EE5907 Pattern Recognition

CA2

A0263693U

Wang Songtao

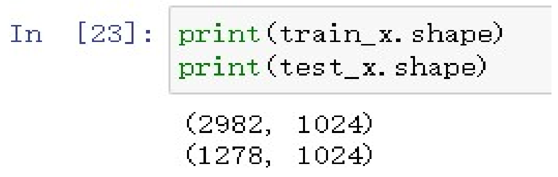
Introduction:

This is the report for CA2 of EE5907 Pattern Recognition. The task is to achieve face recognition system with different algorithms including PCA, NN, LDA, SVM, GMM CNN.
In my assignment, all the code were programmed in Python. In my experiments, I implemented two mandatory tasks which are PCA and LDA. Besides I implemented SVM with Libsvm and CNN. Four parts of code are stored in four independent .ipynb files.

Dataset

Data selection
The datasets I used in this experiment is CMU PIE datasets plus selfie image. Among 68 different group of images, I just selected first 25 group because I think the selection of groups has no influence on face recognition task. Also, I added selfie image into dataset.
There are 170 images in each group and 10 selfie images are included, so the whole dataset contains 4260 images which belong to 26 different classes. What's more, all the images are in form 32*32 pixels resulting in 1024 features for each sample.

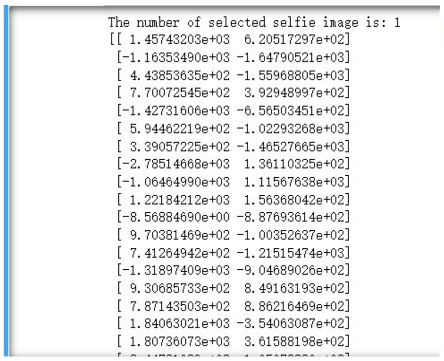
Data split
Next step is to split dataset into training part and testing part. I split origin dataset into 70% for training and 30% for testing. Therefore, the training dataset contains 2982 images and testing dataset contains 1278 images.
The shape of split dataset is shown below.



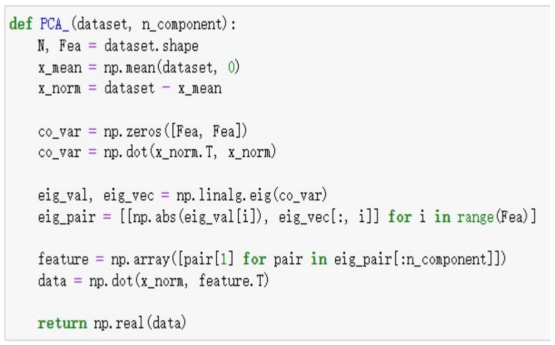
```
In  [23]: print(train_x.shape)
          print(test_x.shape)

          (2982, 1024)
          (1278, 1024)
```

PCA

Reduction dimension
I implemented PCA from scratch, and the code and partial result of processed dataset with 2-demension are shown below.



```
The number of selected selfie image is: 1
[[ 1.45743203e+03  6.20517297e+02]
 [-1.16353490e+03 -1.64790521e+03]
 [ 4.43853635e+02 -1.55968805e+03]
 [ 7.70072545e+02  3.92948997e+02]
 [-1.42731606e+03 -6.56503451e+02]
 [ 5.94462219e+02 -1.02293268e+03]
 [ 3.39057225e+02 -1.46527665e+03]
 [-2.78514668e+03  1.36110325e+02]
 [-1.06464990e+03  1.11567638e+03]
 [ 1.22184212e+03  1.56368042e+02]
 [-8.56884690e+00 -8.87693614e+02]
 [ 9.70381469e+02 -1.00352637e+02]
 [ 7.41264942e+02 -1.21515474e+03]
 [-1.31897409e+03 -9.04689026e+02]
 [ 9.30685733e+02  8.49163193e+02]
 [ 7.87143503e+02  8.86216469e+02]
 [ 1.84063021e+03 -3.54063087e+02]
 [ 1.80736073e+03  3.61588198e+02]
```

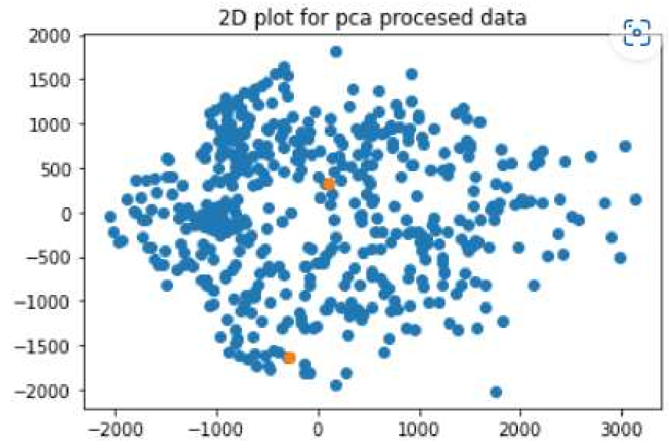


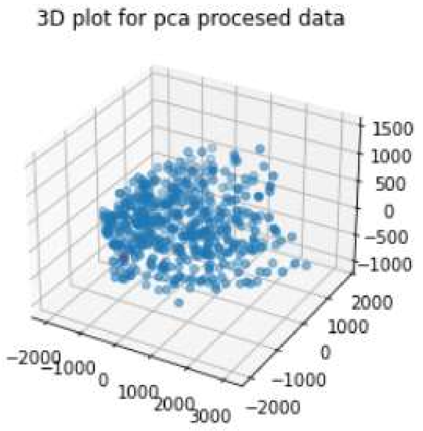
```python
def PCA_(dataset, n_component):
    N, Fea = dataset.shape
    x_mean = np.mean(dataset, 0)
    x_norm = dataset - x_mean

    co_var = np.zeros([Fea, Fea])
    co_var = np.dot(x_norm.T, x_norm)

    eig_val, eig_vec = np.linalg.eig(co_var)
    eig_pair = [[np.abs(eig_val[i]), eig_vec[:, i]] for i in range(Fea)]

    feature = np.array([pair[1] for pair in eig_pair[:n_component]])
    data = np.dot(x_norm, feature.T)

    return np.real(data)
```

Visualization

Randomly sample 500 images from the CMU PIE training set and my own photos. And then the projected image data are plotted in following figure.

```
The number of selected selfie image is: 2
```

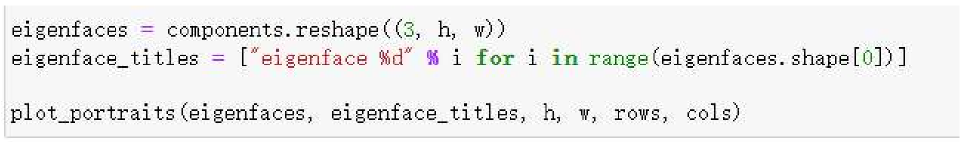




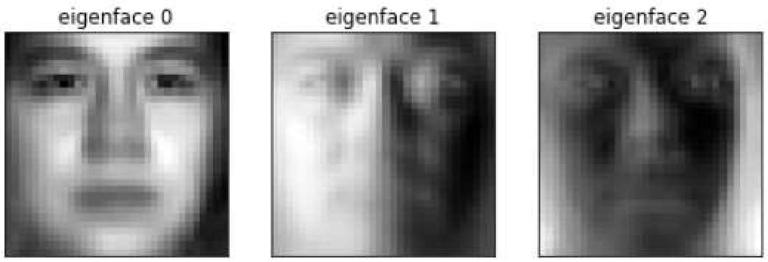
The 2D figure and 3D figure are shown respectively and selfie image data are shown in yellow in 2D figure and red in 3D figure.

Based on the visualization figures, we can easily find that the projected data are all located together and we can barely see the classification situation from the figure directly. Therefore, the PCA process indeed make origin dataset loss some information which would make further classification process difficult or less accurate as normal.

Three eigenfaces used for dimensionality reduction are shown below.



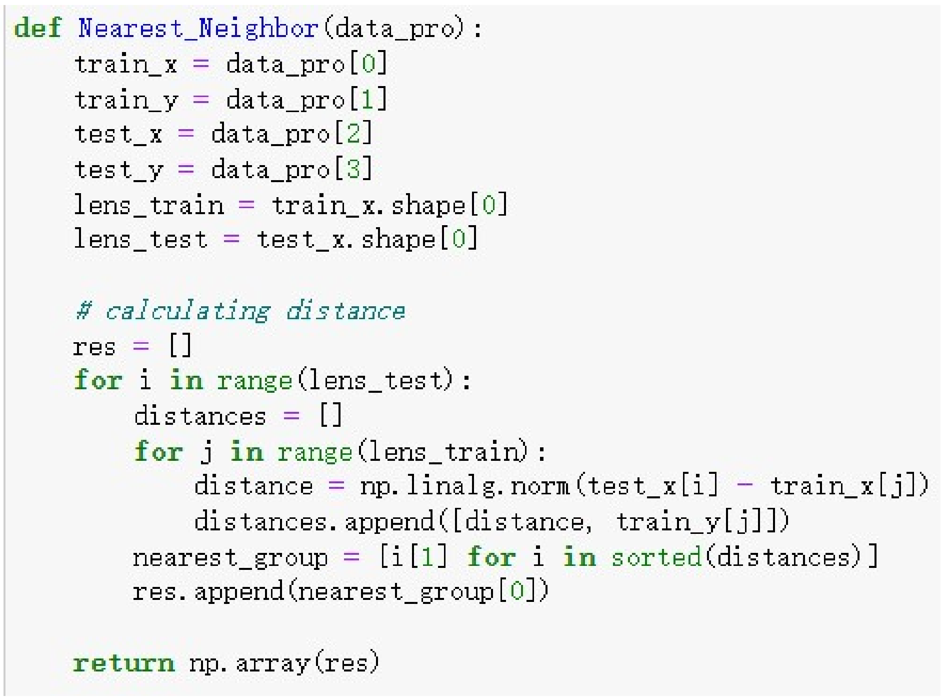
```
eigenfaces = components.reshape((3, h, w))
eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]

plot_portraits(eigenfaces, eigenface_titles, h, w, rows, cols)
```



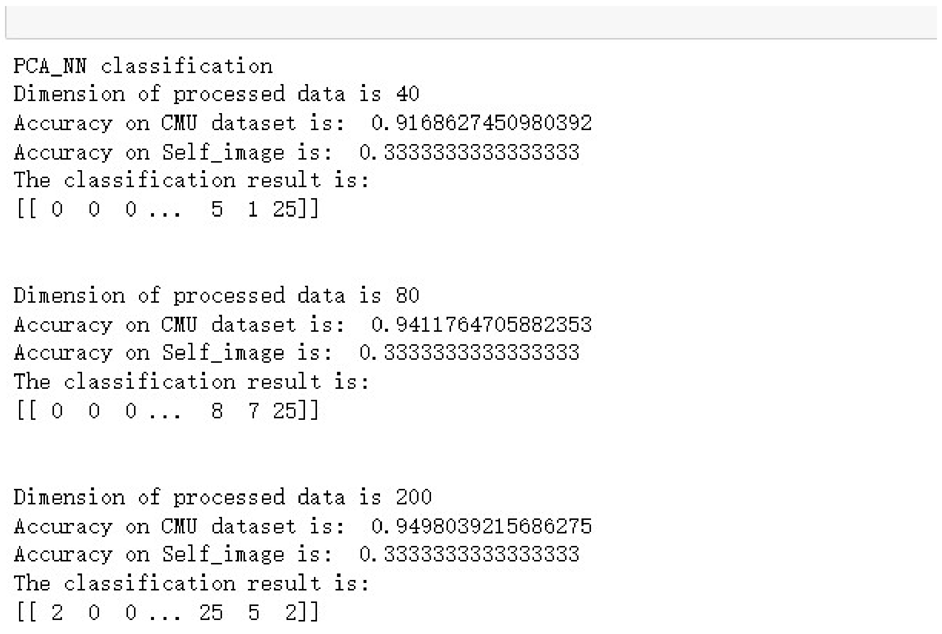
The eigenfaces are another form of their corresponding eigenvector and eigenvalue. The first eigenface is much clearer and more human-like than the last two because based on PCA process, the first eigenvector contain the most information of all dataset which means the most obvious image feature is collected into first eigenvector and eigenface.

Classification

I programmed Nearest Neighbor algorithms and implemented it into PCA processed data for classification task. The NN algorithm and NN_PCA result are shown below.



```
def Nearest_Neighbor(data_pro):
    train_x = data_pro[0]
    train_y = data_pro[1]
    test_x = data_pro[2]
    test_y = data_pro[3]
    lens_train = train_x.shape[0]
    lens_test = test_x.shape[0]

    # calculating distance
    res = []
    for i in range(lens_test):
        distances = []
        for j in range(lens_train):
            distance = np.linalg.norm(test_x[i] - train_x[j])
            distances.append([distance, train_y[j]])
        nearest_group = [i[1] for i in sorted(distances)]
        res.append(nearest_group[0])

    return np.array(res)
```

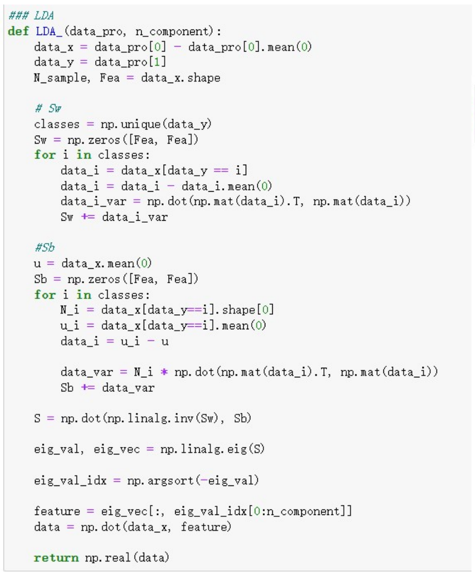


```
PCA_NN classification
Dimension of processed data is 40
Accuracy on CMU dataset is:  0.9168627450980392
Accuracy on Self_image is:  0.3333333333333333
The classification result is:
[[ 0  0  0 ...  5  1 25]]


Dimension of processed data is 80
Accuracy on CMU dataset is:  0.9411764705882353
Accuracy on Self_image is:  0.3333333333333333
The classification result is:
[[ 0  0  0 ...  8  7 25]]


Dimension of processed data is 200
Accuracy on CMU dataset is:  0.9498039215686275
Accuracy on Self_image is:  0.3333333333333333
The classification result is:
[[ 2  0  0 ... 25  5  2]]
```

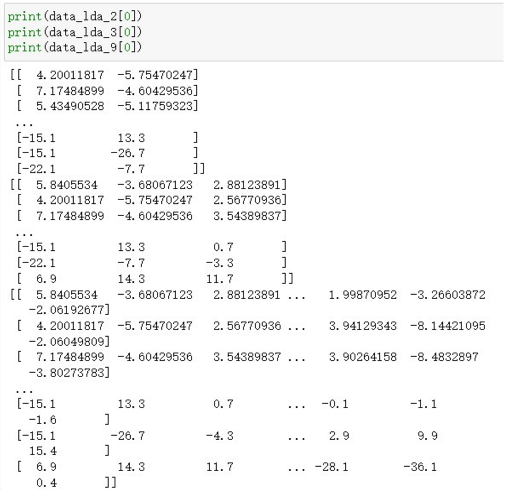
From the results we can see that the accuracy of NN classification is fine on CMU image but low on selfie image. The reason might be the number of training image. NN classification requires a relatively big amount of training data, however, the selfie images have only 7 images on training dataset.
Furthermore, the accuracy is increasing gradually with increasing of dimensionality which is result from more information contained in high-dimension dataset.

LDA

Visualization
I programmed LDA algorithm from scratch and implemented it on origin dataset to reduce its dimension into 2, 3, 9.
The implementation of algorithm and the result of dimension reduction are shown in the following figures.



```python
### LDA
def LDA_(data_pro, n_component):
    data_x = data_pro[0] - data_pro[0].mean(0)
    data_y = data_pro[1]
    N_sample, Fea = data_x.shape

    # Sw
    classes = np.unique(data_y)
    Sw = np.zeros([Fea, Fea])
    for i in classes:
        data_i = data_x[data_y == i]
        data_i = data_i - data_i.mean(0)
        data_i_var = np.dot(np.mat(data_i).T, np.mat(data_i))
        Sw += data_i_var

    #Sb
    u = data_x.mean(0)
    Sb = np.zeros([Fea, Fea])
    for i in classes:
        N_i = data_x[data_y==i].shape[0]
        u_i = data_x[data_y==i].mean(0)
        data_i = u_i - u

        data_var = N_i * np.dot(np.mat(data_i).T, np.mat(data_i))
        Sb += data_var

    S = np.dot(np.linalg.inv(Sw), Sb)

    eig_val, eig_vec = np.linalg.eig(S)

    eig_val_idx = np.argsort(-eig_val)

    feature = eig_vec[:, eig_val_idx[0:n_component]]
    data = np.dot(data_x, feature)

    return np.real(data)
```

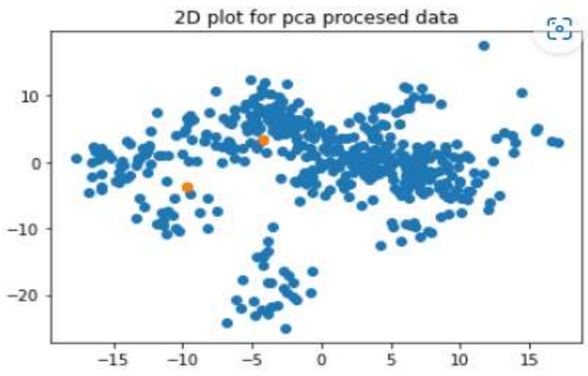


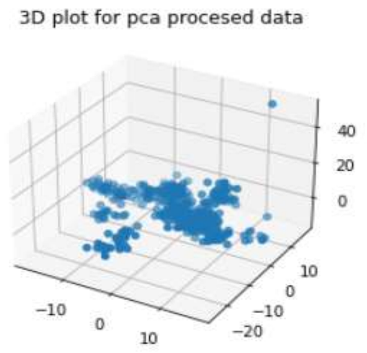
```python
print(data_lda_2[0])
print(data_lda_3[0])
print(data_lda_9[0])
```

```
[[  4.20011817  -5.75470247]
 [  7.17484899  -4.60429536]
 [  5.43490528  -5.11759323]
 ...
 [-15.1         13.3       ]
 [-15.1        -26.7       ]
 [-22.1         -7.7       ]]
[[  5.8405534   -3.68067123   2.88123891]
 [  4.20011817  -5.75470247   2.56770936]
 [  7.17484899  -4.60429536   3.54389837]
 ...
 [-15.1         13.3          0.7       ]
 [-22.1         -7.7         -3.3       ]
 [  6.9         14.3         11.7       ]]
[[  5.8405534   -3.68067123   2.88123891 ...   1.99870952  -3.26603872
   -2.06192677]
 [  4.20011817  -5.75470247   2.56770936 ...   3.94129343  -8.14421095
   -2.06049809]
 [  7.17484899  -4.60429536   3.54389837 ...   3.90264158  -8.4832897
   -3.80273783]
 ...
 [-15.1         13.3          0.7        ...  -0.1         -1.1
   -1.6       ]
 [-15.1        -26.7         -4.3        ...   2.9          9.9
   15.4       ]
 [  6.9         14.3         11.7        ... -28.1        -36.1
    0.4       ]]
```

The randomly selected 500 samples are processed with LDA and are transformed into 2D and 3D data points.
The projected 2D and 3D figures are shown below.

```
The number of selfie image is:  2
```

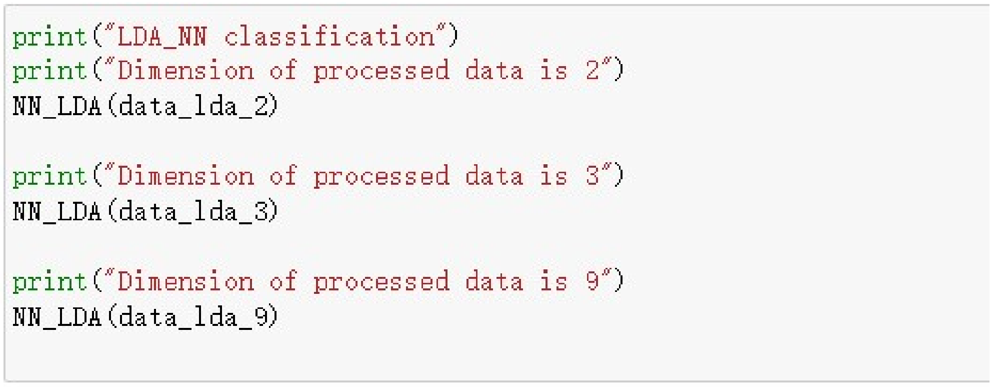




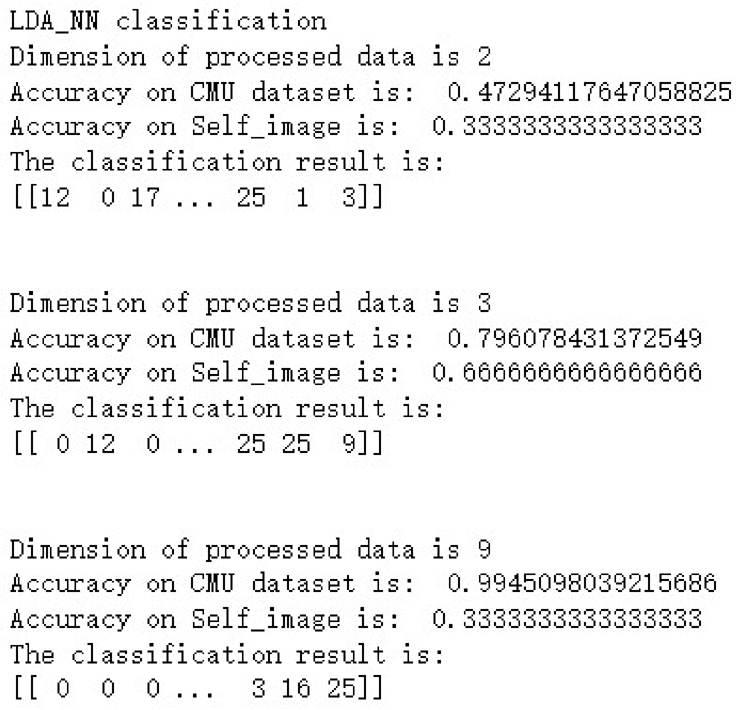
The yellow points are selfie image data selected in 500 random samples in 2D figure.
And the red points are selfie image data in 3D figure.
Because the data points are centered together, the selfie image data is not so obvious.

Besides, from the figures we can see that compared with PCA, the supervised algorithm LDA processed data has the inclination of being centered into several clusters which is the result of the labels of samples.

Classification
I also implemented nearest neighbor for classification. The results are shown below.



```
print("LDA_NN classification")
print("Dimension of processed data is 2")
NN_LDA(data_lda_2)

print("Dimension of processed data is 3")
NN_LDA(data_lda_3)

print("Dimension of processed data is 9")
NN_LDA(data_lda_9)
```



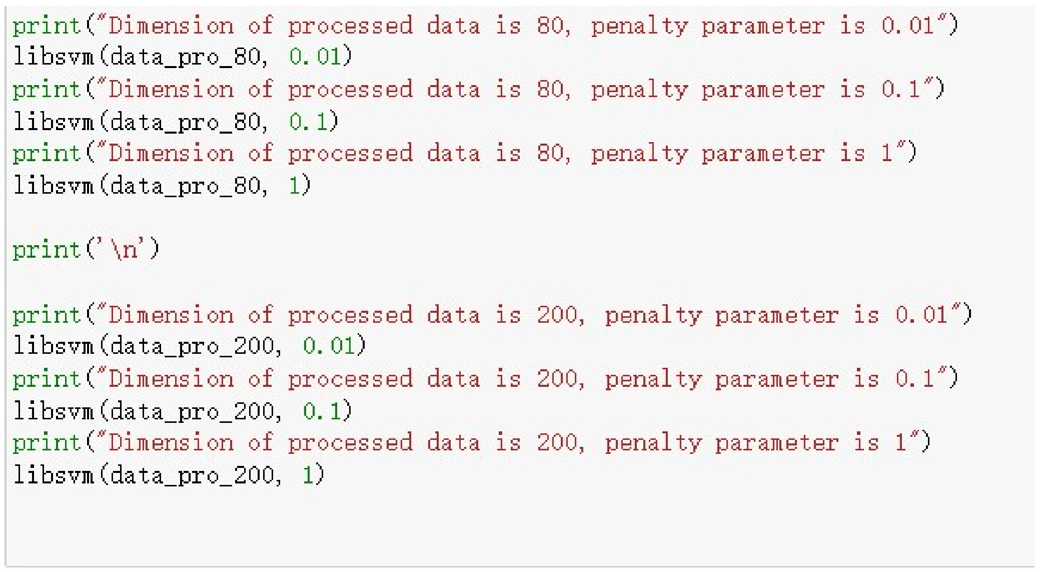
```
LDA_NN classification
Dimension of processed data is 2
Accuracy on CMU dataset is:  0.47294117647058825
Accuracy on Self_image is:  0.3333333333333333
The classification result is:
[[12  0 17 ... 25  1  3]]


Dimension of processed data is 3
Accuracy on CMU dataset is:  0.796078431372549
Accuracy on Self_image is:  0.6666666666666666
The classification result is:
[[ 0 12  0 ... 25 25  9]]


Dimension of processed data is 9
Accuracy on CMU dataset is:  0.9945098039215686
Accuracy on Self_image is:  0.3333333333333333
The classification result is:
[[ 0  0  0 ...  3 16 25]]
```

Following the increasing the dimension, the accuracy of classification has an obvious progress from 0.47 in 2-dimension to 0.99 in 9-dimension. From the results, we can know that 9 dimensionality covers almost all of information of origin data based on LDA process. The accuracy on selfie image is also low, which I think it's the result of the defect of Nearest Neighbor algorithm.

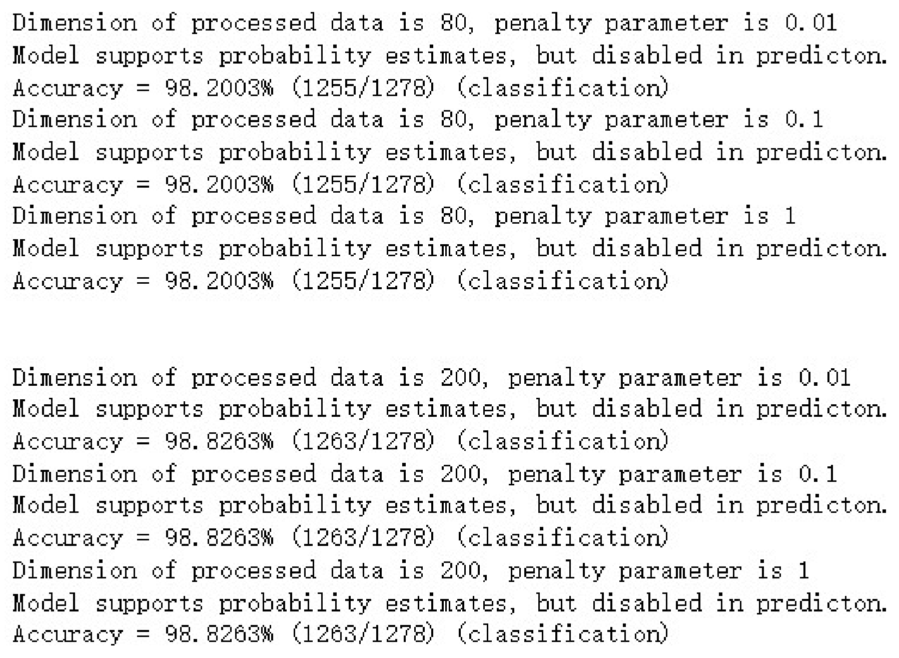
SVM

classification

For SVM task, I used PCA processed datasets with 80 dimension and 200 dimensions. The implementation of svm is based on libsvm. The results with different parameters are shown below.



```
print("Dimension of processed data is 80, penalty parameter is 0.01")
libsvm(data_pro_80, 0.01)
print("Dimension of processed data is 80, penalty parameter is 0.1")
libsvm(data_pro_80, 0.1)
print("Dimension of processed data is 80, penalty parameter is 1")
libsvm(data_pro_80, 1)

print('\n')

print("Dimension of processed data is 200, penalty parameter is 0.01")
libsvm(data_pro_200, 0.01)
print("Dimension of processed data is 200, penalty parameter is 0.1")
libsvm(data_pro_200, 0.1)
print("Dimension of processed data is 200, penalty parameter is 1")
libsvm(data_pro_200, 1)
```

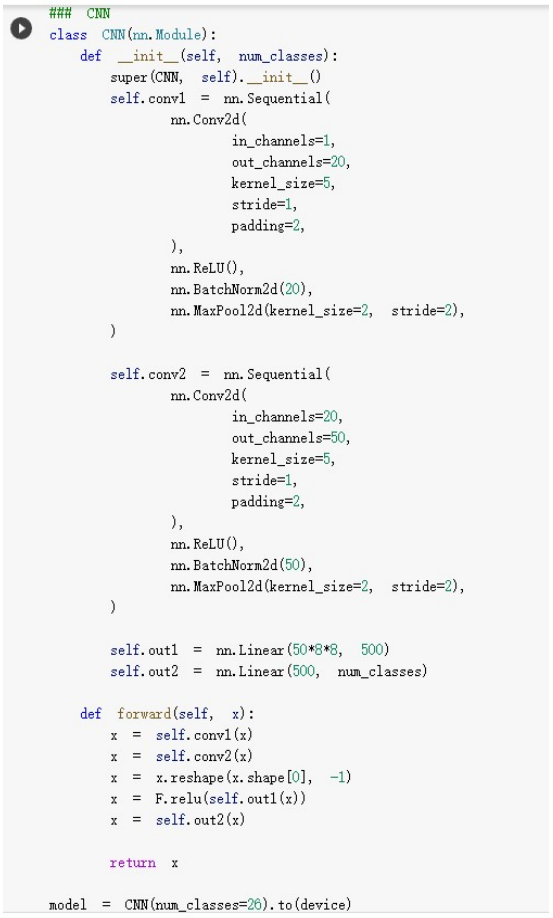


```
Dimension of processed data is 80, penalty parameter is 0.01
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.2003% (1255/1278) (classification)
Dimension of processed data is 80, penalty parameter is 0.1
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.2003% (1255/1278) (classification)
Dimension of processed data is 80, penalty parameter is 1
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.2003% (1255/1278) (classification)


Dimension of processed data is 200, penalty parameter is 0.01
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.8263% (1263/1278) (classification)
Dimension of processed data is 200, penalty parameter is 0.1
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.8263% (1263/1278) (classification)
Dimension of processed data is 200, penalty parameter is 1
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.8263% (1263/1278) (classification)
```

The accuracy got a little improvement with increasing of dimension because even though data with 200 dimension contains more information, these parts are quite trivial. The penalty parameters seem to have no influence on results. The reason might be that the accuracy is really close to 100% and this parameter for linear svm might have more influence on low accuracy part.

CNN

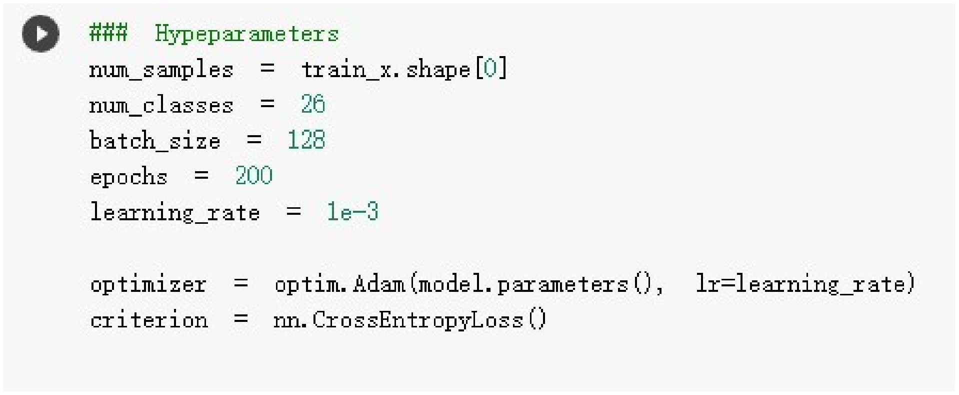
In this part, I used Pytorch to build the CNN structure.
The detail of CNN structure is shown below.



```
### CNN
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
                nn.Conv2d(
                        in_channels=1,
                        out_channels=20,
                        kernel_size=5,
                        stride=1,
                        padding=2,
                ),
                nn.ReLU(),
                nn.BatchNorm2d(20),
                nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.conv2 = nn.Sequential(
                nn.Conv2d(
                        in_channels=20,
                        out_channels=50,
                        kernel_size=5,
                        stride=1,
                        padding=2,
                ),
                nn.ReLU(),
                nn.BatchNorm2d(50),
                nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.out1 = nn.Linear(50*8*8, 500)
        self.out2 = nn.Linear(500, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.reshape(x.shape[0], -1)
        x = F.relu(self.out1(x))
        x = self.out2(x)

        return x

model = CNN(num_classes=26).to(device)
```

I adopted two 2D convolution layers which are all followed by ReLU as activation function. For two convolution layers, the kernel size is both (5, 5), stride is 1, padding is 2.
After that, I added a batch normalization layer and then followed by a max pooling layer. At last, I used two linear layers.

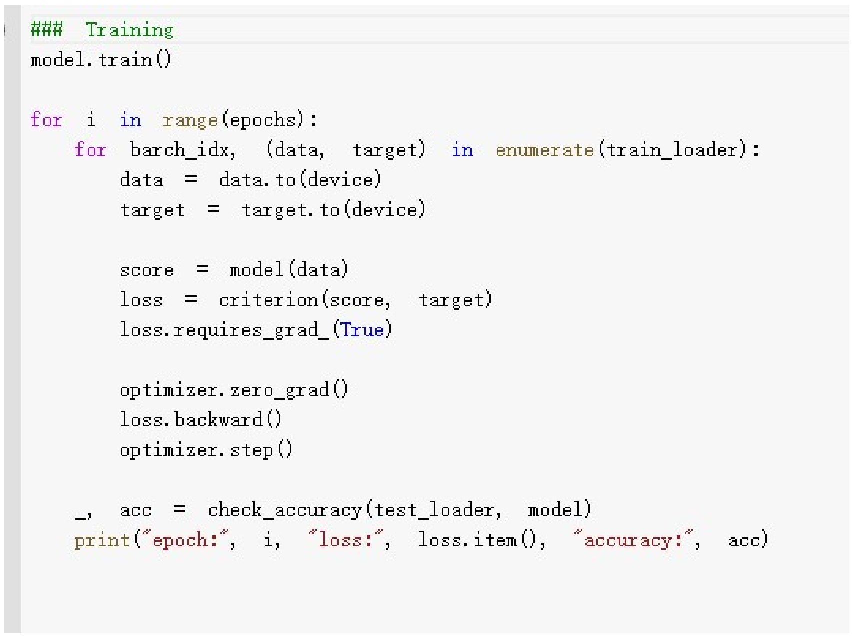
The hype parameters are shown.



```
### Hypeparameters
num_samples = train_x.shape[0]
num_classes = 26
batch_size = 128
epochs = 200
learning_rate = 1e-3

optimizer = optim.Adam(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()
```

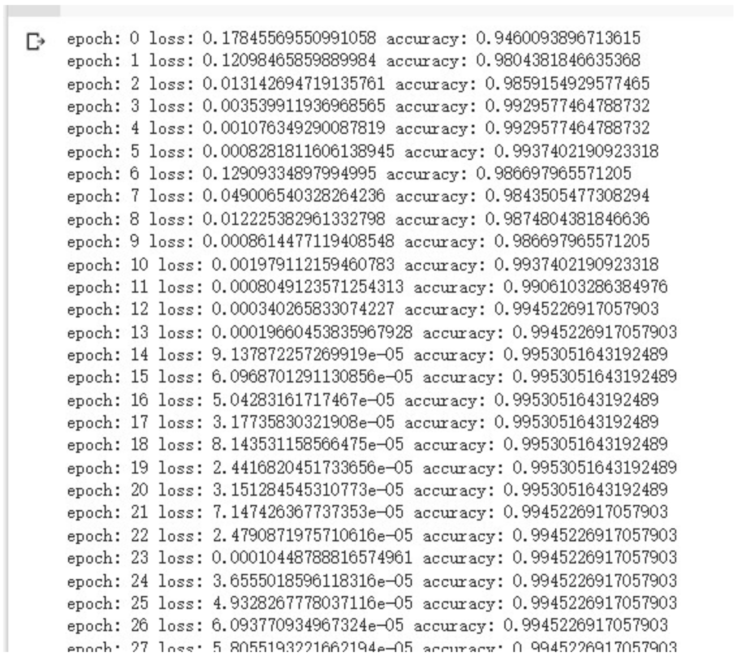
The number of classes are decided by dataset which contains 25 groups of CMU image data and one group of my selfie image. In this experiment, I adopted batch training and batch size is 128. The optimizer I used is Adam and its learning rate is 1e-3. The loss function is cross entropy given that the problem I'm handling with is a classification problem.

The training process is shown.



```
### Training
model.train()

for i in range(epochs):
    for barch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device)
        target = target.to(device)

        score = model(data)
        loss = criterion(score, target)
        loss.requires_grad_(True)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    _, acc = check_accuracy(test_loader, model)
    print("epoch:", i, "loss:", loss.item(), "accuracy:", acc)
```

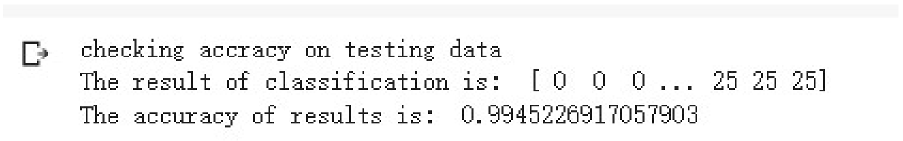
The value of loss function and accuracy on testing data for each training epoch is shown below.



```
epoch: 0 loss: 0.17845569550991058 accuracy: 0.9460093896713615
epoch: 1 loss: 0.12098465859889984 accuracy: 0.9804381846635368
epoch: 2 loss: 0.013142694719135761 accuracy: 0.9859154929577465
epoch: 3 loss: 0.003539911936968565 accuracy: 0.9929577464788732
epoch: 4 loss: 0.001076349290087819 accuracy: 0.9929577464788732
epoch: 5 loss: 0.0008281811606138945 accuracy: 0.9937402190923318
epoch: 6 loss: 0.12909334897994995 accuracy: 0.986697965571205
epoch: 7 loss: 0.049006540328264236 accuracy: 0.9843505477308294
epoch: 8 loss: 0.012225382961332798 accuracy: 0.9874804381846636
epoch: 9 loss: 0.0008614477119408548 accuracy: 0.986697965571205
epoch: 10 loss: 0.001979112159460783 accuracy: 0.9937402190923318
epoch: 11 loss: 0.0008049123571254313 accuracy: 0.9906103286384976
epoch: 12 loss: 0.000340265833074227 accuracy: 0.9945226917057903
epoch: 13 loss: 0.00019660453835967928 accuracy: 0.9945226917057903
epoch: 14 loss: 9.137872257269919e-05 accuracy: 0.9953051643192489
epoch: 15 loss: 6.0968701291130856e-05 accuracy: 0.9953051643192489
epoch: 16 loss: 5.04283161717467e-05 accuracy: 0.9953051643192489
epoch: 17 loss: 3.17735830321908e-05 accuracy: 0.9953051643192489
epoch: 18 loss: 8.143531158566475e-05 accuracy: 0.9953051643192489
epoch: 19 loss: 2.4416820451733656e-05 accuracy: 0.9953051643192489
epoch: 20 loss: 3.151284545310773e-05 accuracy: 0.9953051643192489
epoch: 21 loss: 7.147426367737353e-05 accuracy: 0.9945226917057903
epoch: 22 loss: 2.4790871975710616e-05 accuracy: 0.9945226917057903
epoch: 23 loss: 0.00010448788816574961 accuracy: 0.9945226917057903
epoch: 24 loss: 3.6555018596118316e-05 accuracy: 0.9945226917057903
epoch: 25 loss: 4.9328267778037116e-05 accuracy: 0.9945226917057903
epoch: 26 loss: 6.093770934967324e-05 accuracy: 0.9945226917057903
epoch: 27 loss: 5.8055193221662194e-05 accuracy: 0.9945226917057903
```

Based on the training results, we can see that the loss function declined really fast and the accuracy has reached a high level after 20 epochs training.

The final result is 99.45% accuracy on testing dataset.



```
checking accracy on testing data
The result of classification is:  [ 0  0  0 ... 25 25 25]
The accuracy of results is:  0.9945226917057903
```

Conclusion

From all these experiments, I implemented different methods to perform classification task including PCA with NN, LDA with NN, SVM and CNN.

For two data processed methods PCA and LDA, PCA processing with Nearest Neighbor methods could get 94.98% accuracy on test datasets. LDA processing with Nearest Neighbor methods could get 99.45% accuracy. Therefore, in this case, LDA got better performance on classification.
In terms of feature extraction, PCA and LDA can both achieve that goal greatly.

SVM could get 98.82% accuracy with PCA processed training data.
CNN could get 99.45% accuracy easily. In my experiment, even though I didn't spend much time on tuning parameters, the result is extremely perfect.

Consequently, among all algorithms mentioned above, CNN got the better performance.