As you can see, for each split, each group is either entirely in the training set or entirely in the test set:

In[16]:

```
mglearn.plots.plot_label_kfold()
```

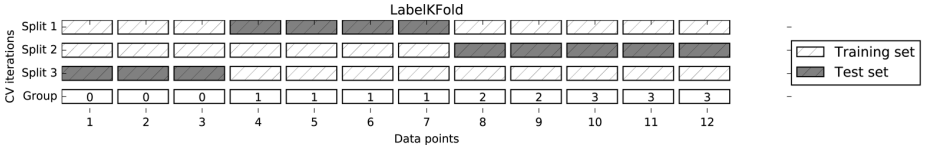

Figure 5-4. Label-dependent splitting with GroupKFold

There are more splitting strategies for cross-validation in `scikit-learn`, which allow for an even greater variety of use cases (you can find these in the `scikit-learn` user guide). However, the standard `KFold`, `StratifiedKFold`, and `GroupKFold` are by far the most commonly used ones.

Grid Search

Now that we know how to evaluate how well a model generalizes, we can take the next step and improve the model's generalization performance by tuning its parameters. We discussed the parameter settings of many of the algorithms in `scikit-learn` in Chapters 2 and 3, and it is important to understand what the parameters mean before trying to adjust them. Finding the values of the important parameters of a model (the ones that provide the best generalization performance) is a tricky task, but necessary for almost all models and datasets. Because it is such a common task, there are standard methods in `scikit-learn` to help you with it. The most commonly used method is *grid search*, which basically means trying all possible combinations of the parameters of interest.

Consider the case of a kernel SVM with an RBF (radial basis function) kernel, as implemented in the `SVC` class. As we discussed in Chapter 2, there are two important parameters: the kernel bandwidth, `gamma`, and the regularization parameter, `C`. Say we want to try the values `0.001`, `0.01`, `0.1`, `1`, `10`, and `100` for the parameter `C`, and the same for `gamma`. Because we have six different settings for `C` and `gamma` that we want to try, we have 36 combinations of parameters in total. Looking at all possible combinations creates a table (or grid) of parameter settings for the SVM, as shown here:

	C = 0.001	C = 0.01	...	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	...	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	...	SVC(C=10, gamma=0.01)
...	...	...	...	...
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	...	SVC(C=10, gamma=100)

Simple Grid Search

We can implement a simple grid search just as `for` loops over the two parameters, training and evaluating a classifier for each combination:

In[18]:

```
# naive grid search implementation
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
print("Size of training set: {}   size of test set: {}".format(
      X_train.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

Out[18]:

```
Size of training set: 112   size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

The Danger of Overfitting the Parameters and the Validation Set

Given this result, we might be tempted to report that we found a model that performs with 97% accuracy on our dataset. However, this claim could be overly optimistic (or just wrong), for the following reason: we tried many different parameters and