eigenfaces_from_scratch

August 21, 2025

1 Eigenfaces — From Scratch

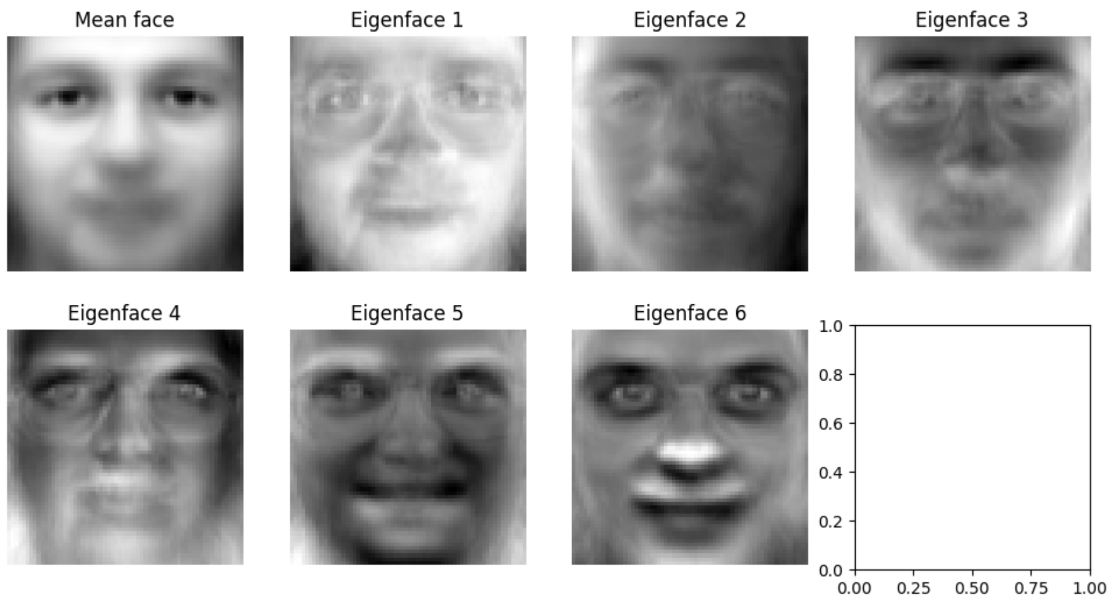
This notebook implements the eigenfaces pipeline using PCA from scratch: - load a face dataset (try Olivetti; fallback to digits) - compute mean face, subtract, compute PCA using SVD/eigendecomposition - show top eigenfaces and reconstructions

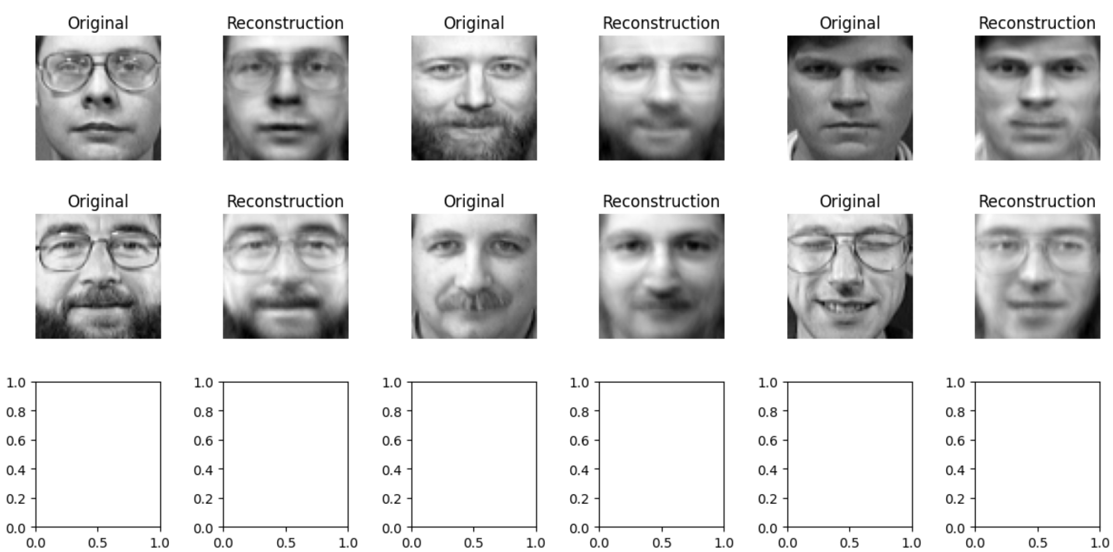
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_olivetti_faces, load_digits
from sklearn.utils import Bunch

def load_faces_fallback():
    # Try Olivetti (may require internet). If unavailable, fallback to digits␣
↪dataset (small images)
    try:
        data = fetch_olivetti_faces(shuffle=True, random_state=0)
        images = data.images  # (n_samples, h, w)
        flat = data.data      # (n_samples, h*w)
        target = data.target
        name = 'olivetti'
    except Exception as e:
        print('Could not fetch Olivetti faces, falling back to digits dataset.␣
↪Error:', e)
        d = load_digits()
        # resize digits (8x8) into 16x16 by simple repeat to make visualization␣
↪nicer
        images = d.images
        images = np.array([np.kron(img, np.ones((2,2))) for img in images])
        flat = images.reshape(images.shape[0], -1)
        target = d.target
        name = 'digits_fallback'
    return Bunch(images=images, flat=flat, target=target, name=name)

faces = load_faces_fallback()
X = faces.flat  # (n_samples, n_pixels)
n_samples, n_pixels = X.shape
h = faces.images.shape[1]; w = faces.images.shape[2]
```

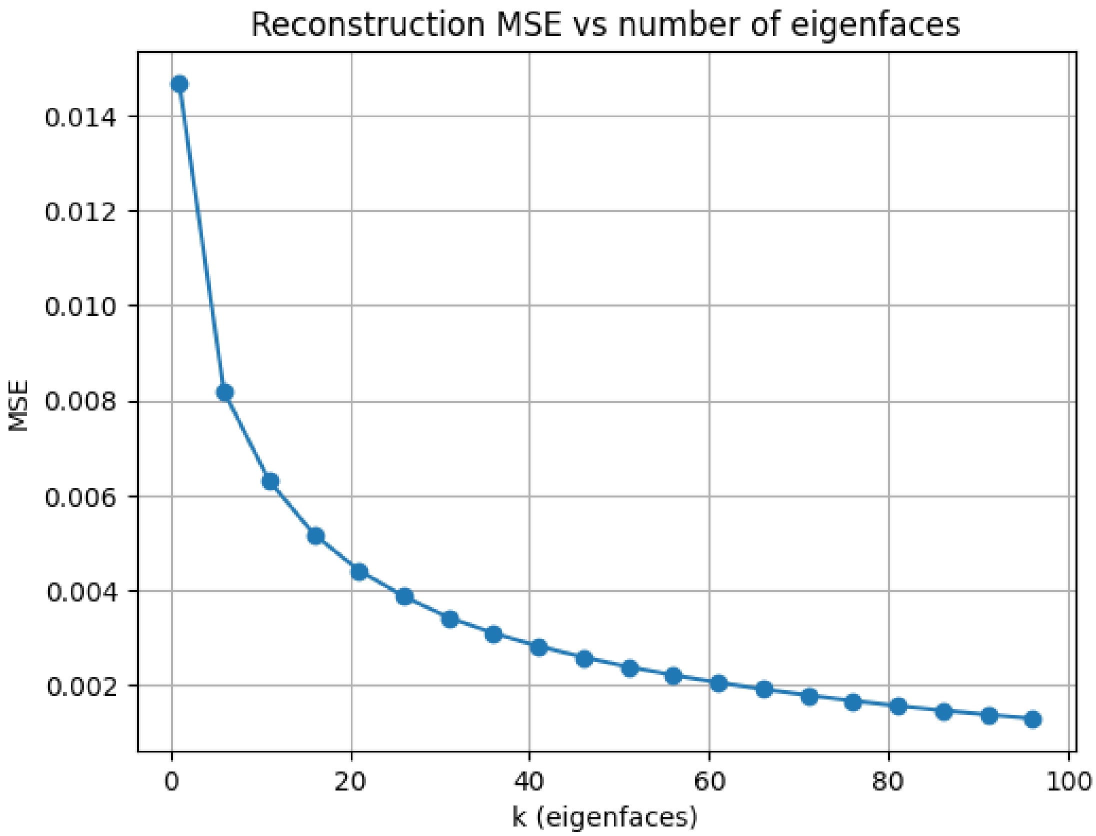
```
print('Dataset:', faces.name, '| n_samples:', n_samples, 'image size:', h, 'x',␣
↪w)

# PCA via SVD (numerically stable)
Xc = X - X.mean(axis=0)
U, S, VT = np.linalg.svd(Xc, full_matrices=False)
components = VT  # rows are principal directions in pixel-space
explained_variance = (S**2) / (n_samples - 1)
explained_ratio = explained_variance / explained_variance.sum()

# Show mean face and top 6 eigenfaces
mean_face = X.mean(axis=0).reshape(h, w)
fig, axes = plt.subplots(2, 4, figsize=(12,6))
axes = axes.ravel()
axes[0].imshow(mean_face, cmap='gray'); axes[0].set_title('Mean face'); axes[0].
↪axis('off')
for i in range(1,7):
    ef = components[i-1].reshape(h, w)
    axes[i].imshow(ef, cmap='gray'); axes[i].set_title(f'Eigenface {i}');␣
↪axes[i].axis('off')
plt.show()

# Reconstruction example using top k eigenfaces
def reconstruct_faces(X, components, k, mean_face_vec):
    comps_k = components[:k]
    proj = (X - mean_face_vec) @ comps_k.T
    rec = proj @ comps_k + mean_face_vec
    return rec

idx = np.random.choice(n_samples, size=6, replace=False)
recs = reconstruct_faces(X, components, k=30, mean_face_vec=X.mean(axis=0))

fig, axes = plt.subplots(3, 6, figsize=(12,6))
axes = axes.ravel()
for i, ind in enumerate(idx):
    axes[2*i].imshow(X[ind].reshape(h,w), cmap='gray'); axes[2*i].
↪set_title('Original'); axes[2*i].axis('off')
    axes[2*i+1].imshow(recs[ind].reshape(h,w), cmap='gray'); axes[2*i+1].
↪set_title('Reconstruction'); axes[2*i+1].axis('off')
plt.tight_layout()
plt.show()

# MSE vs number of components
ks = list(range(1, 101, 5))
mses = []
mean_vec = X.mean(axis=0)
for k in ks:
```

```
    rec_all = reconstruct_faces(X, components, k, mean_vec)
    mses.append(np.mean((X - rec_all)**2))
plt.plot(ks, mses, marker='o')
plt.xlabel('k (eigenfaces)'); plt.ylabel('MSE'); plt.title('Reconstruction MSE␣
  ↪vs number of eigenfaces')
plt.grid(True)
plt.show()
```

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027
to C:\Users\Aryan Gupta\scikit_learn_data
Dataset: olivetti | n_samples: 400 image size: 64 x 64
```







1.1 1. Notes

- If Olivetti can't be downloaded in your environment, the fallback uses scikit-learn digits to demonstrate the pipeline.
- For production face recognition, additional preprocessing (alignment, histogram equalization) is essential.