**Applied Machine Learning**

# Introduction to Supervised Learning

# Supervised Learning

$$(x_i, y_i) \propto p(x, y) \text{ i.i.d.}$$

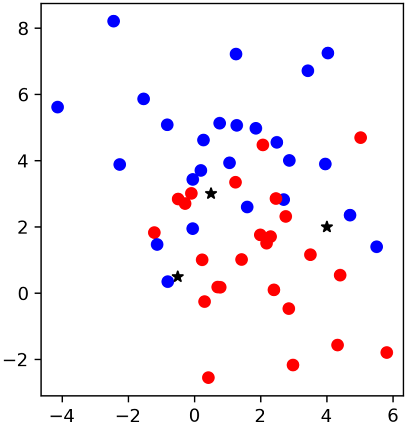
$$x_i \in \mathbb{R}^p$$

$$y_i \in \mathbb{R}$$

$$f(x_i) \approx y_i$$

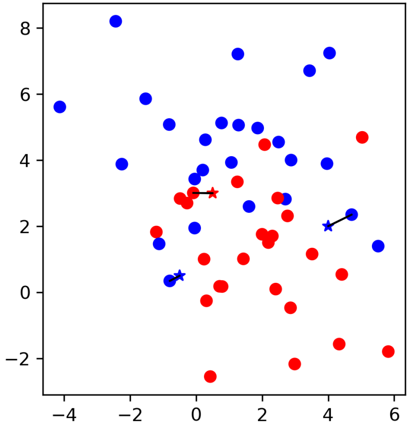
$$f(x) \approx y$$

# Nearest Neighbors

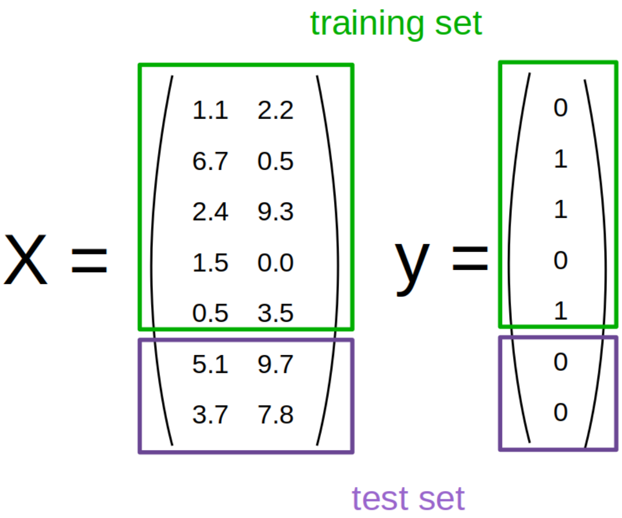


$$f(x) = y_i, i = \text{argmin}_j ||x_j - x||$$

# Nearest Neighbors



$$f(x) = y_i, i = \text{argmin}_j ||x_j - x||$$



training set

$$X = \begin{pmatrix} 1.1 & 2.2 \\ 6.7 & 0.5 \\ 2.4 & 9.3 \\ 1.5 & 0.0 \\ 0.5 & 3.5 \\ 5.1 & 9.7 \\ 3.7 & 7.8 \end{pmatrix} \quad y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

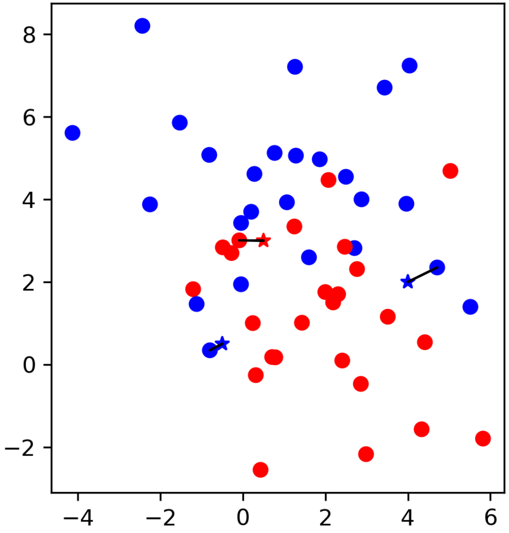
test set

# KNN with scikit-learn

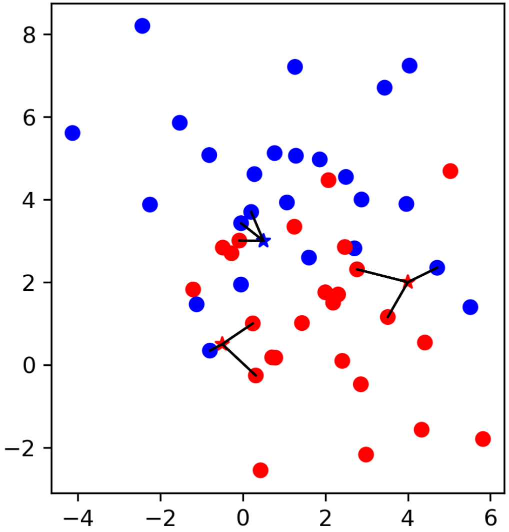
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("accuracy: {:.2f}".format(knn.score(X_test, y_test)))
y_pred = knn.predict(X_test)
```

accuracy: 0.77

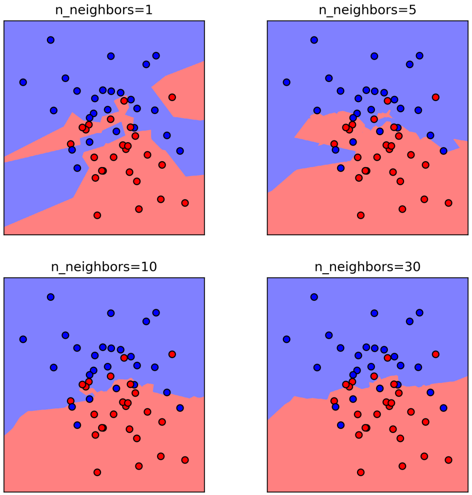
# More neighbors



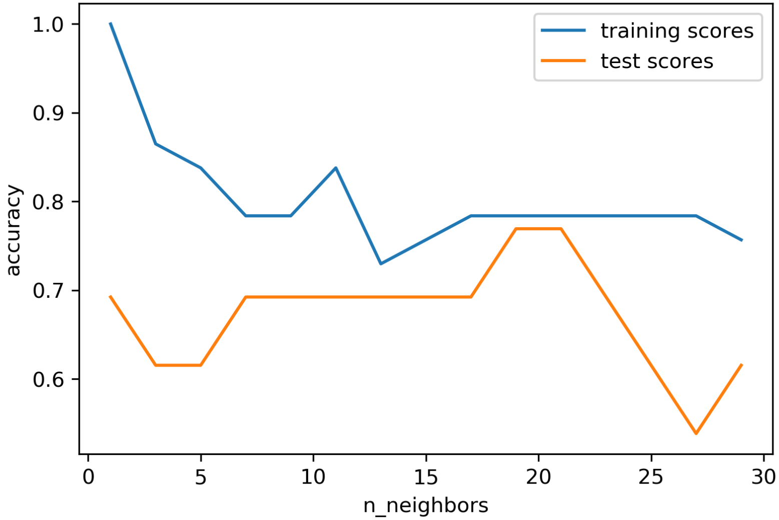
# More neighbors



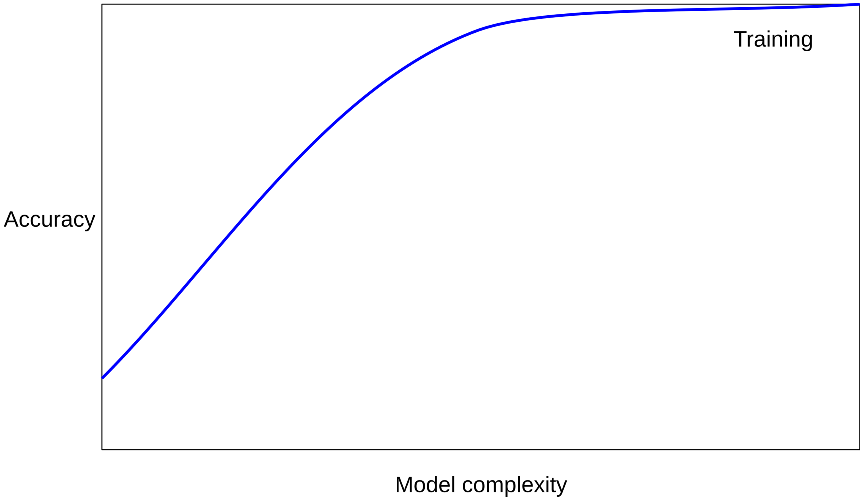
# Influence of n_neighbors



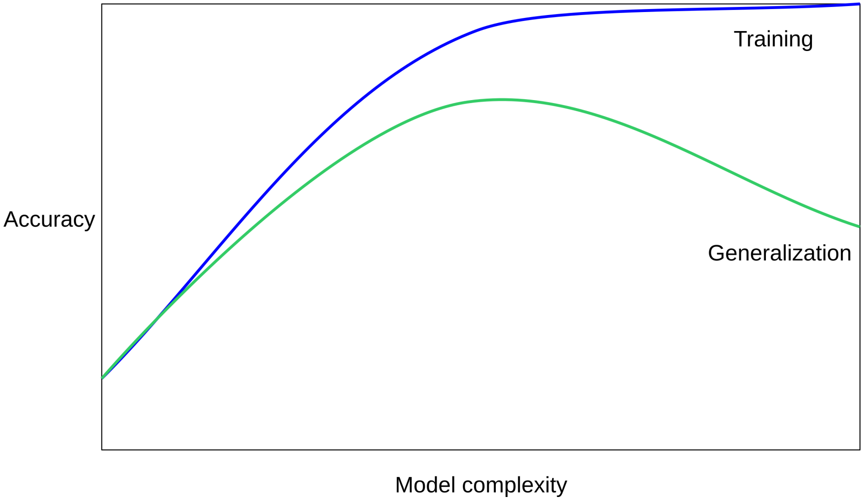
# Model complexity



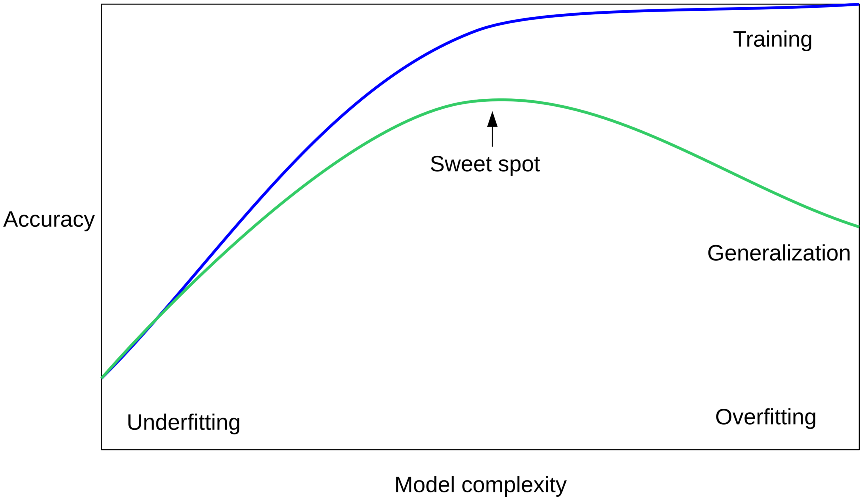
# Overfitting and Underfitting



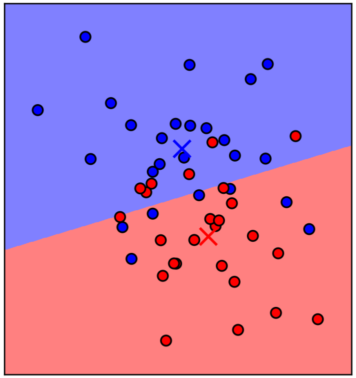
# Overfitting and Underfitting



# Overfitting and Underfitting



# Nearest Centroid



$$f(x) = \mathrm{argmin}_{i \in Y} ||\bar{x}_i - x||$$

# Nearest Centroid with scikit-learn

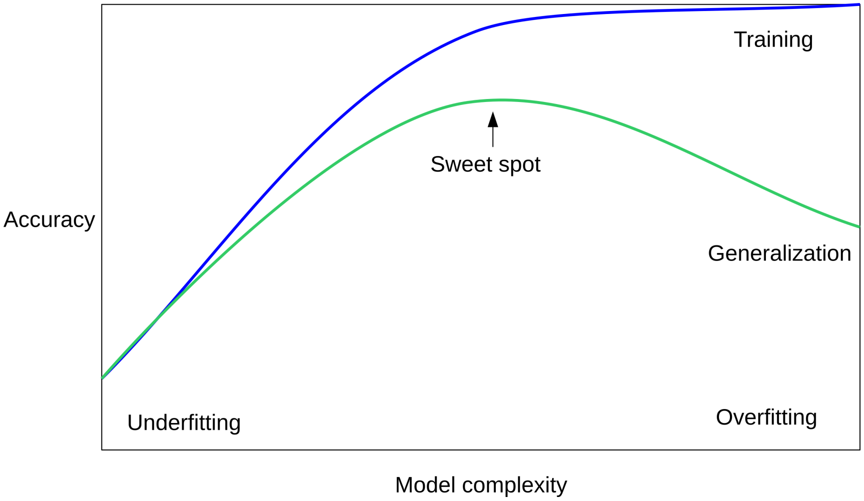
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

from sklearn.neighbors import NearestCentroid
nc = NearestCentroid()
nc.fit(X_train, y_train)
print("accuracy: {:.2f}".format(nc.score(X_test, y_test)))
```

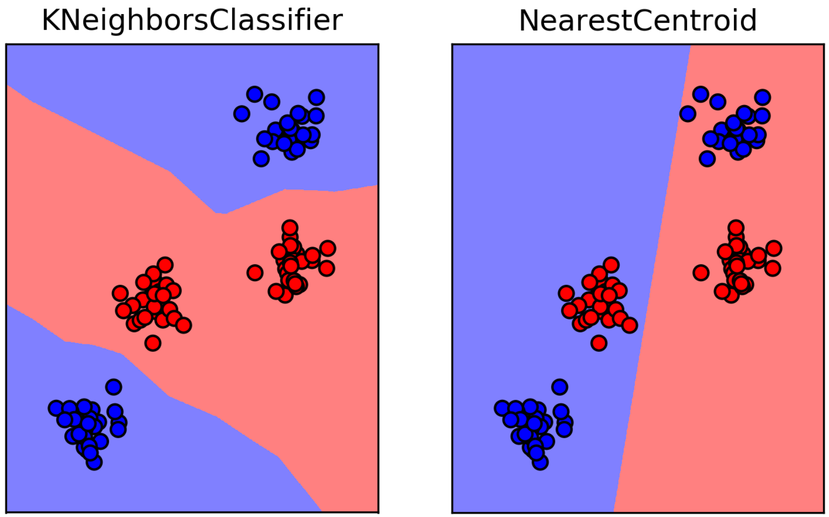
```
accuracy: 0.62
```

# Parametric and non-parametric models

- Parametric model: Number of “parameters” (degrees of freedom) independent of data.
- Non-parametric model: Degrees of freedom increase with more data.

# Overfitting and Underfitting





KNeighborsClassifier

NearestCentroid

training set

$$X = \left(\begin{array}{cc} 1.1 & 2.2 \\ 6.7 & 0.5 \\ 2.4 & 9.3 \\ 1.5 & 0.0 \\ 0.5 & 3.5 \\ \hline 5.1 & 9.7 \\ 3.7 & 7.8 \end{array}\right) \quad y = \left(\begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ \hline 0 \\ 0 \end{array}\right)$$

# Overfitting the validation set

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import scale

data = load_breast_cancer()
X, y = data.data, data.target
X = scale(X)

X_trainval, X_test, y_trainval, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(
    X_trainval, y_trainval)

knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)

print("Validation: {:.3f}".format(knn.score(X_val, y_val)))
print("Test: {:.3f}".format(knn.score(X_test, y_test)))
```

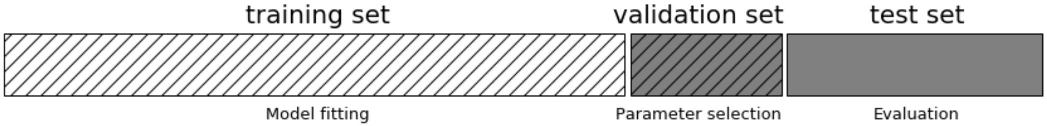
```
Validation: 0.972
Test: 0.965
```

# Overfitting the validation set

```python
val = []
test = []

for i in range(1000):
    rng = np.random.RandomState(i)
    noise = rng.normal(scale=.1, size=X_train.shape)
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train + noise, y_train)
    val.append(knn.score(X_val, y_val))
    test.append(knn.score(X_test, y_test))

print("Validation: {:.3f}".format(np.max(val)))
print("Test: {:.3f}".format(test[np.argmax(val)]))
```

```
Validation: 1.000
Test: 0.958
```

# Threefold split

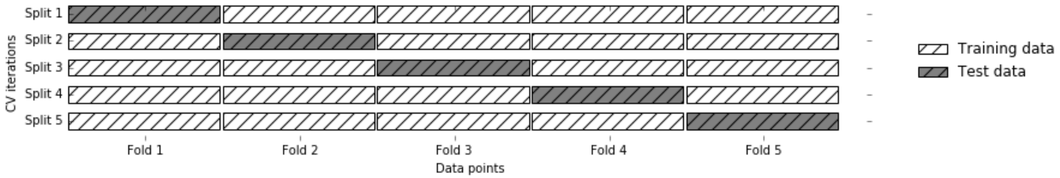


# Implementing threefold split

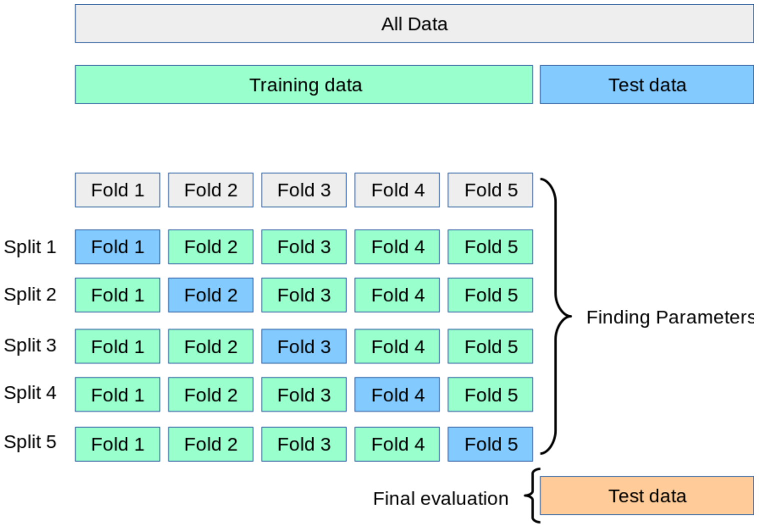
```
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval)

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: 0.991
best n_neighbors: 11
test-set score: 0.951
```

# Cross-validation



# Cross-validation + test set



All Data

Training data | Test data

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| Split 1 | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| Split 2 | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| Split 3 | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| Split 4 | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |
| Split 5 | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 |

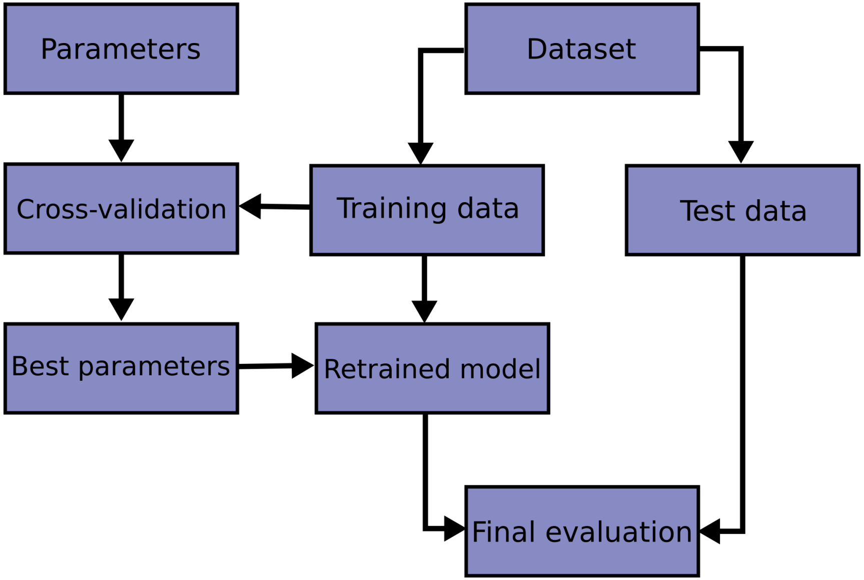
Finding Parameters

Final evaluation | Test data

# Grid-Search with Cross-Validation

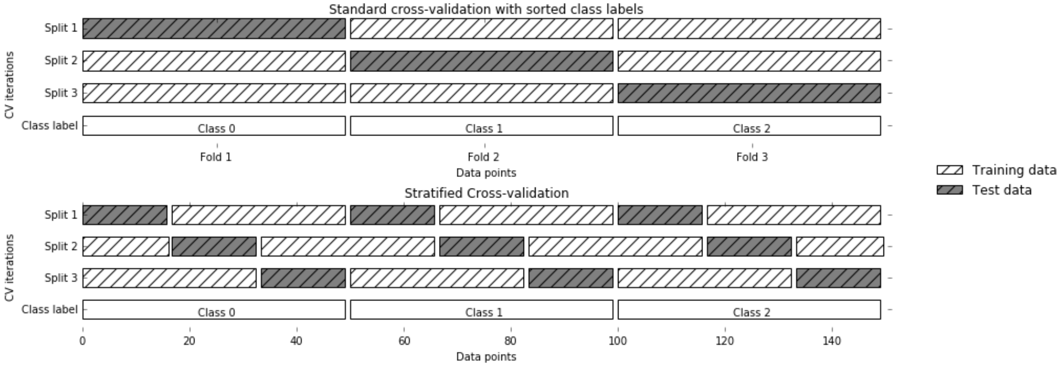
```python
from sklearn.model_selection import cross_val_score

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: 0.967
best n_neighbors: 9
test-set score: 0.965
```


Parameters
Dataset
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation

# Cross-Validation Strategies

# StratifiedKFold



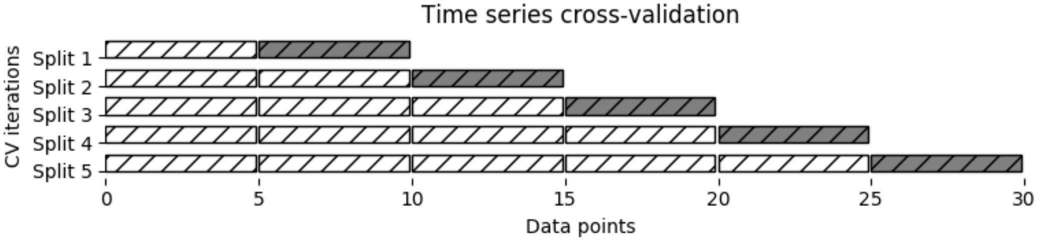
Stratified: Ensure relative class frequencies in each fold reflect relative class frequencies on the whole dataset.

# Repeated KFold and LeaveOneOut

- LeaveOneOut : KFold(n_folds=n_samples) High variance, takes a long time

- Better: RepeatedKFold. Apply KFold or StratifiedKFold multiple times with shuffled data. Reduces variance!

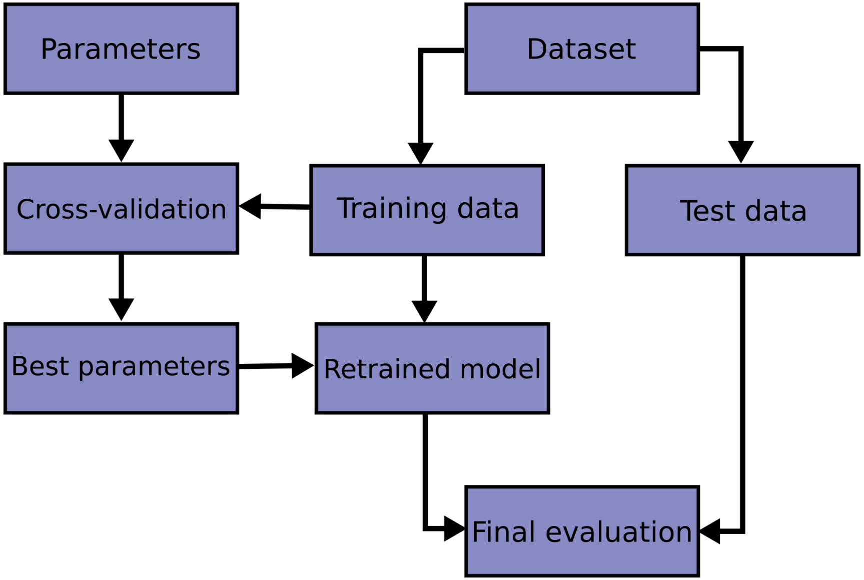
# TimeSeriesSplit



# Using Cross-Validation Generators

```python
from sklearn.model_selection import KFold, StratifiedKFold
kfold = KFold(n_splits=5)
skfold = StratifiedKFold(n_splits=5, shuffle=True)

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

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

```
KFold:
[ 0.93  0.96  0.96  0.98  0.96]
StratifiedKFold:
[ 0.97  0.95  0.98  0.96  0.96]
```


Parameters
Dataset
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation

# GridSearchCV

```
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, 15, 2)}

grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10)
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: 0.967
best parameters: {'n_neighbors': 9}
test-set score: 0.993
```

# GridSearchCV Results

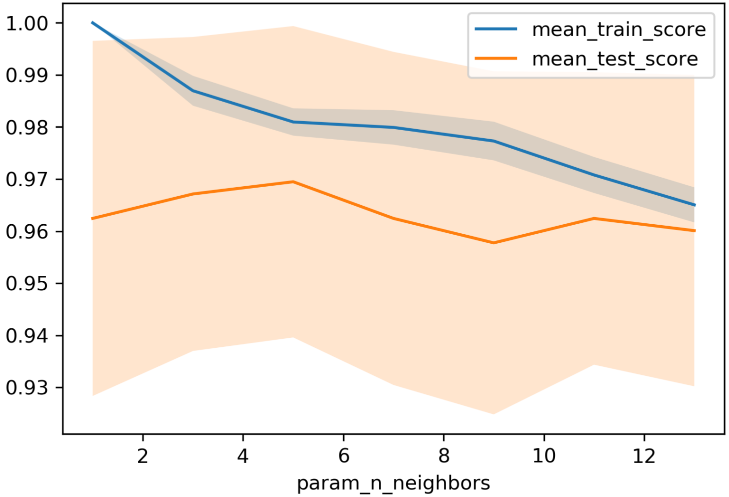
```
import pandas as pd
results = pd.DataFrame(grid.cv_results_)
results.columns
```

```
Index(['mean_fit_time', 'mean_score_time', 'mean_test_score',
       'mean_train_score', 'param_n_neighbors', 'params', 'rank_test_score',
       'split0_test_score', 'split0_train_score', 'split1_test_score',
       'split1_train_score', 'split2_test_score', 'split2_train_score',
       'split3_test_score', 'split3_train_score', 'split4_test_score',
       'split4_train_score', 'split5_test_score', 'split5_train_score',
       'split6_test_score', 'split6_train_score', 'split7_test_score',
       'split7_train_score', 'split8_test_score', 'split8_train_score',
       'split9_test_score', 'split9_train_score', 'std_fit_time',
       'std_score_time', 'std_test_score', 'std_train_score'],
      dtype='object')
```

```
results.params
```

```
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}
Name: params, dtype: object
```

# n_neighbors Search Results



Questions ?