W4995 Applied Machine Learning

Introduction to Supervised Learning

02/04/19

Andreas C. Müller

(Adapted and modified for CC 6021236 @ PCC/Ciencias/UCV by Eugenio Scalise, July 2019)

Supervised Learning

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

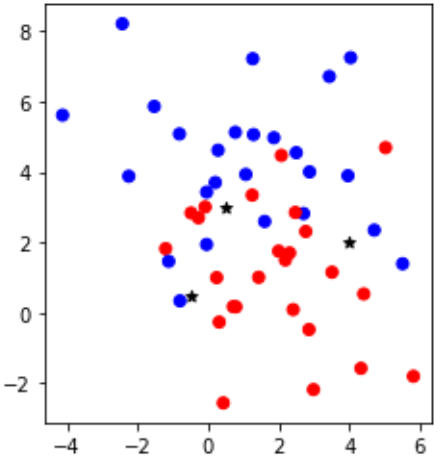
$$x_i \in \mathbb{R}^p$$

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

$$f(x_i) \approx y_i$$

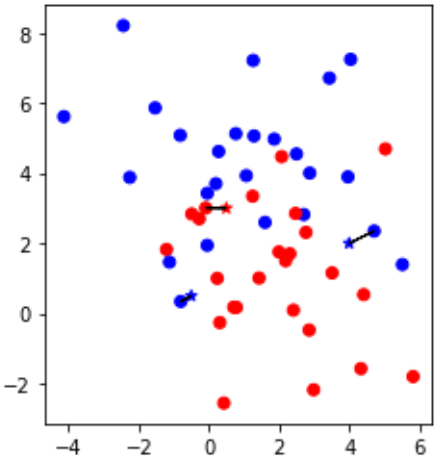
$$f(x) \approx y$$

Nearest Neighbors

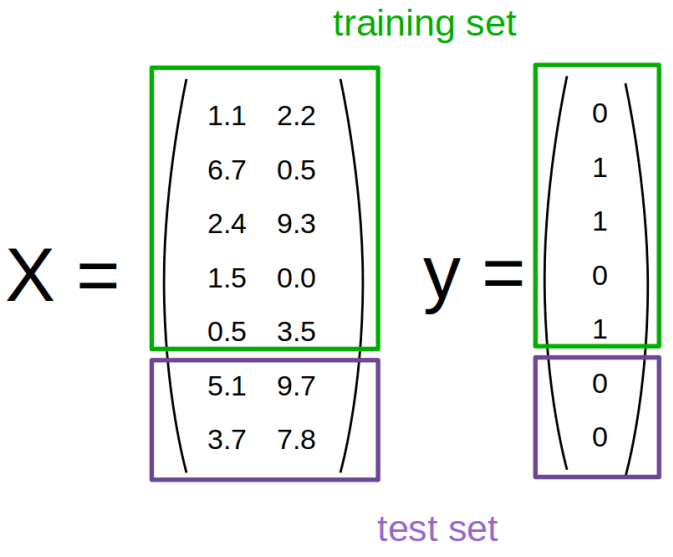


$$f(x) = y_i, i = \text{argmin}_j ||x_j - x||$$

Nearest Neighbors



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

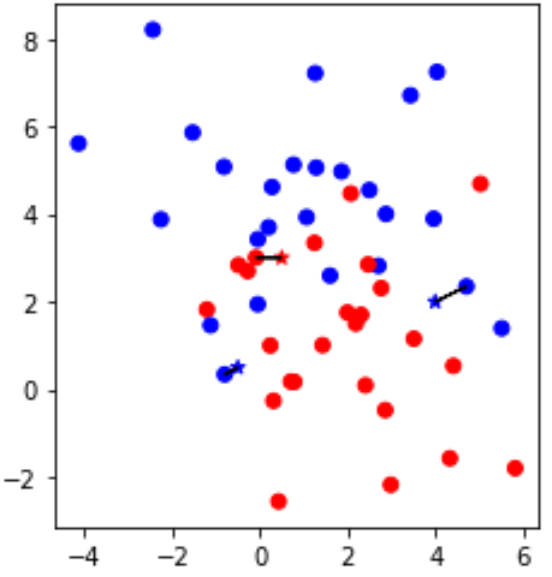

training set
X =
1.1 2.2
6.7 0.5
2.4 9.3
1.5 0.0
0.5 3.5
5.1 9.7
3.7 7.8
y =
0
1
1
0
1
0
0
test set

KNN with scikit-learn

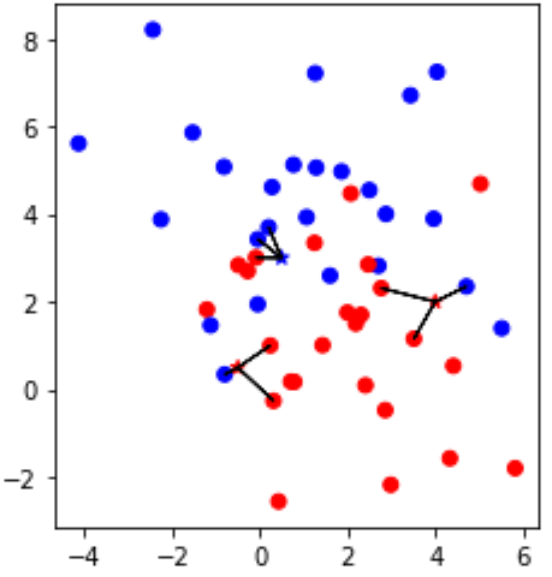
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

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

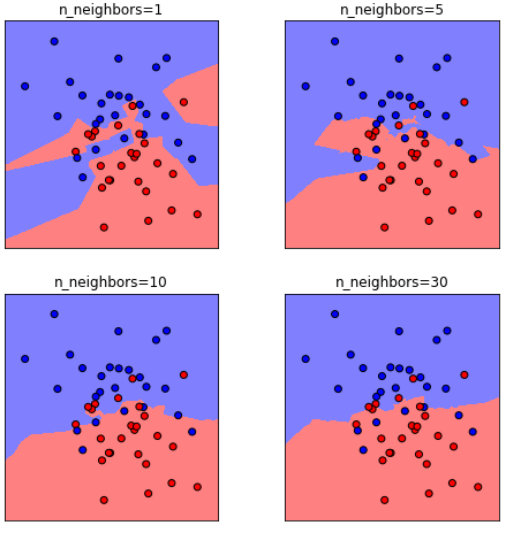
Influence of Number of Neighbors



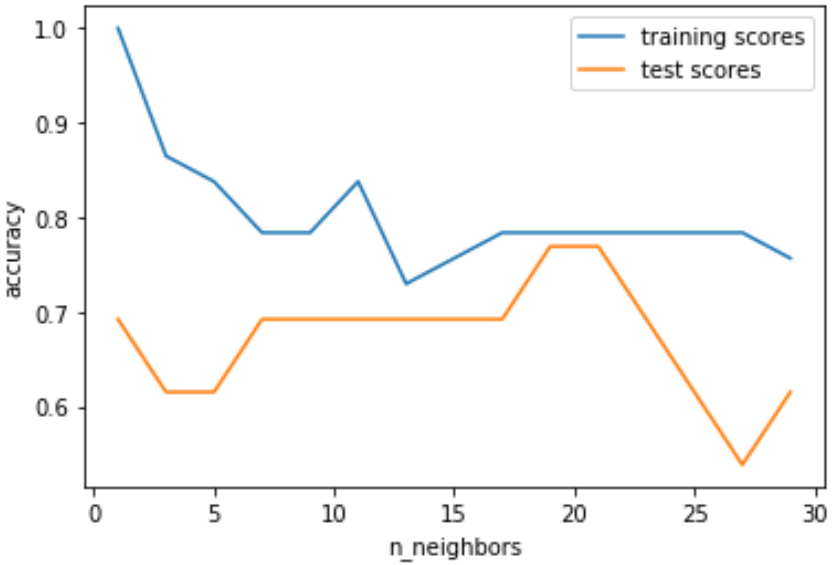
Influence of Number of Neighbors



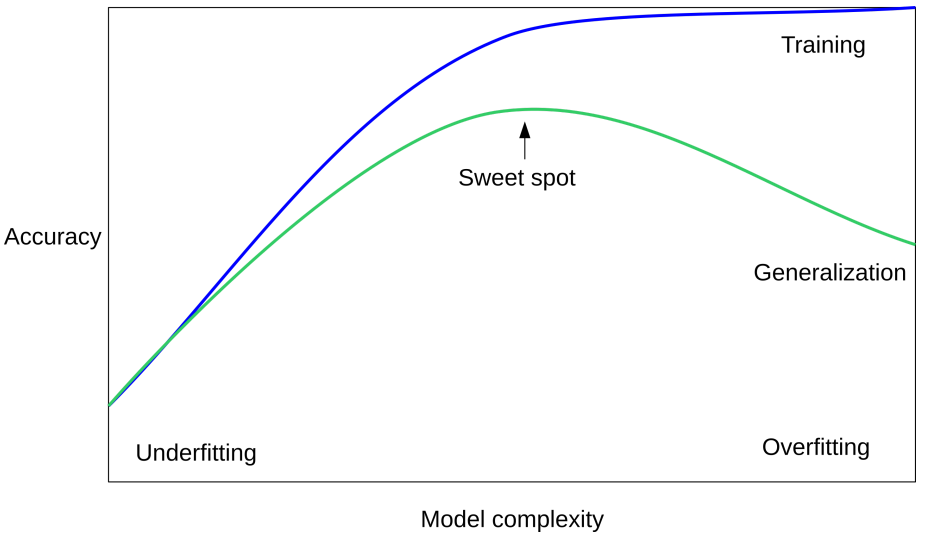
Influence of n_neighbors



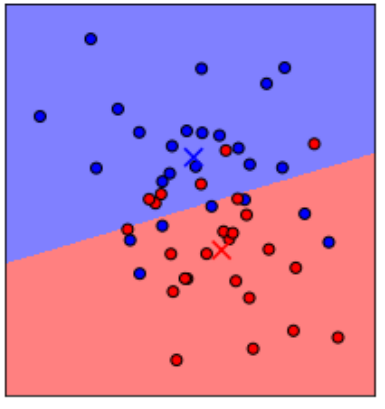
Model complexity



Overfitting and Underfitting



Nearest Centroid

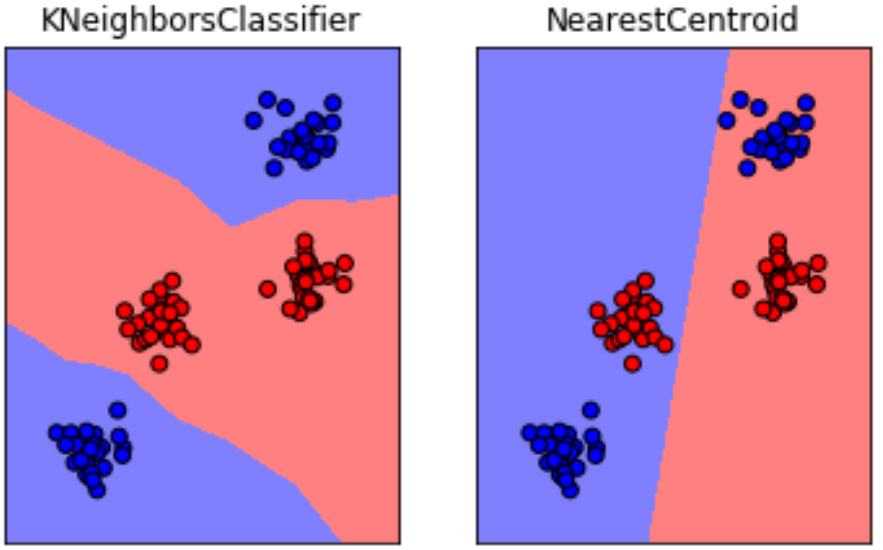


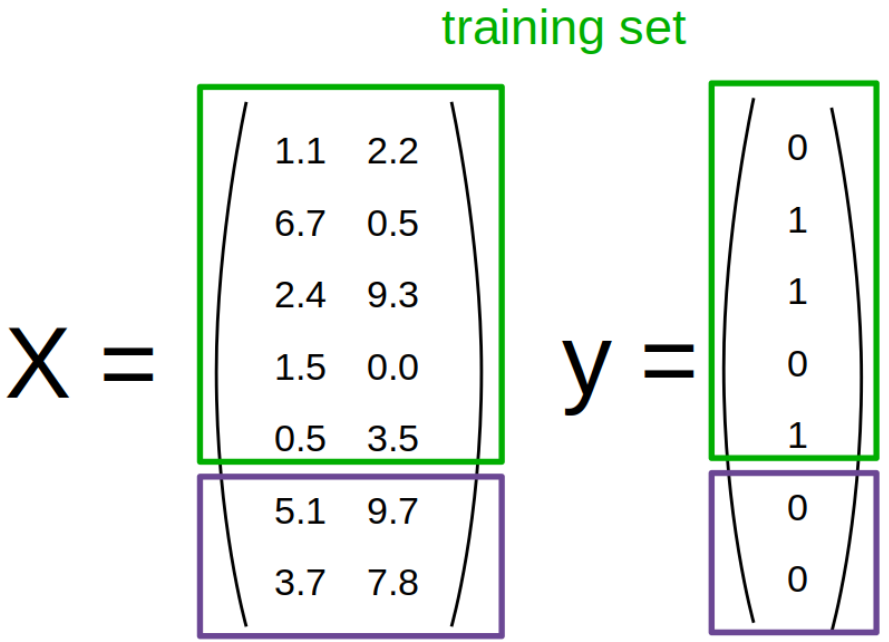
$$f(x) = \text{argmin}_{i \in Y} ||\bar{x}_i - x||$$

Nearest Centroid with scikit-learn

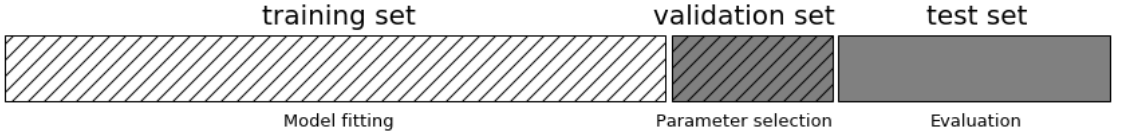
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

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
```

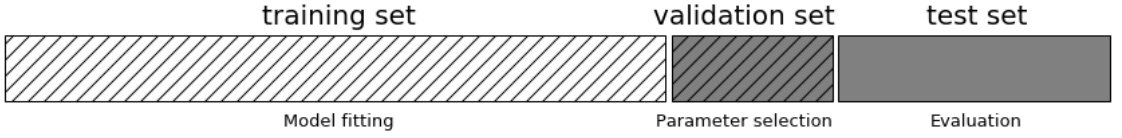

KNeighborsClassifier
NearestCentroid


training set
X =
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
y =
0
1
1
0
1
0
0

Threefold split



Threefold split



pro: fast, simple

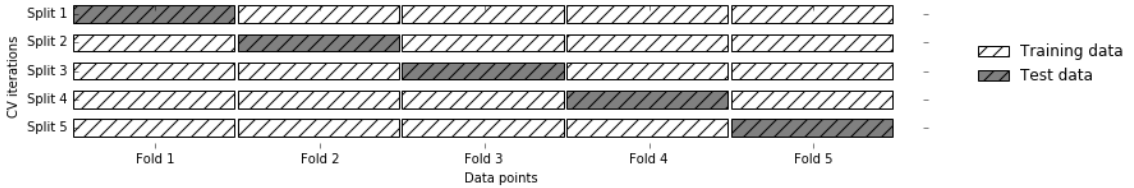
con: high variance, bad use of data

Threefold Split for Hyper-Parameters

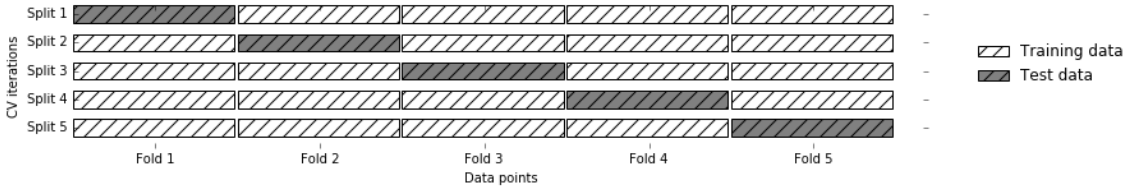
```
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval)

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)))
```

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

Cross-validation



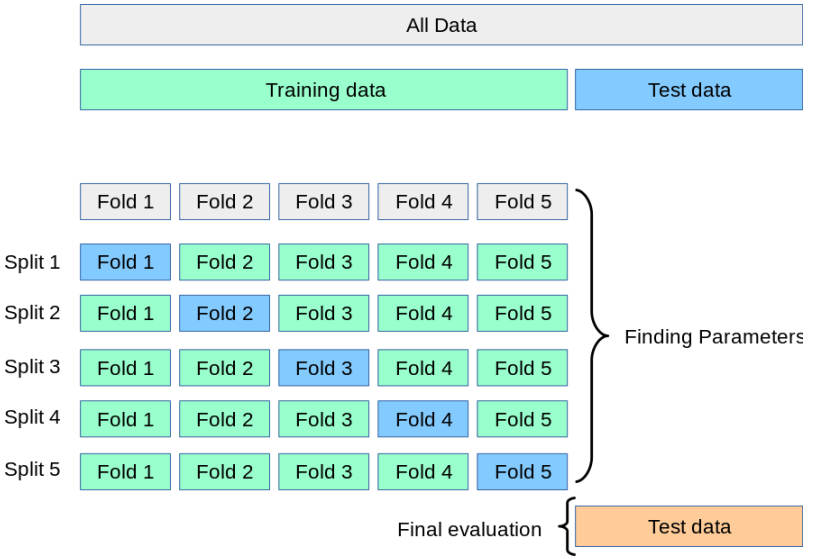
Cross-validation



pro: more stable, more data

con: slower

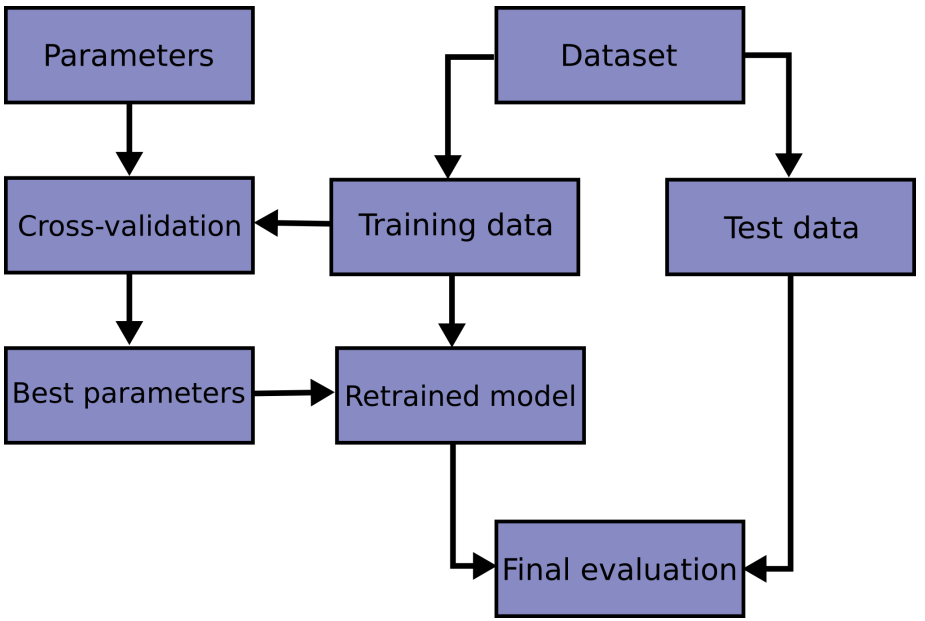
Cross-validation + test set



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
test-set score: 0.965
```


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

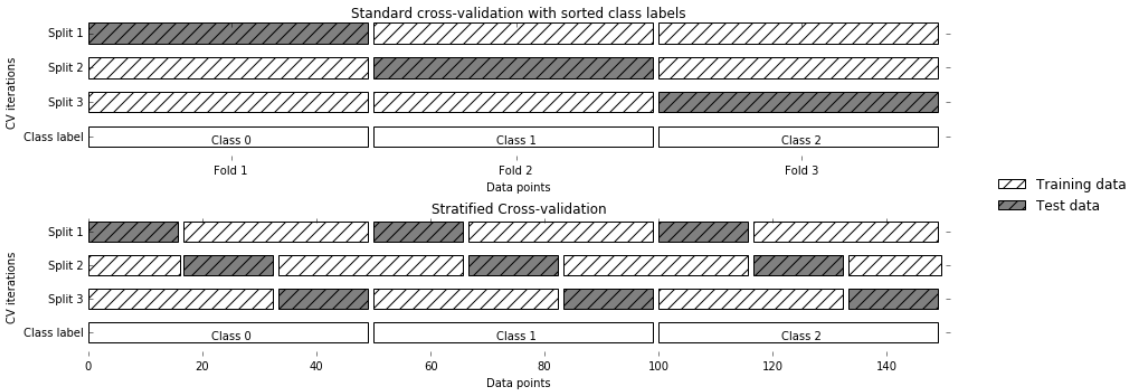
GridSearchCV

```
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, 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
```

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!

Questions ?