COMPSCI 589
Lecture 15: Advanced Experiment Design

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

Data Generating Processes

- In supervised machine learning, we generally assume that the instances in the training dataset are generated independently and identically from a joint probability distribution $P^*(Y, \mathbf{X})$ over feature vectors $\mathbf{X}$ and class labels Y.
- However, in supervised machine learning we care about making predictions on future data, not training data.
- We generally assume that future data instances are generated independently and identically from a joint probability distribution $Q^*(Y, \mathbf{X})$ over feature vectors $\mathbf{X}$ and class labels Y.
- $P^*(Y, \mathbf{X})$ could be the same or different from $Q^*(Y, \mathbf{X})$ depending on the task.

Model Selection and Evaluation

- To solve the problem of optimally selecting capacity hyper-parameters and estimating generalization error of learned models, we need to use appropriate methodology and construct learning experiments carefully.
- **Guiding Principle:** Data used to estimate generalization error can not be used for any other purpose including model training and hyper-parameter selection or the results of the evaluation will be **biased**.

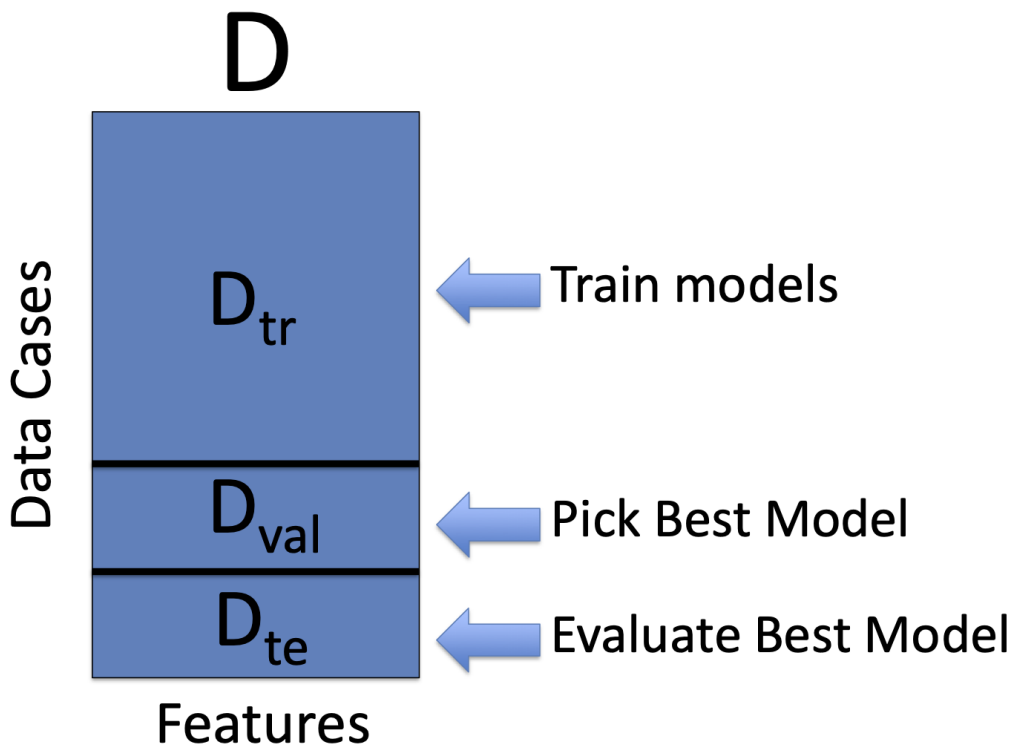
The Train-Validation-Test Experiment

Given a training dataset $\mathcal{D}_{tr}$ sampled from P^*, a validation set $\mathcal{D}_{val}$ sampled from Q^*, a test set $\mathcal{D}_{te}$ sampled from Q^* and a finite set of hyper-parameter values $\{h_1, ..., h_L\}$ to select from:

- For each hyper-parameter value h_i, use the training dataset $\mathcal{D}_{tr}$ to train the model obtaining $f_i(\mathbf{x})$.
- We next compute the validation error rate Val_i of each trained model $f_i(\mathbf{x})$ using the validation set $\mathcal{D}_{val}$.
- Determine the hyper-parameter value h_* that achieves the minimum validation error rate.
- Re-train the model on $\mathcal{D}_{tr} + \mathcal{D}_{val}$ using h_* and denote the corresponding trained model by $f_*(\mathbf{x})$.
- Compute the test error rate of $f_*(\mathbf{x})$ using the test set $\mathcal{D}_{te}$, which is an unbiased estimate of the expected error under Q^*.

Train-Validation-Test Partitioning: Example

Given a single dataset $\mathcal{D}$, we can apply partitioning to implement the train-validation-test experiment design, noting caveats with respect to the OOD problem.



Note: data cases must be shuffled before block partitioning.

Train-Validation-Test Partitioning: Implementation

- The scikit-learn `model_selection` module implements a method for (pseudo) randomly splitting a dataset into two parts called `train_test_split`.
- Given a single dataset, it needs to be applied twice to perform train-val-test partitioning.
- Assume that p is the proportion of the dataset to use for testing and q is the proportion to use for the validation set. We call the method as follows:

```
p=0.2; q=0.2; r=0

Xl, Xte, yl, yte = train_test_split(X, y, test_size=p, random_state=r)
Xtr, Xval, ytr, yval = train_test_split(Xl, yl, test_size=q/(1-p), random_state=r)
```

[4] ✓ 0.0s Python

Train-Validation-Test: Limitations

- When partitioning a single dataset, applying the Train-Validation-Test scheme forces a trade-off between the amount of data available for training a model, selecting hyper-parameters, and evaluating the final model.
- To train the best model for a given set of hyper-parameters, we want a large training dataset.
- To be able to accurately select hyper-parameters, we want a large validation set.
- To be able to accurately assess the performance of the model with the optimal hyper-parameters, we want a large test set.
- Unless we start with a large dataset, we can't do all three of the above at the same time. What else can we do?

The K-Fold Cross Validation-Test Experiment

- The Cross Validation-Test experiment design is a very commonly used alternative to train-validation-test that can make better use of the data allocated to training and validation.
- This experiment design works by partitioning a single learning dataset into K blocks.
- It then rotates through using each block in turn as the validation set and the remaining blocks as the training set.
- Hyper-parameters are selected using average validation loss. This tends to decrease the variance of the validation scores.
- We preserve a single held-out test set exclusively for evaluation of the final model.

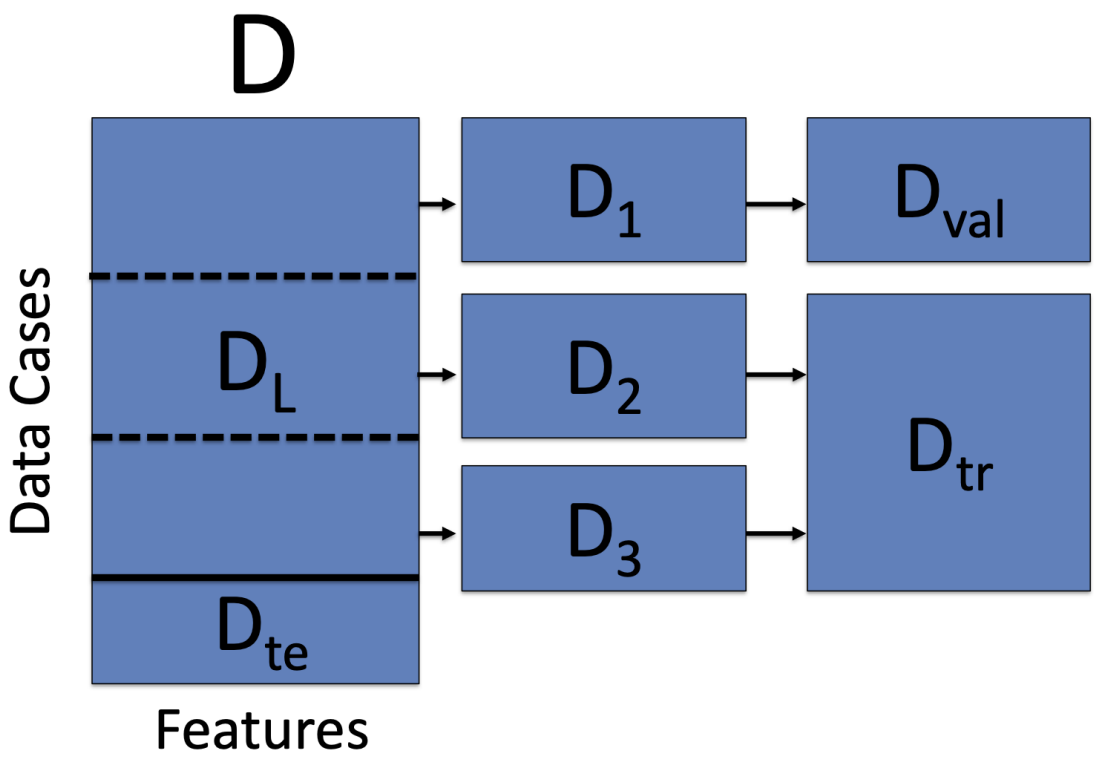
The K-Fold Cross Validation-Test Experiment

Given a value of K, a learning dataset $\mathcal{D}_L$ sampled from P^*, a test set $\mathcal{D}_{te}$ sampled from Q^* and a finite set of hyper-parameter values $\{h_1, ..., h_L\}$ to select from:

- Randomly partition $\mathcal{D}_L$ into a set of K equal sized blocks $\mathcal{D}_1, ..., \mathcal{D}_K$.
- For each cross validation fold $k = 1, ..., K$:
  - Let $\mathcal{D}_{val} = \mathcal{D}_k$ and $\mathcal{D}_{tr} = \mathcal{D}_L - \mathcal{D}_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on $\mathcal{D}_{tr}$ for each choice of hyper-parameters h_i.
  - Compute validation score Val_{ik} of M_{ik} on $\mathcal{D}_{val}$.
- Select hyper-parameters h_* minimizing $\frac{1}{K}\sum_{k=1}^{K} Val_{ik}$ and re-train model on $\mathcal{D}_L$ using these hyper-parameters, yielding final model M_*.
- Estimate generalization performance by evaluating error/accuracy of M_* on $\mathcal{D}_{te}$.

Example: 3-Fold Cross Validation-Test Partitioning

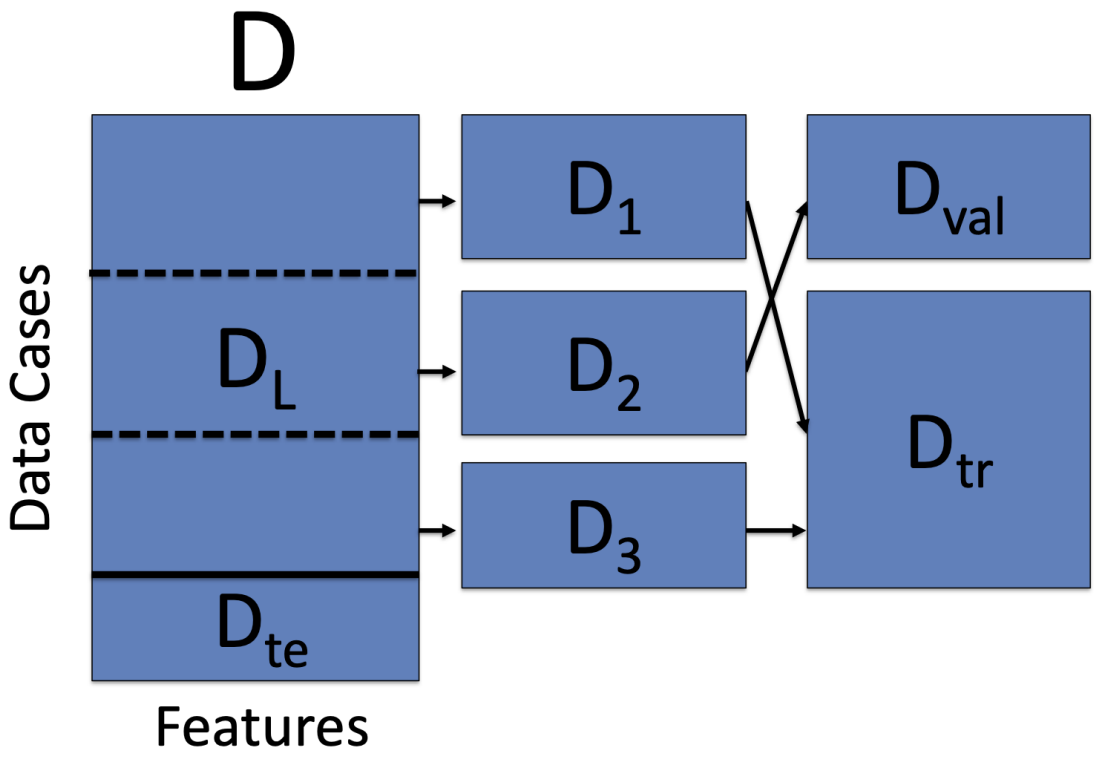
First Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning.

Example: 3-Fold Cross Validation and Test Partitioning

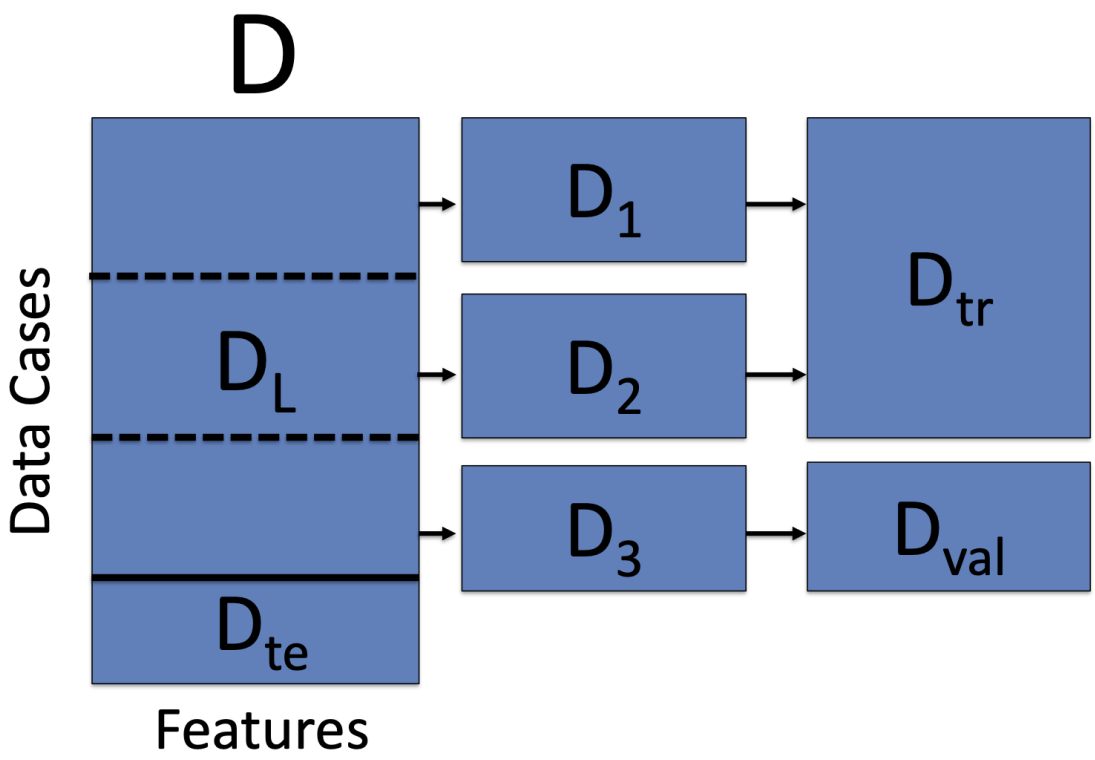
Second Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning.

Example: 3-Fold Cross Validation and Test Partitioning

Third Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning.

K-Fold Cross Validation-Test Special Cases

- Typical applications of K-Fold Cross Validation use $K = 5$ or $K = 10$.
- An extreme case of this design uses $K = N$, often called *leave-one-out* or *LOO* cross validation.
- This results in N folds and a validation set of size 1.

K-Fold Cross Validation-Test: Implementation

- The scikit-learn `model_selection` module implements a method for K-fold random partitioning called `KFold`.
- Given a single dataset, it needs to be applied with `train_test_split` to implement full cross validation-test partitioning.
- Assume that p is the proportion of the dataset to use for testing and K is the number of folds. We call the methods as follows:

```
p=0.2; K=5; r=0

Xl, Xte, yl, yte = train_test_split(X, y, test_size=p, random_state=r)
kf = KFold(n_splits=K,random_state=r, shuffle=True)

for i, (train_index, val_index) in enumerate(kf.split(Xl)):
    Xtr  = Xl[train_index,:]
    Xval = Xl[val_index,:]
    ytr  = yl[train_index]
    yval = yl[val_index]
```

✓ 0.0s Python

K-Fold Cross Validation-Test Limitations

- The K-fold cross validation estimate of generalization performance is biased high due to the fact that the training process sees a fraction $(K-1)/K$ of the data when fitting each model.
- This motivates using K as large as possible so $(K-1)/K$ is as close as possible to 1.
- However, a K-fold cross validation experiment is roughly K times more expensive to perform than a train-val-test experiment (there are some special cases for LOO with certain models). The increased cost motivates using a small value of K.
- If we have a model that is very expensive to train, we might only be able to afford to use $K = 2$ or $K = 3$, which can result in high bias in the validation error estimates.

Random Resampling Validation-Test

- The Random Resampling Validation-Test experiment design is a less commonly used alternative to cross validation-test that also uses multiple train-validation splits.
- This experiment design works by randomly partitioning a learning set such that a fraction p of the data are in the training set and $(1-p)$ of the data are in the validation set.
- This partitioning is repeated completely independently K times. Unlike K-fold cross validation, the size of the validation set and the number of repetitions are decoupled. This allows using smaller K for expensive models while incurring less bias by making p larger than $(K-1)/K$.
- hyper-parameters are selected using average validation loss. This tends to decrease the variance of the validation scores.
- We preserve a single held-out test set exclusively for evaluation

Random Resampling Validation-Test

- Given a value of K, a learning dataset $\mathcal{D}_L$ sampled from P^*, a test set $\mathcal{D}_{te}$ sampled from Q^* and a finite set of hyper-parameter values $\{h_1, ..., h_L\}$ to select from:
- For sample $k = 1, ..., K$:
  - Randomly partition $\mathcal{D}_L$ into $\mathcal{D}_{tr}$ and $\mathcal{D}_{val}$ (80/20, 90/10, etc).
  - Learn M_{ik} on $\mathcal{D}_{tr}$ for each choice of hyper-parameters h_i.
  - Compute Val_{ik} of M_{ik} on $\mathcal{D}_{val}$.
- Select hyper-parameters h_* minimizing $\frac{1}{K}\sum_{k=1}^{K} Val_{ik}$ and re-train model on $\mathcal{D}_L$ using these hyper-parameters, yielding final model M_*.
- Estimate generalization performance by evaluating error/accuracy of M_* on $\mathcal{D}_{te}$.

Random Resampling Validation-Test: Implementation

- The scikit-learn `model_selection` module implements a method for random resampling called `ShuffleSplit`.
- Given a single dataset, it needs to be applied with `train_test_split` to implement full random resampling validation-test partitioning.
- Assume that p is the proportion of the dataset to use for testing and K is the number of sampled train/val splits:

```
p=0.2; K=5; r=0

Xl, Xte, yl, yte = train_test_split(X, y, test_size=p, random_state=r)
ss = ShuffleSplit(n_splits=K,random_state=r)

for i, (train_index, val_index) in enumerate(ss.split(Xl)):
    Xtr  = Xl[train_index,:]
    Xval = Xl[val_index,:]
    ytr  = yl[train_index]
    yval = yl[val_index]
```

✓ 0.0s Python

Random Resampling Validation-Test: Limitations

- Using small K and large p does come at a cost.
- While the bias in generalization performance estimates is lower since each model is trained on more data, the variance of the generalization performance estimates will be larger because each validation set is smaller.

Data Partitioning Considerations

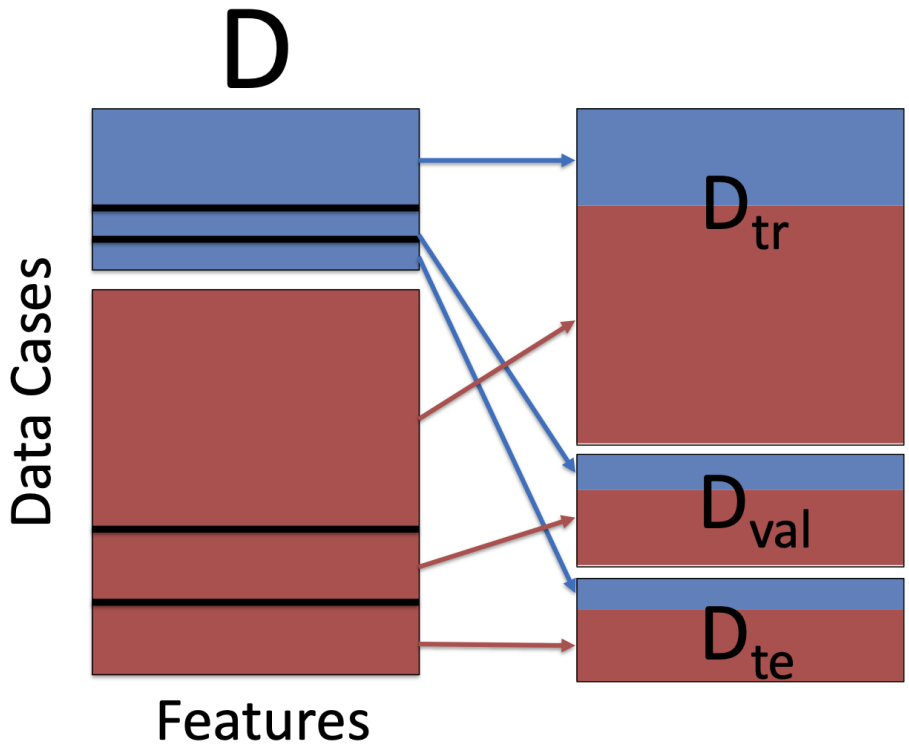
- The experiment designs that we have introduced all assume that data are generated IID from a generative process.
- This leads to simple, random data partitioning strategies being sufficient when designing experiments.
- However, if the data generating process has additional structure, we may need to design data partitioning strategies more carefully.

Stratification

- If we need to partition a dataset in the classification setting where the classes are highly imbalanced, the minority class may not be well represented in smaller validation and test sets leading to high variance error estimates.
- To address this problem, we can use stratified random partitions instead of completely random partitions.
- In a stratified random partition, we first split the data cases by class. We then partition the instances within each class (train-val or cross validation).
- Lastly, we combine the per-class partitions into overall train, validation and test sets.
- This ensures that there are a proportional number of instances from each class in each partition.

Stratified Train-Val-Test

Stratified Train-Val-Test for a Binary Classification Problem



Note that the order of the data cases needs to be randomly shuffled within classes before partitioning.

Stratification: Implementation

- Stratified train-test partition: Call `train_test_split` with the `stratify` parameter set to `y`.
- Stratified Cross Validation: Use the `StratifiedKFold` method.
- Stratified Random Resampling Validation: Use the `StratifiedShuffleSplit` method.

Time Series Data

- If we have time series data, the predictive performance estimated using random train, validation and test partitions can significantly mis-estimate the performance on future data if the time series is autocorrelated.
- In this case, it makes more sense to use a temporal partition of the data than a random partition.
- For example, we could use the first 80% of instances in time order as the training set, the next 10% of instances as the validation set, and the most recent 10% of instances as the test set.

Example: Stock Price Prediction

Time Series Data

- In this case, the validation set performance is an estimate of how well the model works on data from the relative future of the training set.
- The test set performance in an estimate of how well the model works on data from the relative future of both the training and validation sets.
- The resulting performance will be closer to how the model actually performs on future data than if fully random splits were used.
- A drawback of this approach is that it precludes the use of cross validation or resampling methods.

Hierarchical Data

- Another common type of structure in real-world data is hierarchical nesting.
- For example, in recommender systems we have data about people's ratings for a set of items (books, movies, other products).
- Each person will have multiple correlated ratings. There are thus two levels of generalization possible.
- We can ask how well a model predicts future ratings for people whose data was used to train the model.
- We can also ask how well the model predicts ratings for new people not seen during training.

Hierarchical Data: Nested Temporal Split

- If we want to assess how well a model predicts future ratings for a person whose data was used to train the model, we need to use a nested temporal partitioning.
- For each person in the dataset, we split that person's ratings into train, test, and validation sets in time order.
- We train on all of the training ratings, selecting hyper-parameters using all of the validation ratings, and test the final model on all of the test ratings.
- As with the basic temporal split, this has the problem that we can't use cross validation or resampling methods.

Hierarchical Data: Between Person Split

- If we want to assess how well a model predicts ratings for new people not seen during training, we need to split the data at the person level.
- We can randomly assign individuals in the dataset to the train, validation, and test sets in desired proportions and perform a train validation-test experiment.
- We could instead apply a cross validation-test experiment design at the person level.
- In small datasets with few people, it is common to use a leave-one-person-out cross validation experiment design, the generalization of leave-one-out design applied at the data case level.