CS 2750 Machine Learning
Lecture 8

Linear regression

Milos Hauskrecht

milos@cs.pitt.edu

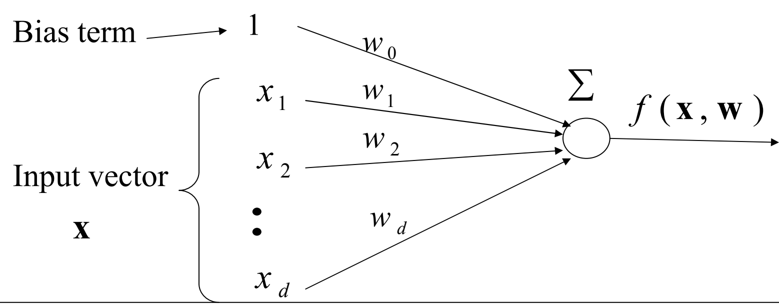
5329 Sennott Square

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. Error.

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

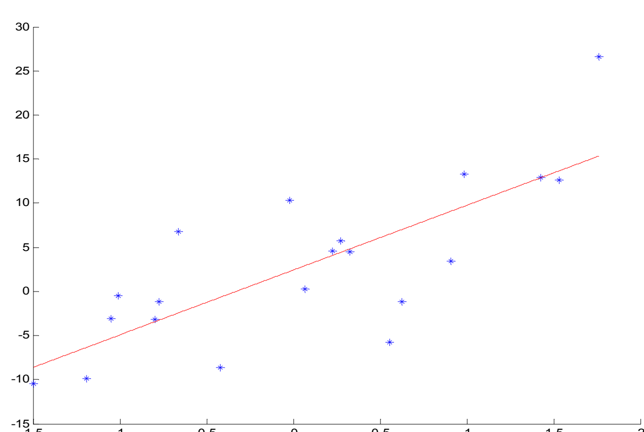
- **Error function**
  - measures how much our predictions deviate from the desired answers

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 !

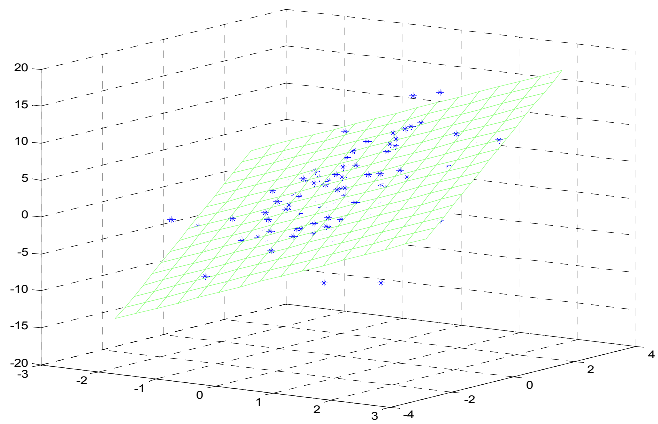
Linear regression. Example

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



Linear regression. Example.

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



Solving linear regression

- The optimal set of weights satisfies:

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

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:

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

- matrix inversion

Gradient descent solution

Goal: the weight optimization in the linear regression model

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

Iterative solution:

- **Gradient descent (first order method)**

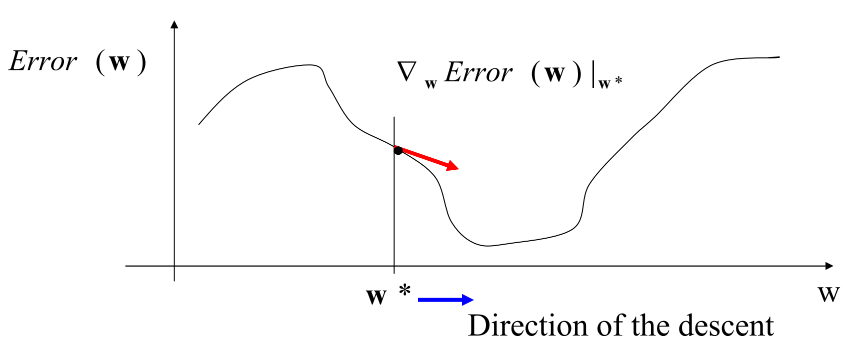
  Idea:
  - Adjust weights in the direction that improves the Error
  - The gradient tells us what is the right direction

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

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

Gradient descent method

- Descend using the gradient information

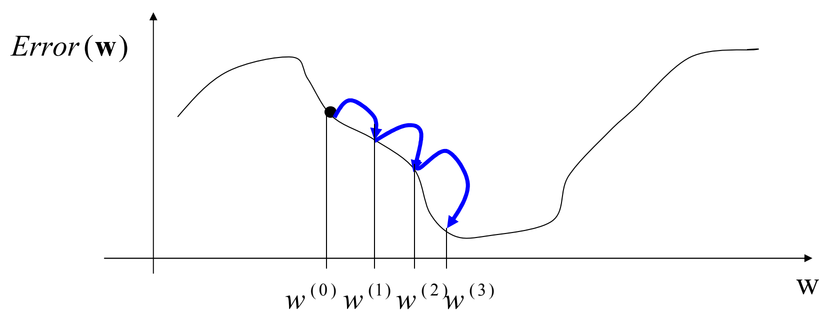


- Change the value of **w** according to the gradient

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

Gradient descent method

- Iteratively approaches the optimum of the Error function



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:

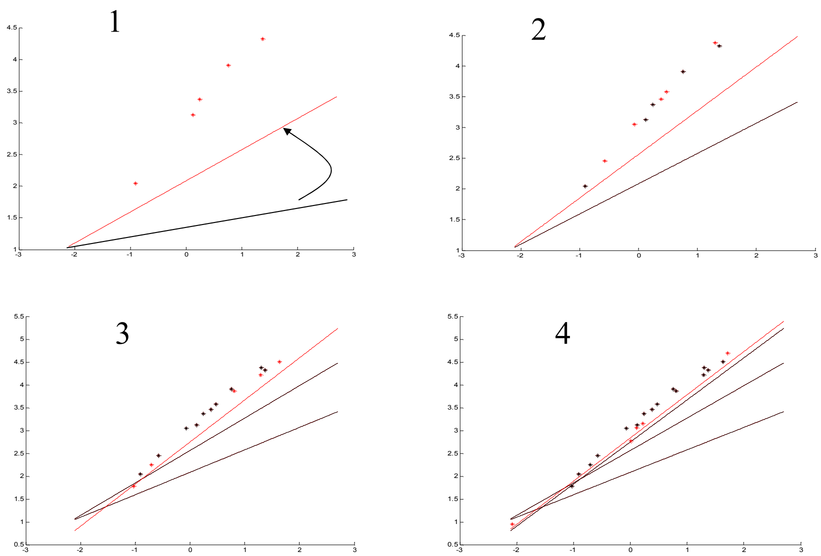
$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i) \frac{\partial Error_i(\mathbf{w})}{\partial w_j} |_{\mathbf{w}^{(i-1)}}$$

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

Fixed learning rate: $\alpha(i) = C$
- Use a small constant

Annealed learning rate: $\alpha(i) \approx \frac{1}{i}$
- Gradually rescales changes

On-line learning. Example

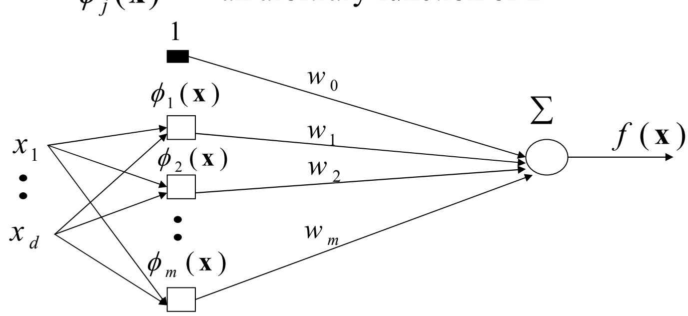


Extensions of simple linear model

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.

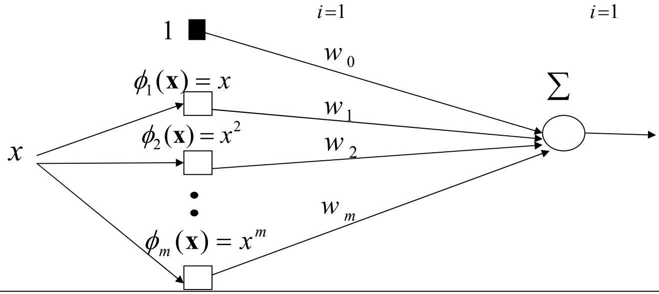
Regression with polynomials of degree m

- **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**

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

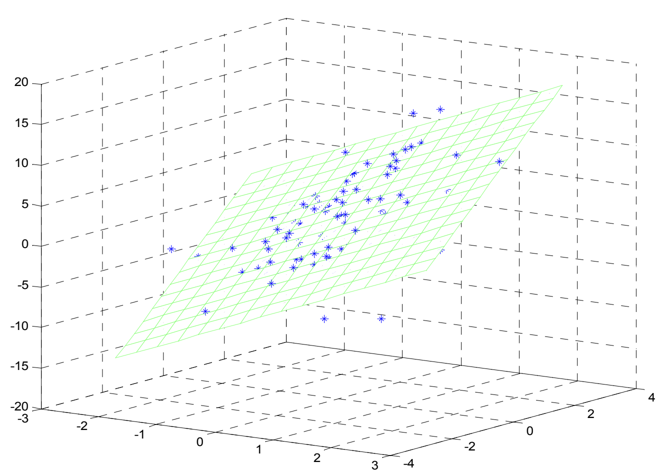
- **On line update** for <*x,y*> pair

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

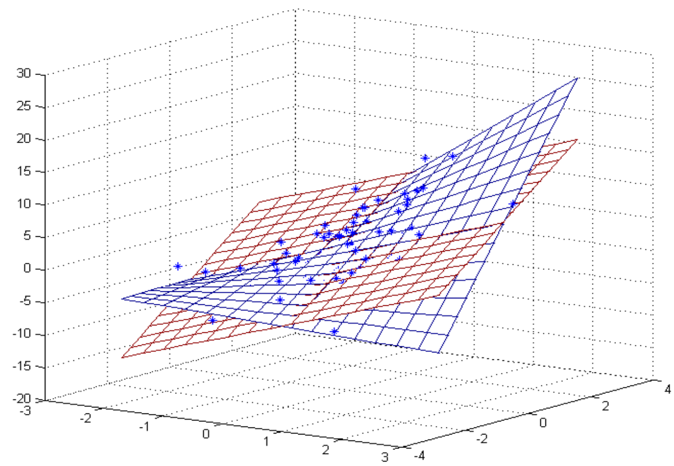
$$\vdots$$

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

Multidimensional additive model example



Multidimensional additive model example

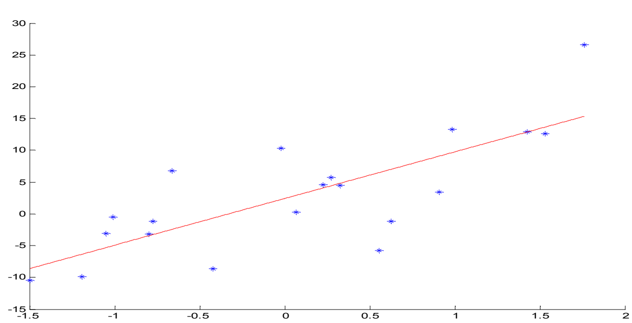


Statistical model of regression

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

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

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



Statistical model of regression

- **Assume a generative model:**

$$y = f(\mathbf{x}, \mathbf{w}) + \varepsilon$$

where $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$ 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** ε models deviations from the mean
- **The model defines the conditional density** of y given $\mathbf{x}, \mathbf{w}, \sigma$

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

ML estimation of the parameters

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

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

- **Maximum likelihood estimation of parameters**
  - parameters maximizing the likelihood of predictions

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

- **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 function**

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 squared prediction error for the best predictor**

Regularized linear regression

- If the number of parameters is large relative to number of data points used to train the model, we face the threat of overfit (generalization error of the model goes up)
- The prediction accuracy can be often improved by setting some coefficients to zero
  - Increases the bias, reduces the variance of estimates
- **Solutions:**
  - **Subset selection**
  - **Ridge regression**
  - **Principal component regression**
- Next: **ridge regression**

Ridge regression

- Error function for the standard least squares estimates:

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

- **We seek:** $\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$
- **Ridge regression:**

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

- Where $\|\mathbf{w}\|^2 = \sum_{i=0}^{d} w_i^2$ and $\lambda \geq 0$
- What does the new error function do?

Ridge regression

- **Standard regression:**

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

- **Ridge regression:**

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

- $\|\mathbf{w}\|^2 = \sum_{i=0}^{d} w_i^2$ penalizes non-zero weights with the cost proportional to λ (a **shrinkage coefficient**)
- If an input attribute x_j has a small effect on improving the error function it is "shut down" by the penalty term
- Inclusion of a shrinkage penalty is often referred to as **regularization**

Regularized linear regression

How to solve the least squares problem if the error function is enriched by the regularization term $\lambda\|\mathbf{w}\|^2$?

Answer: The solution to the optimal set of weights w is obtained again by solving a set of linear equation.

Standard linear regression:

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

Solution: $\mathbf{w}^* = (\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\mathbf{X}^\mathrm{T}\mathbf{y}$

where **X** is an *nxd* matrix with rows corresponding to examples and columns to inputs

Regularized linear regression:

$$\mathbf{w}^* = (\lambda\mathbf{I} + \mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\mathbf{X}^\mathrm{T}\mathbf{y}$$

Regularized linear regression

Problem: How to determine the parameter λ that controls the over-fit?

Overfiting is related to ML estimate.

Bayesian approach alleviates the problem.

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

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

where **X** is an *nxd* matrix with rows corresponding to examples and columns to inputs

$$\mathbf{w}^* = (\lambda\mathbf{I} + \mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{y}$$

Bias and Variance

- **Expected error= Bias + Variance**
  - *Expected error* is the expected discrepancy between the estimated and true function

$$E\left[\left(\hat{f}(X) - E[f(X)]\right)^2\right]$$

  - *Bias* is squared discrepancy between *averaged* estimated and true function

$$\left(E\left[\hat{f}(X)\right] - E[f(X)]\right)^2$$

  - *Variance* is expected divergence of the estimated function vs. its average value

$$E\left[\left(\hat{f}(X) - E\left[\hat{f}(X)\right]\right)^2\right]$$

Bias and Variance

- **Expected error= Bias + Variance**

$$
\begin{aligned}
&E\left[\left(\hat{f}(X) - E[f(X)]\right)^2\right] = \\
&E\left[\left(\hat{f}(X) - E\left[\hat{f}(X)\right] + E\left[\hat{f}(X)\right] - E[f(X)]\right)^2\right] = \\
&E\left\{ \begin{array}{l} \left(\hat{f}(X) - E\left[\hat{f}(X)\right]\right)^2 + \left(E\left[\hat{f}(X)\right] - E[f(X)]\right)^2 + \\ 2\left(\hat{f}(X) - E\left[\hat{f}(X)\right]\right)\left(E\left[\hat{f}(X)\right] - E[f(X)]\right) \end{array} \right\} = \\
&\left(E\left[\hat{f}(X)\right] - E[f(X)]\right)^2 + E\left[\left(\hat{f}(X) - E\left[\hat{f}(X)\right]\right)^2\right] + 0
\end{aligned}
$$

bias variance

Under-fitting and over-fitting

- **Under-fitting:**
  - High bias (models are not accurate)
  - Small variance (smaller influence of examples in the training set)
- **Over-fitting:**
  - Small bias (models flexible enough to fit well to training data)
  - Large variance (models depend very much on the training set)