CS 2750 Machine Learning
Lecture 6

Linear regression

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Administration

- **Matlab:**
  - Statistical and neural network toolboxes are not available on unixs machines
  - Please use Windows Machines in CSSD labs

Outline

Regression

- Linear model
- Error function based on the least squares fit.
- Parameter estimation.
- Gradient methods.
- On-line regression techniques.
- Linear additive models.

Supervised learning

Data: $D = \{D_1, D_2, .., D_n\}$ **a set of *n* examples**

$D_i = < \mathbf{x}_i, y_i >$

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \cdots x_{i,d})$ is an input vector of size d

y_i is the desired output (given by a teacher)

Objective: learn the mapping $f : X \rightarrow Y$

s.t. $y_i \approx f(\mathbf{x}_i)$ for all $i = 1,.., n$

- **Regression:** Y is **continuous**

  Example: earnings, product orders $\rightarrow$ company stock price

- **Classification:** Y is **discrete**

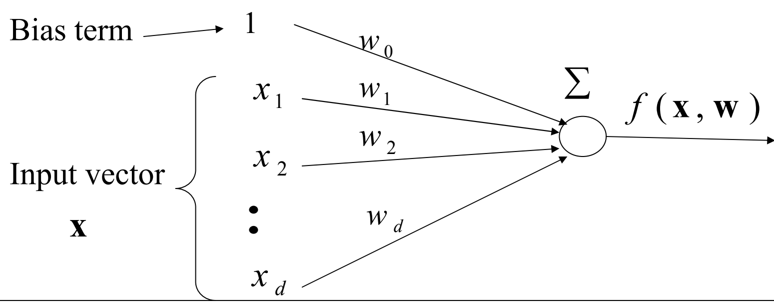
  Example: handwritten digit in binary form $\rightarrow$ digit label

Linear regression

- **Function** $f : X \rightarrow Y$ is a linear combination of input components

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j$$

$w_0, w_1, \ldots w_k$ **- parameters (weights)**



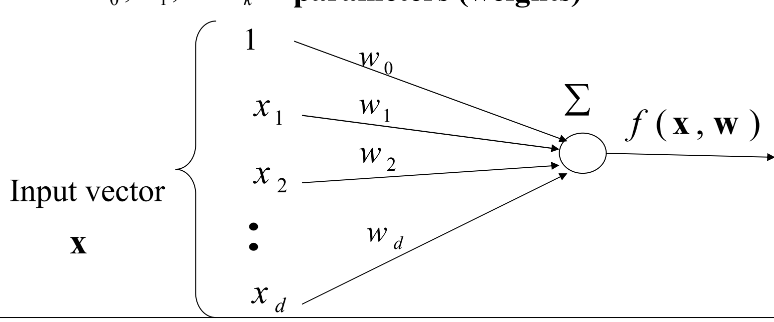
Linear regression

- **Shorter (vector) definition of the model**
  - Include bias constant in the input vector

$$\mathbf{x} = (1, x_1, x_2, \cdots x_d)$$

$$f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = \mathbf{w}^T \mathbf{x}$$

$w_0, w_1, \ldots w_k$ **- parameters (weights)**



Linear regression. Error.

- **Data:** $D_i = < \mathbf{x}_i, y_i >$
- **Function:** $\mathbf{x}_i \rightarrow f(\mathbf{x}_i)$
- We would like to have $y_i \approx f(\mathbf{x}_i)$ for all $i = 1,.., n$

- **Error function** measures how much our predictions deviate from the desired answers

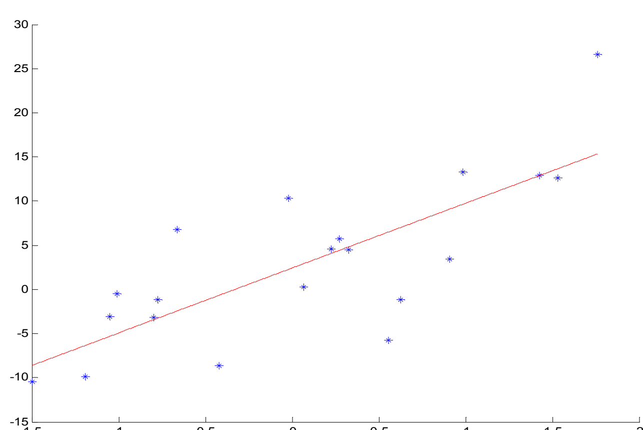
Mean-squared error $J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$

- **Learning:**

  We want to find the weights minimizing the error !

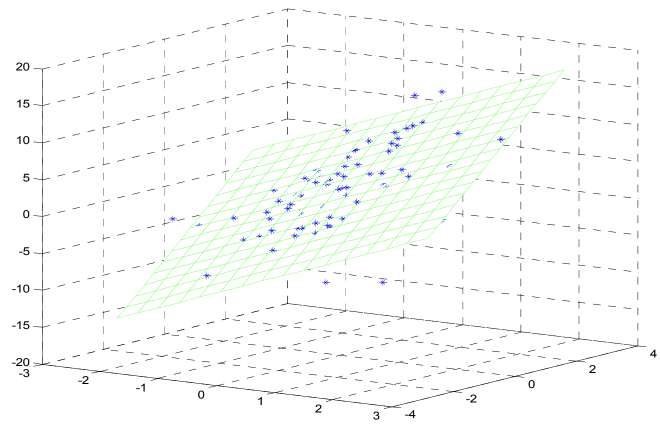
Linear regression. Example

- 1 dimensional input $\mathbf{x} = (x_1)$



Linear regression. Example.

- 2 dimensional input $\mathbf{x} = (x_1, x_2)$



Linear regression. Optimization.

- We want the **weights minimizing the error**

$$J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2 = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0$$

- **Vector of derivatives:**

$$\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)\mathbf{x}_i = \overline{\mathbf{0}}$$

Solving linear regression

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d})x_{i,j} = 0$$

$$\nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \overline{\mathbf{0}}$$

By rearranging the terms we get a **system of linear equations** with *d+1* unknowns

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

Equation for the *j*th component:

$$w_0\sum_{i=1}^{n}x_{i,0}x_{i,j} + w_1\sum_{i=1}^{n}x_{i,1}x_{i,j} + \ldots + w_j\sum_{i=1}^{n}x_{i,j}x_{i,j} + \ldots + w_d\sum_{i=1}^{n}x_{i,d}x_{i,j} = \sum_{i=1}^{n}y_i x_{i,j}$$

Can be solved through matrix inversion, if the matrix is not singular

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$$

Solving linear regression

- Things can be rewritten also in terms of data matrices $\mathbf{X}$ and vectors:

$$J_n(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\nabla J_n(\mathbf{w}) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

- Set derivatives to **0 and solve**

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- What if $\mathbf{X}$ is singular?
- Some columns of the data matrix are linearly dependent
- Then $\mathbf{X}^T\mathbf{X}$ is singular. Multiple possible solutions exist.
- **Remedy:** drop the redundant (linearly dependent) columns

Solving linear regression

- Linear regression problem comes down to the problem of solving a set of linear equations
- Alternative methods: **gradient descent**
  - Iterative method

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J_n(\mathbf{w})$$

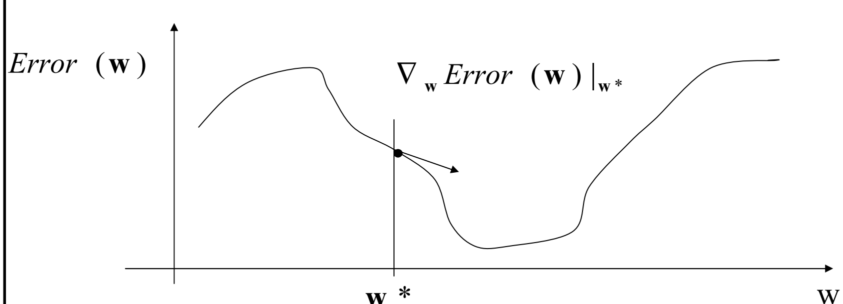
$$J_n(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\nabla J_n(\mathbf{w}) = -2\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha 2\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Gradient descent method

- Descend to the minimum of the function using the gradient information



- Change the value of **w** according to the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J_n(\mathbf{w})$$

Online regression algorithm

- The error function defined for the whole dataset D

$$J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$$

- Instead of the error for all data points we **use error for an individual sample**

$$J_{\text{online}} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$$

- **Change regression weights after every example according to the gradient (delta rule)**:

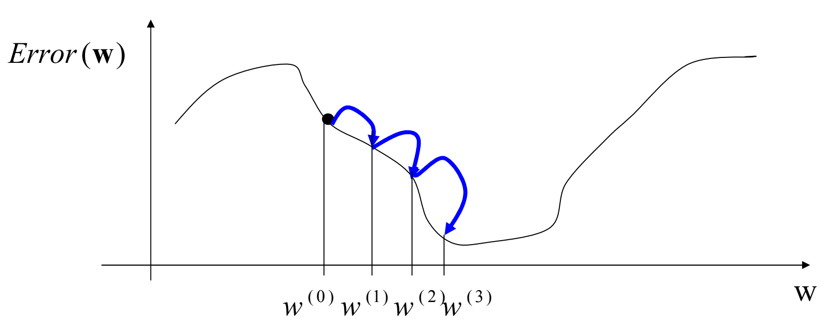
$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$ - Learning rate that depends on the number of updates

On-line gradient descent method

- In every step update weights according to a new example



Gradient for on-line learning

Linear model $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

On-line error $Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$

(i+1)-th update for the linear model:

$$\mathbf{w}^{(i+1)} \leftarrow \mathbf{w}^{(i)} - \alpha(i+1)\nabla_{\mathbf{w}} Error_i(\mathbf{w})|_{\mathbf{w}^{(i)}} = \mathbf{w}^{(i)} + \alpha(i+1)(y_i - f(\mathbf{x}_i))\mathbf{x}_i$$

Typical learning rate $\alpha(i) \approx \frac{1}{i}$

On-line algorithm: repeat online updates for all data points

Online regression algorithm

Online-linear-regression (*D, number of iterations*)

Initialize weights $\mathbf{w} = (w_0, w_1, w_2 \dots w_d)$

for i=1:1*: number of iterations*

do **select** a data point $D_i = (\mathbf{x}_i, y_i)$ from D

set $\alpha = 1/i$

update weight vector

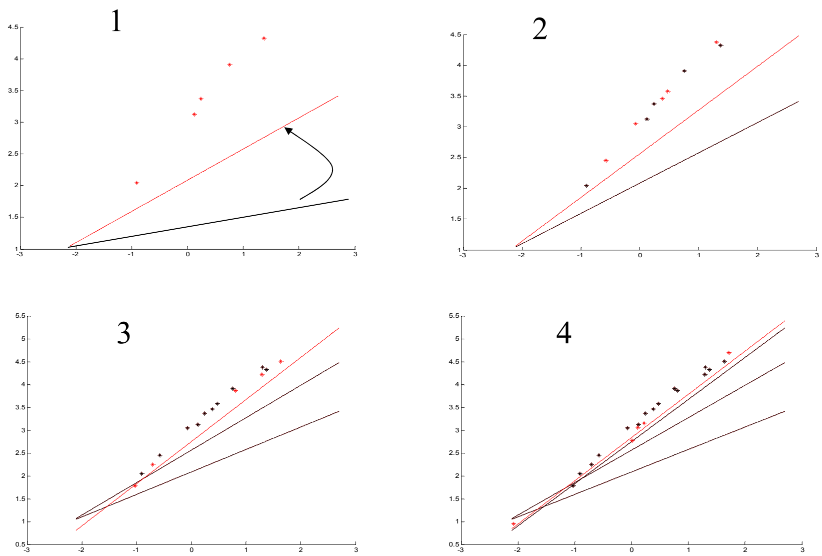
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

end for

return weights $\mathbf{w}$

- **Advantages:** very easy to implement, continuous data streams

On-line learning. Example



Practical concerns: Input normalization

- **Input normalization**
  – makes the data vary roughly on the same scale.
  – Can make a huge difference in **on-line learning**

Assume on-line update (delta) rule for two weights *j,k,*:

$$w_j \leftarrow w_j + \boxed{\alpha(i)(y_i - f(\mathbf{x}_i))} x_{i,j}$$

$$=$$

$$w_k \leftarrow w_k + \boxed{\alpha(i)(y_i - f(\mathbf{x}_i))} x_{i,k}$$

Change depends on the magnitude of the input

For inputs with a large magnitude the change in the weight is huge: changes to the inputs with high magnitude disproportional as if the input was more important

Input normalization

- **Input normalization**:
  - Solution to the problem of different scales
  - Makes all inputs vary in the same range around 0

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j} \qquad \sigma_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i,j} - \bar{x}_j)^2$$

New input: $\tilde{x}_{i,j} = \dfrac{(x_{i,j} - \bar{x}_j)}{\sigma_j}$

More complex normalization approach can be applied when we want to process data with correlations

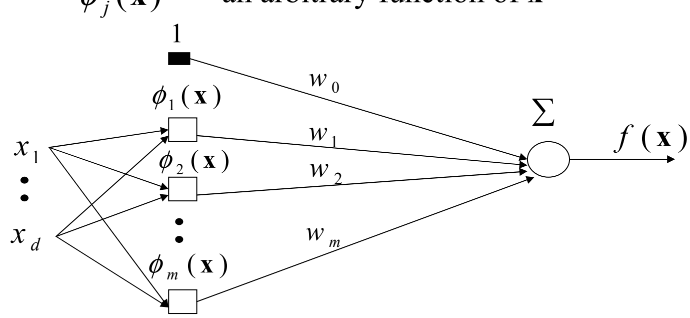
Similarly we can renormalize outputs $\boldsymbol{y}$

Extensions of simple linear model

Replace inputs to linear units with **feature (basis) functions** to model **nonlinearities**

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

$\phi_j(\mathbf{x})$ - an arbitrary function of $\mathbf{x}$



The same techniques as before to learn the weights

Additive linear models

- **Models linear in the parameters we want to fit**

$$f(\mathbf{x}) = w_0 + \sum_{k=1}^{m} w_k \phi_k(\mathbf{x})$$

$w_0, w_1 \ldots w_m$ - parameters

$\phi_1(\mathbf{x}), \phi_2(\mathbf{x}) \ldots \phi_m(\mathbf{x})$ - **feature or basis functions**

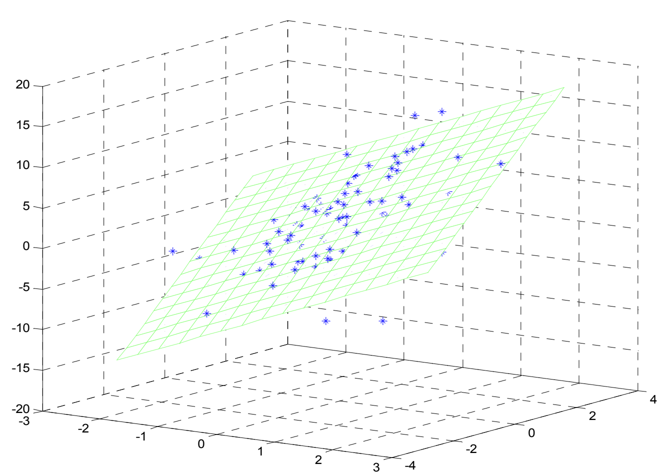
- **Basis functions examples:**
  - a higher order polynomial, one-dimensional input $\mathbf{x} = (x_1)$

    $\phi_1(x) = x \quad \phi_2(x) = x^2 \quad \phi_3(x) = x^3$
  - Multidimensional quadratic $\mathbf{x} = (x_1, x_2)$

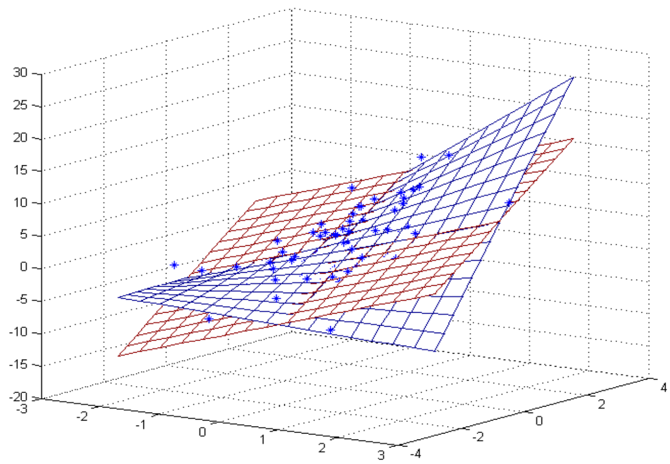
    $\phi_1(\mathbf{x}) = x_1 \quad \phi_2(\mathbf{x}) = x_1^2 \quad \phi_3(\mathbf{x}) = x_2 \quad \phi_4(\mathbf{x}) = x_2^2 \quad \phi_5(\mathbf{x}) = x_1 x_2$
  - Other types of basis functions

    $\phi_1(x) = \sin x \quad \phi_2(x) = \cos x$

Multidimensional additive model example



Multidimensional additive model example



Fitting additive linear models

- **Error function** $J_n(\mathbf{w}) = 1/n \sum_{i=1,..n} (y - f(\mathbf{x}_i))^2$

Assume: $\boldsymbol{\varphi}(\mathbf{x}_i) = (1, \phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \dots, \phi_m(\mathbf{x}_i))$

$$\nabla_{\mathbf{w}} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))\boldsymbol{\varphi}(\mathbf{x}_i) = \overline{\mathbf{0}}$$

- Leads to a **system of *m* linear equations**

$$w_0 \sum_{i=1}^{n} 1\phi_j(\mathbf{x}_i) + \ldots + w_j \sum_{i=1}^{n} \phi_j(\mathbf{x}_i)\phi_j(\mathbf{x}_i) + \ldots + w_m \sum_{i=1}^{n} \phi_m(\mathbf{x}_i)\phi_j(\mathbf{x}_i) = \sum_{i=1}^{n} y_i \phi_j(\mathbf{x}_i)$$

- Can be solved exactly like the linear case

Statistical model of regression

- **A model**:

$$y = f(\mathbf{x}, \mathbf{w}) + \varepsilon, \quad \text{s.t.} \quad \varepsilon \sim N(0, \sigma^2)$$

- The noise models deviations from the parametric linear model
- The model defines the conditional density of y given x

$$p(y \mid \mathbf{x})$$

- Allows not only to predict means but also tries to explain the nature of deviations from it
- As a result we can compute, for a given set of parameters w, σ the probability of a specific prediction

$$p(y \mid \mathbf{x}, \mathbf{w}, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-\frac{1}{2\sigma^2}(y - f(\mathbf{x}, \mathbf{w}))^2]$$

ML estimation of the parameters

- Given the distribution we can compute the probability of all samples *(x, y)* observed in the dataset *D* (values of *y* drawn independently)

$$L(D, \mathbf{w}, \sigma) = \prod_{i=1}^{n} p(y_i \mid \mathbf{x_i}, \mathbf{w}, \sigma)$$

- We want to find the optimal set of parameters
- To do this we can optimize $\mathbf{w}^* = \arg\max_{w} \prod_{i=1}^{n} p(y_i \mid \mathbf{x_i}, \mathbf{w}, \sigma)$

$$l(D, \mathbf{w}, \sigma) = \log(L(D, \mathbf{w}, \sigma)) = \log \prod_{i=1}^{n} p(y_i \mid \mathbf{x_i}, \mathbf{w}, \sigma)$$

Working the math we get

$$= \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x_i}, \mathbf{w}, \sigma) = \sum_{i=1}^{n} \left\{ -\frac{1}{2\sigma^2}(y_i - f(\mathbf{x_i}, \mathbf{w}))^2 - c(\sigma) \right\}$$

$$= -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f(\mathbf{x_i}, \mathbf{w}))^2 + C(\sigma)$$

Equivalent to LSF !!!

ML estimation of parameters

- Loss function and the log likelihood the output are related

$$Loss\,(y_i, \mathbf{x_i}) = \frac{1}{2\sigma^2} \log\ p(y_i \mid \mathbf{x_i}, \mathbf{w}, \sigma) + c(\sigma)$$

- We know how to optimize parameters ***w*** (for a given and fixed variance) – the same approach as for the LSF criterion
- How to estimate the variance of the noise?
- Maximize $l(D, \mathbf{w}, \sigma)$ with respect to variance

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x_i}, \mathbf{w}^*))^2$$

= mean square prediction error for the best predictor