CMPE 482 Final Exam

Spring 2021

Instructions

Please read the instructions carefully.

- **Deadline**: This exam is due June 27th, 2021, 23:00.
- **Exam format**: This exam is in *take-home* format. Unless otherwise specified, all previous instructions for the class assignments apply for this exam as well.
- **Grading**: Each question is equally weighted. The grading will be out of 100 points.
- **Submission format**: Similar to the assignments, you are expected to submit a single `.ipynb` file as your submission. Styling and naming conventions for the assignments apply (replace `_assignment_x.ipynb` with `_final_exam.ipynb`).
- **Data files**: All the data files mentioned in the exam are in the `data/` folder.
- **Mathematical writing**: For questions that require mathematical derivation and/or arguments, please use the math environment in your notebook.
- **Implementation**: For questions that require coding, central functionalities regarding any question should be implemented manually. The rest can be used from previously accepted resources or open source packages. Unless otherwise specified, all instructions for the assignments apply, including those regarding software requirements.
- **References**: Any specific resource used must be cited, including our course textbooks.
- **Bonus questions**: A few questions will be marked with "(Bonus)". You do not lose points by omitting them but can earn extra points by solving them. Maximum exam score will not exceed 100, however.
- **Notation**: In the questions below, unless otherwise specified, scalars are represented with standard letters e.g. $a, N \in \mathbb{R}$, vectors are represented with lower case bold letters e.g. $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d$, and matrices are represented with upper case bold letters $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times d}$. When a subscripted letter is bold face, it corresponds to a vector in a sequence or list of vectors (e.g. $\mathbf{x_k}$ being the k'th iteration of the parameter in gradient descent). Otherwise it corresponds to an indexed element of a vector (e.g. x_i being the i'the element of $\mathbf{x}$).

Questions

1. Suppose you are given two full rank matrices $\mathbf{A} \in \mathbb{R}^{k\times n}$ and $\mathbf{B} \in \mathbb{R}^{k\times m}$. Using the QR factorization, describe how to obtain an orthogonal basis matrix $\mathbf{U}$ for the subspace at the intersection of range($\mathbf{A}$) $\cap$ range($\mathbf{B}$).

   (a) Write a python program `verify` to verify that a given set of points (as columns of a matrix X) are in the intersection subspace.

   (b) Write a python program `intersect` to find the basis matrix $\mathbf{U}$, given $\mathbf{A}$ and $\mathbf{B}$. Your program should determine the size of $\mathbf{U}$ automatically.

   (c) Use `numpy.load` to load the matrices `Q1_A1.npy` and `Q1_B1.npy` as Test Case 1 and `Q1_A2.npy` and `Q1_B2.npy` as Test Case 2 to conduct a test of your algorithm. For each test case, print the basis vectors of the intersection, and for all your basis vectors demonstrate that your `verify` function works correctly as well. Hint: Make sure you can handle the case where the dimensionality of the intersection is 0.

2. Consider the gradient descent algorithm for solving a least squares problem $\mathbf{Ax} \approx \mathbf{b}$.

   (a) Given a fixed learning rate η, write the algorithm in the form of

   $$\mathbf{x}_t = f(\mathbf{x}_{t-1})$$

   where f is an affine function of form $f(\cdot; \mathbf{A}, \mathbf{b}, \eta)$.

   (b) To show that the algorithm converges for a particular learning rate η, one can think of the least square solution $\mathbf{x}^*$ and measure the distance of $\mathbf{x}_t$ to $\mathbf{x}^*$ during the course of the algorithm. In particular, if we can show that f is a contraction, i.e. for any two different parameters $\mathbf{x}$ and $\mathbf{x}'$ in the domain we have

   $$\|f(\mathbf{x}) - f(\mathbf{x}')\| \leq L_\eta \|\mathbf{x} - \mathbf{x}'\|$$

   where $L_\eta < 1$, then the distance shrinks. Compute L_η and write a python program that, for a given least square problem as $\mathbf{A}$ and $\mathbf{b}$, prints the critical η^* over which the algorithm will diverge and under which the algorithm will converge.

   (c) Read `Q2_A1.npy` and `Q2_b1.npy` as Test Case 1 and `Q2_A2.npy` and `Q2_b2.npy` as Test Case 2 and apply your algorithm. Print the critical $\eta*$ for both test cases.

   (d) (Bonus) Estimate and plot the number of steps needed to converge to the optimum as a function of η, given a fixed starting point $\mathbf{x}_0$. Do this for both test cases.

3. In this question, we start by describing an algorithm that changes two N-dimensional vectors iteratively. Let us call these vectors $\mathbf{x}^{(0)}, \mathbf{y}^{(0)} \in \mathbb{R}^N$ where (0) denotes the fact that these are the initial versions of these vectors. Let the sums of $\mathbf{x}^{(0)}, \mathbf{y}^{(0)}$ be equal to zero, i.e. $\sum_{n=1}^N x_n^{(0)} = \sum_{n=1}^N y_n^{(0)} = 0$. We apply the following iterative updates on these vectors:

Algorithm 1:

```
for $k \in \{1, \dots, K\}$ do
    for $n \in \{1, \dots, N\}$ do
        $x_n^{(k)} \leftarrow \frac{x_n^{(k-1)} + x_{n+1}^{(k-1)}}{2}$
        $y_n^{(k)} \leftarrow \frac{y_n^{(k-1)} + y_{n+1}^{(k-1)}}{2}$
    end for
    $\mathbf{x}^{(k)} \leftarrow \frac{1}{\|\mathbf{x}^{(k)}\|}\mathbf{x}^{(k)}$
    $\mathbf{y}^{(k)} \leftarrow \frac{1}{\|\mathbf{y}^{(k)}\|}\mathbf{y}^{(k)}$
end for
```

where indexing of the vectors is circular, that is, $x_{N+1}^{(k)}$ and $y_{N+1}^{(k)}$ are assumed to be equivalent to $x_1^{(k)}$ and $y_1^{(k)}$ for ease of notation. For each value of k, we also define a polygon $\mathcal{P}^{(k)}$ with vertices $\mathbf{p}_1^{(k)}, \dots, \mathbf{p}_N^{(k)}$, where each vertex is a 2D point whose coordinates are determined by $\mathbf{x}^{(k)}$ and $\mathbf{y}^{(k)}$, or more precisely $\mathbf{p}_n^{(k)} \equiv (x_n^{(k)}, y_n^{(k)})$.

(a) Let $N = 20$, initialize $\mathbf{x}^{(0)}$ and $\mathbf{y}^{(0)}$ randomly, and make sure that their element-wise sums are equal to zero. Run Algorithm 1 on them for $K = 1000$ iterations. For each $k \in \{1, 5, 10, 20, 40, 60, 100, 200, 400, 1000\}$, plot the corresponding polygon $\mathcal{P}^{(k)}$.

(b) Describe what this algorithm does to polygon $\mathcal{P}^{(k)}$ in each iteration.

(c) Show that the operations inside the outer loop of Algorithm 1 can be summarized by linear transformations in the form of

$$\mathbf{x}^{(k+1)} = \frac{1}{a^{(k)}}\mathbf{M}\mathbf{x}^{(k)}$$
$$\mathbf{y}^{(k+1)} = \frac{1}{b^{(k)}}\mathbf{M}\mathbf{y}^{(k)}$$

(d) Let $\lambda_1, \lambda_2, \dots, \lambda_N$ be the eigenvalues of matrix $\mathbf{M}$. Show that $\mathbf{M}$ has distinct eigenvalues of form $\lambda_n = (1 + e^{i2\pi n/N})/2$. Also, find a general formula for the eigenvector $\mathbf{v}_n$ corresponding to eigenvalue λ_n.

(e) Show that the eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N$ constitute an orthogonal basis for $\mathbb{R}^N$.

(f) Observe that the random polygon at the beginning converges to an ellipse in each iteration. Prove that as $K \to \infty$, $\mathcal{P}^{(K)}$ converges to a special polygon whose vertices lie on an ellipse.

4. Clustering is a widely used unsupervised learning technique in machine learning, whose goal is grouping similar objects into same partitions (for a more comprehensive understanding, the readers are referred to *IALA*, Chapter 4)[1]. For this question, your task will be clustering a set of points into two groups using constrained nonlinear least squares techniques.

[1] http://vmls-book.stanford.edu/

Let $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$ be a set of points where each $\mathbf{x}^{(n)} \in \mathbb{R}^D$. Let also $c^{(1)}, \dots c^{(N)}$ be binary variables each of which indicates the cluster id of the corresponding point, i.e. $\mathbf{x}^{(n)}$ is in cluster 1 if $c^{(n)} = 1$, otherwise it is in cluster 0. Let $\mathbf{c} = (c^{(1)}, \dots c^{(N)})$ be the vector that contains these cluster id's. Then, one can express the clustering problem as the following constrained nonlinear least squares problem

$$\begin{array}{lll} \text{minimize} & \|f(\mathbf{c})\|^2 = \sum_{d=1}^{D} f_d^2(\mathbf{c}) & \\ \text{subject to} & g_n(c^{(n)}) = 0, & n = 1, \dots N \end{array}$$

where $f_d(\cdot)$ and $g_n(\cdot)$ can be defined as follows

$$f_d(\mathbf{c}) = \sum_{n=1}^{N} c^{(n)} \left| x_d^{(n)} - \frac{\sum_{m=1}^{N} c^{(m)} x_d^{(m)}}{\sum_{m=1}^{N} c^{(m)}} \right| + \sum_{n=1}^{N} (1 - c^{(n)}) \left| x_d^{(n)} - \frac{\sum_{m=1}^{N} (1 - c^{(m)}) x_d^{(m)}}{\sum_{m=1}^{N} (1 - c^{(m)})} \right|$$

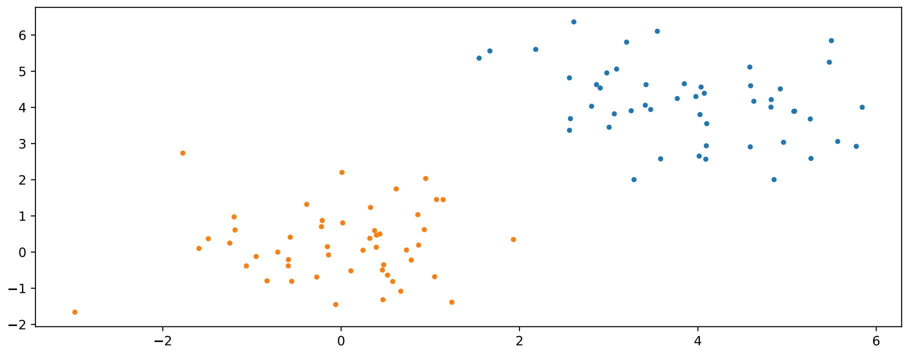
$$g_n(c^{(n)}) = (c^{(n)} - a)^2 + b$$

(a) In order to ensure $c^{(n)} \in \{0, 1\}$, what should be the values of a and b?

(b) Explain the objective of $f_d(\cdot)$. How does the minimization of $\|f(\mathbf{c})\|$ help partitioning data points into two groups?

(c) We will now study this problem on a given data set. Read the set of points from `Q4_points.csv`, and visualize them in a scatter plot.

(d) Solve the optimization problem using *augmented Lagrangian algorithm* (*IALA*, Algorithm 19.2). Print the final value of $\mathbf{c}$, and visualize the resulting clusters in a colored scatter plot as in the figure below:



Note: You are allowed to use JAX for calculating gradients and Jacobians.

5. Dimensionality reduction is a commonly conducted operation in machine learning for many purposes including feature extraction, visualization, or avoiding multicollinearity. Principal components analysis (PCA) is one of the frequently used dimensionality reduction methods, and in this question, we are going to investigate the close relationship between PCA and SVD.

PCA is a method that relies on the projection of the data to a linear subspace. The orthonormal basis vectors spanning this linear subspace are called *principal components*, and the subspace itself is called the *principal subspace*. Given K

principal components[2] $\mathbf{a}_1, \ldots \mathbf{a}_K$, each observation vector $\mathbf{x}$ is projected to a new space using the following linear transformation

$$\mathbf{z} = \underbrace{\begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_K^T \end{bmatrix}}_{\mathbf{A}^T} \mathbf{x} \tag{1}$$

where the projection results in K coordinates. Thus, $\mathbf{x}$ can be represented by the vector $\mathbf{z}$ of size K.

How to select the principal components in PCA? One common derivation of PCA is the "maximum variance formulation". In this line of reasoning, we want to project the data to a linear subspace that *maximizes the variance* of the projected data, so that it retains as much "information" as possible. We will investigate this formulation of the PCA[3].

In this setting, our observations are denoted by $\mathbf{x}_n \in \mathbb{R}^D, n \in \{1, \ldots, N\}$. For simplicity assume that our data has a mean of $\mathbf{0}$, that is, $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n = \mathbf{0}$. If not, we can always *center* the data by subtracting $\bar{\mathbf{x}}$ from each observation. In this formulation, projection to K-dimensional subspace can be written as $\mathbf{z}_n = \mathbf{A}^T\mathbf{x}_n$.

As mentioned earlier, the objective of PCA is maximization of total variance in the projected subspace, where the total variance can be defined as follows:

$$\sigma^2(\mathbf{A}) \equiv \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{z}_n\|^2 = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{A}^T\mathbf{x}_n\|^2 \tag{2}$$

with the orthonormality constraint of $\mathbf{A}$, i.e. $\mathbf{A}^T\mathbf{A} - \mathbf{I} = \mathbf{0}$. For instance, if we were projecting the data onto 1-dimensional linear subspace, the variance of the resulting one dimensional data set would be given by

$$\sigma^2(\mathbf{a}_1) = \frac{1}{N}\sum_{n=1}^{N} (\mathbf{a}_1^T\mathbf{x}_n)^2 = \frac{1}{N}\mathbf{a}_1^T\mathbf{X}^T\mathbf{X}\mathbf{a}_1 \tag{3}$$

where $\mathbf{X}$ denotes a $N \times D$ data matrix, where each observation $\mathbf{x}_n$ is stored in a row. Together with the orthonormality constraint $\mathbf{a}_1^T\mathbf{a}_1 - 1 = 0$, we find a Lagrangian in the form of

$$\mathcal{L}_1(\mathbf{a}_1, \lambda_1) = \sigma^2(\mathbf{a}_1) + \lambda_1(1 - \mathbf{a}_1^T\mathbf{a}_1),$$

where λ_1 is the Lagrange multiplier that enforces $\mathbf{a}_1$ to be an unit vector.

[2] In a D-dimensional space, the number of principal components to be used can range between 1 and D. We will not be discussing the selection methodologies for K here, and will assume it is a given number. However, informally, when choosing K the aim is usually using as little number as dimensions possible without losing too much information.

[3] An alternative formulation for PCA corresponds to aiming for the least (Euclidean) distance or error between the original points and the projected points, and results in exactly the same algorithm.

(a) Let's define the gram-matrix $S \equiv \frac{1}{N}\mathbf{X}^T\mathbf{X}$. What is the condition that $\mathbf{a}_1$ and λ_1 must satisfy in order to maximize $\mathcal{L}_1(\mathbf{a}_1, \lambda_1)$? Hint: Do they have an eigenvector/eigenvalue relationship for a known matrix?

(b) How/where do $\mathbf{a}_1$ and λ_1 appear in the SVD of $\mathbf{X}$?

(c) Derive a similar Lagrangian formula for the case $K = 2$, and show the relationship between $\mathbf{a}_2$ and the SVD of $\mathbf{X}$. Informally generalize your results for $K > 2$ principal components, as well.

(d) Use the **Code Snippet 1** below to load 400 images from the Olivetti Faces Data Set[4] into the variable $\mathbf{D} \in \mathbb{R}^{400\times 64\times 64}$. Reshape $\mathbf{D}$ into a matrix $\mathbf{X} \in \mathbb{R}^{400\times 4096}$ where each picture is flattened (in a row-major fashion) and stacked as a row of the matrix.

(e) Conduct an SVD of the matrix $\mathbf{X}$ (you may use a publicly available package function for this). Print the first 10 elements of the first principal component $\mathbf{a}_1$.

(f) Take the first image in the data set $\mathbf{x}_1$, and obtain and print $\mathbf{z}_1$ for $K = 3$.

(g) Use the full SVD results to find the first three "archetypal" face schemas along which human faces vary the *most*, and depict them in the form of face images (do not forget to reshape them correctly before visualizing). Also depict the three "archetypal" face schemas along which the human faces vary the least.

Hint: Which objects discussed above correspond to these "archetypes"?

6. Examine *orthogonal distance regression* from *IALA*, Pg. 400. In this question, we will examine the relationship between ordinary least squares regression (OLS), principal components analysis (PCA), and orthogonal distance regression (ODR).

(a) Read the bivariate data set `Q6_bivariate.csv`. The first column will be the 'input' variable and the second column will be the 'output' variable. Plot a scatter plot of the data with the input variable being in the horizontal dimension.

(b) Conduct a first order OLS (using only the input feature and the intercept) for the data provided. Find the solution analytically - do not use a statistical package. Plot your solution on top of a scatter plot of the data.

(c) Use the input and output features to create a $N \times 2$ matrix $\mathbf{X}$. After *centering* the data set, conduct a PCA of the data (through SVD), and obtain the first principal component. Now, in order to use the linear space spanned by this basis vector as a summary of the original data, decenter this line based on your original transformation (converting it into an affine space). Plot the resulting line on top of the scatter plot of the data. Hint: This "solution" should look roughly similar to the OLS solution, but not identical.

[4]https://scikit-learn.org/0.19/datasets/olivetti_faces.html

(d) We will now use `scipy.odr` module to conduct a first order ODR of the data set (we are again treating the data as 'input' and 'output'). Examine the **Code Snippet 2**. The parameter `wd` in the code is a weight coefficient for the second term in the ODR objective function on Pg. 400. Using and filling in the code provided, find the ODR solution for `wd` $= 1, 2, 10$, and 100, and plot these on top of the scatter plot.

(e) Which one of the ODR solutions is the closest to your OLS solution? Which one is the closest to the one we obtained through PCA? Why is this the case? Explain.

Hints: Remembering the "minimum error formulation" of PCA could be helpful in the discussion part of this question (see footnote above). Another useful question is, what differentiates (or not) between 'input' and 'output' when conducting ODR?

(f) (Bonus) Imagine a regression with 2 or more input variables. What would the relevant ODR solution correspond to in the PCA solution?

Code Snippets

Code Snippet 1

```
from sklearn import datasets
faces = datasets.fetch_olivetti_faces()
D = faces["images"]
```

Code Snippet 2

```
from scipy import odr
def f(B, x):
    return B[0]*x + B[1]
linear_model = odr.Model(f)
data = odr.Data(x=, y=, wd=) # fill these values appropriately
odreg = ODR(data, linear_model, beta0=[0., 0.])
output = odreg.run()
slope, intercept = output.beta
```