Artificial Intelligence and Machine Learning

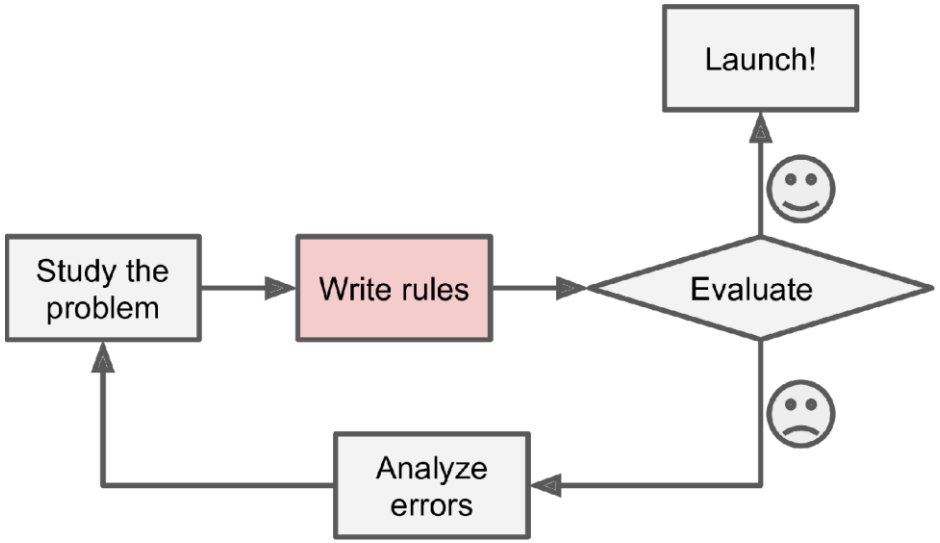
Linear Regression

Lecture 1: Outline

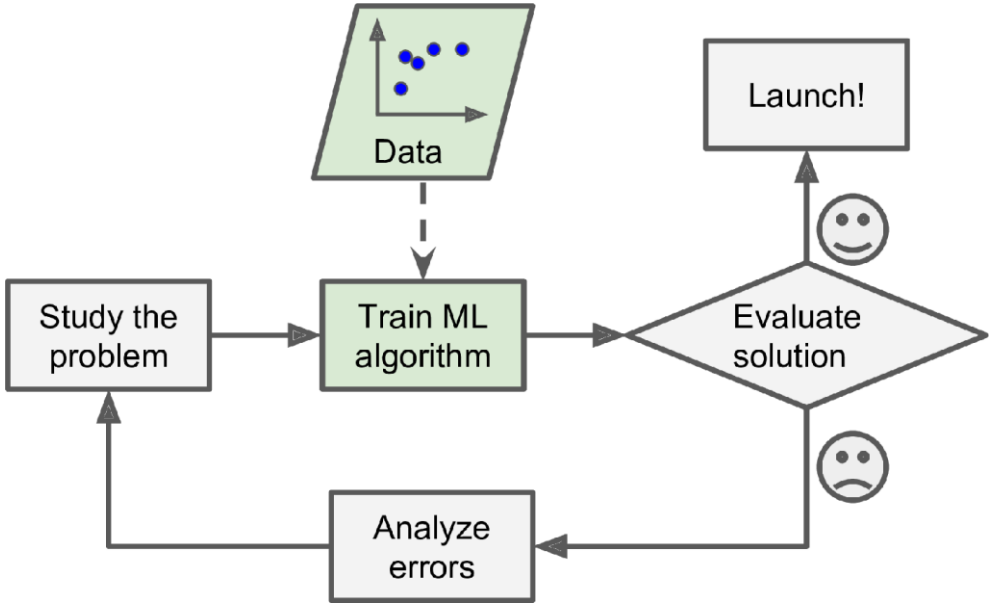
- Introduction to ML
- Linear Regression
- Optimization
- Linear Regression: Probabilistic Interpretation
- Bias-Variance Tradeoff
- Regularization

Introduction to ML

- **Machine Learning** is the science (and art) of programming computers so they can learn from data.



The traditional approach



The Machine Learning approach

Data Types

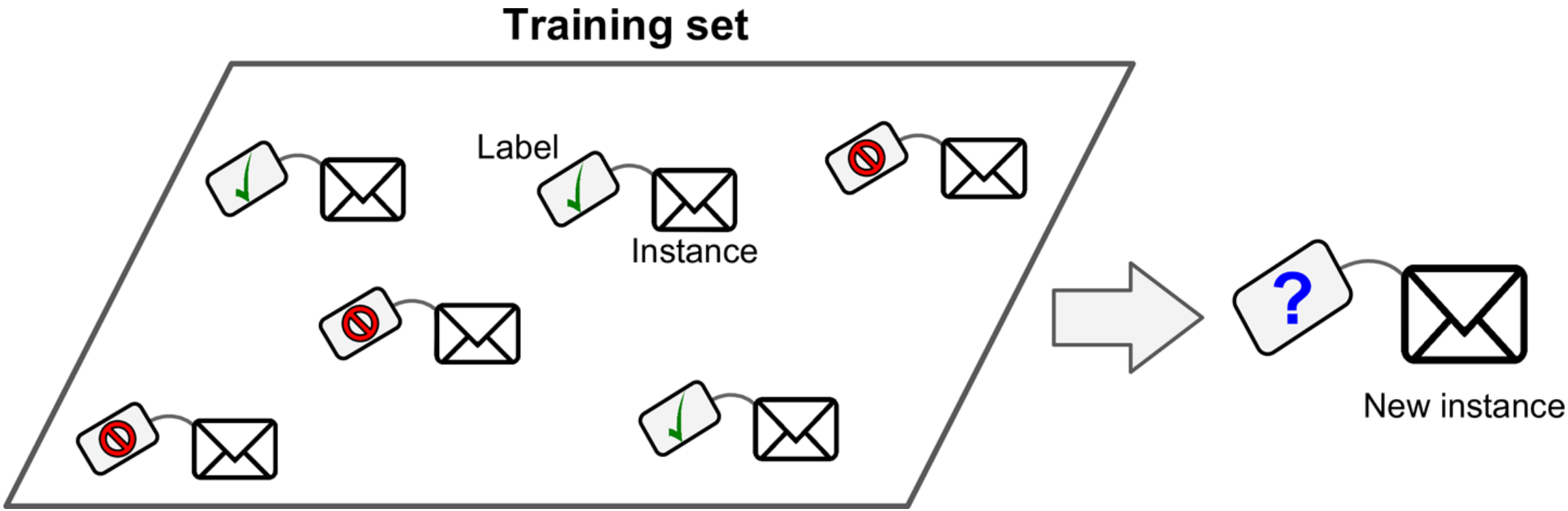
- **Tabular Data** (e.g., spreadsheets, databases)
  - Note: Columns are called **Features**. Rows are called **Samples.**
- **Time-Series Data** (e.g., stock prices, weather forecasts, IoT sensor data)
- **Text Data** (Natural Language Processing, e.g., emails, social media posts, documents)
- **Images and Videos** (Computer Vision, e.g., medical imaging, surveillance, facial recognition)
- **Audio Data** (Speech Recognition, Music Processing, e.g., voice commands, podcasts, sound classification)

ML Algorithms Types

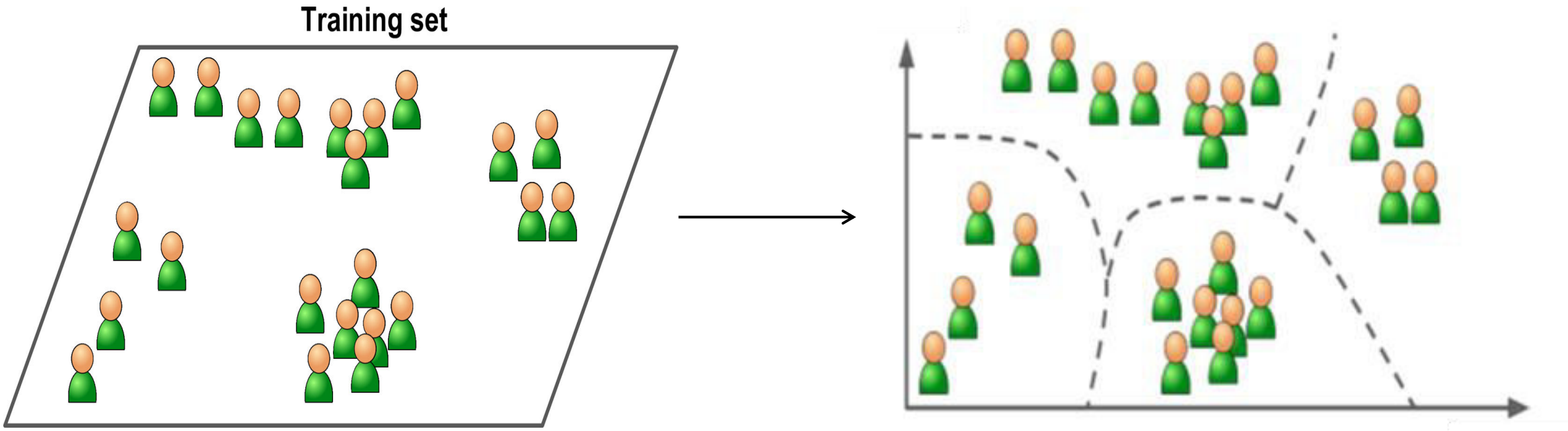
- **Supervised**: There is a target we want to predict.
  - **Regression**: Predict continuous value (e.g. house prices).
  - **Classification**: Predict discrete value (e.g. spam/not-spam).
- **Unsupervised**: There is no target. We are interested in things like:
  - **Clustering**: Grouping
  - **Dimensionality Reduction**: Reducing the Dimensions
  - **Anomaly Detection**: Detecting outliers

ML Algorithms Types cont.

- **Reinforcement Learning**: involves learning to make decisions by interacting with an environment.
  - **Reward Signal:** The agent receives feedback in the form of rewards or penalties, guiding its learning.
  - **Policy:** A strategy the agent learns to decide actions based on the current state.
  - **Value Function:** An estimate of the expected cumulative reward from a state or state-action pair.
  - **Exploration vs Exploitation:** The agent balances exploring new actions to discover rewards and exploiting known actions to maximize them.
  - Really popular in video games and robotics!(Also recently in LLMs, see RLHF)



An example of Supervised Learning: Spam Classification



An example of Unsupervised Learning: Clustering

How Does ML Work?

- Any ML system consists of three main components:

  Hypothesis (Model): The function that approximates the target.
    - E.g. Linear Regression, Logistic Regression, SVM, Decision Trees, NN,…

  Optimizer: The mechanism for improving predictions of our model.

  Loss Function: The measure of how wrong the predictions are.

How Does ML Work?

- How are they related to each other?🤔

How Does ML Work?

- We firstly define our task (classification/regression) then choose an appropriate **model.**
- We will use an **optimization method** to minimize the **loss function**.
- Reached a minima?

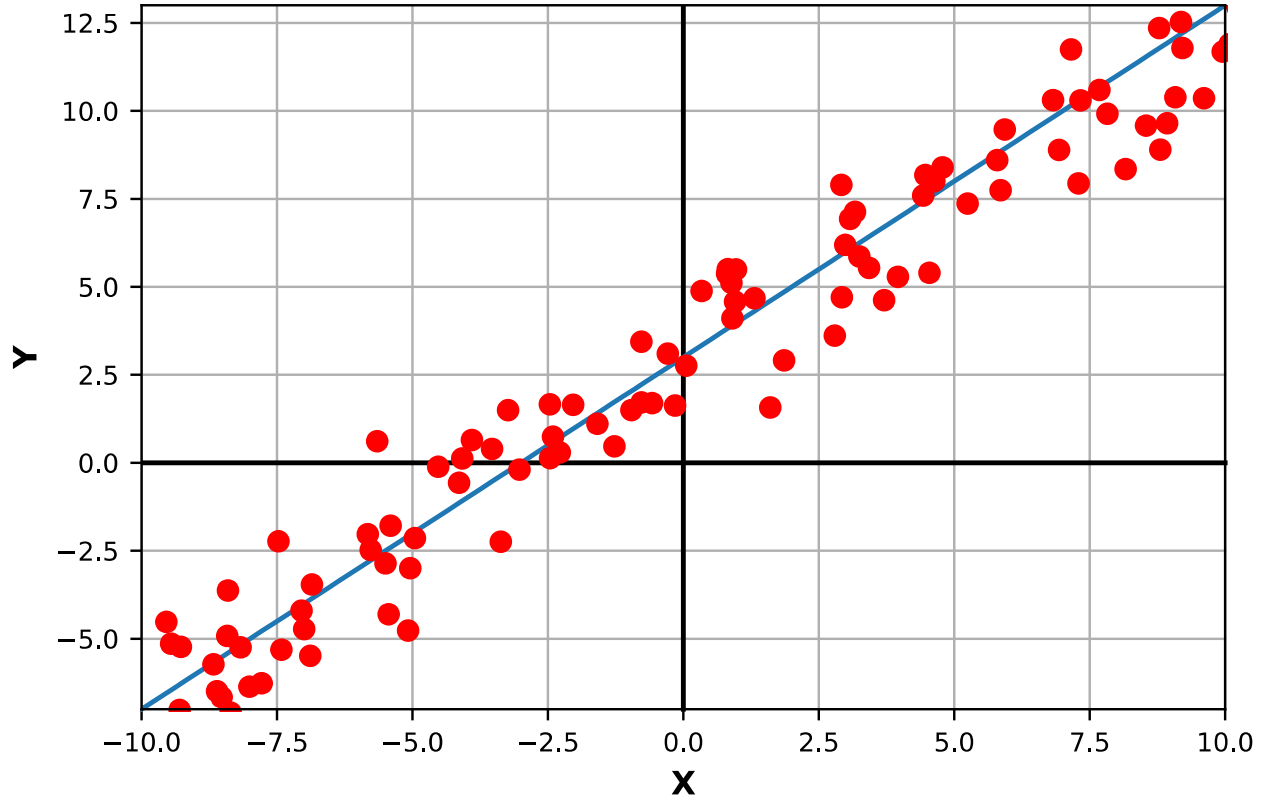
  = Model is making the least possible number of mistakes.

  = **Model trained** 🎉.

Linear Regression: 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

- **Hypothesis**:

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

- **Input**: data (x_i, y_i), $i \in \{1,2, \dots, N\}$
  - (e.g., house size x and price y)

- **Goal**: learn values of variable (m, b)

Notation

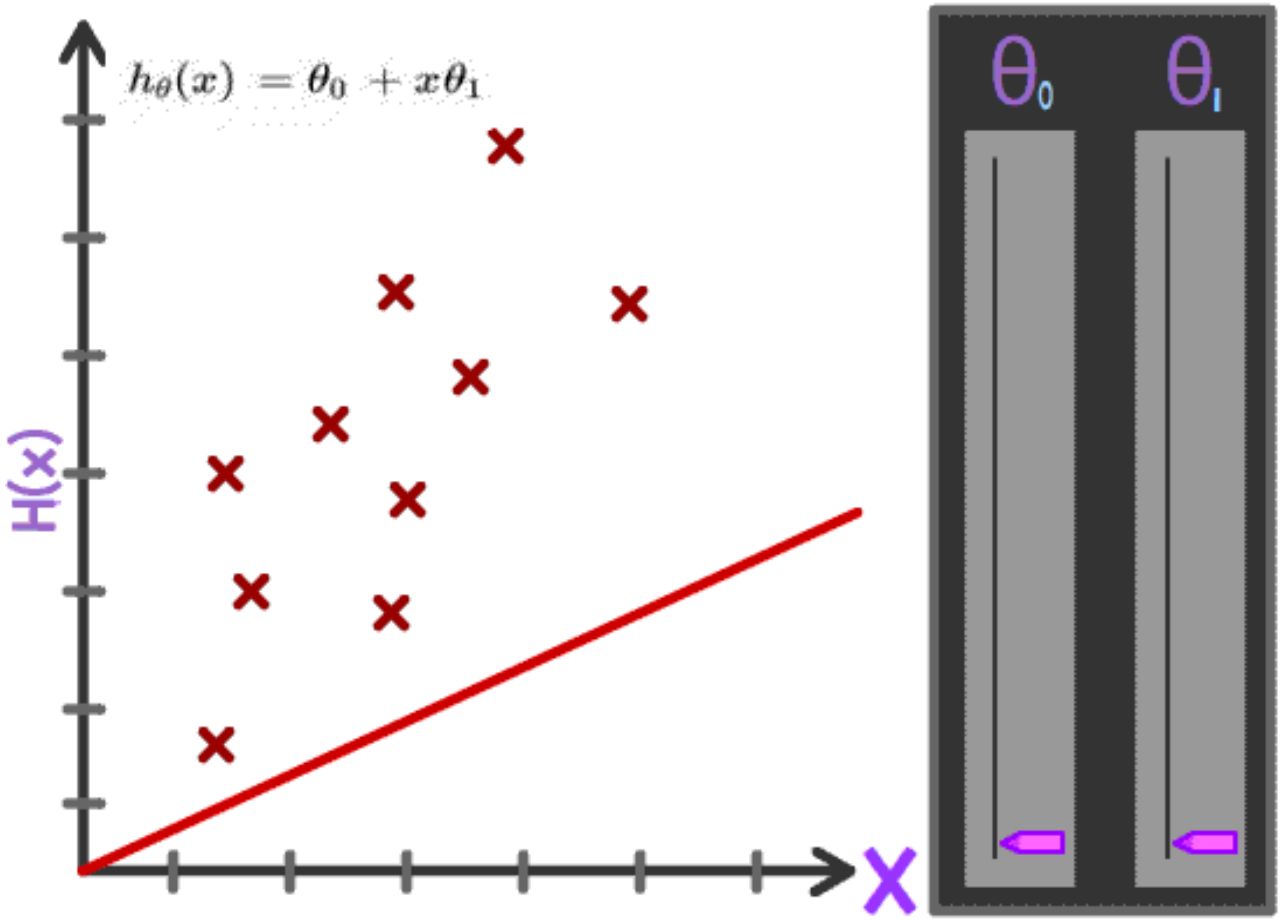
- Some clarification about the notation we will use for this course

$$X_i^{j,[k]}$$

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

Solution Strategy for Solving the Problem

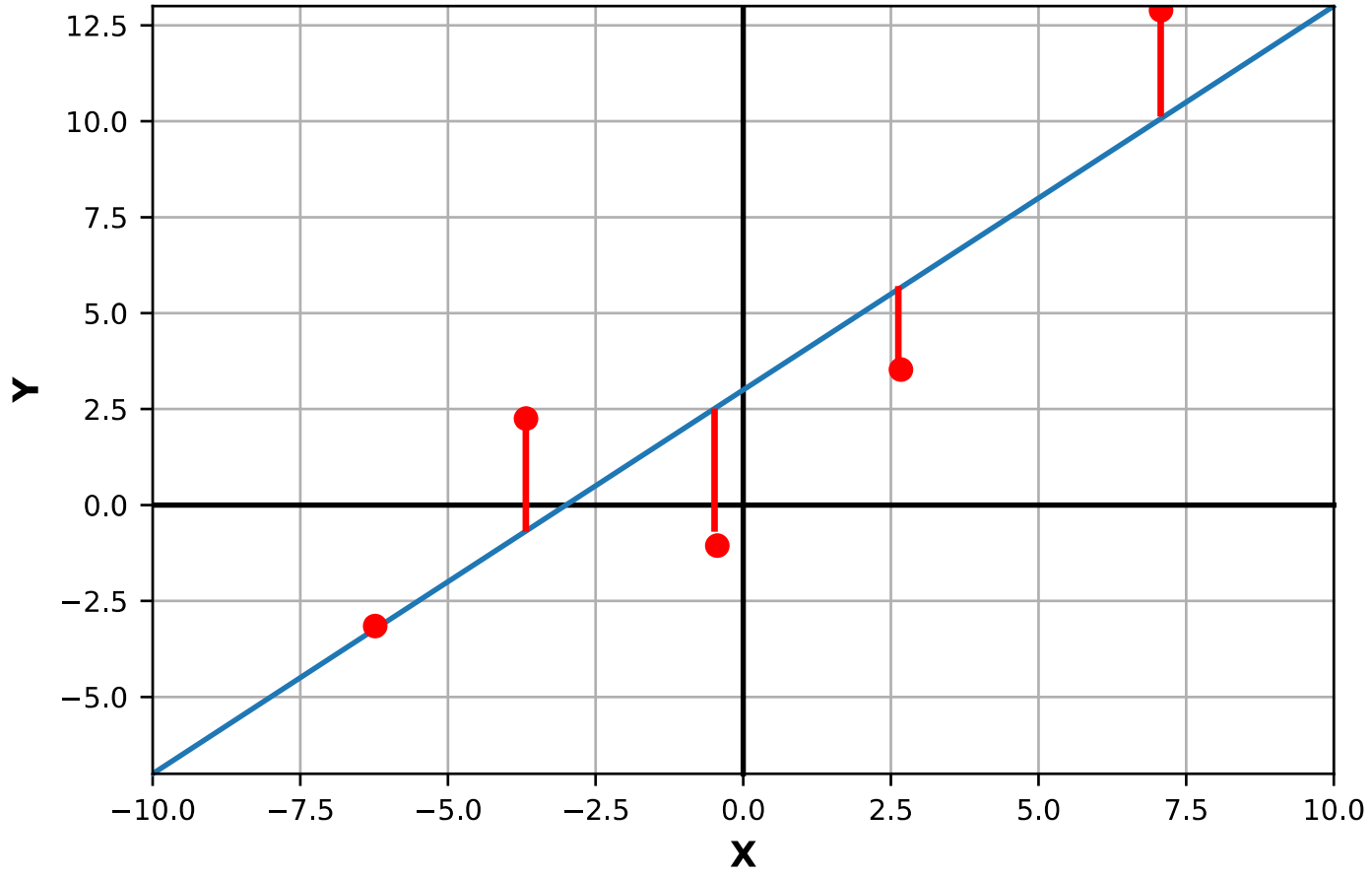
- There are countless possible lines.
- We want a line which is in some sense the “average line” that represents the data.
- Any ideas as to how we can do it?



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

Optimization

- To find the "best line," we should minimize the distances between our line’s predictions and all the data points.
- How to define that mathematically?



Loss Function

- For one sample, this can be represented mathematically by:

$$(y - \hat{y}) \qquad \text{(Error)}$$

- But this could result in negative value if $\hat{y} > y$. Let's square it to remove the negative sign:

$$(y - \hat{y})^2 \qquad \text{(Squared Error)}$$

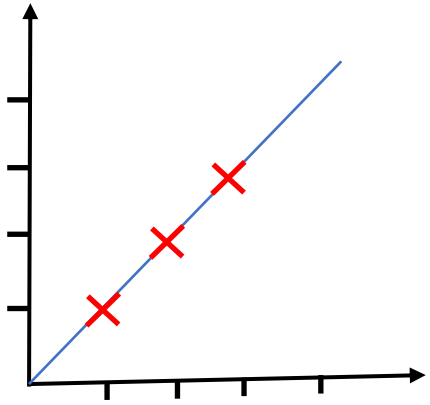
- But we have N samples, not only one. So, let's sum the errors and take the average:

$$Loss\ (MSE) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad \text{(Mean Squared Error)}$$

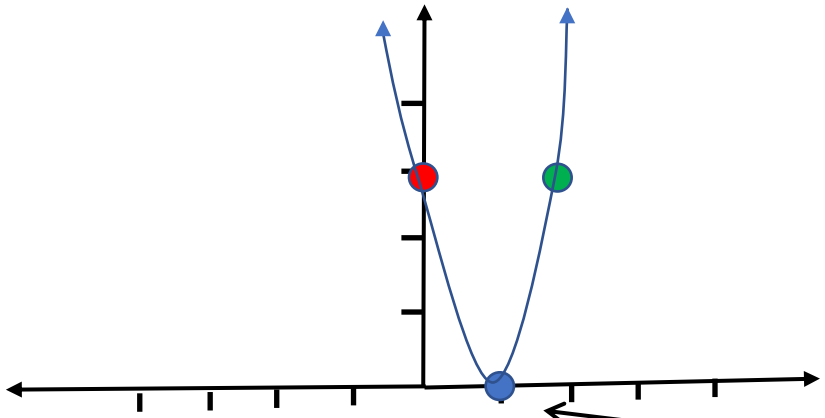
Intuition of Loss Function

$$h(x) = mx$$

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



Hypothesis



Loss Function

Notice: Lower is better.

$J(m = 0) = 14$

$J(m = 1) = 0$

$J(m = 2) = 14$

How to find minima of a function (Review):

- There are two approaches to find the minima:

  - **Exact (Closed-form)**: Directly calculates the solution mathematically by solving for $f'(x) = 0$.

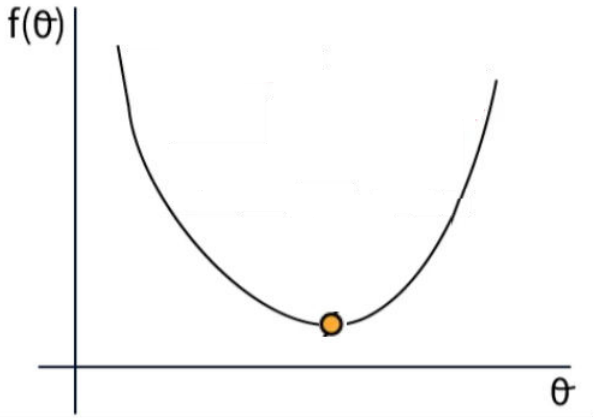
    Important Note: can be used only with a very limited number of algorithms.

  - **Approximation (Iterative approach):** Gradually improves the solution step by step.

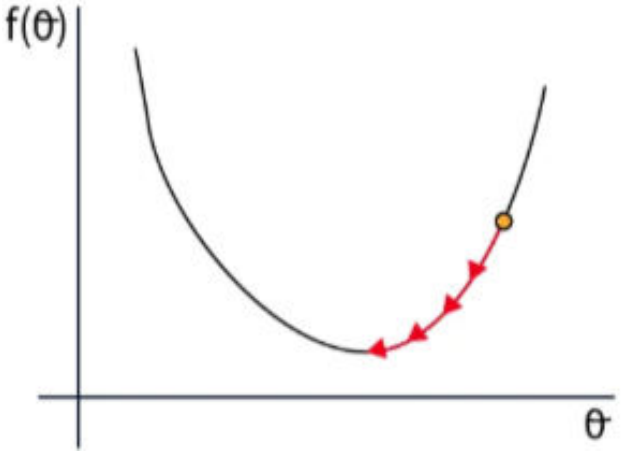
    Done by optimizers (e.g. Gradient Descent, ADAM,…etc).

How to find minima of a function (Review):

Closed-form:



Iterative:



- Example: $y = x^2$ (Solution: $x = 0$)
  - Closed-form Final Result: $x = 0$
  - Iterative Final Result: $x = 0.00001$ (close enough)

How to find minima of a function (Review):

- Let's try to solve this using the closed-form here:

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

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

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

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. m, b

Gradient of the loss 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 loss function

- Simplifying the above expressions, we get:

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

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

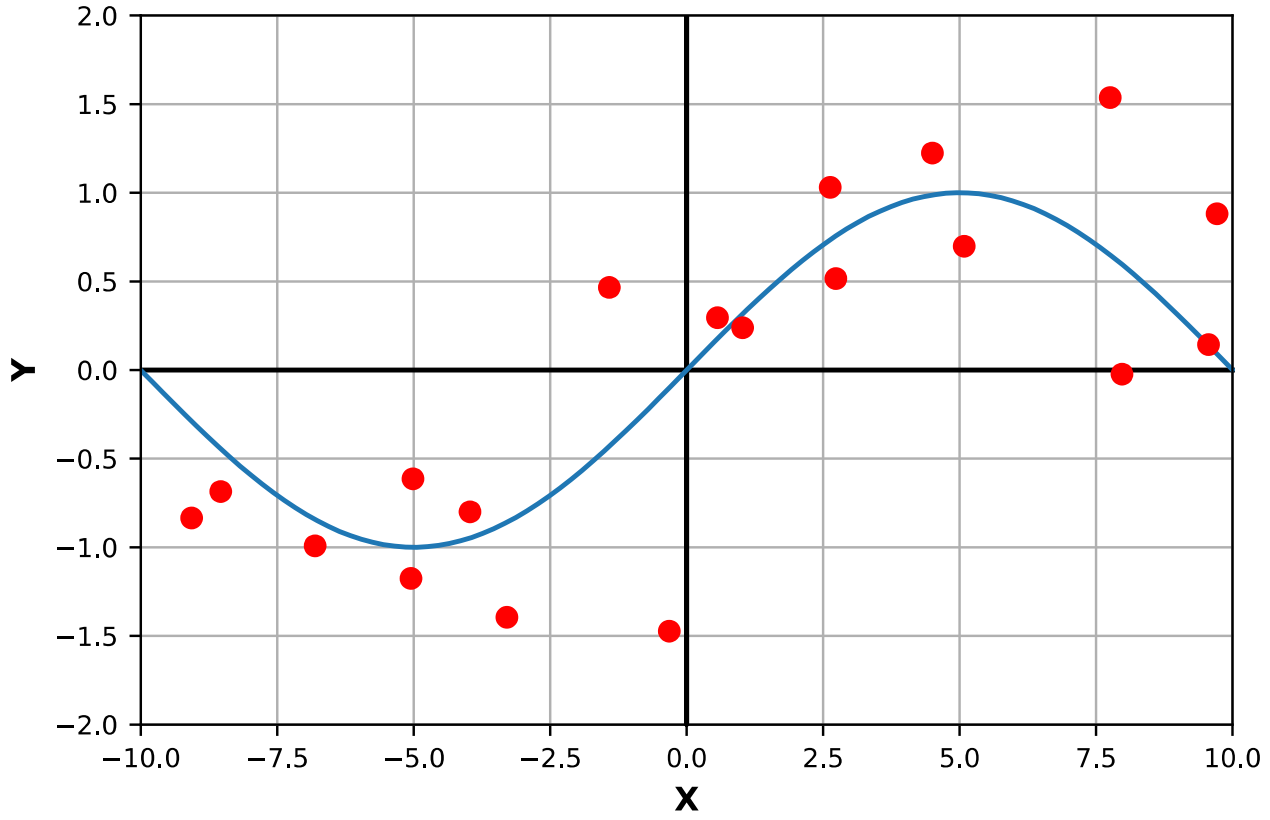
Gradient of the loss function

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

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

Fitting Non-linear Data

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



Transforming the Feature Space (Feature Engineering)

- We can transform features x_i

$$x_i = (x_i^1, x_i^2, x_i^3, ..., x_i^m)$$

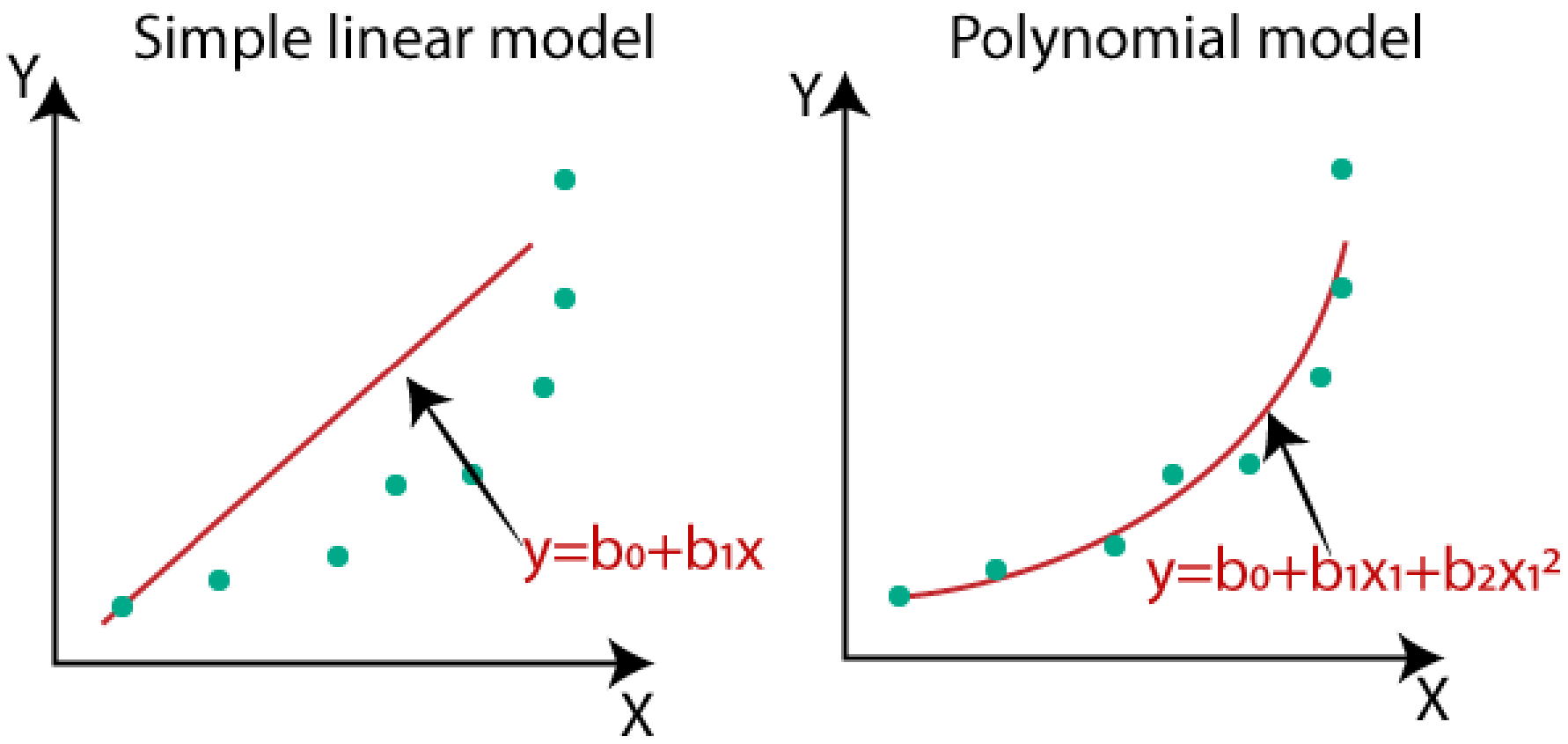
- We will apply some non-linear transformation ϕ:

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

- For example, Polynomial transformation:

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

Transforming the Feature Space (Feature Engineering)



Transforming the Feature Space (Feature Engineering)

Example: assume you have:

$$x_1: Length$$
$$x_2: Width$$

You can add $x_3: Area = x_1 * x_2$ to the dataset.

Other types:

- Cosine, splines, radial basis functions, etc.
- Encoding (Label encoding, One-hot,…)
- Domain-related features (e.g. financial measures)
- Time-related features (Day, month, year,…)

Gradient of the loss function

- Let’s get back to the gradients…

Issues with the Approach

- Assume we have 100 variables instead of 2.
- Calcuting gradients like this can quickly become tedious
- **Notice**: 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** (Writing equations as matrices).

Vectorization

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

$$\hat{y}_i = w_0 + w_1 x_i^1 + w_2 x_i^2 + \cdots + w_M x_i^M$$

- Where w_j are the unknown weights of the data x^j features of the input
- Next, we denote the discrepency between y_i and $\hat{y}_i$ as ϵ_i

$$y_i = \hat{y}_i + \epsilon_i$$

Vectorization

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

$$y_1 = \hat{y}_1 + \epsilon_1$$
$$y_2 = \hat{y}_2 + \epsilon_2$$
$$\vdots$$
$$y_N = \hat{y}_N + \epsilon_N$$

Vectorization

- Replacing the values of $\hat{y}$, we get:

$$y_1 = w_0 + w_1 x_1^1 + w_2 x_1^2 + ... + w_M x_1^M + \epsilon_1$$
$$y_2 = w_0 + w_1 x_2^1 + w_2 x_2^2 + ... + w_M x_2^M + \epsilon_2$$
$$\vdots$$
$$y_N = w_0 + w_1 x_N^1 + w_2 x_N^2 + ... + w_M x_N^M + \epsilon_N$$

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:

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

Vectorization

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

- Let's formalize some notaitons:

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

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

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 $\boldsymbol{\theta}$:

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

Closed-form solution for Linear Regression

ML Algorithms Perspectives:

- We can look into ML algorithms from two perspective:
  - **Loss Minimization** Problem (like what we did).
  - **Probability Maximization** Problem (using Maximum Likelihood Estimation).
- Both should result in the same solution.

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 dataset 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

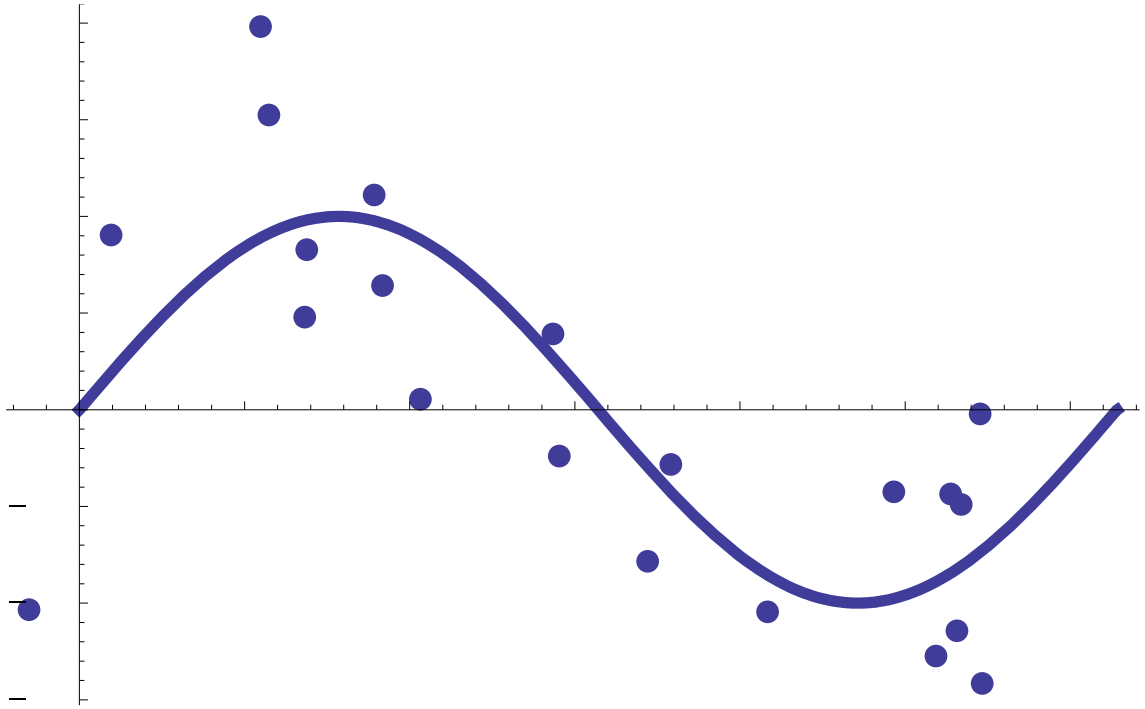
- Furthermore, since the right hand side of the above equation is monotonice in \theta the arg max will not change if we take log of the expression

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

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

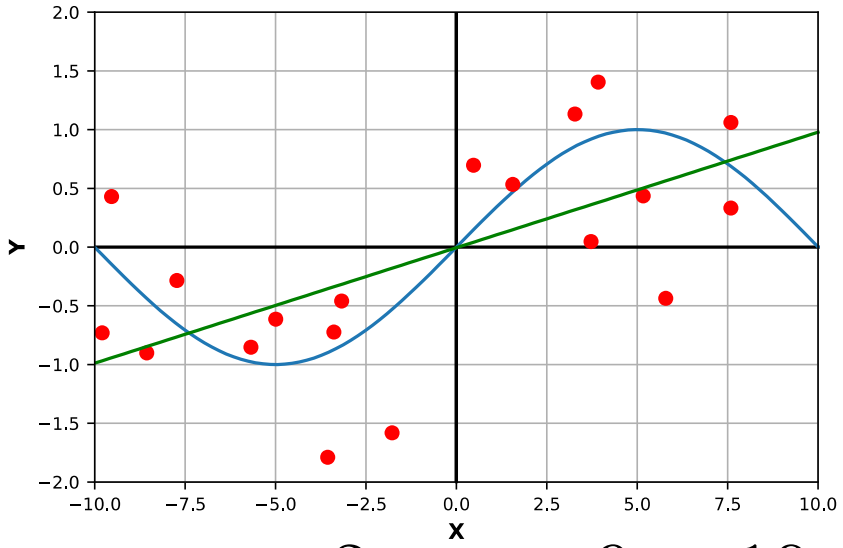
Bias and Variance

- What if Y has a non-linear response?

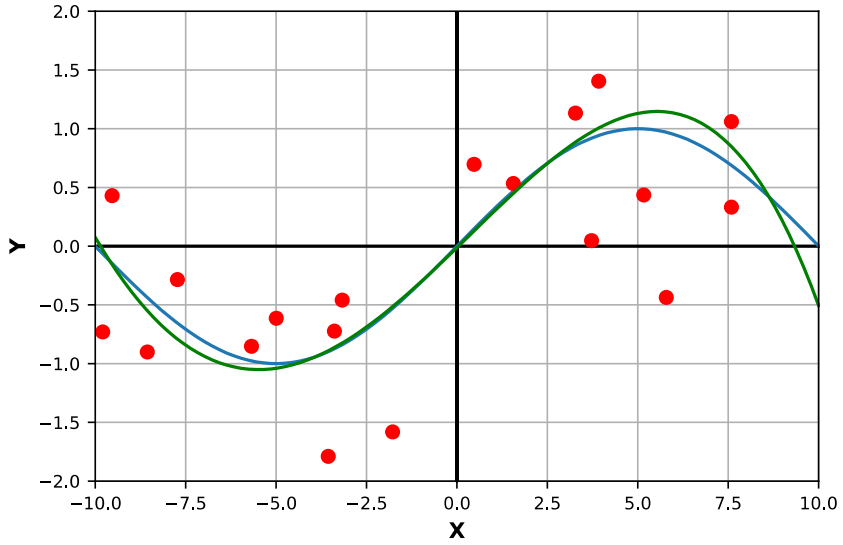


- Can we still use a linear model?

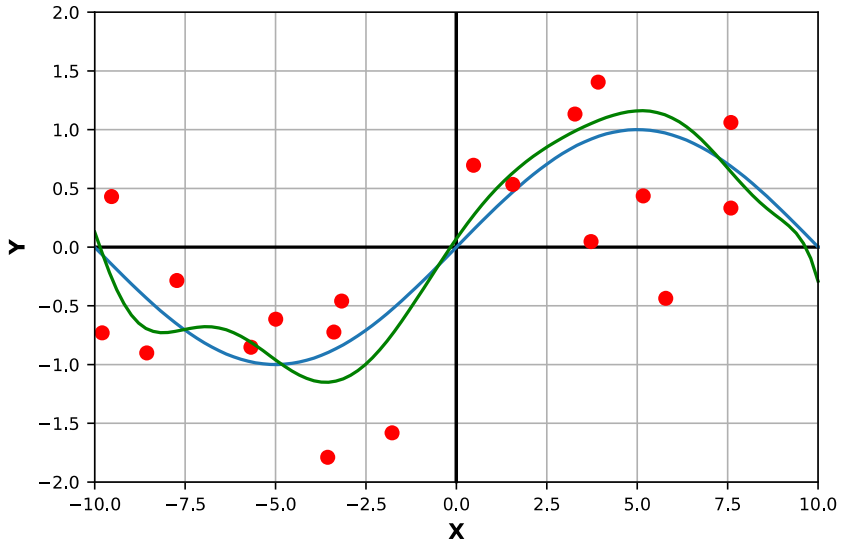
$\{1, x\}$



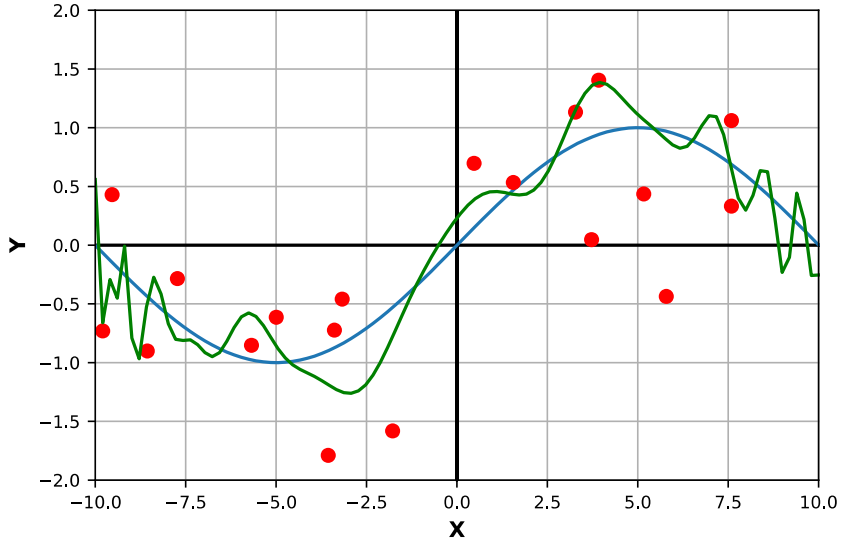
$\{1, x, x^2, x^3, x^4\}$



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

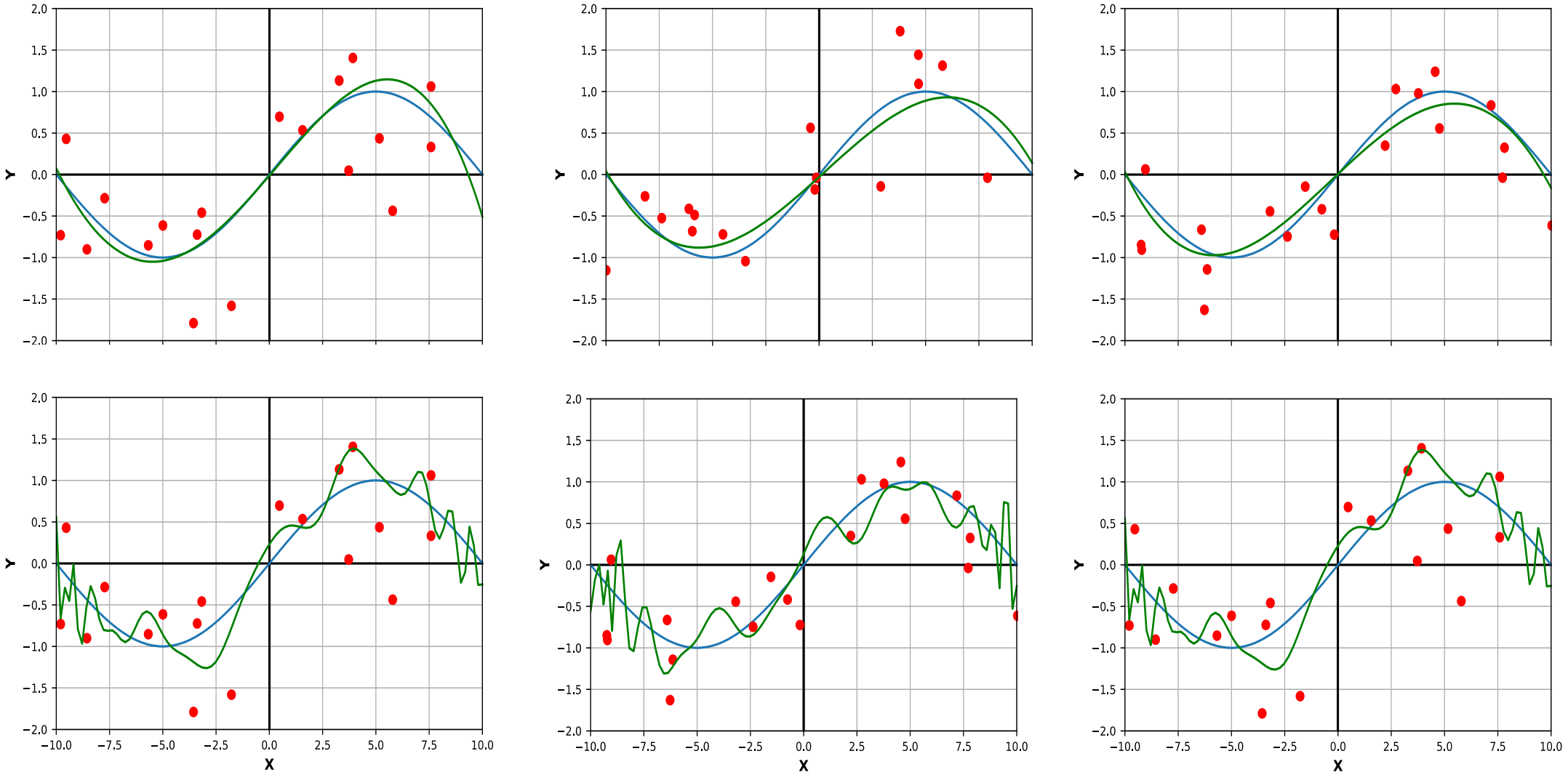


$\{1, x, x^2, ..., x^{99}, x^{100}\}$

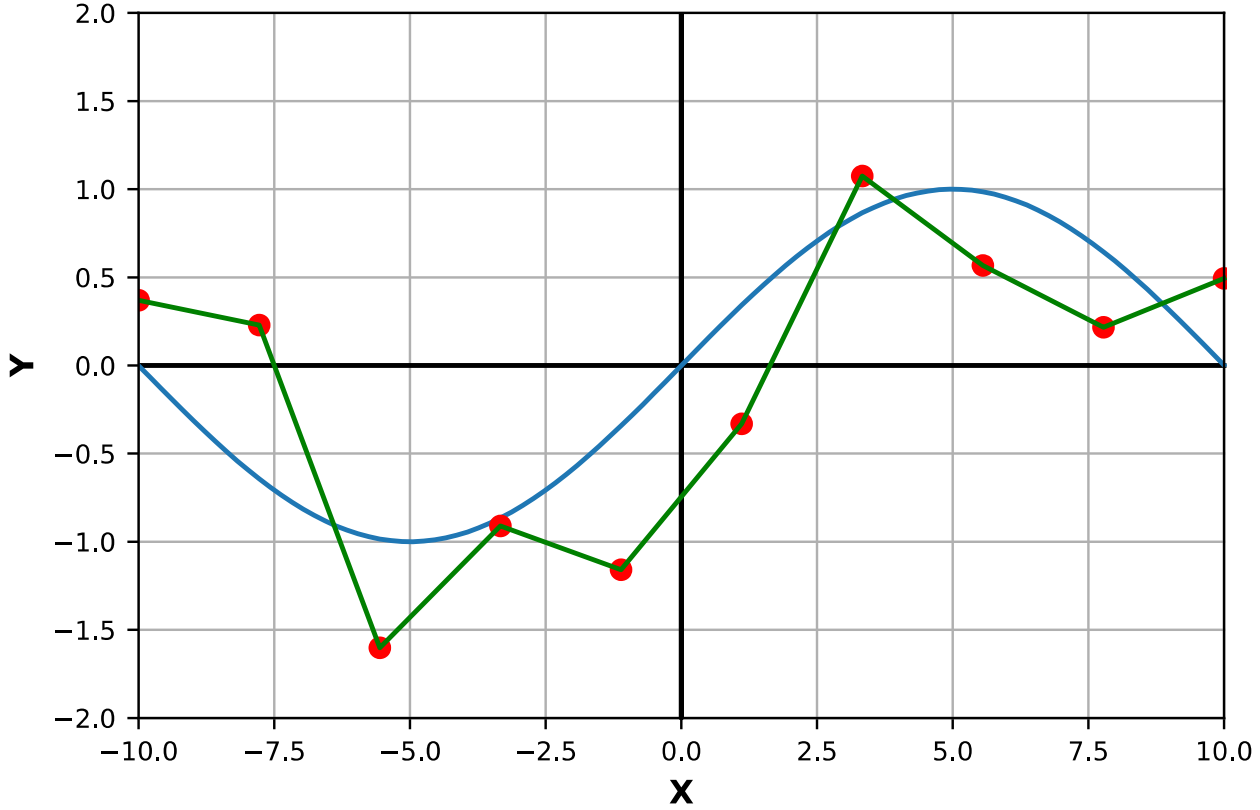


What is Bias and Variance?

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



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 Plot

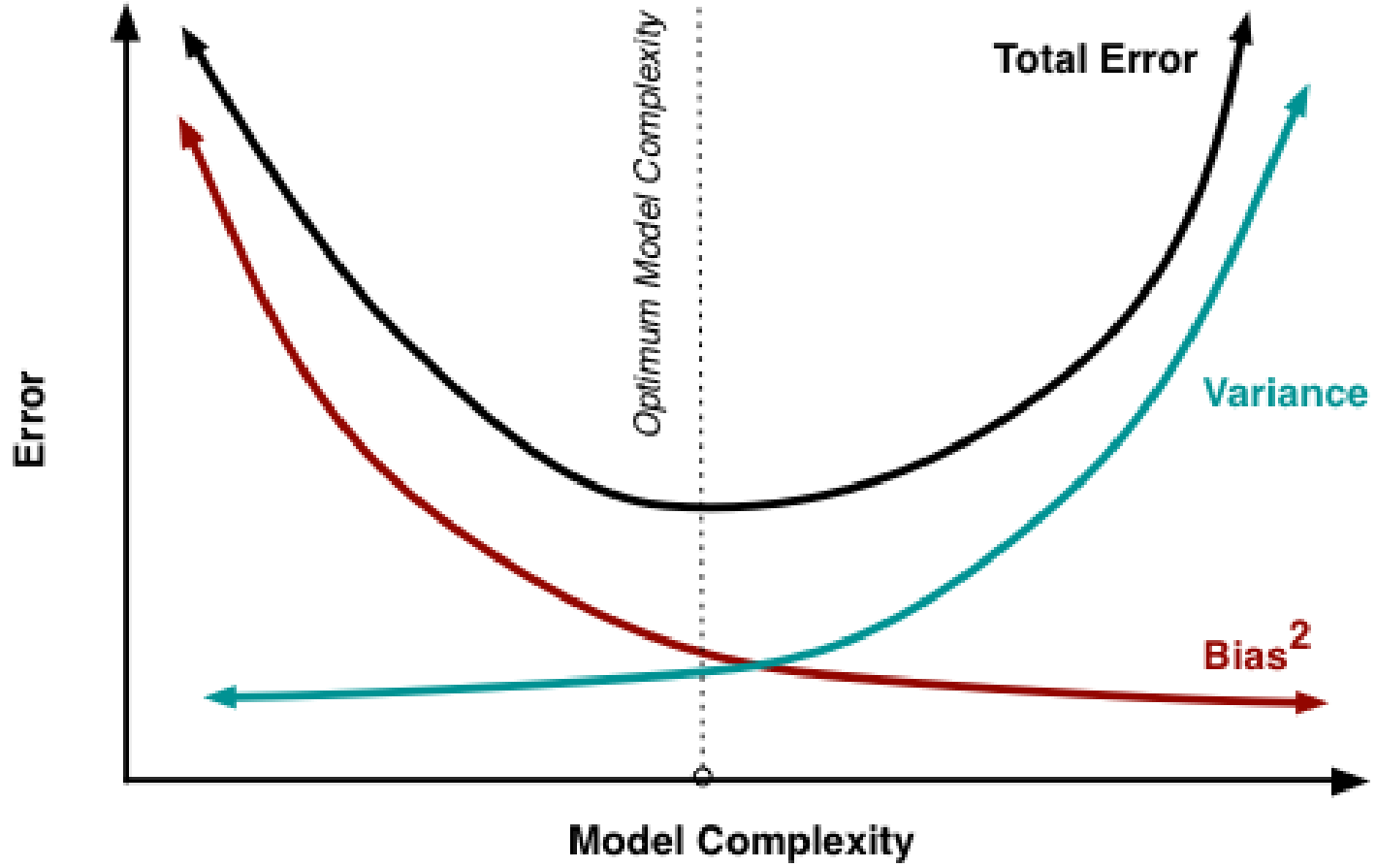


Image from http://scott.fortmann-roe.com/docs/BiasVariance.html

Data Split

- To ensure your model doesn’t overfit to the training data, you should have another subset called **testing data.**
- You will evaluate your model against this subset, and based on its **metric score (e.g. accuracy)** you will decide if it’s overfitting or not.

- But how should I split my data?

Data Split

- **Hold-out set**:
  - A portion of the dataset set aside and not used during training.
  - E.g. 80% for training and 20% for testing.

Issues:

- Imagine you have these labels: [1, 1, 2, 2, 2, 3, 3, 3, 3, 3] and you took last 30% as test: [3,3,3]. You didn't include 1 and 2 in test!

  Solution: Always shuffle before split: [3, 2, 3, 1, 3, 2, 3, 1, 3, 2] ➔ test: [1,3,2]
- My dataset is small. Taking 20% as test would not be representative!

  Solution: Use KFold.

Data Split

- **K-Fold Cross Validation (CV)**:
  - Split data into k parts (folds), trains on $k-1$ folds, test on the remaining fold, and repeats k times then average the scores.

Regularization

Slides from: https://harvard-iacs.github.io/2019-CS109A/lectures/lecture8/presentation/Lecture8a_Regularization.pptx

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 λ 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 $\boldsymbol{l_1}$ 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 β,

- 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 β_{ridge} and β_{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 Set

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

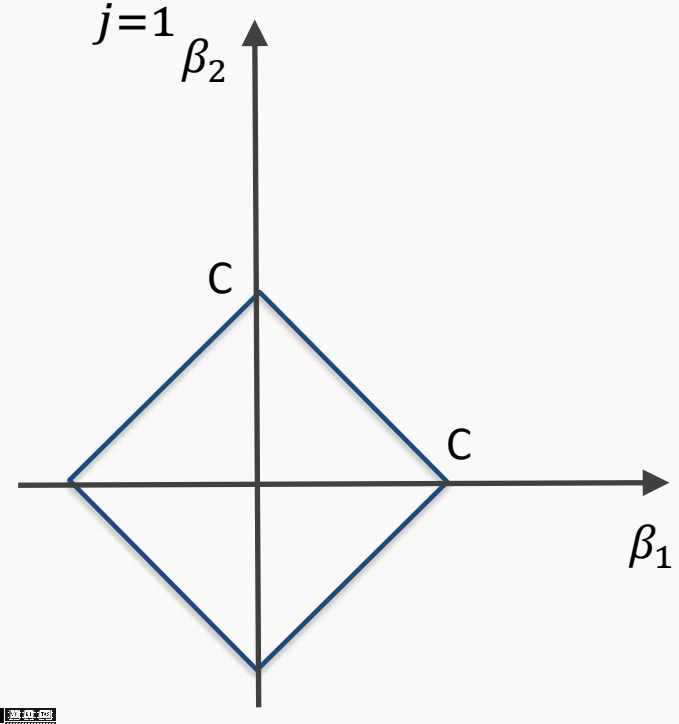
- Select λ
- Find the minimum of the ridge/Lasso regression loss function (using the formula for ridge) and record the *MSE* **on the validation/test set**.
- Find the λ that gives the smallest *MSE*

The Geometry of Regularization (LASSO)

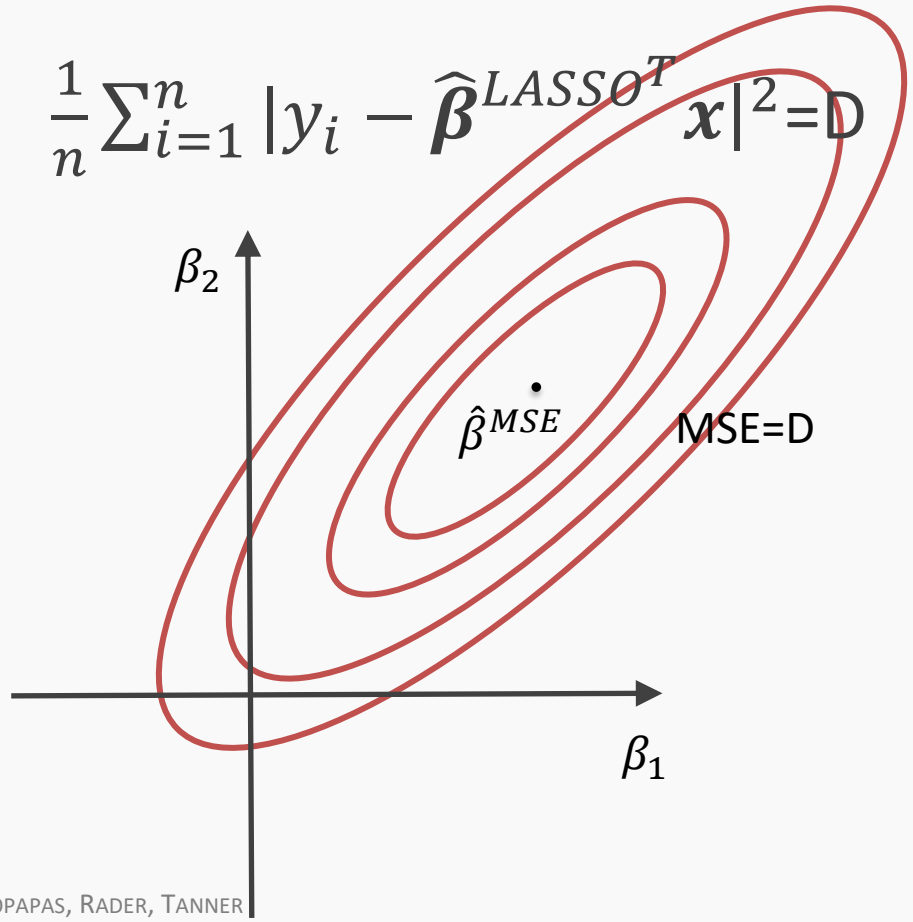
$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}|^2 + \lambda \sum_{j=1}^{J} |\beta_j|$$

$$\widehat{\boldsymbol{\beta}}^{LASSO} = \text{argmin}\, L_{LASSO}(\boldsymbol{\beta})$$

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



$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{\boldsymbol{\beta}}^{LASSO^T} \boldsymbol{x}|^2 = D$$

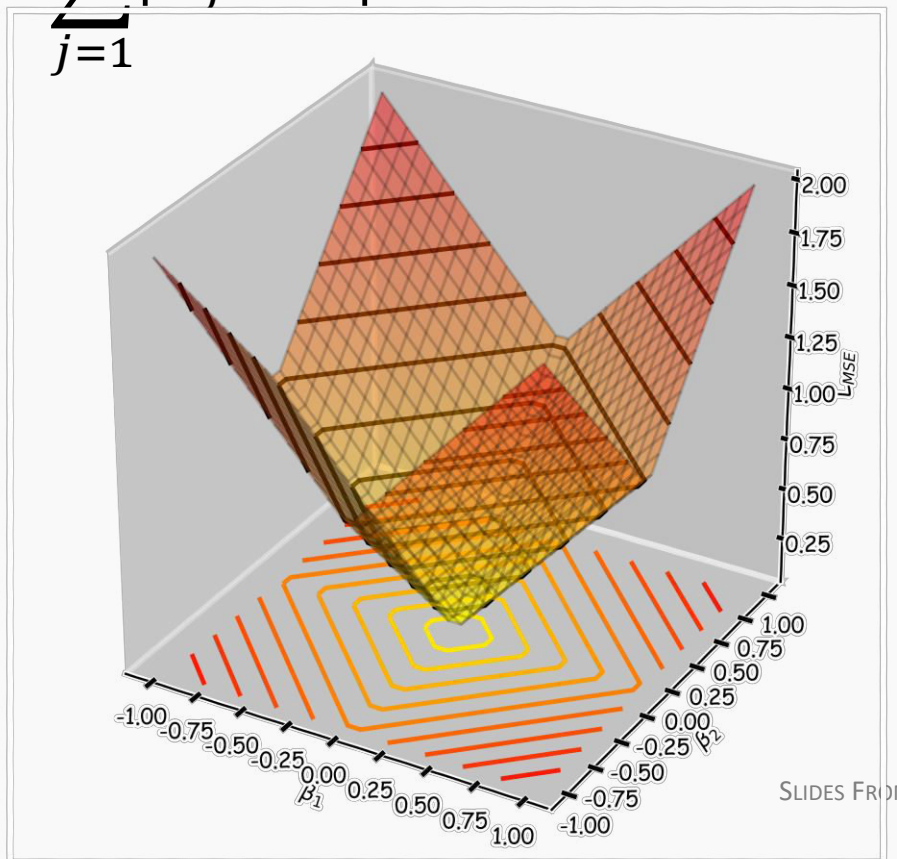


The Geometry of Regularization (LASSO)

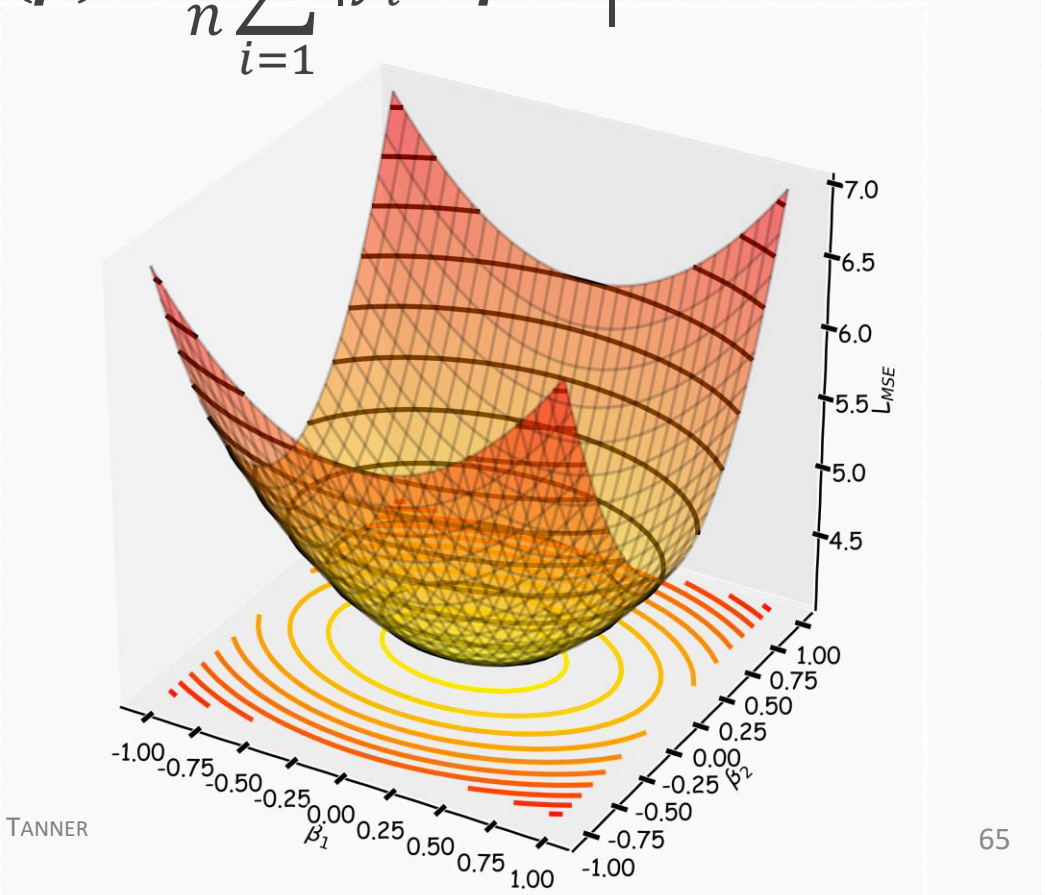
$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2 + \lambda \sum_{j=1}^{J} |\beta_j|$$

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

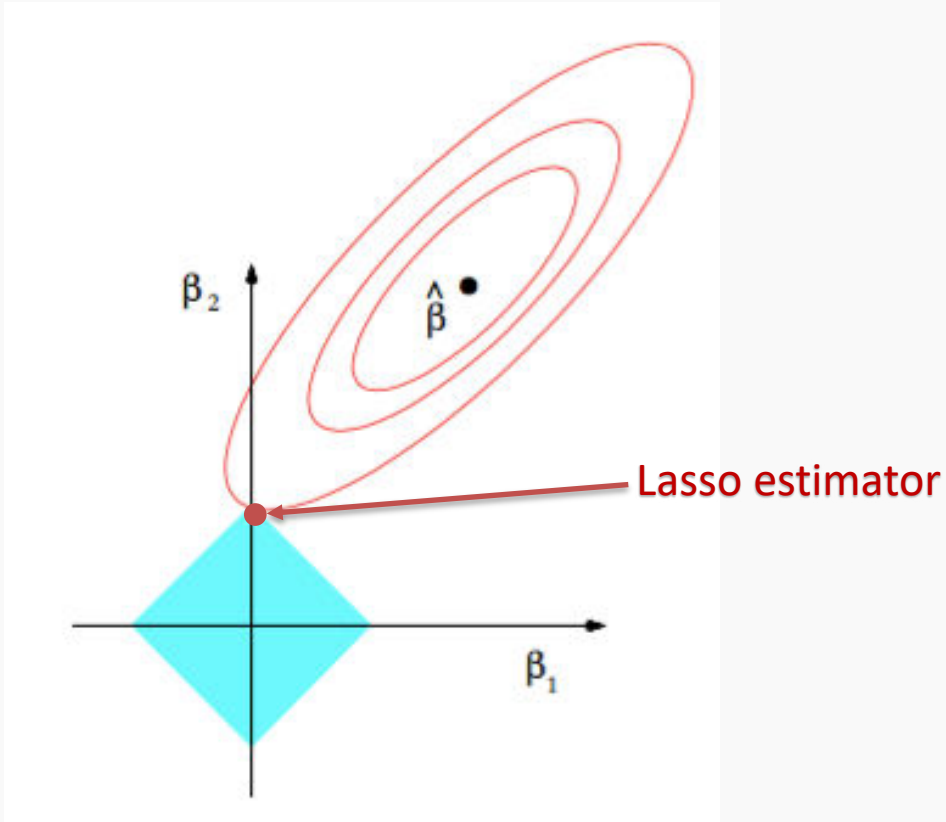
$$L_1 = \lambda \sum_{j=1}^{J} |\hat{\beta}_j^{LASSO}|$$

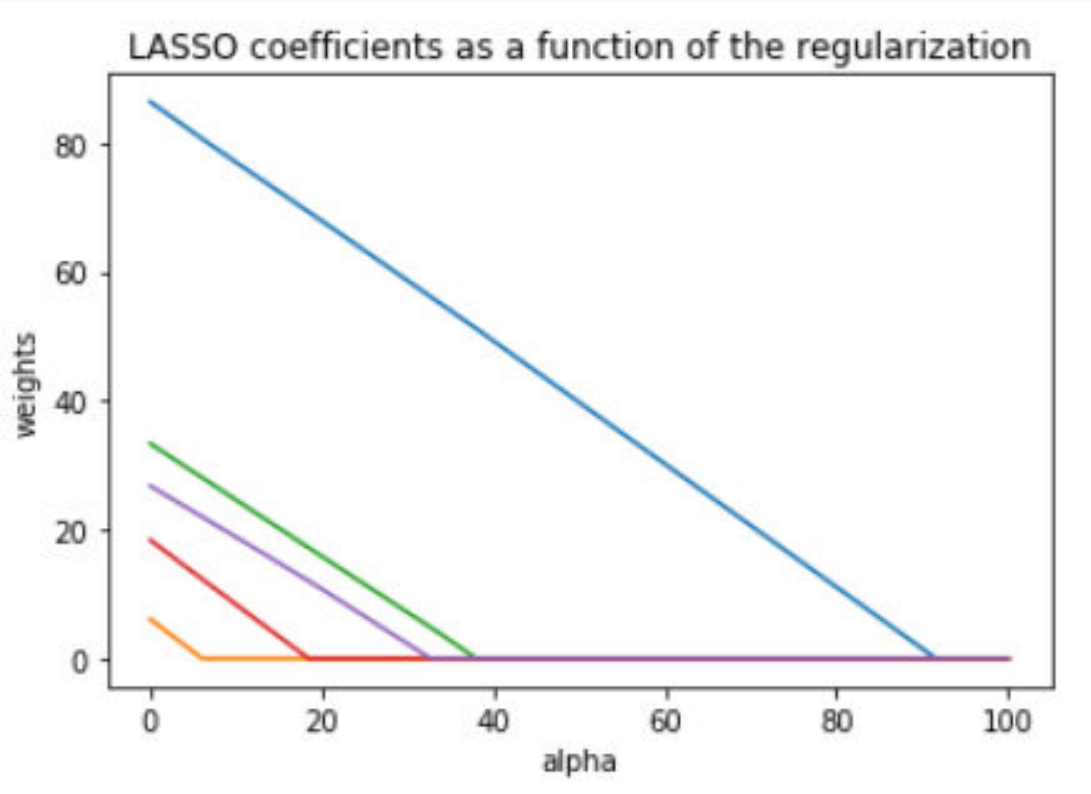


$$L_{MSE}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}\Big|^2$$



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.

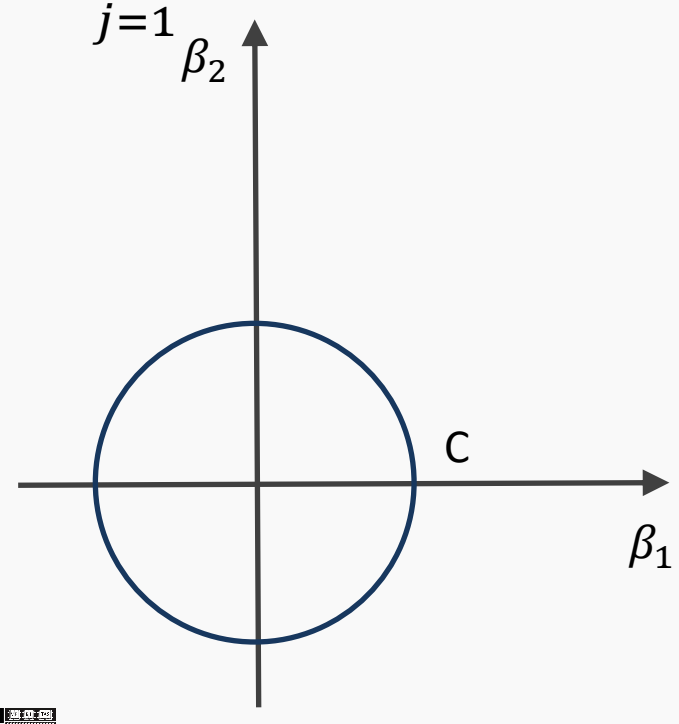
The values of the coefficients decrease as lambda increases, and are nullified fast.

The Geometry of Regularization (Ridge)

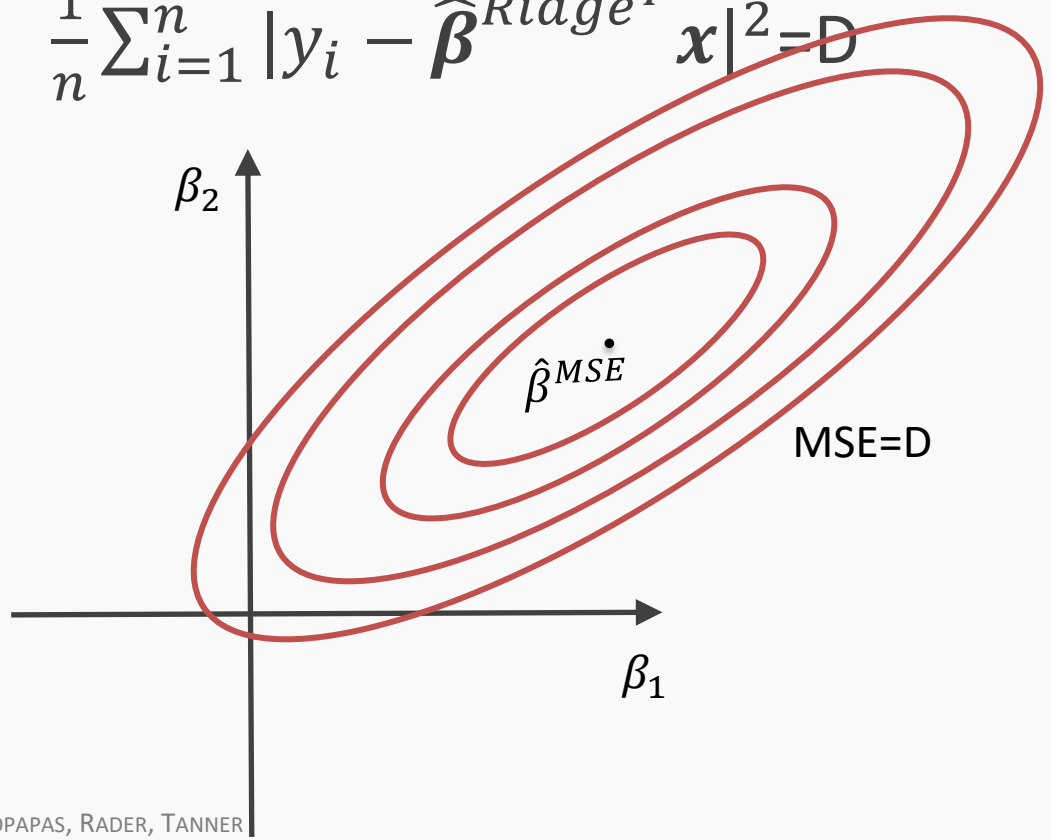
$$L_{Ridge}(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} |y_i - \boldsymbol{\beta}^T \boldsymbol{x}|^2 + \lambda \sum_{j=1}^{J} (\beta_j)^2$$

$$\widehat{\boldsymbol{\beta}}^{Ridge} = \operatorname{argmin} L_{Ridge}(\boldsymbol{\beta})$$

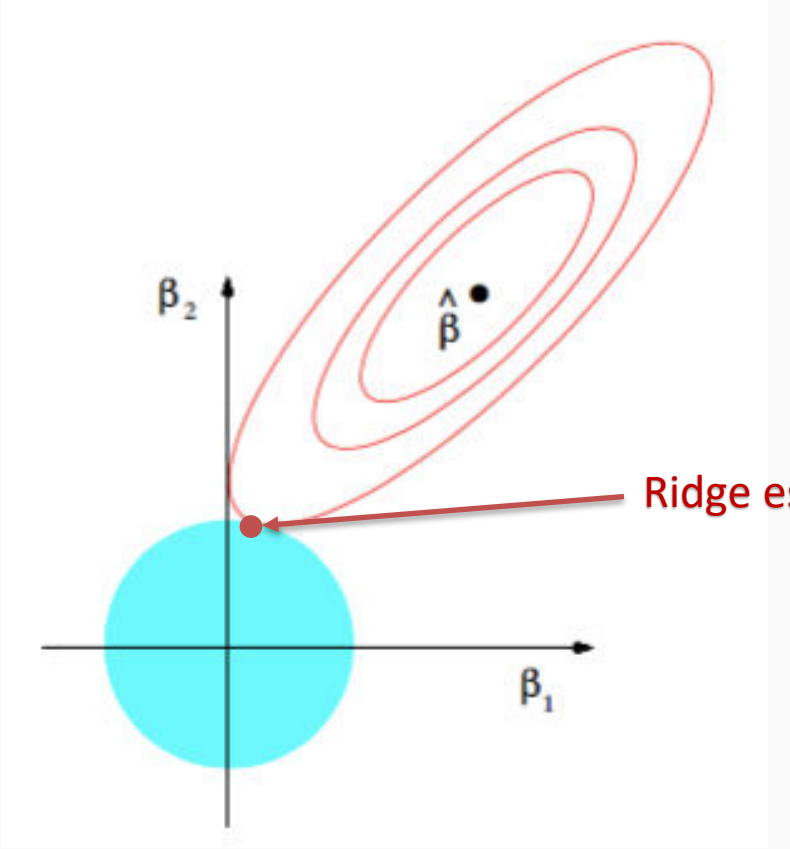
$$\lambda \sum_{j=1}^{J} \left|\hat{\beta}_j^{Ridge}\right|^2 = C$$



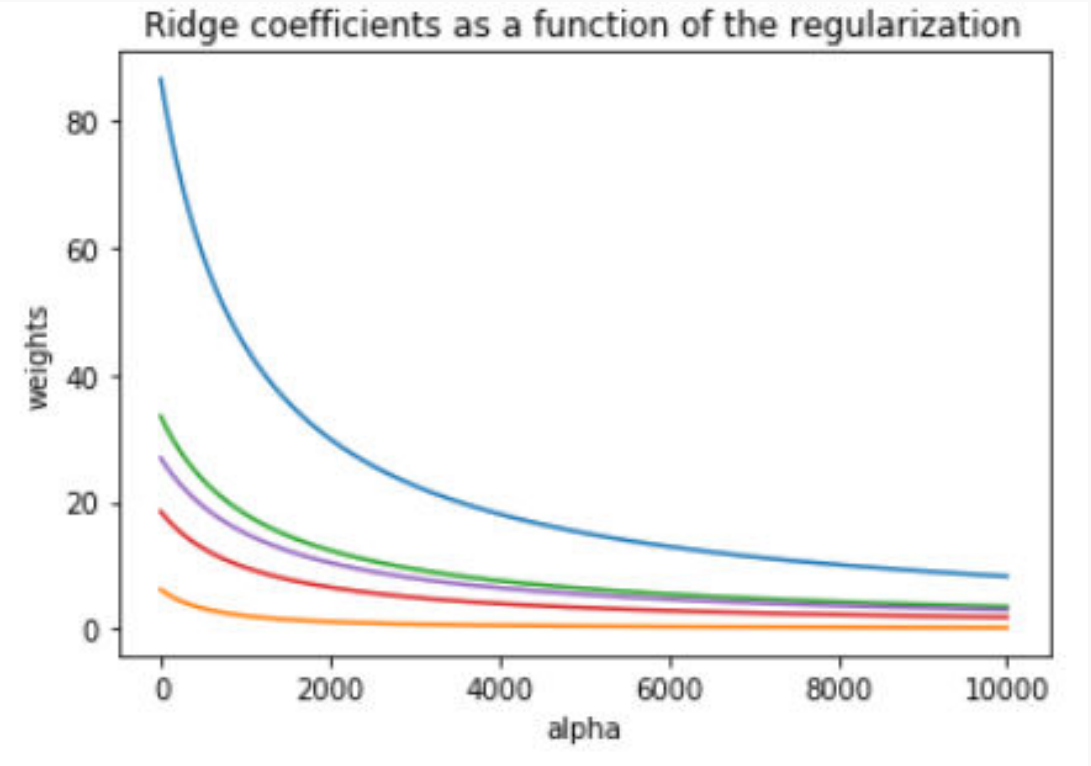
$$\frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{\boldsymbol{\beta}}^{Ridge^T} \boldsymbol{x}|^2 = D$$



Ridge visualized

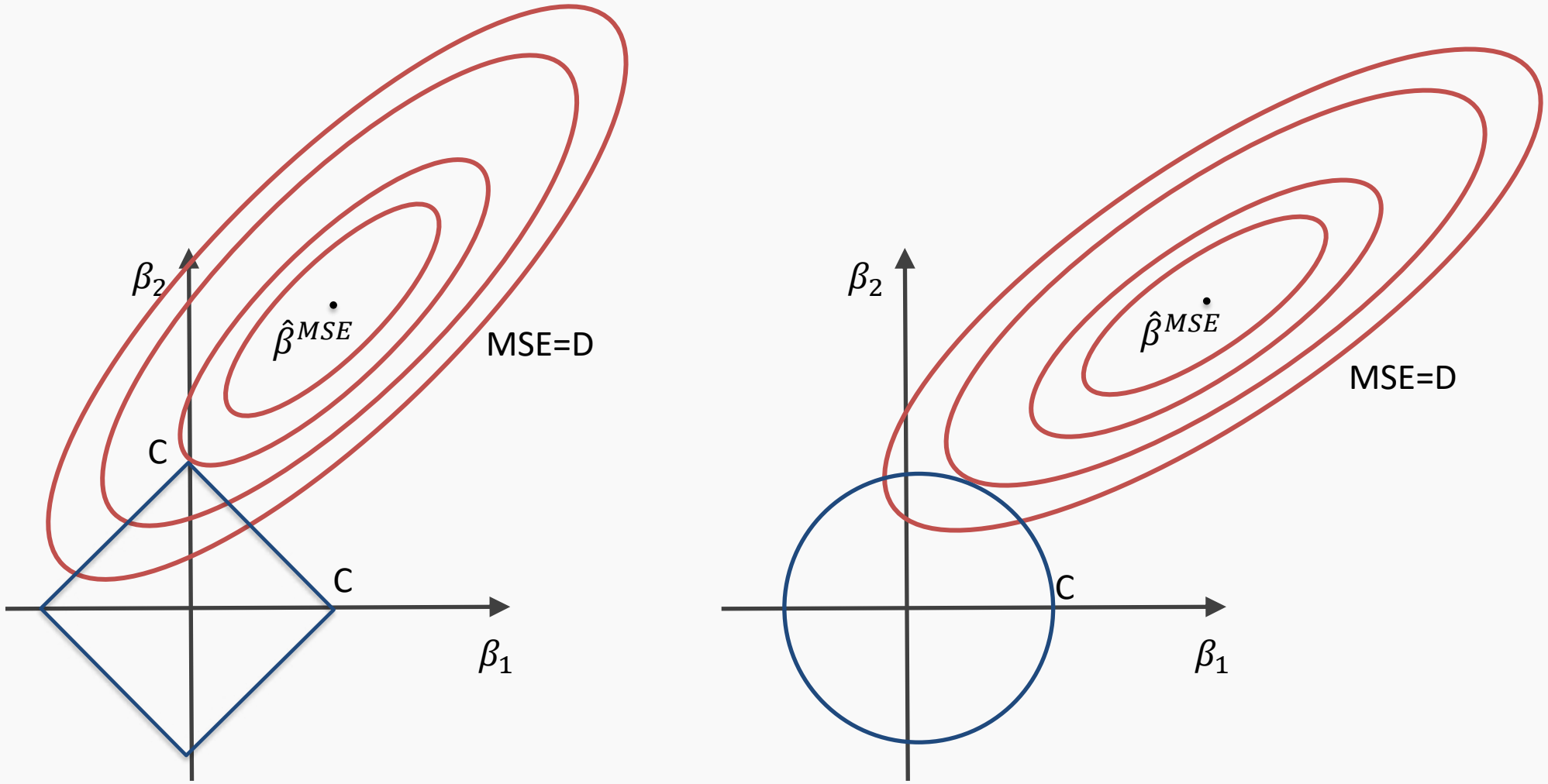


The ridge estimator is where the constraint and the loss intersect.



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

The Geometry of Regularization



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

1. split data into $\{\{X,Y\}_{train},\{X,Y\}_{validation},\{X,Y\}_{test}\}$
2. for λ in $\{\lambda_{min},\dots\lambda_{max}\}$:
    1. determine the β that minimizes the L_{ridge}, $\hat{\beta}_{Ridge}(\lambda) = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}+\lambda I\right)^{-1}X^{T}Y$, using the train data.
    2. record $L_{MSE}(\lambda)$ using validation data.
3. select the λ that minimizes the loss on the validation data, $\lambda_{ridge} = \mathrm{argmin}_{\lambda}\, L_{MSE}(\lambda)$
4. Refit the model using both train and validation data, $\{\{X,Y\}_{train},\{X,Y\}_{validation}\}$, resulting to $\hat{\beta}_{ridge}(\lambda_{ridge})$
5. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{ridge}(\lambda_{ridge})$

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

1. split data into $\{\{X,Y\}_{train}, \{X,Y\}_{validation}, \{X,Y\}_{test}\}$
2. for λ in $\{\lambda_{min}, \ldots \lambda_{max}\}$:
   A. determine the β that minimizes the L_{lasso}, $\hat{\beta}_{lasso}(\lambda)$, using the train data. **This is done using a solver.**
   B. record $L_{MSE}(\lambda)$ using validation data
3. select the λ that minimizes the loss on the validation data,
   $\lambda_{lasso} = \text{argmin}_{\lambda}\, L_{MSE}(\lambda)$
4. Refit the model using both train and validation data, $\{\{X,Y\}_{train}, \{X,Y\}_{validation}\}$, resulting to $\hat{\beta}_{lasso}(\lambda_{lasso})$
5. report MSE or R^2 on $\{X,Y\}_{test}$ given the $\hat{\beta}_{lasso}(\lambda_{lasso})$

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.          ]^T . x
```

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

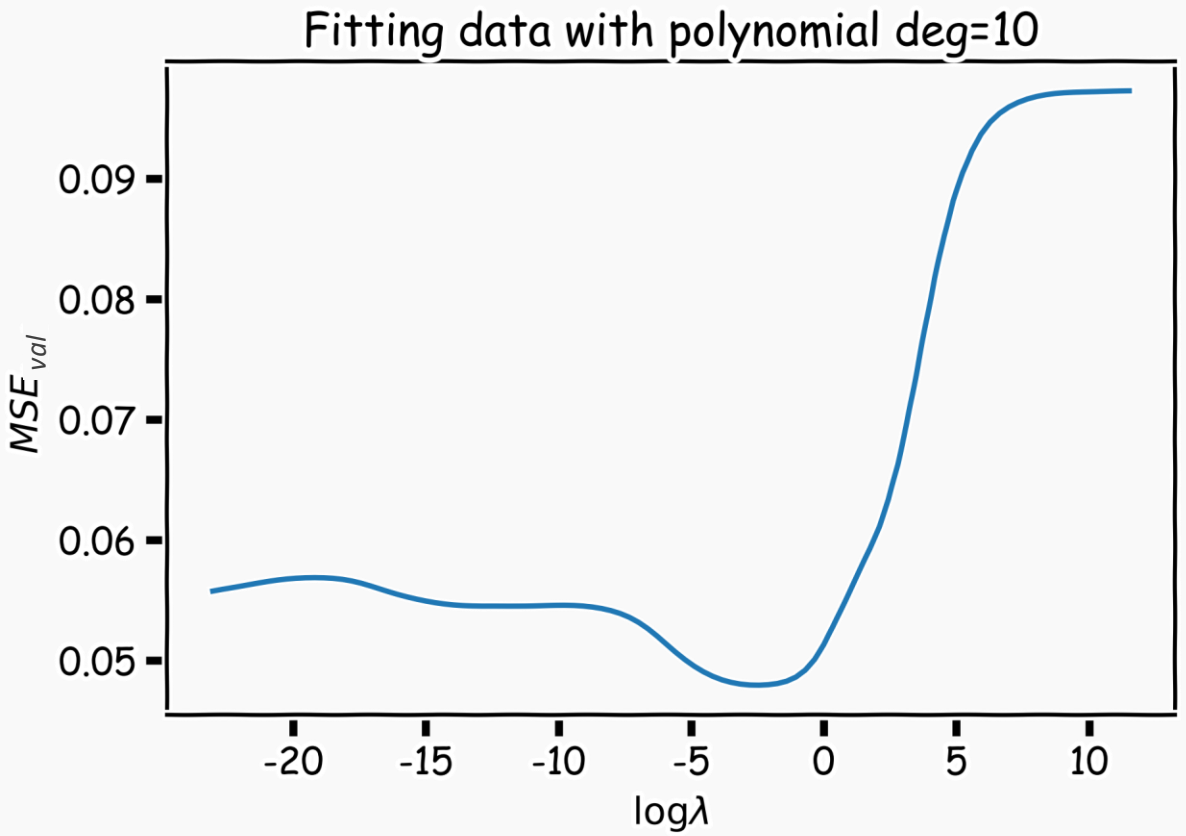
In [20]:
```
X_train = train[all_predictors].values
X_val = validation[all_predictors].values
X_test = test[all_predictors].values

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



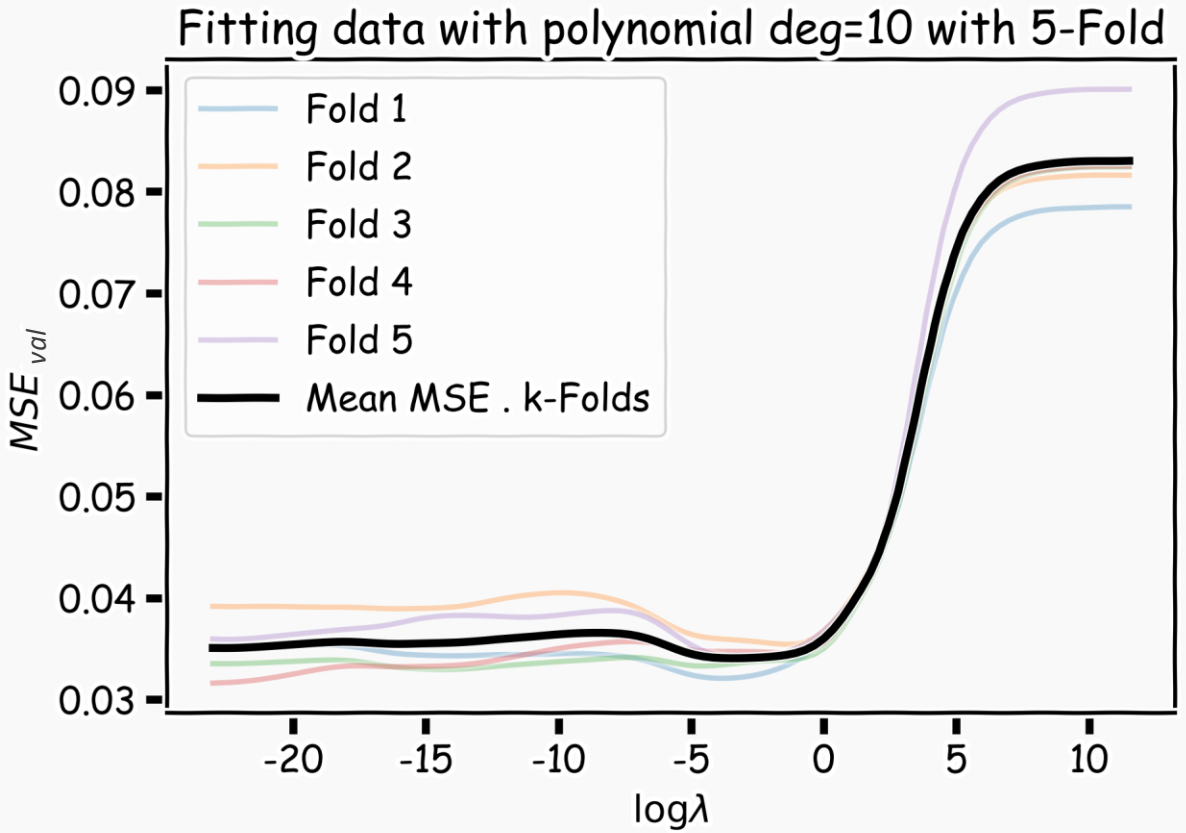
Ridge regularization with **CV**: step by step

	λ_1	λ_2	...	λ_n
k_1	L_{11}	L_{12}	..	...
k_2	L_{21}	...	..	...
...	..	...	..	...
k_n	...	...	...	...
E[]	$\bar{L}_1$	$\bar{L}_2$	...	$\bar{L}_n$

1. remove $\{X, Y\}_{test}$ from data
2. split the rest of data into K folds, $\{\{X, Y\}_{train}^{-k}, \{X, Y\}_{val}^{k}\}$
3. for k in $\{1, \dots, K\}$
    1. for λ in $\{\lambda_0, \dots, \lambda_n\}$:
        A. determine the β that minimizes the L_{ridge}, $\hat{\beta}_{ridge}(\lambda, k) = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X} + \lambda I\right)^{-1} X^T Y$, using the train data of the fold, $\{X, Y\}_{train}^{-k}$.
        B. record $L_{MSE}(\lambda, k)$ using the validation data of the fold $\{X, Y\}_{val}^{k}$

        At this point we have a 2-D matrix, rows are for different k, and columns are for different λ values.
4. Average the $L_{MSE}(\lambda, k)$ for each λ, $\bar{L}_{MSE}(\lambda)$.
5. Find the λ that minimizes the $\bar{L}_{MSE}(\lambda)$, resulting to λ_{ridge}.
6. Refit the model using the full training data, $\{\{X, Y\}_{train}, \{X, Y\}_{val}\}$, resulting to $\hat{\beta}_{ridge}(\lambda_{ridge})$
7. report MSE or R^2 on $\{X, Y\}_{test}$ given the $\hat{\beta}_{ridge}(\lambda_{ridge})$

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?