E0 298: Linear Algebra and Applications

Assignment 6

General Instructions:

- For each of the problems you can find the data in the `Data` folder shared with you. Traverse to the relevant folder, and choose the data corresponding to your number as per the shared record sheet. (Each data is different and if you choose the wrong data you will be penalized).
- `X` is your assigned index number on the record sheet wherever a name of a file is of the form `filename_X.txt`.
- You are supposed to submit a single **`Jupyter Notebook`** with all the solutions made into separate blocks. Use Python 3.10 or above as the kernel.
- No ML library other than `numpy` and `matplotlib` should be used, failing which will attract **0** marks.
- The final evaluation does not depend on the accuracy metrics but is based on the quality of your experiments and observations thereof.
- We will run a thorough plagiarism check on both your report and the codes. Any suspicion of copying would lead to a harsh penalty from negative marks in the assignment to a failing grade in the course, depending upon the severity. Therefore, kindly refrain from copying others' codes and/or reports.

1 Regression

1.1 Unconstrained

1. **Multilinear Regression** - You need to find the position of a particle in 3D space (y1, y2, and y3) given a 10-dimensional feature vector. The data file has name of the form `regression_data_X.txt` in the `lin_regression` folder. The features are readings from 10 sensors in the experiment environment. It is experimentally seen that the position of the particle depends linearly on these readings. Can you figure out the relationship?

2. **Generalised Regression with polynomial kernel** - Now, it turns out that the position only depends on the magnitude of the force along 2 basis vectors (features 1 and 2). That is, the recordings in Q1 are derived quantities from these two independent features and share a polynomial relationship. Hence, use a polynomial kernel to predict the position of the particle given the 2 features. The new data file has name of the form `poly_regression_data_X.txt` in the `poly_regression` folder.

3. You need to generate a correlation plot for each regression task, i.e., 3 independent plots for Q1 and Q2. Metrics - Pearson Correlation, Mean Squared Error, Mean Absolute Error (p-value is not required).

1.2 (Linear) Constrained

1. Modify the problem in Q1 by adding linear constraints from the `constraints_X.txt` file in the `constraints` folder. Please show in a MARKDOWN cell how you are using the formulae given in the lecture notes for this problem.

2 Principle Component Analysis

1. Download MNIST dataset from this link and CIFAR-10 dataset from this link.

2. In this question, you will implement Principal Component Analysis (PCA) from scratch and apply it to reconstruct images from the MNIST and CIFAR-10 dataset. You will use only `NumPy` for the core implementation (**NO** `sklearn.decomposition.PCA` is allowed).

2.1 PCA Implementation

Implement a complete PCA pipeline using only `NumPy`, focusing on image reconstruction. Implement the following :–

1. `standardize_data(X)` to center the data matrix by subtracting the mean.

2. `compute_pca_eigen(X, n_components)` to perform eigen decomposition of the covariance matrix, returning sorted principal components and explained variance ratios.

3. `reconstruct_images(X, components, mean)` to reconstruct images using the computed components

NOTE :– The entire implementation must follow proper train/test split protocols, learning PCA components solely from training data and applying the stored transformation to test data to avoid leakage.

2.2 Analysis and Visualization

Create a comprehensive analysis suite that visualizes reconstruction results and evaluates performance.

1. The visualization should display 10 example images (one from each class) from both MNIST and CIFAR-10 datasets, showing original and reconstructed versions side by side with proper labels.
2. Include quantitative analysis using MSE and PSNR metrics to evaluate reconstruction quality
   (a) Examine the trade-off between compression ratio and image quality for varying numbers of components.
   (b) Document the comparative performance between gray-scale (MNIST) and color (CIFAR-10) reconstructions.

3 Stationary distribution of a Markov Chain

In the `transition_matrices` folder the data file has name of the form `P_X.txt`.

For a given positive state transition matrix $P \in \mathbb{R}^{n \times n}$, compute its stationary distribution π using:

1. Solve directly using the system of linear equations: $\begin{cases} \pi^\top P = \pi^\top \\ \pi^\top \mathbf{1} = 1 \end{cases}$.

   Take this to be the reference solution.
2. Solve using the power method as discussed in class (and in lecture notes).
3. For the matrix in the provided dataset:
   (a) Calculate π using both methods.
   (b) Verify the results:
      - $\|\pi^\top P - \pi^\top\|_1 = 0$
      - $\sum_{i=1}^{n} \pi_i = 1$

   (c) Compare the number of iterations for convergence.
   (d) Record and compare execution times for both methods.
4. Analyze and compare:
   - Accuracy of approaches
   - Computational efficiency
   - Impact of matrix size

4 Spectral Norm of a Matrix

Even after larger number of iterations the power iteration method for calculating the spectral norm of a matrix might not converge for a given tolerance. To overcome this problem a new method using gram iteration has been introduced. You are required to perform a comparative study b/w the two methods as discussed below.

Algorithm 1 Power_iteration(G, N_{iter})

```
1: **Inputs** matrix: $G$, number of iterations: $N_{\text{iter}}$
2: **Initialization**: draw a random vector $u$
3: **for** $1 \dots N_{\text{iter}}$ **do**
4:     $v \leftarrow Gu/\|Gu\|_2$
5:     $u \leftarrow G^\top v/\|G^\top v\|_2$
6: **end for**
7: $\sigma_1 \leftarrow (Gu)^\top v$
8: **return** $\sigma_1$
```

Algorithm 2 Gram_iteration(G, N_{iter})

```
1:  **Inputs** matrix: $G$, number of iterations: $N_{\text{iter}}$
2:  $r \leftarrow 0$                                   // initialize rescaling
3:  $m, n \leftarrow G.shape$
4:  **if** $n > m$ **then**
5:      $G \leftarrow G^\top$
6:  **end if**
7:  **for** $1 \dots N_{\text{iter}}$ **do**
8:      $r \leftarrow 2(r + \log \|G\|_F)$              // cumulate rescaling
9:      $G \leftarrow G/\|G\|_F$                        // rescale to avoid overflow
10:     $G \leftarrow G^\top G$                         // Gram iteration
11: **end for**
12: $\sigma_1 \leftarrow \|G\|_F^{2^{-N_{\text{iter}}}} \exp(2^{-N_{\text{iter}}} r)$
13: **return** $\sigma_1$
```

1. Initialize a 400×500 matrix, say G from the data file `matrix_X.txt` in the `spectral_norm_matrices` folder.
2. Perform power iterations on G for at least 2000 iterations to get its spectral norm as defined in Algo 1.
3. Perform gram iterations on G for 15 iterations to get its spectral norm as defined in Algo 2.
4. Repeat steps 2 and 3 100 times to generate plots mentioned below.
5. Get the avearge run times of the 2 algos. Note the differences b/w them.
6. Use `numpy`'s `svd` method to find spectral norm of G and use it to generate the following figures as seen and described below.
7. Convergence plot in log-log scale for spectral norm computation, comparing Power iteration and Gram iteration, one standard deviation shell $(Mean - StdDev, Mean + StdDev)$ is represented in a light color. Difference at each iteration is defined as $|\sigma_{1method} - \sigma_{1ref}|$.
8. Convergence plot in log-log scale for spectral norm computation, comparing power iteration and Gram iteration, each line corresponds to one run of Power iteration and Gram iteration. Difference at each iteration is defined as $|\sigma_{1method} - \sigma_{1ref}|$.
9. Explain the differences you observed from the figures b/w Algo 1 and 2 in small pointers (Please don't write stories). Mention anything that you can infer.
10. (a) Write an algorithm that uses Algo 1 on a symmetric matrix to get the minimum eigenvalue and its sign.
    (b) What changes are needed if the maximum eigenvalue is to be found ?

    Hint: EVD always exists for a symmetric matrix; Algo 1 can be used at most twice and has to be used at least once.