MAT 167: Applied Linear Algebra
Lecture 21: Classification of Handwritten Digits

Naoki Saito

Department of Mathematics
University of California, Davis

May 17, 2017

Outline

1. The USPS Handwritten Digits Dataset
2. Simple Classification Algorithms
3. Classification Using SVD Bases

Outline

1. The USPS Handwritten Digits Dataset
2. Simple Classification Algorithms
3. Classification Using SVD Bases

The USPS Handwritten Digit Dataset

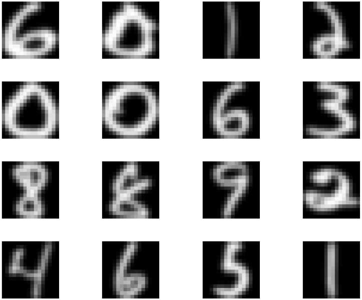
- The USPS digits data were gathered at the Center of Excellence in Document Analysis and Recognition (CEDAR) at SUNY Buffalo, as part of a project sponsored by the US Postal Service.
- The dataset is described in the following paper: J. J. Hull: "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

The USPS Handwritten Digit Dataset

- The USPS digits data were gathered at the Center of Excellence in Document Analysis and Recognition (CEDAR) at SUNY Buffalo, as part of a project sponsored by the US Postal Service.
- The dataset is described in the following paper: J. J. Hull: "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

The USPS Handwritten Digit Dataset

- The USPS digits data were gathered at the Center of Excellence in Document Analysis and Recognition (CEDAR) at SUNY Buffalo, as part of a project sponsored by the US Postal Service.
- The dataset is described in the following paper: J. J. Hull: "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.



- This dataset is available from `http://www.gaussianprocess.org/gpml/data/` .
- There are totally 9298 handwritten single digits between 0 and 9, each of which consists of 16×16 pixel image.
- Half of 9298 digits are designated as *training* and the other half are as *test*: use 4649 digits for constructing a classification algorithm, and use the other 4649 digits to test the performance of that algorithm.
- Pixel values are normalized to be in the range of [-1, 1].
- Each digit image is represented as a 1D vector of length 256 in the MATLAB file, so there are two matrices in the file called `train_patterns` and `test_patterns` each of which is of size 256×4649.
- Note that if you want to see a digit of a particular column of these matrices using MATLAB like in the figure of the previous page, you need to `reshape` that column (a vector of length 256) to a small matrix of size 16×16 and then *transpose* it before rendering it using `imagesc` function.

- This dataset is available from
  http://www.gaussianprocess.org/gpml/data/ .
- There are totally 9298 handwritten single digits between 0 and 9, each of which consists of 16×16 pixel image.
- Half of 9298 digits are designated as *training* and the other half are as *test*: use 4649 digits for constructing a classification algorithm, and use the other 4649 digits to test the performance of that algorithm.
- Pixel values are normalized to be in the range of [-1, 1].
- Each digit image is represented as a 1D vector of length 256 in the MATLAB file, so there are two matrices in the file called `train_patterns` and `test_patterns` each of which is of size 256×4649.
- Note that if you want to see a digit of a particular column of these matrices using MATLAB like in the figure of the previous page, you need to `reshape` that column (a vector of length 256) to a small matrix of size 16×16 and then *transpose* it before rendering it using `imagesc` function.

- This dataset is available from `http://www.gaussianprocess.org/gpml/data/` .
- There are totally 9298 handwritten single digits between 0 and 9, each of which consists of 16×16 pixel image.
- Half of 9298 digits are designated as *training* and the other half are as *test*: use 4649 digits for constructing a classification algorithm, and use the other 4649 digits to test the performance of that algorithm.
- Pixel values are normalized to be in the range of [-1, 1].
- Each digit image is represented as a 1D vector of length 256 in the MATLAB file, so there are two matrices in the file called `train_patterns` and `test_patterns` each of which is of size 256×4649.
- Note that if you want to see a digit of a particular column of these matrices using MATLAB like in the figure of the previous page, you need to `reshape` that column (a vector of length 256) to a small matrix of size 16×16 and then *transpose* it before rendering it using `imagesc` function.

- This dataset is available from `http://www.gaussianprocess.org/gpml/data/` .
- There are totally 9298 handwritten single digits between 0 and 9, each of which consists of 16×16 pixel image.
- Half of 9298 digits are designated as *training* and the other half are as *test*: use 4649 digits for constructing a classification algorithm, and use the other 4649 digits to test the performance of that algorithm.
- Pixel values are normalized to be in the range of [-1, 1].
- Each digit image is represented as a 1D vector of length 256 in the MATLAB file, so there are two matrices in the file called `train_patterns` and `test_patterns` each of which is of size 256×4649.
- Note that if you want to see a digit of a particular column of these matrices using MATLAB like in the figure of the previous page, you need to `reshape` that column (a vector of length 256) to a small matrix of size 16×16 and then *transpose* it before rendering it using `imagesc` function.

- This dataset is available from
  http://www.gaussianprocess.org/gpml/data/ .
- There are totally 9298 handwritten single digits between 0 and 9, each of which consists of 16×16 pixel image.
- Half of 9298 digits are designated as *training* and the other half are as *test*: use 4649 digits for constructing a classification algorithm, and use the other 4649 digits to test the performance of that algorithm.
- Pixel values are normalized to be in the range of [-1, 1].
- Each digit image is represented as a 1D vector of length 256 in the MATLAB file, so there are two matrices in the file called `train_patterns` and `test_patterns` each of which is of size 256×4649.
- Note that if you want to see a digit of a particular column of these matrices using MATLAB like in the figure of the previous page, you need to `reshape` that column (a vector of length 256) to a small matrix of size 16×16 and then *transpose* it before rendering it using `imagesc` function.

- This dataset is available from `http://www.gaussianprocess.org/gpml/data/` .
- There are totally 9298 handwritten single digits between 0 and 9, each of which consists of 16×16 pixel image.
- Half of 9298 digits are designated as *training* and the other half are as *test*: use 4649 digits for constructing a classification algorithm, and use the other 4649 digits to test the performance of that algorithm.
- Pixel values are normalized to be in the range of [-1, 1].
- Each digit image is represented as a 1D vector of length 256 in the MATLAB file, so there are two matrices in the file called `train_patterns` and `test_patterns` each of which is of size 256×4649.
- Note that if you want to see a digit of a particular column of these matrices using MATLAB like in the figure of the previous page, you need to `reshape` that column (a vector of length 256) to a small matrix of size 16×16 and then *transpose* it before rendering it using `imagesc` function.

- The labels are also known for both training and test sets. They are stored in `train_labels` and `test_labels` each of which is of size 10×4649.
- Let $A \in \mathbb{R}^{10 \times 4649}$ be one of these label matrices. Then $A(:,j)$, i.e., jth column of A describes the label of the jth digit in the following way. If that digit represents digit i $(0 \le i \le 9)$, then $A(i+1,j) = +1$ and $A(l,j) = -1$ for $l \ne i+1$.

- The labels are also known for both training and test sets. They are stored in `train_labels` and `test_labels` each of which is of size 10×4649.
- Let $A \in \mathbb{R}^{10 \times 4649}$ be one of these label matrices. Then $A(:,j)$, i.e., jth column of A describes the label of the jth digit in the following way. If that digit represents digit i $(0 \le i \le 9)$, then $A(i+1,j) = +1$ and $A(l,j) = -1$ for $l \ne i+1$.

Notation

- Let $X = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix whose columns represent the digits in the *training* dataset with $d = 256$, $n = 4649$.
- Let $Y = [\boldsymbol{y}_1 \cdots \boldsymbol{y}_m] \in \mathbb{R}^{d \times m}$ be the data matrix whose columns represent the digits in the *test* dataset with $d = 256$, $m = 4649$.

Notation

- Let $X = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ be the data matrix whose columns represent the digits in the *training* dataset with $d = 256$, $n = 4649$.
- Let $Y = [\boldsymbol{y}_1 \cdots \boldsymbol{y}_m] \in \mathbb{R}^{d \times m}$ be the data matrix whose columns represent the digits in the *test* dataset with $d = 256$, $m = 4649$.

Outline

1 The USPS Handwritten Digits Dataset

2 Simple Classification Algorithms

3 Classification Using SVD Bases

The Simplest Classification Algorithm

The simplest classification algorithm is perhaps the following one:

1. Compute the mean (average) digits $\boldsymbol{m}_i$, $i = 0, \ldots, 9$ using the training digits.
2. For each digit $\boldsymbol{y}_j$, classify it as digit k if $\boldsymbol{m}_k$ is the closest mean.

The Simplest Classification Algorithm

The simplest classification algorithm is perhaps the following one:

1. Compute the mean (average) digits $\boldsymbol{m}_i$, $i = 0, \ldots, 9$ using the training digits.
2. For each digit $\boldsymbol{y}_j$, classify it as digit k if $\boldsymbol{m}_k$ is the closest mean.

The Simplest Classification Algorithm

The simplest classification algorithm is perhaps the following one:

1. Compute the mean (average) digits $\boldsymbol{m}_i$, $i = 0, \ldots, 9$ using the training digits.
2. For each digit $\boldsymbol{y}_j$, classify it as digit k if $\boldsymbol{m}_k$ is the closest mean.

The Simplest Classification Algorithm

The simplest classification algorithm is perhaps the following one:

1. Compute the mean (average) digits $\boldsymbol{m}_i$, $i = 0, \ldots, 9$ using the training digits.
2. For each digit $\boldsymbol{y}_j$, classify it as digit k if $\boldsymbol{m}_k$ is the closest mean.

Figure: The mean digits (centroids) in the training dataset.

- Note that there are many choices as the distance between each test and training digits, e.g., ℓ^p-norm with $1 \le p \le \infty$, and cosine between them, etc., we decided to use the simplest one, i.e., the Euclidean, i.e., ℓ^2 distance: $d(\boldsymbol{x}_i, \boldsymbol{y}_j) = \|\boldsymbol{x}_i - \boldsymbol{y}_j\|_2$.
- The over all classification rate was 84.66%. More precisely:

- Note that there are many choices as the distance between each test and training digits, e.g., ℓ^p-norm with $1 \le p \le \infty$, and cosine between them, etc., we decided to use the simplest one, i.e., the Euclidean, i.e., ℓ^2 distance: $d(\boldsymbol{x}_i, \boldsymbol{y}_j) = \|\boldsymbol{x}_i - \boldsymbol{y}_j\|_2$.
- The over all classification rate was 84.66%. More precisely:

	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	363	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	24	3	314

Figure: The worst test digits (the farthest from the means)

The Next Simplest Classification Algorithm

The next simplest one should be the so-called *k-nearest neighbor* (k-NN) classification algorithm as follows:

1. Select k from small odd integers (i.e., 1, 3, 5, etc.)
2. For each test digit $\boldsymbol{y}_j$, do:
   - Compute the distances from $\boldsymbol{y}_j$ to all the training digits $\{\boldsymbol{x}_i\}_{i=1,\ldots,n}$.
   - Choose the k nearest training digits from $\boldsymbol{y}_j$.
   - Check the labels of these k neighbors, and take a majority vote, which is assigned as a class label of $\boldsymbol{y}_j$.

The Next Simplest Classification Algorithm

The next simplest one should be the so-called *k-nearest neighbor* (k-NN) classification algorithm as follows:

1. Select k from small odd integers (i.e., 1, 3, 5, etc.)
2. For each test digit $\boldsymbol{y}_j$, do:
   - Compute the distances from $\boldsymbol{y}_j$ to all the training digits $\{\boldsymbol{x}_i\}_{i=1,\dots,n}$.
   - Choose the k nearest training digits from $\boldsymbol{y}_j$.
   - Check the labels of these k neighbors, and take a majority vote, which is assigned as a class label of $\boldsymbol{y}_j$.

The Next Simplest Classification Algorithm

The next simplest one should be the so-called *k-nearest neighbor* (k-NN) classification algorithm as follows:

1. Select k from small odd integers (i.e., 1, 3, 5, etc.)
2. For each test digit $\boldsymbol{y}_j$, do:
   - Compute the distances from $\boldsymbol{y}_j$ to all the training digits $\{\boldsymbol{x}_i\}_{i=1,\dots,n}$.
   - Choose the k nearest training digits from $\boldsymbol{y}_j$.
   - Check the labels of these k neighbors, and take a majority vote, which is assigned as a class label of $\boldsymbol{y}_j$.

The Next Simplest Classification Algorithm

The next simplest one should be the so-called *k-nearest neighbor* (k-NN) classification algorithm as follows:

1. Select k from small odd integers (i.e., 1, 3, 5, etc.)
2. For each test digit $\boldsymbol{y}_j$, do:
   - Compute the distances from $\boldsymbol{y}_j$ to all the training digits $\{\boldsymbol{x}_i\}_{i=1,\dots,n}$.
   - Choose the k nearest training digits from $\boldsymbol{y}_j$.
   - Check the labels of these k neighbors, and take a majority vote, which is assigned as a class label of $\boldsymbol{y}_j$.

The Next Simplest Classification Algorithm

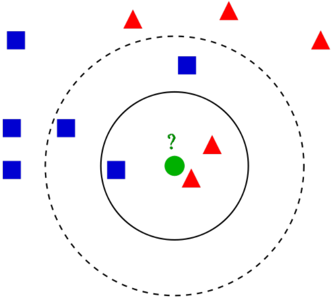
The next simplest one should be the so-called *k-nearest neighbor* (k-NN) classification algorithm as follows:

1. Select k from small odd integers (i.e., 1, 3, 5, etc.)
2. For each test digit $\boldsymbol{y}_j$, do:
    - Compute the distances from $\boldsymbol{y}_j$ to all the training digits $\{\boldsymbol{x}_i\}_{i=1,\ldots,n}$.
    - Choose the k nearest training digits from $\boldsymbol{y}_j$.
    - Check the labels of these k neighbors, and take a majority vote, which is assigned as a class label of $\boldsymbol{y}_j$.

The Next Simplest Classification Algorithm

The next simplest one should be the so-called *k-nearest neighbor* (k-NN) classification algorithm as follows:

1. Select k from small odd integers (i.e., 1, 3, 5, etc.)
2. For each test digit $\boldsymbol{y}_j$, do:
   - Compute the distances from $\boldsymbol{y}_j$ to all the training digits $\{\boldsymbol{x}_i\}_{i=1,\dots,n}$.
   - Choose the k nearest training digits from $\boldsymbol{y}_j$.
   - Check the labels of these k neighbors, and take a majority vote, which is assigned as a class label of $\boldsymbol{y}_j$.



k-NN Classification Results

- I tested with $k = 1, 3, 5$ using the MATLAB function `knnclassify` (in *Bioinformatics Toolbox*).
- The classification rates were considerably better than the previous simplest algorithm, i.e., 96.99%, 97.07%, 96.62%, respectively. More precisely, for $k = 3$:

k-NN Classification Results

- I tested with $k = 1, 3, 5$ using the MATLAB function `knnclassify` (in *Bioinformatics Toolbox*).
- The classification rates were considerably better than the previous simplest algorithm, i.e., 96.99%, 97.07%, 96.62%, respectively. More precisely, for $k = 3$:

k-NN Classification Results

- I tested with $k = 1, 3, 5$ using the MATLAB function `knnclassify` (in *Bioinformatics Toolbox*).
- The classification rates were considerably better than the previous simplest algorithm, i.e., 96.99%, 97.07%, 96.62%, respectively. More precisely, for $k = 3$:

	0	1	2	3	4	5	6	7	8	9
0	778	0	4	2	0	1	0	0	0	1
1	0	643	0	0	1	0	1	0	1	1
2	3	1	435	5	2	1	0	6	1	0
3	1	0	1	402	0	5	0	1	5	3
4	0	2	1	0	420	0	3	1	0	16
5	4	0	1	10	0	332	3	1	2	2
6	3	1	1	0	2	2	405	0	0	0
7	0	0	0	0	1	0	0	394	1	6
8	1	1	1	3	2	3	1	2	314	3
9	0	0	0	1	2	0	0	5	1	390

Outline

1. The USPS Handwritten Digits Dataset
2. Simple Classification Algorithms
3. Classification Using SVD Bases

Classification Using SVD Bases

- We use the first k left singular vectors $\{\boldsymbol{u}_1 \dots, \boldsymbol{u}_k\}$ of the SVD of the training digits. For each digit, we pool the training images corresponding to that digit, and compute the SVD. In other words, for each digit class, we compute the SVD.
- Since we only need the first k terms of the SVD, we only need to use svds function in MATLAB, which can specify k as an input argument.
- Let $X^{(j)}$ be a matrix of size $d \times n_j$ whose columns are training images corresponding to digit j (hence n_j is the number of training images corresponding to digit j), $j = 0, 1, \dots, 9$.
- Let the first k terms of SVD of $X^{(j)}$ be $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\top}$ where $U_k^{(j)} \in \mathbb{R}^{d \times k}$, $\Sigma_k^{(j)} \in \mathbb{R}^{k \times k}$, and $V_k^{(j)} \in \mathbb{R}^{n_j \times k}$.
- $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\top}$ is the *best* rank k approximation of $X^{(j)}$ in the sense of least squares.

Classification Using SVD Bases

- We use the first k left singular vectors $\{\boldsymbol{u}_1 \ldots, \boldsymbol{u}_k\}$ of the SVD of the training digits. For each digit, we pool the training images corresponding to that digit, and compute the SVD. In other words, for each digit class, we compute the SVD.
- Since we only need the first k terms of the SVD, we only need to use `svds` function in MATLAB, which can specify k as an input argument.
- Let $X^{(j)}$ be a matrix of size $d \times n_j$ whose columns are training images corresponding to digit j (hence n_j is the number of training images corresponding to digit j), $j = 0, 1, \ldots, 9$.
- Let the first k terms of SVD of $X^{(j)}$ be $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\top}$ where $U_k^{(j)} \in \mathbb{R}^{d \times k}$, $\Sigma_k^{(j)} \in \mathbb{R}^{k \times k}$, and $V_k^{(j)} \in \mathbb{R}^{n_j \times k}$.
- $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\top}$ is the *best* rank k approximation of $X^{(j)}$ in the sense of least squares.

Classification Using SVD Bases

- We use the first k left singular vectors $\{\boldsymbol{u}_1 \dots, \boldsymbol{u}_k\}$ of the SVD of the training digits. For each digit, we pool the training images corresponding to that digit, and compute the SVD. In other words, for each digit class, we compute the SVD.
- Since we only need the first k terms of the SVD, we only need to use `svds` function in MATLAB, which can specify k as an input argument.
- Let $X^{(j)}$ be a matrix of size $d \times n_j$ whose columns are training images corresponding to digit j (hence n_j is the number of training images corresponding to digit j), $j = 0, 1, \dots, 9$.
- Let the first k terms of SVD of $X^{(j)}$ be $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\top}$ where $U_k^{(j)} \in \mathbb{R}^{d \times k}$, $\Sigma_k^{(j)} \in \mathbb{R}^{k \times k}$, and $V_k^{(j)} \in \mathbb{R}^{n_j \times k}$.
- $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\top}$ is the *best* rank k approximation of $X^{(j)}$ in the sense of least squares.

Classification Using SVD Bases

- We use the first k left singular vectors $\{\boldsymbol{u}_1 \dots, \boldsymbol{u}_k\}$ of the SVD of the training digits. For each digit, we pool the training images corresponding to that digit, and compute the SVD. In other words, for each digit class, we compute the SVD.
- Since we only need the first k terms of the SVD, we only need to use `svds` function in MATLAB, which can specify k as an input argument.
- Let $X^{(j)}$ be a matrix of size $d \times n_j$ whose columns are training images corresponding to digit j (hence n_j is the number of training images corresponding to digit j), $j = 0, 1, \dots, 9$.
- Let the first k terms of SVD of $X^{(j)}$ be $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$ where $U_k^{(j)} \in \mathbb{R}^{d \times k}$, $\Sigma_k^{(j)} \in \mathbb{R}^{k \times k}$, and $V_k^{(j)} \in \mathbb{R}^{n_j \times k}$.
- $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$ is the *best* rank k approximation of $X^{(j)}$ in the sense of least squares.

Classification Using SVD Bases

- We use the first k left singular vectors $\{\boldsymbol{u}_1 \ldots, \boldsymbol{u}_k\}$ of the SVD of the training digits. For each digit, we pool the training images corresponding to that digit, and compute the SVD. In other words, for each digit class, we compute the SVD.
- Since we only need the first k terms of the SVD, we only need to use `svds` function in MATLAB, which can specify k as an input argument.
- Let $X^{(j)}$ be a matrix of size $d \times n_j$ whose columns are training images corresponding to digit j (hence n_j is the number of training images corresponding to digit j), $j = 0, 1, \ldots, 9$.
- Let the first k terms of SVD of $X^{(j)}$ be $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$ where $U_k^{(j)} \in \mathbb{R}^{d \times k}$, $\Sigma_k^{(j)} \in \mathbb{R}^{k \times k}$, and $V_k^{(j)} \in \mathbb{R}^{n_j \times k}$.
- $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$ is the *best* rank k approximation of $X^{(j)}$ in the sense of least squares.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\mathrm{range}\left(X^{(j)}\right) \approx \mathrm{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\mathrm{range}\left(X^{(j)}\right) = \mathrm{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\top}$, i.e., $U_k^{(j)\top}\boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\top}\boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)}\Sigma_k^{(j)}V_k^{(j)\top}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit y_i has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that y_i belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\text{range}\left(X^{(j)}\right) \approx \text{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\text{range}\left(X^{(j)}\right) = \text{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\top}$, i.e., $U_k^{(j)\top}\boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\top}\boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)}\Sigma_k^{(j)}V_k^{(j)\top}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit $\boldsymbol{y}_i$ has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that $\boldsymbol{y}_i$ belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\operatorname{range}\left(X^{(j)}\right) \approx \operatorname{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\operatorname{range}\left(X^{(j)}\right) = \operatorname{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\mathsf{T}}$, i.e., $U_k^{(j)\mathsf{T}}\boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\mathsf{T}}\boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)}\Sigma_k^{(j)}V_k^{(j)\mathsf{T}}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit y_i has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that y_i belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\operatorname{range}\left(X^{(j)}\right) \approx \operatorname{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\operatorname{range}\left(X^{(j)}\right) = \operatorname{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\top}$, i.e., $U_k^{(j)\top}\boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\top}\boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)}\Sigma_k^{(j)}V_k^{(j)\top}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit y_l has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that y_l belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\operatorname{range}\left(X^{(j)}\right) \approx \operatorname{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\operatorname{range}\left(X^{(j)}\right) = \operatorname{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\mathsf{T}}$, i.e., $U_k^{(j)\mathsf{T}} \boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{x}_i \right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit $\boldsymbol{y}_l$ has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that $\boldsymbol{y}_l$ belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\text{range}\left(X^{(j)}\right) \approx \text{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\text{range}\left(X^{(j)}\right) = \text{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\mathsf{T}}$, i.e., $U_k^{(j)\mathsf{T}}\boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\mathsf{T}}\boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)}\Sigma_k^{(j)}V_k^{(j)\mathsf{T}}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit $\boldsymbol{y}_l$ has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that $\boldsymbol{y}_l$ belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\text{range}\left(X^{(j)}\right) \approx \text{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\text{range}\left(X^{(j)}\right) = \text{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\mathsf{T}}$, i.e., $U_k^{(j)\mathsf{T}}\boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\mathsf{T}}\boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)}\Sigma_k^{(j)}V_k^{(j)\mathsf{T}}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit $\boldsymbol{y}_l$ has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that $\boldsymbol{y}_l$ belongs to digit j^*.

- The reasons why we use the first k left singular vectors are:
  - If k is *appropriately* chosen, then $\text{range}\left(X^{(j)}\right) \approx \text{range}\left(U_k^{(j)}\right)$. In fact, if $k = \min(d, n_j)$, then $\text{range}\left(X^{(j)}\right) = \text{range}\left(U_k^{(j)}\right)$.
  - The columns of $U_k^{(j)}$ are a part of ONB for $X^{(j)}$, which allows us to compute the k expansion coefficients of each training image $\boldsymbol{x}_i$ by simply multiplying (from left) $U_k^{(j)\mathsf{T}}$, i.e., $U_k^{(j)\mathsf{T}} \boldsymbol{x}_i$ gives you such expansion coefficients.
  - $U_k^{(j)}\left(U_k^{(j)\mathsf{T}} \boldsymbol{x}_i\right)$ is the best k-term approximation in the least squares sense *if $\boldsymbol{x}_i$ belongs to digit j class*.
- Also, the SVD-based classification algorithm in the next page assume the following (if not, it won't work well):
  - Each $X^{(j)}$ is well characterized and approximated by $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$ for the same value of k for all 10 digits.
  - If you approximate $X^{(m)}$ using $U_k^{(j)}$ with $m \neq j$, the error will be large.
  - If an unlabeled test digit $\boldsymbol{y}_l$ has the least k-term approximation error when using $U_k^{(j^*)}$, then it is likely that $\boldsymbol{y}_l$ belongs to digit j^*.

An SVD Basis Classification Algorithm

- Training: Compute the best rank k approximation of $X^{(j)}$, i.e., $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$.
- Classification: For a given test digit $\boldsymbol{y}_l$, compute the 2-norm of residual errors, $E_j(\boldsymbol{y}_l) := \left\| \boldsymbol{y}_l - U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{y}_l \right) \right\|_2$, $j = 0, 1, \ldots, 9$; If one of them, say, $E_{j^*}(\boldsymbol{y}_l)$ is significantly smaller than all the others, then classify $\boldsymbol{y}_l$ as digit j^*; otherwise give up.

Note: Mathematically, $\boldsymbol{y}_l - U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{y}_l \right) = \left(I_d - U_k^{(j)} U_k^{(j)\mathsf{T}} \right) \boldsymbol{y}_l$, i.e., this is an *orthogonal complement* to the projection of $\boldsymbol{y}_l$ onto range$\left(U_k^{(j)} \right)$. However, *computationally*, you should compute $U_k^{(j)\mathsf{T}} \boldsymbol{y}_l$ first, then multiply $U_k^{(j)}$. As I mentioned previously, if you first try to compute $U_k^{(j)} U_k^{(j)\mathsf{T}}$, it would take a long time or even would be impossible to computing it if d is large. This digit recognition problem has $d = 256$; so you can do either way.

An SVD Basis Classification Algorithm

- Training: Compute the best rank k approximation of $X^{(j)}$, i.e., $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$.
- Classification: For a given test digit $\boldsymbol{y}_l$, compute the 2-norm of residual errors, $E_j(\boldsymbol{y}_l) := \left\| \boldsymbol{y}_l - U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{y}_l \right) \right\|_2$, $j = 0, 1, \ldots, 9$; If one of them, say, $E_{j^*}(\boldsymbol{y}_l)$ is significantly smaller than all the others, then classify $\boldsymbol{y}_l$ as digit j^*; otherwise give up.

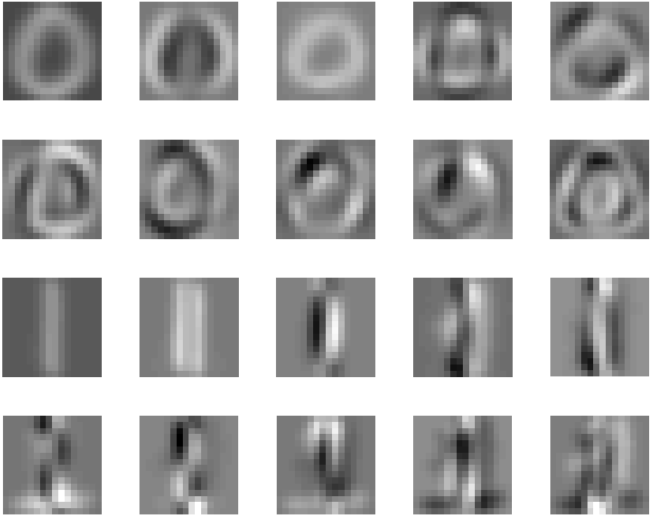
Note: Mathematically, $\boldsymbol{y}_l - U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{y}_l \right) = \left(I_d - U_k^{(j)} U_k^{(j)\mathsf{T}} \right) \boldsymbol{y}_l$, i.e., this is an *orthogonal complement* to the projection of $\boldsymbol{y}_l$ onto range$\left(U_k^{(j)} \right)$. However, *computationally*, you should compute $U_k^{(j)\mathsf{T}} \boldsymbol{y}_l$ first, then multiply $U_k^{(j)}$. As I mentioned previously, if you first try to compute $U_k^{(j)} U_k^{(j)\mathsf{T}}$, it would take a long time or even would be impossible to computing it if d is large. This digit recognition problem has $d = 256$; so you can do either way.

An SVD Basis Classification Algorithm

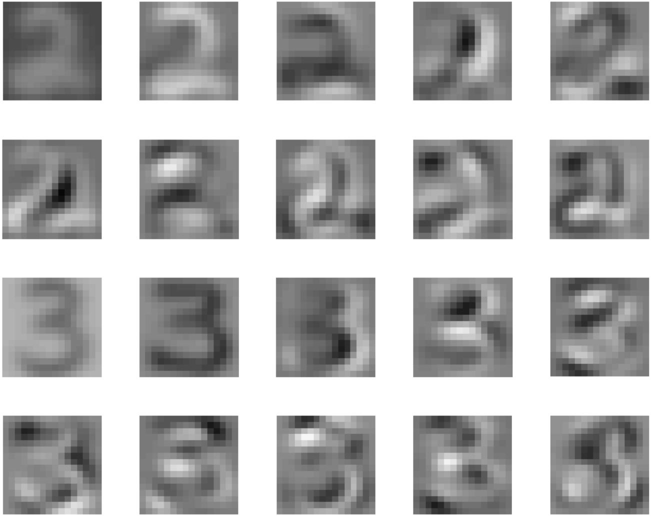
- Training: Compute the best rank k approximation of $X^{(j)}$, i.e., $U_k^{(j)} \Sigma_k^{(j)} V_k^{(j)\mathsf{T}}$.
- Classification: For a given test digit $\boldsymbol{y}_l$, compute the 2-norm of residual errors, $E_j(\boldsymbol{y}_l) := \left\| \boldsymbol{y}_l - U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{y}_l \right) \right\|_2$, $j = 0, 1, \ldots, 9$; If one of them, say, $E_{j^*}(\boldsymbol{y}_l)$ is significantly smaller than all the others, then classify $\boldsymbol{y}_l$ as digit j^*; otherwise give up.

<u>Note:</u> Mathematically, $\boldsymbol{y}_l - U_k^{(j)} \left(U_k^{(j)\mathsf{T}} \boldsymbol{y}_l \right) = \left(I_d - U_k^{(j)} U_k^{(j)\mathsf{T}} \right) \boldsymbol{y}_l$, i.e., this is an *orthogonal complement* to the projection of $\boldsymbol{y}_l$ onto $\mathrm{range}\left(U_k^{(j)} \right)$. However, *computationally*, you should compute $U_k^{(j)\mathsf{T}} \boldsymbol{y}_l$ first, then multiply $U_k^{(j)}$. As I mentioned previously, if you first try to compute $U_k^{(j)} U_k^{(j)\mathsf{T}}$, it would take a long time or even would be impossible to computing it if d is large. This digit recognition problem has $d = 256$; so you can do either way.

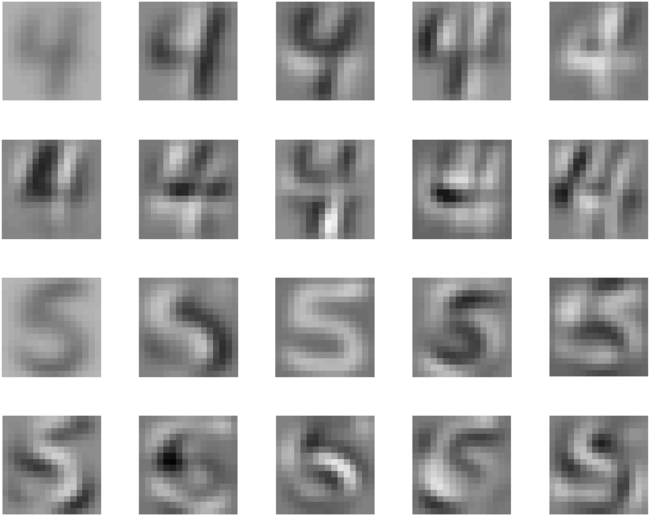
U_{10} of Digits '0' and '1'



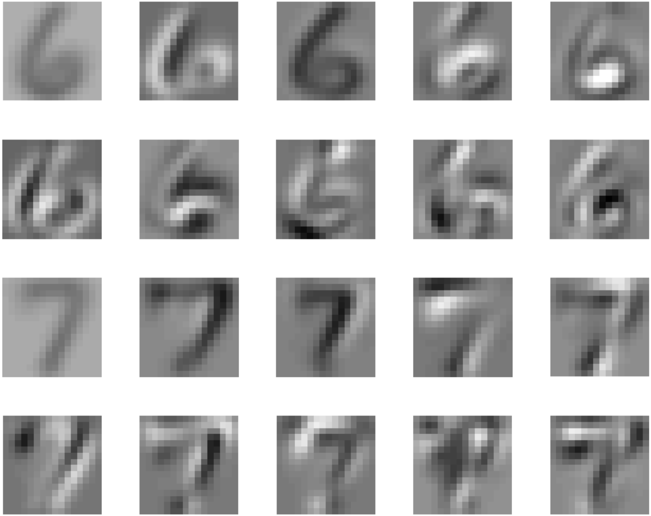
U_{10} of Digits '2' and '3'



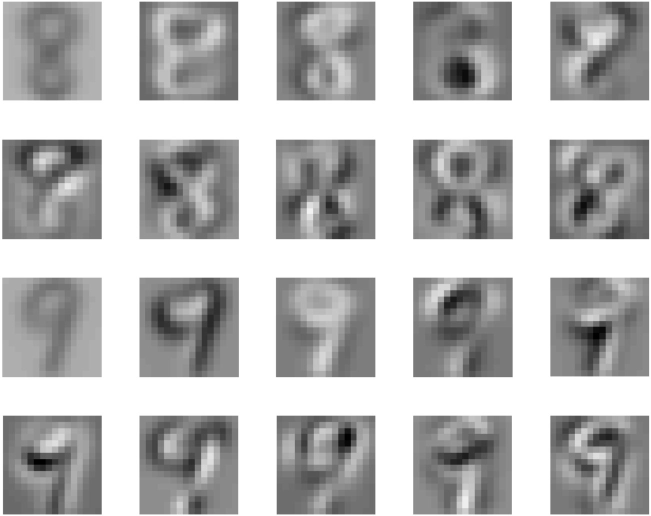
U_{10} of Digits '4' and '5'



U_{10} of Digits '6' and '7'



U_{10} of Digits '8' and '9'



SVD Classification Results with $k = 1:20$

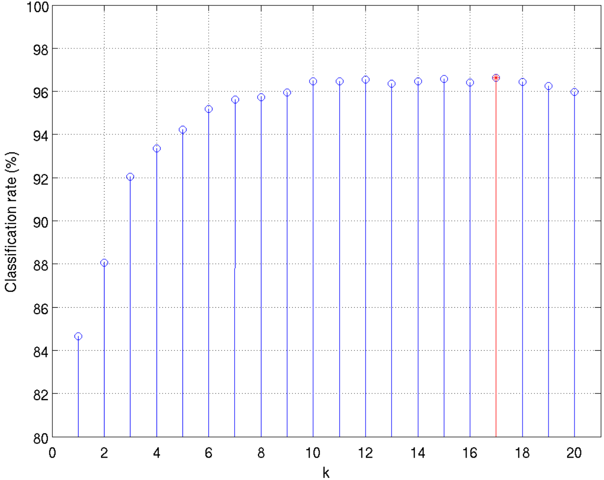


Figure: $k = 17$ gave the best result: 96.62%

Confusion Matrix for $k = 17$

	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