Lecture-5
Linear Regression

CS 277:
Machine Learning
and
Data Science

By:
Dr. Joydeep Chandra
Associate Professor
Dept. of CSE, IIT Patna

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:**

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

Linear Regression Optimization

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

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

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

$$\ldots$$

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

$$\ldots$$

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

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:

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

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:

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

$$\vec{w} = (X^TX)^{-1}X^TY$$

- matrix inversion

Gradient Descent

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:

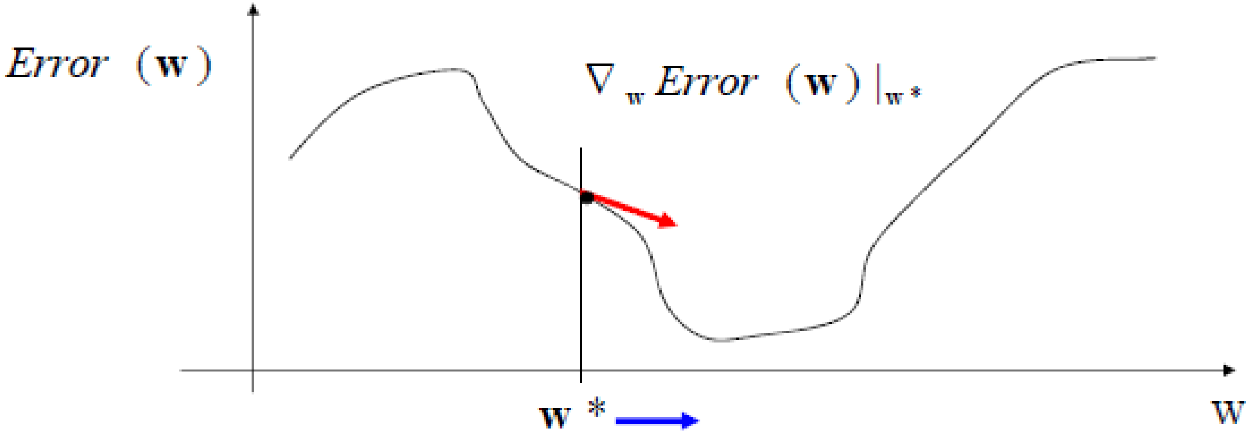
  - Adjust weights in the direction that improves the Error
  - The gradient tells us what is the right direction

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

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

Gradient Descent Method

- Descend using the gradient information

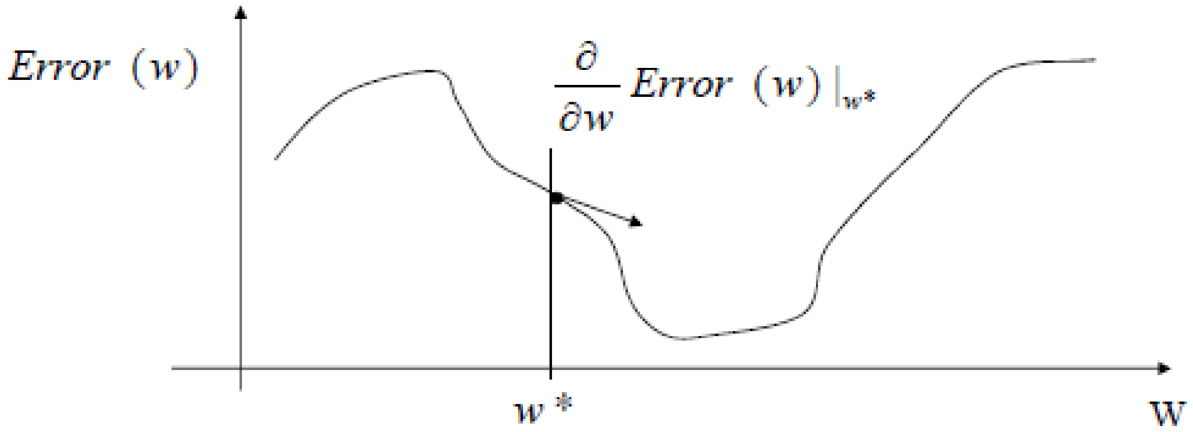


Direction of the descent

- 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



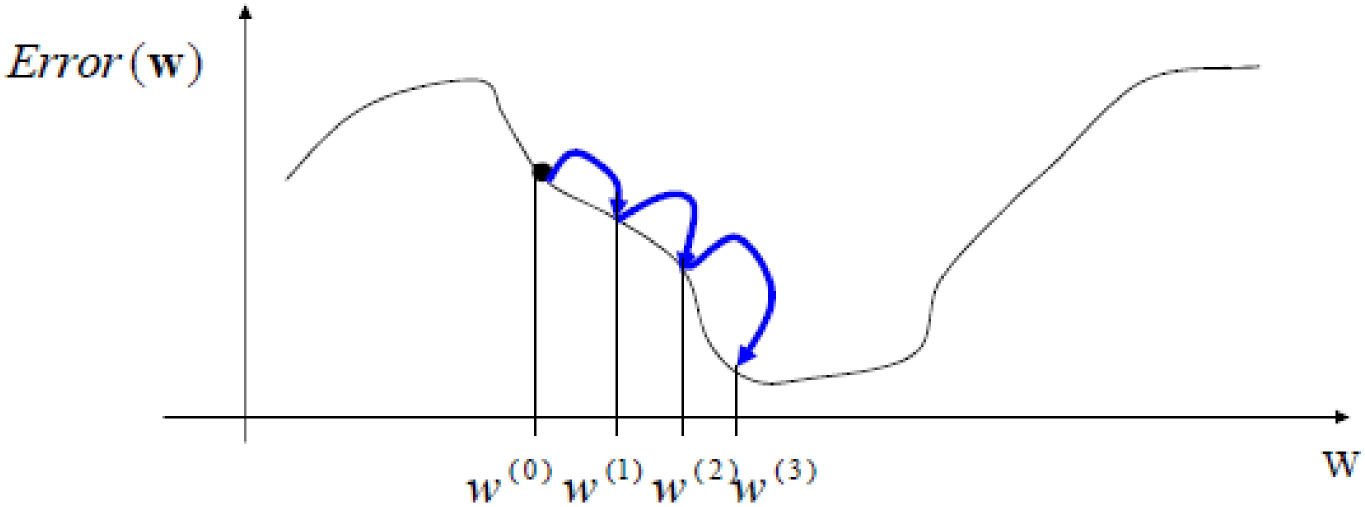
- New value of the parameter

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

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

Gradient Descent Method

- Iteratively approaches the optimum of the Error function



Online Gradient / Stochastic Gradient Descent

- 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 Descent

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:

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

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

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

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

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

set learning rate $\alpha(i)$

update weight vector

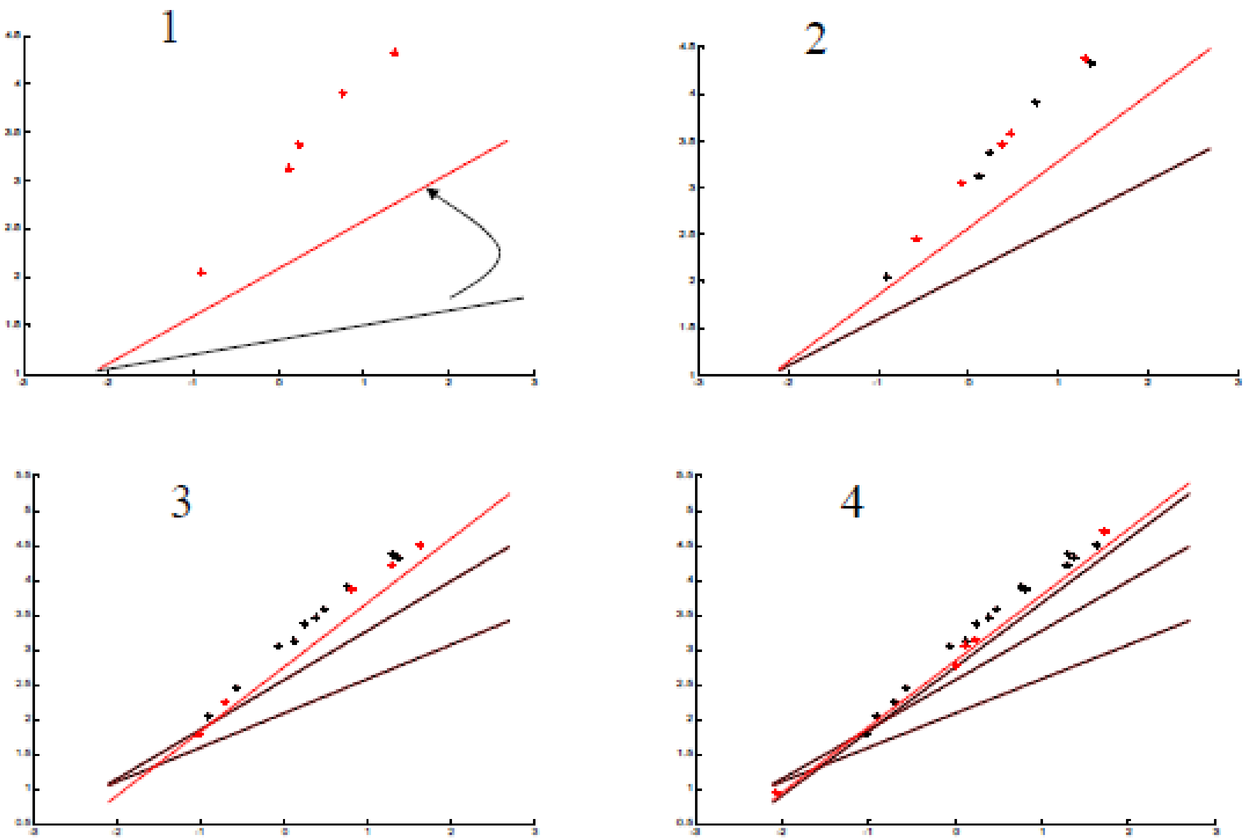
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

end for

return weights $\mathbf{w}$

- Easy to implement on a continuous data stream

How it learns?



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

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

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

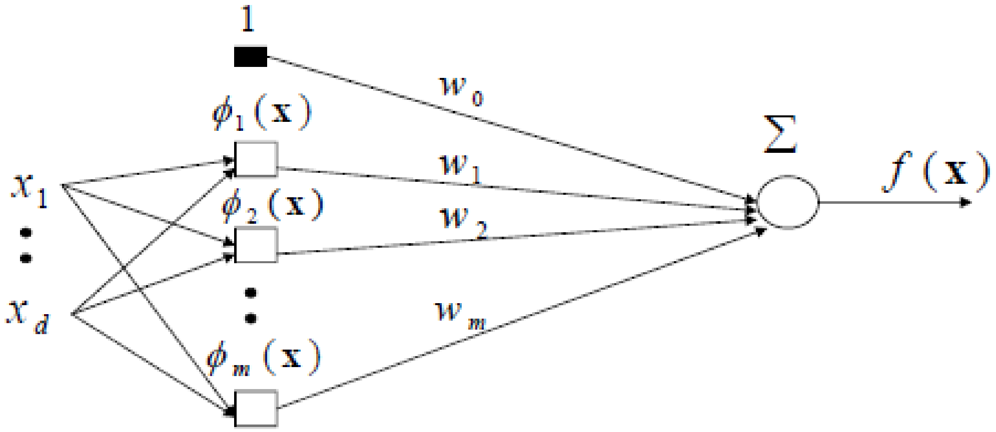
Similarly we can renormalize outputs y

Extensions of the 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

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

MSE and ML equivalence

- **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. $\varepsilon \sim N(0, \sigma^2)$

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

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 $\mathbf{x}, \mathbf{w}, \sigma$

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

ML estimation of the parameters

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

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

- **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 functions**

Regularization of Linear Regression Functions

- If the number of parameters is large relative to the 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**
  - **Lasso regression**
  - **Principal component 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:**

$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_{L2}^2$$

- $\|\mathbf{w}\|_{L2}^2 = \sum_{i=0}^{d} w_i^2$ 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**

Learning the weights in ridge 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:

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

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

where $\mathbf{X}$ is an nxd matrix with rows corresponding to examples and columns to inputs

Regularized linear regression:

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

Lasso Regression

- **Standard regression:**

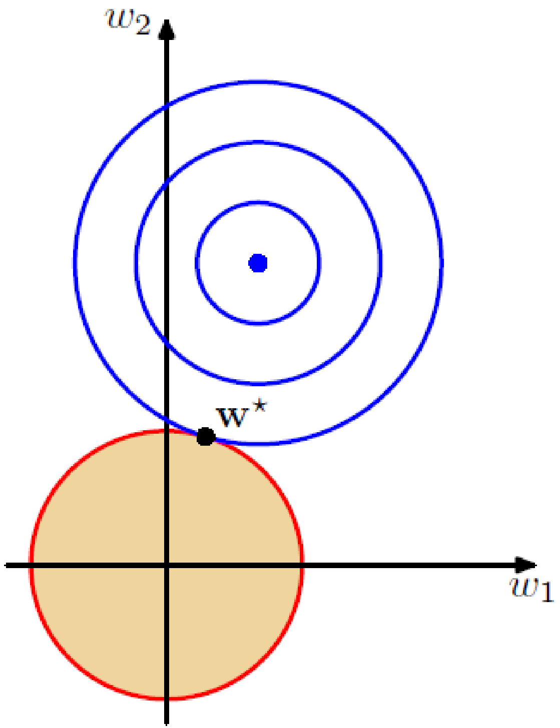
$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- **Lasso regression:**

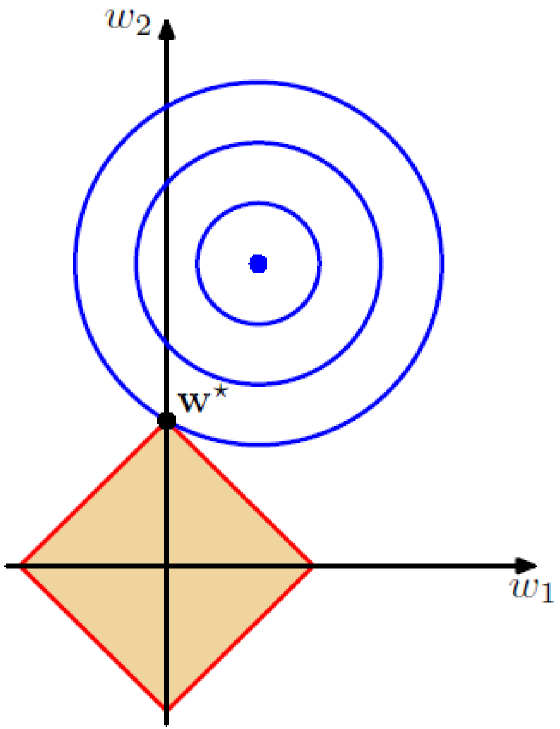
$$J_n(\mathbf{w}) = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_1$$

- $\|\mathbf{w}\|_1 = \sum_{i=0}^{d} |w_i|$ penalizes non-zero weights with the cost proportional to λ
- Lasso regression is more aggressive than the ridge regression in zeroing the weights
- Lasso + ridge regularization combined:
  - **Elastic net regularization**

Geometrically understanding the difference Ridge & Lasso



Ridge



Lasso

Any Questions??