Linear models for classification.
Perceptron. Logistic regression.

Petr Pošík
Linear classification
Binary classification task (dichotomy)

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

- each example is described by a vector of features $x = (x_1, \ldots, x_D)$,
- 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 $x$ belongs.

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

\[
\begin{align*}
  f(x^{(i)}) > 0 & \iff \hat{y}^{(i)} = +1 \\
  f(x^{(i)}) < 0 & \iff \hat{y}^{(i)} = -1
\end{align*}
\]

i.e. $\hat{y}^{(i)} = \text{sign} \left( f(x^{(i)}) \right)$

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

![Graph of f(x)](image1)

![Graph of f(x) and its linear approximations](image2)
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
  \[
  J_{MSE}(\mathbf{w}, T) = \frac{1}{|T|} \sum_{i=1}^{T} \left( y^{(i)} - f(\mathbf{w}, x^{(i)}) \right)^2,
  \]

- minimize it with respect to $\mathbf{w}$,
- and use $\hat{y} = \text{sign}(f(x))$.
- 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
  \[
  J_{MSE}(w, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \left( y^{(i)} - f(w, x^{(i)}) \right)^2,
  \]

- 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
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: \( \{x^{(i)}, y^{(i)}\}, x^{(i)} \in \mathbb{R}^{D+1} \) (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 \( y^{(i)} = -1 \)

begin

1 Initialize the weight vector, e.g. \( w = 0 \).

2 Invert all examples \( x \) belonging to class -1: \( x^{(i)} = -x^{(i)} \) for all \( i \), where \( y^{(i)} = -1 \).

3 Find an incorrectly classified training vector, i.e. find \( j \) such that \( x^{(i)}w^T \leq 0 \), e.g. the worst classified vector: \( x^{(j)} = \text{argmin}_{x^{(i)}} (x^{(i)}w^T) \).

4 if all examples classified correctly then

5 Return the solution \( w \). Terminate.

else

6 Update the weight vector: \( w = w + x^{(j)} \).

7 Go to 4.

Demo: Perceptron

Linear classification
- Perceptron
  - Algorithm
  - Demo
  - Features
  - Result

Logistic regression
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.

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 $T = \{(x^{(i)}, y^{(i)})\}$, 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 $T = \{(x^{(i)}, y^{(i)})\}$, where $y^{(i)} \in \{0, 1\}$.\footnote{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

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

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 \( T = \{(x^{(i)}, y^{(i)})\} \), 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

\[
h_w(x) = g(xw^T) = \frac{1}{1 + e^{-xw^T}},
\]

where

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

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

**Interpretation of the model:**

- \( h_w(x) \) estimates the probability that \( x \) belongs to class 1.
- Logistic regression is a classification model!
- The discrimination function \( h_w(x) \) 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 $J_{\text{MSE}}$ criterion:

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

However, this results in a non-convex multimodal landscape which is hard to optimize.
To train the logistic regression model, one can use the $J_{MSE}$ criterion:

\[ J(w, T) = \frac{1}{|T|} \sum_{i=1}^{|T|} \left( y(i) - h_w(x(i)) \right)^2. \]

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

Logistic regression uses a modified cost function

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

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.