王　兆東

マルチメディア信号解析

2020年7月20日

Report of Generative Model

In this report I will discuss how I design my program to complete the assigned task, it include a implementation of generative classifier with Gaussian distribution.

1. Environment Introduction

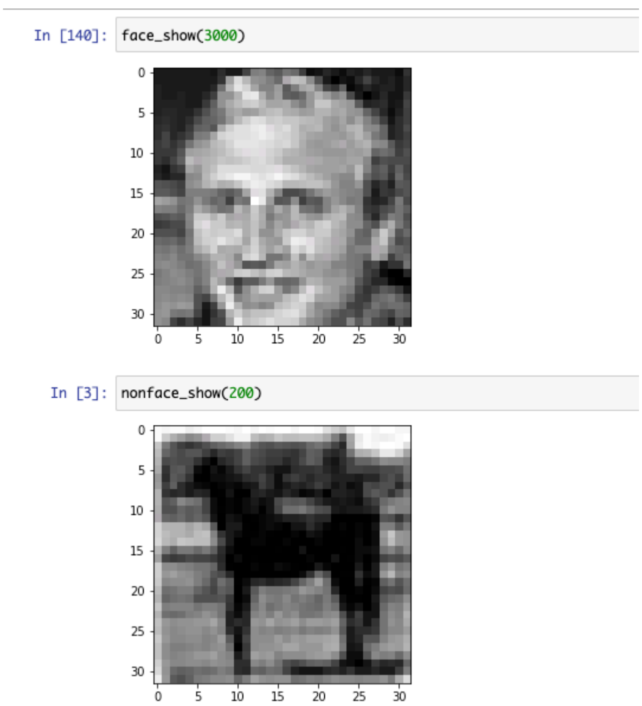
· Programming language: Python3.6

· library would be used: OpenCV [1]

· text book： jupyter [2]

2. Implementation

import the face images and non-face images, and we can check it out in two different cubes of matrixes.



Because the generative model have to calculate the distribution of two butch of images, we don't need to shuffle it, just compute the Gaussian distribution of each set.

```python
def Gaussian_Generative(X):

    Means = X.mean(axis = 0)
    temp = np.zeros((1024, 1024))
    cov = np.cov(X.T, bias=True)
    return Means, cov
```

```python
Face_set = img_set[:int(len(face_cube))*0.5]
Nonface_set = img_set[(int(len(face_cube)))*1.5+1:]

Face_set = img_set[:int(len(face_cube))*0.5]
Nonface_set = img_set[:(int(len(face_cube)))*1.5]
```

```python
Face_Means, Face_cov = Gaussian_Generative(Face_set)
Nonface_Means, Nonface_cov = Gaussian_Generative(Nonface_set)
```

For the prediction function, we use the function of Gaussian distribution,

$$f_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

And calculate with posterior probability.

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

In this formula, the $P(x\,|\,C1)$, $P(x\,|\,C2)$ can calculated from the function of Gaussian distribution.

```python
M = 0.5*(np.linalg.inv(Nonface_cov) - np.linalg.inv(Face_cov))
w = np.linalg.inv(Face_cov) - np.linalg.inv(Nonface_cov)
```

```python
def predict(x, theta):
    x = x.reshape(1024,1)
    result = np.mat(x.T)*np.mat(M)*np.mat(x) + 2*np.matmul(w.T, x)
    #print(result.mean)
    if (np.sum(result)/len(x) > theta):
        predict_label = 1
    else:
        predict_label = 0
    return predict_label
```

In the part of validation, we create a set shuffled from two sets, and the accuracy is 0.966

The confusion matrix is like below:

```
print('--------- P------N-------- ')
print('----T----', TP,'----', TN, '------------')
print('----F----', FP,'----', FN, '------------')
```

```
--------- P------N--------
----T---- 3011 ---- 2782 ------------
----F---- 201 ---- 2 ------------
```

```
accu
```

```
0.9661440960640427
```

Part II PCA method

Using the PCA method to reduce the dimension from size of 1024 into 79.

```
img_set = img_set - img_set.mean(axis = 0)
img_set[:5]
cov_mat = np.cov(img_set.T, bias=True)
eig_val, eig_vecs = np.linalg.eig(cov_mat)
eig_pairs = [(np.abs(eig_val[i]), eig_vecs[:,i]) for i in range(len(eig_val))]
eig_pairs.sort(key = lambda x: x[0], reverse=True)

cond = (eig_val/eig_val.sum()).cumsum()
cond = cond >=0.90
index = cond.argmax()

vector = eig_vecs[:,:index+1]
PCA_result = np.dot(img_set, vector)
PCA_pairs = [(PCA_result[i], label[i]) for i in range(len(PCA_result))]
```

Preparing the PCA Testing set

```
]: random.shuffle(PCA_pairs)

PCA_Test_set = PCA_pairs[int(len(img_set)*0.5)+1:int(len(img_set))]

PCA_Test_label = np.array([PCA_Test_set[i][1] for i in range(len(PCA_Test_set))])
PCA_Test_data = np.array([PCA_Test_set[i][0] for i in range(len(PCA_Test_set))])
```

Here is the final result, the accuracy is 0.938, and the confuse matrix is as bellow:

```
]: TP, TN, FN, FP, accu = compute_accuracy(PCA_Face_cov, PCA_Nonface_cov, PCA_Test_data, PCA_Test_label, 0)

]: print('--------- P------N-------- ')
   print('----T----', TP,'----', TN, '------------')
   print('----F----', FP,'----', FN, '------------')

   --------- P------N--------
   ----T---- 2719 ---- 2909 ------------
   ----F---- 51 ---- 317 ------------

]: accu

]: 0.9386257505003336
```

Conclusion:

The PCA reduce the dimension of image vector but keep the essential features, which lead the accuracy a little bit down but reduce the memory and consuming of computation.