PCA期中报告

一 2DPCA 实现:

1.2D PCA L2

def PCA\_2D(image\_array):

    Eimage\_array=np.zeros\_like(image\_array[0],dtype=float)

    for i in range(len(image\_array)):

        Eimage\_array+=image\_array[i]

    Eimage\_array/=len(image\_array)

    image\_centered=image\_array-Eimage\_array

    G\_t=np.zeros((image\_array[0].shape[1],image\_array[0].shape[1]))

    for i in range(len(image\_array)):

        G\_t+=image\_centered[i].T@image\_centered[i]

    G\_t/=len(image\_array)

    w\_G\_t,v\_G\_t=np.linalg.eigh(G\_t)

return (w\_G\_t[::-1],v\_G\_t[:,::-1])

2.2D PCA L1

def PCA\_2D\_L1\_one\_dimension(image\_centered,max\_turn):

    u\_t=np.random.rand(image\_centered.shape[-1])-0.5

    u\_t/=np.sqrt(u\_t@u\_t.T)

    while True:

        turn\_count=0

        tmp=image\_centered@u\_t

        polarity=np.zeros(image\_centered.shape[:2],dtype=int)

        polarity[tmp>0]=1

        polarity[tmp<=0]=-1

        u\_t\_new=sum([polarity[i]@image\_centered[i] for i in range(len(polarity))])

        u\_t\_new=u\_t\_new/np.sqrt(u\_t\_new@u\_t\_new.T)

        delta=u\_t\_new-u\_t

        if ((delta.max()-delta.min())<1e-5):

            break

        if turn\_count>=max\_turn:

            print('Max turn')

            break

        turn\_count+=1

        u\_t=u\_t\_new

    return u\_t\_new

def PCA\_2D\_L1(image\_centerer,max\_turn,dimension):

    p\_vectors=[PCA\_2D\_L1\_one\_dimension(image\_centered,max\_turn)]

    last\_image=image\_centered

    for i in range(1,dimension):

        last\_v=p\_vectors[-1].reshape(last\_image.shape[-1],1)

        new\_image=last\_image-last\_image@last\_v@last\_v.T

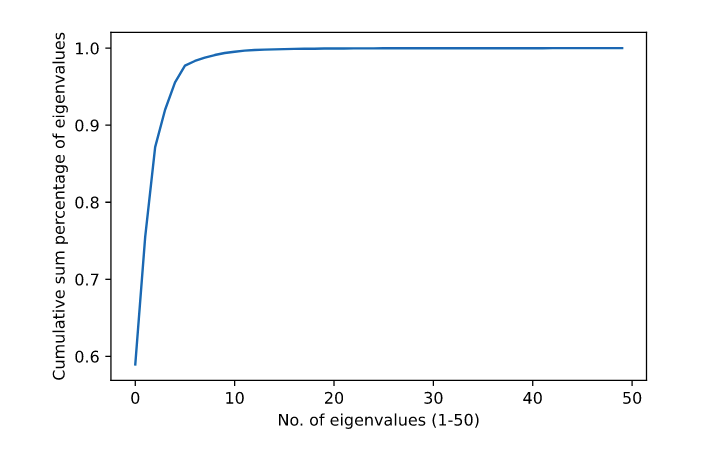
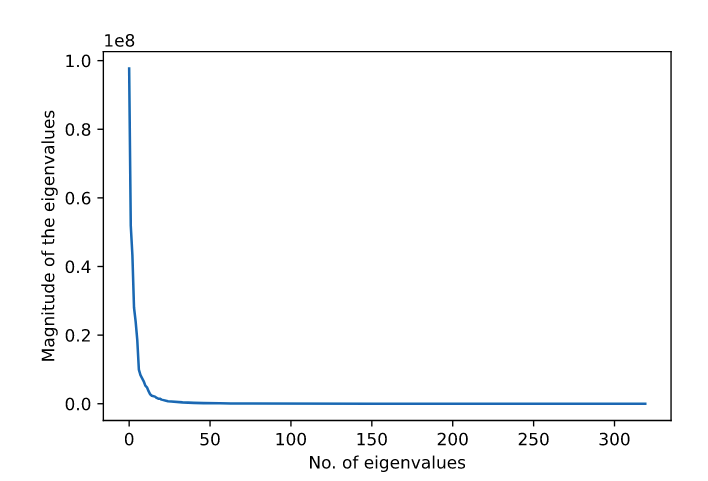
        last\_image=new\_image

        p\_vectors.append(PCA\_2D\_L1\_one\_dimension(new\_image,max\_turn))

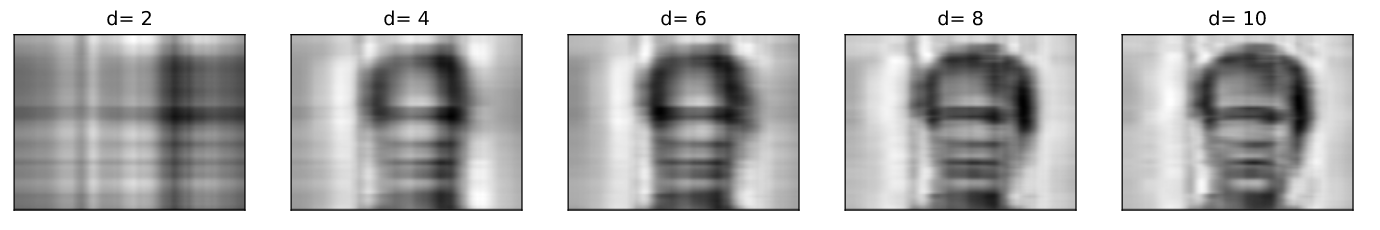
return np.array(p\_vectors).T

二 2D PCA结果分析:

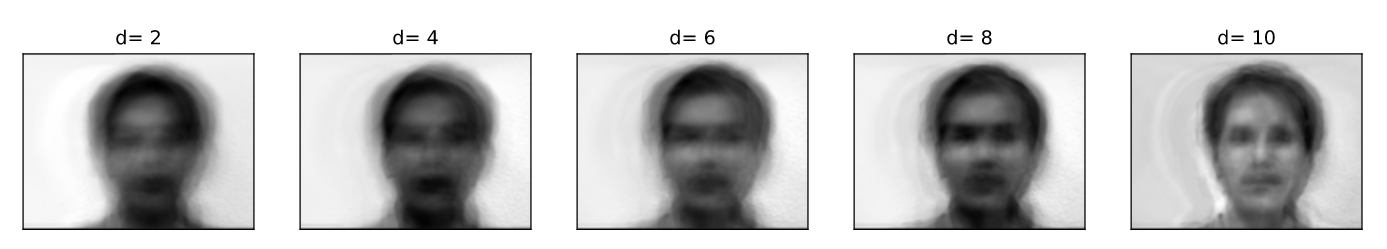
1.特征值随特征向量数的变化如下: 2.平方和所占百分比和:



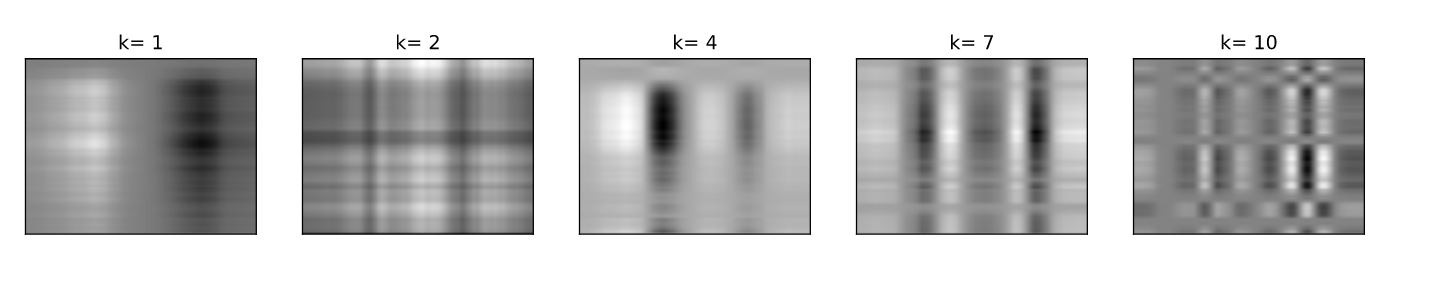
3.基于2D PCA的图像重构, 分别采取2,4,6,8,10个特征值:



与1D PCA比较:



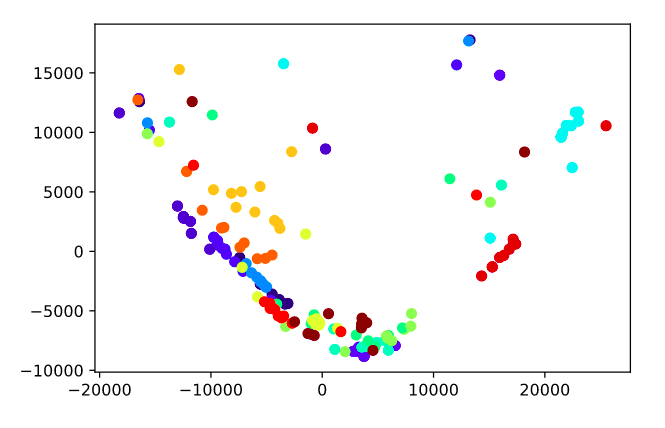
4. 2D PCA图像重构中第1,2,4,7,10个特征得到的子图:



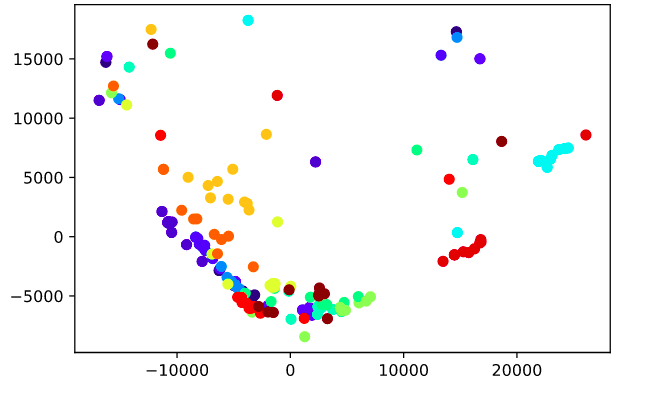
当取4个特征向量时, 所获得的图像已经占了90%的能量, 舍去更低维度是合理的.

三 可视化分布对比:

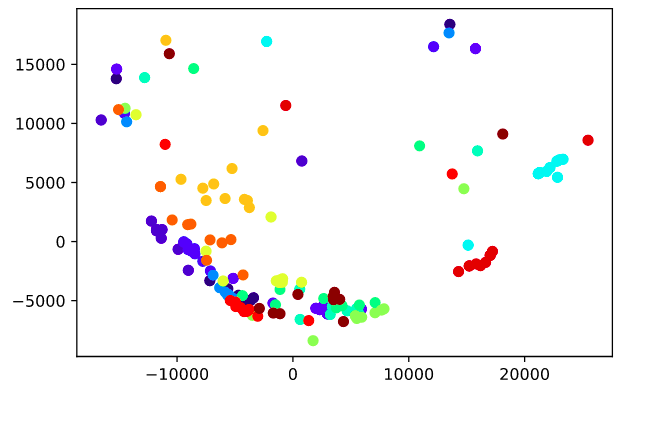
1D PCA(降到二维):



2D PCA(取前4个特征向量后用PCA降到2维):



L1-2D PCA(取前4个特征向量后用PCA降到2维):



四 聚类(k-means):

取1D-PCA-L2前10个特征值进行k-means聚15类,聚10次, 算出F值在0.616-0.711之间

取2D-PCA-L2前4个特征向量进行k-meas聚15类,聚10次, 算出F值在0.658-0.713之间

取2D-PCA-L1前4个特征向量进行k-meas聚15类, 聚10次, 算出F值在 0.640-0.711之间

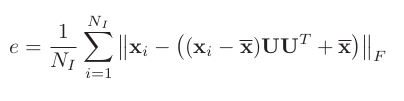
五 分类(KNN):

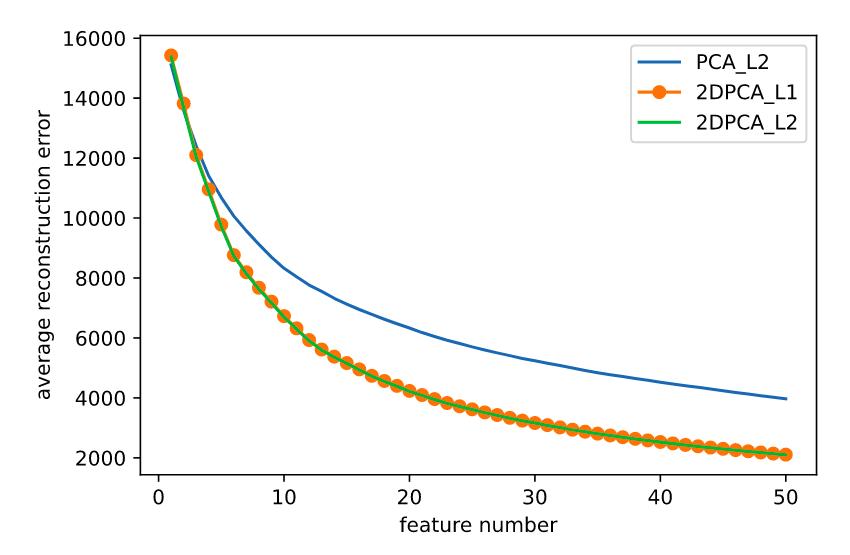
对1D-PCA-L2,在取前17个特征向量,k=1时可获得最小的错误率18.2%

对2D-PCA-L2,在取前6个特征向量,k=1时可获得最小的错误率18.2%

对2D-PCA-L1,在取前4个特征向量,k=1时可获得最小的错误率18.2%

六 计算图像重构损失:





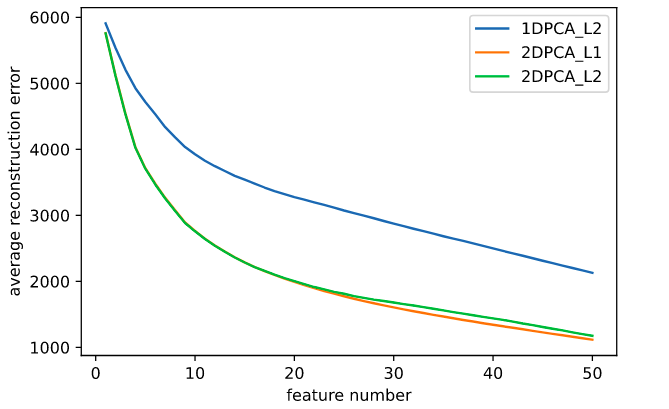
发现2DPCA\_L2和2DPCA\_L1重构误差大致相同, 且均远小于传统的PCA\_L2.

七 基于加入噪声的二维图像PCA:

首先将yale数据裁减成100\*80大小,取出其中的20%加上30\*30-60\*60大小的噪声矩阵(矩阵内数值为(0/255),含有噪声的图如下:



计算图像重构损失如下:



可以发现2D PCA(L1)在维度较高时优于2D PCA(L2), 且2D PCA均优于1D PCA

八 PCA 以及两种 2D PCA 在不同类型数据下的优劣势

传统的PCA在图片维度较大时会产生较大的协方差矩阵, 难以运算.

而2D-PCA(L2)的协方差矩阵较小, 可以快速的计算出结果. 并且性能也优于传统的PCA, 但是获得的特征向量维度太大,可以通过进一步的pca解决.

2D-PCA(L1)需要使用迭代收敛进行计算, 计算效率较低, 但是在数据集有一些离散点时具要比2D -PCA(L2)性能好

九 Robut PCA

1.实现:

class Robust\_PCA:

    def \_\_init\_\_(self,D):

        self.D=D.astype(np.double)

        self.Y=np.zeros\_like(self.D)

        self.S=np.zeros\_like(self.D)

        self.L=np.zeros\_like(self.D)

        self.delta=np.linalg.norm(self.D,ord='fro')\*1e-7

        self.miu=np.prod(D.shape)/(4\*np.linalg.norm(self.D,ord=1))

        self.miu\_inv=1/self.miu

        self.lmbda=1/np.sqrt(np.max(D.shape))

    def S\_tou(self,M,tou):

        return np.sign(M)\*np.maximum((np.abs(M)-tou),np.zeros\_like(M))

    def D\_tou(self,X,tou):

        U, Sigma, V = np.linalg.svd(X, full\_matrices=False)

        return U@np.diag(self.S\_tou(Sigma,tou))@V

    def fit(self,max\_iter=2001,print\_iter=100):

        for i in range(max\_iter):

            self.L=self.D\_tou(self.D-self.S+self.miu\_inv\*self.Y,self.miu\_inv)

            self.S=self.S\_tou(self.D-self.L+self.miu\_inv\*self.Y,self.miu\_inv\*self.lmbda)

            tmp=self.D-self.L-self.S

            self.Y+=self.miu\*tmp

            judge=np.linalg.norm(tmp,ord='fro')

            if(i%print\_iter==0):

                print(i,judge)

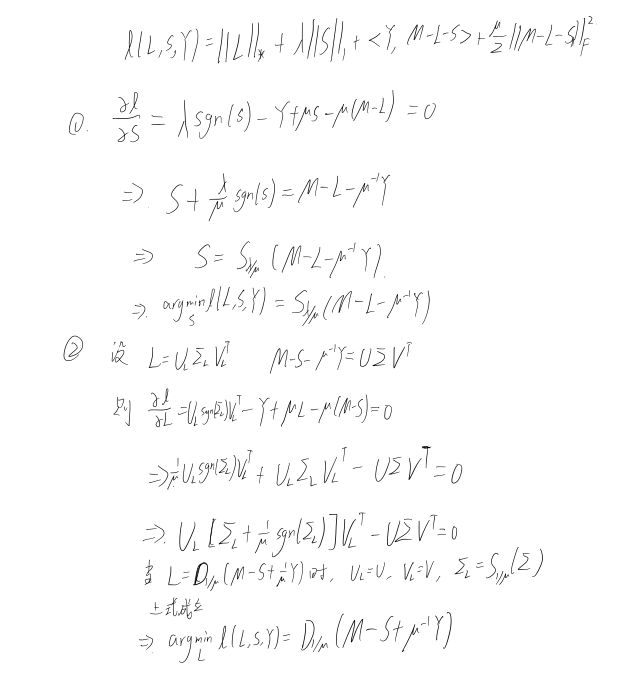
            if(judge<self.delta):

                print(i,judge)

                break

        return (self.L,self.S)

2.证明:



3.实验:

将下图(1.gif):



加上椒盐噪音(noise\_image.gif):

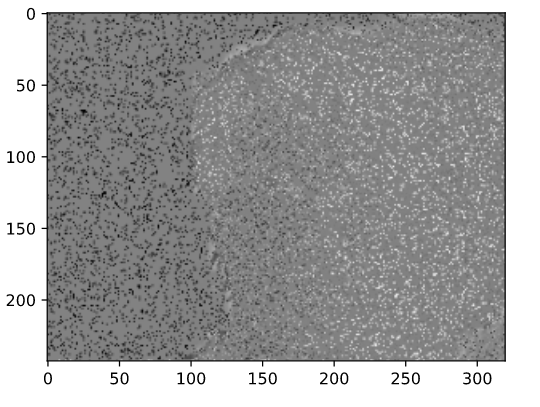
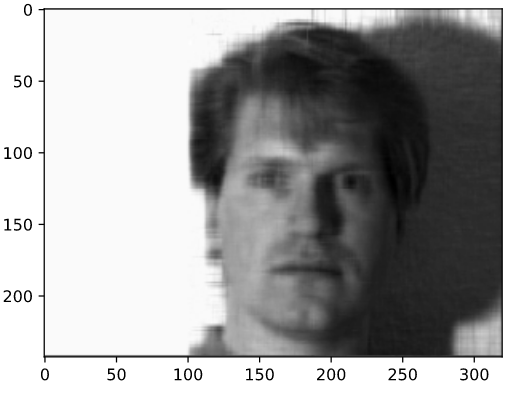


对此加上噪音的图片进行Robust-PCA:

robust\_pca=Robust\_PCA(noise\_image)

L,S=robust\_pca.fit(max\_iter=3000,print\_iter=250)

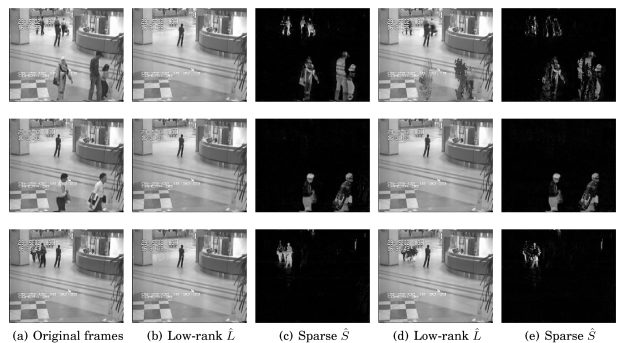
分解出L,S:



成功滤去了噪音

4.应用场景

1)视频监控: 对每一镇进行分解,得到的L为不变的背景, 而稀疏矩阵S为运动变化的物体



2)从人脸图像中去除阴影和光反射: 通过Robust-PCA 可以分离人脸照片中的阴影和反光.

