Applying Bag-of-Words to a Toy Dataset

The bag-of-words representation is implemented in `CountVectorizer`, which is a transformer. Let's first apply it to a toy dataset, consisting of two samples, to see it working:

In[7]:

```
bards_words =["The fool doth think he is wise,",
              "but the wise man knows himself to be a fool"]
```

We import and instantiate the `CountVectorizer` and `fit` it to our toy data as follows:

In[8]:

```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
vect.fit(bards_words)
```

Fitting the `CountVectorizer` consists of the tokenization of the training data and building of the vocabulary, which we can access as the `vocabulary_` attribute:

In[9]:

```
print("Vocabulary size: {}".format(len(vect.vocabulary_)))
print("Vocabulary content:\n {}".format(vect.vocabulary_))
```

Out[9]:

```
Vocabulary size: 13
Vocabulary content:
 {'the': 9, 'himself': 5, 'wise': 12, 'he': 4, 'doth': 2, 'to': 11, 'knows': 7,
  'man': 8, 'fool': 3, 'is': 6, 'be': 0, 'think': 10, 'but': 1}
```

The vocabulary consists of 13 words, from `"be"` to `"wise"`.

To create the bag-of-words representation for the training data, we call the `transform` method:

In[10]:

```
bag_of_words = vect.transform(bards_words)
print("bag_of_words: {}".format(repr(bag_of_words)))
```

Out[10]:

```
bag_of_words: <2x13 sparse matrix of type '<class 'numpy.int64'>'
    with 16 stored elements in Compressed Sparse Row format>
```

The bag-of-words representation is stored in a SciPy sparse matrix that only stores the entries that are nonzero (see Chapter 1). The matrix is of shape 2×13, with one row for each of the two data points and one feature for each of the words in the vocabulary. A sparse matrix is used as most documents only contain a small subset of the words in the vocabulary, meaning most entries in the feature array are 0. Think

about how many different words might appear in a movie review compared to all the words in the English language (which is what the vocabulary models). Storing all those zeros would be prohibitive, and a waste of memory. To look at the actual content of the sparse matrix, we can convert it to a "dense" NumPy array (that also stores all the 0 entries) using the `toarray` method:[4]

In[11]:

```
print("Dense representation of bag_of_words:\n{}".format(
    bag_of_words.toarray()))
```

Out[11]:

```
Dense representation of bag_of_words:
[[0 0 1 1 1 0 1 0 0 1 1 0 1]
 [1 1 0 1 0 1 0 1 1 1 0 1 1]]
```

We can see that the word counts for each word are either 0 or 1; neither of the two strings in `bards_words` contains a word twice. Let's take a look at how to read these feature vectors. The first string (`"The fool doth think he is wise,"`) is represented as the first row in, and it contains the first word in the vocabulary, `"be"`, zero times. It also contains the second word in the vocabulary, `"but"`, zero times. It contains the third word, `"doth"`, once, and so on. Looking at both rows, we can see that the fourth word, `"fool"`, the tenth word, `"the"`, and the thirteenth word, `"wise"`, appear in both strings.

Bag-of-Words for Movie Reviews

Now that we've gone through the bag-of-words process in detail, let's apply it to our task of sentiment analysis for movie reviews. Earlier, we loaded our training and test data from the IMDb reviews into lists of strings (`text_train` and `text_test`), which we will now process:

In[12]:

```
vect = CountVectorizer().fit(text_train)
X_train = vect.transform(text_train)
print("X_train:\n{}".format(repr(X_train)))
```

Out[12]:

```
X_train:
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
    with 3431196 stored elements in Compressed Sparse Row format>
```

4 This is possible because we are using a small toy dataset that contains only 13 words. For any real dataset, this would result in a `MemoryError`.