

Linear models for classification.
Perceptron. Logistic regression.

Petr Pošík

Linear classification	2
Binary class.....	3
Naive approach.....	4
Perceptron	5
Algorithm	6
Demo	7
Features	8
Result	9
Logistic regression	10
Model	11
Cost function.....	12

Binary classification task (dichotomy)

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

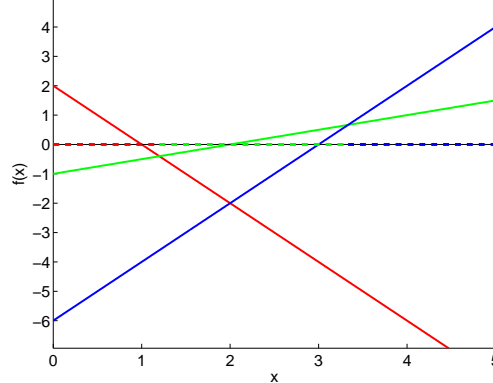
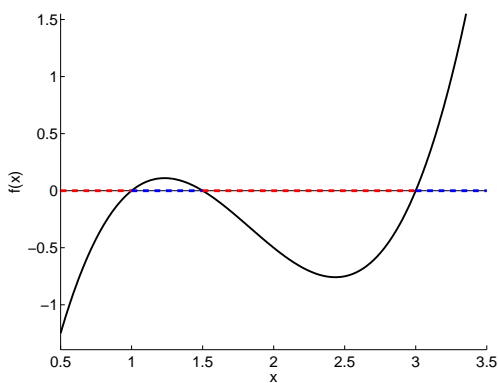
- each example is described by a vector of features $x = (x_1, \dots, x_D)$,
- each example is labeled with the correct class $y \in \{+1, -1\}$.

Discrimination function: a function allowing us to *decide* to which class an example x belongs.

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

$$\left. \begin{aligned} f(x^{(i)}) > 0 &\iff \hat{y}^{(i)} = +1 \\ f(x^{(i)}) < 0 &\iff \hat{y}^{(i)} = -1 \end{aligned} \right\} \text{ i.e. } \hat{y}^{(i)} = \text{sign}(f(x^{(i)}))$$

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



Naive approach

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

Naive solution: fit linear regression model to the data!

- Use cost function

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

- minimize it with respect to w ,
- and use $\hat{y} = \text{sign}(f(x))$.
- 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:

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

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 algorithm

Perceptron [Ros62]:

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

Algorithm 1: Perceptron algorithm

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

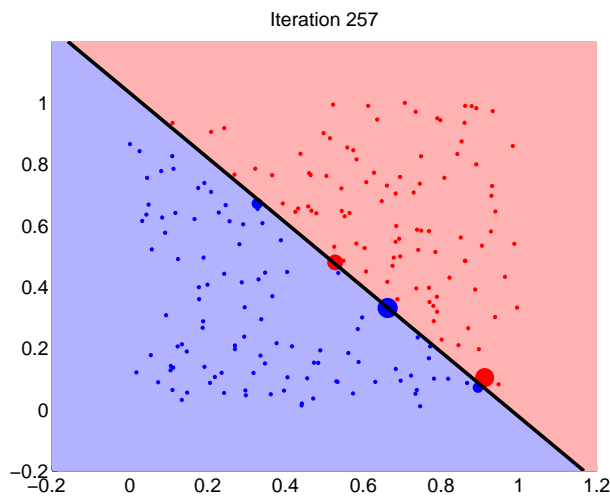
Output: Weight vector w such that $x^{(i)}w^T > 0$ iff $y^{(i)} = +1$ and $x^{(i)}w^T < 0$ iff $y^{(i)} = -1$

```

1 begin
2   Initialize the weight vector, e.g.  $w = 0$ .
3   Invert all examples  $x$  belonging to class -1:  $x^{(i)} = -x^{(i)}$  for all  $i$ , where  $y^{(i)} = -1$ .
4   Find an incorrectly classified training vector, i.e. find  $j$  such that  $x^{(j)}w^T \leq 0$ , e.g. the worst classified vector:  $x^{(j)} = \operatorname{argmin}_{x^{(i)}} (x^{(i)}w^T)$ .
5   if all examples classified correctly then
6     Return the solution  $w$ . Terminate.
7   else
8     Update the weight vector:  $w = w + x^{(j)}$ .
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 algorithm does not have to converge and will iterate forever.

Possible solutions:

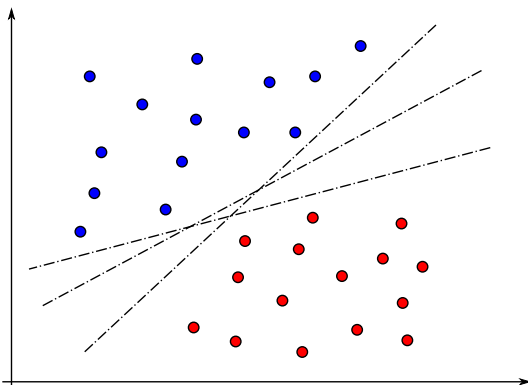
- Pocket algorithm - track the error the perceptron makes in each iteration and store the best weights found so far in a separate memory (pocket).
- Use a different learning algorithm, which finds an approximate solution, if the classes are not linearly separable.

[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 model

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

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

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

where

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

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

Interpretation of the model:

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

^aPreviously, 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:

$$J(w, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} (y^{(i)} - h_w(\mathbf{x}^{(i)}))^2.$$

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

Logistic regression uses a modified cost function

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

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

which can be rewritten in a single expression as

$$\text{cost}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}).$$

Such a cost function is simpler to optimize.