CSCI4390-6390 Assign1

Assign1: Covariance and Eigenvectors

Due Date: Sep 11th, before midnight (11:59:59PM, Alofi Time; GMT-11)

Download the Appliances energy prediction data set from the UCI Machine Learning repository. This dataset has 29 attributes and 19725
will ignore the first attribute, which is a date-time variable. The rest of the attributes are all numeric, and will be used in the assignment.

This assignment consists of two parts. Part I must be done by students in both sections, namely CSCI4390 and CSCI6390. For the second
II-4390 must be done by those in CSCI4390, and Part II-CSCI6390 must be done by students registered for CSCI6390.

Part I (both CSCI-4390 and CSCI-6390): 50 Points

a. Mean vector and total variance

Compute the mean vector μ for the data matrix, and then compute the total variance $var(\mathbf{D})$; see Eq. (1.8) for the latter.

b. Covariance matrix (inner and outer product form)

Compute the sample covariance matrix $\mathbf{\Sigma}$ as **inner products** between the attributes of the centered data matrix (see Eq. (2.38) in chapter
compute the sample covariance matrix as sum of the **outer products** between the centered points (see Eq. (2.39)).

c. Correlation matrix as pair-wise cosines

Compute the correlation matrix for this dataset using the formula for the cosine between centered attribute vectors (see Eq. (2.30)).

Output which attribute pairs are i) the most correlated, ii) the most anti-correlated, and iii) the least correlated?

Create the scatter plots for the threee interesting pairs using matplotlib and visually confirm the trends, i.e., describe how each of the three
results in a particular type of plot.

Part II: Eigenvectors (50 Points)

CSCI-4390 Only: Dominant Eigenvector

Compute the dominant eigenvalue and eigenvector of the covariance matrix $\mathbf{\Sigma}$ via the power-iteration method. One can compute the domi
vector/-value of the covariance matrix iteratively as follows.

Let

$$\mathbf{x}_0 = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

be the starting vector in R^d, where d is the number of dimensions.

In each iteration i, we compute the new vector:

$$\mathbf{x}_i = \mathbf{\Sigma}\ \mathbf{x}_{i-1}$$

We then find the element of $\mathbf{x}_i$ that has the maximum absolute value, say at index m. For the next round, to avoid numerical issues with la
we re-scale $\mathbf{x}_i$ by dividing all elements by x_{im}, so that the largest value is always 1 before we begin the next iteration.

To test convergence, you may compute the norm of the difference between the scaled vectors from the current iteration and the previous o
can stop if this norm falls below some threshold. That is, stop if

$$\|\mathbf{x}_i - \mathbf{x}_{i-1}\|_2 < \epsilon$$

For the final eigen-vector, make sure to normalize it, so that it has unit length.

Also, the ratio $\frac{x_{im}}{x_{i-1,m}}$ gives you the largest eigenvalue. If you did the scaling as described above, then the denominator will be 1, but the nu
will be the updated value of that element before scaling.

Once you have obtained the dominant eigenvector, $\mathbf{u}_1$, project each of the original data points $\mathbf{x}_i$ onto this vector, and print the coordinate
new points along this "direction".

CSCI-6390 Only: First Two Eigenvectors and Eigenvalues

Compute the first two eigenvectors of the covariance matrix $\mathbf{\Sigma}$ using a generalization of the above iterative method.

Let $\mathbf{X}_0$ be a $d \times 2$ (random) matrix with two non-zero d-dimensional column vectors with unit length. We will iteratively multiply $\mathbf{X}_0$ w
the left.

The first column will not be modified, but the second column will be orthogonalized with respect to the first one by subtracting its projecti
the first column (see section 1.3.3 in chapter 1). That is, let $\mathbf{a}$ and $\mathbf{b}$ denote the first and second column of $\mathbf{X}_1$, where

$$\mathbf{X}_1 = \mathbf{\Sigma}\,\mathbf{X}_0$$

Then we orthogonalize $\mathbf{b}$ as follows:

$$\mathbf{b} = \mathbf{b} - \left(\frac{\mathbf{b}^T\mathbf{a}}{\mathbf{a}^T\mathbf{a}}\right)\mathbf{a}$$

After this $\mathbf{b}$ is guaranteed to be orthogonal to $\mathbf{a}$. This will yield the matrix $\mathbf{X}_1$ with the two column vectors denoting the current estimates
first and second eigenvectors.

Before the next iteration, normalize each column to be unit length, and repeat the whole process. That is, from $\mathbf{X}_1$ obtain $\mathbf{X}_2$ and so on, u
convergence.

To test for convergence, you can look at the distance between $\mathbf{X}_i$ and $\mathbf{X}_{i-1}$. If the difference is less than some threshold ϵ then we stop.

Once you have obtained the two eigenvectors: $\mathbf{u}_1$ and $\mathbf{u}_2$, project each of the original data points $\mathbf{x}_i$ onto those two vectors, to obtain the
projected points in 2D. Plot these projected points in the two new dimensions.

Submission

Submit your code via submitty. Name your python script: **assign1.py**.

Your script will be run as follows:

assign1.py FILENAME EPS

Here FILENAME is the name of the input CSV file, EPS the convergence threshold ϵ for the eigen-vector/-value computation.

You may assume that the input CSV data file *energydata_complete.csv* resides in the local directory where the script will be called from. Y
epsilon as 0.001 or 0.0001. You may find the pandas read_csv function useful to read the CSV file.

Save all your output to a pdf file named **assign1.pdf**. The output should comprise the mean vector, total variance, covariance matrix via in
outer product formulas, correlation matrix, the observations, the dominant eigen-vectors and eigenvalues. The scatter plots should also be
output file as well, with any required comments. **You will lose points if you do not include the output PDF file.**

Note that since there are >19k points, you should not print out the full eigenvectors. Just print out the first 10 and last ten values per eigen

Your script must use Python version 3. Please note that you can use built-in NumPy/Python functions for reading and parsing the text inpu
should NOT use any of the built-in functions like **cov** or **eigen** for this assignment. You may however verify your answers by comparing to
from the built-in methods.

Tutorial on Python and NumPy

For those not that familiar with pythoni or NumPy, you may search online for tutorials, e.g. https://docs.python.org/3/tutorial/ or
https://numpy.org/doc/stable/

Policy on Academic Honesty

You are free to discuss how to tackle the assignment, but all coding must be your own. Please do not copy or modify code from anyone els
including code on the web. Any students caught violating the academic honesty principle will get an automatic F grade on the course and
referred to the dean of students for disciplinary action.