W4995 Applied Machine Learning

Introduction to Supervised Learning

02/04/19

Andreas C. Müller

A Note on homeworks

- Submit Zip file
- Don't include .git
- `git archive master -o homework1.zip`

Supervised Learning

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

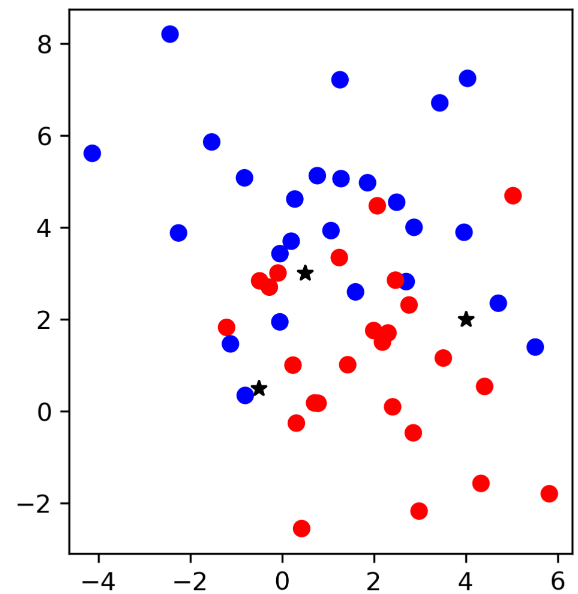
$$x_i \in \mathbb{R}^p$$

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

$$f(x_i) \approx y_i$$

$$f(x) \approx y$$

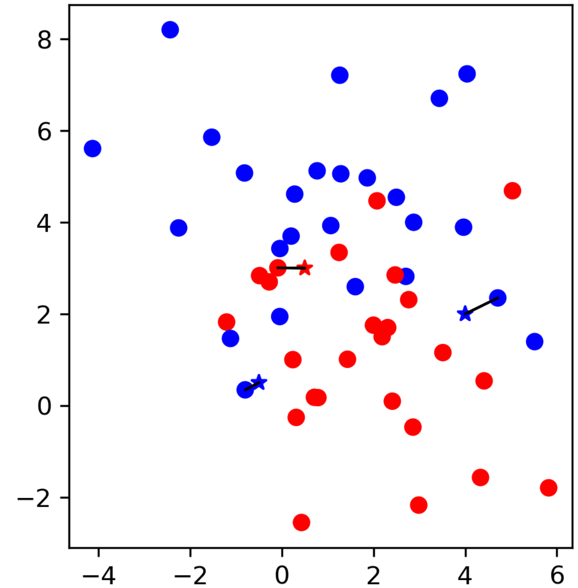
Nearest Neighbors



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

4 / 49

Nearest Neighbors



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

5 / 49

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

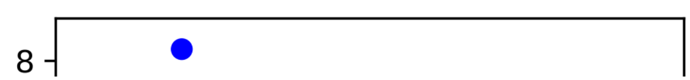
6 / 49

KNN with scikit-learn

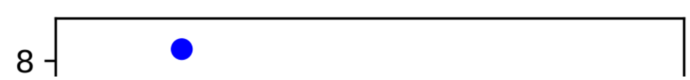
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

from sklearn.neighbors import 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

Influence of Number of Neighbors

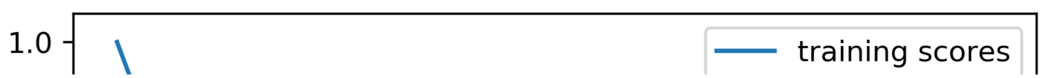


Influence of Number of Neighbors

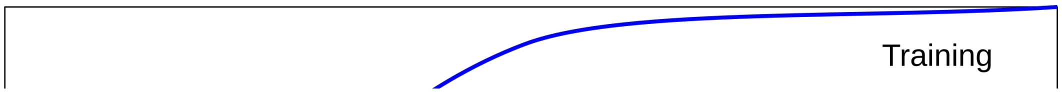


Influence of n_neighbors

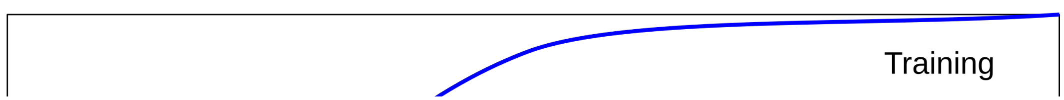
Model complexity



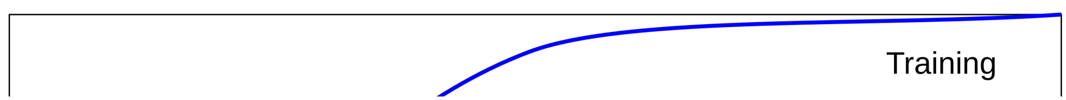
Overfitting and Underfitting



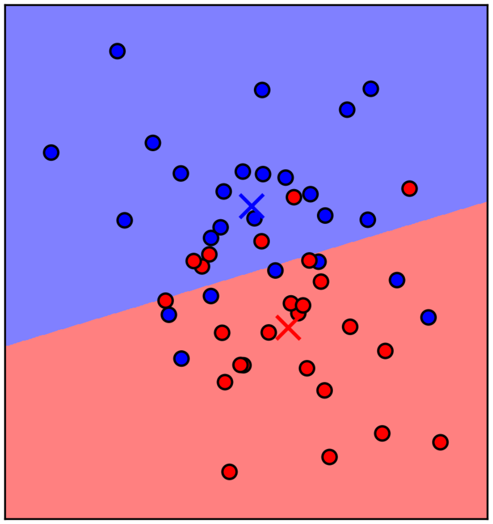
Overfitting and Underfitting



Overfitting and Underfitting



Nearest Centroid



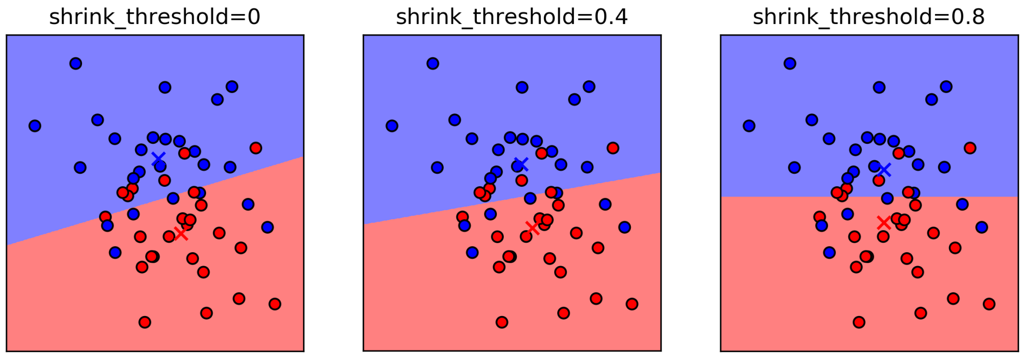
$$f(x) = \text{argmin}_{i \in Y} ||\bar{x}_i - x||$$

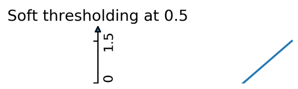
Nearest Centroid with scikit-learn

```
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
```

Nearest Shrunken Centroid





Computational Properties Centroids

Computational Properties Centroids

- fit: O(n * p)
- memory: O(n_classes * p)
- predict: O(n_classes * p)

n = n_samples p = n_features

Computational Properties Neighbors

Naive

- fit: no time
- memory: O(n * p)
- predict: O(n * p)

n=n_samples p=n_features

Computational Properties Neighbors

Naive

- fit: no time
- memory: O(n * p)
- predict: O(n * p)

Kd-tree

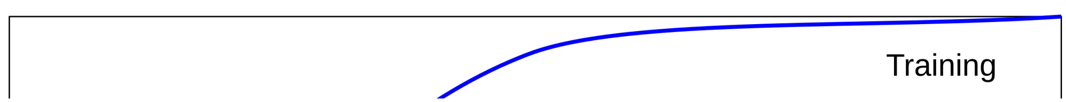
- fit: O(p * n log n)
- memory: O(n * p)
- predict:
- O(k * log(n))
  FOR FIXED p!

n=n_samples p=n_features

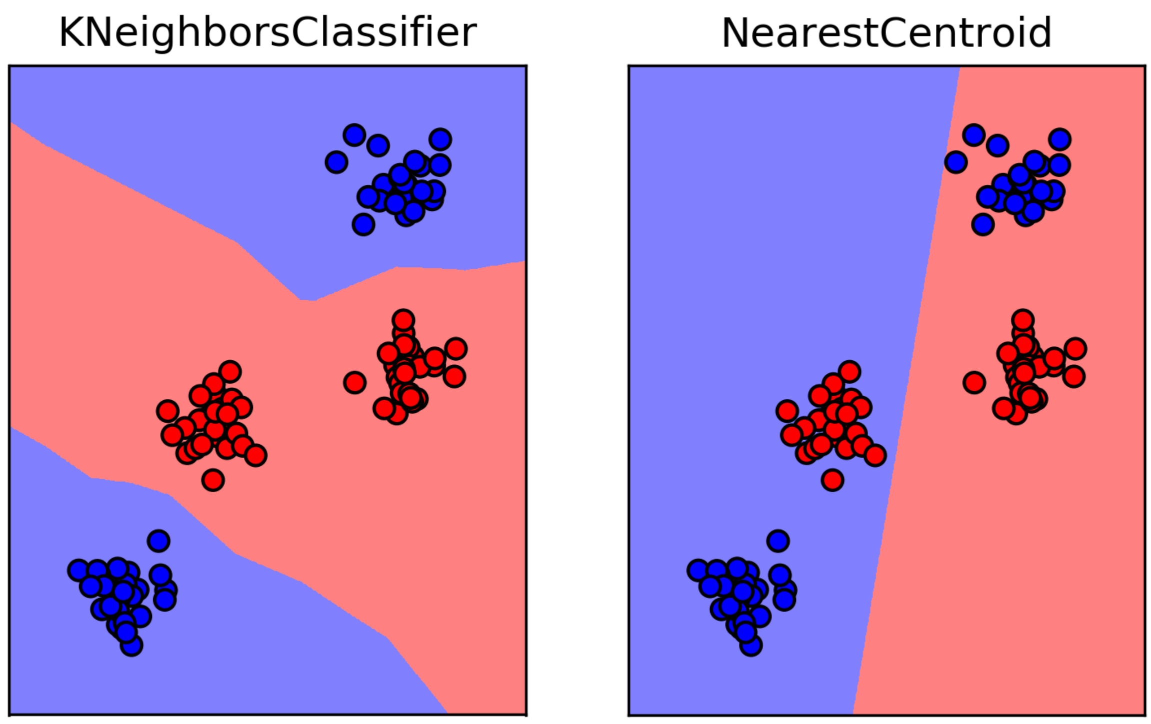
Parametric and non-parametric models

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

Overfitting and Underfitting



23 / 49


KNeighborsClassifier
NearestCentroid

training set

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)))
```

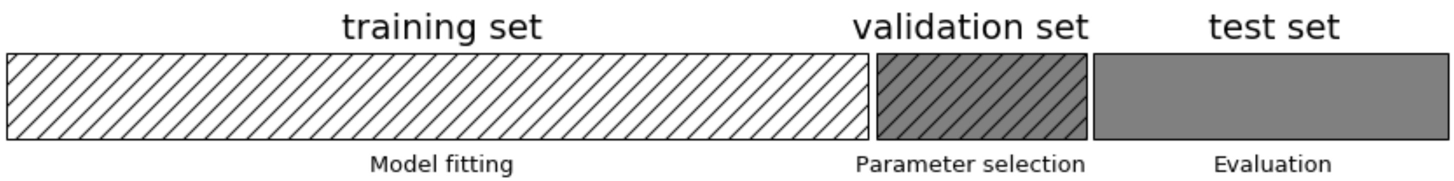
```
Validation: 0.972
Test: 0.965
```

Overfitting the validation set

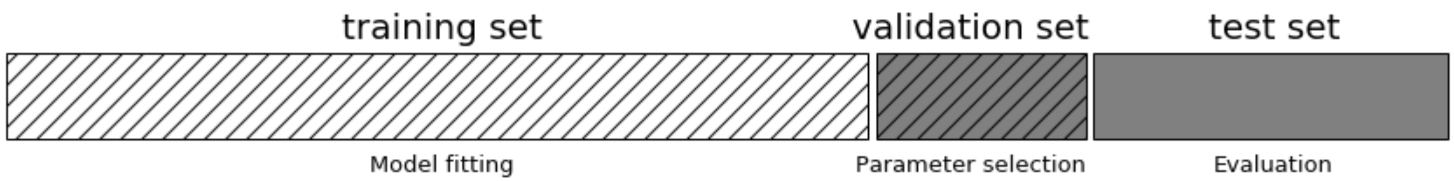
```
val = []
test = []

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



Threefold split



pro: fast, simple

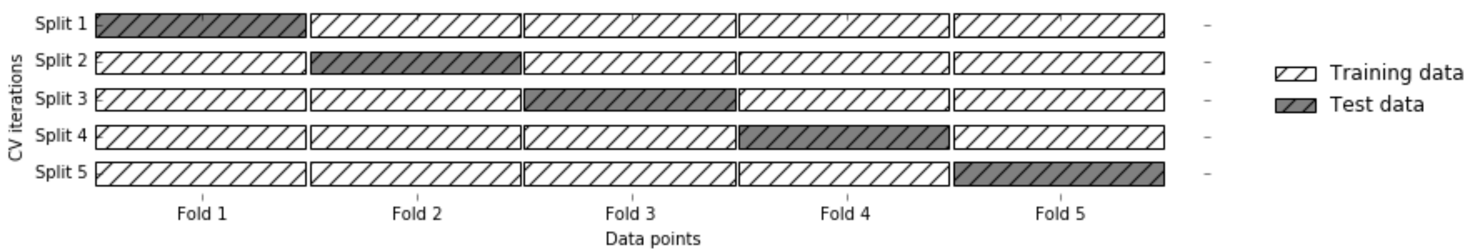
con: high variance, bad use of data

29 / 49

Threefold Split for Hyper-Parameters

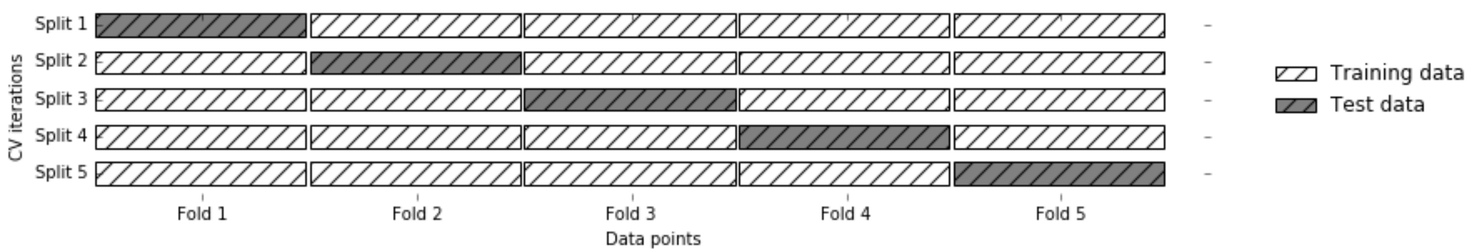
```
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval)

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:", 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)))
```

Cross-validation



31 / 49

Cross-validation



pro: more stable, more data

con: slower

Cross-validation + test set

33 / 49

Grid-Search with Cross-Validation

```
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:", 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
```


Parameters
Dataset

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, 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: 0.967
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

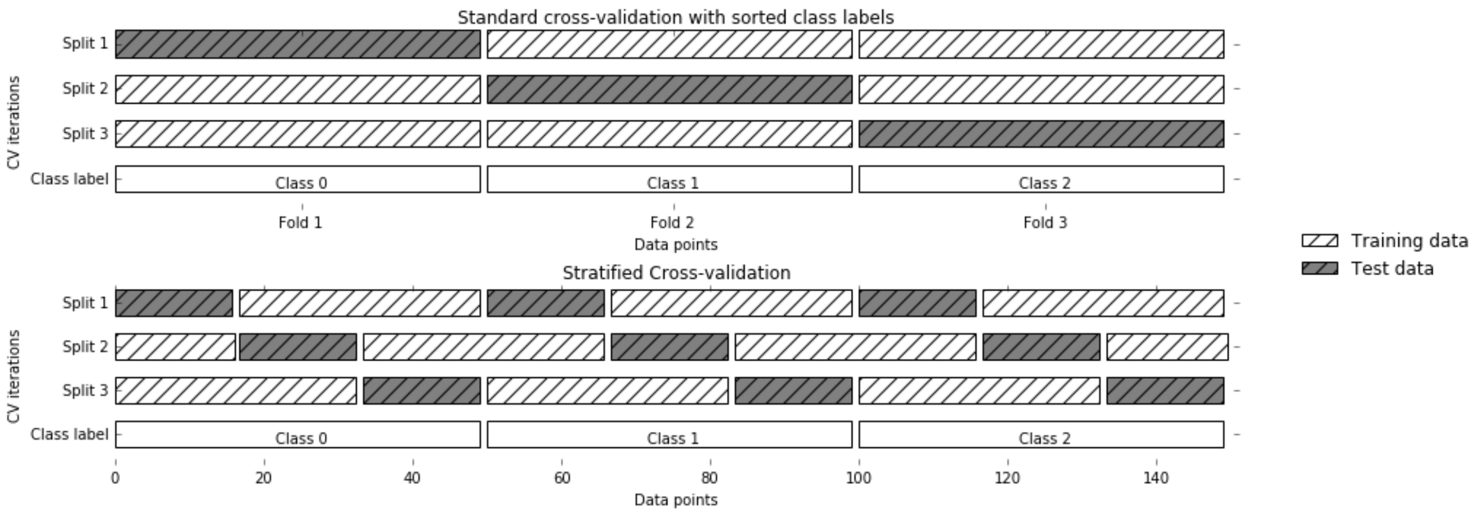
38 / 49

Nested Cross-Validation

- Replace outer split by CV loop
- Doesn’t yield single model (inner loop might have different best parameter settings)
- Takes a long time, not that useful in practice

Cross-Validation Strategies

StratifiedKFold



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

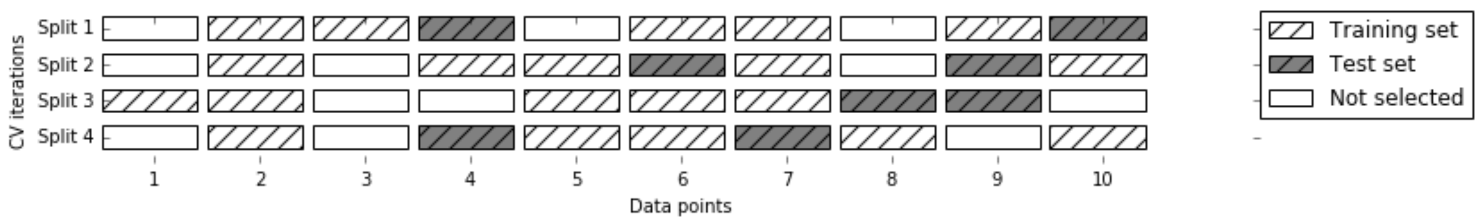
Defaults in scikit-learn

- 3-fold is the deprecated default, 5-fold in 0.22
- For classification cross-validation is stratified
- train_test_split has stratify option: train_test_split(X, y, stratify=y)
- No shuffle by default!

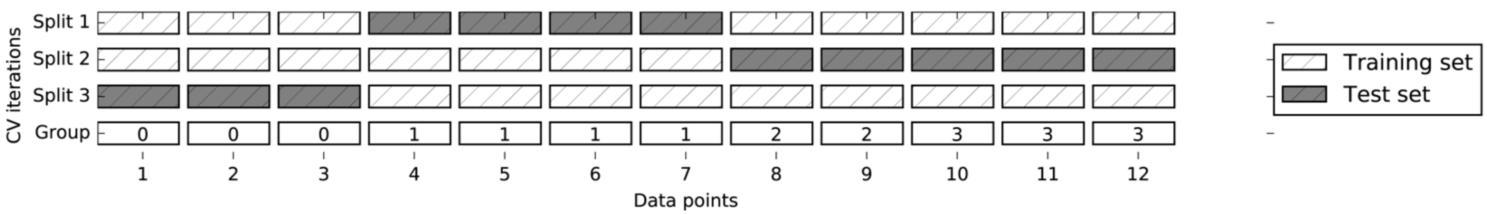
Repeated KFold and LeaveOneOut

- 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!

ShuffleSplit / StratifiedShuffleSplit

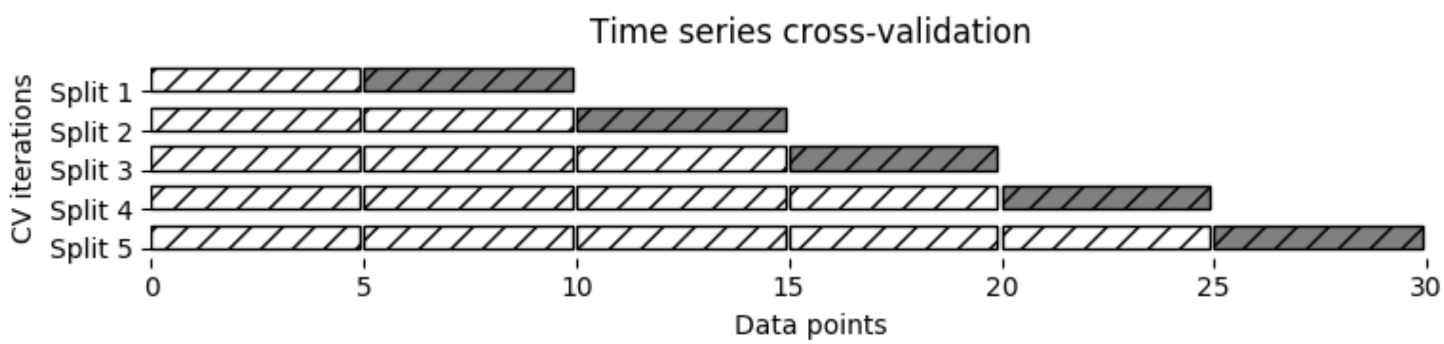


GroupKFold



45 / 49

TimeSeriesSplit



46 / 49

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:")
print(cross_val_score(KNeighborsClassifier(), X, y, cv=kfold))

print("StratifiedKFold:")
print(cross_val_score(KNeighborsClassifier(), X, y, cv=skfold))

print("ShuffleSplit:")
print(cross_val_score(KNeighborsClassifier(), X, y, cv=ss))

print("RepeatedStratifiedKFold:")
print(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

Questions ?