**Out[33]:**

```
Score without poly features: 0.63
```

As we would expect looking at the grid search results visualized in Figure 6-4, using no polynomial features leads to decidedly worse results.

Searching over preprocessing parameters together with model parameters is a very powerful strategy. However, keep in mind that `GridSearchCV` tries *all possible combinations* of the specified parameters. Therefore, adding more parameters to your grid exponentially increases the number of models that need to be built.

## Grid-Searching Which Model To Use

You can even go further in combining `GridSearchCV` and `Pipeline`: it is also possible to search over the actual steps being performed in the pipeline (say whether to use `StandardScaler` or `MinMaxScaler`). This leads to an even bigger search space and should be considered carefully. Trying all possible solutions is usually not a viable machine learning strategy. However, here is an example comparing a `RandomForestClassifier` and an `SVC` on the `iris` dataset. We know that the `SVC` might need the data to be scaled, so we also search over whether to use `StandardScaler` or no preprocessing. For the `RandomForestClassifier`, we know that no preprocessing is necessary. We start by defining the pipeline. Here, we explicitly name the steps. We want two steps, one for the preprocessing and then a classifier. We can instantiate this using `SVC` and `StandardScaler`:

**In[34]:**

```
pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVC())])
```

Now we can define the `parameter_grid` to search over. We want the `classifier` to be either `RandomForestClassifier` or `SVC`. Because they have different parameters to tune, and need different preprocessing, we can make use of the list of search grids we discussed in "Search over spaces that are not grids" on page 271. To assign an estimator to a step, we use the name of the step as the parameter name. When we wanted to skip a step in the pipeline (for example, because we don't need preprocessing for the `RandomForest`), we can set that step to `None`:

**In[35]:**

```
from sklearn.ensemble import RandomForestClassifier

param_grid = [
    {'classifier': [SVC()], 'preprocessing': [StandardScaler(), None],
     'classifier__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
     'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100]},
    {'classifier': [RandomForestClassifier(n_estimators=100)],
     'preprocessing': [None], 'classifier__max_features': [1, 2, 3]}]
```

Now we can instantiate and run the grid search as usual, here on the `cancer` dataset:

**In[36]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best params:\n{}\n".format(grid.best_params_))
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Test-set score: {:.2f}".format(grid.score(X_test, y_test)))
```

**Out[36]:**

```
Best params:
{'classifier':
 SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
     decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf',
     max_iter=-1, probability=False, random_state=None, shrinking=True,
     tol=0.001, verbose=False),
 'preprocessing':
 StandardScaler(copy=True, with_mean=True, with_std=True),
 'classifier__C': 10, 'classifier__gamma': 0.01}

Best cross-validation score: 0.99
Test-set score: 0.98
```

The outcome of the grid search is that `SVC` with `StandardScaler` preprocessing, `C=10`, and `gamma=0.01` gave the best result.

# Summary and Outlook

In this chapter we introduced the `Pipeline` class, a general-purpose tool to chain together multiple processing steps in a machine learning workflow. Real-world applications of machine learning rarely involve an isolated use of a model, and instead are a sequence of processing steps. Using pipelines allows us to encapsulate multiple steps into a single Python object that adheres to the familiar `scikit-learn` interface of `fit`, `predict`, and `transform`. In particular when doing model evaluation using cross-validation and parameter selection using grid search, using the `Pipeline` class to capture all the processing steps is essential for proper evaluation. The `Pipeline` class also allows writing more succinct code, and reduces the likelihood of mistakes that can happen when building processing chains without the `pipeline` class (like forgetting to apply all transformers on the test set, or not applying them in the right order). Choosing the right combination of feature extraction, preprocessing, and models is somewhat of an art, and often requires some trial and error. However, using pipelines, this "trying out" of many different processing steps is quite simple. When