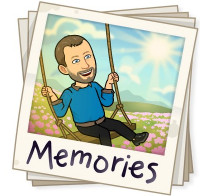


Machine Learning

Textbook: Machine Learning Refined, by Watt, Borhani, and Katsaggelos, 2nd edition:
MLRefined.com

Previous Lecture

- ♦ Generalization concerns performance on new observations, not merely the data used to fit the model.
- ♦ For nested model families, minimum training error cannot increase as model complexity grows.
- ♦ Underfitting occurs when the model family is too restrictive; overfitting occurs when the fitted model adapts too strongly to training-sample peculiarities.
- ♦ A degree- $(P - 1)$ polynomial can interpolate arbitrary responses at P distinct 1D training inputs, so zero training error alone is weak evidence of generalization.
- ♦ Under squared error, expected prediction error decomposes into irreducible noise, squared bias, and variance under stated assumptions.
- ♦ Training data fit parameters, validation data compare model choices, and test data estimate final performance after model selection is complete.



§11.4, 11.7 - Cross-Validation

Last time:

training error could not choose model complexity.

We introduced validation data
to compare hyperparameters.

But a single validation split has two weaknesses:

some observations are withheld from fitting;

the result can depend strongly on one arbitrary split.

Today:

repeatedly change which observations are held out.

This is **cross-validation**.

The Hold-Out Method

The simplest validation method:

split available development data into

training set
and

validation set.

Fit on training data.

Evaluate on validation data.

Use validation performance
to compare hyperparameters.

This is conceptually clean.

But:
one split can be statistically inefficient
when the dataset is small.

Why One Split Can Be Unstable

Suppose the dataset contains
a few unusual observations.

If several happen to fall in the validation set:
validation error may look unusually large.

If they all fall in training:
validation may look unusually easy.

Thus:
validation error depends on the random split.

Cross-validation averages performance
across several different held-out subsets.

K-Fold Cross-Validation

Partition the development observations
into
 K
disjoint folds:
 $F_1, \dots, F_K$.

Requirements:
 $F_j \cap F_k = \emptyset$
for
 $j \neq k$,

and
 $F_1 \cup \dots \cup F_K = \{1, \dots, P\}$.

Each observation belongs
to exactly one validation fold.

One Fold at a Time

For fold k :

validation indices:
 F_k .

Training indices:
 $T_k = \{1, \dots, P\} \setminus F_k$.

Fit a model
 $\hat{f}^{(-k)}$
using only
 T_k .

Then evaluate
 $\hat{f}^{(-k)}$
on
 F_k .

The superscript
 $(-k)$
means:
trained without fold k .

Every Observation Is Evaluated Out of Sample

Observation
 p
belongs to exactly one fold.

Suppose
 $p \in F_k$.

Its cross-validation prediction is
 $\hat{y}_p^{CV} = \hat{f}^{(-k)}(\mathbf{x}_p)$.

Crucially:
observation p
was not used
to fit
 $\hat{f}^{(-k)}$.

Thus every observation receives
an **out-of-fold prediction**.

This is the central cross-validation idea.

Fold Error

For squared-error regression,

fold k error is

$$E_k = \frac{1}{|F_k|} \sum_{p \in F_k} \left(y_p - \hat{f}^{(-k)}(\mathbf{x}_p) \right)^2.$$

For classification:
use the desired held-out loss or metric.

Examples:
cross-entropy;

classification error;

other task-appropriate metrics.

The fitting objective
and evaluation metric
need not be identical.

Combine the Fold Errors

The clean pooled cross-validation error is

$$E_{CV} = \frac{1}{P} \sum_{k=1}^K \sum_{p \in F_k} L\left(y_p, \hat{f}^{(-k)}(\mathbf{x}_p)\right).$$

Equivalently:
$$E_{CV} = \sum_{k=1}^K \frac{|F_k|}{P} E_k.$$

If all folds have equal size:
 $|F_k| = P/K.$

Then:
$$E_{CV} = \frac{1}{K} \sum_{k=1}^K E_k.$$

Why Weight Unequal Folds?

Suppose fold sizes differ.

An unweighted average
$$\frac{1}{K} \sum_k E_k$$

gives every fold
equal influence
regardless of how many observations it contains.

The pooled formula instead gives
each observation equal influence.

Thus:
$$E_{CV} = \sum_k \frac{|F_k|}{P} E_k$$

is the natural sample-average loss
when folds are unequal.

For nearly equal folds:
the distinction is usually small.

But mathematically they are not identical.

The K -Fold Algorithm

For a fixed hyperparameter choice:

1. divide observations into K folds;
2. for each fold k :

fit using all folds except k ;

predict the observations in fold k ;

record their losses;
3. combine all out-of-fold losses.

This requires fitting
 K
models
for one hyperparameter choice.

Use Cross-Validation to Choose a Hyperparameter

Suppose candidate hyperparameters are
 $h_1, \dots, h_M$.

For each candidate:
compute
 $E_{CV}(h_m)$.

Then choose
 $h^* = \underset{h_m}{\operatorname{argmin}} E_{CV}(h_m)$.

Examples of
 h :
polynomial degree;

ridge / lasso strength;

number of basis functions;

later:
kernel parameters.

What Happens After Choosing the Hyperparameter?

During cross-validation:
every fitted model omitted one fold.

But after choosing h^* :
we usually refit the model
using **all available development data**.

Thus:
CV chooses the procedure
and then
all development data fit the final parameters.

If a separate test set exists:
evaluate that final model on the test set
once the procedure is fixed.

Cross-Validation Does Not Eliminate the Test Set

Cross-validation is used to:
compare model choices.

If we repeatedly inspect
cross-validation performance
while designing the model,

then the CV results influence the model-selection process.

Therefore the same CV estimate
should not automatically be treated
as an untouched final performance estimate.

A separate test set provides
a cleaner final evaluation.

For specialized settings without a test set:
nested cross-validation can separate
model selection from performance estimation.

A Running Example

Consider eight observations:

x	-3	-2	-1	0	1	2	3	4
y	9.2	3.9	1.1	0.2	0.9	4.1	8.8	16.2

These are approximately
 $y = x^2$
with small perturbations.

We will compare polynomial degrees

1, 2, 3, 4
using
4-fold cross-validation.

Each fold contains
2
observations.

Create Four Folds

Using one fixed random permutation,
suppose the observation indices
are ordered as
(3,5,4,7,6,1,2,8)
in one-based mathematical indexing.

Then choose:

$$F_1 = \{3,5\},$$

$$F_2 = \{4,7\},$$

$$F_3 = \{6,1\},$$

$$F_4 = \{2,8\}.$$

Every observation appears
in exactly one validation fold.

For each fold:
the remaining six observations
are used for fitting.

Degree 1 Cross-Validation

For each fold:
fit a straight line
using the other six observations.

The four fold MSE values are approximately:
30.8397, 15.4543, 80.8064, 76.3534.

Because all folds have equal size:

$$\begin{aligned} E_{CV}(1) &= \frac{1}{4}(30.8397 + 15.4543 + 80.8064 + 76.3534) \\ &\approx 50.8635. \end{aligned}$$

A straight line performs very poorly.

Degree 2 Cross-Validation

Quadratic fold MSE values:

0.00967, 0.07324, 0.01708, 0.14898.

Average:

$$E_{CV}(2) \\ \approx 0.06224.$$

This is dramatically smaller
than the linear-model CV error.

The quadratic feature family
captures the dominant structure.

Degree 3 Cross-Validation

Cubic fold MSE values:

0.00795, 0.08419, 0.64736, 0.22176.

Average:

$$E_{CV}(3) \\ \approx 0.24032.$$

The cubic is more flexible
than the quadratic.

But its cross-validation error is larger.

More flexibility has not improved
out-of-fold prediction.

Degree 4 Cross-Validation

Quartic fold MSE values:

0.05774, 0.06361, 0.48188, 0.61035.

Average:

$$E_{CV}(4) \\ \approx 0.30339.$$

Again:
more flexibility
produces worse out-of-fold performance.

Among these four candidates:
cross-validation prefers
degree 2.

Cross-Validation Comparison

Polynomial Degree	4-Fold CV MSE
1	50.8635
2	0.06224
3	0.24032
4	0.30339

Choose:
 $d^* = 2.$

Then refit a degree-2 model
 using all eight development observations.

Cross-validation selected the degree.

The final coefficients
 are estimated after that choice
 using all available development data.

Why Not Fit One Model and Reuse It Across Folds?

Suppose we fit once
 using all observations,

then merely calculate the loss
 on each fold.

That is not cross-validation.

Every validation observation
 would already have influenced
 the fitted parameters.

Correct cross-validation requires:
 refit the model separately for every fold.

Each fold creates a different training sample
 and therefore generally
 a different fitted parameter vector.

Preprocessing Must Also Be Refit Inside Each Fold

Suppose features are standardized.

Incorrect procedure:

1. compute mean and standard deviation
 from all development observations;
2. perform cross-validation.

Why incorrect?

Each validation fold contributed
to the preprocessing parameters.

Information leaked across the fold boundary.

Correct procedure:
fit preprocessing using the fold’s training portion only.

Then apply those fitted preprocessing parameters
to the held-out fold.

The Entire Fitting Pipeline Lives Inside Each Fold

For each fold:

fit missing-value rules;

fit scaling parameters;

create any data-dependent feature selection;

fit PCA if PCA is part of preprocessing;

fit model coefficients;

then evaluate on the fold.

General rule:
anything learned from data must be fit without the validation fold.

A transformation that is fixed in advance,
such as
 $x \mapsto (x, x^2)$,

does not need statistical fitting.

But its degree is still a hyperparameter
if degree is being selected.

A Leakage Example

Suppose one fold contains
an unusually large feature value.

If that value is used
when computing the global mean and standard deviation,

then the training transformation
has learned something
about the held-out observation.

The leakage may appear numerically small.

But the principle is exact:
held-out information must not influence any fitted stage.

This becomes especially important
for aggressive feature selection
or dimension reduction.

What Value of K ?

Common choices include:

$$K = 5$$

or

$$K = 10.$$

But there is no universal best value.

Small K :

- fewer model fits;
- larger validation folds;
- each model trains on less data.

Large K :

- more model fits;
- smaller validation folds;
- each model trains on more data.

This creates a computational
and statistical tradeoff.

Training Size in K -Fold Cross-Validation

If folds are equal:

each fold contains

$$P/K$$

observations.

Each fitted model uses

$$P - P/K$$

observations.

Thus:

$$P_{\text{train,fold}} = P(1 - \frac{1}{K}).$$

Examples:

$K = 5$:

train on 80% of the data.

$K = 10$:

train on 90%.

As K grows:

each fold model becomes more similar

in training size

to the final model fit on all development data.

Leave-One-Out Cross-Validation

Extreme case:

$$K = P.$$

Each validation fold contains
exactly one observation.

Each model trains on
 $P - 1$
observations.

This is

leave-one-out cross-validation,

or

LOOCV.

Advantages:

nearly all data used for each fit.

Disadvantages:

requires P fits;

individual fold errors are maximally small-sample;

fits are highly overlapping.

LOOCV is not automatically superior to moderate K .

Repeated K -Fold Cross-Validation

One random division into folds
can still affect the result.

A possible extension:

repeat K -fold cross-validation
with several independently generated fold partitions.

Then average performance
across repetitions.

This can reduce dependence
on one particular partition.

Cost:
many more model fits.

For
 R
repetitions,
 K
folds,

and
 M
hyperparameter candidates:
 RKM
candidate-model fits are required.

Classification: Stratified Folds

Suppose only
10%
of observations belong to the positive class.

A purely random small fold
could contain very few
or even no positive observations.

A common strategy:
stratified cross-validation.

Create folds so that
class proportions are approximately preserved
in every fold.

This does not change the basic CV principle.

It changes how folds are constructed
to make evaluation more stable.

Grouped Data Require Grouped Folds

Suppose we have:
multiple measurements from the same person;

repeated images from the same subject;

several records from the same household.

If related observations are split
between training and validation folds,
the validation task may become artificially easy.

A safer rule:
keep dependent groups together in the same fold.

The definition of an independent observational unit
must reflect the scientific sampling structure.

Time-Ordered Data Are Different

Suppose the goal is
predict the future from the past.

Randomly shuffling all observations
can train on future times
and validate on earlier times.

That does not represent the intended prediction problem.

For time-ordered data:
use time-respecting splits
such as expanding or rolling training windows.

General rule:
cross-validation folds must mimic the intended future use.

Ordinary random K -fold CV assumes
observations are exchangeable enough
for random partitioning to be meaningful.

Cross-Validation Has Sampling Variability

The CV estimate depends on:

the observed dataset;

the fold partition;

the fitted learning procedure.

Therefore:
 E_{CV} is an estimate, not a population constant.

Two candidate models with very similar CV errors
may be practically indistinguishable
given sampling variability.

Do not overinterpret
tiny numerical differences.

Fold-to-Fold Variation Is Informative

Besides the mean CV error,
inspect the fold errors
 $E_1, \dots, E_K$.

Large variation can reveal:

- small-sample instability;
- unusual observations;
- heterogeneous subpopulations;
- sensitivity to the training sample.

However:
fold errors are not independent
because their training sets overlap.

So elementary independent-sample formulas
should not be interpreted too literally
as exact uncertainty statements.

Hyperparameter Search Cost

Suppose:
 $M = 20$
candidate values of
 λ ,

and
 $K = 5$
folds.

Then cross-validation requires
 $20(5) = 100$
model fits.

If we also compare
4
feature families:
 $20(5)(4) = 400$
fits.

Cross-validation is powerful
because it reuses data efficiently.

But:
it can be computationally expensive.

This matters especially
for large nonlinear models.

Cross-Validation in Python: Make the Folds

Use the running example:

```
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)
```

New Python idea:
np
divides an array
into nearly equal pieces.

With eight observations
and four folds:
every fold contains two indices.

Build Polynomial Features

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

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

    return np.column_stack(columns)
```

This function is fixed.

It does not learn
any data-dependent parameters.

The polynomial degree,
however,
is the hyperparameter
being compared.

One Cross-Validation Loop

```
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)
```

Each validation prediction
comes from a model
that did not use that observation.

Compare Polynomial Degrees

```
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
```

Select:

$$d^* = 2.$$

Then fit the degree-2 model
on all eight development observations.

Cross-Validation With Standardization

Inside each fold:

```
X_train = X[train_idx]
X_val = X[val_idx]

mu = np.mean(
    X_train,
    axis=0
)

scale = np.sqrt(
    np.mean(
        (X_train - mu)**2,
        axis=0
    )
)

Z_train = (
    X_train - mu
) / scale

Z_val = (
    X_val - mu
) / scale
```

Notice:

mu

and

scale

come only from

X_train.

The validation fold is transformed
but never used to fit the transformation.

A Useful Pipeline View

For each fold:

raw fold-training data

→

fit preprocessing

→

transform fold-training data

→

fit model

→

transform fold-validation data

→

predict fold-validation data.

Thus cross-validation evaluates
the entire learning procedure,
not merely the final coefficient-fitting formula.

This is why leakage-safe pipelines matter.

After Cross-Validation

Suppose degree 2
wins.

Refit:

```
degree = 2

Phi = polynomial_features(
    x,
    degree
)

w_final = np.linalg.lstsq(
    Phi,
    Y,
    rcond=None
)[0]
```

Now:
all development observations
contribute to the final coefficient estimates.

Cross-validation did not create
one special fold model
to deploy.

It selected
the fitting procedure.

The final model is refit afterward.

A Cross-Validation Checklist

Before running CV:

- identify the observational unit;
- decide whether random folds are scientifically appropriate;
- reserve a final test set if one is available.

Inside every fold:

fit every data-dependent preprocessing step;

fit the candidate model;

evaluate only on the held-out fold.

After CV:

choose hyperparameters;

refit using all development data;

evaluate final performance on untouched test data.

What did we learn?

- ◆ K -fold cross-validation partitions the development data into disjoint folds and gives every observation an out-of-fold prediction.
- ◆ The pooled CV error is the average loss across all out-of-fold predictions; equal-size folds reduce this to the ordinary mean of the fold errors.
- ◆ Hyperparameters are chosen by comparing cross-validation performance, after which the selected model is refit using all development data.
- ◆ Every data-dependent preprocessing step must be fit separately inside each fold to prevent leakage.
- ◆ Larger K uses more observations for each fold fit but requires more model fits; LOOCV is the extreme case $K = P$.
- ◆ Fold construction must reflect the data structure: stratification, grouped folds, or time-respecting validation may be required.



Next

- Nonlinear feature maps can create very high-dimensional representations.
- Explicitly constructing every transformed feature may become expensive.
- In some models we can work with inner products between transformed observations without constructing the transformed vectors themselves.
- This leads to the kernel trick and nonlinear learning through kernels.
- **1.31 - Kernel Methods**