

Exercises - Solutions

§11.4,11.7: Cross-Validation

Problem 1 Given $P = 100$ and $K = 5$: (a) Each equal validation fold contains $100/5 = 20$ observations. (b) Each fold model trains on $100 - 20 = 80$ observations. (c) One hyperparameter value requires one fit per fold: 5 fits. (d) For 12 hyperparameter values: $12(5) = 60$ model fits.

Problem 2 Fold sizes: 4, 4, 2. Total: $P = 10$. Fold MSE values: 1, 2, 10. (a) Unweighted average: $\frac{1+2+10}{3} = \frac{13}{3} \approx 4.333$. (b) Pooled CV MSE: $\frac{4}{10}(1) + \frac{4}{10}(2) + \frac{2}{10}(10) = 0.4 + 0.8 + 2 = 3.2$. (c) The unweighted fold average treats a two-observation fold as equally important as a four-observation fold. The pooled formula weights each fold by its number of observations. Therefore pooled CV gives every observation equal weight.

Problem 3 The observation lies in F_3 . Therefore the corresponding fold-training set is $T_3 = \{1, \dots, P\} \setminus F_3$. (a) Standardization means and scales must be computed using only T_3 . The observation in F_3 must not contribute. (b) Model coefficients are also fit using only T_3 . (c) After preprocessing and model fitting are complete, the held-out observation is transformed using the training-fold parameters and its loss is evaluated. A global standardizer would use F_3 when calculating its mean and scale. Thus held-out information would influence the fitted pipeline. That is leakage.

Problem 4 The positive class contains only 5% of the observations. Unconstrained random folds may have noticeably different positive proportions, especially with smaller datasets. This can make fold metrics unstable and some folds unusually easy or difficult. Stratification approximately preserves the overall class proportions. With $K = 5$: positive observations per fold: $50/5 = 10$. Negative observations per fold: $950/5 = 190$. Thus each fold contains approximately 10 positive and 190 negative observations.

Problem 5 Measurements from the same patient are statistically related. If some measurements from one patient appear in training while other measurements from that same patient appear in validation, the model may benefit from patient-specific information that would not be available for a genuinely new patient. This can make validation performance overly optimistic. Safer approach: assign entire patients to folds. All four measurements from one patient remain together. Thus the basic group is patient, not individual measurement row.

Problem 6 Random shuffling can place later months in the training set while earlier months appear in validation. Then the model is effectively trained using information from the future relative to the validation observation. That does not mimic the intended deployment task. A time-respecting procedure keeps training observations earlier in time than validation observations. For example: train on an initial historical window; validate on the next block of months; expand or move the training window forward; validate again on a later block. The validation structure should mimic forecasting future data from past data.

Problem 7

```
import numpy as np

x = np.array(
    [-3., -2., -1., 0.,
```

```

    1., 2., 3., 4.]
)

y = np.array(
    [9.2, 3.9, 1.1, 0.2,
     0.9, 4.1, 8.8, 16.2]
)

rng = np.random.default_rng(0)

indices = rng.permutation(len(x))

folds = np.array_split(indices, 4)

def polynomial_features(x, degree):
    columns = []

    for k in range(degree + 1):
        columns.append(x**k)

    return np.column_stack(columns)

def cv_mse(x, y, degree, folds):
    losses = []

    all_indices = np.arange(len(x))

    for val_idx in folds:
        train_idx = np.setdiff1d(
            all_indices,
            val_idx
        )

        Phi_train = polynomial_features(
            x[train_idx],
            degree
        )

        Phi_val = polynomial_features(
            x[val_idx],
            degree
        )

        w = np.linalg.lstsq(
            Phi_train,
            y[train_idx],
            rcond=None
        )[0]

        pred = Phi_val @ w

        losses.extend(
            (pred - y[val_idx])**2
        )

    return np.mean(losses)

for degree in [1, 2, 3, 4]:
    error = cv_mse(

```

```

    x,
    y,
    degree,
    folds
)

print(degree, error)

```

Output:

```

1 50.86346579
2 0.06224242
3 0.24031534
4 0.30339225

```

Thus: $E_{CV}(1) \approx 50.8635$, $E_{CV}(2) \approx 0.06224$, $E_{CV}(3) \approx 0.24032$, and $E_{CV}(4) \approx 0.30339$. The selected degree is 2.

Problem 8 Suppose candidate ridge values are $\lambda_1, \dots, \lambda_M$. For each candidate λ_m and each fold k : 1. define fold-training data T_k and validation data F_k ; 2. compute feature means and scales using only T_k ; 3. standardize T_k using those values; 4. standardize F_k using the same fold-training values; 5. fit ridge coefficients with λ_m using the standardized T_k ; 6. predict F_k and record the losses. Combine the out-of-fold losses to obtain $E_{CV}(\lambda_m)$. Then choose $\lambda^* = \arg \min_{\lambda_m} E_{CV}(\lambda_m)$. After selection: fit the standardizer again using all development data; standardize all development data; fit ridge with λ^* using all development data. Finally: transform the untouched test set using the development-data standardizer and evaluate the final model once. The test data are not used to choose λ^* .