Assignment10

December 5, 2018

- Name : Joonyoung-Choi
- Student ID: 20112096
- Description: MNIST 0~9 digits classifier
- github: https://github.com/mydream757/Computer_Vision

1. Import libraries

- import needed libraries.

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import numpy.linalg as lin
```

2. Read data from CSV files

- get ready for using MNIST data

```
In [2]: file_data_train = "mnist_train.csv"
        file_data_test  = "mnist_test.csv"

        h_data_train    = open(file_data_train, "r")
        h_data_test     = open(file_data_test, "r")

        data_train      = h_data_train.readlines()
        data_test       = h_data_test.readlines()

        h_data_train.close()
        h_data_test.close()

        size_row    = 28    # height of the image
        size_col    = 28    # width of the image

        num_train   = len(data_train)   # number of training images
        num_test    = len(data_test)    # number of testing images
```

3. Define functions

- the function of data normalizing

```
In [3]: # normalize the values of the input data to be [0, 1]
        def normalize(data):
            data_normalized = (data - min(data)) / (max(data) - min(data))
            return(data_normalized)
```

- the function of computing distance

```
In [4]: # example of distance function between two vectors x and y
        def distance(x, y):

            d = (x - y) ** 2
            s = np.sum(d)
            # r = np.sqrt(s)
            return(s)
```

- the function of 'feature funtion'. this returns feature upto max P.

```
In [5]: # return the feature function vectors upto Max P
        def featureFun(maxP):
            #feature function: numpy.random.normal
            f = np.empty((size_col*size_row,maxP),dtype=float)
            for i in range(maxP):
                f[:,i]=np.random.normal(0,1,size_col*size_row)
            #result = np.dot(f.T,x.T)
            return f
```

4. Ready for test

- make containers which contain MNIST image data

```
In [6]: #make a matrix each column of which represents an images
        list_image_train    = np.empty((num_train, size_row * size_col), dtype=float)
        list_label_train    = np.empty(num_train, dtype=int)

        list_image_test     = np.empty((num_test, size_row * size_col), dtype=float)
        list_label_test     = np.empty(num_test, dtype=int)
```

- parse the data sets

```
In [7]: #parse the data sets
        count = 0
        for line in data_train:

            line_data   = line.split(',')
            label       = line_data[0]
            im_vector   = np.asfarray(line_data[1:])
            im_vector   = normalize(im_vector)

            list_label_train[count]     = label
            list_image_train[count,:]  = im_vector

            count += 1

        count = 0
        for line in data_test:

            line_data   = line.split(',')
            label       = line_data[0]
            im_vector   = np.asfarray(line_data[1:])
            im_vector   = normalize(im_vector)

            list_label_test[count]    = label
            list_image_test[count,:]  = im_vector

            count += 1
```

- compute average images.

```
In [8]: im_average  = np.zeros((10, size_col*size_row), dtype=float)
        im_count    = np.zeros(10, dtype=int)

        for i in range(num_train):
            im_average[list_label_train[i],:] += list_image_train[i,:]
            im_count[list_label_train[i]] += 1
```

- Ready for test

```
In [9]: #Ready for test
        #P can't be over MaxP
        maxP = pow(2, 14)
        print("MaxP: ",maxP)
        #compute feature function of MaxP
        f = featureFun(maxP)
        #compute coefficient using average image vectors and feature function
        bestP = 0
        bestScore = 0
```

```
MaxP:  16384
```

5. Test and Find the best p

- present confusion matrix, compute P and F1 score at each iteration.

```
In [10]: for i in range(13):

             #set different parameter, P
             p = pow(2,i+1)
```

```
print(i+1,"iteration P: ",p)


im_label = np.zeros((10,10), dtype=float)

for j in range(10):
    fx = np.dot(f[:,:p-1].T,im_average.T)
    inverse = lin.pinv(fx.T)
    for r in range(10):

        #label = j : 1, others : -1 for 0,1....9 digits
        if r==j:
            im_label[r,j] = 1
        else:
            im_label[r,j] = -1

#classifiers of the digits 0,1,2....9
coefficient = np.dot(inverse, im_label)

result = np.dot(np.dot(f[:,:p-1].T, list_image_test.T).T, coefficient)
#experiment result
indexOfMax = np.argmax(result,1)

num = 0

confusionMatrix = np.zeros((10,10),dtype=int)
confusionMatrixTable = np.zeros((11,11),dtype=int)
for b in range(10):
    confusionMatrixTable[0][b+1] = b
    confusionMatrixTable[b+1][0] = b

for a in range(indexOfMax.size):
    if indexOfMax[a] == list_label_test[a]:
        confusionMatrixTable[indexOfMax[a]+1][indexOfMax[a]+1] += 1
        confusionMatrix[indexOfMax[a]][indexOfMax[a]] += 1
    elif indexOfMax[a] != list_label_test[a]:
        confusionMatrixTable[list_label_test[a]+1][indexOfMax[a]+1] += 1
        confusionMatrix[list_label_test[a]][indexOfMax[a]] += 1
print(confusionMatrixTable)

precision = np.zeros(10,dtype=float)
recall = np.zeros(10, dtype=float)

for c in range(10):
    if np.sum(confusionMatrix, axis=1)[c]!=0:
        precision[c] = confusionMatrix[c][c]/np.sum(confusionMatrix, axis=1)[c]
    if np.sum(confusionMatrix, axis=0)[c]!=0:
        recall[c] = confusionMatrix[c][c]/np.sum(confusionMatrix, axis=0)[c]
```

```
            precision = np.sum(precision)
            recall = np.sum(recall)
            F1score = 2 * (precision * recall)/(precision + recall)
            if bestScore<F1score:
                bestP = p
                bestScore = F1score
            print("F1 score :", F1score)
```

```
1 iteration P:  2
[[   0    0    1    2    3    4    5    6    7    8    9]
 [   0  536    0    0    0    0    0  444    0    0    0]
 [   1  105    0    0    0    0    0 1030    0    0    0]
 [   2  159    0    0    0    0    0  873    0    0    0]
 [   3  180    0    0    0    0    0  830    0    0    0]
 [   4   86    0    0    0    0    0  896    0    0    0]
 [   5  344    0    0    0    0    0  548    0    0    0]
 [   6   59    0    0    0    0    0  899    0    0    0]
 [   7  480    0    0    0    0    0  548    0    0    0]
 [   8  203    0    0    0    0    0  771    0    0    0]
 [   9   75    0    0    0    0    0  934    0    0    0]]
F1 score : 0.5747861144100976
2 iteration P:  4
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 650 111   0   0  48   0   7   1   8 155]
 [  1 156 569   0   0   2   0   5   0   4 399]
 [  2 232 454   0   0  51   0  11   0   3 281]
 [  3 220 273   0   0  40   0   7   2   4 464]
 [  4 252 162   0   0  72   0  17   1   9 469]
 [  5 294 261   0   0  22   0   5   2   0 308]
 [  6 131 315   1   0  52   0  29   0   9 421]
 [  7 314 324   0   0   5   0   2  16   1 366]
 [  8 344 362   1   0  36   0  13   0   6 212]
 [  9 111 183   0   0  35   0   9   2   5 664]]
F1 score : 1.9084075268881395
3 iteration P:  8
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 319  44  35 164  59  17 169  72  88  13]
 [  1   0 291   3 366 140   6 138  31 127  33]
 [  2  13  36  80 227 113   6 176  98 269  14]
 [  3  10  30  13 396  98  21 156 162  97  27]
 [  4  17  67  14 213 309   5 107 111  76  63]
 [  5  53  61   3 255  61  46  62 227  96  28]
 [  6  16  72  31 163  92   2 435  64  51  32]
 [  7  24  34   4 118  89  40 108 434 149  28]
 [  8  16  66  24 215 113  10  55 139 330   6]
 [  9   1  38  21 335 175  11 122 131  50 125]]
F1 score : 2.9959241362319595
4 iteration P:  16
```

```
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 130   8  99 213   9 233  85  44  63  96]
 [  1   1 188 169 219 110 162  10   8  85 183]
 [  2  13   4 252 235  15 181 118  56  96  62]
 [  3   5   0  89 373   4 293  73  17  82  74]
 [  4   5   6 125 167 190 241  67  46  44  91]
 [  5   9  10  46 214  19 351  33  44  92  74]
 [  6  10   5 111 185  17 177 281  12  55 105]
 [  7   1   1 108 137  15 313  34 256  41 122]
 [  8   2   6 128 185  26 220  27  24 271  85]
 [  9   4   0 110 208  35 269  47  45  57 234]]
F1 score : 3.065962671583593
5 iteration P:  32
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 383   0  38 118  37 173  45  13  73 100]
 [  1   0 715  29   9  85  50 172   1  47  27]
 [  2   7   9 475 126  67  68  39  29 102 110]
 [  3   5   2  99 544  24 106  32  16  46 136]
 [  4   5   1  32  25 493 135  40  18  29 204]
 [  5   6   4  10 121  88 435   9  27  37 155]
 [  6  18   4  26  36  82 164 522   6  22  78]
 [  7   6   9 125  39  63  73  38 437  25 213]
 [  8   6  15  29  96  84 109  10  24 489 112]
 [  9  11   2  25  56 110 112  40  26  15 612]]
F1 score : 5.3866840688170425
6 iteration P:  64
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 525   0  20  90  26 180  38   8  59  34]
 [  1   0 914   8  60  12  17  24   7  88   5]
 [  2   6  13 566 105  54  43  35  38 107  65]
 [  3   1   5  64 650  26 120  16   9  22  97]
 [  4   3  18  15  32 607  84  40  15  29 139]
 [  5   0   4   4 155  52 558  13   8  31  67]
 [  6   8   8   3  14  96 186 577   3  13  50]
 [  7  13   9  72  28  58  32   8 625  39 144]
 [  8   3  18  17  72  50 114  16  26 582  76]
 [  9  17   0   8  27  93  72  16  38  22 716]]
F1 score : 6.488498567530388
7 iteration P:  128
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 648   1  13  25  21 137  44   2  42  47]
 [  1   0 917   6  10   7  73  13   5 101   3]
 [  2   9  30 658  64  52  22  20  33 107  37]
 [  3   2   4  48 719  14  97  14  12  27  73]
 [  4   1  25  10   6 695  54  34   1  40 116]
 [  5   3   4   5 139  56 590  16   4  29  46]
 [  6  15   6   7   3  71 134 667   1  16  38]
 [  7  11  25  13  24  40  36   7 712  37 123]
 [  8   0  12   9  58  30 122  11  26 644  62]
 [  9  19  12   2  32  86  40   5  29  22 762]]
F1 score : 7.130461836522546
8 iteration P:  256
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 693   1   9  16  20  96  50   1  53  41]
 [  1   0 938  17   1   1  88   2   0  87   1]
 [  2  10  40 660  64  47  30  11  22 107  41]
 [  3   1   4  28 730  14  98  14   8  40  73]
 [  4   1  28   9   3 689  51  21   1  48 131]
 [  5   2   4   2 132  38 611  18   4  42  39]
 [  6   8   7   6   2  82  98 714   0  15  26]
 [  7   7  25  20  25  30  36   4 731  31 119]
 [  8   2  15   8  45  36  98  10  20 691  49]
 [  9  13  10   3  20  87  48   3  30  33 762]]
F1 score : 7.332636265487016
9 iteration P:  512
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 706   0   7  13  17 106  39   2  50  40]
 [  1   0 955   2   1   4  86   2   1  81   3]
 [  2   7  27 651  79  52  23  16  28 102  47]
 [  3   2   9  26 744  10  81  15  10  41  72]
 [  4   1  29   4   6 705  44  14   2  53 124]
 [  5   1   7   1 135  34 603  22   6  39  44]
 [  6  13   4   4   5  69 115 723   1  10  14]
 [  7   4  24  14  22  33  32   4 729  38 128]
 [  8   4  15   9  63  33  77  15  22 681  55]
 [  9  11  14   5  28  66  44   2  30  31 778]]
F1 score : 7.384725919520927
10 iteration P:  1024
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 689   1   6  13  17 111  41   2  55  45]
 [  1   0 956   6   2  20  78   4   0  66   3]
 [  2   7  23 671  74  47  18  15  31 101  45]
 [  3   2   6  25 758   8  81  14   9  36  71]
 [  4   1  25   4   3 718  41  21   2  42 125]
 [  5   3   4   0 136  36 604  21   6  39  43]
 [  6  10   5   5   5  67 112 723   2  11  18]
 [  7   4  26  20  13  37  32   3 730  30 133]
 [  8   4  11   6  65  22  80  15  22 691  58]
 [  9  13  17  10  23  60  52   3  30  26 775]]
F1 score : 7.418989265288842
11 iteration P:  2048
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 675   1   6   6  18 126  46   1  59  42]
 [  1   0 978   6   1   9  55   8   0  73   5]
 [  2   7  29 674  58  52  19  20  25  98  50]
 [  3   2   5  25 766   8  75  13  11  35  70]
 [  4   1  20   4   3 728  30  18   2  41 135]
 [  5   7   2   0 138  38 598  22   7  33  47]
 [  6  11   4   3   5  73 115 715   1  10  21]
 [  7   3  30  24   9  29  26   3 725  32 147]
 [  8   3  12   6  68  26  71  15  17 692  64]
 [  9  15  17  11  21  68  48   3  27  28 771]]
F1 score : 7.421665346400326
12 iteration P:  4096
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 681   0   6   8  17 124  51   0  56  37]
 [  1   0 966   3   1  13  55   5   1  85   6]
 [  2   8  30 675  60  48  15  21  26 100  49]
 [  3   2   4  24 769   8  73  13  10  36  71]
 [  4   1  20   3   3 740  27  20   3  37 128]
 [  5   7   2   0 135  38 606  22   8  31  43]
 [  6  16   3   3   5  79 113 717   1   7  14]
 [  7   3  34  24  13  28  16   4 725  32 149]
 [  8   3  11   8  62  33  65  15  18 699  60]
 [  9  15  14  12  24  63  51   2  27  27 774]]
F1 score : 7.445826674396079
13 iteration P:  8192
[[  0   0   1   2   3   4   5   6   7   8   9]
 [  0 682   0   6   9  14 125  53   0  56  35]
 [  1   0 974   6   1  12  51   5   1  80   5]
 [  2   9  26 674  63  47  16  20  25 100  52]
 [  3   2   4  25 774   6  73  13  10  34  69]
 [  4   1  19   4   3 738  29  20   2  38 128]
 [  5   6   1   0 129  38 607  23   9  34  45]
 [  6  11   3   5   5  76 110 722   2   7  17]
 [  7   3  34  25  12  29  15   2 730  31 147]
 [  8   4  12   9  67  29  60  17  19 702  55]
 [  9  17  15   3  24  64  52   2  29  23 780]]
F1 score : 7.4729627698082215
```

- find best P and best F1 score

```
In [11]: print("Best P: ",bestP)
         print("Best score: ",bestScore)
```

```
Best P:  8192
Best score:  7.4729627698082215
```