Numerical features

- Throughout this lecture we assume that all features are numerical, i.e., feature vectors belong to $\mathbb{R}^n$.

Numerical features

- Throughout this lecture we assume that all features are numerical, i.e., feature vectors belong to $\mathbb{R}^n$.
- Most non-numerical features can be conveniently transformed to numerical ones.

  For example:

  - Colors $\{blue, red, yellow\}$ can be represented by

    $$\{(1,0,0),(0,1,0),(0,0,1)\}$$

    (one-hot encoding)
  - Words can be embedded into vector spaces by various means (word2vec etc.)
  - A black-and-white picture of $x \times y$ pixels can be encoded as a vector of xy numbers that capture the shades of gray of the pixels.

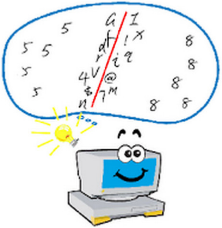
    (Even though this is possibly not the best way of representing images.)

Basic Problems

We consider two basic problems:

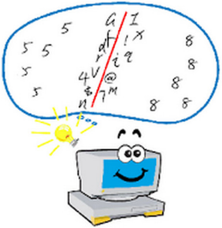
- (Binary) classification

  Our goal: Classify inputs into two categories.



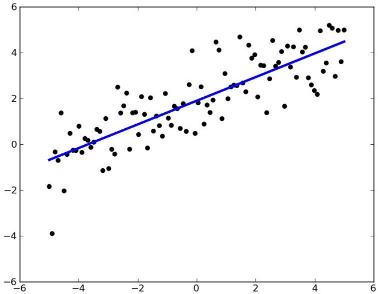
Basic Problems

We consider two basic problems:



- (Binary) classification

  Our goal: Classify inputs into two categories.



- Function approximation (regression)

  Our goal: Find a (hypothesized) functional dependency in data.

Binary classification in $\mathbb{R}^n$

Assume an *unknown* categorization function $c : \mathbb{R}^n \rightarrow \{0, 1\}$.

Our goal:

- Given a set D of training examples of the form $(\vec{x}, c(\vec{x}))$ where $\vec{x} \in \mathbb{R}^n$,

Binary classification in $\mathbb{R}^n$

Assume an *unknown* categorization function $c : \mathbb{R}^n \to \{0, 1\}$.

Our goal:

- Given a set D of training examples of the form $(\vec{x}, c(\vec{x}))$ where $\vec{x} \in \mathbb{R}^n$,
- construct a hypothesized categorization function $h \in \mathcal{H}$ that is consistent with c on the training examples, i.e.,

$$h(\vec{x}) = c(\vec{x}) \text{ for all training examples } (\vec{x}, c(\vec{x})) \in D$$

Binary classification in $\mathbb{R}^n$

Assume an *unknown* categorization function $c : \mathbb{R}^n \to \{0, 1\}$.

Our goal:

- Given a set D of training examples of the form $(\vec{x}, c(\vec{x}))$ where $\vec{x} \in \mathbb{R}^n$,
- construct a hypothesized categorization function $h \in \mathcal{H}$ that is consistent with c on the training examples, i.e.,

$$h(\vec{x}) = c(\vec{x}) \text{ for all training examples } (\vec{x}, c(\vec{x})) \in D$$

Comments:

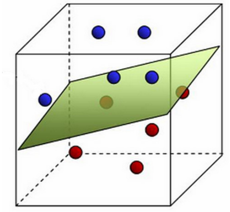
- In practice, we often do not strictly demand $h(\vec{x}) = c(\vec{x})$ for all training examples $(\vec{x}, c(\vec{x})) \in D$ (often it is impossible)
- We are more interested in good **generalization**, that is how well h classifies new instances that do not belong to D.

  (Recall that we usually evaluate accuracy of the resulting hypothesized function h on a test set.)

Hypothesis Spaces

We consider two kinds of hypothesis spaces:

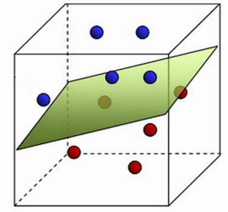
- Linear (affine) classifiers (this lecture)



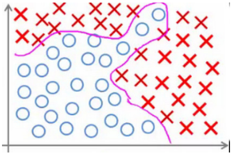
Hypothesis Spaces

We consider two kinds of hypothesis spaces:

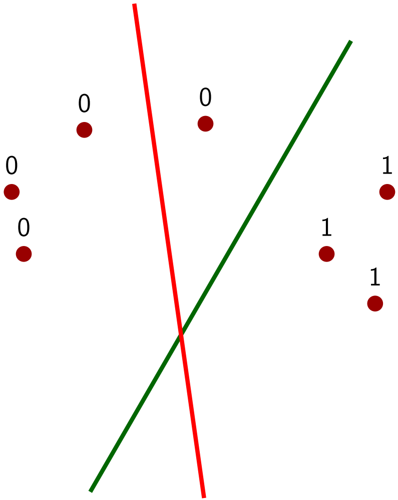
- Linear (affine) classifiers (this lecture)



- Non-linear classifiers (kernel SVM, neural networks) (next lectures)

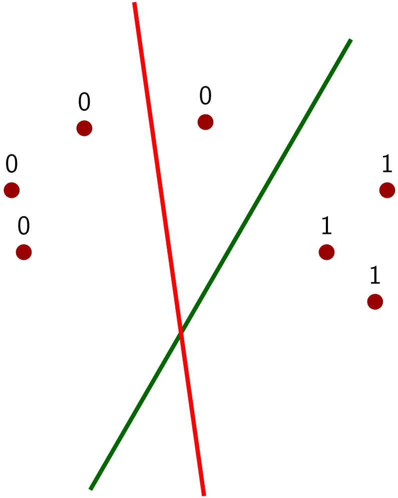


Linear classifier - example



- classification in plane using a linear classifier

Linear classifier - example



- classification in plane using a linear classifier
- if a point is incorrectly classified, the learning algorithm turns the line (hyperplane) to improve the classification.

Length and Scalar Product of Vectors

- We consider vectors $\vec{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$.

Length and Scalar Product of Vectors

- We consider vectors $\vec{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$.
- Euclidean metric on vectors: $|\vec{x}| = \sqrt{\sum_{i=1}^{m} x_i^2}$
  The distance between two vectors (points) $\vec{x}, \vec{y}$ is $|\vec{x} - \vec{y}|$.

Length and Scalar Product of Vectors

- We consider vectors $\vec{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$.
- Euclidean metric on vectors: $|\vec{x}| = \sqrt{\sum_{i=1}^m x_i^2}$
  The distance between two vectors (points) $\vec{x}, \vec{y}$ is $|\vec{x} - \vec{y}|$.
- *Scalar product* $\vec{x} \cdot \vec{y}$ of vectors $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{y} = (y_1, \ldots, y_m)$ defined by

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{m} x_i y_i$$

  - Recall that $\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}| \cos\theta$ where θ is the angle between $\vec{x}$ and $\vec{y}$. That is $\vec{x} \cdot \vec{y}$ is the length of the projection of $\vec{y}$ on $\vec{x}$ multiplied by $|\vec{x}|$.
  - Note that $\vec{x} \cdot \vec{x} = |\vec{x}|^2$

Linear Classifier

A *linear classifier* $h[\vec{w}]$ is determined by a vector of *weights* $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

Linear Classifier

A *linear classifier* $h[\vec{w}]$ is determined by a vector of *weights* $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$,

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \\ 0 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i < 0 \end{cases}$$

Linear Classifier

A *linear classifier* $h[\vec{w}]$ is determined by a vector of *weights* $\vec{w} = (w_0, w_1, \ldots, w_n) \in \mathbb{R}^{n+1}$ as follows:

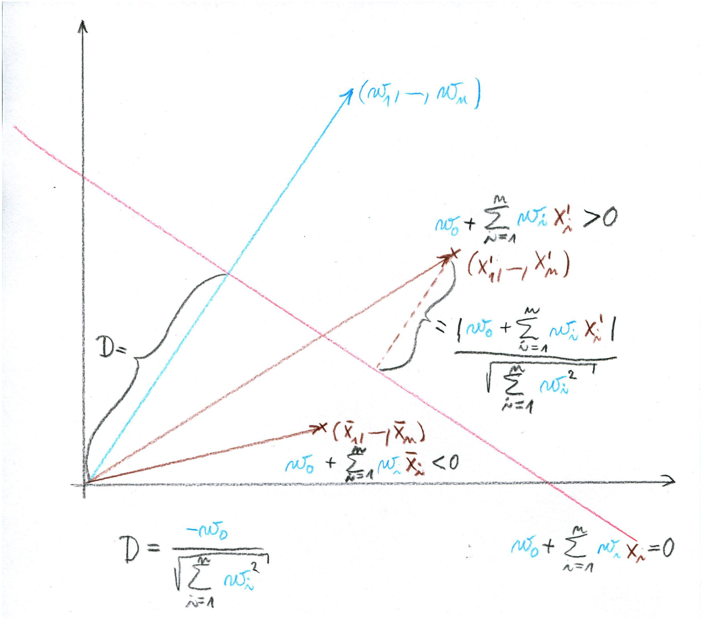
Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$,

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \\ 0 & w_0 + \sum_{i=1}^{n} w_i \cdot x_i < 0 \end{cases}$$

More succinctly:

$$h(\vec{x}) = sgn\left(w_0 + \sum_{i=1}^{n} w_i \cdot x_i\right) \qquad \text{where} \qquad sgn(y) = \begin{cases} 1 & y \geq 0 \\ 0 & y < 0 \end{cases}$$

Linear Classifier – Geometry



Linear Classifier – Notation

Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ we define an *augmented feature vector*

$$\tilde{\mathrm{x}} = (x_0, x_1, \ldots, x_n) \quad \text{where } x_0 = 1$$

Linear Classifier – Notation

Given $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ we define an *augmented feature vector*

$$\tilde{x} = (x_0, x_1, \ldots, x_n) \quad \text{where } x_0 = 1$$

This makes the notation for the linear classifier more succinct:

$$h[\vec{w}](\vec{x}) = sgn(\vec{w} \cdot \tilde{x})$$

Perceptron Learning

- Given a training set

$$D = \{(\vec{x}_1, c(\vec{x}_1)), (\vec{x}_2, c(\vec{x}_2)), \ldots, (\vec{x}_p, c(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

Perceptron Learning

- Given a training set

$$D = \{(\vec{x}_1, c(\vec{x}_1)), (\vec{x}_2, c(\vec{x}_2)), \ldots, (\vec{x}_p, c(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

We write c_k instead of $c(\vec{x}_k)$.
Note that $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

Perceptron Learning

- Given a training set

  $$D = \{(\vec{x}_1, c(\vec{x}_1)), (\vec{x}_2, c(\vec{x}_2)), \ldots, (\vec{x}_p, c(\vec{x}_p))\}$$

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

  We write c_k instead of $c(\vec{x}_k)$.
  Note that $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- A weight vector $\vec{w} \in \mathbb{R}^{n+1}$ is **consistent with** D if

  $$h[\vec{w}](\vec{x}_k) = sgn(\vec{w} \cdot \tilde{\mathrm{x}}_k) = c_k \qquad \text{for all } k = 1, \ldots, p$$

Perceptron Learning

- Given a training set

  $$D = \{(\vec{x}_1, c(\vec{x}_1)), (\vec{x}_2, c(\vec{x}_2)), \ldots, (\vec{x}_p, c(\vec{x}_p))\}$$

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

  We write c_k instead of $c(\vec{x}_k)$.
  Note that $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- A weight vector $\vec{w} \in \mathbb{R}^{n+1}$ is **consistent with** D if

  $$h[\vec{w}](\vec{x}_k) = sgn(\vec{w} \cdot \tilde{\mathrm{x}}_k) = c_k \quad \text{for all } k = 1, \ldots, p$$

  D is **linearly separable** if there is a vector $\vec{w} \in \mathbb{R}^{n+1}$ which is consistent with D.

Perceptron Learning

- Given a training set

$$D = \{(\vec{x}_1, c(\vec{x}_1)), (\vec{x}_2, c(\vec{x}_2)), \ldots, (\vec{x}_p, c(\vec{x}_p))\}$$

  Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $c(\vec{x}_k) \in \{0, 1\}$.

  We write c_k instead of $c(\vec{x}_k)$.
  Note that $\tilde{x}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- A weight vector $\vec{w} \in \mathbb{R}^{n+1}$ is **consistent with** D if

$$h[\vec{w}](\vec{x}_k) = sgn(\vec{w} \cdot \tilde{x}_k) = c_k \quad \text{for all } k = 1, \ldots, p$$

  D is **linearly separable** if there is a vector $\vec{w} \in \mathbb{R}^{n+1}$ which is consistent with D.

- Our goal is to find a consistent $\vec{w}$ assuming that D is linearly separable.

Perceptron – Learning Algorithm

Online learning algorithm:

Idea: Cyclically go through the training examples in D and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to it's correct half-space.

Perceptron – Learning Algorithm

Online learning algorithm:

Idea: Cyclically go through the training examples in D and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to it's correct half-space.

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

Perceptron – Learning Algorithm

Online learning algorithm:

Idea: Cyclically go through the training examples in D and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to it's correct half-space.

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$

Perceptron – Learning Algorithm

Online learning algorithm:

Idea: Cyclically go through the training examples in D and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to it's correct half-space.

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\vec{w}^{(t+1)} \; = \; \vec{w}^{(t)} \; - \; \varepsilon \cdot \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathrm{x}}_k$$

Perceptron – Learning Algorithm

Online learning algorithm:

Idea: Cyclically go through the training examples in D and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to it's correct half-space.

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{x}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \left(sgn\left(\vec{w}^{(t)} \cdot \tilde{x}_k \right) - c_k \right) \cdot \tilde{x}_k\end{aligned}$$

  Here $k = (t \mod p) + 1$, i.e., the examples are considered cyclically, and $0 < \varepsilon \leq 1$ is a **learning rate**.

Perceptron – Learning Algorithm

Online learning algorithm:

Idea: Cyclically go through the training examples in D and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example becomes closer to it's correct half-space.

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k\right) \cdot \tilde{\mathrm{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \left(sgn\left(\vec{w}^{(t)} \cdot \tilde{\mathrm{x}}_k\right) - c_k\right) \cdot \tilde{\mathrm{x}}_k\end{aligned}$$

  Here $k = (t \mod p) + 1$, i.e., the examples are considered cyclically, and $0 < \varepsilon \leq 1$ is a **learning rate**.

Věta (Rosenblatt)

If D is linearly separable, then there is t^* such that $\vec{w}^{(t^*)}$ is consistent with D.

Example

Training set:

$$D = \{((2,-1),1),((2,1),1),((1,3),0)\}$$

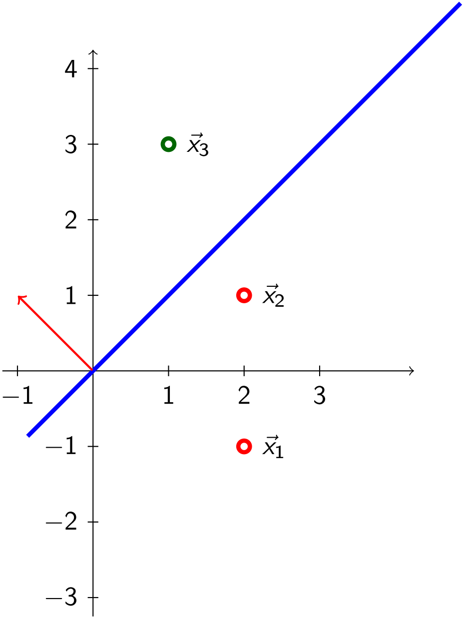
That is

$$\vec{x}_1 = (2,-1) \qquad \tilde{x}_1 = (1,2,-1)$$
$$\vec{x}_2 = (2,1) \qquad \tilde{x}_2 = (1,2,1)$$
$$\vec{x}_3 = (1,3) \qquad \tilde{x}_3 = (1,1,3)$$

$$c_1 = 1$$
$$c_2 = 1$$
$$c_3 = 0$$

Assume that the initial vector $\vec{w}^{(0)}$ is $\vec{w}^{(0)} = (0,-1,1)$.
Consider $\varepsilon = 1$.

Example: Separating by $\vec{w}^{(0)}$



Denoting $\vec{w}^{(0)} = (w_0, w_1, w_2) = (0, -1, 1)$ the blue separating line is given by $w_0 + w_1 x_1 + w_2 x_2 = 0$.

The red vector normal to the blue line is (w_1, w_2).

The points on the side of (w_1, w_2) are assigned 1 by the classifier, the others zero. (In this case $\vec{x}_3$ is assigned one and $\vec{x}_1, \vec{x}_2$ are assigned zero, all of this is inconsistent with $c_1 = 1, c_2 = 1, c_3 = 0$.)

Example: $\vec{w}^{(1)}$

We have

$$\vec{w}^{(0)} \cdot \tilde{\mathrm{x}}_1 = (0, -1, 1) \cdot (1, 2, -1) = 0 - 2 - 1 = -3$$

thus

$$sgn\left(\vec{w}^{(0)} \cdot \tilde{\mathrm{x}}_1\right) = 0$$

and thus

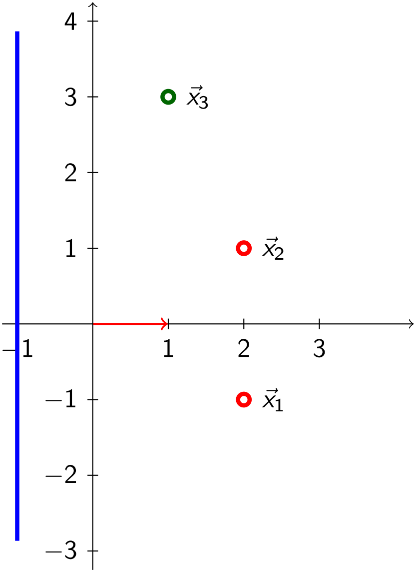
$$sgn\left(\vec{w}^{(0)} \cdot \tilde{\mathrm{x}}_1\right) - c_1 = 0 - 1 = -1$$

(I.e., $\vec{x}_1$ is not correctly classified, and $\vec{w}^{(0)}$ is not consistent with D.)

Hence,

$$\begin{aligned}\vec{w}^{(1)} &= \vec{w}^{(0)} - \left(sgn\left(\vec{w}^{(0)} \cdot \tilde{\mathrm{x}}_1\right) - c_1\right) \cdot \tilde{\mathrm{x}}_1 \\ &= \vec{w}^{(0)} + \tilde{\mathrm{x}}_1 \\ &= (0, -1, 1) + (1, 2, -1) \\ &= (1, 1, 0)\end{aligned}$$

Example



Example: Separating by $\vec{w}^{(1)}$

We have

$$\vec{w}^{(1)} \cdot \tilde{x}_2 = (1,1,0) \cdot (1,2,1) = 1 + 2 = 3$$

thus

$$sgn\left(\vec{w}^{(1)} \cdot \tilde{x}_2\right) = 1$$

and thus

$$sgn\left(\vec{w}^{(1)} \cdot \tilde{x}_2\right) - c_2 = 1 - 1 = 0$$

(I.e., $\vec{x}_2$ *is* currently correctly classified by $\vec{w}^{(1)}$. However, as we will see, $\vec{x}_3$ is not well classified.)

Hence,

$$\vec{w}^{(2)} = \vec{w}^{(1)} = (1,1,0)$$

Example: $\vec{w}^{(3)}$

We have

$$\vec{w}^{(2)} \cdot \tilde{x}_3 = (1,1,0) \cdot (1,1,3) = 1 + 1 = 2$$

thus

$$sgn\left(\vec{w}^{(2)} \cdot \tilde{x}_3\right) = 1$$

and thus

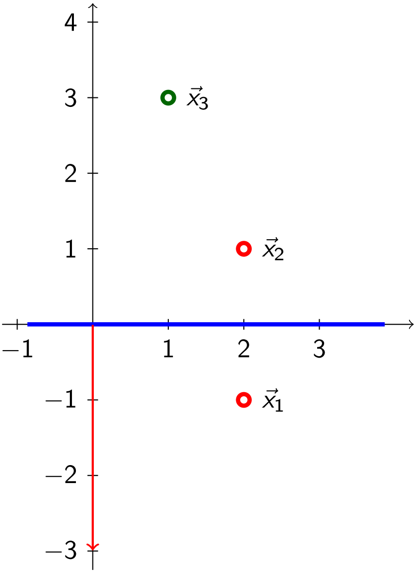
$$sgn\left(\vec{w}^{(2)} \cdot \tilde{x}_3\right) - c_3 = 1 - 0 = 1$$

(This means that $\vec{x}_3$ is not well classified, and $\vec{w}^{(2)}$ is not consistent with D.)

Hence,

$$\begin{aligned} \vec{w}^{(3)} &= \vec{w}^{(2)} - \left(sgn\left(\vec{w}^{(2)} \cdot \tilde{x}_3\right) - c_3\right) \cdot \tilde{x}_3 \\ &= \vec{w}^{(2)} - \tilde{x}_3 \\ &= (1,1,0) - (1,1,3) \\ &= (0,0,-3) \end{aligned}$$

Example: Separating by $\vec{w}^{(3)}$



Example: $\vec{w}^{(4)}$

We have

$$\vec{w}^{(3)} \cdot \tilde{x}_1 = (0, 0, -3) \cdot (1, 2, -1) = 3$$

thus

$$sgn\left(\vec{w}^{(3)} \cdot \tilde{x}_1\right) = 1$$

and thus

$$sgn\left(\vec{w}^{(3)} \cdot \tilde{x}_1\right) - c_1 = 1 - 1 = 0$$

(I.e., $\vec{x}_1$ *is* currently correctly classified by $\vec{w}^{(3)}$. However, we shall see that $\vec{x}_2$ is not.)

Hence,

$$\vec{w}^{(4)} = \vec{w}^{(3)} = (0, 0, -3)$$

Example: $\vec{w}^{(5)}$

We have

$$\vec{w}^{(4)} \cdot \tilde{\mathrm{x}}_2 = (0, 0, -3) \cdot (1, 2, 1) = -3$$

thus

$$sgn\left(\vec{w}^{(4)} \cdot \tilde{\mathrm{x}}_2\right) = 0$$

and thus

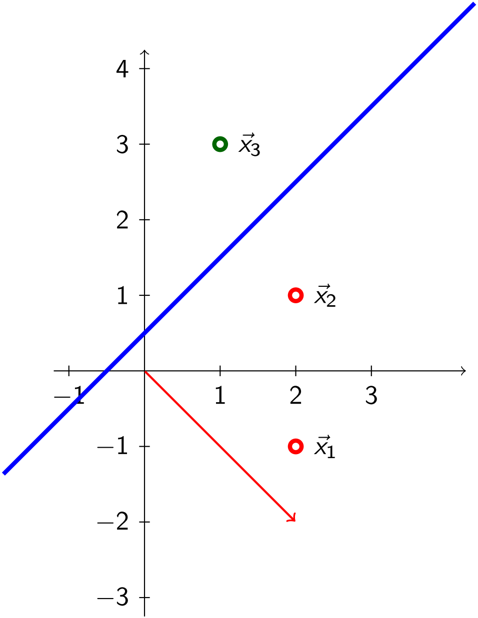
$$sgn\left(\vec{w}^{(4)} \cdot \tilde{\mathrm{x}}_2\right) - c_2 = 0 - 1 = -1$$

(I.e., $\vec{x}_2$ is not correctly classified, and $\vec{w}^{(4)}$ is not consistent with D.)

Hence,

$$\begin{aligned}
\vec{w}^{(5)} &= \vec{w}^{(4)} - \left(sgn\left(\vec{w}^{(4)} \cdot \tilde{\mathrm{x}}_2\right) - c_2\right) \cdot \tilde{\mathrm{x}}_2 \\
&= \vec{w}^{(4)} + \tilde{\mathrm{x}}_2 \\
&= (0, 0, -3) + (1, 2, 1) \\
&= (1, 2, -2)
\end{aligned}$$

Example: Separating by $\vec{w}^{(5)}$



Example: The result

The vector $\vec{w}^{(5)}$ is consistent with D:

$$sgn\left(\vec{w}^{(5)} \cdot \tilde{\mathrm{x}}_1\right) = sgn\left((1, 2, -2) \cdot (1, 2, -1)\right) = sgn(7) = 1 = c_1$$

$$sgn\left(\vec{w}^{(5)} \cdot \tilde{\mathrm{x}}_2\right) = sgn\left((1, 2, -2) \cdot (1, 2, 1)\right) = sgn(3) = 1 = c_2$$

$$sgn\left(\vec{w}^{(5)} \cdot \tilde{\mathrm{x}}_3\right) = sgn\left((1, 2, -2) \cdot (1, 1, 3)\right) = sgn(-3) = 0 = c_3$$

Perceptron – Learning Algorithm

Batch learning algorithm:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

Perceptron – Learning Algorithm

Batch learning algorithm:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$

Perceptron – Learning Algorithm

Batch learning algorithm:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathrm{x}}_k$$

$$= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(sgn\left(\vec{w}^{(t)} \cdot \tilde{\mathrm{x}}_k \right) - c_k \right) \cdot \tilde{\mathrm{x}}_k$$

Here $0 < \varepsilon \leq 1$ is a **learning rate**.

Perceptron – Learning Algorithm

Batch learning algorithm:

Compute a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- $\vec{w}^{(0)}$ is randomly initialized close to $\vec{0} = (0, \ldots, 0)$
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

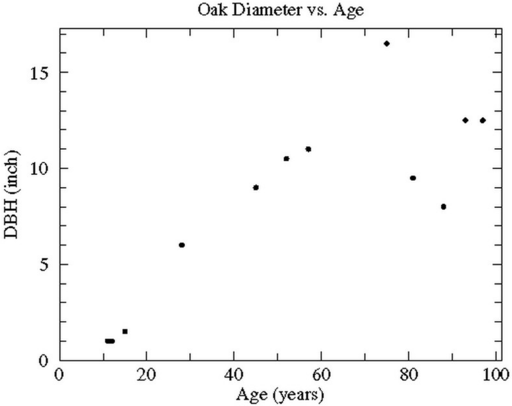
$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\mathrm{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(sgn\left(\vec{w}^{(t)} \cdot \tilde{\mathrm{x}}_k \right) - c_k \right) \cdot \tilde{\mathrm{x}}_k\end{aligned}$$

Here $0 < \varepsilon \leq 1$ is a **learning rate**.

Function Approximation – Oaks in Wisconsin

This example is from *How to Lie with Statistics* by Darrell Huff (1954)

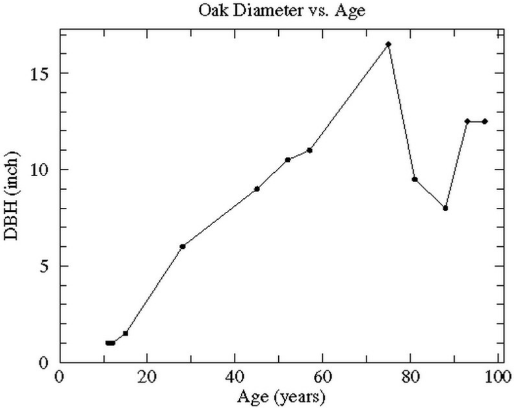
Age (years)	*DBH* (inch)
97	12.5
93	12.5
88	8.0
81	9.5
75	16.5
57	11.0
52	10.5
45	9.0
28	6.0
15	1.5
12	1.0
11	1.0



Function Approximation – Oaks in Wisconsin

This example is from *How to Lie with Statistics* by Darrell Huff (1954)

Age (years)	*DBH* (inch)
97	12.5
93	12.5
88	8.0
81	9.5
75	16.5
57	11.0
52	10.5
45	9.0
28	6.0
15	1.5
12	1.0
11	1.0

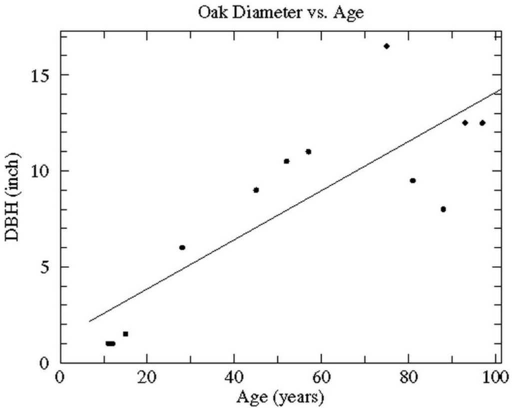


NO!

Function Approximation – Oaks in Wisconsin

This example is from *How to Lie with Statistics* by Darrell Huff (1954)

Age (years)	*DBH* (inch)
97	12.5
93	12.5
88	8.0
81	9.5
75	16.5
57	11.0
52	10.5
45	9.0
28	6.0
15	1.5
12	1.0
11	1.0



possibly YES!

Function Approximation

Assume an *unknown* function $f : \mathbb{R}^n \to \mathbb{R}$.

Our goal:

- Given a set D of training examples of the form $(\vec{x}, f(\vec{x}))$ where $\vec{x} \in \mathbb{R}^n$,

Function Approximation

Assume an *unknown* function $f : \mathbb{R}^n \to \mathbb{R}$.

Our goal:

- Given a set D of training examples of the form $(\vec{x}, f(\vec{x}))$ where $\vec{x} \in \mathbb{R}^n$,
- construct a hypothesized function $h \in \mathcal{H}$ such that

$$h(\vec{x}) \approx f(\vec{x}) \text{ for all training examples } (\vec{x}, f(\vec{x})) \in D$$

  Here $\approx$ means that the values are somewhat close to each other w.r.t. an appropriate *error function* E.

Function Approximation

Assume an *unknown* function $f : \mathbb{R}^n \to \mathbb{R}$.

Our goal:

- Given a set D of training examples of the form $(\vec{x}, f(\vec{x}))$ where $\vec{x} \in \mathbb{R}^n$,
- construct a hypothesized function $h \in \mathcal{H}$ such that

  $$h(\vec{x}) \approx f(\vec{x}) \text{ for all training examples } (\vec{x}, f(\vec{x})) \in D$$

  Here $\approx$ means that the values are somewhat close to each other w.r.t. an appropriate *error function* E.

In what follows we use the *least squares* defined by

$$E = \frac{1}{2} \sum_{(\vec{x}, f(\vec{x})) \in D} (h(\vec{x}) - f(\vec{x}))^2$$

Our goal is to minimize E.

The main reason is that this function has nice mathematical properties (as opposed e.g. to $\sum_{(\vec{x}, f(\vec{x})) \in D} |h(\vec{x}) - f(\vec{x})|$).

Linear Function Approximation

- Given a set D of training examples:

$$D = \{(\vec{x}_1, f(\vec{x}_1)), (\vec{x}_2, f(\vec{x}_2)), \ldots, (\vec{x}_p, f(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k(\vec{x}) \in \mathbb{R}$.

In what follows we use f_k to denote $f(\vec{x}_k)$.

Linear Function Approximation

- Given a set D of training examples:

$$D = \{(\vec{x}_1, f(\vec{x}_1)), (\vec{x}_2, f(\vec{x}_2)), \ldots, (\vec{x}_p, f(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k(\vec{x}) \in \mathbb{R}$.

In what follows we use f_k to denote $f(\vec{x}_k)$.

Our goal: Find $\vec{w}$ so that $h[\vec{w}](\vec{x}) = \vec{w} \cdot \tilde{\mathrm{x}}$ approximates the function f some of whose values are given by the training set.

Recall that $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$.

Linear Function Approximation

- Given a set D of training examples:

$$D = \{(\vec{x}_1, f(\vec{x}_1)), (\vec{x}_2, f(\vec{x}_2)), \ldots, (\vec{x}_p, f(\vec{x}_p))\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k(\vec{x}) \in \mathbb{R}$.

In what follows we use f_k to denote $f(\vec{x}_k)$.

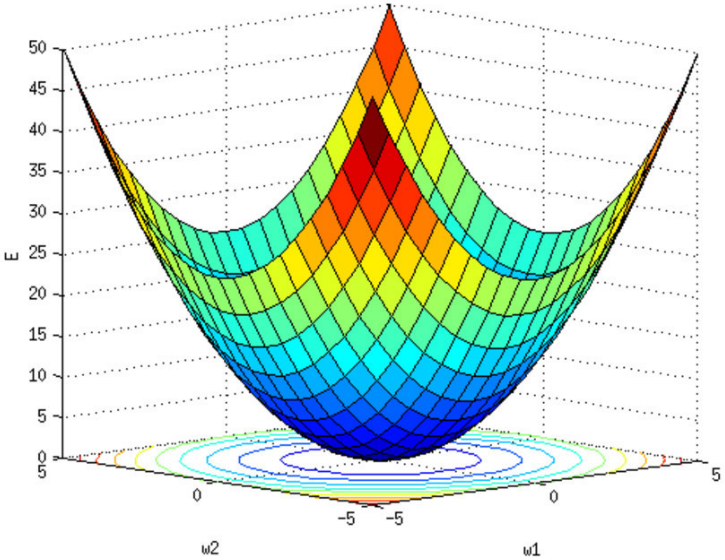
Our goal: Find $\vec{w}$ so that $h[\vec{w}](\vec{x}) = \vec{w} \cdot \tilde{\mathrm{x}}$ approximates the function f some of whose values are given by the training set.

Recall that $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$.

- **Least Squares Error Function:**

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}(\vec{w} \cdot \tilde{\mathrm{x}}_k - f_k)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=0}^{n} w_i x_{ki} - f_k\right)^2$$

Error function



Gradient of the Error Function

Consider the **gradient** of the error function:

$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w}) \right) = \sum_{k=1}^{p} (\vec{w} \cdot \tilde{\mathrm{x}}_k - f_k)) \cdot \tilde{\mathrm{x}}_k$$

What is the gradient $\nabla E(\vec{w})$? It is a vector in $\mathbb{R}^{n+1}$ which points in the direction of the steepest *ascent* of E (it's length corresponds to the steepness). Note that here the vectors $\tilde{\mathrm{x}}_k$ are *fixed* parameters of E!

Gradient of the Error Function

Consider the **gradient** of the error function:

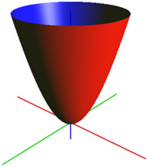
$$\nabla E(\vec{w}) = \left(\frac{\partial E}{\partial w_0}(\vec{w}), \ldots, \frac{\partial E}{\partial w_n}(\vec{w})\right) = \sum_{k=1}^{p} (\vec{w} \cdot \tilde{x}_k - f_k)) \cdot \tilde{x}_k$$

What is the gradient $\nabla E(\vec{w})$? It is a vector in $\mathbb{R}^{n+1}$ which points in the direction of the steepest *ascent* of E (it's length corresponds to the steepness). Note that here the vectors $\tilde{x}_k$ are *fixed* parameters of E!

Fakt

If $\nabla E(\vec{w}) = \vec{0} = (0, \ldots, 0)$, *then* $\vec{w}$ *is a global minimum of* E.

This follows from the fact that E is a convex paraboloid that has a unique extreme which is a minimum.



Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned} \mathcal{T} &= \{((1, 2), 1), ((1, 3), 2), ((1, 4), 5)\} \\ &= ((x_{10}, x_{11}), f_1), ((x_{20}, x_{21}), f_2), ((x_{30}, x_{31}), f_3) \end{aligned}$$

Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned} \mathcal{T} &= \{((1,2),1),((1,3),2),((1,4),5)\} \\ &= ((x_{10},x_{11}),f_1),((x_{20},x_{21}),f_2),((x_{30},x_{31}),f_3) \end{aligned}$$

$$E(w_0,w_1) = \frac{1}{2}[(w_0+w_1\cdot 2-1)^2+(w_0+w_1\cdot 3-2)^2+(w_0+w_1\cdot 4-5)^2]$$

Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned}\mathcal{T} &= \{((1,2),1),((1,3),2),((1,4),5)\} \\ &= ((x_{10},x_{11}),f_1),((x_{20},x_{21}),f_2),((x_{30},x_{31}),f_3)\end{aligned}$$

$$E(w_0,w_1) = \frac{1}{2}[(w_0+w_1\cdot 2-1)^2+(w_0+w_1\cdot 3-2)^2+(w_0+w_1\cdot 4-5)^2]$$

$\frac{\delta E}{\delta w_0}$

Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned} \mathcal{T} &= \{((1,2),1),((1,3),2),((1,4),5)\} \\ &= ((x_{10},x_{11}),f_1),((x_{20},x_{21}),f_2),((x_{30},x_{31}),f_3) \end{aligned}$$

$$E(w_0,w_1) = \frac{1}{2}[(w_0+w_1\cdot 2-1)^2+(w_0+w_1\cdot 3-2)^2+(w_0+w_1\cdot 4-5)^2]$$

$$\frac{\delta E}{\delta w_0} = (w_0+w_1\cdot 2-1)\cdot 1+(w_0+w_1\cdot 3-2)\cdot 1+(w_0+w_1\cdot 4-5)\cdot 1$$

Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned} \mathcal{T} &= \{((1,2),1),((1,3),2),((1,4),5)\} \\ &= ((x_{10},x_{11}),f_1),((x_{20},x_{21}),f_2),((x_{30},x_{31}),f_3) \end{aligned}$$

$$E(w_0,w_1) = \frac{1}{2}[(w_0+w_1\cdot 2-1)^2+(w_0+w_1\cdot 3-2)^2+(w_0+w_1\cdot 4-5)^2]$$

$$\frac{\delta E}{\delta w_0} = (w_0+w_1\cdot 2-1)\cdot 1+(w_0+w_1\cdot 3-2)\cdot 1+(w_0+w_1\cdot 4-5)\cdot 1$$

$$\frac{\delta E}{\delta w_1}$$

Gradient of the error function

Consider $n = 1$, which means that $\vec{w} = (w_0, w_1)$ and we write x instead of $\vec{x}$ since $\vec{x} \in \mathbb{R}^n = \mathbb{R}^1 = \mathbb{R}$.

Then the model is $h[\vec{w}](x) = w_0 + w_1 \cdot x$.

Consider a concrete training set:

$$\begin{aligned}
\mathcal{T} &= \{((1,2),1),((1,3),2),((1,4),5)\} \\
&= ((x_{10},x_{11}),f_1),((x_{20},x_{21}),f_2),((x_{30},x_{31}),f_3)
\end{aligned}$$

$$E(w_0,w_1) = \frac{1}{2}[(w_0+w_1\cdot 2-1)^2+(w_0+w_1\cdot 3-2)^2+(w_0+w_1\cdot 4-5)^2]$$

$$\frac{\delta E}{\delta w_0} = (w_0+w_1\cdot 2-1)\cdot 1+(w_0+w_1\cdot 3-2)\cdot 1+(w_0+w_1\cdot 4-5)\cdot 1$$
$$\frac{\delta E}{\delta w_1} = (w_0+w_1\cdot 2-1)\cdot 2+(w_0+w_1\cdot 3-2)\cdot 3+(w_0+w_1\cdot 4-5)\cdot 4$$

$$\nabla E(\vec{w}) = \left(\frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}\right) =$$
$$(w_0+w_1\cdot 2-1)\cdot(1,2)+(w_0+w_1\cdot 3-2)\cdot(1,3)+(w_0+w_1\cdot 4-5)\cdot(1,4)$$

Function Approximation – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.

Function Approximation – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)})$$

Function Approximation – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}
\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\
&= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(\vec{w}^{(t)} \cdot \tilde{x}_k - f_k \right) \cdot \tilde{x}_k \\
&= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(h[\vec{w}^{(t)}](\vec{x}_k) - f_k \right) \cdot \tilde{x}_k
\end{aligned}$$

Here $k = (t \mod p) + 1$ and $0 < \varepsilon \leq 1$ is the learning speed.

Note that the algorithm is almost similar to the batch perceptron algorithm!

Function Approximation – Learning

Gradient Descent:

- Weights $\vec{w}^{(0)}$ are initialized randomly close to $\vec{0}$.
- In $(t+1)$-th step, $\vec{w}^{(t+1)}$ is computed as follows:

$$\begin{aligned}
\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\
&= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(\vec{w}^{(t)} \cdot \tilde{\mathrm{x}}_k - f_k \right) \cdot \tilde{\mathrm{x}}_k \\
&= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^{p} \left(h[\vec{w}^{(t)}](\vec{x}_k) - f_k \right) \cdot \tilde{\mathrm{x}}_k
\end{aligned}$$

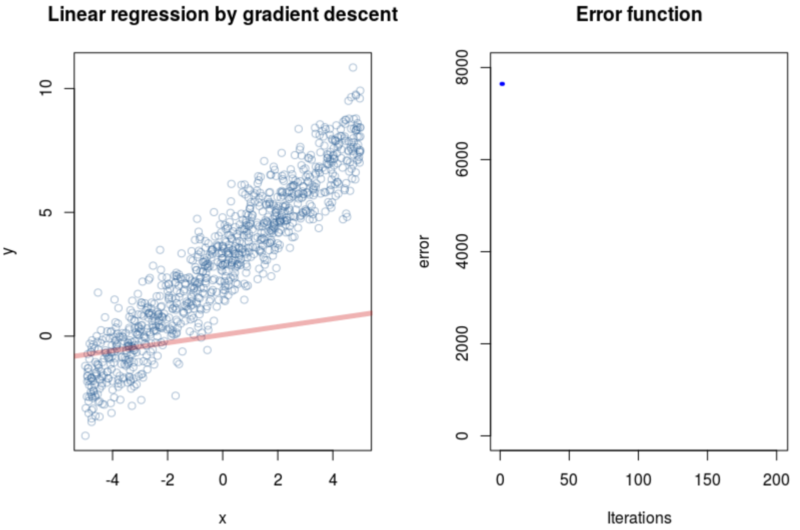
Here $k = (t \mod p) + 1$ and $0 < \varepsilon \leq 1$ is the learning speed.

Note that the algorithm is almost similar to the batch perceptron algorithm!

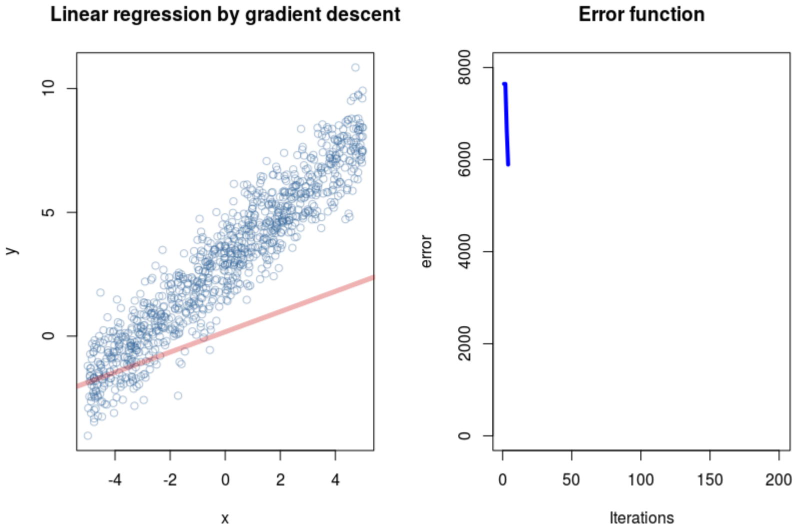
Tvrzení

For sufficiently small $\varepsilon > 0$ the sequence $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ converges (component-wisely) to the global minimum of E.

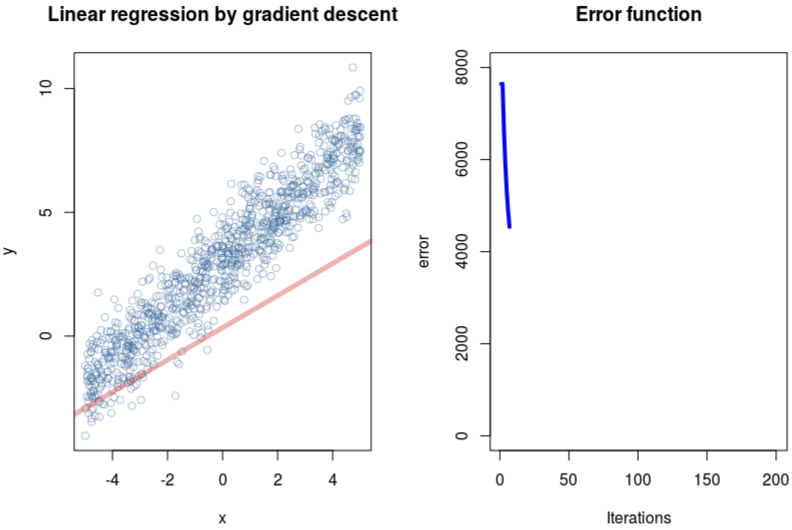
Linear regression - animation



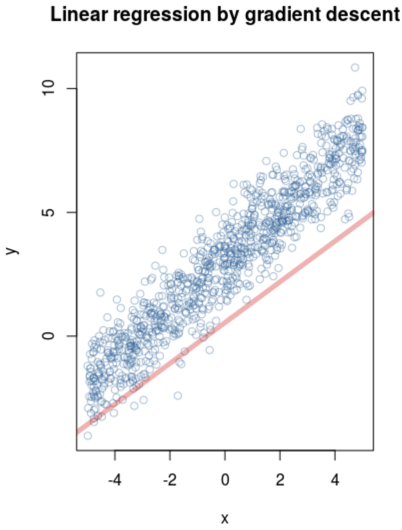
Linear regression - animation

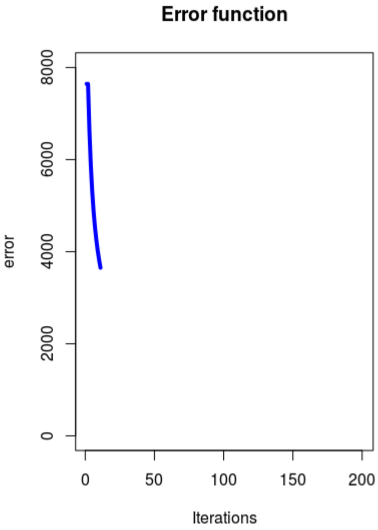


Linear regression - animation

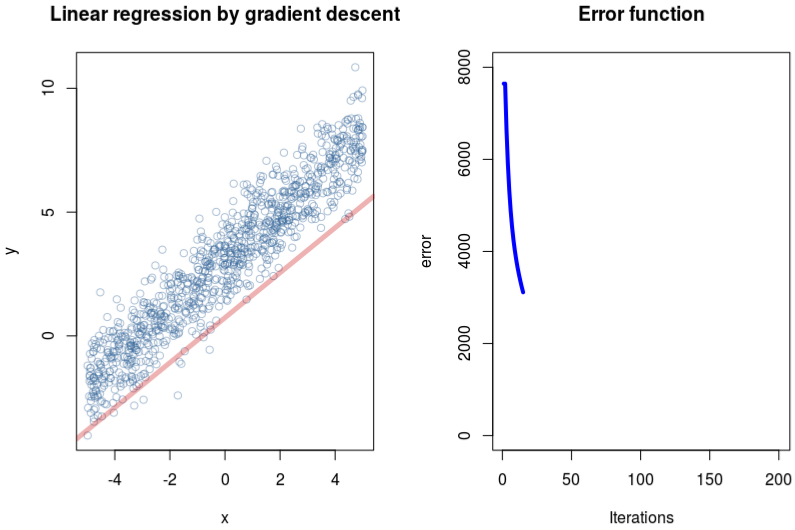


Linear regression - animation

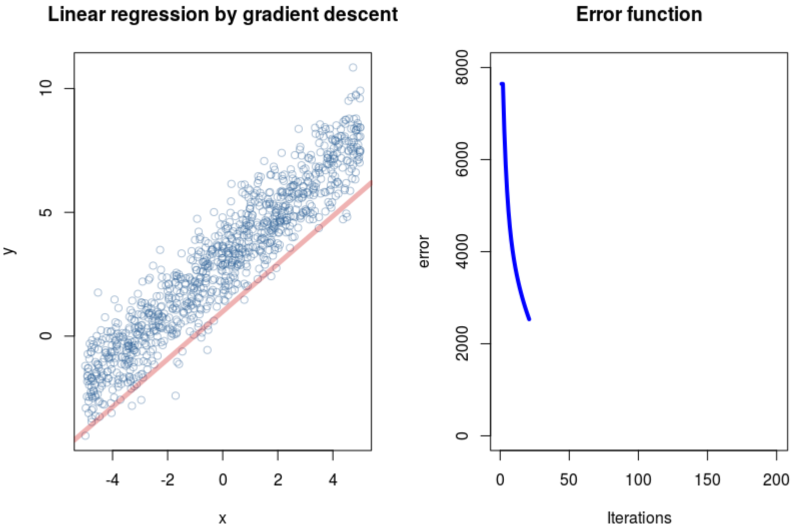




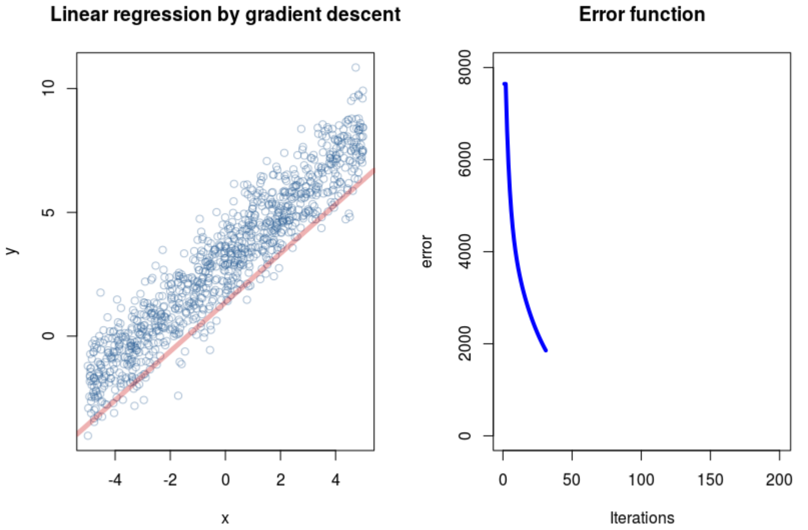
Linear regression - animation



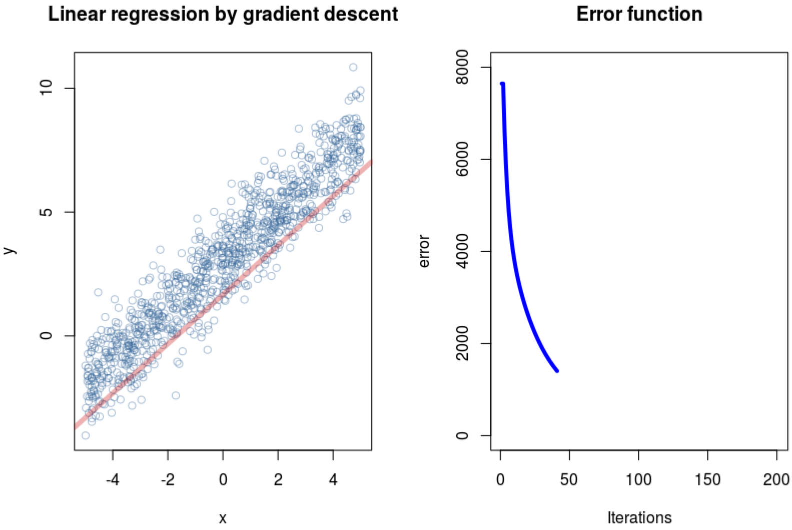
Linear regression - animation



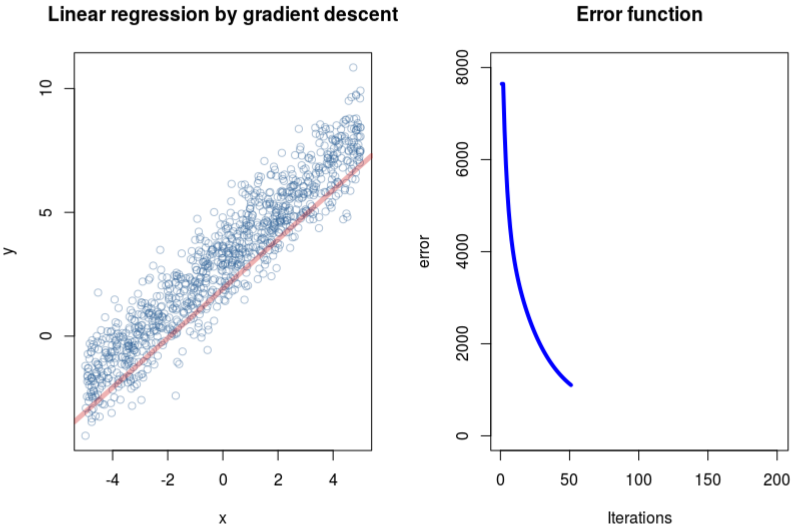
Linear regression - animation



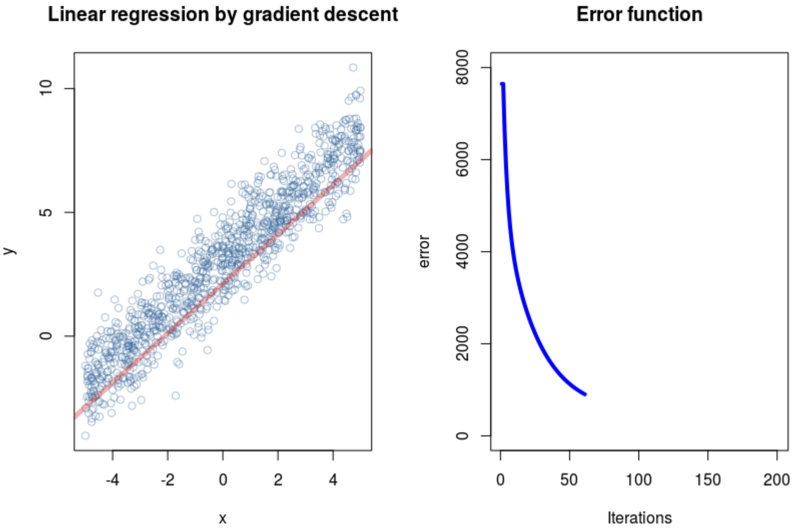
Linear regression - animation



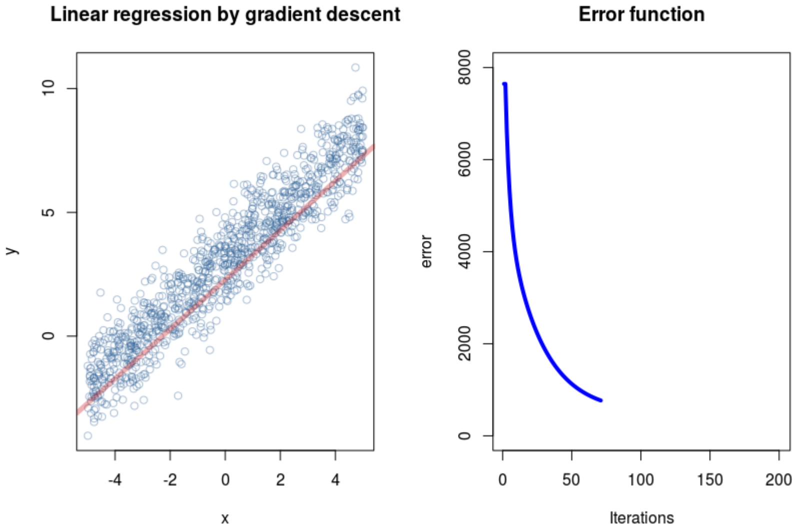
Linear regression - animation



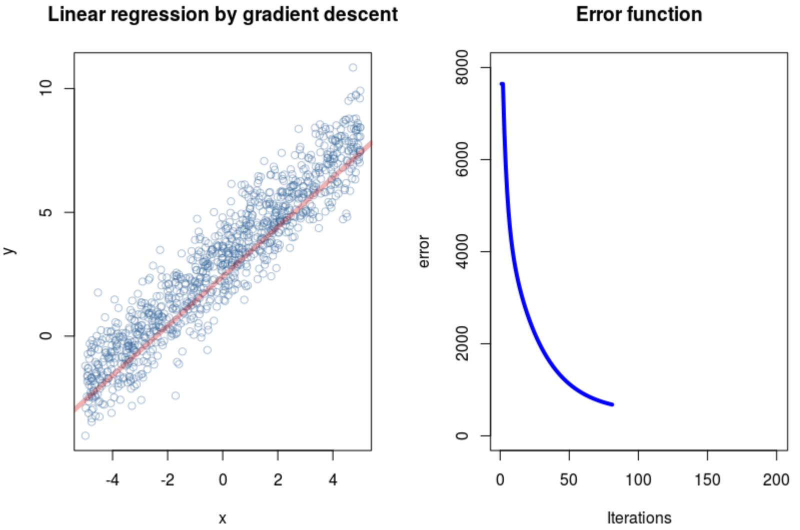
Linear regression - animation



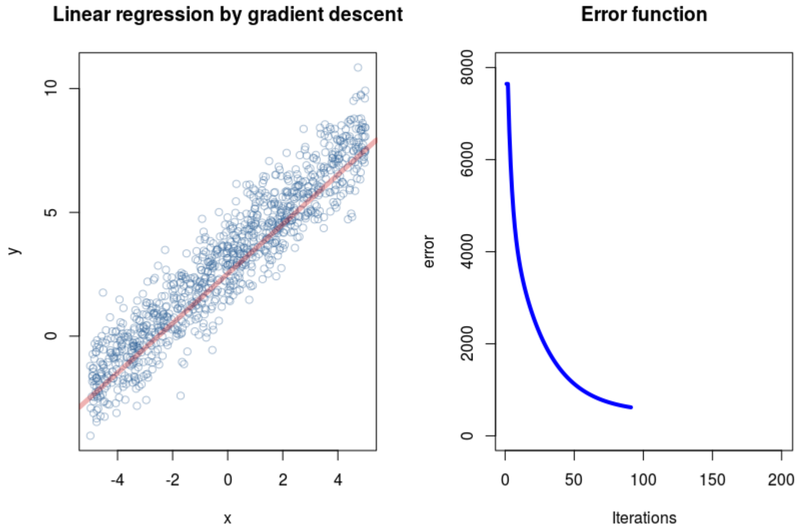
Linear regression - animation



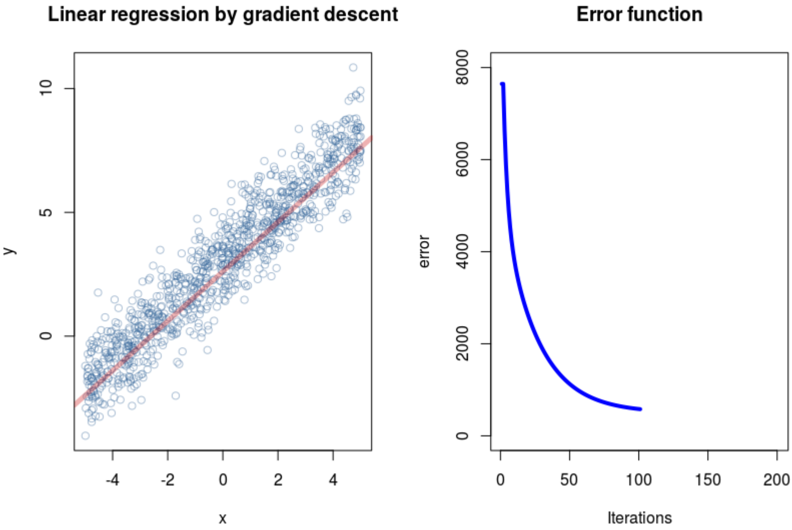
Linear regression - animation



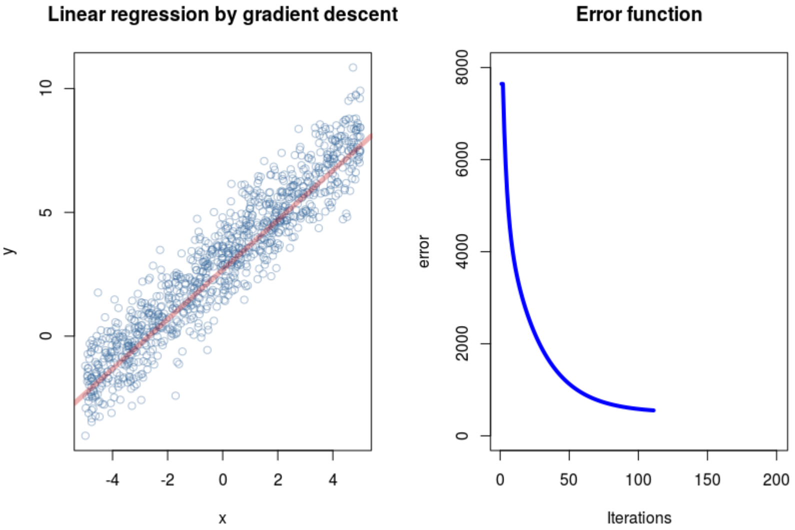
Linear regression - animation



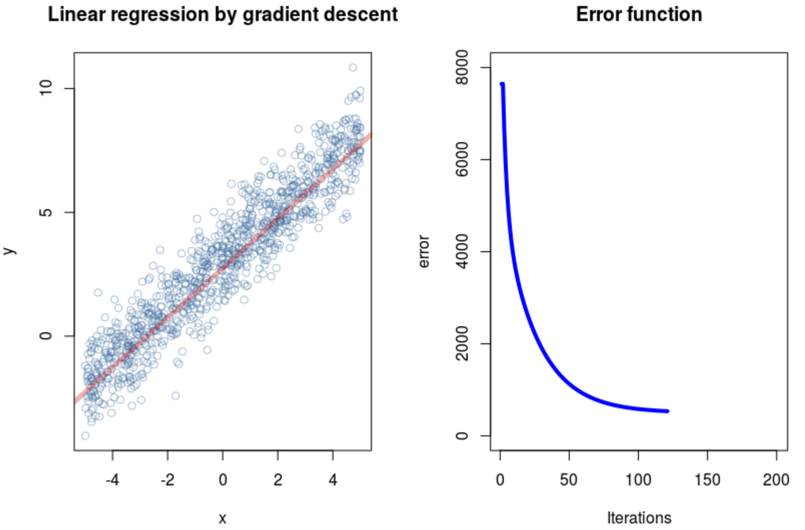
Linear regression - animation



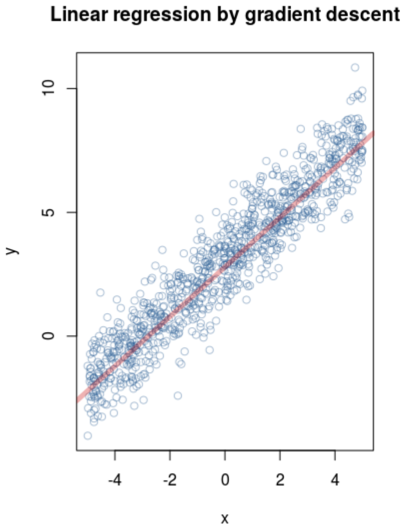
Linear regression - animation

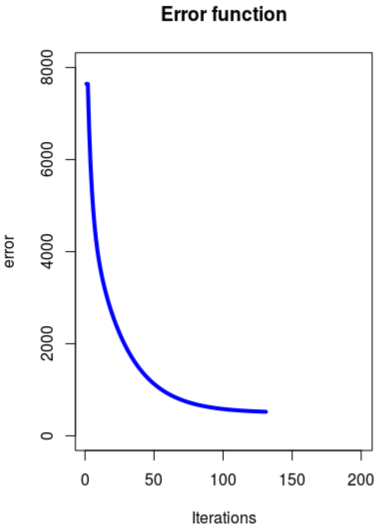


Linear regression - animation

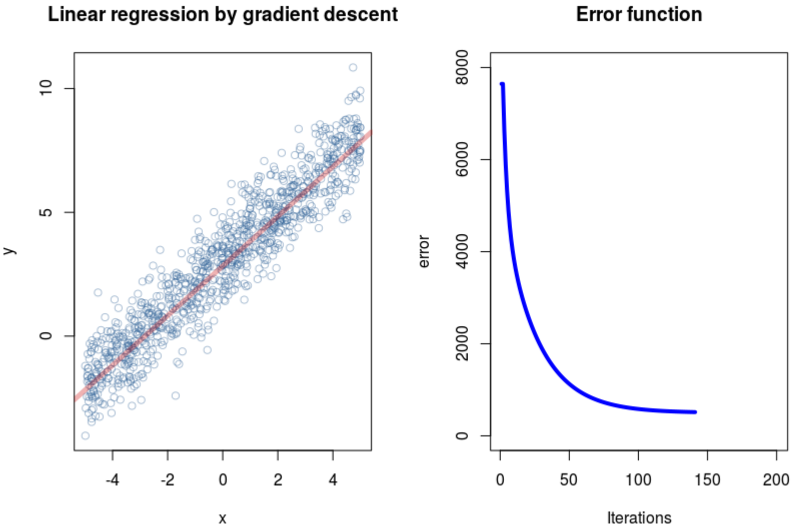


Linear regression - animation

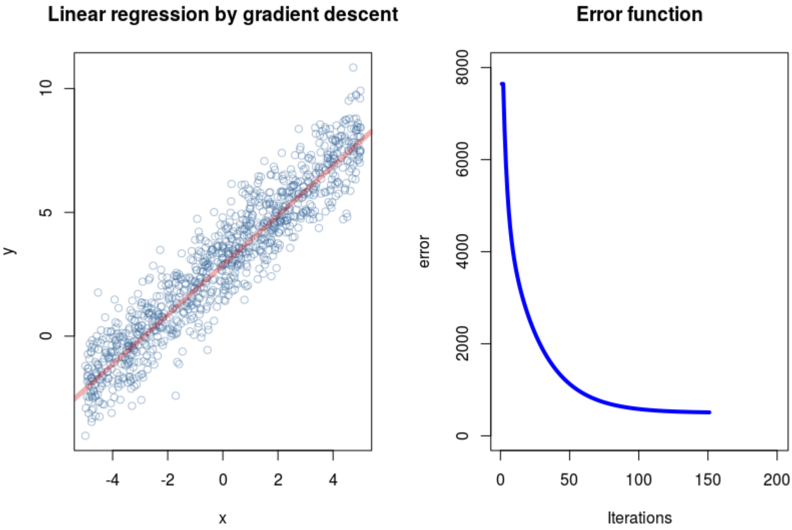




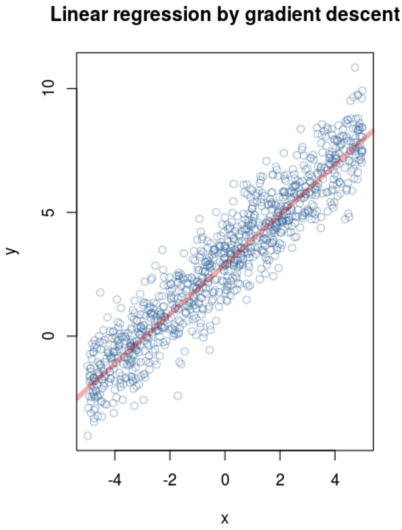
Linear regression - animation

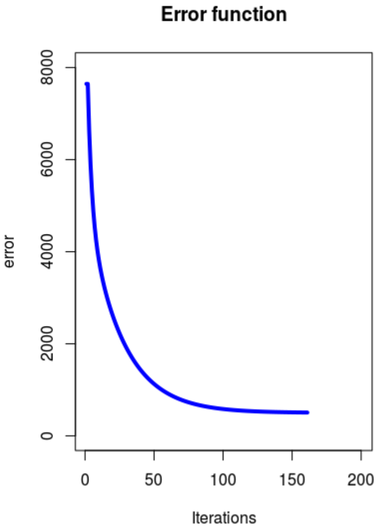


Linear regression - animation

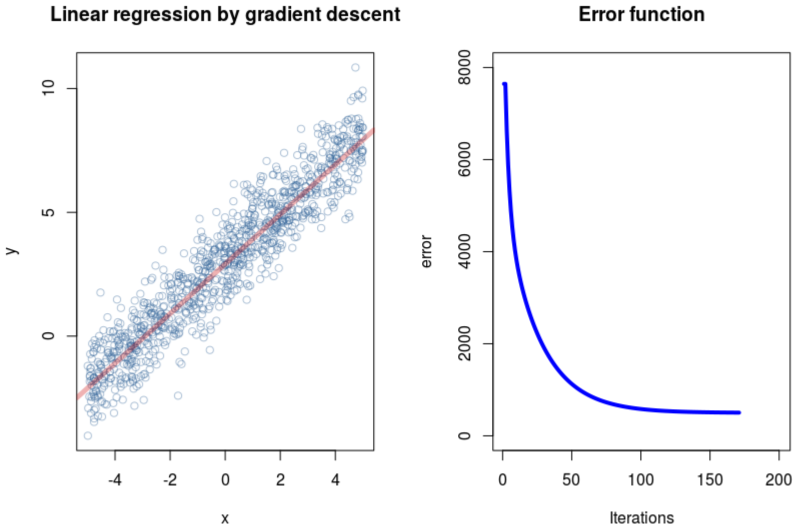


Linear regression - animation

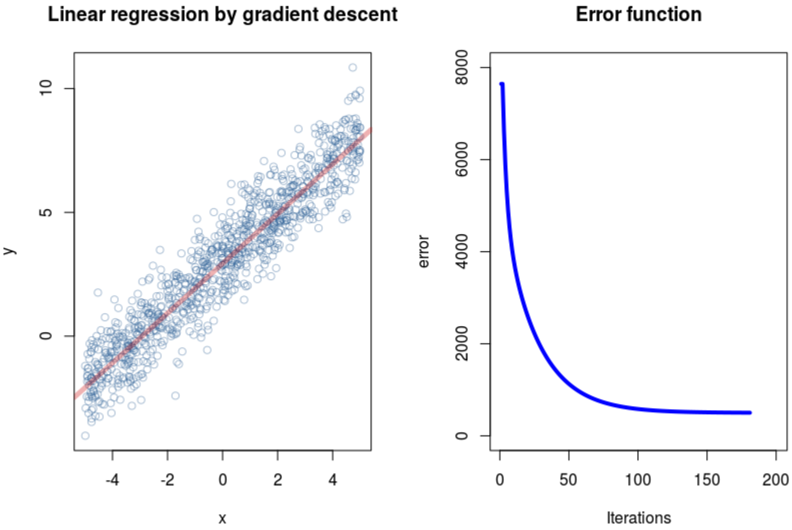




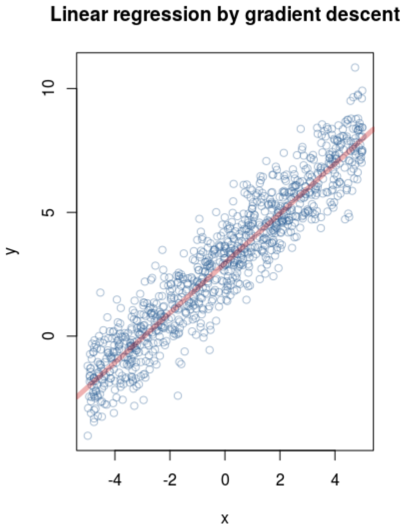
Linear regression - animation

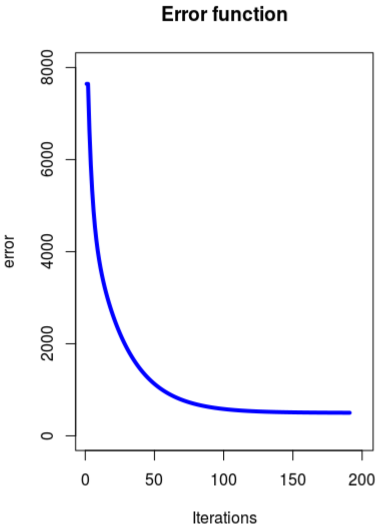


Linear regression - animation

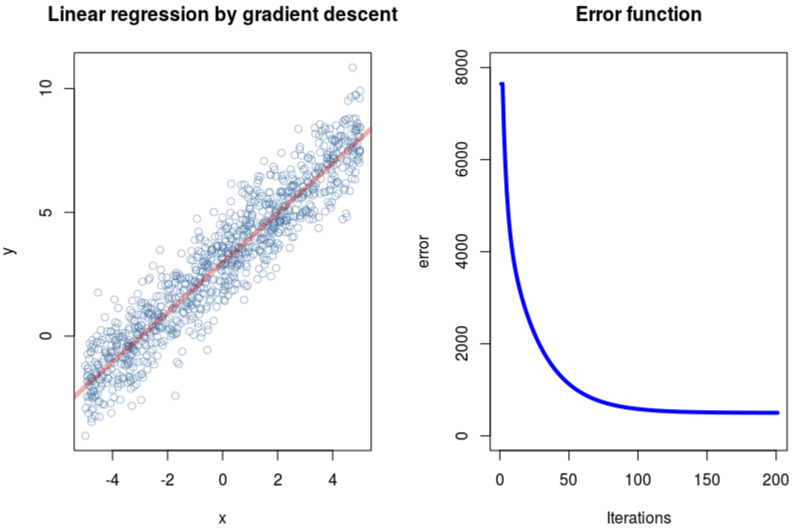


Linear regression - animation





Linear regression - animation



Finding the Minimum in Dimension One

Assume $n = 1$. Then the error function E is

$$E(w_0, w_1) = \frac{1}{2}\sum_{k=1}^{p}(w_0 + w_1 x_k - f_k)^2$$

Finding the Minimum in Dimension One

Assume $n = 1$. Then the error function E is

$$E(w_0, w_1) = \frac{1}{2} \sum_{k=1}^{p} (w_0 + w_1 x_k - f_k)^2$$

Minimize E w.r.t. w_0 a w_1:

$$\frac{\delta E}{\delta w_0} = 0 \quad \Leftrightarrow \quad w_0 = \bar{f} - w_1 \bar{x} \quad \Leftrightarrow \quad \bar{f} = w_0 + w_1 \bar{x}$$

where $\bar{x} = \frac{1}{p} \sum_{k=1}^{p} x_k$ a $\bar{f} = \frac{1}{p} \sum_{k=1}^{p} f_k$

Finding the Minimum in Dimension One

Assume $n = 1$. Then the error function E is

$$E(w_0, w_1) = \frac{1}{2} \sum_{k=1}^{p} (w_0 + w_1 x_k - f_k)^2$$

Minimize E w.r.t. w_0 a w_1:

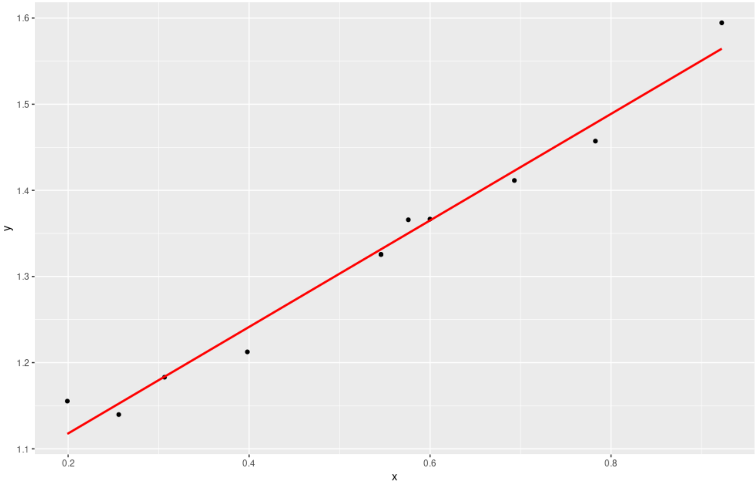
$$\frac{\delta E}{\delta w_0} = 0 \quad \Leftrightarrow \quad w_0 = \bar{f} - w_1 \bar{x} \quad \Leftrightarrow \quad \bar{f} = w_0 + w_1 \bar{x}$$

where $\bar{x} = \frac{1}{p} \sum_{k=1}^{p} x_k$ a $\bar{f} = \frac{1}{p} \sum_{k=1}^{p} f_k$

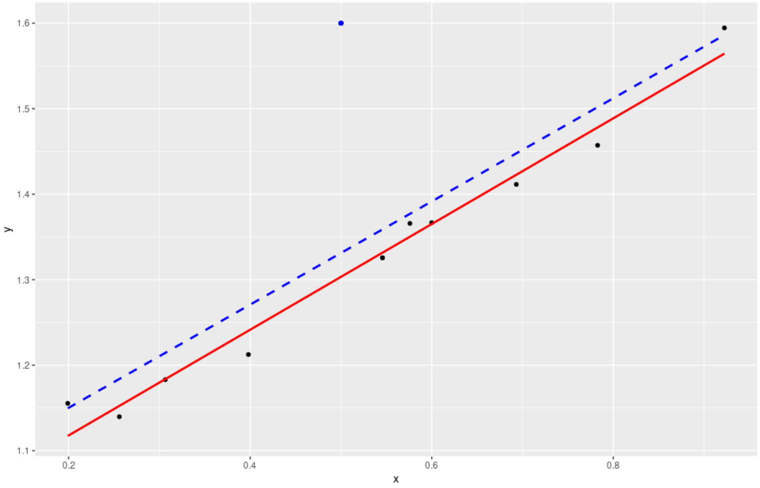
$$\frac{\delta E}{\delta w_1} = 0 \quad \Leftrightarrow \quad w_1 = \frac{\frac{1}{p} \sum_{k=1}^{p} (f_k - \bar{f})(x_k - \bar{x})}{\frac{1}{p} \sum_{k=1}^{p} (x_k - \bar{x})^2}$$

i.e. $w_1 = cov(f, x)/var(x)$

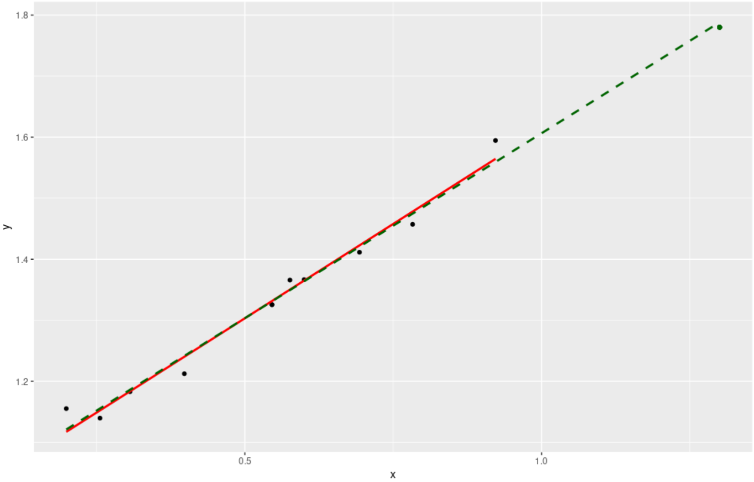
Effect of Outliers



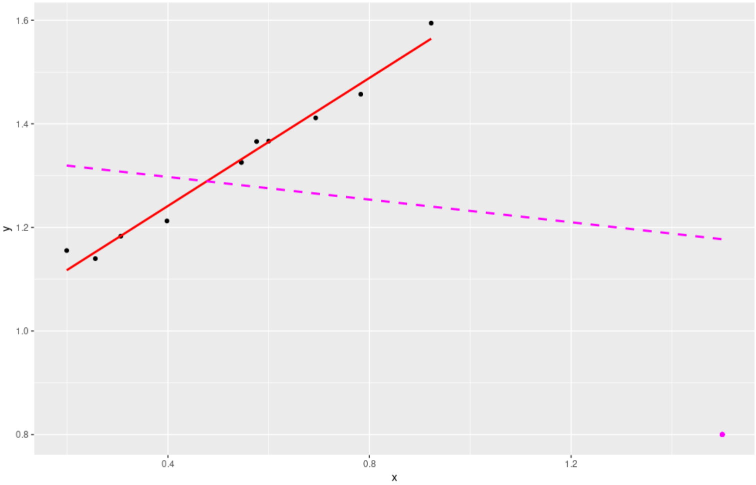
Effect of Outliers



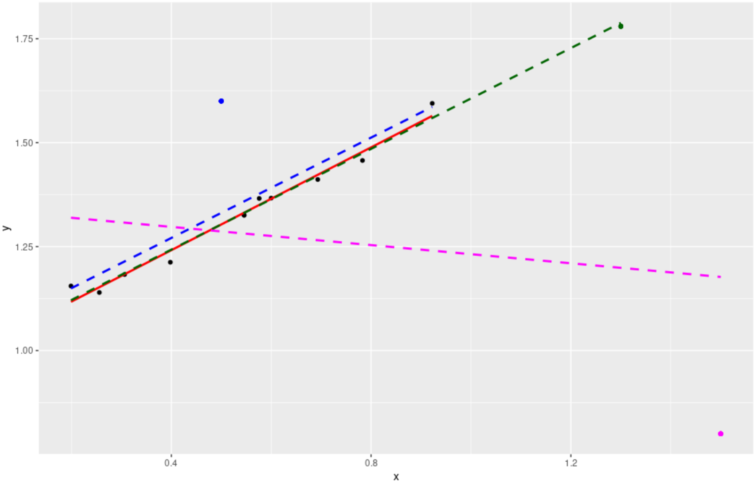
Effect of Outliers



Effect of Outliers



Effect of Outliers

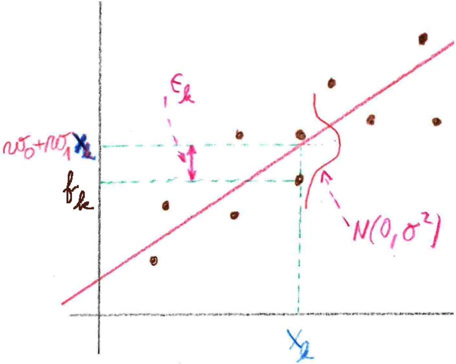


Maximum Likelihood vs Least Squares (Dim 1)

Fix a training set $D = \{(x_1, f_1), (x_2, f_2), \ldots, (x_p, f_p)\}$
Assume that each f_k has been generated randomly by

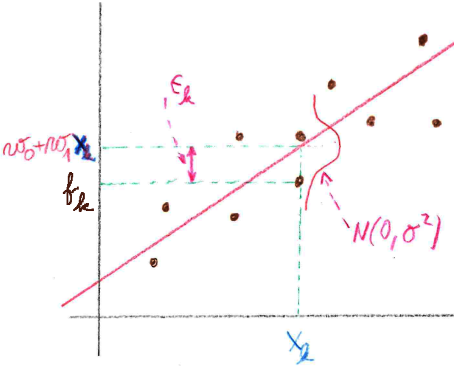
$$f_k = (w_0 + w_1 \cdot x_k) + \epsilon_k$$

where w_0, w_1 are **unknown weights**, and ϵ_k are independent, normally distributed noise values with mean 0 and some variance σ^2



How "probable" is it to generate the correct $f_1, \ldots, f_p$?

Maximum Likelihood vs Least Squares (Dim 1)



How "probable" is it to generate the correct $f_1, \ldots, f_p$?

The following conditions are equivalent:

- w_0, w_1 minimize the squared error E
- w_0, w_1 maximize the likelihood (i.e., the "probability") of generating the correct values $f_1, \ldots, f_p$ using $f_k = (w_0 + w_1 \cdot x_k) + \epsilon_k$

Comments on Linear Models

- Linear models are parametric, i.e., they have a fixed form with a small number of parameters that need to be learned from data (as opposed, e.g., to decision trees where the structure is not fixed in advance).

Comments on Linear Models

- Linear models are parametric, i.e., they have a fixed form with a small number of parameters that need to be learned from data (as opposed, e.g., to decision trees where the structure is not fixed in advance).
- Linear models are stable, i.e., small variations in the training data have only limited impact on the learned model. (tree models typically vary more with the training data).

Comments on Linear Models

- Linear models are parametric, i.e., they have a fixed form with a small number of parameters that need to be learned from data (as opposed, e.g., to decision trees where the structure is not fixed in advance).
- Linear models are stable, i.e., small variations in the training data have only limited impact on the learned model. (tree models typically vary more with the training data).
- Linear models are less likely to overfit (low variance) the training data but sometimes tend to underfit (high bias).

Comments on Linear Models

- Linear models are parametric, i.e., they have a fixed form with a small number of parameters that need to be learned from data (as opposed, e.g., to decision trees where the structure is not fixed in advance).
- Linear models are stable, i.e., small variations in the training data have only limited impact on the learned model. (tree models typically vary more with the training data).
- Linear models are less likely to overfit (low variance) the training data but sometimes tend to underfit (high bias).
- Linear models are prone to outliers.