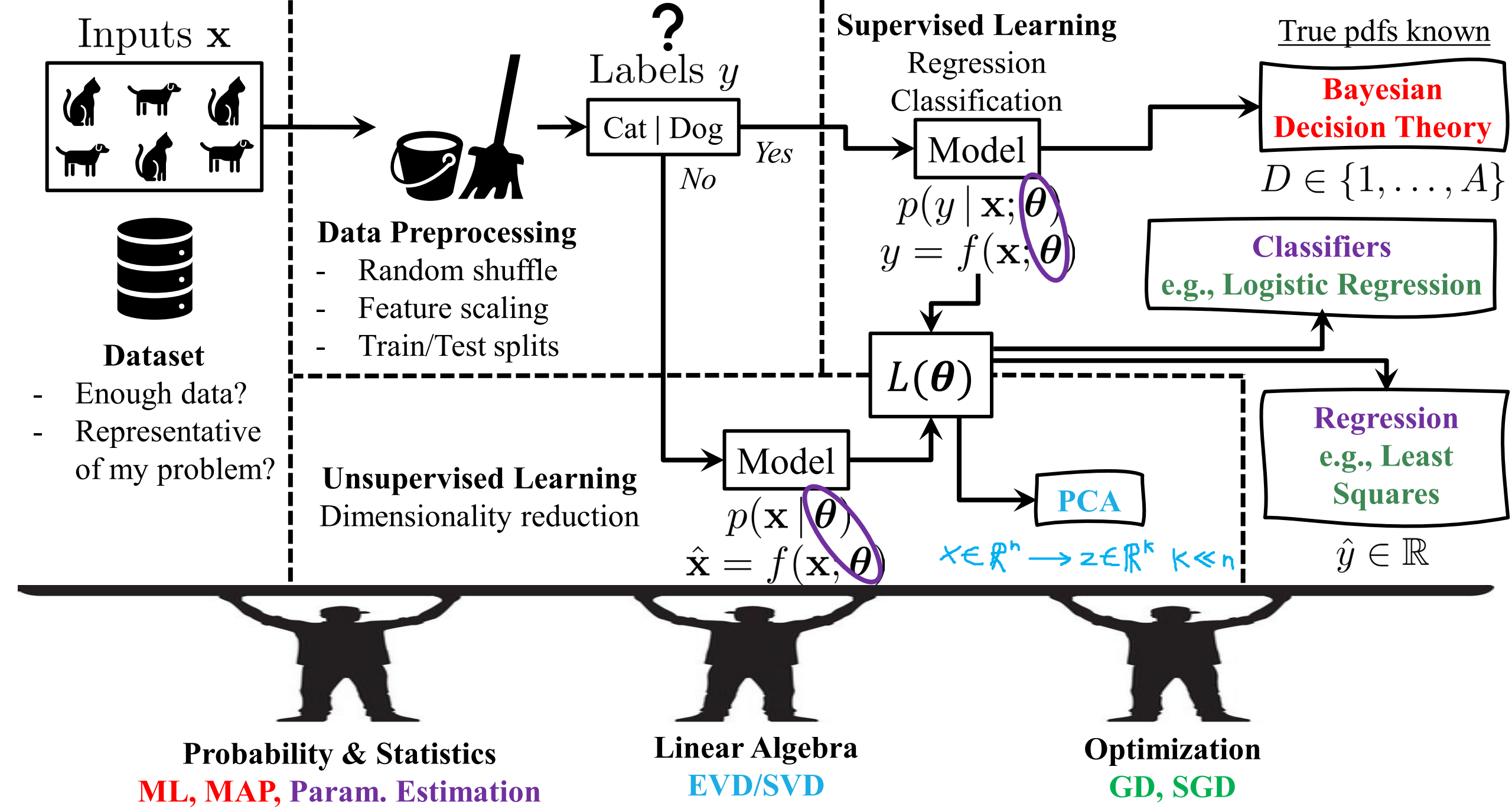
EECE 5644: Model Selection: Hyperparameter Tuning & Cross-validation

Mark Zolotas

E-mail: m.zolotas@northeastern.edu
Webpage: https://coe.northeastern.edu/people/zolotas-mark/

Tentative Course Outline (Wks. 3-4)

Topics	Dates	Assignments	Additional Reading
~~Naïve Bayes Classifier & *Homework 0 Practice Lab*~~	~~07/18~~	~~**Homework 2** released on Canvas on 07/22 **Due 08/01**~~	~~N/A~~
~~Model Fitting/Training: Bayesian Parameter Estimation~~	~~07/19-20~~		~~Chpts. 4.1-4.3, 8.7.2-3 Murphy 2022~~
~~Logistic Regression~~	~~07/21~~		~~Chpt. 10 Murphy 2022~~
Model Selection: Hyperparameter Tuning, k-fold Cross-Validation	07/26	**Homework 3** released on Canvas on 07/29 **Due 08/08**	Chpts. 4.5, 5.2, 5.4.3 Murphy 2022
Regularization, Ridge and Lasso Regression	07/27		Chpts. 4.5, 11.1-11.4 Murphy 2022
Neural Networks: Multilayer Perceptrons & Backpropagation	07/28		Chpts. 13.1-13.5 Murphy 2022


Inputs x
Dataset
- Enough data?
- Representative of my problem?
Data Preprocessing
- Random shuffle
- Feature scaling
- Train/Test splits
?
Labels y
Cat | Dog
Yes
No
Supervised Learning
Regression
Classification
Model
$p(y \mid \mathbf{x}; \boldsymbol{\theta})$
$y = f(\mathbf{x}; \boldsymbol{\theta})$
True pdfs known
Bayesian Decision Theory
$D \in \{1, \ldots, A\}$
Classifiers
e.g., Logistic Regression
$L(\boldsymbol{\theta})$
Regression
e.g., Least Squares
$\hat{y} \in \mathbb{R}$
Unsupervised Learning
Dimensionality reduction
Model
$p(\mathbf{x} \mid \boldsymbol{\theta})$
$\hat{\mathbf{x}} = f(\mathbf{x}; \boldsymbol{\theta})$
PCA
$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k \quad k \ll n$
Probability & Statistics
ML, MAP, Param. Estimation
Linear Algebra
EVD/SVD
Optimization
GD, SGD

ML & MAP Parameter Estimation

- Given i.i.d. samples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ from a dataset, take **log likelihood (LL)**:

$$\mathrm{LL}(\boldsymbol{\theta}) = \log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\mathbf{x}^{(i)} \mid \boldsymbol{\theta})$$

- **MLE:** Good values of $\boldsymbol{\theta}$ should assign high probability to D

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\mathbf{x}^{(i)} \mid \boldsymbol{\theta})$$

*Susceptible to **overfitting**; **underperforms** when **small** N*

- **MAP:** Adds a **prior** (regularization term) to tackle overfitting

$$\hat{\boldsymbol{\theta}}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\boldsymbol{\theta} \mid \mathbf{x}^{(i)}) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \Big[\log p(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\Big]$$

Sigmoid Function

- Takes a probabilistic approach to learning discriminative functions
- Desire $g(\mathbf{w}^T\mathbf{x})$ to output probabilities $p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta})$

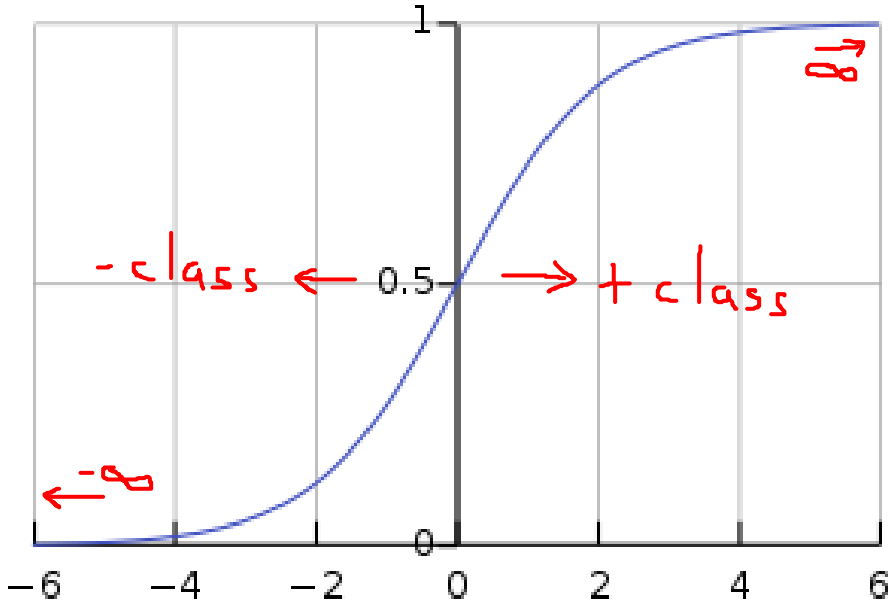
$$0 \leq g(\mathbf{w}^\intercal \mathbf{x}) \leq 1$$

- Model **predictions/hypotheses**:

$$\hat{y} = g(\mathbf{w}^\intercal \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\intercal \mathbf{x}}}$$

where $g(z) = \dfrac{1}{1 + e^{-z}} = \dfrac{e^z}{e^z + 1}$

Sigmoid/Logistic Function



Binary Logistic Regression – Decision Boundary

- In a 0-1 loss setting with $g(\mathbf{w}^T\mathbf{x})$ outputting $p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta})$, our optimal decision rule is to predict $y = 1$ iff:

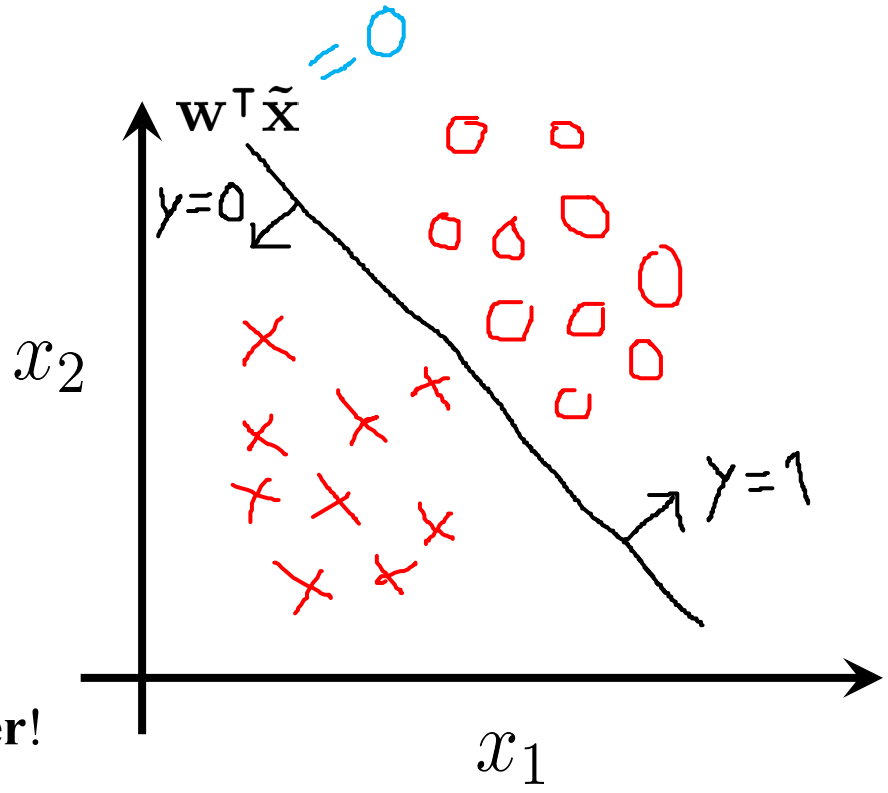
$$p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta}) > p(y = 0 \mid \mathbf{x}; \boldsymbol{\theta})$$
$$g(\mathbf{w}^\intercal \tilde{\mathbf{x}}) > 1 - g(\mathbf{w}^\intercal \tilde{\mathbf{x}})$$

- Rearrange and take logs:

$$\log g(\mathbf{w}^\intercal \tilde{\mathbf{x}}) - \log\left(1 - g(\mathbf{w}^\intercal \tilde{\mathbf{x}})\right) > 0$$

- Decision boundary:

$$\sum_{j=0}^{n} x_j w_j > 0 \quad \textbf{Linear classifier!}$$



Binary Logistic Regression – MLE/NLL

$$\arg\min_{\boldsymbol{\theta}} \text{NLL}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} -\frac{1}{N} \log \prod_{i=1}^{N} p(y^{(i)} \mid \mathbf{x}^{(i)}; \boldsymbol{\theta})$$

Supervised conditional setting

Binary Cross Entropy

$$= \arg\min_{\boldsymbol{\theta}} -\frac{1}{N} \sum_{i=1}^{N} \log \left[g(\mathbf{w}^\intercal \tilde{\mathbf{x}}^{(i)})^{y^{(i)}} (1 - g(\mathbf{w}^\intercal \tilde{\mathbf{x}}^{(i)}))^{(1-y^{(i)})} \right]$$

- Find critical point where $\frac{d\text{NLL}(\boldsymbol{\theta})}{d\mathbf{w}} = 0$, so first derive gradient of $\text{NLL}(\boldsymbol{\theta})$:

$$\frac{d\text{NLL}(\boldsymbol{\theta})}{d\mathbf{w}} = -\frac{1}{N} \sum_{i=1}^{N} \frac{d}{d\mathbf{w}} \left[y^{(i)} \log g(\mathbf{w}^\intercal \tilde{\mathbf{x}}^{(i)}) + (1 - y^{(i)}) \log(1 - g(\mathbf{w}^\intercal \tilde{\mathbf{x}}^{(i)})) \right]$$

$$= -\frac{1}{N} \sum_{i=1}^{N} \left[\left(y^{(i)} - g(\mathbf{w}^\intercal \tilde{\mathbf{x}}^{(i)}) \right) \tilde{\mathbf{x}}^{(i)} \right]$$

Binary Logistic Regression – Gradient Descent

1. Initialize $\boldsymbol{\theta}^{(0)}$
2. Repeat until convergence:

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \alpha \nabla \text{NLL}(\boldsymbol{\theta}^{(t)}) \\ &= \boldsymbol{\theta}^{(t)} - \alpha \left(\frac{1}{N} \sum_{i=1}^{N} \left[\left(g(\mathbf{w}^\intercal \tilde{\mathbf{x}}^{(i)}) - y^{(i)} \right) \tilde{\mathbf{x}}^{(i)} \right] \right)\end{aligned}$$

3. When $\mathbf{g}(\boldsymbol{\theta}) = \mathbf{0}$, then we have derived $\mathbf{w}^* = \hat{\boldsymbol{\theta}}_{MLE}$ using GD

Overfitting

- Training models using MLE, or $L(\boldsymbol{\theta})$ in general, risks perfectly fitting the training data D_{train} and **not generalizing** well to unseen, future data…

Sample Error

$$L(\boldsymbol{\theta}; D_{\text{train}}) = \frac{1}{|D_{\text{train}}|} \sum_{(\mathbf{x},y)\in D_{\text{train}}} l(y, f(\mathbf{x}; \boldsymbol{\theta}))$$

- Assume we had access to the true distribution $p^*(\mathbf{x}, y)$ responsible for generating D_{train}, then the **theoretical expected loss** would be:

True Error

$$L(\boldsymbol{\theta}; p^*) = \mathbb{E}_{p^*(\mathbf{x},y)} \left[l(y, f(\mathbf{x}; \boldsymbol{\theta})) \right]$$

But we don't know p^* and can only measure sample error

- **Generalization gap:** $L(\boldsymbol{\theta}; p^*) - L(\boldsymbol{\theta}; D_{\text{train}})$

- Large generalization gap (low empirical, high theoretical loss) = **Overfitting**

Detecting Overfitting

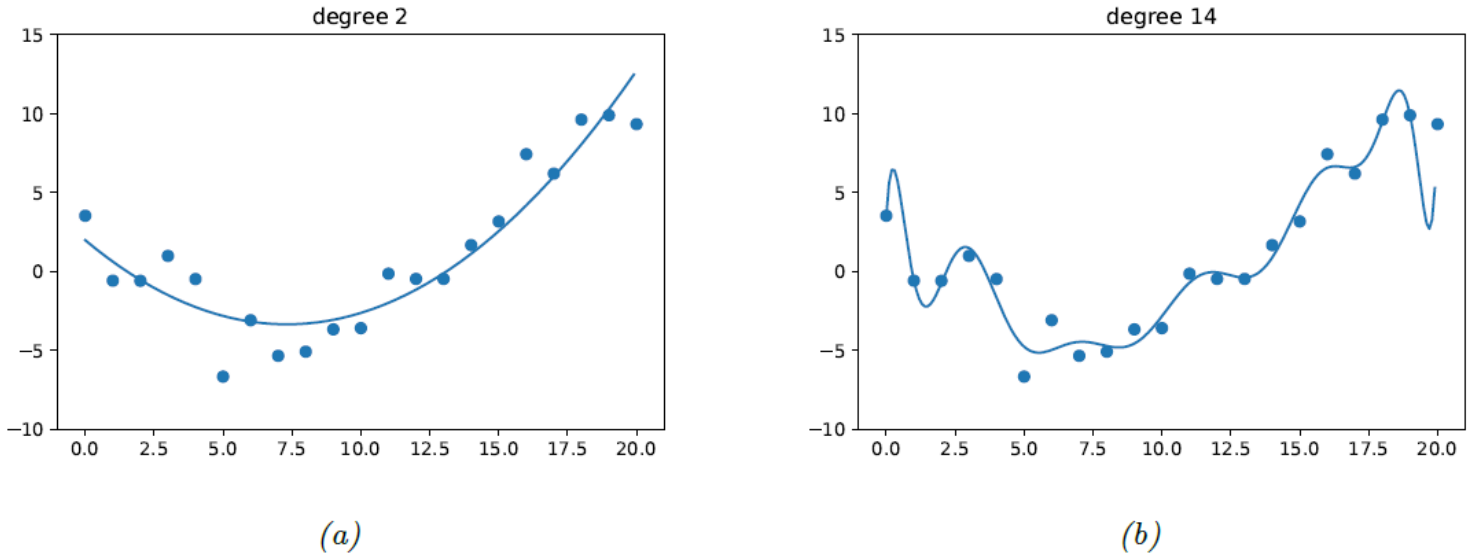


Figure 1.7: (a-c) Polynomials of degrees 2, 14 and 20 fit to 21 datapoints. (d) MSE vs degree.

Murphy, *"Probabilistic Machine Learning: An Introduction"*, 2022

Test Set

- Partition your data into 2 subsets: a training set D_{train} and a **test set** D_{test}
- **Approximation** of true error:

$$L(\boldsymbol{\theta}; D_{\text{test}}) = \frac{1}{|D_{\text{test}}|} \sum_{(\mathbf{x},y)\in D_{\text{test}}} l(y, f(\mathbf{x}; \boldsymbol{\theta}))$$

- If only the training set was used to evaluate models, the most complex one would always dominate → Evaluate based on **test set loss**
- Consider previous **polynomial regression** example
  - Multiple models being trained with varied polynomial degrees
  - Should we select the "best" one based on test set loss?

No Free Lunch Theorem

"All models are wrong, but some models are useful" – George E. P. Box

- No single best machine learning algorithm for all kinds of problems
- Assumptions (**inductive bias**) for one domain may not transfer to another
- Example: the simplest of two models is the preferred (**Occam's razor**)
- Pick a suitable model based on:
  - Domain knowledge
  - …and/or trial and error → Need *another* data subset for **model selection**

Test set is ONLY for model evaluation on unseen data and NOT model selection

Validation Set

- Split data into 3 disjoint sets: training D_{train}, test D_{test} and **validation** D_{valid}
- Often use splits like 60:20:20 or 50:25:25 if you have a lot of data
- **Validation/development/holdout** set is for **model selection**
- Select the model that performs "best", e.g., in terms of $L(\boldsymbol{\theta}; D_{\text{valid}})$, on the validation set from multiple models with a different:
  - Number of training epochs
  - Learning rate
  - Random seed value
  - Initial parameter configuration
  - …

Model Selection using the Validation Set

1. Identify a model (hyper)parameter setting to vary, *e.g.* learning rate α
2. Fit a different model per parameter setting, $\boldsymbol{\theta}_\alpha$, on the **training set** D_{train}, *e.g.* where the GD update step for NLL is:

$$\boldsymbol{\theta}_\alpha^{(t+1)} = \boldsymbol{\theta}_\alpha^{(t)} - \alpha \nabla \text{NLL}(\boldsymbol{\theta}_\alpha^{(t)}; D_{\text{train}})$$

3. For each $\alpha \in \mathcal{A}$ model, evaluate **validation set** performance, *e.g.* as NLL:

$$\text{NLL}(\boldsymbol{\theta}_\alpha^*; D_{\text{valid}}) = -\log p(\mathcal{D}_{\text{valid}} \,|\, \boldsymbol{\theta}_\alpha^*)$$

Optimal for model α

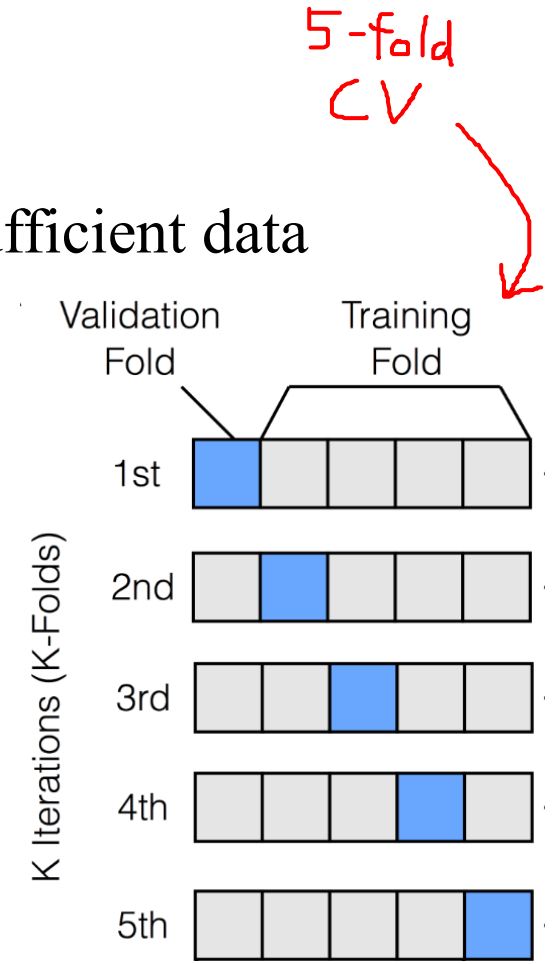
4. Select the model setting that results in the best validation performance:

$$\alpha^* = \arg\min_{\alpha \in \mathcal{A}} \text{NLL}(\boldsymbol{\theta}_\alpha^*; D_{\text{valid}})$$

5. After picking α^*, refit model to entire dataset $D = D_{\text{train}} \cup D_{\text{valid}}$ for $\boldsymbol{\theta}_{\alpha^*}^*$

K-fold Cross-validation – Key Idea

- Previous technique works well when a lot of data is available
- In **small data settings** though, the model **underfits** with insufficient data
- **Cross-validation (CV):** Resampling solution to this problem where different portions of data are used for training and testing at each iteration of the procedure
- **K-fold CV:** Split training data into K folds, $D_1, \dots, D_K$, such that $D_1 \cup \cdots \cup D_K = D$ and $D_i \cap D_j = \emptyset \; \forall \; (i, j)$, then train all folds but k^{th}, and test on k^{th} in **round-robin** manner



Source: https://androidkt.com/pytorch-k-fold-cross-validation-using-dataloader-and-sklearn/

K-fold Cross-validation – Notation

- Let $m \in M$ competing models each have parameters $\boldsymbol{\theta}_{\mathrm{m}}$ optimized according to some training objective $L_{\text{train}}(\boldsymbol{\theta}_m)$
- The "best" m^* is selected based on some validation objective $L_{\text{valid}}(\boldsymbol{\theta}_m)$
- Given a K-fold partition D_k, let $\boldsymbol{\theta}^*_{\mathrm{m},k}$ be the optimal parameters for model m trained **without** D_k:

$$\boldsymbol{\theta}^*_{m,k} = \arg\min_{\boldsymbol{\theta}_m} L_{\text{train}}(\boldsymbol{\theta}_m; D - D_k)$$

All data except D_k

- Validate $\boldsymbol{\theta}^*_{\mathrm{m},k}$ quality using the validation objective function:

Error ⟶ $\epsilon_{m,k} = L_{\text{valid}}(\boldsymbol{\theta}^*_{m,k}; D_k)$ ← Assessed on D_k

K-fold Cross-validation – Algorithm

Algorithm 1 K-fold Cross-validation to estimate (hyper)parameters

Partition D into $D_1, D_2, \ldots, D_K$ with equal partition sizes (or close)

for $m \in \{1, \ldots, M\}$ **do**

  for $k \in \{1, \ldots, K\}$ **do**

    $D_{\text{valid}} = D_k \; ; \; D_{\text{train}} = D - D_k$

    $\boldsymbol{\theta}^*_{m,k} = \arg\min_{\boldsymbol{\theta}_m} L_{\text{train}}(\boldsymbol{\theta}_m; D_{\text{train}})$

    $\epsilon_{m,k} = L_{\text{valid}}(\boldsymbol{\theta}^*_{m,k}; D_{\text{valid}})$

  end for

  $\epsilon_m = \frac{1}{K} \sum_{k=1}^{K} \epsilon_{m,k}$ ← Avg. CV error for model m

end for

Evaluate model m on each k fold of D and record error ϵ_{mk}

"Best" model has smallest average error: $m^* = \arg\min_{m \in \{1,\ldots,M\}} \epsilon_m$

Given m^*, train on entire dataset: $\boldsymbol{\theta}^*_{m^*} = \arg\min_{\boldsymbol{\theta}_{m^*}} L_{\text{train}}(\boldsymbol{\theta}_{m^*}; D)$

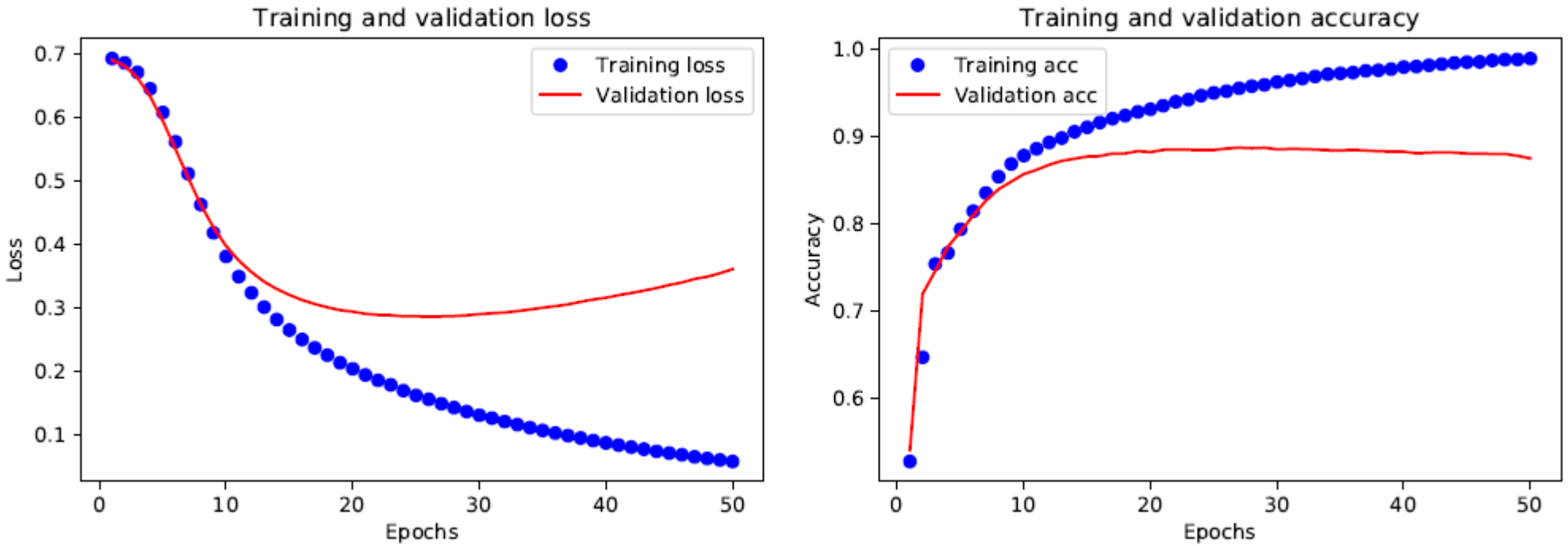
Final trained model

K-fold Cross-validation – Next Steps

- Estimate future performance on **independent** test set D_{test}
- Evaluate based on same measure as for CV: $L_{\text{valid}}(\boldsymbol{\theta}^*_{m^*}; D_{\text{test}})$
- Can also use K-fold CV on test set, known as **Nested CV**
  - Split D into K partitions, assign D_j for testing
  - Use $(K-1)$-fold CV on remaining $D - D_j$
  - Provides K estimates of test performance: $L_{\text{valid}}(\boldsymbol{\theta}^*_{m^*,j}; D_{\text{j}})$
  - Estimate **generalization error** by averaging K test set scores
  - **Unbiased estimate** of generalization performance

Can be v. slow
K*(k-1)*M iterations

Early Stopping



Iterative optimization allows us to monitor validation performance in parallel with training. **Stop early** at signs of overfitting.

Murphy, *"Probabilistic Machine Learning: An Introduction"*, 2022

Coding Break

Concluding Remarks

- **Training set** to fit models, **test set** to evaluate model predictive performance on future data, and **validation set** to select the best model configuration

- Code:

https://github.com/mazrk7/EECE5644_IntroMLPR_LectureCode/blob/main/notebooks/linear_regression/ls_polynomial_reg_cv.ipynb

- For a complete review on model selection:

https://arxiv.org/pdf/1811.12808.pdf

- Questions?