DS-GA.1013 Mathematical Tools for Data Science
Homework 2
Yves Greatti - yg390

1. (Correlation coefficient) The entries of a two-dimensional random vector have a correlation coefficient equal to one. What is the variance of the second principal component? Provide both a proof and an intuitive justification.

   Let X and Y the two components of a random vector. If $\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = 1$ then $\mathrm{Cov}(X,Y) = \sigma_X \sigma_Y$ The covariance matrix is then $\Sigma = \begin{bmatrix} \sigma_X^2 & \mathrm{Cov}(X,Y) \\ \mathrm{Cov}(X,Y) & \sigma_Y^2 \end{bmatrix} = \begin{bmatrix} \sigma_X^2 & \sigma_X \sigma_Y \\ \sigma_X \sigma_Y & \sigma_Y^2 \end{bmatrix}$. $\det(\Sigma - \lambda I) = \lambda(\lambda - (\sigma_X^2 + \sigma_Y^2)), \lambda \in \mathbf{R} \Rightarrow$ the eigenvalues are 0 and $\sigma_X^2 + \sigma_Y^2$. The sample variance of the second principal component is the smaller eigenvalue 0. We can expect such result since there X and Y are positively correlated with a coefficient of one thus the variance of the data is completely expressed by the variance of the first principal component, there is no information on the second principal component.

2. (Not centering) To analyze what happens if we apply PCA without centering, let $\tilde{x}$ be a d-dimensional vector with mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma_{\tilde{x}}$ equal to the identity matrix. If we compute the eigendecomposition of the matrix $\mathrm{E}(\tilde{x}\tilde{x}^T)$ what is the value of the largest eigenvalue? What is the direction of the corresponding eigenvector?

$$\begin{aligned}
\mathrm{E}((\tilde{x}-\mu)(\tilde{x}-\mu)^T)) &= \mathrm{E}(\tilde{x}\tilde{x}^T - 2\mu^T\tilde{x} + \mu^T\mu) \\
&= \mathrm{E}(\tilde{x}\tilde{x}^T) - 2\mu\mathrm{E}(\tilde{x}^T) + \mu\mu^T \\
&= \mathrm{E}(\tilde{x}\tilde{x}^T) - 2\mu\mu^T + \mu\mu^T \\
&= \mathrm{E}(\tilde{x}\tilde{x}^T) - \mu\mu^T \\
\Rightarrow \mathrm{E}(\tilde{x}\tilde{x}^T) &= \mathrm{E}((\tilde{x}-\mu)(\tilde{x}-\mu)^T)) + \mu\mu^T \\
&= \Sigma_{\tilde{x}} + \mu\mu^T
\end{aligned}$$

We have

$$\mathrm{E}(\tilde{x}\tilde{x}^T) = \begin{bmatrix} 1+\mu_1^2 & \mu_1\mu_2 & \dots & \mu_1\mu_d \\ \mu_1\mu_2 & 1+\mu_2^2 & \dots & \mu_2\mu_d \\ \vdots & \ddots & & \vdots \\ \mu_1\mu_d & \dots & & 1+\mu_d^2 \end{bmatrix}$$

We note that an eigenvectors of this matrix is $u = \begin{bmatrix}\mu_1 & \mu_2 & \dots & \mu_d\end{bmatrix}^T$ since $\mathrm{E}(\tilde{x}\tilde{x}^T)u = (1 + \sum_{i=1}^d \mu_i^2)u$.

3. (Financial data) In this exercise you will use the code in the findata folder. For the data loading code to work properly, make sure you have the pandas Python package installed on your system.

   Throughout, we will be using the data obtained by calling *load_data* in *findata_tools.py*. This will give you the names, and closing prices for a set of 18 stocks over a period of 433 days ordered chronologically. For a fixed stock (such as msft), let $P_1, \ldots, P_{433}$ denote its sequence of closing prices ordered in time. For that stock, define the daily returns series $R_i := P_{i+1} - P_i$ for $i = 1, \ldots, 432$. Throughout we think of the daily stock returns as features, and each day (but the last) as a separate datapoint in $\mathbb{R}^{18}$. That is, we have 432 datapoints each having 18 features.

   (a) Looking at the first two principal directions of the centered data, give the two stocks with the largest coefficients (in absolute value) in each direction. Give a hypothesis why these two stocks have the largest coefficients, and confirm your hypothesis using the data. The file *findata_tools.py* has*pretty_print* functions that can help you output your results. You are not required to include the principal directions in your submission.

   The two stocks corresponding to the two principal directions of the centered data with the largest coefficients (in absolute value) in each direction are: "amzn" and "goog". It can be explained by computing the absolute return for each stock over the period of 433 days:

```
      aapl        amzn        msft        goog         xom         apc  \
454.14342  3423.779352  208.844523  2799.940424  258.501684  388.899131

       cvx           c          gs         jpm         aet         jnj  \
381.849154  247.140956  901.311051  277.9143  505.556593  275.823333

       dgx         spy         xlf         sso         sds    uso
257.409551  424.583189  64.368439  277.841774  276.939997  77.16
```

   Figure 1: Stocks returns over a period of 433 days (output of pretty_print).

   In term of return goog and amzn stocks returned about 4 and 3 times more than the next stock after amzn and goog stocks with the highest return: gs, 53 and 43 times more than the last stock in term of return among the 18 stocks: xlf.

amzn/gs	amzn/xlf	goog/gs	goog/xlf
3.8	53.2	3.1	43.5

   So most of the variance in the data will be explained by these two stocks: goog and amzn.

   (b) Standardize the centered data so that each stock (feature) has variance 1 and compute the first 2 principal directions. This is equivalent to computing the principal directions of the correlation matrix (the previous part used the covariance matrix). Using the information in the comments of *generate_findata.py* as a guide to the stocks, give an English interpretation of the first 2 principal directions computed here. You are not required to include the principal directions in your submission.

   We can think of each of the entries of the principal directions as a weighting on the corresponding stock.

   - SPY - A security that roughly tracks the S&P 500, a weighted average of the stock prices of 500 top US companies. We have only 18 stocks and excluding the ETF

(Exchange traded products), we are left with 13 stocks. We can notice that among these 13 stocks, SPY entry in the principal directions, has the same than the other 13 entries for the same principal direction. For a given day, computing a weighted average using the absolute values of entries in the first principal direction and the prices of these 13 stocks, we obtain a weighted average price in the range of the SPY price reported for the same day. It is less true using the entries of the second principal direction.

```
PD0
      aapl      amzn      msft      goog       xom       apc       cvx  \
 -0.184648 -0.168095 -0.227645 -0.197887 -0.208861 -0.194808 -0.223865

         c        gs       jpm       aet       jnj       dgx       spy  \
 -0.280147 -0.260443 -0.27294 -0.188555 -0.150783 -0.196612 -0.336632

       xlf       sso       sds       uso
 -0.198364 -0.334826  0.327259 -0.159213
PD1
      aapl     amzn      msft      goog       xom       apc       cvx  \
  0.285821  0.3302  0.302005  0.389346 -0.293444 -0.329116 -0.308272

         c        gs       jpm       aet       jnj       dgx       spy  \
 -0.133288 -0.137664 -0.155599  0.071286  0.108239  0.165412  0.084872

       xlf       sso       sds       uso
 -0.124933  0.091568 -0.056002 -0.370943
```

Figure 2: First two Principal Directions.

```
--------------------------------------------------------------------------------
Weights for the first 13 stock prices using PD0
      aapl      amzn      msft      goog       xom       apc       cvx  \
 -0.184648 -0.168095 -0.227645 -0.197887 -0.208861 -0.194808 -0.223865

         c        gs       jpm       aet       jnj       dgx
 -0.280147 -0.260443 -0.27294 -0.188555 -0.150783 -0.196612
--------------------------------------------------------------------------------
Absolute values of weights for the first 13 stock prices using PD0
      aapl      amzn      msft      goog       xom       apc       cvx  \
  0.184648  0.168095  0.227645  0.197887  0.208861  0.194808  0.223865

         c        gs       jpm       aet       jnj       dgx
  0.280147  0.260443  0.27294  0.188555  0.150783  0.196612
--------------------------------------------------------------------------------
Day0 prices
        aapl        amzn        msft        goog         xom         apc  \
  101.790649  636.98999  52.433533  741.840027  72.740799  48.801582

        cvx           c          gs         jpm         aet         jnj         dgx
  82.577927  50.149845  172.800156  61.18005  107.291946  95.826782  68.369064
--------------------------------------------------------------------------------
Day0, computed SPY weighted average prices:163.92740408192924
--------------------------------------------------------------------------------
Day0 SPY price
SPY weight:-0.3366318246190488
         spy
  194.027725
```

Figure 3: Weighted average of the 13 stock prices and SPY price using PD0.

- XLF - A security that tracks a weighted average of top US financial companies. Among the 18 stocks, there are only 3 financial stocks: c, gs and jpm. Entry for xlf has the same sign as the entries for these three stocks in the first two principal directions. To some extent the weighted average price of these 3 stocks is somewhat close to XLF price.

```
--------------------------------------------------------------------------------
Weights for the 3 financial stock prices using PD0
         c         gs        jpm
 -0.280147 -0.260443 -0.27294
--------------------------------------------------------------------------------
Absolute values of weights for the 3 financial stock prices using PD0
         c         gs        jpm
  0.280147  0.260443  0.27294
--------------------------------------------------------------------------------
Day100 prices
          c           gs         jpm
  45.329781  155.994034  62.998978
--------------------------------------------------------------------------------
Day100, computed XLF weighted average prices:86.68575537909774
--------------------------------------------------------------------------------
Day100 XLF price
XLF weight:-0.19836417024240857
Day100 XLF price
         xlf
  14.347868
```

Figure 4: Weighted average of the 3 financial stock prices and XLF price using PD0.

- SSO - ProShares levered ETF that roughly corresponds to twice the daily performance of the S&P 500. We have similar entries for SSO and SPY which indicate a strong correlation between these two stocks

	SSO	SPY
First PD	-0.3348	-0.3366
Second PD	0.09156	0.0848

We can also pick two random days (200 and 201) and compare the returns for spy vs. xlf, you can see the return for xlf tracks the return of spy:

```
     aapl      amzn      msft     goog       xom       apc       cvx  \
-0.343941  0.039978 -0.126941  6.23999  0.385491  0.956391  0.461364

        c        gs       jpm       aet       jnj       dgx       spy  \
 0.483455  1.856705  0.639229 -0.366638 -0.798614 -0.71032  0.558686

      xlf       sso       sds   uso
 0.167292  0.33892 -0.240006  0.21
```

Figure 5: Returns for day 200

```
    aapl      amzn      msft      goog       xom       apc       cvx  \
-0.05896 -7.369995 -0.273407 -4.530029  0.038551 -0.169361 -0.384475

         c   gs       jpm       aet       jnj       dgx      spy  xlf  \
 0.098664  0.0 -0.088517 -0.128807  0.272697  1.568627 -0.39206  0.0

       sso       sds   uso
-0.199363  0.200005 -0.23
```

Figure 6: Returns for day 201

- SDS - ProShares inverse levered ETF that roughly corresponds to twice the negative daily performance of the S&P 500. The entries for SDS and SPY are roughly opposite confirming the opposite trend of SDS compared to SPY.

	SDS	SPY
First PD	0.3272	-0.3366
Second PD	-0.0560	0.0848

- USO - Exchange traded product that tracks the price of oil in the US Taking the mean of the entries related to oil company (xom, apc, cvx) from the principal directions and comparing to the entry for USO, they are close, confirming the correlation between uso and (xom, apc, cvx):

	USO	Mean(XOM, APC, CVX)
First PD	-0.1592	-0.2091
Second PD	-0.3709	-0.3102

(c) Assume the stock returns each day are drawn independently from a multivariate distribution $\tilde{x}$ where $\tilde{x}[i]$ corresponds to the ith stock. Assume further that you hold a portfolio with 200 shares of each of appl, amzn, msft, and goog, and 100 shares of each of the remaining 14 stocks in the dataset. Using the sample covariance matrix as an estimator for the true covariance of $\tilde{x}$, approximate the standard deviation of your 1 day portfolio returns $\tilde{y}$ (this is a measure of the risk of your portfolio). Here $\tilde{y}$ is given by

$$\tilde{y} := \sum_{i=1}^{18} \alpha[i]\tilde{x}[i],$$

where $\alpha[i]$ is the number of shares you hold of stock i.

Using the sample covariance matrix and taking the root square of [shares^T $\times$ covariance $\times$ shares], we find that for such portfolio the standard deviation of 1 day is: 4309.94952.

(d) Assume further that $\tilde{x}$ from the previous part has a multivariate Gaussian distribution. Compute the probability of losing 1000 or more dollars in a single day. That is, compute

$$\Pr(\tilde{y} \leq -1000).$$

For each day of the 432 days, we compute the daily return $\tilde{y} := \sum_{i=1}^{18} \alpha[i]\tilde{x}[i]$ and count the number of times over the 432 days: $\Pr(\tilde{y} \leq -1000)$, then divide the result by the number of days (432), we obtain: 0.3425.

Note: The assumptions made in the previous parts are often invalid and can lead to inaccurate risk calculations in real financial situations.

4. The following questions refer to the code in the folder `faces` The Olivetti faces dataset used in `faces` contains images of faces of people associated with a unique numeric id to identify the person.

   (a) Complete the `compute_nearest_neighbors()` function in `nearest_neighbors.py` that finds the image in the training data that is closest to a given test image. Include the generated images in your submitted homework.

   The data set consists in 400 rows of a ravelled face image of original size 64 x 64 pixels . Each row represent a datapoint in $\mathbb{R}^{4096}$. A label is associated to each face image which correspond to the Subject IDs. In the *nearest_neighbors* function we compute the distance between a test image and a set of reference image (train_matrix).

```
def compute_nearest_neighbors(train_matrix, testImage):
    distances = np.sqrt(
            np.sum((train_matrix - testImage) ** 2,
                                axis=1))
        idx_of_closest_point_in_train_matrix =
            np.argsort(distances)
    return idx_of_closest_point_in_train_matrix[0]
```

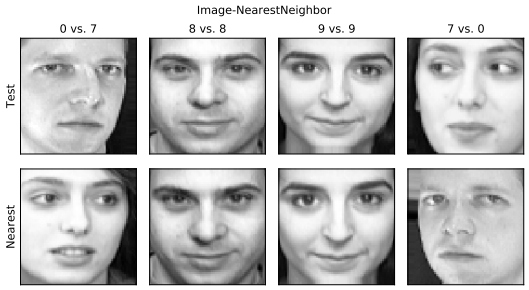

Figure 7: Test images and image found by Nearest Neighbors.

Create a new file in which you must write code to complete the following tasks:

(b) Generate a plot of k vs. σ_k^2, where σ_k^2 is the variance with the kth principal component of the data (e.g., σ_1^2 is the largest variance). Include the plot in your submitted homework document. You can limit the x axis to a reasonable number.

We use PCA decomposition from sklearn which gives us the singular values ordered which correspond to the variance with each principal component, we select k of them and plot them for each principal component:

```
faces = fetch_olivetti_faces().data
n_samples, n_features = faces.shape
faces_centered = faces - faces.mean(axis=0)
cov = np.cov(faces_centered, rowvar=False)
eigvals, _ = np.linalg.eigh(cov)
```

```
k = 40
truncated_eigvals = eigvals[::-1][:k]
fig, ax = plt.subplots(figsize=(10, 6))
k_range = range(1, k+1)
label_str = "variance for largest {}
principal components".format(k)
ax.plot(k_range, truncated_eigvals, "-", color="red",
label=label_str)
ax.set_xlabel("principal component")
ax.set_ylabel("Variance")
ax.set_title(r"Explained Variance of the $k^{th}$ component")
ax.legend()
plt.show();
fig.savefig("pb_4_b.pdf",bbox_inches='tight');
```

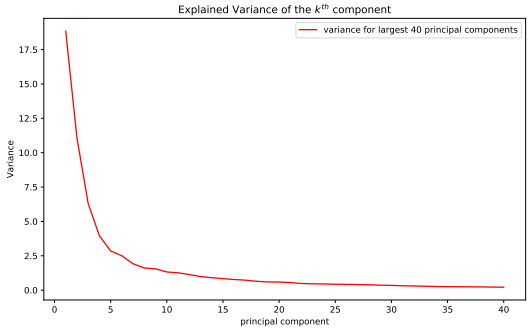

Figure 8: Variance with principal components

(c) Plot (using `plot_image_grid()` in `plot_tools.py`) the vectors corresponding to the top 10 principal directions of the data. Your principal direction vectors should be elements of $\mathbb{R}^{4096}$ (i.e., they should represent images). Include the plot in your submitted homework document.

We compute the sample covariance matrix of the data $\Sigma_{\mathcal{X}}$, the principal directions are the eigenvectors of the eigendecomposition of $\Sigma_{\mathcal{X}}$:

```
faces = fetch_olivetti_faces().data
n_samples, n_features = faces.shape
faces_centered = faces - faces.mean(axis=0)
cov = np.cov(faces_centered, rowvar=False)
_, principal_directions = np.linalg.eigh(cov)
k = 10
top_pd = principal_directions[:, ::-1][:,:k].T
title = "Top {} principal directions vectors".format(k)
plot_tools.plot_image_grid(top_pd, title)
```

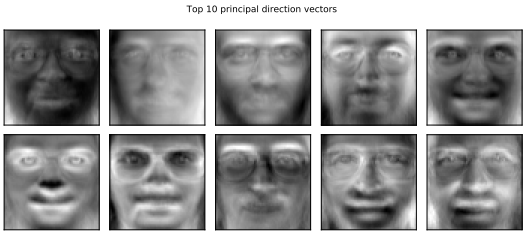

Figure 9: Top 10 principal direction vectors

(d) Use the variance of principal directions plot to determine a realtively small number k of principal components that explains the training data reasonably well. Project the training data and the test data onto the first k principal components, and run nearest neighbors for each test image in this lower dimensional space. Include your choice for k, and the plots of your nearest neighbor results in your submitted homework document. You should use the code from `nearest_neighbors.py` to generate your image plots. Based on the variance plot, most of the variance of the data is captured by the 40 first principal components. Using the first 40 principal components, we project the training and test data set onto these principal components before finding the nearest neighbors.

```
def compute_nearest_neighbors(train_matrix, testImage):
            distances = np.sqrt(
                    np.sum((train_matrix - testImage) ** 2,
                                    axis=1))
                idx_of_closest_point_in_train_matrix =
                    np.argsort(distances)
             return idx_of_closest_point_in_train_matrix[0]

test_idx = [1, 87, 94, 78]

faces = fetch_olivetti_faces()
targets = faces.target
faces = faces.images.reshape((len(faces.images), -1))

train_idx = np.array(list(set(list(range(faces.shape[0])))
- set(test_idx)))

train_set = faces[train_idx]
y_train = targets[train_idx]
test_set = faces[np.array(test_idx)]
y_test = targets[np.array(test_idx)]

k = 40
model = PCA(n_components=k)
# Do PCA and compute principal directions.
model.fit(faces);
```

```
top_principal_components = model.components_

# Projection of training and test data
projected_train_set =
 train_set.dot(top_principal_components.T)
projected_test_set =
test_set.dot(top_principal_components.T)

imgs = list()
est_labels = list()
for i in range(projected_test_set.shape[0]):
    test_image = projected_test_set[i, :]
    # Nearest neighbors in smaller dimension space
    nnIdx = compute_nearest_neighbors(projected_train_set,
    test_image)
    imgs.extend([test_set[i,:], train_set[nnIdx, :]])
    est_labels.append(y_train[nnIdx])

row_titles = ['Test', 'Nearest']
col_titles =
['%d vs. %d' % (i, j) for i, j in zip(y_test, est_labels)]
plot_tools.plot_image_grid(imgs,
                                "PC-Image-NearestNeighbor",
                                (64, 64), len(projected_test_set),
                           n_row=2, bycol=True, row_titles=row_title
                            col_titles=col_titles)
```

Figure 10: Nearest neighbors using top 10 principal component vectors

(e) Give a potential reason why the principal component-based nearest-neighbor approach used in the previous part could be more accurate than using the full training set.

Using the top 40 principal directions, we keep most of the variance of the data along these directions and by projecting into the span of these vectors, we can expect to have capture most of the significant characteristics of the data in a space of smaller dimensions ($\mathbb{R}^{40}$

vs. $\mathbb{R}^{4096}$). So performing nearest neighbors in the space with smaller dimension could be more or as accurate than doing it in the original space.

(f) Use `sklearn.cluster.KMeans` to perform KMeans on the entire dataset (both train and test set) with $k = 40$. Use `plot_image_grid()` to create a picture of all the k cluster centers.

```
rng = RandomState(0)
center = True
n_components = 40

faces = fetch_olivetti_faces().data
faces_centered = faces - faces.mean(axis=0)
estimator =  KMeans(n_clusters=n_components, random_state=rng)
estimator.fit(faces)
kmeans_components = estimator.cluster_centers_
plot_tools.plot_image_grid(kmeans_components,
 "KMEANS_clusters", (64, 64), 10, n_row=4, bycol=True)
```

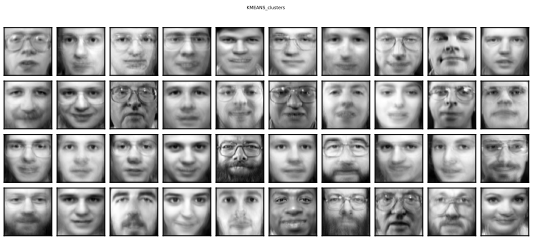

Figure 11: 40 KMeans clusters

Some notes to keep in mind:

i. The function `np.linalg.eig` might return complex eigenvectors.
ii. The data points in the training and test data are given as rows.
iii. Include all new code (or functions) you have filled in your final PDF.