Dimensionality reduction

> Pattern Recognition Homeworks
>
> Student: thu-zxs

Solutions

Problem 1

Maximum-variance approach

1.1

- data matrix:

$$X_N = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$$

- projected data matrix:

$$X_1 = \{\mathbf{x}_1^T\mathbf{u}_1; \mathbf{x}_2^T\mathbf{u}_1; \ldots; \mathbf{x}_N^T\mathbf{u}_1\} = X_N^T\mathbf{u}_1$$

- **covariance** of projected matrix:

$$\begin{aligned} C &= (X_1 - \bar{\mathbf{x}}_1)^T(X_1 - \bar{\mathbf{x}}_1) \\ &= (X_N^T\mathbf{u}_1 - \bar{\mathbf{x}}_N^T\mathbf{u}_1)^T(X_N^T\mathbf{u}_1 - \bar{\mathbf{x}}_N^T\mathbf{u}_1) \\ &= \mathbf{u}_1^T(X_N - \bar{\mathbf{x}}_N)(X_N - \bar{\mathbf{x}}_N)^T\mathbf{u}_1 \\ &= \mathbf{u}_1^T S\mathbf{u}_1 \end{aligned}$$

  where $\bar{\mathbf{x}}_N$ is the mean vector of the sample data vectors.

1.2

$$\max_{\mathbf{u}_1} \mathbf{u}_1^T S\mathbf{u}_1$$
$$s.t. \mathbf{u}_1^T\mathbf{u}_1 = 1$$

By introducing Lagrange multiplier λ we minimize:

$$\min_{\mathbf{u}_1} \mathbf{u}_1^T S\mathbf{u}_1 - \lambda(\mathbf{u}_1^T\mathbf{u}_1 - 1)$$

And take derivative:

$$\frac{\partial f}{\partial \mathbf{u}_1} = 2S\mathbf{u}_1 - 2\lambda\mathbf{u}_1 = 0$$

yield

$$S\mathbf{u}_1 = \lambda\mathbf{u}_1$$

so $\mathbf{u}_1$ is the eigenvector of S. Left multiplying both side with $\mathbf{u}_1^T$ shows that:

$$C = \lambda,$$

which is the variance of the projected data. And the best $\mathbf{u}_1$ is the projection that corresponds to the largest eigenvalue λ, which agrees with the PCA principles.

1.3

Assume the best (Maximum-variance) M projection correspond to the M-largest eigenvalue of covariance matrix. We aim to find the $(M+1)_{th}$ projection $\mathbf{u}_{M+1}$ by maximizing variance:

$$\begin{aligned}
&\max_{\mathbf{u}_{M+1}} \mathbf{u}_{M+1}^T S \mathbf{u}_{M+1} \\
&s.t. \\
&\mathbf{u}_{M+1}^T \mathbf{u}_{M+1} = 1, \\
&\mathbf{u}_{M+1}^T \mathbf{u}_i = 0, \forall i = 1, \ldots, M
\end{aligned}$$

From the same deduction, yielding:

$$\frac{\partial f}{\partial \mathbf{u}_{M+1}} = 2S\mathbf{u}_{M+1} - 2\lambda_{M+1}\mathbf{u}_{M+1} - \lambda_1 \mathbf{u}_1 - \ldots - \lambda_M \mathbf{u}_M = 0$$

and

$$2S\mathbf{u}_{M+1} - 2\lambda_{M+1}\mathbf{u}_{M+1} = \lambda_1 \mathbf{u}_1 + \ldots + \lambda_M \mathbf{u}_M$$

because $\mathbf{u}_{M+1}$ is orthogonal to the subspace spanned by $\mathbf{u}_1, \ldots, \mathbf{u}_M$, so the only possible consequence is:

$$\lambda_1 \mathbf{u}_1 + \ldots + \lambda_M \mathbf{u}_M = 0$$

thus

$$\lambda_1 = \lambda_2 = \ldots = \lambda_M = 0$$

Thus λ_{M+1} is the $(M+1)$-largest eigenvalue of S and the projection $\mathbf{u}_{M+1}$ is the corresponding eigenvector.

1.4

$$\begin{aligned}
J &= \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|_2^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}\Big(\sum_{i=1}^{M}(\mathbf{x}_n^T\mathbf{u}_i - z_{ni})^2 + \sum_{i=M+1}^{D}(\mathbf{x}_n^T\mathbf{u}_i - b_i)^2\Big)
\end{aligned}$$

Solve best b_i:

$$\frac{\partial J}{\partial b_i} = -\frac{2}{N}\sum_{n=1}^{N}(\mathbf{x}_n\mathbf{u}_i - b_i) = 0$$

$$b_i = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{u}_i = \bar{\mathbf{x}}^T\mathbf{u}_j$$

Solve best z_{ni}:

$$\frac{\partial J}{\partial z_{ni}} = -\frac{2}{N}(\mathbf{x_n^T}\mathbf{u_i} - z_{ni}) = 0$$

$$z_{ni} = \mathbf{x}_n^T\mathbf{u}_i$$

Problem 2

This solution is motivated by the work published on *Neural Networks* in 1989. [1]

neural network design

- Input d-dimentional vector: $\mathbf{x}$
- Hidden-layer
  - $k \times d$ weights W_1
- Output-layer
  - $d \times k$ weights W_2
- constraint
  - $k < d$

algorithm

- output reprensentation

  $\mathbf{y}_{hidden} = W_1\mathbf{x}$

  $\mathbf{y}_{out} = W_2\mathbf{y}_{hidden} = W_2(W_1\mathbf{x}) = W_2W_1\mathbf{x}$
- minimizing the error function
  - a set of input vector: $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$
  - and ground truth vector to regress to: $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$

  $$\min_{W_2, W_1, b_2, b_1}\sum_{i=1}^{n}E(W_2, W_1),$$

  where

  $$E(W_2, W_1) = \|W_2W_1\mathbf{x}_i - \mathbf{y}_i\|_2^2$$

We first conclude that E has a unique local and global minimum corresponding to an orthogonal projection onto the subspace spanned by the principal eigenvectors of the convariance matrix associated with the training data X. We will prove this.

We denote by P_M the matrix of the orthogonal projection onto the subspace spanned by the columns of M if M is a $d \times k$ matrix. And P_M has the following properties:

1. $P_M^2 = P_M$
2. $P_M^T = P_M$
3. $P_M = M(M^T M)^{-1} M^T$ if M is of full rank k

   We also have the following facts by proof:

   Fact 1: For any W_2 the error function $E(W_2, W_1)$ is convex in W_1 and reaches minimum for any W_1 satisfying the equation:

$$W_2^T W_2 W_1 \Sigma_{XX} = W_2^T \Sigma_{YX}$$

if Σ_{XX} is invertible and W_2 is of full rank d, then the minimum of E is reached when

$$W_1 = (W_2^T W_2)^{-1} W_2^T \Sigma_{YX} \Sigma_{XX}^{-1}$$

Fact 2: Assume Σ_{XX} is invertible. if W_1 and W_2 define a critical point of E, then $W = W_2 W_1$ is of the form:

$$W = P_{W_2} \Sigma_{YX} \Sigma_{XX}^{-1},$$

where W_2 satisies:

$$P_{W_2} \Sigma = P_{W_2} \Sigma P_{W_2} = \Sigma P_{W_2},$$

where $\Sigma = \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY}$

Fact 3: Assume Σ is full rank with n distinct eigenvalues $\lambda_1 > \ldots > \lambda_d$. Let $\mathcal{F} = \{i_1, \ldots, i_k\} (1 \leq i_1 < \ldots < i_k \leq d)$ is any ordered k-index set. Let $U_{\mathcal{F}} = [\mathbf{u}_{i_1}, \ldots, \mathbf{u}_{i_k}]$ denote the matrix formed by the orthonormal eigenvectors of Σ associated with the eigenvalues $\lambda_{i_1}, \ldots, \lambda_{i_k}$. Then if W_2 and W_1 define a critical point of E, there must exist an $\mathcal{F}$ and an invertible $k \times k$ matrix C such that:

$$\begin{aligned} W_2 &= U_{\mathcal{F}} C \\ W_1 &= C^{-1} U_{\mathcal{F}}^T \Sigma_{YX} \Sigma_{XX}^{-1} \\ W &= P_{U_{\mathcal{F}}} \Sigma_{YX} \Sigma_{XX}^{-1} \end{aligned}$$

Proof: we here assume _Fact 1_ and _Fact 2_ holds and only prove _Fact 3_ for simplicity.

Since Σ is a real symmetric convariance matrix, it can be decomposed in to $U \Lambda U^T$ where U is an orthogonal column matrix of eigenvectors of Σ and Λ is the diagonal matrix with non-increasing eigenvalues on its diagonal.

We have:

$$\begin{aligned} P_{U^T W_2} &= U^T W_2 (W_2^T U U^T W_2)^{-1} W_2^T U = U^T W_2 (W_2^T W_2)^{-1} W_2^T U = U^T P_{W_2} U \\ P_{W_2} &= U P_{U^T W_2} U^T \end{aligned}$$

From _Fact 2_

$$P_{W_2}\Sigma = \Sigma P_{W_2}$$

Namely,

$$UP_{U^TW_2}U^TU\Lambda U^T = U\Lambda U^TUP_{U^TW_2}U^T$$

so,

$$UP_{U^TW_2}\Lambda U^T = U\Lambda P_{U^TW_2}U^T$$

so,

$$P_{U^TW_2}\Lambda = \Lambda P_{U^TW_2}$$

Since Λ is diagonal and $\lambda_1 > \ldots > \lambda_d > 0$, $P_{U^TW_2}$ is diagonal. Also $P_{U^TW_2}$ is an orthogonal projection matrix, so it's diagonal elements are 1 (k times) and 0 ($d-k$ times). So there exist a $\mathcal{F} = \{i_1, \ldots, i_k\}$ such that $P_{U^TW_2} = I_{\mathcal{F}}$. So,

$$P_{W_2} = UP_{U^TW_2}U^T = UI_{\mathcal{F}}U^T = U_{\mathcal{F}}U_{\mathcal{F}}^T = U_{\mathcal{F}}(U_{\mathcal{F}}^TU_{\mathcal{F}})^{-1}U_{\mathcal{F}}^T$$

Where $U_{\mathcal{F}} = [\mathbf{u}_{i_1}, \ldots, \mathbf{u}_{i_k}]$. Thus P_{W_2} is the orthogonnal projection onto the subspace spanned by the columns of $U_{\mathcal{F}}$. And column space of $U_{\mathcal{F}}$ coincides with the column space of W_2, so there exists an invertible matrix C such that $W_2 = U_{\mathcal{F}}C$ and from *Fact 1* $W_1 = C^{-1}U_{\mathcal{F}}^T\Sigma_{YX}\Sigma_{XX}^{-1}$. So,

$$W = P_{U_{\mathcal{F}}}\Sigma_{YX}\Sigma_{XX}^{-1}$$

In auto-associative(自联想) case, $\Sigma_{YX} = \Sigma_{XX}$, so,

$$W = P_{U_{\mathcal{F}}},$$

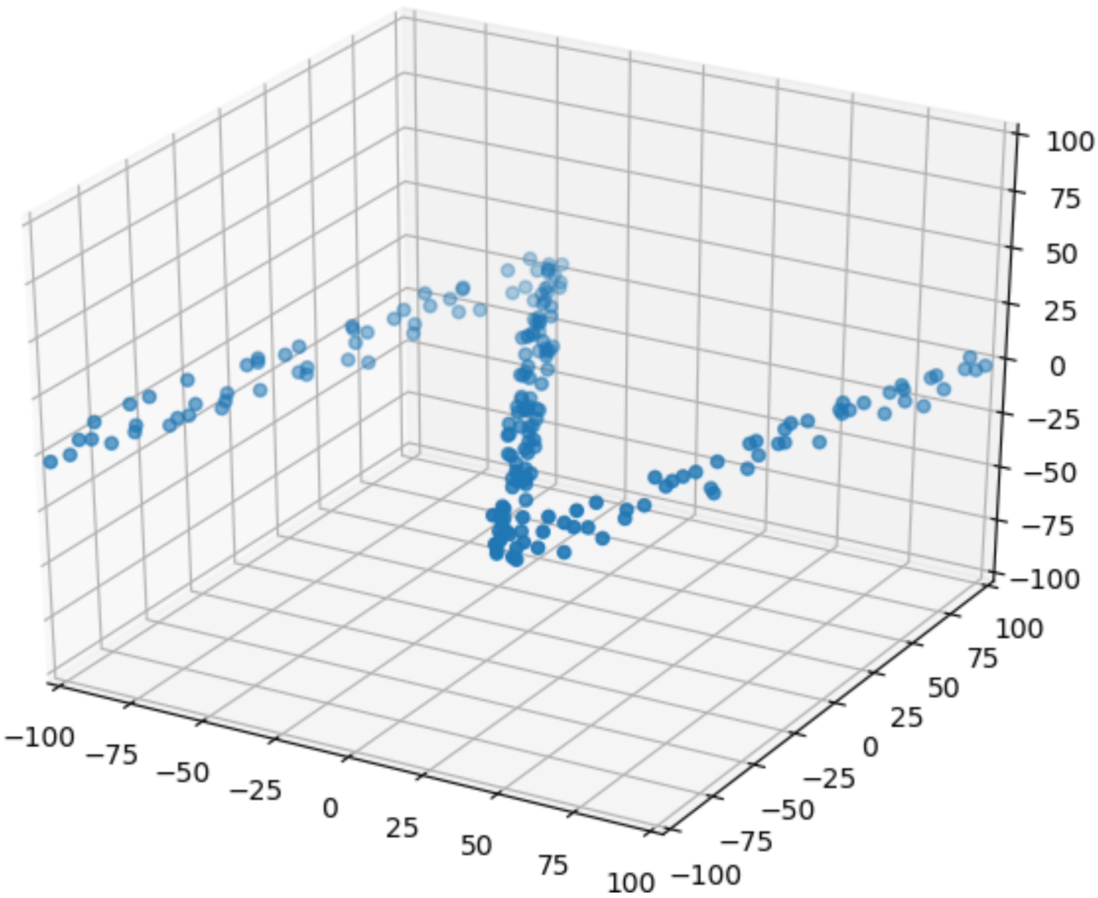
then we reach our conclusion: W is the unique locally and globally optimal solution for the 3-layer neural network function, which is an orthogonal projection onto the subspace spanned by the first k eigenvectors of Σ_{XX}. Thus W's first k columns map X into Σ_{XX}'s eigenspace, resulting in k principal components, which is what we deducted from PCA before.

Programming

Generate N-shaped data

```
def drawN(N):
    """ random generate 3 fold of data and form the `N` shape
        by transformation (rotation or translation).
    """
    x = np.linspace(-100, 100, N)

    delta = np.random.uniform(-10, 10, size=(N,))
```

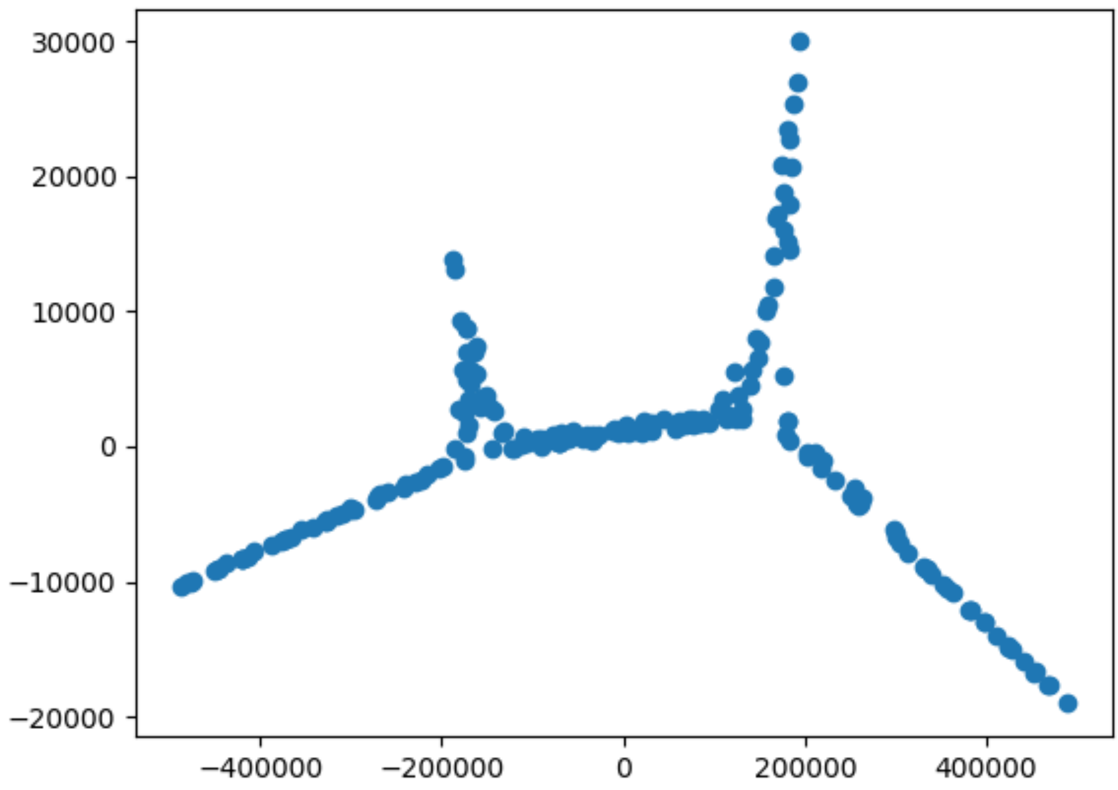
```python
    y1 = 4*x+300
    y1[np.where(x>-50)] = 0
    y3 = 4*x-300
    y3[np.where(x<50)] = 0
    y2 = -2*x
    y2[np.where((x<-50)|(x>50))] = 0
    y = y1 + y2 + y3 + delta

    z = np.random.uniform(-10, 10, size=(N,))

    fig = plt.figure()
    ax = Axes3D(fig)

    ## ratate generated data along direction by theta
    direction = np.array([1,1,0])
    # theta = np.pi/2
    theta = 0
    rmat = rotate_matrix(direction, theta)
    x, y, z = np.dot(rmat, np.array([x,y,z]))
    ax.scatter(x, y, z)

    ## set axis range
    ax.set_xlim([-100, 100])
    ax.set_ylim([-100, 100])
    ax.set_zlim([-100, 100])
    # plt.show()
    plt.savefig('N.png')

    return np.array([x,y,z])
```



reduced N-shaped data

```
def ISOMAP(data, epsilon=30):

    N = data.shape[1]
    A = np.inf*(np.ones((N, N)))
    for i in xrange(N):
        for j in xrange(N):
            d = eculidean(data[:,i], data[:,j])
            if d < epsilon:
                A[i, j] = d

    for i in xrange(N):
        for j in xrange(N):
            for k in xrange(N):
                A[i,j] = min(A[i,j], A[i,k]+A[j,k])

    print(np.any(A==np.inf))
    # print(np.any(A<0))

    A = -0.5*A**2
    H = np.eye(N)-1.0/N*np.ones((N,N))
    B = (H.dot(A)).dot(H)
    # B = data.T.dot(data)
    V,E = np.linalg.eig(B)
```

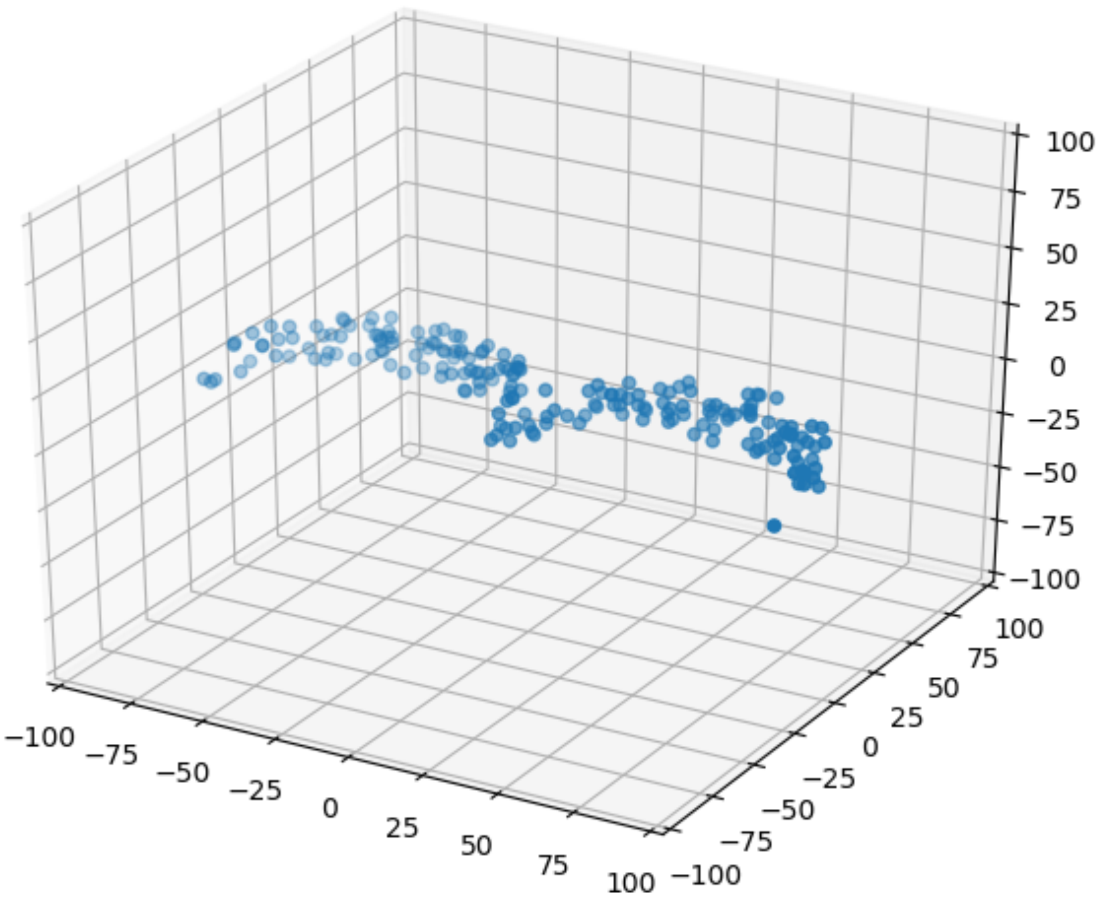
```
    V = np.real(V)
    E = np.real(E)
    reduced = E[:,:2].dot(np.diag(V[:2]))
    # print(reduced)

    return reduced
```



Generate 3-shaped data

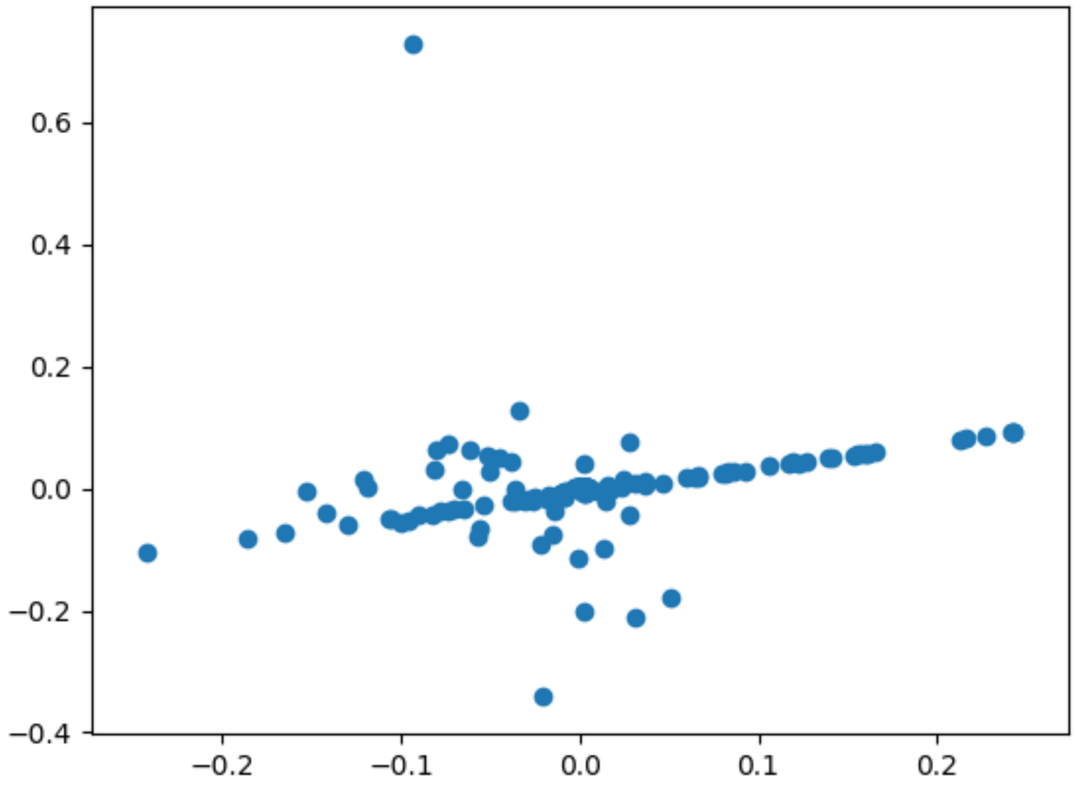
```
def draw3(N):
    """ random generate 4 fold of data and form the `3` shape
    """
    x = np.linspace(-100, 100, N)

    delta = np.random.uniform(-10, 10, size=(N,))
    y1 = np.sqrt(50**2-(x+50)**2)
    y1[np.where(x>0)] = 0
    y2 = np.sqrt(50**2-(x-50)**2)
    y2[np.where(x<=0)] = 0
    # y3 = -np.sqrt(50**2-(x+50)**2)
    # y3[np.where(x>-80)] = 0
    # y4 = -np.sqrt(50**2-(x-50)**2)
    # y4[np.where(x<80)] = 0
    # y = y1 + y2 + y3 + y4 + delta
    y = y1 + y2 + delta
```

```
    y -= 25

    z = np.random.uniform(-10, 10, size=(N,))

    fig = plt.figure(1)
    ax = Axes3D(fig)

    ## ratate generated data along direction by theta
    direction = np.array([1,1,0])
    # theta = np.pi/2
    theta = 0
    rmat = rotate_matrix(direction, theta)
    x, y, z = np.dot(rmat, np.array([x,y,z]))
    ax.scatter(x, y, z)

    ## set axis range
    ax.set_xlim([-100, 100])
    ax.set_ylim([-100, 100])
    ax.set_zlim([-100, 100])
    # plt.show()
    plt.savefig('3.png')

    return np.array([x,y,z])
```



reduced 3-shaped data

```python
def LLE(data, k=5):

    N = data.shape[1]
    p = data.shape[0]
    D = np.zeros((N,N))
    for i in xrange(N):
        for j in xrange(N):
            if i == j:
                D[i,j] = np.inf
            else:
                D[i,j] = eculidean(data[:,i], data[:,j])
    Neighbor = np.zeros((N, k), dtype=np.int)
    for i in xrange(N):
        Neighbor[i,:] = np.argpartition(D[i,:], kth=k-1)[:k]

    # Step 1: Solve W
    Q = np.zeros((N, k, k))
    w = np.zeros((N, k))
    for i in xrange(N):
        diff = data[:,i][:, np.newaxis] - data[:,Neighbor[i,:]]
        Q[i] = diff.T.dot(diff)
        for j in xrange(k):
            Q_inv = np.linalg.inv(Q[i])
            w[i, j] = np.sum(Q_inv[j, :]) / np.sum(Q_inv)

    # Step 2: Solve M
    W = np.zeros((N, N))
    for i in xrange(N):
        W[i, Neighbor[i, :]] = w[i, :]

    M = (np.eye(N)-W).T.dot(np.eye(N)-W)

    V, E = np.linalg.eig(M)
    V = np.real(V)

    reduced = np.real(E[:, -2:])
    return reduced
```



1. Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima, *Neural Networks, Vol. 2, pp. 53-58, 1989*↩