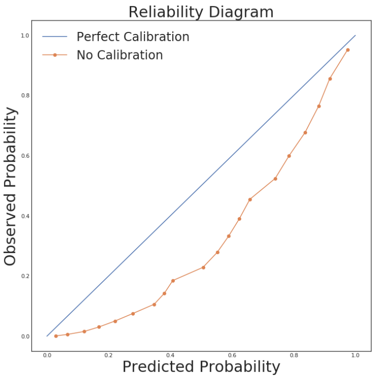
Applied Machine Learning

Performance Evaluation: Calibration Methods & Practices



Learning goals

- How to calibrate probabilities using different methods
- Best practices for data splitting in calibration

CALIBRATING PROBABILITIES

Goal: Calibration Techniques

- Calibrating probabilities involves **post-processing** predictions.
- **Aim:** Ensure predicted probabilities of any event should (on average) match their observed empirical probabilities.
  $\Rightarrow$ Makes predictions interpretable as actual risk/probabilities.
- Calibration should be performed on **new data** not used for model fitting.
- Link function in **logistic regression** can be viewed as a calibration of predictions of a **linear regression**.

CALIBRATING PROBABILITIES

- Let $s(\mathbf{x})$ denote the predicted (uncalibrated) score for input $\mathbf{x}$
- Define $\mathbb{S}$ as the set of possible scores produced by the classifier
- Goal: Construct a *calibration function* C that maps scores $s(\mathbf{x}) \in \mathbb{S}$ to calibrated probabilities $C(s(\mathbf{x})) \in [0, 1]$:

$$C : \mathbb{S} \to [0, 1], \text{such that } C(s(\mathbf{x})) \approx P(y = 1 \mid s(\mathbf{x})) \text{ (well-calibrated)}$$

- To learn calibrator function C, use a separate *calibration dataset*:

$$\mathcal{D}_{\text{cal}} = \{(s^{(i)}, y^{(i)})\}_{i=1}^{n} \subset \mathbb{S} \times \{0, 1\}$$

- Important: $\mathcal{D}_{\text{cal}}$ must be disjoint from the training data used to learn the scoring classifier to avoid bias in estimated calibration

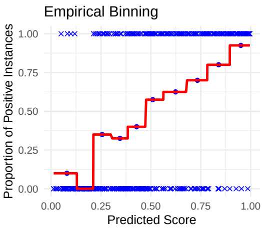
EMPIRICAL BINNING

Empirical binning partitions predicted scores $s \in \mathbb{S}$ into bins $B_1, \ldots, B_M$ and defines a (piecewise constant) calibration function C by:

$$C(s) = \bar{p}_{J(s)} = \frac{\sum_{i=1}^{n} \mathbb{1}[s^{(i)} \in B_{J(s)}] \cdot y^{(i)}}{|B_{J(s)}|}, \quad \text{where}$$

- $J(s) \in \{1, \ldots, M\}$ is the bin index that s belongs to (i.e., $s \in B_{J(s)}$)
- $s^{(i)} = s(\mathbf{x}^{(i)})$ is the score and $y^{(i)}$ the label of instance i
- numerator counts the number of positive instances within $B_{J(s)}$

$\Rightarrow$ C maps s to $\bar{p}_{J(s)}$ (average proportion of positive instances in $B_{J(s)}$)



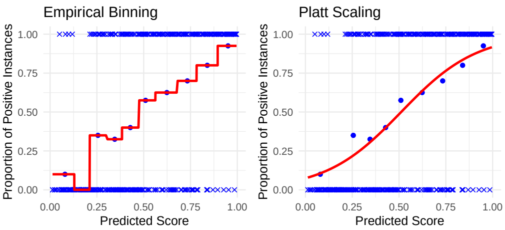
PLATT SCALING

“Niculescu-Mizil et al.” 2005

Platt scaling applies logistic regression to predicted scores $s \in \mathbb{S}$, i.e., it fits a calibration function C minimizing the log-loss on $\mathcal{D}_{cal}$ by:

$$C(s) = \frac{1}{1 + \exp(\theta_0 + \theta_1 \cdot s)}, \quad \text{where}$$

- θ_0 is the intercept, and θ_1 the slope estimated by the logistic regression
- Requires: unbounded outputs before sigmoid/softmax, good for Neural Networks, SVM, boosted trees



ISOTONIC REGRESSION

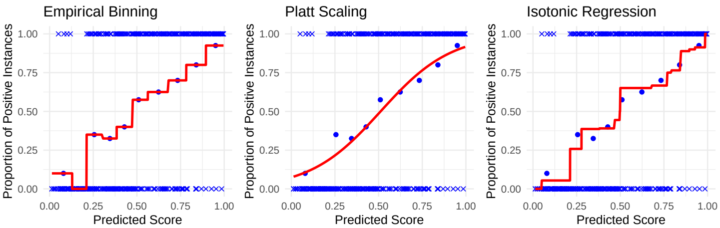
▸ "Kull et al." 2017

- *Isotonic regression* combines the non-parametric character of empirical binning with the monotonicity guarantee of Platt scaling by minimizing

$$\sum_{i=1}^{n} w_i \left(C(s^{(i)}) - y^{(i)} \right)^2$$

subject to the constraint that C is isotonic: $C(s) \leq C(t)$ for $s < t$.

- Note: C is evaluated only at a finite number of points; in-between, one may (linearly) interpolate or assume a piecewise constant function.



ISOTONIC REGRESSION - PAVA

- Optimization can be solved by the pool-adjacent violators algorithm (PAVA) by sorting scores such that

$$s^{(1)} < s^{(2)} < \ldots < s^{(n)} .$$

- Initialize bins B_i for each observation $(s^{(i)}, y^{(i)})$
- Assign to all scores $s \in B_i$: $C(s) = y^{(i)}$ with initial width $w(B_i) = 1$.
- A merge operation combines two adjacent bins B_j and B_k into a new bin $B = B_j \cup B_k$ with width $w(B) = w(B_j) + w(B_k)$ and

$$C(B) = \frac{w(B_j)C(B_j) + w(B_k)C(B_k)}{w(B_j) + w(B_k)} .$$

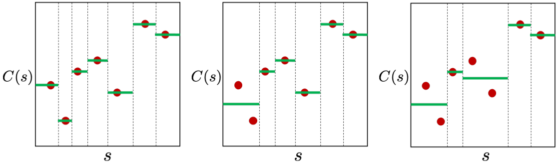
- The algorithm looks for violations of the monotonicity constraint and adjusts them with the best possible fit under the constraint ($\mathcal{O}(n)$).

ISOTONIC REGRESSION - PAVA

PAVA iterates the following steps (simplified to avoid notational overload):

(1) Find first violating adjacent bins B_i and B_{i+1} such that $C(B_i) > C(B_{i+1})$.

(2) Merge B_i and B_{i+1} into a new bin B. Stop if no violation occurred.



ISOTONIC REGRESSION - PAVA

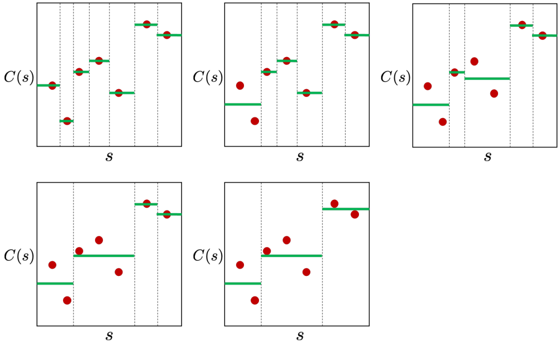
PAVA iterates the following steps (simplified to avoid notational overload):

(1) Find first violating adjacent bins B_i and B_{i+1} such that $C(B_i) > C(B_{i+1})$.

(2) Merge B_i and B_{i+1} into a new bin B. Stop if no violation occurred.

(3) If $C(B) < C(B_{i-1})$ for the left neighbor bin B_{i-1}, merge also these bins and continue until no more violations are found (monotonicity).

(4) Continue with **(1)**.



HOW TO SPLIT DATA FOR CALIBRATION? ▸ “Eskandar” 2023

Problem: How to avoid overfitting the calibrator during training?

Deployment scenario:

- Inducer (ML algorithm & hyperparameters) already selected
- Want to calibrate predictions of resulting model using all available data

HOW TO SPLIT DATA FOR CALIBRATION?

▸ "Eskandar" 2023

Problem: How to avoid overfitting the calibrator during training?

Deployment scenario:

- Inducer (ML algorithm & hyperparameters) already selected
- Want to calibrate predictions of resulting model using all available data

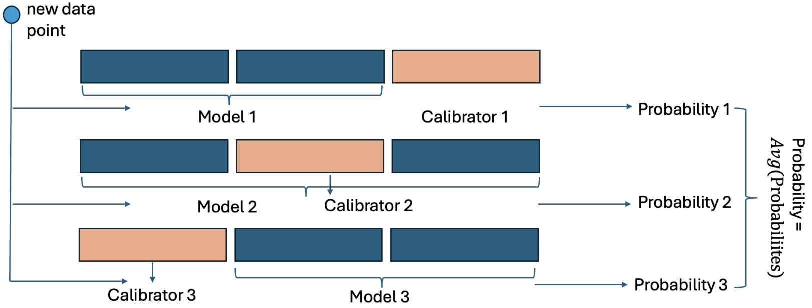
Naive solution: Use hold-out calibration set

Limitation: Does not utilize full dataset for training model and calibrator

Idea: Use *k*-fold CV to use available data more efficiently

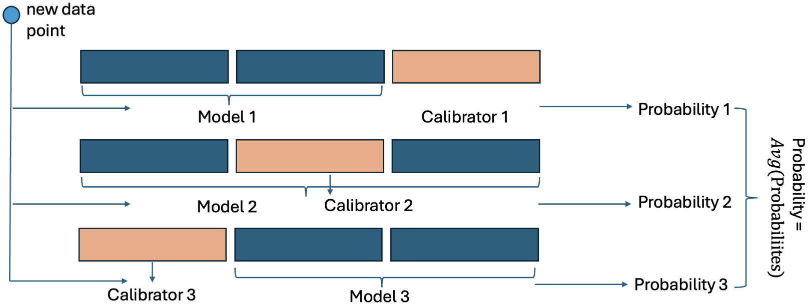
CV FOR CALIBRATION: PER-FOLD STRATEGY

- Fit model and calibrator on each CV fold separately
- Use fold-wise models to generate (uncalibrated) predictions and fold-wise calibrators to calibrate them
- Average fold-wise calibrated predictions



CV FOR CALIBRATION: PER-FOLD STRATEGY

- Fit model and calibrator on each CV fold separately
- Use fold-wise models to generate (uncalibrated) predictions and fold-wise calibrators to calibrate them
- Average fold-wise calibrated predictions

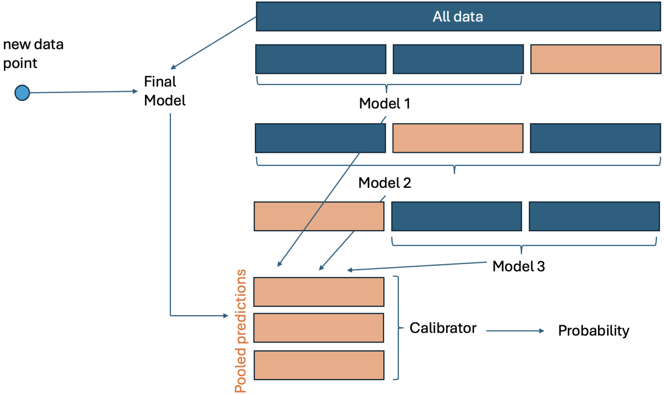


Drawback: Computationally inefficient (requires training k models and k calibrators); none are trained on full data $\Rightarrow$ pessimistic bias in calibration.
Alternative: CV with pooled out-of-fold (OOF) predictions

CV FOR CALIBRATION: POOLED OOF STRATEGY

1. Fit final model on entire dataset
2. Perform CV and generate out-of-fold predictions from CV models
3. Train calibrator on pooled OOF CV predictions (uses all observations)
4. Deploy: predict with final model, calibrate with pooled calibrator

$\Rightarrow$ **Advantage:** One final model + one calibrator, both trained on all data



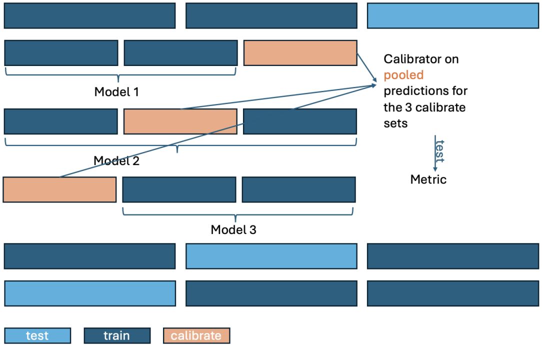
WHEN EVALUATION IS NEEDED

Setting: When the aim is to tune hyperparameters or estimate performance, we require an independent test set (besides the train and calibration set).

Option 1: Simple hold-out split (train set, calibration set, test set)

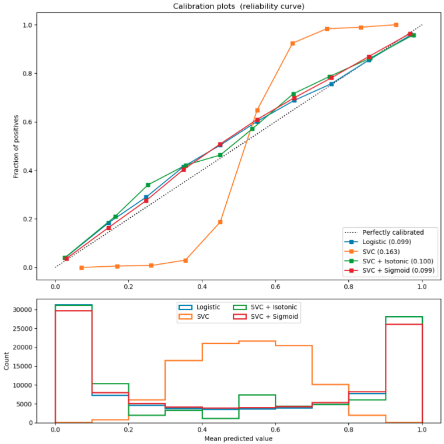
Option 2: Nested Cross-Validation

- Inner loop: Train model & calibrator (per-fold or pooled OOF strategy)
- Outer loop: Evaluate generalization performance



EXAMPLE: CALIBRATION PLOT (REVISITED)

Calibrating a SVC with Platt's scaling and isotonic regression:



⇒ all calibrations improved the original SVC

SUMMARY: WHY IS CALIBRATION IMPORTANT?

- **Interpreting Predicted Probabilities:**
  - Good calibration ensures that predicted probabilities accurately reflect actual risks or frequencies of events.

  $\Rightarrow$ Otherwise, predictions are just scores that happen to lie in $[0, 1]$.
- Instance-based probability evaluation metrics, such as Brier score or log-loss, always measure calibration (plus something else).
- **Example: Medical Diagnosis**
  - Poor calibration: Only half of patients with 90% predicted probability of a disease actually have it.
  - Good calibration: 90% of patients with 90% predicted probability of a disease actually have it.