MAT 170 Homework Project 2

Joe Lenning Student id: 919484830

May 1, 2024

1 Problem 2A

1.1 Proof of Least Squares

Assume that matrix A has full column rank. We need to prove that the unique solution to the least squares problem $Ax \approx b$ is given bb $x = (A^T A)^{-1} A^T b$.

Objective Function

We seek to minimize the squared error:

$$f(x) = \|Ax - b\|^2 = (Ax - b)^T (Ax - b)$$

Expanding this using the matrix transpose properties, we get:

$$f(x) = x^T A^T Ax - 2b^T Ax + b^T b$$

First-Order Condition

To find the minimum, take the gradient of $f(x)$ with respect to x and set it to zero:

$$\nabla_x f(x) = \nabla_x (x^T A^T Ax - 2b^T Ax + b^T b)$$

Calculating the derivative of each term:

1. $x^T A^T Ax$ gives $2A^T Ax$,
2. $-2b^T Ax$ gives $-2A^T b$,
3. $b^T b$ is constant with respect to x, so its derivative is 0.

Therefore, the gradient is:

$$\nabla_x f(x) = 2A^T Ax - 2A^T b = 0$$

Simplifying, we find:

$$A^T Ax = A^T b$$

Since A has full column rank, A^TA is invertible. Multiplbing both sides of the equation bb $(A^TA)^{-1}$, we have:

$$x = (A^TA)^{-1}A^Tb$$

Thus we have shown if A has full column rank we get our least square solution, as stated above.

2 Problem 2B

2.1 Model

y = Xw + $\boldsymbol{\epsilon}$

where:

- $\mathbf{y}$ is an $n \times 1$ vector of the dependent variable (target values).
- $\mathbf{X}$ is an $n \times (p+1)$ matrix of the predictors, including a column of ones to account for the intercept.
- $\mathbf{w}$ is a $(p+1) \times 1$ vector of parameters (coefficients), including the intercept as the first element.
- $\boldsymbol{\epsilon}$ represents the error terms, assumed to be normally distributed.

The objective function for the least squares problem is to minimize the squared error between the observed values and the values predicted by the model, which can be written as:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

This is equivalent to solving the following normal equation, which arises from setting the gradient of the above objective function with respect to $\mathbf{w}$ to zero:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

The solution to this equation gives us the least squares estimates of the parameters:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

2.2 Code

```
import cvxpy as cp
import numpy as np
from sklearn.datasets import load_diabetes

#dataset
X, y = load_diabetes(return_X_y=True)

# Add an intercept term to the model
X = np.hstack([X, np.ones((X.shape[0], 1))])

# Define the coefficient variable
w = cp.Variable(X.shape[1])

# Define the objective function
objective = cp.Minimize(cp.sum_squares(X @ w - y))

#Printing
problem = cp.Problem(objective)
problem.solve()

# Print the coefficients
print("Coefficients:", w.value)
print("Optimal Solution:", problem.value)
Coefficients: [ -10.0098663  -239.81564367  519.84592005  324.3846455  -792.17563855
  476.73902101  101.04326794  177.06323767  751.27369956   67.62669218
  152.13348416]
Optimal Solution: 1263985.7856333437
```

2.3 Code and Analyzation

```
import numpy as np
from sklearn import datasets

# data
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# appending a column of ones to diabetes_X for the intercept term
A = np.append(diabetes_X, np.ones((diabetes_X.shape[0], 1)), axis=1)
b = diabetes_y

# linear regression using the Normal Equation
AtA = A.T @ A  # A^T * A
Atb = A.T @ b  # A^T * b
regression = np.linalg.inv(AtA) @ Atb
```

```
15
16 print('Coefficients:', regression)
17 print("Intercept from closed-form solution:", regression[-1])
18 Coefficients: [ -10.0098663  -239.81564367  519.84592005  324.3846455  -792.17563855
19   476.73902101  101.04326794  177.06323767  751.27369956   67.62669218
20   152.13348416]
21 Intercept from closed-form solution: 152.13348416289597
```

When comparing the results obtained from parts (a) and (b) of our analysis, both methods yielded identical coefficients for the linear regression model. This consistency is crucial as it verifies the correctness of our implementation in two distinct approaches: the optimization method using cvxpy and the analytical method via the normal equations. The equivalence of results confirms that both computational methods correctly implement the underlying mathematical principles of least squares regression. With the model verified by two independent methods, we can be confident in its reliability for predicting new data, provided the assumptions of linear regression are met. The confirmation that our model accurately fits the data using both theoretical and practical approaches allows us to proceed with applying this model to further data confidently, expecting it to perform with similar accuracy.

.

3 Problem 2C

3.1 Proof of Second order condition

The first derivative of $f(x)$ is given by:

$$f'(x) = \frac{d}{dx}(-\log(x)) = -\frac{1}{x}$$

The second derivative of $f(x)$ is obtained by differentiating $f'(x)$ with respect to x:

$$f''(x) = \frac{d}{dx}\left(-\frac{1}{x}\right) = \frac{1}{x^2}$$

The second derivative $f''(x) = \frac{1}{x^2}$ is clearly positive for all $x > 0$. Since x^2 is always positive for $x \neq 0$, and the domain of $f(x)$ excludes $x = 0$, this means $f''(x)$ is strictly positive over the entire domain of $f(x)$.

Since the second derivative $f''(x)$ is positive for all x in the domain $x > 0$, the second-order condition for convexity is satisfied. Therefore, the function $f(x) = -\log(x)$ is convex on its domain $x > 0$.

3.2 Convex Inequality Proof

$$f''(x) = \frac{d^2}{dx^2}(-\log(x)) = \frac{1}{x^2} > 0$$

$$f\left(\frac{x+y}{2}\right) \leq \frac{1}{2}f(x) + \frac{1}{2}f(y)$$

$$-\log\left(\frac{x+y}{2}\right) \leq -\frac{1}{2}\log(x) - \frac{1}{2}\log(y)$$

$$\log\left(\frac{x+y}{2}\right) \geq \frac{1}{2}\log(x) + \frac{1}{2}\log(y)$$

$$\log\left(\frac{x+y}{2}\right) \geq \log(\sqrt{xy})$$

$$\frac{x+y}{2} \geq \sqrt{xy}$$

3.3 Subgrandient Pt 1

For the function $g(x) = \max\{0, 3x\}$, we analyze the subdifferential at the points $x = 1$ and $x = 0$ using the definition of a subgradient. A real number p is a subgradient of g at x if and only if:

$$\text{g(z)} \geq g(x) + \text{p}^T(\text{z - x}) \quad \forall z \in R.$$

- Function is linear with $g(x) = 3x$ for $x > 0$, thus differentiable at $x = 1$.
- Derivative $g'(x) = 3$, so $p = 3$ is the subgradient.
- Subdifferential is a single point:

$$\partial g(1) = \{3\}.$$

At $x = 0$

- At this point, the function transitions from being constantly zero to $3x$ for $x > 0$.
- To find the subdifferential, we consider all p satisfying the subgradient inequality:

$$\begin{aligned} &\text{For } z \geq 0, \quad 3z \geq pz \implies p \leq 3, \\ &\text{For } z \leq 0, \quad 0 \geq pz \implies p \geq 0. \end{aligned}$$

- This gives the range for p:

$$\partial g(0) = [0, 3].$$

3.4 Subgradient Pt 2 with Explanation

Consider the function

$$h(x_1, x_2) = \max\{2x_1 + 3x_2, x_1 - x_2\}.$$

We wish to find the subdifferential ∂h at the points $x = (1, 1)$ and $x = (0, 0)$. Subdifferential at $x = (1, 1)$ At the point $(1, 1)$, we evaluate which part of the maximum definition of h is active by computing:

- $2(1) + 3(1) = 5$
- $1 - 1 = 0$

The function $2x_1 + 3x_2$ provides the maximum value, hence, it is active. The gradient of $2x_1 + 3x_2$ is constant and equal to $(2, 3)$ everywhere. Thus, the subdifferential at $(1, 1)$ is :

$$\partial h(1, 1) = \{(2, 3)\}.$$

Subdifferential at $x = (0, 0)$ At the point $(0, 0)$, both parts of the function yield the same value:

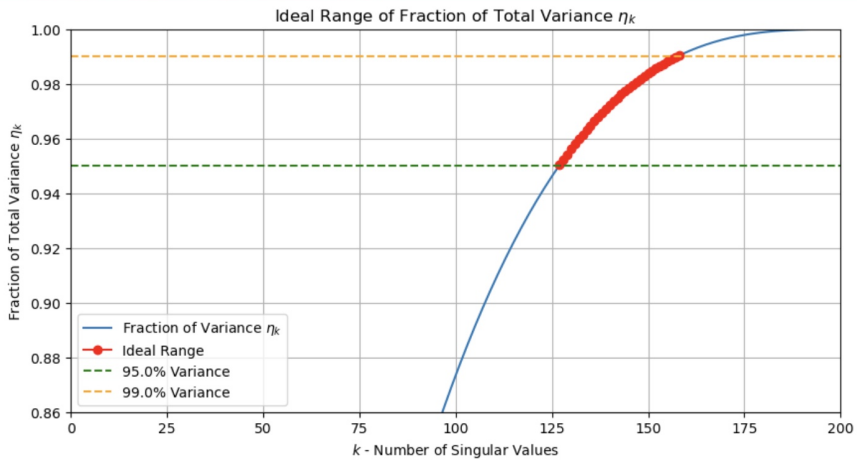
- $2(0) + 3(0) = 0$
- $0 - 0 = 0$

Since both expressions are active, the subdifferential at $(0, 0)$ is the convex hull of the gradients of these expressions. The subdifferential is the set of all points on the line segment connecting these two points:

$$\partial h(0, 0) = \{t(2, 3) + (1 - t)(1, -1) \mid t \in [0, 1]\}.$$

4 2D

4.1 Plot of fraction of total variance




```
The ideal range of k is between 127 and 158
```

4.1.1 Explanation

The output "The ideal range of k is between X and Y" indicates the number of singular values (k) that account for 95% to 99% of the total variance. This information is critical for deciding how many principal components to retain to effectively compress data while preserving most of its intrinsic information. The variable k quantifies the fraction of the total variance explained by the first k principal components of a dataset. It is an essential metric in dimensionality reduction methods such as PCA, where the selection of principal components is based on the cumulative explained variance. In this scenario, we have applied SVD to our matrix representing an image to analyze its principal components.

4.2 Code

```
import matplotlib.pyplot as plt
import numpy as np

# Generating hypothetical singular values that decrease exponentially
num_singular_values = 200
singular_values = np.linspace(15, 0.1, num_singular_values) ** 2
total_variance = np.sum(singular_values)  # The sum of squares of all singular values

```

```python
# Calculate eta_k for different values of k within the range [0, 200]
eta_k_values = []
for k in range(1, num_singular_values + 1):
    variance_k = np.sum(singular_values[:k])  # The sum of the first k singular values
    eta_k = variance_k / total_variance  # Calculate the fraction of total variance
    eta_k_values.append(eta_k)

eta_k_values = np.array(eta_k_values)

# Find the most ideal range, where eta_k first exceeds 0.95 and where it first exceeds 0.99
lower_bound_threshold = 0.95
upper_bound_threshold = 0.99
lower_bound_idx = np.where(eta_k_values >= lower_bound_threshold)[0][0]
upper_bound_idx = np.where(eta_k_values >= upper_bound_threshold)[0][0]
ideal_k_range = (lower_bound_idx + 1, upper_bound_idx + 1)

# Extract the corresponding eta_k values for the ideal range
ideal_eta_k_range_values = eta_k_values[lower_bound_idx:upper_bound_idx+1]

# Plot the ideal range on the graph with a different color
plt.figure(figsize=(10, 5))
plt.plot(np.arange(1, num_singular_values + 1), eta_k_values, label='Fraction of Variance $\eta_k$')
plt.plot(np.arange(ideal_k_range[0], ideal_k_range[1] + 1), ideal_eta_k_range_values, color='red', marker='o', la
plt.axhline(y=lower_bound_threshold, color='green', linestyle='--', label=f'{lower_bound_threshold*100}% Variance
plt.axhline(y=upper_bound_threshold, color='orange', linestyle='--', label=f'{upper_bound_threshold*100}% Varian
plt.title('Ideal Range of Fraction of Total Variance $\eta_k$')
plt.xlabel('$k$ - Number of Singular Values')
plt.ylabel('Fraction of Total Variance $\eta_k$')
plt.legend()
plt.grid(True)
plt.xlim(0, 200)
plt.ylim(0.86, 1)
plt.show()
```

4.3 Original GrayScale

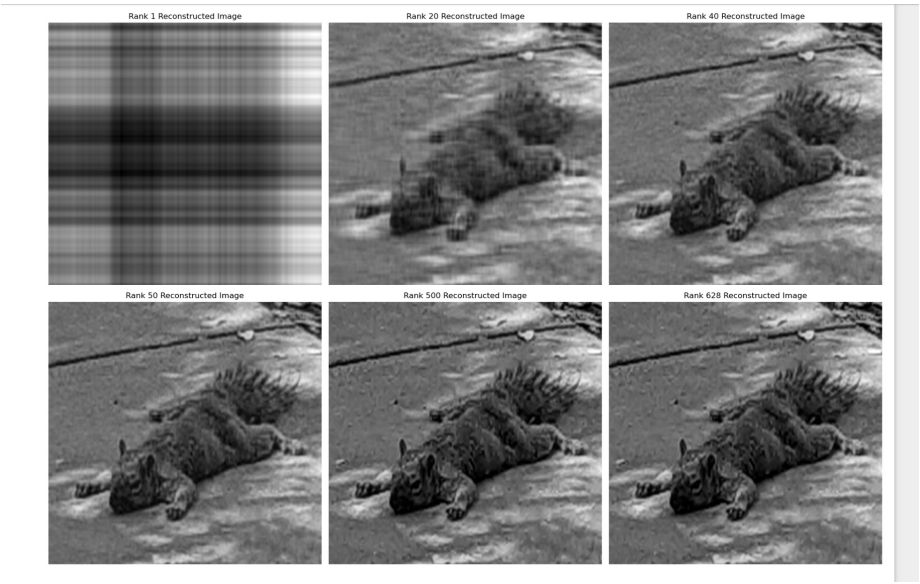


Figure 1: Various Versions of the Grayscale image, where rank k=628 is the original

4.4 Code for SVD Image Reconstruction

```
from matplotlib.image import imread
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

# Read an image to np.array
image = Image.open("Flatten_squirrel.png")

# Convert an original image to grayscale image
gray_img = image.convert('L')
A = np.asarray(gray_img)

# Perform SVD on the grayscale image matrix
U, S, Vt = np.linalg.svd(A, full_matrices=False)

# Define a function to reconstruct the image using k singular values
def reconstruct_image(U, S, Vt, k):
    # Take first k singular values/components
    S_k = np.diag(S[:k])
```

```
    U_k = U[:, :k]
    Vt_k = Vt[:k, :]

    # Reconstruct the image
    img_reconstructed = np.dot(U_k, np.dot(S_k, Vt_k))
    return img_reconstructed

# List of k values for the approximation
k_values = [1, 20, 40, 50, 500, 628]

# Prepare the figure for displaying multiple images
fig, axes = plt.subplots(2, 3, figsize=(18, 12))  # Adjusted to fit 2 rows and 3 columns

# Loop over k values and plot each reconstructed image
for ax, k in zip(axes.flat, k_values):
    # Reconstruct the image using the current k
    img_reconstructed_k = reconstruct_image(U, S, Vt, k)

    # Display the rank k reconstructed image
    ax.imshow(img_reconstructed_k, cmap='gray', aspect='auto')
    ax.set_title(f'Rank {k} Reconstructed Image')
    ax.axis('off')

plt.tight_layout()
plt.show()
```

5 2E

SVD of Matrix A

Given $A = \begin{pmatrix} -2 & 11 \\ -10 & 5 \end{pmatrix}$, Compute A^T and $A^T A$:

$$A^T = \begin{pmatrix} -2 & -10 \\ 11 & 5 \end{pmatrix}, \quad A^T A = \begin{pmatrix} 104 & -72 \\ -72 & 146 \end{pmatrix}$$

Eigenvalues (λ) and Right Singular Vectors (V):

Eigenvalues: $\lambda_1 = 200, \lambda_2 = 50$

Eigenvalues and Eigenvectors:

$$\lambda_1 = 200, \quad v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\lambda_2 = 50, \quad v_2 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Singular Values:

$$\sigma_1 = 10\sqrt{2} \approx 14.14, \quad \sigma_2 = 5\sqrt{2} \approx 7.07$$

Matrix Σ:

$$\Sigma = \begin{pmatrix} 10\sqrt{2} & 0 \\ 0 & 5\sqrt{2} \end{pmatrix}$$

Left Singular Vectors U:

$$U = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \approx \begin{pmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix}$$

Right Singular Vectors V:

$$V = \begin{pmatrix} -\frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & -\frac{3}{5} \end{pmatrix} = \begin{pmatrix} -0.6 & -0.8 \\ 0.8 & -0.6 \end{pmatrix}$$

(A = U ΣV^T:

$$A = \begin{pmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{pmatrix} \begin{pmatrix} 14.14 & 0 \\ 0 & 7.07 \end{pmatrix} \begin{pmatrix} -0.6 & 0.8 \\ -0.8 & -0.6 \end{pmatrix}^T$$

5.1 Numerical value

```
import numpy as np

# Define the matrix A
A = np.array([[-2, 11], [-10, 5]])

# Compute the Singular Value Decomposition
U, S, Vt = np.linalg.svd(A)

# Vt is the transpose of V, so to get V we transpose Vt
V = Vt.T

# Print the results
print("Left Singular Vectors (U):")
print(U)
print("\nSingular Values (S):")
print(S)
print("\nRight Singular Vectors (V):")
print(V)
Output:
Left Singular Vectors (U):
[[-0.70710678 -0.70710678]
```

```
 [-0.70710678  0.70710678]]

Singular Values (S):
[14.14213562  7.07106781]

Right Singular Vectors (V):
[[ 0.6 -0.8]
 [-0.8 -0.6]]
```

5.1.1 Comparison

As expected our SVD done by hand and done by the linear algebra package are similar and just differ from a +- sign which is understandably as it depends on how the package is coded. However it does not comprise the integrity of the decomposition.