Lecture 11 : Learning parametric models

- We have until now looked at two simple parametric models
  - → linear regression
  - → logistic regression
  - → generalized linear models

- Parametric models assume a functional form described a fixed number of parameters

- Learning a parametric model implies tuning the parameters to fit the training dataset

- In the next two lectures, we will discuss basic principles for learning these models

- Key components of any parametric ML algorithm:

  - The data $\{ \underline{x}_i , y_i \}_{i=1}^{N}$

  - A model with parameters $\underline{\theta}$ [e.g. $\hat{y} = \underline{x}^T \underline{\theta}$]

  - A loss function $L(y, \hat{y})$ to quantify goodness of model predictions during training

  - An optimization algorithm to tune the model's parameters so as to minimize the loss

- Key components of any parametric ML algorithm:

  - The data $\{\underline{x}_i, y_i\}_{i=1}^{N}$

  → - A model with parameters $\underline{\theta}$ [e.g. $\hat{y} = \underline{x}^T \underline{\theta}$]

  - A loss function $L(y, \hat{y})$ to quantify goodness of model predictions during training

  - An optimization algorithm to tune the model's parameters so as to minimize the loss

Nonlinear Parametric Functions (Regression)

- A nonlinear regression model

$$y = \underbrace{f_{\underline{\theta}}(\underline{x})} + \epsilon$$

sometimes also written as $f(\underline{x}\,;\underline{\theta})$

- $f_{\underline{\theta}}(\underline{x})$ could be any nonlinear function, which depend on some model parameter $\underline{\theta}$ that control the shape of the function

- Different values of $\underline{\theta}$ result in different functions $f_{\underline{\theta}}(\cdot)$

- In mathematical terms, we say we have a parametric family of functions

$$\{ f_{\underline{\theta}}(\cdot) \ \text{ s.t } \ \underline{\theta} \in \underbrace{\underline{\Theta}} \}$$

space of all possible parameter vectors

Nonlinear Parametric Functions (Regression)

A nonlinear regression model

$$y = f_{\underline{\theta}}(\underline{x}) + \epsilon$$

- If noise ϵ is taken to be Gaussian with zero mean and variance σ_ϵ^2 we obtain a Gaussian likelihood

$$p(y \mid \underline{x}; \underline{\theta}) = \mathcal{N}\left(f_{\underline{\theta}}(\underline{x}), \sigma_\epsilon^2 \right)$$

- Note that in the linear case, $f_{\underline{\theta}}(\underline{x}) = \underline{x}^T \underline{\theta}$ and

$$p(y \mid \underline{x}; \underline{\theta}) = \mathcal{N}\left(\underline{x}^T \underline{\theta}, \sigma_\epsilon^2 \right)$$

Nonlinear Parametric Functions (Classification)

- Recall linear classification i.e. logistic regression
  - Logit : $z = \underline{x}^T \underline{\theta}$
  - $p(y=1 | \underline{x})$: $h(z) = \dfrac{e^z}{1+e^z} = \dfrac{e^{\underline{x}^T \underline{\theta}}}{1+e^{\underline{x}^T \underline{\theta}}}$

- Nonlinear classification can be constructed by considering $z = f_{\underline{\theta}}(\underline{x})$ as a generalization of the logistic regression

$$p(y=1 | \underline{x}) = \frac{e^{f_{\underline{\theta}}(\underline{x})}}{1 + e^{f_{\underline{\theta}}(\underline{x})}}$$

- Similarly, we could also have a multi-class nonlinear classifier by generalizing multi-class logistic regression model (recall softmax)

- Key components of any parametric ML algorithm:

  - The data $\{\underline{x}_i , y_i\}_{i=1}^{N}$

  - A model with parameters $\underline{\theta}$ [e.g. $\hat{y} = \underline{x}^T \underline{\theta}$]

→ - A loss function $L(y, \hat{y})$ to quantify goodness of model predictions during training

  - An optimization algorithm to tune the model's parameters so as to minimize the loss

Loss functions

- After choosing a certain parametric model class, the next step is to "learn" the model — find suitable values of parameters so that the model describes the true (but often unknown) input-output relationship as accurately as possible

- For parametric models, "learning" is typically framed as an optimization

$$\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} \underbrace{\frac{1}{N} \sum_{i=1}^{N} \overbrace{L\left(y_i \, , f_{\underline{\theta}}(\underline{x}_i)\right)}^{\text{loss function}}}_{\text{Cost function } J(\underline{\theta})} \quad \Bigg]$$

Usually solved using some numerical optimization method

- We seek to minimize a cost function which is the average of some user-defined loss function L evaluated on the training data

Loss function is a proxy of Generalization

- Natural Idea : Find the value of $\underline{\theta}$ that fits the training data well

- However, our ultimate goal is not to fit the training data very well but rather to find a model that can generalize to new data

  - In other words, we are actually interested in solving :

$$\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} \underbrace{E_{new}(\underline{\theta})} \longrightarrow \mathbb{E}_{*}\left[E\left(y, \hat{y}(\underline{x}_{*}; \underline{\theta})\right)\right]$$

  - Issue is $E_{new}(\underline{\theta})$ is unknown since $p(\underline{x}, y)$ is unknown

However, it is still important to keep in mind that

Training objective $\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} \frac{1}{N} \sum_{i=1}^{N} L\left(y_i \, , f_{\underline{\theta}}(\underline{x}_i)\right)$

is only a proxy for the actual objective

→ Viewing the training objective as a proxy for generalization will help us in choosing how we approach the optimization problem!

- Optimization accuracy vs Statistical accuracy

  It is not meaningful to optimize $J(\underline{\theta})$ with greater accuracy than the statistical error in the estimate.

  ↳ difficult to determine though

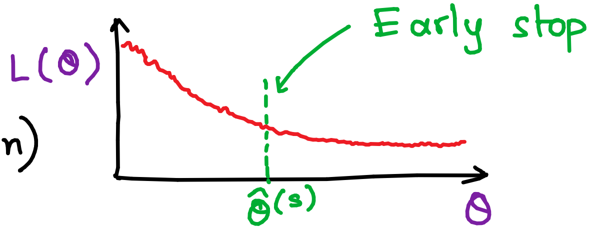
- Choice of loss function (loss function ≠ error function)

  One should choose a loss function with the aim of making the optimization problem easier to solve

  $$\hat{\underline{\theta}} = \underset{\underline{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, f_{\underline{\theta}}(\underline{x}_i)\right)$$

  (e.g. use convex loss functions)

- Early Stopping (or implicit regularization)



$\hat{\theta}^{(0)} \rightarrow \hat{\theta}^{(1)} \rightarrow \hat{\theta}^{(2)} \ldots\ldots \hat{\theta}^{(s)} \ldots\ldots \rightarrow$

- Explicit regularization by adding a parameter-penalty term in the cost function (e.g. L_2-regularization)

Different loss functions (for regression) → give different solutions $\hat{\Theta}$

- Squared error loss: $L(y, \hat{y}) = (y - \hat{y})$ ← default choice for linear regression
    - Maximum likelihood perspective: Noise ϵ is Gaussian, $\epsilon \sim N(0, \sigma_\epsilon^2)$
- Absolute error loss: $L(y, \hat{y}) = |y - \hat{y}|$ ← robust to outliers since it grows more slowly for large errors
    - Maximum likelihood perspective: Noise ϵ is Laplace, $\epsilon \sim \mathcal{L}(0, b_\epsilon)$

$$\mathcal{L}(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

- ϵ-insensitive loss (extension of absolute error loss)

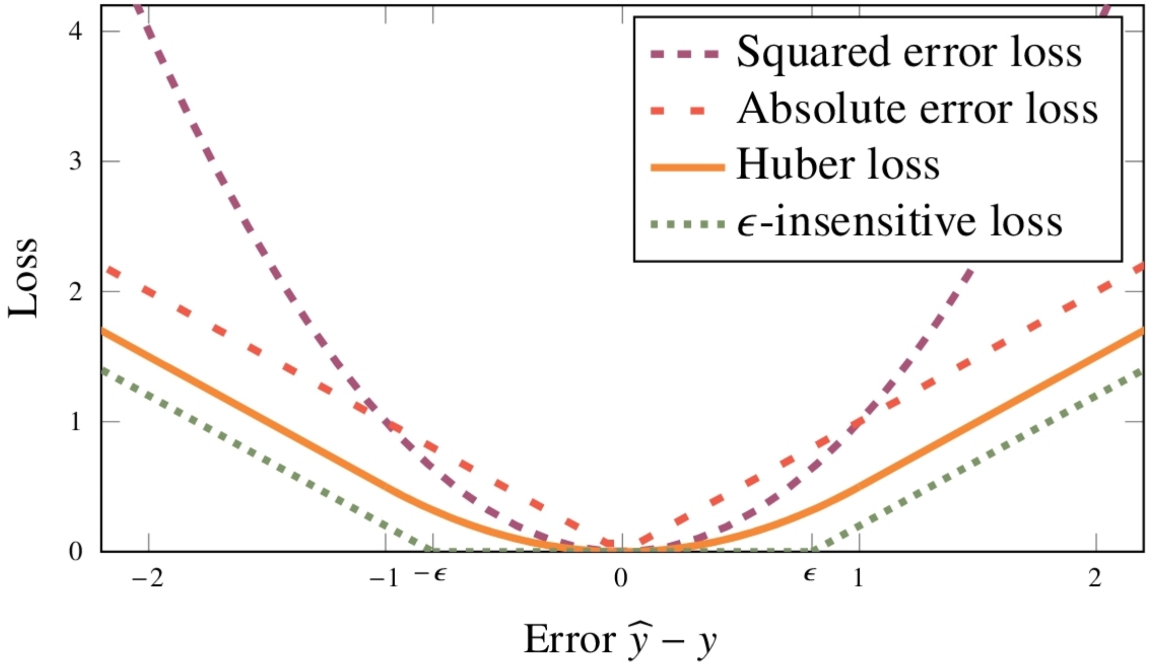
$$L(y, \hat{y}) = \begin{cases} 0 & \text{if } |y-\hat{y}| < \epsilon \\ |y-\hat{y}| - \epsilon & \text{otherwise} \end{cases}$$

↖ chosen by user

- Huber loss (hybrid between squared error loss and absolute error loss)

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & , \ |y-\hat{y}| < 1 \\ |y-\hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases}$$

Different loss functions (for regression)



Squared error loss

$$L(y, \hat{y}) = (y - \hat{y})^2$$

Absolute error loss

$$L(y, \hat{y}) = |y - \hat{y}|$$

ϵ - insensitive loss

$$L(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}$$

ϵ - insensitive loss will turn out to be useful for support vector regression (SVR)

Huber loss

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases}$$

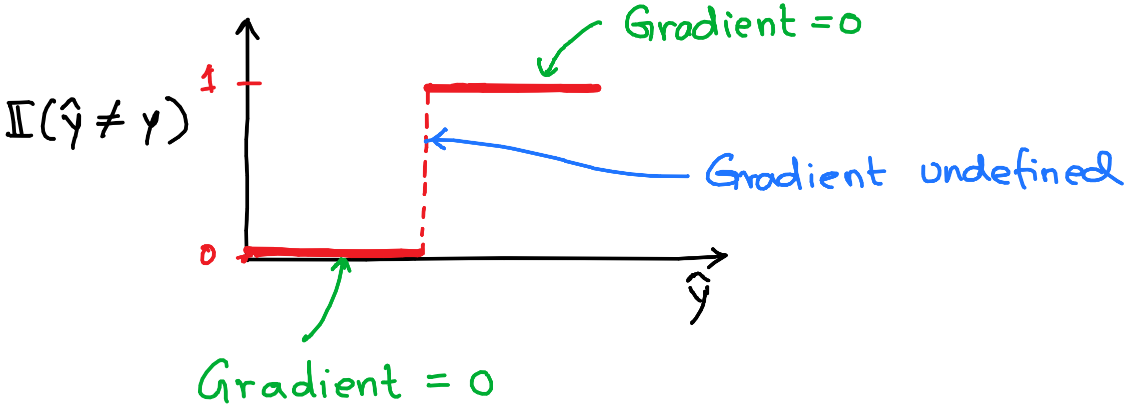
Different loss functions (for classification)

Let's look at loss functions for binary classification first

- An intuitive loss function for is the misclassification loss

$$L(y, \hat{y}) = \underset{\text{indicator function}}{\underset{\uparrow}{\mathbb{I}}}(\hat{y} \neq y = \begin{cases} 0 & y = \hat{y} \\ 1 & y \neq \hat{y} \end{cases}$$

Although intuitive, this loss is rarely used in practice, because it is hard to optimize → has zero gradients



Different loss functions (for classification)

- Cross-entropy loss forms a natural choice for a binary classifier that predicts class probabilities $p(y=1|\underline{x})$ in terms of $g(\underline{x})$

$$L(y, \underset{\overset{\uparrow}{g(\underline{x})}}{\hat{y}}) = \begin{cases} \ln g(\underline{x}) & \text{if } y = 1 \\ 1 - \ln g(\underline{x}) & \text{if } y = -1 \end{cases}$$

- Another useful class of loss functions can be defined using the concept of margins

- Many classifiers can be constructed by thresholding some real-valued function $f(\underline{x};\underline{\theta})$ at 0. We can write the class prediction as

$$\boxed{\hat{y}(\underline{x}) = \text{sign}\{f(\underline{x};\underline{\theta})\}}$$

(for binary classes) $\{-1, 1\}$

E.g. Logistic regression can be brought into this form by $f(\underline{x}) = \underline{x}^T\underline{\theta}$

Concept of margin for (binary) classifiers

- The decision boundary of any classifier of the form

$$\hat{y}(\underline{x}) = \text{sign}\{ f(\underline{x}; \underline{\theta}) \}$$

$$\begin{tikzcd} & +1 \\ \hat{y}(x) \arrow[ur] \arrow[dr] & \\ & -1 \end{tikzcd}$$

is given by the values of $\underline{x}$ for which $f(\underline{x}) = 0$

- The margin of a classifier for a data point $(\underline{x}, y)$ is $y \cdot f(\underline{x})$

$$\left.\begin{aligned} f(\underline{x}) &\rightarrow + \\ y &\rightarrow + \end{aligned}\right\} \rightarrow y \cdot f(\underline{x}) \rightarrow +\text{ve margin}$$

$$\left.\begin{aligned} f(\underline{x}) &\rightarrow - \\ y &\rightarrow - \end{aligned}\right\} \rightarrow y \cdot f(\underline{x}) \rightarrow +\text{ve margin}$$

- If classification is correct, margin is <u>positive</u>
- If y and $f(\underline{x})$ have different signs, margin is <u>negative</u>

  (meaning incorrect classification)
- Data points with small margins are closer to decision boundary

Margin-based perspective of logistic loss

- In the lecture on logistic regression, we started out with a class probability perspective, modelling using $p(y=1 \mid \underline{x}) = g(\underline{x})$, then arrived at cross-entropy loss, and later for $g(x)$ modelled using the logistic function, we obtained the logistic loss

$$L(y, \hat{y}) = \ln\left(1 + e^{-y\cdot(\underline{x}^T\underline{\theta})}\right)$$

- Without linking the probabilistic perspective, we could consider the logistic loss as a generic margin-based loss

$$L(y, f(\underline{x})) = \ln\left(1 + e^{-\overbrace{y\cdot f(\underline{x})}^{\text{margin of the classifier}}}\right)$$

Hence,

- we postulate a classifier according to $\hat{y}(\underline{x}) = \text{sign}\{f(\underline{x};\underline{\theta})\}$, and
- then learn the parameters of $f(\underline{x};\underline{\theta})$ by minimizing $L(y, f(\underline{x}))$

Other margin-based loss functions for classification

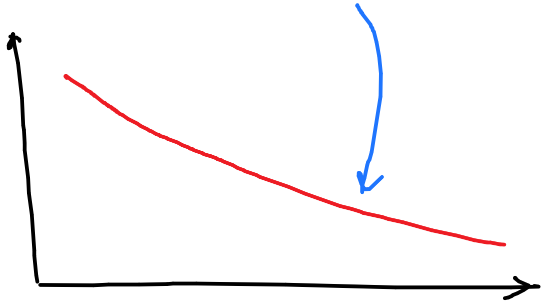
- Misclassification loss as margin-based loss

Misclassification loss

$$L(y, \hat{y}) = \mathbb{I}(\hat{y} \neq y) = \begin{cases} 0 & y = \hat{y} \\ 1 & y \neq \hat{y} \end{cases}$$

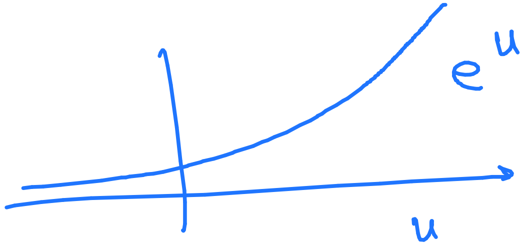
$$L(y, f(\underline{x})) = \begin{cases} 1 & \text{if } \; y \cdot f(\underline{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

- In principle, any DECREASING function is a candidate loss function



- e.g. Exponential loss

$$L(y, f(\underline{x})) = \exp\left(-y \cdot f(x)\right)$$



- Not very robust to outliers, due to the exponential growth for negative margins

- Hinge loss (will be used in support vector machine) [no probabilistic interpretation possible]

$$L(y, f(\underline{x})) = \begin{cases} 1 - y \cdot f(\underline{x}) & \text{for } y \cdot f(\underline{x}) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- Squared hinge loss (has probabilistic interpretation)

$$L(y, f(\underline{x})) = \begin{cases} (1 - y \cdot f(\underline{x}))^2 & \text{for } y \cdot f(\underline{x}) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

  - less robust to outliers

- Huberized squared hinge loss

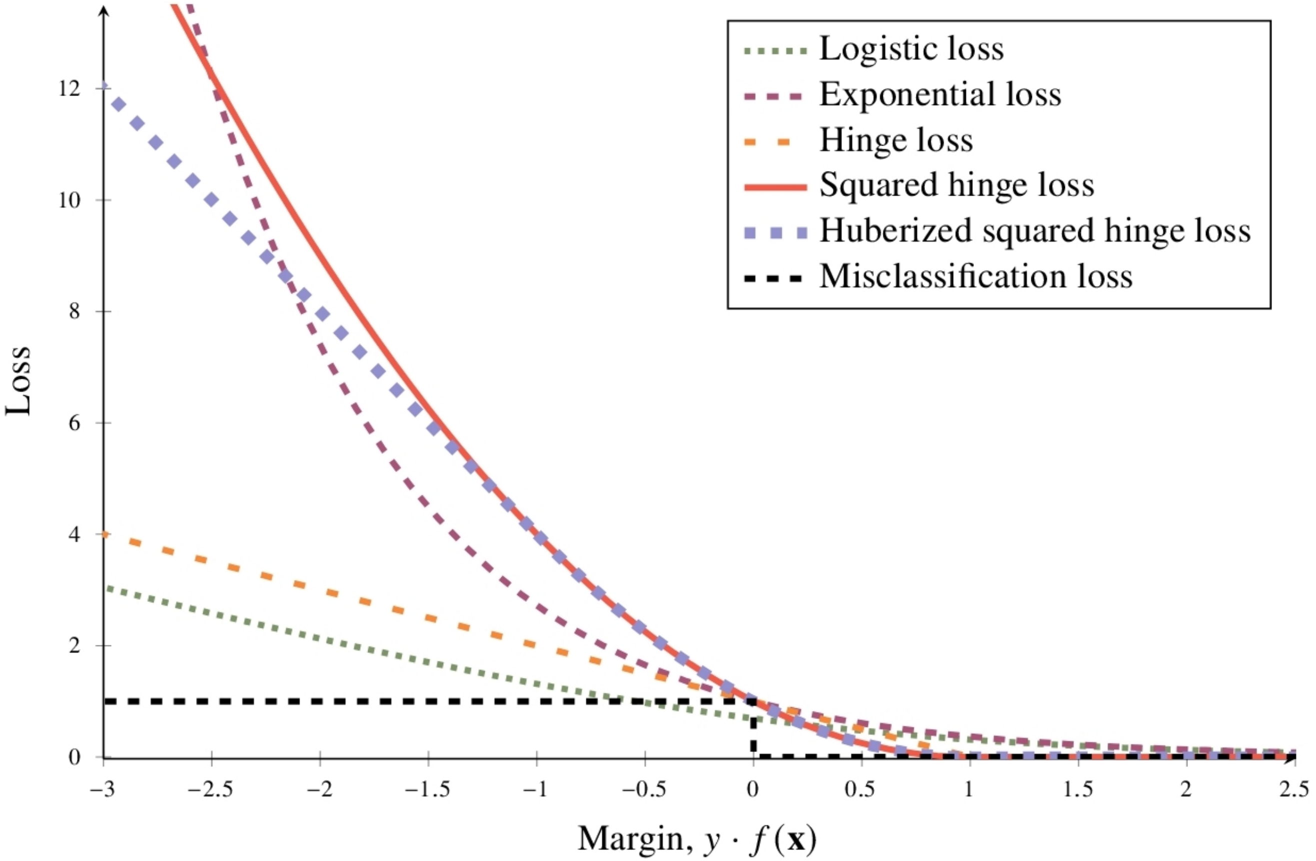
$$L(y, f(\underline{x})) = \begin{cases} -4y \cdot f(\underline{x}) & \text{for } y \cdot f(\underline{x}) \leq -1 \quad \text{(linear margin)} \\ (1 - y \cdot f(\underline{x}))^2 & \text{for } -1 \leq y \cdot f(\underline{x}) \leq 1 \quad \text{(squared hinge loss)} \\ 0 & \text{otherwise} \end{cases}$$

Misclassification loss

$$L(y, f(\underline{x})) = \begin{cases} 1 & \text{if } \; y \cdot f(\underline{x}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

Exponential loss

$$L(y, f(\underline{x})) = \exp\left(-y \cdot f(x)\right)$$



Hinge loss

$$L(y, f(\underline{x})) = \begin{cases} 1 - y \cdot f(\underline{x}) & \text{for } \; y \cdot f(\underline{x}) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Squared hinge loss

$$L(y, f(\underline{x})) = \begin{cases} (1 - y \cdot f(\underline{x}))^2 & \text{for } \; y \cdot f(\underline{x}) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Huberized squared hinge loss

$$L(y, f(\underline{x})) = \begin{cases} -4\, y \cdot f(\underline{x}) & \text{for } \; y \cdot f(\underline{x}) \leq -1 \quad (\\ (1 - y \cdot f(\underline{x}))^2 & \text{for } \; -1 \leq y \cdot f(\underline{x}) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Regularization

- The idea of regularization in a parametric model is to

"keep the parameters $\hat{\underline{\theta}}$ small unless the data really convinces us otherwise"

- Two types of regularization

  → Explicit regularization, e.g L_2-regularization

  → Implicit regularization, e.g. early stopping

Explicit regularization

L_2 - regularization

$$\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} \frac{1}{N} \| \underline{y} - \underline{\underline{X}}\,\underline{\theta} \|_2^2 + \lambda \| \underline{\theta} \|_2^2$$

- Admits closed-form solution

$$\hat{\underline{\theta}} = \left(\underline{\underline{X}}^T \underline{\underline{X}} + N \lambda \underline{\underline{I}} \right)^{-1} \underline{\underline{X}}^T \underline{y}$$

- Typically does not produce sparse solution

L_1 - regularization (LASSO)

$$\hat{\underline{\theta}} = \arg\min_{\underline{\theta}} \frac{1}{N} \| \underline{y} - \underline{\underline{X}}\,\underline{\theta} \|_2^2 + \lambda \| \underline{\theta} \|_1^2$$

$$\| \underline{\theta} \|_1 = | \theta_0 | + | \theta_1 | + \cdots + | \theta_p |$$

- No closed-form solution available

Have to do numerical optimization

- Produces sparse solutions, where only a few of the parameters are non-zero

In a sense, L_1-regularization can "switch-off" some inputs (by setting the corresponding parameter θ_k to zero)

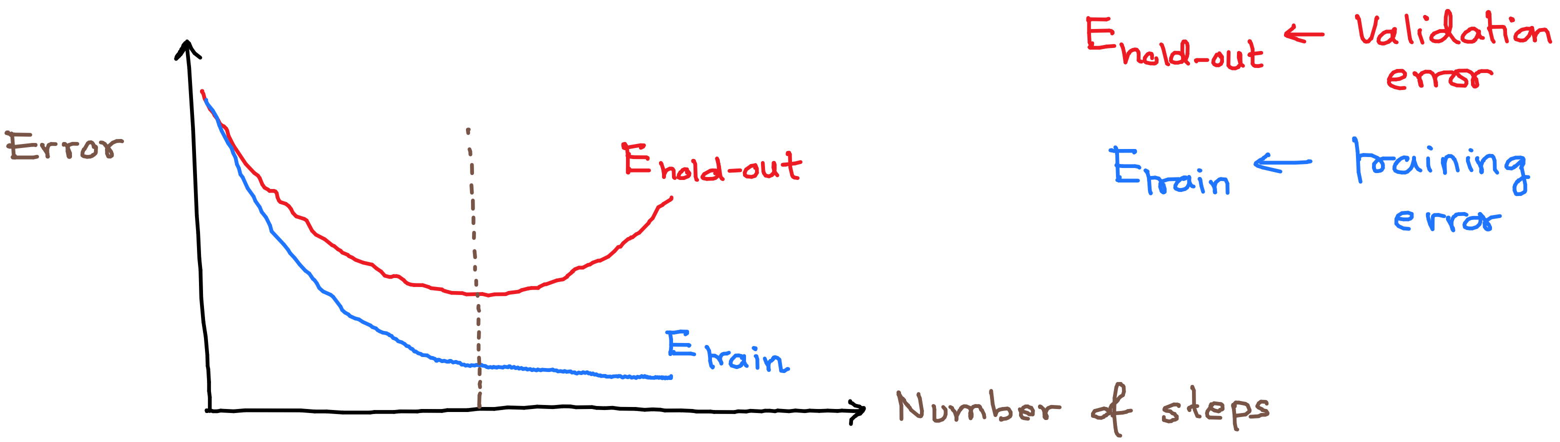
Implicit Regularization

- There are alternative ways to achieve regularization without explicitly modifying the cost function

- One such way is Early Stopping

  ↳ aborting an iterative numerical optimization before it has reached the minimum of the cost function

- Set aside some hold-out validation data for computing $E_{hold\text{-}out}$ and use it to determine the stopping point


Error
Number of steps
$E_{hold\text{-}out}$
E_{train}
$E_{hold\text{-}out}$ ← Validation error
E_{train} ← training error