CS 2750 Machine Learning
Lecture 7

Linear regression

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

Outline

Regression

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

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 $\longrightarrow$ company stock price
- **Classification:** Y is **discrete**

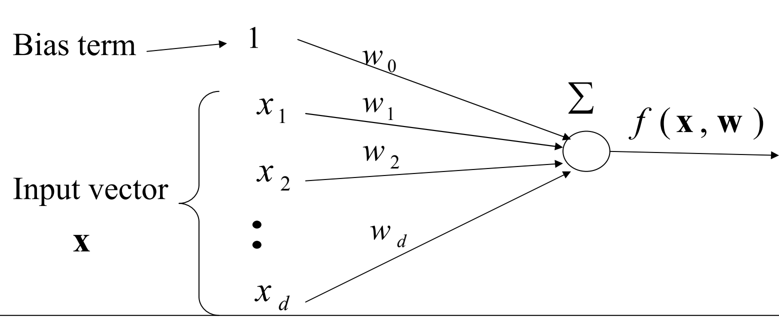
  Example: handwritten digit in binary form $\longrightarrow$ digit label

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)**



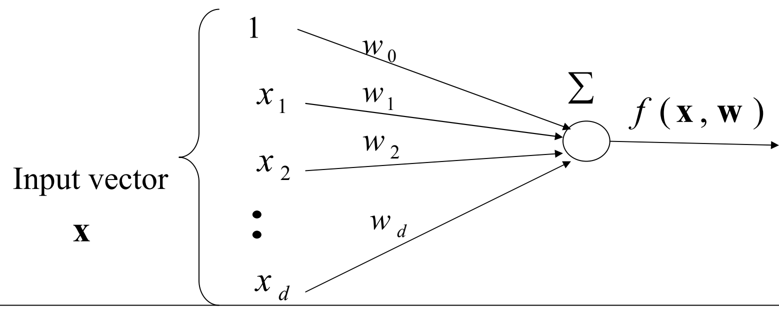
Linear regression

- **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

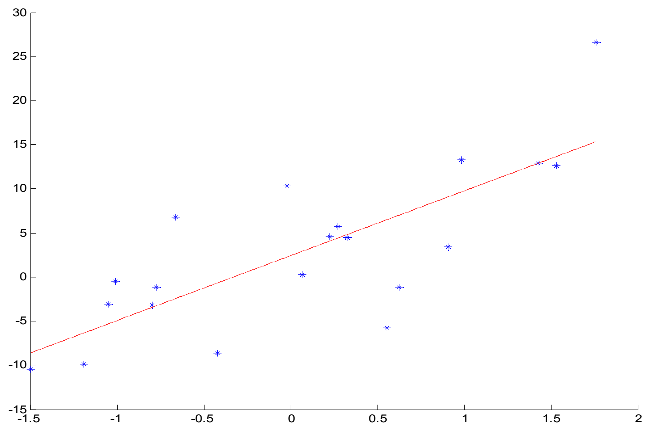
Mean-squared error $J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$

- **Learning:**

  We want to find the weights minimizing the error !

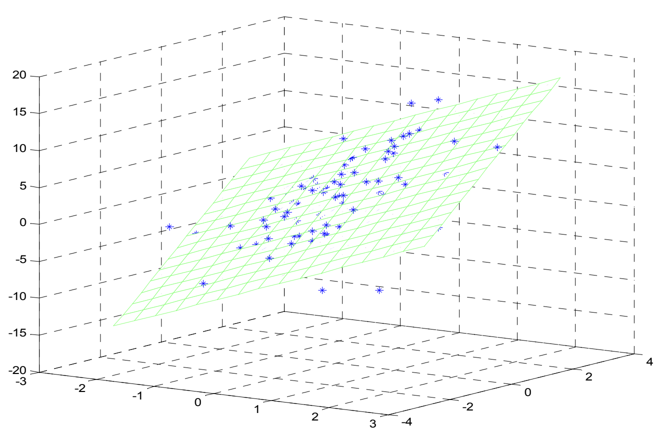
Linear regression. Example

- 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:**

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

Linear regression. Optimization.

- $\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \bar{\mathbf{0}}$ defines a set of equations in **w**

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

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

$$\ldots$$

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

$$\ldots$$

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

Solving linear regression

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

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

$$\boxed{\mathbf{Aw} = \mathbf{b}}$$

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

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

$$\bullet\bullet\bullet$$

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

$$\bullet\bullet\bullet$$

Solving linear regression

- The optimal set of weights satisfies:

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

Leads to a **system of linear equations (SLE)** with *d+1* unknowns of the form

$$\mathbf{Aw} = \mathbf{b}$$

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

Solution to SLE: ?

Solving linear regression

- The optimal set of weights satisfies:

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

Leads to a **system of linear equations (SLE)** with *d+1* unknowns of the form

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

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

Solution to SLE:

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

- matrix inversion

Gradient descent solution

Goal: the weight optimization in the linear regression model

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

An alternative to SLE solution:

- **Gradient descent**

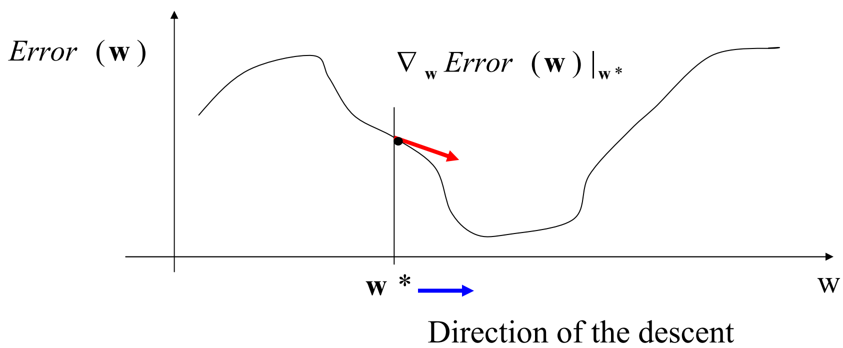
  Idea:
  - Adjust weights in the direction that improves the Error
  - The gradient tells us what is the right direction

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

$\alpha > 0$ - a **learning rate** (scales the gradient changes)

Gradient descent method

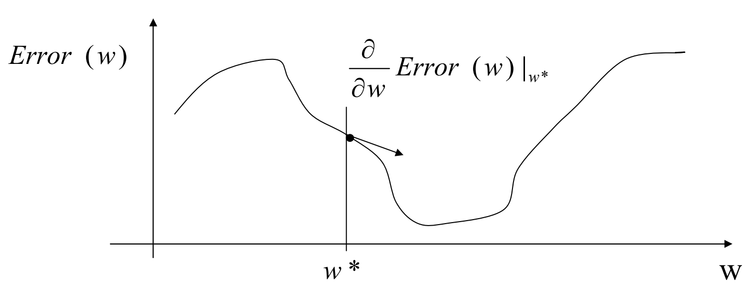
- Descend using the gradient information



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

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

Gradient descent method



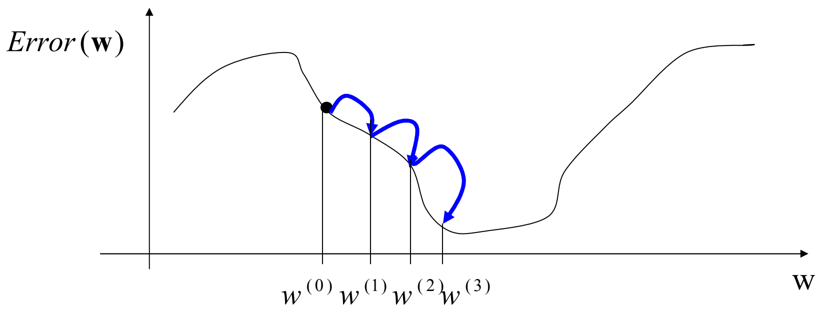
- New value of the parameter

$$w_j \leftarrow w_j^* - \alpha \frac{\partial}{\partial w_j} Error(w)|_{w^*} \qquad \text{For all j}$$

$\alpha > 0$ - a learning rate (scales the gradient changes)

Gradient descent method

- Iteratively approaches the optimum of the Error function



Online gradient algorithm

- The error function is defined for the whole dataset D

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

- **error for a sample** $D_i = < \mathbf{x}_i, y_i >$

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

- **Online gradient method: changes weights after every sample**

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

- **vector form:**

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

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

Online gradient method

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

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

On-line algorithm: generates a sequence of online updates

(i)-th update step with : $D_i = < \mathbf{x}_i, y_i >$

j-th weight:

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial Error_i(\mathbf{w})}{\partial w_j} |_{\mathbf{w}^{(i-1)}}$$

$$\boxed{w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))x_{i,j}}$$

Fixed learning rate: $\alpha(i) = C$

- Use a small constant

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

- Gradually rescales changes

Online regression algorithm

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

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

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

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

set learning rate $\alpha(i)$

update weight vector

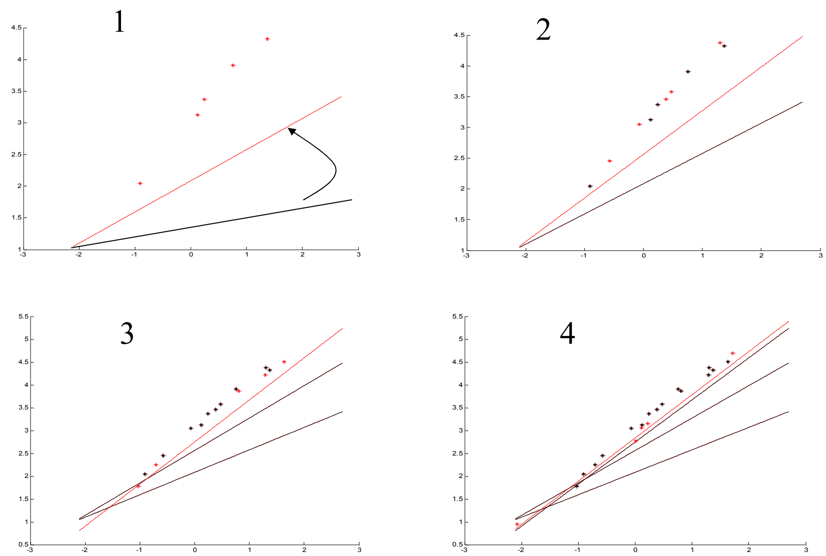
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

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

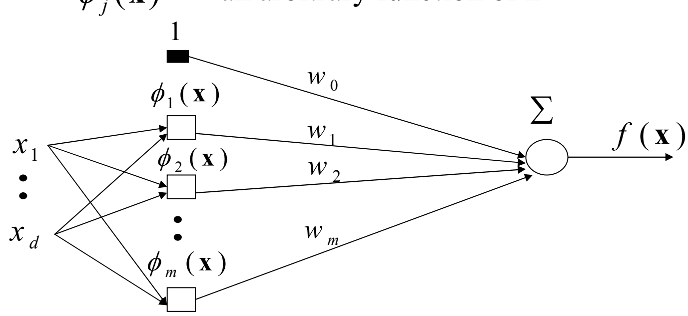
Similarly we can renormalize outputs $\boldsymbol{y}$

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**

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

$w_0, w_1 \dots w_m$ - parameters

$\phi_1(\mathbf{x}), \phi_2(\mathbf{x}) \dots \phi_m(\mathbf{x})$ - **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

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

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

Example. Regression with polynomials.

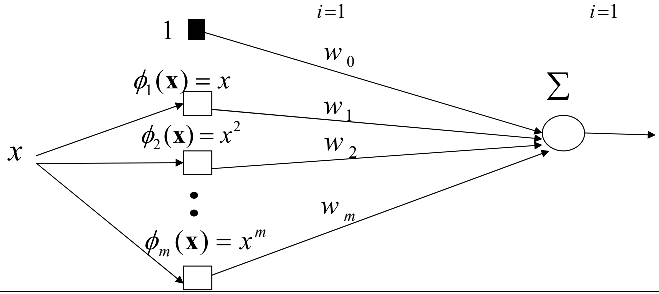
Regression with polynomials of degree m

- **Data points:** **pairs of** $< x, y >$
- **Feature functions:** **m feature functions**

$$\phi_i(x) = x^i \qquad i = 1,2,\dots,m$$

- **Function to learn:**

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i$$



Learning with feature functions.

Function to learn:

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{k} w_i \phi_i(x)$$

On line gradient update for the <*x,y*> pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x}, \mathbf{w}))$$

$$\vdots$$

$$w_j = w_j + \alpha(y - f(\mathbf{x}, \mathbf{w}))\phi_j(\mathbf{x})$$

Gradient updates are of the same form as in the linear and logistic regression models

Example. Regression with polynomials.

Example: **Regression with polynomials of degree m**

$$f(x,\mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i$$

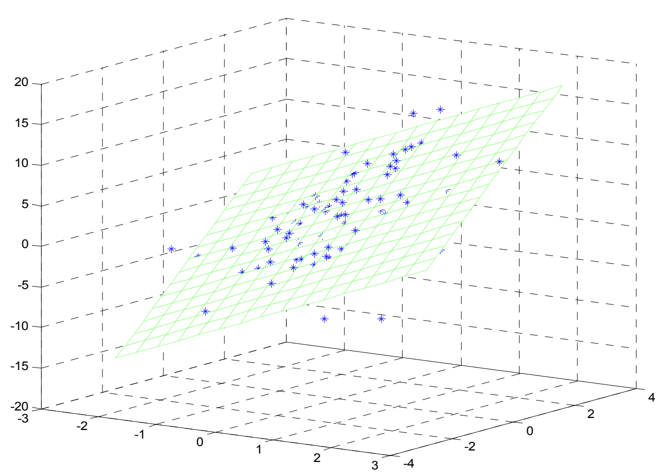
- **On line update** for <*x,y*> pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x},\mathbf{w}))$$

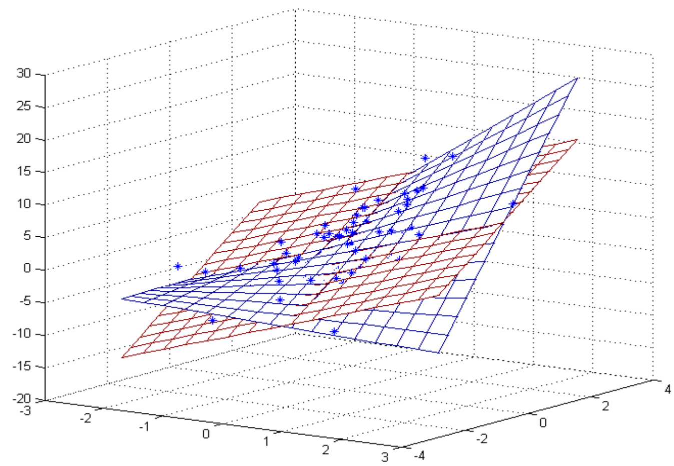
$$\vdots$$

$$w_j = w_j + \alpha(y - f(\mathbf{x},\mathbf{w}))x^j$$

Multidimensional additive model example



Multidimensional additive model example



Statistical model of regression

- **A generative model:** $y = f(\mathbf{x}, \mathbf{w}) + \varepsilon$

  $f(\mathbf{x}, \mathbf{w})$ is a deterministic function

  ε is a random noise , represents things we cannot capture with $f(\mathbf{x}, \mathbf{w})$, e.g. $\varepsilon \sim \mathrm{N}(0, \sigma^2)$

Assume $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$ is a linear model, and $\varepsilon \sim \mathrm{N}(0, \sigma^2)$

Then: $f(\mathbf{x}, \mathbf{w}) = E(y \mid \mathbf{x})$ models the mean of outputs y for $\mathbf{x}$

and the **noise** models deviations from the mean

- **The model defines the conditional density** of y given $\mathbf{x}, \mathbf{w}, \sigma$

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

ML estimation of the parameters

- **likelihood of predictions** = the probability of observing outputs y in D given $\mathbf{w}, \sigma$

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

- We want parameters maximizing the likelihood of predictions

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} p(y_i \mid \mathbf{x}_i, \mathbf{w}, \sigma)$$

Maximum likelihood estimation of parameters (see handout)

- **Log-likelihood** trick for the ML optimization
  - Maximizing the log-likelihood is equivalent to maximizing the likelihood

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

ML estimation of the parameters

- **Using conditional density**

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

- **We can rewrite the log-likelihood as**

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

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

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

- Maximizing with regard to **w**, is **equivalent to minimizing squared error functions**

ML estimation of parameters

- **Criteria based on mean squares error function and the log likelihood of the output are related**

$$J_{online}(y_i, \mathbf{x}_i) = \frac{1}{2\sigma^2} \log \; p(y_i \mid \mathbf{x}_i, \mathbf{w}, \sigma) + c(\sigma)$$

- **We know how to optimize parameters *w***
  – the same approach as used for the least squares fit
- **But what is the ML estimate of 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**