CHAPTER 5

Model Evaluation and Improvement

Having discussed the fundamentals of supervised and unsupervised learning, and having explored a variety of machine learning algorithms, we will now dive more deeply into evaluating models and selecting parameters.

We will focus on the supervised methods, regression and classification, as evaluating and selecting models in unsupervised learning is often a very qualitative process (as we saw in Chapter 3).

To evaluate our supervised models, so far 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[2]:

```
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[2]:

```
Test set score: 0.88
```

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.

In this chapter, we will expand on two aspects of this evaluation. We will first introduce cross-validation, a more robust way to assess generalization performance, and discuss methods to evaluate classification and regression performance that go beyond the default measures of accuracy and R^2 provided by the `score` method.

We will also 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. 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 the first fold as the test set, and the remaining folds (2–5) are used as the training set. The model is built using the data in folds 2–5, and then the accuracy is evaluated on fold 1. 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 into training and test sets, we compute the accuracy. In the end, we have collected five accuracy values. The process is illustrated in Figure 5-1:

`In[3]:`

```
mglearn.plots.plot_cross_validation()
```

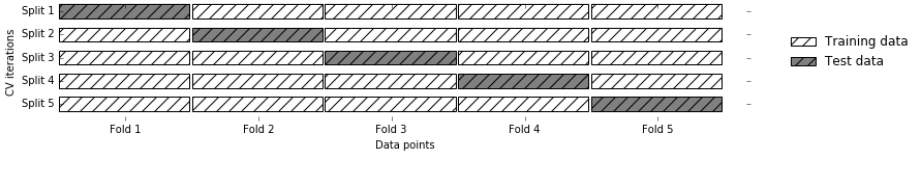

Figure 5-1. Data splitting in five-fold cross-validation

Usually, the first fifth of the data is the first fold, the second fifth of the data is the second fold, and so on.