Lecture 4: Linear Regression (Part 3)

CIS 4190/5190
Spring 2023
Administrivia

• Class roster is now stable, and add/drop deadline has passed.

• HW1 due **tonight 8 p.m.**, and HW2 will be posted tonight/tomorrow morning, on linear regression.
  ▪ Class late policy reminder: 0.5% points for every late hour, up to a max of 48 hours.

• TA office hours:
  ▪ Any changes will be posted to the TA office hours thread on EdSTEM, at least 48 hours ahead of time.
  ▪ We are moving the 3 p.m. OH on Monday and Wed to 3.30 p.m. to avoid clashing. More news on that soon.

• Recitations tomorrow at 5 p.m. on Python, Numpy, Pandas, Scikit-Learn. See EdSTEM post.

• No quiz for week 1. We’ll fix the webpage.
Last Lecture Summary

• The Train/Test Split Protocol for Measuring Underfitting / Overfitting

• Bias and variance as functions of a model class
  ▪ Tuning them by selecting hypothesis spaces / feature maps
  ▪ Tuning them by modifying the loss function
    ▪ $L_{\text{new}}(\beta; Z) = L(\beta; Z) + \lambda \cdot R(\beta)$

• Today:
  ▪ Selecting hyperparameters like $\lambda$
  ▪ Finally unveil the mystery about how to find $\hat{\beta}(Z) = \arg \min_\beta L(\beta; Z)$
Cross-Validation for Model Selection
Hyperparameter Tuning, or “Model Selection”

• \( \lambda \) is a **hyperparameter** that must be tuned (satisfies \( \lambda \geq 0 \))

• **Naïve strategy:** Try a few different candidates \( \lambda_t \) and choose the one that minimizes the test loss
Test Data Contamination

• Suppose you have tried 100 different hyperparameter values, that all have the same value of generalization MSE, if evaluated on the full data distribution.

• But the test dataset is only a finite sample of this distribution, so test MSE is a noisy estimate of true generalization MSE. For example

Note how, in selecting based on test MSE, you have “overfit” your hyperparameter choice to your test set!
• $\lambda$ is a **hyperparameter** that must be tuned (satisfies $\lambda \geq 0$)

• **Naïve strategy:** Try a few different candidates $\lambda_t$ and choose the one that minimizes the test loss

• **Problem:** We may overfit the test set!
  ▪ Major problem if we have more hyperparameters

• **Solution:** A new subset of data just for selecting hyperparameters
Train/Val/Test Split Protocol for Model Selection

- **Goal:** Choose best hyperparameter $\lambda$
  - Can also compare different model families, feature maps, etc.
- **Solution:** Optimize $\lambda$ on a **held-out validation data**
  - **Rule of thumb:** 60/20/20 split (usually shuffle before splitting)

---

Given data $Z$
- 1, 2, ..., samples
- $\ldots$, $n-1$, $n$

<table>
<thead>
<tr>
<th>$x_1, y_1$</th>
<th>$x_2, y_2$</th>
<th>$\ldots$</th>
<th>$x_n, y_n$</th>
</tr>
</thead>
</table>

- Training data $Z_{\text{train}}$
- Val data $Z_{\text{val}}$
- Test data $Z_{\text{test}}$
Basic Cross Validation Algorithm: “Holdout”

**Step 1:** Split $Z$ into $Z_{\text{train}}$, $Z_{\text{val}}$, and $Z_{\text{test}}$

| Training data $Z_{\text{train}}$ | Val data $Z_{\text{val}}$ | Test data $Z_{\text{test}}$ |

**Step 2:** For $t \in \{1, \ldots, h\}$ hyperparameter choices:

- **Step 2a:** Run linear regression with $Z_{\text{train}}$ and $\lambda_t$ to obtain $\hat{\beta}(Z_{\text{train}}, \lambda_t)$

- **Step 2b:** Evaluate validation loss $L_{\text{val}}^t = L(\hat{\beta}(Z_{\text{train}}, \lambda_t); Z_{\text{val}})$

**Step 3:** Use best $\lambda_t$

- Choose $t' = \arg\min_t L_{\text{val}}^t$ with lowest validation loss

- Re-run linear regression with $Z_{\text{train}}$ and $\lambda_{t'}$ to obtain $\hat{\beta}(Z_{\text{train}}, \lambda_{t'})$
Cross Validation Hygiene

- The moment that test data is used for hyperparameter selection or to iterate on ML design choices, it should be treated as “contaminated”.
- Remember: Performance on contaminated test data is an overly **optimistic** estimate of the “true” test performance.

Q: What about validation data performance then? (yes, this is also overly optimistic)
Alternative Cross-Validation Algorithms

• If \( Z \) is small, then splitting it can reduce performance
  - Can use \( Z_{\text{train}} \cup Z_{\text{val}} \) in Step 3

• Alternative more thorough CV strategy: “\( k \)-fold” cross-validation
  - Split \( Z \) into \( Z_{\text{train}} \) and \( Z_{\text{test}} \)
  - Split \( Z_{\text{train}} \) into \( k \) disjoint sets \( Z_{\text{val}}^s \), and let \( Z_{\text{train}}^s = \bigcup_{s' \neq s} Z_{\text{val}}^s \)
  - Use \( \lambda' \) that works best on average across \( s \in \{1, \ldots, k\} \) with \( Z_{\text{train}} \)
  - Chooses better \( \lambda' \) than above strategy
Example: \( k = 3 \)-Fold Cross Validation

<table>
<thead>
<tr>
<th>Training data ( Z_{\text{train}}^3 )</th>
<th>Val data ( Z_{\text{val}}^3 )</th>
<th>Test data ( Z_{\text{test}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train data ( Z_{\text{val}}^2 )</td>
<td>Val data ( Z_{\text{val}}^2 )</td>
<td>Train data ( Z_{\text{val}}^2 )</td>
</tr>
<tr>
<td>Val data ( Z_{\text{val}}^1 )</td>
<td>Train data ( Z_{\text{train}}^1 )</td>
<td>Test data ( Z_{\text{test}} )</td>
</tr>
<tr>
<td>Train data ( Z_{\text{train}} )</td>
<td></td>
<td>Test data ( Z_{\text{test}} )</td>
</tr>
</tbody>
</table>

Compute vs. accuracy tradeoff: As \( k \to N \), model selection becomes more accurate, but algorithm becomes more computationally expensive.
Note: What Exactly Are “Hyperparameters”?

• Cross-Validation is a general, systematic trial-and-error procedure for selecting hyperparameters.

• Other hyperparameters too, not just the regularization $\lambda$.

• “Hyperparameters” are ML system properties / design choices that are not directly set in the optimization problem.

$$\hat{\beta}(Z) = \arg\min_{\beta} L(\beta; Z)$$

• Examples of other hyperparameters you could set with cross-validation:
  ▪ choice of feature maps in linear regression.
  ▪ data selection and other preprocessing procedures (coming up soon).
  ▪ linear regression versus another ML algorithm, altogether.
Minimizing the MSE Loss

• Recall that linear regression minimizes the loss

\[ L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 \]

• Closed-form solution: Compute a matrix expression derived using calculus

• Iterative Optimization-based solution: Search over candidate \( \beta \)
Vectorizing Linear Regression
Vectorizing Linear Regression

\[
\begin{bmatrix}
  f_\beta(x_1) \\
  \vdots \\
  f_\beta(x_n)
\end{bmatrix}
\]
Vectorizing Linear Regression

\[
\begin{bmatrix}
  f_\beta(x_1) \\
  \vdots \\
  f_\beta(x_n)
\end{bmatrix} =
\begin{bmatrix}
  \beta^T x_1 \\
  \vdots \\
  \beta^T x_n
\end{bmatrix}
\]
Vectorizing Linear Regression

\[
\begin{bmatrix}
    f_\beta(x_1) \\
    \vdots \\
    f_\beta(x_n)
\end{bmatrix} =
\begin{bmatrix}
    \beta^\top x_1 \\
    \vdots \\
    \beta^\top x_n
\end{bmatrix} =
\begin{bmatrix}
    \sum_{j=1}^{d} \beta_j x_{1,j} \\
    \vdots \\
    \sum_{j=1}^{d} \beta_j x_{n,j}
\end{bmatrix}
\]
Vectorizing Linear Regression

\[
\begin{bmatrix}
  f_\beta(x_1) \\
  \vdots \\
  f_\beta(x_n)
\end{bmatrix}
= \begin{bmatrix}
  \beta^\top x_1 \\
  \vdots \\
  \beta^\top x_n
\end{bmatrix}
= \begin{bmatrix}
  \sum_{j=1}^d \beta_j x_{1,j} \\
  \vdots \\
  \sum_{j=1}^d \beta_j x_{n,j}
\end{bmatrix}
= \begin{bmatrix}
  x_{1,1} & \cdots & x_{1,d} \\
  \vdots & \ddots & \vdots \\
  x_{n,1} & \cdots & x_{n,d}
\end{bmatrix}
\begin{bmatrix}
  \beta_1 \\
  \vdots \\
  \beta_d
\end{bmatrix}
\]
Vectorizing Linear Regression

\[
\begin{bmatrix}
f_\beta(x_1) \\
\vdots \\
f_\beta(x_n)
\end{bmatrix} = \begin{bmatrix}
\beta^\top x_1 \\
\vdots \\
\beta^\top x_n
\end{bmatrix} = \begin{bmatrix}
\sum_{j=1}^{d} \beta_j x_{1,j} \\
\vdots \\
\sum_{j=1}^{d} \beta_j x_{n,j}
\end{bmatrix} = \begin{bmatrix}
x_{1,1} & \cdots & x_{1,d} \\
\vdots & \ddots & \vdots \\
x_{n,1} & \cdots & x_{n,d}
\end{bmatrix} \begin{bmatrix}
\beta_1 \\
\vdots \\
\beta_d
\end{bmatrix}
\]
Vectorizing Linear Regression

$$
\begin{bmatrix}
  f_\beta(x_1) \\
  \vdots \\
  f_\beta(x_n)
\end{bmatrix} = \begin{bmatrix}
  \beta^\top x_1 \\
  \vdots \\
  \beta^\top x_n
\end{bmatrix} = \begin{bmatrix}
  \sum_{j=1}^{d} \beta_j x_{1,j} \\
  \vdots \\
  \sum_{j=1}^{d} \beta_j x_{n,j}
\end{bmatrix} = \begin{bmatrix}
  x_{1,1} & \cdots & x_{1,d} \\
  \vdots & \ddots & \vdots \\
  x_{n,1} & \cdots & x_{n,d}
\end{bmatrix} \begin{bmatrix}
  \beta_1 \\
  \vdots \\
  \beta_d
\end{bmatrix} = X\beta
$$
Vectorizing Linear Regression

\[
\begin{bmatrix}
  f_\beta(x_1) \\
  \vdots \\
  f_\beta(x_n)
\end{bmatrix} = 
\begin{bmatrix}
  \beta^T x_1 \\
  \vdots \\
  \beta^T x_n
\end{bmatrix} = 
\begin{bmatrix}
  \sum_{j=1}^{d} \beta_j x_{1,j} \\
  \vdots \\
  \sum_{j=1}^{d} \beta_j x_{n,j}
\end{bmatrix} = 
\begin{bmatrix}
  x_{1,1} & \cdots & x_{1,d} \\
  \vdots & \ddots & \vdots \\
  x_{n,1} & \cdots & x_{n,d}
\end{bmatrix} \begin{bmatrix}
  \beta_1 \\
  \vdots \\
  \beta_d
\end{bmatrix} = X\beta
\]

\[\iff\]

\[
\begin{bmatrix}
  y_1 \\
  \vdots \\
  y_n
\end{bmatrix}
\]
Vectorizing Linear Regression

\[
\begin{bmatrix}
    f_\beta(x_1) \\
    \vdots \\
    f_\beta(x_n)
\end{bmatrix}
= \begin{bmatrix}
    \beta^\top x_1 \\
    \vdots \\
    \beta^\top x_n
\end{bmatrix}
= \begin{bmatrix}
    \sum_{j=1}^{d} \beta_j x_{1,j} \\
    \vdots \\
    \sum_{j=1}^{d} \beta_j x_{n,j}
\end{bmatrix}
= \begin{bmatrix}
    x_{1,1} & \cdots & x_{1,d} \\
    \vdots & \ddots & \vdots \\
    x_{n,1} & \cdots & x_{n,d}
\end{bmatrix}
\begin{bmatrix}
    \beta_1 \\
    \vdots \\
    \beta_d
\end{bmatrix}
= X\beta
\]

\[\leq\]

\[
\begin{bmatrix}
    y_1 \\
    \vdots \\
    y_n
\end{bmatrix} = Y
\]

**Summary:** \( Y \approx X\beta \)

Note: n equations, d variables
Vectorizing Linear Regression

\[ Y \approx X \beta \]

\[ Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} \]
Vectorizing Mean Squared Error
Vectorizing Mean Squared Error

\[ L(\beta; Z) \]
Vectorizing Mean Squared Error

\[ L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2 \]
Vectorizing Mean Squared Error

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 = \frac{1}{n} \|Y - X\beta\|_2^2$$

$$\|Z\|_2^2 = \sum_{i=1}^{n} z_i^2$$
Intuition on Vectorized Linear Regression

• Rewriting the vectorized loss:

\[ n \cdot L(\beta; Z) = \|Y - X\beta\|_2^2 = \|Y\|_2^2 - 2Y^T X\beta + \|X\beta\|_2^2 \]
\[ = \|Y\|_2^2 - 2Y^T X\beta + \beta^T (X^T X)\beta \]

• **Side note:** Quadratic function of \( \beta \) with leading “coefficient” \( X^T X \)
  
  ▪ In one dimension, “width” of parabola \( ax^2 + bx + c \) is \( a^{-1} \)
  
  ▪ In multiple dimensions, “width” along direction \( v_i \) is \( \lambda_i^{-1} \), where \( v_i \) is an eigenvector of \( X^T X \) with eigenvalue \( \lambda_i \)
    
    ▪ Large width (small \( \lambda_i \)) along a direction \( v_i \) implies that parameter values along that direction affect the loss value less.
    
    ▪ This will be interesting to us later in class (“PCA”)
Intuition on Vectorized Linear Regression

Directions/magnitudes are given by eigenvectors/eigenvalues of $X^TX$
Strategy 1: Closed-Form Solution

• Recall that linear regression minimizes the loss:

\[ L(\beta; Z) = \frac{1}{n} \| Y - X\beta \|_2^2 \]

• Minimum solution has gradient equal to zero:

\[ \nabla_\beta L(\hat{\beta}(Z); Z) = 0 \]
Strategy 1: Closed-Form Solution

• Recall that linear regression minimizes the loss

\[ L(\beta; Z) = \frac{1}{n} \| Y - X\beta \|_2^2 \]

• Minimum solution has gradient equal to zero:

\[ \nabla_\beta L(\hat{\beta}; Z) = 0 \]
Strategy 1: Closed-Form Solution

• The gradient is

\[ \nabla_{\beta} L(\beta; Z) \]
Strategy 1: Closed-Form Solution

• The gradient is

\[ \nabla_\beta L(\beta; Z) = \nabla_\beta \frac{1}{n} \| Y - X\beta \|_2^2 \]
Strategy 1: Closed-Form Solution

The gradient is

\[
\nabla_\beta L(\beta; Z) = \nabla_\beta \frac{1}{n} \|Y - X\beta\|_2^2 = \nabla_\beta \frac{1}{n} (Y - X\beta)^\top (Y - X\beta)
= \frac{2}{n} \left[ \nabla_\beta (Y - X\beta)^\top \right] (Y - X\beta)
= -\frac{2}{n} X^\top (Y - X\beta)
= -\frac{2}{n} X^\top Y + \frac{2}{n} X^\top X\beta
\]
Strategy 1: Closed-Form Solution

• The gradient is

$$
\nabla_\beta L(\beta; Z) = \nabla_\beta \frac{1}{n} \|Y - X\beta\|_2^2 = -\frac{2}{n} X^T Y + \frac{2}{n} X^T X \beta
$$

• Setting $\nabla_\beta L(\hat{\beta}; Z) = 0$, we have $X^T X \hat{\beta} = X^T Y$

• Assuming $X^T X$ is invertible, we have

$$
\hat{\beta}(Z) = (X^T X)^{-1} X^T Y
$$

Compare this to: $Y \approx X\beta$

This is called the “pseudoinverse” of $X$
Strategy 1: Closed-Form Solution

• Setting $\nabla_\beta L(\hat{\beta}; Z) = 0$, we have $X^T X \hat{\beta} = X^T Y$

• Assuming $X^T X$ is invertible, we have

$$\hat{\beta}(Z) = (X^T X)^{-1} X^T Y$$
Note on Invertibility

• Closed-form solution only **unique** if $X^TX$ (size dxd) is invertible
  • Otherwise, **multiple solutions exist** to $X^TX\hat{\beta} = X^TY$
  • **Intuition**: Underconstrained system of linear equations

• Example:

\[
\begin{bmatrix}
1 & 1 \\
2 & 2
\end{bmatrix}
\begin{bmatrix}
\hat{\beta}_1 \\
\hat{\beta}_2
\end{bmatrix}
= \begin{bmatrix}
2 \\
4
\end{bmatrix}
\]

• In this case, any $\hat{\beta}_2 = 2 - \hat{\beta}_1$ is a solution
When Can this Happen?

• Case 1
  ▪ Fewer data examples than feature dimension (i.e., $n < d$)
    ▪ Remember: we are solving something like $Y \approx X\beta$
  ▪ **Solution**: Remove features so $d \leq n$
  ▪ **Solution**: Collect more data until $d \leq n$

• Case 2: Some feature is a linear combination of the others
  ▪ **Special case (duplicated feature)**: For some $j$ and $j'$, $x_{i,j} = x_{i,j'}$ for all $i$
  ▪ **Solution**: Remove linearly dependent features
  ▪ **Solution**: Use $L_2$ regularization (we will soon see why)
Shortcomings of Closed-Form Solution

• Computing $\hat{\beta}(Z) = (X^T X)^{-1} X^T Y$ can be challenging

• Computing $(X^T X)^{-1}$ is $O(d^3)$
  - $d = 10^4$ features $\rightarrow O(10^{12})$
  - Even storing $X^T X$ requires a lot of memory

• Numerical accuracy issues due to “ill-conditioning”
  - $X^T X$ is “barely” invertible
  - Then, $(X^T X)^{-1}$ has large variance along some dimension
  - Regularization helps (more on this later)
Iterative Optimization Algorithms

• Recall that linear regression minimizes the loss

\[ L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 \]

• Iteratively optimize \( \beta \)
  - Initialize \( \beta_1 \leftarrow \text{Init}(...) \)
  - For some number of iterations \( T \), update \( \beta_t \leftarrow \text{Step}(...) \)
  - Return \( \beta_T \)
Iterative Optimization Algorithms

- **Global search**: Try random values of $\beta$ and choose the best
  - I.e., $\beta_t$ independent of $\beta_{t-1}$
  - Very unstructured, can take a long time (especially in high dimension $d$)!

- **Local search**: Start from some initial $\beta$ and make local changes
  - I.e., $\beta_t$ is computed based on $\beta_{t-1}$
  - What is a “local change”, and how do we find good one?
Strategy 2: Gradient Descent

- **Gradient descent:** Update $\beta$ based on gradient $\nabla_\beta L(\beta; Z)$ of $L(\beta; Z)$:

  $$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- **Intuition:** The gradient is the direction along which $L(\beta; Z)$ changes most quickly as a function of $\beta$

- $\alpha \in \mathbb{R}$ is a hyperparameter called the **learning rate**
  - More on this later
Strategy 2: Gradient Descent

• Choose initial value for $\beta$

• Until we reach a minimum:
  ▪ Choose a new value for $\beta$ to reduce $L(\beta; Z)$

$L(\beta; Z)$

Figure by Andrew Ng
Strategy 2: Gradient Descent

- Choose initial value for $\beta$
- Until we reach a minimum:
  - Choose a new value for $\beta$ to reduce $L(\beta; Z)$
Strategy 2: Gradient Descent

• Choose initial value for $\beta$
• Until we reach a minimum:
  ▪ Choose a new value for $\beta$ to reduce $L(\beta; Z)$

Linear regression loss is convex, so no local minima

Figure by Andrew Ng
Strategy 2: Gradient Descent

- Initialize $\beta_1 = \vec{0}$
- Repeat until convergence:
  $$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla \beta L(\beta_t; Z)$$
- For linear regression, know the gradient from strategy 1

For in-place updates $\beta \leftarrow \beta - \alpha \cdot \nabla \beta L(\beta; Z)$, compute all components of $\nabla \beta L(\beta; Z)$ before modifying $\beta$. 
Strategy 2: Gradient Descent

- Initialize $\beta_1 = \overrightarrow{0}$
- Repeat until convergence:

\[
\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)
\]

- For linear regression, know the gradient from strategy 1
Strategy 2: Gradient Descent

- Initialize $\beta_1 = \vec{0}$
- Repeat until $\|\beta_t - \beta_{t+1}\|_2 \leq \epsilon$:

  $$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1

Hyperparameter defining convergence
Aside: Gradient As Sum of Sample-Wise Gradients

(Equivalent to our earlier matrix expression of gradient)

• By linearity of the gradient, we have

\[
\nabla_\beta L(\beta; Z) = \sum_{i=1}^{n} \nabla_\beta (y_i - \beta^T x_i)^2 = \sum_{i=1}^{n} 2(y_i - \beta^T x_i)x_i
\]

• The gradient term induced by a single training data sample is:

\[
\nabla_\beta (y_i - \beta^T x_i)^2 = 2(y_i - \beta^T x_i)x_i
\]

• I.e., the current error \(y_i - \beta^T x_i\) times the feature vector \(x_i\)

“Large error samples induce large changes to \(\beta\), proportional to their feature values.”
Strategy 2: Gradient Descent

\[ h(x) = -900 - 0.1x \]

\[ f_\beta(x) \quad \text{and} \quad L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]
Strategy 2: Gradient Descent

\[ f_\beta(x) \]

\[ L(\beta; Z) \]

Minimizer of loss function
Choice of Learning Rate $\alpha$

Problem: $\alpha$ too small
- $L(\beta; Z)$ decreases slowly

Problem: $\alpha$ too large
- $L(\beta; Z)$ increases!

Plot $L(\beta_t; Z_{\text{train}})$ vs. $t$ to diagnose these problems
Choice of Learning Rate $\alpha$

- $\alpha$ is a hyperparameter for gradient descent that we need to choose
  - Can set just based on training data

- Rule of thumb
  - $\alpha$ too small: Loss decreases slowly
  - $\alpha$ too large: Loss increases!

- Try rates $\alpha \in \{1.0, 0.1, 0.01, \ldots\}$ (can tune further once one works)
Comparison of Strategies

• Closed-form solution
  ▪ No hyperparameters
  ▪ Slow if \( n \) or \( d \) are large

• Gradient descent
  ▪ Need to tune \( \alpha \)
  ▪ Scales to large \( n \) and \( d \)

• For linear regression, there are better optimization algorithms, but gradient descent is very general
  ▪ Accelerated gradient descent is an important tweak that improves performance in practice (and in theory)
Loss Minimization View of ML

• Two design decisions
  ▪ **Model family**: What are the candidate models $f$? (E.g., linear functions)
  ▪ **Loss function**: How to define “approximating”? (E.g., MSE loss)
Loss Minimization View of ML

• **Three design decisions**
  - **Model family:** What are the candidate models $f$? (E.g., linear functions)
  - **Loss function:** How to define “approximating”? (E.g., MSE loss)
  - **Optimizer:** How do we minimize the loss? (E.g., gradient descent)