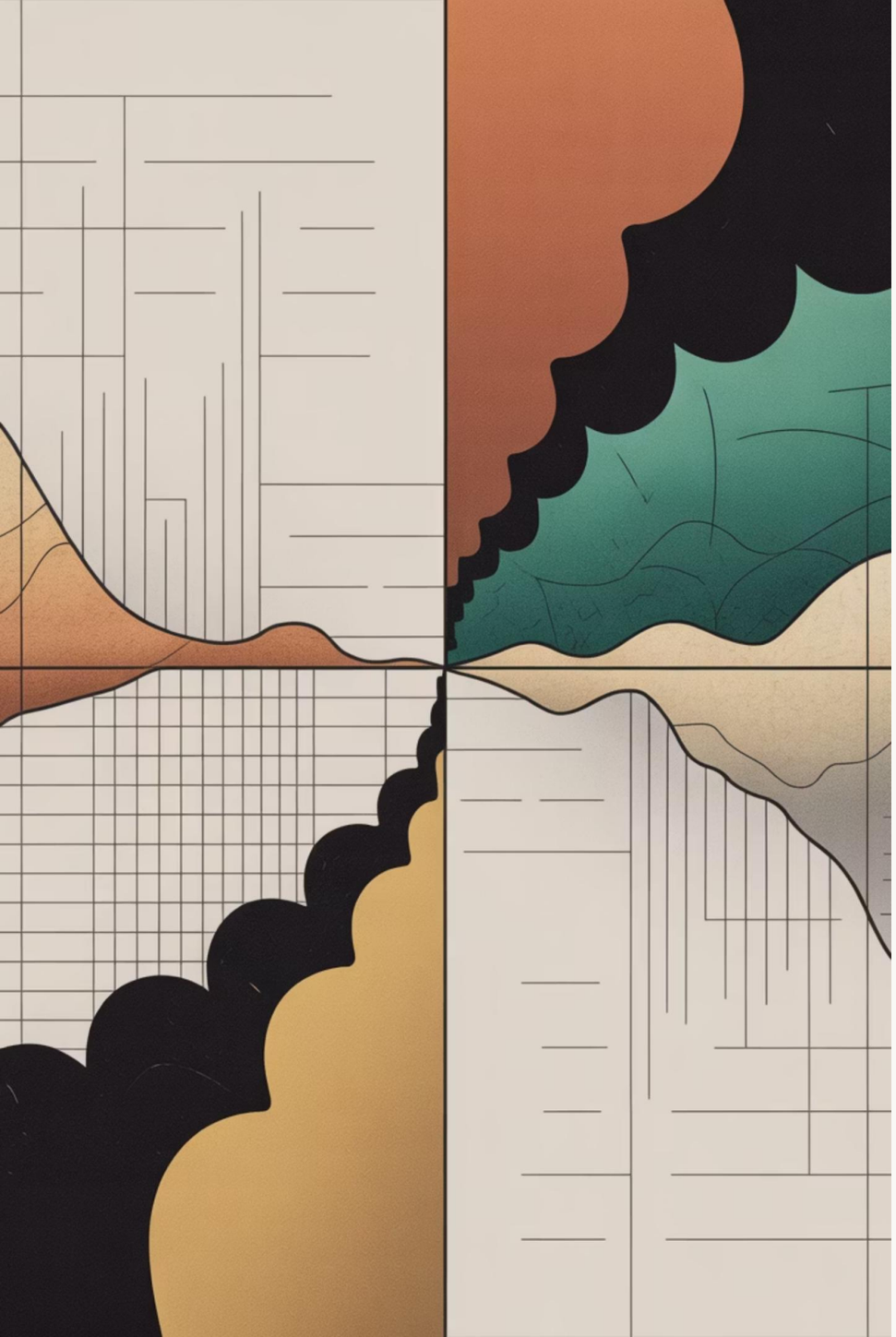

CEFET/RJ

Data Sampling Strategies for Machine Learning

Effective model evaluation begins with proper data splitting. Whether you're building predictive models or classification systems, how you divide your dataset between training and testing sets fundamentally impacts your results. Random and stratified sampling represent two core approaches, each with distinct advantages for different scenarios.

This presentation explores both methods using the DAL Toolbox in R, providing practical guidance for data science practitioners. We'll examine the theory behind each approach, demonstrate implementation with real code examples, and help you choose the right strategy for your machine learning projects.

Eduardo Ogasawara
eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

Random Sampling: Theory and Principles

How Random Sampling Works

Random sampling splits datasets arbitrarily, selecting observations without regard to class distribution or other characteristics. Each data point has an equal probability of assignment to either training or testing sets. This straightforward approach works quickly and requires minimal computational overhead.

While random sampling preserves overall proportions on average across multiple runs, individual splits may produce imbalanced datasets. For example, in a binary classification problem with 70% positive cases, one random split might yield 65% positive in training and 75% in testing—creating potential evaluation bias.

Speed

Fast execution with minimal computation

Simplicity

Easy to implement and understand

Risk

May create imbalanced splits

Reference: Han, J., Kamber, M., & Pei, J. – Data Mining: Concepts and Techniques (3rd Ed.); Witten, I. H., Frank, E., Hall, M. A., & Pal, C. – Data Mining: Practical Machine Learning Tools and Techniques (4th Ed.)

Practice: Implementing Random Sampling

Random Sampling with DAL Toolbox

The DAL Toolbox provides a clean interface for random sampling through the `sample_random()` function. When combined with `train_test()`, it automatically splits your dataset into training and testing subsets. The example below demonstrates this using the classic iris dataset.

```
tt <- train_test(sample_random(), datasets::iris)
table(tt$train$Species)
table(tt$test$Species)
```

After splitting, the `table()` function reveals the distribution of Species classes in both sets. Notice how the counts may vary from the original proportions—this illustrates the potential imbalance inherent in random sampling. Running this code, multiple times produces different distributions, highlighting the variability of this approach.

The `train_test()` function returns a list containing two data frames: `$train` and `$test`. By default, it uses a 70-30 split, but you can customize this ratio based on your dataset size and evaluation needs.

Full code available at: https://github.com/cefet-rj-dal/daltoolbox/blob/main/examples/transf/sample_random.md

Stratified Sampling: Maintaining Balance

Proportional Preservation

Maintains exact class ratios from the original dataset in both training and testing sets

Classification Essential

Critical for imbalanced datasets where minority classes require adequate representation

Reliable Evaluation

Produces consistent, reproducible results that better reflect true model performance

Stratified sampling addresses the key limitation of random sampling by ensuring class proportions remain constant across splits. When your dataset contains 50 observations each of three classes, stratified sampling guarantees these ratios persist in training and testing sets. This precision proves especially valuable for imbalanced classification problems where minority classes might otherwise be underrepresented or entirely absent from one subset.

The stratification process divides the dataset by target variable, then samples proportionally from each stratum. This methodology ensures every class receives fair representation during model training and evaluation, leading to more reliable performance metrics and better generalization estimates.

Reference: Han, J., Kamber, M., & Pei, J. – Data Mining: Concepts and Techniques (3rd Ed.); Witten, I. H., Frank, E., Hall, M. A., & Pal, C. – Data Mining: Practical Machine Learning Tools and Techniques (4th Ed.)

Practice: Implementing Stratified Sampling

Stratified Sampling with DAL Toolbox

Implementing stratified sampling requires only one modification: specify the stratification variable. The `sample_stratified()` function accepts the column name containing your target variable—in this case, "Species".

```
tt <- train_test(sample_stratified("Species"), datasets::iris)
table(tt$train$Species)
table(tt$test$Species)
```

Examining the output tables reveals identical proportions across both sets, demonstrating the precision of stratified sampling.

Unlike random sampling, running this code multiple times produces consistent class distributions. The exact counts may vary slightly due to rounding with small datasets, but proportions remain stable—providing reliable foundations for model evaluation.

Full code available at: https://github.com/cefet-rj-dal/daltoolbox/blob/main/examples/transf/sample_stratified.md

Comparing Random vs Stratified Approaches

Random Sampling

Advantages: Quick execution, minimal setup, works well with balanced datasets

Limitations: May produce imbalanced splits, less consistent results, risky for skewed data

Best for: Large balanced datasets, exploratory analysis, regression problems

Stratified Sampling

Advantages: Preserves class proportions, consistent results, reliable for classification

Limitations: Requires stratification variable, slightly more complex setup

Best for: Classification tasks, imbalanced datasets, rigorous evaluation needs

The choice between random and stratified sampling depends on your specific use case. For classification problems—especially those with imbalanced classes—stratified sampling should be your default choice. It eliminates a significant source of evaluation variance and ensures minority classes receive adequate representation in both training and testing sets.

Random sampling remains appropriate for well-balanced datasets or regression problems where maintaining class proportions isn't applicable. However, given the minimal additional complexity of stratified sampling in DAL Toolbox, many practitioners default to stratification as a best practice across most scenarios.

Reference: Han, J., Kamber, M., & Pei, J. – Data Mining: Concepts and Techniques (3rd Ed.); Witten, I. H., Frank, E., Hall, M. A., & Pal, C. – Data Mining: Practical Machine Learning Tools and Techniques (4th Ed.)

Advanced: Stratified K-Fold Cross-Validation

Cross-validation extends stratified sampling principles to multiple splits, providing more robust performance estimates. K-fold cross-validation divides data into k subsets (folds), iteratively using each fold as a test set while training on the remaining folds. Stratified k-fold ensures each fold maintains the original class proportions.

Implementation in DAL Toolbox

```
folds <- k_fold(sample_stratified("Species"), datasets::iris, 4)
do.call(rbind, lapply(folds, function(f) table(f$Species)))
```

Understanding the Output

The `k_fold()` function returns a list of k data frames, each representing one fold. The subsequent code combines these into a matrix showing class distributions across all folds, confirming stratification worked correctly.

Why Use K-Fold?

K-fold cross-validation reduces evaluation variance by averaging results across multiple train-test splits. This provides more reliable performance estimates, especially valuable for smaller datasets where a single split might not represent overall model behavior.

Full code available at: https://github.com/cefet-rj-dal/daltoolbox/blob/main/examples/transf/sample_stratified.md

Key Takeaways: Sampling Best Practices

1 Sampling Drives Evaluation Quality

How you split your data directly impacts model performance estimates. Poor sampling strategies lead to misleading metrics, overconfident predictions, and models that fail in production. Always treat sampling as a critical first step in your machine learning pipeline.

2 Random Sampling Has Limitations

While simple and fast, random sampling introduces unnecessary risk through potential class imbalance. Reserve it for well-balanced datasets or cases where speed outweighs evaluation precision. For most classification tasks, the risks outweigh the benefits.

3 Default to Stratified Sampling

Stratified sampling ensures fair representation of all classes, produces consistent results, and costs almost nothing in additional complexity. Make it your standard approach for classification problems, especially with imbalanced datasets or when rigorous evaluation matters.

4 DAL Toolbox Simplifies Implementation

The DAL Toolbox provides clean, intuitive functions for both sampling strategies. With consistent syntax and minimal setup, you can focus on modeling rather than data plumbing. Extend to k-fold cross-validation for even more robust evaluation.