Finally, if you want some practice building your own estimator, you might tackle building a similar Bayesian classifier using Gaussian mixture models instead of KDE.

Application: A Face Detection Pipeline

This chapter has explored a number of the central concepts and algorithms of machine learning. But moving from these concepts to real-world application can be a challenge. Real-world datasets are noisy and heterogeneous, may have missing features, and may include data in a form that is difficult to map to a clean `[n_samples, n_features]` matrix. Before applying any of the methods discussed here, you must first extract these features from your data; there is no formula for how to do this that applies across all domains, and thus this is where you as a data scientist must exercise your own intuition and expertise.

One interesting and compelling application of machine learning is to images, and we have already seen a few examples of this where pixel-level features are used for classification. In the real world, data is rarely so uniform and simple pixels will not be suitable, a fact that has led to a large literature on *feature extraction* methods for image data (see "Feature Engineering" on page 375).

In this section, we will take a look at one such feature extraction technique, the Histogram of Oriented Gradients (HOG), which transforms image pixels into a vector representation that is sensitive to broadly informative image features regardless of confounding factors like illumination. We will use these features to develop a simple face detection pipeline, using machine learning algorithms and concepts we've seen throughout this chapter. We begin with the standard imports:

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

HOG Features

The Histogram of Gradients is a straightforward feature extraction procedure that was developed in the context of identifying pedestrians within images. HOG involves the following steps:

1. Optionally prenormalize images. This leads to features that resist dependence on variations in illumination.
2. Convolve the image with two filters that are sensitive to horizontal and vertical brightness gradients. These capture edge, contour, and texture information.
3. Subdivide the image into cells of a predetermined size, and compute a histogram of the gradient orientations within each cell.

4. Normalize the histograms in each cell by comparing to the block of neighboring cells. This further suppresses the effect of illumination across the image.
5. Construct a one-dimensional feature vector from the information in each cell.

A fast HOG extractor is built into the Scikit-Image project, and we can try it out relatively quickly and visualize the oriented gradients within each cell (Figure 5-149):

```
In[2]: from skimage import data, color, feature
       import skimage.data

       image = color.rgb2gray(data.chelsea())
       hog_vec, hog_vis = feature.hog(image, visualise=True)

       fig, ax = plt.subplots(1, 2, figsize=(12, 6),
                              subplot_kw=dict(xticks=[], yticks=[]))
       ax[0].imshow(image, cmap='gray')
       ax[0].set_title('input image')

       ax[1].imshow(hog_vis)
       ax[1].set_title('visualization of HOG features');
```

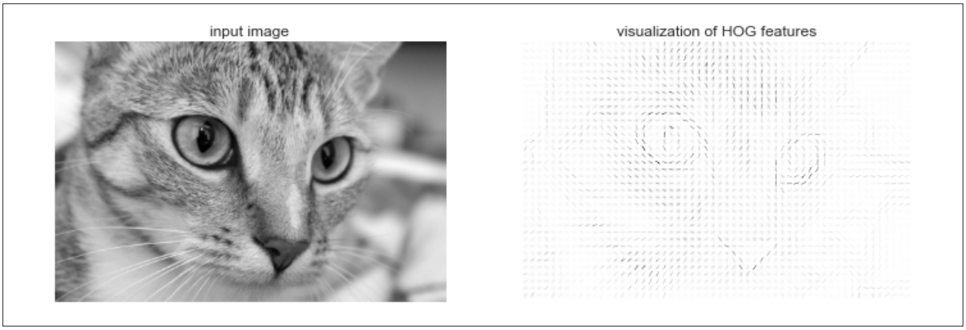

Figure 5-149. Visualization of HOG features computed from an image

HOG in Action: A Simple Face Detector

Using these HOG features, we can build up a simple facial detection algorithm with any Scikit-Learn estimator; here we will use a linear support vector machine (refer back to "In-Depth: Support Vector Machines" on page 405 if you need a refresher on this). The steps are as follows:

1. Obtain a set of image thumbnails of faces to constitute "positive" training samples.
2. Obtain a set of image thumbnails of nonfaces to constitute "negative" training samples.
3. Extract HOG features from these training samples.