# Perceptron and Logistic Regression

Milan Straka

October 17, 2022

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

# Binary Classification

Binary classification is a classification in two classes.

The simplest way to evaluate classification is **accuracy**, which is the ratio of input examples that were classified correctly – i.e., where the predicted class and the target class match.

To extend linear regression to binary classification, we might seek a **threshold** and then classify an input as negative/positive depending on whether $y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w} + b$ is smaller/larger than a given threshold.

Zero value is usually used as the threshold, both because of symmetry and also because the **bias** parameter acts as a trainable threshold anyway.

The set of points with prediction 0 is called a **decision boundary**.

# Binary Classification

- Consider two points on the decision boundary. Because $y(\boldsymbol{x}_1; \boldsymbol{w}) = y(\boldsymbol{x}_2; \boldsymbol{w})$, we have $(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{w} = 0$, and so $\boldsymbol{w}$ is orthogonal to every vector on the decision surface – $\boldsymbol{w}$ is a **normal** of the boundary.
- Consider $\boldsymbol{x}$ and let $\boldsymbol{x}_\perp$ be orthogonal projection of $\boldsymbol{x}$ to the boundary, so we can write $\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$. Multiplying both sides by $\boldsymbol{w}^T$ and adding $b$, we get that the distance of $\boldsymbol{x}$ to the boundary is $r = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|}$.
- The distance of the decision boundary from the origin is therefore $\frac{|b|}{\|\boldsymbol{w}\|}$.

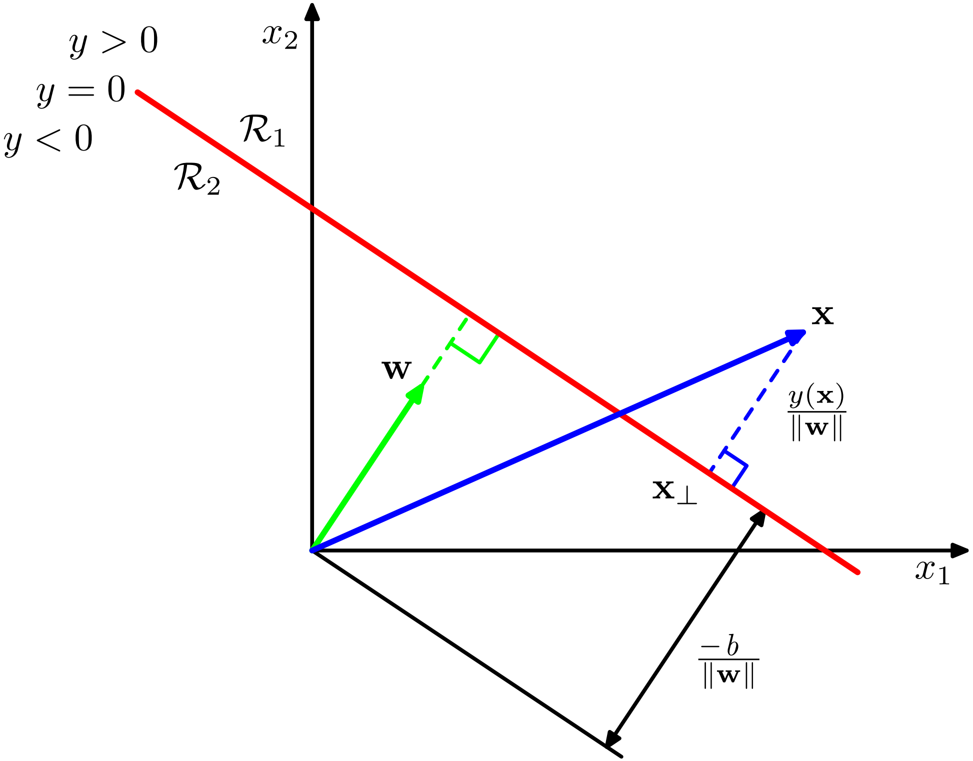


*Figure 4.1 of Pattern Recognition and Machine Learning.*

# Perceptron

The perceptron algorithm is probably the oldest one for training weights of a binary classification. Assuming the target value $t \in \{-1, +1\}$, the goal is to find weights $\boldsymbol w$ such that for all train data,

$$\operatorname{sign}(y(\boldsymbol x_i; \boldsymbol w)) = \operatorname{sign}(\boldsymbol x_i^T \boldsymbol w) = t_i,$$

or equivalently,

$$t_i y(\boldsymbol x_i; \boldsymbol w) = t_i \boldsymbol x_i^T \boldsymbol w > 0.$$

Note that a set is called **linearly separable**, if there exists a weight vector $\boldsymbol w$ such that the above equation holds.

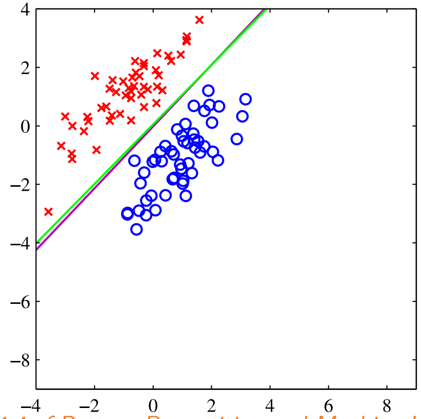


*Figure 4.4 of Pattern Recognition and Machine Learning.*

# Perceptron

The perceptron algorithm was invented by Rosenblatt in 1958.

**Input**: Linearly separable dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t} \in \{-1, +1\}^N$).
**Output**: Weights $\boldsymbol{w} \in \mathbb{R}^D$ such that $t_i \boldsymbol{x}_i^T \boldsymbol{w} > 0$ for all $i$.

```
• $\boldsymbol{w} \leftarrow \mathbf{0}$
• until all examples are classified correctly, process example $i$:
    ◦ $y \leftarrow \boldsymbol{x}_i^T \boldsymbol{w}$
    ◦ if $t_i y \leq 0$ (incorrectly classified example):
        ▪ $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_i \boldsymbol{x}_i$
```

We will prove that the algorithm always arrives at some correct set of weights $\boldsymbol{w}$ if the training set is linearly separable.

# Perceptron as SGD

Consider the main part of the perceptron algorithm:

- $y \leftarrow \boldsymbol{x}_i^T \boldsymbol{w}$
- if $t_i y \leq 0$ (incorrectly classified example):
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_i \boldsymbol{x}_i$

We can derive the algorithm using on-line gradient descent, using the following loss function

$$L(y(\boldsymbol{x}; \boldsymbol{w}), t) \stackrel{\text{def}}{=} \begin{cases} -t\boldsymbol{x}^T\boldsymbol{w} & \text{if } t\boldsymbol{x}^T\boldsymbol{w} \leq 0 \\ 0 & \text{otherwise} \end{cases} = \max(0, -t\boldsymbol{x}^T\boldsymbol{w}) = \operatorname{ReLU}(-t\boldsymbol{x}^T\boldsymbol{w}).$$

In this specific case, the value of the learning rate does not influence the model behavior (different learning rates would produce models with same predictions), because multiplying $\boldsymbol{w}$ by a constant does not change the prediction and the loss derivative does not depend on $\boldsymbol{w}$.

Note that the *second* condition is crucial; the first holds also for logistic regression, but the learning rate matters there.

# Perceptron Example

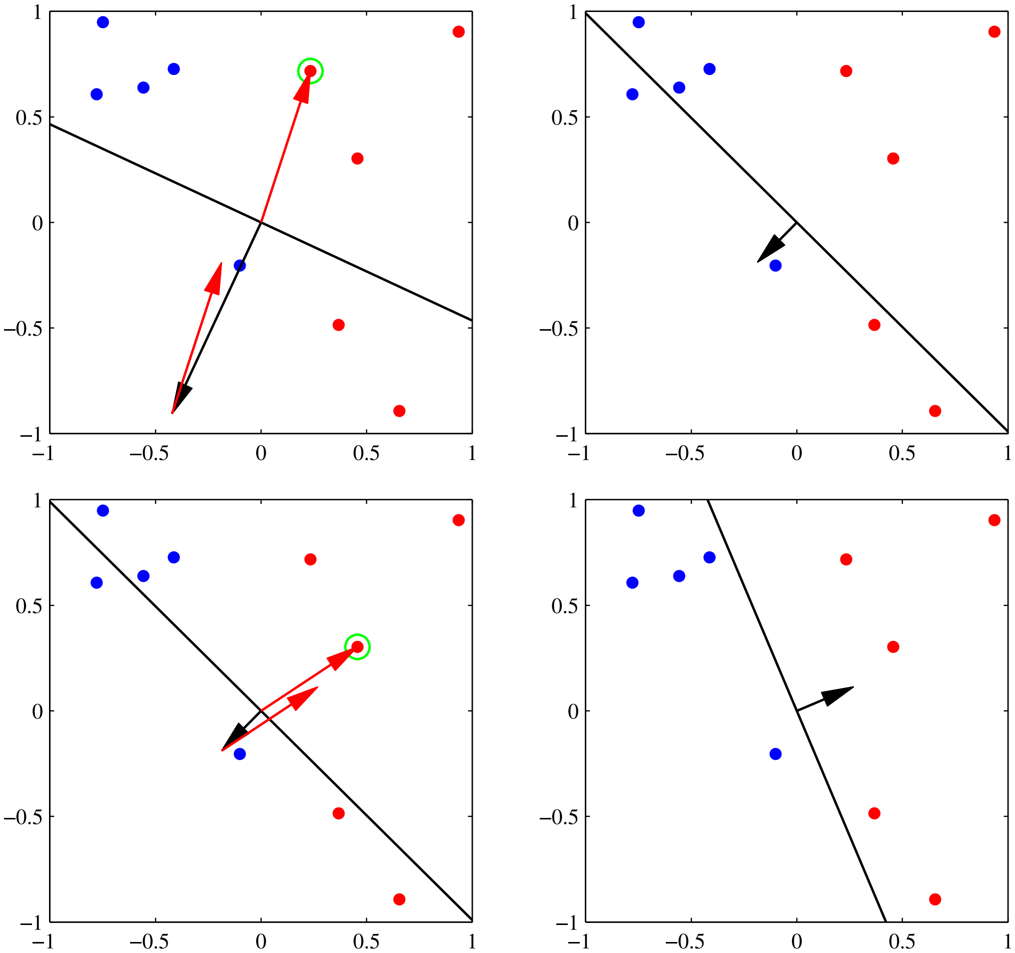


*Figure 4.7 of Pattern Recognition and Machine Learning.*

# Proof of Perceptron Convergence

Let $\boldsymbol{w}_*$ be some weights correctly classifying (separating) the training data, and let $\boldsymbol{w}_k$ be the weights after $k$ nontrivial updates of the perceptron algorithm, with $\boldsymbol{w}_0$ being 0.

We will prove that the angle $\alpha$ between $\boldsymbol{w}_*$ and $\boldsymbol{w}_k$ decreases at each step. Note that

$$\cos(\alpha) = \frac{\boldsymbol{w}_*^T \boldsymbol{w}_k}{\|\boldsymbol{w}_*\| \cdot \|\boldsymbol{w}_k\|}.$$

# Proof of Perceptron Convergence

Assume that the maximum norm of any training example $\|\boldsymbol{x}\|$ is bounded by $R$, and that $\gamma$ is the minimum margin of $\boldsymbol{w}_*$, so for each training example $(\boldsymbol{x}, t)$, $t\boldsymbol{x}^T\boldsymbol{w}_* \geq \gamma.$

First consider the dot product of $\boldsymbol{w}_*$ and $\boldsymbol{w}_k$:

$$\boldsymbol{w}_*^T\boldsymbol{w}_k = \boldsymbol{w}_*^T(\boldsymbol{w}_{k-1} + t_k\boldsymbol{x}_k) \geq \boldsymbol{w}_*^T\boldsymbol{w}_{k-1} + \gamma.$$

By iteratively applying this equation, we get

$$\boldsymbol{w}_*^T\boldsymbol{w}_k \geq k\gamma.$$

Now consider the length of $\boldsymbol{w}_k$:

$$\|\boldsymbol{w}_k\|^2 = \|\boldsymbol{w}_{k-1} + t_k\boldsymbol{x}_k\|^2 = \|\boldsymbol{w}_{k-1}\|^2 + 2t_k\boldsymbol{x}_k^T\boldsymbol{w}_{k-1} + \|\boldsymbol{x}_k\|^2.$$

Because $\boldsymbol{x}_k$ was misclassified, we know that $t_k\boldsymbol{x}_k^T\boldsymbol{w}_{k-1} \leq 0$, so $\|\boldsymbol{w}_k\|^2 \leq \|\boldsymbol{w}_{k-1}\|^2 + R^2$. When applied iteratively, we get $\|\boldsymbol{w}_k\|^2 \leq k \cdot R^2$.

# Proof of Perceptron Convergence

Putting everything together, we get

$$\cos(\alpha) = \frac{\boldsymbol{w}_*^T \boldsymbol{w}_k}{\|\boldsymbol{w}_*\| \cdot \|\boldsymbol{w}_k\|} \geq \frac{k\gamma}{\sqrt{kR^2}\|\boldsymbol{w}_*\|}.$$

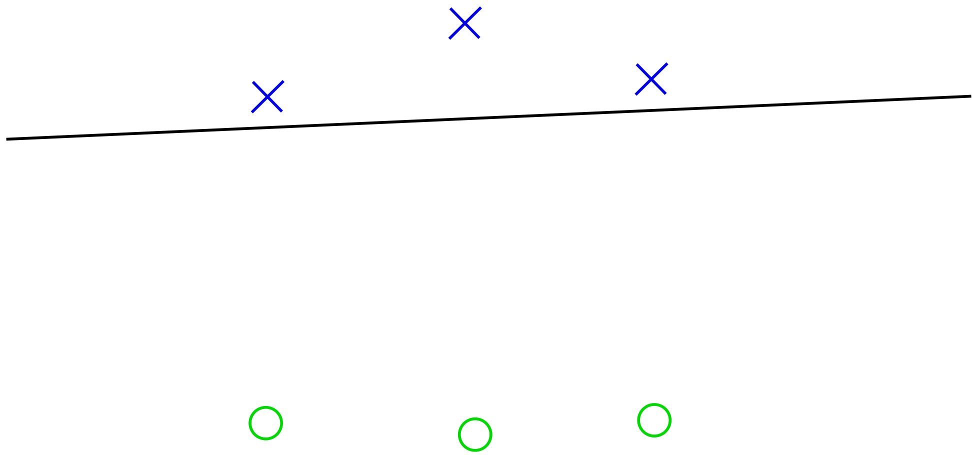
Therefore, the $\cos(\alpha)$ increases during every update. Because the value of $\cos(\alpha)$ is at most one, we can compute the upper bound on the number of steps when the algorithm converges as

$$1 \geq \frac{\sqrt{k}\gamma}{\sqrt{R^2}\|\boldsymbol{w}_*\|} \text{ or } k \leq \frac{R^2\|\boldsymbol{w}_*\|^2}{\gamma^2}.$$

# Perceptron Issues

Perceptron has several drawbacks:

- If the input set is not linearly separable, the algorithm never finishes.
- The algorithm performs only prediction, it is not able to return the probabilities of predictions.
- Most importantly, Perceptron algorithm finds *some* solution, not necessarily a good one, because once it finds some, it cannot perform any more updates.



# Common Probability Distributions

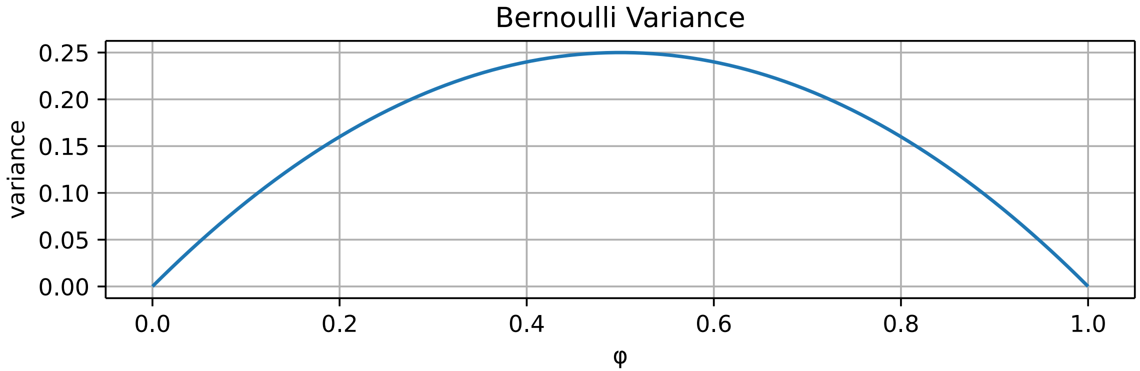
## Bernoulli Distribution

The Bernoulli distribution is a distribution over a binary random variable. It has a single parameter $\varphi \in [0, 1]$, which specifies the probability that the random variable is equal to 1.

$$P(x) = \varphi^x (1-\varphi)^{1-x}$$

$$\mathbb{E}[x] = \varphi$$

$$\mathrm{Var}(x) = \varphi(1-\varphi)$$



## Categorical Distribution

Extension of the Bernoulli distribution to random variables taking one of $K$ different discrete outcomes. It is parametrized by $\boldsymbol{p} \in [0,1]^K$ such that $\sum_{i=0}^{K-1} p_i = 1$.

We represent outcomes as vectors $\in \{0,1\}^K$ in the **one-hot encoding**. Therefore, an outcome $x \in \{0, 1, \ldots, K-1\}$ is represented as a vector

$$\mathbf{1}_x \stackrel{\text{def}}{=} \big([i = x]\big)_{i=0}^{K-1} = \big(\underbrace{0, \ldots, 0}_{x}, 1, \underbrace{0, \ldots, 0}_{K-x-1}\big).$$

The outcome probability, mean, and variance are very similar to the Bernoulli distribution.

$$P(\boldsymbol{x}) = \prod\nolimits_{i=0}^{K-1} p_i^{x_i}$$

$$\mathbb{E}[x_i] = p_i$$

$$\mathrm{Var}(x_i) = p_i(1 - p_i)$$

## Self Information

Amount of **surprise** when a random variable is sampled.

- Should be zero for events with probability 1.
- Less likely events are more surprising.
- Independent events should have **additive** information.

$$I(x) \stackrel{\text{def}}{=} -\log P(x) = \log \frac{1}{P(x)}$$

# Information Theory

## Entropy

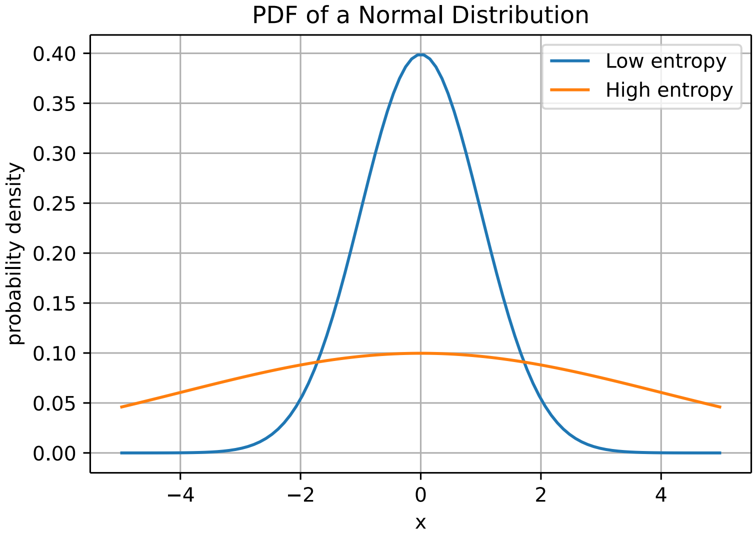
Amount of **surprise** in the whole distribution.

$$H(P) \overset{\text{def}}{=} \mathbb{E}_{\mathrm{x} \sim P}[I(x)] = -\mathbb{E}_{\mathrm{x} \sim P}[\log P(x)]$$

- for discrete $P$: $H(P) = -\sum_x P(x) \log P(x)$
- for continuous $P$: $H(P) = -\int P(x) \log P(x)\,\mathrm{d}x$

Because $\lim_{x \to 0} x \log x = 0$, for $P(x) = 0$ we consider $P(x) \log P(x)$ to be zero.

Note that in the continuous case, the continuous entropy (also called *differential entropy*) has slightly different semantics, for example, it can be negative.



For binary logarithms, the entropy is measured in **bits**.
However, from now on, all logarithms are *natural logarithms* with base *e* (and then the entropy is measured in units called **nats**).

# Information Theory

## Cross-Entropy

distribuce, kterou očekáváme

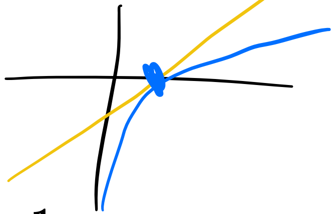
$$H(P, Q) \stackrel{\text{def}}{=} -\mathbb{E}_{\mathrm{x} \sim P}[\log Q(x)]$$

distribuce, kterou samplujeme

## Gibbs Inequality

- $H(P, Q) \geq H(P) = H(P,P)$
- $H(P) = H(P, Q) \Leftrightarrow P = Q$

Proof: Consider $H(P) - H(P, Q) = \sum_x P(x) \log \frac{Q(x)}{P(x)}$.



Using the fact that $\log x \leq (x - 1)$ with equality only for $x = 1$, we get

$$\sum_x P(x) \log \frac{Q(x)}{P(x)} \leq \sum_x P(x) \left( \frac{Q(x)}{P(x)} - 1 \right) = \underbrace{\sum_x Q(x)}_{1} - \underbrace{\sum_x P(x)}_{1} = 0.$$

For the equality to hold, $\frac{Q(x)}{P(x)}$ must be 1 for all $x$, i.e., $P = Q$.

# Information Theory

## Corollary of the Gibbs inequality

For a categorical distribution with $n$ outcomes, $H(P) \leq \log n$, because for $Q(x) = 1/n$ we get $H(P) \leq H(P, Q) = -\sum_x P(x) \log Q(x) = \log n$.

## Nonsymmetry

Note that generally $H(P, Q) \neq H(Q, P)$.

## Kullback-Leibler Divergence (KL Divergence)

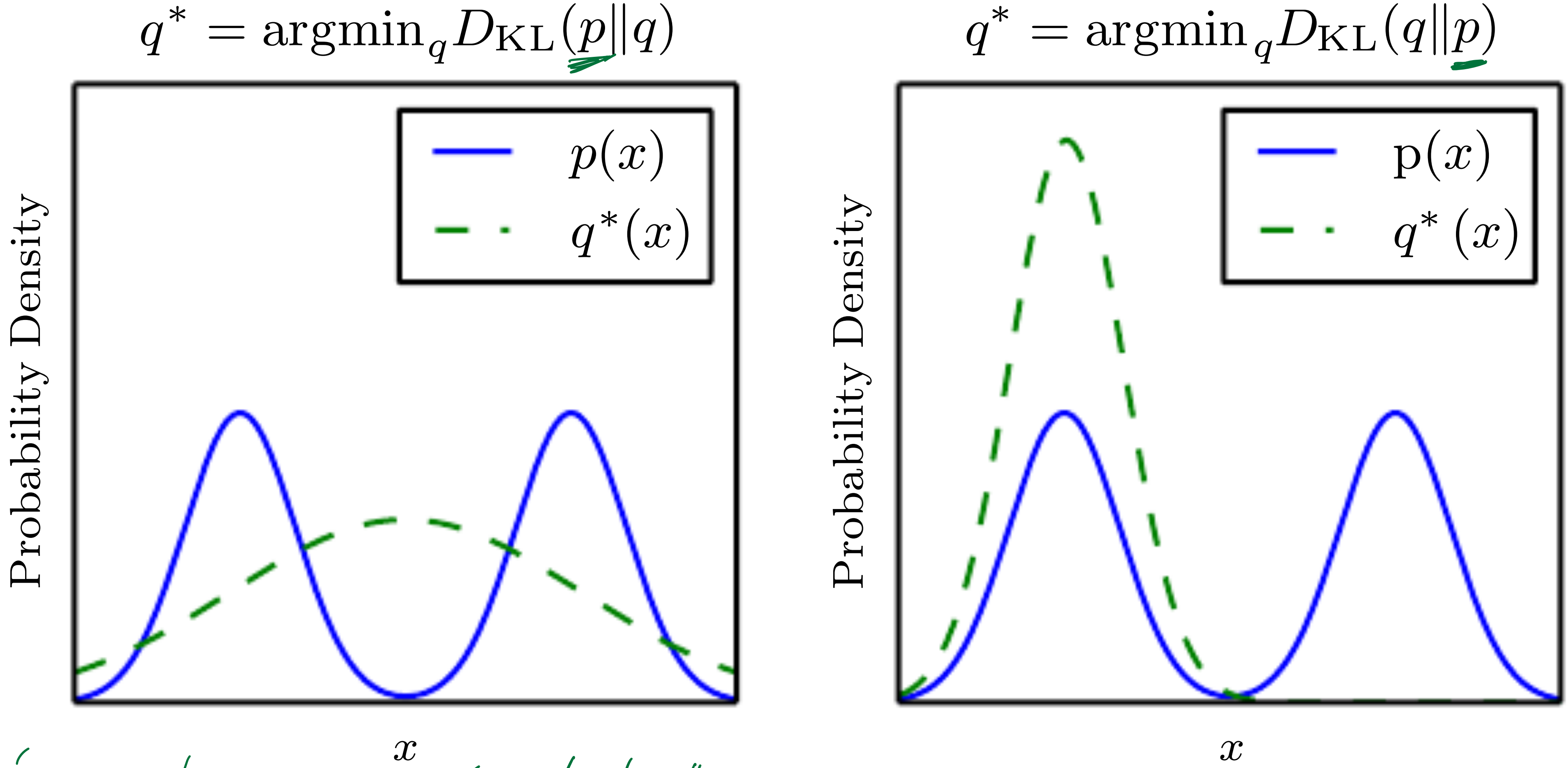
Sometimes also called **relative entropy**.

$$D_{\mathrm{KL}}(P\|Q) \overset{\text{def}}{=} H(P,Q) - H(P) = \mathbb{E}_{\mathrm{x}\sim P}[\log P(x) - \log Q(x)]$$

- consequence of Gibbs inequality: $D_{\mathrm{KL}}(P\|Q) \geq 0$, $D_{\mathrm{KL}}(P\|Q) = 0$ iff $P = Q$
- generally $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$

⇒ proto to není metrika

# Nonsymmetry of KL Divergence



$$q^* = \operatorname{argmin}_q D_{\mathrm{KL}}(\underline{p}\|q)$$

$$q^* = \operatorname{argmin}_q D_{\mathrm{KL}}(q\|\underline{p})$$

Figure 3.6 of "Deep Learning" book, https://www.deeplearningbook.org

Známe p, chceme q, co má „jeden kopec" a je nejblíž té p distribuci

## Normal (or Gaussian) Distribution

Distribution over real numbers, parametrized by a mean $\mu$ and variance $\sigma^2$:

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

For standard values $\mu = 0$ and $\sigma^2 = 1$ we get $\mathcal{N}(x; 0, 1) = \sqrt{\frac{1}{2\pi}} e^{-\frac{x^2}{2}}$.

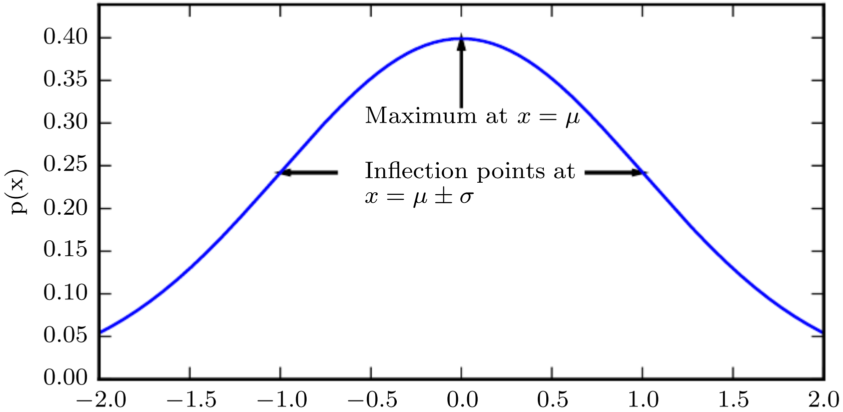


*Figure 3.1 of "Deep Learning" book, https://www.deeplearningbook.org.*

# Why Normal Distribution

## Central Limit Theorem

The sum of independent identically distributed random variables with finite variance converges to normal distribution.

## Principle of Maximum Entropy

Given a set of constraints, a distribution with maximal entropy fulfilling the constraints can be considered the most general one, containing as little additional assumptions as possible.

Considering distributions with a given mean and variance, it can be proven (using variational inference) that such a distribution with **maximum entropy** is exactly the normal distribution.

# Maximum Likelihood Estimation

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be training data drawn independently from the data-generating distribution $p_{\text{data}}$.

We denote the **empirical data distribution** as $\hat{p}_{\text{data}}$, where

$$\hat{p}_{\text{data}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \frac{\big|\{i : \boldsymbol{x}_i = \boldsymbol{x}\}\big|}{N}.$$

Let $p_{\text{model}}(\mathbf{x}; \boldsymbol{w})$ be a family of distributions.

- If the weights are fixed, $p_{\text{model}}(\mathbf{x}; \boldsymbol{w})$ is a probability distribution.
- If we instead consider the fixed training data $\boldsymbol{X}$, then

tohle už se nemusí vysčítat na 1 ?

$$L(\boldsymbol{w}) = p_{\text{model}}(\boldsymbol{X}; \boldsymbol{w}) = \prod\nolimits_{i=1}^{N} p_{\text{model}}(\boldsymbol{x}_i; \boldsymbol{w})$$

is called the **likelihood**. Note that even if the value of the likelihood is in range $[0, 1]$, it is not a probability, because the likelihood is not a probability distribution.

# Maximum Likelihood Estimation

Let $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be training data drawn independently from the data-generating distribution $p_\text{data}$. We denote the empirical data distribution as $\hat{p}_\text{data}$ and let $p_\text{model}(\mathbf{x}; \boldsymbol{w})$ be a family of distributions.

The **maximum likelihood estimation** of $\boldsymbol{w}$ is:

$$\begin{aligned}
\boldsymbol{w}_\text{MLE} &= \arg\max_{\boldsymbol{w}} p_\text{model}(\boldsymbol{X}; \boldsymbol{w}) = \arg\max_{\boldsymbol{w}} \prod\nolimits_{i=1}^N p_\text{model}(\boldsymbol{x}_i; \boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum\nolimits_{i=1}^N -\log p_\text{model}(\boldsymbol{x}_i; \boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_\text{data}} [-\log p_\text{model}(\boldsymbol{x}; \boldsymbol{w})] \\
&= \arg\min_{\boldsymbol{w}} H(\underbrace{\hat{p}_\text{data}(\mathbf{x})}_{\text{data}}, \underbrace{p_\text{model}(\mathbf{x}; \boldsymbol{w})}_{\text{co model vrací}}) \\
&= \arg\min_{\boldsymbol{w}} D_\text{KL}(\hat{p}_\text{data}(\mathbf{x}) \| p_\text{model}(\mathbf{x}; \boldsymbol{w})) + H(\hat{p}_\text{data}(\mathbf{x}))
\end{aligned}$$

ztrátové funkce chceme obecně minimalizovat, proto to obrátíme

hledat max index u f(x) a log(f(x)) je stejné!

# Maximum Likelihood Estimation

MLE can be easily generalized to the conditional case, where our goal is to predict $t$ given $\boldsymbol{x}$:

$$\begin{aligned}
\boldsymbol{w}_{\text{MLE}} &= \arg\max_{\boldsymbol{w}} p_{\text{model}}(\boldsymbol{t}|\boldsymbol{X};\boldsymbol{w}) = \arg\max_{\boldsymbol{w}} \prod_{i=1}^{N} p_{\text{model}}(t_i|\boldsymbol{x}_i;\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} -\log p_{\text{model}}(t_i|\boldsymbol{x}_i;\boldsymbol{w}) \quad \text{NLL, negative log likelihood} \\
&= \arg\min_{\boldsymbol{w}} \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\hat{p}_{\text{data}}}[-\log p_{\text{model}}(t|\boldsymbol{x};\boldsymbol{w})] \\
&= \arg\min_{\boldsymbol{w}} H(\hat{p}_{\text{data}}(\mathrm{t}|\mathbf{x}), p_{\text{model}}(\mathrm{t}|\mathbf{x};\boldsymbol{w})) \quad \text{CE, cross-entropy} \\
&= \arg\min_{\boldsymbol{w}} D_{\text{KL}}(\hat{p}_{\text{data}}(\mathrm{t}|\mathbf{x})\|p_{\text{model}}(\mathrm{t}|\mathbf{x};\boldsymbol{w})) + H(\hat{p}_{\text{data}}(\mathrm{t}|\mathbf{x})) \quad \text{KL divergence}
\end{aligned}$$

where the conditional entropy is defined as $H(\hat{p}_{\text{data}}) = \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\hat{p}_{\text{data}}}[-\log(\hat{p}_{\text{data}}(t|\boldsymbol{x};\boldsymbol{w}))]$ and the conditional cross-entropy as $H(\hat{p}_{\text{data}}, p_{\text{model}}) = \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\hat{p}_{\text{data}}}[-\log(p_{\text{model}}(t|\boldsymbol{x};\boldsymbol{w}))]$.

The resulting *loss function* is called **negative log-likelihood** (**NLL**), or **cross-entropy**, or **Kullback-Leibler divergence**.

# Properties of Maximum Likelihood Estimation

Assume that the true data-generating distribution $p_\text{data}$ lies within the model family $p_\text{model}(\cdot; \boldsymbol w)$. Furthermore, assume there exists a unique $\boldsymbol w_{p_\text{data}}$ such that $p_\text{data} = p_\text{model}(\cdot; \boldsymbol w_{p_\text{data}})$.

- MLE is a *consistent* estimator. If we denote $\boldsymbol w_m$ to be the parameters found by MLE for a training set with $m$ examples generated by the data-generating distribution, then $\boldsymbol w_m$ converges in probability to $\boldsymbol w_{p_\text{data}}$.

  Formally, for any $\varepsilon > 0$, $P(\|\boldsymbol w_m - \boldsymbol w_{p_\text{data}}\| > \varepsilon) \rightarrow 0$ as $m \rightarrow \infty$.

- MLE is in a sense the most *statistically efficient*. For any consistent estimator, let us consider the average distance of $\boldsymbol w_m$ and $\boldsymbol w_{p_\text{data}}$: $\mathbb{E}_{\mathbf{x}_1,\ldots,\mathbf{x}_m \sim p_\text{data}}\left[\|\boldsymbol w_m - \boldsymbol w_{p_\text{data}}\|^2\right]$.
  It can be shown (Rao 1945, Cramér 1946) that no consistent estimator has lower mean squared error than the maximum likelihood estimator.

Therefore, for reasons of consistency and efficiency, maximum likelihood is often considered the preferred estimator for machine learning.

# Logistic Regression

An extension of perceptron, which models the conditional probabilities of $p(C_0|\boldsymbol{x})$ and of $p(C_1|\boldsymbol{x})$. Logistic regression can in fact handle also more than two classes, which we will see in the next lecture.

Logistic regression employs the following parametrization of the conditional class probabilities:

$C_0, C_1$ jsou klasifikační třídy.

$$\begin{aligned}p(C_1|\boldsymbol{x}) &= \sigma(\boxed{\boldsymbol{x}^T\boldsymbol{w} + b})\\ p(C_0|\boldsymbol{x}) &= 1 - p(C_1|\boldsymbol{x}),\end{aligned}$$

lineární predikce

rozšlapávací funkce

where $\sigma$ is a **sigmoid function**

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

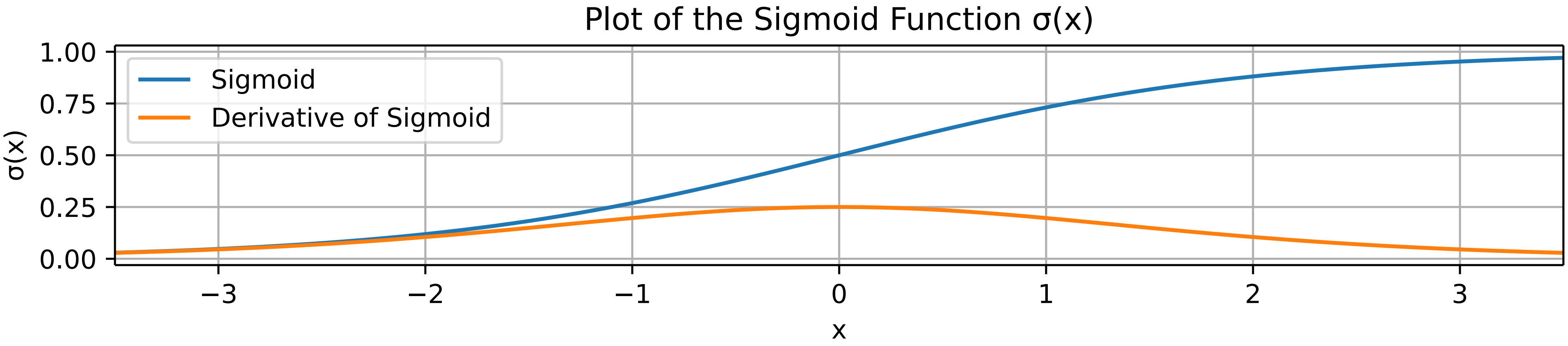
It can be trained using the SGD algorithm.

# Sigmoid Function

The sigmoid function has values in range $(0, 1)$, is monotonically increasing and it has a derivative of $\frac{1}{4}$ at $x = 0$.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$$



# Logistic Regression

We denote the output of the “linear part” of the logistic regression as

$$\bar{y}(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w},$$

⇐ výstup lineární části

and the overall prediction as

$$y(\boldsymbol{x}; \boldsymbol{w}) = \sigma(\bar{y}(\boldsymbol{x}; \boldsymbol{w})) = \sigma(\boldsymbol{x}^T \boldsymbol{w}).$$

# Logistic Regression

The logistic regression output $y(\boldsymbol{x}; \boldsymbol{w})$ models the probability of class $C_1$, $p(C_1|\boldsymbol{x})$.

To give some meaning to the output of the linear part $\bar{y}(\boldsymbol{x}; \boldsymbol{w})$, starting with

$$p(C_1|\boldsymbol{x}) = \sigma(\bar{y}(\boldsymbol{x}; \boldsymbol{w})) = \frac{1}{1 + e^{-\bar{y}(\boldsymbol{x}; \boldsymbol{w})}},$$

we arrive at

← poměr pravděpodobností náležitosti do dané třídy.

$$\bar{y}(\boldsymbol{x}; \boldsymbol{w}) = \log\left(\frac{p(C_1|\boldsymbol{x})}{1 - p(C_1|\boldsymbol{x})}\right) = \log\left(\frac{p(C_1|\boldsymbol{x})}{p(C_0|\boldsymbol{x})}\right),$$

which is called a **logit** and it is a logarithm of odds of the probabilities of the two classes.

# Logistic Regression

To train the logistic regression, we use MLE (the maximum likelihood estimation). Its application is straightforward, given that $p(C_1|\boldsymbol{x};\boldsymbol{w})$ is directly the model output $y(\boldsymbol{x};\boldsymbol{w})$.

Therefore, the loss for a minibatch $\mathbb{X} = \{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2), \ldots, (\boldsymbol{x}_N, t_N)\}$ is

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_i -\log(\underline{p(C_{t_i}|\boldsymbol{x}_i;\boldsymbol{w})}).$$

$$\begin{cases} y & : t_i = 1 \\ 1-y & : t_i = 0 \end{cases}$$

```
Input: Input dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t} \in \{0, +1\}^N$), learning rate $\alpha \in \mathbb{R}^+$.

• $\boldsymbol{w} \leftarrow \mathbf{0}$ or we initialize $\boldsymbol{w}$ randomly
• until convergence (or patience runs out), process a minibatch of examples $\mathbb{B}$:
    ○ $\boldsymbol{g} \leftarrow \frac{1}{|\mathbb{B}|}\sum_{i\in\mathbb{B}} \nabla_{\boldsymbol{w}} \underline{\Big(-\log\big(p(C_{t_i}|\boldsymbol{x}_i;\boldsymbol{w})\big)\Big)}$   nutno spočítat derivaci
    ○ $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\boldsymbol{g}$
```

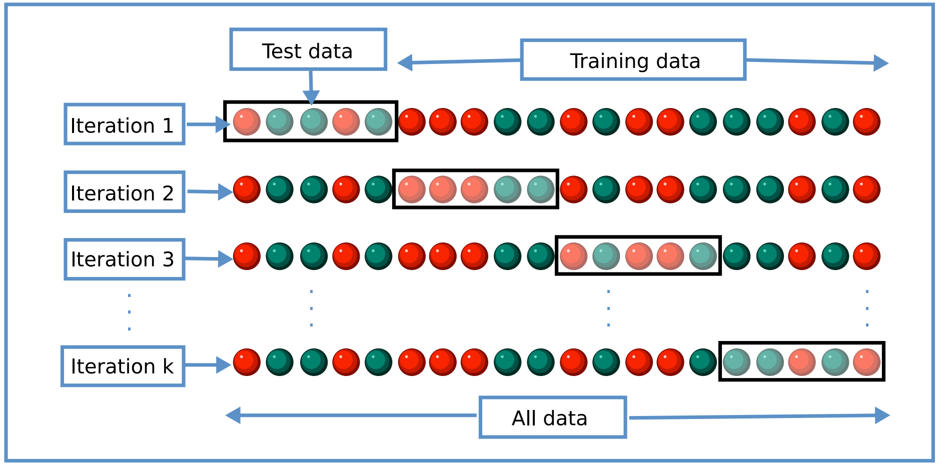
# Cross-Validation

We already discussed a **train set** and a **test set**. Given that the main goal of machine learning is to perform well on unseen data, the test set must not be used during training or hyperparameter selection. Ideally, it is hidden to us altogether.

Therefore, to evaluate a machine learning model (for example to select model architecture, features, or hyperparameter value), we normally need the **validation** or a **development** set.

However, using a single development set might give us noisy results. To obtain less noisy results (i.e., with smaller variance), we can use **cross-validation**.

In cross-validation, we choose multiple validation sets from the training data, and for each one, we train a model on the rest of the training data and evaluate on the chosen validation sets. A commonly used strategy to choose the validation sets is called **k-fold cross-validation**. Here the training set is partitioned into $k$ subsets of approximately the same size, and each subset takes turns to play a role of a validation set.



*https://commons.wikimedia.org/wiki/File:K-fold_cross_validation_EN.svg*

# Cross-Validation

An extreme case of the **k-fold cross-validation** is **leave-one-out cross-validation**, where every element is considered a separate validation set.

Computing leave-one-out cross-validation is usually extremely inefficient for larger training sets, but in the case of linear regression with $L^2$-regularization, it can be evaluated efficiently.

- If you are interested, see:

  *Ryan M. Rifkin and Ross A. Lippert: Notes on Regularized Least Square*
  http://cbcl.mit.edu/publications/ps/MIT-CSAIL-TR-2007-025.pdf

- Implemented by `sklearn.linear_model.RidgeCV`.