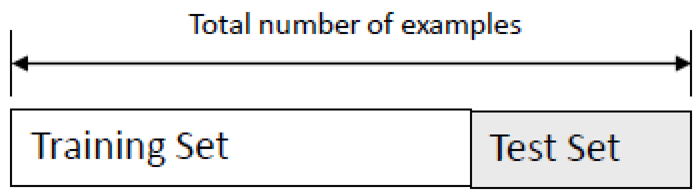
Cross Validation

The holdout method

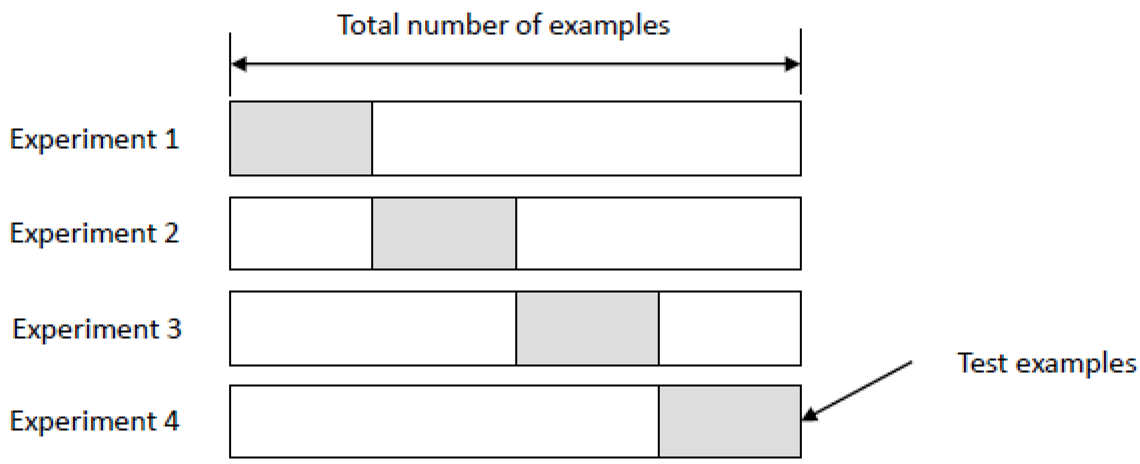
- Split dataset into two groups
  - Training set: used to train the classifier
  - Test set: used to estimate the error rate of the trained classifier



- Drawbacks
  - Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an "unfortunate" split
  - In problems where we have a sparse dataset we may not be able to afford the "luxury" of setting aside a portion of the dataset for testing

K-fold cross validation (KFCV)

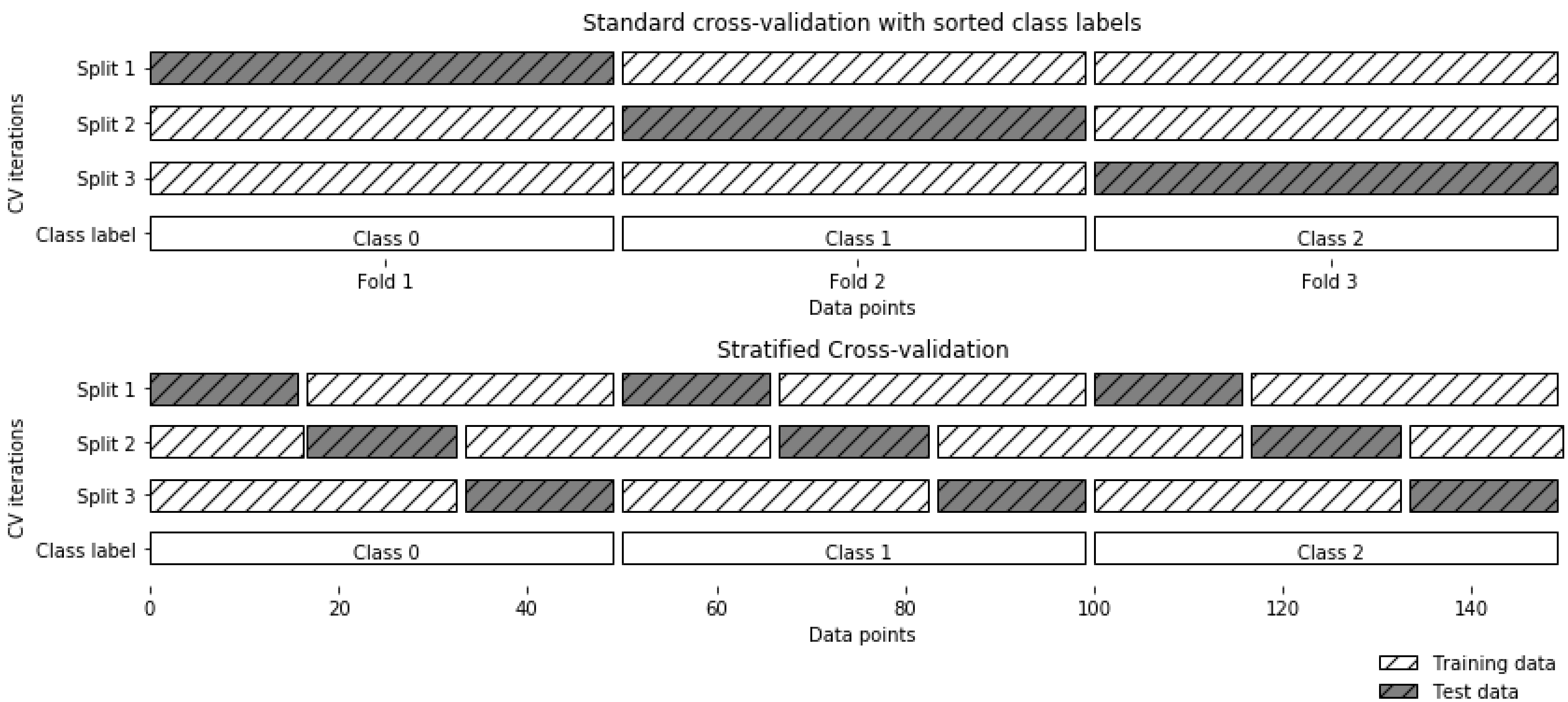
- Create a K-fold partition of the dataset
  - For each of K experiments, use $K-1$ folds for training and a different fold for testing
  - This procedure is illustrated in the following figure for K=4



- Advantages
  - The advantage of KFCV is that all the examples in the dataset are eventually used for both training and testing
  - The error is estimated as the average error rate on test examples

Stratified KFCV

- Stratified KFCV rearranges the data as to ensure each fold is a good representative of the whole
- It is generally a better scheme, both in terms of bias and variance, when compared to standard cross-validation



```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree, X_train, y_train); scores
```

```
array([ 0.93555556,  0.93541203,  0.93303571])
```

```python
# Change k
scores = cross_val_score(tree, X_train, y_train, cv=5); scores
```

```
array([ 0.92962963,  0.93703704,  0.92962963,  0.94052045,  0.9291
0448])
```

```python
print("Mean: {:.3f}\nMin: {:.3f}\nMax: {:.3f}".format(
    scores.mean(), scores.min(), scores.max()))
```

```
Mean: 0.933
Min: 0.929
Max: 0.941
```

```python
# Change performance measure
cross_val_score(tree, X_train, y_train, cv=5, scoring='roc_auc')
```

```
array([ 0.8243408 ,  0.86625514,  0.87768633,  0.92283951,  0.8773
8398])
```

Other cross validation methods

- Leave-one-out cross-validation (LOOCV)
- Shuffle-split cross-validation
- Cross-validation with groups
- Nested cross-validation

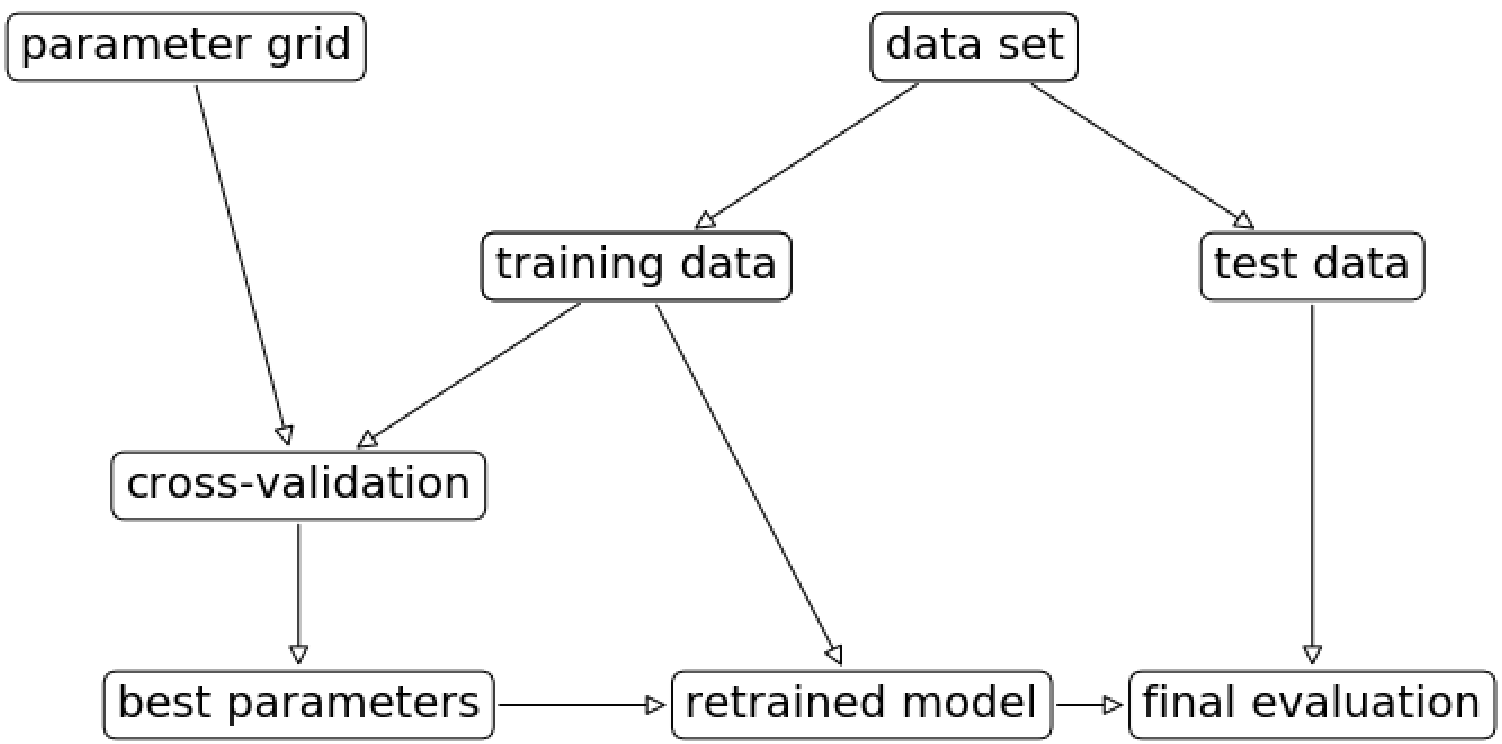
For more information, refer to :

- *http://scikit-learn.org/stable/modules/cross_validation.html*
- *"Introduction to Machine Learning", pp.312-316, pp.332-334*

Model Tuning

Model(Hyper-parameter) Tuning Workflow

- Parameter estimation using grid search with cross-validation
  - grid search: an exhaustive searching through a manually specified subset of the hyper-parameter space of a learning algorithm
  - using sklearn.model_selection.GridSearchCV



Set the parameters for grid search

```
# param_grid: dictionary with parameters names as keys and
# lists of parameter settings to try as values
param_grid = {'n_estimators': [100, 200, 300],
              'max_features': range(7,10)}
param_grid
```

```
{'max_features': range(7, 10), 'n_estimators': [100, 200, 300]}
```

Grid search with cross-validation

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(max_depth=5, random_state=0)
grid_search = GridSearchCV(rf, param_grid, cv=5, n_jobs=-1)
```

```
# grid search is very time-consuming
grid_search.fit(X_train, y_train)
```

Evaluate the model with best parameters

```
grid_search.score(X_test, y_test)
```

```
0.9666666666666667
```

```
print("Best parameters: {}".format(grid_search.best_params_))
print("Best CV score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'max_features': 9, 'n_estimators': 200}
Best CV score: 0.97
```

```
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
Best estimator:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=5, max_features=9, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

AutoML

- The goal is to automate the building of ML pipelines
- Open-source AutoML packages
  - The Tree-Based Pipeline Optimization Tool (TPOT)
  - Hyperopt
  - scikit-optimize