# Evaluating Model Performance

# Section 8.1 Model Error

**Bias**

**Bias** measures how much the model’s predictions systematically differ from the actual values due to simplifying assumptions.

**High bias** occurs when a model makes strong assumptions (like assuming linearity in a nonlinear relationship) and consistently mispredicts because it cannot capture the complexity of the data.

**Low Bias** indicates that the model can accurately represent the data's underlying patterns without overly simplifying. A low-bias model typically fits the data well by capturing essential relationships, though it may become complex in the process.

In essence, high bias means the model is "stuck" with a rigid, oversimplified view, causing it to make similar types of errors regardless of the dataset. Low bias models, on the other hand, adapt more flexibly to the data's true patterns, reducing systematic prediction errors.

**Variance**

**Variance** measures how sensitive the model is to fluctuations in the training data. **High variance** means the model captures the details and noise in a specific dataset, so it performs well on that dataset but changes its predictions significantly when presented with new or slightly different data.

**Low Variance** means that the model’s predictions remain stable across different datasets from the **same population**. This stability is often due to a simpler model that generalizes better, rather than memorizing specific patterns or outliers in the training data.

In other words, a high-variance model has predictions that "jump around" significantly on different data samples, while a low-variance model keeps its predictions consistent and reliable across datasets.

**Underfitting**

A graph with a red line and green line

Description automatically generated

**High Bias**: Comes from the model being too simple, leading to consistent mispredictions because it can't capture the true complexity of the data.

**Low Variance**: Results because a simple model (like a straight line) doesn’t change much, even with new or slightly different data; it remains stable with minimal fluctuation.

**Overfitting**

A graph with red line and blue x

Description automatically generated

**Low Bias**: The model accurately represents the training data’s specifics, closely following the points.

**High Variance**: This model is highly sensitive to small changes, meaning it would likely perform poorly on new data (**from the same population**), as it has essentially memorized the training points rather than generalizing a pattern.

# Section 8.2 Training Validation, and Testing

In machine learning, datasets are typically split into three or two distinct subsets:

* **training**, **validation**, and **test sets** or
* **training and test sets.**

**Example:**

**Training set**: Start by using 70% of the dataset to train several models, each with different algorithm. Fit multiple models (e.g., model 1, model 2, and model 3) to capture different patterns and complexities in the data.

**Validation set**: Setting aside 15% of the data as validation set to evaluate each trained model. Use performance metrics like accuracy to compare the models. Select the model with the highest performance (e.g., accuracy) on validation set.

**Test set**: The remaining 15% of the dataset is used for a final evaluation. The final test ensures the model performs well on completely unseen data and is ready for real world deployment.

Code:

from sklearn.model\_selection import **train\_test\_split**

import numpy as np

# Sample data

X = np.arange(20).reshape((10, 2)) # 10 samples, 2 features

# Split data without a target variable

X\_train, X\_test = **train\_test\_split** (X, test\_size=0.3, random\_state=42)

from sklearn.model\_selection import train\_test\_split

import numpy as np

# Sample data

X = np.arange(20).reshape((10, 2)) # 10 samples, 2 features

y = np.array([0, 1] \* 5) # 10 samples, binary target (0 or 1)

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

## Section 8.3

MSE = mean\_squared\_error (y, yPrediction)

RMSE = mean\_squared\_error (y, yPrediction, **squared=False**)

MAE = mean\_absolute\_error(y, yPrediction)

## Section 8.4:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a math problem

Description automatically generated

Overall absolute loss:

To calculate the absolute loss for the entire dataset, take the average of the absolute loss values across all instances.

A screenshot of a computer

Description automatically generated

A white paper with black text and numbers

Description automatically generated

A white paper with black text

Description automatically generated

A white text with black text

Description automatically generated

## 8.5 Binary Classification Metrics

A screenshot of a test

Description automatically generated

A screenshot of a test

Description automatically generated



A white text on a white background

Description automatically generated



A white text on a white background

Description automatically generated

A white text on a white background

Description automatically generated

**Code implementation**

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import accuracy\_score

* score = recall\_score (y, yPrediction)
* score = accuracy\_score (y, yPrediction)
* score = precision\_score (y, yPrediction)

ROC curve:

X-axis: a high FPR means that **a high proportion of actual negatives are incorrectly classified as positive**. Specifically, FPR is calculated as: FPR=FPTN+FP\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}}FPR=TN+FPFP​ So, a high FPR implies that the model has difficulty distinguishing between positive and negative cases, often incorrectly labeling negatives as positives.

Y-axis: A high TPR (or recall) means the model is **effectively identifying the true positive cases**. It’s finding a large proportion of the actual positives, which is desirable when detecting important events (like diseases or fraud).

Ideal case: The ideal point on the ROC curve would be the top-left corner (FPR = 0, TPR = 1), where the model has no false positives and identifies all true positives. In other words, a low FPR and a high TPR represent a model that’s effective at distinguishing between positive and negative cases.

Summary: A high FPR indicates the model is frequently mistaking negatives for positives, while a high TPR means it’s good at identifying positives. An ideal model would achieve a high TPR with a low FPR, which would place it toward the top-left corner of the ROC space. Remember that FPR is specifically about the proportion of actual negatives misclassified as positives.

8.6 Cross Validation:

**1. Introduction to Cross-Validation**

Cross-validation is a technique used to assess how well a model will generalize to unseen data by splitting a dataset into multiple parts for training and validation. This approach provides a more reliable estimate of model performance than a single train-test split, especially when data is limited.

Cross-validation methods can be divided into two main types:

* **Non-Exhaustive Cross-Validation**: Methods that do not explore every possible way of splitting the dataset, such as K-Fold Cross-Validation.
* **Exhaustive Cross-Validation**: Methods that cover all possible splits of the data, like Leave-One-Out Cross-Validation (LOOCV), where each data point is used as the validation set once.

**2. K-Fold Cross-Validation (Non-Exhaustive) Explained with Example (K = 5)**

Let’s walk through **5-Fold Cross-Validation** with a dataset of 100 data points. Here, we divide the data into 5 equal parts, or folds, with each fold containing 20 data points.

1. **Divide the Data into 5 Folds**:
   * With 100 data points and K=5, each fold contains 20 points.
2. **Perform 5 Iterations**:
   * **Iteration 1**: Use Fold 1 (20 points) as the validation set, and Folds 2–5 (80 points) as the training set.
   * **Iteration 2**: Use Fold 2 as the validation set, and Folds 1, 3, 4, and 5 as the training set.
   * **Iteration 3**: Use Fold 3 as the validation set, and Folds 1, 2, 4, and 5 as the training set.
   * **Iteration 4**: Use Fold 4 as the validation set, and Folds 1, 2, 3, and 5 as the training set.
   * **Iteration 5**: Use Fold 5 as the validation set, and Folds 1–4 as the training set.
3. **Calculate the Average Performance**:
   * For each fold, we record a performance metric (e.g., accuracy or mean squared error) on the validation set.
   * The average performance across all 5 folds represents the model’s overall performance on unseen data.

A screenshot of a test

Description automatically generated

A white paper with black text

Description automatically generated

Note:

A close-up of a test

Description automatically generated

### 4. **Choosing K in K-Fold Cross-Validation**

Choosing the number of folds K affects the balance between bias and variance in the model’s performance estimate:

* **Larger K (e.g., approaching LOOCV)**:
  + **Lower Bias**: With a larger K, each fold used for validation is smaller, making the training set nearly the size of the full dataset. This results in a model with low bias, as it has more data to learn from in each iteration.
  + **Higher Variance**: A larger K means each validation set contains fewer data points, making the model’s performance more sensitive to each specific fold. This leads to higher variability in the error estimates across folds.
  + **Risk of Overfitting**: With a large training set in each iteration, the model may closely fit the data (including noise), which can lead to overfitting.
* **Smaller K** 
  + **Higher Bias:** As the validation set grows, the amount of data left for training decreases. With fewer training examples, the model has less information to learn from, which can prevent it from capturing the full complexity of the patterns in the data
  + **Low Variance:** A large validation set makes each fold’s performance estimate more representative of the overall data. With more validation examples, each fold provides a more stable, consistent performance estimate, reducing the variance in performance across folds.
  + **Risk of Underfitting.**

In summary, **K-Fold Cross-Validation** (commonly with K=5 or 10) is a practical and efficient method for model evaluation. While **LOOCV** provides an exhaustive evaluation, it is generally reserved for small datasets due to its high computational demands.

Code: train\_test\_split(pines, test\_size = 0.15, random\_state=rng)

-cross\_val\_score(quadPolyRegModel, X, y, scoring='neg\_mean\_squared\_error', cv=10)

-cross\_val\_score(quadPolyRegModel, X, y, scoring='neg\_mean\_squared\_error', cv=len(X))

## 8.7 Bootstrapping:

**Objective**

To understand the two primary goals of bootstrapping:

1. **Estimating the Sampling Distribution** of a sample statistic, such as the mean or proportion, to make inferences about a population.
2. **Evaluating Model Performance** by estimating the distribution of error metrics, providing insights into how well a model generalizes to new data.

**Part 1: Estimating the Sampling Distribution of a Sample Statistic**

**Goal:**

The main objective of this application of bootstrapping is to approximate the **sampling distribution** of a sample statistic (like the sample mean or sample proportion) when we have only one dataset. This is particularly useful for understanding the variability in the sample statistic and making inferences about the population.

**Process of Estimating the Sampling Distribution with Bootstrapping:**

1. **Original Sample**:
   * Start with a single sample dataset, which provides an initial estimate of the population parameter
2. **Generate Bootstrap Samples**:
   * Draw multiple "bootstrap samples" by randomly sampling **with replacement** from the original dataset. Each bootstrap sample is typically the same size as the original dataset.
   * This resampling process may result in some data points appearing **multiple times** in a single bootstrap sample, while others may not appear at all.
3. **Calculate the Sample Statistic for Each Bootstrap Sample**:
   * For each bootstrap sample, calculate the statistic of interest (e.g., sample mean or proportion).
   * Repeat this process across all bootstrap samples, generating a distribution of the statistic (e.g., multiple sample means).
4. **Construct the Approximate Sampling Distribution**:
   * The collection of sample means (or proportions) from all bootstrap samples forms an **approximate sampling distribution** of the sample statistic.
   * This distribution can be used to estimate the **variability** of the statistic and to construct **confidence intervals**.

**Example:**

Suppose we have a dataset of 50 test scores with a sample mean of 96.6. We want to understand how this sample mean might vary if we repeatedly took samples from the population.

1. Draw multiple bootstrap samples from the original data.
2. Calculate the sample mean for each bootstrap sample.
3. The distribution of these bootstrap means provides an estimate of the **sampling distribution of the sample mean**. From this distribution, we can create a confidence interval to capture the likely range of the true population mean.

A screenshot of a graph

Description automatically generated

Once we have the approximate sampling distribution of the sample mean from bootstrapping, we can indeed use it to calculate a **confidence interval around the sample mean**. This confidence interval provides a range in which we believe the **true population mean** is likely to lie.

**Part 2: Evaluating Model Performance Using Bootstrapping**

**Goal:**

The main objective here is to evaluate the **model’s generalization performance** by estimating the distribution of error metrics, such as **Mean Squared Error (MSE)**, using bootstrap resampling. This helps us understand how well a model is likely to perform on unseen data.

**Process of Bootstrap Model Evaluation:**

1. **Draw Bootstrap Samples**:
   * Generate a specified number of bootstrap samples (e.g., 30 or more) by resampling **with replacement** from the original dataset. Each bootstrap sample will be used to train a separate model.
2. **Generate Out-of-Bag (OOB) Samples**:
   * Since each bootstrap sample is drawn with replacement, not all data points from the original sample will be included in each bootstrap sample.
   * **The data points not selected form the out-of-bag (OOB) sample.** This OOB sample serves as a validation set for the model trained on the corresponding bootstrap sample.
   * Note: **Out-of-bag (OOB) samples** used as validation data in bootstrapping **can vary in size**. This variability arises because each bootstrap sample is created by drawing **with replacement** from the original dataset, which means some data points will appear multiple times in a bootstrap sample, while others will not appear at all. The instances that are **not selected** in each bootstrap sample become the OOB sample for that specific bootstrap model.
3. **Train Models and Calculate Errors**:
   * For each bootstrap sample, train a model and evaluate it using the OOB sample.
   * Calculate an error metric (e.g., MSE) for each model on its corresponding OOB sample. This error represents the model's performance on data not used during training.
4. **Examine the Distribution of Errors**:
   * After generating errors for all models, examine the **distribution of errors** (e.g., the distribution of MSE values).
   * **A low average error and low variance** indicate that the model performs consistently and generalizes well, while high variance in the error suggests that the model’s performance may be unstable.

Code implementation:

resample(loblolly, replace=True, n\_samples=44, random\_state=rng)

pines[~pines.index.isin(bootstrap.index)]

bootSample = resample(pines, replace=True, n\_samples=29, random\_state=rng)

oob = pines[~pines.index.isin(bootSample.index)]

# Section 8.8 Model Selection!

**Section 1: One-Standard Error Method**

**1.1 Understanding the One-Standard Error Method**

The **one-standard error method** is a model selection technique used to choose a model that balances simplicity and accuracy. **The idea is to avoid overfitting by selecting a model that performs close to the best-performing model but is simpler and more likely to generalize well**.

1. **Calculate Performance Scores**: Use cross-validation to calculate a performance metric (e.g., error rate) for each model.
2. **Identify the Best Model’s Mean Score**: Find the model with the lowest mean error across folds.
3. **Define the One-Standard Error Range**: Add one standard deviation to this lowest mean error to create a range.
4. **Select the Simplest Model in Range**: Choose the simplest model (with fewer parameters) whose mean error falls within this one-standard error range.

This approach reduces the risk of overfitting by allowing a slightly less accurate but simpler model if its performance is comparable.

**1.2 Example of the One-Standard Error Method**

Suppose you are predicting housing prices and are evaluating polynomial models of increasing complexity using 5-fold cross-validation. Here are the results:

| **Model Degree** | **Mean RMSE** | **Standard Error** |
| --- | --- | --- |
| Degree 1 | 24,000 | 2,500 |
| Degree 2 | 20,000 | 2,000 |
| Degree 3 | 19,500 | 1,800 |
| Degree 4 | 19,300 | 1,700 |
| Degree 5 | 19,200 | 1,600 |

1. **Best Model (Minimum RMSE)**: The model with the lowest RMSE is **Degree 5** with an RMSE of 19,200.
2. **One-Standard Error Range**: Adding one standard error (1,600) gives 20,800 as the threshold.
3. **Simplest Model in Range**: The **Degree 2 model** has an RMSE of 20,000, within the one-standard error range.

Using the one-standard error method, we select the simpler **Degree 2 model** instead of Degree 5, trading a slight increase in error for a model that’s less likely to overfit.

**Section 2: Alternative Methods for Model Selection**

In addition to the one-standard error method, other model selection criteria—such as **AIC** (Akaike Information Criterion), **BIC** (Bayesian Information Criterion), and **Adjusted R\_Squared**—are widely used to balance **fit** and **complexity**. These metrics help us choose models that are complex enough to capture meaningful patterns but simple enough to avoid overfitting.

**2.1 Information Criteria (AIC and BIC)**

**AIC** and **BIC** are information-based criteria that measure a model’s quality by balancing **goodness of fit** with **penalties for complexity** (number of parameters). Here’s how each criterion approaches this balance:

* **AIC (Akaike Information Criterion)**:
  + **Goal**: Minimize prediction error while allowing for model complexity, accepting slightly more complex models if they significantly improve fit.
  + **Formula**: ), where k is the number of parameters, and Likelihood measures how well the model fits the data (higher likelihood means better fit). Note that the term 2k is the complexity penalty, which is constant regardless of how many data points (n) you have.
  + **Intuition**: AIC is relatively lenient about adding parameters. It focuses on reducing **bias** (error due to underfitting) by capturing as much data pattern as possible. This flexibility allows AIC to select slightly more complex models that fit better, even if there’s a small risk of **overfitting**.
  + **When to Use**: AIC is useful when your priority is minimizing prediction error, as it balances fit with a moderate penalty on complexity.
* **BIC (Bayesian Information Criterion)**:
  + **Goal**: Select a simpler model that is likely to generalize better, especially as sample size (n) grows.
  + **Formula**: BIC= , where n is the sample size. The term increases with n, making BIC’s penalty for complexity much stronger than AIC’s.
  + **Intuition**: BIC is stricter about complexity, favoring models with fewer parameters to reduce **variance** (error due to sensitivity to fluctuations in the data). This means BIC often selects simpler models to avoid overfitting, which can sometimes lead to **underfitting** if the penalty discourages adding important features.
  + **When to Use**: BIC is preferred when simplicity and generalizability are key goals, especially with large datasets, as its higher penalty discourages excessive complexity.

**2.2 Adjusted**

**Adjusted**  is commonly used in regression and is a modified form of that accounts for the number of predictors in the model. Unlike standard , which can increase with any added feature, adjusted increases only if the added feature genuinely improves the model.

* **Intuition**: Adjusted penalizes complexity, increasing only when the additional feature improves the model meaningfully. A higher adjusted suggests a better balance of fit and simplicity.
* **When to Use**: Adjusted is useful in regression to assess if each added feature provides substantial explanatory power.

**2.3 Example: AIC and BIC with Conflicting Values**

Let’s see how AIC, BIC, and guide model selection in a scenario where they conflict slightly, revealing the pros and cons of each criterion.

Suppose you’re building a model to predict house prices. Here’s a comparison between **Model 1** (one feature) and **Model 2** (two features):

This example illustrates the subtle differences and trade-offs between AIC, BIC, and , emphasizing how each criterion can guide model selection based on specific goals, such as fit quality, complexity, and generalizability.

**Given Results:**

| **Model** | **AIC** | **BIC** |  |
| --- | --- | --- | --- |
| Model 1 (One Feature) | 42.9 | 45.6 | 0.55 |
| Model 2 (Two Features) | 41.3 | 46.8 | 0.58 |

**Metric-by-Metric Analysis**

1. **AIC (Akaike Information Criterion)**:
   * **Interpretation**: AIC measures model fit with a penalty for complexity (number of features).
   * **Comparison**: Model 2 has a lower AIC (41.3) than Model 1 (42.9).
   * **Conclusion**: AIC suggests that Model 2 is slightly better, as it achieves a better balance between fit and complexity.
2. **BIC (Bayesian Information Criterion)**:
   * **Interpretation**: Like AIC, BIC penalizes complexity but more heavily, especially with larger sample sizes.
   * **Comparison**: Model 1 has a lower BIC (45.6) than Model 2 (46.8).
   * **Conclusion**: BIC suggests Model 1, as it implies that adding the second feature may not justify the complexity.
3. **(Coefficient of Determination)**:
   * **Interpretation**: indicates the proportion of variance in the target variable explained by the model. Higher generally means better fit.
   * **Comparison**: Model 2 has a slightly higher (0.58) compared to Model 1 (0.55), indicating that the second feature adds some predictive power.
   * **Conclusion**: The higher suggests that Model 2 fits the data slightly better than Model 1.

**Balancing the Metrics**

* **Model 2** has a lower AIC and higher, indicating a slightly better fit to the data.
* **Model 1** has a lower BIC, which suggests that adding a second feature may not sufficiently improve the model to justify the added complexity.

**Decision-Making Logic**

1. **If You Prioritize Fit Over Simplicity**:
   * Model 2’s lower AIC and higher suggest that it fits the data better, capturing a bit more of the variance. If your primary goal is to improve the model’s explanatory power and you are less concerned about model simplicity, **Model 2** is preferable.
2. **If You Prioritize Simplicity and Generalization**:
   * Model 1’s lower BIC suggests that, given the slight improvement in fit with Model 2, it may not be worth the added complexity, especially if you have a smaller dataset or if interpretability and simplicity are priorities. In this case, **Model 1** would be a good choice as it’s simpler and might generalize better.

**Final Recommendation**

Since the differences in AIC, BIC, and are relatively small, the decision depends on the balance you’re aiming for:

* **If fit improvement is key**, go with **Model 2**.
* **If simplicity and generalizability are preferred**, go with **Model 1**.