CS498 Applied Machine Learning Homework3

Huamin Zhang && Rongzi Wang

Monday, Feburary 20, 2017

4.10

CIFAR-10 is a dataset of 32x32 images in 10 categories, collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. It is often used to evaluate machine learning algorithms. You can download this dataset from https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html).

(a)

For each category, compute the mean image and the first 20 principal components. Plot the error resulting from representing the images of each category using the first 20 principal components against the category.

Answer:

In [1]:

```
from sklearn.decomposition import PCA
import pickle
import numpy as np
```

In [2]:

```
# read data of all training data
wd = "C:\\Users\\98302\\Desktop\\cifar-10-batches-py\\data_batch_"
batch_file = ["1","2", "3", "4", "5"]
data = []
label = []
for file in batch_file:
    input_file = wd + file
    fo = open(input_file, 'rb')
    dict = pickle.load(fo)
    fo.close()
    data.append(dict['data'])
    label.append(dict['labels'])
```

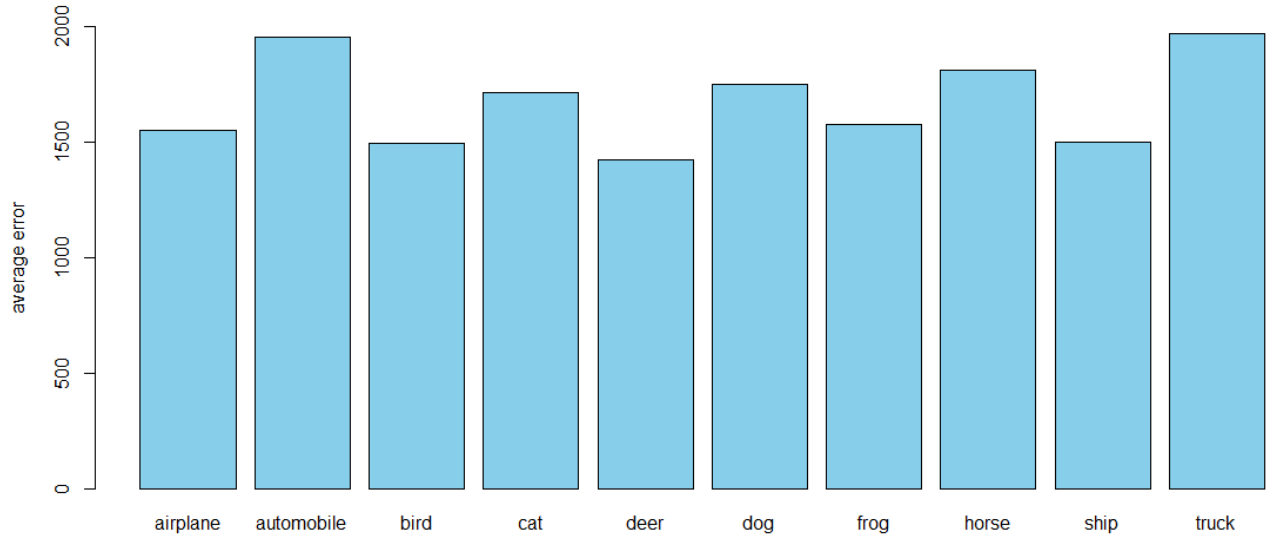
In [3]:

```
#############4.10(a)#####################################################
class_data=[]
class_pca=[]
class_mean_image=[]
error = []
for i in range(0,10):
    x=[]
    y=[]
    x=np.array(x,dtype = np.uint8)
    new_dict = {'data':x,'labels':y}

    #classfier
    for j in range(0,5):
        for k in range(0,10000):
            if label[j][k] == i:
                new_dict['labels'].append(label[j][k])
                new_dict['data'] = np.concatenate((new_dict['data'],data[j][k]))
    new_dict['data'] = new_dict['data'].reshape(new_dict['data'].shape[0]/3072,3072)
    class_data.append(new_dict)

    #do pca
    pca = PCA(copy=True,n_components=20) #pca para
    new_data = pca.fit_transform(class_data[i]['data']) #transform 10000*3072 to 10000*20
    #mean image
    class_mean_image.append(pca.mean_)
    class_pca.append(new_data)

    #Constructing a low-dimensional representation
    represent_data = pca.inverse_transform(new_data)

    #calculate error
    total_dis = 0
    for image in range(5000):
        dist = np.sqrt(np.sum(np.square(represent_data[image] - class_data[i]['data'][image])))
        total_dis = total_dis + dist

    ave_err = total_dis/5000
    error.append(ave_err)
error
```

Out[3]:

```
[1553.1472639290423,
 1955.6153325333514,
 1497.7214125909268,
 1714.0358977724477,
 1422.0067679182389,
 1749.0792671009447,
 1576.0715795335107,
 1811.8674420936179,
 1501.9119009409098,
 1972.6160563358567]
```

In the question, we use the formula $\frac{1}{N}\sum\sqrt{\sum_{i=1}^{3072}(image[m]_i - image[n]_i)^2}$ to calculate the average error between two classes. So the error resulting from representing the images of each category using the first 20 principal components against the category is (1553.15382446685, 1955.6179886516406,1497.7200032304538,1714.0308064609096, 1422.0085695761902,1749.0835367137101, 1576.0729220159319,1811.8635062055494, 1501.9083558302561,1972.6160816894055). The barplot of the error is showed as following.



(b)

Compute the distances between mean images for each pair of classes. Use principal coordinate analysis to make a 2D map of the means of each categories. For this exercise, compute distances by thinking of the images as vectors.

Answer:

In [4]:

```
#############4.10(b)#####################################################

#create distance matrix
D = []
for i in range(10):
    for j in range(10):
        dist = np.sum(np.square(class_mean_image[i] - class_mean_image[j]))
        D.append(dist)
D = np.array(D).reshape(10,10)
#Form A,W and get the eigenvectors andeigenvalues of W
I = np.identity(10)
A = I - 0.1 * np.array([1]*100).reshape(10,10)
W = -0.5 * np.dot(np.dot(A,D),np.transpose(A))
eigval, eigvec = np.linalg.eig(W)
eigval
```

Out[4]:

```
array([  6.11872731e+06,   1.43736291e+06,   5.23337909e+05,
         1.50913516e+05,   1.39065005e+05,  -3.15542556e-10,
         1.22613194e+04,   4.19451647e+04,   2.88933057e+04,
         3.18558605e+04])
```

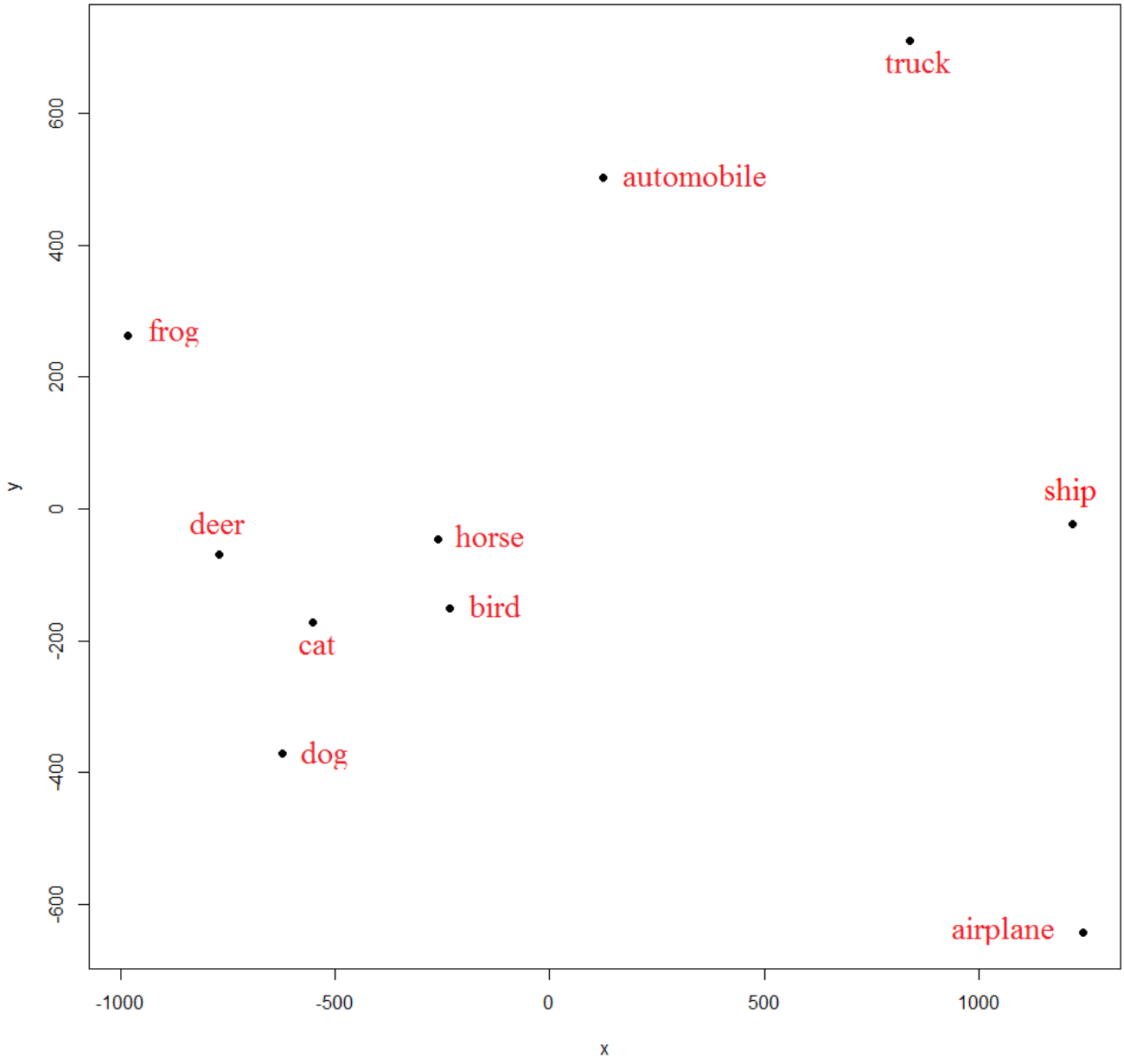
In [5]:

```
#the top left r × r block , r =2
sort1 = eigval.argsort()[-1]
sort2 = eigval.argsort()[-2]
v1 = eigval[sort1]
v2 = eigval[sort2]
vec1 = eigvec[:,sort1]
vec2 = eigvec[:,sort2]
diag = np.array([np.sqrt(v1),0,0,np.sqrt(v2)]).reshape(2,2)
eigvec_r = np.concatenate((vec1,vec2)).reshape(2,10)
V_T = np.dot(diag,eigvec_r)
V = np.transpose(V_T)
V
```

Out[5]:

```
array([[ 1240.98660652,  -642.90599027],
       [  124.59424171,   502.18646509],
       [ -232.62552266,  -151.03722973],
       [ -551.42585341,  -171.53378739],
       [ -770.97022443,   -68.40681696],
       [ -623.48772513,  -371.41540556],
       [ -983.32529277,   262.41028018],
       [ -260.2478262 ,   -46.83221196],
       [ 1218.75239766,   -23.36626178],
       [  837.7491987 ,   710.90095838]])
```

So the V matrix is showed as above. The 2D map of the means of each categories is showed as following and we can find animals are closest to each other.



(c)

Here is another measure of the similarity of two classes. For class A and class B, define $E(A \rightarrow B)$ to be the average error obtained by representing all the images of class A using the mean of class A and the first 20 principal components of class B. Now define the similarity between classes to be $(1/2)(E(A \rightarrow B) + E(B \rightarrow A))$. Use principal coordinate analysis to make a 2D map of the classes. Compare this map to the map in the previous exercise are they different? why?

Answer:

In [6]:

```
#############4.10(c)#####################################################

#create E matrix E(i->j) = E_ij
E = []
for i in range(10):
    for j in range(10):
        pca_i = PCA(copy=True,n_components=20) #pca para
        data_i = pca_i.fit_transform(class_data[i]['data']) #transform 10000*3072 to 10000*20
        pca_j = PCA(copy=True,n_components=20) #pca para
        data_j = pca_j.fit_transform(class_data[j]['data']) #transform 10000*3072 to 10000*20
        pca_i.components_ = pca_j.components_
        represent_data_i = pca_i.inverse_transform(data_i)
        total_dis = 0
        for image in range(5000):
            dist = np.sqrt(np.sum(np.square(represent_data_i[image] - class_data[i]['data'][imag
e]))) 
            total_dis = total_dis + dist
        ave_err = total_dis/5000
        E.append(ave_err)
E = np.array(E).reshape(10,10)
```

In [7]:

```
D2 = []
for i in range(10):
    for j in range(10):
        similarity = 0.5 * (E[i][j] + E[j][i])
        if i == j:
            similarity = 0
        D2.append(similarity)
D2 = np.array(D2).reshape(10,10)
#Form A,W and get the eigenvectors andeigenvalues of W
I2 = np.identity(10)
A2 = I2 - 0.1 * np.array([1]*100).reshape(10,10)
W2 = -0.5 * np.dot(np.dot(A2,D2),np.transpose(A2))
eigval2, eigvec2 = np.linalg.eig(W2)
eigval2
```

Out[7]:

```
array([  4.45452415e+03,  -4.85232501e-13,   2.32606753e+03,
         2.12965284e+03,   1.32628504e+03,   1.76288247e+03,
         1.72046181e+03,   1.66284661e+03,   1.51763770e+03,
         1.49982041e+03])
```

In [8]:

```
# the top left r × r block , r =2
sort_c1 = eigval2.argsort()[-1]
sort_c2 = eigval2.argsort()[-2]
v_c1 = eigval2[sort_c1]
v_c2 = eigval2[sort_c2]
vec_c1 = eigvec2[:,sort_c1]
vec_c2 = eigvec2[:,sort_c2]
diag2 = np.array([np.sqrt(v_c1),0,0,np.sqrt(v_c2)]).reshape(2,2)
eigvec_r2 = np.concatenate((vec_c1,vec_c2)).reshape(2,10)
#compute V
V_T2 = np.dot(diag2,eigvec_r2)
V2 = np.transpose(V_T2)
V2
```

Out[8]:

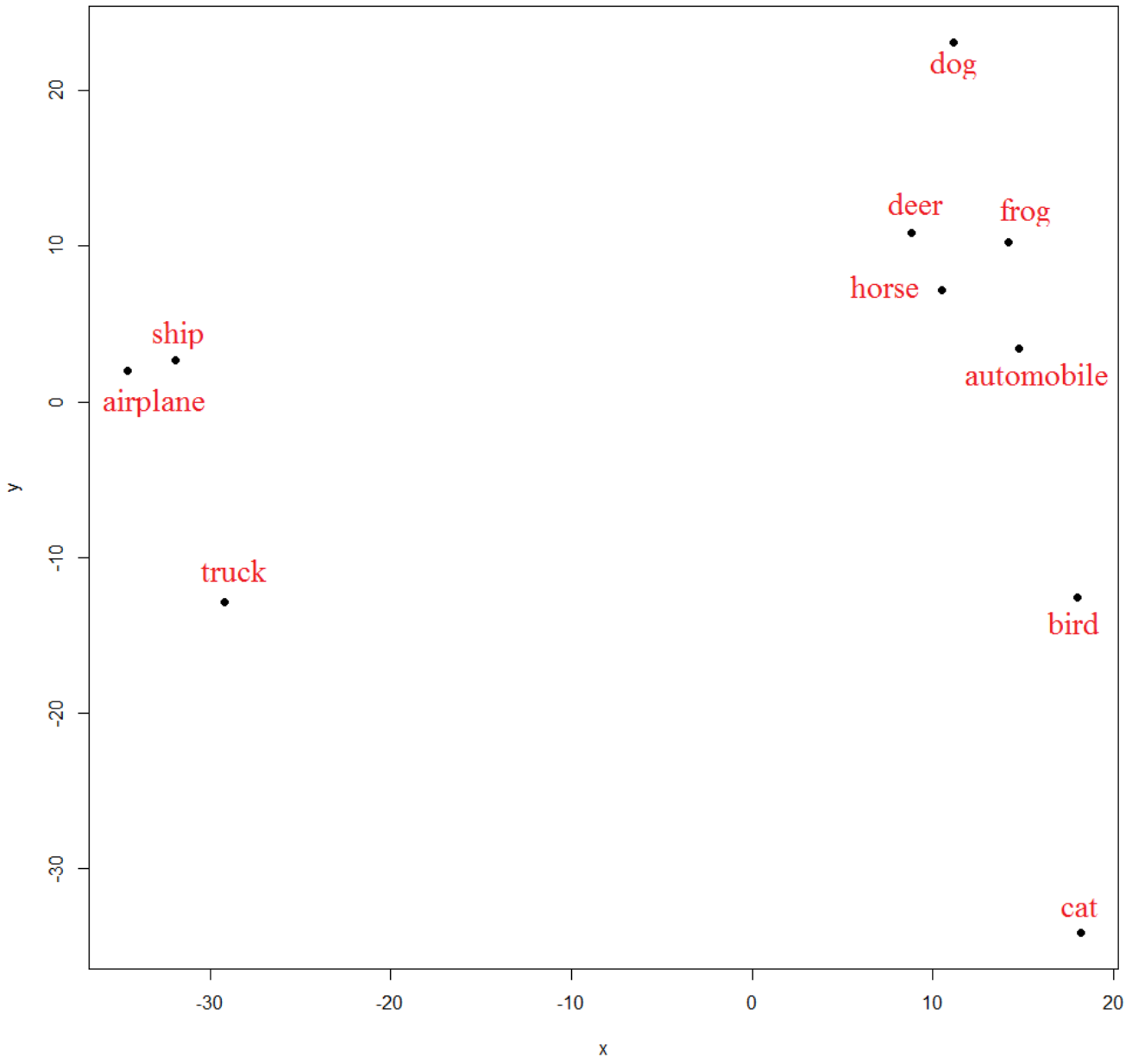
```
array([[-34.59375249,   1.91904878],
       [ 14.82201305,   3.76305174],
       [ 17.99645891, -12.64307444],
       [ 18.12126984, -34.2490615 ],
       [  8.84147255,  10.69525577],
       [ 11.15202304,  23.24730935],
       [ 14.24731381,  10.25328994],
       [ 10.50577661,   7.03159535],
       [-31.92532567,   2.6006685 ],
       [-29.16724964, -12.61808349]])
```

The PCA model from sklearn uses mean and principal components. The mean and pcs are typically set during the call of the function fit which calculates mean and pcs based on a set of data. There are also function transform and inverse transform that take data and convert it to their representation in terms pcs or the opposite direction. The transform functions only behave based on the mean and pcs. The mean and pcs can be set manually. Therefore we can do part c by manually setting the mean and components.

Here we make diagonals of distance matrix to be 0.

So the V matrix for another measure of the similarity of two classes is showed as above. The 2D map of the means of each categories is showed as following.

By comparing the scatter plots we obtain from part b with part c, we observe that they are different. For part b, we start by projecting the distance between each classes into an L2 distance in a larger number of dimensions, and the newly constructed 2D map captures the largest amount of variation from the n-1 dimension space. We observe that classes with different animals are close to each other. However, in part c, the average error is obtained by taking the difference between a mean vector and 20 principal components.The mean vector is not correspond to 20 principal components when we construct a low-dimensional representation. Such difference may not reflect the largest variance between classes, and the constructed 2D map might not very well reflect the similarity/difference between all classes.


dog
deer
frog
horse
ship
automobile
airplane
truck
bird
cat
20
10
0
-10
-20
-30
y
-30
-20
-10
0
10
20
x

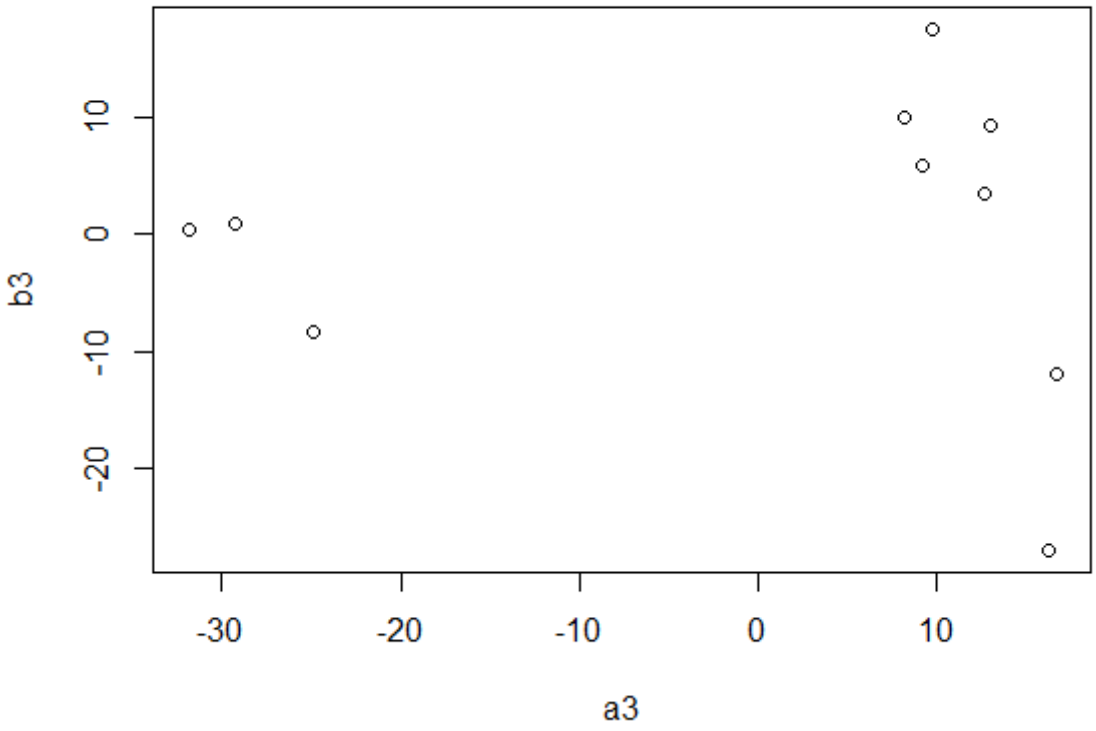
In [14]:

```
#############extra#########################################################
D3 = []
for i in range(10):
    for j in range(10):
        similarity = 0.5 * (E[i][j] + E[j][i])
        D3.append(similarity)
D3 = np.array(D3).reshape(10,10)
#Form A,W and get the eigenvectors andeigenvalues of W
I3 = np.identity(10)
A3 = I3 - 0.1 * np.array([1]*100).reshape(10,10)
W3 = -0.5 * np.dot(np.dot(A3,D3),np.transpose(A3))
eigval3, eigvec3 = np.linalg.eig(W3)
#the top left r × r block , r =2
sort_d1 = eigval3.argsort()[-1]
sort_d2 = eigval3.argsort()[-2]
v_d1 = eigval3[sort_d1]
v_d2 = eigval3[sort_d2]
vec_d1 = eigvec3[:,sort_d1]
vec_d2 = eigvec3[:,sort_d2]
diag3 = np.array([np.sqrt(v_d1),0,0,np.sqrt(v_d2)]).reshape(2,2)
eigvec_r2 = np.concatenate((vec_d1,vec_d2)).reshape(2,10)
#compute V
V_T3 = np.dot(diag3,eigvec_r2)
V3 = np.transpose(V_T3)
V3
```

Out[14]:

```
array([[-31.90196183,   0.32197992],
       [ 12.69776374,   3.46280928],
       [ 16.7461959 , -11.96139016],
       [ 16.2738602 , -26.98157014],
       [  8.24179315,   9.95928546],
       [  9.82406533,  17.53399166],
       [ 13.03246106,   9.19088388],
       [  9.22275975,   5.80136343],
       [-29.276331  ,   0.92405535],
       [-24.86060629,  -8.25140868]])
```

Here we try another diagonals possible(non-zero) and the result is following. We find the result are almost the same. The result may be that the centering matrix is a symmetric and idempotent matrix, which when multiplied with a vector has the same effect as subtracting the mean of the components of the vector from every component so it dose not matter.


b3
10
0
-10
-20
-30
-20
-10
0
10
a3