|  |  |  |  |
| --- | --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | | |
| Факультет | Компьютерных сетей и систем | | |
| Кафедра | Информатики  Дисциплина: Конструирование те технологии электронных вычислительных средств | | |
|  |  | | |
| **лабораторная работа №7**  по курсу Машинное обучение  на тему  **Метод главных компонент** | | | |
| Магистрант:  гр. 858341  Кудин Н.И. | |  | Проверил:  Стержанов М.В. |
| Минск, 2019 | | | |

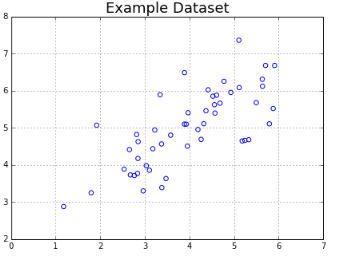
**Постановка задачи**

Набор данных **ex7data1.mat** представляет собой файл формата \*.mat (т.е. сохраненного из Matlab). Набор содержит две переменные X1 и X2 - координаты точек, для которых необходимо выделить главные компоненты.

Набор данных **ex7faces.mat** представляет собой файл формата \*.mat (т.е. сохраненного из Matlab). Набор содержит 5000 изображений 32x32 в оттенках серого. Каждый пиксель представляет собой значение яркости (вещественное число). Каждое изображение сохранено в виде вектора из 1024 элементов. В результате загрузки набора данных должна быть получена матрица 5000x1024.

**Ход работы**

1. Загрузите данные ex7data1.mat из файла и построим график загруженного набора данных.



1. Реализуем функцию вычисления матрицы ковариации данных. Вычислим координаты собственных векторов для набора данных с помощью сингулярного разложения матрицы ковариации (разрешается использовать библиотечные реализации матричных разложений). Построим на графике собственные векторы матрицы ковариации.

*def featureNormalize(myX):*

*#Feature-normalize X, return it*

*means = np.mean(myX,axis=0)*

*myX\_norm = myX - means*

*stds = np.std(myX\_norm,axis=0)*

*myX\_norm = myX\_norm / stds*

*return means, stds, myX\_norm*

*def getUSV(myX\_norm):*

*# Compute the covariance matrix*

*cov\_matrix = myX\_norm.T.dot(myX\_norm)/myX\_norm.shape[0]*

*# Run single value decomposition to get the U principal component matrix*

*U, S, V = scipy.linalg.svd(cov\_matrix, full\_matrices = True, compute\_uv = True)*

*return U, S, V*

*# Feature normalize*

*means, stds, X\_norm = featureNormalize(X)*

*# Run SVD*

*U, S, V = getUSV(X\_norm)*

*# "...output the top principal component (eigen- vector) found,*

*# and you should expect to see an output of about [-0.707 -0.707]"*

*print 'Top principal component is ',U[:,0]*

*#Quick plot, now including the principal component*

*plt.figure(figsize=(7,5))*

*plot = plt.scatter(X[:,0], X[:,1], s=30, facecolors='none', edgecolors='b')*

*plt.title("Example Dataset: PCA Eigenvectors Shown",fontsize=18)*

*plt.xlabel('x1',fontsize=18)*

*plt.ylabel('x2',fontsize=18)*

*plt.grid(True)*

*#To draw the principal component, you draw them starting*

*#at the mean of the data*

*plt.plot([means[0], means[0] + 1.5\*S[0]\*U[0,0]],*

*[means[1], means[1] + 1.5\*S[0]\*U[0,1]],*

*color='red',linewidth=3,*

*label='First Principal Component')*

*plt.plot([means[0], means[0] + 1.5\*S[1]\*U[1,0]],*

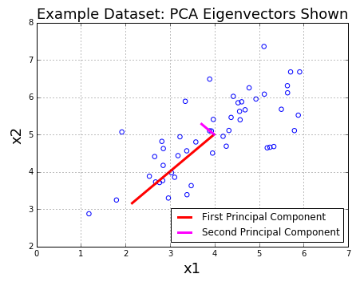
*[means[1], means[1] + 1.5\*S[1]\*U[1,1]],*

*color='fuchsia',linewidth=3,*

*label='Second Principal Component')*

*leg = plt.legend(loc=4)*

**Top principal component is [-0.70710678 -0.70710678]**



1. Реализуем функцию проекции из пространства большей размерности в пространство меньшей размерности с помощью метода главных компонент.

Реализуем функцию вычисления обратного преобразования.

Также построим график исходных точек и их проекций на пространство меньшей размерности (с линиями проекций).

*def projectData(myX, myU, K):*

*"""*

*Function that computes the reduced data representation when*

*projecting only on to the top "K" eigenvectors*

*"""*

*#Reduced U is the first "K" columns in U*

*Ureduced = myU[:,:K]*

*z = myX.dot(Ureduced)*

*return z*

*# "...project the first example onto the first dimension*

*# "and you should see a value of about 1.481"*

*z = projectData(X\_norm,U,1)*

*print 'Projection of the first example is %0.3f.'%float(z[0])*

**Projection of the first example is 1.496.**

*def recoverData(myZ, myU, K):*

*Ureduced = myU[:,:K]*

*Xapprox = myZ.dot(Ureduced.T)*

*return Xapprox*

*X\_rec = recoverData(z,U,1)*

**Recovered approximation of the first example is [-1.05805279 -1.05805279]**

*#Quick plot, now drawing projected points to the original points*

*plt.figure(figsize=(7,5))*

*plot = plt.scatter(X\_norm[:,0], X\_norm[:,1], s=30, facecolors='none',*

*edgecolors='b',label='Original Data Points')*

*plot = plt.scatter(X\_rec[:,0], X\_rec[:,1], s=30, facecolors='none',*

*edgecolors='r',label='PCA Reduced Data Points')*

*plt.title("Example Dataset: Reduced Dimension Points Shown",fontsize=14)*

*plt.xlabel('x1 [Feature Normalized]',fontsize=14)*

*plt.ylabel('x2 [Feature Normalized]',fontsize=14)*

*plt.grid(True)*

*for x in xrange(X\_norm.shape[0]):*

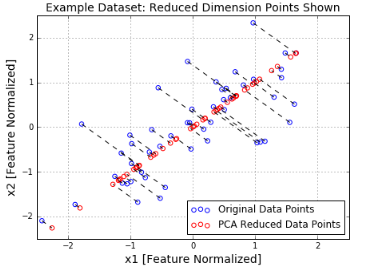
*plt.plot([X\_norm[x,0],X\_rec[x,0]],[X\_norm[x,1],X\_rec[x,1]],'k--')*

*leg = plt.legend(loc=4)*

*#Force square axes to make projections look better*

*dummy = plt.xlim((-2.5,2.5))*

*dummy = plt.ylim((-2.5,2.5))*



1. Загрузим данные **ex7faces.mat** из файла.

Визуализируем 100 случайных изображений из набора данных.

*def getDatumImg(row):*

*"""*

*Function that is handed a single np array with shape 1x1032,*

*crates an image object from it, and returns it*

*"""*

*width, height = 32, 32*

*square = row.reshape(width,height)*

*return square.T*

*def displayData(myX, mynrows = 10, myncols = 10):*

*"""*

*Function that picks the first 100 rows from X, creates an image from each,*

*then stitches them together into a 10x10 grid of images, and shows it.*

*"""*

*width, height = 32, 32*

*nrows, ncols = mynrows, myncols*

*big\_picture = np.zeros((height\*nrows,width\*ncols))*

*irow, icol = 0, 0*

*for idx in xrange(nrows\*ncols):*

*if icol == ncols:*

*irow += 1*

*icol = 0*

*iimg = getDatumImg(myX[idx])*

*big\_picture[irow\*height:irow\*height+iimg.shape[0],icol\*width:icol\*width+iimg.shape[1]] = iimg*

*icol += 1*

*fig = plt.figure(figsize=(10,10))*

*img = scipy.misc.toimage( big\_picture )*

*plt.imshow(img,cmap = cm.Greys\_r)*

*displayData(X)*



1. С помощью метода главных компонент вычислим собственные векторы.

Визуализируем 36 главных компонент с наибольшей дисперсией.

*# Feature normalize*

*means, stds, X\_norm = featureNormalize(X)*

*# Run SVD*

*U, S, V = getUSV(X\_norm)*

*# Visualize the top 36 eigenvectors found*

*displayData(U[:,:36].T,mynrows=6,myncols=6)*



**Качество ухудшилось, изображение стало расплывчатым.**

1. Визуализируем 100 главных компонент с наибольшей дисперсией.



**В компонентах с наибольшей дисперсией изображения стали более темные.**

1. С помощью метода главных компонент визуализируем данное изображение в 3D и 2D.

*# Use PCA to go from 3->2 dimensions*

*z = projectData(A\_norm,U,2)*

*# Make the 2D plot*

*subX = []*

*for x in xrange(len(np.unique(idxs))):*

*subX.append(np.array([A[i] for i in xrange(A.shape[0]) if idxs[i] == x]))*

*fig = plt.figure(figsize=(8,8))*

*for x in xrange(len(subX)):*

*newX = subX[x]*

*plt.plot(newX[:,0],newX[:,1],'.',alpha=0.3)*

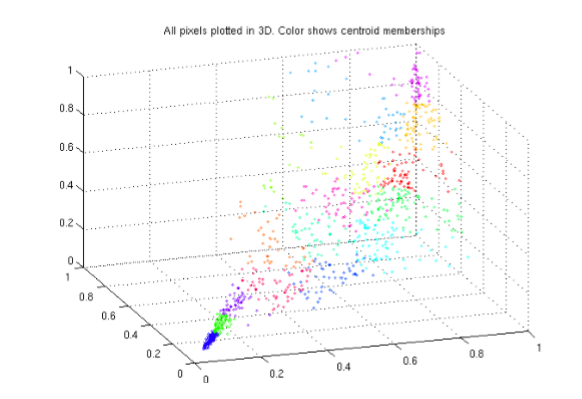
*plt.xlabel('z1',fontsize=14)*

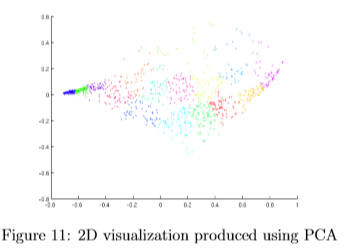
*plt.ylabel('z2',fontsize=14)*

*plt.title('PCA Projection Plot',fontsize=16)*

*plt.grid(True)*

*3D:*



2D:  


**Заключение**

В ходе работы над лабораторной работой были приобретены навыки использования различных библиотек и программных средства, а также применили метод главных компонент в обработке изображений