INTRODUCTION

To Statistical Learning

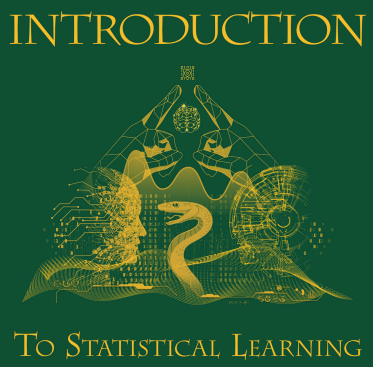
Introduction to Statistical Learning

Model Selection - Part A - Class 5

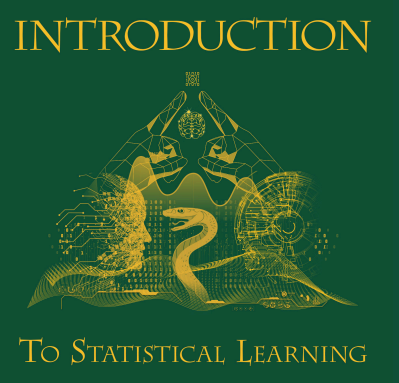
Giora Simchoni

gsimchoni@gmail.com and add #intro2sl in subject

Stat. and OR Department, TAU



Expected Prediction Error



Previously with the Bias-Variance Tradeoff

- For regression, take the standard model: $y = f(x) + \varepsilon \ , \ \varepsilon \sim (0, \sigma^2)$
- Modeling approach (e.g. OLS), given training data T, gives model $\hat{f}(x)$
- Assume we want to predict at new point x_0, and understand our expected (squared) prediction error:

$$\mathbb{E}_{y_0,T}(y_0 - \hat{f}(x_0))^2 = \text{irreducible error} + \text{squared bias} + \text{variance}$$

Expected Prediction Error

$$\mathbb{E}_{y_0,T}(y_0 - \hat{f}(x_0))^2 = \text{irreducible error} + \text{squared bias} + \text{variance}$$

- Note we treat both the training data T (and hence $\hat{f}$) and the response y_0 as random variables in our expectations
- So, more generally we decomposed: $\mathbb{E}_{y_0,T}(L(y_0, \hat{f}(x_0))|X = x_0)$
- Expected prediction error is when we average over all x_0:

$$Err = \mathbb{E}_{x_0,y_0,T}(L(y_0, \hat{f}(x_0))) = \mathbb{E}_X \left[\mathbb{E}_{y_0,T}(L(y_0, \hat{f}(x_0))|X = x_0)\right]$$

- This could also be written as:

$$Err = \mathbb{E}_{x_0,y_0,T}(L(y_0, \hat{f}(x_0))) = \mathbb{E}_T \left[\mathbb{E}_{x_0,y_0}(L(y_0, \hat{f}(x_0))|T)\right] = \mathbb{E}\left[Err_T\right]$$

- Some would say the *conditional* Err_T is even more interesting!

What EPE is for?

1. Model Selection: select between a set of models (e.g. one with 5 parameters and the other with 6 parameters) the one with lowest prediction error
2. Model Assessment: know how accurate the model would be, estimate the prediction error itself

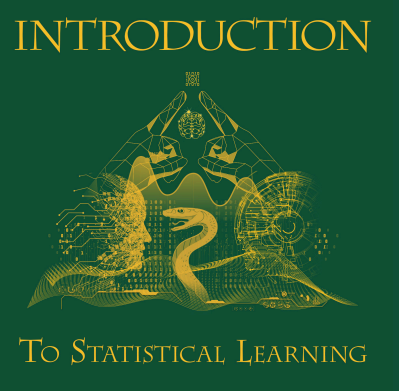
How to estimate EPE?

What do you mean how, why not training error:

$$\overline{err} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}(x_i))$$

1. Data splitting: Train-Validation-Test
2. Cross Validation
3. The Bootstrap
4. Training error + Optimism

Data splitting



Data splitting: Train-Validation-Test

Divide the given sample to 3:

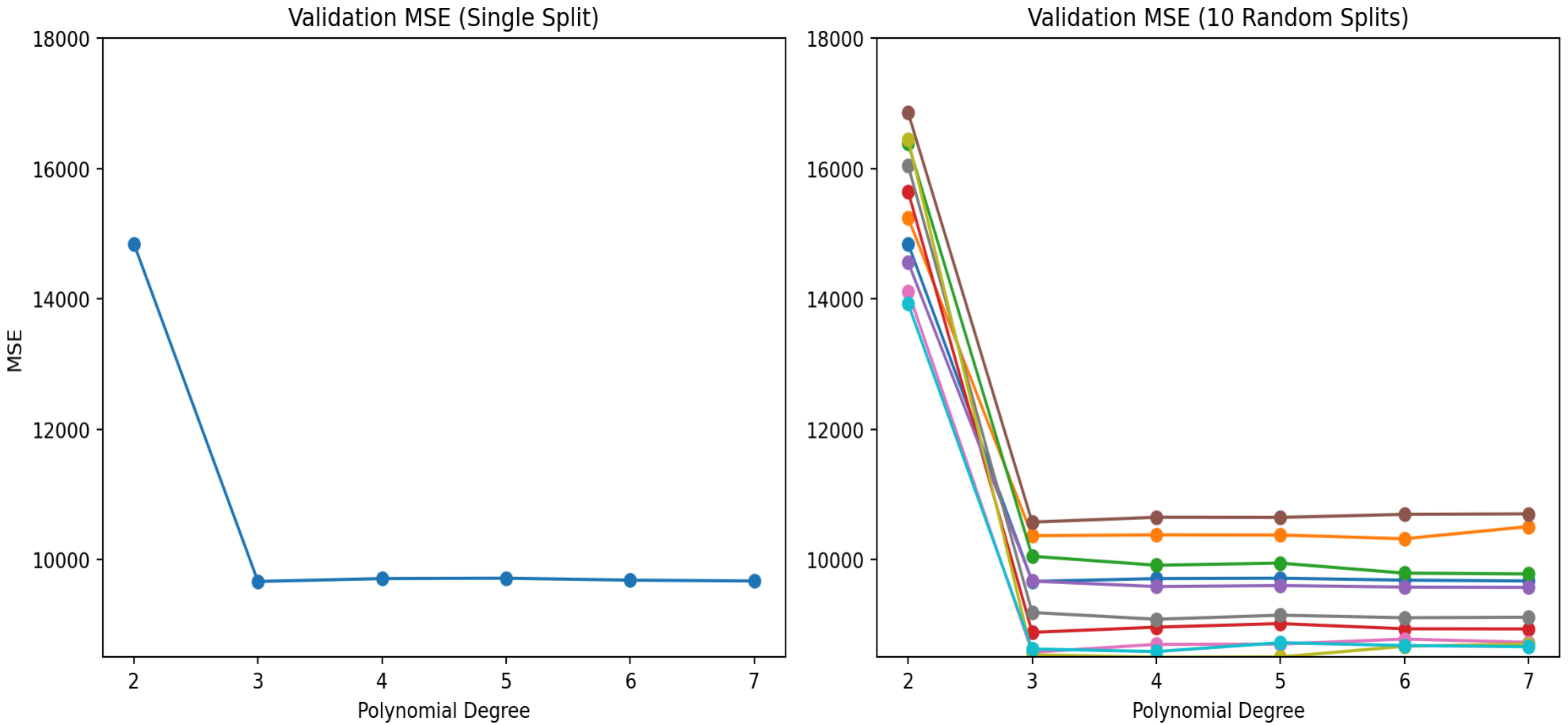
- Training set: learn different models
- Validation set: decide on final model (model selection, tuning)
- Test set: estimate final model's performance (model assessment)

Why not use the final model's performance on *validation* set as estimate?

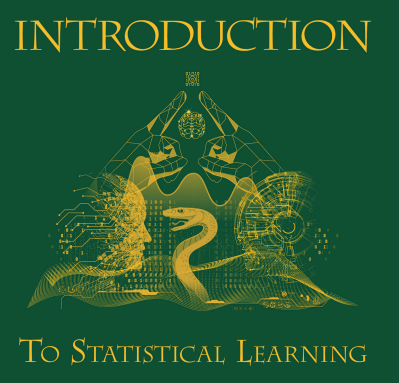
Data splitting: comments

- Test set is used only once!
- Since larger n should improve model's performance, after model selection:
  - unite {training + validation}, train final model and then assess it on the test set
- No clear guidelines: 60-20-20%, 50-25-25%, … (depends on n and nature of f)
- Disadvantages:
  - Lose half the data? Only if we are rich in data
  - Model assessment highly variable
- Additional assumption rarely mentioned

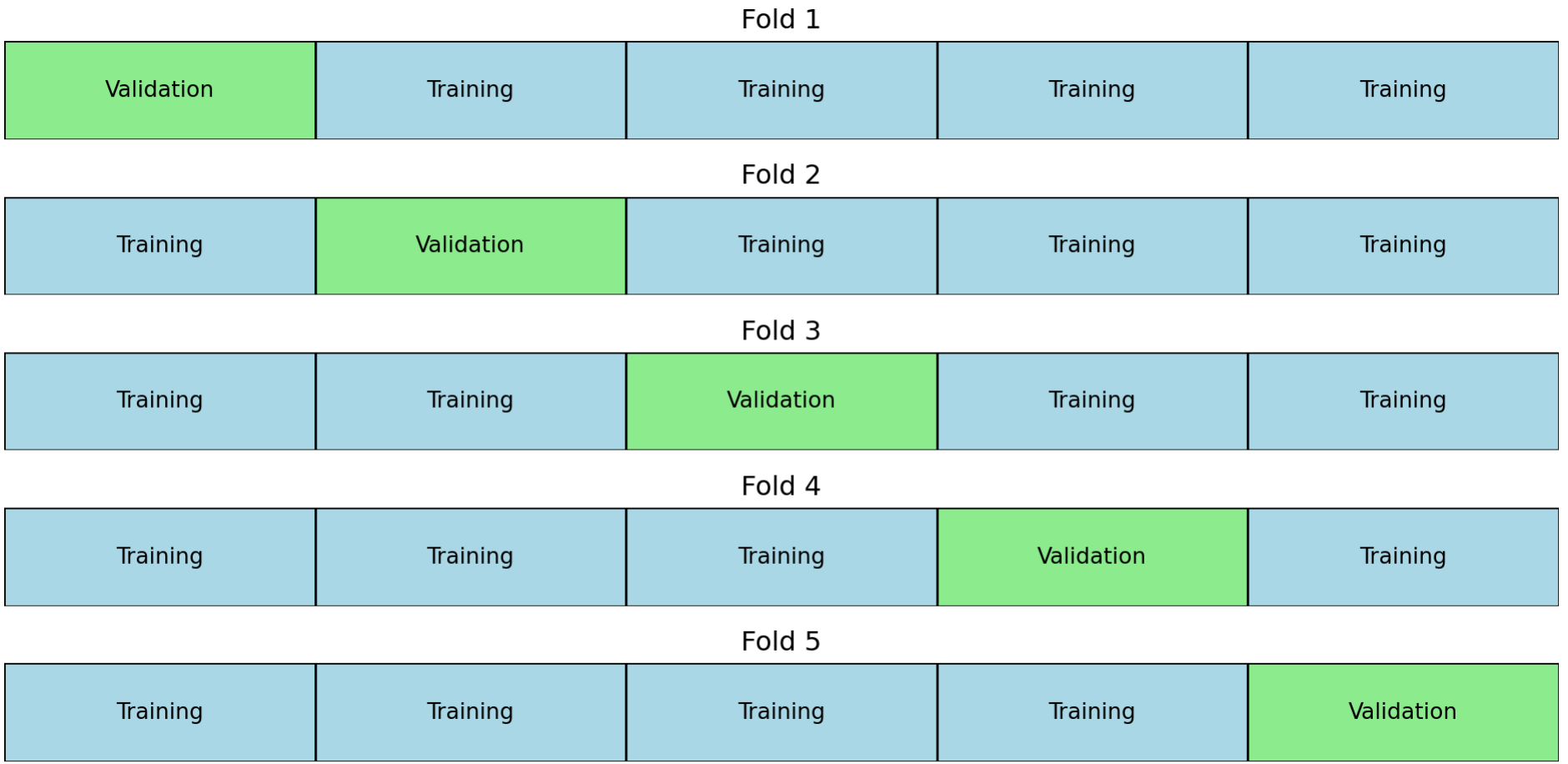
Single split: need 2nd opinion



Cross Validation



K-fold Cross Validation



K-fold Cross Validation

More formally:

- Define a random partition of the n data points into K sets, each of size n/K:

$$\kappa : \{1, \ldots, n\} \to \{1, \ldots, K\}$$

- For $k = 1, \ldots, K$ do:
  - Build a model $\hat{f}^{(-k)}$ using all folds except the k-th fold (total $n \cdot (K-1)/K$ data points)
  - Predict on remaining k-th fold: $L_k = \sum_{i \in k\text{th fold}} L(y_i, \hat{f}^{(-k)}(x_i))$
- CV estimate for prediction error: $CV = \frac{1}{n} \sum_{k=1}^{K} L_k$

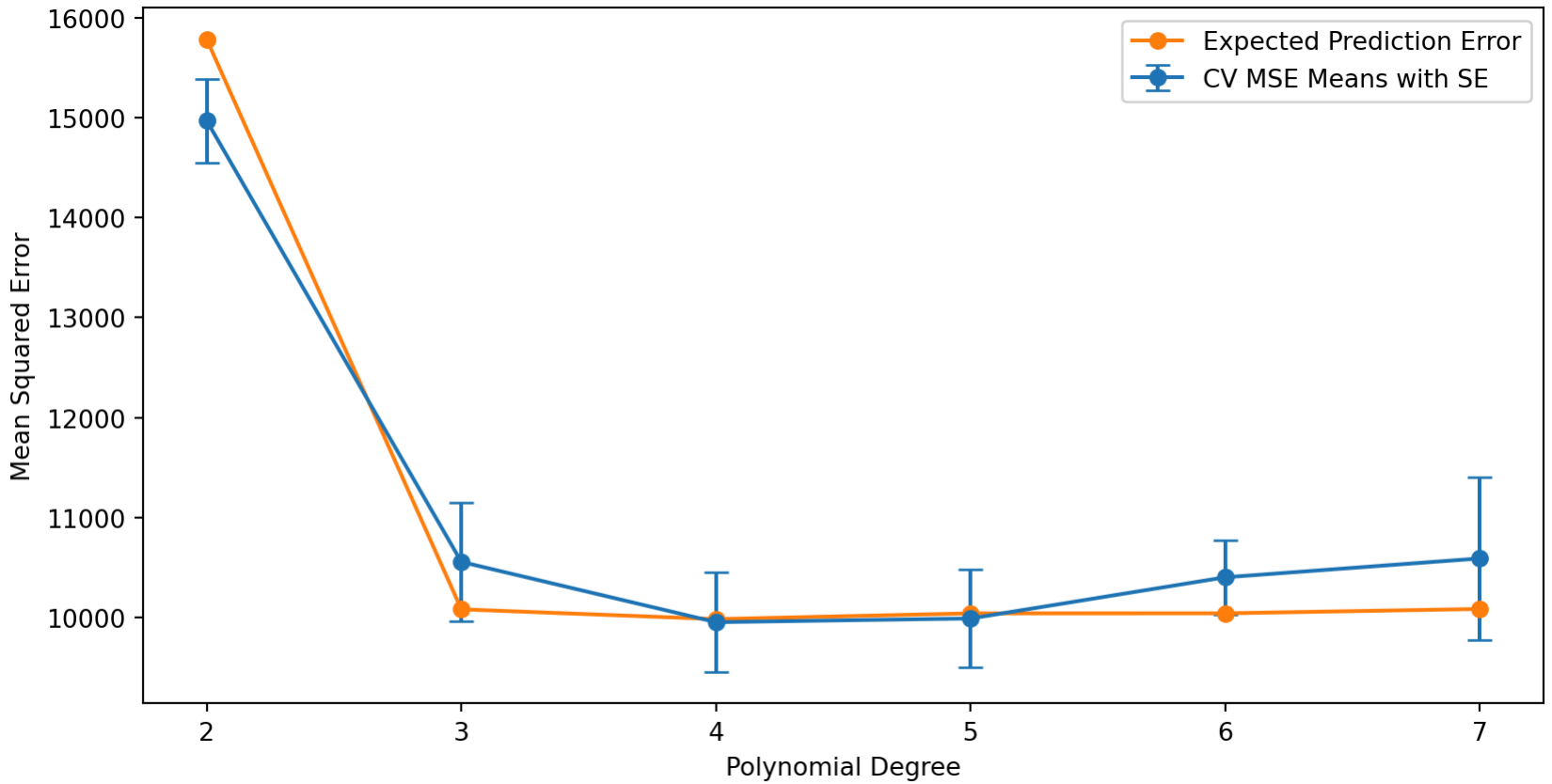
Model selection and assessment with CV

- To perform model selection on tuning parameter λ:

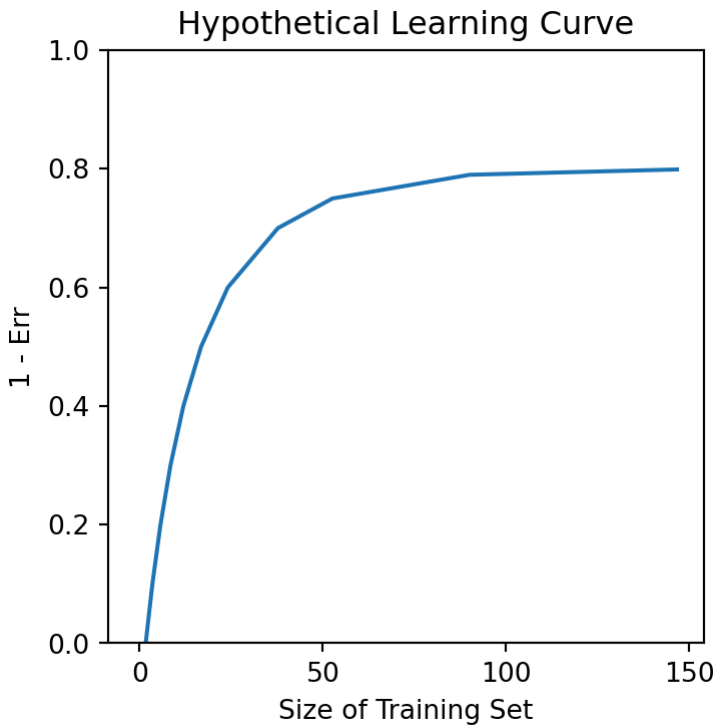
$$L_k(\lambda) = \sum_{i \in k\text{th fold}} L(y_i, \hat{f}_\lambda^{(-k)}(x_i)), \quad CV(\lambda) = \frac{1}{n} \sum_{k=1}^{K} L_k, \quad \hat{\lambda} = \arg\min_\lambda CV(\lambda)$$

- Alternatively use the 1-SE rule (see next slide)
- Model assessment: Optionally, train final model using $\hat{\lambda}$ on all n observations and assess it on a fresh test set

What does CV error estimate?



Can K-fold CV be biased?



- Assume our model requires at least 50 data points to reach its potential
- What will happen if we do 5-fold CV on a training set that contains exactly 50 observations?
- CV error would be biased, over-estimated

Leave-one-out CV (LOOCV)

- So why not $K = n$?
- Train on $n - 1$ observations excluding observation i, test on observation i

- Might be computationally intensive (though see Generalized CV (GCV))
- Test on 1 observations, training sets extremely dependent $\Rightarrow$ low bias but high variance!

CV - Interim Summary

Single-split:

- "$K = 1$"
- Not efficient use of data
- Highly variable, "need 2nd opinion"
- For large n

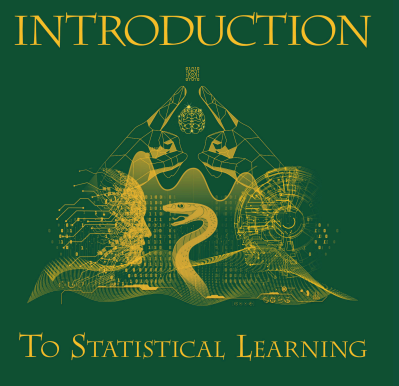
K-fold CV:

- $K = 5, 10$
- More efficient use of data but "Learning Curve"
- Can combine with an additional final test set
- Can become computationally intensive (2+ tuning params, finer grid)
- Probably best compromise

LOOCV:

- $K = n$
- Most efficient use of data, unbiased
- But computational intensive for modern ML techniques, even for 1 param
- High variance

Cross Validation - Common Pitfalls



CV - Common Pitfalls (I)

- Suppose we have many days of sequential inter-daily data (time series)
- We do CV and randomly divide the data into parts
- Can we rely on the variance reduction of the parts?
- Did we leak information between the training part and the test part?

- No and Yes!
- we cannot assume the parts are independent, and we probably have an information leak $\Rightarrow$ we under-estimate the error
- How can we correct this?

CV - Common Pitfalls (II)

- Suppose we have a data with many variables, can we do the following process to estimate the prediction error:
  - On all the data find a small subset of the strongest variables
  - Perform CV using only this set of variables
- Extremely problematic and overly optimistic!
- Where is the information leakage?
- Similar principle: standardizing the data based on all n observations
- How can we correct this?

Use Pipelines

```
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.pipeline import Pipeline

kf = KFold(n_splits=5, shuffle=True, random_state=0)

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('knn', KNeighborsRegressor())
])

param_grid = {
    'feature_selection__k': range(1, n_features + 1),
    'knn__n_neighbors': [3, 5, 7, 9],
    'knn__metric': ['euclidean', 'manhattan']
}

grid_search = GridSearchCV(pipeline, param_grid, cv=kf, scoring='neg_mean_squa
grid_search.fit(X, y)
```