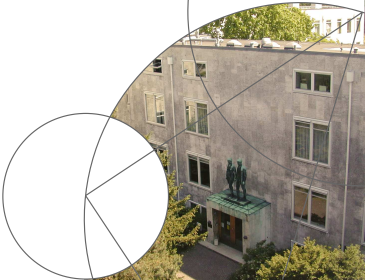
Faculty of Science

Linear Classification

Machine Learning

Christian Igel
Department of Computer Science



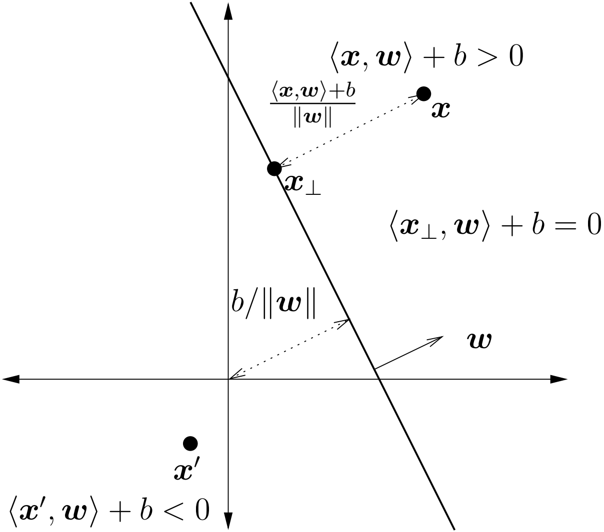
Outline

Outline

1. Logistic Regression
2. Linear Classification and Margins
3. Perceptron Learning
4. Convergence of Perceptron Learning
5. Summary

Linear functions

$f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle + b$



Decision functions

- Classification assigns an input $x \in \mathcal{X}$ to one of a finite set of classes $\mathcal{Y} = \{\mathcal{C}_1, \dots, \mathcal{C}_m\}$, $2 \leq m$.

- One approach is to learn discrimination functions $\delta_k : \mathcal{X} \to \mathbb{R}$, $1 \leq k \leq m$, and assign a pattern x to class $\hat{y}$ using

$$\hat{y} = h(x) = \operatorname{argmax}_k \delta_k(x) \ .$$

Linear classification

- We build affine linear decision functions

$$\delta(\boldsymbol{x}) = \sum_{i=1}^{d} w_i x_i + b = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b$$

  with $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$.
- For convenience, we define $\tilde{\boldsymbol{x}}_i^\mathsf{T} = (x_1, \dots, x_d, 1)$ for $i = 1, \dots, N$ and $\tilde{\boldsymbol{w}}^\mathsf{T} = (w_1, \dots, w_d, b)$ and consider the equivalent formulation

$$\delta(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{d+1} \tilde{w}_i \tilde{x}_i = \tilde{\boldsymbol{w}}^\mathsf{T}\tilde{\boldsymbol{x}} \enspace .$$

  We omit the tilde in the following.

Binary decision functions

- If we have only two classes, we can consider a single function

$$\delta(x) = \delta_1(x) - \delta_2(x)$$

  and the hypothesis

$$h(x) = \begin{cases} \mathcal{C}_1 & \text{if } \delta(x) > 0 \\ \mathcal{C}_2 & \text{otherwise} \end{cases} .$$

- For $\mathcal{Y} = \{-1, 1\}$ this is equal to

$$h(x) = \operatorname{sgn}(\delta(x)) = \begin{cases} 1 & \text{if } \delta(x) > 0 \\ -1 & \text{otherwise} \end{cases} .$$

Decision functions and class posteriors

- If we know the class posteriors $P(Y \mid X)$ we can perform optimal classification: a pattern x is assigned to class $\mathcal{C}_k$ with maximum $P(Y = \mathcal{C}_k \mid X = x)$, i.e.,

$$\hat{y} = h(x) = \operatorname{argmax}_k P(Y = \mathcal{C}_k \mid X = x)$$

or in the binary case with $\mathcal{Y} = \{-1, 1\}$

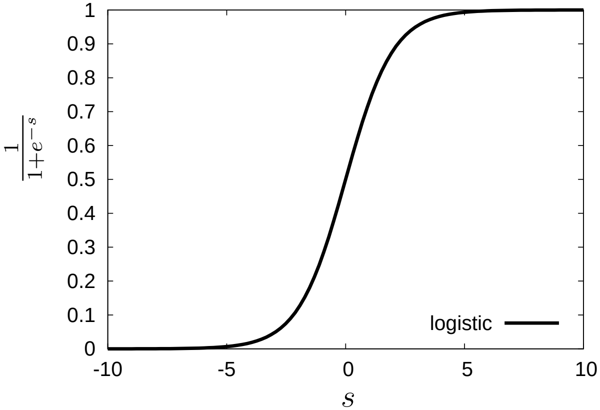
$$\delta(\boldsymbol{x}) = P(Y = \mathcal{C}_1 \mid X = x) - P(Y = \mathcal{C}_2 \mid X = x)$$

and $\hat{y} = h(x) = \operatorname{sgn}(\delta(x))$.

- $P(Y = \mathcal{C}_k \mid X = x)$ is proportional to the class-conditional density $p(X = x \mid Y = \mathcal{C}_k)$ times the class prior $P(Y = \mathcal{C}_k)$:

$$P(Y = \mathcal{C}_k \mid X = x) = \frac{p(X = x \mid Y = \mathcal{C}_k) P(Y = \mathcal{C}_k)}{p(X = x)}$$

Logistic function



$$\theta(s) = \frac{1}{1 + e^{-s}}$$

Predicting probabilities

- Instead of predicting the class label, we want to learn

$$f(\boldsymbol{x}) = P(Y = 1 \,|\, X = \boldsymbol{x})$$

assuming that the data is generated by

$$P(Y = y \,|\, X = \boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{for } y = 1 \\ 1 - f(\boldsymbol{x}) & \text{for } y = -1 \end{cases} .$$

- In the binary case, our model takes the form $h : \mathcal{X} \to [0, 1]$:

$$h(\boldsymbol{x}) = \theta(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{x})$$

Likelihood function

- Our hypothesis h describes the probability distribution:

$$P(Y = y \,|\, X = \boldsymbol{x}; \boldsymbol{w}) = \theta(y\boldsymbol{w}^\mathsf{T}\boldsymbol{x}) = \begin{cases} \theta(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}) & \text{for } y = 1 \\ 1 - \theta(\boldsymbol{w}^\mathsf{T}\boldsymbol{x}) & \text{for } y = -1 \end{cases}$$

- $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\} \subseteq (\mathbb{R}^n \times \{-1, 1\})^N$
- Likelihood (function) of the parameters $\boldsymbol{w}$ given training data S is the probability of observing S when the data is generated by h with parameters $\boldsymbol{w}$.
- Likelihood for i.i.d. S:

$$\prod_{i=1}^{N} P(Y = y_i \,|\, X = \boldsymbol{x}_i; h) \text{ or short } \prod_{i=1}^{N} P(y_i \,|\, \boldsymbol{x}_i)$$

Maximum likelihood

- Learning principle: Maximize the likelihood function!
- Equivalently, we can minimize the negative logarithmic likelihood.
- Negative log-likelihood (divided by N):

$$-\frac{1}{N}\ln\left(\prod_{i=1}^{N} P(y_i \mid \boldsymbol{x}_i)\right) = -\frac{1}{N}\sum_{n=1}^{N}\ln\left(P(y_i \mid \boldsymbol{x}_i)\right)$$

- Plugging in our linear hypothesis gives the error function:

$$-\frac{1}{N}\sum_{n=1}^{N}\ln\left(\theta(y_n w^{\mathsf{T}}\boldsymbol{x}_n)\right) = \frac{1}{N}\sum_{n=1}^{N}\ln\left(1+e^{-y_n\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_n}\right)$$

Recall: Gradient

- The *gradient*

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \dots, \frac{\partial f(\boldsymbol{x})}{\partial x_d} \right)^{\mathsf{T}}$$

points in the direction $\nabla f(\boldsymbol{x})/\|\nabla f(\boldsymbol{x})\|$ giving maximum rate of change $\|\nabla f(\boldsymbol{x})\|$.

Gradient descent

- Consider learning by iteratively changing the parameters:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$$

- Simplest choice is (steepest) gradient descent

$$\Delta \boldsymbol{w} = -\eta \nabla f|_{\boldsymbol{w}}$$

with learning rate $\eta > 0$.

Logistic regression algorithm (steepest descent)

Algorithm 1: Logistic regression

Input: data $\{(\boldsymbol{x}_1, y_1), \dots, \boldsymbol{x}_N, y_N)\} \subseteq (\mathbb{R}^n \times \{-1, 1\})^N$,
learning rate η

Output: weights of linear hypothesis $h(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

initialize $\boldsymbol{w}$

repeat

    `// gradient of negative log-likelihood over` N

    $\boldsymbol{g} \leftarrow -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \boldsymbol{x}_n}{1+e^{y_n \boldsymbol{w}^\mathsf{T} \boldsymbol{x}_n}}$

    `// model parameter update`

    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{g}$

until *stopping criterion is met*

Logistic regression algorithm (stochastic gradient descent, SGD)

Algorithm 2: Logistic regression

Input: data $\{(\boldsymbol{x}_1, y_1), \dots\} \subseteq (\mathbb{R}^n \times \{-1, 1\})^N$, learning rate η

Output: weights of linear hypothesis $h(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

initialize $\boldsymbol{w}$

repeat

  pick $(\boldsymbol{x}, y) \in S$

  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \frac{y\boldsymbol{x}}{1+e^{y\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}}}$

until *stopping criterion is met*

Logistic regression algorithm (mini-batch gradient descent)

Algorithm 3: Logistic regression

Input: data $\{(\boldsymbol{x}_1, y_1), \dots\} \subseteq (\mathbb{R}^n \times \{-1, 1\})^N$, learning rate η

Output: weights of linear hypothesis $h(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

initialize $\boldsymbol{w}$

repeat

pick $S' \subset S$

$\boldsymbol{g} \leftarrow -\frac{1}{|S'|} \sum_{(\boldsymbol{x},y)\in S'} \frac{y\boldsymbol{x}}{1+e^{y\boldsymbol{w}^\mathsf{T}\boldsymbol{x}}}$

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{g}$

until *stopping criterion is met*

Multiple classes

- Binary, single decision function, $y \in \{0, 1\}$:

$$P(Y = 1 \,|\, \boldsymbol{x}) = \frac{1}{1 + e^{-\delta(\boldsymbol{x})}} = \frac{e^{\delta(\boldsymbol{x})}}{1 + e^{\delta(\boldsymbol{x})}}$$

- Binary, two decision functions, $y \in \{1, 2\}$:

$$P(Y = y \,|\, \boldsymbol{x}) = \frac{e^{\delta_y(\boldsymbol{x})}}{e^{\delta_1(\boldsymbol{x})} + e^{\delta_2(\boldsymbol{x})}} = \frac{e^{\delta_y(\boldsymbol{x})+C}}{e^{\delta_1(\boldsymbol{x})+C} + e^{\delta_2(\boldsymbol{x})+C}}$$

  for every constant C, thus logistic function is special case for $C = -\delta_1(\boldsymbol{x})$.

- Multiple classes, $y \in \{1, \dots, m\}$:

$$P(Y = y \,|\, \boldsymbol{x}) = \underbrace{\frac{e^{\delta_y(\boldsymbol{x})}}{\sum_{i=1}^{m} e^{\delta_i(\boldsymbol{x})}}}_{\text{softmax function}}$$

Outline

Margins I

The functional margin of an example $(\boldsymbol{x}_i, y_i)$ with respect to a hyperplane $(\boldsymbol{w}, b)$ is

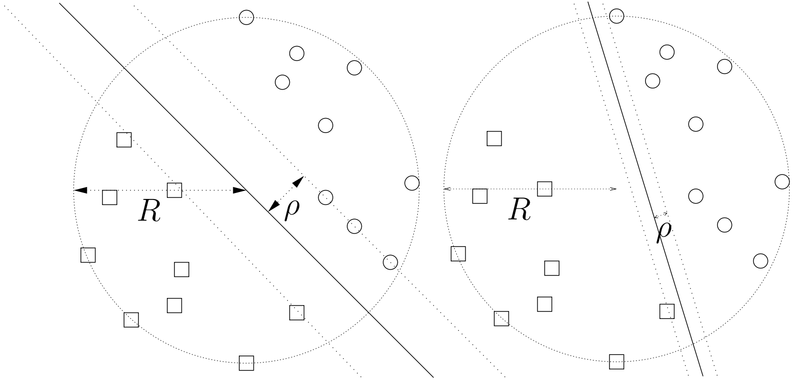
$$\gamma_i := y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \ .$$

The geometric margin of an example $(\boldsymbol{x}_i, y_i)$ with respect to a hyperplane $(\boldsymbol{w}, b)$ is

$$\rho_i := y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)/\|\boldsymbol{w}\| = \gamma_i/\|\boldsymbol{w}\| \ .$$

A positive margin implies correct classification.
The margin of a hyperplane $(\boldsymbol{w}, b)$ with respect to a training set S is $\min_i \rho_i$. The margin of a training set S is the maximum geometric margin over all hyperplanes. A hyperplane realizing this margin is called maximum margin hyperplane.

Margins II



Outline

Analyzing the Perceptron

Why should we look at the Perceptron?

- Linear classifiers such as perceptrons are the basis of technical neurocomputing
- Support Vector Machines are basically linear classifiers
- Basic concepts of learning theory can be explained easily:
  - Margins
  - Dual representation
  - Bounds involving margins and the radius of the ball containing the data

Perceptron learning algorithm (primal form)

For simplicity, consider hyperplanes with no bias ($b = 0$), i.e., $\mathcal{H} = \{h(\boldsymbol{x}) = \operatorname{sgn}(\langle \boldsymbol{w}, \boldsymbol{x} \rangle) \mid \boldsymbol{w} \in \mathbb{R}^n\}$.

Algorithm 4: Perceptron

Input: separable data $\{(\boldsymbol{x}_1, y_1), \dots\} \subseteq (\mathbb{R}^n \times \{-1, 1\})^N$

Output: hypothesis $h(\boldsymbol{x}) = \operatorname{sgn}(\langle \boldsymbol{w}_k, \boldsymbol{x} \rangle)$

$\boldsymbol{w}_0 \leftarrow \mathbf{0}; k \leftarrow 0$

repeat

  for $i = 1, \dots, N$ **do**

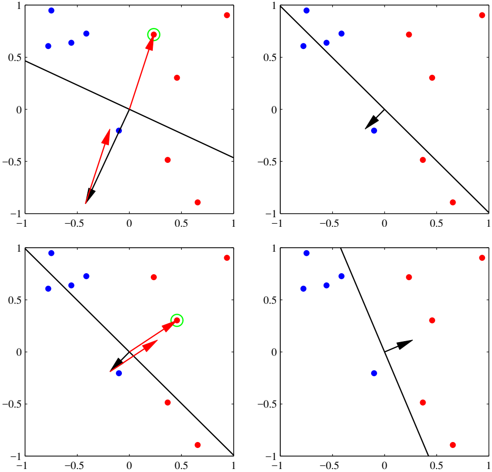
    if $y_i \langle \boldsymbol{w}_k, \boldsymbol{x}_i \rangle \leq 0$ **then**

      $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + y_i \boldsymbol{x}_i$

      $k \leftarrow k + 1$

until *no mistake made within* **for** *loop*

Perceptron learning in pictures



C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006

Outline

Novikoff

Theorem (Novikoff)

Let $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ be a non-trivial training set (i.e., containing patterns of both classes), $b = \mathbf{0} = \sum_{i=1}^m 0\boldsymbol{x}_i$ and let

$$R \leftarrow \max_{1 \leq i \leq N} \|\boldsymbol{x}_i\| \ .$$

Suppose that there exists $\boldsymbol{w}_{\mathit{opt}}$ and $\rho > 0$ such that $\|\boldsymbol{w}_{\mathit{opt}}\| = 1$ and

$$y_i \langle \boldsymbol{w}_{\mathit{opt}}, \boldsymbol{x}_i \rangle \geq \rho > 0$$

for $1 \leq i \leq N$. Then the number of updates k made by the online perceptron algorithm on S is at most

$$\left(\frac{R}{\rho}\right)^2 \ .$$

Novikoff, sketch of proof I

Let i be the index of the example in update k

$$
\begin{aligned}
\|\boldsymbol{w}_{k+1}\|^2 &= \langle \boldsymbol{w}_k + y_i \boldsymbol{x}_i, \boldsymbol{w}_k + y_i \boldsymbol{x}_i \rangle \\
&= \|\boldsymbol{w}_k\|^2 + 2y_i \langle \boldsymbol{w}_k, \boldsymbol{x}_i \rangle + \|\boldsymbol{x}_i\|^2 \\
&\leq \|\boldsymbol{w}_k\|^2 + R^2 \\
&\leq (k+1)R^2
\end{aligned}
$$

Novikoff, sketch of proof II

$$\begin{aligned}\left\langle \boldsymbol{w}_{\mathsf{opt}}, \boldsymbol{w}_{k+1} \right\rangle &= \left\langle \boldsymbol{w}_{\mathsf{opt}}, \boldsymbol{w}_{k} \right\rangle + y_i \left\langle \boldsymbol{w}_{\mathsf{opt}}, \boldsymbol{x}_{i} \right\rangle \\ &\geq \left\langle \boldsymbol{w}_{\mathsf{opt}}, \boldsymbol{w}_{k} \right\rangle + \rho \\ &\geq (k+1)\rho\end{aligned}$$

$$k^2\rho^2 \leq \left\langle \boldsymbol{w}_{\mathsf{opt}}, \boldsymbol{w}_{k} \right\rangle^2 \leq \|\boldsymbol{w}_{\mathsf{opt}}\|^2 \|\boldsymbol{w}_{k}\|^2 \leq kR^2$$

$$k \leq \frac{R^2}{\rho^2}$$

Dual representation

- Weight vector of hyperplane computed by online perceptron algorithm can be written as

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

- Function $h(\boldsymbol{x}) = \operatorname{sgn}(\delta(\boldsymbol{x}))$ can be written in dual coordinates

$$\begin{aligned}
\delta(\boldsymbol{x}) &= \langle \boldsymbol{w}, \boldsymbol{x} \rangle \\
&= \left\langle \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i, \boldsymbol{x} \right\rangle \\
&= \sum_{i=1}^{N} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle
\end{aligned}$$

Perceptron learning algorithm (dual form)

Algorithm 5: Perceptron (dual form)

Input: separable data $\{(\boldsymbol{x}_1, y_1), \dots\} \subseteq (\mathbb{R}^n \times \{-1, 1\})^N$

Output: hypothesis $h(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{N} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle\right)$

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
repeat
    for $i = 1, \dots, N$ **do**
        if $y_i \sum_{j=1}^{N} \alpha_j y_j \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle \leq 0$ **then**
            $\alpha_i \leftarrow \alpha_i + 1$
until *no mistake made within* **for** *loop*

Outline

Summary I

Logistic regression

- is easy to use, has in its simplest form no hyperparameters (not counting η),
- gives surprisingly good results, is highly recommended as baseline method,
- does typically not tend to overfit (assuming $d \ll N$), but does not capture non-linearities,
- can be used with non-linear transformations,
- can be parallelized and is applicable to "Big Data".

Summary II

Hey, we also now know about

- perceptron learning,
- margins,
- dual representation,
- bounds involving margins and the radius of the ball containing the data.