580 Homework 5

Yiming Mao

March 27, 2025

The Power Method

(a)

For any $v \in \mathbb{R}^d$, since $\{u_1, \ldots, u_d\}$ is an orthonormal basis, we can write

$$v \;=\; \sum_{i=1}^{d} \alpha[i]\, u_i,$$

where

$$\alpha[i] \;=\; u_i^\top v.$$

This representation is unique by the orthonormality of the u_i.

(b)

The coordinates $\alpha[i]$ in the expansion

$$v \;=\; \sum_{i=1}^{d} \alpha[i]\, u_i$$

are given by $\alpha[i] = u_i^\top v$. Hence the choice of α is unique and always exists.

(c)

The Power Method iterates

$$x_{t+1} \;=\; \frac{M\, x_t}{\|M\, x_t\|_2}, \quad \text{starting from } x_0 \text{ with } \|x_0\|_2 = 1.$$

Write

$$x_t \;=\; \sum_{i=1}^{d} \alpha_t[i]\, u_i.$$

Then

$$\hat{x}_{t+1} \;=\; M\, x_t \;=\; \sum_{i=1}^{d} \lambda_i\, \alpha_t[i]\, u_i.$$

Thus

$$x_{t+1} \;=\; \frac{\hat{x}_{t+1}}{\|\hat{x}_{t+1}\|_2} \;=\; \sum_{i=1}^{d} \frac{\lambda_i\, \alpha_t[i]}{\sqrt{\sum_{j=1}^{d} \lambda_j^2\, \alpha_t[j]^2}}\, u_i.$$

Hence, if we denote by $\alpha_{t+1}[i]$ the new coefficient of u_i, we have

$$\alpha_{t+1}[i] \;=\; \frac{\lambda_i\, \alpha_t[i]}{C},$$

where

$$C \;=\; \sqrt{\sum_{j=1}^{d} \lambda_j^2\, \alpha_t[j]^2}.$$

Note that C does not depend on i.

(d)

Assume $\alpha_0[i] > 0$ for all i. By iterating the relation

$$\alpha_{t+1}[i] \;=\; \frac{\lambda_i\,\alpha_t[i]}{C_t},$$

one obtains

$$\alpha_t[i] \;=\; \frac{\lambda_i^t\,\alpha_0[i]}{\sqrt{\sum_{j=1}^d \lambda_j^{2t}\,\alpha_0[j]^2}} \;=\; \frac{\lambda_i^t\,\alpha_0[i]}{C_t},$$

where

$$C_t \;=\; \sqrt{\sum_{j=1}^d \lambda_j^{2t}\,\alpha_0[j]^2}.$$

(e)

We want to show

$$\sum_{i=2}^d \alpha_t[i]^2 \;\le\; \left(\frac{\lambda_2}{\lambda_1}\right)^{2t} \;\frac{\sum_{i=2}^d \alpha_0[i]^2}{\alpha_0[1]^2}.$$

Using the closed-form expression,

$$\sum_{i=2}^d \alpha_t[i]^2 \;=\; \frac{\sum_{i=2}^d \lambda_i^{2t}\,\alpha_0[i]^2}{\sum_{j=1}^d \lambda_j^{2t}\,\alpha_0[j]^2}.$$

Since $\lambda_2 \ge \lambda_i$ for $i \ge 2$ and λ_1 is the largest eigenvalue, we have

$$\sum_{j=1}^d \lambda_j^{2t}\,\alpha_0[j]^2 \;\;\ge\;\; \lambda_1^{2t}\,\alpha_0[1]^2, \quad \lambda_i^{2t} \;\le\; \lambda_2^{2t} \text{ for } i \ge 2.$$

Hence

$$\sum_{i=2}^d \alpha_t[i]^2 \;\le\; \frac{\lambda_2^{2t}\,\sum_{i=2}^d \alpha_0[i]^2}{\lambda_1^{2t}\,\alpha_0[1]^2} \;=\; \left(\frac{\lambda_2}{\lambda_1}\right)^{2t} \;\frac{\sum_{i=2}^d \alpha_0[i]^2}{\alpha_0[1]^2},$$

as required.

(f)

Since x_t is always a unit vector, the distance to u_1 can be measured by the coefficients in the directions $u_2, \ldots, u_d$. From part (e),

$$\sum_{i=2}^d \alpha_t[i]^2 \;\le\; \left(\frac{\lambda_2}{\lambda_1}\right)^{2t} \;\frac{\sum_{i=2}^d \alpha_0[i]^2}{\alpha_0[1]^2}.$$

To ensure

$$\|x_t - u_1\|_2^2 \;=\; \sum_{i=2}^d \alpha_t[i]^2 \;\le\; \varepsilon,$$

it suffices to require

$$\left(\frac{\lambda_2}{\lambda_1}\right)^{2t} \;\frac{\sum_{i=2}^d \alpha_0[i]^2}{\alpha_0[1]^2} \;\le\; \varepsilon.$$

Solving for t gives

$$t \;\ge\; \frac{1}{2\,\ln(\lambda_1/\lambda_2)}\;\ln\!\Big(\frac{\sum_{i=2}^d \alpha_0[i]^2}{\varepsilon\,\alpha_0[1]^2}\Big).$$

Thus, on the order of

$$O\Big(\log(1/\varepsilon)\;/\;\log(\lambda_1/\lambda_2)\Big)$$

iterations are needed to get within ε of the principal eigenvector.

$$\boxed{\text{After } T \;\ge\; \frac{1}{2\,\ln(\lambda_1/\lambda_2)}\;\ln\!\Big(\tfrac{\sum_{i=2}^d \alpha_0[i]^2}{\varepsilon\,\alpha_0[1]^2}\Big) \text{ iterations, we have } \|x_T - u_1\|_2^2 \;\le\; \varepsilon.}$$

Optimality of PCA for Minimizing Reconstruction Error

(a)

Let $X \in \mathbb{R}^{n\times d}$ with $x_i^\top$ as the ith row of X, and $V \in \mathbb{R}^{k\times d}$ with $v_i^\top$ as the ith row of V. Then:

$$\sum_{i=1}^{n} \left\| x_i - \sum_{j=1}^{k} (x_i^\top v_j) v_j \right\|_2^2 = \left\| X - XV^\top V \right\|_F^2$$

where $\|M\|_F = \sqrt{\mathrm{tr}(M^\top M)} = \sqrt{\mathrm{tr}(MM^\top)}$ is the Frobenius norm.

(b)

Let $V \in \mathbb{R}^{d\times k}$ be the matrix whose columns are the vectors $v_1, v_2, \ldots, v_k$, i.e.,

$$V = \begin{bmatrix} | & & | \\ v_1 & \cdots & v_k \\ | & & | \end{bmatrix}.$$

Then:

$$(V^\top V)_{ij} = v_i^\top v_j.$$

So, the matrix $V^\top V \in \mathbb{R}^{k\times k}$ has entries: - 1 on the diagonal when $i = j$ because $\|v_i\|_2 = 1$, - 0 off the diagonal when $i \neq j$ because $v_i^\top v_j = 0$.

Hence,

$$V^\top V = I_k.$$

Now consider the matrix $VV^\top \in \mathbb{R}^{d\times d}$, which is a projection matrix onto the column space of V.

However, in the context of the formulation in part (a), the reconstruction is $XVV^\top$, and since V has orthonormal columns, it satisfies:

$$V^\top V = I_k.$$

Thus, the conditions $\|v_i\| = 1$ and $v_i^\top v_j = 0$ for $i \neq j$ are equivalent to:

$$V^\top V = I_k,$$

which is the standard orthonormality constraint for the columns of V.

Therefore, the set of constraints:

$$\|v_i\| = 1 \text{ for all } i, \quad v_i^\top v_j = 0 \text{ for all } i \neq j$$

is equivalent to:

$$V^\top V = I_k.$$

(c)

Let $V \in \mathbb{R}^{d\times k}$ be any matrix satisfying $V^\top V = I_k$. Then define:

$$M := XVV^\top.$$

This matrix M has rank at most k because:

$$\mathrm{rank}(M) = \mathrm{rank}(XVV^\top) \leq \mathrm{rank}(VV^\top) \leq k.$$

Therefore, M is a feasible candidate for problem (3). Hence:

$$\|X - XVV^\top\|_F^2 = \|X - M\|_F^2 \geq \min_{\mathrm{rank}(M)\leq k} \|X - M\|_F^2.$$

This inequality holds for *any* V satisfying $V^\top V = I_k$, so it also holds for the optimal V^* that minimizes (2).

Therefore, the optimal value of problem (2) is lower bounded by the optimal value of problem (3).

(d)

From the SVD of X:

$$X = \sum_{i=1}^{d} \sigma_i w_i z_i^\top, \quad \text{so} \quad X^\top = \sum_{i=1}^{d} \sigma_i z_i w_i^\top.$$

Then,

$$X^\top X = \left(\sum_{i=1}^{d} \sigma_i z_i w_i^\top \right) \left(\sum_{j=1}^{d} \sigma_j w_j z_j^\top \right) = \sum_{i=1}^{d} \sum_{j=1}^{d} \sigma_i \sigma_j z_i (w_i^\top w_j) z_j^\top.$$

Since the w_i are orthonormal ($w_i^\top w_j = \delta_{ij}$), only the terms with $i = j$ survive:

$$X^\top X = \sum_{i=1}^{d} \sigma_i^2 z_i z_i^\top.$$

This is the eigen-decomposition of $X^\top X$, where: - each z_i is an eigenvector, - the corresponding eigenvalue is σ_i^2.

(e)

From the Eckart–Young–Mirsky theorem, the optimal rank-k approximation to X (in Frobenius norm) is given by:

$$M^* = \sum_{i=1}^{k} \sigma_i w_i z_i^\top,$$

where: - σ_i are the top k singular values of X, - w_i are the left singular vectors of X, - z_i are the right singular vectors of X.

Let:

$$V^* := [z_1, z_2, \ldots, z_k] \in \mathbb{R}^{d \times k}, \quad W^* := [w_1, w_2, \ldots, w_k] \in \mathbb{R}^{n \times k}, \quad \Sigma_k := \operatorname{diag}(\sigma_1, \ldots, \sigma_k).$$

Then the rank-k SVD of X is:

$$M^* = W^* \Sigma_k (V^*)^\top.$$

Now observe:

$$X = \sum_{i=1}^{d} \sigma_i w_i z_i^\top = W \Sigma V^\top,$$

so:

$$XV^* = W \Sigma V^\top V^*.$$

Because $V^\top V^*$ picks the top k columns of Σ, we get:

$$XV^* = W^* \Sigma_k, \quad \text{and} \quad XV^*(V^*)^\top = W^* \Sigma_k (V^*)^\top = M^*.$$

Therefore, the optimal rank-k approximation M^* from the Eckart–Young–Mirsky theorem can be written as:

$$M^* = XV^*(V^*)^\top,$$

where V^* contains the top k right singular vectors (i.e., eigenvectors of $X^\top X$).

(f)

The matrix V^* in part (e) contains the top k eigenvectors of $X^\top X$, which are the optimal directions that minimize the projection error in the optimization problem (2), where we seek to minimize $\|X - XVV^\top\|_F^2$ subject to $V^\top V = I_k$.

(g)

Since the optimal V^* minimizes the reconstruction error in (2) and its columns are the top-k eigenvectors of $X^\top X$, the optimal basis vectors $v_1, \ldots, v_k$ in (1) must be exactly those top-k eigenvectors.

1 Implementing Eigen-Faces

(a)

Input

```
k_list = [1,2,4,8,16, 32,64,128,256]


# ------------------------------
X_train_np = X_train.cpu().numpy()  # shape (280, 4096)
n_samples, d = X_train_np.shape

# Center the data (subtract the mean)

X_mean = np.mean(X_train_np, axis=0)
X_centered = X_train_np - X_mean

# Compute PCA (SVD)
U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
explained_variance = S**2 / (n_samples - 1)
total_variance = np.sum(explained_variance)
explained_variance_ratio = explained_variance / total_variance  # shape: (4096,)


plt.figure(figsize=(8,8))
# ------------------------------
# Plot Explained Variance vs. k and k/total_dimension curve
cumulative_variance = np.cumsum(explained_variance_ratio)

plt.figure(figsize=(8, 6))

# Plot: cumulative explained variance
plt.plot(k_list, [cumulative_variance[k-1]*100 for k in k_list], marker='o', label='Explained␣
    Variance␣(%)')

# Plot: fraction of dimension used (k/d)
plt.plot(k_list, [k/d*100 for k in k_list], marker='s', label='k␣/␣d␣(%)')
# ------------------------------
plt.xlabel("Number␣of␣Principal␣Components␣(k)")
plt.ylabel("Percentage␣(%)")
plt.title("Explained␣Variance␣and␣k/d␣vs.␣k")
plt.legend()
plt.grid(True)

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

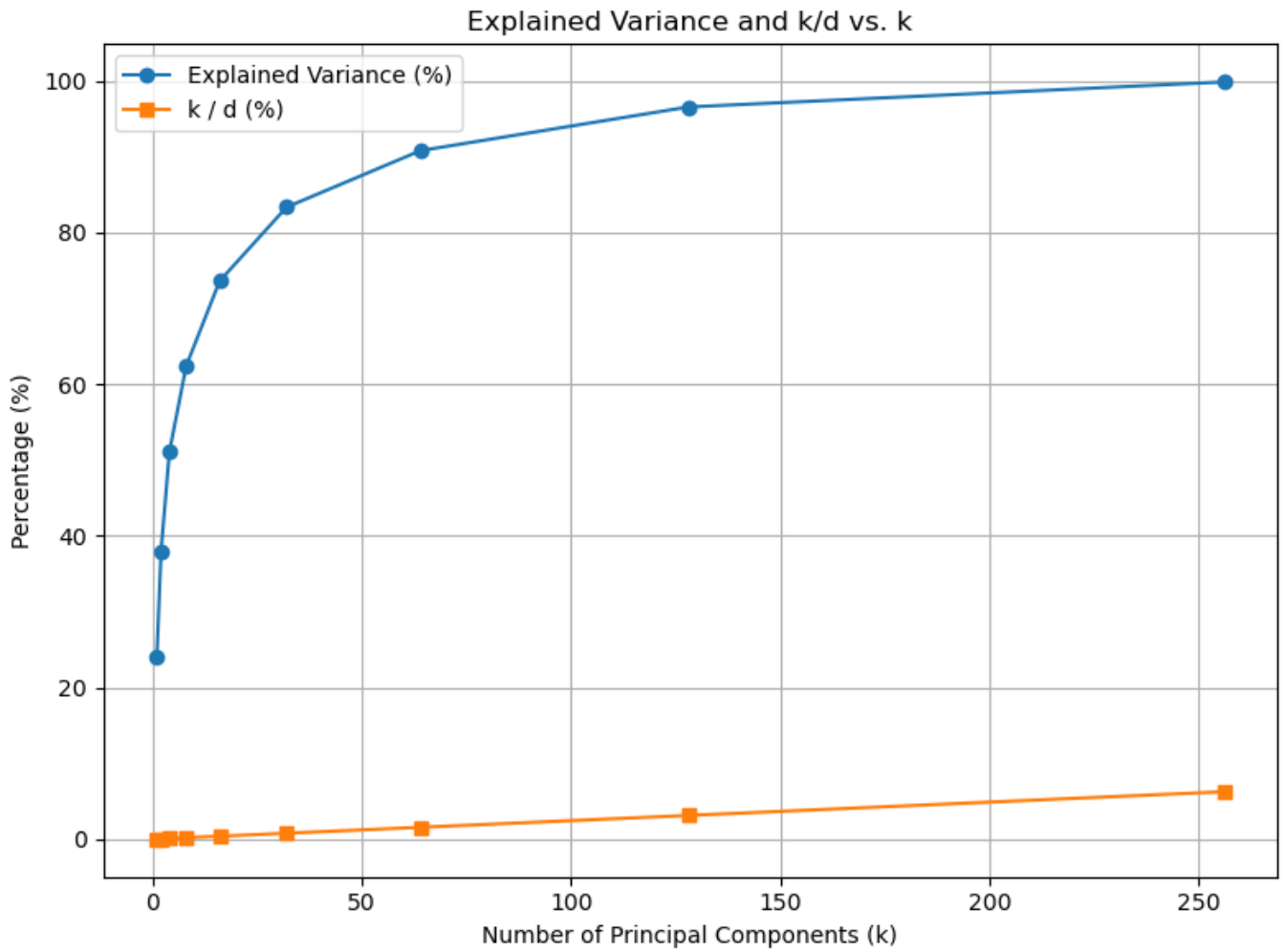

Figure 1: 3(a)

(b)

Input

```
# Create a 1x6 grid for plotting
fig, axes = plt.subplots(1, 6, figsize=(15, 5))
fig.suptitle("Top 6 Eigenvectors", fontsize=16)

for i in range(6):
    # ------------------------------
    eigenface = Vt[i].reshape(64, 64)  # reshape the i-th eigenvector
    axes[i].imshow(eigenface, cmap='gray')
    # ------------------------------
    axes[i].axis('off')
    axes[i].set_title(f"Eigenvector {i+1}")

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

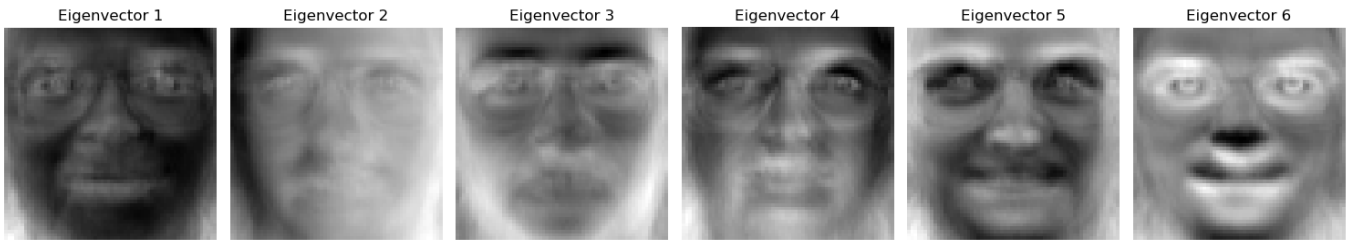

Figure 2: 3(b)

(c)

Input

```
# Parameters
k_reconstruct = [10, 50, 100]
num_faces = 6  # first 6 training images

# Prepare 4x6 subplot grid
fig, axes = plt.subplots(4, 6, figsize=(12, 8))
fig.suptitle("PCA Reconstruction with Varying Number of Components", fontsize=16)

# Original 6 faces (top row)
for i in range(num_faces):
    original_face = X_train_np[i].reshape(64, 64)
    axes[0, i].imshow(original_face, cmap='gray')
    axes[0, i].axis('off')
    if i == 0:
        axes[0, i].set_ylabel("Original", fontsize=12)

# Reconstructions using top-k PCs
for row, k in enumerate(k_reconstruct):
    # Project into k-dimensional space and back
    V_k = Vt[:k]  # shape: (k, 4096)
    X_proj = (X_centered[:num_faces] @ V_k.T)  # shape: (6, k)
    X_reconstructed = X_proj @ V_k  # shape: (6, 4096)
    X_reconstructed += X_mean  # add the mean back

    for i in range(num_faces):
        img = X_reconstructed[i].reshape(64, 64)
        axes[row+1, i].imshow(img, cmap='gray')
        axes[row+1, i].axis('off')
        if i == 0:
            axes[row+1, i].set_ylabel(f"k={k}", fontsize=12)

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

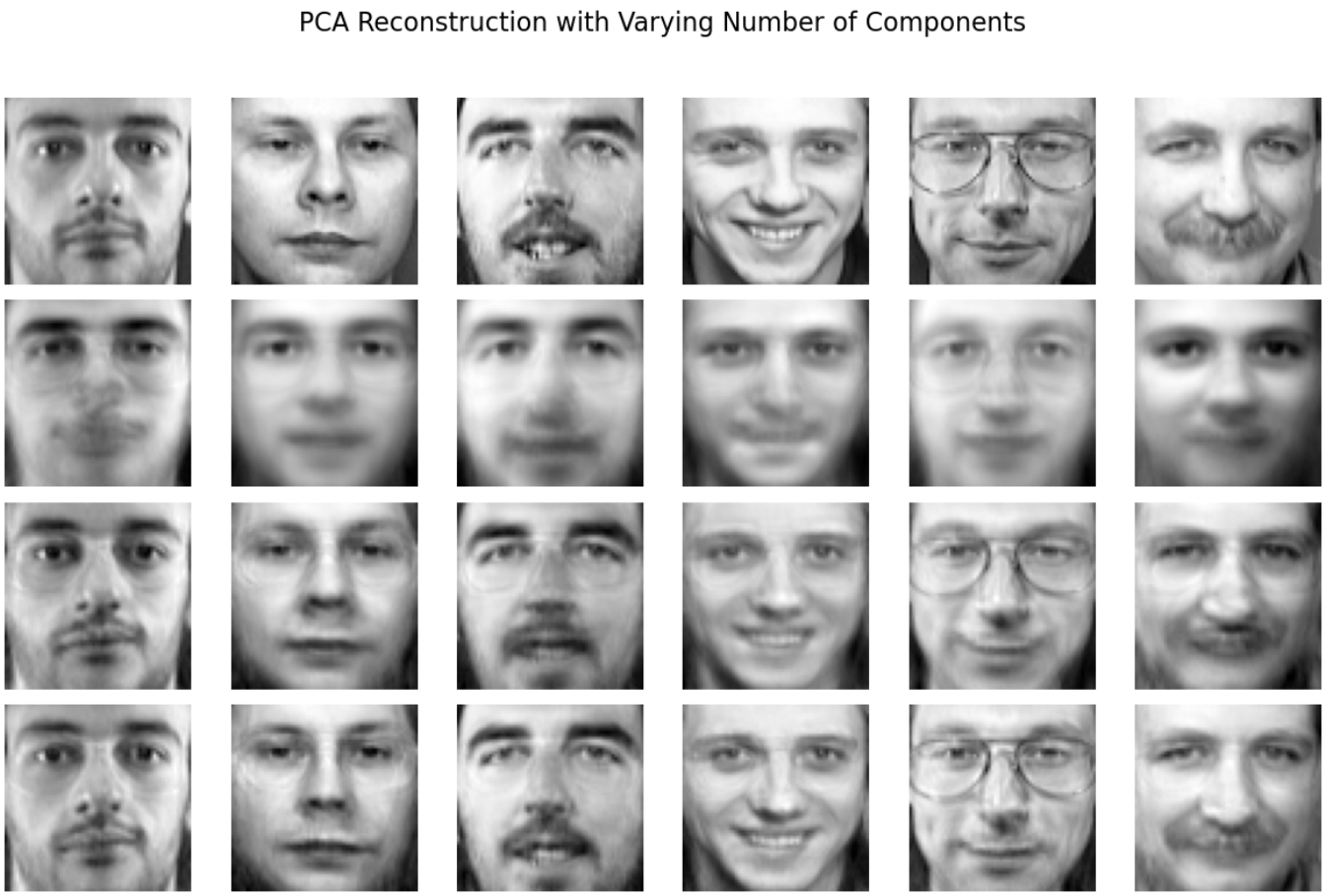

Figure 3: 3(c)

(d)

Input

```
# ------------------------------
# Define logistic regression model and solve (using pytorch+SGD or scikitlearn)
# remember to print your test accuracy on full dimensions.
# ------------------------------
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Step 1: Convert tensors to NumPy arrays
X_train_np = X_train.cpu().numpy()      # shape: (280, 4096)
X_test_np = X_test.cpu().numpy()        # shape: (120, 4096)
y_train_np = y_train.cpu().numpy()      # shape: (280,)
y_test_np = y_test.cpu().numpy()        # shape: (120,)

# Step 2: Train logistic regression on full features
clf_raw = LogisticRegression(max_iter=1000, solver='saga', multi_class='multinomial')
clf_raw.fit(X_train_np, y_train_np)

# Step 3: Predict and evaluate on test set
y_pred_raw = clf_raw.predict(X_test_np)
accuracy_raw = accuracy_score(y_test_np, y_pred_raw) * 100


print(f"Test␣accuracy␣using␣all␣4096␣raw␣pixels:␣{accuracy_raw:.2f}%")
```

(d)

Output

```
Test accuracy using all 4096 raw pixels: 98.33%
```

(e)

Input

```
eigvecs = torch.from_numpy(Vt.T).float().to(device)  # shape (4096, 4096)
eigvals = torch.from_numpy((S**2 / (280 - 1))).float().to(device)  # shape (4096,)
total_variance = torch.sum(eigvals)

X_train_centered = torch.from_numpy(X_train_np - X_mean).float().to(device)
X_test_centered  = torch.from_numpy(X_test_np - X_mean).float().to(device)

num_classed = 40

# ------------------------------
# PCA + Logistic Regression for different k values
# ------------------------------
k_list = [1,2,4,8,16, 32,64,128,256] # you can change this list if needed
pca_accuracies = []
explained_variances = []
k_ratios = []  # k / total_dimension (total_dimension = 4096)

for k in k_list:
    # Select top-k eigenvectors (principal components)
    pcs = eigvecs[:, :k]  # shape (4096, k)

    # Project centered data onto these components
    X_train_pca = X_train_centered @ pcs  # shape (n_train, k)
    X_test_pca  = X_test_centered @ pcs   # shape (n_test, k)

    # Compute explained variance ratio for top-k components (in percent)
    explained = torch.sum(eigvals[:k]) / total_variance
    explained_variances.append(explained.item() * 100)

    # Compute ratio of k to total dimension (as percent)
    k_ratios.append((k / X_train_centered.shape[1]) * 100)

    # Train logistic regression on PCA-reduced data
    X_train_pca_np = X_train_pca.cpu().numpy()
    X_test_pca_np = X_test_pca.cpu().numpy()
    y_train_np = y_train.cpu().numpy()
    y_test_np = y_test.cpu().numpy()

    clf = LogisticRegression(max_iter=1000, solver='saga', multi_class='multinomial')
    clf.fit(X_train_pca_np, y_train_np)
    y_pred = clf.predict(X_test_pca_np)
    accuracy_pca = accuracy_score(y_test_np, y_pred) * 100
    pca_accuracies.append(accuracy_pca)

    print(f"k = {k:3d}: Explained Variance = {explained.item()*100:5.2f}%, k/4096 = {k
        /4096*100:5.2f}%, Test Accuracy = {accuracy_pca:5.2f}%")

# ------------------------------
# Plot Test Accuracy and Explained Variance vs. k
# ------------------------------
plt.figure(figsize=(12, 5))

# Plot Test Accuracy vs. k
plt.plot(k_list, pca_accuracies, marker='o', label='PCA + Logistic Regression')
plt.axhline(y=accuracy_raw, color='red', linestyle='--', label='Raw (centered)')
plt.xlabel("Number of Principal Components (k)")
plt.ylabel("Test Accuracy (%)")
plt.title("Test Accuracy vs. Number of Principal Components")
plt.legend()
plt.grid(True)

plt.show()
```

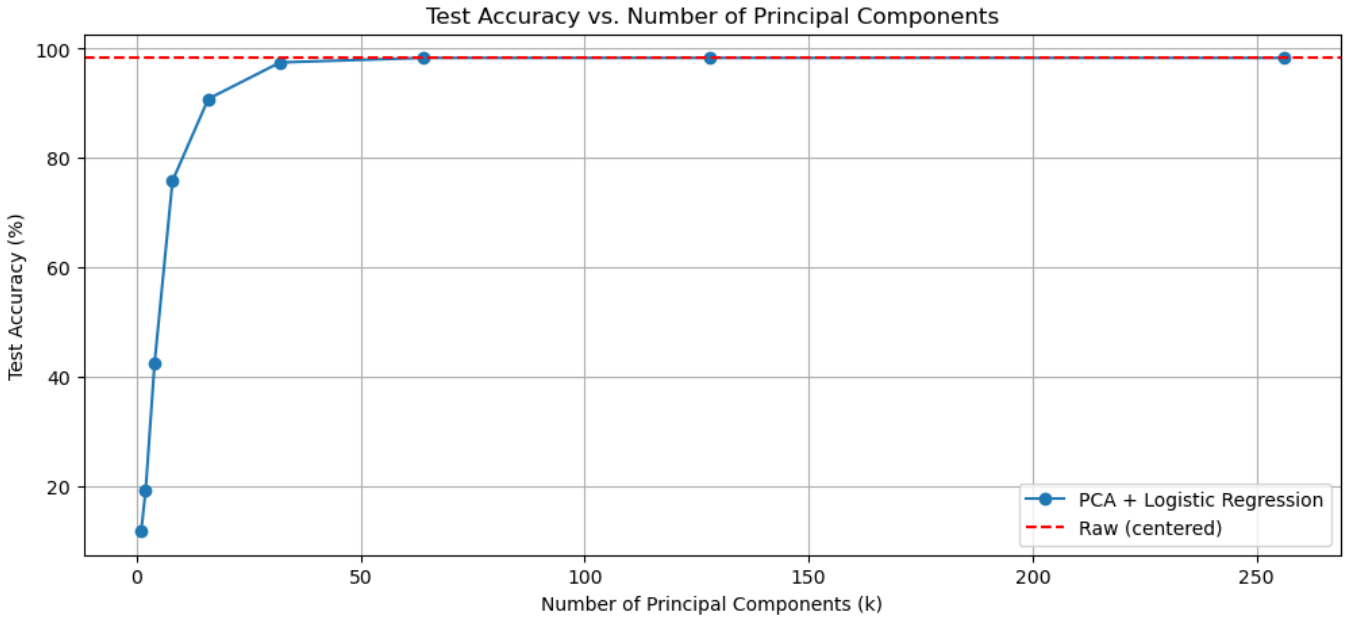

Figure 4: 3(e)

(f)

According to the output of part 3(e), we need $k = 16$ principal components to achieve over 90% test accuracy, and this corresponds to a fraction of the original dimension given by $k/d = \frac{16}{4096} = 0.0039 = \mathbf{0.39}\%$.