Machine Learning: Module 2 Notes

Dr. Likhit Nayak

September 21, 2025

1 Training vs Testing Error

The core challenge in supervised machine learning is building a model that not only performs well on the data it was trained on but also generalizes to new, unseen data. Understanding the difference between training error and test error is crucial to this goal, as it helps us identify and avoid a common pitfall called *overfitting.*

Imagine you're studying for a major exam. Your only study material is the exact set of questions and answers from last year's test. You could memorize these questions perfectly and, if tested on them again, you would score 100%. This perfect score on the practice material is analogous to having a very low **training error**.

However, the real exam will have new questions that cover the same concepts but are worded differently. Because you only memorized specific answers instead of learning the underlying principles, your performance on the new exam will likely be poor. This failure to perform well on new, unseen problems is a failure to **generalize**. The model (your brain) has **overfit** to the training data (last year's test).

1.1 Key Concepts and Formal Definitions

1. **Supervised Learning:** The task of learning a function $f(X)$ that maps input features X to an output variable Y based on a labeled training dataset of example pairs (x_i, y_i).
2. **Loss Functions:** A loss function $L(Y, f(X))$ quantifies the cost or error of predicting $f(X)$ when the true value is Y. It tells us how bad a single prediction is.
   - **Squared Error Loss:**
     $$L(Y, f(X)) = (Y - f(X))^2,$$
     typically used in regression. It heavily penalizes larger errors.
   - **0–1 Loss:**
     $$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X), \\ 0, & Y = f(X). \end{cases}$$
     Used in classification; it simply counts whether a prediction is right or wrong.
3. **Training Error:** The average loss of the model calculated over the training dataset. It measures how well the model fits the data it learned from.
   $$\mathrm{err}_T = \frac{1}{N}\sum_{i=1}^{N} L\big(y_i, f(x_i)\big),$$
   where N is the number of samples in the training set. A low training error is easy to achieve with a complex enough model, but it can be misleading.
4. **Test (Generalization) Error:** The expected loss of the model on a new, unseen data point drawn from the true data distribution. This is the true measure of our model's performance in the real world.
   $$\mathrm{Err}_T = \mathbb{E}\big[L(Y, f(X))\big].$$

Since we cannot compute this expectation directly (the true distribution is unknown), we *estimate* it using a separate **test set**—data held out during training. The average loss on this test set serves as a proxy for the true generalization error.

Our real goal in machine learning is to minimize the test error, not the training error.

1.2 The Problem of Overfitting

Overfitting occurs when a model is too complex and begins to learn the noise and random fluctuations in the training data, rather than the underlying signal or pattern. Such a model performs exceptionally well on the training data but fails to generalize to new data.

The U-Shaped Curve of Test Error

When we plot error against model complexity, we typically see two distinct trends:

- **Training Error:** As model complexity increases, training error decreases monotonically. A more complex model can fit every training point, including noise.
- **Test Error:** Test error follows a U-shaped curve:
  1. *Underfitting (Left Side):* Very low complexity leads to high bias. Both training and test errors are high.
  2. *Optimal Complexity (Bottom):* The model captures the true pattern without fitting the noise. Test error is minimized.
  3. *Overfitting (Right Side):* Excessive complexity leads the model to fit noise. Training error continues to drop, but test error rises again.

What "Model Complexity" Means in Practice

- **Linear/Polynomial Regression:** Complexity is the degree of the polynomial. A linear model (degree 1) is simple; a high-degree polynomial (e.g., degree 10) is flexible and prone to overfitting.
- **k-Nearest Neighbors (k-NN):** Complexity is inversely related to k. A small k (e.g., $k = 1$) yields a highly flexible model with a jagged decision boundary (overfitting). A large k averages more neighbors, yielding a smoother, less complex model.

2 The Bias–Variance Decomposition

The **Bias-Variance Decomposition** is a fundamental concept in machine learning that breaks down the expected prediction error of a model into three distinct components. Understanding this decomposition helps us diagnose model performance issues like underfitting and overfitting, and guides model selection and regularization. The objective is to analyze the expected generalization error of a learning algorithm for a particular problem.

- **Model Setup:** We assume there is a true, underlying relationship between our features X and target Y, given by

$$Y = f(X) + \varepsilon,$$

  where:

  - $f(X)$ is the true, unknown function we want to learn.
  - ε is random noise with $\mathbb{E}[\varepsilon] = 0$ and $\mathrm{Var}(\varepsilon) = \sigma^2$.

- **Our Model:** We train a model $\hat{f}(X)$ using a specific training dataset. Since $\hat{f}$ depends on the data, it is treated as a random variable.

- **Goal:** Minimize the expected squared error for a new, unseen point x_0:

$$\mathrm{Err}(x_0) = \mathbb{E}\big[(Y - \hat{f}(x_0))^2\big],$$

  where the expectation is over both the sampling of the training set and the noise ε.

2.1 Derivation for Squared-Error Loss

We decompose the expected squared error $\mathrm{Err}(x_0)$ at the point x_0 step by step.

Step 1: Definition of the expected error

$$\mathrm{Err}(x_0) = \mathbb{E}\big[(Y - \hat{f}(x_0))^2\big].$$

Step 2: Substitute $Y = f(x_0) + \varepsilon$

$$\mathrm{Err}(x_0) = \mathbb{E}\Big[\big(f(x_0) + \varepsilon - \hat{f}(x_0)\big)^2\Big].$$

Step 3: Expand and separate terms

$$\begin{aligned}\mathrm{Err}(x_0) &= \mathbb{E}\big[((f(x_0) - \hat{f}(x_0)) + \varepsilon)^2\big] \\ &= \mathbb{E}\big[(f(x_0) - \hat{f}(x_0))^2\big] + 2\,\mathbb{E}[\varepsilon\,(f(x_0) - \hat{f}(x_0))] + \mathbb{E}[\varepsilon^2] \\ &= \mathbb{E}\big[(f(x_0) - \hat{f}(x_0))^2\big] + \sigma^2,\end{aligned}$$

since $\mathbb{E}[\varepsilon] = 0$ and $\mathbb{E}[\varepsilon^2] = \sigma^2$.

Step 4: Decompose the remaining term

Add and subtract $\mathbb{E}[\hat{f}(x_0)]$:

$$\begin{aligned}\mathbb{E}\big[(f(x_0) - \hat{f}(x_0))^2\big] &= \mathbb{E}\Big[\big(f(x_0) - \mathbb{E}[\hat{f}(x_0)] + \mathbb{E}[\hat{f}(x_0)] - \hat{f}(x_0)\big)^2\Big] \\ &= \underbrace{(f(x_0) - \mathbb{E}[\hat{f}(x_0)])^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\big[(\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)])^2\big]}_{\text{Variance}}.\end{aligned}$$

Step 5: Final decomposition

Combining all terms yields:

$$\mathrm{Err}(x_0) = \sigma^2 + \underbrace{(\mathbb{E}[\hat{f}(x_0)] - f(x_0))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}\big[(\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)])^2\big]}_{\text{Variance}}.$$

Thus,

$$\mathrm{Err}(x_0) = \text{Irreducible Error } (+\sigma^2) + \text{Bias}^2 + \text{Variance}.$$

2.2 Explanation of Terms

Irreducible Error (σ^2): The inherent noise in the data that no model can eliminate. It arises from unmeasured factors or randomness in the process.

Bias ($\mathrm{Bias}(\hat{f}(x_0)) = \mathbb{E}[\hat{f}(x_0)] - f(x_0)$): The error due to systematic deviations of the model's average prediction from the true function. High bias leads to underfitting.

Variance ($\mathrm{Var}(\hat{f}(x_0)) = \mathbb{E}[(\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)])^2]$): The error due to fluctuations of the model's prediction around its mean when trained on different datasets. High variance leads to overfitting.

2.3 The Bias–Variance Tradeoff

There is a fundamental tradeoff between bias and variance:

- **Low-Complexity Models** (e.g., linear regression): tend to have high bias and low variance (underfitting).
- **High-Complexity Models** (e.g., deep trees, k-NN with $k = 1$): tend to have low bias and high variance (overfitting).

The goal is to find a model complexity that minimizes the total error (bias^2+variance) for optimal generalization.

3 The Bayesian Perspective on Model Selection

In classical or frequentist statistics, model selection often involves finding a single "best" model based on a chosen criterion (e.g., minimizing a loss function, hypothesis testing). The Bayesian approach is fundamentally different. It doesn't seek a single best model but instead assigns a probability to every candidate model, quantifying our belief in each model after observing the data.

Key Concepts

1. **Posterior Probability of a Model**: The core of Bayesian model selection is computing the posterior probability of a model M_m given the observed data Z. This is calculated using Bayes' theorem:

$$\Pr(M_m \mid Z) = \frac{\Pr(Z \mid M_m)\,\Pr(M_m)}{\Pr(Z)}$$

   where:

   - $\Pr(M_m \mid Z)$ is the *posterior probability* of model M_m given data Z.
   - $\Pr(M_m)$ is the *model prior*, our belief in M_m before seeing data.
   - $\Pr(Z \mid M_m)$ is the *marginal likelihood* (or model evidence):

$$\Pr(Z \mid M_m) = \int \Pr(Z \mid \theta_m, M_m)\,\Pr(\theta_m \mid M_m)\,d\theta_m.$$

   - $\Pr(Z)$ is a normalizing constant, $\sum_m \Pr(Z \mid M_m)\Pr(M_m)$, common to all models and thus ignorable for comparison.

2. **Bayes Factor**: To compare two models M_i and M_j, consider the ratio of their posterior probabilities:

$$\frac{\Pr(M_i \mid Z)}{\Pr(M_j \mid Z)} = \frac{\Pr(Z \mid M_i)}{\Pr(Z \mid M_j)}\,\frac{\Pr(M_i)}{\Pr(M_j)}.$$

   The ratio of marginal likelihoods

$$\mathrm{BF}_{ij} = \frac{\Pr(Z \mid M_i)}{\Pr(Z \mid M_j)}$$

   is called the *Bayes Factor*, measuring evidence in favor of M_i over M_j.

3.1 From Bayes to BIC: The Derivation

The main practical challenge in Bayesian model selection is evaluating the high-dimensional integral for $\Pr(Z \mid M_m)$. The Bayesian Information Criterion (BIC) arises from a Laplace approximation to this integral.

The Laplace Approximation

Let

$$\log \Pr(Z \mid M_m) = \log \int \Pr(Z \mid \theta_m, M_m)\,\Pr(\theta_m \mid M_m)\,d\theta_m.$$

Define the likelihood $L(\theta_m) = \Pr(Z \mid \theta_m, M_m)$. For large sample size n, the integrand is sharply peaked around its mode, approximately the maximum likelihood estimate $\hat{\theta}_{\mathrm{MLE}}$. A second-order Taylor expansion of the log-integrand around this mode yields the Laplace approximation:

$$\log \Pr(Z \mid M_m) \approx \log L(\hat{\theta}_{\mathrm{MLE}}) - \frac{k}{2}\log n,$$

where k is the number of parameters in model M_m and n is the number of observations.

The BIC Formula

Multiplying the approximation by -2 gives the Bayesian Information Criterion:

$$\text{BIC} = -2 \log \Pr(Z \mid M_m) \approx -2 \log L(\hat{\theta}_{\text{MLE}}) + k \log n.$$

Here

- $-2 \log L(\hat{\theta}_{\text{MLE}})$ measures model fit,
- $k \log n$ penalizes model complexity.

In practice, one computes the BIC for each candidate model and selects the model with the lowest BIC, approximating the model with the highest posterior probability under a uniform prior.

4 K-Fold Cross-Validation

4.1 The Problem of Scarce Data

In machine learning, our primary goal is to build a model that generalizes well to new, unseen data. A standard approach when we have a large amount of data is to split it into three distinct sets:

1. **Training Set**: Used to train the model (i.e., learn the parameters).
2. **Validation Set**: Used to tune the model's hyperparameters (e.g., the k in k-nearest neighbors, or the depth of a decision tree) and select the best-performing model architecture. It acts as a proxy for the test set during development.
3. **Test Set**: Held aside until the very end. It's used only once to provide an unbiased estimate of the final model's performance on unseen data.

However, in many real-world scenarios, data is a scarce and valuable resource. Splitting the data into three parts significantly reduces the amount of data available for training, which can lead to a suboptimal model. The challenge becomes:

> **How can we get a reliable estimate of the test error without sacrificing a large chunk of our data for a dedicated validation set?**

4.2 The Solution

K-fold cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. It provides a more robust estimate of model performance than a single train/validation split.

4.3 The Process

The mechanics of K-fold cross-validation are as follows:

1. **Split**: The entire dataset is randomly shuffled and partitioned into K equal-sized, non-overlapping subsets, called *folds*.
2. **Iterate**: Repeat the following for each $k = 1, \dots, K$:
   (a) *Hold-out*: Use the k-th fold as the validation set.
   (b) *Train*: Train the model on the remaining $K - 1$ folds combined.
   (c) *Evaluate*: Compute a performance score (e.g., accuracy, mean squared error) on the held-out fold and record it.
3. **Average**: After K iterations, compute the mean of the K recorded scores. This average is the cross-validation estimate of the model's performance on unseen data.

4.4 Diagrammatic Representation (for $K = 5$)

Imagine our dataset as a single block:

```
[ Dataset ]
```

We split it into five folds:

```
[ Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 ]
```

Then we iterate 5 times:

1. Train on [F2, F3, F4, F5], Test on [F1] $\rightarrow$ Error_1
2. Train on [F1, F3, F4, F5], Test on [F2] $\rightarrow$ Error_2

3. Train on [F1, F2, F4, F5], Test on [F3] $\rightarrow$ Error_3
4. Train on [F1, F2, F3, F5], Test on [F4] $\rightarrow$ Error_4
5. Train on [F1, F2, F3, F4], Test on [F5] $\rightarrow$ Error_5

The final cross-validation error estimate is given by:

$$\text{CV-error} = \frac{\text{Error}_1 + \text{Error}_2 + \text{Error}_3 + \text{Error}_4 + \text{Error}_5}{5}.$$

This procedure ensures each data point is used exactly once as validation and $K - 1$ times in training.

4.5 Choosing the Value of K: A Bias–Variance Tradeoff

The choice of K affects both the performance estimate and computational cost.

4.5.1 Leave-One-Out Cross-Validation (LOOCV)

A special case where $K = N$, the total number of samples. Each iteration uses one point for validation and the other $N - 1$ for training, repeated N times.

Bias: Very low, since each training set is nearly the entire dataset.

Variance: High, because the N training sets are almost identical, leading to highly correlated estimates.

Computational Cost: Very high, requiring N model trainings.

4.5.2 $K = 5$ or $K = 10$ (Common Choices)

Moderate K (e.g., 5 or 10) balances bias, variance, and cost.

Bias: Slightly higher than LOOCV, as each training set excludes 10% (for $K = 10$) or 20% (for $K = 5$) of the data.

Variance: Lower than LOOCV, because the training sets overlap less, reducing correlation among estimates.

Computational Cost: Feasible, requiring only K model trainings.

In summary, K-fold cross-validation is an essential technique for model evaluation when data is limited. By systematically creating multiple train/validation splits, it provides a more stable and less biased performance estimate than a single split, with $K = 5$ or $K = 10$ commonly offering the best tradeoff between bias, variance, and computational expense.

5 The Right Way to do Cross-Validation

The critical rule in machine learning model validation is: all aspects of model fitting, including preprocessing and feature selection, must be performed inside the cross-validation (CV) loop to avoid a common and serious error known as data leakage.

5.1 Key Concepts

- **Cross-Validation (CV):** A resampling procedure used to evaluate machine learning models on a limited data sample. It works by partitioning the data into folds, then iteratively training the model on some folds and testing it on the remaining fold, ensuring that every data point gets to be in a test set once.
- **Data Leakage:** The introduction of information about the test data into the model training process. This leads to an overly optimistic evaluation of the model's performance, as the model has inadvertently "cheated" by seeing information from the data it is being tested on. The model will perform well on the test set but will likely fail to generalize to new, unseen data.

5.2 The Scenario: The $p \gg N$ Problem

Consider a challenging classification scenario where the number of predictors (features) is much larger than the number of samples (observations). This is common in fields like genomics or text analysis.

- $N = 50$ samples (e.g., 50 patients)
- $p = 5000$ predictors (e.g., 5000 gene expression levels)
- **Task:** Binary classification (e.g., patient has disease or not)

5.3 The Wrong Way: Inducing Data Leakage

A common but incorrect approach is to perform feature selection before starting the cross-validation process.

1. **Select Predictors Using All Data:** Analyze all 5000 predictors across all 50 samples to find the 100 predictors with the highest correlation to the class labels. This is the pre-filtering step.
2. **Perform Cross-Validation:** Take the reduced dataset (50 samples, 100 selected predictors) and perform a standard 10-fold cross-validation to estimate the model's error rate.

The Result

The procedure yields a shockingly low and misleading CV error, perhaps around 3%. This suggests the model is almost perfectly accurate.

Why This is Wrong

The error lies in the first step. When the 100 best predictors were selected, information from the *entire* dataset was used, including labels of samples that would later be part of the test data inside the CV loop. For any given test fold, the predictors were chosen because they had a strong correlation with the labels in that fold. The model received a hint about the test set's answers before training. This is a classic case of data leakage.

5.4 The Right Way: The Correct CV Procedure

To obtain an honest estimate of model performance, every step of the model-building pipeline must be contained within the CV loop. The CV loop must be the outermost operation. Using 10-fold CV as an example:

1. Partition the 50 samples into 10 folds (5 samples each).
2. Iterate from $k = 1$ to 10:
    (a) Designate fold k as the **test set** and the remaining 9 folds as the **training set**.
    (b) **Perform Predictor Selection:** Analyze the 5000 predictors using only the training set to find the 100 predictors most correlated with the labels.
    (c) **Train the Model:** Build the classification model using only the selected predictors and the training set.
    (d) **Test the Model:** Evaluate the trained model on the held-out test set (fold k) and record the error.
3. After the loop, average the 10 error rates. This average is the unbiased estimate of the model's generalization error.

5.5 The Golden Rule of Cross-Validation

The core principle is that the test data in each fold must be treated as if it were truly unseen. It cannot be used, in any way, to inform the model-building process. This includes:

- Predictor/Feature Selection
- Hyperparameter Tuning
- Outlier Removal
- Data Scaling and Normalization (fit only on the training data, then transform both training and test data)

By following this rule, the cross-validation result is a trustworthy estimate of how the model will perform on new, real-world data.

6 Bootstrap and Prediction Error Estimation

6.1 What is the Bootstrap?

The bootstrap is a powerful and widely used resampling technique in statistics. Its core idea is to simulate the process of obtaining new datasets from an underlying population, using only the original training dataset you have.

How it works

- **Sampling with Replacement:** Given an original training dataset of size N, a "bootstrap dataset" is created by randomly drawing N samples *with replacement* from the original data.
- **Simulation:** Because we sample with replacement, some original data points may appear multiple times in a single bootstrap dataset, while others may not appear at all. This process mimics drawing a new, independent dataset from the true population distribution that the original data came from.
- **Purpose:** By creating many (e.g., $B = 200$) of these bootstrap datasets, we can train our model on each one and analyze the variability of its performance or parameters. This is extremely useful for estimating quantities like standard errors, confidence intervals, and, in this context, prediction error.

6.2 Leave-One-Out Bootstrap (LOOB) Error Estimation

The bootstrap can be used to estimate the prediction error of a model, similar to cross-validation. This specific method is called the Leave-One-Out Bootstrap.

The Bias Problem

When we create a bootstrap sample, not all original data points are included. The probability that any given observation is *not* picked in a single draw is $(1 - 1/N)$. The probability it is never picked in N draws is

$$(1 - 1/N)^N.$$

As N becomes large, this probability approaches $1/e \approx 0.368$. This means, on average, each bootstrap dataset contains only about $1 - 1/e \approx 0.632$ (or 63.2%) of the unique observations from the original training set. The remaining 36.8% are called "out-of-bag" (OOB) samples.

Calculating the LOOB Error ($\mathrm{Err}^{(1)}$)

1. Generate a large number of bootstrap datasets, B.
2. For each bootstrap dataset $b = 1, \ldots, B$:
   (a) Train the model using the bootstrap dataset.
   (b) Test the trained model on the out-of-bag (OOB) samples (the original data points that were *not* included in this bootstrap dataset).
   (c) Calculate the error (e.g., misclassification rate) on these OOB predictions.
3. The *Leave-One-Out Bootstrap Error*, denoted $\mathrm{Err}^{(1)}$, is the average of these errors across all B bootstrap samples.

The Issue with $\mathrm{Err}^{(1)}$

Because each model is trained on only about 63.2% of the data, it is likely to perform worse than a model trained on the full 100% of the data. Therefore, $\mathrm{Err}^{(1)}$ is generally a *pessimistic* estimate—it overestimates the true prediction error.

6.3 The .632 Estimator

The .632 estimator was designed to correct for the pessimistic bias of the LOOB error.

The idea is to combine the pessimistic LOOB error, $\mathrm{Err}^{(1)}$, with the highly optimistic training error, err (the error rate when the final model is tested on the very same data it was trained on). The formula is

$$\mathrm{Err}_{.632} = 0.368\,\mathrm{err} + 0.632\,\mathrm{Err}^{(1)}.$$

This formula creates a weighted average: it pulls the pessimistic $\mathrm{Err}^{(1)}$ downward, using the optimistic err as a correction factor. The weights (0.368 and 0.632) are derived from the bootstrap sampling probabilities.

7 Evaluating Models: Beyond Errors & Accuracy

When evaluating a classification model, the most intuitive metric is accuracy, or its inverse, the error rate ($1 -$ Accuracy). However, relying solely on accuracy can be dangerously misleading, especially with imbalanced datasets.

Motivation Example: Rare Disease Diagnosis

Imagine a test for a rare disease that affects 1 in 1000 people. A model that simply predicts "No Disease" for every person would be 99.9% accurate. While this accuracy score seems excellent, the model is completely useless because it fails to identify a single sick person (its entire purpose). This demonstrates that we need to understand the *types* of errors a model makes, not just the total number of errors.

7.1 The Confusion Matrix: Deconstructing Model Performance

A confusion matrix is a table that visualizes the performance of a classification algorithm. It breaks down predictions into four categories, comparing the actual outcomes to the predicted outcomes.

Consider a spam filter example where "Positive" = Spam and "Negative" = Not Spam (Ham).

	Predicted: Spam	**Predicted: Not Spam**
Actual: Spam	True Positive (TP)	False Negative (FN)
Actual: Not Spam	False Positive (FP)	True Negative (TN)

Table 1: Confusion matrix for a spam filter.

7.2 Derived Metrics from the Confusion Matrix

Using the four values from the matrix, we can calculate more nuanced performance metrics.

1. Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. Recall (Sensitivity or True Positive Rate)

$$\text{Recall} = \frac{TP}{TP + FN}$$

4. Specificity (True Negative Rate)

$$\text{Specificity} = \frac{TN}{TN + FP}$$

7.3 The Precision–Recall Tradeoff

Precision and Recall often have an inverse relationship. Improving one tends to lower the other. Many classifiers output a probability or a confidence score. By changing the threshold to classify an instance as positive, we can trade precision for recall.

- **High Threshold (e.g., ¿0.9 confidence to be spam):** The model will be very conservative, only flagging emails it's extremely sure about. This leads to *high precision* (few FPs; important emails are safe) but *low recall* (many FNs; more spam gets through).
- **Low Threshold (e.g., ¿0.4 confidence to be spam):** The model will be aggressive, flagging many emails as spam. This leads to *high recall* (most spam is caught) but *low precision* (many FPs; more legitimate emails land in the spam folder).

7.4 The F1-Score: A Balanced Metric

Since we often want a balance between Precision and Recall, we use the F1-Score, which combines them into a single number. It is the *harmonic mean* of precision and recall.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The harmonic mean punishes extreme values more than a simple average. A model with a high F1-score must have both high precision and high recall. A model with excellent precision but terrible recall (or vice versa) will have a low F1-score, making it a more robust metric for evaluating a model's overall performance, especially on imbalanced datasets.

8 Receiver Operating Characteristic (ROC) Curve and AUC

8.1 Key Concepts

- **Classifier Threshold:** A value used to convert a classifier's continuous output (like a probability or score) into a discrete class label (e.g., if probability ¿ 0.5, predict "Spam").
- **Receiver Operating Characteristic (ROC) Curve:** A graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
- **True Positive Rate (TPR):** Also known as Sensitivity or Recall. It measures the proportion of actual positives that are correctly identified.
  - **Formula:** $\text{TPR} = \frac{TP}{TP+FN}$, where TP = True Positives, FN = False Negatives.
- **False Positive Rate (FPR):** It measures the proportion of actual negatives that are incorrectly identified as positives.
  - **Formula:** $\text{FPR} = \frac{FP}{FP+TN}$, where FP = False Positives, TN = True Negatives. Note that $\text{FPR} = 1 - \text{Specificity}$.
- **Area Under the Curve (AUC):** The area under the ROC curve, which serves as a single-number summary of the classifier's performance across all thresholds.

8.2 The Need for a Threshold-Independent Metric

Most classification models, like logistic regression or neural networks, don't just output a binary label (e.g., "Spam" or "Not Spam"). Instead, they output a probability score between 0 and 1. To make a final decision, we apply a threshold. For example, we might classify an email as "Spam" if its predicted probability is greater than 0.5. However, this 0.5 threshold is an arbitrary choice. A different threshold (e.g., 0.8) would change the model's performance metrics like accuracy, precision, and recall. An ROC curve allows us to evaluate the model's ability to discriminate between classes, independent of any single, chosen threshold.

8.3 Building the ROC Curve

Process:

1. For every instance in the validation set, get the model's predicted probability for the positive class.
2. Create a list of all possible thresholds. In practice, this is done by using the unique predicted probability scores as the thresholds.
3. For each threshold, classify all instances and calculate the corresponding TPR and FPR.
4. Plot each (FPR, TPR) pair on a graph, with FPR on the x-axis and TPR on the y-axis.
5. Connect the points to form the ROC curve.

Example Trace

Imagine a dataset with 3 positive and 3 negative instances. Our model gives the following scores:

- Scores for positives: [0.9, 0.8, 0.6]
- Scores for negatives: [0.7, 0.5, 0.4]

The ROC curve points are generated by sliding a threshold from above 1.0 down to 0:

- **Threshold = 1.0:** All are classified as negative. $TP = 0, FP = 0$. $\text{TPR} = 0/3 = 0, \text{FPR} = 0/3 = 0$. Point: $(0, 0)$.

- **Threshold = 0.9:** The instance with score 0.9 is now positive. It's a true positive. $TP = 1, FP = 0$. TPR $= 1/3$, FPR $= 0$. Point: $(0, \frac{1}{3})$.
- **Threshold = 0.8:** The instance with score 0.8 is now positive. Another true positive. $TP = 2, FP = 0$. TPR $= 2/3$, FPR $= 0$. Point: $(0, \frac{2}{3})$.
- **Threshold = 0.7:** The instance with score 0.7 is now positive. It's a false positive. $TP = 2, FP = 1$. TPR $= 2/3$, FPR $= 1/3$. Point: $(\frac{1}{3}, \frac{2}{3})$.
- **Threshold = 0.6:** The instance with score 0.6 is now positive. A true positive. $TP = 3, FP = 1$. TPR $= 1$, FPR $= 1/3$. Point: $(\frac{1}{3}, 1)$.
- And so on, until the threshold is below the lowest score, where everything is classified as positive. Final Point: $(1, 1)$.

Meaning of Key Points on the Curve

- (0, 0): The classifier predicts everything as negative. It has a 0% True Positive Rate and a 0% False Positive Rate.
- (1, 1): The classifier predicts everything as positive. It correctly identifies all positives (100% TPR) but also incorrectly identifies all negatives (100% FPR).
- (0, 1): The perfect classifier. It achieves a 100% TPR without any false positives (0% FPR). This is the top-left corner of the plot.

8.4 Interpreting ROC Curves and AUC

- **Curve Shape:** A better classifier has an ROC curve that is closer to the top-left corner (0, 1). This indicates that the model can achieve a high TPR while maintaining a low FPR.
- **Random Guessing:** A diagonal line from (0,0) to (1,1) represents a random classifier (e.g., flipping a coin). A model whose curve falls below this line is worse than random.
- **Area Under the Curve (AUC):** AUC is a single number that summarizes the entire ROC curve. It represents the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.
  - AUC = 1.0: A perfect classifier.
  - AUC = 0.5: A model with no discriminative ability, equivalent to random guessing.
  - 0.5 ¡ AUC ¡ 1.0: The model has some ability to discriminate. The higher the AUC, the better the model.

8.5 When to Use ROC vs. Precision–Recall

- **ROC/AUC:** This is a good general-purpose metric, especially when the classes are balanced or when you care equally about the performance on both the positive and negative classes. It is robust to changes in the class distribution.
- **Precision–Recall (PR) Curve:** This metric is more informative for highly imbalanced datasets where the positive class is rare and of primary interest (e.g., fraud detection, rare disease diagnosis). In such cases, ROC curves can be misleadingly optimistic because a large number of true negatives can make the FPR appear very low, even if the model performs poorly on the few positive instances. A PR curve, which plots Precision vs. Recall (TPR), provides a better view of performance on the minority class.

Disclaimer

These notes were generated with the assistance of artificial intelligence. While efforts have been made to ensure accuracy, they may contain errors or omissions. Please verify any critical information and use these notes as a supplement to, not a replacement for, topics discussed in the lectures.