Due date: March 25, 2025

Problem 1: High Dimensional Cubes? Spheres! As you may have known, high-dimensional geometry is *very* weird. Many phenomena that we take for granted break down as we move into higher dimensional spaces.

(i) Consider the difference between the d-dimensional unit cube $C^d = [-1/2, 1/2]^d$ and the d-dimensional unit ball $B^d = \{x \in \mathbb{R}^d : \|x\|_2 \leqslant 1\}$. Show by example that for small d, C^d is entirely contained in B^d, but as d increases, argue that the vertices of C^d lie far outside the unit ball B^d. Part (ii) shows things are much crazier than just this!

(ii) The *Weak Law of Large Numbers* says if you have independent and identically distributed (i.i.d.) random variables $X_1, X_2, \ldots$, then as $n \to \infty$, the mean $Y_n = \sum_{i=1}^n X_i/n$ converges to $\mathbb{E}X_1$, and the larger n is, the more likely Y_n will be *close* (this is formally called *convergence in probability*) to $\mathbb{E}X_1$. For example if you flip a fair coin 10 times, you may find it unsurprising to get $0.4 \cdot 10 = 4$ heads. However, if you flip the coin 1000 times and observe only $0.4 \cdot 1000 = 400$ heads, you may start to think that the coin is biased, even though the fraction of heads remains 0.4.

Consider randomly sampling a point $X = (X_1, \ldots, X_n)$ in C^d by sampling each X_i uniformly from $[-1/2, 1/2]$. Then $\mathbb{E}X_i^2 = \int_{-0.5}^{0.5} x^2 \, \mathrm{d}x = 1/12$. What do you suspect

$$S_n = \sqrt{X_1^2 + \ldots + X_n^2}$$

to converge to, as $n \to \infty$? How large is this value compared to the radius of B^d? Argue (informally) that in high-dimensions, (i) the unit cube is essentially *almost* just a sphere (state its radius) and (ii) *almost all* volume of C^d lies outside of B^d.

Problem 2: PCA vs. Least Squares

As discussed in lecture, both PCA and least squares (LS) regression are ways to infer linear relationships between data. In this problem you will develop some intuition for the differences between these two approaches, and an understanding of when to use which one.

In this problem we'll use a simple synthetic example with two variables x and y, where the hidden relationship is $y = 3x$.

A quick linear algebraic recap: given inputs (x_i, y_i), the LS fit aims to find a line $\ell(x)$ such that least-squares error $\sum_{i=1}^n (\ell(x_i) - y_i)^2$ is minimized. If μ_x, μ_y are the means of x and y, respectively, the slope can be precisely computed:

$$\ell^* = \frac{\langle x - \mu_x \cdot \mathbf{1}, y - \mu_y \cdot \mathbf{1} \rangle}{\|x - \mu_x\|_2^2}$$

where $\mu_x \cdot \mathbf{1}$ is the n-dimensional vector where each component equals μ_x and likewise for $\mu_y \cdot \mathbf{1}$. You will need two core functions throughout this assignment:

- `pca_recover`, which takes $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ as inputs, and returns the slope of the first component of the PCA — in our case, this would be the second coordinate divided by the first. **In this problem, you ARE allowed to use any existing functions or libraries as you wish.**
- `ls_recover`, which also takes x, y as inputs, and returns the slope ℓ^* of the least-squares fit.

As a sanity check, you can set $x = (1/1000, 2/1000, \ldots, 999/1000, 1)$ and $y = 3x$. If implemented correctly, both functions should return 3. Once you have implemented these two functions, please answer the following:

(i) Noise-less x and noisy y. In this part, let $x = (1/1000, 2/1000, \ldots, 1)$. For a $\sigma > 0$, each component y_i of y is set to be $3x_i$ plus some random noise:

$$\hat{y}_i \sim 3x_i + \mathcal{N}(0, \sigma^2) = \frac{3i}{1000} + \mathcal{N}(0, \sigma^2).$$

Make a scatterplot with σ on the x-axis, and the outputs of `pca_recover` and `ls_recover` on the y-axis (you can plot the outputs of both functions separately or on the same plot). For each $\sigma \in \{0, 0.05, 0.1, \ldots, 0.45, 0.5\}$, repeat the above 40 times. There are 11 values for σ, so your scatterplot should contain 440 points for each function.

(ii) Noisy x and noisy y. Now, for a given $\epsilon > 0$ and $1 \leqslant i \leqslant 1000$, let

$$\hat{x}_i \sim x_i + \mathcal{N}(0, \sigma^2) = \frac{i}{1000} + \mathcal{N}(0, \sigma^2)$$

and likewise

$$\hat{y}_i \sim 3x_i + \mathcal{N}(0, \sigma^2) = \frac{3i}{1000} + \mathcal{N}(0, \sigma^2).$$

Like in the previous part, for each $\sigma \in \{0, 0.05, 0.1, \ldots, 0, 45, 0.5\}$, repeat this experiment 40 times. Display the outputs of both functions as scatterplots.

(iii) Why does PCA do poorly in (i), where only y is noisy? Why does PCA do well, and why does LS do poorly when both x and y are noisy, as in (ii)? This part is somewhat open-ended — feel free to share your thoughts.

Problem 3: PCA on MNIST Digits. In this implementation problem you will be playing around with PCA on the MNIST dataset, which consists of 784-dimensional inputs corresponding to pixels of 28×28 image showing handwritten digits from 0 to 9, where each pixel ranges from 0 to 255 representing its grayness scale. In Python, the dataset can be obtained via:

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
X = mnist.data.astype(np.float32)
y = mnist.target.astype(int)
```

If you are using other programming languages, you may find the dataset on openML or here (file too large to be uploaded to Canvas, so I used Google Drive). Given n samples and $d = 784$, the input X has dimension $\mathbb{R}^{n\times d}$. Your code should contain the core features (not all of them need to be implemented in standalone functions), along with any other help functions you wish to add (e.g. functions to visualize 784-dimensional vectors as 32×32 images). **In this problem, you may NOT directly use publicly available PCA functions such as Python's `sklearn.decomposition.PCA`** (though you are welcome to use them to verify your answers).

- **Normalization**. Compute sample mean μ and variance σ^2 and normalize X via $X \leftarrow (X - \mu)/\sigma$ so the new dataset has mean 0 and variance 1.
- **Singular Value Decomposition (SVD)**. Perform SVD on $X^T X \in \mathbb{R}^{d\times d}$, yielding singular values $\sigma_1, \ldots, \sigma_d$ and (right) singular vectors $v_1, \ldots, v_d$ which correspond to the principal components of X.
- **Reconstruction**. Given the number p, output the linear projection of X into a p-dimensional space with minimized mean square error between X and the transformed version. (This is achieved by projecting X onto the subspace spanned by the first p principal components.)

Your write-up should contain the following:

(i) With abuse of notation, let X be the normalized input. Perform SVD on X and obtain singular values $\sigma_1, \ldots, \sigma_d$ with associated singular vectors $v_1, \ldots, v_d$. Recall the cumulative % of explained variance of the first p principal components is $\sum_{i=1}^{p} \sigma_i^2 / \sum_{i=1}^{d} \sigma_i^2$.

Plot the cumulative sum of these percentages vs. the number of components (or do it below). Visualize the k^{th} principal component for each $k \in \{1, 2, 5, 20, 100, 784\}$. Describe your findings as k increases. Specifically, describe the case $k = 1$ and $k = 784$. Why do you think they look like this?

(ii) Apply PCA to the input with $p \in \{1, 2, 5, 20, 100, 784\}$. Pick one image from each class (i.e. each image digit from 0 to 9). Display this image, along with its reduced-dimension versions for each p. Also compute each image's normalized reconstruction error with respect to Frobenius norm: if M is the input matrix and $\hat{M}_p$ is the recovered image by the top p principal components, then the normalized error is

$$\frac{\|M - \hat{M}_p\|_F}{\|M\|_F} \qquad \text{where} \qquad \|M\|_F = \left(\sum_{i,j} |M_{i,j}|^2\right)^{1/2}.$$

Which classes (digits) are "easy" to be approximated? Which ones are "hard?" Does it make sense visually? This part is also somewhat open-ended, so feel free to share your thoughts.