

Foundations of Model Evaluation and Overfitting

Classification Error

1 Definition

- The classification error is the **average misclassification rate**:

$$E_{in} = \frac{\text{Number of misclassified points}}{\text{Total number of points}}.$$

- It tells us how many samples the model predicted incorrectly, on average.
-

2 General formula

The image writes the formula like this:

$$E_{in} = \frac{1}{N} \sum_{i=1}^N I(\hat{y}_i \neq y_i),$$

Where:

- N: total number of samples.
- \hat{y}_i : predicted label for sample ii.
- y_i : actual label for sample ii.
- $I(\hat{y}_i \neq y_i)$: an **indicator function** that equals:
 - 1 if $\hat{y}_i \neq y_i \rightarrow$ the model made an error.
 - 0 if $\hat{y}_i = y_i \rightarrow$ the model predicted correctly.

So this formula counts how many times the prediction is wrong and divides by the total number of samples.

Example — Bank Locker Fingerprint ID

This is a practical example:

- The classifier predicts whether the fingerprint matches the owner's.
- So for each input sample x , the indicator function $I(x)$ is:

$$I(x) = \begin{cases} 1 & \text{if } \hat{y} \neq y \quad (\text{prediction is wrong}) \\ 0 & \text{if } \hat{y} = y \quad (\text{prediction is correct}) \end{cases}$$

This gives a simple way to compute how often the fingerprint system fails.

Takeaways

1. **Classification error is intuitive** — it's just the fraction of incorrect predictions.
 2. **Indicator function is important** — it formalizes the logic:
 - ✓ If wrong → count as 1 error
 - ✓ If correct → count as 0
 3. **Works for any classification problem** — from spam filters to fingerprint recognition.
 4. **Training vs. test error** — you compute this same way, but on different datasets depending on what you want to measure.
-

Summary

- The classification error measures how often a model's predictions are wrong.
 - It's calculated by summing up the number of mistakes and dividing by the total points.
 - The indicator function $I(\hat{y}_i \neq y_i)$ counts mistakes as 1 and correct predictions as 0.
 - This approach applies broadly, including real-life cases like verifying fingerprints.
-

Overfitting in Classification

Definition

Overfitting happens when a classification model learns not only the true underlying patterns in the training data but also the **random noise or small fluctuations** that are specific to that dataset. As a result:

- ✓ The model performs **very well on training data**
 - ✗ But performs **poorly on unseen test data** — it fails to generalize.
-

Why does overfitting happen?

- The model is **too flexible**, meaning it has many parameters or can form very complex decision boundaries.
 - It tries to perfectly classify all training examples, including noise or outliers.
 - As a result, it loses the ability to classify new data points correctly.
-

Example — Classification overfitting

Imagine you are training a model to distinguish cats vs dogs:

- A simple model might create a smooth boundary between the two classes.
- A highly flexible model might create a boundary that loops around every single training point to perfectly separate them → this is overfitting.

On new images, this model would likely fail because it learned the quirks of the training set rather than general characteristics.

How overfitting shows up

1. **Training error is very low** — the model memorizes the data.
 2. **Test error is high** — the model doesn't generalize well.
 3. Complex decision boundaries with sharp twists and turns.
 4. The model may react strongly to small changes in input features.
-

Why overfitting is a problem

- Poor real-world performance — the model cannot handle data variations.
 - Wasted computational resources — complexity doesn't translate into accuracy.
 - It's especially risky with small datasets or noisy data.
-

How to detect overfitting

- Compare **training error vs test error**.
 - If training error is much lower than test error → likely overfitting.
 - Use **cross-validation** to see how performance generalizes across data splits.
-

How to prevent or reduce overfitting

1. Simplify the model

- Reduce number of features or parameters.

2. Regularization

- Techniques like Ridge or Lasso add penalties for large weights.

3. Collect more data

- A larger dataset helps the model learn true patterns.

4. Early stopping

- Stop training before the model starts fitting noise.

5. Cross-validation

- Evaluate on different data splits to ensure generalization.

6. Data augmentation

- Introduce variability to make the model more robust.

7. Pruning (for decision trees)

- Remove parts of the model that don't add predictive power.
-

Key takeaway:

Overfitting is a tradeoff between bias and variance — too much flexibility causes high variance, which hurts generalization. The goal is to build models that learn patterns but avoid memorizing noise.

Overfitting with Polynomials

What's happening?

In polynomial regression or classification, you try to fit a curve (or boundary) to your data points using polynomial terms like X , X^2 , X^3 , etc.

- **Low-degree polynomial** → simple curve, smooth boundary.
- **High-degree polynomial** → complex curve with many bends and twists → it can pass through every training point.

When the polynomial degree is too high, the model tries to perfectly fit the training data, even the noise or outliers → **overfitting**.

Example: 1D polynomial fit

Imagine you are trying to classify points along the x-axis.

- With a **linear model (X)**, the boundary is a straight line → might miss some subtle patterns but generalizes reasonably.
 - With a **cubic model (X^3)**, the boundary can curve more → capture some nonlinearities without overfitting much.
 - With a **degree-10 polynomial**, the boundary wiggles dramatically → fits every point exactly → captures noise → overfits.
-

Key insights

1. **Training error decreases** as the polynomial degree increases → it fits the training data better and better.
2. **Test error first decreases, then increases** →

- At first, the model learns useful patterns → better predictions.
 - After a point, the model starts memorizing the noise → poor generalization.
3. This is a perfect example of the **bias–variance tradeoff**:
- Low-degree polynomial → high bias, underfitting.
 - High-degree polynomial → high variance, overfitting.
-

Visual intuition

- The training curve starts to twist and bend sharply as the polynomial degree grows.
 - It passes through all training points but looks unnatural.
 - On new data, it performs poorly because it expects the same noise patterns.
-

Why does overfitting happen with polynomials?

- Higher-degree terms allow more flexibility → more parameters → easier to “memorize” the data.
 - Without constraints or regularization, the model interprets noise as signal.
-

How to address polynomial overfitting

1. **Choose the right degree**
 - Use validation or cross-validation to pick a degree that generalizes well.
 2. **Use regularization**
 - Penalize large polynomial coefficients (e.g. Ridge, Lasso).
 3. **Collect more data**
 - Reduces variance and helps the model learn the true relationship.
 4. **Early stopping or pruning**
 - Limit complexity where unnecessary.
-

✓ Takeaways

- Overfitting with polynomials shows how adding complexity can lead to memorizing noise instead of learning patterns.
- There's a sweet spot: not too simple (underfit), not too complex (overfit).
- Always test on unseen data to ensure the model generalizes.
- Regularization, cross-validation, and data collection help prevent overfitting.

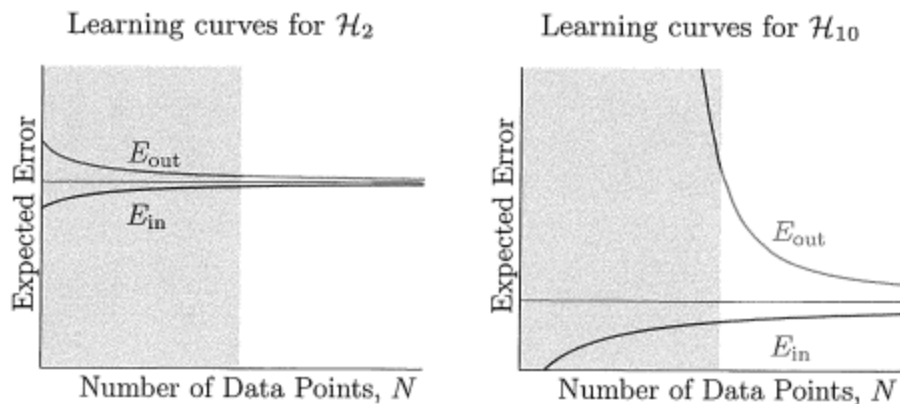


Figure 4.2: Overfitting is occurring for N in the shaded gray region because by choosing \mathcal{H}_{10} which has *better* E_{in} , you get *worse* E_{out} .

Types of Noise

1 Stochastic Noise (Random Noise)

📖 Definition

Stochastic noise is the **random variability in the data** that cannot be explained by the predictors (features). It's the uncertainty or error inherent in the process of data generation.

- It's caused by factors outside the model's control.
- It's unpredictable and cannot be reduced, only accounted for.
- Often referred to as **irreducible error**.

Example

You're trying to predict a person's weight based on their height and age.

- Even if you had perfect data and model structure, factors like daily water intake, temporary health conditions, or measurement inaccuracies will cause variations → stochastic noise.

Characteristics

- ✓ Random and unpredictable
 - ✓ Always present
 - ✓ It can't be removed by better models
 - ✓ You can only try to minimize its impact statistically
-

2 Deterministic Noise (Systematic Noise)

Definition

Deterministic noise arises when the **true relationship between predictors and response cannot be perfectly captured by the chosen model**. It's a systematic error caused by model limitations.

- It's the error due to **model misspecification**.
- If the model is too simple (underfitting), it can't explain the underlying patterns → deterministic noise increases.
- If you use a more flexible or appropriate model, you can reduce deterministic noise.

Example

You're trying to predict the same weight data using a **linear model** while the true relationship is quadratic.

- The model's inability to capture the true pattern leads to deterministic noise → it's not random but due to the model's structure.

Characteristics

- ✓ Caused by model limitations
- ✓ It can be reduced by choosing a better or more complex model
- ✓ Not random — systematic error in prediction
- ✓ Related to bias and underfitting

✓ Key Differences

Feature	Stochastic Noise	Deterministic Noise
Source	Random, external factors	Model inadequacy or simplification
Predictability	Unpredictable	Systematic and predictable (given model choice)
Reducible?	No	Yes (by improving the model)
Related to	Irreducible error	Bias, underfitting
Example	Measurement error, natural variations	Using a linear model for a nonlinear relationship

✓ Why this matters

- **Stochastic noise** sets a lower limit on how good any model can be — even with perfect data and structure, some error will remain.
- **Deterministic noise** is what we often try to minimize by choosing better models, using more features, or applying regularization techniques.

Understanding both helps you diagnose whether poor performance is due to **data limitations** (stochastic noise) or **model limitations** (deterministic noise).

Bias–variance decomposition of the out-of-sample error (E_{out})

1 Without stochastic noise

$$E_{out} = bias + variance$$

This means the out-of-sample error is made up of:

- **Bias** → how much the model's average prediction deviates from the true function.
- **Variance** → how much the model's predictions fluctuate with different training datasets.

This version assumes that there's **no randomness or noise** in the data.

2 With stochastic noise

$$E_{out} = bias + variance + noise$$

Now we add **stochastic noise**, which represents random errors that cannot be explained by the model or data structure.

3 The full mathematical decomposition

$$E_{out} = \mathbb{E}_{D,X} [(g_D(x) - \bar{g}(x))^2] + \mathbb{E}_X [(\bar{g}(x) - f(x))^2] + \mathbb{E}_{\epsilon,X} [(\epsilon(x))^2]$$

Breaking it down:

✓ Variance term:

$$\mathbb{E}_{D,X} [(g_D(x) - \bar{g}(x))^2]$$

- Measures how much the model's predictions $g_D(x)$ vary when trained on different datasets.
 - It's due to sensitivity to training data.
-

✓ Bias term:

$$\mathbb{E}_X [(\bar{g}(x) - f(x))^2]$$

- Measures how far the model's average prediction $\bar{g}(x)$ is from the true function $f(x)$.
 - It's caused by model assumptions or simplifications.
-

✓ Noise term:

$$\mathbb{E}_{\epsilon, X} [(\epsilon(x))^2]$$

- Represents stochastic (random) noise in the data, denoted by $\epsilon(x)$.
 - It's inherent variability that can't be removed by the model.
-

✓ Deterministic vs Stochastic Noise

✓ **Deterministic noise** → comes from model limitations and misspecification → bias.

✓ **Stochastic noise** → random fluctuations or measurement errors → noise.

✓ Why is this important

1. Helps diagnose errors

You can see whether your model is overfitting (high variance), underfitting (high bias), or if randomness is unavoidable (stochastic noise).

2. Guides model improvement

- If bias is high → try a more complex model.
 - If variance is high → collect more data or regularize the model.
 - If stochastic noise is high → data collection or measurement methods may need improvement, but you can't reduce it through modeling alone.
-

✓ Final Takeaway

This image summarizes how total prediction error arises from three sources:

1. **Bias** → due to assumptions or simplifications in the model.
2. **Variance** → due to fluctuations from dataset changes.
3. **Noise** → due to randomness that cannot be predicted.

Understanding this helps you interpret learning curves, adjust models, and manage expectations about performance.

How do we deal with deterministic noise

1 Validation (through resampling) – stick to the bottom line

- **Purpose:** Helps ensure that the model you choose generalizes well on unseen data, rather than just fitting the training data.
 - **Key points:**
 - **Give priority to E_{out} over E_{in} :**

Even if your model performs well on training data (E_{in}), that doesn't guarantee it will perform well on new data (E_{out}).
 - **You've already experimented with this in choosing the fit based on the dataset size N:**

You've likely tuned models or hyperparameters during training, so validation helps confirm which choice generalizes better.
 - **There's more!**

Validation not only confirms performance but also helps in choosing the best among different models.
-

2 Regularization – putting breaks

Regularization helps reduce overfitting by controlling model complexity.

✓ Subset selection:

Select only the most important features instead of using all of them.

✓ Shrinkage (penalization):

Allow all parameters but limit their magnitudes to avoid over-reliance on specific features.

✓ Dimension reduction (like PCA):

Reduce the number of features while retaining the most important information — helps in removing irrelevant noise.

Why validation or estimate E_{out} ?

1 Applies to all problems

Validation isn't limited to specific cases — it's necessary in almost every machine learning problem.

2 Improve a given model's performance

- Once you have a model, you want to choose the best version or hyperparameter setting among many possibilities.
- This process is called **model selection** → helps you decide which hypothesis or algorithm yields the lowest out-of-sample error.

3 Model selection

- Validation helps in choosing between models — whether it's linear regression, KNN, or others.

4 K-fold cross-validation

- A method where the data is split into K parts.
- Each part is used as a validation set while the others are used for training.
- Note: The "K" here is different from the K in KNN, so don't confuse them!

✓ Overall Summary

These slides explain how deterministic noise (bias due to model limitations) can be addressed by:

1. **Validation** → Testing the model's performance on new data to ensure it generalizes.
2. **Regularization** → Controlling complexity by limiting feature usage or weights.

Additionally, validation plays a crucial role in **model selection**, helping you pick the best model configuration using techniques like **K-fold cross-validation**.

✓ What is K-fold Cross-Validation?

K-fold cross-validation is a technique used to evaluate how well a model generalizes to unseen data. It helps you estimate the **out-of-sample error** (E_{out}) without needing a separate test set or wasting data.

How it works:

1. The data is split into **K equally sized folds** (or parts).
 2. For each fold:
 - That fold is used as the **validation set**.
 - The remaining $K - 1$ folds are used as the **training set**.
 3. The model is trained on the training set and evaluated on the validation set.
 4. This process is repeated **K times**, each time with a different fold as the validation set.
 5. The final error estimate is the **average** of the validation errors across all folds.
-

✓ Step-by-step Example

Let's say you have **100 samples** and you choose **K = 5**.

1. Split data into 5 folds:

- Fold 1 → samples 1–20
- Fold 2 → samples 21–40
- Fold 3 → samples 41–60
- Fold 4 → samples 61–80
- Fold 5 → samples 81–100

2. First iteration:

- Train on folds 2, 3, 4, 5 → 80 samples
- Validate on fold 1 → 20 samples

3. Second iteration:

- Train on folds 1, 3, 4, 5
- Validate on fold 2

...and so on until each fold has been used as validation once.

4. Compute error:

For each iteration, calculate the validation error (e.g., classification error, RMSE).

Finally, averaging these errors → gives the **cross-validation error estimate**.

✓ Why use K-fold Cross-Validation?

- ✓ It helps **use all data efficiently** → each sample is used for both training and validation.
- ✓ It provides a **better estimate of model performance** → reduces bias from random data splits.
- ✓ It helps in **model selection and hyperparameter tuning** → you can compare models reliably.

✓ Common choices for K

K	Use case
5	Often a good balance between bias and variance
10	Commonly used in practice
N	Leave-one-out cross-validation → used for small datasets

✓ Example: Using K-fold in Python (scikit-learn)

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression

# Generate synthetic data
```

```
X, y = make_regression(n_samples=100, n_features=5, noise=10)

# Initialize model
model = LinearRegression()

# 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation
scores = cross_val_score(model, X, y, cv=kf, scoring='neg_mean_squared_error')

print("Cross-validation MSE scores:", -scores)
print("Average cross-validation MSE:", -scores.mean())
```

✓ Important Notes

- ✓ **K** should be chosen based on the dataset size and computational cost.
- ✓ Always shuffle the data before splitting to avoid bias.
- ✓ Cross-validation gives a more robust performance measure than a single train-test split.
- ✓ It's widely used in hyperparameter tuning, especially in grid search or random search.

What is Bootstrap?

Bootstrap is a method used to estimate statistics (like the mean, variance, or model parameters) by sampling from the original dataset **with replacement**. It helps measure uncertainty in estimates when you have limited data.

The idea is:

1. Create multiple new datasets by sampling from the original data.
2. Fit a model or calculate a statistic on each sample.

3. Aggregate the results to understand variability, confidence intervals, etc.

Original Data (Z)

This table is the **original dataset**, with 3 observations (Obs) and columns X and Y.

Obs	X	Y
1	4.3	2.4
2	2.1	1.1
3	5.3	2.8

This dataset is used to create new samples.

Bootstrap Samples (Z^1, Z^2, \dots, Z^{*B})

- Each new sample is drawn **with replacement** from the original data.
- That means some rows may appear more than once, while others may not appear at all.

Example bootstrap samples:

Z^{*1}

Obs	X	Y
3	5.3	2.8
1	4.3	2.4
3	5.3	2.8

- Row 3 is selected twice.

Z^{*2}

Obs	X	Y
2	2.1	1.1
3	5.3	2.8
1	4.3	2.4

- All rows were selected once, but in a different order.

Z^{*B}

Obs	X	Y
2	2.1	1.1
2	2.1	1.1
1	4.3	2.4

- Row 2 is selected twice.

► Model Fitting:

For each bootstrap sample, a model is fitted. The result is a set of parameter estimates:

- From Z^{*1} , you get $\hat{\alpha}^{*1}$.
- From Z^{*2} , you get $\hat{\alpha}^{*2}$.
- From Z^{*B} , you get $\hat{\alpha}^{*B}$.

These estimates are then aggregated to find the average, variance, and confidence intervals.

✓ Key Points

✓ **Sampling with replacement** → Some data points appear multiple times, others not at all.

✓ **Useful for small datasets** → Bootstrap simulates variability without needing more data.

✓ **Multiple bootstrap samples** → Each provides a slightly different version of the dataset to explore uncertainty.

✓ Applications:

- Estimating standard errors
- Constructing confidence intervals
- Model validation
- Bias correction

✓ Why is this important?

Bootstrap is widely used because:

- It's simple and intuitive.
 - It doesn't rely on strong assumptions about data distribution.
 - It provides reliable estimates of uncertainty, especially when theoretical formulas are complex or unknown.
-