Machine Learning Explained

Lecture Notes of CS385, SJTU, taught by Prof. Quanshi Zhang

Summarized and Written by Tony Fang (galaxies@sjtu.edu.cn, tony.fang.galaxies@gmail.com)

Chapter 1. Linear Model

1.1 Linear Regression

Suppose that a regression problem receives p inputs and produces one output. The dataset of the regression problem consists n input-output instances of the regression problem, and the i-th instance shows that with input $(x_{i,1}, x_{i,2}, \cdots, x_{i,p})$, output y_i is observed. We add an extra "1" into the input for convenience (notice it is not the real input, but the "imaginable" input used for convenience), so the i-th input becomes a $(p+1) \times 1$ vector $X_i = (1, x_{i,1}, x_{i,2}, \cdots, x_{i,p})^T$. Hence, we can write all inputs of all instances into a $n \times (p+1)$ matrix $\boldsymbol{X}^T = (X_1^T, X_2^T, \cdots, X_n^T)^T$. Similarly, we have an output $n \times 1$ vector $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)^T$. Hence ***the linear regression model*** is of the following form.

$$\boldsymbol{y} = \boldsymbol{X}^T\beta + \varepsilon$$

where $\beta = (\beta_0, \beta_1, \cdots, \beta_p)^T$ is a $(p+1) \times 1$ coefficient vector, and $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)^T$ is a $n \times 1$ residual vector. WE assume that the **error/residual/disturbance term** $\varepsilon_i \sim N(0, \sigma^2)$ independently.

The process of estimating β from $\boldsymbol{X}, \boldsymbol{y}$ are called learning from training data, and the purpose is twofold.

(1) Explanation: understanding the relationship between output y and its corresponding inputs $x_1, \cdots, x_p$;

(2) Prediction: learning to predict output y using inputs $x_1, \cdots, x_p$ in testing stage.

In the prediction stage, we predict the output $\hat{y}$ using the estimation $\hat{y} = X^T\beta$, where X is the given input.

1.2 Logistic Regression

Given training instances $\boldsymbol{X}, \boldsymbol{y}^*$ that are similar to linear regression, except that $y_i \in \{0,1\}$ are the boolean labels. Suppose that $y_i \mid X_i \sim B(p_i)$, *i.e.*, $\Pr(y_i = 1 \mid X_i) = p_i$ and $\Pr(y_i = 0 \mid X_i) = 1 - p_i$. Then we assume $\text{logit}(p_i)$ can be estimated using linear regression, that is,

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

Therefore,

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

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

Similarly, let $\{(X_i, y_i^*)\}_{i=1,2,\cdots,n}$ be the training instances, where y_i^* is the ground-truth label. The probability for correct estimation is

$$\Pr(y_i = y_i^* \mid X_i) = p_i^{y_i^*}(1-p_i)^{1-y_i^*} = \left(\frac{e^{X_i^T\beta}}{1+e^{X_i^T\beta}}\right)^{y_i^*}\left(1-\frac{e^{X_i^T\beta}}{1+e^{X_i^T\beta}}\right)^{1-y_i^*} = \frac{e^{y_i^*X_i^T\beta}}{1+e^{X_i^T\beta}}$$

Notice that we can also use $\Pr(y_i = y_i^* \mid X_i) = p_i y_i^* + (1-p_i)(1-y_i^*)$, but it is inconvenient in the following calculations and optimizations.

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

$$\Pr(\beta) = \prod_{i=1}^{n}\Pr(y_i = y_i^* \mid X_i) = \prod_{i=1}^{n}\frac{e^{y_i^*X_i^T\beta}}{1+e^{X_i^T\beta}}$$

The corresponding log-likelihood is

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

The maximum (log-)likelihood is to find the most possible explanation to the observed data.

$$\hat{\beta} = \arg\max_{\beta}\Pr(\beta) = \arg\max_{\beta}\log\Pr(\beta)$$

Classifications. In context of classifications, we usually use $y_i^* = 1$ or $y_i^* = -1$ to represent positive and negative sample respectively, instead of using $y_i^* \in \{0,1\}$. Hence, $\Pr(y_i = -1 \mid X_i) = 1 - p_i$. Therefore,

$$\Pr(y_i \mid X_i) = \frac{1}{1 + e^{-y_i X_i^T \beta}}$$

Here, β may be called a classifier, since $\langle X_i, \beta \rangle = X_i^T \beta$ is the projection of X_i on β, so vector β is the direction that reveals the difference between positive and negative samples. Thus, β should be aligned with the positives and negatively aligned with the negatives, *i.e.*, β should point from negatives to positives.

1.3 Perceptron

A deterministic version of the logistic regression is the perceptron model

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

where $\text{sign}(x) = \begin{cases} +1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$ is the sign function.

The notation of the perceptron model in machine learning literature is usually written as

$$y_i = \text{sign}(X_i^T w + b)$$

where $w = \left(w_j \mid j = 1,2,\cdots,p \right)^T$ are the connection weights, and b is the bias term. (w, b) corresponds to β.

1.4 Loss Functions

In order to learn β from the training data $\{(X_i, y_i^*)\}_{i=1,2,\cdots,n}$, we can minimize the loss function. In this section, we will use y_i^* to denote the ground-truth and use y_i to denote the predicted value. In linear models, $y_i = X_i^T \beta$ holds where β is the parameter of the model.

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} L(y_i^*, y_i) = \sum_{i=1}^{n} L(y_i^*, X_i^T \beta)$$

where $L(y_i^*, y_i)$ is the loss for each training sample (X_i, y_i^*), and $y_i = X_i^T \beta$ is the prediction of our model. We are going to define different $L(\cdot)$'s next.

Square Loss.

For linear regression, the square loss is usually used.

$$L(y_i^*, y_i) = (y_i^* - y_i)^2 = (y_i^* - X_i^T \beta)^2$$

Why? Because the least square loss can be derived from the maximum likelihood estimation if we assume that the errors follow a standard normal distribution $N(0,1)$.

$$y_i^* \mid X_i = X_i^T \beta + \varepsilon \sim N(X_i^T \beta, 1) \implies \log \Pr(y_i^* \mid X_i) = -\frac{(y_i^* - X_i^T \beta)^2}{2} + C$$

where C is a constant. Therefore,

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} L(y_i^*, y_i) = C - \sum_{i=1}^{n} -\frac{(y_i^* - X_i^T \beta)^2}{2} = \sum_{i=1}^{n} \frac{(y_i^* - X_i^T \beta)^2}{2}$$

Here, C can be omitted since it is a constant. Our loss function here is defined as negated log-likelihood, since we want to maximize log-likelihood and minimize loss function.

Mean Absolute Loss.

$$L(y_i^*, y_i) = |y_i^* - y_i| = |y_i^* - X^T \beta|$$

The mean absolute loss penalizes big difference between y_i^* and y_i to a less degree than the least square loss, so that the estimated β is less affected by the outliers.

Huber loss[1].

$$L(y_i^*, y_i) = \begin{cases} \frac{1}{2}(y_i^* - y_i)^2 = \frac{1}{2}(y_i^* - X^T \beta)^2 & |y_i^* - y_i| \le \delta \\ \delta |y_i^* - y_i| - \frac{\delta^2}{2} = \delta |y_i^* - X^T \beta| - \frac{\delta^2}{2} & |y_i^* - y_i| > \delta \end{cases}$$

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

---

[1] Huber, Peter J. "Robust Estimation of a Location Parameter." *The Annals of Mathematical Statistics* (1964): 73-101.

Logistic Regression Loss with 0/1 Responses.

For logistic regression, we usually minimize the negated (log-)likelihood function. Previously, we have derived the form of log-likelihood function of logistic regression with 0/1 response. Therefore,

$$L(y_i^*, y_i) = -y_i^* y_i + \log(1 + \exp y_i) = -[y_i^* X_i^T \beta - \log(1 + \exp X_i^T \beta)]$$

Logistic Loss (Logistic Regression Loss with +/- Responses).

Similarly, for +/- responses,

$$L(y_i^*, y_i) = \log(1 + exp(-y_i^* y_i)) = \log(1 + \exp(-y_i^* X_i^T \beta))$$

which is called logistic loss.

Classification Loss Function.

1. Logistic Loss: introduced before, used by logistic regression;
2. Exponential Loss: $L(y_i^*, y_i) = \exp(-y_i^* y_i) = \exp(-y_i^* X_i^T \beta)$, used by adaboost;
3. Hinge Loss: $L(y_i^*, y_i) = \max(0,1 - y_i^* y_i) = \max(0,1 - y_i^* X_i^T \beta)$, used by support vector machine;
4. Zero-one Loss: $L(y_i^*, y_i) = [y_i^* y_i < 0] = [y_i^* X_i^T \beta < 0]$, where $[\cdot]$ is the predicate function, only used to count the number of mistakes and not for training since it is not differentiable.

The previous loss are based on $m_i = y_i^* y_i = y_i^* X_i^T \beta$, which is called **margin**, the larger the margin is, the more confident we are with regard to the classification result. Margin will be further discussed in Chapter 2.

1.5 Least Squares

Consider a linear regression model $Y = X\beta + \varepsilon$, where X is an $n \times p$ matrix with $n > p$, Y is an $n \times 1$ vector of ground truth, and $\varepsilon \sim N(0, \sigma^2 I_n)$. Notice that for convention reasons, here we use X to represent the original X^T in the previous section, where we write the model inference as $X^T\beta$ instead of $X\beta$.

By using the square loss, we want to find $\hat{\beta}$ with minimal loss.

$$\hat{\beta} = \arg\min_{\beta} \mathcal{L}(\beta) = \arg\min_{\beta} \|Y - X\beta\|^2$$

The first order condition of the optimization problem is

$$\frac{\partial \mathcal{L}(\beta)}{\partial \beta} = \frac{\partial}{\partial \beta}(Y - X\beta)^T(Y - X\beta) = \frac{\partial}{\partial \beta}[-\beta^T X^T Y - Y^T X\beta + \beta^T X^T X\beta] = 0$$

Since $\beta^T XY$ is an 1×1 matrix, then $\beta^T X^T Y = (\beta^T X^T Y)^T = Y^T X\beta$; since $X^T X$ is symmetric matrix, $\frac{\partial}{\partial \beta}\beta^T X^T X\beta = 2X^T X\beta$. Therefore,

$$2X^T Y = 2X^T X\beta \implies \hat{\beta} = (X^T X)^{-1} X^T Y$$

We use the estimator to reconstruct Y as follows.

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

In essence, Y can be seen as the sum of its projection onto the space spanned by columns of X, plus a vector of residuals $\varepsilon = Y - \hat{Y}$, which is orthogonal to that space. These two parts also represent the signal that can be encoded by the current model, and the signal that cannot be encoded by the current model.

Furthermore, we discuss the distribution of random variable $\hat{\beta}$. Suppose the true value of β is β^*. Therefore,

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

Since $\varepsilon \sim N(0, \sigma^2 I_n)$, hence $E[\hat{\beta}] = \beta^*$, which means $\hat{\beta}$ is an unbiased estimator. Moreover, we can prove that $\hat{\beta}$ is also a consistent estimator and an asymptotic efficient estimator.

1.6 Entropy, Cross Entropy and Kullback-Leibler Divergence

Entropy in physics measure the degree of chaos in a certain system. But here the definition of "entropy" has been brought to the fields of information theory. Define the entropy of a distribution $p(x)$ as

$$Entropy(p) = E_p[-\log p(X)] = \sum_x p(x)[-\log p(x)]$$

Shannon's Source Coding Theorem states that the entropy represents the optimal coding length of a variable X with distribution $p(X)$. The cross entropy represents the coding length if we encode $X \sim p(x)$ by a wrong distribution $q(x)$ with code length $-\log q(x)$ instead of optimal $-\log p(x)$, that is,

$$CrossEntropy(p, q) = E_{p(X)}[-\log q(X)] = \sum_x p(X)[-\log q(X)]$$

The cross entropy is usually used as a loss function to learn a neural network.

The Kullback-Leibler divergence $KL(p \mid q)$ is used to measure the dissimilarity between two distributions $p(x)$ and $q(x)$, *i.e.*, the redundancy between the wrong coding length and the optimal coding length, that is,

$$KL(p|q) = E_{p(X)}[-\log q(X)] - E_{p(X)}[-\log p(X)] = E_{p(X)}\left[\log \frac{P(X)}{q(X)}\right] = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

According to *Shannon's Source Coding Theorem*, $KL(p \mid q) \geq 0$. Here is another interesting proof.

Proof of the non-negative property of KL divergence.

$$E_{p(X)}\left[\frac{q(X)}{p(X)}\right] = \sum_x \frac{q(x)}{p(x)} \cdot p(x) = \sum_x q(x) = 1$$

According to Jensen's Inequality, for the concave function $\log(\cdot)$, $E\left[\log \frac{q(X)}{p(X)}\right] \leq \log E\left[\frac{q(X)}{p(X)}\right] = 0$.

Q.E.D.

Other properties of KL divergence are sometimes useful, listed as follows.

- $KL(p \mid q) = 0$ iff $\forall x, p(x) = q(x)$;
- Asymmetricity: $KL(p \mid q) \neq KL(q \mid p)$;
- $p(x)$ 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.

The conditional KL divergence is specially defined as follows.

$$KL\big(p(y \mid x) \mid q(y \mid x)\big) = E_{p(X,Y)}\left[\log \frac{p(y \mid x)}{q(y \mid x)}\right] = E_{p(X)}E_{p(Y|X)}\left[\log \frac{p(y \mid x)}{q(y \mid x)}\right]$$

According to the definition of conditional KL divergence, another useful property is that

$$KL\big(p(x,y) \mid q(x,y)\big) = KL\big(p(x) \mid q(x)\big) + KL\big(p(y \mid x) \mid q(y \mid x)\big)$$

Proof of the property of KL divergence.

$$\begin{aligned} KL\big(p(x,y) \mid q(x,y)\big) &= E_{p(x,y)}\left[\log \frac{p(x,y)}{q(x,y)}\right] = E_{p(x,y)}\left[\log \frac{p(x)p(y \mid x)}{q(x)q(y \mid x)}\right] \\ &= E_{p(x,y)}\left[\log \frac{p(x)}{q(x)}\right] + E_{p(x,y)}\left[\log \frac{p(y \mid x)}{q(y \mid x)}\right] \\ &= KL\big(p(x) \mid q(x)\big) + KL\big(p(y \mid x) \mid q(y \mid x)\big) \end{aligned}$$

Notice that the first term doesn't contain y, therefore we can omit y in $p(x,y)$. (More precisely, we can integrate y and then $\int p(x,y)\mathrm{d}y = p(x)$).

Q.E.D.

1.7 Generative, Discriminative and Descriptive Models[2]

A father has three kids, kid A, kid B and kid C. Kid A has a special character whereas he can learn everything in depth. Kid B has a special character whereas he can only learn the differences between what he saw. Kid C has a special character whereas he can learn to describe everything in some certain aspects.

One fine day, The father takes three kids to a zoo. This zoo is a very small one and has only two kinds of animals, say a lion and an elephant. After they came out of the zoo, the father showed them an animal and asked both of them "is this animal a lion or an elephant?"

Kid A suddenly draw the image of lion and elephant in a piece of paper based on what he saw inside the zoo. He compared both the images with the animal standing before and answered based on the closest match of image & animal, he answered: "The animal is lion". Kid B knows only the differences, based on different properties learned, he answered: "The animal is a lion". Kid C only learnt the distribution of the animals that his father has shown to him, hence he can play the same game with his brothers, as an author!

Here, we can see both kid A and kid B of them is finding the kind of animal, but the way of learning and the way of finding answer is entirely different, and kid C provides a descriptive view of the animals. In machine learning, we call kid A ***generative model****, kid B* ***discriminative model*** *and kid C* ***descriptive model****.*

Originally written by *Prathap Manohar Joshi*, published in Medium, adapted by *Tony Fang*.

[2] Wu, Ying Nian, et al. "A tale of three probabilistic families: Discriminative, descriptive, and generative models." *Quarterly of Applied Mathematics* 77.2 (2019): 423-465.

Generative Models.

Given an optional label h and an observable variable X, a generative model is a statistical model of the joint probability distribution of $h \times X$, *i.e.*, $p_\theta(h, X)$, where θ are the parameters of the model. If there is no labels provided, then the generative models is a statistical model of the probability distribution of X, *i.e.*, $p_\theta(X)$.

The generative models can be represented as the following diagram:

$$\text{hidden } h_i \rightarrow \text{input } X_i$$

where h_i is not a vector of features extracted from signal X_i, actually h_i is a vector of hidden variables that is used to generate X_i.

In supervised learning settings, *i.e.*, hidden variables h are provided as labels, the generative model can learn accurate mapping from h to X, and the linear interpolation in the space of h leads to very realistic non-linear interpolation in the space of X.

In unsupervised learning settings, *i.e.*, hidden variables h are unknown, the generative model is given as

$$X = g_\theta(h) + \varepsilon; \;\; \varepsilon \sim N\left(0, \sigma^2 I_p\right)$$

where $h \sim N(0, I_d)$ is d-dimensional hidden vector for latent factors, which is also the input of the generative neural network, which is a low-dimension vector; g_θ is a neural network that maps d-dimensional vector h to p-dimensional signal X, which is the network output, *e.g.*, a generated image. Notice that here $d \leq p$.

What we want to learn is the joint distribution of hidden vector h and signal X.

$$p_\theta(h, X) = \Pr(h)\, p_\theta(X \mid h)$$

Since $X \mid h \sim N(g_\theta(h), \sigma^2 I)$, we know that

$$p_\theta(h, X) = \text{const} \cdot \exp\left(-\frac{\|h\|^2}{2}\right) \cdot \exp\left(-\frac{\|X - g_\theta(h)\|^2}{2\sigma^2}\right)$$

If we take logarithm in both side of the previous formula, we can get

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

We may be interested in the posterior distribution in models, that is,

$$p_\theta(h \mid X) = \frac{p_\theta(h, X)}{p_\theta(X)} \propto p_\theta(h, X)$$

where $p_\theta(X)$ is the marginal distribution of X defined as follows.

$$p_\theta(X) = \int p_\theta(h, x)\mathrm{d}h = \int \Pr(h)\, p_\theta(X \mid h)\mathrm{d}h$$

Discriminative Models.

Given an observable variable X and a target variable Y, a discriminative model is a statistical model of the conditional probability distribution of $Y \mid X$, *i.e.*, $p_\theta(Y \mid X)$, where θ are the parameters of the model.

The discriminative models can be represented as the following diagram:

$$\text{input } X_i \rightarrow \text{features or hidden variables } h_i \rightarrow \text{output } y_i$$

where the vector of features h_i is computed from X_i via $h_i = h(X_i)$. In a non-hierarchical model, the feature vector h_i is designed, not learnt, *i.e.*, $h(\cdot)$ is a pre-specified transformation. In a hierarchical model, the feature vector h_i is usually learnt from the training samples.

In deep learning, the neural networks in general and the convolutional neural networks in particular were initially designed for discriminative models. Let X be p-dimensional input vector, and Y be the output. As for regression problems, we want to predict Y by $\hat{Y}$ which is a non-linear transformation of X: $\hat{Y} = f_\theta(X)$, where f_θ is the network parametrized by parameter θ.

As for classification problems, suppose there are K categories, the conditional probability of category k given input X is given by the following soft-max probability.

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

where $f_\theta(X)$ is a K-dimensional network output for input X, $f_\theta^{(k)}(X)$ indicates the k-th output dimension, and

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

Descriptive Models.

Given observations X, the descriptive model aims to estimate the data distribution of X, *i.e.*, $p_\theta(X)$.

The descriptive models can be represented as the following diagram:

$$\text{input } X \to \text{features or hidden variables } h_i \to p_\theta(X)$$

The descriptive models usually learn in unsupervised settings. The probability distribution is calculated by

$$p_\theta(X) = \left[\frac{1}{Z(\theta)}\exp f_\theta(X)\right] p_0(X)$$

where

$$Z(\theta) = \int f_\theta(x)\mathrm{d}x$$

and $p_0(X)$ is the reference distribution such as Gaussian white noise model $p_0(X) \propto \exp\left(-\frac{\|X\|^2}{2\sigma^2}\right)$.

1.8 Maximum Likelihood

Let $\{(X_i, y_i) \sim P_{data}(X, y)\}_{i=1,2,\cdots,n}$ be the training instances. Here we use y_i instead of y_i^* for convenience.

Discriminative Models.

The log-likelihood can be written as

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

According to *the law of large numbers*, we have

$$\mathfrak{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i \mid X_i) \approx \sum_X P_{data}(X) \log p_\theta(y_i \mid X_i) = E_{P_{data}(X)}[\log p_\theta(y \mid X)]$$

Hence,

$$\max_\theta \mathfrak{L}(\theta) = \max_\theta E_{P_{data}(X)}[\log p_\theta(y \mid X)] = \min_\theta\left(E_{P_{data}(X)}[\log P_{data}(y \mid X)] - E_{P_{data}(X)}[\log p_\theta(y \mid X)]\right)$$

Notice that the first term is a constant with regard to θ. Since $E_{P_{data}(X)} = E_{P_{data}(X,y)}$ (in the instances, X corresponds to only one ground-truth output y), we have

$$\max_\theta \mathfrak{L}(\theta) = \min_\theta KL\left(P_{data}(y \mid X) \mid p_\theta(y \mid X)\right)$$

The result shows that for discriminative models, the essence of maximum likelihood estimation is to minimize the KL divergence between the model $p_\theta(y \mid X)$ and the ground-truth $P_{data}(y \mid X)$.

Generative Models and Descriptive Models.

The descriptive models and the generative models learn from $\{X_i \sim P_{data}(X), i = 1,2,\cdots,n\}$ by maximizing the log-likelihood.

$$\mathfrak{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(X_i) \approx E_{P_{data}(X)}[\log p_\theta(X)]$$

Hence,

$$\max_\theta \mathfrak{L}(\theta) = \max_\theta E_{P_{data}(X)}[\log p_\theta(X)] = \min_\theta\left(E_{P_{data}(X)}[\log P_{data}(X)] - E_{P_{data}(X)}[\log p_\theta(X)]\right)$$

Notice that the first term is a constant with regard to θ. Therefore,

$$\max_\theta \mathfrak{L}(\theta) = \min_\theta KL\left(P_{data}(X) \mid p_\theta(X)\right)$$

The result shows that for discriminative and generative models, the essence of the maximum likelihood estimation is to minimize the KL divergence between the model $p_\theta(X)$ and the ground-truth $P_{data}(X)$.

1.9 Gradient Descent and Optimization Methods

Let $\{(X_i, y_i) \sim P_{data}(X, y)\}_{i=1,2,\cdots,n}$ be the training instances. Let $L_i(\theta) = L(y_i, X_i; \theta)$ be the loss caused by (X_i, y_i). For regression, we use the least square loss $L(y_i, X_i; \theta) = \left(y_i - f(X_i)\right)^2$ where $f(X_i)$ is parametrized by a model with parameters θ. For classification, we use the cross-entropy loss $L(y_i, X_i; \theta) = -\log p_\theta(y_i \mid X_i)$ where $p_\theta(y_i \mid X_i)$ is modeled by a model with parameters θ. Let $\mathcal{L}(\theta) = \sum_{i=1}^{n} L(y_i, X_i; \theta)$ be the overall loss averaged over the entire training dataset. The **gradient descent algorithm** is

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

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

The gradient descent algorithm may be time-consuming because we need to compute $\mathcal{L}'(\theta)$ by summing over all examples. If the number of examples is large, the computation may consume a large amount of time. We may use the following **stochastic gradient descent algorithm**. At each step, we randomly select i from $\{1,2,\cdots,n\}$. Then we update θ by

$$\theta_{t+1} = \theta_t - \eta_t \frac{\partial}{\partial\theta} L(y_i, X_i; \theta)$$

where η_t is the step size or learning rate, and $\frac{\partial}{\partial\theta} L(y_i, X_i; \theta)$ is the gradient descent only for the i-th sample.

Instead of randomly selecting a single sample, we may randomly select a **mini-batch**, and replace $\frac{\partial}{\partial\theta} L(y_i, X_i; \theta)$ by the average of this mini-batch, that is,

$$\theta_{t+1} = \theta_t - \eta_t \cdot \frac{1}{|\text{batch}|} \cdot \sum_{i\in\text{batch}\subset\text{allSamples}} \frac{\partial}{\partial\theta} L(y_i, X_i; \theta)$$

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.

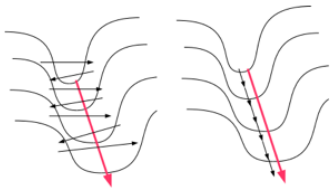
(1) $\sum_{t=1}^{\infty} \eta_t = \infty$, which ensures that the algorithm can go the distance toward the minimum.

(2) $\sum_{(i=1)}^{\infty} \eta_t^2 < \infty$, which ensures that the algorithm won't run away from the local minimum once 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.

Momentum.

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 the figure. In the left subfigure, 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 subfigure. We want to accumulate the momentum, and let it guide the descent. The following is the stochastic mini-batch gradient descent with momentum[3].



$$v_t = \gamma v_{t-1} + \eta_t g_t$$
$$\theta_t = \theta_{t-1} - v_t$$

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 0.9, for accumulating the momentum, and θ is updated based on the momentum.

Adagrad[4].

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 let

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

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.

Adadelta[5] and RMSprop.

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 use the following scheme.

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

where β can be set at 0.9. It can be shown that $G_t = (1-\beta)(\beta^{t-1} g_1^2 + \beta^{t-2} g_2^2 + \cdots + \beta g_{t-1}^2 + g_t^2)$, which is a sum over time with decaying weights.

---

[3] Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning.* PMLR, 2013.

[4] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research* 12.7 (2011).

[5] Zeiler, Matthew D. "Adadelta: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).

Adam[6].

The Adam optimizer combines the idea of RMSprop and the idea of momentum.

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

Gradient Ascent.

To minimize loss, we use the gradient descent algorithm; and to maximize the log-likelihood, we usually use the gradient ascent algorithm, which is similar to the former one with only one sign changed.

$$\theta_{t+1} = \theta_t + \eta \mathfrak{L}'(\theta_t)$$

The gradient ascent algorithm also has the optimization methods that stated before.

1.10 Gradient of Log-likelihood

In this section, we define the batch log-likelihood as the average log-likelihood of the samples.

Discriminative Models.

The log-likelihood method is usually used for classification task, hence the soft-max probability is given by the following formula.

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

where $Z(\theta) = \sum_k exp\, f_\theta^{(k)}(X)$, and $f_\theta^{(k)}(X)$ indicates the k-th output dimension.

If $y = k$, that is, the ground-truth of y is k, then

$$\begin{aligned}
\frac{\partial}{\partial\theta} \log p(y \mid X) &= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \frac{\partial}{\partial\theta} \log Z(\theta) \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \frac{1}{Z(\theta)} \sum_{k'} \frac{\partial}{\partial\theta} \exp f_\theta^{(k')}(X) \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \sum_{k'} \frac{\exp f_\theta^{(k')}(X)}{Z(\theta)} \cdot \frac{\partial}{\partial\theta} f_\theta^{(k')}(X) \\
&= \frac{\partial}{\partial\theta} f_\theta^{(k)}(X) - \sum_{k'} p_{k'} \cdot \frac{\partial}{\partial\theta} f_\theta^{(k')}(X) \\
&= \sum_{k'} [1(k' = k) - p_{k'}] \frac{\partial}{\partial\theta} f_\theta^{(k')}(X) \\
&= \frac{\partial}{\partial\theta} f_\theta(X)^T (Y - p) \\
&= \frac{\partial}{\partial\theta} f_\theta(X)^T (Y - E_\theta(Y \mid X))
\end{aligned}$$

where p is the predicted value, and Y is one-hot version of $y = k$, that is,

$$Y_i = \begin{cases} 0 & i \neq k \\ 1 & i = k \end{cases}$$

which can be viewed as the ground-truth.

For observation (X_i, y_i),

$$\frac{\partial}{\partial\theta} \log p(y_i \mid X_i) = \frac{\partial}{\partial\theta} f_\theta(X_i)^T \big(Y_i - E_\theta(Y_i \mid X_i)\big)$$

Therefore, the gradient of the log-likelihood is:

$$\frac{\partial}{\partial\theta} \mathfrak{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial\theta} f_\theta(X_i)^T \big(Y_i - E_\theta(Y_i \mid X_i)\big)$$

---

[6] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

This leads to a mini-batch stochastic gradient ascent algorithm for maximizing $\mathfrak{L}(\theta)$:

$$\theta_{t+1} = \theta_t + \eta_t \left[\frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} f_{\theta_t}(X_i)^T \left(Y_i - E_{\theta_t}(Y_i \mid X_i) \right) \right]$$

where η_t is the learning rate. Therefore, the learning process can be interpreted as "***learn from errors***", since $Y_i - E_\theta(Y_i \mid X_i)$ is the error term between the ground-truth and the model prediction, and $f_\theta(X)^T$ is irrelevant with the ground-truth Y_i. At the maximum likelihood estimate $\hat{\theta}$, the model matches the training data:

$$\left[\frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} f_{\theta_t}(X_i)^T \left(Y_i - E_{\theta_t}(Y_i \mid X_i) \right) \right] = 0$$

Descriptive Models[7]**.**

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 $Z(\theta) = \int \exp f_\theta(x)\, \mathrm{d}x$. Therefore,

$$\begin{aligned}
\frac{\partial}{\partial \theta} \log p_\theta(X) &= \frac{\partial}{\partial \theta} f_\theta(X) - \frac{\partial}{\partial \theta} \log Z(\theta) \\
&= \frac{\partial}{\partial \theta} f_\theta(X) - \frac{1}{Z(\theta)} \int \frac{\partial}{\partial \theta} \exp f_\theta(X) dX \\
&= \frac{\partial}{\partial \theta} f_\theta(X) - \frac{1}{Z(\theta)} \int \exp(f_\theta(X)) \frac{\partial}{\partial \theta} f_\theta(X) dX \\
&= \frac{\partial}{\partial \theta} f_\theta(X) - \sum_{X'} p_\theta(X') \frac{\partial}{\partial \theta} f_\theta(X') \\
&= \frac{\partial}{\partial \theta} f_\theta(X) - E\left[\frac{\partial}{\partial \theta} f_\theta(X) \right]
\end{aligned}$$

For an observation X_i,

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

Therefore, the gradient of the log-likelihood is:

$$\frac{\partial}{\partial \theta} \mathfrak{L}(\theta) = \frac{\partial}{\partial \theta} \cdot \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(X_i) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} f_\theta(X_i) - E\left[\frac{\partial}{\partial \theta} f_\theta(X) \right]$$

This leads to a mini-batch stochastic gradient ascent algorithm for maximizing $\mathfrak{L}(\theta)$:

$$\theta_{t+1} = \theta_t + \eta_t \left[\frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \theta} f_{\theta_t}(X_i) - \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \frac{\partial}{\partial \theta} f_{\theta_t}(\tilde{X}_i) \right]$$

where $\{ \tilde{X}_i \mid i = 1,2, \cdots, \tilde{n} \}$ are random samples from p_{θ_t}, and η_t is the learning rate. Therefore, the learning process can be interpreted as "***learn by synthesis***". The $\{\tilde{X}_i\}$ are the synthesized data generated by the current model. The learning algorithm updates the parameters in order to make the synthesized data similar to the observed data in terms of features statistics. At the maximum likelihood estimate $\hat{\theta}$, the model matches the data:

$$E_{\hat{\theta}}\left[\frac{\partial}{\partial \theta} f_\theta(X) \right] = E_{P_{data}}\left[\frac{\partial}{\partial \theta} f_\theta(X) \right]$$

Generative Models.

The generative models aim to find out how the data is generated. Since the hidden variables h_i are often unknown, we should compute $p_\theta(X)$ instead of directly calculate $p_\theta(h, X)$ as the log-likelihood.

$$p_\theta(X) = \int \Pr(h)\, p_\theta(h, X) \mathrm{d}h$$

---

[7] Wu, Ying Nian, et al. "A tale of three probabilistic families: Discriminative, descriptive, and generative models." *Quarterly of Applied Mathematics* 77.2 (2019): 423-465.

Therefore,

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

For an observation X_i,

$$\frac{\partial}{\partial\theta}\log p_\theta(X_i) = E_{p_\theta(h|X_i)}\left[\frac{\partial}{\partial\theta}log\ p_\theta(h,X_i)\right]$$

Therefore, the gradient of the log-likelihood is:

$$\frac{\partial}{\partial\theta}\mathfrak{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} E_{p_\theta(h|X_i)}\left[\frac{\partial}{\partial\theta}\log p_\theta(h,X_i)\right]$$

This leads to a mini-batch stochastic gradient ascent algorithm for maximizing $\mathfrak{L}(\theta)$.

$$\theta_{t+1} = \theta_t + \eta_t\left[\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta}\log p_{\theta_t}(h_i,X_i)\right]$$

where η_t is the learning rate, and in each iteration, for each X_i, h_i is sampled from $p_{\theta_t}(h\mid X_i)$. Notice that here we use Monte-Carlo approximation methods in calculating $\frac{\partial}{\partial\theta}\mathfrak{L}(\theta_t)$. The convergence of the algorithm is proved in some literatures[8].

By the way, we can calculate $\frac{\partial}{\partial\theta}\log p_{\theta_t}(h_i,X_i)$ using the result in section 1.7.

$$\frac{\partial}{\partial\theta}\log p_{\theta_t}(h_i,X_i) = \frac{\partial}{\partial\theta}\left[-\frac{1}{2}\|h_i\|^2 - \frac{1}{2\sigma^2}\left\|X_i - g_{\theta_t}(h_i)\right\|^2 + \text{const}\right] = \frac{1}{\sigma^2}\left\|X_i - g_{\theta_t}(h_i)\right\|\frac{\partial}{\partial\theta}g_{\theta_t}(h_i)$$

1.11 Langevin Dynamics

In previous derivation of the descriptive models' and the generative models' gradients of log-likelihood, we omit the process of sampling points from a given distribution. In descriptive model, we want to sample some elements from p_{θ_t} randomly; while in generative model, we want to sample an element from $p_{\theta_t}(h\mid X_i)$.

Actually, this can be solved using Langevin Dynamics, which will be introduced later. Before that, we want to introduce Brownian motion first.

Brownian Motion.

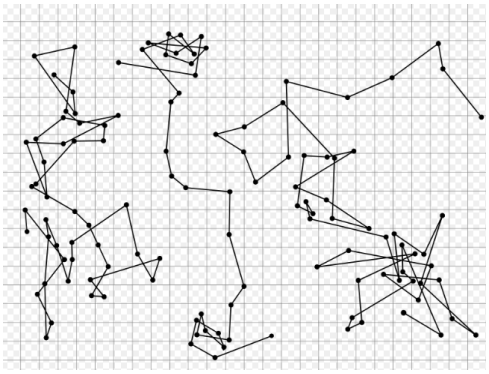
The Brownian motion is defined by

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

where $E(\varepsilon_t) = 0$ and $Var(\varepsilon_t) = 1$. Let $X_0 = x$, if we divide interval $[0,t]$ into n periods, so that $\Delta t = t/n$. Write ε_t within each period as ε_i for $i = 1,2,\cdots,n$. Then

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

according to *the central limit theorem*.



[8] Younes, Laurent. "On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates." *Stochastics: An International Journal of Probability and Stochastic Processes* 65.3-4 (1999): 177-228.

The above equation also shows why we need $\sqrt{\Delta t}$ scaling in order to make the $Var(X_t)$ 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 $\frac{dX(t)}{dt} = \frac{\sigma\varepsilon_t}{\sqrt{\Delta t}} \to \infty$.

Langevin Dynamics.

The Langevin dynamics has the following term.

$$X_{t+\Delta t} - X_t = -\frac{1}{2}\nabla U(X_t)\Delta t + \varepsilon_t\sqrt{\Delta t}$$

This term is the solution of Fokker-Planck equation. You can go watching "Langevin dynamics for sampling and global optimization" video[9] for detailed derivation.

Sometimes, the $\sqrt{\Delta t}$ can be re-written as η, which denotes step size or learning rate. Hence, the discrete version of Langevin dynamics can also be stated as follows.

$$X_{t+1} - X_t = -\frac{\eta^2}{2}\nabla U(X_t) + \eta\varepsilon_t$$

Notice that in the previous formula, the gradient descent term $-\frac{\eta^2}{2}\nabla U(X_t)$ decreases the free energy of the model, while the Brownian motion $\eta\varepsilon_t$ increases the entropy. The discrete version Langevin dynamics is useful in sampling, since the gradient descent term makes those with high probabilities appears more often, and the Brownian motion term ensures that other elements with low probabilities possible to be chosen. The sampling process should carefully choose η, and we can add Metropolis-Hasting step to correct for the fitness of s.

Langevin Dynamics Sampling in Descriptive Model.

In descriptive model, we want to sample $\tilde{n}$ examples from p_{θ_t}. Simply let $U = -p_{\theta_t}$ and apply the Langevin dynamics can sample the examples according to the distribution p_{θ_t}.

Langevin Dynamics Sampling in Generative Model.

In generative model, we want to sample h_i from $p_{\theta_t}(h \mid X_i)$. Hence, we can set the energy function U as $-\log p_{\theta_t}(h \mid X_i)$ for convenience. Therefore, according to conditional probability,

$$\frac{\partial}{\partial h}U = -\frac{\partial}{\partial h}\log p_{\theta_t}(h \mid X_i) = -\frac{\partial}{\partial h}\log p_{\theta_t}(h, X_i) + \frac{\partial}{\partial h}\log p_{\theta_t}(X_i) = -\frac{\partial}{\partial h}\log p_{\theta_t}(h, X_i)$$

Notice that $\log p_{\theta_t}(X_i)$ is irrelevant with h, hence its derivative is 0. Therefore, the sampling strategy is

$$h_{t+1} - h_t = \frac{\eta^2}{2}\frac{\partial}{\partial h}\log p_{\theta_t}(h_t, X_i) + \eta\varepsilon_t = \frac{\eta^2}{2}\left[\frac{1}{\sigma^2}\left\|X_i - g_{\theta_t}(h_t)\right\|\frac{\partial}{\partial\theta}g_{\theta_t}(h_t)\right] + \eta\varepsilon_t$$

We can also apply several steps of the Langevin dynamics to get a new sample h_{t+1} from h_t.

1.12 Logistic Regression Revised: Optimizing Log-likelihood via Gradient Ascent

For logistic regression, the log-likelihood function is

$$\mathfrak{L}(\theta) = \sum_{i=1}^{n}[y_i X_i^T\beta - \log(1 + \exp X_i^T\beta)]$$

Therefore, the gradient of the log-likelihood is

$$\frac{\partial}{\partial\theta}\mathfrak{L}(\theta) = \sum_{i=1}^{n}\left[y_i X_i - \frac{e^{X_i^T\beta}}{1 + e^{X_i^T\beta}}X_i\right] = \sum_{i=1}^{n}(y_i - p_i)X_i$$

where $p_i = \frac{e^{X_i^T\beta}}{1+e^{X_i^T\beta}} = \frac{1}{1+e^{-X_i^T\beta}}$. Therefore, we can use the gradient ascent algorithm to iteratively update β and maximize the log-likelihood, that is,

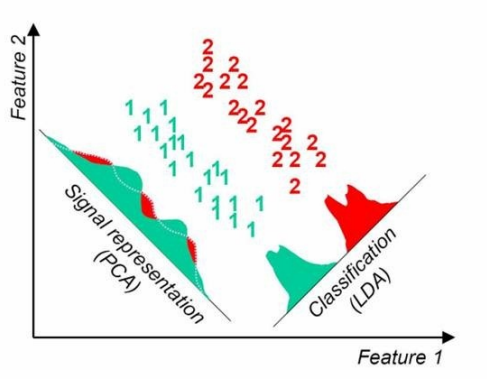
$$\beta_{t+1} = \beta_t + \eta_t\sum_{i=1}^{n}(y_i - p_i)X_i$$

where η_t is the learning rate of the t-th iteration. We can also regard the negated log-likelihood as loss function and use gradient descent algorithm. Therefore, the algorithm can be interpreted as "***learn from errors***" like the previous discriminative model, since $y_i - p_i$ is the error term, and X_i is given sample irrelevant with model.

---

[9] https://www.youtube.com/watch?v=3-KzIjoFJy4&t=1904s

1.13 Linear Discriminant Analysis



Let us consider the situation that only has two classes $y = +1$ and $y = -1$. $\Omega = \{(X_i, y_i)\}_{i=1,2,\cdots,n}$ denotes the training set, and Ω^+, Ω^- denote the set of positive training samples and the negative training samples respectively. Obviously, $\Omega = \Omega^+ \cup \Omega^-$. In the scenario of linear discriminant analysis, we assume that

1. $\forall X_i \in \Omega^+, p(X_i \mid y = +1) \sim N(\mu^+, \Sigma^+)$;
2. $\forall X_i \in \Omega^-, p(X_i \mid y_i = -1) \sim N(\mu^-, \Sigma^-)$.

The task is to learn a linear function that projects X_i to $z = X_i^T\beta$, so that the projection maximizes the inter-class variance and minimizes the intra-class variance. The intra-class variance $\sigma^2_{between}$ and the inter-class variance σ^2_{within} are given as follows.

$$\begin{aligned}\sigma^2_{between} &= (E_{i\in\Omega^+}[X_i^T\beta] - E_{i\in\Omega^-}[X_i^T\beta])^2 \\ &= \left((E_{i\in\Omega^+}[X_i] - E_{i\in\Omega^-}[X_i])^T\beta\right)^2 \\ &= [(\mu^+ - \mu^-)^T\beta]^2\end{aligned}$$

$$\sigma^2_{within} = n_{pos}\sigma^2_{pos} + n_{neg}\sigma^2_{neg} = |\Omega^+|\sigma^2_{pos} + |\Omega^-|\sigma^2_{neg}$$

where

$$\begin{aligned}\sigma^2_{pos} &= E_{i\in\Omega^+}\left[(X_i^T\beta - E_{i'\in\Omega^+}[X_{i'}^T\beta])^2\right] \\ &= E_{i\in\Omega^+}\left[(\beta^T X_i - E_{i'\in\Omega^+}[\beta^T X_{i'}])(X_i^T\beta - E_{i'\in\Omega^+}[X_{i'}^T\beta])\right] \\ &= E_{i\in\Omega^+}\left[\beta^T(X_i - E_{i'\in\Omega^+}[X_{i'}])(X_i^T - E_{i'\in\Omega^+}[X_{i'}^T])\beta\right] \\ &= \beta^T E_{i\in\Omega^+}\left[(X_i - E_{i'\in\Omega^+}[X_{i'}])(X_i^T - E_{i'\in\Omega^+}[X_{i'}^T])\right]\beta \\ &= \beta^T\Sigma^+\beta\end{aligned}$$

and similarly, $\sigma^2_{neg} = \beta^T\Sigma^-\beta$. Therefore, the objective is

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

where $S_B = (\mu^+ - \mu^-)(\mu^+ - \mu^-)^T$, and $S_W = n_{pos}\Sigma^+ + n_{neg}\Sigma^-$. Hence, the optimization problem is

$$\max_\beta S = \frac{\beta^T S_B\beta}{\beta^T S_W\beta}$$

Consider that β may have different scales and produces the same score of S. Hence, we normalize the scale of β by setting $\beta^T S_W\beta = 1$. Therefore, the optimization problem is modified as

$$\max_\beta \beta^T S_B\beta \qquad \text{s.t. } \beta^T S_W\beta = 1$$

By using Lagrange multipliers, we construct the function $L = \beta^T S_B\beta - \lambda(\beta^T S_W\beta - 1)$. The optimal point must have the following condition:

$$\frac{\partial L}{\partial\beta} = 0 \implies S_W^{-1}S_B\beta = \lambda\beta$$

Hence, the optimal β should be the eigenvector of matrix $S_W^{-1}S_B$, and the orientation of β should follows $S_W^{-1}(\mu^+ - \mu^-)$ since $(\mu^+ - \mu^-)\beta$ is a scalar constant.