Cat and Dog Recognition

Shuaiheng Xiao

May 2, 2020

Professor: Rensheng Wang

EE-551 Python Final Project

Stevens Institute of Technology

INSTRUCTIONS

This project based on PCA algorithm, KNN algorithm and SVM algorithm.

Principal Component Analysis (PCA) Algorithm is an unsupervised machine learning algorithm that attempts to reduce the dimensionality within a dataset while still retaining as much as possible.

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems.

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.

The process

Dataset

Dataset is from Kaggle Website, it contains thousands of dog and cat images separately. I choose 50 cat images and 50 dog images as my train set, and choose 10 cat images and 10 dog images as my test set.

Pre-process data

First, I do pre-processing for my images. Since different images from this dataset have different size (i.e. pixel), I uniform them as 50*50. Then convert images from RGB scale to gray scale. Finally, I convert them from 50*50 matrix to 2500*1 column vector.

```
path='/Users/xiaoshuaiheng/Desktop/EE 551 Python/final project/cat'
trainFiles = os.listdir(path)
numFiles = len(trainFiles)
image_size=(50,50)
cat_set=[]
for i in range(numFiles-1):
    image = cv.imread(path+'/'+str(i)+'.jpg',cv.IMREAD_GRAYSCALE)
    image = cv.resize(image,image_size) # resize all images as uniform pixel
    image = image.reshape(image.size)  # reshape image matrix as 2500*1
    cat_set.append(image)
```

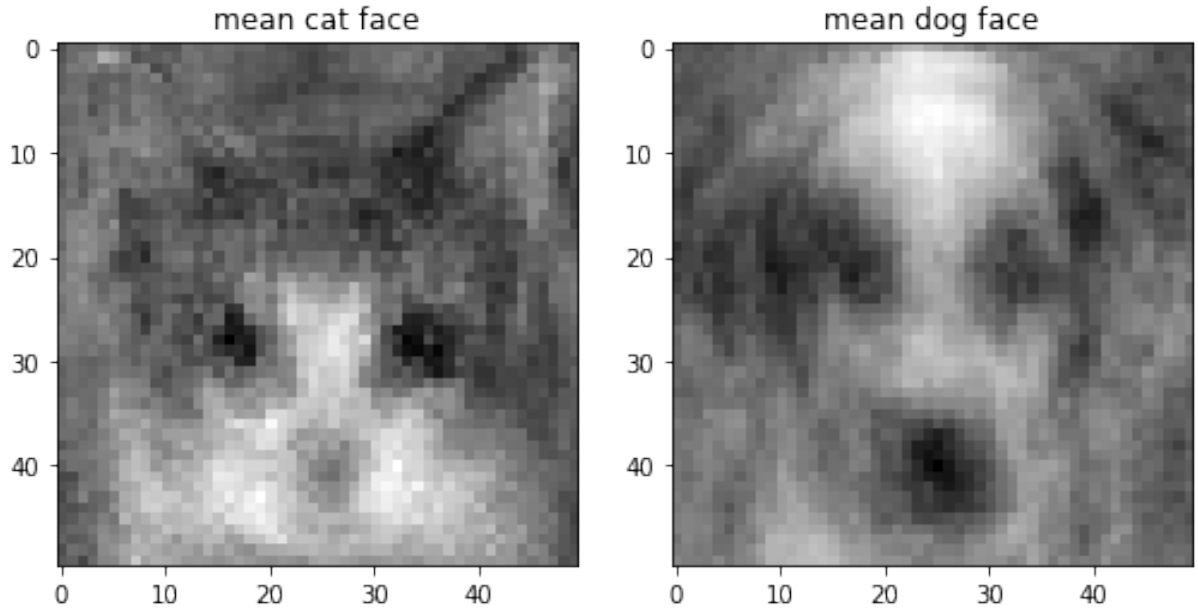
Packages:

```
In [144]: import os
          import cv2 as cv
```

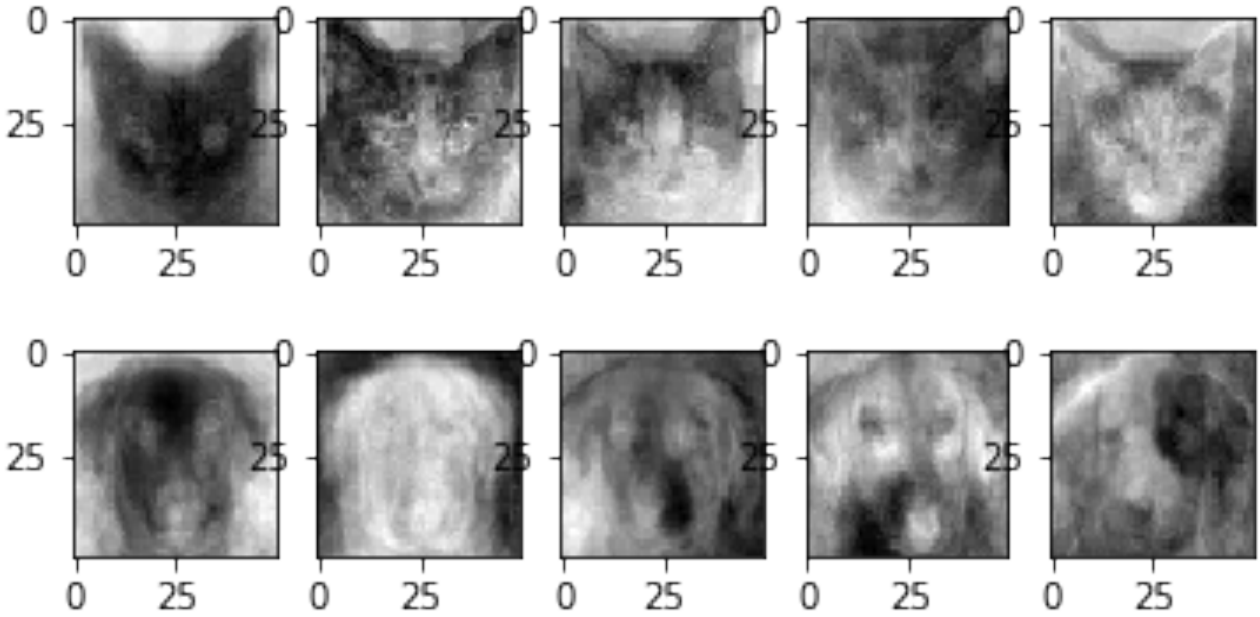
Os is for reading my dataset from my local file
Cv2 is a package can operate images.

PCA algorithm

Firstly I calculate the mean of cat image set and dog image set to get the mean cat face and mean dog face:



And then calculate scatter matrix. By decomposing scatter matrix we can get eigenvectors corresponding eigenvalues. By choosing top 5 eigenvalues, we are able to obtain corresponding 5 eigenvectors as our train samples. These pictures below are eigenfaces from reconstructing eigenvectors.



In [152]:
```
scatter_matrix=np.zeros(shape=(2500,2500)) # set up an empty scatter_matrix
for i in range(50):
    a=cat_set[:,i].reshape(2500,1)-mean_cat.reshape(2500,1)
    b=a.transpose()
    scatter=a*b
    scatter_matrix+=scatter
```

In [153]:
```
(eigenvalues,eigenvectors)=la.eig(scatter_matrix) # eigendecomposition
```

In [154]:
```
eigenvalues=eigenvalues.real   # convert complex nums to real nums
print(eigenvalues)
eigenvectors=eigenvectors.real
print(eigenvectors)
```

```
[ 1.15694143e+08  7.57828387e+07  3.88803378e+07 ... -1.30229520e-12
 -1.30229520e-12 -2.17339666e-11]
[[ 0.04019428 -0.00796898  0.01868556 ...  0.0048468   0.0048468
  -0.00181273]
 [ 0.0379048  -0.00873341  0.01737816 ...  0.00023081  0.00023081
  -0.00062048]
 [ 0.04025592 -0.00884969  0.01784118 ... -0.00076511 -0.00076511
   0.00015262]
 ...
 [ 0.00266401 -0.00775287  0.03830061 ...  0.00024987  0.00024987
  -0.00786179]
 [ 0.00485449 -0.00731148  0.03471125 ...  0.03955391  0.03955391
  -0.05962925]
 [ 0.00745458 -0.00996759  0.03146117 ...  0.03887626  0.03887626
  -0.01885738]]
```

I import numpy to build array here, and import scipy.linalg to calculate eigen decomposition.

KNN algorithm

Since KNN algorithm is intuitive, I define my own function to implement it.

```
def k_nearestNeighbor(k,test,eigen_cat,eigen_dog):
    length_cat,width_cat=eigen_cat.shape  # calculate distance between test and eigen_cat
    result_cat=[]
    for i in range(width_cat):
        sum_cat=sum((eigen_cat[:,i]-test)**2)
        dis_cat=np.sqrt(sum_cat)
        result_cat.append(dis_cat)
    length_dog,width_dog=eigen_dog.shape # calculate distance between test and eigen_dog
    result_dog=[]
    for j in range(width_dog):
        sum_dog=sum((eigen_dog[:,j]-test)**2)
        dis_dog=np.sqrt(sum_dog)
        result_dog.append(dis_dog)
    result_cat.sort()
    result_dog.sort()
    dic={}
    for ii in range(k):
        dic[result_cat[ii]]='cat'
        dic[result_dog[ii]]='dog'
    for jj in range(k):
        del dic[max(dic.keys())]
    return dic
```

The basic idea of KNN algorithm here is by calculating Euclidean distance between train sample and test sample to classify different class. Here I design k = 3 which means that obtain 3 nearest data point.

Test part:

```
path='/Users/xiaoshuaiheng/Desktop/EE 551 Python/final project/test_dog'
trainFiles = os.listdir(path)
numFiles = len(trainFiles)
image_size=(50,50)
for i in range(numFiles-1):
    image = cv.imread(path+'/'+str(i)+'.jpg',cv.IMREAD_GRAYSCALE)
    image = cv.resize(image,image_size)
    image = image.reshape(image.size)
    #image = image-mean_dog
    image = image/np.std(image,ddof=1)
    print(k_nearestNeighbor(3,image,eigen_cat,eigen_dog))
```

```
{124.02775965373459: 'cat', 124.16580953522205: 'dog', 124.21641685855643: 'cat'}
{113.78734460228323: 'cat', 114.02552060206128: 'cat', 114.02918077827113: 'cat'}
{97.86986411030796: 'cat', 98.02630341692452: 'cat', 98.16069626269928: 'cat'}
{125.99752798662203: 'cat', 126.2255554646807: 'cat', 126.39983807633254: 'cat'}
{153.0387018680535: 'cat', 153.08811469572163: 'dog', 153.11182198012426: 'cat'}
{120.68916170388007: 'cat', 121.02975652636577: 'dog', 120.9965464934432: 'cat'}
{115.56404245784434: 'cat', 115.79425350259537: 'dog', 115.80795750882629: 'cat'}
{123.82996995217232: 'cat', 124.1098650281999: 'cat', 124.11594992045663: 'cat'}
{165.15236421325466: 'cat', 165.3233051122957: 'dog', 165.16016447805524: 'cat'}
{137.75208411876835: 'cat', 138.06096250684817: 'dog', 137.9746648938852: 'cat'}
```

The result here is not all that good, it can only classify the cat part, and in dog part, the accuracy is low. The reason here I believe the KNN algorithm is not

perfectly suitable for image recognition. And also by reading others researches, it confirms my conclusion. So, I import SVM classifier.

SVM Algorithm:

Importing SVM classifier from Sklearn, it is a very useful classifier which can fit many problems. Lots of kernel functions can chose, here I selected 'Polynomial kernel' and set the degree of kernel function 2.

```
In [119]: from sklearn import svm

In [120]: clf = svm.SVC(kernel='poly')

In [124]: clf.fit(train,np.ravel(response,order='C'))

Out[124]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
              decision_function_shape='ovr', degree=3, gamma='scale', kernel='poly',
              max_iter=-1, probability=False, random_state=None, shrinking=True,
              tol=0.001, verbose=False)
```

Using 'fit' function to build my own classifier and set up parameters.

Test part:

In [180]:
```
path='/Users/xiaoshuaiheng/Desktop/EE 551 Python/final project/test_cat'
trainFiles = os.listdir(path)
numFiles = len(trainFiles)
image_size=(50,50)
sum_=0
for i in range(numFiles-1):
    image = cv.imread(path+'/'+str(i)+'.jpg',cv.IMREAD_GRAYSCALE)
    image = cv.resize(image,image_size)
    image = image.reshape(image.size)
    image = image-mean_cat
    image = image/np.std(image,ddof=1)
    image = image.reshape((1,-1))
    sum_+=clf.predict(image)
    if clf.predict(image)[0]==0:
        print('Wrong, this is dog')
    else:
        print('Correct, this is cat')
    accuracy=sum_[0]/(numFiles-1)
print('\naccuracy is %.1f' %(accuracy))
```

```
Correct, this is cat
Correct, this is cat
Correct, this is cat
Correct, this is cat
Wrong, this is dog
Wrong, this is dog
Correct, this is cat
Correct, this is cat
Wrong, this is dog
Correct, this is cat

accuracy is 0.7
```

I chose 10 cat images as test set, and it can classify 7 images, which means that the accuracy here is roughly 0.7.

In [179]:
```
path='/Users/xiaoshuaiheng/Desktop/EE 551 Python/final project/test_dog'
trainFiles = os.listdir(path)
numFiles = len(trainFiles)
image_size=(50,50)
sum_=0
for i in range(numFiles-1):
    image = cv.imread(path+'/'+str(i)+'.jpg',cv.IMREAD_GRAYSCALE)
    image = cv.resize(image,image_size)
    image = image.reshape(image.size)
    image = image-mean_dog
    image = image/np.std(image,ddof=1)
    image = image.reshape((1,-1))
    sum_+=clf.predict(image)
    if clf.predict(image)[0]==0:
        print('Correct, this is dog')
    else:
        print('Wrong, this is cat')
    accuracy=(numFiles-sum_[0])/(numFiles-1)
print('\naccuracy is %.1f' %(accuracy))
```

```
Correct, this is dog
Correct, this is dog
Correct, this is dog
Wrong, this is cat
Wrong, this is cat
Correct, this is dog
Correct, this is dog
Correct, this is dog
Wrong, this is cat
Correct, this is dog

accuracy is 0.8
```

Test 10 dog images here, and the accuracy here is 0.8.

Therefore, the total accuracy is 0.75.

Conclusion

Given limited time under this special circumstance, I cannot explore other different algorithms to further increase the performance of my classifier. But the SVM classifier has good performance. By finishing this final project, I have more confidence with my Python coding skill and I will keep improving my code structure and learn more advanced algorithm.