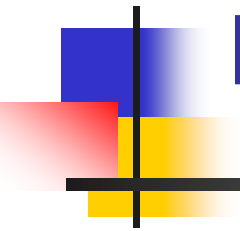


# Data Science: Evaluation-1

Won Kim

2022

# Roadmap: End-to-End Process

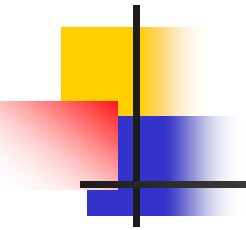
- 1. Objective Setting
- 2. Data Curation
- 3. Data Inspection
- 4. Data Preparation
- 5. Data Analysis
- 6. Evaluation
- 7. Deployment

# Roadmap: Evaluation

- Motivation
- Evaluation Methods
- Ensemble Learning
- Evaluation Metrics

# Acknowledgments


- https://bcourses.berkeley.edu/courses/1377158/files/61598112/download?verifier=5lpPFMM751cYXH0vVWOfWI2KzLhiU6pDNXeprQd5&wrap=1

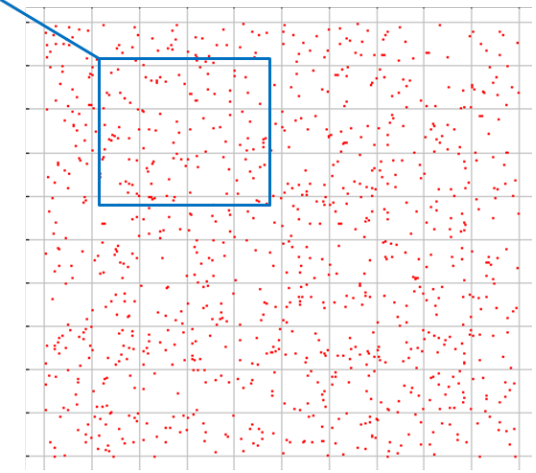


# Motivation

- For supervised learning, many algorithms (predictive models) are available.
- Each model has a number of parameters that should be set.
- No model is ever perfect, that is, every model results in error.
- Need to choose the best model (and parameters), or tune the models for performance improvement.

# Source of Error: Sample-Based Modeling

- Most datasets are samples from an infinite population.
- We are most interested in models of the population, but we have access only to a sample of it.



- For datasets consisting of (X,y)

  features X + label y

  a model is a prediction y = f(X)
- We train on a training sample D and we denote the model as $f_D(X)$

# Bias and Variance

- Our data-generated model $f_D(X)$ is a statistical estimate of the true function $f(X)$.
  - Because of this, it is subject to bias and variance.
- bias: If we train models $f_D(X)$ on many training sets $D$, bias is the expected difference between their predictions and the true $y$'s.

$$Bias = \mathrm{E}[f_D(X) - y]$$

where E[] is taken over points X and datasets D

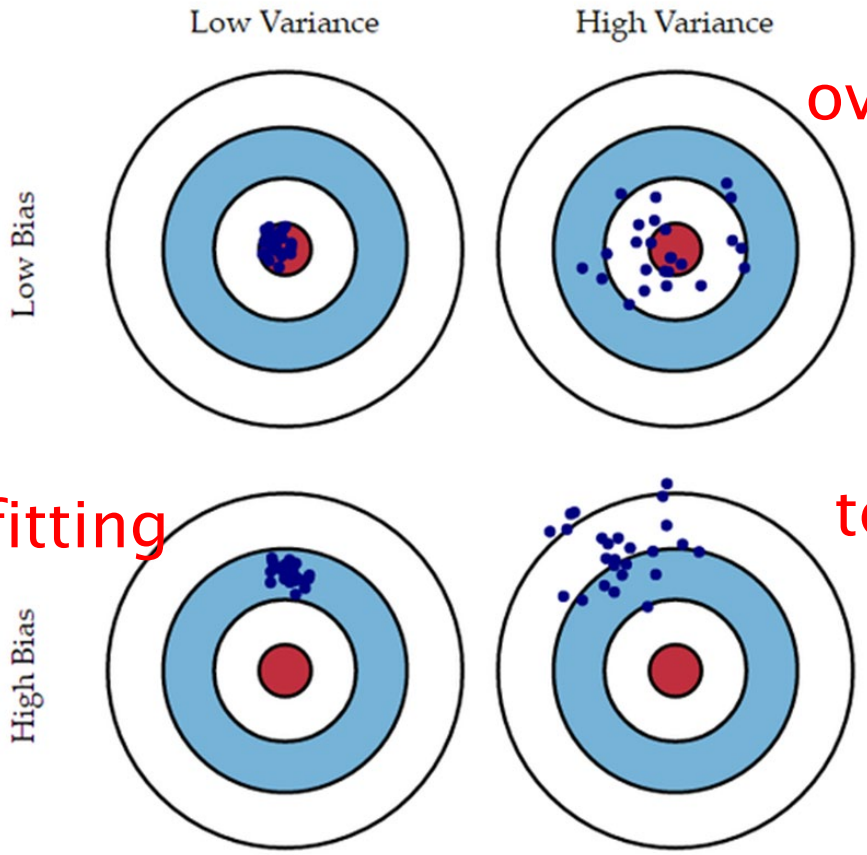
- variance: If we train models $f_D(X)$ on many training sets $D$, variance is the difference among the estimates

$$Variance = \mathrm{E}[(f_D(X) - \bar{f}(X))\text{^}2]$$

where $\bar{f}(X) = \mathrm{E}[f_D(X)]$ is the average prediction on X.

# Illustration of Bias and Variance

- Bias: distance from the bullseye
- Variance: distance between estimates




overfitting

underfitting

terrible

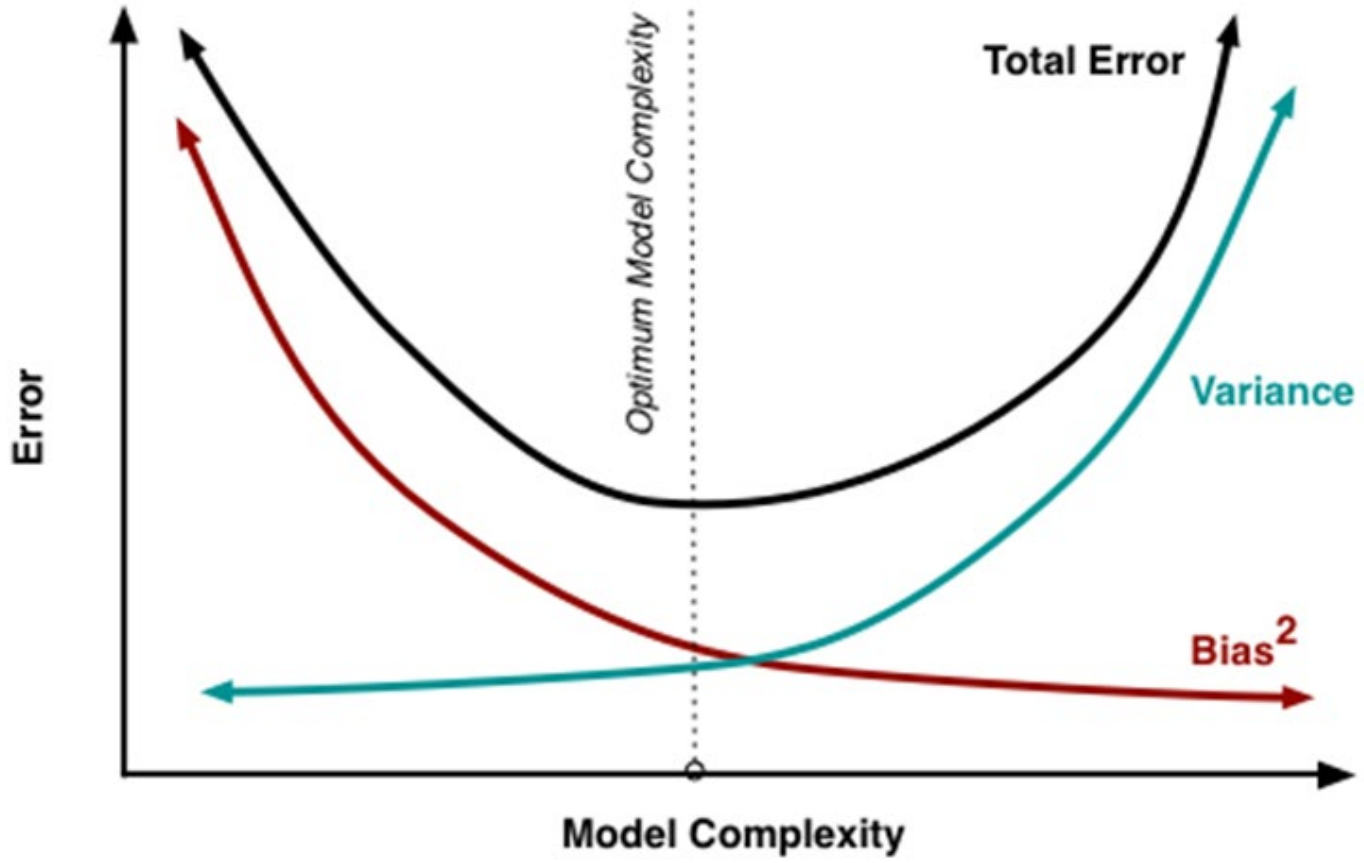
# Model Overfitting and Underfitting

- The total expected error is

  $bias^2 + variance$

- Because of the bias-variance tradeoff, we need to balance these two contributions.
- If variance strongly dominates, it means there is too much variation between models. This is called overfitting.
- If bias strongly dominates, then the models are not fitting the data well enough. This is called underfitting.

# Minimum Total Error and Optimal Complexity



# Bias and Variance Tradeoff

- There is usually a bias-variance tradeoff caused by model complexity.
- Simple models (few parameters) have higher bias, but lower variance.

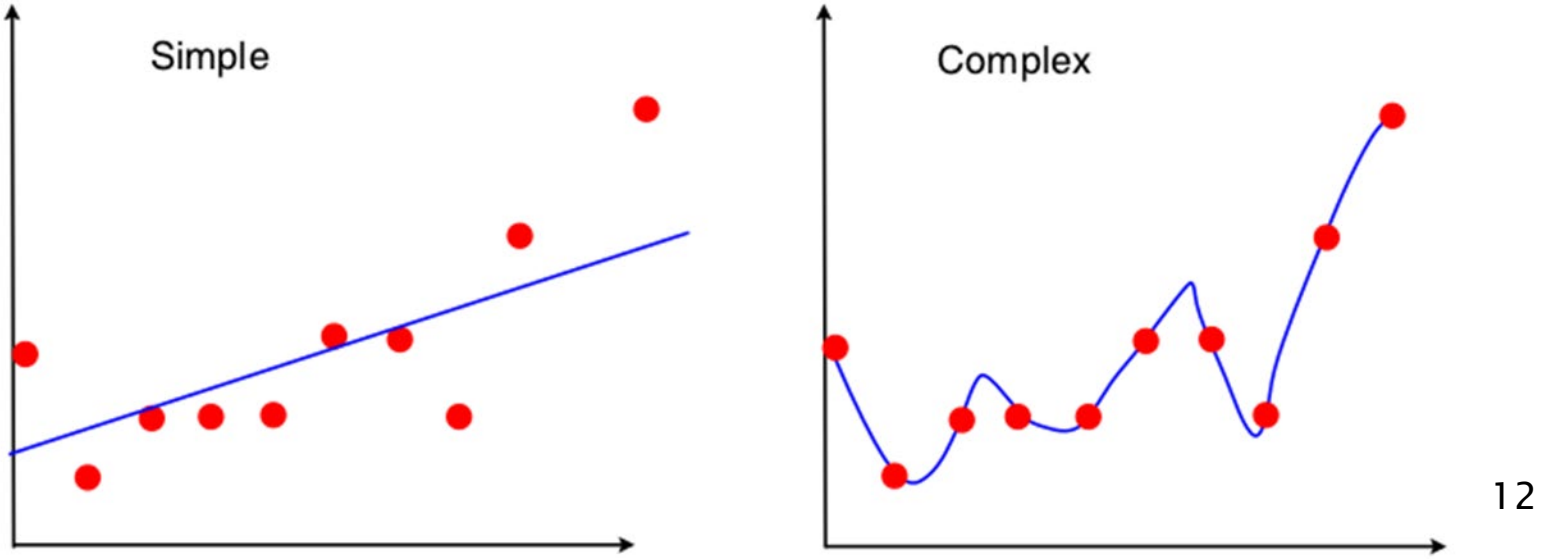
$$\hat{y} = a + bX$$

- Complex models (many parameters) usually have lower bias, but higher variance.

$$\hat{y} = a + b_1 x_1 + b_2 x_2$$

# Example

- A linear model can only fit a straight line.
- A high-degree polynomial can fit a complex curve.
- But the polynomial can fit the individual sample, rather than the population.
- Its shape can vary from sample to sample, so it has high variance.



# Issues in Performance Measurement (1/3)

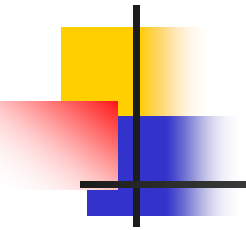
- If the training and test data are skewed towards one classification, the model will predict everything as being that class.
  - (e.g.) In Titanic training dataset, 68% of the people died. If the model is trained to predict everyone died, it would be 68% accurate.
- The data may be fitted against a feature that is not relevant.
  - In image classification, if all images of one class have similar background, the model may match based on the background, not the main object in the image.

# Issues in Performance Measurement (2/3)

- Different costs are associated with different errors.
  - (e.g.) mammography
    - Say 99% of the population has no apparent disease.
    - So if we say "no disease," we'll be correct 99% of the time!
    - But the cost is great (death) if we are wrong!
    - So let's say disease is always present.
      - We'll be wrong 99% of the time!
      - We'll need to follow up with biopsy.
  - (e.g.) in predicting consumer credit-worthiness
    - Are costs of loaning money to someone who then defaults the same as costs of not lending money to someone who would have repaid the loan?

# Issues in Performance Measurement (3/3)

- Cutoff threshold
  - For example, we have a k-nearest neighbor classifier with 1 output unit, and we code '1 = YES' and '0 = NO'
  - Should we set the threshold at 0.5, and turn anything > 0.5 into a 1, anything =< 0.5 into a 0?



# Roadmap: Evaluation

- Motivation
- Evaluation Methods
- Ensemble Learning
- Evaluation Metrics

# Roadmap: Evaluation Methods

- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

# Acknowledgments

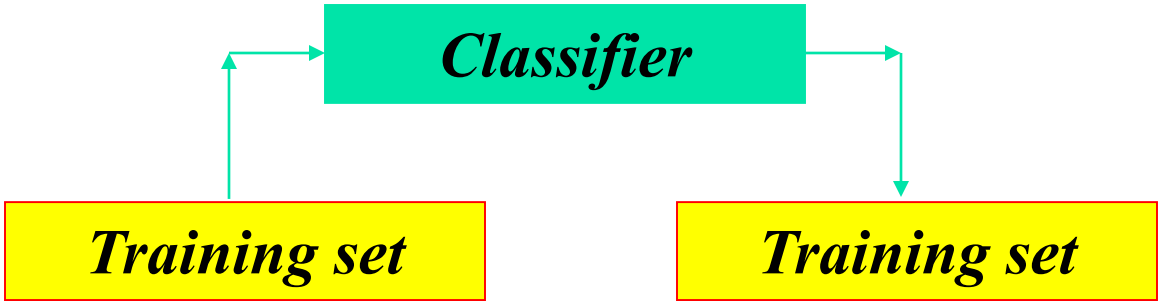

- https://project.dke.maastrichtuniversity.nl/datamining/2013-Slides/lecture-02.ppt
- https://www.cs.waikato.ac.nz/ml/weka/slides/Chapter5.pptx

# Roadmap: Evaluation Methods

- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

# Estimation Using Training Dataset (dumb)

- The accuracy/error estimates on the training data measure the degree of classifier's overfitting.
- Accuracy/error estimates on the training dataset are *not* good indicators of performance on future data.
  - Because future data will probably not be the same as the training data!

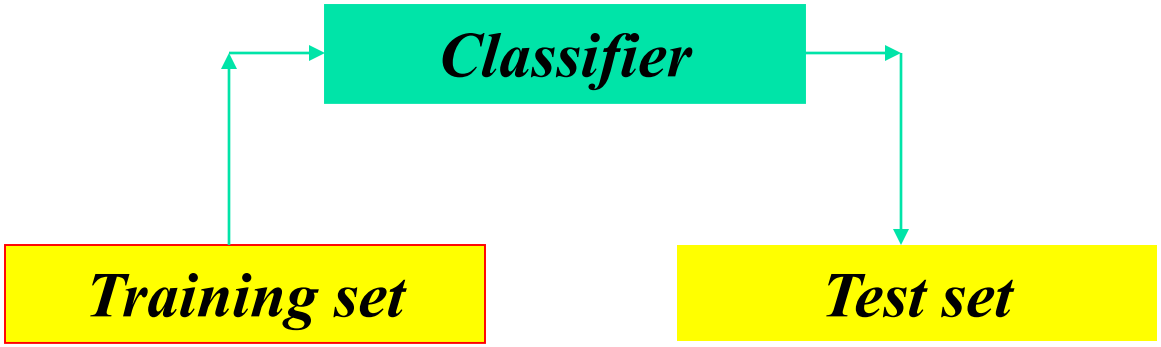


# Roadmap: Evaluation Methods

- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

# Estimation Using Independent Test Dataset

- Estimation with independent test data is used when we have plenty of data and there is a natural way to form training and test data.



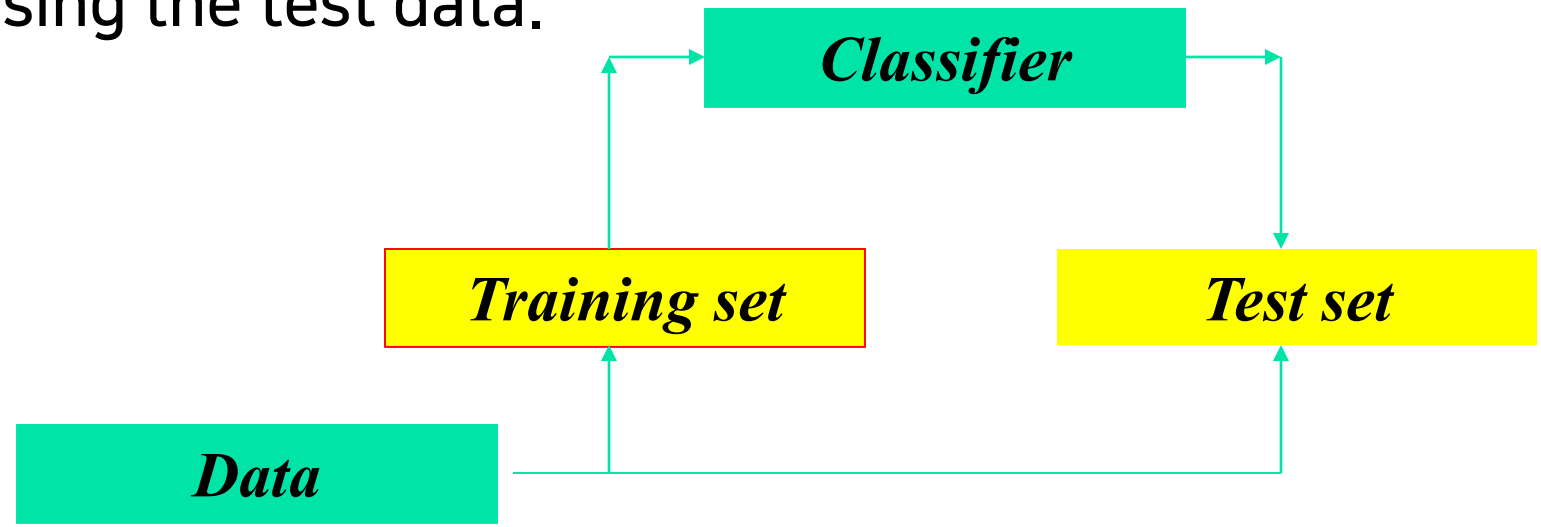

- For example: Classifiers are trained on data from 1985 and tested on data from 1986.

# Roadmap: Evaluation Methods

- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

# Holdout Method

- This method splits the data into training data and test data (e.g., 2/3 for training, 1/3 for test).
- Build a classifier using the training data and test it using the test data.




- The method is usually used when there are thousands of instances, including several hundred instances from each class.

# Making the Most of the Data

- Generally, the larger the training data the better the classifier (but returns diminish).
- The larger the test data, the more accurate the error estimate.
- Once evaluation is complete, *all the data* can be used to build the final classifier.

# Stratification

- The *holdout* method reserves a certain amount for testing and uses the remainder for training.
- For "unbalanced" datasets, samples might not be representative.
  - *few or no instances of some classes.*
- Stratified sampling ensures that each class is represented with approximately equal proportions in both the training and testing datasets.

# Repeated Holdout Method

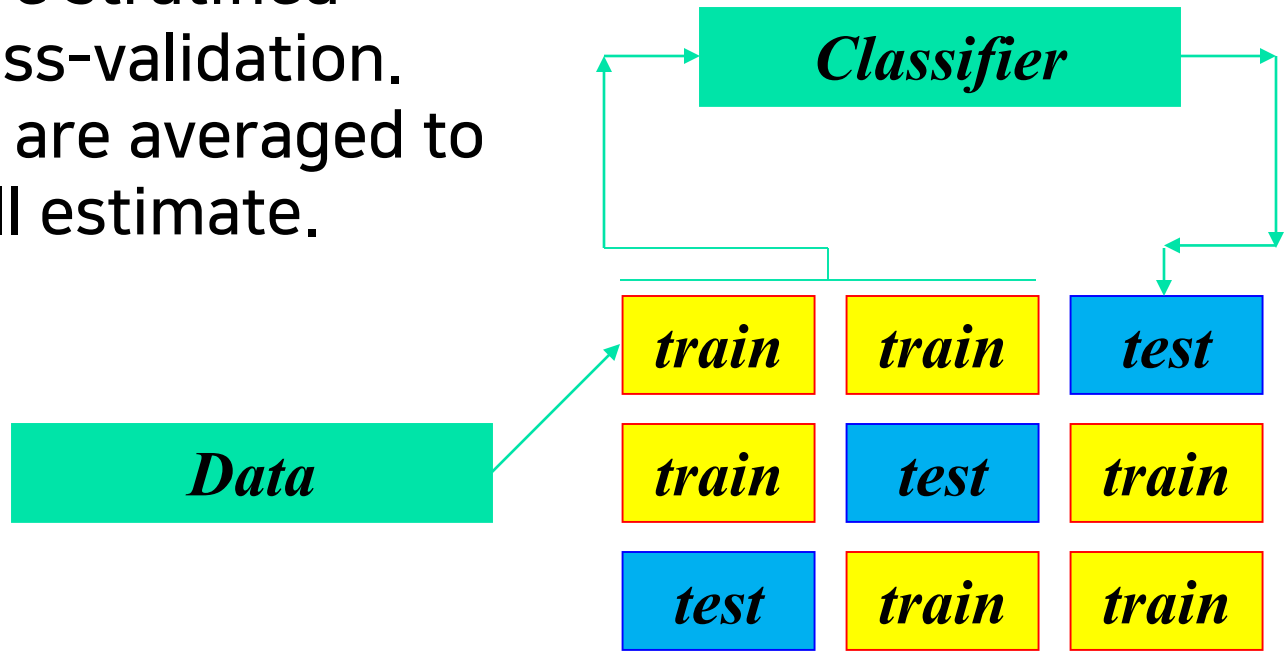
- Holdout estimate can be made more reliable by repeating the process with different subsamples.
  - In each iteration, a certain proportion is randomly selected for training (possibly with stratification).
  - The error rates on the different iterations are averaged to yield an overall error rate.
- Still not optimum
  - The different test datasets overlap.

# Roadmap: Evaluation Methods

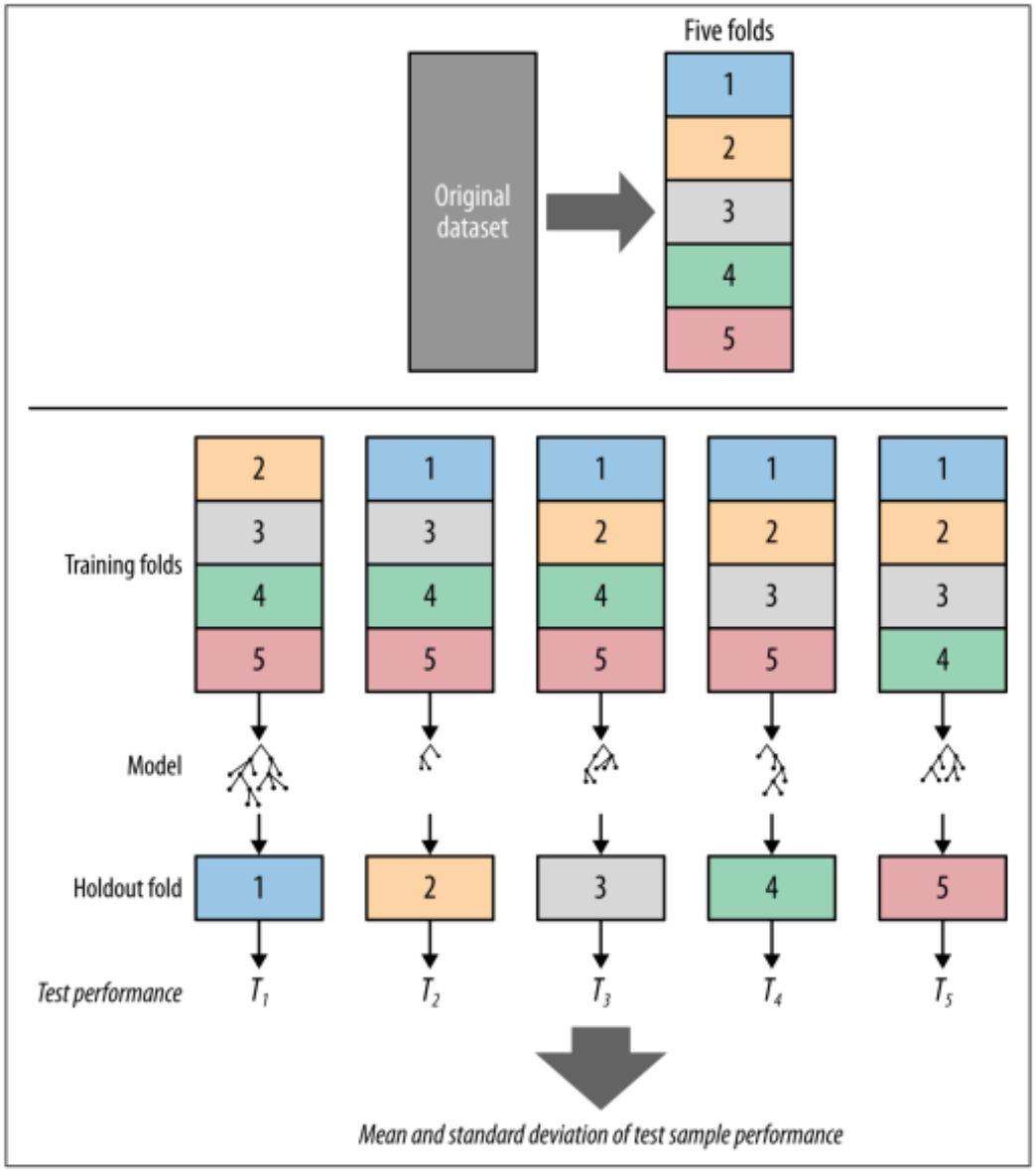
- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

# *k*-Fold Cross-Validation (1/2)

- Most widely used method
- *k-fold cross-validation* avoids overlapping test sets:
  - First, the dataset is split into *k* subsets of equal size.
  - Second, each subset in turn is used for testing and the remainder for training.
- The subsets are stratified before the cross-validation.
- The estimates are averaged to yield an overall estimate.



# Illustration: 5-fold cross-validation



# k-Fold Cross-Validation (2/2)

- Standard method for evaluation: stratified 10-fold cross-validation.
  - Why 10?
  - Extensive experiments have shown that this is the best choice to get an accurate estimate.
- Stratification reduces the estimate's variance.
- Repeated stratified cross-validation is even better.
  - e.g. 10-fold cross-validation is repeated 10 times and results are averaged (reduces the variance).

# 10-Fold Cross-Validation

- 1. Randomly divide your data into 10 pieces, 1 through k.
- 2. Treat the 1st tenth of the data as the test dataset. Fit the model to the other nine-tenths of the data (which are now the training data).
- 3. Apply the model to the test data.
- 4. Repeat this procedure for all 10 tenths of the data.
- 5. Calculate statistics of model accuracy and fit from the test data only.

# Cross Validation
# Using Pandas and Scikit-Learn

# Acknowledgments


- https://machinelearningmastery.com/k-fold-validation/

# Cross Validation: Example 1

- Dataset with 6 observations

  [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

- 3 Folds

  Fold1: [0.5, 0.2]
  Fold2: [0.1, 0.3]
  Fold3: [0.4, 0.6]

- 3 Classifiers

  Classifier1: trained on Fold1+Fold2, tested on Fold3
  Classifier2: trained on Fold2+Fold3, tested on Fold1
  Classifier3: trained on Fold1+Fold3, tested on Fold2

# Cross Validation API (1/2)

```
# prepare cross validation
from sklearn.model_selection import KFold
kfold = KFold(3, True, 1)
```

- 3: number of folds (dataset splits)
- True: shuffle the dataset before split
- 1: the seed for the pseudo random number generator used prior to the shuffle

# Cross Validation API (2/2)

```
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s,  test: %s'  % (train, test))
```

- The split() function splits the dataset.
- Called repeatedly, it returns each group (array) of train dataset and test dataset.
- The arrays contain the indexes into the original dataset.

# Full Code

```
# scikit-learn k-fold cross-validation
 from numpy import array
 from sklearn.model_selection import KFold

# data sample
 data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])

# prepare cross validation
 kfold = KFold(3, True, 1)

# enumerate splits
 for train, test in kfold.split(data):
        print('train: %s, test: %s' % (data[train], data[test]))
```

# The Result

```
# generates the first test set with 2 numbers,
# and generates a train set with the
  remaining 4 numbers
train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
# generates the second test set with 2 numbers,
# and generates a train set with the
  remaining 4 numbers
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
# generates the third test set with 2 numbers,
# and generates a train set with the
  remaining 4 numbers
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
```

# Cross Validation: Example 2 (example from an earlier class)

```
from sklearn.neighbors import   KNeighborsClassifier

# Build a KNN model
# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 3)

# Fit the classifier to the data
knn.fit(X_train,y_train)
```

# Test (Predict Using) the Model

#show first 5 model predictions on the test data
knn.predict(X_test)[0:5]

```
array([0, 0, 0, 0, 1])
```

# 'no diabetes' for the first 4 patients

# Check Model Accuracy

```
#check accuracy of our model on the test data
knn.score(X_test, y_test)
```

```
0.6688311688311688
```

# Try to Improve the Model Using 5-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score
import numpy as np
# create a KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)

# train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)

# print each cv score (accuracy) and average them
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

# Variations of K-Fold

- StratifiedKFold
  - Returns stratified folds. The folds preserve the percentage of samples for each class.
  - (That is, this avoids building folds with imbalanced class distributions.)
- GroupKFold
  - Ensures that the same group is not represented in both testing/validation and training sets.
  - (That is, the same group will not appear in two different folds.)
  - (The number of distinct groups is at least equal to the number of folds).
- RepeatedKFold
  - Repeats K-Fold n times.

# Roadmap: Evaluation Methods

- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

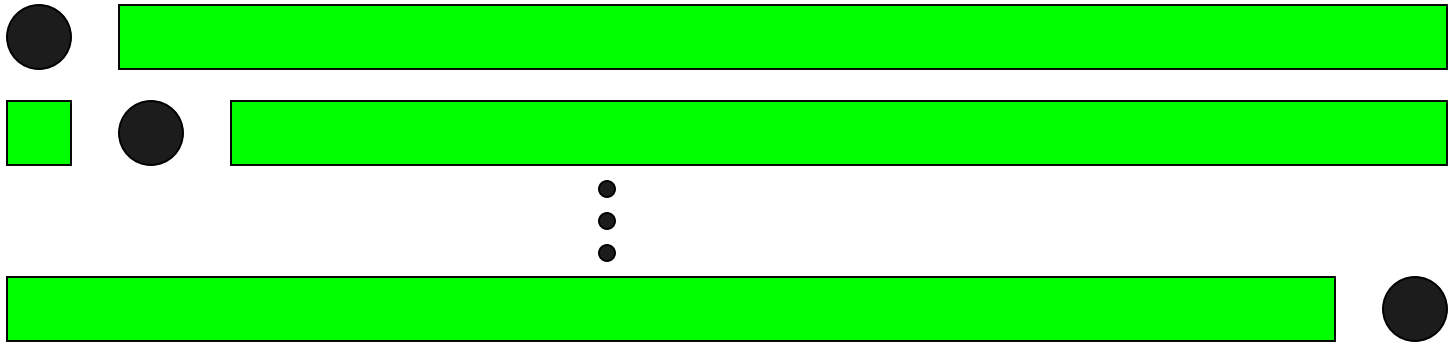
# Leave-One-Out Cross-Validation

- Leave-One-Out is a special case of cross-validation.
  - Set number of folds to number of training instances
  - (i.e.,for *n* training instances, build classifier *n* times.)
- Makes the best use of a small dataset
- Involves no random subsampling
- But, computationally very expensive

# K-Fold CV vs. LOO-CV

- k-fold cross-validation


- Leave-one-out (n-fold cross-validation)



# Problems with Leave-One-Out CV

- Stratification is not possible
  - It *guarantees* a non-stratified sample because there is only one instance in the test set!
- extreme example: Random dataset splits equally into two classes.
  - Best inducer predicts majority class.
  - 50% accuracy on fresh data
  - Leave-One-Out-CV estimate is 100% error!

# Roadmap: Evaluation Methods

- Training dataset
- Independent test dataset
- Hold-out method
- *k*-fold cross-validation method
- Leave-one-out cross-validation method
- Bootstrap method
- many more…

# Bootstrap Method

- Cross validation uses sampling *without replacement.*
  - The same instance, once selected, cannot be selected again for a particular training/test set.
- The *bootstrap* uses sampling *with replacement* to form the training set.
  - Sample a dataset of *n* instances *n* times *with replacement* to form a new dataset of *n* instances.
  - Use this data as the training set
  - Use the instances from the original dataset that are not in the new training set for testing.

# Bootstrap Method

- The bootstrap method is also called the *0.632 bootstrap:*
  - A particular instance has a probability of 1–1/*n* of *not* being picked.
  - Thus its probability of ending up in the test set (i.e., not in the training set) is

$$\left(1-\frac{1}{n}\right)^{n} \approx e^{-1} = 0.368$$

  - This means the training set will contain approximately 63.2% of the instances and the test set will contain approximately 36.8% of the original instances.

# Estimating Error with the Bootstrap Method

- The error estimate on the test set will be very pessimistic, because the classifier is trained on just ~63% of the instances.
- Therefore, combine it with the training error:

$$err = 0.632 \cdot e_{\text{test instances}} + 0.368 \cdot e_{\text{training instances}}$$

- The training error gets less weight than the error on the test data.
- Repeat process several times with different replacement samples; average the results.

# Evaluation Methods Summary

- Use independent test sets and the holdout method for “large” data;
- Use the cross-validation method for “middle-sized” data;
- Use the leave-one-out and bootstrap methods for small data;
- Do not use test data for parameter tuning - use separate validation data.

# Hyperparameter Optimization

# Acknowledgments


- https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d
- https://towardsdatascience.com/hyper-parameter-tuning-with-randomized-grid-search-54f865d27926

# Hyperparameter Optimization

- First, establish a baseline model
  - Build a model
  - Train it
  - Test (predict with) it
  - Evaluate the model's performance
- Then, if necessary, tune the model by hyperparameter optimization
  - (e.g.) determine the best 'k' for a k-means clustering model or k-nearest neighbor classifier
  - Often there are many sets of parameter combinations

# Three Methods (1/2)

- Manual search (obvious)
- Grid search
  - Create a grid (matrix or Python dictionary) of parameter-value combinations
  - Try all possible sets of parameter combinations in the grid, along with k-fold cross validation
  - ** Becomes very expensive with ensemble learning models (later)
  - Scikit-learn GridSearchCV()

# Three Methods (2/2)

- Randomized search (usually the best)
  - Try random subsets of parameter combinations, along with k-fold cross validation
  - Scikit-learn RandomizedSearchCV()
  - Rule of thumb: "In 95% of the time, with 60 iterations (combinations), best 5% sets of parameters can be found, regardless of the grid size."

# Testing Using Different Hyperparameters (k)

- For our kNN model, we will specify a range of values for ‘n_neighbors’ in order to see which value works best for our model.
- To do this, we will create a dictionary, setting ‘n_neighbors’ as the key and using Numpy to create an array of values from 1 to 24.

# Hypertuning Using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
# create new a knn model
knn2 = KNeighborsClassifier()

# create a dictionary of all values we want to test for
n_neighbors
param_grid = {'n_neighbors': np.arange(1, 25)}

# use GridSearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn2, param_grid, cv=5)

# fit model to data
knn_gscv.fit(X, y)
```

# Checking the Result of Hypertuning

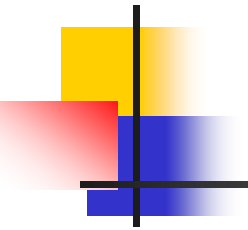
# check top performing n_neighbors value
knn_gscv.best_params

```
{'n_neighbors': 14}
```

# 14 is the optimal value of k

# check the mean score for the top performing value of n_neighbors
knn_gscv.best_score_

```
0.7578125
```



# End of SubModule