And for good measure, plot the confusion matrix (Figure 5-79):

```
In[16]: from sklearn.metrics import confusion_matrix
        mat = confusion_matrix(ytest, ypred)
        sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)
        plt.xlabel('true label')
        plt.ylabel('predicted label');
```

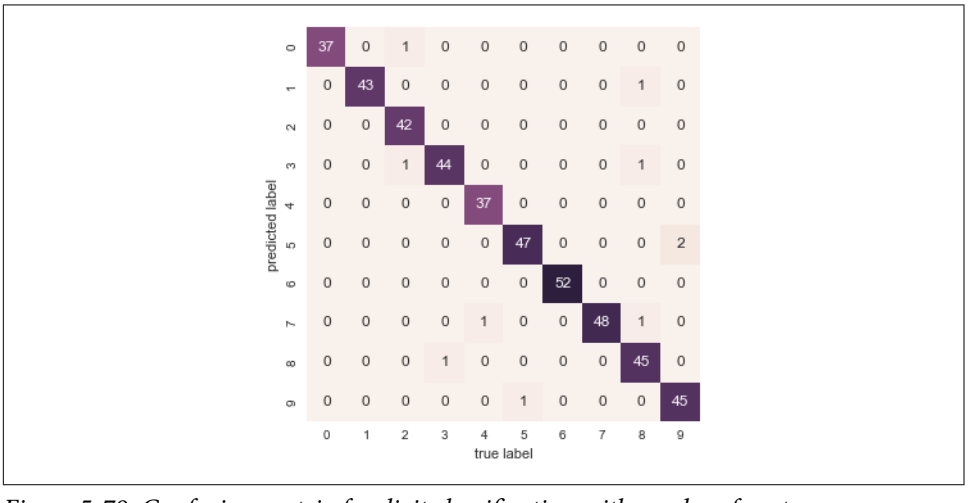

Figure 5-79. Confusion matrix for digit classification with random forests

We find that a simple, untuned random forest results in a very accurate classification of the digits data.

Summary of Random Forests

This section contained a brief introduction to the concept of *ensemble estimators*, and in particular the random forest model—an ensemble of randomized decision trees. Random forests are a powerful method with several advantages:

- Both training and prediction are very fast, because of the simplicity of the underlying decision trees. In addition, both tasks can be straightforwardly parallelized, because the individual trees are entirely independent entities.
- The multiple trees allow for a probabilistic classification: a majority vote among estimators gives an estimate of the probability (accessed in Scikit-Learn with the `predict_proba()` method).
- The nonparametric model is extremely flexible, and can thus perform well on tasks that are underfit by other estimators.

A primary disadvantage of random forests is that the results are not easily interpretable; that is, if you would like to draw conclusions about the *meaning* of the classification model, random forests may not be the best choice.

In Depth: Principal Component Analysis

Up until now, we have been looking in depth at supervised learning estimators: those estimators that predict labels based on labeled training data. Here we begin looking at several unsupervised estimators, which can highlight interesting aspects of the data without reference to any known labels.

In this section, we explore what is perhaps one of the most broadly used of unsupervised algorithms, principal component analysis (PCA). PCA is fundamentally a dimensionality reduction algorithm, but it can also be useful as a tool for visualization, for noise filtering, for feature extraction and engineering, and much more. After a brief conceptual discussion of the PCA algorithm, we will see a couple examples of these further applications. We begin with the standard imports:

```
In[1]: %matplotlib inline
       import numpy as np
       import matplotlib.pyplot as plt
       import seaborn as sns; sns.set()
```

Introducing Principal Component Analysis

Principal component analysis is a fast and flexible unsupervised method for dimensionality reduction in data, which we saw briefly in "Introducing Scikit-Learn" on page 343. Its behavior is easiest to visualize by looking at a two-dimensional dataset. Consider the following 200 points (Figure 5-80):

```
In[2]: rng = np.random.RandomState(1)
       X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
       plt.scatter(X[:, 0], X[:, 1])
       plt.axis('equal');
```

By eye, it is clear that there is a nearly linear relationship between the x and y variables. This is reminiscent of the linear regression data we explored in "In Depth: Linear Regression" on page 390, but the problem setting here is slightly different: rather than attempting to *predict* the *y* values from the *x* values, the unsupervised learning problem attempts to learn about the *relationship* between the *x* and *y* values.