Classifying Digits by machine learning

Yukai Yan

March 10th, 2021

Abstract

In this paper, we will use the MNIST data set (both training and test sets and labels) to train our machine learning model and verify its accuracy by comparing its predicted result with the test data. We will use singular value decomposition, principal component analysis, and linear discriminant analysis to interpret data and classify digits. Later in the paper, we will compare our model with decision tree classifiers and support vector machines.

1 Introduction and Overview

1.1 Classify digits

The Modified National Institute of Standards and Technology (MNIST) database contains a large dataset that is commonly used for training various image processing systems. The dataset includes handwritten digits and their labels that are widely used for training and testing in the field of machine learning. In this paper, we will train our model with the training data provided by MNIST and verify the performance of our classifier by the test data.

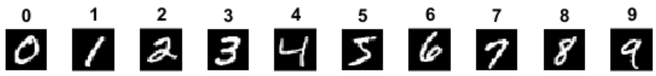

Figure 1: Handwritten digits with their labels

1.2 Approach the problem

We will use machine learning to train our computers. The primary strategy is to use edge detections in our image classifier. Our approach will be implemented as follows: use a wavelet transform on each image to detect edges and find the principal components of the wavelet transforms to see how digits differ in principle component analysis. Then, we will use linear discriminant analysis to determine some thresholds that separate numbers. In the end, we will test the algorithm on the test data to see its accuracy.

1.3 Overview

We will discuss singular value decomposition and linear discriminant analysis in Section Two. In Section Three, we will develop the solution and implement the corresponding algorithms. Section Four will include the computational results derived from the approach, as the summary and conclusion will be in Section Five.

2 Theoretical Background

2.1 Singular Value Decomposition

In matrix form, the singular value decomposition of the matrix A with size $m \times n$ can be represented with:

$$A = U\Sigma V^*$$

Where U is an $m \times m$ complex unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ complex unitary matrix. The singular value decomposition of matrix A is connected to the eigenvalue decomposition of AA^T. To account for the $\frac{1}{n-1}$ factor, consider:

$$A = \frac{1}{\sqrt{n-1}}X$$

Then, for principle component analysis, we have:

$$C_X = \frac{1}{n-1}XX^T = AA^T = U\Sigma^2U^T$$

So, the eigenvalues of the covariance matrix are the squared of the scaled singular values. Since we used a change of basis to work in the basis of the principal components, the data in the new coordinate and its covariance can be obtained by:

$$Y = U^TX, \qquad C_Y = \Sigma^2$$

2.2 Linear Discriminant Analysis

The goal of linear discriminant analysis(LDA) is to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. To implement LDA for 2 datasets, we need to find the means for each of our groups for each feature, μ_1 and μ_2. Note that these μ are column vectors since they are means across each row. We can then define the between-class scatter matrix:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$$

It measures the variance between the means. Then we can define the within-class scatter matrix:

$$S_w = \sum_{j=1}^{2}\sum_{x}(x - \mu_j)(x - \mu_j)^T$$

It measures the variance within each group. Our goal is find a vector w such that

$$w = argmax\frac{w^TS_Bw}{w^TS_ww}$$

Note that the vector w maximizes the above quotient when the eigenvector corresponds to the largest eigenvalue of the generalized eigenvalue problem:

$$S_Bw = \lambda S_ww$$

The LDA can also be used to classify between more than two groups with the following changes:

$$S_B = \sum_{j=1}^{N}(\mu_j - \mu)(\mu_j - \mu)^T$$
$$S_w = \sum_{j=1}^{N}\sum_{x}(x - \mu_j)(x - \mu_j)^T$$

Where μ is the overall mean and μ_j is the mean of each of the N groups/classes. Then, w is found as it was above.

2.3 Decision Tree Classifiers and Support-Vector Machines

Decision tree learning uses a decision tree to go from observations about an item to conclusion about the item's target value. Tree models where the target variable can take a discrete set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.

Support-vector machines(SVMs) are supervised learning models with associated learning algorithms that analysis data for classification and regression analysis. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernal trick, implicitly mapping their inputs into high-dimensional feature spaces.

3 Algorithm Implementation and Development

Since our goal is to identify the digits, it is essential for us to train our model effectively. To achieve this, we will implement the code into four main parts (the code implemented in MatLab is included in Appendix B):

1. Load and parse data, then apply the wavelet transformation to each image.
2. Analysis data by singular value decomposition and determine the number of features.
3. Build a two-digit/three-digit classifier by linear Discriminant analysis.
4. Build a Decision Tree classifier and Support-Vector Machines.

Algorithm 1: Load and parse data, then apply the wavelet transformation to each image.

Clean work space
Load and parse images in the form of $X \times Y \times N$, where X, Y represents the size of the image.
for *image in the training data* **do**
    Convert the image to double precision.
    Use Haar wavelet to find cH, cV of the image.
    Take the absolute value of cH, cV, then rescale.
    Combine cH, cV, reshape data to the appropriate size.
end

Note that cH records the horizontal details and cV records the vertical details. We need to rescale them to be between 0 and 1 for double precision. The absolute value keeps the information about which values are large, so we take the absolute value before scaling. In order to do edge detection, we combine horizontal details cH and vertical details cV by adding them together.

Algorithm 2: Analysis data by singular value decomposition and determine the number of features.

Combine and reshape each image int o a column vector where each column is a different image.
Apply the singular value decomposition to obtain $[U, S, V]$
Based on the singular values S, determine the number of features used for classifier.

The number of features determine the accuracy of our model. In the paper, we determine the number of features of by the number of singular values when its cumulative energy reaches 80%.

Algorithm 3: Build a two-digit/three-digit classifier by linear Discriminant analysis.

Pick the digits of interest and combine them as column vectors
Apply the singular value decomposition to obtain $[U, S, V]$
Define digits as $S \times V'$, and principle component $U = U(:, 1 : feature)$
Set $d1 = digits(1 : feature, 1 : nd1)$, where $nd1$ is the number of image1.
Set $d2 = digits(1 : feature, nd1 + 1 : nd1 + nd2)$, where $nd2$ is the number of image2.
Find the mean of $d1, d2$, then calculate S_w and S_b stated in section 2.2
Find the eigenvector v and eigenvalue d of S_b, S_w, then solve for w.
Project the data, set $vd1 = w' \times d1$ and $vd2 = w' \times d2$.
Find the appropriate threshold and separates $vd1, vd2$.

The above algorithm only works for classifying two digits. For three digits, we can find $\mu = mean(digits(1 : feature, 1 : nd1 + nd2 + nd3))$ and $d3 = digits(1 : feature, nd1 + nd2 + 1 : nd1 + nd2 + nd3)$.Follow section 2.2 to find S_b and S_w. In the end, we need to order the group and find 2 thresholds that classify three digits. To classify digits using our model, we need to project the test data as well, then compare it with the threshold(s) to determine the label of the digits.

Algorithm 4: Build a Decision Tree classifier and Support-Vector Machines

---

Apply wavelet transformation to the training data.
For decision tree, set the max number of splits to 10 and train the decision tree by the data and labels.
For SVM, train the model with the data and labels.
Find the class error of the tree and the prediction error made by the SVM.

The results of the computed data (along with the graph) are included in Section 4.

4 Computational Results

4.1 Data Analysis

With principle component Analysis by singular value decomposition, we can find that

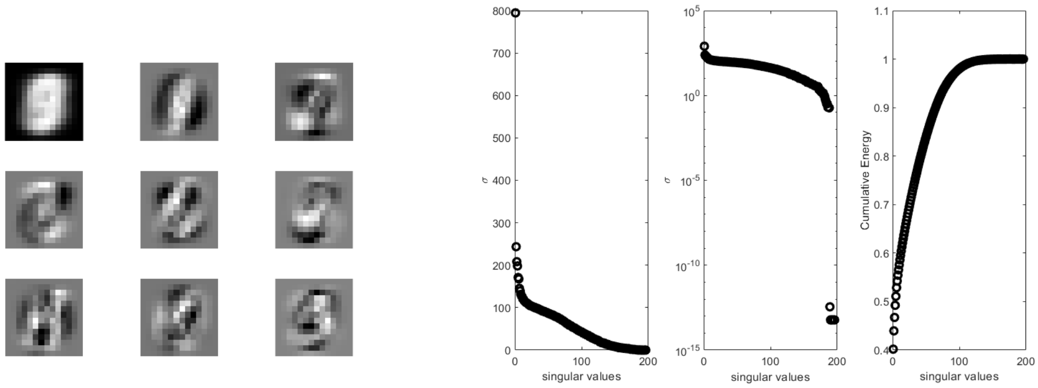

Figure 2: The first 9 principle component U and singular values, respectively

With an 80 percent cutoff, our training model requires roughly 43 features to classify digits. Based on the singular value decomposition, we can find that the emphasis is on circles. Digits like 0,3,8 dominate the principal components. This is one of the main features we used to differentiate between numbers. By looking at singular values, it is clear the first mode is dominant. On a log scale, We can also see another sharp drop-off when $x = 188$, which indicates that our model works best with 188 features.

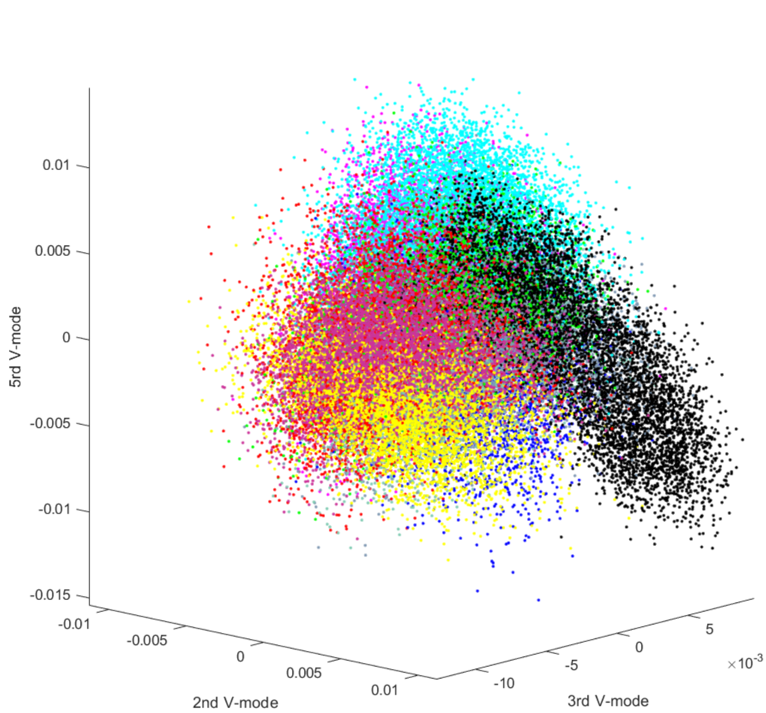

Figure 3: Projection of V mode 2,3,5

On inspection, some digits are clearly separable. For example, 1 and 7 do not overlap on the 2nd mode and 5th mode, making our classifier easier to identify the difference. Some digits like 3 and 5 do overlap on these selected modes, requiring our classifier to check other modes.

4.2 Two-Digit Classifier

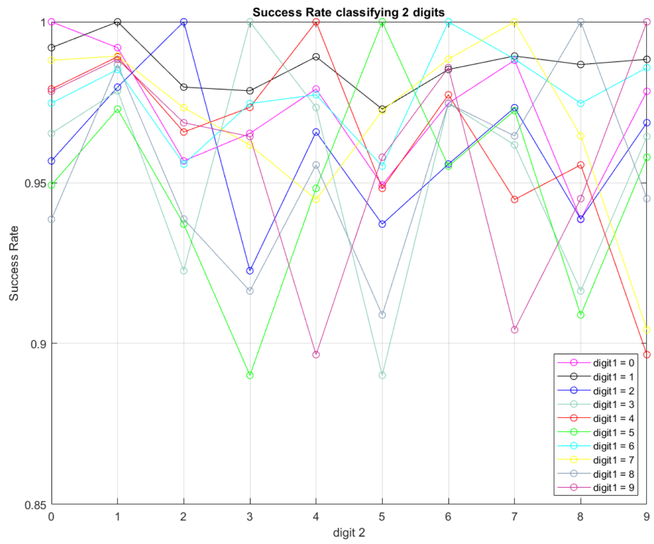

Figure 4: Success rate of classifying two digits

As you might see in the graph, our linear classifier is capable of classifying two digits with roughly 90% accuracy. Based on the data, 0 and 1 are the easiest numbers to separate (with 99.2% accuracy). 3 and 5 are the most difficult numbers to separate(with 89% accuracy). 4 and 9 are the second most difficult numbers to separate(with 89.5% accuracy). The result verifies that our classifier is using features like circles and lines to differentiate between numbers.

4.3 Three-Digit Classifier

The three-digit classifier we implemented requires the digits' order (either from the smallest to biggest or vice versa). To guarantee availability, Our code in appendix B prints out the debugging message to help clients ordering the digits. Note that there are 3! ways of ordering three digits. When we compare $1, 3, 5$ instead of $3, 5$, the accuracy drops from 89% to 67.3%. When we compare $0, 1, 2$ instead of $0, 1$, the accuracy drops from 99.2% to 89.6%. It is reasonable to see a dropoff in accuracy as the number of digits increases. However, this phenomenon also illustrates that our linear model may not perform well on predicting digits when the possible choices increase.

4.4 Decision Tree Classifier and Support Vector Machines

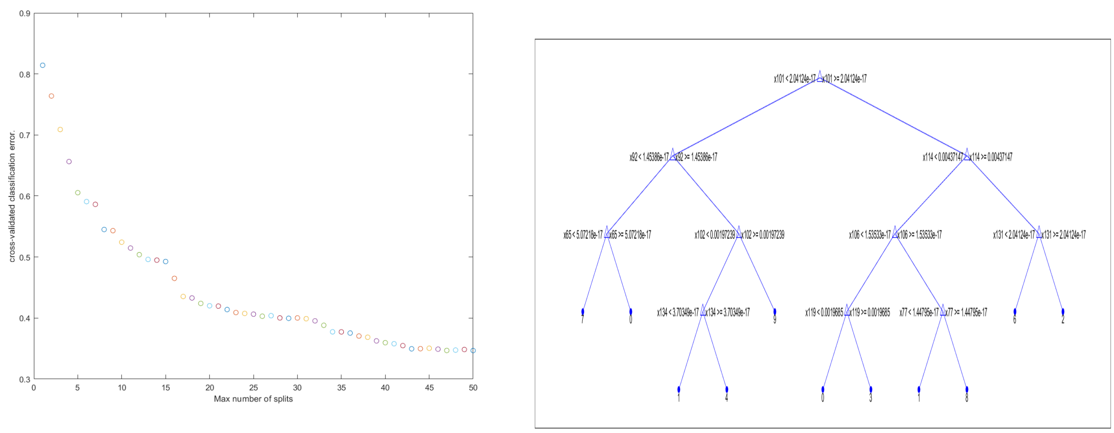

Figure 5: Cross Validation error and the tree, respectively

Unlike linear classifiers, we are using all possible digits to train the model. As the max number of split increases in the decision tree classifier, the classification error decreases. However, it becomes longer to train the model as the max number of split increases. The support vector machine can classify all digits as well. However, it takes a long time to train. Once it is trained, differentiating numbers can be done effectively, with 88.3% accuracy.

5 Summary and Conclusions

The singular value decomposition allows us to analyze data and determine the number of features needed in our classifier and predict which digits might be hard to differentiate. The linear discriminate analysis makes a projection that maximizes the distance between the inter-class data while minimizing the intra-class data. By building a linear classifier, we can efficiently train our model and classify digits. However, as the number of possible choices increases, it becomes harder for our classifier to differentiate digits. Other standard machine learning models like the Decision Tree classifier and Support Vector Machine can classify all numbers simultaneously. It might take some time to train the model, but differentiating numbers can be done effectively once it is trained.

Appendices

Appendix A: MatLab functions used and brief explanations

The following list includes the MatLab functions used in Appendix B. It also contains a brief implementation explanation of their functionalities. More information can be found at MatLab Documentation at MathWorks.

- *Y = swapbytes(X)* swaps the byte ordering of each element in array X from little endian to big endian (or vice versa).
- *szdim = size(A,dim)*: returns the length of dimension *dim* when *dim* is a positive integer scalar.
- *I2 = im2double(I)*: converts the image I to double precision.
- *[cA,cH,cV,cD] = dwt2(X,wname)*: computes the single-level 2-D discrete wavelet transform (DWT) of the input data X using the wname wavelet. dwt2 returns the approximation coefficients matrix cA and detail coefficients matrices cH, cV, and cD (horizontal, vertical, and diagonal, respectively).
- *B = rescale(A)* scales the entries of an array to the interval [0,1]
- *B = reshape(A,sz)* reshapes A using the size vector, sz, to define size(B)
- *k = find(X)*: returns a vector with the same orientation as X.
- *B = cumsum(A)* returns the cumulative sum of A.
- *imshow(I)* displays the grayscale image I in a figure
- *M = mean(A)*: returns the mean of the elements of A
- *[V,D] = eig(A)* returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that $A \times V = V \times D$.
- *[U,S,V] = svd(A,'econ')*: produces an economy-size decomposition of m-by-n matrix A
- *D = diag(v)*: returns a square diagonal matrix with the elements of vector v on the main diagonal.
- *n = norm(v,p)*: returns the generalized vector p-norm.
- *tree = fitctree(Tbl,formula)*: returns a fitted binary classification decision tree based on the input variables contained in the table *Tbl*. $formula$ is an explanatory model of the response and a subset of predictor variables in *Tbl* used to fit tree.
- *Mdl = fitcecoc(Tbl,formula)* returns an ECOC model using the predictors in table *Tbl* and the class labels. $formula$ is an explanatory model of the response and a subset of predictor variables in *Tbl* used for training.
- *label = predict(Mdl,X)* returns a vector of predicted class labels for the predictor data in the table or matrix X, based on the trained, full or compact classification tree Mdl.

Appendix B: MatLab Code

The main executable file:

```
%% clean and load data
clear variables; close all; clc;

% load data
[train_images, train_labels] = mnist_parse("train-images.idx3-ubyte", "train-labels.idx1-ubyte", false);
[test_images, test_labels] = mnist_parse("t10k-images.idx3-ubyte", "t10k-labels.idx1-ubyte", false);

% transform the entire dataset
train_wavelet = d_wavelet(train_images);
test_wavelet = d_wavelet(test_images);

% classify data by their labels
[Digit0,Digit1,Digit2,Digit3,Digit4,Digit5,Digit6,Digit7,Digit8,Digit9] = sortDigits(train_images, train_labels);
[Test0,Test1,Test2,Test3,Test4,Test5,Test6,Test7,Test8,Test9] = sortDigits(test_images, test_labels);

% cutoff for adjusting features
cutoff = 0.80;

% default feature
feature = 25;

% Example: display a few data in the training set
image_size = size(train_images, 1);
figure(1)
image_index = 0;
while image_index < 10
    for i = 1:size(test_labels)
        if (train_labels(i) == image_index)
            subplot(1,10,image_index + 1);
            imshow(train_images(:,:,i));
            title(num2str(image_index));
            break;
        end
    end
    image_index = image_index + 1;
end


%% Perform wavelet transformation and Singular Value Decomposition
digit_wave = zeros(size(train_images,1)*size(train_images,2)/4,size(train_images,3));
for i = 1:size(train_images,3)
     digit_wave(:,i) = d_wavelet(train_images(:,:,i));
end
[U,S,V] = svd(digit_wave,'econ');

sig = diag(S);
cumulative_energy = cumsum(sig.^2)/sum(sig.^2);
for i = 1:length(cumulative_energy)
     if cumulative_energy(i) >= cutoff
          break
     end
end
```

```
feature = i;

%% Perform analysis
% plot the first nine principal components
figure(2)
for k = 1:9
    subplot(3,3,k)
    ut1 = reshape(U(:,k),14,14);
    ut2 = rescale(ut1);
    imshow(ut2);
end


% plot singular values
figure(3)
subplot(1,3,1); plot(sig,'ko','Linewidth',2);
ylabel('\sigma');xlabel('singular values')
subplot(1,3,2); semilogy(sig,'ko','Linewidth',2);
ylabel('\sigma');xlabel('singular values')
subplot(1,3,3);plot(cumulative_energy,'ko','Linewidth',2)
ylabel('Cumulative Energy');xlabel('singular values');


% Project selected V-modes
% for k = 1:3
%     subplot(3,1,k);
%     plot(t,V(t,k), 'ko-');
%     legend(['Mode ', num2str(k)], 'Location','SouthEast');
% end
figure(4)
C = {'m','k','b',[.5 .8 .7],'r','g','c','y',[.5 .6 .7],[.8 .2 .6]}; % Cell array of colors.
legends = cell(10,1);
for label = 0:9
    indices = find(train_labels == label);
    plot3(V(indices,2), V(indices,3),V(indices,5), '.','color',C{label+1});
    legends{label+1} = num2str(label);
    hold on, drawnow;
end
xlabel('2nd V-mode');
ylabel('3rd V-mode');
zlabel('5rd V-mode');
legend(legends);



%% Find two digits that are most easy/difficult to separate
minRate = 1; minDigits = zeros(1,2);
maxRate = 0; maxDigits = zeros(1,2);
figure(5)
for i = 0:9
    correct_rates = zeros(10,1);
    for j = 0:9
        if (i == j)
            correct_rates(j+1) = 1;
            continue
        end
        digit1 = ['Digit',num2str(i)];
        digit2 = ['Digit',num2str(j)];
        test1 = ['Test',num2str(i)];
        test2 = ['Test',num2str(j)];
```

```
        correct_rate = twoDigitsClassifier(eval(digit1),eval(digit2),eval(test1),eval(test2),feature);
        correct_rates(j+1) = correct_rate;
        if (correct_rate < minRate)
            minRate = correct_rate;
            minDigits(1) = i;
            minDigits(2) = j;
        end
        if (correct_rate > maxRate)
            maxRate = correct_rate;
            maxDigits(1) = i;
            maxDigits(2) = j;
        end
    end

    plot(0:9,correct_rates, '-o','color',C{i+1});
    xlabel('digit 2'), ylabel('Success Rate')
    xlim([0,9]),ylim([0.85,1.0])
    legends{i+1} = ['digit1 = ', num2str(i)];
    title(legends{i+1});
    hold on, grid on, drawnow
end
title('Success Rate classifying 2 digits');
legend(legends,'Location','SouthEast');



%% Try classify three digits
% may not work on certain orders, see output in the console to debug.
clc;

% an example of 'buggy' code
% threeDigitsClassifier(Digit1,Digit3,Digit5,Test1,Test3,Test5);

% fix the order
% threeDigitsClassifier(Digit1,Digit5,Digit3,Test1,Test5,Test3);
rate1 = threeDigitsClassifier(Digit3,Digit5,Digit1,Test3,Test5,Test1,feature); % 1,3,5
rate2 = threeDigitsClassifier(Digit9,Digit4,Digit1,Test9,Test4,Test1,feature); % 1,4,9
rate3 = threeDigitsClassifier(Digit0,Digit2,Digit1,Test0,Test2,Test1,feature); % 0,1,2


%% Other Machine Learning Methods
clc; close all;

% classification tree
% figure(6)
for numSplits = 1:50
    tree=fitctree(train_wavelet',train_labels,'MaxNumSplits',numSplits,'CrossVal','on');
    % view(tree.Trained{1},'Mode','graph');
    % fprintf(['done ' , num2str(numSplits), '\n']);
    classError = kfoldLoss(tree);
    plot(numSplits, classError,'-o');
    hold on, drawnow;
end
xlabel('Max number of splits');
ylabel('cross-validated classification error.');
```

```
tree=fitctree(train_wavelet',train_labels,'MaxNumSplits',10,'CrossVal','on');
view(tree.Trained{1},'Mode','graph');


% SVM classifier with training data, labels and test set
Mdl = fitcecoc(train_wavelet',train_labels);
predict_labels = Mdl.predict(test_wavelet');
SVMError = length(find(predict_labels - test_labels == 0))/length(test_labels)
```

mnist_parse file:

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels, showMsg)

% The function is curtesy of stackoverflow user rayryeng from Sept. 20,
% 2016. Link: https://stackoverflow.com/questions/39580926/how-do-i-load-in-the-mnist-digits-and-label-data-in-matlab

% Open files
fid1 = fopen(path_to_digits, 'r');

% The labels file
fid2 = fopen(path_to_labels, 'r');

% Read in magic numbers for both files
A = fread(fid1, 1, 'uint32');
magicNumber1 = swapbytes(uint32(A)); % Should be 2051
if (showMsg)
    fprintf('Magic Number - Images: %d\n', magicNumber1);
end

A = fread(fid2, 1, 'uint32');
magicNumber2 = swapbytes(uint32(A)); % Should be 2049
if (showMsg)
    fprintf('Magic Number - Labels: %d\n', magicNumber2);
end

% Read in total number of images
% Ensure that this number matches with the labels file
A = fread(fid1, 1, 'uint32');
totalImages = swapbytes(uint32(A));
A = fread(fid2, 1, 'uint32');
if totalImages ~= swapbytes(uint32(A))
    error('Total number of images read from images and labels files are not the same');
end
if showMsg
    fprintf('Total number of images: %d\n', totalImages);
end

% Read in number of rows
A = fread(fid1, 1, 'uint32');
numRows = swapbytes(uint32(A));

% Read in number of columns
A = fread(fid1, 1, 'uint32');
numCols = swapbytes(uint32(A));

if showMsg
    fprintf('Dimensions of each digit: %d x %d\n', numRows, numCols);
end

% For each image, store into an individual slice
```

```
images = zeros(numRows, numCols, totalImages, 'uint8');
for k = 1 : totalImages
    % Read in numRows*numCols pixels at a time
    A = fread(fid1, numRows*numCols, 'uint8');

    % Reshape so that it becomes a matrix
    % We are actually reading this in column major format
    % so we need to transpose this at the end
    images(:,:,k) = reshape(uint8(A), numCols, numRows).';
end

% Read in the labels
labels = fread(fid2, totalImages, 'uint8');

% Close the files
fclose(fid1);
fclose(fid2);

end
```

d_wavelet file

```
% wavelet transform
function dData = d_wavelet(dfile)

    [m,~, n] = size(dfile);  % 28*28*n
    nw = m^2/4;              % wavelet resolution
    dData = zeros(nw,n);

    for k = 1:n
        X = im2double(dfile(:,:,k));
        [~,cH,cV,~] = dwt2(X,'haar');
        cod_cH1 = rescale(abs(cH));
        cod_cV1 = rescale(abs(cV));
        cod_edge = cod_cH1 + cod_cV1;
        dData(:,k) = reshape(cod_edge,nw,1);
    end
end
```

SortDigits file

```
% Classify digits and perform wavelet transformation
function [digit0,digit1,digit2,digit3,digit4,digit5,digit6,digit7,digit8,digit9]
    = sortDigits(images, labels)

Digit0 = []; count0 = 1;
Digit1 = []; count1 = 1;
Digit2 = []; count2 = 1;
Digit3 = []; count3 = 1;
Digit4 = []; count4 = 1;
Digit5 = []; count5 = 1;
Digit6 = []; count6 = 1;
Digit7 = []; count7 = 1;
Digit8 = []; count8 = 1;
Digit9 = []; count9 = 1;
for i = 1:size(images,3)
    if labels(i) == 0
        Digit0(:,:,count0) = images(:,:,i);
        count0 = count0 + 1;
    elseif labels(i) == 1
        Digit1(:,:,count1) = images(:,:,i);
        count1 = count1 + 1;
    elseif labels(i) == 2
```

```
            Digit2(:,:,count2) = images(:,:,i);
            count2 = count2 + 1;
        elseif labels(i) == 3
            Digit3(:,:,count3) = images(:,:,i);
            count3 = count3 + 1;
        elseif labels(i) == 4
            Digit4(:,:,count4) = images(:,:,i);
            count4 = count4 + 1;
        elseif labels(i) == 5
            Digit5(:,:,count5) = images(:,:,i);
            count5 = count5 + 1;
        elseif labels(i) == 6
            Digit6(:,:,count6) = images(:,:,i);
            count6 = count6 + 1;
        elseif labels(i) == 7
            Digit7(:,:,count7) = images(:,:,i);
            count7 = count7 + 1;
        elseif labels(i) == 8
            Digit8(:,:,count8) = images(:,:,i);
            count8 = count8 + 1;
        elseif labels(i) == 9
            Digit9(:,:,count9) = images(:,:,i);
            count9 = count9 + 1;
        end
end

digit0 = d_wavelet(Digit0);
digit1 = d_wavelet(Digit1);
digit2 = d_wavelet(Digit2);
digit3 = d_wavelet(Digit3);
digit4 = d_wavelet(Digit4);
digit5 = d_wavelet(Digit5);
digit6 = d_wavelet(Digit6);
digit7 = d_wavelet(Digit7);
digit8 = d_wavelet(Digit8);
digit9 = d_wavelet(Digit9);

end
```

Two-Digit classifier file

```
% main classifier for two given digits
function [correct_rate] = twoDigitsClassifier(digit1_wave,digit2_wave,Test_wave1
    ,Test_wave2,feature)

[U,~,~,threshold,w,~,~] = trainer(digit1_wave, digit2_wave,feature);

Test_Mat1 = U'* Test_wave1;      % PCA Projection
Test_Mat2 = U'* Test_wave2;
pVal1 = w' * Test_Mat1;
pVal2 = w' * Test_Mat2;

ResVec1 = (pVal1 < threshold);  % is digit 1
ResVec2 = (pVal2 > threshold);  % is digit 2

% calculate stats
total_cases = size(Test_wave1,2) + size(Test_wave2,2);
correct_cases = sum(ResVec1(:)) + sum(ResVec2(:));
correct_rate = correct_cases/total_cases;

end

```

```matlab
% train data
function [U,S,V,threshold,w,sortd1,sortd2] = trainer(d1_wave,d2_wave,feature)

    nd1 = size(d1_wave,2);
    nd2 = size(d2_wave,2);
    [U,S,V] = svd([d1_wave d2_wave],'econ');
    digits = S*V';
    U = U(:,1:feature); % Add this in
    d1 = digits(1:feature,1:nd1);
    d2 = digits(1:feature,nd1+1:nd1+nd2);
    md1 = mean(d1,2);
    md2 = mean(d2,2);

    Sw = 0;
    for k=1:nd1
        Sw = Sw + (d1(:,k)-md1)*(d1(:,k)-md1)';
    end
    for k=1:nd2
        Sw = Sw + (d2(:,k)-md2)*(d2(:,k)-md2)';
    end
    Sb = (md1-md2)*(md1-md2)';

    [V2,D] = eig(Sb,Sw);         % linear discriminant analysis
    [~,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    vd1 = w' * d1;
    vd2 = w' * d2;

    % adjust the order to be vd1 < vd2
    if mean(vd1) > mean(vd2)
        w = -w;
        vd1 = -vd1;
        vd2 = -vd2;
    end

    sortd1 = sort(vd1);
    sortd2 = sort(vd2);
    t1 = length(sortd1);
    t2 = 1;
    while sortd1(t1) > sortd2(t2)
        t1 = t1 - 1;
        t2 = t2 + 1;
    end
    threshold = (sortd1(t1) + sortd2(t2))/2;

end
```

Three-Digit classifier file

```matlab
% main classifier for three given digits
function [correct_rate] = threeDigitsClassifier(digit1_wave,digit2_wave,
    digit3_wave,Test_wave1,Test_wave2,Test_wave3,feature)

[U,~,~,threshold1,threshold2,w] = trainer(digit1_wave, digit2_wave, digit3_wave,
    feature);

if (threshold1 == -1 && threshold2 == -1)
    fprintf("Error: cannot order variables (threeDigitsClassifier)\n");
    correct_rate = 0;
    return
end
```

```
Test_Mat1 = U'* Test_wave1;          % PCA Projection
Test_Mat2 = U'* Test_wave2;
Test_Mat3 = U'* Test_wave3;

pVal1 = w' * Test_Mat1;
pVal2 = w' * Test_Mat2;
pVal3 = w' * Test_Mat3;

ResVec1 = (pVal1 < threshold1);                          % is digit 1
ResVec2 = (pVal2 > threshold1 & pVal2 < threshold2);     % is digit 2
ResVec3 = (pVal3 > threshold2);                          % is digit 3

% calculate stats
total_cases = size(Test_wave1,2) + size(Test_wave2,2) + size(Test_wave3,2);
correct_cases = sum(ResVec1(:)) + sum(ResVec2(:)) + sum(ResVec3(:));
correct_rate = correct_cases/total_cases;

end

function [U,S,V,threshold1, threshold2,w] = trainer(d1_wave,d2_wave,d3_wave,
    feature)

    nd1 = size(d1_wave,2);
    nd2 = size(d2_wave,2);
    nd3 = size(d3_wave,2);

    [U,S,V] = svd([d1_wave,d2_wave,d3_wave],'econ');
    digits = S*V';
    U = U(:,1:feature); % Add this in

    d1 = digits(1:feature,1:nd1);
    d2 = digits(1:feature,nd1+1:nd1+nd2);
    d3 = digits(1:feature,nd1+nd2+1:nd1+nd2+nd3);
    d = digits(1:feature,1:nd1+nd2+nd3);

    md1 = mean(d1,2);
    md2 = mean(d2,2);
    md3 = mean(d3,2);
    md = mean(d,2);

    Sw = 0;
    for k=1:nd1
        Sw = Sw + (d1(:,k)-md1)*(d1(:,k)-md1)';
    end
    for k=1:nd2
        Sw = Sw + (d2(:,k)-md2)*(d2(:,k)-md2)';
    end
    for k=1:nd3
        Sw = Sw + (d3(:,k)-md3)*(d3(:,k)-md3)';
    end

    %Sb = (md1-md2)*(md1-md2)';
    Sb = (md1-md)*(md1-md)';
    Sb = Sb + (md2-md)*(md2-md)';
    Sb = Sb + (md3-md)*(md3-md)';

    [V2,D] = eig(Sb,Sw);          % linear discriminant analysis
    [~,ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
```

```
    vd1 = w' * d1;
    vd2 = w' * d2;
    vd3 = w' * d3;

    % adjust the order to be vd1 < vd2 < vd3
    if mean(vd1) > mean(vd2)
        % vd1 > vd2 > vd3
        if mean(vd2) > mean(vd3)
            w = -w;
            vd1 = -vd1;
            vd2 = -vd2;
            vd3 = -vd3;
        end

        % TODO: implement more
    end

    if (mean(vd1) < mean(vd2) && mean(vd2) < mean(vd3))
        sortd1 = sort(vd1);
        sortd2 = sort(vd2);
        sortd3 = sort(vd3);

        t1 = length(sortd1);
        t2 = 1;
        while sortd1(t1) > sortd2(t2)
            t1 = t1 - 1;
            t2 = t2 + 1;
        end
        threshold1 = (sortd1(t1) + sortd2(t2))/2;

        t2 = length(sortd2);
        t3 = 1;
        while sortd2(t2) > sortd3(t3)
            t2 = t2 - 1;
            t3 = t3 + 1;
        end
        threshold2 = (sortd2(t2) + sortd3(t3))/2;
        return;
    else
        % for debugging
        threshold1 = -1;
        threshold2 = -1;

        fprintf("Please order the variables(either order is fine).\n");
        fprintf("Current mean: %f, %f, %f.\n",mean(vd1),mean(vd2),mean(vd3))
    end

end
```