Implementation  We implement the greedy nonnegative MF algorithm described in (Seung and Lee, 1999). Note: only one of the two re-normalization steps (columns of $W$ and rows of $H$) is needed, such that $WH$ has the same scale as the data matrix $X$. We choose to renormalize the rows of $H$, so as to make the architypes have row sum one. The convergence criterion is set to be

$$\max \left\{ \frac{\|W^t - W^{t+1}\|_F}{\|W^t\|_F}, \frac{\|H^t - H^{t+1}\|_F}{\|H^t\|_F} \right\} \leq 0.001.$$ 

On the MNIST dataset, the algorithm will converge in at most 1000 steps.

Results

1. Figure 1 plots the 10 leading eigenvectors of the sample covariance. As is seen, the first eigenvector picks up the digit 3, and the subsequent eigenvectors show some signal in the data but in a more complicated way. Eigenvectors 6-10 barely have any visible structure that is related to digits.

![Figure 1: 10 leading eigenvectors of the sample covariance.](image)

2. We perform nonnegative matrix factorization with $r \in \{3, 6, 10\}$ and plot the architypes in figure 2. NMF gives much better architypes than PCA: with $r = 3$, it picks up the digits 0, 9, 1. With $r = 6$ and $r = 10$, it successfully recognizes more major patterns in handwritten digits. In particular, the patterns in $r = 10$ seem to cover most of the structures needed to completely determine the label of any digit.

3. Figure 3 plots the coefficient vectors of digits $\{0, \ldots, 9\}$, randomly chosen from the dataset. PCA coefficients are very messy and uniform. NMF coefficient are more peaky, with each digit having one or two high coefficients. For example, we can see that 0 and 6 shares architype 0; (2, 3, 5) shares architype 8. These suggest that NMF gives a more accurate low-dimensional representation of the handwritten digits.
A Python code

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

def plotgs(x, ax, m=28, n=28, rescale=False):
    if rescale == True:
        x = 255 * (x - np.min(x)) / (np.max(x) - np.min(x))
    ax.imshow(np.reshape(x, (m,n)), cmap='gray')

def Division(A, B):
    return np.nan_to_num(A / B)
```

Figure 2: Architypes from NMF.

Figure 3: Coefficient vectors of digits 0-9.
Nonnegative matrix factorization

Args:
  X: n*d data matrix
  r: hidden dimension

Returns:
  W, H: nonnegative matrices such that X \approx W * H

```python
def NMF(X, r, eps=1e-3):
    n, d = X.shape
    W_new = np.random.rand(n, r)
    H_new = np.random.rand(r, d)
    while True:
        W = W_new
        H = H_new
        W_tilde = W * np.dot(Division(X, np.dot(W, H)), H.T)
        # W_new = np.dot(W_tilde, np.diag(1.0/np.sum(W_tilde,0)))
        W_new = W_tilde
        H_tilde = H * np.dot(W_new.T, Division(X, np.dot(W_new, H)))
        H_new = np.dot(np.diag(1.0/np.sum(H_tilde,1)), H_tilde)
        H_err = np.linalg.norm(H_new - H) / np.linalg.norm(H)
        W_err = np.linalg.norm(W_new - W) / np.linalg.norm(W)
        print(H_err, W_err)
        if H_err <= eps and W_err <= eps:
            break
    return W, H
```
for i in range(num_r):
    r = r_vec[i]
    W, H = NMF(pixels_rsc, r)
    fn = fn_format % r
    fig, axes = plt.subplots(1, r, figsize=(r+1, 1))
    for j in range(r):
        ax = axes[j]
        ax.get_xaxis().set_ticks([])
        ax.get_yaxis().set_ticks([])
        plotgs(H[j, :], ax, rescale=True)
    plt.savefig(fn)
    plt.close()

# Plot coefficients (implicitly using r=10)
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(14, 6))
ax0.set_title('PCA')
ax1.set_title('Nonnegative Matrix Factorization')
for i in range(10):
    idx = np.where(labels == i)[0]
    idx_i = np.random.choice(idx)
    ax0.plot(np.dot(V[:, :r].T, pixels[idx_i, :].T), label='%d' % i)
    ax1.plot(W[idx_i, :], label='%d' % i)
ax0.legend(loc='best', prop={'size':10})
ax1.legend(loc='best', prop={'size':10})
plt.savefig('mnist_coefficients.png')
plt.close()