appearing words, by setting the `max_df` option of `CountVectorizer` and see how it influences the number of features and the performance.

Rescaling the Data with tf–idf

Instead of dropping features that are deemed unimportant, another approach is to rescale features by how informative we expect them to be. One of the most common ways to do this is using the *term frequency–inverse document frequency* (tf–idf) method. The intuition of this method is to give high weight to any term that appears often in a particular document, but not in many documents in the corpus. If a word appears often in a particular document, but not in very many documents, it is likely to be very descriptive of the content of that document. `scikit-learn` implements the tf–idf method in two classes: `TfidfTransformer`, which takes in the sparse matrix output produced by `CountVectorizer` and transforms it, and `TfidfVectorizer`, which takes in the text data and does both the bag-of-words feature extraction and the tf–idf transformation. There are several variants of the tf–idf rescaling scheme, which you can read about on Wikipedia. The tf–idf score for word *w* in document *d* as implemented in both the `TfidfTransformer` and `TfidfVectorizer` classes is given by:[7]

$$\text{tfidf}(w, d) = \text{tf} \log\left(\frac{N+1}{N_w+1}\right) + 1$$

where N is the number of documents in the training set, N_w is the number of documents in the training set that the word w appears in, and *tf* (the term frequency) is the number of times that the word w appears in the query document d (the document you want to transform or encode). Both classes also apply L2 normalization after computing the tf–idf representation; in other words, they rescale the representation of each document to have Euclidean norm 1. Rescaling in this way means that the length of a document (the number of words) does not change the vectorized representation.

Because tf–idf actually makes use of the statistical properties of the training data, we will use a pipeline, as described in Chapter 6, to ensure the results of our grid search are valid. This leads to the following code:

7 We provide this formula here mostly for completeness; you don't need to remember it to use the tf–idf encoding.

In[23]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(TfidfVectorizer(min_df=5, norm=None),
                     LogisticRegression())
param_grid = {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10]}

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(text_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

Out[23]:

```
Best cross-validation score: 0.89
```

As you can see, there is some improvement when using tf–idf instead of just word counts. We can also inspect which words tf–idf found most important. Keep in mind that the tf–idf scaling is meant to find words that distinguish documents, but it is a purely unsupervised technique. So, "important" here does not necessarily relate to the "positive review" and "negative review" labels we are interested in. First, we extract the `TfidfVectorizer` from the pipeline:

In[24]:

```
vectorizer = grid.best_estimator_.named_steps["tfidfvectorizer"]
# transform the training dataset
X_train = vectorizer.transform(text_train)
# find maximum value for each of the features over the dataset
max_value = X_train.max(axis=0).toarray().ravel()
sorted_by_tfidf = max_value.argsort()
# get feature names
feature_names = np.array(vectorizer.get_feature_names())

print("Features with lowest tfidf:\n{}".format(
    feature_names[sorted_by_tfidf[:20]]))

print("Features with highest tfidf: \n{}".format(
    feature_names[sorted_by_tfidf[-20:]]))
```

Out[24]:

```
Features with lowest tfidf:
['poignant' 'disagree' 'instantly' 'importantly' 'lacked' 'occurred'
 'currently' 'altogether' 'nearby' 'undoubtedly' 'directs' 'fond' 'stinker'
 'avoided' 'emphasis' 'commented' 'disappoint' 'realizing' 'downhill'
 'inane']
Features with highest tfidf:
['coop' 'homer' 'dillinger' 'hackenstein' 'gadget' 'taker' 'macarthur'
 'vargas' 'jesse' 'basket' 'dominick' 'the' 'victor' 'bridget' 'victoria'
 'khouri' 'zizek' 'rob' 'timon' 'titanic']
```

Features with low tf–idf are those that either are very commonly used across documents or are only used sparingly, and only in very long documents. Interestingly, many of the high-tf–idf features actually identify certain shows or movies. These terms only appear in reviews for this particular show or franchise, but tend to appear very often in these particular reviews. This is very clear, for example, for "pokemon", "smallville", and "doodlebops", but "scanners" here actually also refers to a movie title. These words are unlikely to help us in our sentiment classification task (unless maybe some franchises are universally reviewed positively or negatively) but certainly contain a lot of specific information about the reviews.

We can also find the words that have low inverse document frequency—that is, those that appear frequently and are therefore deemed less important. The inverse document frequency values found on the training set are stored in the `idf_` attribute:

In[25]:

```
sorted_by_idf = np.argsort(vectorizer.idf_)
print("Features with lowest idf:\n{}".format(
    feature_names[sorted_by_idf[:100]]))
```

Out[25]:

```
Features with lowest idf:
['the' 'and' 'of' 'to' 'this' 'is' 'it' 'in' 'that' 'but' 'for' 'with'
 'was' 'as' 'on' 'movie' 'not' 'have' 'one' 'be' 'film' 'are' 'you' 'all'
 'at' 'an' 'by' 'so' 'from' 'like' 'who' 'they' 'there' 'if' 'his' 'out'
 'just' 'about' 'he' 'or' 'has' 'what' 'some' 'good' 'can' 'more' 'when'
 'time' 'up' 'very' 'even' 'only' 'no' 'would' 'my' 'see' 'really' 'story'
 'which' 'well' 'had' 'me' 'than' 'much' 'their' 'get' 'were' 'other'
 'been' 'do' 'most' 'don' 'her' 'also' 'into' 'first' 'made' 'how' 'great'
 'because' 'will' 'people' 'make' 'way' 'could' 'we' 'bad' 'after' 'any'
 'too' 'then' 'them' 'she' 'watch' 'think' 'acting' 'movies' 'seen' 'its'
 'him']
```

As expected, these are mostly English stopwords like "the" and "no". But some are clearly domain-specific to the movie reviews, like "movie", "film", "time", "story", and so on. Interestingly, "good", "great", and "bad" are also among the most frequent and therefore "least relevant" words according to the tf–idf measure, even though we might expect these to be very important for our sentiment analysis task.

Investigating Model Coefficients

Finally, let's look in a bit more detail into what our logistic regression model actually learned from the data. Because there are so many features—27,271 after removing the infrequent ones—we clearly cannot look at all of the coefficients at the same time. However, we can look at the largest coefficients, and see which words these correspond to. We will use the last model that we trained, based on the tf–idf features.

The following bar chart (Figure 7-2) shows the 25 largest and 25 smallest coefficients of the logistic regression model, with the bars showing the size of each coefficient: