COMPSCI 389
Introduction to Machine Learning

Evaluation Part 4

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Model Evaluation (Review)

- Often ML texts evaluate models by doing the following:
  - Partition the data into train/test.
  - Train the model on the training data.
  - Evaluate the model on the testing data.
  - Report a performance metric and a number representing the *uncertainty* in this performance metric.
    - Format: performance $\pm$uncertainty

	Model	MSE	RMSE	MAE
0	k-NN k=1 sigma=None	1.104 ± 0.075	1.051 ± 0.029	0.803 ± 0.029
1	k-NN k=100 sigma=None	**0.565 ± 0.041**	**0.752 ± 0.020**	**0.586 ± 0.020**
2	k-NN k=110 sigma=90	0.565 ± 0.041	0.752 ± 0.020	0.586 ± 0.020

Algorithm Evaluation

- Recall that the previous method evaluates the performance of a single model (learned with each algorithm).
- To evaluate algorithms, we must evaluate them with different training sets.

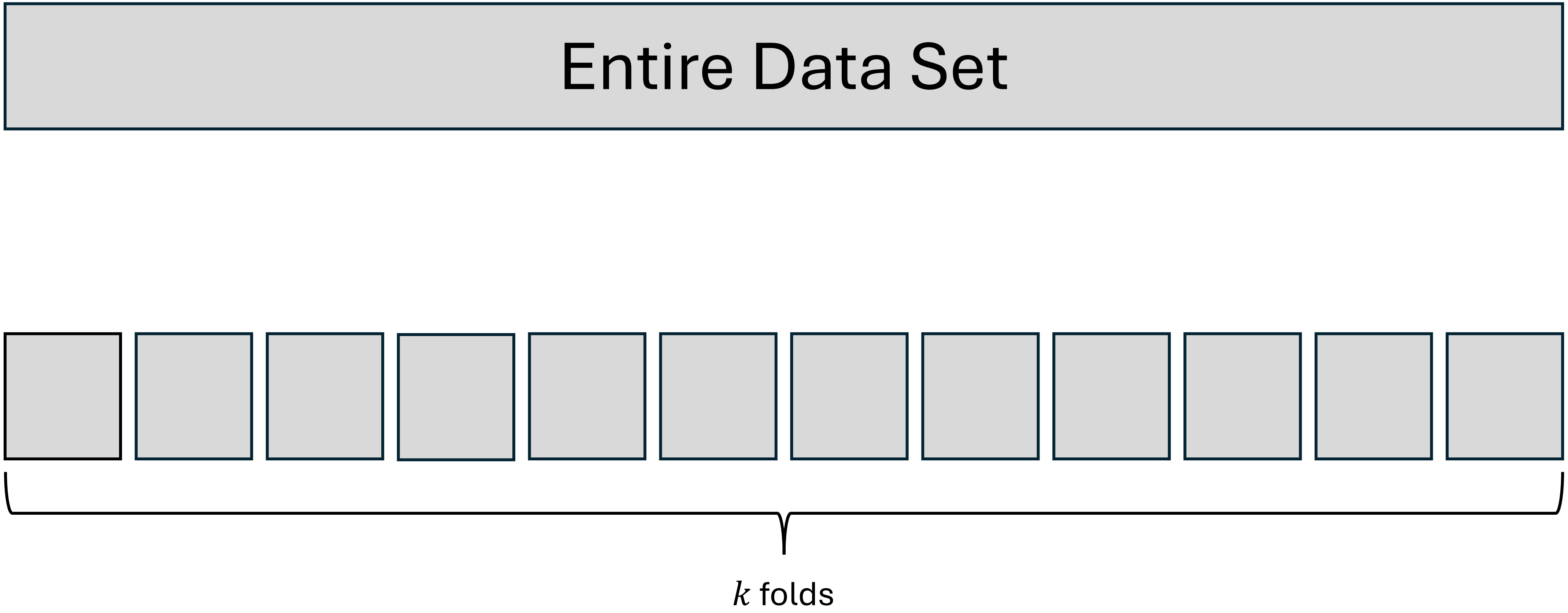
Algorithm Evaluation (Ideal)

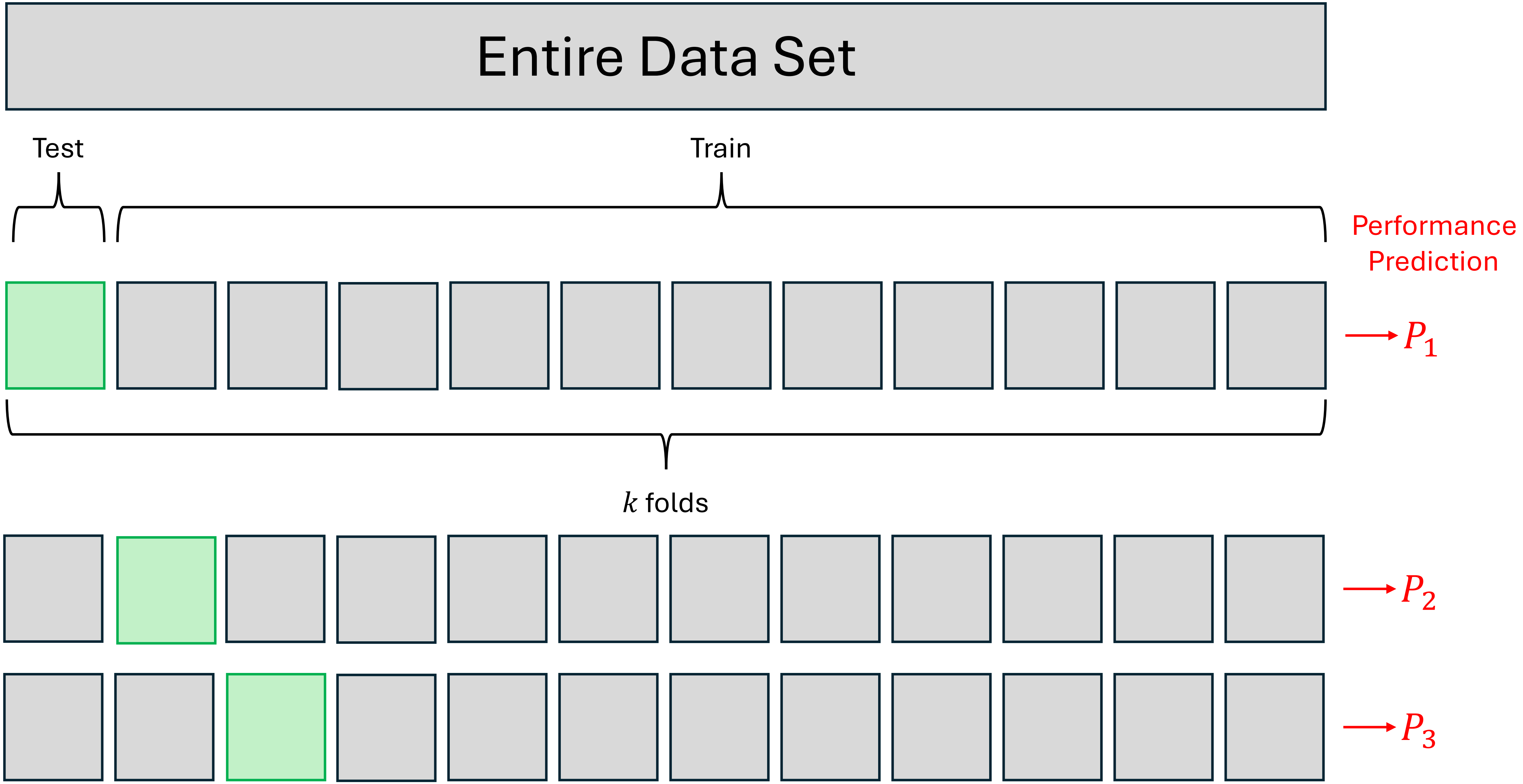
In practice, we can't do this step!

- Specify a number of trials, `num_trials`
- For each trial i in $1, \ldots, \mathrm{num\_trials}$ do:
  - Sample a data set (ideally independent of the data sets for other trials)
  - Split the data set into training and testing sets
  - Use the ML algorithm to train a model on the training set.
  - Use the trained model to make predictions for the testing set.
  - Compute the sample performance metric (e.g., sample MSE) for the test set. Call this Z_i.
- Compute and report the average sample MSE.
- Compute and report the standard error of $Z_1, \ldots, Z_{\mathrm{num\_trials}}$.

Cross-Validation

- **Idea**: Repeatedly define different parts of the data set to be training and testing data.
  - Different training sets result in different models.
  - The testing set for each model will always be independent of the data used to train the model.
- To do this, we will split the data D into k equally-sized subsets.
  - Each of these subsets is called a *fold*.
  - This k is not related to the k in nearest neighbor.
- We will train on all but one fold and test on the held-out fold.
  - These individual evaluations on test sets containing one fold have high variance!
  - We can average these high-variance evaluations to obtain a better estimate of performance.


Entire Data Set
k folds



Repeat for $P_1, \ldots, P_k$ Performance Estimate = mean($P_1, \ldots, P_k$) Uncertainty quantification = SE($P_1, \ldots, P_k$)

K-Fold Cross-Validation Pseudocode

- **Input:** Dataset `D`, Number of folds `k`, Machine Learning Algorithm `ML_Algo`
- **Output:** Cross-validated performance estimate

Procedure:

1. Split `D` into `k` equal-sized subsets (folds) `F1, F2, ..., Fk`.
2. For `i` from 1 to `k`:
   - Set aside fold `Fi` as the validation set, and combine the remaining `k-1` folds to form a training set.
   - Train the model `M` using `ML_Algo` on the `k-1` training folds.
   - Evaluate the performance of model `M` on the validation fold `Fi`. Store the performance metric `P_i`.
3. Calculate the average of the performance metrics: `Average_Performance = mean(P_1, P_2, ..., P_k)`.
4. Optionally, calculate other statistics (like standard deviation or standard error) of the performance metrics across the folds.

Leave-One-Out (LOO) Cross-Validation

- The number of folds equals the number of points in the data set.
- Each test set contains only a single point!
- Provides the best estimates of performance.
- Often too computationally intensive to perform.

k-Fold Cross-Validation Implementation

```python
from sklearn.model_selection import KFold
```

```python
# Choose number of folds for k-fold Cross-Validation
k = 20
kf = KFold(n_splits=k, shuffle=True, random_state=1)
```

```python
display(kf)
```

```
KFold(n_splits=20, random_state=1, shuffle=True)
```

k-Fold Cross-Validation Implementation

```
from sklearn.model_selection import KFold
```

```
# Choose number of folds for k-fold Cross-Validation
k = 20
kf = KFold(n_splits=k, shuffle=True, random_state=1)
```

```
display(kf)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
```

```
KFold(n_splits=20, random_state=1, shuffle=True)

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   10    44    45 ... 43267 43290 43296]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   40    93   134 ... 43246 43256 43261]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [    3    23    34 ... 43262 43286 43288]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [    8    19    25 ... 43277 43293 43299]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   11    33    58 ... 43255 43282 43292]

TRAIN: [    0     1     3 ... 43300 43301 43302] TEST: [    2    22    24 ... 43271 43284 43298]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   21    36    55 ... 43241 43249 43275]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   26    46    62 ... 43223 43231 43265]

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   29    35    43 ... 43229 43234 43242]

TRAIN: [    1     2     3 ... 43300 43301 43302] TEST: [    0    13    31 ... 43274 43278 43281]
```

k-Fold Cross-Validation Implementation

```python
from sklearn.model_selection import KFold
```

```python
# Choose number of folds for k-fold Cross-Validation
k = 20
kf = KFold(n_splits=k, shuffle=True, random_state=1)
```

```python
display(kf)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    mse_score = mse_for_fold(train_index, test_index, model, X, y)
    print("MSE Score for this fold:", mse_score)
```

```python
# Function to compute MSE for each fold
def mse_for_fold(train_index, test_index, model, X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    return mean_squared_error(y_test, predictions)
```

```python
# Choose number of folds for k-fold Cross-Validation
k = 20
kf = KFold(n_splits=k, shuffle=True, random_state=1)
```

```python
display(kf)
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    mse_score = mse_for_fold(train_index, test_index, model, X, y)
    print("MSE Score for this fold:", mse_score)
```

```
KFold(n_splits=20, random_state=1, shuffle=True)

TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   10    44    45 ... 43267 43290 43296]
MSE Score for this fold: 0.5807234989808185
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   40    93   134 ... 43246 43256 43261]
MSE Score for this fold: 0.5630048290694765
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [    3    23    34 ... 43262 43286 43288]
MSE Score for this fold: 0.555346701084036З
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [    8    19    25 ... 43277 43293 43299]
MSE Score for this fold: 0.6129428000450592
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   11    33    58 ... 43255 43282 43292]
MSE Score for this fold: 0.5933726084007112
TRAIN: [    0     1     3 ... 43300 43301 43302] TEST: [    2    22    24 ... 43271 43284 43298]
MSE Score for this fold: 0.564414182778922б
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   21    36    55 ... 43241 43249 43275]
MSE Score for this fold: 0.57366751853279
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   26    46    62 ... 43223 43231 43265]
MSE Score for this fold: 0.5764896819599702
TRAIN: [    0     1     2 ... 43300 43301 43302] TEST: [   29    35    43 ... 43229 43234 43242]
MSE Score for this fold: 0.5443248559898528
TRAIN: [    1     2     3 ... 43300 43301 43302] TEST: [    0    13    31 ... 43274 43278 43281]
MSE Score for this fold: 0.5647024320496056
```

k-Fold Cross-Validation for Weighted k-Nearest Neighbor ($k = 300, \sigma = 100$)

- Average the mse_score values from each fold (code not shown).
- Compute the standard error (code not shown).

```
Average MSE: 0.571
MSE Standard Error: ±0.004
```

- Notice that this is fundamentally different from what we evaluated before.
  - **Model evaluation**: Use one train-test split, report the performance and uncertainty on the test set.
  - **Algorithm evaluation**: Use k-fold cross-validation to estimate the performance (and report the uncertainty) if an algorithm were to be applied to a new data set of a given size.

End