Machine Learning

Session 8 Linear Regression and Regularization

Introduction to Regression

Problem formulation: Loss / Cost function

Solution methods:

- Gradient descent
- Closed-form solution

Geometrical Interpretation

Problem formulation: Maximum Likelihood

Regularization

Bibliography:

Bishop, CH 3, 3.1, 3.2

Introduction

- **Goal of Regression:**

  predict the value of a continuous target variable y for a new D-dimensional vector $\mathbf{x}$ of input features

- Ingredients:
  - A set of training examples (the training set):

    $$\mathbf{X} = \left\{\left(\mathbf{x}^{(n)}, y^{(n)}\right)\right\}, \quad n = 1, \dots, N \quad \mathbf{x}^{(n)} \in \mathbb{R}^D,\ y^{(n)} \in \mathbb{R}$$

  - A model $h_{\mathbf{w}}$ with parameters $\mathbf{w}$ to be estimated.
    We expect: $h_{\mathbf{w}}(\mathbf{x}) \approx y$ for unseen examples

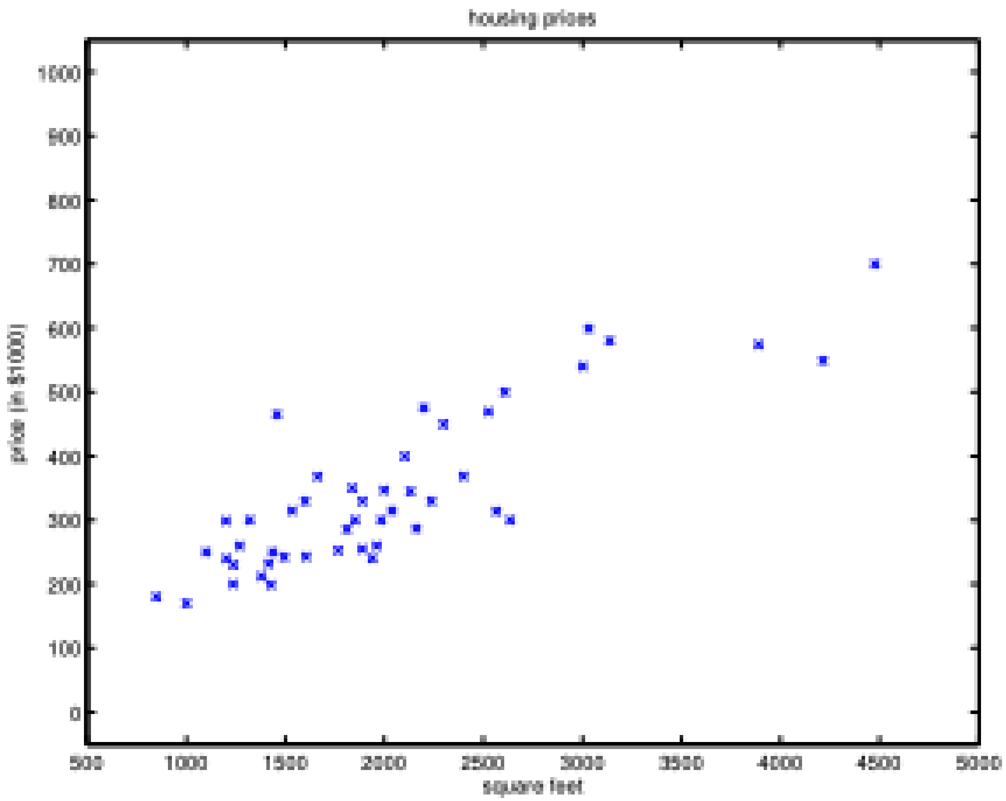
  - The y variable is **continuous** (it does not represent a class)

Introduction

- Example: living areas of homes and its prices

Living area ($feet^2$)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

1 feet = 0.30 m
1 $feet^2$ = 0.09 m^2

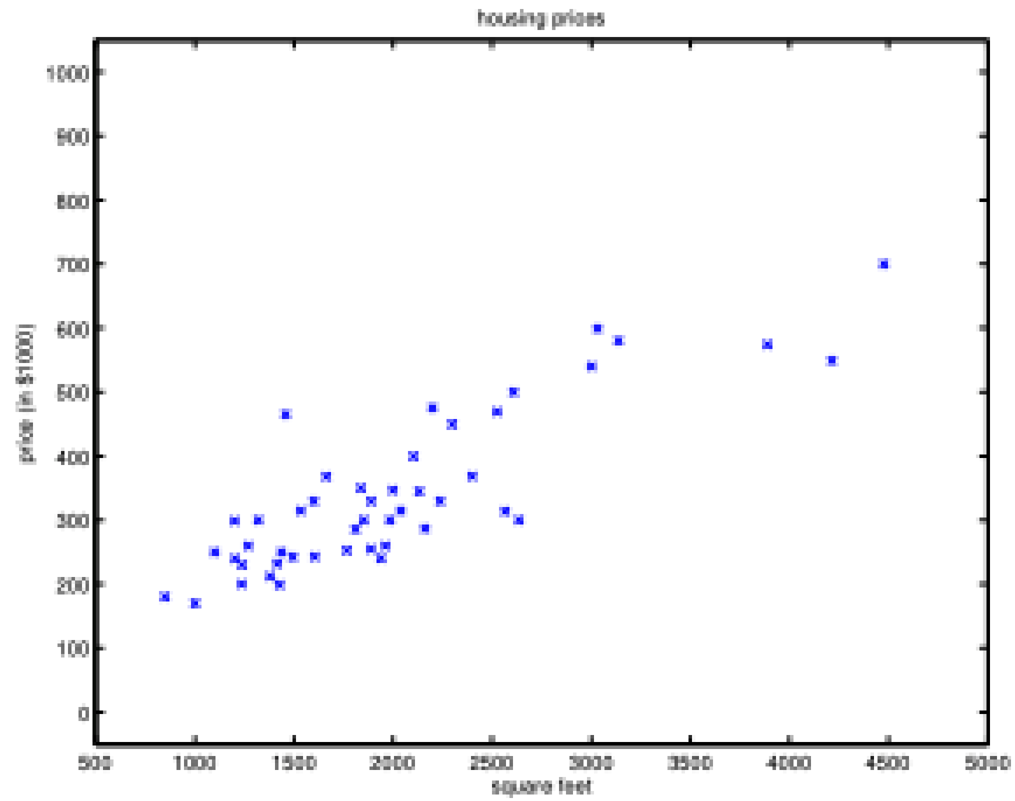


Introduction

- Example: living areas of homes and its prices

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

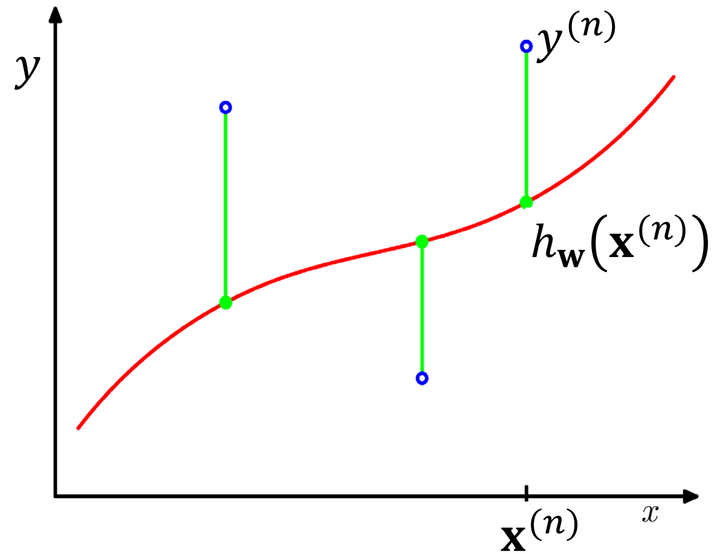
1 feet = 0.30 m
1 feet^2 = 0.09 m^2



- Another example:
  predicting the grade of the ML course from
  - The grade of Algebra?
  - The grade of Algebra and Calculus?

Problem formulation: Loss / Cost function

- The error corresponds to the green lines



- Minimize the following objective: the **least-squares** cost function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2$$

$E(\mathbf{w})$ is zero when $h_w(\mathbf{x})$ passes exactly through each training data point

Goal: learn the parameters $\mathbf{w}$ that minimize $E(\mathbf{w})$

Problem formulation: Loss / Cost function

- **Linear** regression is regression with a **linear** model

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \ \dots + w_D x_D$$

- The **model parameters are the linear coefficients**
- For convenience $\mathbf{w} = (w_0, \dots, w_D)^\top$ and $x_0 = 1$

- The **least-squares** cost function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^\top \mathbf{x}^{(n)} - y^{(n)}\right)^2$$

Solution methods: Gradient Descent

- How can we find $\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} E(\mathbf{w})$?

  The error at that point will be: $E(\hat{\mathbf{w}}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\hat{\mathbf{w}}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2$

- **1st. Method** : Least Mean Squares (LMS) algorithm
  1. Start with an initial guess $\mathbf{w}_0$
  2. Update your guess iteratively to make $E(\mathbf{w})$ smaller
  3. Stop when some convergence criterion is satisfied

Solution methods: Gradient Descent

- How can we find $\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} E(\mathbf{w})$?

  The error at that point will be: $E(\hat{\mathbf{w}}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\hat{\mathbf{w}}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2$

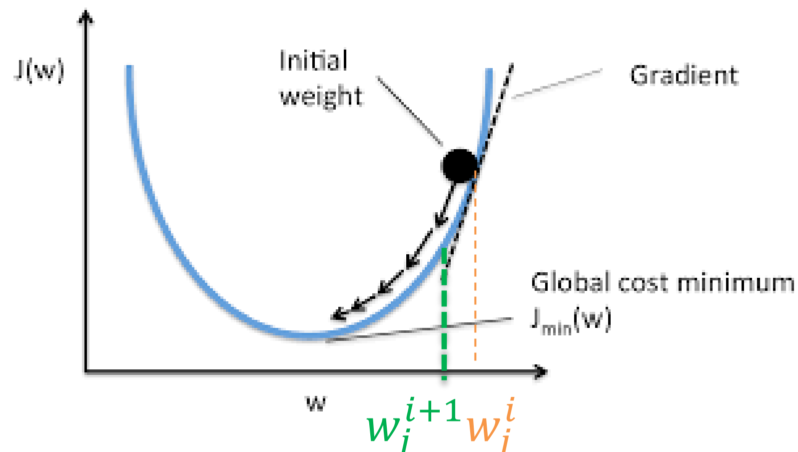
- **1st. Method** : Least Mean Squares (LMS) algorithm
  1. Start with an initial guess $\mathbf{w}_0$
  2. Update your guess iteratively to make $E(\mathbf{w})$ smaller
  3. Stop when some convergence criterion is satisfied

- Gradient descent updates

  $$w_j^{i+1} \leftarrow w_j^i - \alpha \frac{\partial E(\mathbf{w})}{\partial w_j}$$

  For all values of $j = 0, \dots, D$

  α is called ***learning rate***



- In this case the minimum exists, so at each step $E(\mathbf{w})$ decreases

Solution methods: Gradient Descent

- What form has $\frac{\partial E(\mathbf{w})}{\partial w_j}$ in our case?

  Consider one training example $(\mathbf{x}, y)$ [no super index (n)]

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}(h_{\mathbf{w}}(\mathbf{x}) - y)^2$$

$$= 2 \cdot \frac{1}{2}(h_{\mathbf{w}}(\mathbf{x}) - y) \cdot \frac{\partial}{\partial w_j}(h_{\mathbf{w}}(\mathbf{x}) - y)$$

$$= (h_{\mathbf{w}}(\mathbf{x}) - y) \cdot \frac{\partial}{\partial w_j}\left(\sum_{j=0}^{D} w_j x_j - y\right)$$

$$= (h_{\mathbf{w}}(\mathbf{x}) - y) \cdot x_j$$

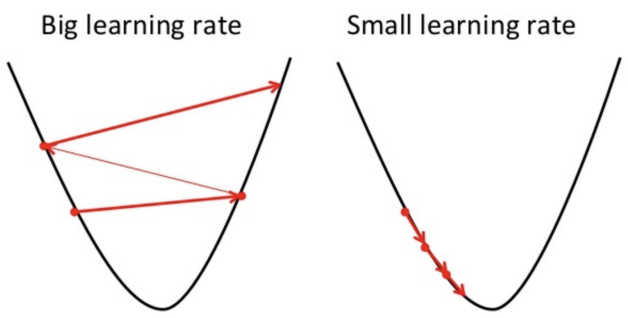
Note that: $\frac{\partial E(\mathbf{w})}{\partial w_0} = (h_{\boldsymbol{w}}(\mathbf{x}) - y) \cdot 1$

Solution methods: Gradient Descent

- For a single training example $(\mathbf{x}_n, y_n)$, the update becomes

$$\boxed{w_j^{i+1} \leftarrow w_j^i - \alpha\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right) \cdot x_j^{(n)}}$$

- Important
  - The magnitude of the update is *proportional to the error*
    - No error -> no update
    - Large error -> large update
  - The value of α needs to be chosen with care
    - Large α -> too large steps (no convergence
    - Small α -> too small steps (slow convergence)



Solution methods: Gradient Descent

- **Online** (stochastic) gradient descent : sequential updates using a single training example $(\mathbf{x}^{(n)}, y^{(n)})$ at each iteration

$$w_j^{i+1} \leftarrow w_j^i - \alpha\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right) \cdot x_j^{(n)}$$

Solution methods: Gradient Descent

- **Online** (stochastic) gradient descent : sequential updates using a single training example $(\mathbf{x}^{(n)}, y^{(n)})$ at each iteration

$$w_j^{i+1} \leftarrow w_j^i - \alpha\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right) \cdot x_j^{(n)}$$

- **Batch** gradient descent : process *all* training examples at each iteration
  – Repeat until convergence

$$w_j^{i+1} \leftarrow w_j^i - \alpha \sum_{n=1}^{N} \left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right) \cdot x_j^{(n)}$$

convergence means that the difference between two iterations is smaller than ϵ, e.g., $\epsilon = 10^{-4}$

Solution methods: Gradient Descent

- Example: 1D problem (Housing dataset)

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Ellipses are contours of error function

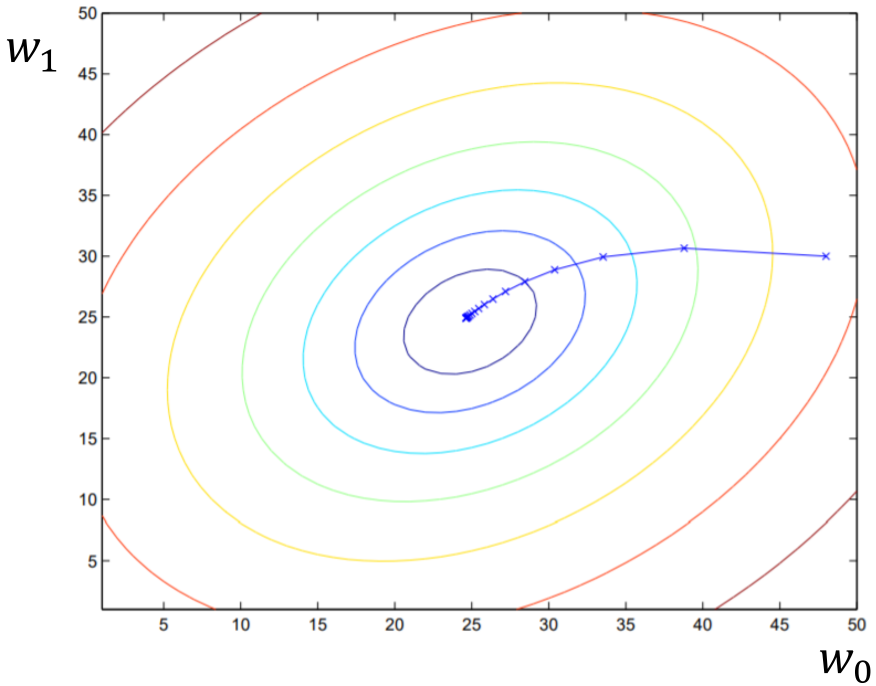
Why?

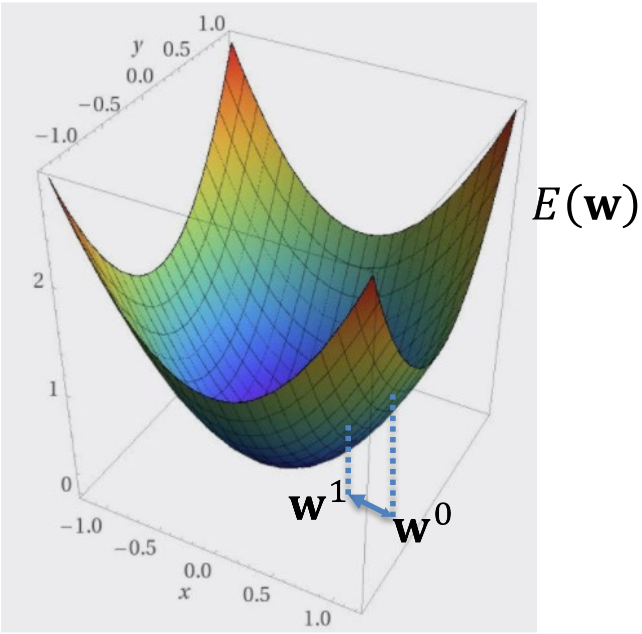
Initial guess $\mathbf{w}^0 = (48, 30)$

After convergence $\hat{\mathbf{w}} = (71.27, 0.135)$

CS229 Lecture notes Andrew Ng

https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf





Solution methods: Gradient Descent

- Example: 1D problem (Housing dataset)

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

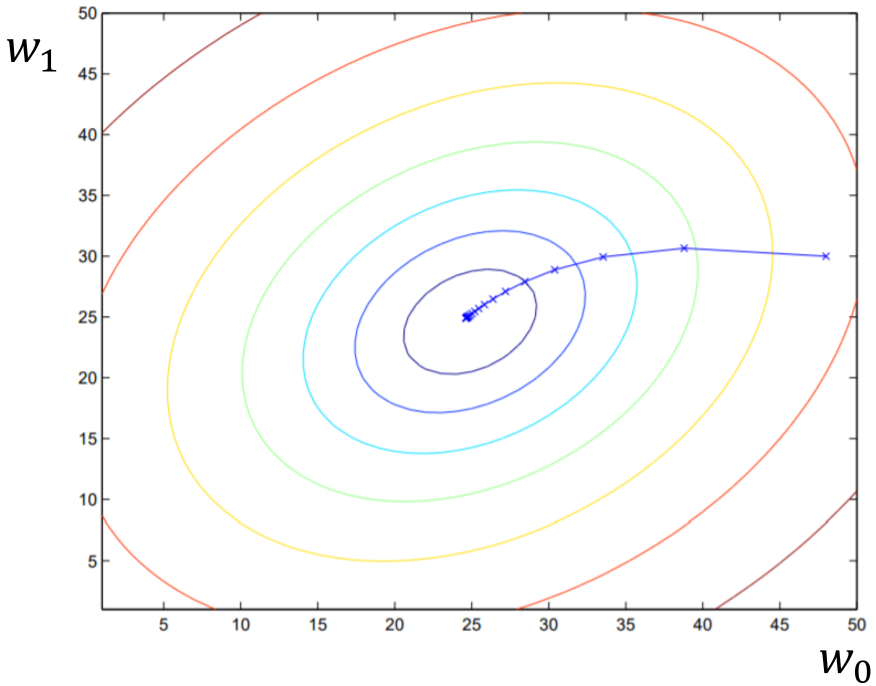
Ellipses are contours of error function

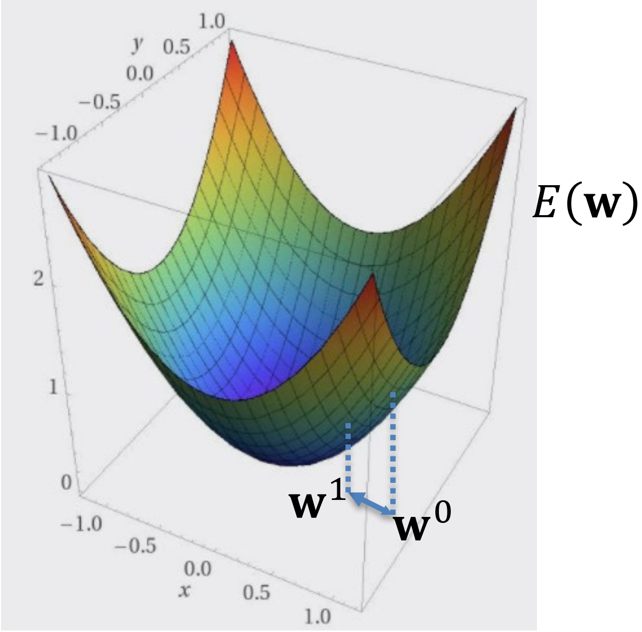
Why?

Initial guess $\mathbf{w}^0 = (48, 30)$

After convergence $\hat{\mathbf{w}} = (71.27, 0.135)$

Another numerical example:
https://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html





Solution methods: Closed-form Solution

- How can we find $\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} E(\mathbf{w})$?
- 2nd. Method : Closed-form solution
- The dataset in matrix form

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & \cdots & x_D^{(N)} \end{bmatrix}, \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{pmatrix}, \mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

$x_j^{(n)}$ is the j-th. component of the n-th. training example

- The cost function $E(\mathbf{w})$ in matrix form,

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2 = \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$$

Solution methods: Closed-form Solution

- To minimize $E(\mathbf{w})$ we take derivatives

Are the same number, transposed, so they are equal

$$\frac{\partial \mathbf{w}^\top \mathbf{A}\mathbf{w}}{\partial \mathbf{w}} = 2\mathbf{A}\mathbf{w}$$

$$\frac{\partial \mathbf{w}^\top \mathbf{z}}{\partial \mathbf{w}} = \mathbf{z}$$

$$\begin{aligned}
\nabla_{\mathbf{w}} E(\mathbf{w}) &= \nabla_{\mathbf{w}} \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \nabla_{\mathbf{w}} \frac{1}{2} (\mathbf{w}^\top \mathbf{X}^\top - \mathbf{y}^\top)(\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= \nabla_{\mathbf{w}} \frac{1}{2} (\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} \boxed{- \mathbf{w}^\top \mathbf{X}^\top \mathbf{y}} \boxed{- \mathbf{y}^\top \mathbf{X}\mathbf{w}} + \mathbf{y}^\top \mathbf{y}) \\
&= \nabla_{\mathbf{w}} \frac{1}{2} (\boxed{\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}} - \boxed{2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y}} + \mathbf{y}^\top \mathbf{y}) \\
&= \frac{1}{2} (2\mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{X}^\top \mathbf{y}) = \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y}
\end{aligned}$$

- Setting derivatives to zero, we obtain the *normal equations*

$$\begin{aligned}
\mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} &= \mathbf{0} \\
\mathbf{X}^\top \mathbf{X}\mathbf{w} &= \mathbf{X}^\top \mathbf{y}
\end{aligned}$$

- The value of $\mathbf{w}$ that minimizes $E(\mathbf{w})$ is given in closed form

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

Solution methods: Closed-form Solution

- What can go wrong?

$$\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\,\mathbf{X}^\top\mathbf{y}$$

$(\mathbf{X}^\top\mathbf{X})^{-1}$ numerically difficult if input covariance matrix $\mathbf{X}^\top\mathbf{X}$ is close to singular

  - When some features are co-linear
  - When $D \gg N$

- In practice, we use the pseudo-inverse $\mathbf{X}^+$
  - Consider solving $\mathbf{X}\mathbf{w} = \mathbf{y}$
  - Get Singular Value Decomposition $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathrm{T}$
  - Get $\mathbf{S}_{\mathrm{nz}}$ discarding zero entries in $\mathbf{S}$ (very small singular values)
  - $\hat{\mathbf{w}} = \mathbf{V}\mathbf{S}_{\mathrm{nz}}^{-1}\mathbf{U}^\top\mathbf{y}$

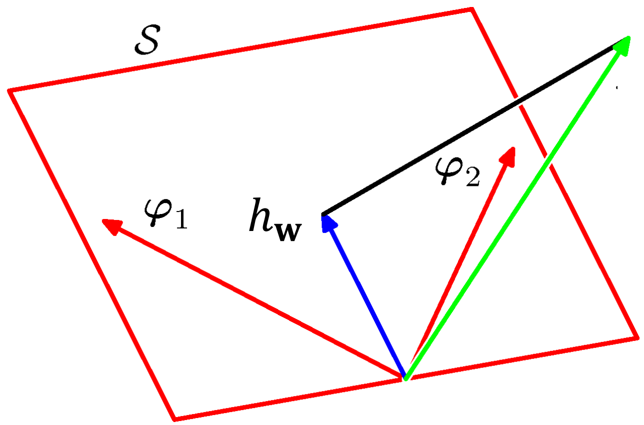
Geometrical Interpretation

- N-dimensional space with components given by $y^{(n)}$
- $\mathbf{y} = [y_1, \dots, y_N]^\top$ is N-dimensional vector in this space
- Feature Design matrix $\mathbf{X} = \begin{bmatrix} 1 & \cdots & x_D^{(1)} \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_D^{(N)} \end{bmatrix}$

N !!!

- Each column $\boldsymbol{\varphi}_j$, $j = 0,1,\dots,D$ is a vector of $\mathbb{R}^N$
- $\mathbf{y} = \left[y^{(1)}, \dots, y^{(N)}\right]^\top$ is a linear combination of the vectors $\boldsymbol{\varphi}_j$

The least-squares solution corresponds to the **orthogonal projection** of **y** onto subspace spanned by $\boldsymbol{\varphi}_j$

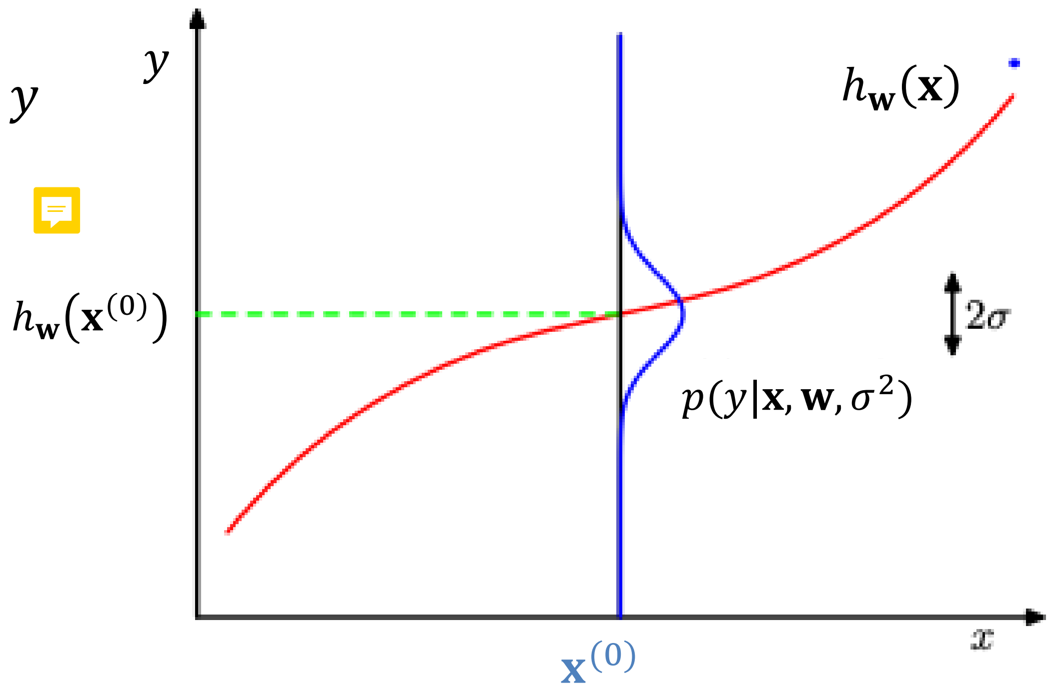


Problem formulation: Maximum Likelihood

Consider learning a noisy linear mapping where the target variable y is generated from a linear model plus zero-mean Gaussian noise

$$y = h_{\mathbf{w}}(\mathbf{x}) + \epsilon = \mathbf{w}^{\top}\mathbf{x} + \epsilon$$

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$$



The probability density function of the target can be written as

$$p(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(y \mid h_{\mathbf{w}}(\mathbf{x}), \sigma^2) = \mathcal{N}(y \mid \mathbf{w}^{\top}\mathbf{x}, \sigma^2)$$

$$\mathcal{N}(y \mid h_{\mathbf{w}}(\mathbf{x}), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{w}^{\top}\mathbf{x} - y)^2\right)$$

Problem formulation: Maximum Likelihood

If examples are generated **independently** from $\mathcal{N}(y \mid h_{\mathbf{w}}(\mathbf{x}), \sigma^2)$, the probability of generating the *entire* dataset $\mathcal{D}$ is

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}\left(y^{(n)} \middle| h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right), \sigma^2\right)$$

$$\mathcal{L}(\mathbf{w}; \mathcal{D}) = \log p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \sigma^2) = \sum_{n=1}^{N} \log \mathcal{N}\left(y^{(n)} \mid h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right), \sigma^2\right)$$

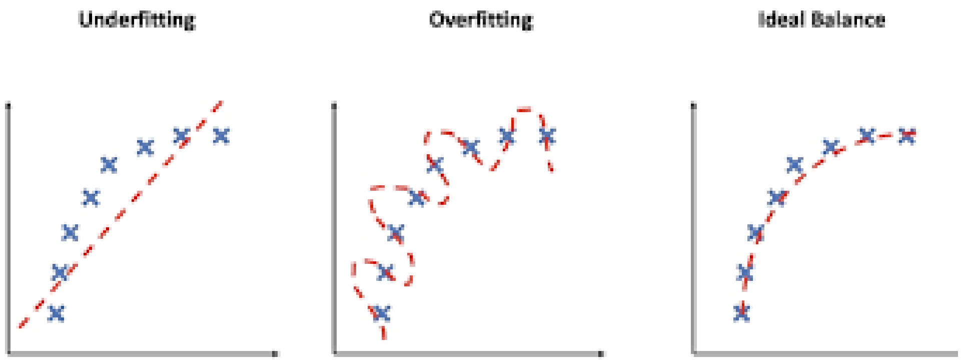
$$= \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2\right)\right)$$

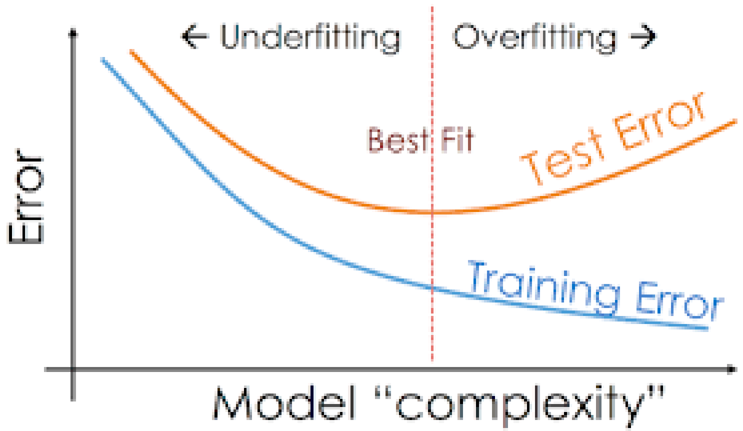
$$= -\frac{N}{2}\log\sigma^2 - \frac{N}{2}\log(2\pi) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2$$

Minimizing the least-squares error is **equivalent** to **maximizing** the log probability of the correct answer under a Gaussian centered at the predicted target and implicit *unknown* variance

Regularization

- Finding the right model (neither too simple nor too complex)





- How can we prevent overfitting?
  - Divide the data in training and test:
  - Use the training data for learning the parameters
  - Use the test data to test the generalization capability given the parameters learned
- Another way to prevent overfitting: **Regularization**

Regularization

Regularized Least Squares

Introduce an **extra term** in the error function

- The term is data-independent
- It is used to incorporate prior knowledge of the parameters $\mathbf{w}$

$$E_R(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2 + \lambda R(\mathbf{w})$$

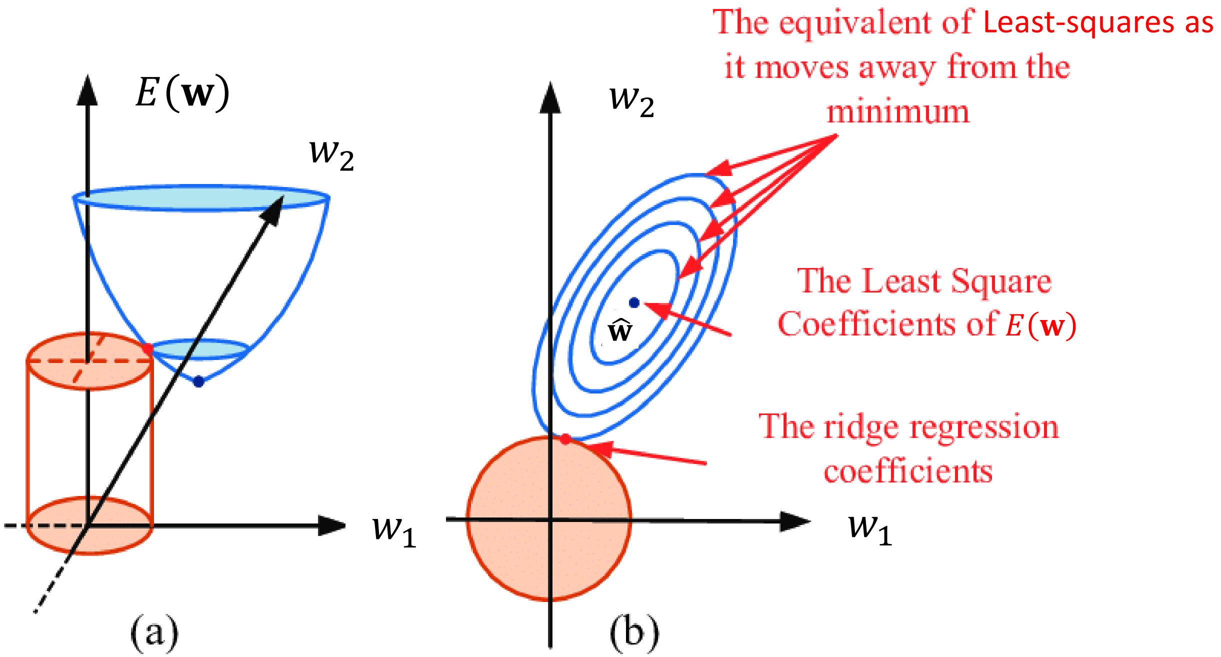
- The regularized objective $E_R(\mathbf{w})$ penalizes undesired solutions
- Hyperparameter λ controls the relative importance of $R(\mathbf{w})$
- Two common regularization terms
  - **Ridge Regression** (penalizes large values in magnitude): $R(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top\mathbf{w}$
  - **Lasso** (penalizes non-sparse solutions): $R(\mathbf{w}) = \frac{1}{2}\sum_{d=1}^{D}|w_d|$

Regularization

- **Ridge Regression**

$$E_R(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2 + \lambda\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

  - Least squares with L2 norm (quadratic) penalty (a.k.a. weight decay)
  - The error function remains quadratic



Regularization

- **Ridge Regression**

$$E_R(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2 + \lambda\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

  – Least squares with L2 norm (quadratic) penalty (a.k.a. weight decay)
  – The error function remains quadratic

- Closed form solution

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

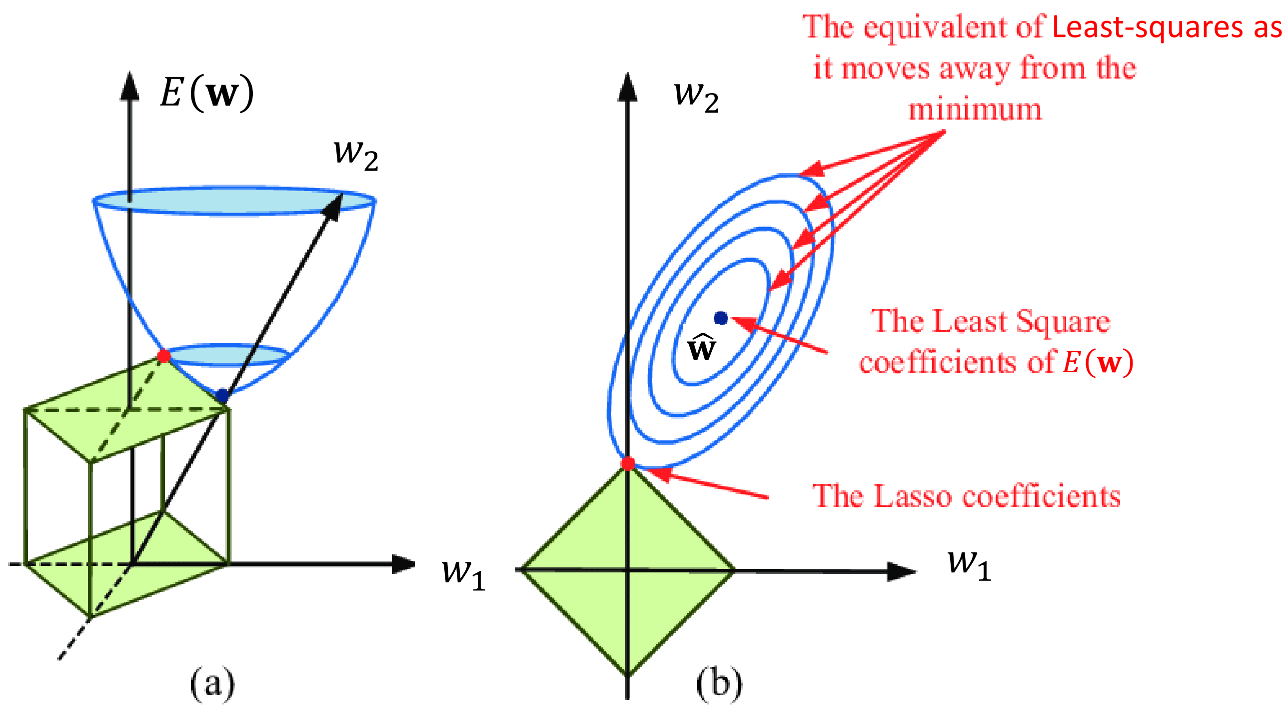
Remark: the diagonal of the covariance matrix increases with λ and makes it nonsingular

Regularization

- **Lasso**

$$E_R(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2 + \lambda\sum_{d=1}^{D}|w_d|$$

  - Least squares with L1 norm penalty
  - Solutions tend to be sparse (many components of **w** are zero)



Regularization

- **Lasso**

$$E_R(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)^2 + \lambda\sum_{d=1}^{D}|w_d|$$

  - Least squares with L1 norm penalty
  - Solutions tend to be sparse (many components of $\mathbf{w}$ are zero)

- No closed form, requires quadratic programming
  Efficient algorithm LARS (least angle regression)

Linear Regression

- Summary
  - Solving least squares problems
    - Online (incremental) methods
      - Online or stochastic gradient descent:
        - » Used in massive datasets
        - » Converges faster
      - Batch Gradient descent:
        - » More stable updates
        - » Typical trade-off (mini-batches)
    - Closed-form solution
      - Requires matrix inversion
      - Typically solved using SVD
  - Regularization and bias-variance trade-off
    - A technique to prevent overfitting
    - Different forms: weight decay, LASSO, etc, …