Assignment 3

Due: 1.00pm Monday 24^{th} May 2021

Rules

1. This is a group assignment. (There are approximately 3 people per group and by now you should know your assigned group.)
2. While `R` is the default package / programming language for this course you are free to use `R` or `Python` for the programming components of this assignment.
3. Within each group **I strongly encourage each person to attempt each question by his / herself first** before discussing it with other members of the group.
4. Students should **not** consult students in other groups when working on their assignments.
5. Late assignments will **not** be accepted and all assignments must be submitted through the Hub with one assignment submission per group. Your submission should include a PDF report with your answers to each question as well as any relevant code. Make sure your PDF clearly identifies each member of the group by CID and name.

---

1. **Eigen-Faces (40 marks)**
Open the *Eigen_Faces_Fragment* `R` Notebook (posted on the Hub) and familiarise yourself with the Olivetti faces data-set. (The `Matlab` .mat file containing the data is also available on the Hub and the `R` Notebook shows you how to read in the data.) Then use PCA to construct k-dimensional approximations to the data-points. (To be clear, each data-point is a vector $\mathbf{x} \in \mathbb{R}^{4,096}$ corresponding to a particular face.) Some other points you may wish to consider:

   - By default the *prcomp* function in `R` first de-means the data. In addition, setting `scale=TRUE` as an argument to *prcomp* ensures that each component of the data has standard deviation 1. (There are 4,096 components or variables in this data-set.) In general it is a good idea to do this!
   - The means and standard deviations of the original data are some of the outputs of *prcomp*. You will need to use these when constructing your k-dimensional approximations because the PCA is applied to the de-meaned and standardized data (assuming you set `scale=TRUE`).

   Your answer to this question should show plots of the original 4 faces that are currently displayed in the R Notebook in addition to k-dimensional reconstructions of them for $k = 3, 10, 25$ and 50. You should also provide a short mathematical expression for how you construct the approximation. (This will be identical to the approximation provided in the slides except you also need to account appropriately for the mean and standard deviation terms.)

Solution The code to do everything can be found in the `R` Notebook *Eigen_Faces.Rmd*. (You can vary k in the code to construct the required plots.)

---

2. **Clustering & PCA – Q10.10 from *ISLR* (60 marks)**
In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data. If you are not already familiar with K-means clustering then you should read Section 10.3.1 of ISLR. (This is less than 4 pages in length and is very easy to follow.) You should also read Section 10.5.1 (just 2 pages) to see how to implement K-means clustering in `R` using the `kmeans` function.

   (a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables. (10 marks)

   Hint: There are a number of functions in `R` that you can use to generate data. One example is the `rnorm` function; `runif` is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.

   (b) Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors. (10 marks)

   (c) Perform K-means clustering of the observations with $K = 3$. How well do the clusters that you obtained in K-means clustering compare to the true class labels? (10 marks)

   Hint: You can use the table() function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.

   (d) Perform K-means clustering with $K = 2$. Describe your results. (5 marks)

   (e) Now perform K-means clustering with $K = 4$, and describe your results. (5 marks)

   (f) Now perform K-means clustering with $K = 3$ on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60×2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results. (10 marks)

   (g) Using the *scale()* function, perform K-means clustering with $K = 3$ on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain. (10 marks)

A Note on Evaluating Clusters
Suppose we wish to evaluate the performance of a clustering algorithm. We can do this as follows when we know the true class assignment of each point (which of course is unknown to the clustering algorithm). Evaluating the quality of the clusters would appear to be a simple matter but a problem arises because the clustering algorithm will just assign some arbitrary class labels to the clusters. Trying to link these class labels to the true classes can be difficult, especially when the clustering is imperfect (as will typically be the case). So how then can we evaluate the quality of the clusters when we know the true class labels of the data-points? We can do this as follows. Define a matrix $\mathbf{Y} \in \{0,1\}^{n \times n}$ according to

$$\mathbf{Y}_{ij} = \begin{cases} 1, & y(i) = y(j) \\ 0, & y(i) \neq y(j) \end{cases}$$

where $y(i)$ denotes the true class label of the i^{th} data-point for $i = 1, \ldots, n$. The matrix $\mathbf{Y}$ encodes the pair-wise labeling of samples.

Now let $z \in \mathbb{R}^n$ denote the cluster labeling obtained from the clustering algorithm, e.g. from K-means clustering. Let $\mathbf{Z} \in \{0,1\}^{n \times n}$ denote the matrix

$$\mathbf{Z}_{ij} = \begin{cases} 1, & z(i) = z(j), \\ 0, & z(i) \neq z(j). \end{cases}$$

We can then "score" the clustering $\mathbf{Z}$ according to the metric

$$\rho(\mathbf{Z}) = \sum_{i \neq j} \Big(\mathbf{Y}_{ij}(1 - \mathbf{Z}_{ij}) + (1 - \mathbf{Y}_{ij})\mathbf{Z}_{ij} \Big).$$

This metric encodes the fact that when $\mathbf{Y}_{ij} = 1$ (resp. $\mathbf{Y}_{ij} = 0$) the clustering has an error when $\mathbf{Z}_{ij} = 0$ (resp. $\mathbf{Z}_{ij} = 1$). If the clustering perfectly recovers the class labels, $\rho(\mathbf{Z}) = 0$; otherwise $\rho(\mathbf{Z}) > 0$. In particular, smaller values of $\rho(\mathbf{Z})$ denote a superior clustering.

Solution: See the `R` Notebook *PCA_Clustering_ISLR_Q10_10.Rmd* for the solutions to this question. It's important to note that the results depend on how the classes were defined originally and just how much separation there is between the classes in the original "$\mathbf{x}$" space. That said, a key takeaway is that using the first few principal components (in this case two) to do the clustering can yield a better performance than using the original data. Why might this be the case? Well, if most of the principal components are (more or less) noise, then throwing that noise away can yield a better performance. You can think of this in the usual bias-variance decomposition framework as well. Loosely speaking, working with just the top few principal components is akin to introducing a bias but the gain is a reduction in variance.

In light of these comments it would be worthwhile seeing under what circumstances you expect clustering on the first 2 PC's to do a better job than clustering on the original 50-dimensional data. (Once your code is written you can easily check if your intuition is right.)

3. **Missing Data and the Bootstrap (40 marks)**
Consider the student-exam data in the table below. There are 22 students and their results across 5 exams (labelled A to E) are presented in columns 2 to 6 of the table. Unfortunately results for some students in exams A and E are missing and these missing values are denoted with a "?". The goal of the analysis is to (i) estimate the largest eigen-value θ of the 5×5 variance-covariance matrix $\mathbf{\Sigma}$ of the exam scores and (ii) estimate the uncertainty in our estimate $\widehat{\theta}$. If there was no missing data then we could easily do (i) by (for example) computing the sample covariance matrix and then computing its largest eigen-value. We could then use the bootstrap to do (ii). But how do we do (i) and (ii) when there is missing data?

   A common and standard way to handle missing data is known as *list deletion* and this simply means deleting all data-points / records (in this case, students) who have at least one missing feature (in this case, exam score). This often works fine when there is very little missing data and only a relatively small number of data-points needs to be deleted. In many real-world settings this is not the case, however. In health-care, for example, feature vectors are often high-dimensional and it's very common that every patient (data-point) has some missing data. In our example, list deletion would result in having to delete 17 of the 22 records. This would mean discarding an awful lot of information and would therefore be very inefficient.

   Instead we will have to *impute* the missing data, i.e. use the observed data to infer or impute the values of the missing data. An example of an imputed data-set is displayed in columns 7 to 11 in the table below. Note that the imputed and observed data-sets coincide on the observed values. (We will see one way to impute the missing data below.) In fact, a common approach for handling missing data is a Bayesian approach known as *multiple imputation.* Multiple imputation imputes several complete data-sets. We then analyze each imputed data-set separately as if it was the true data-set, and combine the results appropriately. We will not discuss multiple imputation here but note that it has some similarities with the bootstrap approach we describe below.

	Observed Data					An Imputed Data-set				
Student	**A**	**B**	**C**	**D**	**E**	**A**	**B**	**C**	**D**	**E**
1	?	63	65	70	63	56.21	63	65	70	63
2	53	61	72	64	73	53	61	72	64	73
3	51	67	65	65	?	51	67	65	65	58.94
4	?	69	53	53	53	47.96	69	53	53	53
5	?	69	61	55	45	48.46	69	61	55	45
6	?	49	62	63	62	49.96	49	62	63	62
7	44	61	52	62	?	44	61	52	62	51.69
8	49	41	61	49	?	49	41	61	49	46.94
9	30	69	50	52	45	30	69	50	52	45
10	?	59	51	45	51	42.46	59	51	45	51
11	?	40	56	54	?	39.54	40	56	54	44.33
12	42	60	54	49	?	42	60	54	49	48.19
13	?	63	53	54	?	46.21	63	53	54	50.99
14	?	55	59	53	?	45.21	55	59	53	49.99
15	?	49	45	48	?	36.87	49	45	48	41.66
16	17	53	57	43	51	17	53	57	43	51
17	39	46	46	32	?	39	46	46	32	37.69
18	48	38	41	44	33	48	38	41	44	33
19	46	40	47	29	?	46	40	47	29	37.44
20	30	34	43	46	18	30	34	43	46	18
21	?	30	32	35	21	20.46	30	32	35	21
22	?	26	15	20	?	9.87	26	15	20	14.66

Note: *Left panel:* 22 students have each taken 5 exams, labeled A, B, C, D, and E. Some of the scores for A and E, indicated by "?", are missing. *Right panel:* The missing data have been imputed from a two-way additive model. The full data set, taken from Mardia, Kent, and Bibby (1979), appears in table 1 of Efron (1992a).

Load the data-set from the file *Efron94_MissingData-Bootstrap.csv* and then answer the following questions. (Note that you will only need to use the data in columns 2 to 6.)

(a) Write a piece of code that uses a two-way linear model to estimate the missing data. A two-way linear model assumes

$$o_{ij} \approx \nu + \alpha_i + \beta_j. \tag{1}$$

Note that you can easily fit (1) using the second baseline estimator from the *Dimension Reduction Techniques* slides! (8 marks)

(b) Use your code to construct an imputed data-set and use it compute $\widehat{\theta}$, the estimated largest eigen-value of the variance-covariance matrix $\mathbf{\Sigma}$. What is your imputed value of $\widehat{\theta}$? (8 marks)

Solution: Depending on how exactly you estimated (1) to impute the missing data, you should have obtained a value of $\widehat{\theta} \approx 652$. (Other similar values, e.g. $\widehat{\theta} \approx 630$ are also possible if you estimated (1) via least-squares.)

(c) In order to estimate the uncertainty in $\widehat{\theta}$, run a bootstrap analysis where each bootstrap simulates 22 samples *with replacement* of the rows in the original data-set. For each of $B = 2000$ bootstrapped data-sets use your code from parts (a) and (b) to impute the missing values and estimate $\widehat{\theta}_b$ for $b = 1, \ldots, B$. (8 marks)

(d) Use your bootstrapped samples $\widehat{\theta}_b$ for $b = 1, \ldots, B$ to estimate the bias in $\widehat{\theta}$ and construct an approximate 95% confidence interval for θ. (8 marks)

Solution: The bias is ≈ -30 but values of ≈ -20 are also possible. (Many of you stated the bias was ≈ 30 but this is wrong! The sign matters and you should have noted the bias was negative.)

The approximate 95% confidence interval you obtained will also have depended on the specific method you used to estimate (1) as well as whether or not you accounted for the bias (as discussed in class) in your CI. You should have obtained an approximate 95% CI of [250, 1,100] but this could easily be off by 100-150 so a CI of [350, 1,300] would also have been reasonable.

Note that at least one of you suggested that the imputation method was a poor method because the approx 95% CI was so wide. This is **not true**. You might have the best imputation method in the world (based on the true unknown missing data mechanism) but if there is a lot of noise in the data then there is nothing you can do about having wide CI's.)

(e) Note that this bootstrap approach depends on (1) being a good imputation method. Can you suggest any improvements to (1)? For example, should we consider adding some noise to (1)? (8 marks)

Solution: Yes, even assuming the missing data was MCAR or MCAR (see below), other imputation methods could have been used including:

i. Maximum likelihood estimation (MLE) using the so-called EM algorithm. (You're not expected to be familiar with the EM algorithm!)

ii. (1) is likely to underestimate the noise in the data. This could have been addressed by using the model

$$o_{ij} \approx \nu + \alpha_i + \beta_j + \epsilon_{ij} \tag{2}$$

instead of (1) and where the ϵ_{ij}'s represent random noise. Model (2) could be fit using standard regression methods. Let $\widehat{\sigma}_j^2$ be our estimate of $\mathrm{Var}(\epsilon_{ij})$ for all i and let $\widehat{\nu} + \widehat{\alpha}_i + \widehat{\beta}_j$ be the estimated parameters from the regression. Missing values x_{ij} could then be imputed as $\tilde{x}_{ij} \approx \widehat{\nu} + \widehat{\alpha}_i + \widehat{\beta}_j + z_{ij}$ where the z_{ij}'s are generated as $\mathrm{N}(0, \widehat{\sigma}_j^2)$.

iii. We could also account for correlations in the z_{ij}'s. Can you see how you might do this?

Remark: Imputing missing data always involves some assumption regarding the pattern of *missingness*. For example, if the missing exam scores for A and E are missing because the students failed the exams then this pattern would correspond to *missing not at random*

(MNAR), and would require us to model the *missingness mechanism* when computing $\widehat{\theta}$ in any of the bootstrapped data-sets. In this case (1) would be a very poor model and the resulting inference would be inaccurate and misleading.

On the other hand, if the missing data is *missing completely at random* (MCAR) then there is no need to model the missingness mechanism. Unfortunately, MCAR data is quite rare. Instead we hope our missingness pattern is *missing at random* (MAR) which lies between the MNAR and MCAR assumptions. MAR means that the missingness mechanism depends on the *observed* data. Ultimately domain specific knowledge will be required to tell which of MNAR, MCAR or MAR is the appropriate assumption. Our imputation method, i.e. (1), (or the other ones that we discussed in part (e)) should work reasonably well for data that's MCAR or MAR.

Note that the 1994 paper by Efron outlines an alternative bootstrap approach which requires modeling the missing-data mechanism.

---

4. **Collaborative Filtering – ENTIRELY OPTIONAL!**
You do not need to do this question and will not receive any credit if you do. I only include it here as the `MovieLens` data-set is a famous machine-learning data-set and you now have the tools to tackle this question! (Maybe you'll try it after the exams or over the summer when you have nothing better to do :-))

Download the `Excel` workbook *Assignment_MovieData.xlsx*. The workbook contains 100k movie ratings from the `MovieLens` data-set. The data consists of ratings from 1 to 5 from a total of 943 users on 1682 movies. These ratings are split into "train" and "test" work-sheets, respectively. The "test" worksheet contains 9,430 observations with exactly 10 ratings from each user.

Code is not provided as part of the solutions for this question. That said, the answers are provided below so that you can compare your answers with them if at some point you decide to do the question.

(a) Construct the baseline estimator where we use the average rating (across all ratings in the training data), $\bar{\mathbf{x}}$, as our estimator. What is the test error for this estimator? (Here and in the other parts below we mean RMSE when we refer to (test) error.)

Solution: We found $\bar{x} = 3.5238$, with a test error of 1.1220.

(b) Now construct biases for each movie and user according to

$$b_i := \frac{\sum_u x_{ui}}{M_i} - \bar{x} \tag{3}$$

$$b_u := \frac{\sum_i x_{ui}}{M_u} - \bar{x} \tag{4}$$

where $M_i = \#$ users that rated movie i and $M_u = \#$ movies rated by user u. The new baseline estimator is

$$\hat{x}_{ui} = \bar{\mathbf{x}} + b_u + b_i.$$

What is the test error of this estimator?

Solution: We obtained a test error of 0.9919.

(c) Repeat part (b) but now use regularization and validation on the test set to choose the biases. That is, solve

$$\min_{b_i, b_u} \sum_{(u,i)} (x_{ui} - \hat{x}_{ui})^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 \right). \tag{5}$$

where the sum is over observations (u,i) in the training data and choose $\lambda \geq 0$ to be that value which gives the best performance on the test set. What is the test error here? (Note that we are really using the test set as a validation set here and in part (e) below.)

Hint: Note that (5) is an unconstrained concave optimization problem and the first order conditions will be sufficient to find the global optimum. You can check that these first order conditions (for user u and movie i) are:

$$b_u = \frac{\sum_{i:i \text{ rated by } u} (x_{ui} - b_i) - M_u \bar{x}}{\lambda + M_u} \tag{6}$$

$$b_i = \frac{\sum_{u:u \text{ rated } i} (x_{ui} - b_u) - M_i \bar{x}}{\lambda + M_i} \tag{7}$$

Note that (6) and (7) is a system of $M + I$ linear equations in $M + I$ unknowns (where $M = \#$ of movies and $I = \#$ of users) and will have a unique solution for any $\lambda > 0$. You can either solve this system directly or by using (6) and (7) to construct an iterative scheme.

Solution: The best value of λ that we found was $\lambda = 3.9$ with a corresponding test set error of 0.9585. A graph of the test error as a function of λ is displayed below.

(d) Use your best estimator from parts (a), (b) and (c) to construct the residual matrix, $\tilde{\mathbf{X}}$.

(e) Now use a neighborhood method (as described in the slides) applied to the residual matrix to construct a new estimator of the form

$$\hat{x}_{ui}^N = \bar{\mathbf{x}} + b_u + b_i + \frac{\sum_{j \in \mathcal{L}_i} d_{ij}\, \tilde{x}_{uj}}{\sum_{j \in \mathcal{L}_i} |d_{ij}|}$$

where $\mathcal{L}_i$ denotes the neighborhood of movie i and the d_{ij}'s are as defined in the slides. You can choose L, the size of a neighborhood, via validation on the test set. What is error on the test set now?

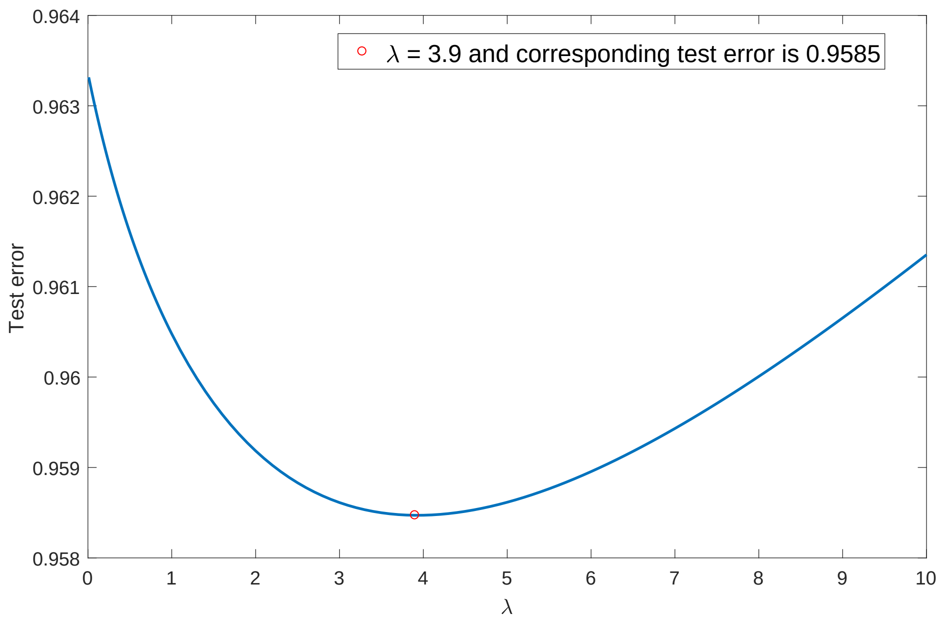

Figure 1: Test error as a function of λ

Remark: You could also try playing around with some of the matrix factorization methods (instead of the neighborhood method) for part (e). (They both yield similar results.)