I_p$), this can be simplified, and $\hat{\beta} = I_p^{-1} X^T Y = X^T Y$.

Note that $X\hat{\beta}$ is simply the projection of Y onto the space spanned by the p columns of X. Define that space as $\Pi(X)$. The linear map that projects any vector onto $\Pi(X)$ is given by $P_X = X(X^T X)^{-1} X^T$. Thus, the projection of Y onto $\Pi(X)$ (let's call it $\hat{Y}$) is given by:

$$\hat{Y} = \Pi(X)Y = X(X^T X)^{-1} X^T Y = X\hat{\beta}, \qquad \text{reconstruction of } Y$$

In essence, Y can be seen as the sum of its projection onto the space spanned by the columns of X, plus a vector of residuals $\varepsilon = Y - \hat{Y}$, which is orthogonal to that space.

The reconstructed sample $\hat{Y} = X\hat{\beta}$ reflects the signal that can be represented by the current model, while the residual $\varepsilon = Y - \hat{Y}$ represents the signal that cannot be encoded by the model.

Distribution of $\hat{\beta}$

The distribution of $\hat{\beta}$ indicates the stability or robustness of the parameters with respect to noises in training samples.

We can derive the distribution of $\hat{\beta}$. Assume that the model above is true, and that the true β is β_{true}.

Then, we have:

$$\begin{aligned}\hat{\beta} &= (X^T X)^{-1} X^T Y \\ &= (X^T X)^{-1} X^T (X\beta_{true} + \varepsilon) \\ &= \beta_{true} + X^{-1}\varepsilon \\ &= \beta_{true} + X^T \varepsilon \qquad \text{because } X \text{ is orthonormal.}\end{aligned}$$

Given $\varepsilon \sim N(0, \sigma^2 I_n)$, and assuming X is given, this equation implies that each $\hat{\beta}_j$ is normally distributed, and thus the distribution for $\hat{\beta}$ is multivariate normal. Let's find its first and second moments:

Taking the expectation of $\hat{\beta}$: $E[\hat{\beta}] = \beta_{true}$.

And $Var(\hat{\beta}) = X^T Var(\varepsilon) X = X^T \sigma^2 I_n X = \sigma^2 X^T X$.
This implies:

$$\hat{\beta} = N(\beta_{true}, \Sigma = \sigma^2 X^T X)$$

A special case: we used the property that X is orthonormal.
And $Var(\hat{\beta}) = \sigma^2 X^T X = \sigma^2 I_p$.

$$\hat{\beta} = N(\beta_{true}, \sigma^2 I_p)$$

1.9 Kullback-Leibler divergence and cross entropy

Coding and entropy

We define the entropy of a distribution $p(x)$ as

$$\text{entropy}(p) = \mathrm{E}_p[-\log p(X)] = \sum_x p(x)[-\log p(x)].$$

Consider a distribution $p(x)$ on an alphabet A, B, C, D, $p(A) = 1/2$, $p(B) = 1/4$, $p(C) = 1/8$, $p(D) = 1/8$. We can generate $X \sim p(x)$ using coin flipping. We flip a coin, if it is head, we output A. If it is tail, we continue to flip. If it is head, we output B. If it is tail, we continue to flip. If it is head, we output C. If it is tail, we output D. So $A = H$, $B = TH$, $C = TTH$, $D = TTT$. The expected number of coin flippings is entropy(p).

x	A	B	C	D
$p(x)$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$
$-\ log_2 p(x)$	1	2	3	3
$code$	1	01	001	000

Figure 16: Distribution, coin flipping, and code

We can also change the above coin flippings into a binary code $A = 1$, $B = 01$, $C = 001$, $D = 000$. For a sequence generate from $p(x)$, we can code it and transmit it as a binary sequence. There are two important observations. (1) We can recover the original sequence from the binary sequence. This is because the binary

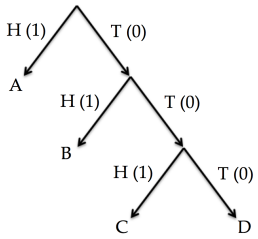


Figure 17: Coin flipping and coding

code of any letter is not the beginning part of the binary code of any other letter. That is, the code is prefix code. This is because in the coin flipping experiment, when we output a letter, the experiment then stops. So the sequence of coin flippings that produces one letter will not be the beginning part of the sequence of coin flippings that produces another letter. (2) For a random sequence produced by $p(x)$, the corresponding binary sequence is a sequence of iid realizations from Bernoulli(1/2), i.e., a sequence of random coin flippings. This is because the binary code is based on the coin flipping experiment that reproduces $p(x)$. So entropy(p) is the expected coding length. The unit of coding length is termed "bit."

This has a profound consequence. The random binary sequence cannot be further compressed in general. Otherwise, for example, for a random sequence of length n, if we can shorten it to a sequence of $.9n$, then there can only be $2^{.9n}$ sequences. That cannot cover all the 2^n sequences. So entropy(p) is the shortest expected coding length.

Let us compare the following two types of coding. 1) $A = 00$, $B = 01$, $C = 10$, $D = 11$ (a balanced coding) and 2) $A = 1$, $B = 01$, $C = 001$, $D = 000$ (the coding based on the probability). Given a sequence of n alphabets, the first type of coding produces a binary sequence of length $2n$. Considering $p(A) = 1/2$, $p(B) = 1/4$, $p(C) = 1/8$, $p(D) = 1/8$, the second type of coding produces a compressed sequence of length $1.75n$, $1.75n = 0.5n \cdot 1 + 0.25n \cdot 2 + 0.125n \cdot 3 + 0.125n \cdot 3$, which is more compressed than the first binary sequence.

We can also interpret the entropy of $p(x)$ as the average number of coin flipping it amounts to.

Kullback-Leibler divergence and cross entropy

The Kullback-Leibler divergence KL$(p|q)$ is used to measure the dissimilarity between two distributions $p(x)$ and $q(x)$.

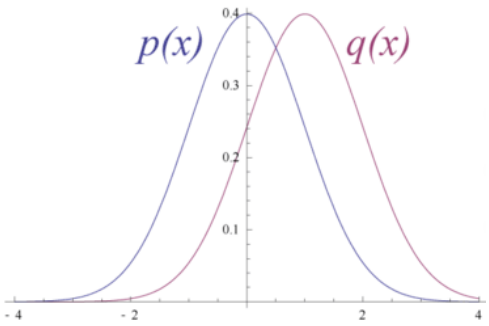


Figure 18: KL$(p|q)$ measures the dissimilarity between the distribution p and the distribution q. `https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence`

If we code the letter $X \sim p(x)$ by a wrong distribution $q(x)$ with code length $-\log_2 q(x)$ instead of $-\log_2 p(x)$, then the coding length is

$$\mathrm{E}_{X\sim p(X)}[-\log q(X)] \overset{\text{def}}{=} \mathrm{E}_p[-\log q(X)] = -\sum_X p(X)\log q(X),$$

and this is called the cross entropy. **The cross entropy is usually used as a loss function to learn a neural network.** We use $\mathrm{E}_p[\cdot]$ to denote $\mathrm{E}_{X\sim p(X)}[\cdot]$ for simplicity.

The redundancy is

$$\mathrm{KL}(p|q) = \mathrm{E}_p[-\log q(X)] - \mathrm{E}_p[-\log p(X)] = \mathrm{E}_p[\log(p(X)/q(X)],$$

which is the Kullback-Leibler divergence from p to q.

Proof of $\mathrm{KL}(p|q) \geq 0$

$$\mathrm{E}_p[q(X)/p(X)] = \sum_x [q(x)/p(x)]p(x) = \sum_x q(x) = 1.$$

According to Jensen inequality, for the concave function $\log(a)$,

$$\mathrm{E}[\log(A)] \leq \log(\mathrm{E}(A)),$$

so

$$\mathrm{E}[\log(q(X)/p(X))] \leq \log \mathrm{E}[q(X)/p(X)] = 0.$$

Thus $\mathrm{KL}(p|q) \geq 0$.

- $\mathrm{KL}(p|q) \geq 0$
- $\mathrm{KL}(p|q) = 0$ if and only if $\forall x, p(x) = q(x)$.
- $\mathrm{KL}(p|q) \neq \mathrm{KL}(q|p)$
- $p(x)$ is usually referred to as the ground-truth distribution, and the estimated distribution q (usually the output of the model) aims to approximate the ground-truth distribution p.
- $\mathrm{KL}(p(y|x)|q(y|x)) \overset{\text{def}}{=} \mathrm{E}_{p(x)}\mathrm{E}_{p(y|x)}\left[\log \frac{p(Y|X)}{q(Y|X)}\right]$.
- $\mathrm{KL}(p(x,y)|q(x,y)) = \mathrm{KL}(p(x)|q(x)) + \mathrm{KL}(p(y|x)|q(y|x))$

1.10 Maximum likelihood

Let $P_{\text{data}}(X,y)$ denote the true distribution of all potentially samples (usually containing infinite samples). n training samples (X_i, y_i) are sampled from $P_{\text{data}}(X,y)$. The discriminative model is trained on $\{(X_i,y_i) \sim P_{\text{data}}(X,y), i = 1,...,n\}$ by maximizing

$$\mathscr{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i|X_i)$$

$$\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i|X_i) \approx \sum_{X\in\Omega} P_{\text{data}}(X)\log p_\theta(y|X) = \mathrm{E}_{P_{\text{data}}}[\log p_\theta(y|X)]$$

according to the law of large number.

Let $h(X)$ be any arbitrary function. In the following, we will assume n is large enough so that

$$\mathrm{E}_{P_{\text{data}}}(h(X)) \approx \sum_X P_{\text{data}}(X)h(X) = \frac{1}{n}\sum_{i=1}^{n} h(X_i)$$

Maximizing log-likelihood is equivalent to

$$\begin{aligned}
\max_\theta \mathscr{L}(\theta) &\equiv \max_\theta \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i|X_i) \\
&\equiv \max_\theta \mathrm{E}_{P_{\text{data}}}[\log p_\theta(y|X)] \\
&\equiv \min_\theta \big\{ -\mathrm{E}_{P_{\text{data}}}[\log p_\theta(y|X)] \big\} \\
&\equiv \min_\theta \mathrm{E}_{P_{\text{data}}}[\log P_{\text{data}}(y|X)] - \mathrm{E}_{P_{\text{data}}}[\log p_\theta(y|X)] \qquad \text{because } \mathrm{E}_{P_{\text{data}}}[\log P_{\text{data}}(y|X)] \text{ is a constant w.r.t. } \theta \\
&\equiv \min_\theta \mathrm{KL}(P_{\text{data}}(y|X)|p_\theta(y|X)).
\end{aligned}$$

where $P_{\text{data}}(y|X)$ denotes the ground-truth conditional probability from the God's perspective, which does not depend on the model θ. Thus, $\mathrm{E}_{P_{\text{data}}}[\log P_{\text{data}}(y|X)]$ is a constant *w.r.t.* θ.

- **For discriminative models, the essence of the maximum likelihood estimation is to minimize the KL divergence between the model $p_\theta(y|X)$ and the ground-truth $P_{\text{data}}(y|X)$.**

The descriptive and generative models learn from $\{X_i \sim P_{\text{data}}(X), i = 1, ..., n\}$ by maximizing the log-likelihood

$$\mathscr{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(X_i) \to \mathrm{E}_{P_{\text{data}}}[\log p_\theta(X)],$$

which is equivalent to minimizing $\mathrm{KL}(P_{\text{data}}|p_\theta)$, because

$$\mathrm{KL}(P_{\text{data}}|p_\theta) = \mathrm{E}_{P_{\text{data}}}[\log P_{\text{data}}(X)] - \mathrm{E}_{P_{\text{data}}}[\log p_\theta(X)].$$

- **For descriptive models, the essence of the maximum likelihood estimation is to minimize the KL divergence between the model $p_\theta(X)$ and the ground-truth $P_{\text{data}}(X)$.**

1.11 Kullback-Leibler of conditionals

Note that for conditional distribution, KL involves expectation with respect to the random variable being conditional on:

$$\mathrm{KL}(p(y|x)|q(y|x)) \stackrel{\text{def}}{=} \mathrm{E}_{p(x,y)}\left[\log \frac{p(Y|X)}{q(Y|X)}\right] = \mathrm{E}_{p(x)}\mathrm{E}_{p(y|x)}\left[\log \frac{p(Y|X)}{q(Y|X)}\right].$$

Let $p(x)$ and $q(x)$ be the marginal distributions, then

$$\begin{aligned}
\mathrm{KL}(p(x,y)|q(x,y)) &= \mathrm{E}_p\left[\log \frac{p(x,y)}{q(x,y)}\right] \\
&= \mathrm{E}_p\left[\log \frac{p(x)p(y|x)}{q(x)q(y|x)}\right] \\
&= \mathrm{E}_p\left[\log \frac{p(x)}{q(x)}\right] + \mathrm{E}_p\left[\log \frac{p(y|x)}{q(y|x)}\right] \\
&= \mathrm{KL}(p(x)|q(x)) + \mathrm{KL}(p(y|x)|q(y|x)).
\end{aligned}$$

1.12 Stochastic gradient descent (SGD)

Mini-batch

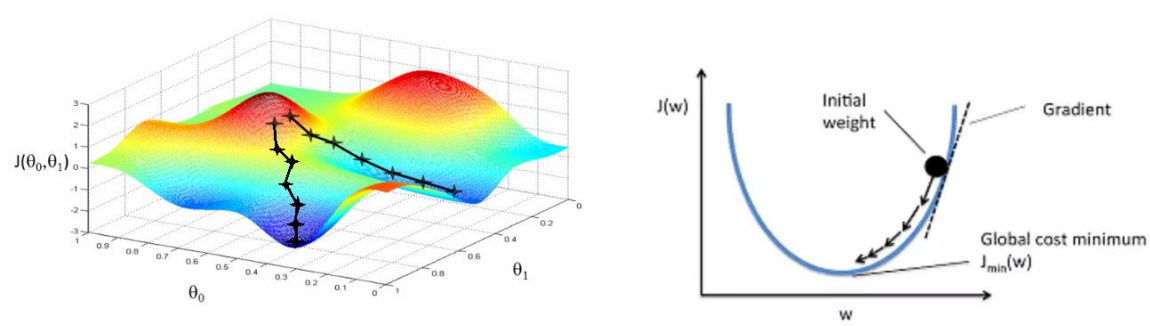


Figure 19: Gradient descent. `https://blog.csdn.net/qq_24133491/article/details/84399195`

Let the training data be (X_i, y_i), $i = 1, ..., n$. Let $L_i(\theta) = L(y_i, X_i; \theta)$ be the loss caused by (X_i, y_i). For regression, $L(y_i, X_i; \theta) = (y_i - f(X_i))^2$, where $f(X_i)$ is parametrized by a model with parameters θ. For classification, $L(y_i, X_i; \theta) = -\log p_\theta(y_i|X_i)$ where $p_\theta(y_i|X_i)$ is modeled by a model with parameters θ. Let $\mathscr{L}(\theta) = \frac{1}{n}\sum_{i=1}^n L_i(\theta)$ be the overall loss averaged over the entire training dataset. The gradient descent is

$$\theta_{t+1} = \theta_t - \eta \mathscr{L}'(\theta_t),$$

where θ_t denotes model parameters at the time step t, η is the step size or learning rate, and $\mathscr{L}'(\theta)$ is the gradient.

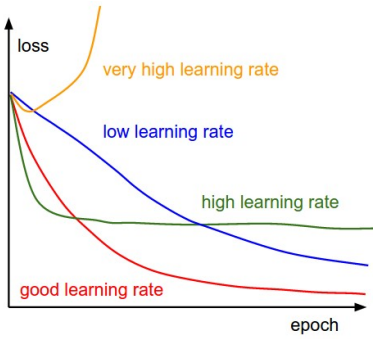


Figure 20: Effects of different learning rates during the learning of deep neural networks. `https://blog.csdn.net/qq_24133491/article/details/84399195`

Gradient descent algorithm: The above gradient descent may be time consuming because we need to compute $\mathscr{L}'(\theta) = \frac{1}{n}\sum_{i=1}^n L_i'(\theta)$ by summing over all the examples. If the number of examples is large, the computation can be time consuming. We may use the following stochastic gradient descent algorithm. At each step, we randomly select i from $\{1, 2, ..., n\}$. Then we update θ by

$$\theta_{t+1} = \theta_t - \eta_t L_i'(\theta_t),$$

where η_t is the step size or learning rate, and $L_i'(\theta_t)$ is the gradient only for the i-th example. Because i is randomly selected, the above algorithm is called the stochastic gradient descent algorithm.

Mini-batch: Instead of randomly selecting a single example, we may randomly select a mini-batch, and replace $L_i'(\theta_t)$ by the average of this mini-batch.

$$\theta_{t+1} = \theta_t - \eta_t \sum_{i \in \text{batch} \subset \text{allSamples}} L_i'(\theta_t)$$

About learning rate: According to the Robbins-Monroe theory for stochastic approximation, we usually need the following conditions to ensure the algorithm to converge to a local minimum. (1) $\sum_{t=1}^{\infty} \eta_t = \infty$. (2) $\sum_{t=1}^{\infty} \eta_t^2 < \infty$. The first condition ensures that the algorithm can go the distance toward the minimum. The second condition ensures that the algorithm will not run away from the local minimum once it arrives. One simple example is $\eta_t = c/t$ for a constant c. In practice, we need more sophisticated scheme for η_t. For instance, reducing η_t after a certain number of steps, or reducing η_t if the training error stops decrease.
for example,

$$\eta_1 = 10^{-3}, \eta_2 = 10^{-3.01}, \eta_3 = 10^{-3.01}, \ldots, \eta_{100} = 10^{-3.99}, \eta_{101} = 10^{-4}, \eta_{102} = 10^{-4}, \eta_{103} = 10^{-4}, \ldots$$

Momentum, Adagrad, RMSprop, Adam

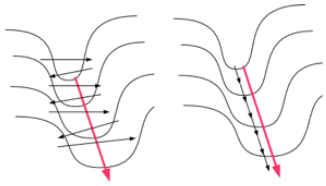


Figure 21: Momentum. The left figure illustrates the original gradient descent algorithm. The black arrow is the gradient direction, and the red arrow is the preferred direction. The right figure illustrates the gradient descent algorithm with momentum, which is along the red arrow.

The gradient descent algorithm goes downhills in the steepest direction in each step. However, the steepest direction may not be the best direction, as illustrated by Figure 21. In the left figure, the black arrows are the gradient direction. The red arrows are the preferred direction, which is the direction of momentum. It is better to move along the direction of the momentum, as illustrated by the right figure. We want to accumulate the momentum, and let it guide the descent. The following is the stochastic gradient descent with momentum[4, 5]:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta_t g_t, \\ \theta_t &= \theta_{t-1} - v_t. \end{aligned}$$

where g_t is the average gradient computed from the current mini-batch, and v_t is the momentum or velocity. γ is usually set at .9, for accumulating the momentum, and θ is updated based on the momentum.

Adagrad modifies the gradient descent algorithm in another direction. The magnitudes of the components of g_t may be very uneven, and we need to be adaptive to that. The Adagrad[1] let

$$\begin{aligned} G_t &= G_{t-1} + g_t^2, \\ \theta_{t+1} &= \theta_t - \eta_t \frac{g_t}{\sqrt{G_t + \xi}}, \end{aligned}$$

where ξ is a small number (*e.g.* 10^{-4}) to avoid dividing by 0. In the above formula, g_t^2 and $g_t/\sqrt{G_t+\xi}$ denote component-wise square and division.

In Adagrad, G_t is the sum over all the time steps. It is better to sum over the recent time steps. Adadelta and RMSprop[6] use the following scheme:

$$G_t = \beta G_{t-1} + (1-\beta) g_t^2,$$

where β can be set at .9. It can be shown that

$$G_t = (1-\beta)(\beta^{t-1} g_1^2 + \beta^{t-2} g_2^2 + ... + \beta g_{t-1}^2 + g_t^2),$$

which is a sum over time with decaying weights.

The Adam[3] optimizer combines the idea of RMSprop and the idea of momentum.

$$\begin{aligned}
&v_t = \gamma v_{t-1} + (1-\gamma) g_t, \\
&G_t = \beta G_{t-1} + (1-\beta) g_t^2, \\
&v_t \leftarrow v_t/(1-\gamma), G_t \leftarrow G_t/(1-\beta), \\
&\theta_{t+1} = \theta_t - \eta_t \frac{v_t}{\sqrt{G_t+\xi}}.
\end{aligned}$$

1.13 Gradient of log-likelihood

In deep learning, a discriminative neural network with the softmax output layer is

$$p_\theta(y=k|X) = p_k = \frac{1}{Z(\theta)} \exp(f_\theta^{(k)}(X)),$$

where $f_\theta^{(k)}(X)$ is a K-dimensional network output with respect to the input X before the softmax layer. $f_\theta^{(k)}(X)$ indicates the k-th output dimension.

for $k = 1, ..., K$, and

$$Z(\theta) = \sum_k \exp(f_\theta^{(k)}(X)).$$

• For discriminative model, if $y = k$ (when the ground-truth y is k), then

$$\begin{aligned}
\frac{\partial}{\partial\theta} \log p_\theta(y|X) &= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \frac{\partial}{\partial\theta} \log Z(\theta) \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \frac{1}{Z(\theta)} \frac{\partial}{\partial\theta} Z(\theta) \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \frac{1}{Z(\theta)} \frac{\partial}{\partial\theta} \left[\sum_{k'} \exp(f_\theta^{(k')}(X)) \right] \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \left[\sum_{k'} \frac{\exp(f_\theta^{(k')}(X))}{Z(\theta)} \frac{\partial}{\partial\theta} f_\theta^{(k')}(X) \right] \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \left[\sum_{k'} p_{k'} \frac{\partial}{\partial\theta} f_\theta^{(k')}(X) \right] \\
&= \sum_{k'} (1(y=k') - p_{k'}) \frac{\partial}{\partial\theta} f_\theta^{(k')}(X) \\
&= \frac{\partial}{\partial\theta} f_\theta(X)^\top (Y - p) \qquad \text{Learn from errors} \\
&= \frac{\partial}{\partial\theta} f_\theta(X)^\top (Y - \mathrm{E}_\theta(Y|X)),
\end{aligned}$$

where we define Y to be the one-hot version of y,

$$Y = [Y_1, Y_2, \ldots, Y_n], \qquad Y_{k'} = \begin{cases} 1, & k' = k \\ 0, & k' \neq k \end{cases}$$

Define $p = (p_{k'}, k' = 1, ..., K)^\top$. Then $p = \mathrm{E}_\theta(Y|X)$. $\log p_\theta(y|X) = Y^\top \log p$. Define $f = (f_{k'}, k' = 1, ..., K)^\top$. For an observation (X_i, y_i),

$$\begin{aligned} \frac{\partial}{\partial \theta} \log p_\theta(y_i|X_i) &= \sum_{k'} (1(y_i = k') - p_{k'}) \frac{\partial}{\partial \theta} f_\theta^{(k')}(X_i) \\ &= \frac{\partial}{\partial \theta} f_\theta(X_i)^\top (Y_i - p_i) \\ &= \frac{\partial}{\partial \theta} f_\theta(X_i)^\top (Y_i - \mathrm{E}_\theta(Y_i|X_i)), \end{aligned}$$

• The descriptive model aims to estimate the data distribution $p_\theta(X)$, which can be given as

$$p_\theta(X) = \frac{1}{Z(\theta)} \exp(f_\theta(X)),$$

where

$$Z(\theta) = \int \exp(f_\theta(x)) dx.$$

We get

$$\begin{aligned} \frac{\partial}{\partial \theta} \log p_\theta(X) &= \frac{\partial}{\partial \theta} f_\theta(X) - \frac{\partial}{\partial \theta} \log Z(\theta) \\ &= \frac{\partial}{\partial \theta} f_\theta(X) - \frac{1}{Z(\theta)} \frac{\partial}{\partial \theta} Z(\theta) \\ &= \frac{\partial}{\partial \theta} f_\theta(X) - \frac{1}{Z(\theta)} \int \exp(f_\theta(X)) \frac{\partial}{\partial \theta} f_\theta(X) dx \\ &= \frac{\partial}{\partial \theta} f_\theta(X) - \int \frac{1}{Z(\theta)} \exp(f_\theta(X)) \frac{\partial}{\partial \theta} f_\theta(X) dx \\ &= \frac{\partial}{\partial \theta} f_\theta(X) - \sum_X p_\theta(X) \frac{\partial}{\partial \theta} f_\theta(X) \\ &= \frac{\partial}{\partial \theta} f_\theta(X) - \mathrm{E}_\theta \left[\frac{\partial}{\partial \theta} f_\theta(X) \right]. \end{aligned}$$

For an observation X_i,

$$\frac{\partial}{\partial \theta} \log p_\theta(X_i) = \frac{\partial}{\partial \theta} f_\theta(X_i) - \mathrm{E}_\theta \left[\frac{\partial}{\partial \theta} f_\theta(X) \right].$$

- For generative model $p_\theta(h,X)$,

$$\begin{aligned}
\frac{\partial}{\partial\theta}\log p_\theta(X) &= \frac{1}{p_\theta(X)}\frac{\partial}{\partial\theta}\int p_\theta(h,X)dh \\
&= \frac{1}{p_\theta(X)}\int\left[\frac{\partial}{\partial\theta}p_\theta(h,X)\right]dh \\
&= \frac{1}{p_\theta(X)}\int\left[\frac{\partial}{\partial\theta}\log p_\theta(h,X)\right]p_\theta(h,X)dh \\
&= \int\left[\frac{\partial}{\partial\theta}\log p_\theta(h,X)\right]\frac{p_\theta(h,X)}{p_\theta(X)}dh \\
&= \int\left[\frac{\partial}{\partial\theta}\log p_\theta(h,X)\right]p_\theta(h|X)dh \\
&= \mathrm{E}_{p_\theta(h|X)}\left[\frac{\partial}{\partial\theta}\log p_\theta(h,X)\right]
\end{aligned}$$

For an observation X_i,

$$\frac{\partial}{\partial\theta}\log p_\theta(X_i) = \mathrm{E}_{p_\theta(h_i|X_i)}\left[\frac{\partial}{\partial\theta}\log p_\theta(h_i,X_i)\right]$$

$$p_\theta(h_i,X_i) = p(h_i)p_\theta(X_i|h_i), \quad h_i \sim \mathrm{N}(\mathbf{0},\mathbf{I}), \quad X_i = g_\theta(h_i)+\varepsilon, \quad \varepsilon \sim \mathrm{N}(\mathbf{0},\sigma^2\mathbf{I}).$$

Thus,

$$\begin{aligned}
p_\theta(h_i,X_i) =& \mathrm{N}(h_i|\mu=\mathbf{0},\Sigma=\mathbf{I})\mathrm{N}(\varepsilon|\mu=\mathbf{0},\Sigma=\sigma^2\mathbf{I}) \\
=& \mathrm{N}(h_i|\mu=\mathbf{0},\Sigma=\mathbf{I})\mathrm{N}(X_i|h_i|\mu=g_\theta(h_i),\Sigma=\sigma^2\mathbf{I})
\end{aligned}$$

$$\log p_\theta(h_i,X_i) = -\frac{1}{2\sigma^2}\|X_i-g_\theta(h_i)\|^2 - \frac{1}{2}\|h_i\|^2 + \text{constant}$$

which is analytically tractable.

For a mini-batch, we average the above gradient.

Optimizing logistic regression via gradient ascent

The likelihood function is

$$L(\beta) = \prod_{i=1}^n p_i^{y_i}(1-p_i)^{1-y_i} = \prod_{i=1}^n \frac{e^{y_i X_i^\top \beta}}{1+e^{X_i^\top\beta}}.$$

The log-likelihood is

$$l(\beta) = \log L(\beta) = \sum_{i=1}^n \left[y_i X_i^\top\beta - \log(1+\exp X_i^\top\beta)\right].$$

The maximum likelihood is to find the most plausible explanation to the observed data.

To find the maximum of $l(\beta)$, we first calculate the gradient

$$l'(\beta) = \sum_{i=1}^n\left[y_iX_i - \frac{e^{X_i^\top\beta}}{1+e^{X_i^\top\beta}}X_i\right] = \sum_{i=1}^n(y_i-p_i)X_i. \qquad \text{Learn from mistakes}$$

where

$$p_i = \frac{e^{X_i^\top \beta}}{1+e^{X_i^\top \beta}} = \frac{1}{1+e^{-X_i^\top \beta}}$$

We use gradient ascent to iteratively update β and maximize the log-likelihood,

$$\beta^{(t+1)} = \beta^{(t)} + \gamma_t \sum_{i=1}^{n} (y_i - p_i) X_i,$$

where γ is the learning rate. This is a hill climbing algorithm, where each step we take the steepest direction uphill. If we minimize a loss function such as $-l(\beta)$, then we use the gradient descent algorithm, which means each step we take the steep direction downhill.

The algorithm learns from mistakes $y_i - p_i$ by trial and error. If a mistake is made such that p_i is very different from y_i, then β accumulates X_i if $y_i - p_i$ is positive, and $-X_i$ if $y_i - p_i$ is negative.

1.14 Langevin

Brownian motion, $\sqrt{\Delta t}$ notation, second order Taylor expansion

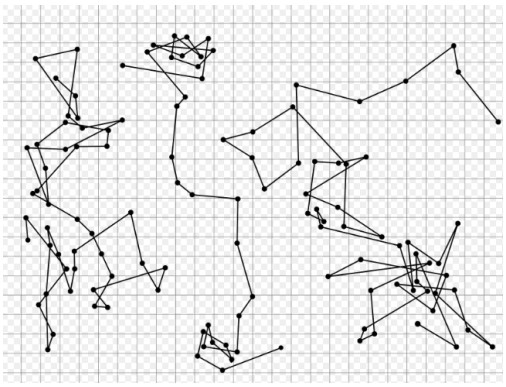


Figure 22: Brownian motion. `https://en.wikipedia.org/wiki/Brownian_motion`

In this section, we need to consider X_t as the model parameter β that is learned after t epochs, rather than an image or an input sample. We can consider U as the loss function of a task, and $U'(X_t)$ denotes $\frac{\partial U}{\partial X}\Big|_{X=X_t}$.

The Brownian motion is defined by

$$X_{t+\Delta t} = X_t + \sigma \varepsilon_t \sqrt{\Delta t},$$

where $\mathrm{E}(\varepsilon_t) = 0$ and $\mathrm{Var}(\varepsilon_t) = 1$. Let $X_0 = x$. If we divide the interval $[0,t]$ into n periods, so that $\Delta t = t/n$. Write ε_t within each period as ε_i for $i = 1, ..., n$. Then

$$X_t = x + \sum_{i=1}^{n} \sigma \varepsilon_i \sqrt{\frac{t}{n}} = x + \sigma\sqrt{t}\frac{1}{\sqrt{n}} \sum_{i=1}^{n} \varepsilon_i \to \mathrm{N}(x, \sigma^2 t \mathbf{I})$$

according to the central limit theorem. The above equation also shows why we need $\sqrt{\Delta t}$ scaling in order to make the $\mathrm{Var}(X_t)$ to be independent of n or Δt. This $\sqrt{\Delta}$ scaling was first noticed by Einstein in his 1905 analysis of Brownian motion. Einstein explained that the velocity $dX(t)/dt = (X_{t+\Delta t} - X_t)/\Delta t = \sigma \varepsilon_t / \sqrt{\Delta t} \to \infty$.

The $\sqrt{\Delta t}$ notation also makes it clear that we should use second order Taylor expansion in our analysis. The second order Taylor expansion underlies much of the calculations in Brownian motion, including the Ito calculus.

Langevin: energy and entropy

For the descriptive model, we can approximate it using a tractable model q by minimizing the free energy $F(q)$. The Langevin dynamics

$$X_{t+\Delta t} = X_t - \frac{1}{2}U'(X_t)\Delta t + \sqrt{\Delta t}\varepsilon_t,$$

is consistent with minimizing $F(q)$, where the gradient descent $X_t - U'(X_t)\Delta t/2$ decreases the energy, and the Brownian motion $\sqrt{\Delta t}\varepsilon_t$ increases the entropy. The Langevin dynamics decreases the KL-divergence between the distribution of X_t and p_θ monotonically.

U' can be considered as the loss function of a discriminative model.

For the generative model, we can sample h_i from $p_\theta(h_i|X_i)$ by Langevin dynamics

$$h_{t+\Delta t} = h_t + \frac{1}{2}\frac{\partial}{\partial h}\log p_\theta(h, X_i)\Delta t + \sqrt{\Delta t}\varepsilon_t,$$

where $-\log p_\theta(h, X_i)$ plays the role of energy. The Langevin dynamics decreases the KL-divergence between the distribution of h_t and $p_\theta(h|X_i)$ monotonically. Here we drop the subscript i in h_i for convenience.

1.15 Non-linear and non-parametric functions

We may replace the linear function $X_i^\top \beta$ by a more flexible non-linear function $f(X_i)$. The function $f(X)$ (here we use X to denote a generic X_i) is sometimes called non-parametric, but it actually parametrized by many parameters. In what follows, we shall study the more flexible $f(X)$ parametrized by models.

1.16 Kernel Regression

Suppose we have training examples (y_i, x_i), $i = 1, ..., n$. Consider the Gaussian kernel $K(x, x') = e^{-\gamma\|x-x'\|^2}$. Suppose we want to learn a regression curve of the form

$$f(x) = \sum_{i=1}^{n} c_i K(x, x_i)$$

by minimizing

$$\sum_{i=1}^{n} ||y_i - \sum_{j=1}^{n} c_j K(x_i, x_j)||^2 + \lambda \sum_{i,j} c_i c_j K(x_i, x_j).$$

Let kernel K be an $n \times n$ matrix with $K_{ij} = K(x_i, x_j)$, then the objective function in matrix notation is

$$\ell(c) = \sum_{i=1}^{n} \|y_i - \sum_{j=1}^{n} c_j K_{ij}\|^2 + \lambda \sum_{i,j} c_i c_j K_{ij} = \|Y - Kc\|^2 + \lambda c^\top K c.$$

We want to find $\hat{c}_\lambda$ with minimal loss

$$\hat{c}_\lambda = \arg\min_c \ell(c).$$

By taking the derivative of the loss function w.r.t. c, we get the first order condition

$$\frac{\partial \ell(c)}{\partial c}\,|_{c=\hat{c}_\lambda} = -2K(Y - K\hat{c}_\lambda) + 2\lambda K\hat{c}_\lambda = 0.$$

Note $K = K^T$ by definition of $K(\cdot,\cdot)$. Hence, the estimate of c is

$$\hat{c}_\lambda = (K^2 + \lambda K)^{-1} KY = (K + \lambda I_n)^{-1} Y.$$

1.17 Linear Discriminant Analysis (LDA)

LDA for two classes

credit: `https://blog.csdn.net/u012303532/article/details/42973897`

Let us consider two classes $y = +1$ and $y = -1$. $\Omega \stackrel{\text{def}}{=} \{X_i, y_i\}_{i=1,\ldots,n}$ denotes the training set. Ω^+ and Ω^- denote the set of positive training samples and the set of negative training samples. $\Omega = \Omega^+ \cup \Omega^-$. In the scenario of linear discriminant analysis, we assume that

- $\forall X_i \in \Omega^+,\ p(X_i|y=+1) \sim \mathrm{N}(\mu^+, \Sigma^+)$
- $\forall X_i \in \Omega^-,\ p(X_i|y=-1) \sim \mathrm{N}(\mu^-, \Sigma^-)$

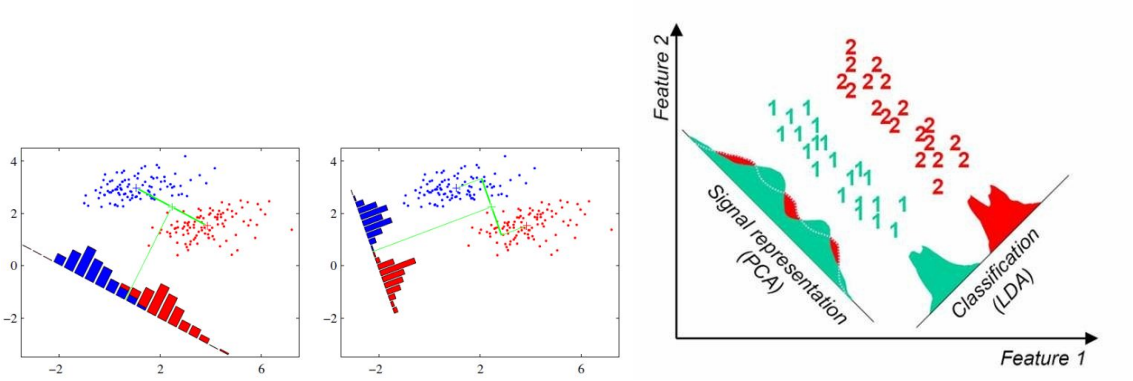


Figure 23: LDA

The task is to learn a linear function that projects X_i to $z = X_i^\top \beta$, so that the projection maximizes the inter-class variance and minimizes the intra-class variance.

The inter-class variance is given as

$$\begin{aligned}
\sigma^2_{\text{between}} &= (E_{i\in\Omega^+}[X_i^\top\beta] - E_{i\in\Omega^-}[X_i^\top\beta])^2 \\
&= (E_{i\in\Omega^+}[X_i^\top]\beta - E_{i\in\Omega^-}[X_i^\top]\beta)^2 \\
&= ((\mu^+)^\top\beta - (\mu^-)^\top\beta)^2 \\
&= [(\mu^+ - \mu^-)^\top\beta]^2 \\
\sigma^2_{\text{within}} &= n_{\text{pos}}\sigma^2_{\text{pos}} + n_{\text{neg}}\sigma^2_{\text{neg}}, \qquad n_{\text{pos}} = |\Omega^+|,\ n_{\text{neg}} = |\Omega^-| \\
\sigma^2_{\text{pos}} &= E_{i\in\Omega^+}\left[(X_i^\top\beta - E_{i'\in\Omega^+}[X_{i'}^\top\beta])^2\right] \\
&= E_{i\in\Omega^+}\left[(\beta^\top X_i - E_{i'\in\Omega^+}[\beta^\top X_{i'}])(X_i^\top\beta - E_{i'\in\Omega^+}[X_{i'}^\top\beta])\right] \\
&= E_{i\in\Omega^+}\left[\beta^\top(X_i - E_{i'\in\Omega^+}[X_{i'}])(X_i^\top - E_{i'\in\Omega^+}[X_{i'}^\top])\beta\right] \\
&= \beta^\top E_{i\in\Omega^+}\left[(X_i - E_{i'\in\Omega^+}[X_{i'}])(X_i^\top - E_{i'\in\Omega^+}[X_{i'}^\top])\right]\beta \\
&= \beta^\top\Sigma^+\beta \\
\sigma^2_{\text{neg}} &= \beta^\top\Sigma^-\beta
\end{aligned}$$

Thus, the objective is

$$S = \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}}, \qquad \max_{\beta} S$$

$$\begin{aligned} S &= \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}} \\ &= \frac{[(\mu^+ - \mu^-)^\top \beta]^2}{n_{\text{pos}} \beta^\top \Sigma^+ \beta + n_{\text{neg}} \beta^\top \Sigma^- \beta} \\ &= \frac{(\beta^\top (\mu^+ - \mu^-))((\mu^+ - \mu^-)^\top \beta)}{\beta^\top (n_{\text{pos}} \Sigma^+ + n_{\text{neg}} \Sigma^-) \beta} \\ &= \frac{\beta^\top S_B \beta}{\beta^\top S_W \beta} \end{aligned}$$

where

$$\begin{aligned} S_B &= (\mu^+ - \mu^-)(\mu^+ - \mu^-)^\top \\ S_W &= n_{\text{pos}} \Sigma^+ + n_{\text{neg}} \Sigma^- \end{aligned}$$

Considering β may have different scales, *i.e.* $\beta' = 2\beta$ may produce the score of S. Thus, we require

$$\beta^\top S_W \beta = 1$$

and then the objective is modified as

$$\max_{\beta} \beta^\top S_B \beta, \quad \text{s.t.} \quad \beta^\top S_W \beta = 1.$$

We can use Lagrange multipliers for this constrained maximization.

$$\begin{aligned} & L = \beta^\top S_B \beta - \lambda (\beta^\top S_W \beta - 1) \\ \Rightarrow \quad & \frac{\partial L}{\partial \beta} = 2 S_B \beta - 2 \lambda S_W \beta = 0 \\ \Rightarrow \quad & S_B \beta = \lambda S_W \beta \\ \Rightarrow \quad & S_W^{-1} S_B \beta = \lambda \beta \end{aligned}$$

Thus, β is the eigenvector of the matrix $S_W^{-1} S_B$.

Because the orientation of $S_B \beta$ is always $\mu^+ - \mu^-$, the result is

$$\beta \propto S_W^{-1} (\mu^+ - \mu^-)$$

References

[1] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[2] P. J. Huber. Robust estimation of a location parameter. *The annals of mathematical statistics*, pages 73–101, 1964.

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[5] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[6] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.