8. Model Selection

CPDSAI

Chantri Polprasert

Outline

- Model Selection
  - Validation-test
  - Leave-One-Out Cross-Validation
  - k-fold Cross-Validation

Recall

For regression, we often aim to reduce the mean square error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

For classification, we often consider the error rate:

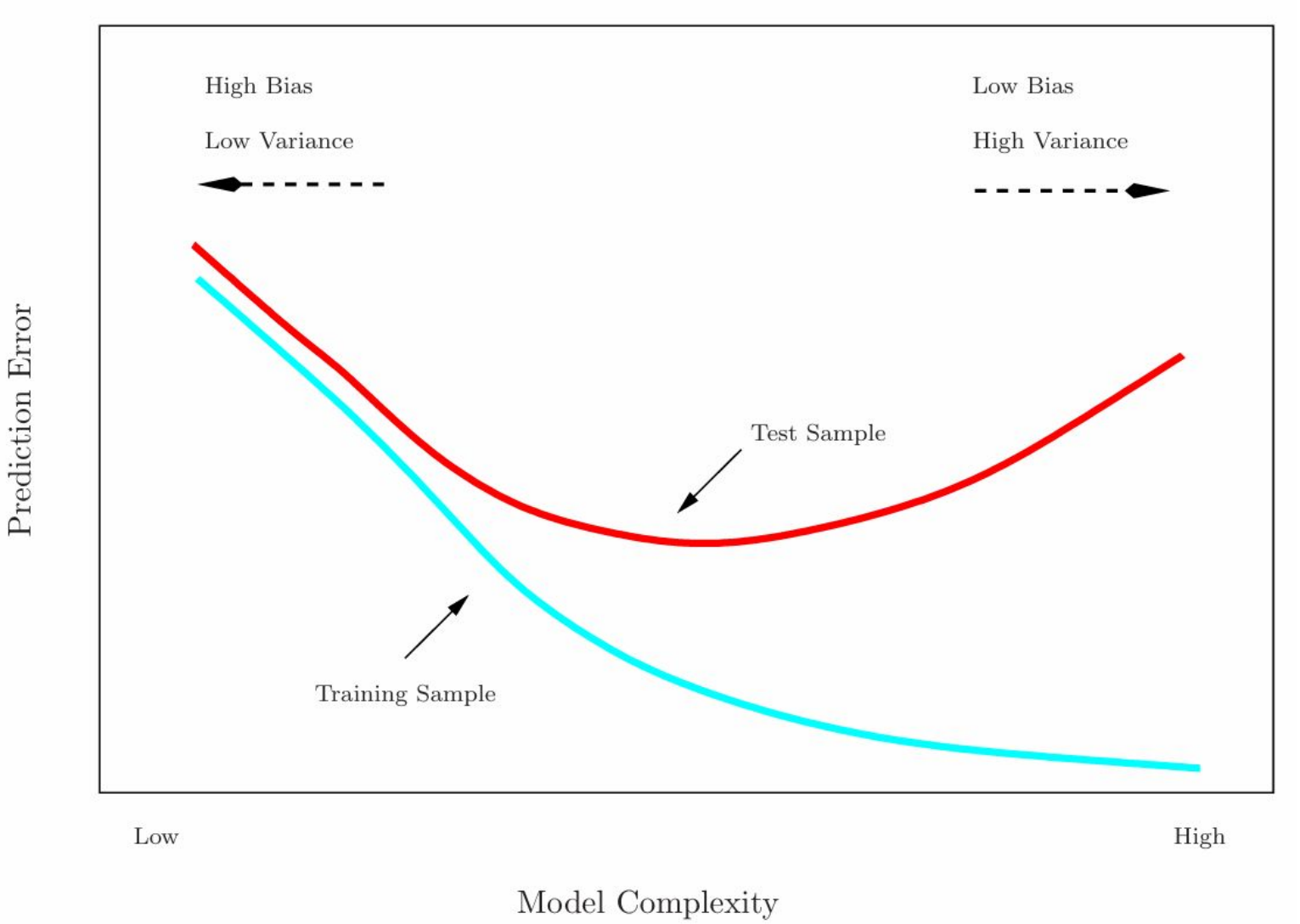
$$\frac{1}{N}\sum_{i=1}^{N} I(y_i \neq \hat{y}_i)$$

y_i and $\hat{y}_i$ are the ith observation and its estimate. *N* is the total number of observations.

Training Error versus Test error

- For supervised learning in regression problem, we fit the model to the training set and estimate the MSE using the testing set.
- **The training error**: the average error that results from fitting a model to the training dataset.
- **The test error**: the average error that results from using a trained model through some statistical learning methods to predict the response on a new observation, one that was not used in training the method.
- The training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter. Why?

Training Error versus Test error

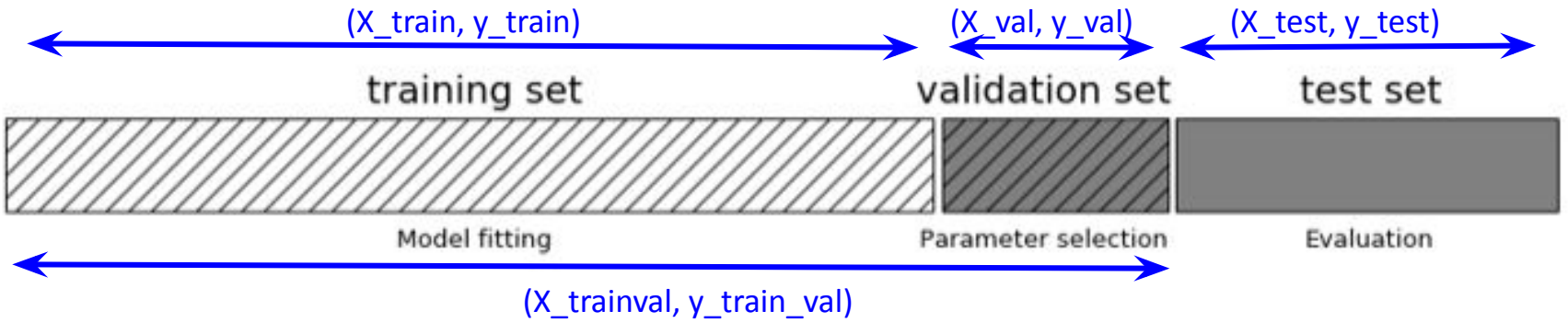


- Models that are too complex for the amount of training data available are said to <u>overfit</u> (high variance, low bias) and are not likely to generalize well to new examples.
- Can very large datasets solve this problem?

Generalization, Overfitting, and Underfitting

- Generalization ability refers to an algorithm's ability to give accurate predictions for new, previously unseen data.
- Assumptions:
  - Future unseen data (test set) will have the same properties (distribution) as the current training sets.
  - Thus, models that are accurate on the training set are expected to be accurate on the test set.
  - But that may not happen if the trained model is tuned too specifically to the training set.

Threefold split

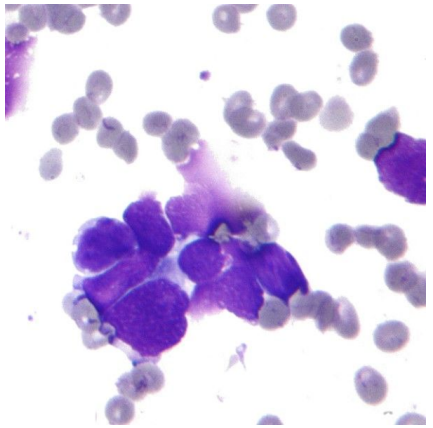


- To avoid overfitting using the test set multiple times to select hyperparameters
  - hyperparameters = a parameter that can be set in order to define any configurable part of a model's learning process

Validation-test approach

- Approach to get the validation test: Holding out a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.
- Pro: fast, simple
- Con:
  - Highly variable (the results depend on the particular sampling)
  - Validation error tends to overestimate the true error

Threefold Split

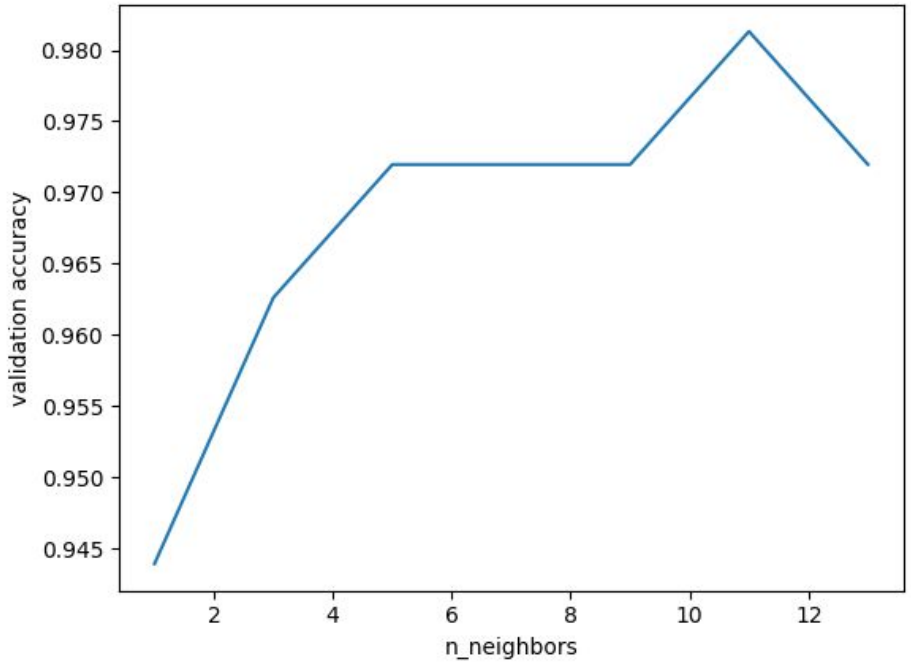


Breast Cancer Wisconsin (Diagnostic) Data Set

- Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
- Sample feature Information: radius, texture, perimeter etc…

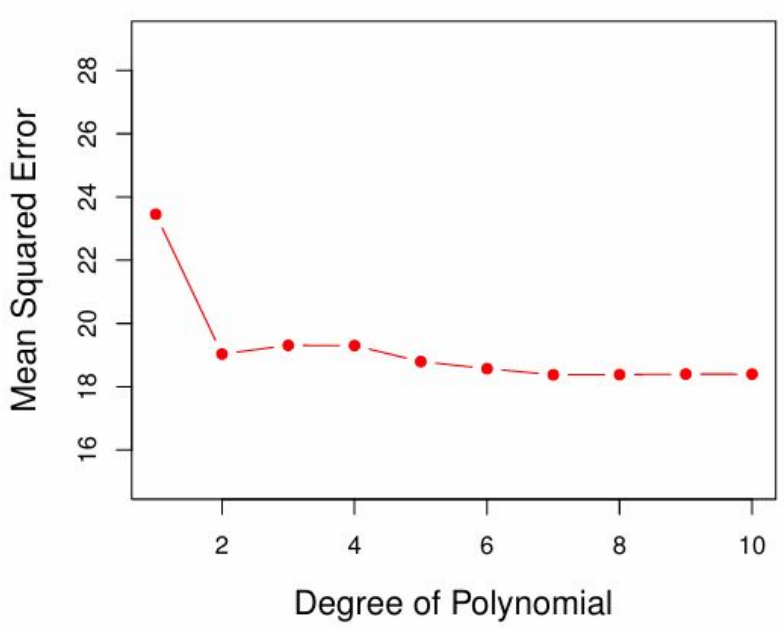
```
val_scores = []
neighbors = np.arange(1, 15, 2)
for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    val_scores.append(knn.score(X_val, y_val))
print("best validation score: {:.3f}".format(np.max(val_scores)))
best_n_neighbors = neighbors[np.argmax(val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_trainval, y_trainval)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

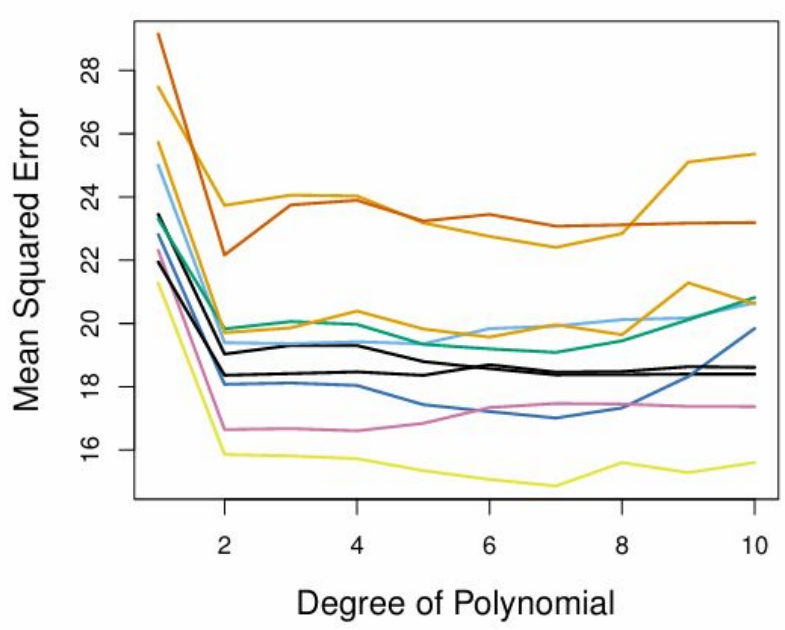
```
best validation score: 0.981
best n_neighbors: 11
test-set score: 0.965
```



Example: automobile data

- Want to compare linear vs higher-order polynomial terms in a linear regression to predict the mpg
- We randomly split the 392 observations into two sets, a training set containing 196 of the data points, and a validation set containing the remaining 196 observations.





Left figure shows single split. Right figure shows multiple splits.

Auto Data

Gas mileage, horsepower, and other information for 392 vehicles.

A data frame with 392 observations on the following 9 variables.

- `mpg`: miles per gallon
- `cylinders`: Number of cylinders between 4 and 8
- `displacement`: Engine displacement (cu. inches)
- `horsepower`: Engine horsepower
- `weight`: Vehicle weight (lbs.)
- `acceleration`: Time to accelerate from 0 to 60 mph (sec.)
- `year`: Model year (modulo 100)
- `origin`: Origin of car (1. American, 2. European, 3. Japanese)
- `name`: Vehicle name

https://islp.readthedocs.io/en/latest/datasets/Auto.html

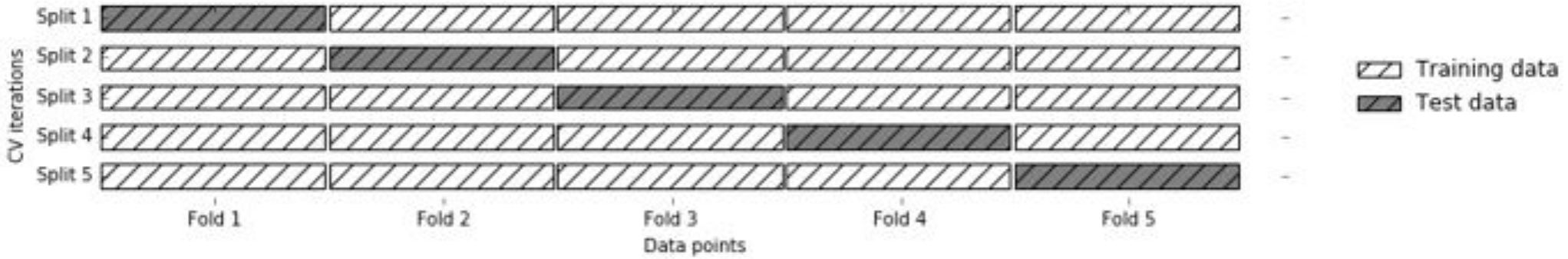
Drawbacks of the validation set approach

- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- Only a subset of the observations – those that are included in the training set rather than in the validation set – are used to train the model.
- This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set. Why?

Resampling method

- To combat this, it would be nice if we could have multiple testing and training sets.
- While this is not a luxury we usually have, we can fudge it using resampling methods. This technique involves:
  - repeatedly drawing samples from a training set
  - refitting a model of interest on each sample
- This will provide additional information about the fitted model that we would not otherwise get.
- Resampling approaches can be computationally expensive (less so now due to recent advances in computing power)
- Resampling techniques: cross-validation (CV) and bootstrapping
  - To assess model performance (eg. estimate test MSE),
  - Perform model selection (eg. choose k for knn)

Cross-validation



- Split the data into multiple folds, usually 5 or 10, and built multiple models.
- Start by using fold-1 as the test data, and the remaining ones as the training data.
- build your model on the training data, and evaluate it on the test fold.
- For each of the splits of the data, you get a model evaluation and a score.
- In the end, aggregate the scores, for example by taking the mean/median.
- Pro: more stable (multiple models), more data (e.g. 10-fold we have 90% for training)
- Con: ???

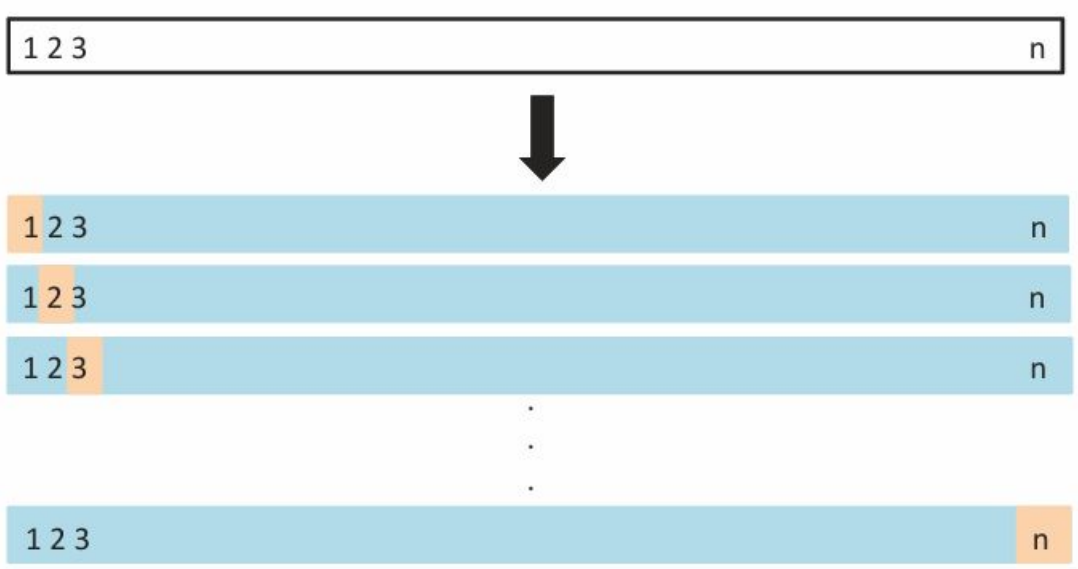
Details (CV for Regression)

- Let the K parts be $C_1, C_2, .. C_K$ where C_k denotes the indices of the observations in part k. There are n_k observations in part k: if n is a multiple of K then, $n_k = n/K$.
- Compute

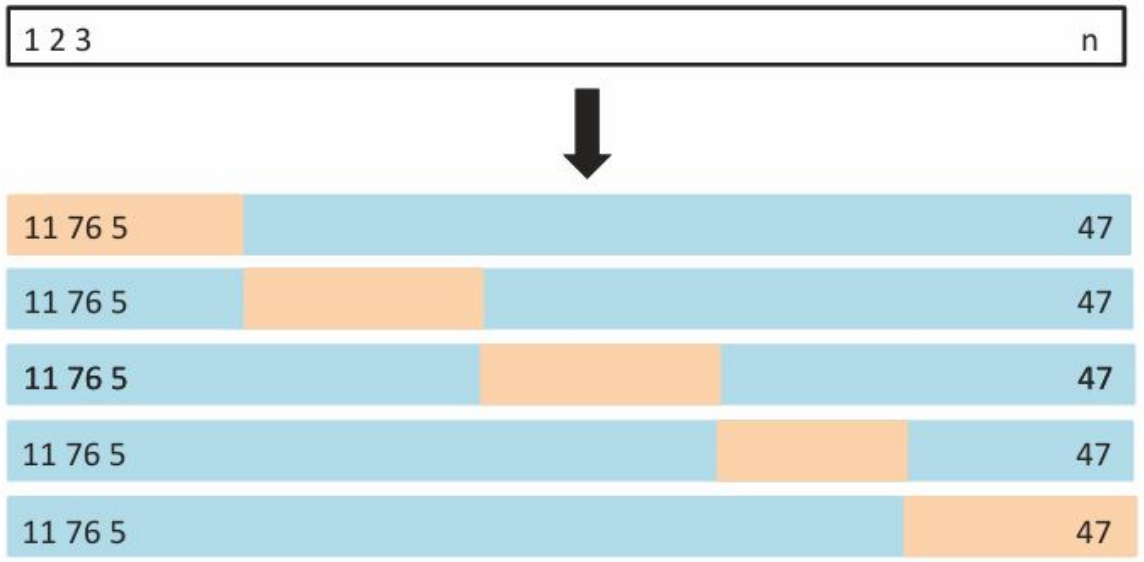
$$\mathrm{CV}_{(K)} = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{MSE}_k$$

where $\mathrm{MSE}_k = \sum_{i \in C_k} (y_i - \hat{y}_i)^2 / n_k$ and $\hat{y}_i$ is the fit for observation i, obtained from the data with part k excluded.

- Setting $K=n$ yields n-fold or LOOCV

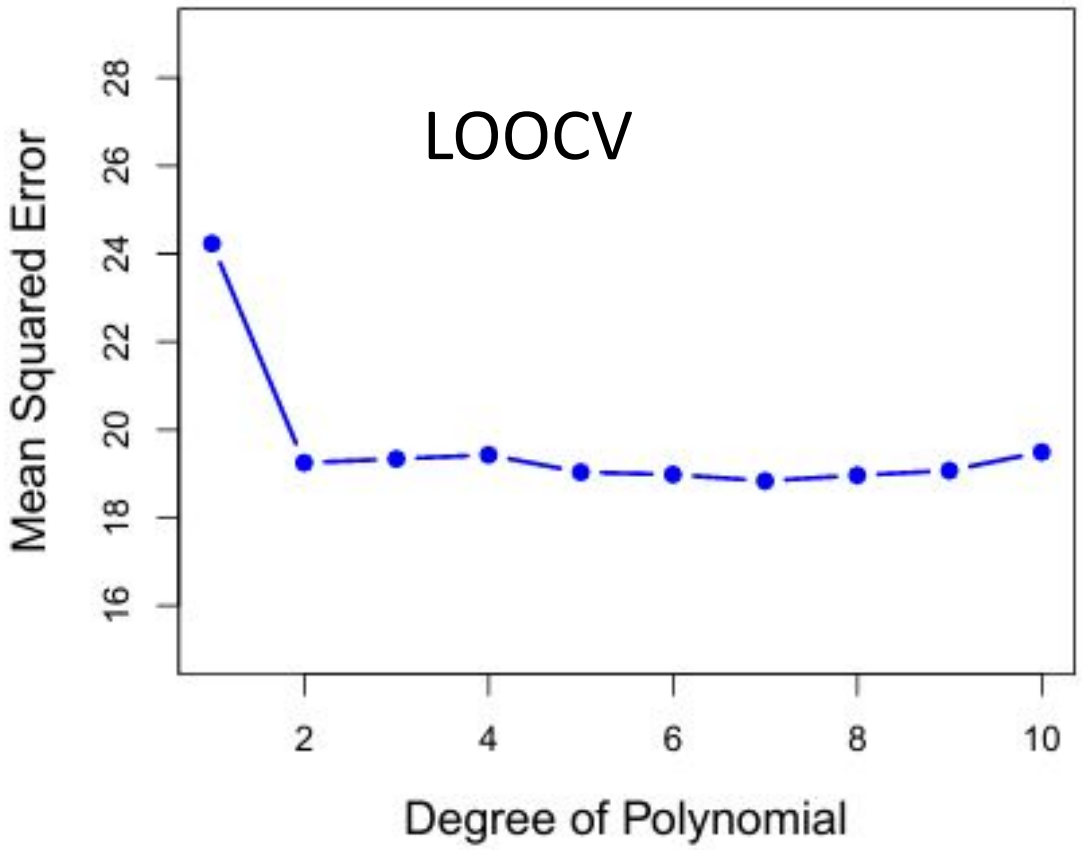


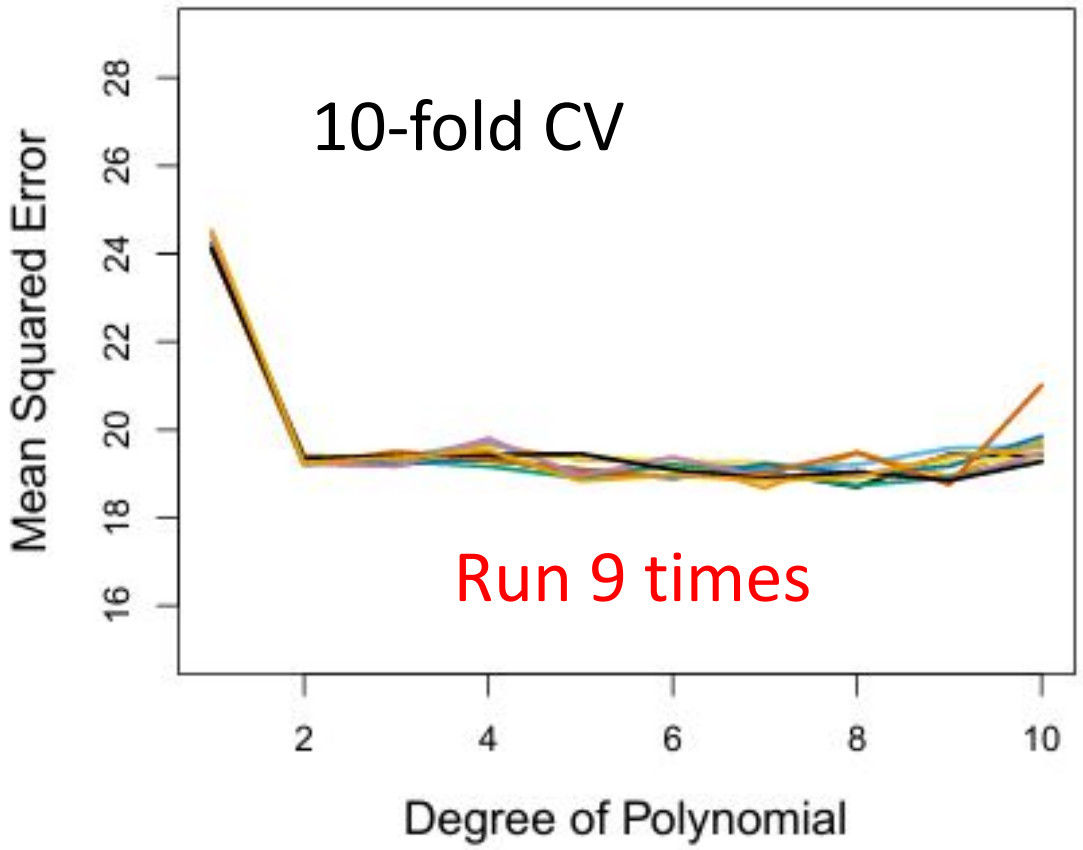
LOOCV



5-fold CV

LOOCV vs k-fold CV (k<N)

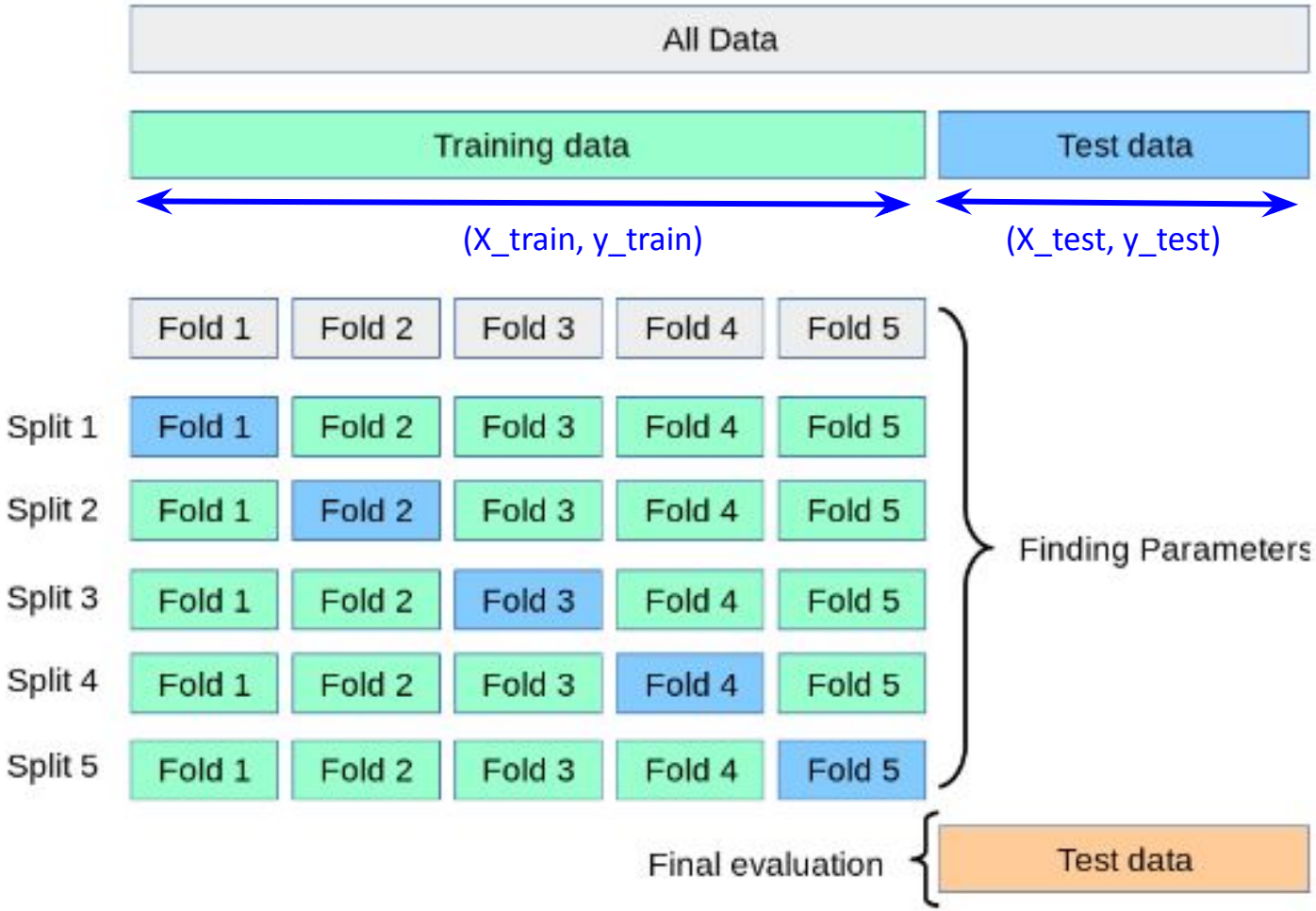




LOOCV vs k-fold CV (k<N)

	Complexity	Bias	Variance
LOOCV			
k-fold CV (k<N)			

Cross-validation + test set



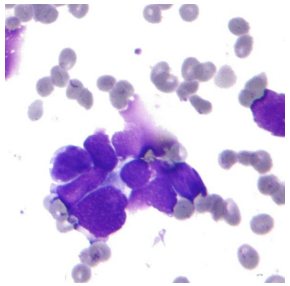
- Split whole data to training and testing
- In testing use k-fold to find parameters
- Given best parameters, retrain with the whole training data, and use this final model for final evaluation
- <u>Gold standard</u>

Grid-Search with Cross-Validation

```
sklearn.model_selection.cross_val_score(estimator, X, y=None, *, groups=None,
scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, params=None,
pre_dispatch='2*n_jobs', error_score=nan)
```

```python
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y)

cross_val_scores = []

for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    cross_val_scores.append(np.mean(scores))

print("best cross-validation score: {:.3f}".format(np.max(cross_val_scores)))
best_n_neighbors = neighbors[np.argmax(cross_val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_train, y_train)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

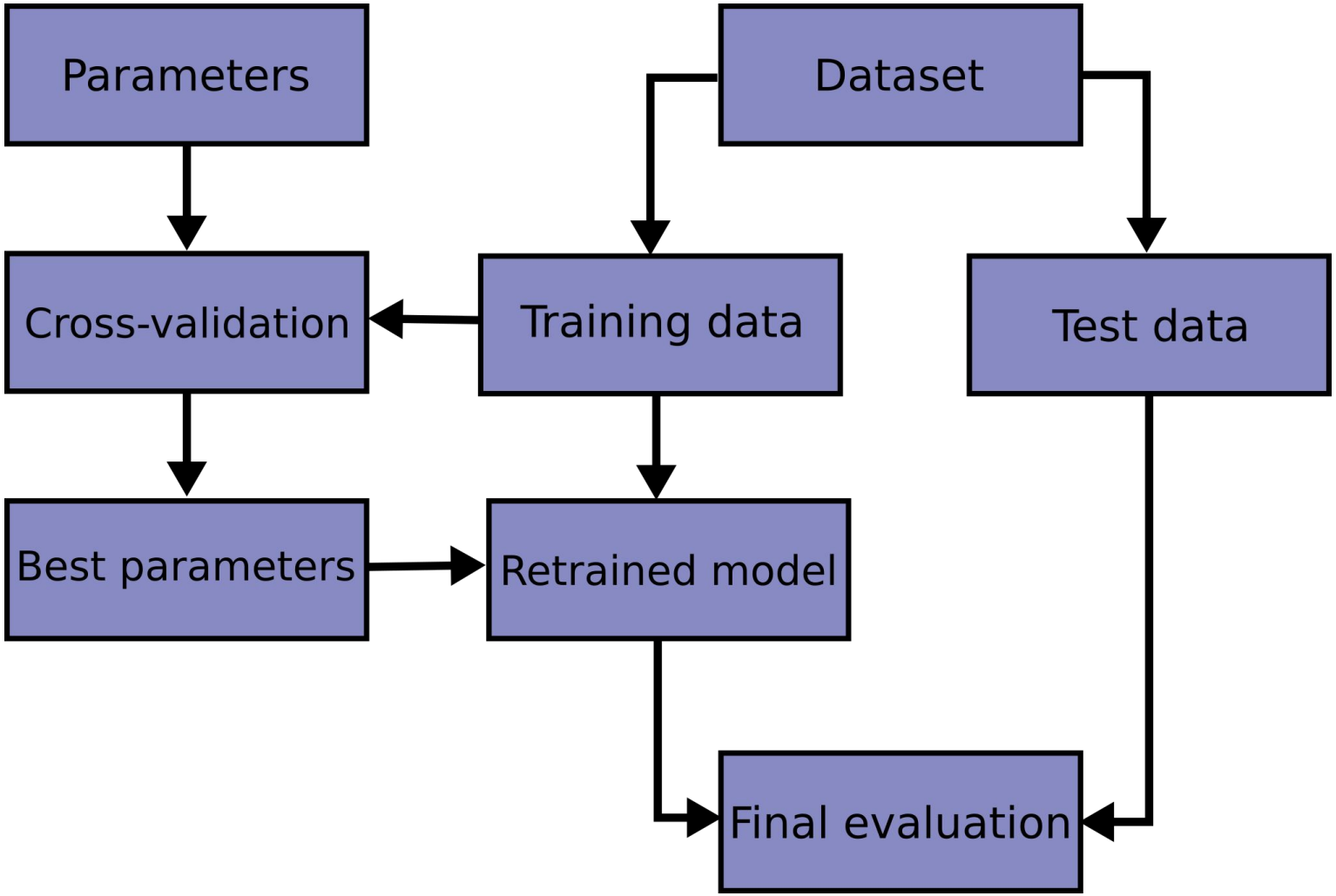
```
best cross-validation score: 0.969
best n_neighbors: 9
test-set score: 0.944
```



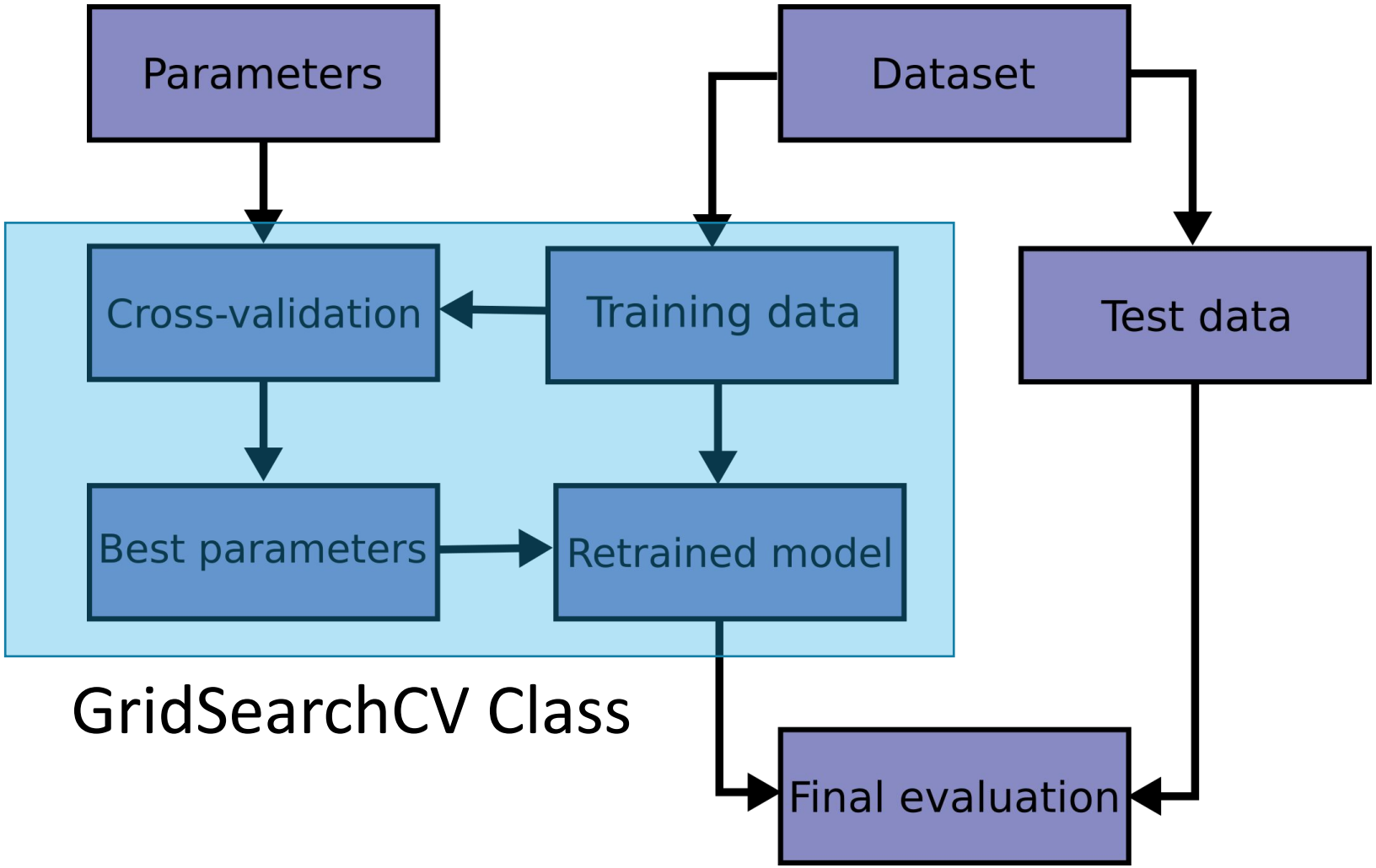
cv

- For int/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used.
- In all other cases, KFold is used.
- CV iterators

Basic Supervised Learning Workflow



Scikit-learn has a tool



GridSearchCV

- Called Meta-estimator
- Can specify the estimator, parameters
- Retrain the model with the best parameter settings.

Grid search is the process of performing hyperparameter tuning in order to determine the optimal values for a given model.

```
from sklearn.model_selection import GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)


param_grid = {'n_neighbors':  np.arange(1, 30, 2)}
grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10, return_train_score=True)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

print("test-set score: {:.3f}".format(grid.score(X_test, y_test)))
```

```
best mean cross-validation score: 0.972
best parameters: {'n_neighbors': 3}
test-set score: 0.958
```

GridSearchCV

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, *,
scoring=None, n_jobs=None, refit=True, cv=None, verbose=0,
pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False) #
```

- Exhaustive search over specified parameter values for an estimator.
- GridSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.
- The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid

GridSearchCV

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, *,
scoring=None, n_jobs=None, refit=True, cv=None, verbose=0,
pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False) #
```

- estimator: the scikit-learn estimator interface (e.g. kNNClassifier)
- param_grid: Dictionary with parameters names (str) as keys and lists of parameter settings to try as values
- scoring: Default: accuracy for classification, R2 for regression
- n_jobs: no. of jobs to run in parallel
- refit: an estimator using the best found parameters on the whole dataset
- cv: Number of folds (default:5)

GridSearchCV output

Three different groups for the GridSearchCV method:

- A result log
  - cv_results_
- The best results
  - best_index_, best_params_, best_score_, best_estimator_
- Extra information
  - scorer_, n_splits_, refit_time_

The .cv_results_

```
results = pd.DataFrame(grid.cv_results_)
print(results.shape)
```

```
(15, 31)
```

- The cv_results_ property: 15 rows with 31 columns
- Contain the dictionary of all parameters
- Others: time, train_score, test_score etc..

	split8_train_score	split9_train_score	mean_train_score	std_train_score
0	1.000000	1.000000	1.000000	0.000000
1	0.986979	0.986979	0.985392	0.002917
2	0.981771	0.981771	0.979915	0.002632
3	0.981771	0.979167	0.977046	0.002287
4	0.981771	0.979167	0.977829	0.002923
5	0.973958	0.973958	0.973135	0.003702
6	0.973958	0.976562	0.971568	0.004292
7	0.973958	0.976562	0.967916	0.004391
8	0.963542	0.973958	0.965309	0.003710

0	{'n_neighbors': 1}
1	{'n_neighbors': 3}
2	{'n_neighbors': 5}
3	{'n_neighbors': 7}
4	{'n_neighbors': 9}
5	{'n_neighbors': 11}
6	{'n_neighbors': 13}
7	{'n_neighbors': 15}
8	{'n_neighbors': 17}
9	{'n_neighbors': 19}
10	{'n_neighbors': 21}
11	{'n_neighbors': 23}
12	{'n_neighbors': 25}
13	{'n_neighbors': 27}
14	{'n_neighbors': 29}

The best result

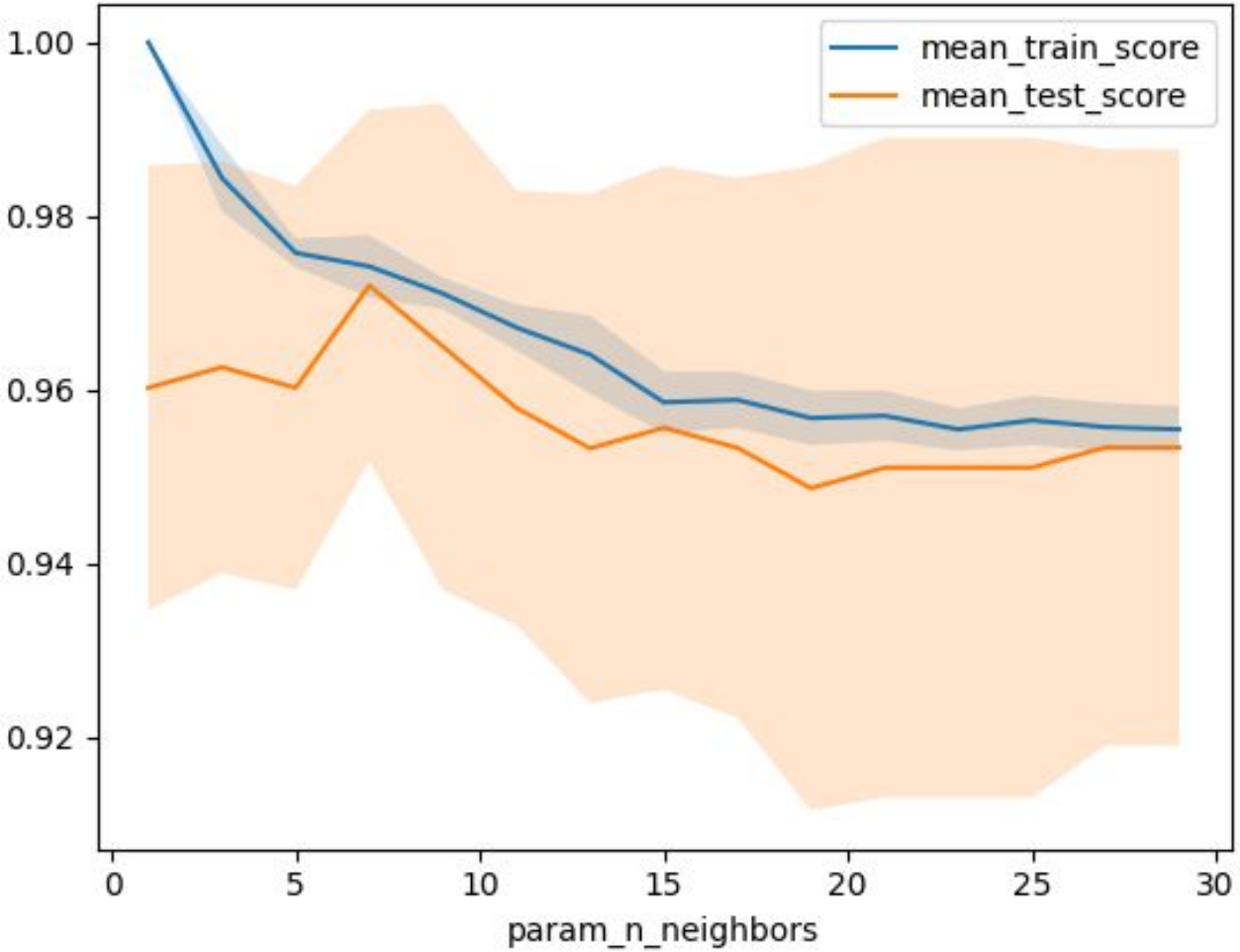
Information on the best grid is summarized in the following three properties:

- best_score_: the actual best score
- best_params_: the dictionary of parameters that give the best score

```
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

print("test-set score: {:.3f}".format(grid.score(X_test, y_test)))
```

```
best mean cross-validation score: 0.972
best parameters: {'n_neighbors': 7}
test-set score: 0.965
```

Example: n_neighbors Search Results



Extra information

Additional information related to training, testing

- refit_time_: the number of seconds for refitting the best model on the whole dataset
- n_splits_: How many cross-validation splits (cv)
- scorer_: the scorer function used on the held out data (

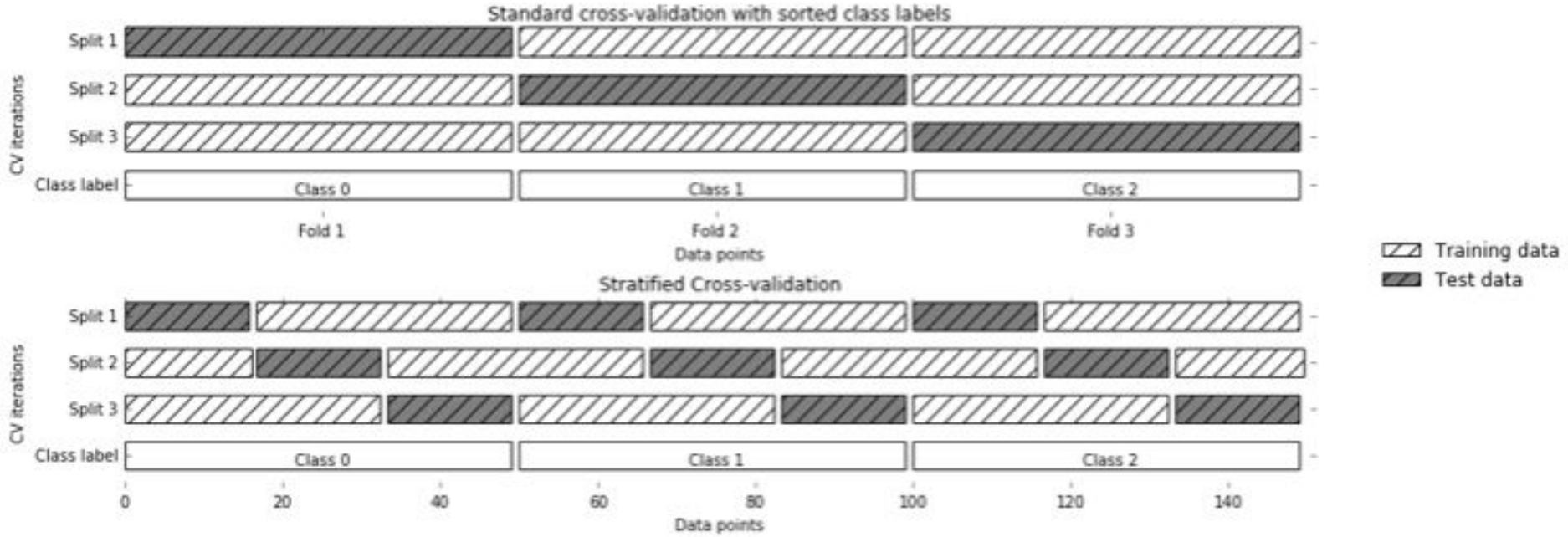
```
print(grid.refit_time_)
```

```
0.0003075599670410156
```

```
print(grid.n_splits_)
```

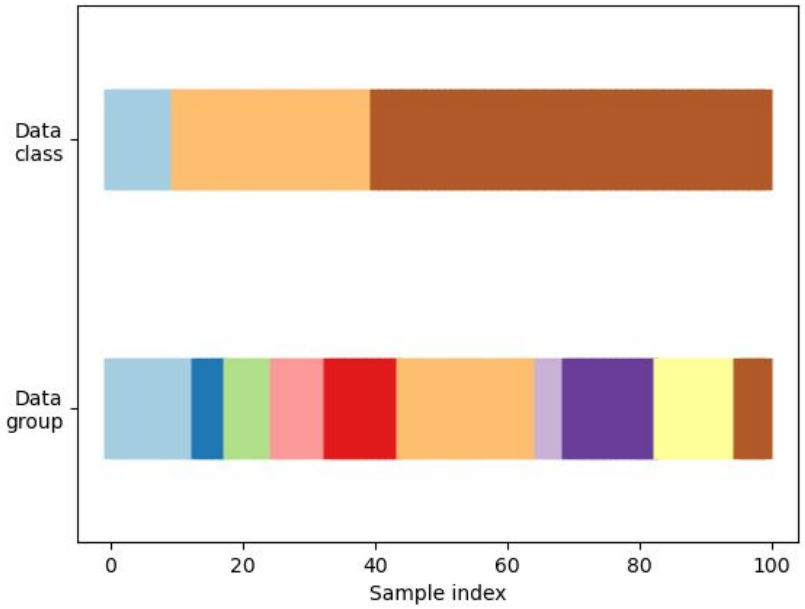
```
10
```

Cross-Validation Strategies: StratifiedKFold



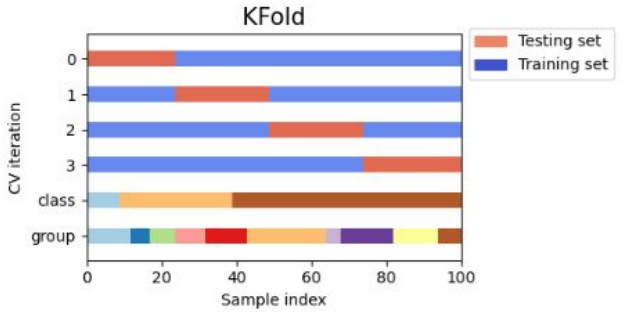
- Dataset sorted by class, if you don't stratify, accuracy will be zero.
- Stratified: Ensure relative class frequencies in each fold reflect relative class frequencies on the whole dataset.
- This is also helpful if your data is very imbalanced. If some of the classes are very rare, it could otherwise happen that a class is not present at all in a particular fold.

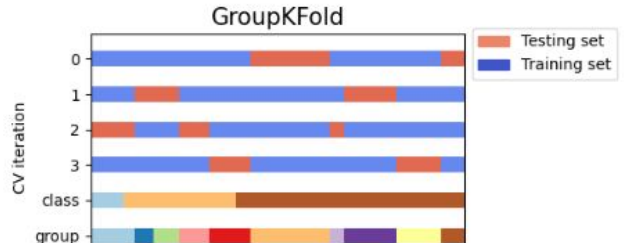
Cross-validation iterators in sklearn



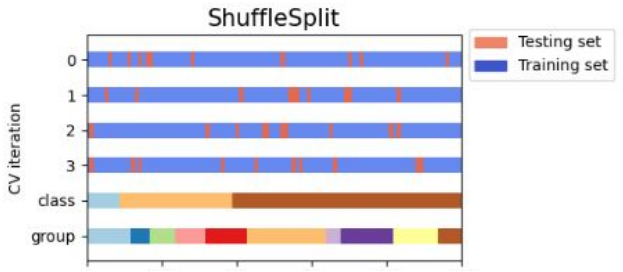
100 randomly generated input data points, 3 classes split unevenly across data points, and 10 groups split evenly across data points

Which one is suitable for i.i.d. data?

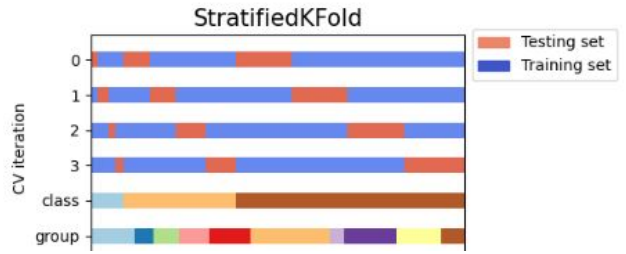




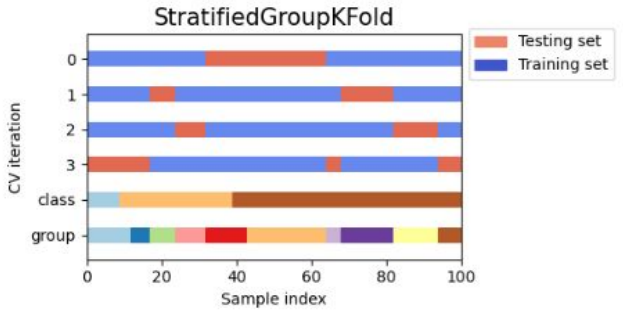
same group is not represented in both testing and training sets.

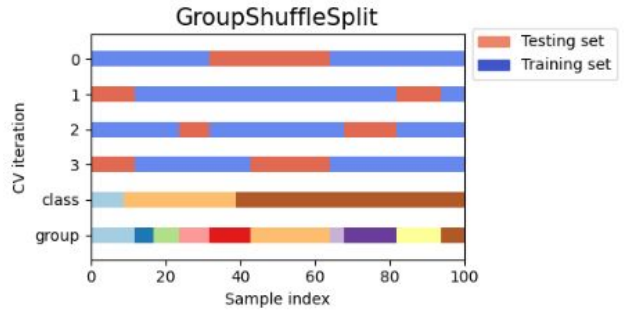


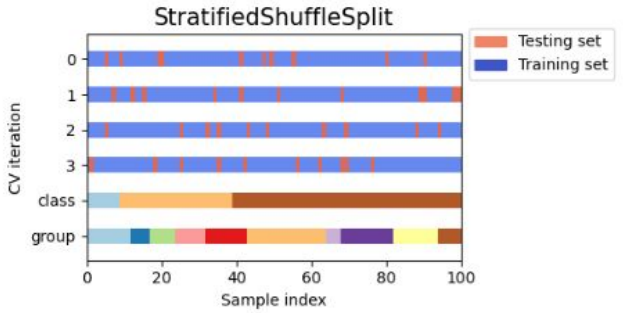
Random permutation cross-validator.

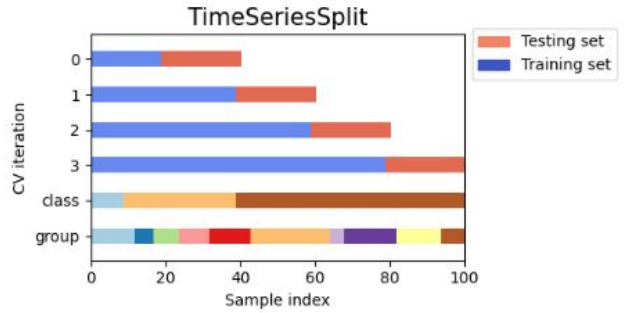


preserve the percentage of samples for each class



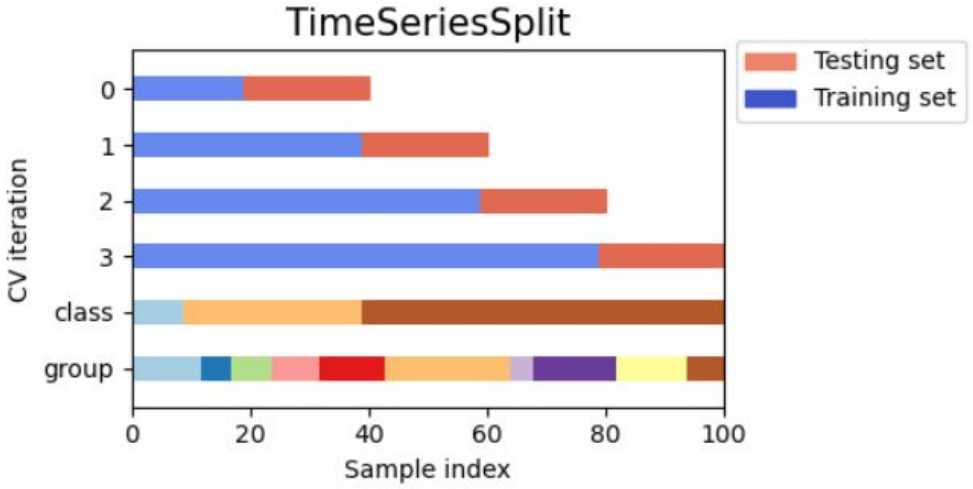






Time Series Split

- A variation of k-fold which returns first *k* folds as train set and the *(k+1)th* fold as test set.
- Unlike standard cross-validation methods, successive training sets are supersets of those that come before them.
- Used to cross-validate time series data samples that are observed at fixed time intervals.



Using Cross-Validation Generators

```
from sklearn.model_selection import KFold, StratifiedKFold, ShuffleSplit, RepeatedStratifiedKFold
kfold = KFold(n_splits=5)
skfold = StratifiedKFold(n_splits=5, shuffle=True)
ss = ShuffleSplit(n_splits=20, train_size=.4, test_size=.3)
rs = RepeatedStratifiedKFold(n_splits=5, n_repeats=10)

print("KFold:\n{}".format(
      cross_val_score(KNeighborsClassifier(), X, y, cv=kfold)))

print("StratifiedKFold:\n{}".format(
      cross_val_score(KNeighborsClassifier(), X, y, cv=skfold)))

print("ShuffleSplit:\n{}".format(
      cross_val_score(KNeighborsClassifier(), X, y, cv=ss)))

print("RepeatedStratifiedKFold:\n{}".format(
      cross_val_score(KNeighborsClassifier(), X, y, cv=rs)))
```

```
KFold:
[0.93 0.96 0.96 0.98 0.96]
StratifiedKFold:
[0.98 0.96 0.96 0.97 0.96]
ShuffleSplit:
[0.98 0.96 0.96 0.98 0.94 0.96 0.95 0.98 0.97 0.92 0.94 0.97 0.95 0.92
 0.98 0.98 0.97 0.94 0.97 0.95]
RepeatedStratifiedKFold:
[0.99 0.96 0.97 0.97 0.95 0.98 0.97 0.98 0.97 0.96 0.97 0.99 0.94 0.96
 0.96 0.98 0.97 0.96 0.96 0.97 0.97 0.96 0.96 0.96 0.98 0.96 0.97 0.97
 0.97 0.96 0.96 0.95 0.96 0.99 0.98 0.93 0.96 0.98 0.98 0.96 0.96 0.95
 0.97 0.97 0.96 0.97 0.97 0.97 0.96 0.96]
```

cross_validate function

```
from sklearn.model_selection import cross_validate
res = cross_validate(KNeighborsClassifier(), X, y, return_train_score=True,
                     cv=5, scoring=["accuracy", "roc_auc"])
res_df = pd.DataFrame(res)
```

fit_time	score_time	test_accuracy	test_roc_auc	train_accuracy	train_roc_auc
0.000839	0.010204	0.965217	0.996609	0.980176	0.997654
0.000870	0.014424	0.956522	0.983689	0.975771	0.998650
0.000603	0.009298	0.982301	0.999329	0.971491	0.996977
0.000698	0.006670	0.955752	0.984071	0.978070	0.997820
0.000611	0.006559	0.964602	0.994634	0.978070	0.998026

Defaults in scikit-learn

- 5-fold is a default number of folds (since v.0.22)
  - Usually 5 or 10 folds
- Stratified:
  - train_test_split has stratify option: train_test_split(X, y, stratify=y)
  - cross_val_score, GridSearchCV: int, to specify the number of folds in a (Stratified)KFold,
- Shuffle:
  - train_test_split: by default (through random_state)
  - KFold and StartifiedKFold!
    - No shuffle by default for cross-validation
    - If your data is sorted, you are bad.
    - So if you run cross-validation twice with the default parameters, it will yield exactly the same results.

References

- Gareth James et al., ‘An Introduction to Statistical Learning with Python’, Ch5.
- Andreas C. Müller and Sarah Guido. ***Introduction to Machine Learning with Python: A Guide for Data Scientists***. O'Reilly Media; 1 edition.
- https://scikit-learn.org/stable/modules/cross_validation.html
- Nuwee Wiwattanawatana DS511 Data Science SWU

LOO CV

- For the following datasets, compute the Leave-One-Out Cross-validation accuracy:
  - 1-NN
  - 3-NN