Final project

MAST 387 Data Science Lab
Concordia University, Fall 2024

Due date: Wed, December 18, 2024, 23:59 ET, **NO extension will be assigned.**

Instructions

Please read this instruction carefully and complete your exam following all the instruction. If you don't follow these instructions, you will loose marks.

- This assignment should be completed and submitted **individually**.
- Coding generated by ChatGPT, or other AI generators will not be accepted. **The minimum penalty for generating or plagiarizing code is a zero grade for your work.**
- You should **upload one .ipynb file AND one .html (or .pdf) file on Moodle following the instructions**:
    1. **one single .ipynb file**, created following these steps:
        1. You should start with a markdown cell, which includes the title of the assignment, your name and student ID. For example, using the following formate in markdown cell
           ```
           # MAST 387 Final Project
           ## Student:  Your Full Name (ID 12345678)
           ```
        2. For each question and subquestion, you should start with a markdown cell, which includes the question number as the title. Then add a code cell to include your code. You will have to run the code and include the output of each subquestion. For example, using the following formate in markdown cell
           ```
           ## Question 1
           ### Question 1-1
           ```
           code for Q1-1 (in code cell)
           outputs for Q1-1 (out cell)
           ```
           ### Question 1-2
           ```
           code for Q1-2 (in code cell)
           outputs for Q1-2 (out cell)
        3. Coding in each question have to be run successfully and individually, for example, you must import the needed module for each question (not for sub-questions).

4. Rename your final ipynb file as "MAST387_ Midterm_ YourName_ StudentID.ipynb".

2. **one .html file** corresponding to your final .ipynb file with name "MAST387_ Midterm_ YourName_ StudentID.html'.

- Your answers should not only include code, you should give comments of your code if necessary. You need to give interpretation or mathematical equations (use math mode) of the output in a markdown cell. For example, I run a regression, and print the plot, you should write down your regression model, then give a clear interpretation of what this regression plot means, what results you can tell based on it.

- The data in this exam are intended for this exercise only. Individuals analyzing the data for other purposes are not allowed.

Important remarks

- Your code should be readable (e.g., by giving meaningful names to variables and functions) and properly commented where appropriate. Your code should also be efficient. For example, vectorized operations are always better than loops.

- Unless stated otherwise, you are only allowed to use `NumPy`, `pandas`, `matplotlib`, `seaborn` or the libraries explicitly mentioned in each problem, nothing more!

- For problems involving visualization, marking will take into account the appropriate choice of plot, its clarity and overall quality.

- For each subquestions, you have to run your code and include the outputs in your .ipynb file. If you only write code without run it and include the outputs of your code, I will assign zero grade for the question.

Good luck!

Problem A - Sparse recovery

This exercise deals with *sparse recovery*, a popular signal processing technique that allows one to reconstruct signals (such as, e.g., images) from linear measurements. It has applications in many areas, including medical imaging (e.g., MRI). A vector is called s-sparse if it has at most s nonzero entries. For example, the vector $(1, 0, 1, 0, 0, 2, -3, 0, 0)^\top \in \mathbb{R}^9$ is 4-sparse. It is also, for example, 5- and 6-sparse, but not 3-sparse.

1. Create a function `sparse_data()` that takes as input integers s, N, and d, and returns a dataset of d random N-dimensional s-sparse vectors. The s-sparse vectors composing the dataset should be generated independently as follows. For each vector, first generate a random subset of s unique elements from $\{1, \ldots, N\}$. Then, populate the corresponding entries with s independent random variables distributed according to the standard distribution $N(0, 1)$.

2. Print the output of `sparse_data(s=4, N=10, d=5)`.

We will now consider two algorithms for signal recovery from linear measurements. Assume $x \in \mathbb{R}^N$ to be a ground truth signal and let $A \in \mathbb{R}^{m \times N}$ be a measurement matrix. Then, the measurement of x is given by the vector $y = Ax \in \mathbb{R}^m$. The objective of signal recovery is to compute an approximation $\hat{x}$ to x, from the measurement data A and y.

3. Implement the *Matching Pursuit (MP)* algorithm with a function `MP()`, defined as follows.

   Inputs: measurement matrix $A \in \mathbb{R}^{m \times N}$ (assumed to have columns with unit 2-norm), measurement vector $y \in \mathbb{R}^m$, and number of iterations K.

   Algorithm: The MP algorithm produces a series of approximations $x_0, \ldots, x_K \in \mathbb{R}^N$ iteratively defined as follows for every $k = 0, 1, \ldots$:

   $$\begin{aligned} r_k &= y - Ax_k, \\ j_k &= \text{any index } j \in \{1, \ldots, N\} \text{ that maximizes the quantity } |a_j^\top r_k|, \\ x_{k+1} &= x_k + (a_{j_k}^\top r_k) e_{j_k}, \end{aligned}$$

   where $x_0 = 0$, $a_j \in \mathbb{R}^m$ is the jth column of A, and $e_j \in \mathbb{R}^N$ is the jth vector of the canonical basis of $\mathbb{R}^N$ (i.e., the jth column of the $N \times N$ identity matrix).

   Output: A matrix $X \in \mathbb{R}^{N \times (K+1)}$ containing the iterates $x_0, \ldots, x_K$ as columns.

   Given measurements $y = Ax$, the approximation to the signal x produced by MP is the vector $\hat{x} = x_K$.

4. Test the MP algorithm on the data stored in `sparse_A.npy` and `sparse_y.npy` with $K = 2s$, where s is the number of nonzero entries of the ground truth signal x, stored in `sparse_x.npy`. Visualize the ground truth signal x and its MP approximation $\hat{x}$ in the same plot using `matplotlib.pyplot.stem()`. Compute the relative approximation error with respect to the 2-norm, i.e., $\|x - \hat{x}\|_2 / \|x\|_2$.

5. In the same setting as part 4, plot the relative approximation error as a function of the iteration k. What do you observe?

6. Implement the *Orthogonal Matching Pursuit (OMP)* algorithm with a function `OMP()`, defined as follows.

   Inputs: Same as `MP()`.

   Algorithm: The OMP algorithm produces a series of approximations $x_0, \dots, x_K \in \mathbb{R}^N$ iteratively defined as follows for every $k = 0, 1, \dots$:

$$\begin{aligned} r_k &= y - Ax_k, \\ j_k &= \text{any index } j \in \{1, \dots, N\} \text{ that maximizes the quantity } |a_j^\top r_k|, \\ S_{k+1} &= S_k \cup \{j_k\}, \\ z_{k+1} &= \text{ solution to the least squares problem } \min_z \|A_{S_{k+1}} z - y\|_2^2, \\ x_{k+1} &= \begin{cases} z_{k+1} & \text{for entries in } S_{k+1} \\ 0 & \text{otherwise} \end{cases}, \end{aligned}$$

   where $x_0 = 0$, $S_0 = \emptyset$, A_S denotes the restriction of A to the columns in S (hence it is an $m \times |S|$ matrix), and $a_j \in \mathbb{R}^m$, $e_j \in \mathbb{R}^N$ are as in MP. Note that $S_0, S_1, \dots$ is a sequence of subsets of $\{1, \dots, N\}$.

   Use `numpy.linalg.lstsq()` to solve the least squares problem.

   Output: Same as `MP()`.

7. Repeat part 4 for OMP.

8. Compare the convergence of MP and OMP as a function of the iteration. Which method performs better?

Problem B - Image compression

This problem deals with image compression. We will compare the SVD-based compression we studied in class with a different method based on *wavelet decomposition*.

1. Load the image from the file `image_phantom.bmp` and visualize it. This is the Shepp-Logan phantom (`https://en.wikipedia.org/wiki/Shepp-Logan_phantom`). It is a simplified model of the human head used in medical imaging tests.

2. **[1 mark]** Using `scikit-image`, resize the image to a 64×64 size (without using antialiasing), store it in a matrix called `img` and visualize it. From now on, we will work with this resized 64×64 version.

3. Compute the SVD of `img` and study its singular values. What do you observe?

4. How many singular values do you need to explain 95% of the image variance? Plot the corresponding compressed image.

We now consider a different compression method based on wavelet decomposition. This yields an efficient multi-level decomposition of an image. We will use specific wavelets called *Haar* wavelets.

Before proceeding further, install the package `PyWavelets` (see `https://pywavelets.readthedocs.io/en/latest/index.html`). Assuming that your image is stored in a `NumPy` array called `img`, you can perform a wavelet decomposition of your image as follows:

```
import pywt
wave_deco = pywt.wavedec2(img, 'haar', mode='symmetric', axes=(-2, -1))
```

Vice versa, given a wavelet decomposition `img_deco`, you can reconstruct the image as follows:

```
img_reco = pywt.waverec2(wave_deco, 'haar', mode='symmetric', axes=(-2, -1))
```

5. Compute the wavelet decomposition of `img` and then its reconstruction `img_reco`. Plot `img_reco` and compute the reconstruction error, given by the Frobenius norm of the difference `img` $-$ `img_reco`. Is the reconstruction error acceptable?

6. Now, we want to visualize the wavelet decomposition of `img`. This will require some data manipulation since the output `wave_deco` of `pywt.wavedec2()` is stored in a very specific format. In fact, `wave_deco` is a tuple organized as follows:

   - The element `wave_deco[0]` is a 1×1 `NumPy` array containing the *scaling coefficient* of the image. This value stores most of the energy of the image.
   - The following elements of the tuple, i.e., `wave_deco[i]` with `i` > 0, are, in turn, tuples containing three $2^{i-1} \times 2^{i-1}$ `NumPy` arrays each. These represent, respectively, the *horizontal*, *vertical*, and *diagonal details* of the image at level `i`.[1]

[1] You can take a look at the official documentation for further details (see `https://pywavelets.readthedocs.io/en/latest/ref/2d-dwt-and-idwt.html`). However, it might be difficult to understand if you are not a wavelet expert. Here I am telling you all you need to know to solve the problem.

Display the horizontal, vertical, and diagonal details at levels $2, 4, 6$ of `img`.

7. Create a function called `from_deco_to_img()` that takes a wavelet decomposition `wave_deco` as input and returns as output a 2D `NumPy` array called `deco_img` constructed as follows:

   - The element $(0, 0)$ of `deco_img` contains the scaling coefficient.
   - For every level `i`, copy the horizontal, vertical and diagonal details of the image at level `i` into suitable $2^{i-1} \times 2^{i-1}$ submatrices of `deco_img`, as follows:
     - the submatrix corresponding to the rows from 0 (included) to 2^{i-1} (excluded) and the columns from 2^{i-1} (included) to 2^i (excluded) should contain the horizontal detail;
     - the submatrix corresponding to the rows from 2^{i-1} (included) to 2^i (excluded) and the columns from 0 (included) to 2^{i-1} (excluded) should contain the vertical detail;
     - the submatrix corresponding to the rows from 2^{i-1} (included) to 2^i (excluded) and the columns from 2^{i-1} (included) to 2^i (excluded) should contain the diagonal detail.

8. Test `from_deco_to_img()` on the `wave_deco` tuple obtained in part 5. Visualize the corresponding image `deco_img`. You should get an image with most of the information in the top left corner.

   To enhance visibility, visualize also a suitable transformation of `deco_img`.

9. Repeat part 8 a 32×32 resized version of `img`

10. **[1 mark]** Reshape the matrix `deco_img` into a vector called `deco_vec`. This vector represents the wavelet coefficients of the image. First, plot the entries of `deco_vec` using `matplotlib.pyplot.stem()`. Then, plot the *absolute* wavelet coefficients, sorted in descending order, using a curve plot. Do they seem to decay fast?

    The idea of wavelet compression (at the basis of the JPEG 2000 compression standard) is to only store the k largest wavelet coefficients and discard the rest (i.e., setting them to zero). This only requires storing $2k$ scalars (i.e., the coefficients and their positions) and can lead to dramatic memory savings.

11. Finally, let's create a plot to compare the decay of the absolute wavelet coefficients and the decay of the singular values of the image, as a function of the *memory used* (measured as the number of scalars stored) by each compression method. In the case of wavelet coefficients, plot the kth largest absolute wavelet coefficient as a function of the number of scalars needed to store k wavelet coefficients (see explanation in part 10). In the case of singular values, plot the kth singular value as a function of the amount of memory used to store the first k singular values and singular vectors.

    Which compression method seems to be most memory efficient?

Problem C - House pricing

In this problem, we will consider a dataset containing different features and the sale price of several houses. The files associated with this problem are `house_info.txt`, `house_train.csv` and `house_test.csv`.

1. Load the dataset `house_train.csv` into a DataFrame called `df_train`. In the following exercises, we will work with `df_train`, unless stated otherwise.

2. Provide an overview of `df_train`. How many categorical/numerical features are there? Which columns contain missing values? What percentage of the data is missing in those columns? Is there any other interesting aspect worth noting? Use suitable visualization strategies to illustrate.

3. Using an appropriate visualization strategy, show the distribution of `SalePrice` values. Define a new column called `SalePriceLog` containing the $\log_{10}$ transformation of `SalePrice`. What is the effect of this transformation on the distribution?

4. Plot `SalePrice` as a function of `YearBuilt`. What do you observe?

5. Define a DataFrame called `df_by_zone` that has `MSZoning` as index and `MSSubClass`, `YearBuilt`, `YearRemodAdd`, `GarageType`, `Fence`, and `SaleCondition` as columns. The values should be computed either as the mean value (for numerical variables) or the most frequent value (for categorical variables) of the corresponding column, for different values of `MSZoning`. Print the DataFrame.

6. Let's do some data cleaning. Create a DataFrame `df_train_filled` from `df_train`, where missing values are replaced with values that you deem appropriate. Explain the rationale behind your data-filling strategy, while taking into account the information provided in `house_info.txt`. Print the values of `df_train_filled` relative to the columns `PoolQC`, `MiscFeature`, `LotFrontage`, `GarageYrBlt` and `MasVnrArea` and the indices 234 and 287.

7. Plot `SalePriceLog` as a function of every other relevant numerical variable (i.e., excluding `SalePrice` and `Id`) and show the corresponding linear regression curves. From a visual inspection, what are the numerical variables that seem *most* linearly correlated with `SalePriceLog`?

8. Now, compute the correlation coefficient between `SalePriceLog` and every other relevant numerical variable. What are the 10 most correlated variables? Does the log transformation made to obtain `SalePriceLog` have any impact here?

9. What categorical features seem to be *least* correlated with `SalePriceLog`? Use a suitable visualization strategy to gain insights.

The last two parts of this problem involve regression and machine learning. You are allowed to use `scikit-learn`, `statsmodels` , `patsy`, and others.

10. Apply PCA (with 2 and 3 components) to the dataset `df_train`, restricted to columns with numerical features not containing any missing value and excluding `Id`, `SalePrice` and `SalePriceLog`. Visualize the first 2 and 3 principal components.

    Does PCA give any insights on `SalePrice` (or its log transformation)? Use a suitable visualization strategy to address this question. Note that visualizations might be negatively impacted by outliers. If this is the case, try solutions to alleviate this issue.

11. In this exercise, we want to predict the sale price of houses in the dataset `house_test.csv`. With this aim, you are invited to test different regression methods among those we explored in class (and beyond, if you want to!).

    You can conduct further data explorations to select the most relevant regression variables, make suitable transformations, deal with outliers, select hyperparameters, apply for L1 or L2 constraints, cross-validation, deal with missing values, etc. Experiment with different techniques! Print the predicted prices for the 11 houses in `house_test.csv`.

    Note that you are invited to experiment with different methods, but you can propose only *one* set of predicted values for the test houses based on the method you deem most reliable. Marking will particularly take into account presentation quality, the justification of your choices, and the accuracy of your predictions (note that the test set does not contain `SalePrice`).

Have a good holiday!