ificial Intelligence and Machine Learn

Linear Regression

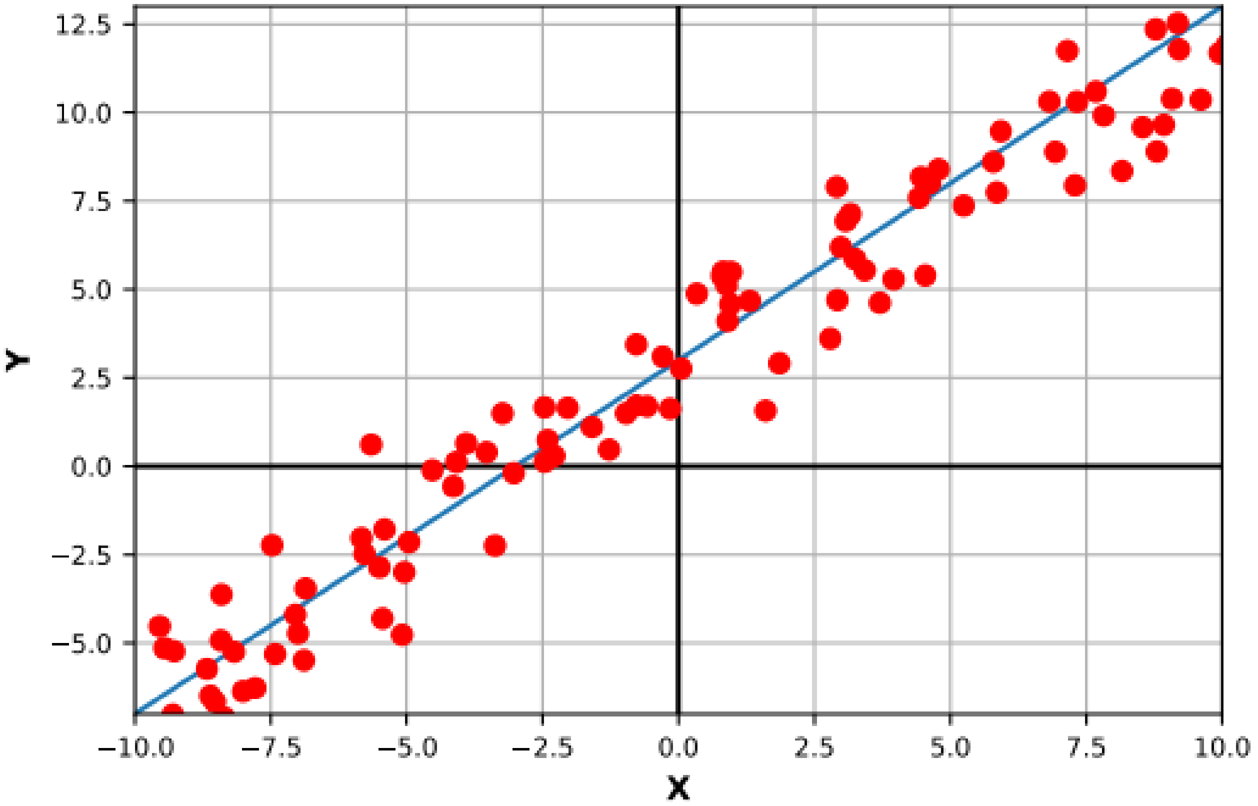
Lecture 1: Outline

- Linear Regression
- Optimization
- Applications

Motivation

- Linear Regression is “still” one of the more widely used ML/DL Algorithms
- Easy to understand and implement
- Efficient to Solve
- We will use Linear Regression to Understand the concepts of:
  - Data
  - Models
  - Loss
  - Optimization

Simple Linear Regression



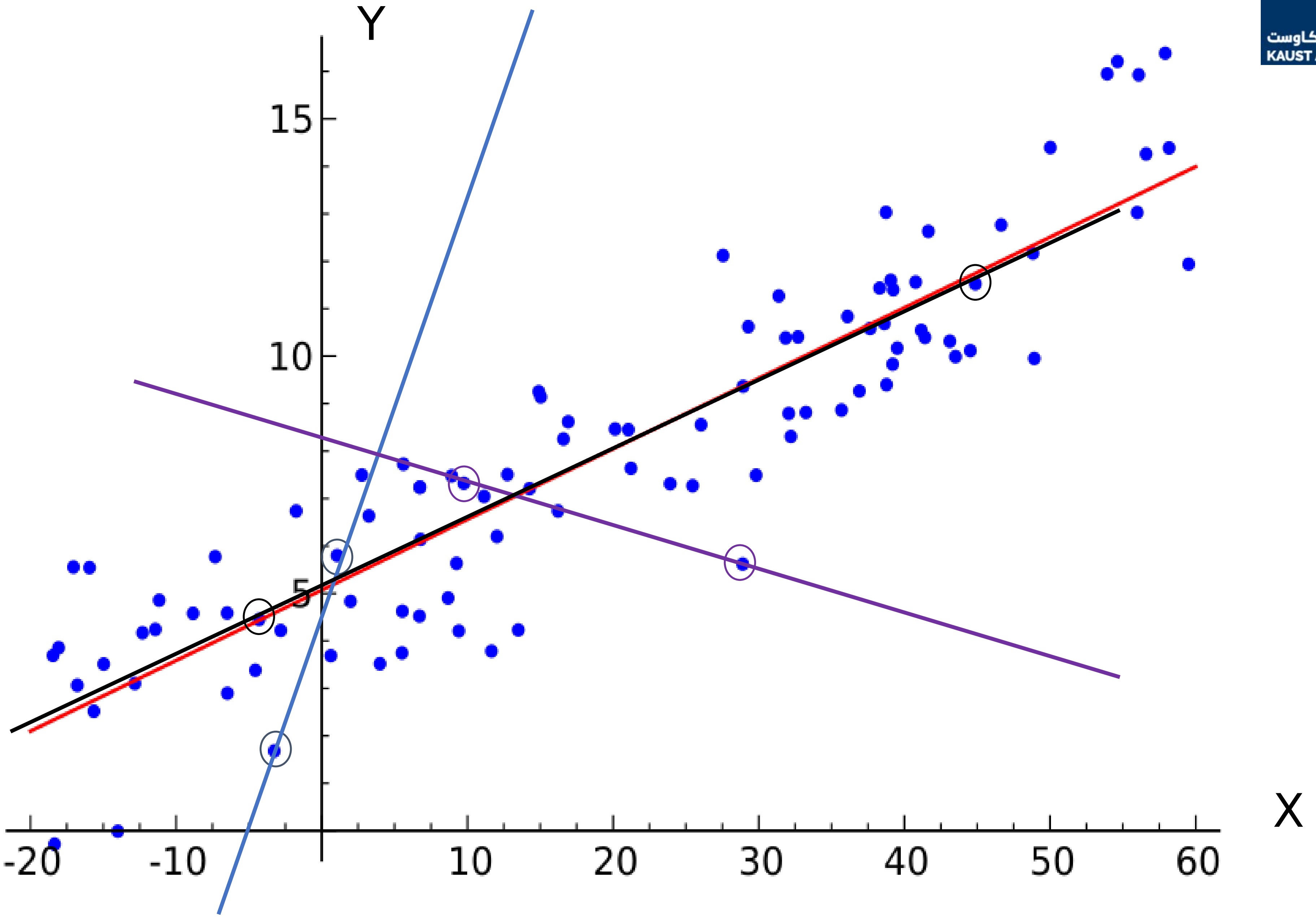
Model (*Linear*) $Y = mX + b$

Y: Response Variable
X: Covariate / Ind., var/Regressors
m: slope
b: bias $\theta = \{m, b\}$

Simple Linear Regression

- Input: data (),
- Goal: learn values of variable ()

- Question: How many points in a plane do we need to fit a line through it?



Y
15
10
5
-20
-10
10
20
30
40
50
60
X

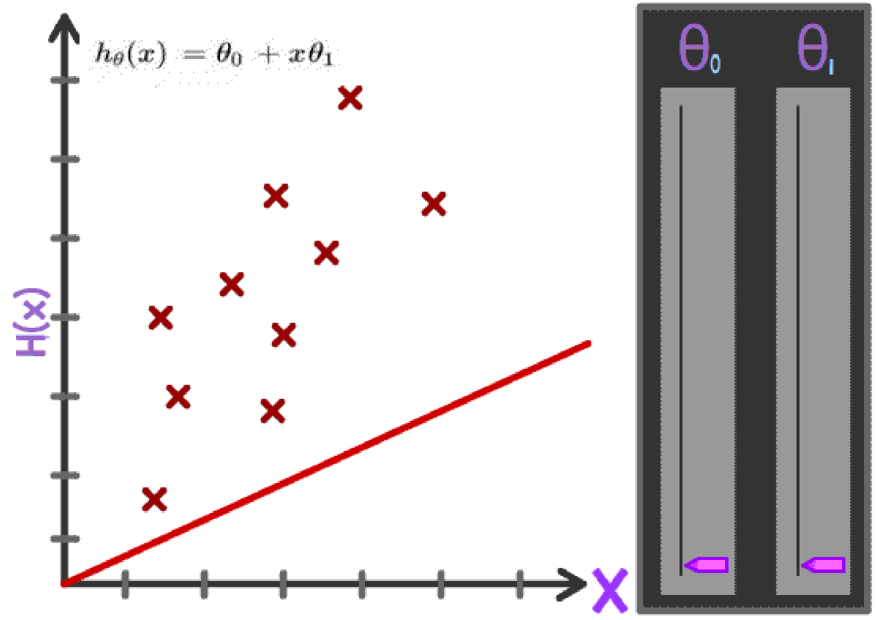
Notation

- Some clarification about the notation we will use for this course

- is the index of the data, is the feature number, and is the power.

Solution Strategy for Solving the Problem

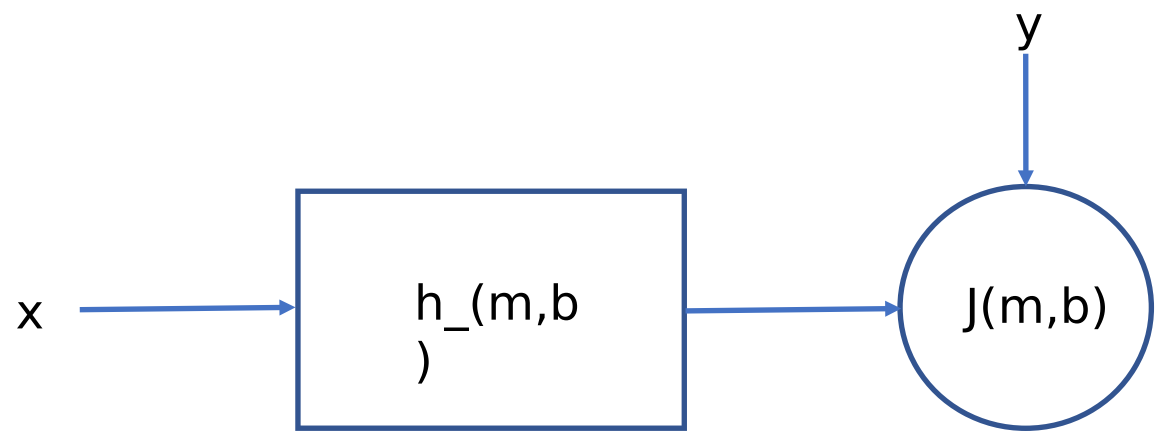
- We want a line which is in some sense the "average line" that represents the data.
- Any ideas as to how we can do it?



https://medium.com/analytics-vidhya/machine-learning-intuition-b4b49a671f65

Cost Function

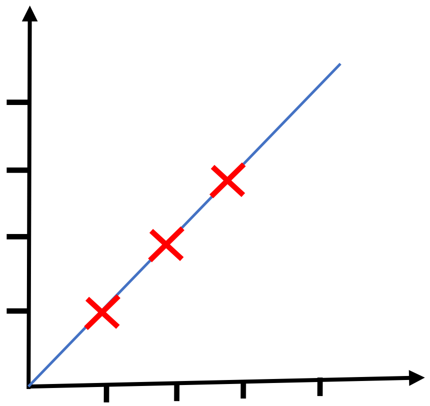
- We want to minimize the discrepancy between our model hypothesis and the observed label.



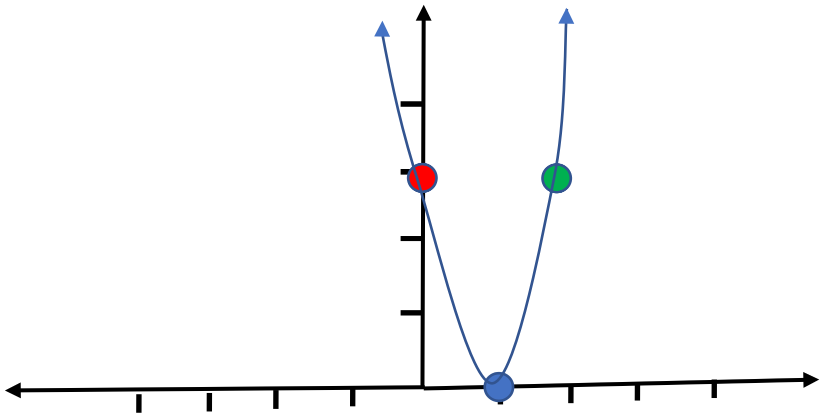
Intuition of Cost Function

$$h(x) = mx$$

$$J(m) = \sum_{i=1}^{3} (y_i - mx_i)^2$$



Hypothesis



Cost Function

$\boxed{J(1) = 0}$ $\boxed{J(0) = 14}$ $\boxed{J(2) = 14}$

How to find minima of a function (Review):

$$J(m) = \sum_{i=1}^{3} (y_i - mx_i)^2 \qquad J(m) = \sum_{i=1}^{3} (i - mi)^2$$

$$\frac{dJ(m)}{dm} = \frac{d}{dm} \sum_{i=1}^{3} (i - mi)^2 \quad \frac{dJ(m)}{dm} = \sum_{i=1}^{3} \frac{d}{dm} (i - mi)^2$$

$$\frac{dJ(m)}{dm} = \sum_{i=1}^{3} -2i(i - mi) \quad -2\sum_{i=1}^{3} i^2 + 2m \sum_{i=1}^{3} i^2 = 0 \quad m = 1$$

Hypothesis Function with 2 Variables

- Let’s setup regression for linear function in two variables:
- The hypothesis function is:

$$\hat{y}_i = mx_i + b$$

- Similar to the previous problem our loss function is:

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Let’s calculate the partial derivatives of the loss function w.r.t.

Gradient of the cost function

- We get the following expressions for the gradient of the cost function

$$\frac{\partial J}{\partial m} = \frac{1}{N} \sum_{i=1}^{N} -2(y_i - \hat{y}_i)x_i$$

$$\frac{\partial J}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} -2(y_i - \hat{y}_i)$$

Gradient of the cost function

- Simplifying the above expressions, we get:

$$\frac{\partial J}{\partial m} = \frac{-2}{N}\sum_{i=1}^{N} y_i x_i + \frac{2m}{N}\sum_{i=1}^{N} x_i^2 + \frac{2b}{N}\sum_{i=1}^{N} x_i$$

$$\frac{\partial J}{\partial b} = \frac{-2}{N}\sum_{i=1}^{N} y_i + \frac{2m}{N}\sum_{i=1}^{N} x_i + \frac{2b}{N}\sum_{i=1}^{N} 1$$

Gradient of the cost function

- Setting the Gradient equal to 0, and solving for m and b, we get

$$\underbrace{\begin{bmatrix} \frac{\sum_i x_i^2}{N} & \frac{\sum_i x_i}{N} \\ \frac{\sum_i x_i}{N} & 1 \end{bmatrix}} \underbrace{\begin{bmatrix} m \\ b \end{bmatrix}} = \underbrace{\begin{bmatrix} \frac{\sum_i x_i y_i}{N} \\ \frac{\sum_i y_i}{N} \end{bmatrix}}$$

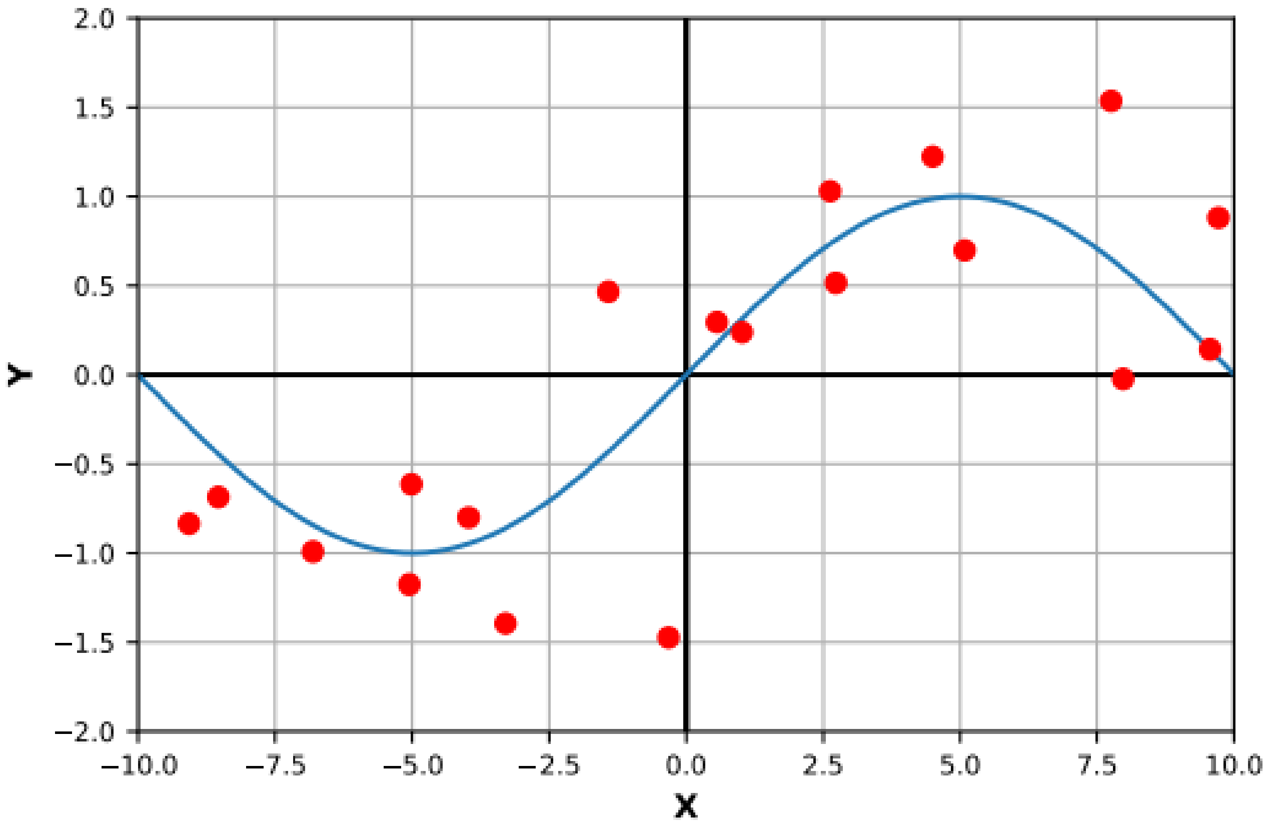
Issues with the Approach

- Calcuting graients like this can quickly become tedious
- Each term on either side of the experssion can be written a dot product of two vectors (maybe we can calculate it more efficiently)?

- Let’s explore if we can do something better through vectorization
- Before, we step into vectorization, let’s consider regression for non-linear functions

Fitting Non-linear Data

- What if y is a non-linear function of x, will this approach still work?



Transforming the Feature Space

- We can transform features x_i

$$x_i = (x_i^1, x_i^2, x_i^3, \ldots, x_i^m)$$

- We will apply some non-linear transformation ϕ:

$$\phi : \mathbb{R}^m \rightarrow \mathbb{R}^M$$

- For example, Polynomial transformation:

$$\phi(x_i) = \{1, x_i^1, x_i^{1,[2]}, \ldots, x_i^{1,[k]}, x_i^2, x_i^{2,[2]}, \ldots, x_i^{2,[k]}, \ldots, x_i^m, x_i^{m,[2]}, \ldots, x_i^{m,[k]}\}$$

  - others: cosine, splines, radial basis functions, etc.
  - Expert engineered features (modeling)

Vectorization

- To truly appreciate the power of vectorization. Let’s make the problem a little more complex. The hypothesis function is now

- Where are the unknow weightns of the data features of the input
- Next, we denote the discrepency between and as

Vectorization

- Now let's collect the above equation for all datapoints

$$y_1 = \hat{y}_1 + \epsilon_1$$

$$y_2 = \hat{y}_2 + \epsilon_2$$

$$\cdot$$

$$\cdot$$

$$\cdot$$

$$y_N = \hat{y}_N + \epsilon_N$$

Vectorization

- Replacing the values of , we get:

$$y_1 = w_0 + w_1 x_1^1 + w_2 x_1^2 + \ldots + w_M x_1^M + \epsilon_1$$

$$y_2 = w_0 + w_1 x_2^1 + w_2 x_2^2 + \ldots + w_M x_2^M + \epsilon_2$$

$$\vdots$$

$$y_N = w_0 + w_1 x_N^1 + w_2 x_N^2 + \ldots + w_M x_N^M + \epsilon_N$$

Vectorization

- Collecting the equations in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^M \\ 1 & x_2^1 & x_2^2 & \dots & x_2^M \\ 1 & x_3^1 & x_3^2 & \dots & x_3^M \\ & . & & & \\ & . & & & \\ & . & & & \\ 1 & x_N^1 & x_N^2 & \dots & x_N^M \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ . \\ . \\ . \\ \epsilon_N \end{bmatrix}$$

Vectroization

- Notice the rows of the matrix on the right are data samples:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_N \end{bmatrix} = \begin{bmatrix} \cdots & \mathbf{x_1} & \cdots \\ \cdots & \mathbf{x_2} & \cdots \\ \cdots & \mathbf{x_3} & \cdots \\ & . & \\ & . & \\ & . & \\ \cdots & \mathbf{x_N} & \cdots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ . \\ . \\ . \\ \epsilon_N \end{bmatrix}$$

Vectorization

$$\mathcal{D} = \{(\mathbf{x_i}, \mathbf{y_i})\}_{\mathbf{i=1}}^{\mathbf{N}}$$

- Let's formalize some notaitons:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_N \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x_1} & \cdots \\ \cdots & \mathbf{x_2} & \cdots \\ \cdots & \mathbf{x_3} & \cdots \\ & . & \\ & . & \\ & . & \\ \cdots & \mathbf{x_N} & \cdots \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_M \end{bmatrix} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ . \\ . \\ . \\ \epsilon_N \end{bmatrix}$$

$$\boldsymbol{y} = \boldsymbol{X\theta} + \boldsymbol{\epsilon}$$

Cost function for the Vectorized form

- Notice that we are using the MSE cost function:

$$J = \frac{1}{N} \sum_{i} (y_i - \widehat{y}_i)^2$$

- Using the defintion of epsilon we can write the above as:

$$J = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^{N} (\epsilon_i)^2$$

- Using the definition of dot product the above can be written as:

$$J = \frac{1}{N} \sum_{i=1}^{N} (\epsilon_i)^2 = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

Optimization

- The optimization problem is now:

$$\min_{\boldsymbol{\theta}} \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

$$\min_{\boldsymbol{\theta}} \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = \min_{\boldsymbol{\theta}} (\boldsymbol{y} - (\boldsymbol{X}\boldsymbol{\theta}))^T (\boldsymbol{y} - (\boldsymbol{X}\boldsymbol{\theta}))$$

- We will use chain rule to calculate the gradient of the cost function:

$$\frac{\partial}{\partial \boldsymbol{\theta}} J = \frac{dJ}{d\boldsymbol{\epsilon}} \nabla_{\boldsymbol{\theta}} \boldsymbol{\epsilon}$$

Linear Least Squares

- We get:

$$\frac{\partial}{\partial \boldsymbol{\theta}} J = \boldsymbol{X}^T 2(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})$$

- Setting it equal to zero we can solve for $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

Probalistic Interpretration of Linear Regression and MLE

- We can also look at the probalistic Interpretation of Linear Regression.
- Keeping everything else same as the previous formulation

$$y_i = \boldsymbol{x}_i^T \boldsymbol{\theta} + \epsilon_i$$

- Now assume that $\epsilon_i \sim \mathcal{N}(0,\, \sigma^2)$, then $y_i \sim \mathcal{N}(\boldsymbol{x}_i^T \boldsymbol{\theta}, \sigma^2)$
- We can write the conditional distribution as :

$$\mathbb{P}(y_i | \boldsymbol{x}_i) \sim \mathcal{N}(0, \sigma^2)$$

Probalistic Interpretation of LR

- Let’s assume that all data point in the dataser are i.i.d. (independent identically distributed). Then we have:

$$\mathbb{P}(\mathcal{D}) = \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i, y_i)$$

- Using Bayes Theorem we can write:

$$\prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i, y_i) = \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i)\mathbb{P}(y_i|\boldsymbol{x}_i)$$

Maximum Likelihoold Estimator

- In simple words, given the Dataset we want to find the values of the unknow parameters which maximize the probability of the Dataset.
- Using the definition of the conditional distrubtion we have

$$\mathbb{P}(y_i|\boldsymbol{x}_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta})\right)$$

- Using the definition we get

$$\prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i, y_i) = \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i) \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta})\right)$$

Maximum Likelihood Estimator

- Let's try to maximaize:

$$\prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i, y_i) = \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i) \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta})\right)$$

- Note that

$$\arg\max_{\theta} \prod_{i=1}^{N} \mathbb{P}(\boldsymbol{x}_i, y_i) = \arg\max_{\theta} \prod_{i=1}^{N} \exp\left(-(y_i - \boldsymbol{x}_i^T \boldsymbol{\theta})^2\right)$$

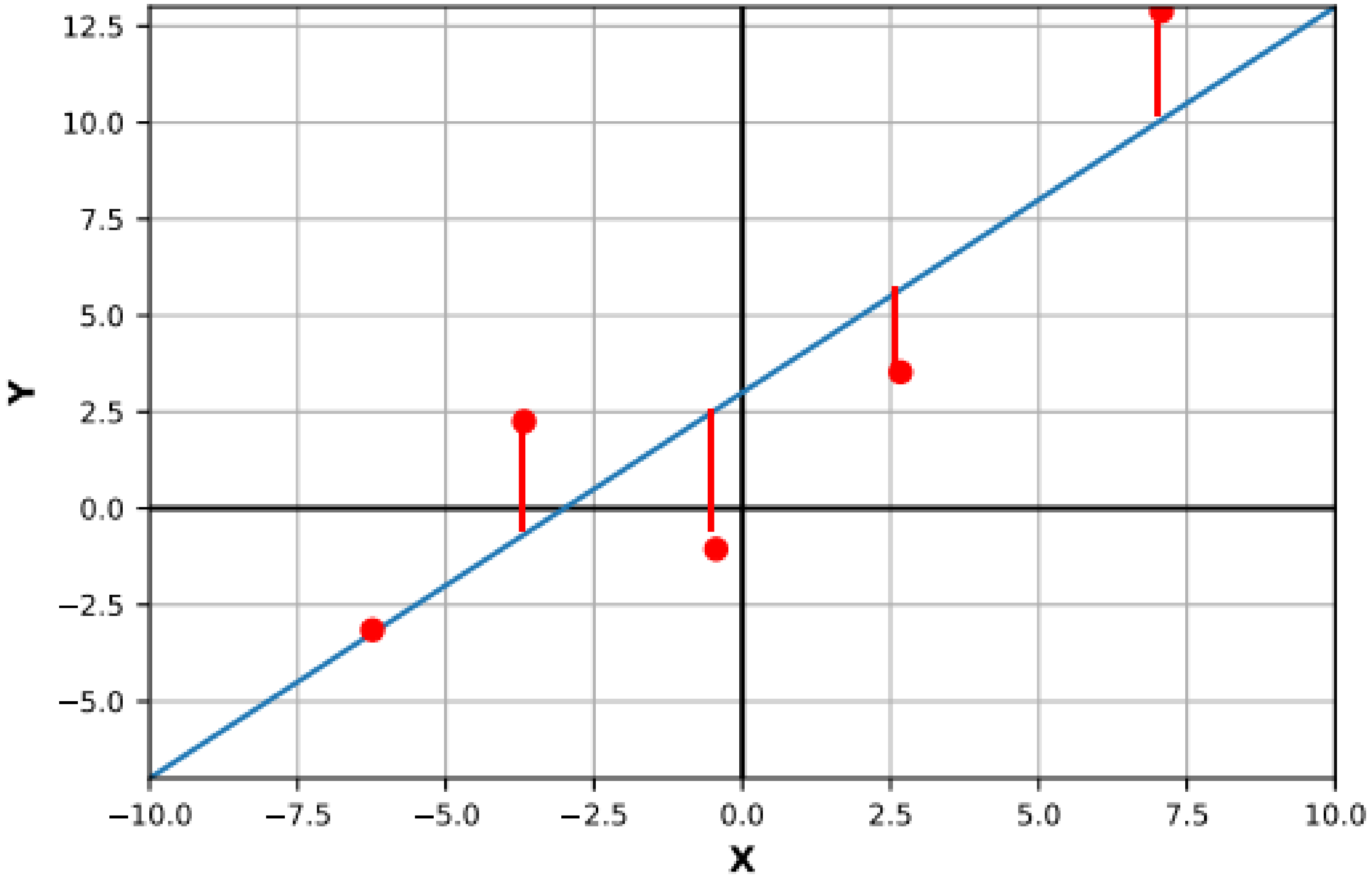
Maximum Likelihood Estimator

- Furthermore, since the right hand side of the above equation is monotonice in \theta the arg max will not change if we take log of the expression

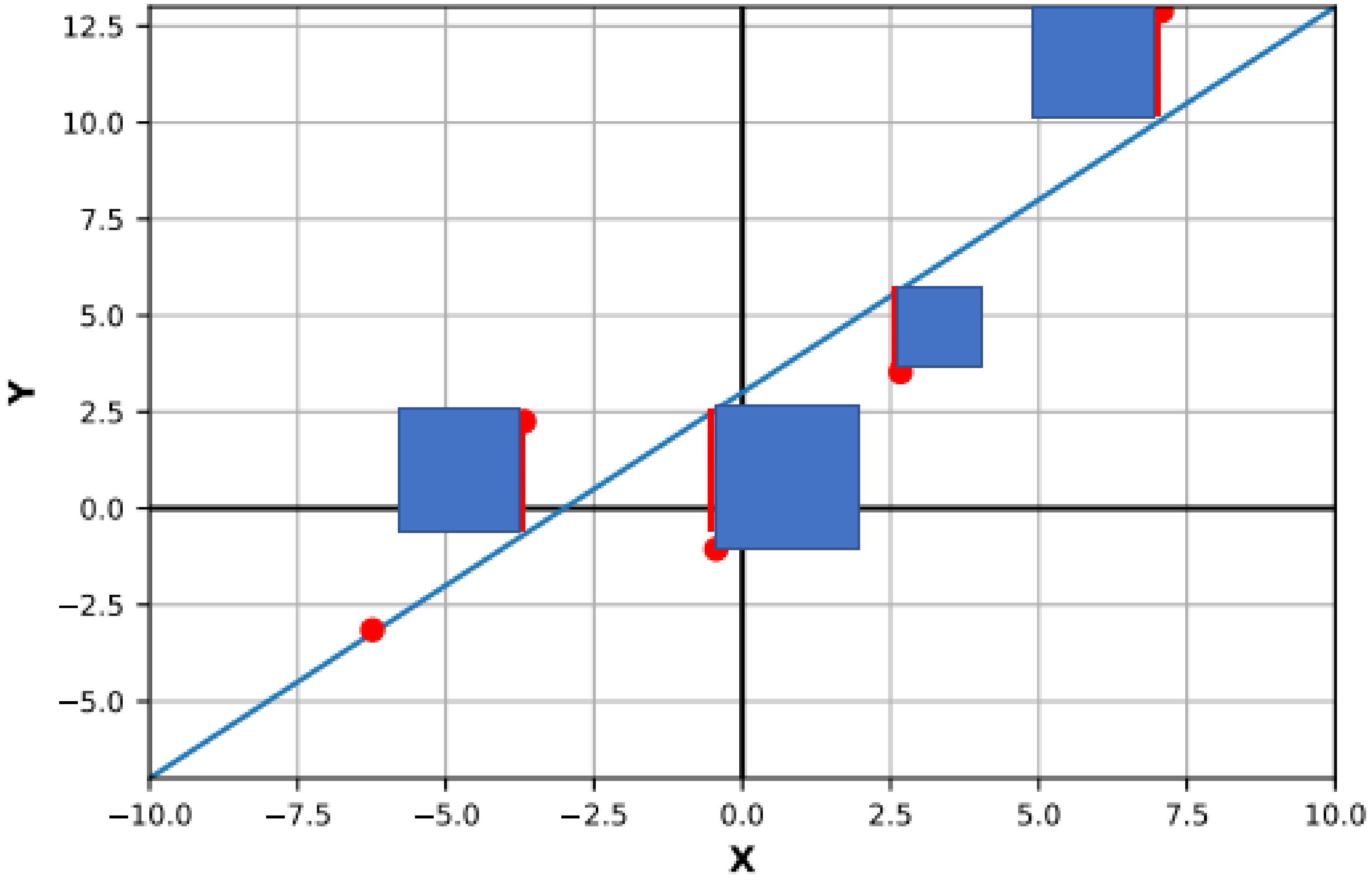
$$\arg\max_{\theta} \prod_{i=1}^{N} \exp\left(-(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta})^2\right) = \arg\max_{\theta} \sum_{i=1}^{N} \left(-(y_i - \boldsymbol{x}_i^T\boldsymbol{\theta})^2\right)$$

- Notice that the right hand side is minising the MSE.
- Hence solution of minimizing the MSE is equivalend to Maximum Likelihood Estimator for linear regression

Error Visualization

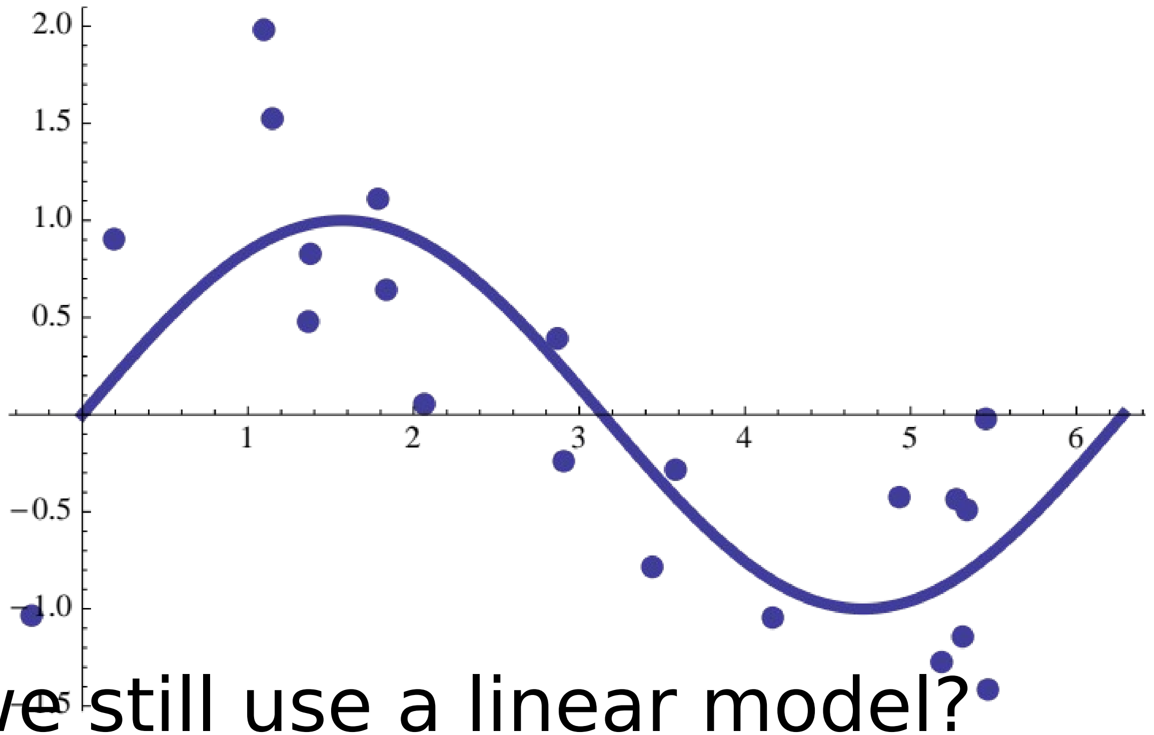


Error Visualization



Fitting Non-linear Data

- What if Y has a non-linear response?



- Can we still use a linear model?

Transforming the Feature Space

- Transform features x_i

$$x_i = (X_{i,1}, X_{i,2}, \ldots, X_{i,p})$$

- By applying non-linear transformation ϕ:

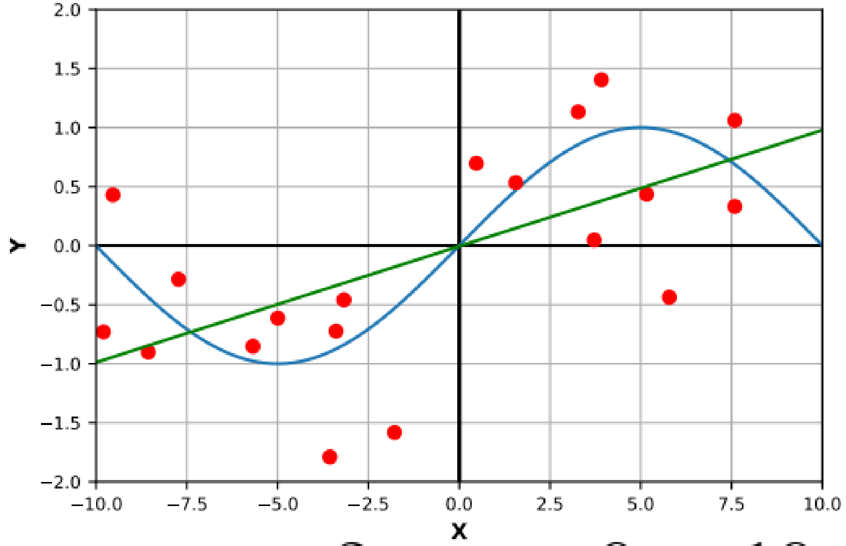
$$\phi : \mathbb{R}^p \to \mathbb{R}^k$$

- Example:

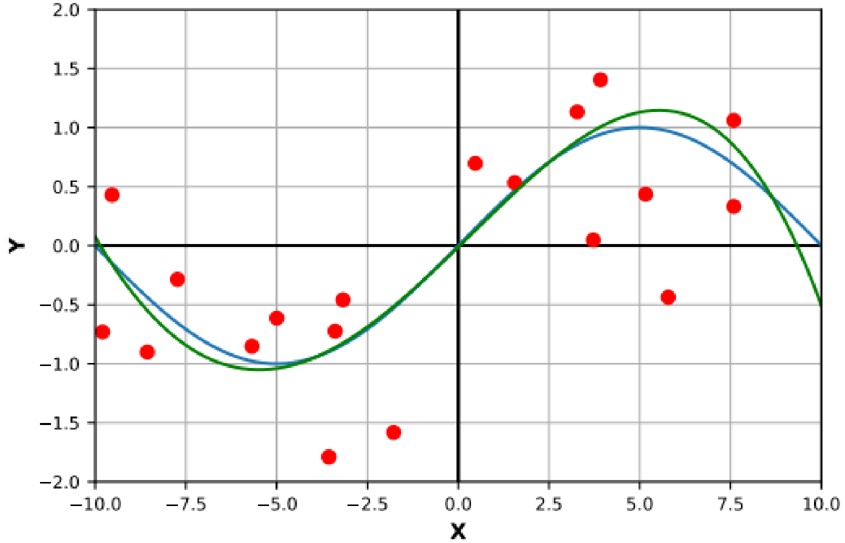
$$\phi(x) = \{1, x, x^2, \ldots, x^k\}$$

  - others: splines, radial basis functions, …
  - Expert engineered features (modeling)

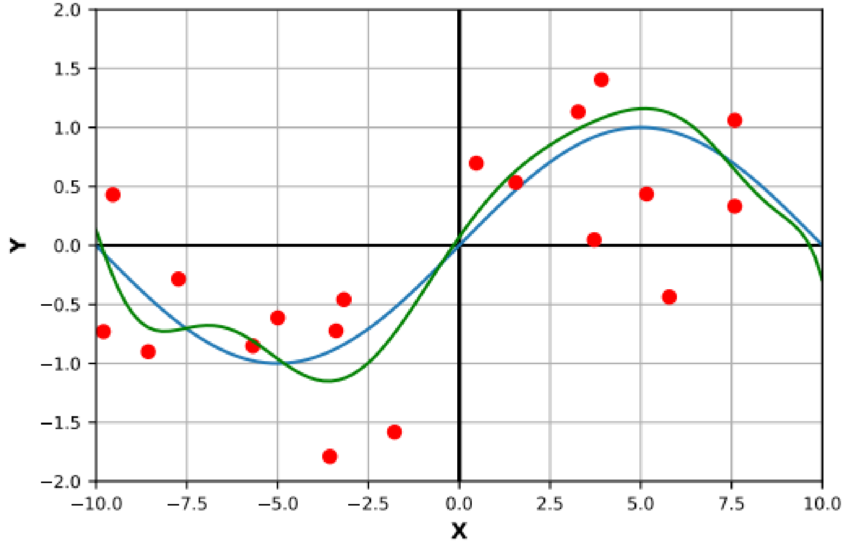
$$\{1, x\}$$



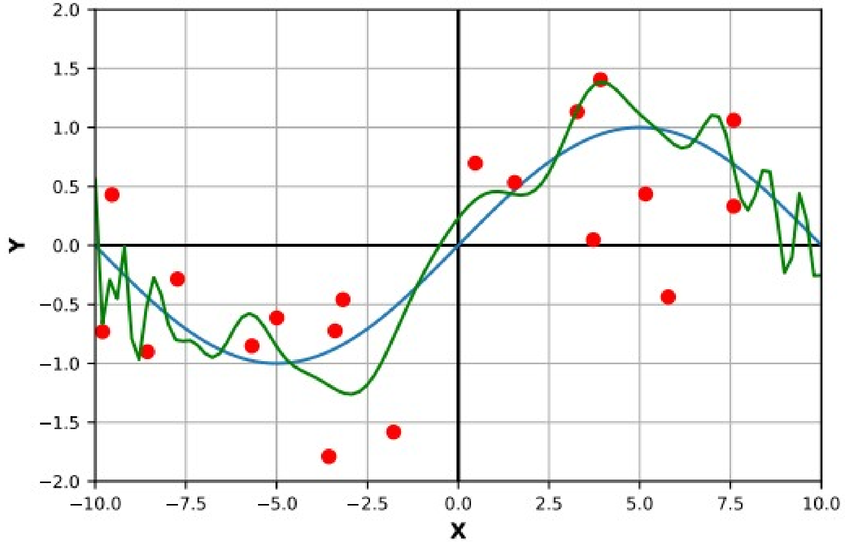
$$\{1, x, x^2, x^3, x^4\}$$



$$\{1, x, x^2, \ldots, x^9, x^{10}\}$$

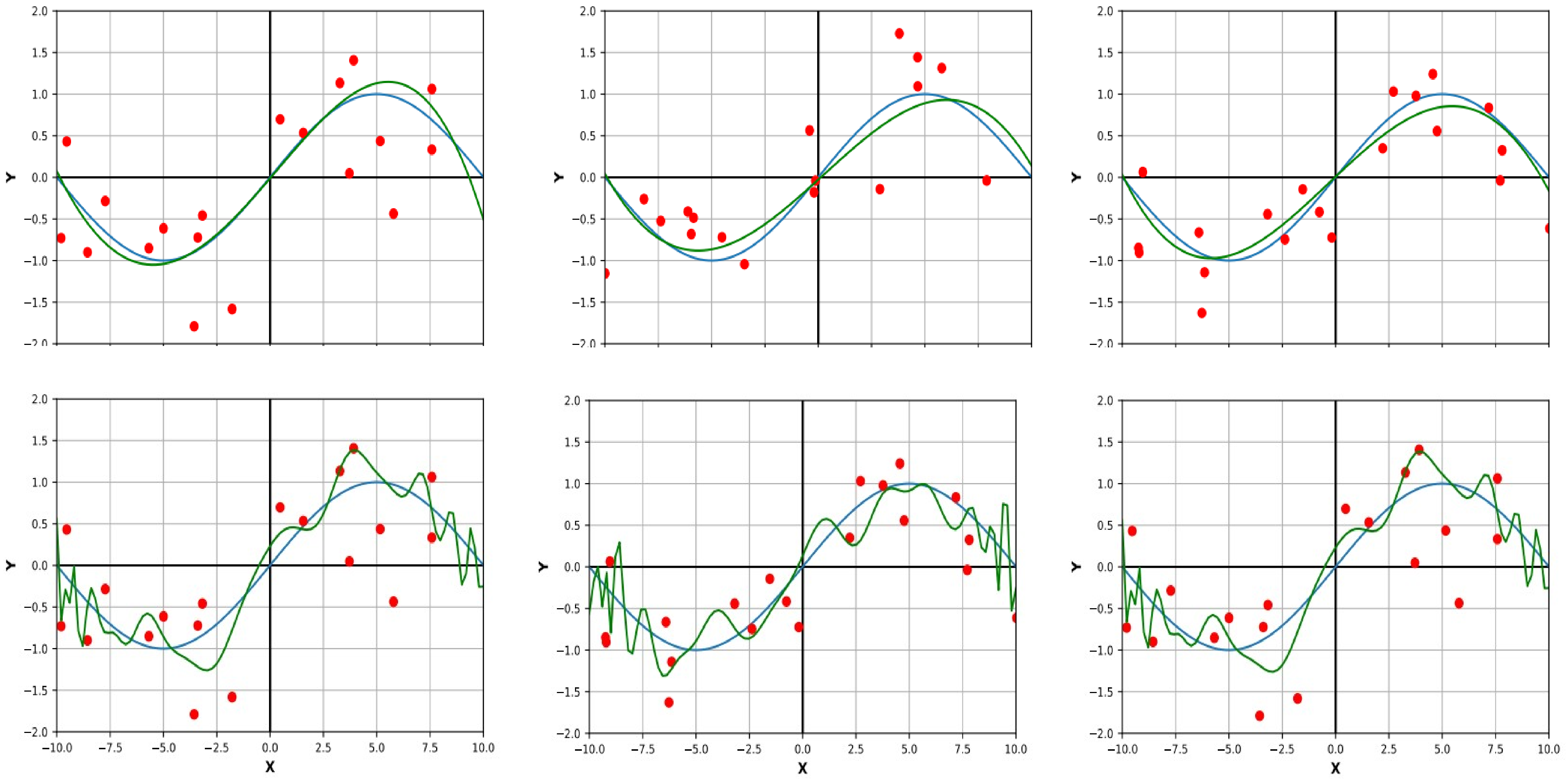


$$\{1, x, x^2, \ldots, x^{99}, x^{100}\}$$

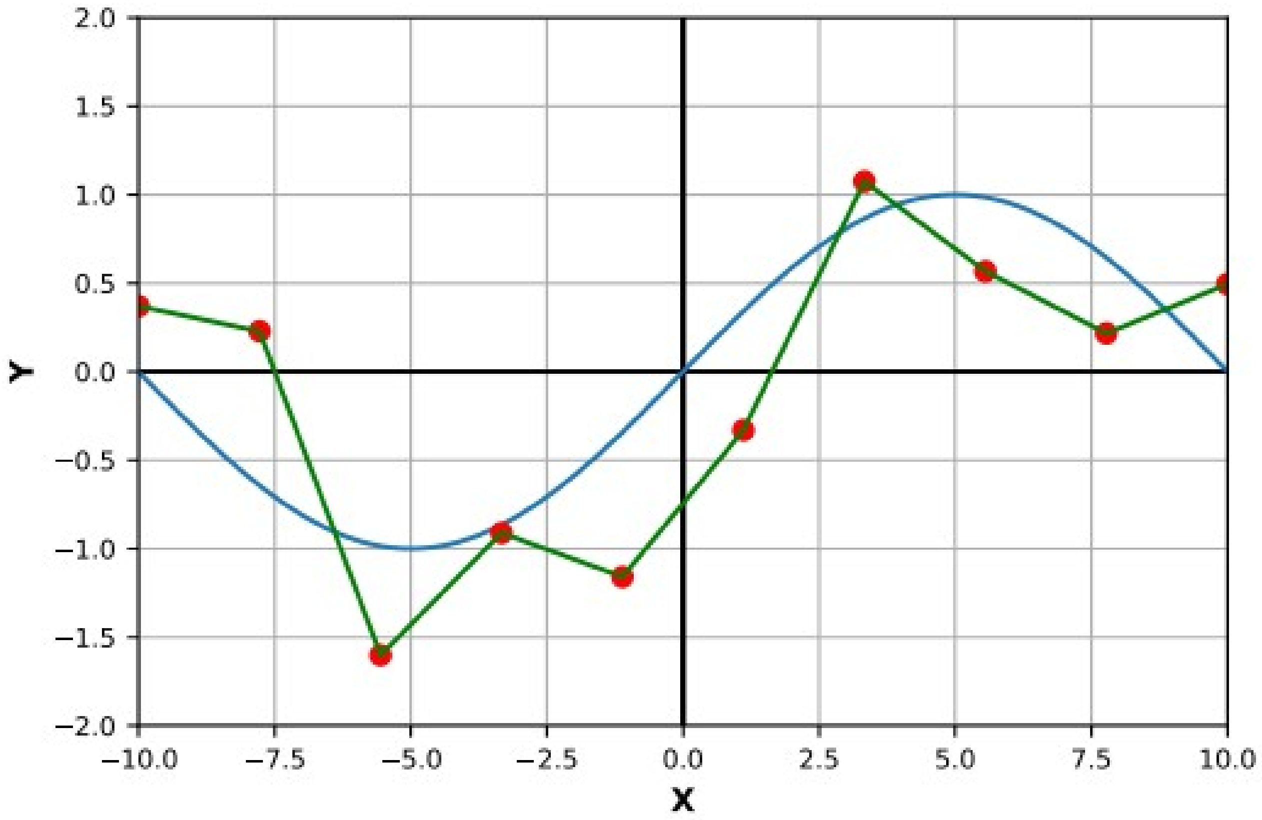


What is Bias and Variance?

$$\{1, x, x^2, x^3, x^4\}$$



Real Bad Overfit?



Bias-Variance Tradeoff

- So far we have minimized the error (loss) with respect to **training data**
  - Low training error does not imply good expected performance: **over-fitting**
- We would like to reason about the **expected loss (Prediction Risk)** over:
  - Training Data: $\{(y_1, x_1), \ldots, (y_n, x_n)\}$
  - Test point: (y_*, x_*)
- We will decompose the expected loss into:

$$\mathbf{E}_{D,(y_*,x_*)}\left[(y_* - f(x_*|D))^2\right] = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

Bias-Variance Trade Off (Intuition)

Bias Variance Plot

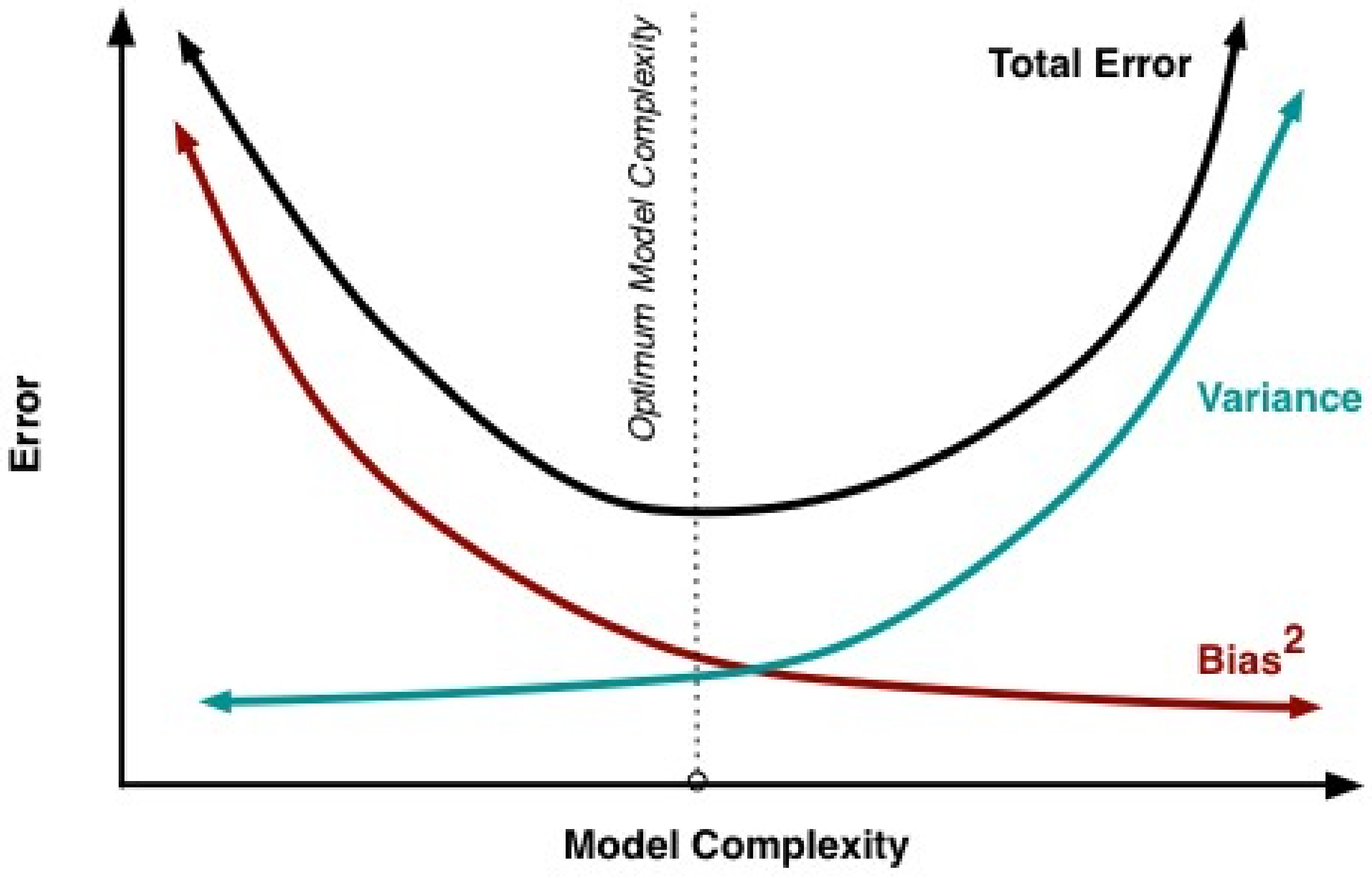


Image from http://scott.fortmann-roe.com/docs/BiasVariance.html

Bias Variance Application in Training

Regularization

Slides from: https://harvard-iacs.github.io/2019-CS109A/lectures/lecture8/presentation/

Regularization: An Overview

The idea of regularization revolves around modifying the loss function L; in particular, we add a regularization term that penalizes some specified properties of the model parameters

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta),$$

where is a scalar that gives the weight (or importance) of the regularization term.

Fitting the model using the modified loss function L_{reg} would result in model parameters with desirable properties (specified by R).

LASSO Regression

Since we wish to discourage extreme values in model parameter, we need to choose a regularization term that penalizes parameter magnitudes. For our loss function, we will again use MSE.

Together our regularized loss function is:

$$L_{LASSO}(\beta) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J}|\beta_j|.$$

Note that $\sum_{j=1}^{J}|\beta_j|$ is the l_1 norm of the vector $\boldsymbol{\beta}$

$$\sum_{j=1}^{J}|\beta_j| = \|\boldsymbol{\beta}\|_1$$

Ridge Regression

Alternatively, we can choose a regularization term that penalizes the squares of the parameter magnitudes. Then, our regularized loss function is:

$$L_{Ridge}(\beta) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^\top \boldsymbol{x}_i|^2 + \lambda \sum_{j=1}^{J} \beta_j^2.$$

Note that $\sum_{j=1}^{J} |\beta_j|^2$ is the square of the l_2 norm of the vector $\boldsymbol{\beta}$

$$\sum_{j=1}^{J} \beta_j^2 = \|\boldsymbol{\beta}\|_2^2$$

Choosing λ

In both ridge and LASSO regression, we see that the larger our choice of the **regularization parameter** λ, the more heavily we penalize large values in $\boldsymbol{\beta}$,

- If λ is close to zero, we recover the MSE, i.e. ridge and LASSO regression is just ordinary regression.
- If λ is sufficiently large, the MSE term in the regularized loss function will be insignificant and the regularization term will force $\boldsymbol{\beta}_{\text{ridge}}$ and $\boldsymbol{\beta}_{\text{LASSO}}$ to be close to zero.

To avoid ad-hoc choices, we should select λ using cross-validation.

Ridge, LASSO - Computational complexity

Solution to ridge regression:

$$\beta = (X^T X + \lambda I)^{-1} X^T Y$$

The solution to the LASSO regression:

LASSO has no conventional analytical solution, as the L1 norm has no derivative at 0. We can, however, use the concept of subdifferential or subgradient to find a manageable expression. See a–sec2 for details.

Regularization Parameter with a Validation Seet

The solution of the Ridge/Lasso regression involves three steps:

- Select λ
- Find the minimum of the ridge/Lasso regression loss function (using the formula for ridge) and record the *MSE* **on the validation set**.
- Find the λ that gives the smallest *MSE*

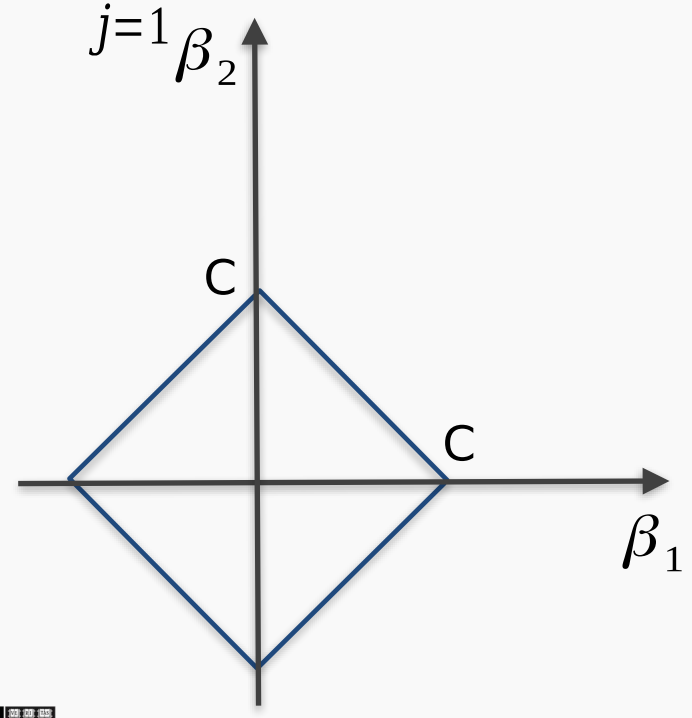
The Geometry of Regularization (LASSO)

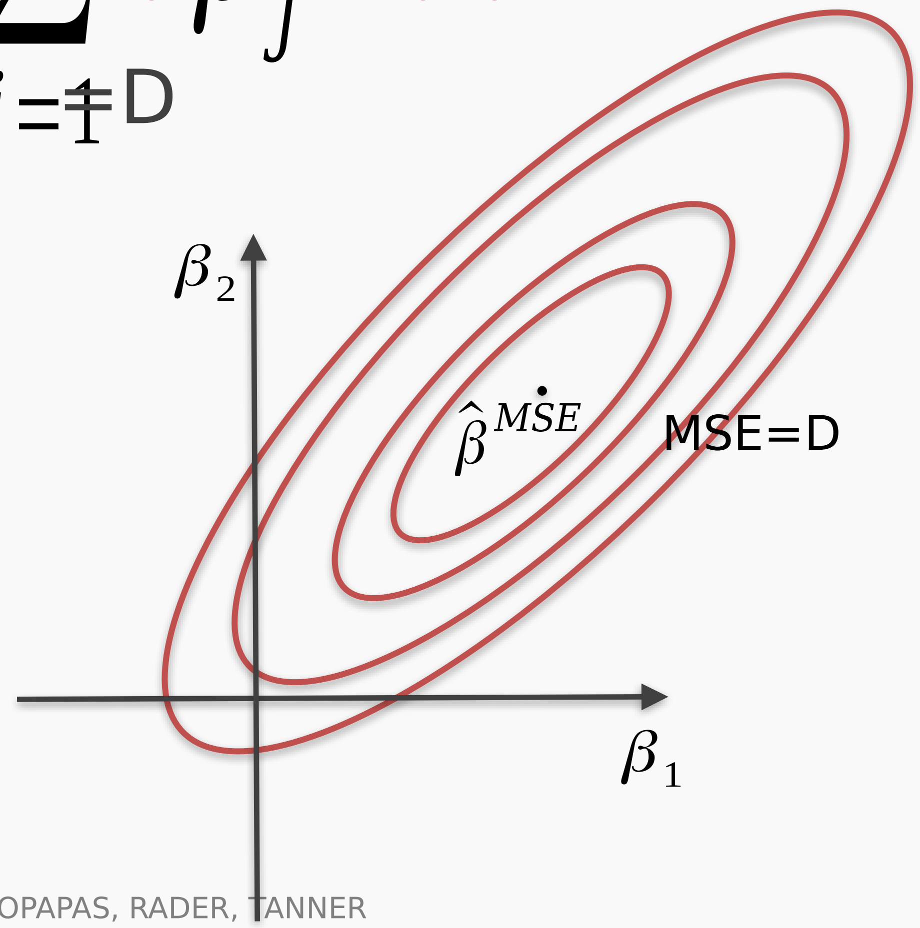
$$L_{LASSO}(\beta)=\frac{1}{n}\sum_{i=1}^{n} ¿\, y_i-\beta^T x\,|^2+\lambda\sum_{j=1}^{J} ¿\,\beta_j \vee ¿¿$$

$$\hat{\beta}^{LASSO}=\operatorname{argmin} L_{LASSO}(\beta)$$

$$\lambda\sum_{j=1}^{J}\left|\hat{\beta}_j^{LASSO}\right|=C$$

$=D$





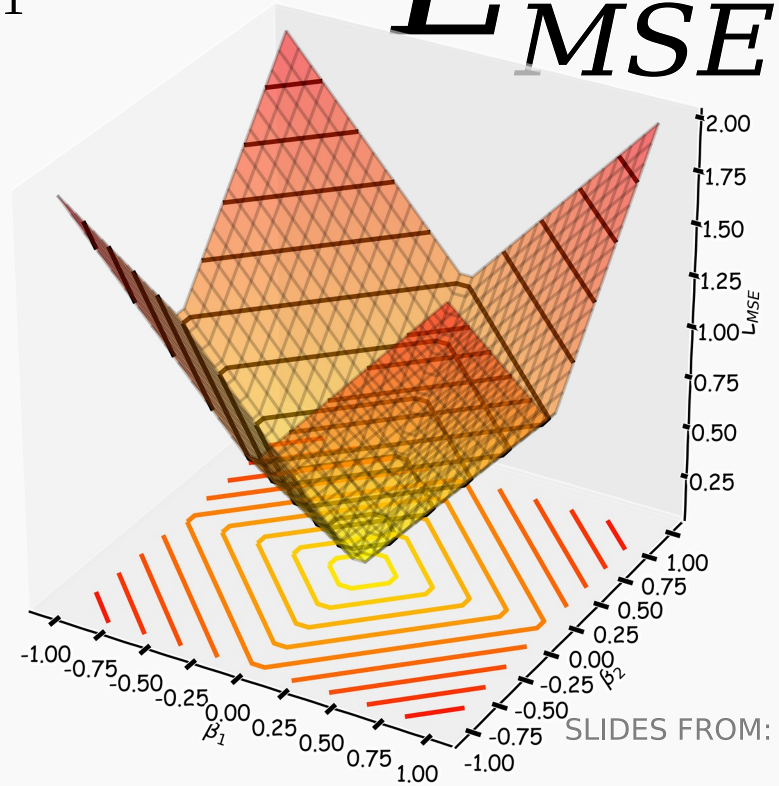
The Geometry of Regularization (LASSO)

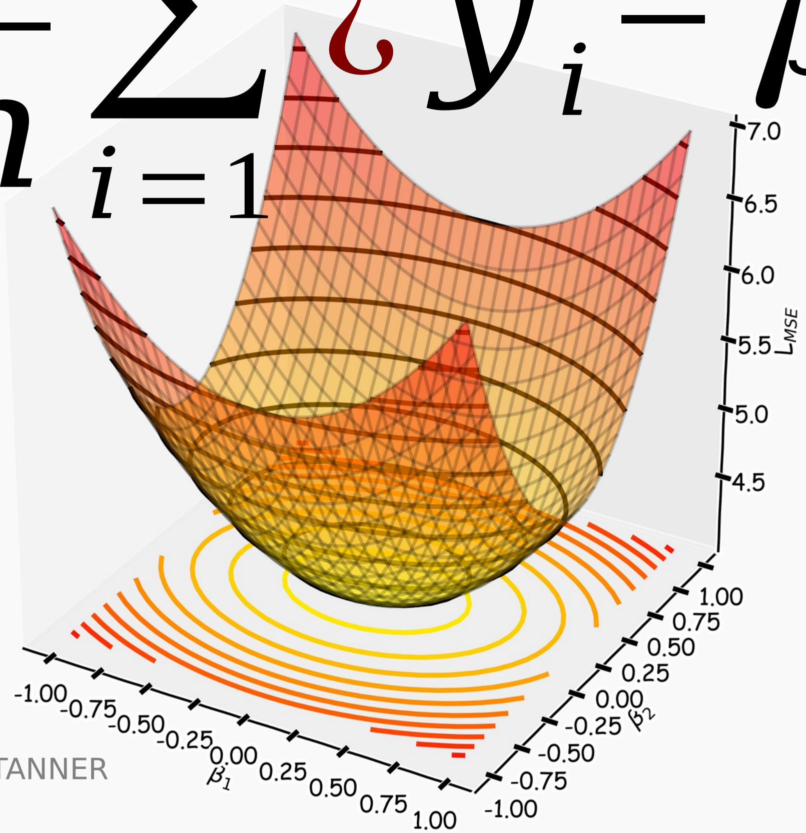
$$L_{LASSO}(\beta) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \beta^T x|^2 + \lambda \sum_{j=1}^{J} |\beta_j|$$

$$\hat{\beta}^{LASSO} = \arg\min L_{LASSO}(\beta)$$

$$L_1 = \lambda \sum_{j=1}^{J} |\hat{\beta}_j^{LASSO}|$$

$$L_{MSE}(\beta) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \beta^T x|^2$$





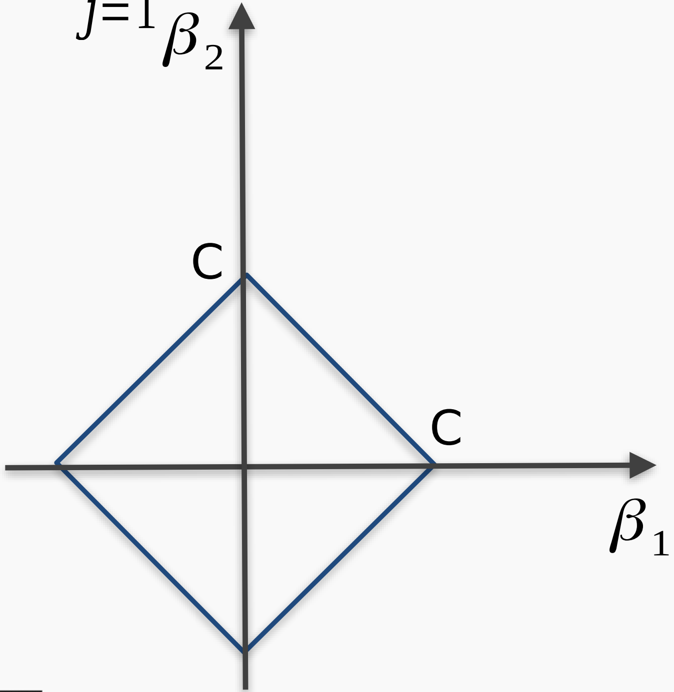
The Geometry of Regularization (LASSO)

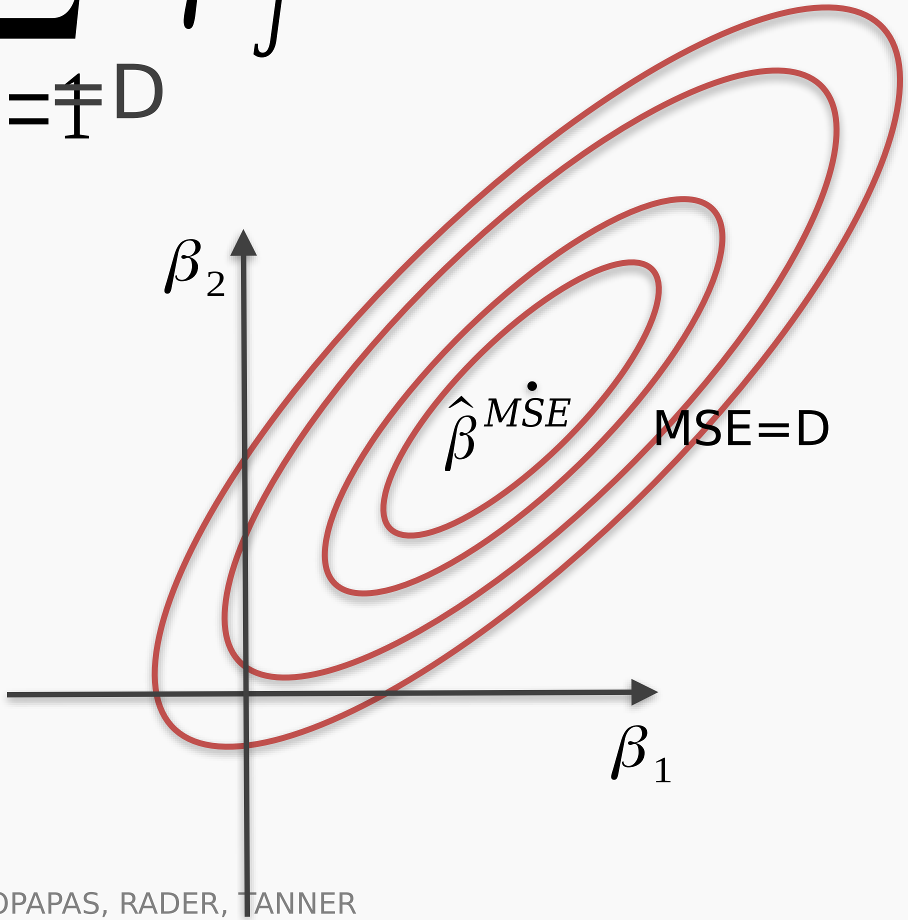
$$\ldots_{O}\left(\beta\right)=\frac{1}{n}\sum_{i=1}^{n} ¿\, y_i-\beta^T x\,|^2+\lambda\sum_{j=1}^{J} ¿\,\beta_j \vee ¿\,¿$$

$$\hat{\boldsymbol{\beta}}^{LASSO}=argmin\, L_{LASSO}\left(\boldsymbol{\beta}\right)$$

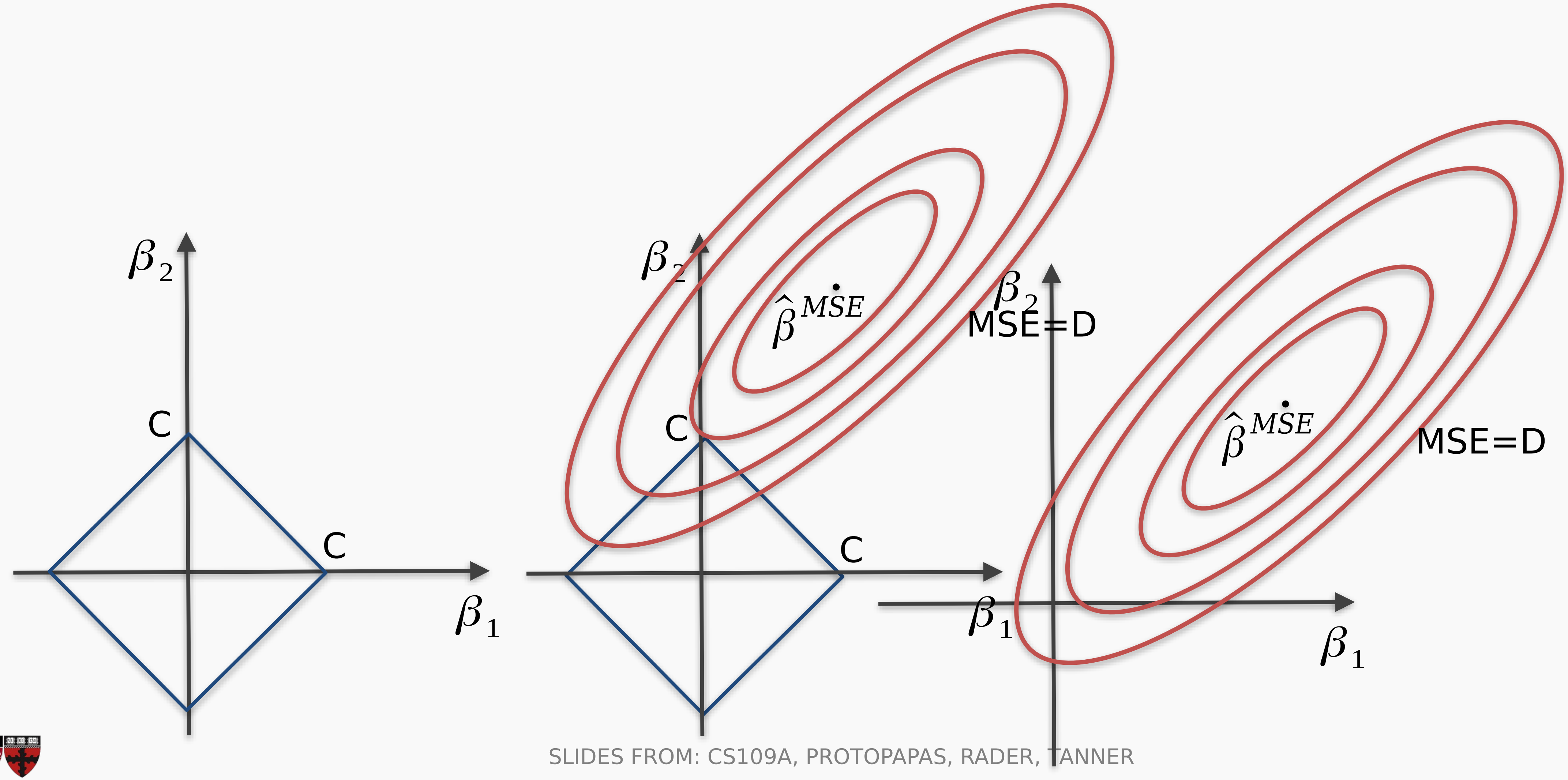
$$\lambda\sum_{j=1}^{J}\left|\hat{\beta}_j^{LASSO}\right|=C$$

=D





The Geometry of Regularization (LASSO)



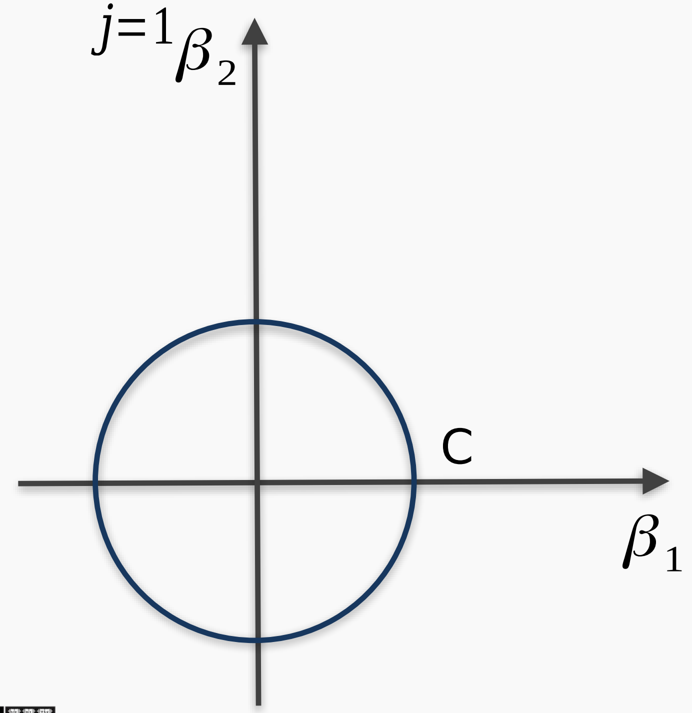
The Geometry of Regularization (Ridge)

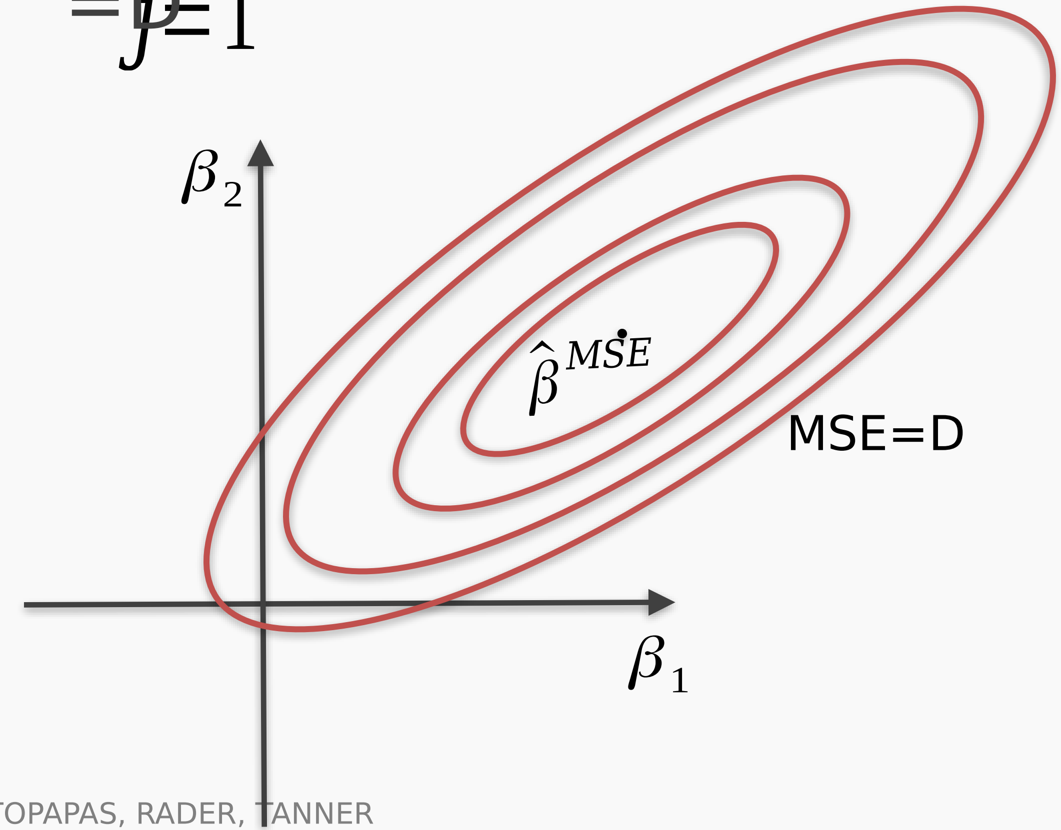
$$L_{Ridge}(\beta) = \frac{1}{n}\sum_{i=1}^{n} ¿\, y_i - \beta^T x |^2 + \lambda \sum_{j=1}^{J} (\beta_j)^2$$

$$\hat{\beta}^{Ridge} = argmin\, L_{Ridge}(\beta)$$

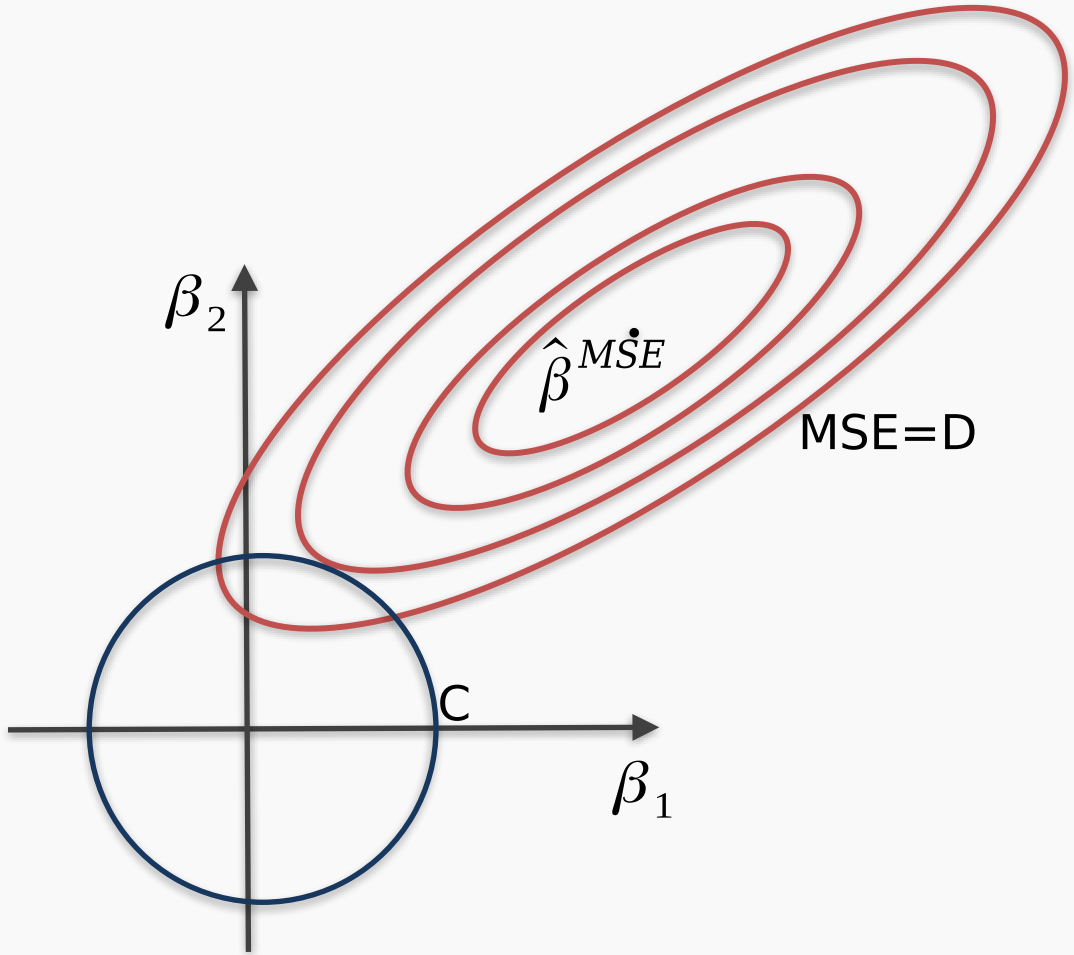
$$\lambda \sum_{j=1}^{J} |\hat{\beta}_j^{Ridge}|^2 = C$$

$= D$

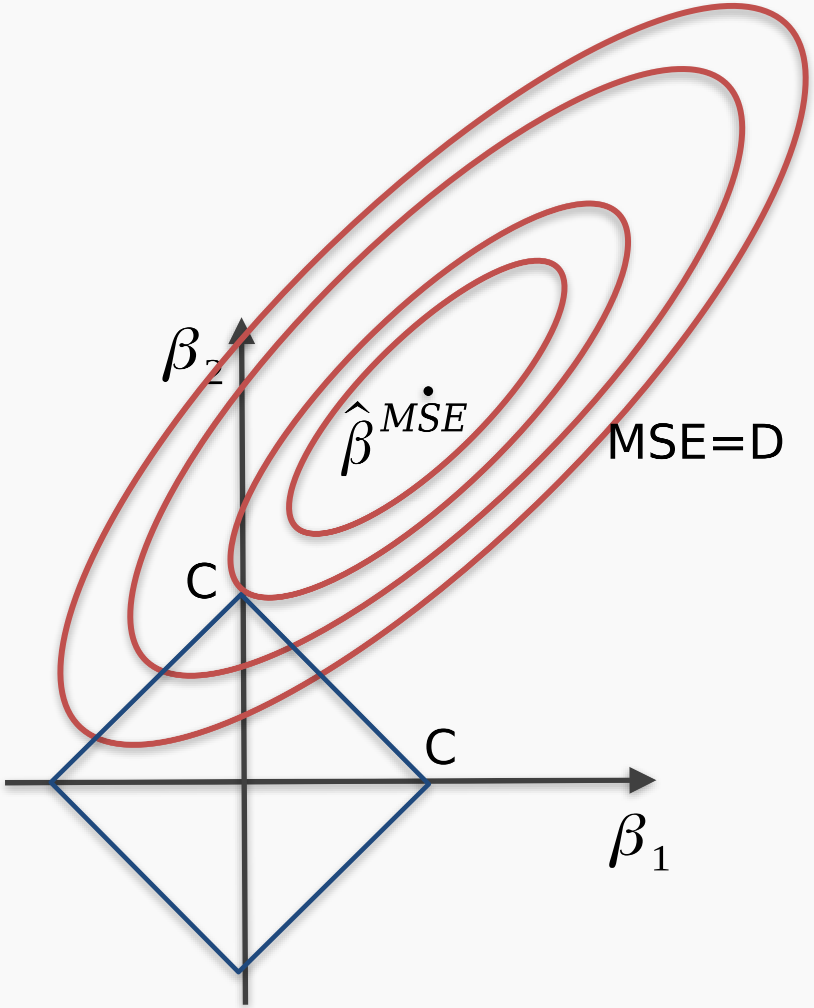


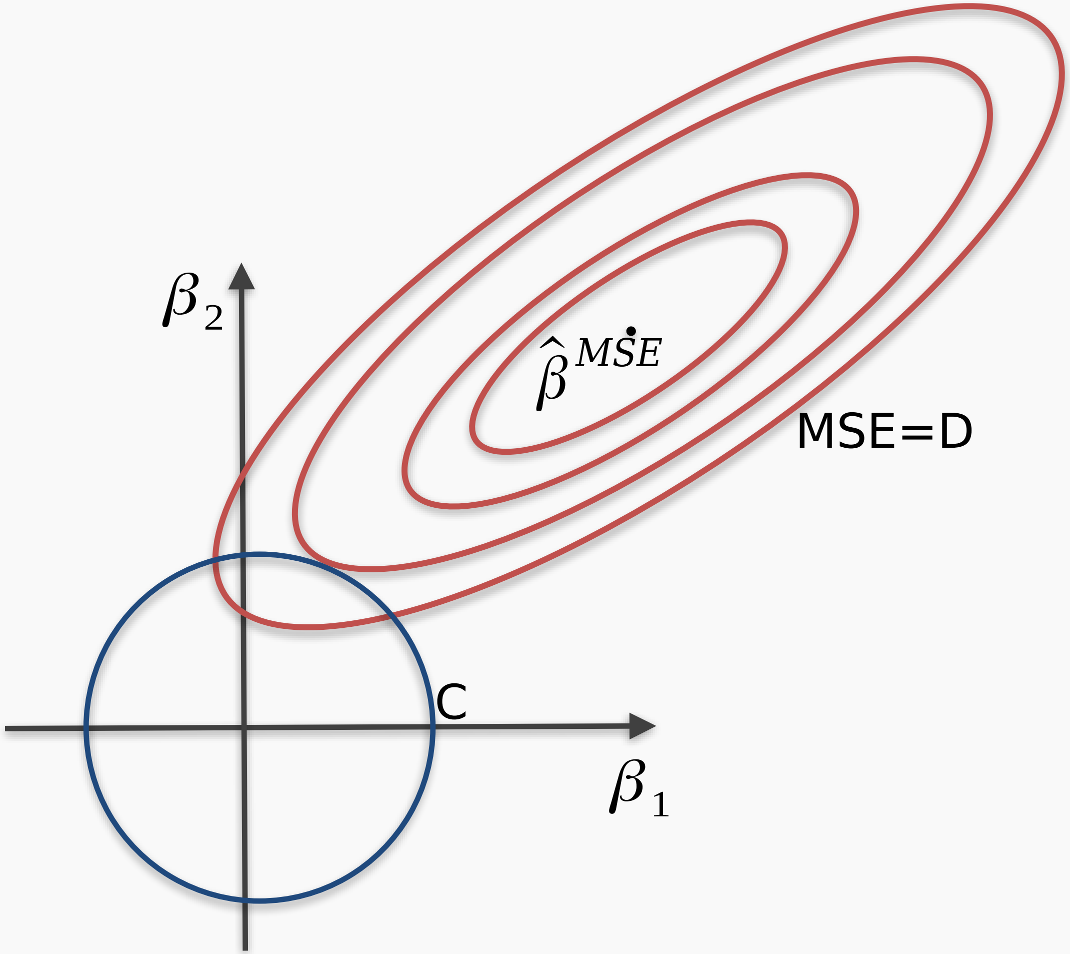


The Geometry of Regularization (Ridge)



The Geometry of Regularization





Examples

In [ ]:
```
from sklearn.linear_model import Lasso
```

In [22]:
```
lasso_regression = Lasso(alpha=1.0, fit_intercept=True)
lasso_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

print('Lasso regression model:\n {} + {}^T . x'.format(lasso_regression.intercept_, lasso_regression.coe
```

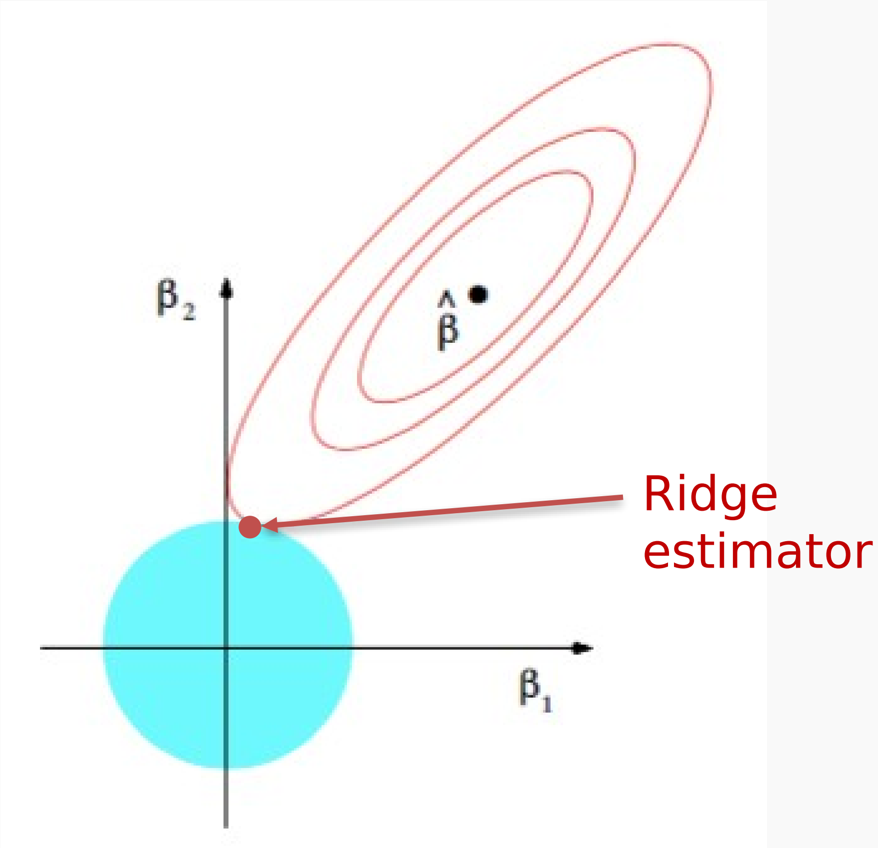
```
Lasso regression model:
 10.424895873901445 + [ 0.24482603  3.48164594  1.84836859 -0.06864603 -0.          -0.
 -0.02249766 -0.          0.          0.          0.          0.        ]^T . x
```

In [ ]:
```
from sklearn.linear_model import Ridge
```

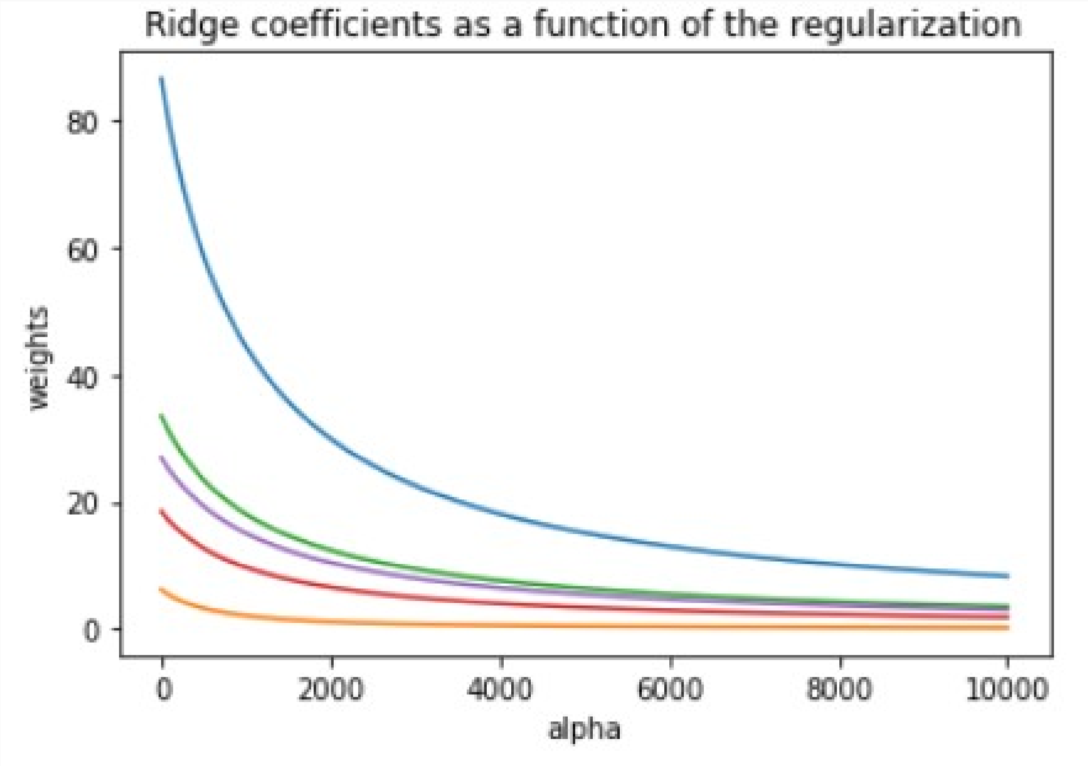
In [20]:
```
X_train = train[all_predictors].values
X_val = validation[all_predictors].values
X_test = test[all_predictors].values

ridge_regression = Ridge(alpha=1.0, fit_intercept=True)
ridge_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

print('Ridge regression model:\n {} + {}^T . x'.format(ridge_regression.intercept_, ridge_regression.coe
```

```
Ridge regression model:
 -525.7662550875951 + [ 0.24007312  8.42566029  2.04098593 -0.04449172 -0.01227935  0.41902475
 -0.50397312 -4.47065168  4.99834262  0.          0.          0.29892679]^T . x
```

Ridge visualized

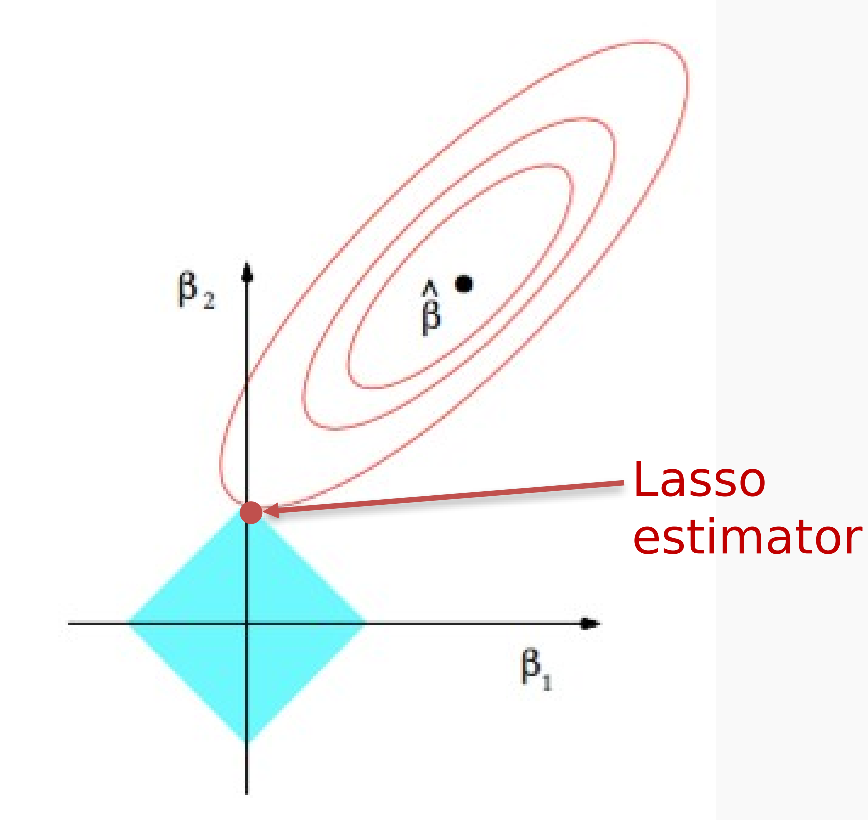


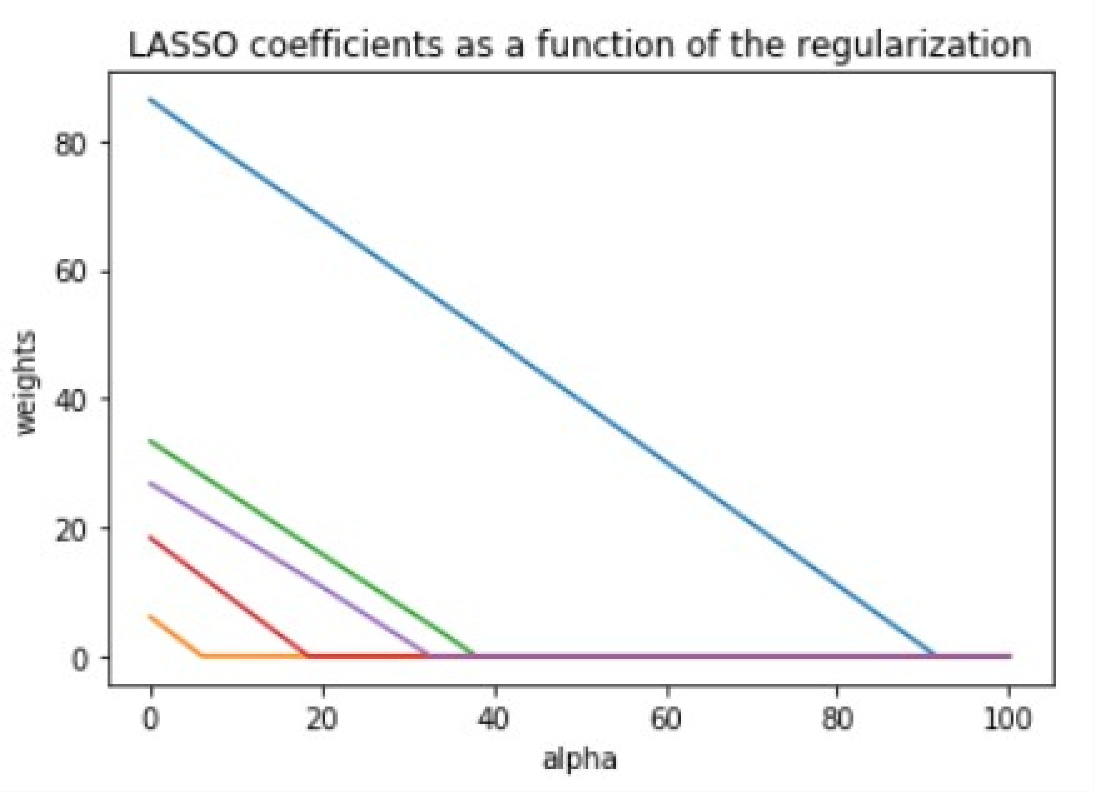
The ridge estimator is where the constraint and the loss intersect.



The values of the coefficients decrease as lambda increases, but they are not nullified.

LASSO visualized





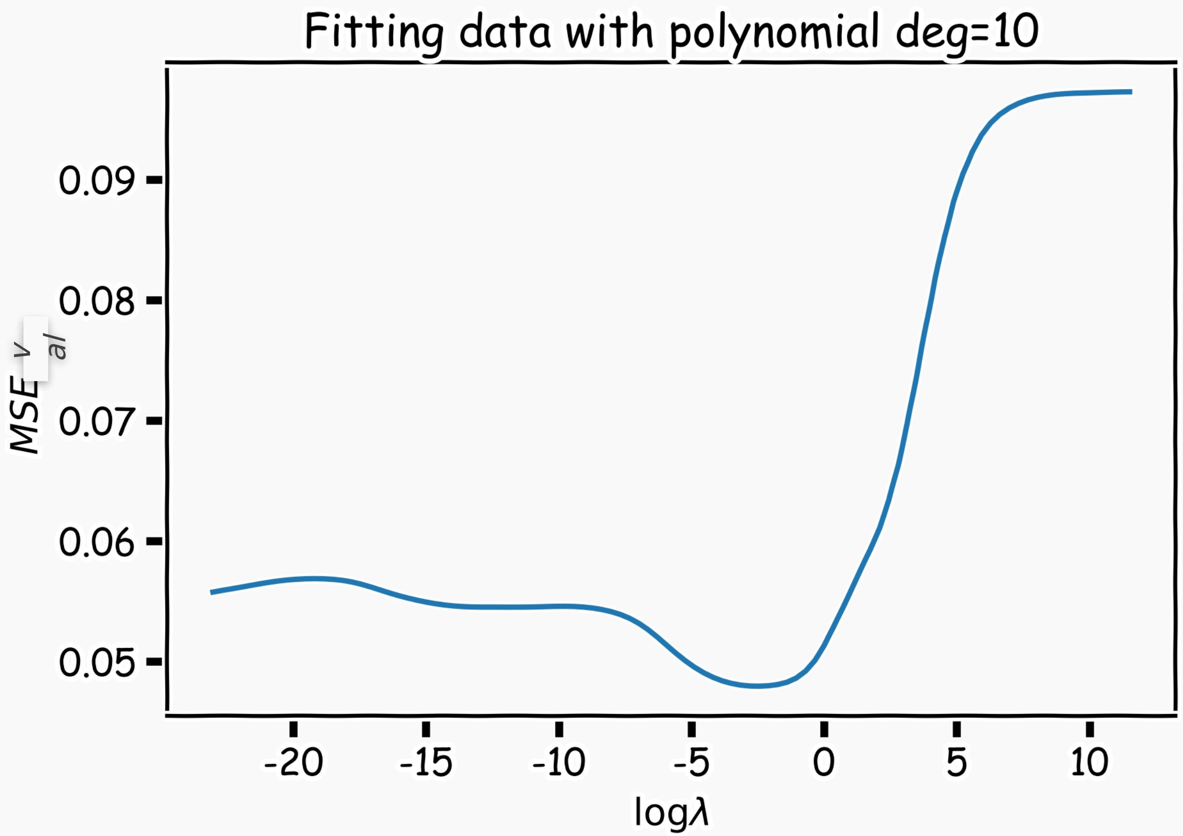
The Lasso estimator tends to zero out parameters as the OLS loss can easily intersect with the constraint on one of the axis.

The values of the coefficients decrease as lambda increases, and are nullified fast.

Ridge regularization with only **validation** : step by step

1. split data into
2. for
   1. determine the that minimizes the , , using the train data.
   2. record using validation data.
3. select the that minimizes the loss on the validation data,
4. Refit the model using both train and validation data, }, resulting to
5. report MSE or R^2 on given the

Ridge regularization with **validation** only: step by step



Lasso regularization with **validation** only: step by step

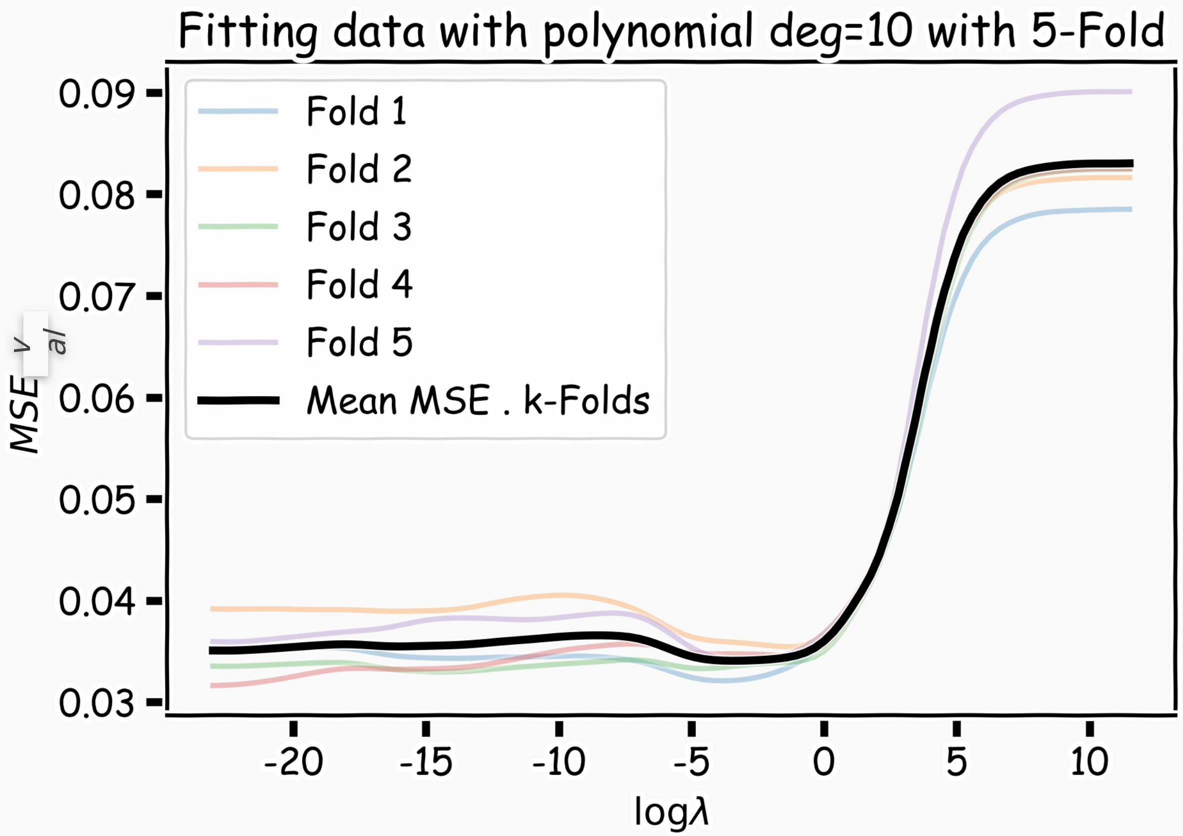
1. split data into
2. for
   A. determine the that minimizes the , , using the train data. **This is done using a solver.**
   B. record using validation data
3. select the that minimizes the loss on the validation data,
4. Refit the model using both train and validation data, }, resulting to
5. report MSE or R^2 on given the

Ridge regularization with **CV**: step by step

			...	
			..	...
		...	..	...
...	..	...	..	...
	...	...	...	...
E[]			...	

1. remove from data
2. split the rest of data into K folds,
3. for *k* in
   1. for
      A. determine the that minimizes the , , using the train data of the fold, .
      B. record using the validation data of the fold
      At this point we have a 2-D matrix, rows are for different k, and columns are for different values.
4. Average the for each , .
5. Find the that minimizes the , resulting to .
6. Refit the model using the full training data, }, resulting to
7. report MSE or R^2 on given the

Ridge regularization with **validation** only: step by step



Variable Selection as Regularization

Since LASSO regression tend to produce zero estimates for a number of model parameters - we say that LASSO solutions are **sparse** - we consider LASSO to be a method for variable selection.

Many prefer using LASSO for variable selection (as well as for suppressing extreme parameter values) rather than stepwise selection, as LASSO avoids the statistic problems that arises in stepwise selection.

Question: What are the pros and cons of the two approaches?