**In[70]:**

```
from sklearn.metrics.scorer import SCORERS
print("Available scorers:\n{}".format(sorted(SCORERS.keys())))
```

**Out[70]:**

```
Available scorers:
['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'f1_macro',
 'f1_micro', 'f1_samples', 'f1_weighted', 'log_loss', 'mean_absolute_error',
 'mean_squared_error', 'median_absolute_error', 'precision', 'precision_macro',
 'precision_micro', 'precision_samples', 'precision_weighted', 'r2', 'recall',
 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted', 'roc_auc']
```

## Summary and Outlook

In this chapter we discussed cross-validation, grid search, and evaluation metrics, the cornerstones of evaluating and improving machine learning algorithms. The tools described in this chapter, together with the algorithms described in Chapters 2 and 3, are the bread and butter of every machine learning practitioner.

There are two particular points that we made in this chapter that warrant repeating, because they are often overlooked by new practitioners. The first has to do with cross-validation. Cross-validation or the use of a test set allow us to evaluate a machine learning model as it will perform in the future. However, if we use the test set or cross-validation to select a model or select model parameters, we "use up" the test data, and using the same data to evaluate how well our model will do in the future will lead to overly optimistic estimates. We therefore need to resort to a split into training data for model building, validation data for model and parameter selection, and test data for model evaluation. Instead of a simple split, we can replace each of these splits with cross-validation. The most commonly used form (as described earlier) is a training/test split for evaluation, and using cross-validation on the training set for model and parameter selection.

The second point has to do with the importance of the evaluation metric or scoring function used for model selection and model evaluation. The theory of how to make business decisions from the predictions of a machine learning model is somewhat beyond the scope of this book.[7] However, it is rarely the case that the end goal of a machine learning task is building a model with a high accuracy. Make sure that the metric you choose to evaluate and select a model for is a good stand-in for what the model will actually be used for. In reality, classification problems rarely have balanced classes, and often false positives and false negatives have very different consequences.

7 We highly recommend Foster Provost and Tom Fawcett's book *Data Science for Business* (O'Reilly) for more information on this topic.

Make sure you understand what these consequences are, and pick an evaluation metric accordingly.

The model evaluation and selection techniques we have described so far are the most important tools in a data scientist's toolbox. Grid search and cross-validation as we've described them in this chapter can only be applied to a single supervised model. We have seen before, however, that many models require preprocessing, and that in some applications, like the face recognition example in Chapter 3, extracting a different representation of the data can be useful. In the next chapter, we will introduce the `Pipeline` class, which allows us to use grid search and cross-validation on these complex chains of algorithms.

CHAPTER 6

# Algorithm Chains and Pipelines

For many machine learning algorithms, the particular representation of the data that you provide is very important, as we discussed in Chapter 4. This starts with scaling the data and combining features by hand and goes all the way to learning features using unsupervised machine learning, as we saw in Chapter 3. Consequently, most machine learning applications require not only the application of a single algorithm, but the chaining together of many different processing steps and machine learning models. In this chapter, we will cover how to use the `Pipeline` class to simplify the process of building chains of transformations and models. In particular, we will see how we can combine `Pipeline` and `GridSearchCV` to search over parameters for all processing steps at once.

As an example of the importance of chaining models, we noticed that we can greatly improve the performance of a kernel SVM on the `cancer` dataset by using the `MinMaxScaler` for preprocessing. Here's code for splitting the data, computing the minimum and maximum, scaling the data, and training the SVM:

**In[1]:**

```
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# load and split the data
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

# compute minimum and maximum on the training data
scaler = MinMaxScaler().fit(X_train)
```