Model Validation

Department of Computer Languages and Systems

General objectives

Given

- a Machine Learning problem and
- several classifiers/regressors applied to such a problem,

we seek answers to the following questions:

- how to assess the results of any classifier/regressor for the given problem?
- how to make sure that the results provided by the classifier/regressor are not artificial and will hold true when the model faces unseen data?

Basics of model validation

- any conclusion is limited to the given problem
- data = learning [training + validation] + test
- the learning/evaluation process should be repeated, and using some statistic of the error distribution
- there are multiple measures to choose a classifier or a regressor among several: classification error, computational complexity (space/time) of the training and/or test, interpretability of the classification model, implementation complexity, …

Basics of model validation (ii)

Given a labeled data set, we have to divide it into three random subsets:

- training set (*learning*), to create the model
- validation set (*learning*), to optimize hyperparameters of the model (e.g., *k* of the *k*-NN, number of neurons in the hidden layer, etc.)
- test set (*evaluation*), to assess the model with data not used in the creation of the model

Generation of subsets

Given a labeled data set, we want to create random subsets (training, validation, test),

- as independent as possible (to minimize overlap)
- as large as possible (to obtain robust results)
- as stratified as possible (to maintain the original proportions of samples per class)

Generation of subsets (ii)

How many samples for learning and how many for testing?

- if we have many samples to learn and few to evaluate, it can lead to overfitting
- if we have many samples to evaluate and few to learn, it can lead to underfitting

Thus the idea is to resample the whole data set in a way that we have enough training samples to approximate the generalization error, tune the hyperparameters controlling the model complexity and reduce the test error

Generation of subsets (iii)

Resampling methods for validating models:

- arbitrarily large sets (rare case)
  - holdout
- moderate or small size set (common case)
  - K-fold cross-validation
  - leaving-one-out
  - 5x2 cross-validation
  - bootstrapping

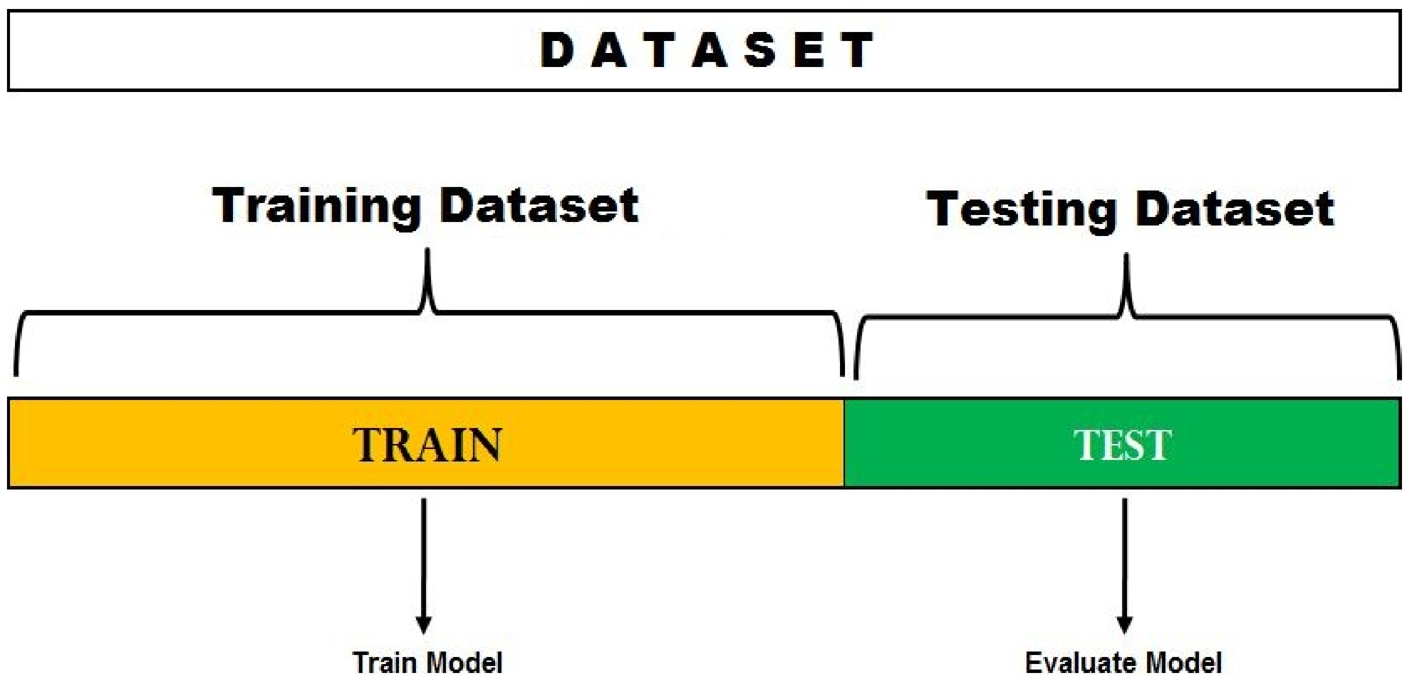
Holdout

Given a data set *X*, we randomly divide *X* into 2 disjoint blocks: one for training (*T*) and one for test/validation (*V*)

$$T \cup V = X$$
$$T \cap V = \emptyset$$

- typically the training data set *T* is bigger than the evaluation data set *V*
- common ratios used for splitting *X* are 60:40, 70:30, 80:20
- we can shuffle the data K different times and repeat the process for each shuffled data set (repeated holdout)
- used when we only have one model to evaluate and no hyperparameters to tune

Holdout (ii)


DATASET
Training Dataset
Testing Dataset
TRAIN
TEST
Train Model
Evaluate Model

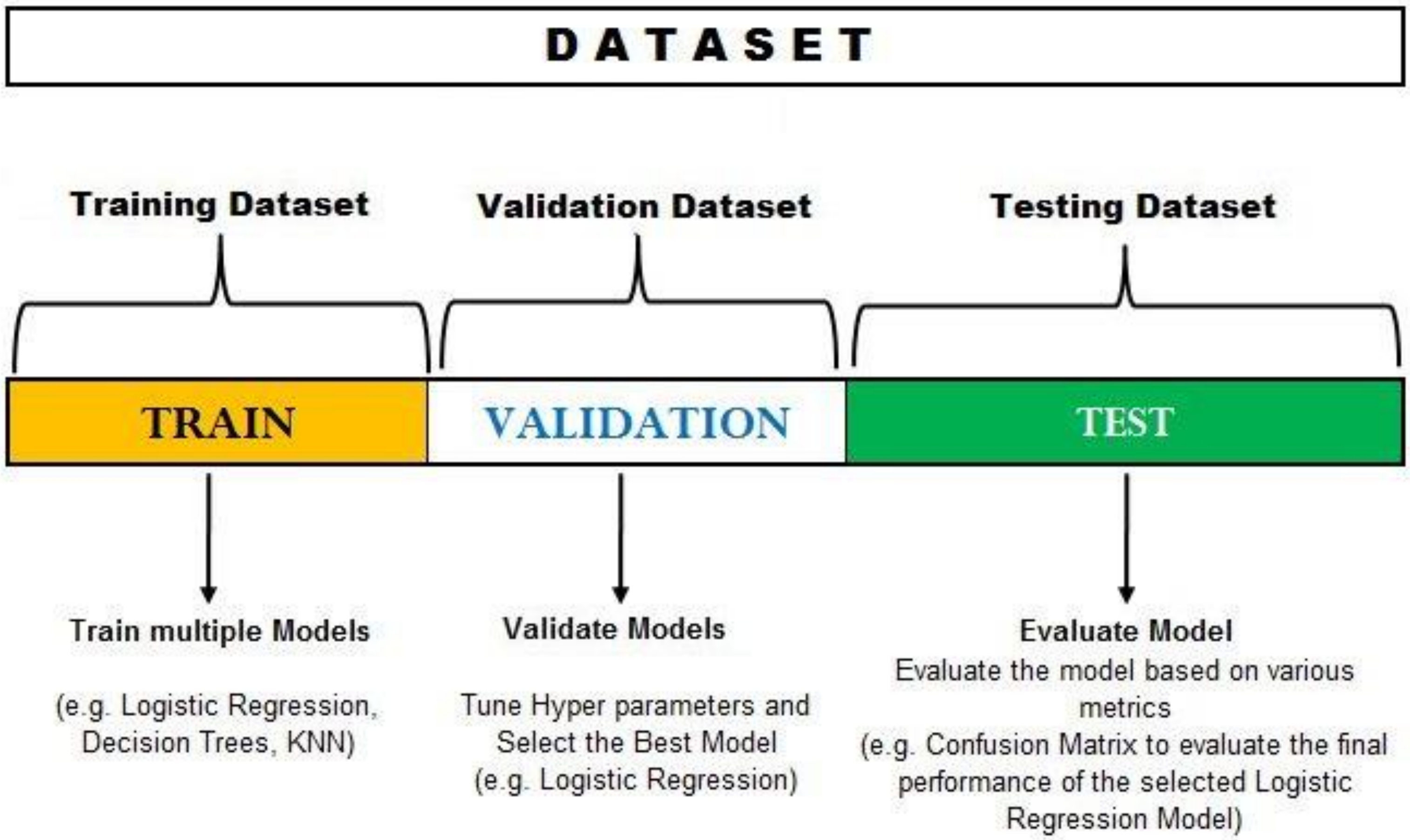
A variation of Holdout

Holdout is not effective for comparing multiple models and tuning their hyperparameters

Solution: splitting of data into not two, but three disjoint sets: training, validation and test

- the training block is divided into a portion for training and another for validation
- the validation set is used to tune the hyperparameters and select the best performing algorithm
- the test set is used to evaluate the final selected model

A variation of Holdout (ii)



K-fold cross-validation

Given a data set *X*, we randomly divide *X* into K equal sized blocks X^1, X^2, …, X^K:

- we leave out a part and train the model on the other K–1 blocks, using the left part to evaluate the model
- this process is repeated K times so that each part is used as testing set
- the results from each fold are then combined and averaged to come up with the final error
- typical values of K are 5 and 10, but there is no formal rule

K-fold cross-validation (ii)

Given a data set *X*, we randomly divide *X* into K equal sized blocks X^1, X^2, …, X^K and generate K training sets and K test sets:

- Iteration 1: $V_1 = \mathrm{X}^1$, $T_1 = \mathrm{X}^2 \cup \mathrm{X}^3 \cup \cdots \cup \mathrm{X}^\mathrm{K}$
- Iteration 2: $V_2 = \mathrm{X}^2$, $T_2 = \mathrm{X}^1 \cup \mathrm{X}^3 \cup \cdots \cup \mathrm{X}^\mathrm{K}$

…

- Iteration K: $V_\mathrm{K} = \mathrm{X}^\mathrm{K}$, $T_\mathrm{K} = \mathrm{X}^1 \cup \mathrm{X}^2 \cup \cdots \cup \mathrm{X}^{\mathrm{K}-1}$

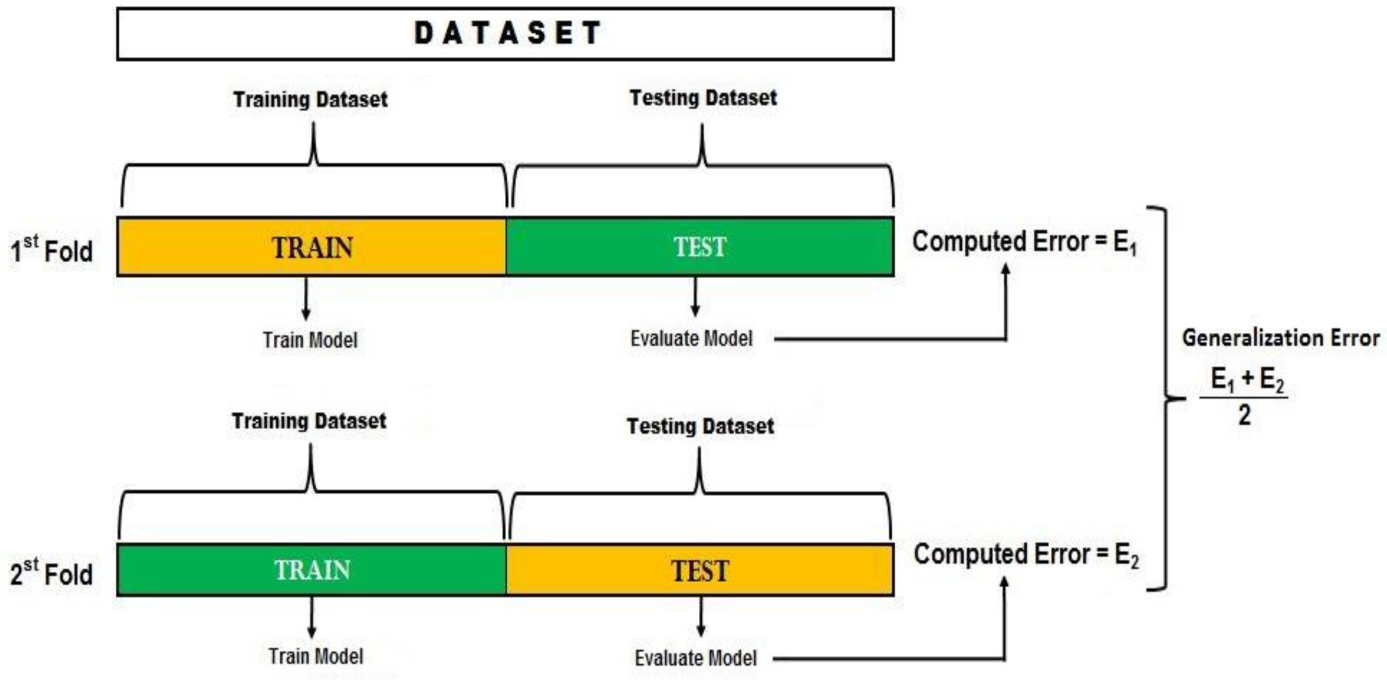
K-fold cross-validation (iii)

Example (2-fold cross-validation)

- we have a subset-A with 50% of the data and a subset-B with the other 50% of the whole data set
- we train the model on subset-A and evaluate the model on subset-B
- we then repeat the process but this time subset-B is for training and subset-A is used as the testing set
- we then average the two results and consider this value as our generalization error

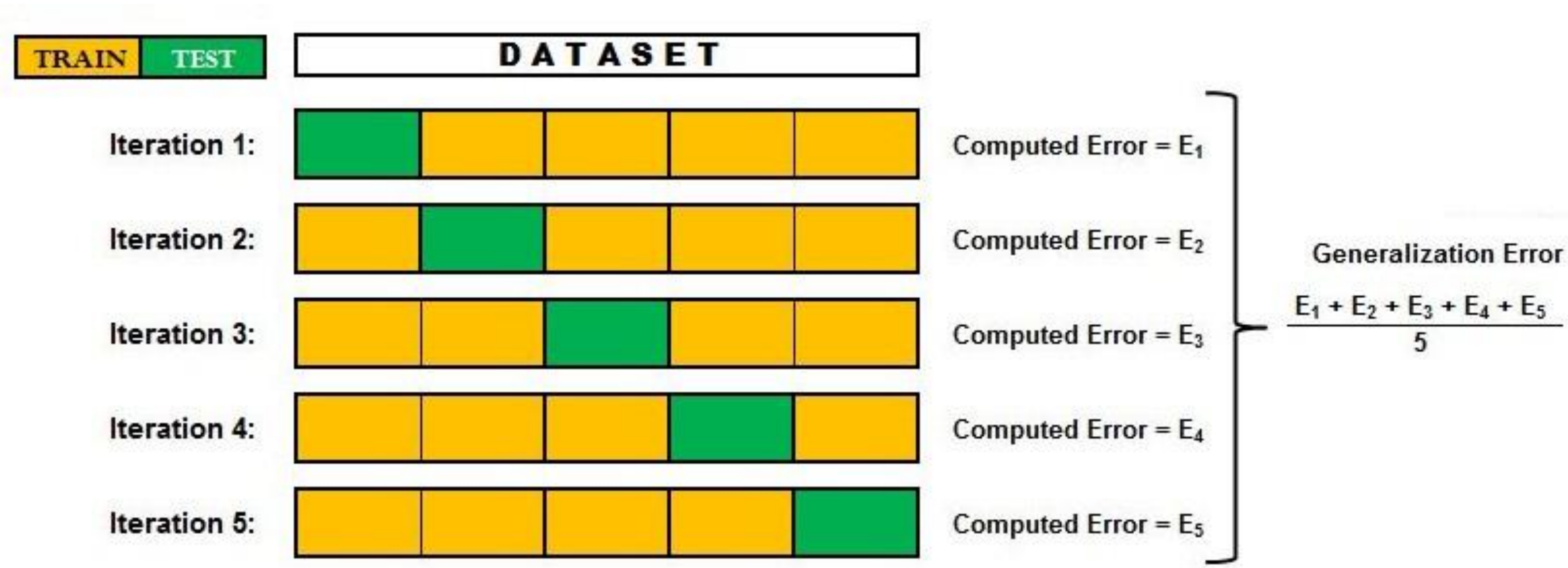
K-fold cross-validation (iv)

Example (2-fold cross-validation)



K-fold cross-validation (v)

General case (e.g. 5-fold cross-validation)



K-fold cross-validation (vi)

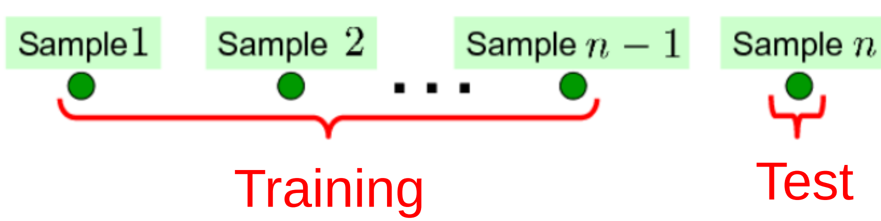
Limitations

- test sets V_i are small
- any two training sets T_i, T_j share K–2 partitions
- as K increases, the size of T_i increases
- as $|X|$ increases, K can be lower (we reduce importance of limitations)

Leaving-one-out

Given a data set *X* with *n* samples, the leaving-one-out method is a particular case of K-fold cross-validation:

- we divide the whole data set *X* into *n* blocks
- at each iteration, one sample is for testing and the remaining *n*–1 samples are used to train the model
- we then average the *n* results and consider this value as our generalization error



Leaving-one-out (ii)

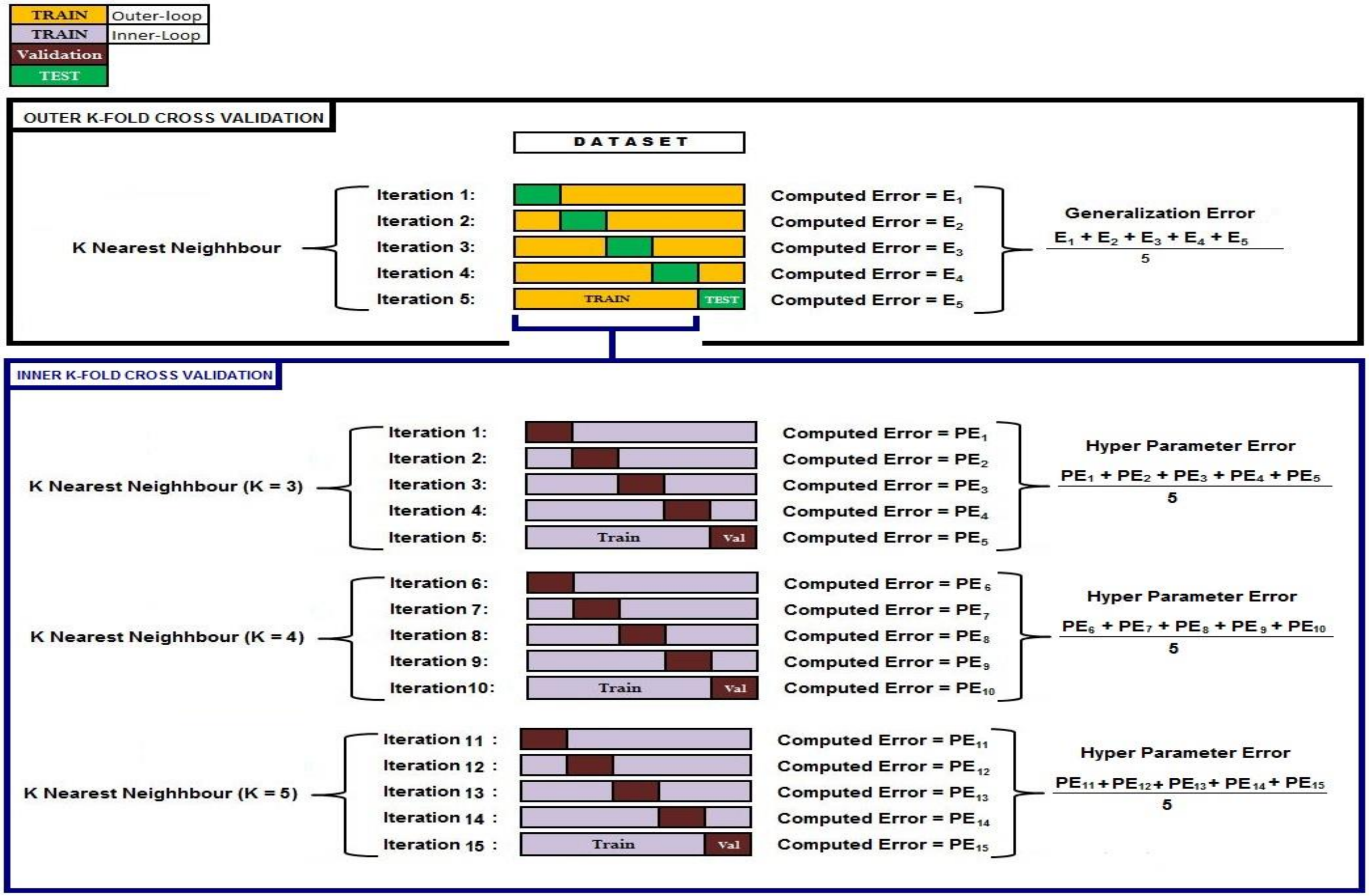
- it is appropriate when you have a small data set since it can be a time-consuming process to use when n is large
- it can also be time-consuming if a model is particularly complex and takes a long time to fit to a data set
- it does not allow stratification

Nested K-fold cross-validation

This is a variation of K-fold cross-validation:

- an inner K-fold cross-validation is performed within each training fold of the outer cross-validation, often to tune hyperparameters during model evaluation
- at each iteration of the outer cross-validation, the inner fold is divided into K equal random parts
- the inner cross-validation is repeated K times where at every iteration, K–1 parts form the training fold while the remaining K part forms the validation fold

Nested K-fold cross-validation (ii)



5×2 cross-validation

Given a data set *X*, we randomly divide *X* into 2 equal sized blocks 5 times:

$$\mathbf{X}_1^1, \mathbf{X}_1^2, \mathbf{X}_2^1, \mathbf{X}_2^2, \ldots, \mathbf{X}_5^1, \mathbf{X}_5^2$$

- we then generate 10 training sets (T_i) and 10 test sets (V_i):
  - $T_1 = \mathbf{X}_1^1, V_1 = \mathbf{X}_1^2 \rightarrow$ Compute Error = E_1
  - $T_2 = \mathbf{X}_1^2, V_2 = \mathbf{X}_1^1 \rightarrow$ Compute Error = E_2

  … …
  - $T_9 = X_5^1, V_9 = X_5^2 \rightarrow$ Compute Error = E_9
  - $T_{10} = X_5^2, V_{10} = X_5^1 \rightarrow$ Compute Error = E_{10}

$$\mathrm{E} = \sum \mathrm{E}_i$$

5×2 cross-validation (II)

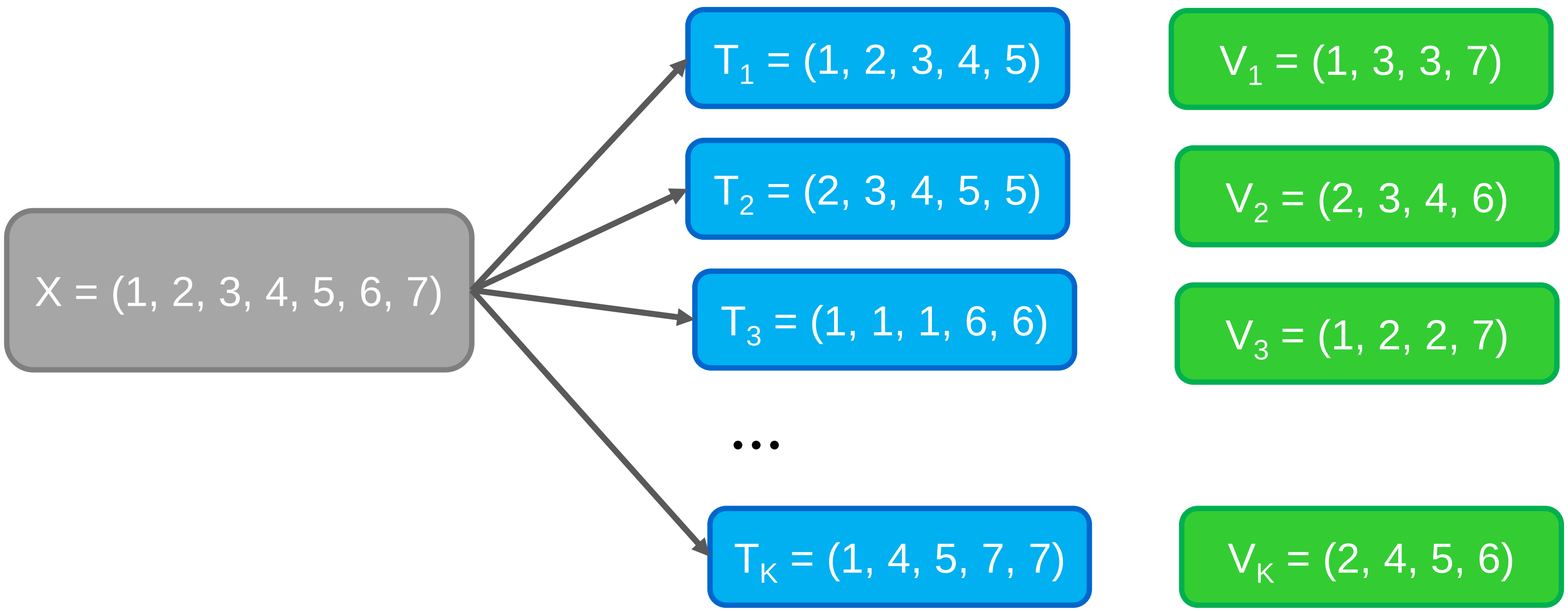
It is possible divide *X* more than 5 times (partitions), but …

- sets share many instances (overlap)
- error rates become dependent
- more error rates do not provide new information

Less than 5 partitions are too few and insufficient to establish an error distribution

Bootstrapping (i)

Given a data set *X*, we generate K pairs of sets (T_i, V_i) by resampling *X* with replacement



Bootstrapping (ii)

- One of the most appropriate methods with small size data sets
- Some drawbacks:
  - higher levels of overlap than in cross-validation
  - its estimated errors are more dependent