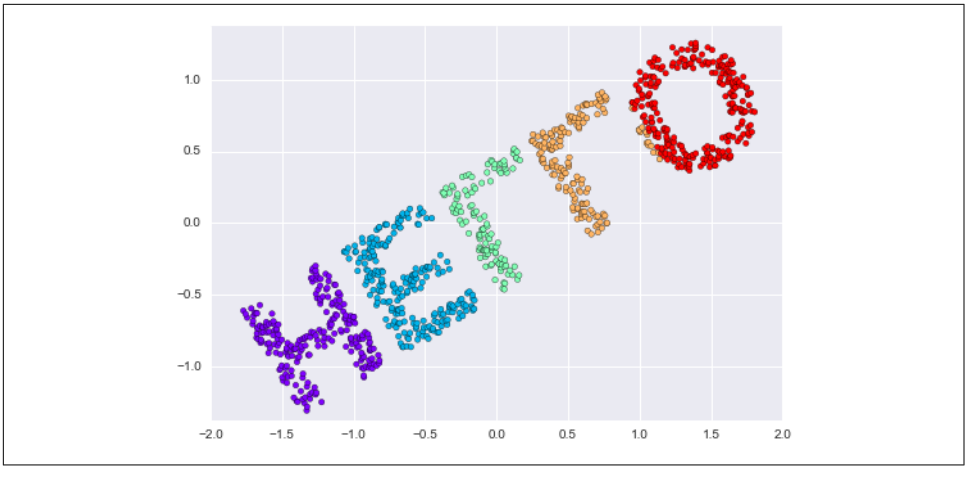

Figure 5-97. An MDS embedding computed from the pairwise distances

The MDS algorithm recovers one of the possible two-dimensional coordinate representations of our data, using *only* the $N \times N$ distance matrix describing the relationship between the data points.

MDS as Manifold Learning

The usefulness of this becomes more apparent when we consider the fact that distance matrices can be computed from data in *any* dimension. So, for example, instead of simply rotating the data in the two-dimensional plane, we can project it into three dimensions using the following function (essentially a three-dimensional generalization of the rotation matrix used earlier):

```
In[9]: def random_projection(X, dimension=3, rseed=42):
           assert dimension >= X.shape[1]
           rng = np.random.RandomState(rseed)
           C = rng.randn(dimension, dimension)
           e, V = np.linalg.eigh(np.dot(C, C.T))
           return np.dot(X, V[:X.shape[1]])

       X3 = random_projection(X, 3)
       X3.shape

Out[9]: (1000, 3)
```

Let's visualize these points to see what we're working with (Figure 5-98):

```
In[10]: from mpl_toolkits import mplot3d
        ax = plt.axes(projection='3d')
        ax.scatter3D(X3[:, 0], X3[:, 1], X3[:, 2],
                     **colorize)
        ax.view_init(azim=70, elev=50)
```

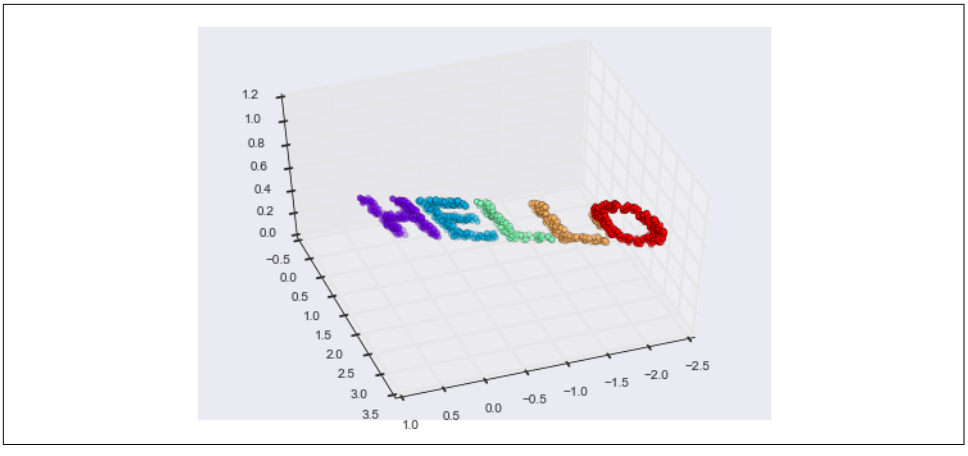

Figure 5-98. Data embedded linearly into three dimensions

We can now ask the `MDS` estimator to input this three-dimensional data, compute the distance matrix, and then determine the optimal two-dimensional embedding for this distance matrix. The result recovers a representation of the original data (Figure 5-99):

```
In[11]: model = MDS(n_components=2, random_state=1)
        out3 = model.fit_transform(X3)
        plt.scatter(out3[:, 0], out3[:, 1], **colorize)
        plt.axis('equal');
```

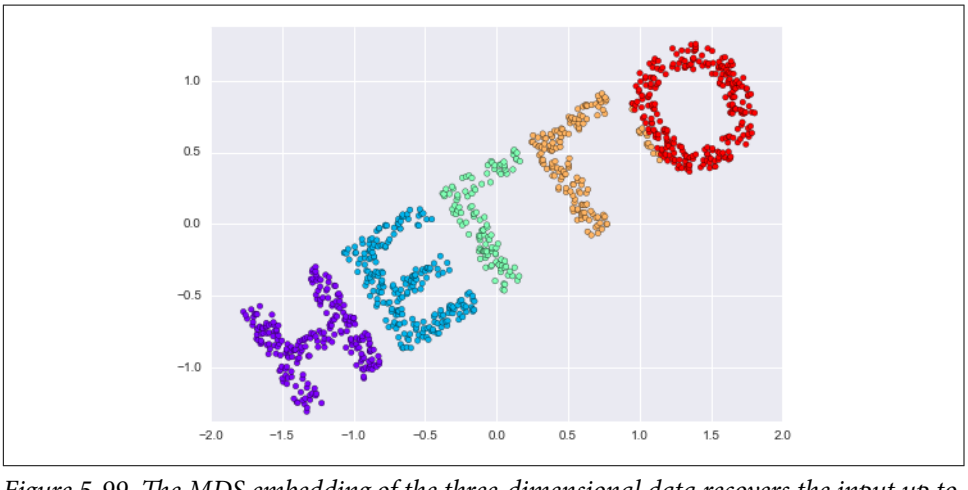

Figure 5-99. The MDS embedding of the three-dimensional data recovers the input up to a rotation and reflection

This is essentially the goal of a manifold learning estimator: given high-dimensional embedded data, it seeks a low-dimensional representation of the data that preserves

certain relationships within the data. In the case of MDS, the quantity preserved is the distance between every pair of points.

Nonlinear Embeddings: Where MDS Fails

Our discussion so far has considered *linear* embeddings, which essentially consist of rotations, translations, and scalings of data into higher-dimensional spaces. Where MDS breaks down is when the embedding is nonlinear—that is, when it goes beyond this simple set of operations. Consider the following embedding, which takes the input and contorts it into an "S" shape in three dimensions:

```
In[12]: def make_hello_s_curve(X):
            t = (X[:, 0] - 2) * 0.75 * np.pi
            x = np.sin(t)
            y = X[:, 1]
            z = np.sign(t) * (np.cos(t) - 1)
            return np.vstack((x, y, z)).T

        XS = make_hello_s_curve(X)
```

This is again three-dimensional data, but we can see that the embedding is much more complicated (Figure 5-100):

```
In[13]: from mpl_toolkits import mplot3d
        ax = plt.axes(projection='3d')
        ax.scatter3D(XS[:, 0], XS[:, 1], XS[:, 2],
                     **colorize);
```

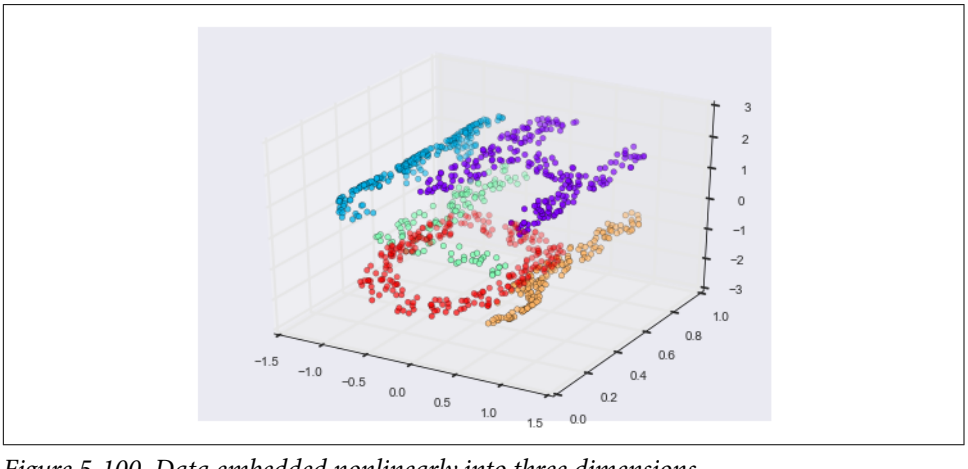

Figure 5-100. Data embedded nonlinearly into three dimensions

The fundamental relationships between the data points are still there, but this time the data has been transformed in a nonlinear way: it has been wrapped up into the shape of an "S."