Randomized PCA

Scikit-Learn offers yet another option to perform PCA, called *Randomized PCA*. This is a stochastic algorithm that quickly finds an approximation of the first d principal components. Its computational complexity is $O(m \times d^2) + O(d^3)$, instead of $O(m \times n^2) + O(n^3)$, so it is dramatically faster than the previous algorithms when d is much smaller than n.

```
rnd_pca = PCA(n_components=154, svd_solver="randomized")
X_reduced = rnd_pca.fit_transform(X_mnist)
```

Kernel PCA

In Chapter 5 we discussed the kernel trick, a mathematical technique that implicitly maps instances into a very high-dimensional space (called the *feature space*), enabling nonlinear classification and regression with Support Vector Machines. Recall that a linear decision boundary in the high-dimensional feature space corresponds to a complex nonlinear decision boundary in the *original space*.

It turns out that the same trick can be applied to PCA, making it possible to perform complex nonlinear projections for dimensionality reduction. This is called *Kernel PCA* (kPCA).[6] It is often good at preserving clusters of instances after projection, or sometimes even unrolling datasets that lie close to a twisted manifold.

For example, the following code uses Scikit-Learn's `KernelPCA` class to perform kPCA with an RBF kernel (see Chapter 5 for more details about the RBF kernel and the other kernels):

```
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

Figure 8-10 shows the Swiss roll, reduced to two dimensions using a linear kernel (equivalent to simply using the `PCA` class), an RBF kernel, and a sigmoid kernel (Logistic).

6 "Kernel Principal Component Analysis," B. Schölkopf, A. Smola, K. Müller (1999).

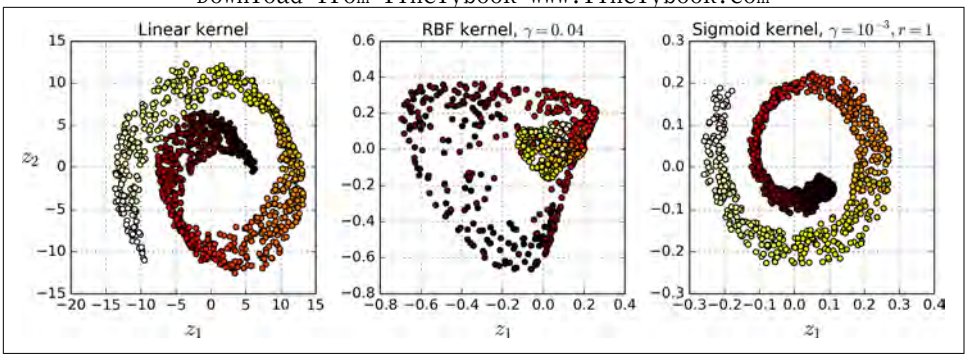

Figure 8-10. Swiss roll reduced to 2D using kPCA with various kernels

Selecting a Kernel and Tuning Hyperparameters

As kPCA is an unsupervised learning algorithm, there is no obvious performance measure to help you select the best kernel and hyperparameter values. However, dimensionality reduction is often a preparation step for a supervised learning task (e.g., classification), so you can simply use grid search to select the kernel and hyperparameters that lead to the best performance on that task. For example, the following code creates a two-step pipeline, first reducing dimensionality to two dimensions using kPCA, then applying Logistic Regression for classification. Then it uses `GridSearchCV` to find the best kernel and gamma value for kPCA in order to get the best classification accuracy at the end of the pipeline:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression())
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

The best kernel and hyperparameters are then available through the `best_params_` variable:

```
>>> print(grid_search.best_params_)
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}
```