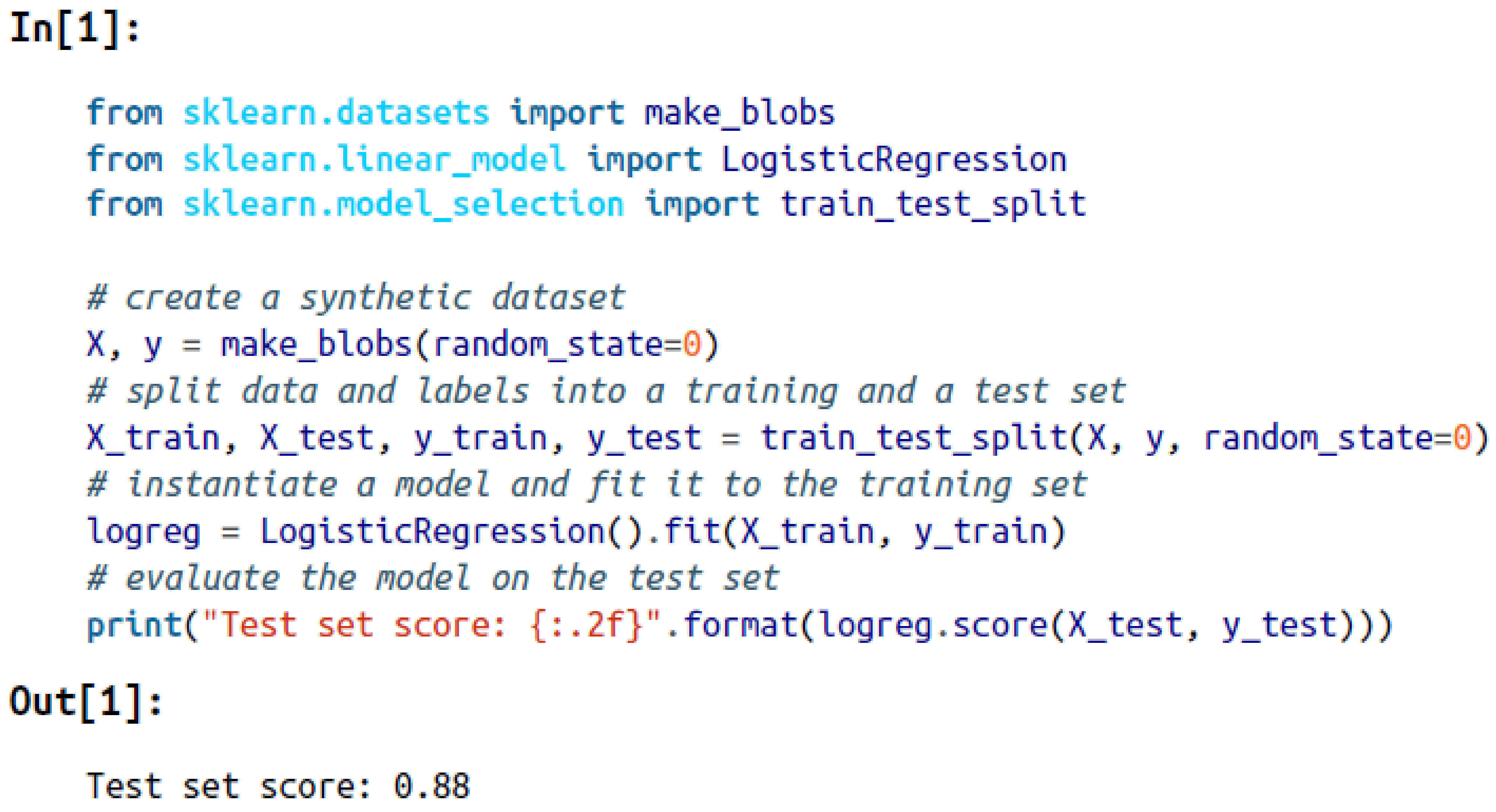
Cross-Validation

APT 3025: APPLIED MACHINE LEARNING

The Basic Machine Learning Process

- To evaluate our supervised models, we have split our dataset into a training set and a test set using the train_test_split function, built a model on the training set by calling the fit method, and evaluated it on the test set using the score method, which for classification computes the fraction of correctly classified samples.
- Here’s an example of that process:



In[1]:

```python
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# create a synthetic dataset
X, y = make_blobs(random_state=0)
# split data and labels into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# instantiate a model and fit it to the training set
logreg = LogisticRegression().fit(X_train, y_train)
# evaluate the model on the test set
print("Test set score: {:.2f}".format(logreg.score(X_test, y_test)))
```

Out[1]:

```
Test set score: 0.88
```

Why we split the data

- Remember, the reason we split our data into training and test sets is that we are interested in measuring how well our model generalizes to new, previously unseen data.
- We are not interested in how well our model fit the training set, but rather in how well it can make predictions for data that was not observed during training.

Better evaluation approaches

- In this lecture, we will expand on two aspects of this evaluation.
- We will first introduce cross-validation, a more robust way to assess generalization performance.
- We will discuss ways to evaluate classification and regression performance better than with the default score method.
- Later we will discuss grid search, an effective method for adjusting the parameters in supervised models for the best generalization performance.

Cross-Validation

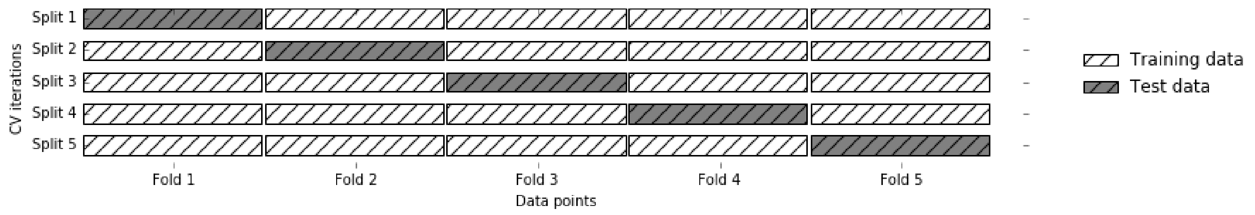
- Cross-validation is a statistical method of evaluating generalization performance that is more stable and thorough than using a split into a training and a test set.
- In cross-validation, the data is instead split repeatedly and multiple models are trained.
- The most commonly used version of cross-validation is k-fold cross-validation, where k is a user-specified number, usually 5 or 10.

Cross-Validation

- When performing five-fold cross-validation, the data is first partitioned into five parts of (approximately) equal size, called folds.
- Next, a sequence of models is trained.
- The first model is trained using 4 folds (folds 2-5) as the training set and the remaining fold (fold 1) as the test set.

Cross-Validation

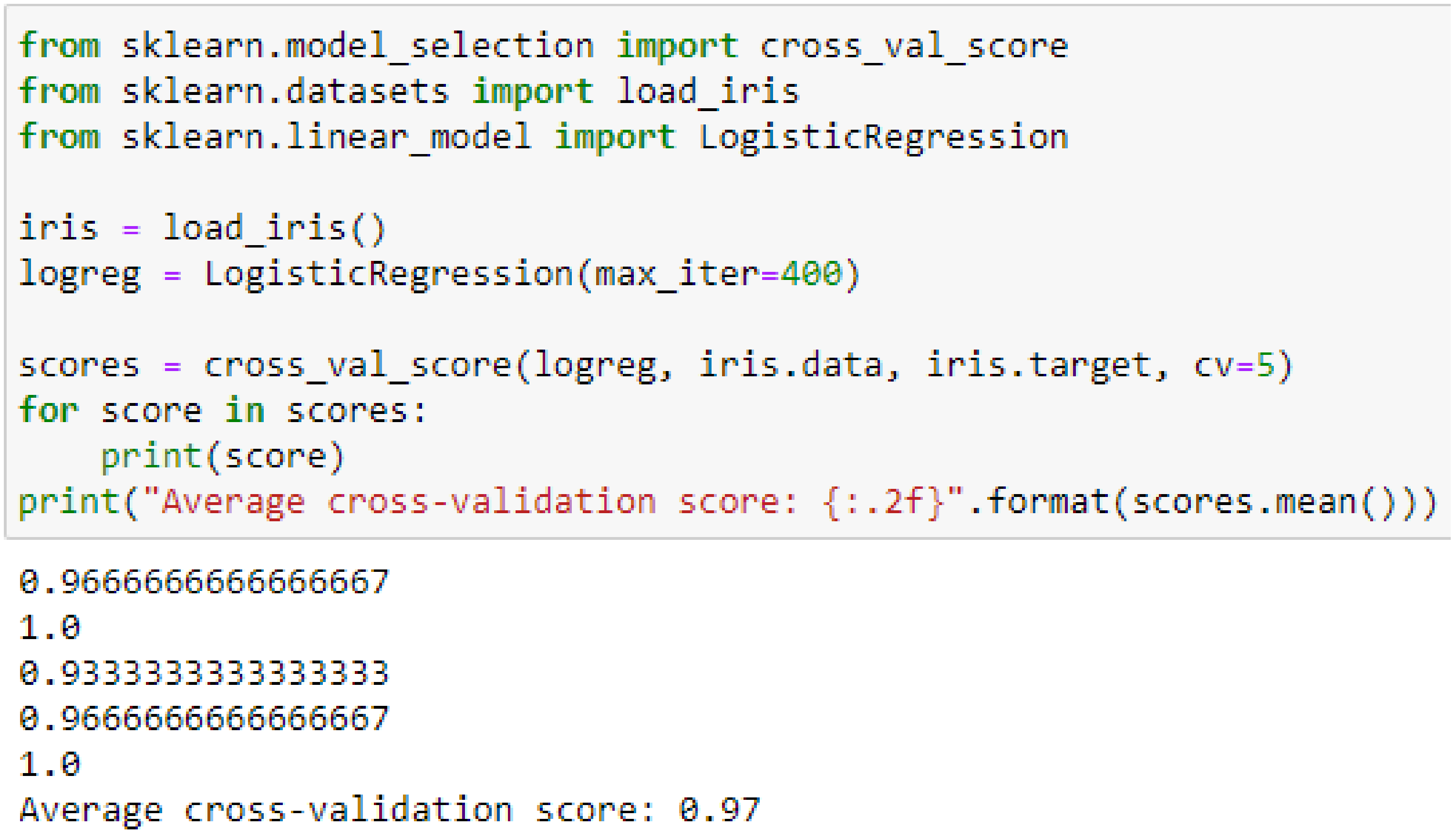
- Then another model is built, this time using fold 2 as the test set and the data in folds 1, 3, 4, and 5 as the training set.
- This process is repeated using folds 3, 4, and 5 as test sets.
- For each of these five splits of the data, we compute the accuracy, and in the end, we have collected five accuracy values.



Cross-Validation in Scikit-Learn

- Cross-validation is implemented in scikit-learn using the cross_val_score function from the model_selection module.
- The parameters of the cross_val_score function are the model we want to evaluate, the training data, and the ground-truth labels.
- Let’s evaluate LogisticRegression on the iris dataset:

Cross-Validation



```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression(max_iter=400)

scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
for score in scores:
    print(score)
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

```
0.9666666666666667
1.0
0.9333333333333333
0.9666666666666667
1.0
Average cross-validation score: 0.97
```

Cross-Validation

- Using the mean cross-validation we can conclude that we expect the model to be around 97% accurate on average.
- Looking at all five scores, we can also conclude that there is a relatively high variance in the accuracy between folds, ranging from 93% accuracy to 100% accuracy.
- This could imply that the model is very dependent on the particular folds used for training, but it could also just be a consequence of the small size of the dataset.

Benefits of Cross-Validation

- There are several benefits to using cross-validation instead of a single split into a training and a test set.
- First, remember that train_test_split performs a random split of the data.
- Imagine that we are "lucky" when randomly splitting the data, and all examples that are hard to classify end up in the training set.
- In that case, the test set will only contain "easy" examples, and our test set accuracy will be unrealistically high.

Benefits of Cross-Validation

- Conversely, if we are “unlucky,” we might have randomly put all the hard-to-classify examples in the test set and consequently obtain an unrealistically low score.
- However, when using cross-validation, each example will be in the test set exactly once: each example is in one of the folds, and each fold is the test set once.
- Therefore, the model needs to generalize well to all of the samples in the dataset for all of the cross-validation scores (and their mean) to be high.

Benefits of Cross-Validation

- Having multiple splits of the data also provides some information about how sensitive our model is to the selection of the training dataset.
- For the iris dataset, we saw accuracies between 93% and 100%.
- This is quite a range, and it provides us with an idea about how the model might perform in the worst case and best case scenarios when applied to new data.

Benefits of Cross-Validation

- Another benefit of cross-validation as compared to using a single split of the data is that we use our data more effectively.
- When using train_test_split, we usually use 75% of the data for training and 25% of the data for evaluation.
- When using five-fold cross-validation, in each iteration we can use four-fifths of the data (80%) to fit the model.
- When using 10-fold cross-validation, we can use nine-tenths of the data (90%) to fit the model.
- More training data will usually result in more accurate models.

Disadvantage of Cross-Validation

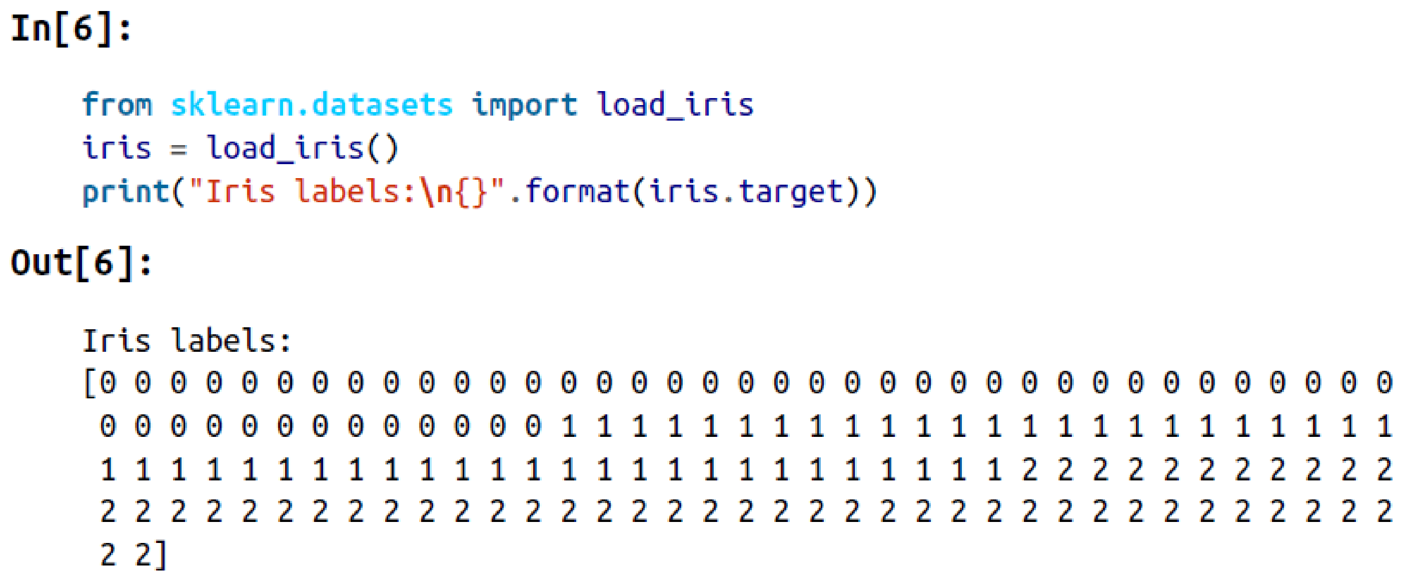
- The main disadvantage of cross-validation is increased computational cost.
- As we are now training k models instead of a single model, cross-validation will be roughly k times slower than doing a single split of the data.

Cross-validation does not return a model

- It is important to keep in mind that cross-validation is not a way to build a model that can be applied to new data.
- Cross-validation does not return a model.
- When calling cross_val_score, multiple models are built internally, but the purpose of cross-validation is only to evaluate how well a given algorithm will generalize when trained on a specific dataset.

Problem with k-Fold Cross-Validation

- Splitting the dataset into k folds by starting with the first 1/k of the data might not always be a good idea.
- For example, let’s have a look at the iris dataset



In[6]:

```python
from sklearn.datasets import load_iris
iris = load_iris()
print("Iris labels:\n{}".format(iris.target))
```

Out[6]:

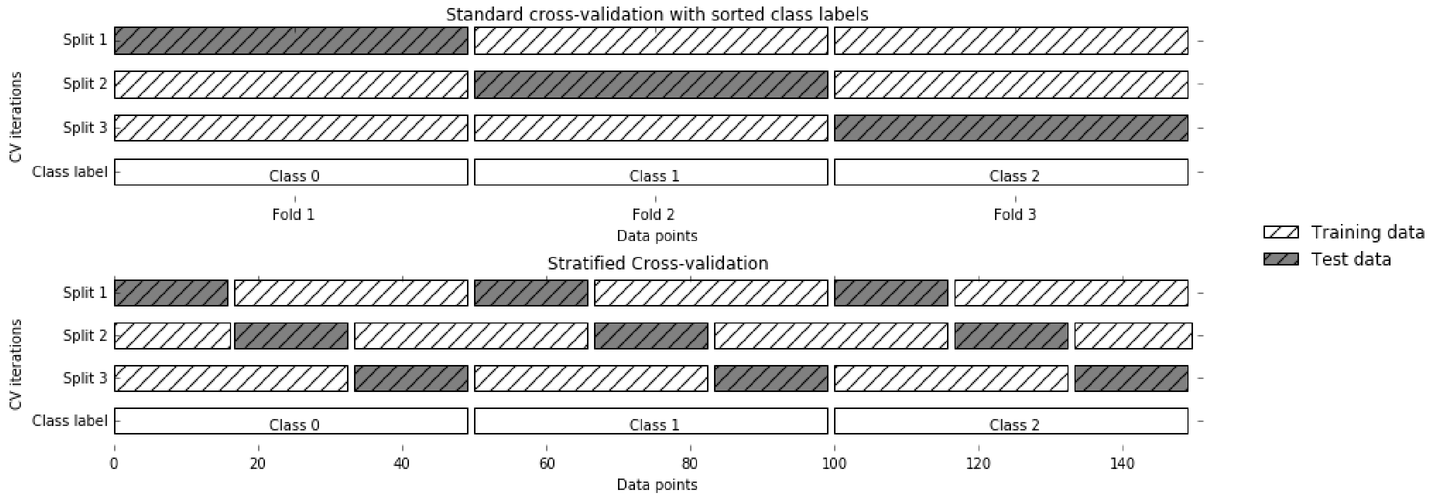
```
Iris labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Problem with k-Fold Cross-Validation

- As you can see, the first third of the data is the class 0, the second third is the class 1, and the last third is the class 2. Imagine doing three-fold cross-validation on this dataset.
- The first fold would be only class 0, so in the first split of the data, the test set would be only class 0, and the training set would be only classes 1 and 2.
- As the classes in training and test sets would be different for all three splits, the three-fold cross-validation accuracy would be zero on this dataset.

Stratified k-Fold Cross-Validation

- As the simple k-fold strategy fails here, scikit-learn does not use it for classification, but rather uses stratified k-fold cross-validation.
- In stratified cross-validation, we split the data such that the proportions between classes are the same in each fold as they are in the whole dataset.



Stratified k-fold cross-validation is recommended

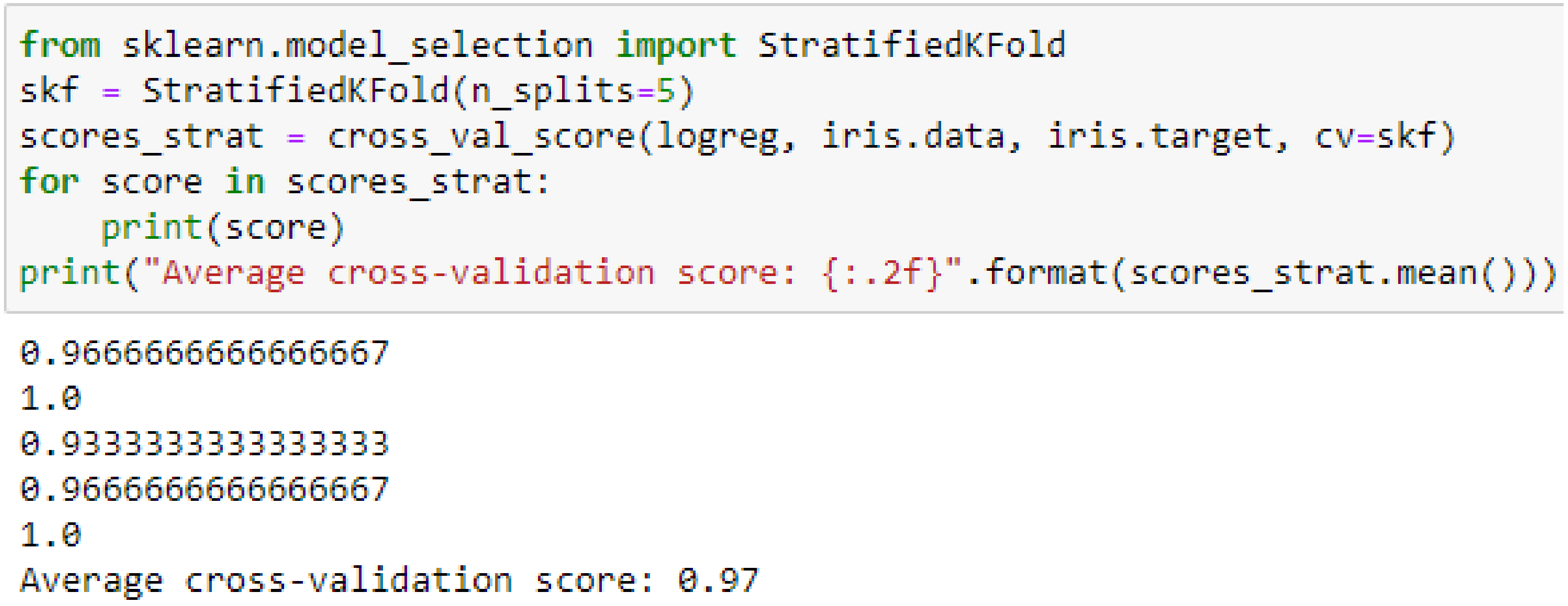
- It is usually a good idea to use stratified k-fold cross-validation instead of k-fold cross-validation to evaluate a classifier, because it results in more reliable estimates of generalization performance.
- In the case of 90% of samples belonging to class A and only 10% belonging to class B, using standard k-fold runs the risk of one fold only containing samples of class A.
- Using this fold as a test set would not be very informative about the overall performance of the classifier.

Classification uses stratified cross-validation

- As the simple k-fold strategy fails for the iris dataset, scikit-learn does not use it for classification, but rather uses stratified k-fold cross-validation be default.
- For regression tasks, however, Scikit-Learn uses the standard k-fold cross-validation by default.

Classification uses stratified cross-validation

- We can use stratified k-fold cross validation explicitly. For classification, the results are the same as before as expected.



```python
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5)
scores_strat = cross_val_score(logreg, iris.data, iris.target, cv=skf)
for score in scores_strat:
    print(score)
print("Average cross-validation score: {:.2f}".format(scores_strat.mean()))
```

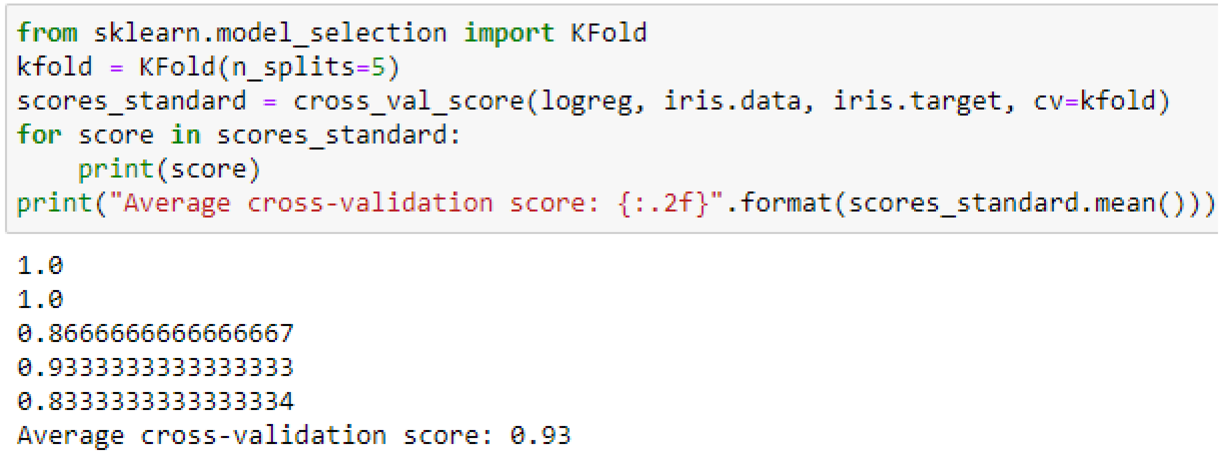
```
0.9666666666666667
1.0
0.9333333333333333
0.9666666666666667
1.0
Average cross-validation score: 0.97
```

Changing the Defaults

- For most use cases, the defaults of k-fold cross-validation for regression and stratified k-fold for classification work well, but there are some cases where you might want to use a different strategy.
- Say, for example, we want to use the standard k-fold cross-validation on a classification dataset to reproduce someone else’s results.

Using Standard K-Fold Cross-Validation in Classification

- To do this, we first have to import the KFold splitter class from the model_selection module and instantiate it with the number of folds we want to use.
- Then, we can pass the kfold splitter object as the cv parameter to cross_val_score.

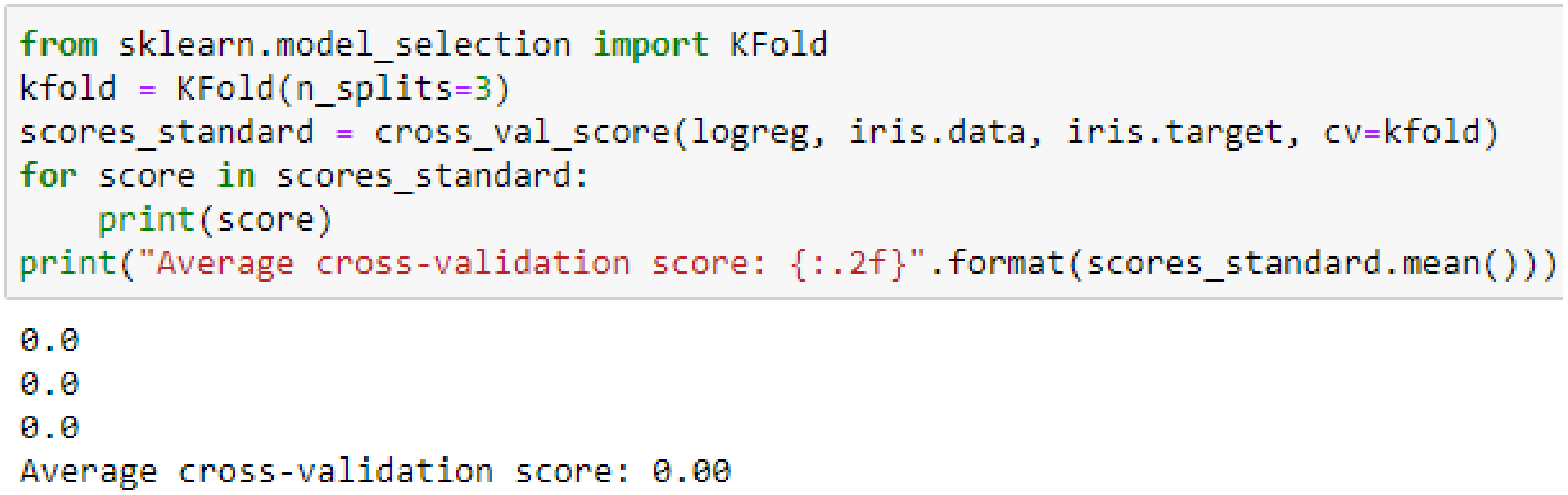


```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5)
scores_standard = cross_val_score(logreg, iris.data, iris.target, cv=kfold)
for score in scores_standard:
    print(score)
print("Average cross-validation score: {:.2f}".format(scores_standard.mean()))
```

```
1.0
1.0
0.8666666666666667
0.9333333333333333
0.8333333333333334
Average cross-validation score: 0.93
```

Using Standard K-Fold Cross-Validation in Classification

- We already see a significant performance drop, the lowest going from the previous 93% to now 83%.
- It is much worse if we use 3 folds:



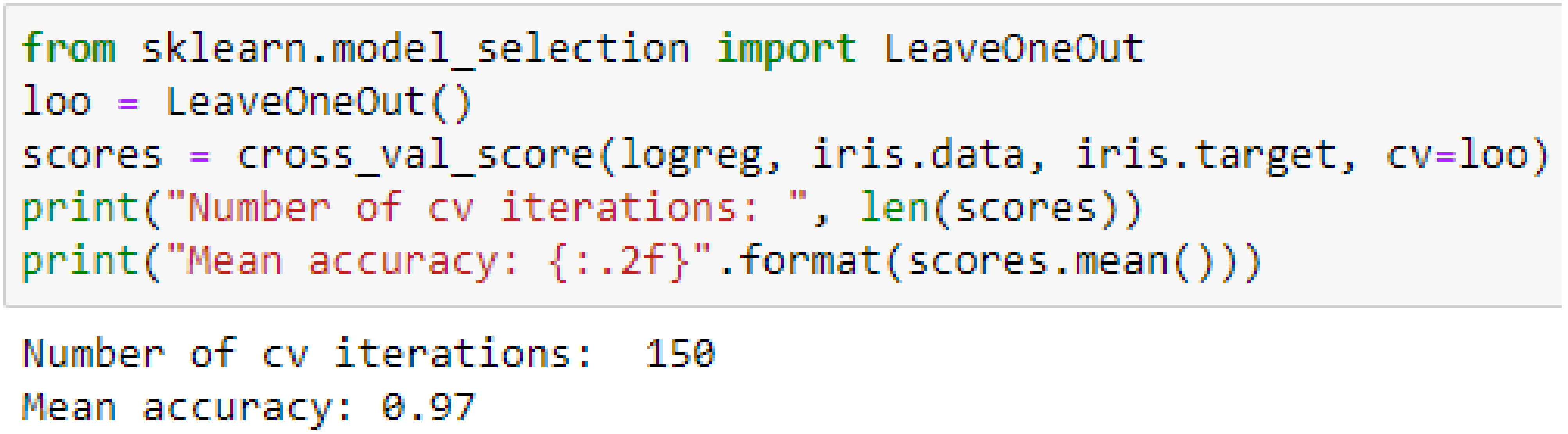
```python
from sklearn.model_selection import KFold
kfold = KFold(n_splits=3)
scores_standard = cross_val_score(logreg, iris.data, iris.target, cv=kfold)
for score in scores_standard:
    print(score)
print("Average cross-validation score: {:.2f}".format(scores_standard.mean()))
```

```
0.0
0.0
0.0
Average cross-validation score: 0.00
```

Leave-One-Out Cross-Validation

- Another frequently used cross-validation method is leave-one-out.
- You can think of leave-one-out cross-validation as k-fold cross-validation where each fold is a single sample.
- For each split, you pick a single data point to be the test set.
- This can be very time consuming, particularly for large datasets, but sometimes provides better estimates on small datasets

Leave-One-Out Cross-Validation

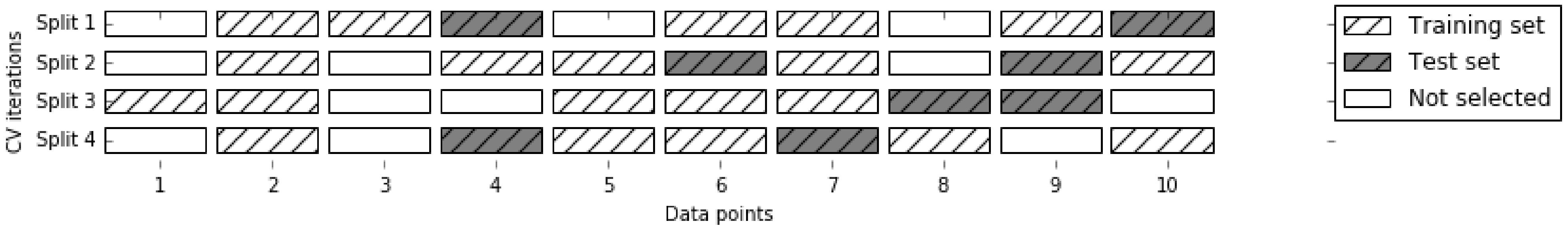


```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
print("Number of cv iterations: ", len(scores))
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

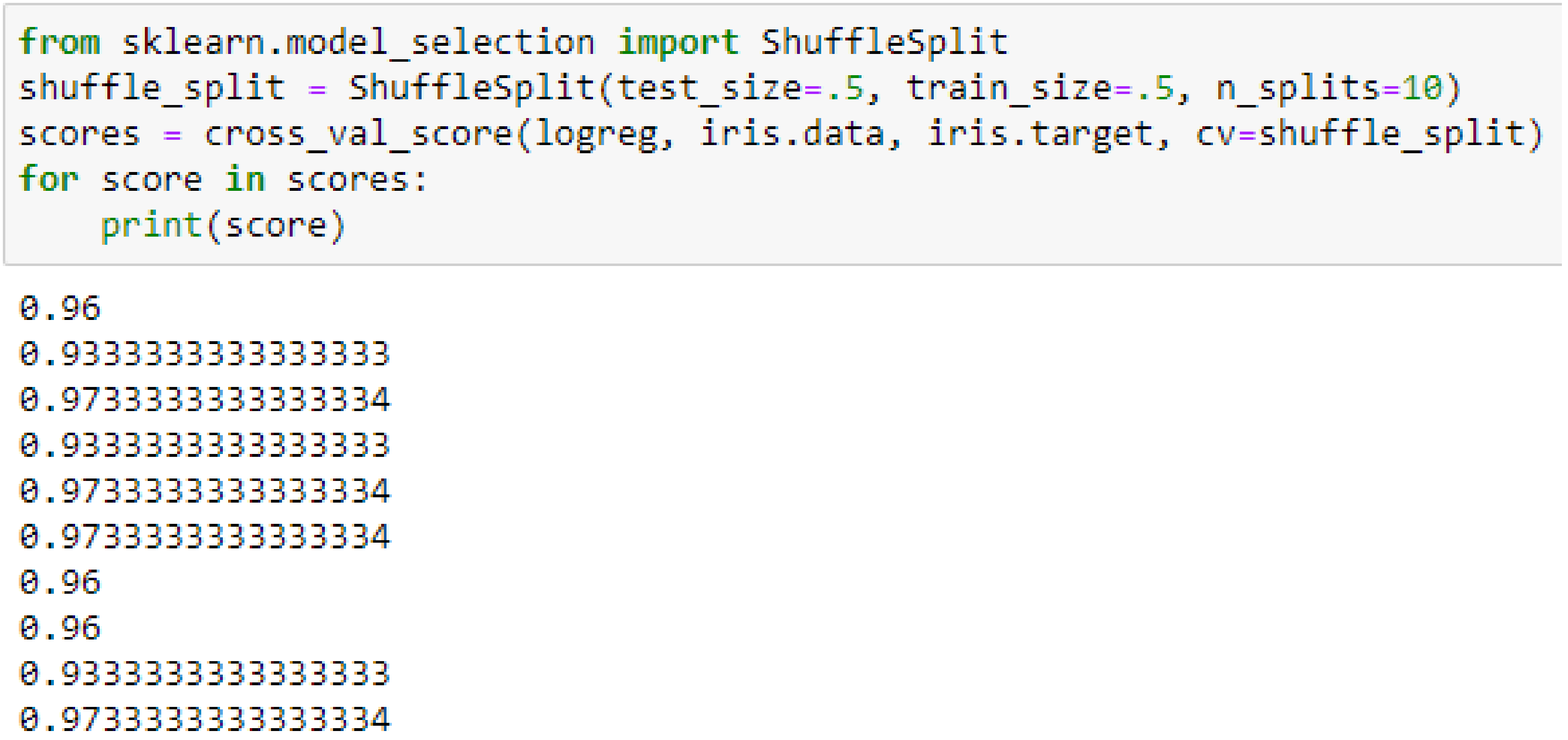
```
Number of cv iterations:  150
Mean accuracy: 0.97
```

Shuffle-split cross-validation

- Another, very flexible strategy for cross-validation is shuffle-split cross-validation. In shuffle-split cross-validation, each split samples train_size points for the training set and test_size (disjoint) points for the test set. This splitting is repeated n_splits times.



Shuffle-split cross-validation



```
from sklearn.model_selection import ShuffleSplit
shuffle_split = ShuffleSplit(test_size=.5, train_size=.5, n_splits=10)
scores = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
for score in scores:
    print(score)
```

```
0.96
0.9333333333333333
0.9733333333333334
0.9333333333333333
0.9733333333333334
0.9733333333333334
0.96
0.96
0.9333333333333333
0.9733333333333334
```

Shuffle-split cross-validation

- Shuffle-split cross-validation allows for control over the number of iterations independently of the training and test sizes, which can sometimes be helpful.
- It also allows for using only part of the data in each iteration, by providing train_size and test_size settings that don’t add up to one. Subsampling the data in this way can be useful for experimenting with large datasets.
- There is also a stratified variant of ShuffleSplit, aptly named StratifiedShuffleSplit, which can provide more reliable results for classification tasks.

Cross-Validation with Groups

- Another very common setting for cross-validation is when there are groups in the data that are highly related.
- Say you want to build a system to recognize emotions from pictures of faces, and you collect a dataset of pictures of 100 people where each person is captured multiple times, showing various emotions.
- The goal is to build a classifier that can correctly identify emotions of people not in the dataset.

Cross-Validation with Groups

- You could use the default stratified cross-validation to measure the performance of a classifier here.
- However, it is likely that pictures of the same person will be in both the training and the test set.
- It will be much easier for a classifier to detect emotions in a face that is part of the training set, compared to a completely new face.
- To accurately evaluate the generalization to new faces, we must therefore ensure that the training and test sets contain images of different people.

Cross-Validation with Groups

- To achieve this, we can use GroupKFold, which takes an array of groups as argument that we can use to indicate which person is in the image.
- The groups array here indicates groups in the data that should not be split when creating the training and test sets.

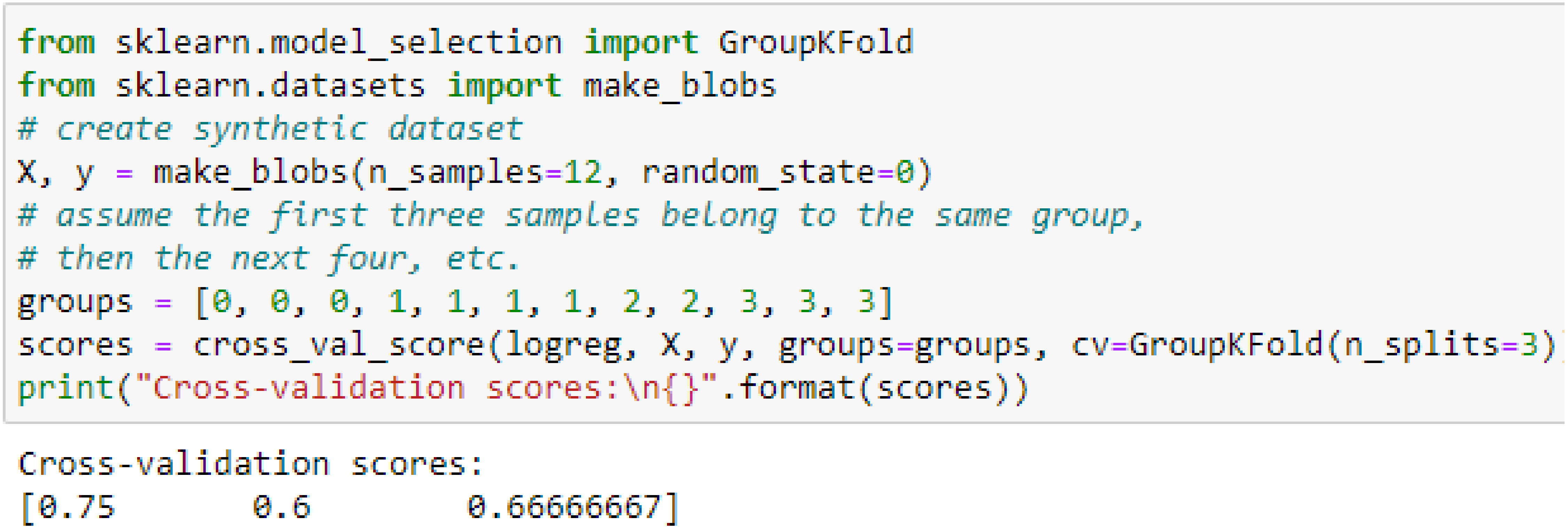
Cross-Validation with Groups

- This example of groups in the data is common in medical applications, where you might have multiple samples from the same patient, but are interested in generalizing to new patients.
- Similarly, in speech recognition, you might have multiple recordings of the same speaker in your dataset, but are interested in recognizing speech of new speakers.

Cross-Validation with Groups

- The following is an example of using a synthetic dataset with a grouping given by the groups array.
- The dataset consists of 12 data points, and for each of the data points, groups specifies which group (think patient) the point belongs to.
- The groups specify that there are four groups, and the first three samples belong to the first group, the next four samples belong to the second group, and so on.

Cross-Validation with Groups



```python
from sklearn.model_selection import GroupKFold
from sklearn.datasets import make_blobs
# create synthetic dataset
X, y = make_blobs(n_samples=12, random_state=0)
# assume the first three samples belong to the same group,
# then the next four, etc.
groups = [0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3]
scores = cross_val_score(logreg, X, y, groups=groups, cv=GroupKFold(n_splits=3)
print("Cross-validation scores:\n{}".format(scores))
```

```
Cross-validation scores:
[0.75       0.6        0.66666667]
```

Reference

- *Introduction to Machine Learning with Python*: A Guide for Data Scientists by Andreas C. Müller & Sarah Guido, O'Reilly (2017).