Model Selection, Model Evaluation

Machine Learning 1 UE (INP.33761UF)

Thomas Wedenig

May 15, 2024

Institute of Theoretical Computer Science
Graz University of Technology, Austria

Table of contents

The Problem – Overfitting, Underfitting

- We have data $\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(N)}, y^{(N)}\right) \right\}$
- We split it into **training data** $\mathcal{D}_{\text{train}}$ and **test data** $\mathcal{D}_{\text{test}}$
  - e.g., 80% train and 20% test
- Given a model $f_{\boldsymbol{\theta}}$, we compute the **average loss** over all **training** examples:

$$\mathcal{L}_{\text{train}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)}\right)$$

- We have data $\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(N)}, y^{(N)} \right) \right\}$
- We split it into **training data** $\mathcal{D}_{\text{train}}$ and **test data** $\mathcal{D}_{\text{test}}$
  - e.g., 80% train and 20% test
- Given a model $f_{\boldsymbol{\theta}}$, we compute the **average loss** over all **training** examples:

$$\mathcal{L}_{\text{train}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell \left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)} \right)$$

Question 🤔

Is the final goal of supervised learning to minimize the average training loss? That is, finding

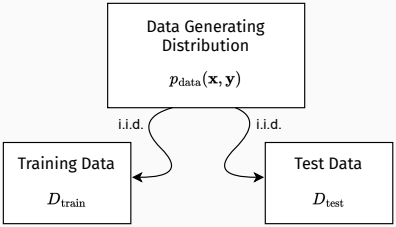
$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}_{\text{train}}(\boldsymbol{\theta})$$

The Goal of Supervised Learning

- No !
- There's an **unknown data generation distribution** $p_{\text{data}}(\mathbf{x}, y)$
- If we had access to $p_{\text{data}}(\mathbf{x}, y)$, we want to minimize the **generalization error**:

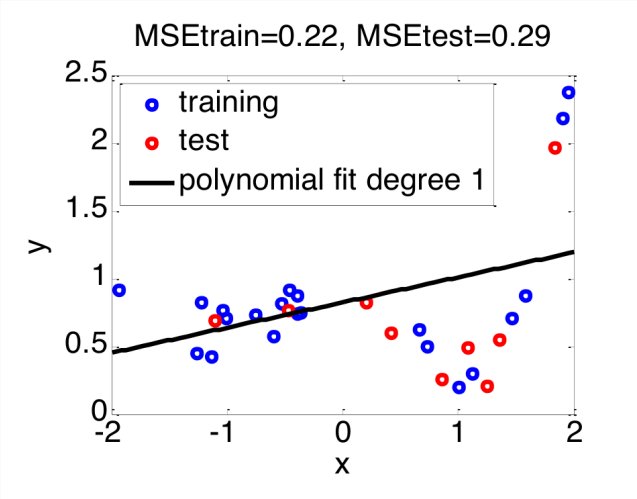
$$\mathcal{L}_{p_{\text{data}}}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}}\left[\ell(f_{\boldsymbol{\theta}}(\mathbf{x}), y)\right]$$

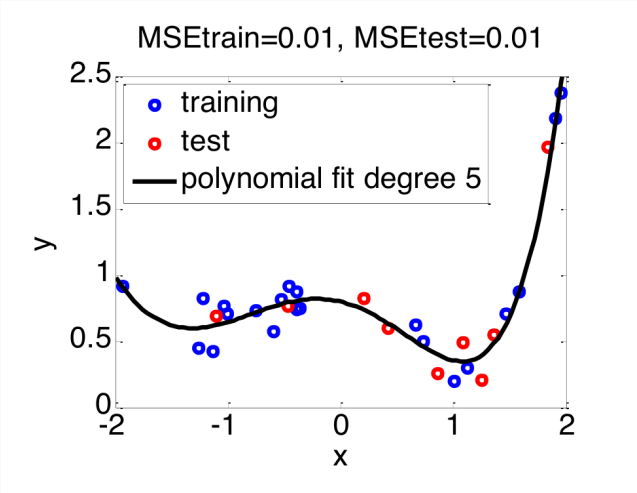
- But since we don't know p_{data}, we minimize $\mathcal{L}_{\text{train}}$
- However, we **truly care about the average test loss** $\mathcal{L}_{\text{test}}$, since this is an **unbiased estimate of the generalization error** !

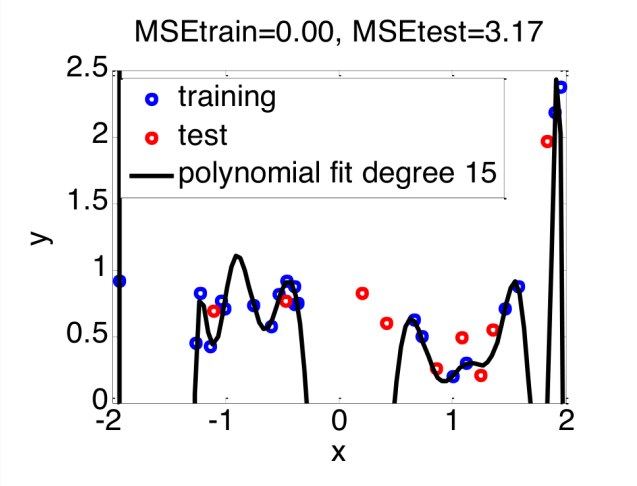


Overfitting – Example

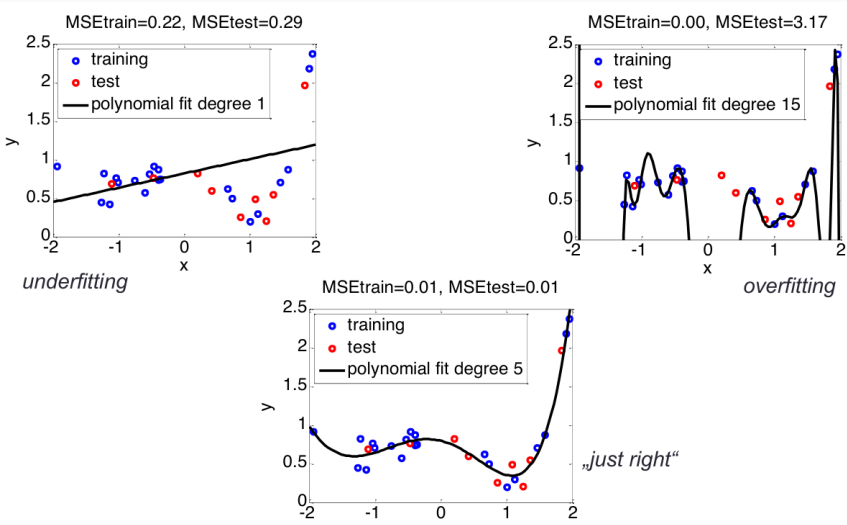
- Say we do **polynomial regression**, i.e., our targets $y^{(i)} \in \mathbb{R}$
- The **polynomial degree** of our model is a hyperparameter
- What's the **"right"** hyperparameter setting for our task ❓


MSEtrain=0.22, MSEtest=0.29
training
test
polynomial fit degree 1
2.5
2
1.5
1
0.5
0
-2
-1
0
1
2
y
x

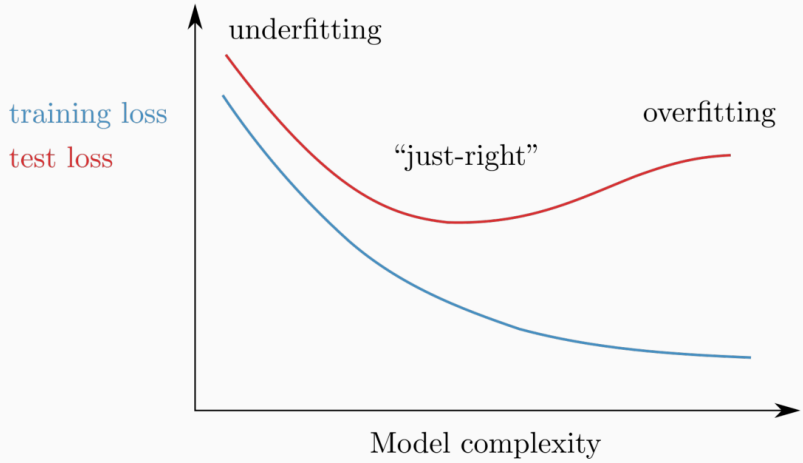

MSEtrain=0.01, MSEtest=0.01
training
test
polynomial fit degree 5
y
x
2.5
2
1.5
1
0.5
0
-2
-1
0
1
2


MSEtrain=0.00, MSEtest=3.17
training
test
polynomial fit degree 15
2.5
2
1.5
1
0.5
0
y
-2
-1
0
1
2
x

Underfitting and Overfitting

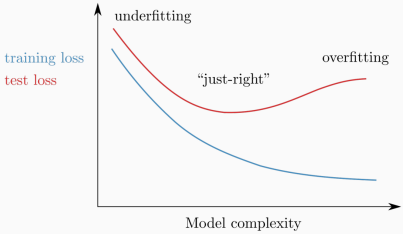


Underfitting and Overfitting



Underfitting and Overfitting

- Underfitting 😩
  - Model is **too simple**
  - e.g., Neural Net too small, polynomial degree too low
  - **Large training loss, large test loss**
- Overfitting 😩
  - Model is **too complex**
  - e.g., Neural Net too large, polynomial degree too high
  - **Small training loss, large test loss**
- "Just right" 😍
  - Model is neither **too complex** nor **too simple**
  - **Smallest test loss**



- **However**, selecting a model based on the **test set** is a **methodological issue** 😱
- **We can't touch the test set during model selection**
- We spare it until the very end to get an unbiased estimate of the **generalization error**
- But how should we decide which model to use then? 🤔

Model Selection

Test Set

- Let's **set aside** a chunk of our data, the **test set**
- For example, use 80% of the data as **training data** and 20% as **test set**
- **We do not touch the test set during model selection** ❗
  - For what's to come, forget that the test data exists at all

Holdout Method

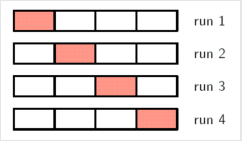
- Split remaning data into **train and validation set**
  - e.g., 80% for train and 20% for validation
- Use validation set to calculate **validation loss** for all model candidates
- Pick model candidate with **lowest validation loss**

Holdout Method

- Split remaning data into **train and validation set**
  - e.g., 80% for train and 20% for validation
- Use validation set to calculate **validation loss** for all model candidates
- Pick model candidate with **lowest validation loss**

K-Fold Cross Validation (CV)

- Split remaining data into K equally sized disjoint sets (**folds**)



- Train K separate models
  - Each model trains on $K - 1$ folds and is evaluated using the remaining fold
- Calculate emprical mean and variance of validation loss over K runs
  - Pick model with lowest mean CV loss

Holdout Method vs. Cross-Validation

Holdout Method

- Commonly used when
  - We have enough data
  - Model Training is relatively expensive
- Little training data $\rightarrow$ parameters are fitted poorly
- Little validation data $\rightarrow$ generalization error is poorly estimated

Holdout Method vs. Cross-Validation

Holdout Method

- Commonly used when
  - **We have enough data**
  - **Model Training is relatively expensive**
- **Little training data** $\rightarrow$ parameters are fitted poorly
- **Little validation data** $\rightarrow$ generalization error is poorly estimated

K-Fold Cross Validation (CV)

- Commonly used when
  - **We have little data**
  - **Model Training is relatively cheap**
- Allows you to **recycle the data**
- Needs K **times as much computation** as the Holdout Method...

(Some) Techniques against Overfitting

- Idea: use **flexible, complex model architectures**
  - e.g., large Neural Network, high degree polynomial
- This way, we **definitely won't underfit**
- But how to avoid **overfitting** ❓

Regularization

- Techniques against overfitting are often called **Regularization**
- Techniques include
  - **L1/L2 Regularization** (“penalty” term in the loss)
  - **Early Stopping** (used when training **Neural Networks**)

- In the **overfitting** regime, the **magnitude of parameter values often grows large**
- Thus, we prefer models with **smaller magnitude of parameter values**
- **L2 Regularization**: Instead of minimizing some loss $\mathcal{L}(\boldsymbol{\theta})$, we seek

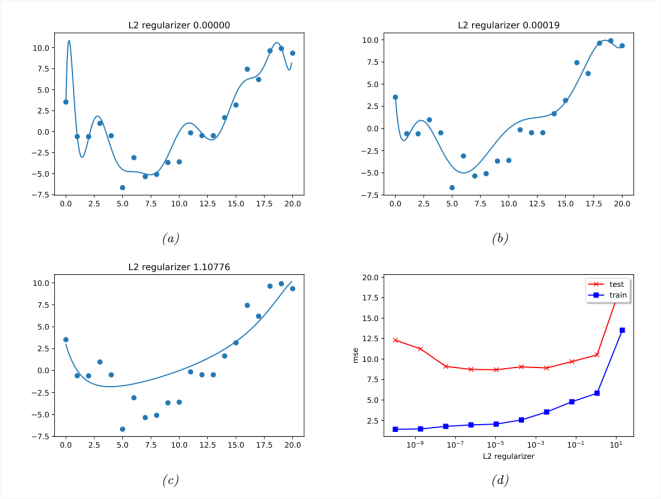
$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2$$

- **L1 Regularization**: Instead of minimizing some loss $\mathcal{L}(\boldsymbol{\theta})$, we seek

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1$$

- λ controls the **strength of regularization**

L2 Regularization – Example



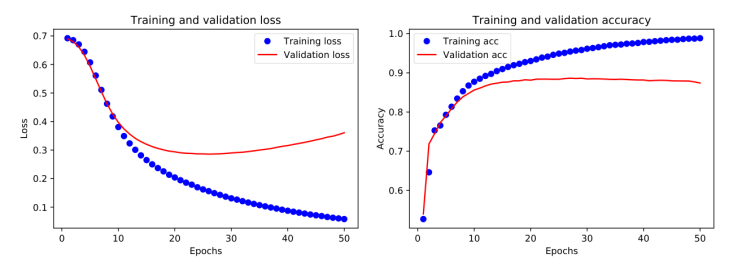
Probabilistic Machine Learning, An Introduction (Murphy, 2022)

Early Stopping

- Type of **regularization** used primarily in Neural Network training
  - Model needs to be trained in an **iterative manner**
  - e.g., Stochastic Gradient Descent (SGD), or similar
- During training, monitor **validation loss**
- **Stop training when validation loss increases**

Early Stopping

- Type of **regularization** used primarily in Neural Network training
  - Model needs to be trained in an **iterative manner**
  - e.g., Stochastic Gradient Descent (SGD), or similar
- During training, monitor **validation loss**
- **Stop training when validation loss increases**



Probabilistic Machine Learning, An Introduction (Murphy, 2022)

Early Stopping as Regularization

- Parameters $\boldsymbol{\theta}$ are initialized with **small values**
- SGD will take many steps to go away from our initialization
- Stop training if it goes “too far away”

Performance Metrics

- In our selection process, we pick the model with the **lowest validation loss**
  - Or lowest mean **CV loss**
- We can then – for the first time – use the test set to compute the **test loss**
- The loss is **usually not interpretable easily**
  - e.g., **Cross-Entropy Loss**
  - Loss might involve a **regularization term**
- How can we **evaluate the model's performance in a more interpretable way** ?

- **Regression:** $y^{(i)} \in \mathbb{R}$
  - **Note:** In the following, $\mathbf{x}^{(i)}, y^{(i)}$ come from the **test set**
- **Mean Squared Error** (MSE):

$$MSE(\boldsymbol{\theta}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

- **Root Mean Squared Error** (RMSE):

$$RMSE(\boldsymbol{\theta}) = \sqrt{MSE(\boldsymbol{\theta})}$$

- **Mean Absolute Error** (MAE):

$$MAE(\boldsymbol{\theta}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} | f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} |$$

- Prefer RMSE and MAE, since they have the **same unit as the target**

- **Binary classification:** $y^{(i)} \in \{0, 1\}$. A model's prediction can be:
  - **True Positive** (TP): $y = 1$ and model correctly said $\hat{y} = 1$
  - **True Negative** (TN): $y = 0$ and model correctly said $\hat{y} = 0$
  - **False Positive** (FP): $y = 0$ but model incorrectly said $\hat{y} = 1$
  - **False Negative** (FN): $y = 1$ but model incorrectly said $\hat{y} = 0$

- **Binary classification:** $y^{(i)} \in \{0, 1\}$. A model's prediction can be:
  - **True Positive** (TP): $y = 1$ and model correctly said $\hat{y} = 1$
  - **True Negative** (TN): $y = 0$ and model correctly said $\hat{y} = 0$
  - **False Positive** (FP): $y = 0$ but model incorrectly said $\hat{y} = 1$
  - **False Negative** (FN): $y = 1$ but model incorrectly said $\hat{y} = 0$
- Say the test data has P positive and N negative examples: $N_{\text{test}} = P + N$
- **Accuracy:**
$$ACC = \frac{TP + TN}{N_{\text{test}}}$$
- **Precision:**
$$PREC = \frac{TP}{TP + FP}$$
- **Recall:**
$$REC = \frac{TP}{P}$$

- F_1 **Score**:

$$F_1 = 2 \cdot \frac{PREC \cdot REC}{PREC + REC}$$

- F_1 is the **harmonic mean** of **precision** and **recall**
  - Balances precision and recall
- Accuracy, Precision, Recall, F_1 are all $\in [0, 1]$

Question 🤔

You want to build a binary classification model that predicts if a patient has a rare disease. In your test set, 99% of patients are healthy.

What's the **test accuracy** of a naïve model that **always predicts** $\hat{y} = 0$ (no disease) ❓

Accuracy as a Performance Metric

Question 🤔

You want to build a binary classification model that predicts if a patient has a rare disease. In your test set, 99% of patients are healthy.

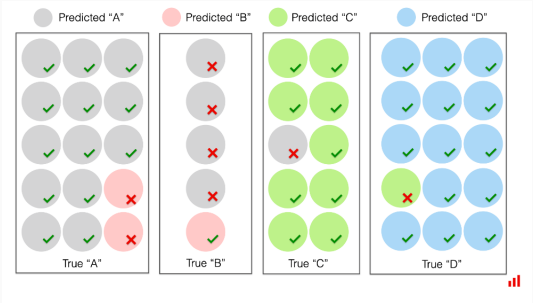
What's the **test accuracy** of a naïve model that **always predicts** $\hat{y} = 0$ (no disease) ❓

Answer

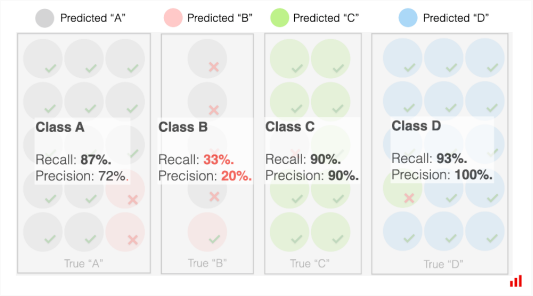
- **Test accuracy is** 99% ❗
- Accuracy **is very misleading if we have class inbalances**
- Better: Report **Precision, Recall** and F_1

What about **multiclass** classification?

- Report metric **per class**
  - e.g., Accuracy, Precision, Recall, F_1



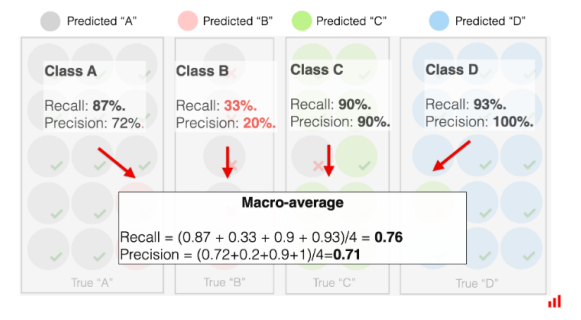
https://www.evidentlyai.com/classification-metrics/multi-class-metrics



https://www.evidentlyai.com/classification-metrics/multi-class-metrics

Multiclass Classification (cont.)

- We can also **average** the individual per-class metric (**macro** average)
  - Gives equal weight to each class



https://www.evidentlyai.com/classification-metrics/multi-class-metrics

- We can also globally count true predictions and false predictions (**micro** average)
  - Gives a global type of **accuracy**

Model Selection & Evaluation Demo (`MLPClassifier`)