CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering
Department of Cybernetics

Linear models for classification.
Perceptron. Logistic regression.

Petr Pošík

Linear classification

Binary classification task (dichotomy)

Let's have the training dataset $T = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(|T|)}, y^{(|T|)})\}$:

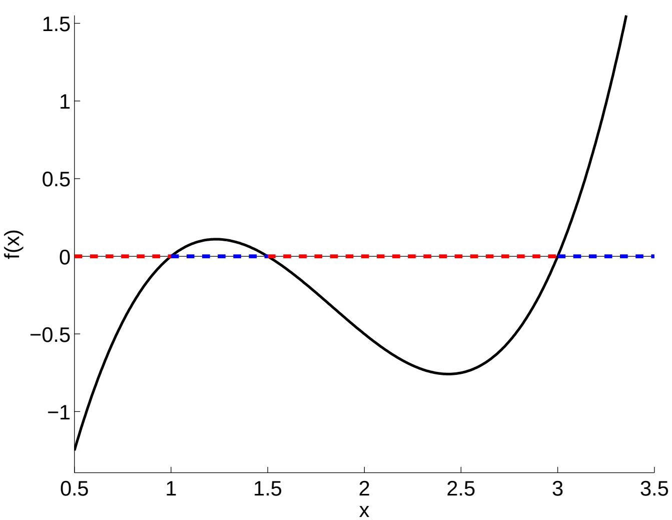
- each example is described by a vector of features $\boldsymbol{x} = (x_1, \ldots, x_D)$,
- each example is labeled with the correct class $y \in \{+1, -1\}$.

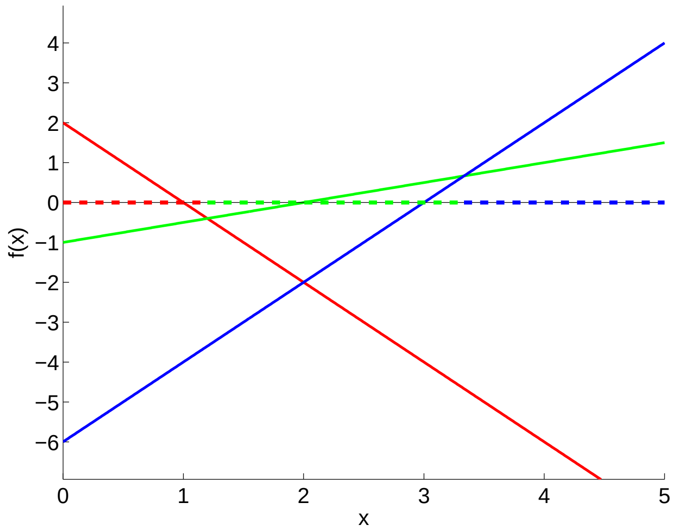
Discrimination function: a function allowing us to *decide* to which class an example $\boldsymbol{x}$ belongs.

- For 2 classes, 1 discrimination function is enough.
- Decision rule:

$$\left.\begin{array}{l} f(\boldsymbol{x}^{(i)}) > 0 \Longleftrightarrow \widehat{y}^{(i)} = +1 \\ f(\boldsymbol{x}^{(i)}) < 0 \Longleftrightarrow \widehat{y}^{(i)} = -1 \end{array}\right\} \qquad \text{i.e.} \qquad \widehat{y}^{(i)} = \operatorname{sign}\left(f(\boldsymbol{x}^{(i)})\right)$$

- *Learning* then amounts to finding (parameters of) function f.





Naive approach

Problem: Learn a linear discrimination function f from data T.

Naive approach

Problem: Learn a linear discrimination function f from data T.

Naive solution: fit linear regression model to the data!

- Use cost function

$$J_{MSE}(\boldsymbol{w}, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \left(y^{(i)} - f(\boldsymbol{w}, \boldsymbol{x}^{(i)}) \right)^2,$$

- minimize it with respect to $\boldsymbol{w}$,
- and use $\widehat{y} = \mathrm{sign}(f(\boldsymbol{x}))$.
- Issue: Points far away from the decision boundary have *huge effect* on the model!

Naive approach

Problem: Learn a linear discrimination function f from data T.

Naive solution: fit linear regression model to the data!

- Use cost function

$$J_{MSE}(\boldsymbol{w}, T) = \frac{1}{|T|}\sum_{i=1}^{|T|}\left(y^{(i)} - f(\boldsymbol{w}, \boldsymbol{x}^{(i)})\right)^2,$$

- minimize it with respect to $\boldsymbol{w}$,
- and use $\widehat{y} = \text{sign}(f(\boldsymbol{x}))$.
- Issue: Points far away from the decision boundary have *huge effect* on the model!

Better solution: fit a linear discrimination function which minimizes the number of errors!

- Cost function:

$$J_{01}(\boldsymbol{w}, T) = \frac{1}{|T|}\sum_{i=1}^{|T|}\mathbb{I}(y^{(i)} \neq \widehat{y}^{(i)}),$$

  where $\mathbb{I}$ is the indicator function: $\mathbb{I}(a)$ returns 1 iff a is True, 0 otherwise.
- The cost function is non-smooth, contains plateaus, not easy to optimize, but there are algorithms which attempt to solve it, e.g. perceptron, Kozinec's algorithm, etc.

Perceptron

Perceptron algorithm

Perceptron [Ros62]:

- a simple model of a neuron
- linear classifier (in this case a classifier with linear discrimination function)

Algorithm 1: Perceptron algorith

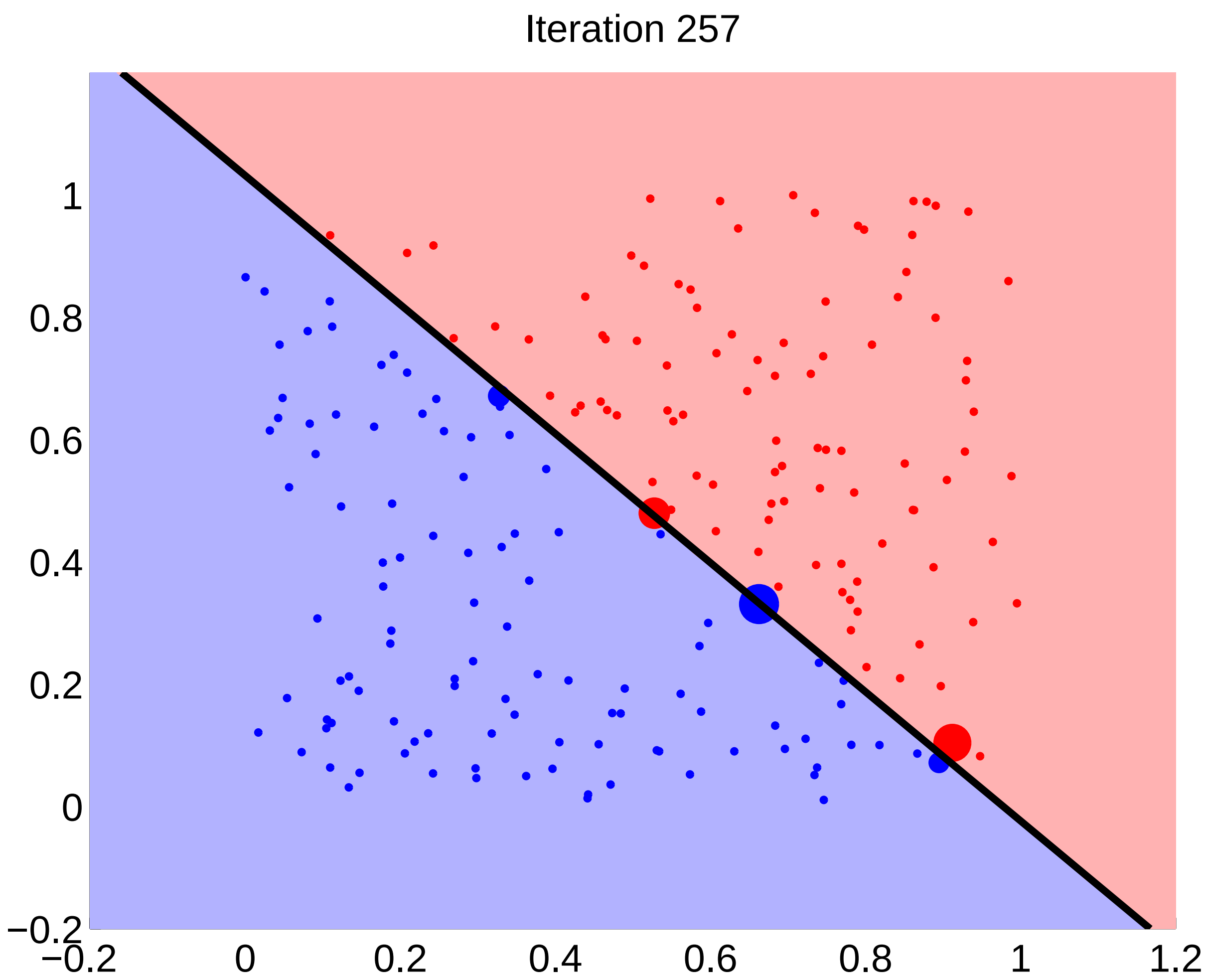
Input: Linearly separable training dataset: $\{\boldsymbol{x}^{(i)}, y^{(i)}\}, \boldsymbol{x}^{(i)} \in \mathcal{R}^{D+1}$ (homogeneous coordinates), $y^{(i)} \in \{+1, -1\}$

Output: Weight vector $\boldsymbol{w}$ such that $\boldsymbol{x}^{(i)}\boldsymbol{w}^T > 0$ iff $y^{(i)} = +1$ and $\boldsymbol{x}^{(i)}\boldsymbol{w}^T < 0$ iff $y^{(i)} = -1$

1 **begin**
2   Initialize the weight vector, e.g. $\boldsymbol{w} = \boldsymbol{0}$.
3   Invert all examples $\boldsymbol{x}$ belonging to class -1: $\boldsymbol{x}^{(i)} = -\boldsymbol{x}^{(i)}$ for all i, where $y^{(i)} = -1$.
4   Find an incorrectly classified training vector, i.e. find j such that $\boldsymbol{x}^{(i)}\boldsymbol{w}^T \leq 0$, e.g. the worst classified vector: $\boldsymbol{x}^{(j)} = \text{argmin}_{\boldsymbol{x}^{(i)}}(\boldsymbol{x}^{(i)}\boldsymbol{w}^T)$.
5   **if** *all examples classified correctly* **then**
6   Return the solution $\boldsymbol{w}$. Terminate.
7   **else**
8   Update the weight vector: $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}^{(j)}$.
9   Go to 4.

[Ros62] Frank Rosenblatt. *Principles of Neurodynamics: Perceptron and the Theory of Brain Mechanisms*. Spartan Books, Washington, D.C., 1962.

Demo: Perceptron



Features of the perceptron algorithm

Perceptron convergence theorem [Nov62]:

- Perceptron algorithm eventually finds a hyperplane that separates 2 classes of points, if such a hyperplane exists.
- If no separating hyperplane exists, the alorithm does not have to converge and will iterate forever.

Possible solutions:

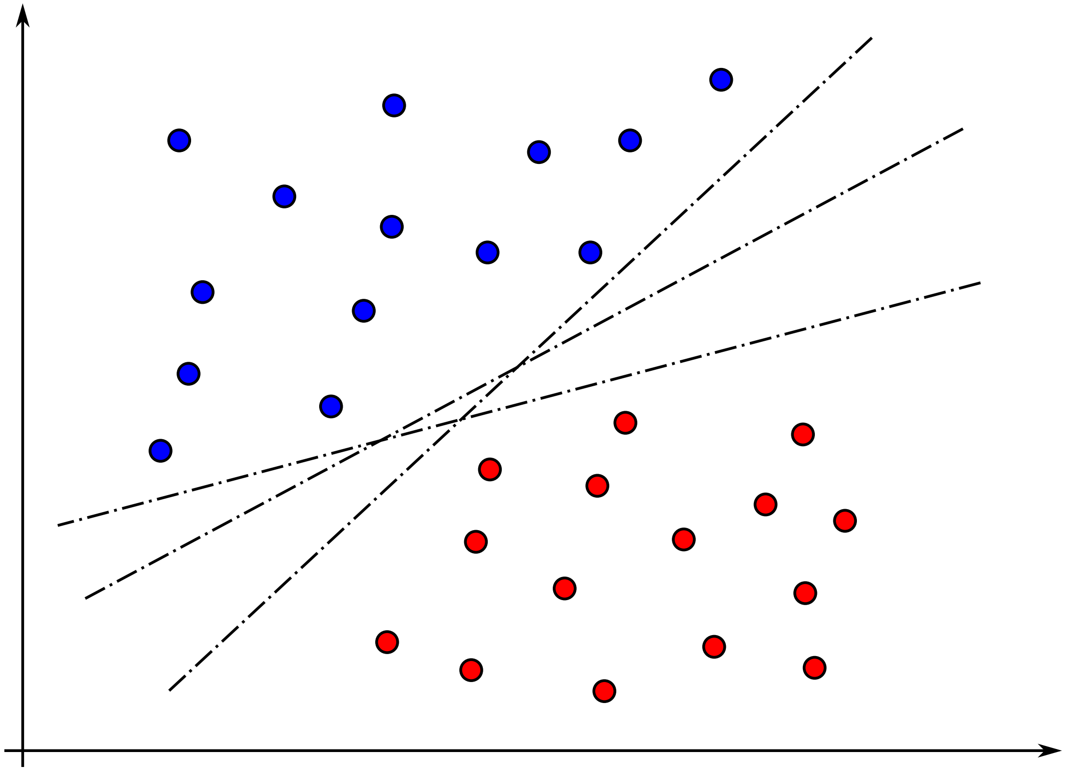
- Pocket algorithm - track the error the perceptron makes in each iteration and store the best weights found so far in a separate memory (pocket).
- Use a different learning algorithm, which finds an approximate solution, if the classes are not linearly separable.

[Nov62] Albert B. J. Novikoff. On convergence proofs for perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, volume 12, Brooklyn, New York, 1962.

The hyperplane found by perceptron

The perceptron algorithm

- finds a separating hyperplane, if it exists;
- but if a single separating hyperplane exists, then there are infinitely many (equally good) separating hyperplanes



- and perceptron finds *any* of them!

Which separating hyperplane is the optimal one? What does "optimal" actually mean? (Possible answers in the SVM lecture.)

Logistic regression

Logistic regression model

Problem: Learn a binary **classifier** for the dataset $T = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, where $y^{(i)} \in \{0, 1\}$.[1]

To reiterate: when using linear regression, the examples far from the decision boundary have a huge impact on h. How to limit their influence?

Logistic regression model

Problem: Learn a binary **classifier** for the dataset $T = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, where $y^{(i)} \in \{0, 1\}$.[1]

To reiterate: when using linear regression, the examples far from the decision boundary have a huge impact on h. How to limit their influence?

Logistic regression uses a transformation of the values of linear function

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\boldsymbol{x}\boldsymbol{w}^T) = \frac{1}{1 + e^{-\boldsymbol{x}\boldsymbol{w}^T}},$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is the **sigmoid** function (a.k.a **logistic** function).

Logistic regression model

Problem: Learn a binary **classifier** for the dataset $T = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, where $y^{(i)} \in \{0, 1\}$.[1]

To reiterate: when using linear regression, the examples far from the decision boundary have a huge impact on h. How to limit their influence?

Logistic regression uses a transformation of the values of linear function

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = g(\boldsymbol{x}\boldsymbol{w}^T) = \frac{1}{1 + e^{-\boldsymbol{x}\boldsymbol{w}^T}},$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is the **sigmoid** function (a.k.a **logistic** function).

Interpretation of the model:

- $h_{\boldsymbol{w}}(\boldsymbol{x})$ estimates the probability that $\boldsymbol{x}$ belongs to class 1.
- Logistic *regression* is a *classification model!*
- The discrimination function $h_{\boldsymbol{w}}(\boldsymbol{x})$ itself is not linear anymore; but the *decision boundary is still linear!*

[1] Previously, we have used $y^{(i)} \in \{-1, +1\}$, but the values can be chosen arbitrarily, and $\{0, 1\}$ is convenient for logistic regression.

Cost function

To train the logistic regression model, one can use the J_{MSE} criterion:

$$J(\boldsymbol{w}, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \left(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) \right)^2.$$

However, this results in a non-convex multimodal landscape which is hard to optimize.

Cost function

To train the logistic regression model, one can use the J_{MSE} criterion:

$$J(\boldsymbol{w}, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \left(y^{(i)} - h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)}) \right)^2 .$$

However, this results in a non-convex multimodal landscape which is hard to optimize.

Logistic regression uses a modified cost function

$$J(\boldsymbol{w}, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \mathrm{cost}(y^{(i)}, h_{\boldsymbol{w}}(\boldsymbol{x}^{(i)})), \text{ where}$$

$$\mathrm{cost}(y, \widehat{y}) = \begin{cases} -\log(\widehat{y}) & \text{if } y = 1 \\ -\log(1 - \widehat{y}) & \text{if } y = 0 \end{cases},$$

which can be rewritten in a single expression as

$$\mathrm{cost}(y, \widehat{y}) = -y \log(\widehat{y}) - (1 - y) \log(1 - \widehat{y}).$$

Such a cost function is simpler to optimize.