Mud card

- **Why is cv score higer than train score in some cases?**
  - we use MSE as our evaluation metric, so the cv score should normally be higher than the train score
  - the unusual thing is when the cv score is smaller than the train score
    - we work with small datasets due to the hub's limited computational resources
    - unlucky splits happen

- **For k-fold, when shuffle is true, when is the data shuffled? is the data shuffled before splitting into k folds? but isn't data shuffled before?**
  - the data is shuffled first, then split into folds
  - train_test_split by default shuffles the data so if you use train_test_split first, it's OK to not shuffle in KFold
  - ALWAYS CHECK THAT THE CODE DOES WHAT YOU INTEND IT TO DO!

- **how is the deviation of k-fold test mse smaller than the basic pipeline?**
  - KFold CV was run only once, so we only had one set of points in test
    - the only source of uncertainty came from changing the CV fold
  - we ran the basic ML pipeline 10 times, so we had 10 different train/CV/test sets

- **What is the meaning of "rank_test_score"?**
  - rank_test_score column in the results tells you what GridSearchCV believes to be the best hyperparameter combination

In [1]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.metrics import make_scorer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib

np.random.seed(10)
def true_fun(X):
    return np.cos(1.5 * np.pi * X)

n_samples = 100

X = np.random.rand(n_samples)
y = true_fun(X) + np.random.randn(n_samples) * 0.1
```

In [2]:
```
def ML_pipeline_kfold(X,y,random_state,n_folds):
    # split the data
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=
0.2, random_state = random_state)
    CV_scores = []
    test_scores = []
    # k folds - each fold will give us a CV and a test score
    kf = KFold(n_splits=n_folds,shuffle=True,random_state=random_state)
    for train_index, CV_index in kf.split(X_other,y_other):
        X_train, X_CV = X_other[train_index], X_other[CV_index]
        y_train, y_CV = y_other[train_index], y_other[CV_index]
        # simple preprocessing
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_c = scaler.transform(X_CV)
        X_t = scaler.transform(X_test)
        # tune ridge hyper-parameter, alpha
        alpha = np.logspace(-5,2,num=8)
        train_score = []
        CV_score = []
        regs = []
        for a in alpha:
            reg = Ridge(alpha = a)
            reg.fit(X_train,y_train)
            train_score.append(mean_squared_error(y_train,reg.predict(X_
train)))
            CV_score.append(mean_squared_error(y_CV,reg.predict(X_c)))
            regs.append(reg)
        # find the best alpha in this fold
        best_alpha = alpha[np.argmin(CV_score)]
        # grab the best model
        reg = regs[np.argmin(CV_score)]
        CV_scores.append(np.min(CV_score))
        # calculate test score using thee best model
        test_scores.append(mean_squared_error(y_test,reg.predict(X_t)))
    return CV_scores,test_scores
```

In [3]:
```
def ML_pipeline_kfold_GridSearchCV(X,y,random_state,n_folds):
    # create a test set
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=
0.2, random_state = random_state)
    # splitter for _other
    kf = KFold(n_splits=n_folds,shuffle=True,random_state=random_state)
    # create the pipeline: preprocessor + supervised ML method
    scaler = StandardScaler()
    pipe = make_pipeline(scaler,Ridge())
    # the parameter(s) we want to tune
    param_grid = {'ridge__alpha': np.logspace(-3,4,num=8)}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,scoring = make_score
r(mean_squared_error,greater_is_better=False),
                        cv=kf, return_train_score = True)
    # do kfold CV on _other
    grid.fit(X_other, y_other)
    return grid, grid.score(X_test, y_test)
```

In [4]:
```
grid, test_score = ML_pipeline_kfold_GridSearchCV(X[:,np.newaxis],y,42,5
)
results = pd.DataFrame(grid.cv_results_)
print('CV MSE:',-np.around(results[results['rank_test_score'] == 1]['mea
n_test_score'].values[0],2),\
      '+/-',np.around(results[results['rank_test_score'] == 1]['std_test
_score'].values[0],2))
print('test MSE:',-np.around(test_score,2))
print(grid.best_estimator_)
print(grid.best_score_)
print(grid.best_index_)
results
```

```
CV MSE: 0.19 +/- 0.03
test MSE: 0.16
Pipeline(memory=None,
         steps=[('standardscaler',
                 StandardScaler(copy=True, with_mean=True, with_std=Tru
e)),
                ('ridge',
                 Ridge(alpha=0.1, copy_X=True, fit_intercept=True,
                       max_iter=None, normalize=False, random_state=Non
e,
                       solver='auto', tol=0.001))],
         verbose=False)
-0.18765006501383993
2
```

Out[4]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_ridge__alpha	parar
0	0.001283	0.000445	0.000439	0.000177	0.001	{'ridge__alph 0.0(
1	0.000757	0.000086	0.000250	0.000008	0.01	{'ridge__alph 0.(
2	0.000871	0.000126	0.000379	0.000062	0.1	{'ridge__alph 0
3	0.000792	0.000164	0.000258	0.000033	1	{'ridge__alph 1
4	0.000729	0.000075	0.000252	0.000013	10	{'ridge__alph 10
5	0.000684	0.000038	0.000283	0.000070	100	{'ridge__alph 100
6	0.000764	0.000087	0.000250	0.000014	1000	{'ridge__alph 1000
7	0.000934	0.000243	0.000329	0.000114	10000	{'ridge__alph 10000

8 rows × 21 columns

Cross Validation with iid and non-iid data

By the end of this lecture, you will be able to

- use GridSearchCV with pipelines
- apply stratified splits to imbalanced data
- split based on group ID and time

Cross Validation with iid and non-iid data

By the end of this lecture, you will be able to

- **use GridSearchCV with pipelines**
- apply stratified splits to imbalanced data
- split based on group ID and time

Some notable differences between my KFold and KFold with GridSearchCV

- if multiple parameters give an equally good CV score, GridSearchCV returns the largest
  - my function returns the smallest
- GridSearchCV calculates only one test score
  - my function returns n_folds test scores
  - the GridSearchCV approach refits the best model to X_other and y_other and that model is used to calculate the test score
  - it's unclear which one is better
    - my approach allows to calculate some uncertainty due to splitting (not on test)
    - the GridSearchCV approach returns one test score but it is based on more data (likely more accurate)
- 7 lines of code in GridSearchCV
  - 28 lines of code in my function

Estimate the uncertainty from random test sets in KFold CV

Exercise 1

Calculate the test score for 10 different random splits. What's the mean and std test score?

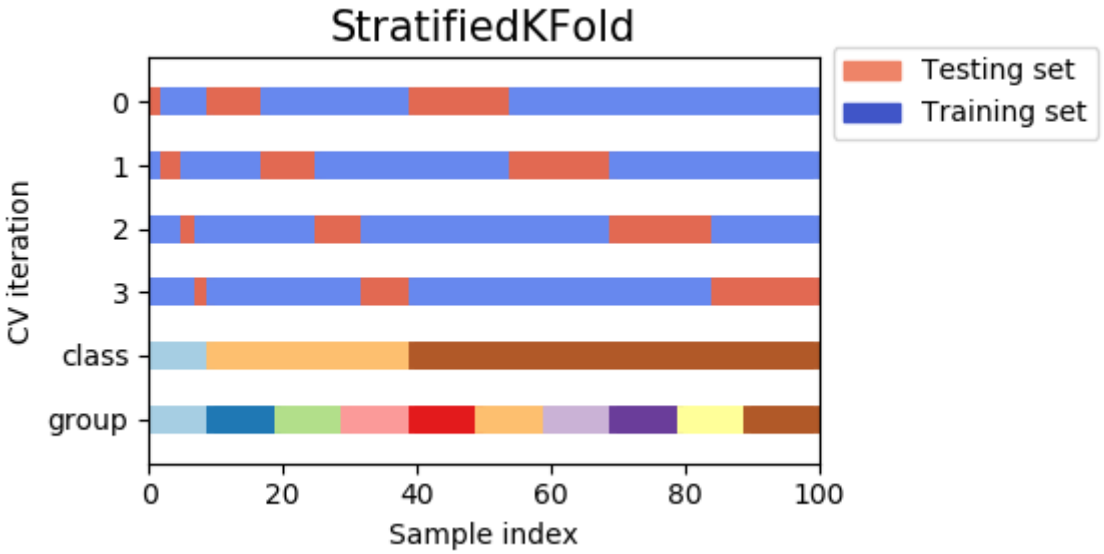
In [ ]:

Cross Validation with iid and non-iid data

By the end of this lecture, you will be able to

- use GridSearchCV with pipelines
- **apply stratified splits to imbalanced data**
- split based on group ID and time

Imbalanced data: use stratified folds



In [6]:
```
from sklearn.model_selection import StratifiedKFold
help(StratifiedKFold)
```

```
Help on class StratifiedKFold in module sklearn.model_selection._split:

class StratifiedKFold(_BaseKFold)
 |  Stratified K-Folds cross-validator
 |
 |  Provides train/test indices to split data in train/test sets.
 |
 |  This cross-validation object is a variation of KFold that returns
 |  stratified folds. The folds are made by preserving the percentage o
f
 |  samples for each class.
 |
 |  Read more in the :ref:`User Guide <cross_validation>`.
 |
 |  Parameters
 |  ----------
 |  n_splits : int, default=3
 |      Number of folds. Must be at least 2.
 |
 |      .. versionchanged:: 0.20
 |          ``n_splits`` default value will change from 3 to 5 in v0.2
2.
 |
 |  shuffle : boolean, optional
 |      Whether to shuffle each class's samples before splitting into b
atches.
 |
 |  random_state : int, RandomState instance or None, optional, default
=None
 |      If int, random_state is the seed used by the random number gene
rator;
 |      If RandomState instance, random_state is the random number gene
rator;
 |      If None, the random number generator is the RandomState instanc
e used
 |      by `np.random`. Used when ``shuffle`` == True.
 |
 |  Examples
 |  --------
 |  >>> import numpy as np
 |  >>> from sklearn.model_selection import StratifiedKFold
 |  >>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
 |  >>> y = np.array([0, 0, 1, 1])
 |  >>> skf = StratifiedKFold(n_splits=2)
 |  >>> skf.get_n_splits(X, y)
 |  2
 |  >>> print(skf)  # doctest: +NORMALIZE_WHITESPACE
 |  StratifiedKFold(n_splits=2, random_state=None, shuffle=False)
 |  >>> for train_index, test_index in skf.split(X, y):
 |  ...    print("TRAIN:", train_index, "TEST:", test_index)
 |  ...    X_train, X_test = X[train_index], X[test_index]
 |  ...    y_train, y_test = y[train_index], y[test_index]
 |  TRAIN: [1 3] TEST: [0 2]
 |  TRAIN: [0 2] TEST: [1 3]
 |
 |  Notes
 |  -----
```

```
 |  Train and test sizes may be different in each fold, with a differen
ce of at
 |  most ``n_classes``.
 |
 |  See also
 |  --------
 |  RepeatedStratifiedKFold: Repeats Stratified K-Fold n times.
 |
 |  Method resolution order:
 |      StratifiedKFold
 |      _BaseKFold
 |      BaseCrossValidator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_splits='warn', shuffle=False, random_state=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  split(self, X, y, groups=None)
 |      Generate indices to split data into training and test set.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_samples, n_features)
 |          Training data, where n_samples is the number of samples
 |          and n_features is the number of features.
 |
 |          Note that providing ``y`` is sufficient to generate the spl
its and
 |          hence ``np.zeros(n_samples)`` may be used as a placeholder
for
 |          ``X`` instead of actual training data.
 |
 |      y : array-like, shape (n_samples,)
 |          The target variable for supervised learning problems.
 |          Stratification is done based on the y labels.
 |
 |      groups : object
 |          Always ignored, exists for compatibility.
 |
 |      Yields
 |      ------
 |      train : ndarray
 |          The training set indices for that split.
 |
 |      test : ndarray
 |          The testing set indices for that split.
 |
 |      Notes
 |      -----
 |      Randomized CV splitters may return different results for each c
all of
 |      split. You can make the results identical by setting ``random_s
tate``
 |      to an integer.
 |
```

```
 |  ----------------------------------------------------------------------
---
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
---
 |  Methods inherited from _BaseKFold:
 |
 |  get_n_splits(self, X=None, y=None, groups=None)
 |      Returns the number of splitting iterations in the cross-validat
or
 |
 |      Parameters
 |      ----------
 |      X : object
 |          Always ignored, exists for compatibility.
 |
 |      y : object
 |          Always ignored, exists for compatibility.
 |
 |      groups : object
 |          Always ignored, exists for compatibility.
 |
 |      Returns
 |      -------
 |      n_splits : int
 |          Returns the number of splitting iterations in the cross-val
idator.
 |
 |  ----------------------------------------------------------------------
---
 |  Methods inherited from BaseCrossValidator:
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
---
 |  Data descriptors inherited from BaseCrossValidator:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Stratified train_test_split

In [7]:
```
help(train_test_split) # give the class labels to the stratify parameter
```

```
Help on function train_test_split in module sklearn.model_selection._sp
lit:

train_test_split(*arrays, **options)
    Split arrays or matrices into random train and test subsets

    Quick utility that wraps input validation and
    ``next(ShuffleSplit().split(X, y))`` and application to input data
    into a single call for splitting (and optionally subsampling) data
in a
    oneliner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float, int or None, optional (default=None)
        If float, should be between 0.0 and 1.0 and represent the propo
rtion
        of the dataset to include in the test split. If int, represents
the
        absolute number of test samples. If None, the value is set to t
he
        complement of the train size. If ``train_size`` is also None, i
t will
        be set to 0.25.

    train_size : float, int, or None, (default=None)
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test si
ze.

    random_state : int, RandomState instance or None, optional (default
=None)
        If int, random_state is the seed used by the random number gene
rator;
        If RandomState instance, random_state is the random number gene
rator;
        If None, the random number generator is the RandomState instanc
e used
        by `np.random`.

    shuffle : boolean, optional (default=True)
        Whether or not to shuffle the data before splitting. If shuffle
=False
        then stratify must be None.

    stratify : array-like or None (default=None)
        If not None, data is split in a stratified fashion, using this
as
        the class labels.
```

```
    Returns
    -------
    splitting : list, length=2 * len(arrays)
        List containing train-test split of inputs.

        .. versionadded:: 0.16
            If the input is sparse, the output will be a
            ``scipy.sparse.csr_matrix``. Else, output type is the same
as the
            input type.

    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>> X
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7],
           [8, 9]])
    >>> list(y)
    [0, 1, 2, 3, 4]

    >>> X_train, X_test, y_train, y_test = train_test_split(
    ...     X, y, test_size=0.33, random_state=42)
    ...
    >>> X_train
    array([[4, 5],
           [0, 1],
           [6, 7]])
    >>> y_train
    [2, 0, 3]
    >>> X_test
    array([[2, 3],
           [8, 9]])
    >>> y_test
    [1, 4]

    >>> train_test_split(y, shuffle=False)
    [[0, 1, 2], [3, 4]]
```

Cross Validation with iid and non-iid data

By the end of this lecture, you will be able to

- use GridSearchCV with pipelines
- apply stratified splits to imbalanced data
- **split based on group ID and time**

When the iid assumption breaks down

- What is the intended use of the model? What is it supposed to do/predict?
- What data do you have available at that time?
- Your cross validation must simulate the intended use of the model!

An example: seizure project

- you can read the publication here (https://ieeexplore.ieee.org/document/8857552)
- classification problem:
  - epileptic seizures vs. non-epileptic psychogenic seizures
- data from empatica wrist sensor
  - heart rate, skin temperature, EDA, blood volume pressure, acceleration
- data collection:
  - patients come to the hospital for a few days
  - eeg and video recording to determine seizure type
  - wrist sensor data is collected
- question:
  - Can we use the wrist sensor data to differentiate the two seizure types on new patients?

In [16]:
```
df = pd.read_csv('data/seizure_data.csv')
print(df[df['patient ID'] == 32])
```

```
    patient_ID            seizure_ID  ACC_mean  BVP_mean  EDA_mean    HR_mean  \
5          32  ID32__day3_arm_1_sz1  1.028539 -0.092102  0.112795  64.748167
6          32  ID32__day3_arm_1_sz1  1.027986  0.745437  0.130486  63.715667
7          32  ID32__day2_arm_1_sz0  1.002146  0.150810  0.189272  61.838500
8          32  ID32__day2_arm_1_sz0  1.005410  0.482859  1.226038  66.240833
9          32  ID32__day1_arm_1_sz0  0.997017 -0.925122  0.200990  56.103667
10         32  ID32__day1_arm_1_sz0  1.009207  1.618456  1.679754  64.668167
27         32  ID32__day1_arm_1_sz0  1.000290  0.046690  0.123165  54.289500
28         32  ID32__day1_arm_1_sz0  1.010351  0.125039  0.471180  65.060667
29         32  ID32__day2_arm_1_sz0  1.018163  0.254302  0.206010  61.875833
30         32  ID32__day2_arm_1_sz0  1.016785  1.242893  0.954649  66.216167
34         32  ID32__day3_arm_1_sz1  1.008867  0.070180  0.195966  65.995667
35         32  ID32__day3_arm_1_sz1  1.009554  0.222872  0.229909  63.871000
58         32  ID32__day3_arm_1_sz0  1.008873 -0.550857  0.177822  67.750833
79         32  ID32__day3_arm_1_sz0  1.026840  0.355953  0.205273  69.124667

    TEMP_mean  ACC_stdev   BVP_stdev  EDA_stdev  ...  BVP_50th  EDA_50th  \
5   36.944833   0.007469   36.486091   0.003905  ...     1.815   0.112710
6   36.676333   0.028190   84.964155   0.018598  ...     2.210   0.131921
7   38.600333   0.003747   64.194294   0.024278  ...     6.985   0.186026
8   39.296083   0.035257  165.665784   0.891139  ...     1.140   1.062333
9   34.656667   0.022648   77.013336   0.132008  ...     3.800   0.142159
10  34.678000   0.046047  146.515297   0.438236  ...     5.585   1.690537
27  38.467417   0.019826   51.176639   0.014530  ...     7.765   0.124259
28  38.448000   0.077142   61.205657   0.156170  ...     3.290   0.510114
29  37.681583   0.006805   40.982246   0.017099  ...     1.455   0.202632
30  37.979500   0.032493  219.277839   0.612229  ...    -5.785   1.028171
34  40.659458   0.021812   49.981175   0.013259  ...     3.480   0.198570
35  40.481333   0.048531   37.409681   0.031963  ...     0.695   0.228676
```

```
58  39.906667   0.021431   27.472002   0.003085  ...     1.955  0.17807
3
79  34.490167   0.008165   40.742936   0.003550  ...     3.090  0.20620
7

    HR_50th  TEMP_50th  ACC_75th  BVP_75th  EDA_75th  HR_75th  TEMP_75t
h  \
5    65.060      36.95  1.029947   16.3725  0.115591  65.8175     36.99
0
6    62.175      36.81  1.029947   21.1625  0.147611  66.2100     36.84
0
7    61.840      38.61  1.006085   43.8850  0.209086  61.9000     38.79
0
8    62.325      39.37  1.008872   49.4325  2.313129  71.0625     39.39
0
9    56.110      34.66  0.996821   35.2700  0.176739  56.6050     34.66
0
10   65.790      34.66  1.021497   70.4800  1.998868  67.7725     34.73
5
27   53.960      38.49  1.002073   39.8525  0.133226  54.7425     38.50
0
28   65.285      38.45  1.014302   25.4625  0.577047  69.4975     38.53
0
29   61.910      37.68  1.022811   29.2125  0.219282  61.9300     37.75
0
30   64.700      38.00  1.022811   65.5000  1.503002  69.5725     38.03
0
34   66.145      40.68  1.013700   13.1300  0.199852  67.0425     40.71
0
35   64.395      40.49  1.016106   12.9650  0.260383  65.9625     40.53
0
58   68.170      39.93  1.015264   17.8625  0.179354  68.5725     40.03
0
79   69.810      34.37  1.033260   13.4550  0.207488  70.0000     34.68
0

    label
5     0.0
6     0.0
7     0.0
8     0.0
9     0.0
10    0.0
27    0.0
28    0.0
29    0.0
30    0.0
34    0.0
35    0.0
58    0.0
79    0.0

[14 rows x 48 columns]
```

In [17]:
```
y = df['label']
patient_ID = df['patient ID']
seizure_ID = df['seizure_ID']
X = df.drop(columns=['patient ID','seizure_ID','label'])
classes, counts = np.unique(y,return_counts=True)
print('balance:',np.max(counts/len(y)))
```

```
balance: 0.6884057971014492
```

In [10]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold
def ML_pipeline_kfold_GridSearchCV(X,y,random_state,n_folds):
    # create a test set
    X_other, X_test, y_other, y_test = train_test_split(X, y, test_size=
0.2, random_state = random_state,stratify=y)
    # splitter for _other
    kf = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=rand
om_state)
    # create the pipeline: preprocessor + supervised ML method
    scaler = StandardScaler()
    pipe = make_pipeline(scaler,SVC())
    # the parameter(s) we want to tune
    param_grid = {'svc__C': np.logspace(-3,4,num=8),'svc__gamma': np.log
space(-3,4,num=8)}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,scoring = make_score
r(accuracy_score),
                        cv=kf, return_train_score = True,iid=True)
    # do kfold CV on _other
    grid.fit(X_other, y_other)
    return grid, grid.score(X_test, y_test)
```

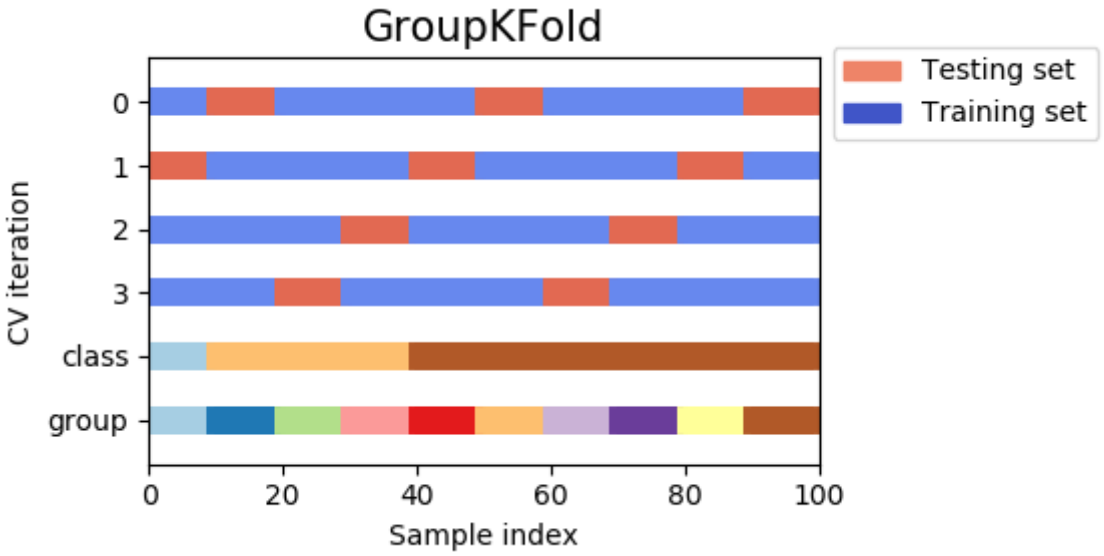
In [11]:
```
test_scores = []
for i in range(5):
    grid, test_score = ML_pipeline_kfold_GridSearchCV(X,y,i*42,5)
    print(grid.best_params_)
    print('best CV score:',grid.best_score_)
    print('test score:',test_score)
    test_scores.append(test_score)
print('test accuracy:',np.around(np.mean(test_scores),2),'+/-',np.around
(np.std(test_scores),2))
```

```
{'svc__C': 100.0, 'svc__gamma': 0.001}
best CV score: 0.9136363636363637
test score: 0.9285714285714286
{'svc__C': 10.0, 'svc__gamma': 0.01}
best CV score: 0.9454545454545454
test score: 0.9285714285714286
{'svc__C': 10.0, 'svc__gamma': 0.01}
best CV score: 0.9227272727272727
test score: 0.9464285714285714
{'svc__C': 10.0, 'svc__gamma': 0.01}
best CV score: 0.9363636363636364
test score: 0.9285714285714286
{'svc__C': 10.0, 'svc__gamma': 0.01}
best CV score: 0.9454545454545454
test score: 0.9107142857142857
test accuracy: 0.93 +/- 0.01
```

This is wrong! A very bad case of data leakage!

- the textbook case of information leakage!
- if we just do KFold CV blindly, the points from the same patient end up in different sets
  - when you deploy the model and apply it to data from new patients, that patient's data will be seen for the first time
- the ML pipeline needs to mimic the intended use of the model!
  - we want to split the points based on the patient ID!
  - we want all points from the same patient to be in either train/CV/test

Group-based split: GroupKFold



In [12]:
```
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import GroupShuffleSplit
def ML_pipeline_groups_GridSearchCV(X,y,groups,random_state,n_folds):
    # create a test set based on groups
    splitter = GroupShuffleSplit(n_splits=1,test_size=0.2,random_state=r
andom_state)
    for i_other,i_test in splitter.split(X, y, groups):
        X_other, y_other, groups_other = X.iloc[i_other], y.iloc[i_other
], groups.iloc[i_other]
        X_test, y_test, groups_test = X.iloc[i_test], y.iloc[i_test], gr
oups.iloc[i_test]
    # check the split
#     print(pd.unique(groups))
#     print(pd.unique(groups_other))
#     print(pd.unique(groups_test))
    # splitter for _other
    kf = GroupKFold(n_splits=n_folds)
    # create the pipeline: preprocessor + supervised ML method
    scaler = StandardScaler()
    pipe = make_pipeline(scaler,SVC())
    # the parameter(s) we want to tune
    param_grid = {'svc__C': np.logspace(-3,4,num=8),'svc__gamma': np.log
space(-3,4,num=8)}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,scoring = make_score
r(accuracy_score),
                        cv=kf, return_train_score = True,iid=True)
    # do kfold CV on _other
    grid.fit(X_other, y_other, groups_other)
    return grid, grid.score(X_test, y_test)
```

In [13]:
```
test_scores = []
for i in range(5):
    grid, test_score = ML_pipeline_groups_GridSearchCV(X,y,patient_ID,i*
42,5)
    print(grid.best_params_)
    print('best CV score:',grid.best_score_)
    print('test score:',test_score)
    test_scores.append(test_score)
print('test accuracy:',np.around(np.mean(test_scores),2),'+/-',np.around
(np.std(test_scores),2))
```

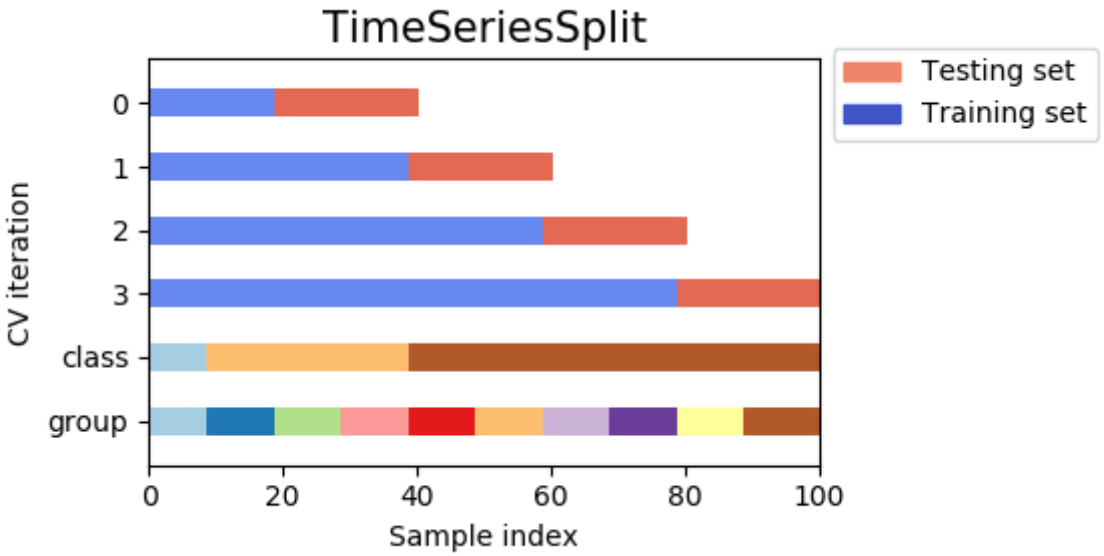
```
{'svc__C': 10.0, 'svc__gamma': 0.001}
best CV score: 0.8143459915611815
test score: 0.6410256410256411
{'svc__C': 10000.0, 'svc__gamma': 0.001}
best CV score: 0.6455696202531646
test score: 0.5847457627118644
{'svc__C': 10.0, 'svc__gamma': 0.001}
best CV score: 0.6494845360824743
test score: 0.9390243902439024
{'svc__C': 10.0, 'svc__gamma': 0.001}
best CV score: 0.7907949790794979
test score: 0.43243243243243246
{'svc__C': 10000.0, 'svc__gamma': 0.001}
best CV score: 0.6756756756756757
test score: 0.8901098901098901
test accuracy: 0.7 +/- 0.19
```

The takeaway

- an incorrect cross validation pipeline gives misleading results
  - usually the model appears to be pretty accurate
  - but the performance is poor when the model is deployed
- this can be avoided by a careful cross validation pipeline
  - think about how your model will be used
  - mimic that future use in CV

Data leakage in time series data is similar!

- do NOT use information in CV which will not be available once your model is deployed
  - don't use future information!



Now you can

- use GridSearchCV with pipelines
- apply stratified splits to imbalanced data
- split based on group ID and time**

In [ ]: