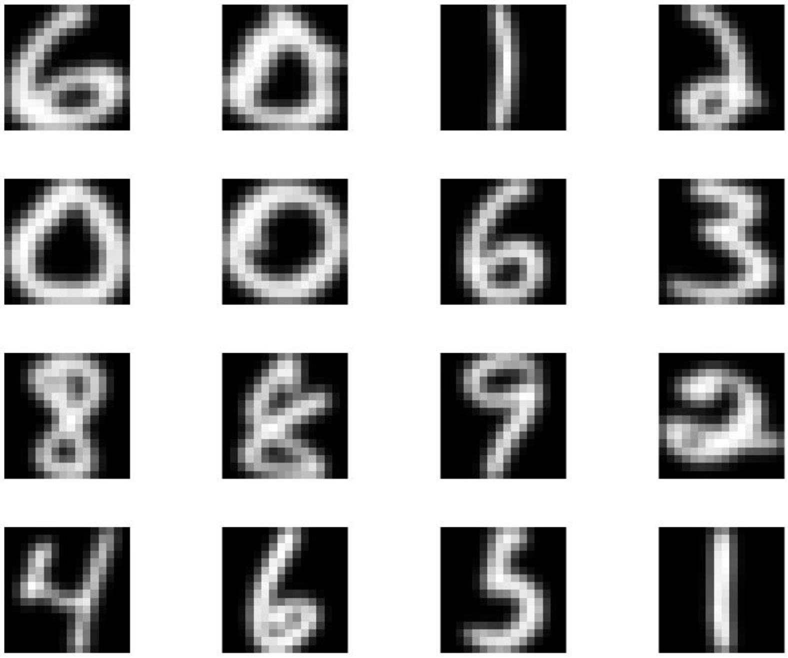
Quilvio Hernandez
MAT 167
Kathryn G. Link
March 17, 2021

Project 2

Problem 2A. I used MATLAB to classify handwritten digit recognition using both the nearest residual and SVD-based classification.
First, we examine the data.

(a) The handwritten digit database file "usps.mat" from Canvas Project 2 folder contains 4 arrays: `train_patterns`, `test_patterns` of size 256×4649, and `train_labels`, `test_labels` of size 10×4649. The `train_patterns` and `test_patterns` contain a raster scan of the 16×16 gray level pixel intensities, which have been normalized to range within [..1; 1]. The `train_labels` and `test_labels` variables contain the ground truth information of the digit images. That is, if the jth handwritten digit image in `train_patterns` truly represents digit i, then the $(i + 1; j)$th entry of `train_labels` is +1, and all the other entries of the jth column of `train_labels` are -1. We can examine the first 16 images in the `train_patterns` using `subplot(4,4,k)` and `imagesc` functions in MATLAB. The first 16 images and the code used is shown below.

```
%Part (a)
load usps.mat

figure(1)
title('First Sixteen Digits')
for j=1:16
    subplot(4,4,j)
    imagesc(reshape(train_patterns(:,j),[16 16])');
    axis image; axis off; colormap(gray);
end
```



(b) Now, we compute the mean digits in the `train_patterns`, and put them in a matrix called `train_aves` of size 256×10. We make use of `subplot(2,5,k)` and `imagesc` to display these 10 mean digit images.

```
%Part (b)
train_aves = zeros(256,10);
figure(2)
for k=1:10
    train_aves(:,k) = mean(train_patterns(:,train_labels(k,:) ==1),2);
    subplot(2,5,k);
    imagesc(reshape(train_aves(:,k),[16 16])');
    axis image; axis off; colormap(gray);
end
```

(c) Let's conduct the simplest classification experiments as follows:

(c.1) First, we prepare a matrix called `test_classif` of size 10×4649 and fill this matrix by computing the Euclidean distance (or its square) between each image in the test patterns and each mean digit image in `train_patterns`. The following line computes the squared Euclidean distances between all the test digit images and the kth mean digit of the training dataset by one line:

```
%Part (c)
%Part (c1)
test_classif = zeros(10,4649);
for k=1:10
    test_classif(k,:) = sum((test_patterns-repmat(train_aves(:,k), [1
end
```

(c.2) Then, we compute the classification results by finding the position index of the minimum of each column of `test_classif`. We put the results in a vector `test_classif_res` of size 1×4649.

```
%Part (c2)
test_classif_res = zeros(1, 4649);
for i=1:4649
    [M, ind] =  min(test_classif(:, i));
    test_classif_res(i) =  ind;
end
```

(c.3) Finally, we compute the confusion matrix `test_confusion` of size 10×10, print out this matrix, and submit your results. The `tmp` array contains the results of your classification of the test digits whose true digit is $k-1 (1 \leq k \leq 10)$. In other words, if your classification results were perfect, all the entries of `tmp` would be k. But in reality, this simplest classification algorithm makes mistakes, so `tmp` contains values other than k. You need to count how many entries have the value j in `tmp`, $j = 1 : 10$. That would give you the kth row of the test confusion matrix.

```
%Part (c3)
test_confusion = zeros(10, 10);
for k=1:10
    tmp = test_classif_res(test_labels(k,:)==1);
    n = size(tmp, 2);
    % use test labels and see where they are equal to 1
    % and those indices are going to define which test_classif_res
    % we are going to use
    for i=1:n
        for j=1:10
            if tmp(i) == j
                test_confusion(k, j) = test_confusion(k, j) + 1;
            end
        end
    end
end
disp(test_confusion);
```

	0	1	2	3	4	5	6	7	8	9
0	656	1	3	4	10	19	73	2	17	1
1	0	644	0	1	0	0	1	0	1	0
2	14	4	362	13	25	5	4	9	18	0
3	1	3	4	368	1	17	0	3	14	7
4	3	16	6	0	636	1	8	1	5	40
5	13	3	3	20	14	271	9	0	16	6
6	23	11	13	0	9	3	354	0	1	0
7	0	5	1	0	7	1	0	351	3	34
8	9	19	5	12	6	6	0	1	253	20
9	1	15	0	1	39	2	0	0	24	314

(d) Finally, let's conduct the SVD-based classification experiments.

(d.1) We can pool together all the images corresponding to the kth digit `train_patterns`, then compute the rank 17 SVD of that set of images (i.e., the first 17 singular values and vectors), and put the left singular vectors (or the matrix U) of kth digit into the array `train_u` of size $256 \times 17 \times 10$. For $k = 1 : 10$, use the following code:

```
%Part (d)
%Part (d.1)
train_u = zeros(256, 17,10);
for k=1:10
    [train_u(:,:,k),tmp,tmp2] = svds(train_patterns(:,train_labels(k,:
end
```

We do not need the singular values and right singular vectors in this experiment.

(d.2) Now, we compute the expansion coefficients of each test digit image with respect to the 17 singular vectors of each train digit image set. In other words, you need to compute 17×10 numbers for each test digit image. Put the results in the 3D array `test_svd17` of size $17 \times 4649 \times 10$. This can be done by

```
%Part (d.2)
test_svd17 = zeros(17, 4649,10);

for k=1:10
    test_svd17(:,:,k) = train_u(:,:,k)'*test_patterns;
end
```

(d.3) Next, compute the error between each original test digit image and its rank 17 approximation using the kth digit images in the training dataset. The idea of this classification is that if a test digit image should belong to class of kth digit if the corresponding rank 17 approximation is the best approximation (i.e., the smallest error) among 10 such approximations. (See my Lecture 21 for the details). Prepare a matrix `test_svd17res` of size 10×4649, and put those approximation errors into this matrix.

```
%Part (d.3)
```

```
test_svd17res = zeros(10, 4649);
rank17approx = zeros(256,4649);

for k=1:10
    rank17approx = train_u(:,:,k)*test_svd17(:,:,k);
    test_svd17res(k,:) = sum((test_patterns-rank17approx).^2);
end
```

(d.4) Finally, compute the confusion matrix using this SVD-based classification method by following the same strategy as Parts c.2 and c.3 above. Let's name this confusion matrix `test_svd17_confusion`. Print out this matrix, and submit your results.

```
%Part (d4)
test_svd17res_min = zeros(1, 4649);
for j=1:4649
    [z, ind] = min(test_svd17res(:, j));
    test_svd17res_min(j) = ind;
end

test_svd17_confusion = zeros(10, 10);
for k=1:10
    tmp = test_svd17res_min(test_labels(k,:)==1);
    n = size(tmp, 2);
    for i=1:n
        for j=1:10
            if tmp(i) == j
                test_svd17_confusion(k, j) = test_svd17_confusion(k, j
            end
        end
    end
end

disp(test_svd17_confusion);

save 'Project2_data.txt' test_confusion test_svd17_confusion
```

	0	1	2	3	4	5	6	7	8	9
0	772	2	1	3	1	1	2	1	3	0
1	0	646	0	0	0	0	0	0	0	1
2	3	6	431	6	0	3	1	2	2	0
3	1	1	4	401	0	7	0	0	4	0
4	2	8	1	0	424	1	1	5	0	1
5	2	0	0	5	2	335	7	1	1	2
6	6	4	0	0	2	3	399	0	0	0
7	0	2	0	0	2	0	0	387	0	11
8	2	9	1	5	1	1	0	0	309	3
9	0	5	0	1	0	0	0	4	1	388

It is clear from the confusion matrices that SVD-based classification outperformed residual classification. The number 1 was easy for both methods to classify, with SVD being near-perfect. 5, 0, and 9 seemed to be rather problematic for the residual classification, while SVD-based classification was able to handle these two numbers with far fewer errors. For accurate results, we should be using SVD even if it may be slighlty more comptuationally expensive.

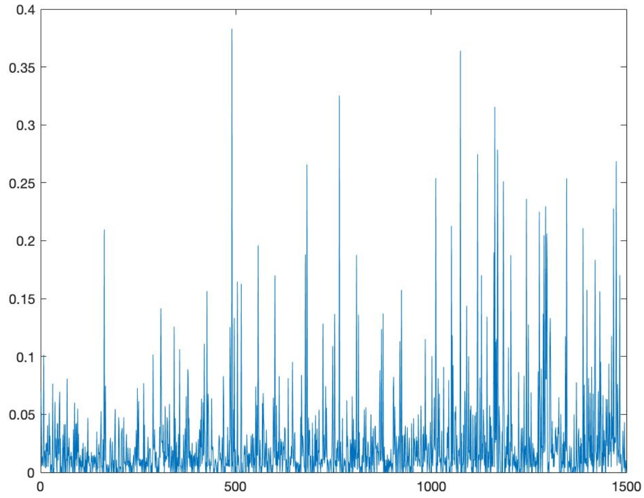
Problem 2B. I used MATLAB to conduct the following text mining experiments.

(a) Using the NIPS dataset file "nips.mat" from Canvas Project 2 folder. This file contains a term-document matrix A of size 12419×1500 as discussed in Lecture 22. Actual 12,419 terms are included in an array `terms` in that file. We want to retrieve the documents containing the following three terms: 'principal', 'component', 'analysis'. Construct the query vector q in MATLAB.

```
%Part (a)
load nips.mat

q = zeros(12419,1);

for i=1:12419
    if (strcmp('principal',terms(i)))
        q(i) = 1;
    elseif (strcmp('component', terms(i)))
        q(i) = 1;
    elseif (strcmp('analysis', terms(i)))
        q(i) = 1;
    end
end
```

(b) Now, we compute the cosine similarities between this query vector q and each document (i.e., each column vector) $a_j, j = 1 : 1500$. Then, plot this cosine similarities. Also, we compute the number of retrieved documents by varying the tolerance `tol = 0:05; 0:15; 0:25; 0:35`. The four numbers retrieved are reported below.

```
%Part (b)
figure(1)
cosine = zeros(1500,1);

for j=1:1500
    cosine(j) = (q'*A(:, j))/(norm(q) * norm(A(:, j)));
end

plot(cosine); axis([0 1500 0 0.4]);

tol = [0.05 0.15 0.25 0.35];
docsReturned = zeros(4,1);

for i=1:4
    for j=1:1500
        if (cosine(j) > tol(i))
            docsReturned(i) = docsReturned(i) + 1;
        end
    end
end

disp(docsReturned);
```



Documents received according to tolerance D(tol):

$$
\begin{aligned}
D(0.05) &= 179 \\
D(0.15) &= 37 \\
D(0.25) &= 11 \\
D(0.35) &= 2
\end{aligned}
$$

(c) We compute the first 100 terms of SVD of A using MATLAB's `svds` function by:

```
%Part (c)
[U100,S100,V100] = svds(A, 100);

A100 =  U100*S100*V100'; %formula
error100 = norm(A-A100, 'fro')/norm(A, 'fro');%your error formula
```

Then, we compute the relative Frobenius error between $A100$ and A, and report the results.

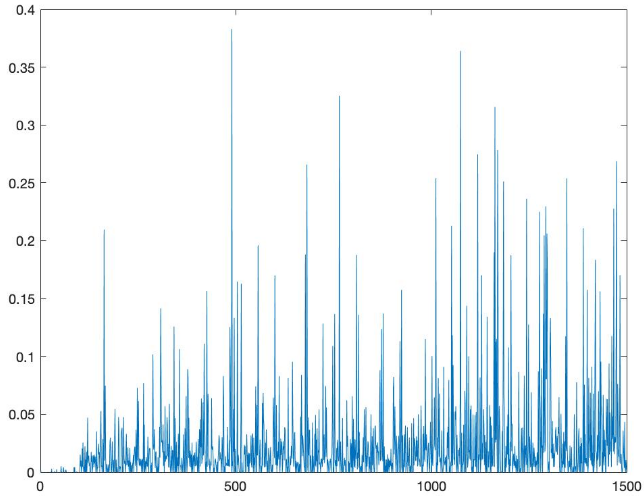
$$error100 = 0.6074$$

(d) Instead of A, let's use the rank 100 approximation of A. Without forming $A100$ explicitly, we can repeat Part (b) using the cosine similarity formula discussed in the class, i.e.,

$$cos\theta_j := \frac{q_k^T h_j}{||q||_2 ||h_j||_2}, q_k := U_k^T q :$$

```
%Part (d)
figure(2)
for j=1:100
    cosine(j) = ((U100'*q)'*A(1:100,j)) / (norm(U100'*q) * norm(A(:, j)));
end

plot(cosine); axis([0 1500 0 0.4]);
```

```
%print similiarityA100.jpg

docsReturnedA100 = zeros(4,1);

for i=1:4
    for j=1:1500
        if (cosine(j) > tol(i))
            docsReturnedA100(i) = docsReturnedA100(i) + 1;
        end
    end
end
disp(docsReturnedA100);
```



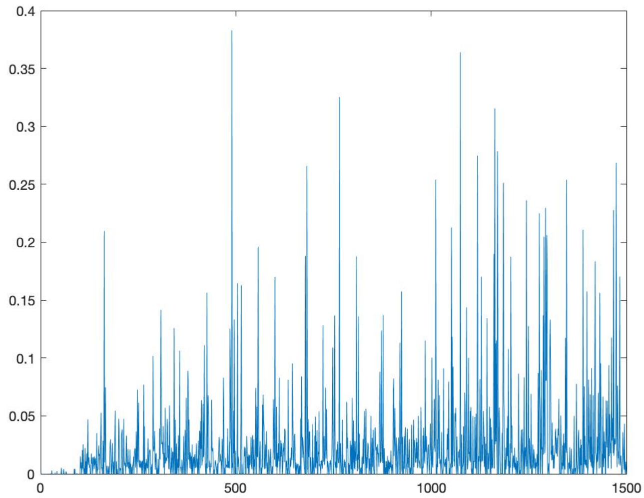
Documents received according to tolerance D(tol):

$$D(0.05) = 179$$
$$D(0.15) = 37$$
$$D(0.25) = 11$$
$$D(0.35) = 2$$

(e) We place $k = 100$ with $k = 50$ to analyze the difference.

```
%Part (e)
[U50,S50,V50] = svds(A, 50);

A50 =  U50*S50*V50'; %formula
error50 = norm(A-A50, 'fro')/norm(A, 'fro');%your error formula

%d
figure(3)
for j=1:50
    cosine(j) = ((U50'*q)'*A(1:50,j)) / (norm(U50'*q) * norm(A(:, j))); %
end

plot(cosine); axis([0 1500 0 0.4]);

docsReturnedA50 = zeros(4,1);
```

```

for i=1:4
    for j=1:1500
        if (cosine(j) > tol(i))
            docsReturnedA50(i) = docsReturnedA50(i) + 1;
        end
    end
end
disp(docsReturnedA50);
```



Error between A and $A50$:

$$error50 = 0.6857$$

Documents received according to tolerance D(tol):

$$D(0.05) = 169$$
$$D(0.15) = 37$$
$$D(0.25) = 11$$
$$D(0.35) = 2$$

We find the k = 50 and k = 100 perform similarly; however, k=50 is slightly more sensitive. The error for k = 50 is also about 13% higher than that of k = 100. It appears that k = 50 misses a few documents that k = 100 flags as similar enough at the .05 tolerance level. We conclude k = 100 is a better choice for k.