⇒ Assignment (1)

(1) Non-negative matrix Factorization is an alternative to PCA when data and factors can be cast as non-negative. We seek to factorize $N \times p$ data matrix X as

$$X \approx WH$$

where W is $N \times r$ and H is $r \times p$, with $r \leq \max(N, p)$ and we assume that $x_{ij}, w_{ik}, h_{kj} \geq 0$

(2) Suppose that $x_{ij} \in N$. If we model each random variables x_{ij} as a (independent) Poisson random variable with mean $(WH)_{ij}$. Show that the log-likelihood of the model is (up to a constant)

$$L(W,H) = \sum_{i,j} \left[x_{ij} \log (WH)_{ij} - (WH)_{ij} \right].$$

The following alternating algorithm converges to a local maximum of $L(W,H)$

$$W_{ik} \leftarrow W_{ik} \frac{\sum_j h_{kj} x_{ij} / (WH)_{ij}}{\sum_j h_{kj}}$$

$$h_{kj} \leftarrow h_{kj} \frac{\sum_i W_{ik} x_{ij} / (WH)_{ij}}{\sum_i W_{ik}}$$

A function $g(x,y)$ is said to minorize a function $f(x)$ if

$$f(x,y) \cdot g(x,y) \leq f(x) \cdot g(x,x) = f(x)$$

$\Rightarrow$ Solution

The likelihood of observing x_{ij} given the mean $(WH)_{ij}$ for a poisson distribution is given by

$$P(x_{ij} | (WH)_{ij}) = \frac{e^{-(WH)_{ij}} (WH)_{ij}^{x_{ij}}}{x_{ij}!}$$

→ log likelihood

$$\log P(x_{ij} | (WH)_{ij}) = \log\left(\frac{e^{-(WH)_{ij}} (WH)_{ij}^{x_{ij}}}{x_{ij}!}\right)$$

Expanding this gives.

$$\log P(x_{ij} | (WH)_{ij}) = -(WH)_{ij} + x_{ij} \log((WH)_{ij}) - \log(x_{ij}!)$$

The term $\log(x_{ij}!)$ is independent of W and H, so its a constan and can be ignored. Since the x_{ij} are IID, the log likelihood for all observations is just sum of all observations

$$L(W,H) = \sum_{ij} \left[-(WH)_{ij} + x_{ij} \log((WH)_{ij}) \right]$$

Thus concluded.

(b) Show that under the update

$$x^{t+1} = \arg\max_x g(x, x^t),$$

the sequence $f_t = f(x^t)$ is non-decreasing

⇒ Solution.

given that,

$$x^{t+1} = \arg\max_x g(x, x^t)$$

This implies

$$g(x^{t+1}, x^t) \geq g(x, x^t) \text{ for all } x.$$

we assume that g is somehow related to the gradient or direction of steepest ascent for f, then maximizing g would lead to an increase in f.

→ given the update rule and the relationship between g and f, we can infer that:

$$f^{t+1} = f(x^{t+1}) \geq f(x^t) = f^t$$

This shows f^t is non decreasing.

(c) Using concavity of the logarithm, show that for any set of r values $y_k \geq 0$ and $0 \leq c_k \leq 1$ with $\sum_{1 \leq k \leq r} c_k = 1$

$$\log\left(\sum_{1 \leq k \leq r} y_k\right) \geq \sum_{1 \leq k \leq r} c_k \log\left(y_k / c_k\right)$$

→ The concavity of the logarithm function allows us to use Jensen's inequality, which states that for a concave function f and a convex combination of points

$$f\left(\sum_i \lambda_i x_i\right) \geq \sum_i \lambda_i f(x_i)$$

where λ_i are non-negative weights such that $\sum_i \lambda_i = 1$.

→ Applying Jensen's Inequality.

$$\log\left(\sum_{k\le r} y_r\right) \ge \sum_{k\le r} c_k \log\left(y_k\right)$$

→ Introduce Multiplicative Identity.

Using the fact that $\sum_{k\le r} c_k = 1$, we can multiply each term y_k by $\frac{c_k}{c_k}$

$$\log\left(\sum_{k\le r} y_k\right) = \log\left(\sum_{k\le r} y_k \frac{c_k}{c_k}\right)$$

$$= \log\left(\sum_{k\le r} \frac{y_k}{c_k} c_k\right)$$

Using jensen inequality to modify

$$\log\left(\sum_{k\le r} \frac{y_k}{c_k} c_k\right) \ge \sum_{k\le r} c_k \log\left(\frac{y_k}{c_k}\right)$$

$$\Rightarrow \log\left(\sum_{k\le r} y_k\right) \ge \sum_{k\le r} c_k \log\left(\frac{y_k}{c_k}\right)$$

(d) Deduce that $\forall i \in \{1, N\}, j \in \{1, P\}$

$$\log\left(\sum_{k \leq r} w_{ik} h_{kj}\right) \geq \sum_{k \leq r} c_{kij} \log\left(w_{ik} h_{kj} / c_{kij}\right)$$

where $c_{kij} = \dfrac{w_{ik}^t h_{kj}^t}{\sum_{k' \leq r} w_{ik'}^t h_{k'j}^t}$

and t is the current iteration.

→ from the inequality

$$\log\left(\sum_{k \leq r} y_k\right) \geq \sum_{k \leq r} c_k \log\left(\frac{y_k}{c_k}\right)$$

also substituting

$y_k \to w_{ik} h_{kj}$

and $c_k \to c_{kij}$

$$\log\left(\sum_{k \leq r} w_{ik} h_{kj}\right) \geq \sum_{k \leq r} c_{kij} \log\left(\frac{w_{ik} h_{kj}}{c_{kij}}\right)$$

(e) Ignoring constants, show that

$$g(W,H;W^t,H^t) = \sum_{i,j,k} \Big[x_{ij} c_{kij} \left(\log w_{ik} + \log h_{kj} - \log c_{kij} \right) - w_{ik} h_{kj} \Big]$$ minorizes $L(W,H)$

⟹ Solution

To show that the function g minorizes L, we need to ensure that:

(1) $g(W,H;W^t,H^t) \leq L(W,H)$ for all W,H.

(2) $g(W,H;W,H) = L(W,H)$

→ Apply Jensen's Inequality.

rearranging

$$x_{ij} c_{kij} \left(\log w_{ik} + \log h_{kj} - \log c_{kij} \right) - w_{ik} h_{kj}$$

$$\Rightarrow \left[x_{ij} c_{kij} \log\left(\frac{w_{ik} h_{kj}}{c_{kij}} \right) - w_{ik} h_{kj} \right]$$

Now, $$c_{kij} = \frac{w_{ik}^t h_{kj}^t}{\sum_{k'=1}^{r} w_{ik'}^t h_{k'j}^t}$$

we get,

$$g(w, H; W^t, H^t) = \sum_{i,j,k} \left[x_{ij} \frac{w_{ik}^t h_{kj}^t}{\sum_{k'=1}^{r} w_{ik'}^t h_{k'j}^t} \log\left(\frac{w_{ik} h_{kj}}{\frac{w_{ik}^t h_{kj}^t}{\sum_{k'=1}^{r} w_{ik'}^t h_{k'j}^t}} \right) - w_{ik} h_{kj} \right]$$

Simplifying:

$$g(W, H; W^t, H^t) = \sum_{i,j,k} \left[x_{ij} \frac{w^t_{ik} h^t_{kj}}{\sum_{k' \le r} w^t_{ik'} h^t_{k'j}} \cdot \log\left(\frac{w_{ik} h_{kj} \sum_{k' \le r} w^t_{ik'} h^t_{k'j}}{w^t_{ik} h^t_{kj}} \right) - w_{ik} h_{kj} \right]$$

$\rightarrow$ Compare it with $L(W, H)$

$$L(W,H) = \sum_{i,j} \left[x_{ij} \log\left(\sum_{k \le r} w_{ik} h_{kj} \right) - \sum_{k \le r} w_{ik} h_{kj} \right]$$

$$f\left(\sum_i a_i x_i \right) \ge \sum_i a_i f(x_i)$$

from Jensen's Inequality

$$\log\left(\sum_{k \le r} w_{ik} h_{kj} \right) \ge$$

$$\sum_{k \le r} \frac{w^t_{ik} h^t_{kj}}{\sum_{k' \le r} w^t_{ik'} h^t_{k'j}} \log\left(\frac{w_{ik} h_{kj} \sum_{k' \le r} w^t_{ik'} h^t_{k'j}}{w^t_{ik} h^t_{kj}} \right)$$

From this we conclude that each term of g is $\leq$ to L thus proving minorization.

(f) Finally, derive update steps (3,4) by setting to zero the partial derivatives of g.

We know,

$$g(w, H, w^t, H^t) = \sum_{i,j,k} \Big[x_{ij} c_{kij} (\log w_{ik} + \log h_{kj} - \log c_{kij}) - w_{ik} h_{kj} \Big]$$

Partial derivation wrt w_{ik}

$$\frac{\partial g}{\partial w_{ik}} = \sum_j \left[x_{ij} c_{kij} \frac{1}{w_{ik}} - h_{kj} \right] = 0$$

$$\frac{\partial g}{\partial h_{kj}} = \sum_i \left[x_{ij} c_{kij} \frac{1}{h_{kj}} - w_{ik} \right] = 0$$

Now,

$$\sum_j \left[x_{ij} c_{kij} \frac{1}{w_{ik}} \right] = \sum_j h_{kj}$$

and

$$\sum_j \left[x_{ij} c_{kij} \frac{1}{h_{kj}} \right] = \sum_i w_{ik}$$

$$\therefore w_{ik} = \frac{\sum_j x_{ij} c_{kij}}{\sum_j h_{kj}}$$

$$h_{kj} = \frac{\sum_i x_{ij} c_{kij}}{\sum_i w_{ik}}$$

Hence derived.

Q3 Maximum Entropy distributions.

(a) Let $X = \{x_1, \dots, x_N\}$ be a discrete set and $p \in P(X)$ be the space of probability distributions defined over X. Define the entropy

$$H(p) := -\sum_{i=1}^{n} p_i \log(p_i)$$

By identifying $p \in P(X)$ with a point $\vec{p} = (p(x_1), \dots, p(x_N))$ in the N-dimensional simplex

$$\Delta_N := \{y \in \mathbb{R}^N, y_i \geq 0, \Sigma_i y_i = 1\},$$

show that H is a concave function in Δ_N.

$\Rightarrow$ Solution

$$H(p) = -\sum_{i=1}^{n} p_i \log(p_i)$$

Taking a second derivative of H,

$$\frac{\partial^2 H(p)}{\partial p_i \partial p_j} = \begin{cases} -\frac{1}{p_i}, & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

So, the Hessian matrix $H''(p)$ is diagonal elements $-\frac{1}{p_i}$ which are non positive since $p_i > 0$. therefore H. is concave.

(b) Show that H is non-negative in the simplex, and that its maximum is attained at the uniform distribution $(1/N, \ldots, 1/N)$, with maximum entropy $\log N$.

$$H(p) = -\sum_{i=1}^{N} p_i \log(p_i)$$

each term $p_i \log(p_i)$ is non-positive because $\log(p_i)$ is negative for $0 < p_i < 1$, and $p_i \log(p_i) = 0$ for $p_i = 0$. Thus, $H(p)$ is non-negative

To find the maximum of H, we take the derivative and set it to zero:

$$\frac{\partial H(p)}{\partial p_i} = -\log(p_i) - 1 = 0 \Rightarrow$$

$$p_i = \frac{1}{e}$$

Since $\sum_{i=1}^{N} p_i = 1$, we have

$$N \cdot \frac{1}{c} = 1 \Rightarrow p_i = \frac{1}{N}$$

Substituting back into $H(p)$ gives,

$$H_{max} = -\sum_{i=1}^{N} \frac{1}{N} \log\left(\frac{1}{N}\right) = \log(N)$$

(c) Now let us move from a discrete to a continuous domain, by setting $X = [0, R]$. Let $P(X)$ now denote the space of probability distributions admitting a density $p(x)$, $x \in X$, and define the Shann entropy as

$$H(p) = -\int_X p(x) \log p(x) \, dx$$

Show that the maximum entropy distribution in $P(X)$ is the uniform measure in X, with entropy $\log R$.

→ We need to maximize $H(p)$ subject to two constraints

(1) The probability density function $p(x)$ must be normalized $\int_0^R p(x)\,dx = 1$

(2) $p(x)$ should be non-negative for all x in $[0, R]$.

→ Apply Lagrange multipliers.

$$\mathcal{L}(p, \lambda) = -\int_x p(x) \log(p(x))\,dx + \lambda\left(1 - \int_x p(x)\,dx\right)$$

$$\frac{\partial \mathcal{L}}{\partial p(x)} = -\log(p(x)) - 1 - \lambda = 0.$$

Step 3:- Solve for the optimal $p(x)$

$$\log(p(x)) = -1 - \lambda \Rightarrow p(x) = e^{-1-\lambda}$$

Applying the normalization constraint:

$$\int_0^R e^{-1-\lambda}\,dx = 1 \Rightarrow e^{-1-\lambda} = \frac{1}{R}$$

So maximum entropy distribution is

$$p(x) = \frac{1}{R}$$

$$H_{max} = \int_0^R \frac{1}{R} \log\left(\frac{1}{R}\right) dx$$

$$= \log(R).$$

We showed that MED on the interval $[0, R]$ is the Uniform distribution with density $1/R$ and its entropy is $\log R$.

(d) We now attempt to understand maximum entropy distributions defined over $X = \mathbb{R}$. For $p \in P(\mathbb{R})$, denote the spread $\Delta_p := \sup_{p(x)\neq 0} x - \inf_{p(x)\neq 0} x$ as the smallest interval containing the support of p (where we abuse notation and identify a probability distribution in $P(\mathbb{R})$ with given mean c and spread $\Delta < \infty$ is the uniform distribution over the interval $(c - \Delta/2, c + \Delta/2)$, with entropy $\log \Delta$

→ We are given that the spread $\Delta p := \sup p(x) - \inf p(x)$ is finite, and we know the mean c. We want to maximum the entropy

$$H(p) = -\int_{-\infty}^{\infty} p(x) \cdot \log(p(x))\, dx.$$

→ To maximize $H(p)$

(1) The pDF $p(x)$ must be normalized

$$\int_{-\infty}^{\infty} p(x)\, dx = 1$$

(2) the mean c: $\int_{-\infty}^{\infty} x \cdot p(x)\, dx = c.$

→ Lagrangian is

$$\mathcal{L}(p, \lambda_1, \lambda_2) = \int_{-\infty}^{\infty} p(x) \log(p(x))\, dx + \lambda_1 \left(1 - \int_{-\infty}^{\infty} p(x)\, dx\right) + \lambda_2 \left(c - \int_{-\infty}^{\infty} x \cdot p(x)\, dx\right)$$

$$\frac{\delta J}{\delta p(x)} = -\log(p(x)) - 1 - \lambda_1 - \lambda_2 x = 0$$

Since Δ is finite, the support of $p(x)$ must be finite. With given mean c, the optimal distribution is:

$$p(x) = \begin{cases} \frac{1}{\Delta}, & \text{for } c - \Delta/2 \le x \le c + \Delta/2 \\ 0, & \text{otherwise} \end{cases}$$

Substituting:

$$H_{max} = \int_{c-\Delta/2}^{c+\Delta/2} \frac{1}{\Delta} \log\left(\frac{1}{\Delta}\right) dx$$

$$= \log(\Delta).$$

(e) Concludes that the maximum entropy distribution over $P(R)$ with given mean does not exist. How does the answer change if now we consider distribution over $P(R_+)$

Since the Δ is finite, the support of the probability is bounded meaning there is no probability distribution with maximum entropy defined over entire $\mathbb{R}$

However, if we consider $P(\mathbb{R}^+)$, the support can be unbounded to the right allowing for a distribution with a given mean to exist. A common example is the exponential distribution, which has maximum entropy among all distribution defined over $\mathbb{R}^+$ with given mean.

⇒ Conclusion

for a given mean C and finite spread Δ, the uniform distribution over the interval $(C - \Delta/2, C + \Delta/2)$ has maximum equal to $\log(\Delta)$

No maximum entropy distribution exists over the entire real line $\mathbb{R}$

with a given mean.

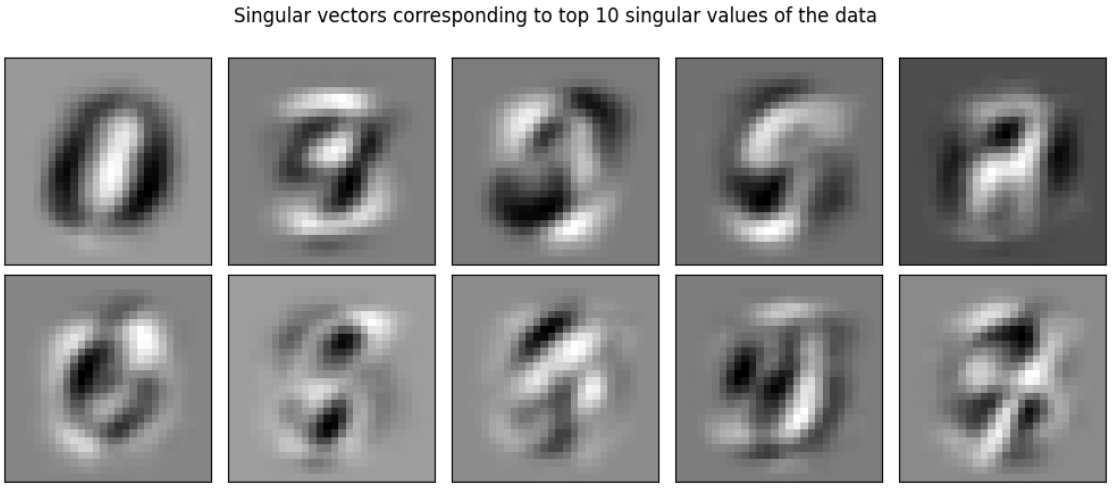
For Distribution defined over $\mathbb{R}^+$ with a given mean, maximum entropy distributions do exist such as the exponential distribution.

nmf

October 4, 2023

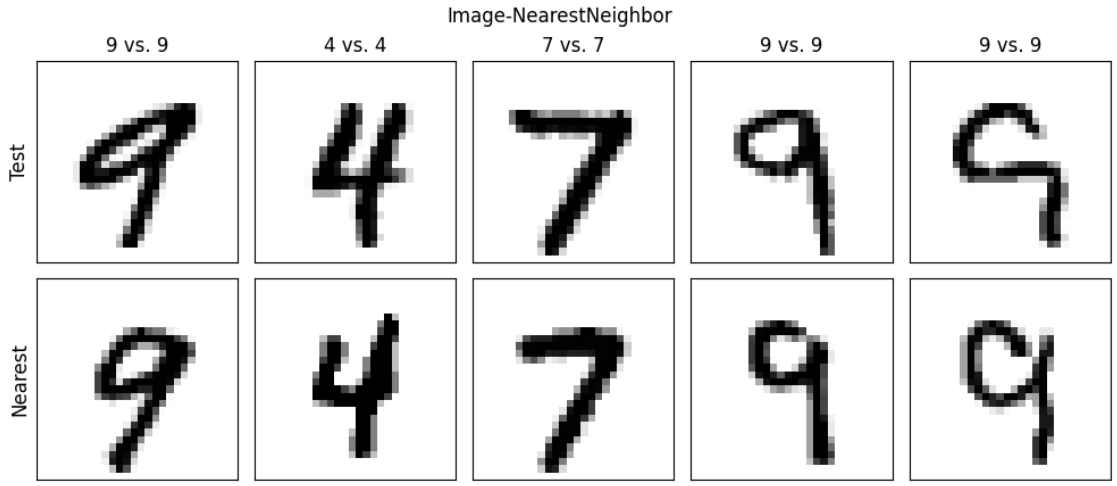
[5]:
```
"""
Inference & Representation - 2022 Fall HW1
Question 2 PCA and Non-negative matrix factorization.

"""

"""
Tools for loading the MNIST Data.
From Optimization Based Data Analysis HW1
@author: Brett
"""
import numpy as np
from mnist_tools import *
from plot_tools import *
import matplotlib.pyplot as plt

"""
Given train (in the format returned by load_train_data in mnist_tools),
and a 1d numpy array testImage you should return a tuple (digit,imageIdx). digit is
an integer giving the numerical digit value of the training image closest
to the testImage in Euclidean distance.  imageIdx is the row number of the closest
training image in the 2d array train[digit].
"""

"""
Assumes the data file is in 'mnist_all.mat'.
"""

datafile = "mnist_all.mat" #Change if you put the file in a different path
train = load_train_data(datafile)

trainarr=np.asarray(train)
trainarr = np.reshape(trainarr, (trainarr.shape[0]*trainarr.shape[1],-1))
trainarr = trainarr.astype(float)
trainarr=trainarr-trainarr.mean(axis=0)
```

[6]:
```
"""
Plot of the singular vectors corresponding
to top 10 singular values of the data.
@author: Vlad
"""
U, s, V = np.linalg.svd(trainarr, full_matrices=True)
n=10
imgs = [V[i,:] for i in range(n)]
plot_image_grid(imgs,
           "Singular vectors corresponding to top 10 singular values of the␣
 ↪data")
```



[7]:
```
"""
Plot of the results of the nearest neighbour test applied
to a principal component projection.
@author: Vlad
"""

def project(V, Images) :
     return np.dot(V.T, np.dot(V, Images))

def compute_nearest_neighbors(train, testImage, V) :
    train=[np.array(i, dtype=float) for i in train]
    testImage= np.array(testImage, dtype=float)
    digit=0
    imageIdx=0
    dist=np.linalg.norm (project (V, train[digit][imageIdx])-project (V,␣
 ↪testImage))
    for i in range(len(train)):
        for j in range (train[i].shape[0]):
```

```
            tempDist=np.linalg.norm (project(V,train[i][j])-project(V,␣
 ↪testImage))
            if tempDist<dist:
                digit=i
                imageIdx =j
                dist= tempDist
    return digit, imageIdx

n=8
U, s, V = np.linalg.svd(trainarr, full_matrices=False)
V=V[0:n,:]


test,testLabels = load_test_data(datafile)


imgs = []
TestLabels = []
for i in range(len(testLabels)) :
    trueDigit = testLabels[i]
    testImage = test[i]
    (nnDig,nnIdx) = compute_nearest_neighbors(train,testImage,V)
    imgs.extend( [testImage,train[nnDig][nnIdx,:]] )
    TestLabels.append(nnDig)

row_titles = ['Test','Nearest']
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(testLabels,testLabels)]
plot_image_grid(imgs,
                    "Image-NearestNeighbor",
                    ␣
 ↪(28,28),len(testLabels),2,True,row_titles=row_titles,col_titles=col_titles)
```



[8]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

# Loading the MNIST data
datafile = "mnist_all.mat"
mnist_data = scipy.io.loadmat(datafile)

# Checking the keys in the loaded data
mnist_data.keys()
```

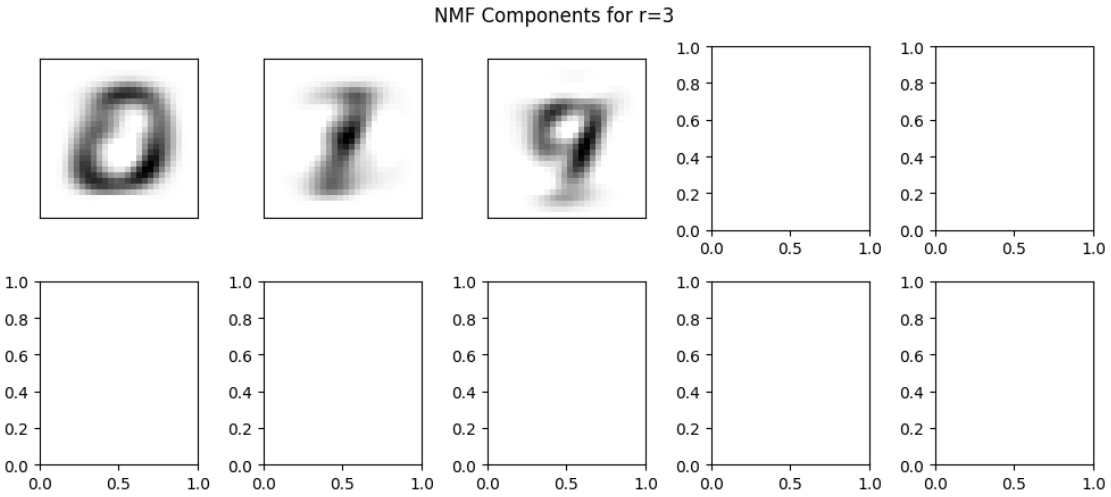
[8]:
```
dict_keys(['__header__', '__version__', '__globals__', 'train0', 'test0',
'train1', 'test1', 'train2', 'test2', 'train3', 'test3', 'train4', 'test4',
'train5', 'test5', 'train6', 'test6', 'train7', 'test7', 'train8', 'test8',
'train9', 'test9'])
```

[9]:
```
# Extracting training and testing data and labels
def extract_data_and_labels(mnist_data):
    data = []
    labels = []
    for i in range(10):
        train_data_key = f'train{i}'
        test_data_key = f'test{i}'
        train_data = mnist_data[train_data_key]
        test_data = mnist_data[test_data_key]
        combined_data = np.vstack((train_data, test_data))
        data.append(combined_data)
        labels.append(np.full((combined_data.shape[0],), i))
    return data, labels

# Getting the data and labels
data, labels = extract_data_and_labels(mnist_data)

# Checking the shape of extracted data for a specific digit
data[0].shape, labels[0].shape
```

[9]:
```
((6903, 784), (6903,))
```

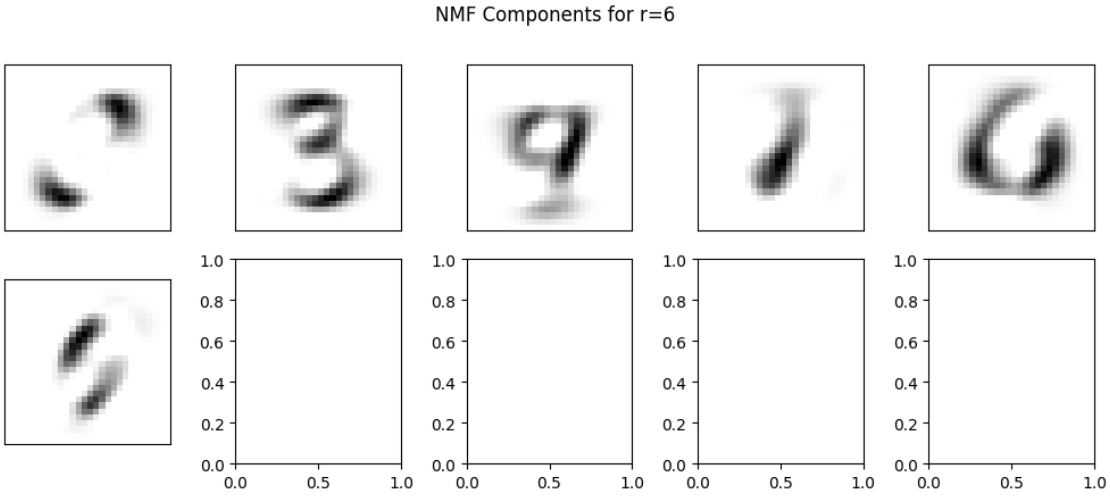
[12]:
```
from sklearn.decomposition import NMF
import plot_tools

# Function to apply NMF and plot the components
def apply_nmf_and_plot(data, n_components):
    # Combining all digit data
    all_data = np.vstack(data)
```

```
    # Applying NMF
    nmf = NMF(n_components=n_components, init='random', random_state=0)
    W = nmf.fit_transform(all_data)
    H = nmf.components_

    # Plotting the components
    plot_tools.plot_image_grid(H, f"NMF Components for r={n_components}",␣
 ↪image_shape=(28, 28))

# Plotting function (from the provided plot_tools.py with minor modifications)
# def plot_image_grid(images, title, image_shape=(28, 28), n_col=5, n_row=2):
#     fig, axes = plt.subplots(nrows=n_row, ncols=n_col, figsize=(2. * n_col, 2.
 ↪26 * n_row))
#     axes = axes.flatten()  # Flattening the axes array to simplify the␣
 ↪indexing
#     for i, comp in enumerate(images):
#         ax = axes[i]
#         ax.imshow(comp.reshape(image_shape), cmap=plt.cm.gray_r,␣
 ↪interpolation='nearest')
#         ax.set_xticks(())
#         ax.set_yticks(())

#     fig.suptitle(title)
#     plt.show()

# Re-running the NMF and plotting for r in {3, 6, 10}
for r in [3, 6, 10]:
    apply_nmf_and_plot(data, r)
```

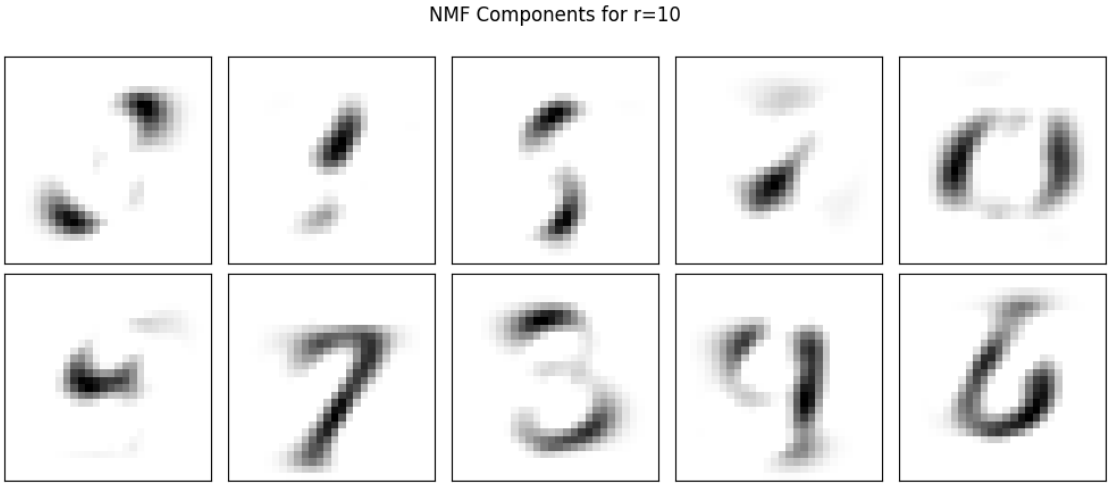


/home/karanvora/miniconda3/lib/python3.8/site-

```
packages/sklearn/decomposition/_nmf.py:1710: ConvergenceWarning: Maximum number
of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
```

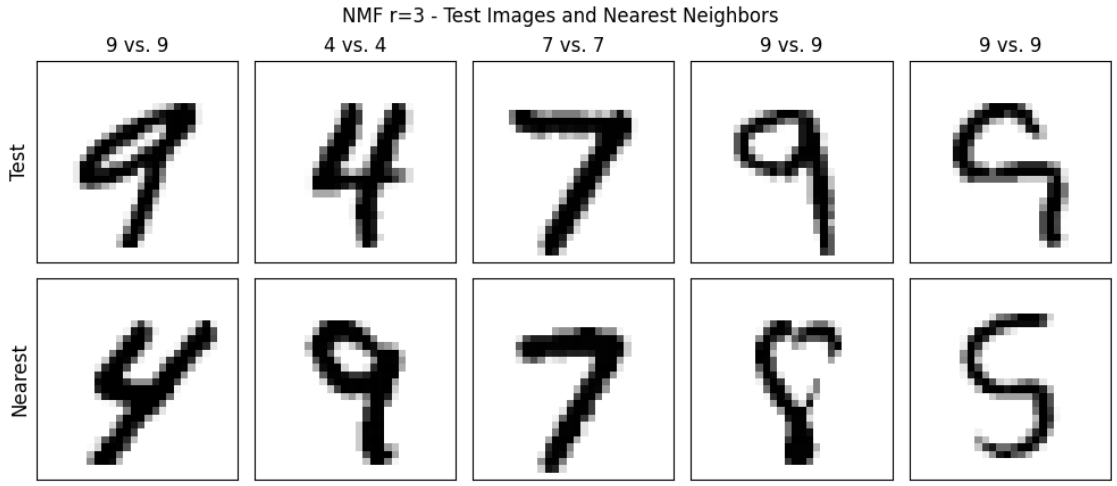



```
/home/karanvora/miniconda3/lib/python3.8/site-
packages/sklearn/decomposition/_nmf.py:1710: ConvergenceWarning: Maximum number
of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
```



[24]:
```
# Correcting the dimension mismatch error in the compute_nearest_neighbors_NMF
  ↪function
def compute_nearest_neighbors_NMF(train, testImage, H):
```

```
    projected_train = [np.dot(np.array(i, dtype=float), H.T) for i in train]  # Corrected this line
    projected_test = np.dot(np.array(testImage, dtype=float), H.T)  # Corrected this line
    digit = 0
    imageIdx = 0
    dist = np.linalg.norm(projected_train[digit][imageIdx] - projected_test)

    for i in range(len(train)):
        for j in range(train[i].shape[0]):
            tempDist = np.linalg.norm(projected_train[i][j] - projected_test)
            if tempDist < dist:
                digit = i
                imageIdx = j
                dist = tempDist
    return digit, imageIdx

# Apply NMF with r=3
r = 3
nmf = NMF(n_components=r, init='random', random_state=0)
W = nmf.fit_transform(trainarr)
H = nmf.components_

# Find the nearest neighbors for all test images again
nmf_imgs = []
for i in range(len(testLabels)):
    nnDig, nnIdx = compute_nearest_neighbors_NMF(train, test[i], H)
    nmf_imgs.extend([test[i], train[nnDig][nnIdx,:]])

# Visualize the test images and their nearest neighbors
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(testLabels,testLabels)]
plot_image_grid(nmf_imgs, "NMF r=3 - Test Images and Nearest Neighbors",
                (28,28), len(testLabels), 2, True, row_titles=['Test','Nearest'], col_titles=col_titles)
```



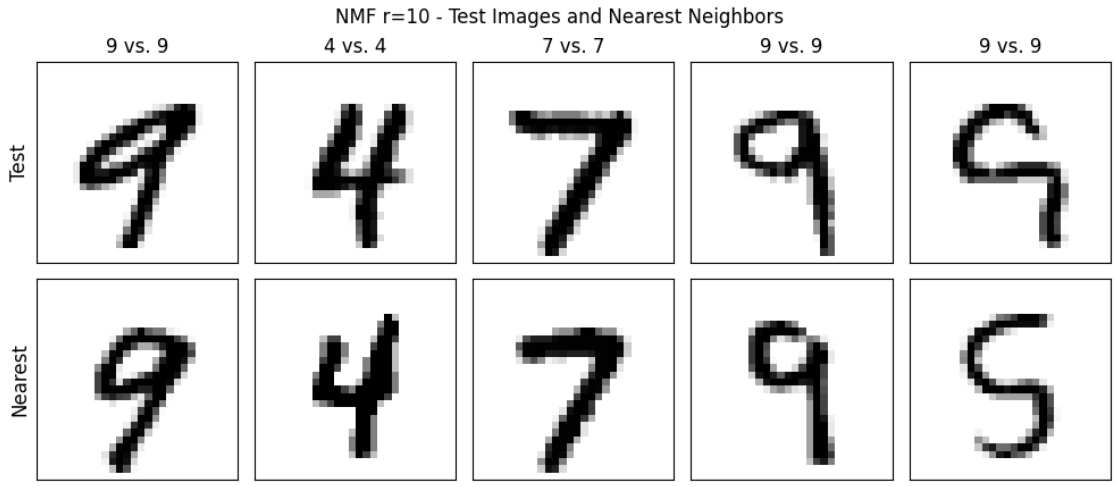
[22]:
```
# Correcting the dimension mismatch error in the compute_nearest_neighbors_NMF function
def compute_nearest_neighbors_NMF(train, testImage, H):
    projected_train = [np.dot(np.array(i, dtype=float), H.T) for i in train]  # Corrected this line
    projected_test = np.dot(np.array(testImage, dtype=float), H.T)  # Corrected this line
    digit = 0
    imageIdx = 0
    dist = np.linalg.norm(projected_train[digit][imageIdx] - projected_test)

    for i in range(len(train)):
        for j in range(train[i].shape[0]):
            tempDist = np.linalg.norm(projected_train[i][j] - projected_test)
            if tempDist < dist:
                digit = i
                imageIdx = j
                dist = tempDist
    return digit, imageIdx

# Apply NMF with r=3
r = 6
nmf = NMF(n_components=r, init='random', random_state=0)
W = nmf.fit_transform(trainarr)
H = nmf.components_

# Find the nearest neighbors for all test images again
nmf_imgs = []
for i in range(len(testLabels)):
    nnDig, nnIdx = compute_nearest_neighbors_NMF(train, test[i], H)
```

```
    nmf_imgs.extend([test[i], train[nnDig][nnIdx,:]])

# Visualize the test images and their nearest neighbors
col_titles = ['%d vs. %d'%(i,j) for i,j in zip(testLabels,testLabels)]
plot_image_grid(nmf_imgs, "NMF r=6 - Test Images and Nearest Neighbors",
                (28,28), len(testLabels), 2, True, row_titles=['Test',␣
 ↪'Nearest'], col_titles=col_titles)
```

[25]:
```
# Correcting the dimension mismatch error in the compute_nearest_neighbors_NMF␣
 ↪function
def compute_nearest_neighbors_NMF(train, testImage, H):
    projected_train = [np.dot(np.array(i, dtype=float), H.T) for i in train]  #␣
 ↪Corrected this line
    projected_test = np.dot(np.array(testImage, dtype=float), H.T)  # Corrected␣
 ↪this line
    digit = 0
    imageIdx = 0
    dist = np.linalg.norm(projected_train[digit][imageIdx] - projected_test)

    for i in range(len(train)):
        for j in range(train[i].shape[0]):
            tempDist = np.linalg.norm(projected_train[i][j] - projected_test)
            if tempDist < dist:
                digit = i
                imageIdx = j
                dist = tempDist
    return digit, imageIdx

# Apply NMF with r=3
r = 10
```

```
nmf = NMF(n_components=r, init='random', random_state=0)
W = nmf.fit_transform(trainarr)
H = nmf.components_

# Find the nearest neighbors for all test images again
nmf_imgs = []
for i in range(len(testLabels)):
    nnDig, nnIdx = compute_nearest_neighbors_NMF(train, test[i], H)
    nmf_imgs.extend([test[i], train[nnDig][nnIdx,:]])

# Visualize the test images and their nearest neighbors
plot_image_grid(nmf_imgs, "NMF r=10 - Test Images and Nearest Neighbors",
                (28,28), len(testLabels), 2, True, row_titles=['Test',␣
 ↪'Nearest'], col_titles=col_titles)
```

```
/home/karanvora/miniconda3/lib/python3.8/site-
packages/sklearn/decomposition/_nmf.py:1710: ConvergenceWarning: Maximum number
of iterations 200 reached. Increase it to improve convergence.
  warnings.warn(
```



For C, Ir is clear with PCA the matches are far more closer and accurate then NMF. NMF requires higher r number to get accurate enough Information.