There is one clear disadvantage of this approach: if your category has many possible values, this can *greatly* increase the size of your dataset. However, because the encoded data contains mostly zeros, a sparse output can be a very efficient solution:

```
In[5]: vec = DictVectorizer(sparse=True, dtype=int)
       vec.fit_transform(data)
```

```
Out[5]: <4x5 sparse matrix of type '<class 'numpy.int64'>'
            with 12 stored elements in Compressed Sparse Row format>
```

Many (though not yet all) of the Scikit-Learn estimators accept such sparse inputs when fitting and evaluating models. `sklearn.preprocessing.OneHotEncoder` and `sklearn.feature_extraction.FeatureHasher` are two additional tools that Scikit-Learn includes to support this type of encoding.

Text Features

Another common need in feature engineering is to convert text to a set of representative numerical values. For example, most automatic mining of social media data relies on some form of encoding the text as numbers. One of the simplest methods of encoding data is by *word counts*: you take each snippet of text, count the occurrences of each word within it, and put the results in a table.

For example, consider the following set of three phrases:

```
In[6]: sample = ['problem of evil',
                 'evil queen',
                 'horizon problem']
```

For a vectorization of this data based on word count, we could construct a column representing the word "problem," the word "evil," the word "horizon," and so on. While doing this by hand would be possible, we can avoid the tedium by using Scikit-Learn's `CountVectorizer`:

```
In[7]: from sklearn.feature_extraction.text import CountVectorizer

       vec = CountVectorizer()
       X = vec.fit_transform(sample)
       X
```

```
Out[7]: <3x5 sparse matrix of type '<class 'numpy.int64'>'
            with 7 stored elements in Compressed Sparse Row format>
```

The result is a sparse matrix recording the number of times each word appears; it is easier to inspect if we convert this to a `DataFrame` with labeled columns:

```
In[8]: import pandas as pd
       pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

```
Out[8]:    evil  horizon  of  problem  queen
        0     1        0   1        1      0
        1     1        0   0        0      1
        2     0        1   0        1      0
```

There are some issues with this approach, however: the raw word counts lead to features that put too much weight on words that appear very frequently, and this can be suboptimal in some classification algorithms. One approach to fix this is known as *term frequency–inverse document frequency* (*TF–IDF*), which weights the word counts by a measure of how often they appear in the documents. The syntax for computing these features is similar to the previous example:

```
In[9]: from sklearn.feature_extraction.text import TfidfVectorizer
       vec = TfidfVectorizer()
       X = vec.fit_transform(sample)
       pd.DataFrame(X.toarray(), columns=vec.get_feature_names())
```

```
Out[9]:        evil   horizon        of   problem     queen
        0  0.517856  0.000000  0.680919  0.517856  0.000000
        1  0.605349  0.000000  0.000000  0.000000  0.795961
        2  0.000000  0.795961  0.000000  0.605349  0.000000
```

For an example of using TF–IDF in a classification problem, see "In Depth: Naive Bayes Classification" on page 382.

Image Features

Another common need is to suitably encode *images* for machine learning analysis. The simplest approach is what we used for the digits data in "Introducing Scikit-Learn" on page 343: simply using the pixel values themselves. But depending on the application, such approaches may not be optimal.

A comprehensive summary of feature extraction techniques for images is well beyond the scope of this section, but you can find excellent implementations of many of the standard approaches in the Scikit-Image project. For one example of using Scikit-Learn and Scikit-Image together, see "Application: A Face Detection Pipeline" on page 506.

Derived Features

Another useful type of feature is one that is mathematically derived from some input features. We saw an example of this in "Hyperparameters and Model Validation" on page 359 when we constructed *polynomial features* from our input data. We saw that we could convert a linear regression into a polynomial regression not by changing the model, but by transforming the input! This is sometimes known as *basis function regression*, and is explored further in "In Depth: Linear Regression" on page 390.