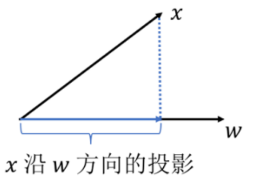


图 10.1 $\boldsymbol{x}$ 沿 $\boldsymbol{w}$ 方向的投影

主成分分析法

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{t=1}^{m} \boldsymbol{x}^{(t)}$$

For $t = 1,2,\dots,m$:

$$\boldsymbol{x}^{(t)} \leftarrow \boldsymbol{x}^{(t)} - \boldsymbol{\mu}$$

$$\boldsymbol{X} = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ {\boldsymbol{x}^{(2)}}^T \\ \dots \\ {\boldsymbol{x}^{(m)}}^T \end{bmatrix}$$

Compute eigenvalues of $\boldsymbol{X}^T\boldsymbol{X}$: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$

Let $\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \dots, \boldsymbol{w}^{(n)}$ be the corresponding eigenvectors

$$\boldsymbol{W} = \left(\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \dots, \boldsymbol{w}^{(d)}\right)$$

Return $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$

图 10.2　主成分分析的算法描述

machine_learning.lib.pca

```python
import numpy as np

class PCA:
    def __init__(self, n_components):
        self.d = n_components

    def fit_transform(self, X):
        self.mean = X.mean(axis = 0)
        X = X - self.mean
        eigen_values, eigen_vectors = np.linalg.eig(X.T.dot(X))
        n = len(eigen_values)
        pairs = [(eigen_values[i], eigen_vectors[:, i]) for i in range(n)]
        pairs = sorted(pairs, key = lambda pair: pair[0], reverse = True)
        self.W = np.array([pairs[r][1] for r in range(self.d)]).T
        return X.dot(self.W)

    def inverse_transform(self, Z):
        return Z.dot(self.W.T) + self.mean
```

图 10.3　主成分分析法

```python
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
from machine_learning.lib.pca import PCA

mnist = input_data.read_data_sets("MNIST_data/", one_hot=False)
X, Y = mnist.train.images, mnist.train.labels
model = PCA(n_components = 100)
Z = model.fit_transform(X)
X_recovered = model.inverse_transform(Z).astype(int)

plt.figure(0)
plt.imshow(X[0].reshape(28,28))
plt.figure(1)
plt.imshow(X_recovered[0].reshape(28,28))
plt.figure(2)
plt.scatter(Z[:,0], Z[:,1], c = Y)
plt.show()
```

图 10.4　手写数字图片降维的主成分分析法

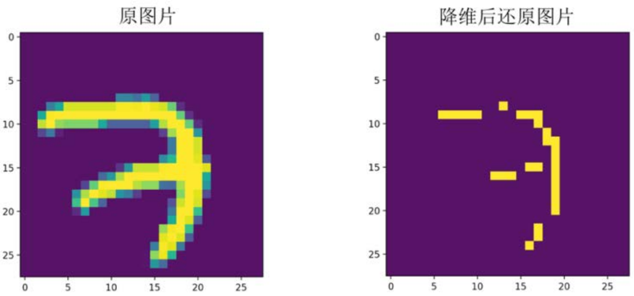


图 10.5　经数据重构的图片与原图的对比

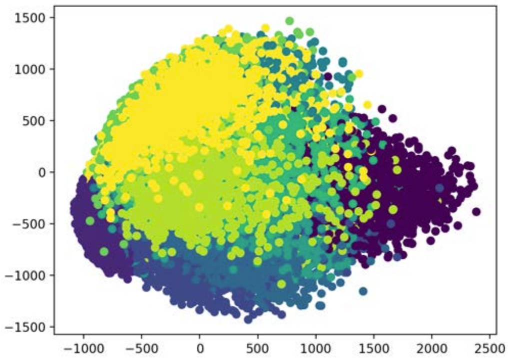


图 10.6　MNIST 中数据的二维展示

machine_learning.lib.pca_svd

```python
import numpy as np

class PCA:
    def __init__(self, n_components):
        self.d = n_components

    def fit_transform(self, X):
        self.mean = X.mean(axis = 0)
        X = X - self.mean
        U, D, VT = np.linalg.svd(X)
        self.W = VT[0: self.d].T
        return X.dot(self.W)

    def inverse_transform(self, Z):
        return Z.dot(self.W.T) + self.mean
```

图 10.7　矩阵奇异值分解的 PCA 法

PCA 法的等价描述

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{t=1}^{m} \boldsymbol{x}^{(t)}$$

For $t = 1,2,\ldots,m$:

$$\boldsymbol{x}^{(t)} \leftarrow \boldsymbol{x}^{(t)} - \boldsymbol{\mu}$$

$$\boldsymbol{X} = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ {\boldsymbol{x}^{(2)}}^T \\ \ldots \\ {\boldsymbol{x}^{(m)}}^T \end{bmatrix}$$

Compute eigenvalues of $\boldsymbol{X}\boldsymbol{X}^T$: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$

Let $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(m)}$ be the corresponding eigenvectors

Return $\boldsymbol{Z} = \left(\sqrt{\lambda_1}\boldsymbol{u}^{(1)}, \sqrt{\lambda_2}\boldsymbol{u}^{(2)}, \ldots, \sqrt{\lambda_d}\boldsymbol{u}^{(d)}\right)$

图 10.8　主成分分析法的等价描述

主成分分析的核方法

For $t, s = 1,2,\cdots,m$:

$$K_{t,s} = K_{\phi}\left(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(s)}\right)$$

$$\widehat{\boldsymbol{K}} = \boldsymbol{K} - \boldsymbol{J}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{J} + \boldsymbol{J}\boldsymbol{K}\boldsymbol{J}$$

Compute eigenvalues of $\widehat{\boldsymbol{K}}$: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$

Let $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \dots, \boldsymbol{u}^{(m)}$ be the corresponding eigenvectors

Return $\boldsymbol{Z} = \left(\sqrt{\lambda_1}\boldsymbol{u}^{(1)}, \sqrt{\lambda_2}\boldsymbol{u}^{(2)}, \dots, \sqrt{\lambda_d}\boldsymbol{u}^{(d)}\right)$

图 10.9　主成分分析的核方法描述

machine_learning.lib.kernel_pca

```python
import numpy as np

def default_kernel(x1, x2):
    return x1.dot(x2.T)

class KernelPCA:
    def __init__(self, n_components, kernel = default_kernel):
        self.d = n_components
        self.kernel = kernel

    def fit_transform(self, X):
        m,n = X.shape
        K = np.zeros((m,m))
        for s in range(m):
            for r in range(m):
                K[s][r] = self.kernel(X[s],X[r])
        J = np.ones((m,m)) * (1.0 / m)
        K = K - J.dot(K) - K.dot(J) + J.dot(K).dot(J)
        eigen_values, eigen_vectors = np.linalg.eig(K)
        pairs = [(eigen_values[i], eigen_vectors[:,i]) for i in range(m)]
        pairs = sorted(pairs, key = lambda pair: pair[0], reverse = True)
        Z = np.array([pairs[i][1] * np.sqrt(pairs[i][0]) for i in range(self.d)]).T
        return Z
```

图 10.10　主成分分析的核方法

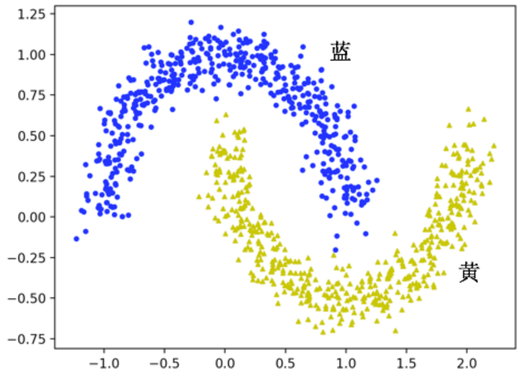
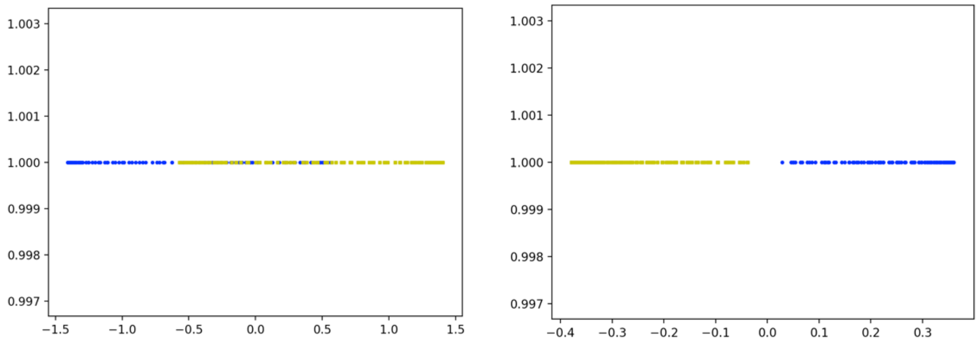


图 10.11 月亮数据集采样

```python
import numpy as np
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
from machine_learning.lib.kernel_pca import KernelPCA
from machine_learning.lib.pca import PCA

def rbf_kernel(x1, x2):
    sigma = 1.0 / 15
    return np.exp(- np.linalg.norm(x1 - x2, 2) ** 2 / sigma)

np.random.seed(0)
X, y = make_moons(n_samples=500, noise=0.01)
plt.figure(0)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='rainbow')

pca = PCA(n_components = 1)
X_pca = pca.fit_transform(X).reshape(-1)
kpca = KernelPCA(n_components = 1, kernel = rbf_kernel)
X_kpca = kpca.fit_transform(X).reshape(-1)

plt.figure(1)
plt.scatter(X_pca, np.ones(X_pca.shape), c=y, cmap='rainbow')
plt.figure(2)
plt.scatter(X_kpca, np.ones(X_kpca.shape), c=y, cmap='rainbow')
plt.show()
```

图 10.12 月亮数据降维的主成分分析法与主成分分析法的核方法



(a) (b)

图 10.13 主成分分析法与主成分分析法的核方法对比

线性判别分析法

Compute eigenvalues of $\boldsymbol{S}_{\boldsymbol{w}}^{-1}\boldsymbol{S}_{\boldsymbol{b}}$: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$

Let $\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(n)}$ be the corresponding eigenvectors

$\boldsymbol{W} = \left(\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(d)}\right)$

Return $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$

图 10.14　线性判别分析算法描述

machine_learning.lib.lda

```python
import numpy as np

class LDA:
    def __init__(self, n_components):
        self.d = n_components

    def fit_transform(self, X, y):
        sums = dict()
        counts = dict()
        m,n = X.shape
        for t in range(m):
            i = y[t]
            if i not in sums:
                sums[i] = np.zeros((1,n))
                counts[i] = 0
            sums[i] += X[t].reshape(1,n)
            counts[i] += 1
        X_mean = np.mean(X, axis=0).reshape(1,n)
        S_b = np.zeros((n,n))
        for i in counts:
            v = X_mean - 1.0 * sums[i] / counts[i]
            S_b += counts[i] * v.T.dot(v)
        S_w = np.zeros((n,n))
        for t in range(m):
            i = y[t]
            u = X[t].reshape(1,n) - 1.0 * sums[i] / counts[i]
            S_w += u.T.dot(u)
        A = np.linalg.pinv(S_w).dot(S_b)
        values, vectors = np.linalg.eig(A)
        pairs = [(values[j], vectors[:, j]) for j in range(len(values))]
        pairs = sorted(pairs, key = lambda pair: pair[0], reverse = True)
        W = np.array([pairs[j][1] for j in range(self.d)]).T
        return X.dot(W)
```

图 10. 15　线性判别分析算法

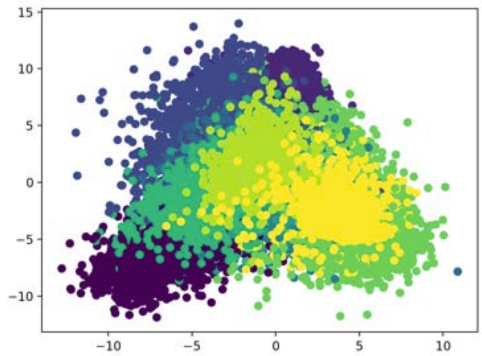


图 10.16　手写数字数据的线性判别分析法降维效果

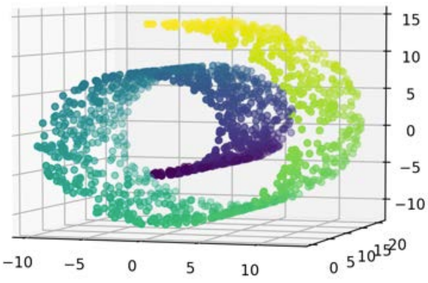


图 10.17　流形数据示意

局部线性嵌入法

For $i = 1,2,\ldots,m$:

Pick k nearest neighbors of $\boldsymbol{x}^{(i)}$: $\boldsymbol{x}^{(i_1)}, \boldsymbol{x}^{(i_2)}, \ldots, \boldsymbol{x}^{(i_k)}$

$$\boldsymbol{U} = \begin{bmatrix} {\boldsymbol{x}^{(i_1)}}^T - {\boldsymbol{x}^{(i)}}^T \\ {\boldsymbol{x}^{(i_2)}}^T - {\boldsymbol{x}^{(i)}}^T \\ \ldots \\ {\boldsymbol{x}^{(i_k)}}^T - {\boldsymbol{x}^{(i)}}^T \end{bmatrix}$$

$$\boldsymbol{v} = (\boldsymbol{U}\boldsymbol{U}^T)^{-1}\mathbf{1}_k$$

$$\boldsymbol{w}^{(i)} = \frac{\boldsymbol{v}}{\boldsymbol{v}^T\mathbf{1}_k}$$

$$\boldsymbol{W} = \left(w_{i,j}\right)_{1\le i,j\le m} = \mathbf{0}$$

For $i = 1,2,\ldots,m$:

For $t = 1,2,\ldots,k$:

$$w_{i,i_t} = w_t^{(i)}, 1 \le t \le k$$

$$\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{W})^T(\boldsymbol{I} - \boldsymbol{W})$$

Compute eigenvalues of $\boldsymbol{M}$: $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_m$

Let $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(m)}$ be the corresponding eigenvectors

Return $\boldsymbol{Z} = \left(\boldsymbol{v}^{(m-d)}, \ldots, \boldsymbol{v}^{(m-1)}\right)$

图 10.18　局部线性嵌入法的算法描述

machine_learning.lib.lle

```
import numpy as np
from sklearn.neighbors import NearestNeighbors

class LLE:
    def __init__(self, n_components, n_neighbors):
        self.d = n_components
        self.k = n_neighbors

    def get_weights(self, X, knn):
        m, n = X.shape
        W = np.zeros((m,m))
        for i in range(m):
            U = X[knn[i]].reshape(-1,n)
            k = len(U)
            for t in range(k):
                U[t] -= X[i]
            C = U.dot(U.T)
            w = np.linalg.inv(C).dot(np.ones((k,1)))
            w /= w.sum(axis=0)
            for t in range(k):
                W[i][knn[i][t]] = w[t]
        return W

    def fit_transform(self,X):
        m, n = X.shape
        model = NearestNeighbors(n_neighbors = self.k + 1).fit(X)
        knn = model.kneighbors(X, return_distance = False)[:, 1:]
        W = self.get_weights(X, knn)
        M = (np.identity(m) - W).T.dot(np.identity(m) - W)
        eigen_values, eigen_vectors = np.linalg.eig(M)
        pairs = [(eigen_values[i], eigen_vectors[:, i]) for i in range(m)]
        pairs = sorted(pairs, key = lambda pair: pair[0])
        Z = np.array([pairs[i+1][1] for i in range(self.d)]).T
        return Z
```

图 10.19　局部线性嵌入算法

```python
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from machine_learning.lib.lle impor LLE

np.random.seed(0)
X, color = datasets.samples_generator.make_swiss_roll(n_samples=1500)
model = LLE(n_components=2, n_neighbors=12)
Z = model.fit_transform(X)
plt.scatter(Z[:, 0], Z[:, 1], c=color)
plt.show()
```

图 10.20　瑞士卷数据降维的局部线性嵌入

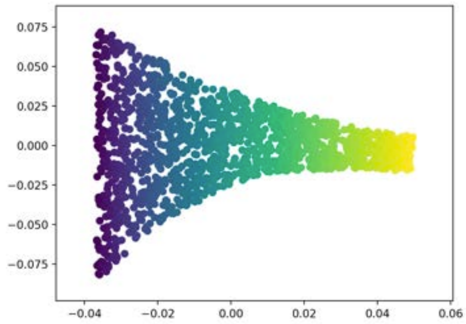


图 10.21　瑞士卷数据的局部线性嵌入法降维

多维放缩算法

$\boldsymbol{\mu} = \frac{1}{m}\sum_{t=1}^{m} \boldsymbol{x}^{(t)}$

For $t = 1,2,\ldots,m$:

$\boldsymbol{x}^{(t)} \leftarrow \boldsymbol{x}^{(t)} - \boldsymbol{\mu}$

$$\boldsymbol{X} = \begin{bmatrix} {\boldsymbol{x}^{(1)}}^T \\ {\boldsymbol{x}^{(2)}}^T \\ \ldots \\ {\boldsymbol{x}^{(m)}}^T \end{bmatrix}$$

Compute eigenvalues of $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{X}^T$: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$

Let $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(m)}$ be the corresponding eigenvectors

Return $\boldsymbol{Z} = \left(\sqrt{\lambda_1}\boldsymbol{v}^{(1)}, \sqrt{\lambda_2}\boldsymbol{v}^{(2)}, \ldots, \sqrt{\lambda_d}\boldsymbol{v}^{(d)}\right)$

图 10.22 多维放缩算法描述

machine_learning.lib.mds

```
import numpy as np

class MDS:
    def __init__(self, n_components):
        self.d = n_components

    def fit_transform(self, X):
        m, n = X.shape
        self.mean = X.mean(axis = 0)
        X = X - self.mean
        B = X.dot(X.T)
        eigen_values, eigen_vectors = np.linalg.eig(B)
        pairs = [(eigen_values[i], eigen_vectors[:,i]) for i in range(m)]
        pairs = sorted(pairs, key = lambda pair: pair[0], reverse = True)
        Z = np.array([pairs[i][1] * np.sqrt(pairs[i][0]) for i in range(self.d)]).T
        return Z
```

图 10.23　多维放缩算法

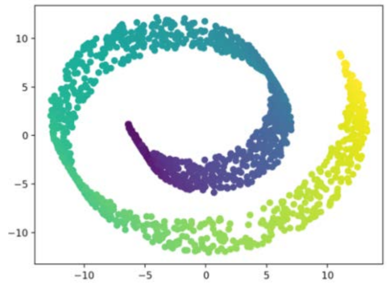


图 10.24　瑞士卷数据的多维放缩算法降维

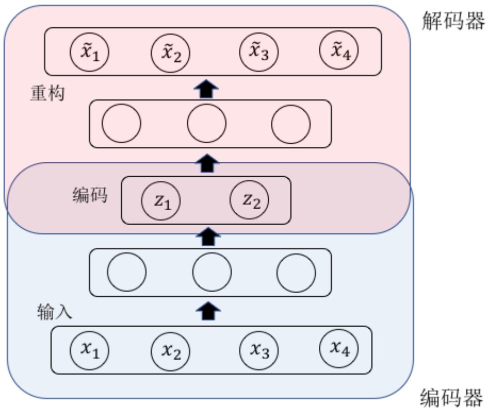


图 10.25　自动编码器结构

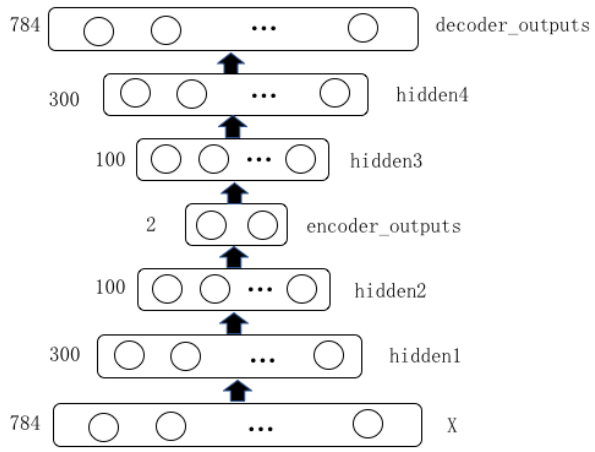


图 10.26　手写数字数据降维的自动编码器结构

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

n_features = 28 * 28
n_hidden1 = 300
n_hidden2 = 100
n_encoder_outputs = 2
n_hidden3 = n_hidden2
n_hidden4 = n_hidden1
n_decoder_ouputs = n_features

X = tf.placeholder(tf.float32, shape=(None, n_features))
hidden1 = tf.layers.dense(X, n_hidden1, activation = tf.nn.relu)
hidden2 = tf.layers.dense(hidden1, n_hidden2, activation = tf.nn.relu)
encoder_outputs = tf.layers.dense(hidden2, n_encoder_outputs)
hidden3 = tf.layers.dense(encoder_outputs, n_hidden3, activation = tf.nn.relu)
hidden4 = tf.layers.dense(hidden3, n_hidden4, activation = tf.nn.relu)
decoder_outputs = tf.layers.dense(hidden4, n_decoder_ouputs)

recover_loss = tf.reduce_mean(tf.square(decoder_outputs - X))
optimizer = tf.train.AdamOptimizer(learning_rate = 1e-4)
train_op = optimizer.minimize(recover_loss)

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=False)
    n_epoches = 10
    batch_size = 150
    for epoch in range(n_epoches):
        for batch in range(mnist.train.num_examples // batch_size):
            X_batch, _ = mnist.train.next_batch(batch_size)
            sess.run(train_op, feed_dict = {X: X_batch})

    Z = sess.run(encoder_outputs, feed_dict = {X:mnist.train.images})
    plt.scatter(Z[:,0], Z[:,1], c = mnist.train.labels)
    plt.show()
```

图 10.27　手写数字数据降维的自动编码器算法

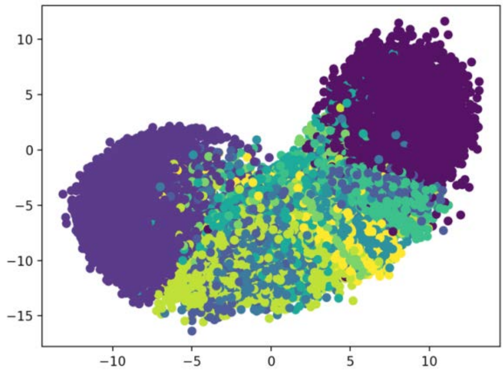


图 10.28　自动编码器对手写数字数据降维效果