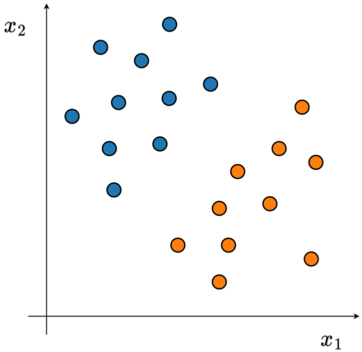
Linear classifiers

Classification problem

For the **classification** task, the target variable y to predict is **discrete**, *i.e.* for K classes (or labels), y can only take K distinct values.

For example, we can have $y \in \{0, \dots, K-1\}$ or $y \in \{-1, +1\}$.

We will first focus on the **binary classification** problem, where $K = 2$.



Linear model for classification

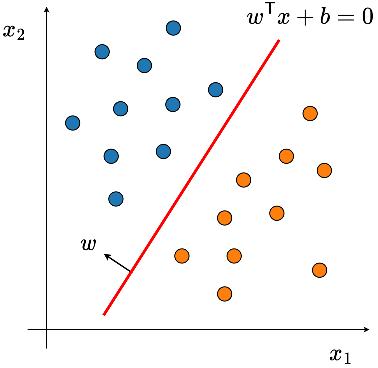
Our model will still be a **linear function**, but also with an **activation function** (also called *discriminant function*):

$$y = f(\boldsymbol{x}; \boldsymbol{w}, b) = \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b)$$

For example with:

$$\sigma(x) = \operatorname{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

this is called the **perceptron** model.



Logistic regression

If we use the **sigmoid** function (or *logistic* function) as the activation function, the model is called **logistic regression**.

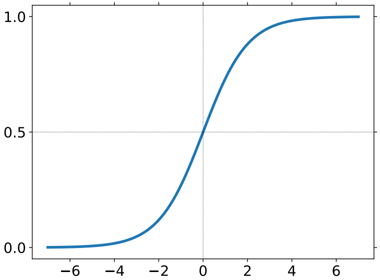
$$y = \sigma(z) \quad \text{with} \quad z = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b \quad \text{and} \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

The scalar z before the activation is called a **logit**.

The output of the model can be interpreted as a **probability**.

We can take the final classification decision by thresholding:

$$\begin{cases} \text{class 1} & \text{if } y \geq 0.5 \\ \text{class 0} & \text{otherwise} \end{cases}$$



Maximum likelihood for logistic regression

With logistic regression formulation, we can write:

$$p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}, b) = \underbrace{\sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b)}_{\hat{y}_i}{}^{y_i} (1 - \underbrace{\sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b)}_{\hat{y}_i})^{(1-y_i)}$$

We can apply maximum likelihood (ML) estimation on the training data:

$$\boldsymbol{w}_{\mathrm{ML}}, b_{\mathrm{ML}} = \arg\max_{\boldsymbol{w}, b} \ \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, b)$$

Assuming *i.i.d.* samples, we obtain:

$$\begin{aligned}
\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, b) &= \log \prod_{i=1}^{N} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}, b) = \sum_{i=1}^{N} \log p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}, b) \\
&= \sum_{i=1}^{N} \log \left((\hat{y}_i)^{y_i} (1 - \hat{y}_i)^{(1-y_i)} \right) \\
&= \sum_{i=1}^{N} \left(y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right)
\end{aligned}$$

Binary cross-entropy

Following the maximum likelihood principle, we typically want to **minimize** the loss function:

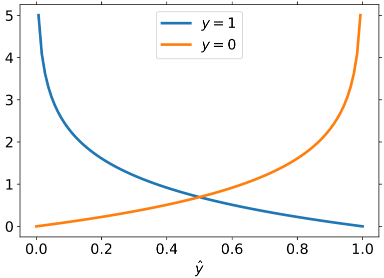
$$\mathcal{L}_{\mathrm{CE}}(\boldsymbol{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right)$$

where $\hat{y}_i = \sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_i + b)$ is the prediction of the model.

This loss function is called **binary cross-entropy**.

This loss function is **convex**. But unlike linear regression, there is **no closed-form analytical solution** to this minimization problem.

$\rightarrow$ Use **gradient descent**.



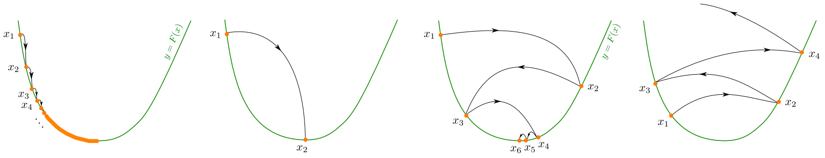
Gradient descent

Principle

- Update the parameters by taking steps in the opposite direction of the gradient of the function to minimize:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \gamma \nabla \mathcal{L}(\boldsymbol{w}^{(t)}),\ \gamma > 0.$$

- The hyperparameter γ is called the **learning rate** (or *step size*).



Small step size Slow convergence	Good step size Fast convergence	Large step size Slow convergence	Too large step size Divergence

Gradient descent for binary cross-entropy

Recall the binary cross-entropy loss:

$$\mathcal{L}_{\mathrm{CE}}(\boldsymbol{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log \hat{y_i} + (1 - y_i)\log(1 - \hat{y_i})\right)$$

where $\hat{y_i} = \sigma(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_i + b)$ is the prediction of the model.

By successive applications of the chain rule, we find that:

$$\nabla_{\boldsymbol{w}}\,\mathcal{L}_{\mathrm{CE}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y_i} - y_i)\,\boldsymbol{x}_i$$

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y_i} - y_i).$$

Extension to multiclass classification

Logistic regression can be extended to the **multiclass** setting (K classes):

- Output targets are encoded into one-hot vectors $\boldsymbol{y} \in \mathbb{R}^K$ (for example: $\boldsymbol{y} = [0, 1, 0, 0]^\mathsf{T}$ for class 2 out of 4)
- Learnable parameters are $\boldsymbol{W} \in \mathbb{R}^{D \times K}$ and $\boldsymbol{b} \in \mathbb{R}^K$
- Predictions are $\hat{\boldsymbol{y}} = \sigma(\boldsymbol{W}^\mathsf{T}\boldsymbol{x} + \boldsymbol{b})$, where σ is applied component-wise
- σ is the $\mathrm{softmax}$ function: $\sigma(\boldsymbol{x})_i = \dfrac{\exp x_i}{\sum_{j=1}^{K} \exp x_j}$
- The loss is the **categorical cross-entropy** function:

$$\mathcal{L}_{\mathrm{CCE}}(\boldsymbol{W}, \boldsymbol{b}) = -\frac{1}{N} \sum_{i=1}^{N} \ell(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) \quad \text{with} \quad \ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{j=1}^{K} y_j \log \hat{y}_j$$