# 8 Logistic regression

Logistic regression, despite its name, is a model used for classification problems. The output variable can be *binary* or *categorical*. The procedure is quite similar for both cases, so binary classification will be analyzed first. The model is trained to directly fit the posterior distribution of the labels $p(y \mid \mathbf{x}, \boldsymbol{\theta})$, because this is a discriminative classifier.

## 8.2 Model specification

Given a features vector, $\mathbf{x}$, a binary classifier returns the most probable class $y \in \{0, 1\}$. So, the conditional distribution of $y$ must be:

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \text{Ber}(y; f(\mathbf{x}))$$

for a function $f : \mathbb{R}^D \to [0, 1]$. The **logistic regression** model, defines $f$ to be the *sigmoid* function,

$$\text{sigm}(x) \coloneqq \frac{e^x}{e^x + 1} = \frac{1}{1 - e^{-x}}$$

which maps values to the appropriate range (the only parameter $\theta$ of a Bernoulli is the probability of the event '$y = 1$'). The most simple choice to reduce the features vector to a single scalar is the linear function, $\mathbf{w}^T\mathbf{x}$. Putting all together, logistic regression is the following model:

$$p(y \mid \mathbf{x}, \mathbf{w}) = \text{Ber}(y; \text{sigm}(\mathbf{w}^T\mathbf{x})) \tag{45}$$

The decision rule will return $y = 1$, if $\text{sigm}(\cdot)$ is higher than 0.5; zero otherwise. From the model definition, we see that the decision boundary is the line $\mathbf{w}^T\mathbf{x} = 0$ (the line orthogonal to the vector $\mathbf{w}$).

This model seems a completely unmotivated choice. Yet, this decision rule arises in some generative classifiers, such as those with features distributed as Poissons, or in linear discriminant analysis (see equation (22)). To understand how this is possible, we need to define the **log-odds**. The log-odds (LO) of an event $A$ are:

$$\text{LO}(A) \coloneqq \log\left(\frac{p(A)}{p(\neg A)}\right)$$

We find that $\text{sigm}(\text{LO}(A)) = p(A)$, so log-odds can be transformed to probabilities trough the logistic function. The opposite transformation is given by the **logit** function:

$$\text{logit}(x) \coloneqq \text{sigm}^{-1}(x) = \log\left(\frac{x}{1 - x}\right)$$

Thus, $\text{LO}(A) = \text{logit}(p(A))$. The transformation is bijective for probabilities in $(0, 1)$, so log-odds are an equivalent representation of probabilities, and

it's possible to directly work with them. From equation (45), we see that the log-odds of $(y = 1 \mid \mathbf{x}, \mathbf{w})$ are: $\mathbf{w}^T\mathbf{x}$. In log-odds, the Bayes theorem becomes the sum of the transformed prior and each of the log likelihood ratios of new evidences. For our classification problem, this means:

$$\begin{aligned}\operatorname{logit}(p(y \mid \mathbf{x})) = \operatorname{logit}(p(y)) + \\ + \log\left(\frac{p(x_1 \mid y)}{p(x_1 \mid 1-y)}\right) + \cdots + \log\left(\frac{p(x_{D-1} \mid y)}{p(x_{D-1} \mid 1-y)}\right)\end{aligned}$$

for $D-1$ conditionally independent features. The vector $\mathbf{w}$ has size $D$ and $w_1$ is the intercept. In conclusion, the only assumption made by logistic regression is to say that each of the addends is $w_i x_i$ (the simplest model in each input $x_i$) and that $\operatorname{logit}(p(y)) = w_1$ is the intercept.

Logistic regression builds a discriminative model, so the Bayes theorem is never used on the features vector. This has been useful just to show that it can reproduce the posterior of a generative model without the need to explicitly assume each class conditional density.

## 8.3 Model fitting

As usual, we want to choose the parameters so to minimize the negative log likelihood. For a binary variable $y$ and a dataset of size $N$, it can be written:

$$\operatorname{NLL}(\mathbf{w}) = -\sum_{i=1}^{N} \log(y_i \log \mu_i + (1-y_i)\log(1-\mu))$$

where $\mu_i \coloneqq \operatorname{sigm}(\mathbf{w}^T\mathbf{x}_i)$. It takes the form of a negative *cross-entropy*. The NLL is a nonlinear function in $\mathbf{w}$, and there is no closed form expression for the MLE as it was for linear regression. So, in order to find a global minimum, it's necessary to use algorithms for unconstrained optimization. The two most important quantities for a continuous functions are the gradient and the Hessian. In vector form, they are:

$$\begin{aligned}\mathbf{g}(\mathbf{w}) &\coloneqq \left(\frac{d}{d\mathbf{w}}\operatorname{NLL}(\mathbf{w})\right)^T = \mathbf{X}^T(\boldsymbol{\mu} - \mathbf{y}) \\ \mathbf{H}(\mathbf{w}) &\coloneqq \frac{d}{d\mathbf{w}}\mathbf{g}(\mathbf{w})^T = \mathbf{X}^T\mathbf{S}\mathbf{X}\end{aligned} \tag{46}$$

where $\mathbf{X}$ is the design matrix, and $\mathbf{S} \coloneqq \operatorname{diag}[\mu_1(1-\mu_1), \ldots, \mu_N(1-\mu_N)]$.

Now we will see three basic optimization algorithms for continuous scalar functions $f(\boldsymbol{\theta})$, that have a wide range of applications. For a positive definite Hessian matrix, as it is for this NLL, there is a unique global minimum. So, a simple algorithm such as **steepest descent** is guaranteed to reach it, eventually. Steepest (gradient) descent, at each step $k$, moves on segments of length $\eta_k$ along the steepest direction:

$$\boldsymbol{\theta}_{k+1} \coloneqq \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$$

where $\mathbf{g}_k \coloneqq \mathbf{g}(\boldsymbol{\theta}_k)$. Big step sizes lead to instability, because the linear approximation is not valid anymore. Instead of a fixed step, **line minimization** method chooses $\eta_k$ so that $f(\boldsymbol{\theta}_{k+1})$ is the minimum on that segment. We find that it should be picked such that $\mathbf{g}_k \perp \mathbf{g}_{k+1}$. This algorithm ensures *global convergence*.

The **heavy ball method** is an heuristic to reduce this zig-zag effect, through the addition of a momentum term:

$$\boldsymbol{\theta}_{k+1} \coloneqq \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k + \mu_k(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})$$

with $0 \leq \mu_k \leq 1$.

In optimization, the **Newton's method** is an iterative algorithm that uses the second order approximation of the function. At each step $k$, this is:

$$f(\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}_k) + (\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \mathbf{g}_k + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \mathbf{H}_k (\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

The successive value is chosen to minimize this paraboloid, and this is given by:

$$\boldsymbol{\theta}_{k+1} \coloneqq \boldsymbol{\theta}_k - \mathbf{H}_k^{-1} \mathbf{g}_k$$

Often, each update step is scaled down by a constant, $\eta \in (0, 1)$, and the algorithm is called *relaxed* Newton's method.

Back to the problem of fitting the logistic model, the straightforward application of the Newton's method is known as the **iteratively reweighted least squares** (IRLS) algorithm. The generic step can be expressed in two ways:

$$\begin{aligned} \mathbf{w}_{k+1} &\coloneqq \mathbf{w}_k - \mathbf{H}_k^{-1} \mathbf{g}_k \\ &= (\mathbf{X}^T \mathbf{S}_k \mathbf{X})^{-1} \mathbf{X}^T \mathbf{S}_k \mathbf{z}_k \end{aligned}$$

with $\mathbf{z}_k \coloneqq \mathbf{X}\mathbf{w}_k + \mathbf{S}_k^{-1}(\mathbf{y} - \boldsymbol{\mu}_k)$. The first equation is immediate to implement. The second form, instead, shows how each vector of parameters, $\mathbf{w}_{k+1}$, can be seen as solution of a *weighted least squares problem*. In fact, it is computed with the weighted left-pseudoinverse of the design matrix. The vector $\mathbf{z}$ is called the *working response*. Also note that $\mathbf{S}$ is a diagonal matrix, so each $z_i$ can be computed independently. This method is a good trade-off of simplicity and efficiency.

However, bigger models require more sophisticated algorithms, which, for example, do not require the inversion of any matrix. The *truncated Newton* compute the solution of $\mathbf{H}_k \mathbf{d}_\mathbf{k} = -\mathbf{g}_k$ with the conjugate gradient method. The solution is approximated, as not all basis vector are used. The Levenberg Marquardt algorithm mix adaptively steepest descent and Newton method. The BFGS is an important algorithm that updates at each step approximations of the Hessian or its inverse, involving just inner products. The most used algorithm is limited memory BFGS, which follows the same

procedure of BFGS, but it doesn't need to store all $D^2$ elements of the Hessian. In fact, it is sufficient to remember a some of the last updates to reconstruct an approximation that is good enough.

With one of these techniques it's possible to compute the MLE. Yet, as previously noticed, this estimator can easily overfit. If the data is linearly separable, the model can achieve zero error on the training set. As result, the MLE would increase the probability of the observed data close to 1, with the effect: $\|\hat{\mathbf{w}}\| \to \infty$. With such high parameters, the model would become the **linear threshold unit**, $\mathbb{I}(\hat{\mathbf{w}}^T\mathbf{x} > 0)$. To overcome this issue, we introduce a $l_2$ regularization, as it was done in ridge regression. The modified objective function now includes $l_2$ norm as a second goal to be minimized:

$$f_r(\mathbf{w}) \coloneqq \text{NLL}(\mathbf{w}) + \lambda \mathbf{w}^T\mathbf{w} \tag{47}$$

Hence, we need to pass to the optimization algorithm the following gradient and Hessian:

$$\begin{aligned}\mathbf{g}_r(\mathbf{w}) &= \mathbf{g}(\mathbf{w}) + 2\lambda\mathbf{w} \\ \mathbf{H}_r(\mathbf{w}) &= \mathbf{H}(\mathbf{w}) + 2\lambda\mathbf{I}\end{aligned}$$

Now, we're ready to extend the same procedures to the case of a categorical response variable. The generalization of the logistic function is the **softmax**, that has been introduced in equation (21). Therefore, **multi-class logistic regression** is the model:

$$p(y_{\text{cat}} = c \mid \mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^T\mathbf{x})}{\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^T\mathbf{x})} = \mathcal{S}(\mathbf{W}^T\mathbf{x})_c \tag{48}$$

where $\mathbf{w}_c$ is the $c$-th column of $\mathbf{W}$.

In classification, each value of the categorical variable identifies a distinct element of a set. Another common representation is the **dummy encoding** (also called *1-hot*), in which the output is defined:

$$\mathbf{y} \coloneqq [\mathbb{I}(y_{\text{cat}} = 1), \dots, \mathbb{I}(y_{\text{cat}} = C-1)]^T$$

If the encoding would have been defined with $C$ entries, the redundancy of the representation would be an issue when training the classifier. In fact, a selected class, called the *control group*, can be represented by the vector of all zeros. The control group must be a well determined class as any other. It has no associated parameters to estimate, and we can assume $\mathbf{w}_C = \mathbf{0}$. The only difference is that the explanatory variables, $\mathbf{w}_c^T\mathbf{x}$, are expressed w.r.t the reference class, which has a null value.

The gradient and the Hessian in the $D(C-1)$ parameters, $\mathbf{W}$, can be written concisely with the *Kronecker product* operator. It is defined:

$$\mathbf{A} \otimes \mathbf{B} \coloneqq \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n1}\mathbf{B} & \dots & a_{nm}\mathbf{B} \end{bmatrix}$$

So, we have:

$$\mathbf{g}(\mathbf{W}) = \sum_{i=1}^{N} (\boldsymbol{\mu}_i - \mathbf{y}_i) \otimes \mathbf{x}_i$$
$$\mathbf{H}(\mathbf{W}) = \sum_{i=1}^{N} (\text{diag}[\boldsymbol{\mu}_i] - \boldsymbol{\mu}_i \boldsymbol{\mu}_i^T) \otimes (\mathbf{x}_i \mathbf{x}_i{}^T)$$

in which $\mathbf{y}_i$ is the dummy encoding for the sample $i$, and $\boldsymbol{\mu}_i \coloneqq [p(y_{i_{\text{cat}}} = 1 \mid \mathbf{x}_i, \mathbf{W}), \ldots, p(y_{i_{\text{cat}}} = C-1 \mid \mathbf{x}_i, \mathbf{W})]$, computed from (48). The $l_2$ regularized versions toward the origin, with a fixed variance matrix $\mathbf{V}_0$, are:

$$\mathbf{g}_r(\mathbf{W}) = \mathbf{g}(\mathbf{W}) + \mathbf{V}_0^{-1} \mathbf{W}\, \mathbf{1}$$
$$\mathbf{H}_r(\mathbf{W}) = \mathbf{H}(\mathbf{W}) + \mathbf{I} \otimes \mathbf{V}_0^{-1}$$

where $\mathbf{1}$ is a column of ones.

## 8.4 Bayesian logistic regression

The parameters we obtain with a regularized negative log likelihood are exactly the MAP estimates, corresponding to a Gaussian prior. In fact, just as it was for linear regression, we can assume:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, \lambda \mathbf{I})$$

Then, maximizing the posterior distribution is equivalent to minimizing (47). Still, we have just the mode of $p(\mathbf{w} \mid \mathcal{D})$, not the actual distribution. In fact, the closed form can't be found easily without a convenient conjugate prior. If the posterior is still needed, for example to compute confidence intervals, it can be approximated.

The approximation we will use is a Gaussian. First, we observe that the posterior can be put in exponential form,

$$p(\mathbf{w} \mid \mathcal{D}) = \frac{1}{Z} e^{-E(\mathbf{w})} \qquad \text{with} \qquad \begin{aligned} Z &= p(\mathbf{y}) \\ E(\mathbf{w}) &= -\log p(\mathbf{y}, \mathbf{w} \mid \mathbf{X}) \end{aligned}$$

The *energy function*, $E$, is known, because it is the joint probability of the data:

$$E(\mathbf{w}) = -\sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i, \mathbf{w}) - \log p(\mathbf{w})$$

Now, if we approximate it with its second order Taylor expansion around the mode, $\hat{\mathbf{w}}$, the joint becomes a Gaussian function:

$$p(\mathbf{y}, \mathbf{w} \mid \mathbf{X}) \approx e^{-E(\hat{\mathbf{w}})} \exp\left( -\frac{1}{2} (\mathbf{w} - \hat{\mathbf{w}})^T \mathbf{H} (\mathbf{w} - \hat{\mathbf{w}}) \right)$$

because $\mathbf{g}(\hat{\mathbf{w}}) = \mathbf{0}$. The constant $Z$ is called the **Laplace approximation** to the marginal likelihood, and it is chosen to normalize the last equation to a Gaussian in $\mathbf{w}$. So, the final result is:

$$p(\mathbf{w} \mid \mathcal{D}) \approx \mathcal{N}(\mathbf{w}; \hat{\mathbf{w}}, \mathbf{H}^{-1})$$

By construction, it is centred at the mode of $E(\mathbf{w})$. The precision is the Hessian matrix.

Even with such a simple posterior, we can't easily integrate it to a posterior predictive analytically. As usual, it's possible to use a plug-in estimate:

$$p(y = 1 \mid \mathbf{x}, \mathcal{D}) \approx \text{sigm}(\mathbb{E}_{\mathbf{w}|\mathcal{D}}[\mathbf{w}]^T \mathbf{x})$$

However, a better alternative is to approximate it with Monte Carlo:

$$p(y = 1 \mid \mathbf{x}, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} \text{sigm}((\mathbf{w}_s)^T \mathbf{x})$$

with $S$ samples $\mathbf{w}_s \sim p(\mathbf{w} \mid \mathcal{D})$ from the posterior.

A third possibility is to approximate the likelihood with the **probit model**. The probit, $\Phi(\cdot)$, is a S-shaped function with the same domains of the sigmoid, defined as the cumulative density function of the standard normal. The advantage is that it can be convoluted with a Gaussian in closed form. First, we have to express the integral on a scalar variable:

$$\begin{aligned} p(y = 1 \mid \mathbf{x}, \mathcal{D}) &= \int_{\mathcal{W}} p(y = 1 \mid \mathbf{x}, \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, d\mathbf{w} \\ &\approx \int_{\mathcal{W}} \text{sigm}(\mathbf{w}^T \mathbf{x})\, \mathcal{N}(\mathbf{w}; \hat{\mathbf{w}}, \mathbf{H}^{-1})\, d\mathbf{w} \\ &\approx \int_{-\infty}^{+\infty} \text{sigm}(a)\, \mathcal{N}(a; \mu_a, \sigma_a)\, da \end{aligned}$$

with $\mu_a \coloneqq \hat{\mathbf{w}}^T \mathbf{x}$ and $\sigma_a \coloneqq \mathbf{x}^T \mathbf{H}^{-1} \mathbf{x}$. Then, using the approximation, $\text{sigm}(a) \approx \Phi(a\sqrt{\pi/8})$, we get the distribution:

$$p(y = 1 \mid \mathbf{x}, \mathcal{D}) \approx \text{sigm}(\mu_a (1 + \pi \sigma_a^2 / 8)^{-1/2})$$

## 8.5 Online learning and stochastic optimization

What has been considered up to now is the most common situation: **offline** learning. Training a model offline means that the complete dataset is available and the computation can be made on the whole **batch** of data. In **online** learning, instead, we need to provide an estimate (or response) as new samples arrive.

In case of *streaming data*, this is the only option. However, online algorithms can be also applied to complete datasets. In this case, they would

scan multiple times the same data. Each pass is called an **epoch**. If the dataset is randomly permuted, performances can improve between epochs. **Mini-batches** are smaller portions of data that are used together, useful with large datasets. If the size of a mini-batch is reduced to a single sample, inputs are treated as streaming data.

The frequentist approach well adapts to online learning, because it conveniently focuses on the future inputs. They are the only true random variables, in fact. Usually, we want to optimize risk functions of the form:

$$\bar{f}(\boldsymbol{\theta}) \coloneqq \mathbb{E}_{\mathbf{z}}[f(\boldsymbol{\theta}, \mathbf{z})]$$

Here, $\mathbf{z}$ is a generic sample, that could be either a single value or an input-output couple; $f$ could be a negative log-likelihood or any other cost function. Taking the derivative of this convex function, we get to the problem of finding roots of:

$$\bar{\mathbf{g}}(\boldsymbol{\theta}) \coloneqq \mathbb{E}_{\mathbf{z}}[\mathbf{g}(\boldsymbol{\theta}, \mathbf{z})]$$

This can be solved with **stochastic approximation** algorithms. The Robbins-Monro algorithm is an iterative method that approximates a zero of the function $\bar{\mathbf{g}}(\boldsymbol{\theta})$, given noisy measurements $\mathbf{g}(\boldsymbol{\theta}, \mathbf{z}_i)$. For each new input $\mathbf{z}_i$, it updates:

$$\boldsymbol{\theta}_{i+1} \coloneqq \boldsymbol{\theta}_i - \eta_i \mathbf{g}(\boldsymbol{\theta}_i, \mathbf{z}_i) \tag{49}$$

With an appropriate choice of steps, $\eta_i$, this estimate converges to the zero $\bar{\mathbf{g}}(\boldsymbol{\theta}^*) = 0$. This equation is similar to steepest descent, but the gradient is computed on the loss for a single sample. Common learning rates are: $\eta_i \coloneqq 1/i$; or $\eta_i \coloneqq (\tau_0 + i)^{-\kappa}$, for $\tau_0 \geq 0$ and $\kappa \in (0.5, 1]$. A variant of the algorithm (Polyak-Ruppert), with a faster convergence, uses as estimate the average:

$$\hat{\boldsymbol{\theta}}_i \coloneqq \frac{1}{i} \sum_{i=1}^{i} \boldsymbol{\theta}_i = \hat{\boldsymbol{\theta}}_{i-1} - \frac{1}{i}(\hat{\boldsymbol{\theta}}_{i-1} - \boldsymbol{\theta}_i)$$

We can now apply equation (49) to linear and logistic regression. Substituting the gradients we obtain two learning rules with the same form:

$$\mathbf{w}_{k+1} \coloneqq \mathbf{w}_k - \eta_k(\mu_i - y_i)\mathbf{x}_i$$

where $\mu_i$ is:

$$\begin{aligned} \mu_i &\coloneqq \mathbf{w}_i^T \mathbf{x}_i && \text{for linear regression} \\ \mu_i &\coloneqq \text{sigm}(\mathbf{w}_i^T \mathbf{x}_i) && \text{for logistic regression} \end{aligned}$$

Note that, in linear regression, $\mu_i$ is an estimated output, while in logistic regression, it is the estimated probability of a positive response.

If the gradient of the logistic model is approximated to $(\hat{y}_i - y_i)\mathbf{x}_i$, where $\hat{y}_i$ is the most probable binary output, both equations compare the estimated

and the actual response. The modified learning rule for binary classification is:

$$\mathbf{w}_{k+1} \coloneqq \mathbf{w}_k - \eta_i(\hat{y}_i - y_i)\mathbf{x}_i$$

in which, $(\hat{y}_i - y_i)$ only gives the sign of the update. This is the **perceptron algorithm**. It only works well with linearly separable data. In fact, under this assumption, there exists a finite upper bound to the total number of mistakes made by the algorithm. Also, convergence time is independent of the parameter $\eta_i$, so it is always set to 1. We can see that parameters are not updated if a sample is correctly classified. To conclude, if there exists an hyperplane, $\mathbf{w}^*$, that perfectly separates all points, the perceptron will eventually reach a null misclassification rate.

## 8.6 Generative vs discriminative classifiers

Some differences of these two type of classifiers are:

**Easy to fit** Generative classifiers are easy to fit, because usually we are able to write a closed form expression of the estimators. Instead, training a discriminative models requires to solve a nonlinear optimization problem.

**Fit classes separately** In a generative classifier, parameters of class conditional features are independent. So, if the model is modified by changing the number of classes, the only parameters to be found are the new ones. A discriminative classifier needs to be retrained completely.

**Handle missing data** Generative classifiers can handle randomly missing features or labels, in any combination. In fact, they can still give a distribution of outputs, marginalizing out the unobserved information.

**Assumptions** Generative classifiers make an higher number of assumptions. This can simplify the training process. However, if the assumptions are not valid, performances will be poor. Instead, discriminative classifiers directly provide a decision rule, which could be common to various type of models.

**Input preprocessing** In a discriminative classifier, inputs can be transformed arbitrarily with a function $\boldsymbol{\phi}(\mathbf{x})$. The same modification would me much more difficult to introduce in a generative classifier.

All classifiers behave badly if the data is too sparse. As already noticed, this is particularly true for high dimensional inputs. The most common solution is to project the inputs in a lower dimensional space using a linear function: $\mathbf{z} \coloneqq \mathbf{W}\mathbf{x}$. A method, called **Fisher's linear discriminant analysis** (FLDA), differs from other projections because it chooses $\mathbf{W}$ such that the low dimensional vectors can be well separated by a Gaussian generative

model. This technique requires that the dimension $L$ of the low dimensional space is $L \leq C - 1$. So, for binary classification, each vector is projected to a scalar $z = \mathbf{w}^T \mathbf{x}$. We will report here just the final result for this case, that is:

$$\mathbf{w} \coloneqq \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

where $\mathbf{S}_W$ is the within-class scatter matrix,

$$\mathbf{S}_W \coloneqq \sum_{i:y_i=1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^T + \sum_{i:y_i=2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^T$$

and $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$ are the class-conditional means:

$$\boldsymbol{\mu}_1 \coloneqq \frac{1}{N_1} \sum_{i:y_i=1} \mathbf{x}_i \qquad \boldsymbol{\mu}_2 \coloneqq \frac{1}{N_2} \sum_{i:y_i=2} \mathbf{x}_i$$