AMATH 482 Homework 4

Avery Milandin

Due February 10th, 2021

Abstract

In this project we develop machine learning models to classify pictures of hand-drawn digits from 0 to 9. We train our models using 60,000 training images, and then we test our models on a set of 10,000 images. We develop models using linear discriminant analysis, decision trees, and support vector machines.

1 Introduction and Overview

Our training and test data comes from `http://yann.lecun.com/exdb/mnist/`. We use this data to develop our own models using linear discriminant analysis (LDA), and we also use it to train decision trees and support vector machines (SVM) using built-in MATLAB commands. For the LDA models, we develop a model to classify between two digits for every possible digit pair: (0,1), (0,2), (0,3)...(1,2), (1,3), (1,4)...(8,9). To do so, we take the SVD of our image data matrix and examine the singular value plot to see how many features we need to include in order to develop reasonably accurate models, and then we create the models. We also develop one LDA model to distinguish between three different digits. After developing these models we test them on the test data and measure their accuracy. We also develop a decision tree classifier and an SVM classifier to classify all 10 digits. We train these models using built-in commands and measure their accuracy on the test data. Lastly, we develop decision tree classifiers and SVM classifiers to distinguish between just two digits rather than all 10, and we do this for the pairs of digits that our LDA classifiers had the highest and lowest accuracy on. We then compare the accuracy of these classifiers to that of the LDA classifiers.

2 Theoretical Background

The first thing that is important to understand for this project is the SVD of our training data. Our training data is 60,000 pictures, each of which is 28×28 pixels, so the tensor is $28 \times 28 \times 60000$. Since we need a 2-D matrix, we reshape each 28×28 picture into a 784×1 vector, so our training data matrix is a 784×6000 where each column represents and image. When we take the SVD of this matrix, we get three matrices U, Σ, V.

The columns of U can be thought of as representing different "features" of the digits that can be used as classifiers. The first column is the most important feature for classification, the second is the second most important, and so on. The singular values in Σ quantify the relative importance of each of these features. The square of each individual singular value tells us how much of the variation in the digits can be described by the "feature" corresponding to the singular value. The columns of V^T tell us how much of each "feature" makes up one image, so the first value of the first column of V^T tells us how much of the feature described by $\vec{u_1}$ is present in the first image, and the second tells us how much of the second feature is present, and so on.

It is also important to understand Linear Discriminant Analysis. LDA starts with two clusters of high dimensional data. More than two clusters can be used, as will be demonstrated later, but typically LDA is used with only a small number of data clusters. The goal is to project the data from both of these clusters on to one dimension such that the two means corresponding to the projected data clusters are as far apart as possible, and the intra-class projected data for each cluster has as little spread as possible.

We can define a between-class scatter matrix as:

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

where μ_2 and μ_1 are vectors containing means of each principal component that we projected our training data onto. In the case of our project, we projected our training data onto the first 30 principal components, so these vectors have 30 components each.

We can define an within-class scatter matrix as:

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

where the X vectors are images, so we are summing over the distance from each image to its own mean.

It can be proven that solving the following problem for $\vec{W}$ gives us the line that maximizes S_B and minimizes S_W:

$$S_B \vec{W} = \Lambda S_W \vec{W}$$

where Λ is the largest possible eigenvalue. This means that $\vec{W}$ is the line that we should project onto.

Once we have found this line, we can find a threshold, which is just a point on the line, where anything below that threshold is classified as a digit of the first type, and anything above is classified as a digit of the second type. We can then take our test data and project it onto its first 30 principal components, and then project onto this line, and compare all of the resulting points to the determined threshold value to classify images from our test data.

3 Algorithm Implementation and Development

Algorithm 1

The first thing that we need is a way to extract data corresponding to specific digits from our training and test data. Our data contains data for all 10 digits, but for the classifiers we make, we will usually only need data on two or three digits. The steps to extract data for two digits from a data tensor and a vector of labels corresponding to images in the tensor is as follows:

1. Reshape our $28 \times 28 \times n$ tensor into a $784 \times n$ matrix.
2. Initialize 2 matrices to be used for holding columns corresponding to two different digits.
3. Loop over the columns of the reshaped matrix and perform steps 4-5.
4. If the label corresponding to the current column is the first digit we are looking for, add the current column to the first matrix we initialized.
5. If the label corresponds to the other digit we are looking for, add the current column to the other matrix we initialized.
6. After finishing the loop, convert the two matrices that were generated from uint8 to doubles so that we will be able to work with them.

We created a function for this algorithm, and the function code is below. We also created a function to do this for 3 digits, but it is essentially the same, so we did not include it in this section, but it can be found in the attached MATLAB code.

```
function [mat1, mat2] = extract_data(val1, val2, mat, labels)
    reshapedMat = reshape(mat, 28*28, length(mat));
    mat1 = [];
    mat2 = [];
    for j = 1: length(reshapedMat)
        if (labels(j) == val1)
```

```
            mat1 = [mat1, reshapedMat(:,j)];
        elseif (labels(j) == val2)
            mat2 = [mat2, reshapedMat(:,j)];
        end
    end
    mat1 = im2double(mat1);
    mat2 = im2double(mat2);
end
```

Algorithm 2

We now need to train an LDA classifier to distinguish between two digits. We created a function to do this that returns the U, S, V matrices from the SVD, the threshold value, and the line $\vec{W}$ to project our training data onto. These returned values will be used in a later algorithm. The steps for this algorithm are as follows:

1. Extract data matrices for the two digits that we are going to train our model on using Algorithm 1, and put the two returned matrices into a single data matrix.
2. Since we only need edge data and not the full images, apply a wavelet transform to our data matrix using the provided dc_wavelet function.
3. Take the SVD of our data matrix and project our data onto the first 30 principal components.
4. Truncate the resultant U matrix to contain only the first 30 columns (this is the U matrix returned by this function)
5. Break the projection matrix into two matrices, one for each digit, and save these matrices in variables.
6. Save the means of the columns of both projection matrices in two variables.
7. Populate a within-class scatter matrix as defined in the Theoretical Background section using a loop.
8. Define a between-class scatter matrix as defined in the Theoretical Background section
9. Solve the eigenvalue problem from the Theoretical Background section and save the result in a variable w.
10. Determine the threshold value using a loop(see code).
11. Return all variables that are to be returned.

The code for this algorithm is below:

```
function [U,S,V,threshold,w] = digit_trainer(dOne, dTwo, images, labels)
[firstDigit, secondDigit] = extract_data(dOne, dTwo, images, labels);
digits = dc_wavelet([firstDigit, secondDigit]);
features = 30;
[U,S,V] = svd(digits, 'econ');
U = U(:,1:features);
numFirst = size(firstDigit, 2);
numSecond = size(secondDigit, 2);
digitsProj = S*V';
firstProj = digitsProj(1:features, 1:numFirst);
secondProj = digitsProj(1:features, numFirst+1:numFirst + numSecond);
mFirst = mean(firstProj,2);
mSecond = mean(secondProj,2);
Sw = 0; % within class variances
for k = 1:numFirst
    Sw = Sw + (firstProj(:,k) - mFirst)*(firstProj(:,k) - mFirst)';
```

```
end
for k = 1:numSecond
    Sw =  Sw + (secondProj(:,k) - mSecond)*(secondProj(:,k) - mSecond)';
end
Sb = (mFirst-mSecond)*(mFirst-mSecond)'; % between class
[V2,D] = eig(Sb,Sw);
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
vFirst = w'*firstProj;
vSecond = w'*secondProj;
if mean(vFirst) > mean(vSecond)
    w = -w;
    vFirst = -vFirst;
    vSecond = -vSecond;
end
sortFirst = sort(vFirst);
sortSecond = sort(vSecond);
t1 = length(sortFirst);
t2 = 1;
while sortFirst(t1) > sortSecond(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sortFirst(t1) + sortSecond(t2))/2;
end
```

We also need a similar function for training and LDA classifier to distinguish between three digits. The function is nearly the same but with some differences. The code for this algorithm can be found in the attached MATLAB code, but it is fairly redundant, so we have not included it in this section. However, the differences are listed below:

1. Instead of returning a threshold, just return the mean of the projections of the first, second, and third clusters.
2. When defining the between-class scatter matrix, instead of considering just the distance between two means, considered the distance between means one and two, means two and three, and means one and three, and sum these values. This way we will maximize the distance between all three clusters.

Algorithm 3
In order to determine which pairs of digits were easiest and hardest to classify, we looped over all possible pairs of digits, created a model for each, and tested it on the test data. The steps to test a model's accuracy on a pair of digits are below:

1. Train a model to classify digits j and k using algorithm 2
2. Extract test data for digits j and k using algorithm 1, and apply a wavelet transform.
3. Project onto the first 30 PCA modes by multiplying the transformed test data by U'.
4. Project onto $\vec{W}$ by multiplying this data by $\vec{W}$
5. Classify all values in the resultant vector as digit j if they are above the threshold and digit k otherwise.
6. compare these classifications to the known image labels and calculate the percentage correct; save this value as the accuracy for the classifier.

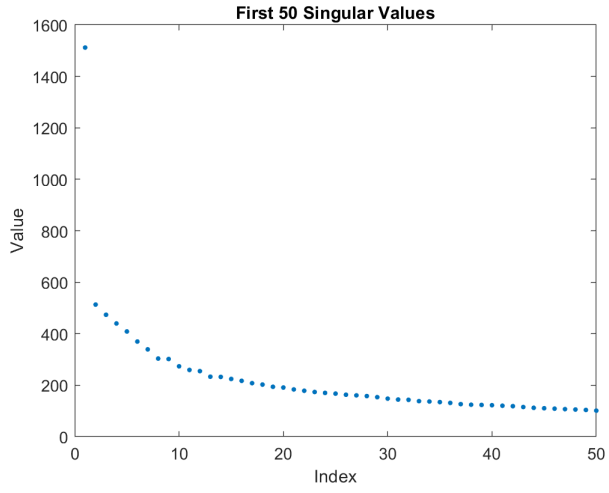
Below is the code for this algorithm within a nested loop over all possible combinations of digits. The output gives the accuracy of each model, as well as the least accurate and most accurate models, and the results of this can be found in the computational results section.

```
for j = 0:9
    for k = (j+1):9
        [U,S,V,threshold,w] = digit_trainer(j,k,images,labels);
        [digit1, digit2] = extract_data(j,k,testImages, testLabels);
        Test_wave = dc_wavelet([digit1, digit2]);
        TestMat = U'*Test_wave; % PCA projection
        pval = w'*TestMat;
        ResVec = (pval > threshold);
        solutions = [zeros(1, length(digit1)) ones(1, length(digit2))];
        incorrect = abs(ResVec - solutions);
        accuracy = 100 - sum(incorrect)/length(solutions)*100;
        if (accuracy > highestAccuracy)
            highestAccuracy = accuracy;
            easiest = [j,k];
        end
        if (accuracy < lowestAccuracy)
            lowestAccuracy = accuracy;
            hardest = [j,k];
        end
        allAccuracy = [allAccuracy; [j, k, accuracy]]
    end
end
```

For the LDA classifier for 3 digits, the difference is that we classify digits based simply on which of the three means values are closest to instead of attempting to define thresholds. The code for this is not included in this section but can be found in the attached MATLAB code.

We also developed algorithms to train decision trees and SVM models, but they are rather trivial and not the main focus of the project, so they are not included here but can be found in the attached MATLAB code. We trained a decision tree as well as an SVM model to classify all 10 digits, and we also trained decision trees and SVM models to classify just the digit pairs that were easiest and hardest for the LDA models. The accuracy of these models can be found in the Computational Results section and are discussed further in the Summary and Conclusions Section.

4 Computational Results



We can see here that Singular values of our training data matrix drop off quite quickly, and keep in mind that this is only the first 50 of 784 singular values, but it is clear from this plot that using just the first 30 singular values for this project will give us accurate models

Digit	0	1	2	3	4	5	6	7	8	9
0	X	99.2	95.2	96.2	97.0	94.8	97.1	98.9	94.3	97.6
1	X	X	97.6	97.9	98.9	97.2	98.8	98.6	98.4	98.6
2	X	X	X	91.6	96.3	93.5	95.2	96.9	93.7	96.8
3	X	X	X	X	96.8	87.6	97.0	95.9	90.7	95.7
4	X	X	X	X	X	93.6	97.1	93.7	95.2	89.2
5	X	X	X	X	X	X	95.4	96.9	90.1	95.4
6	X	X	X	X	X	X	X	98.6	97.1	98.5
7	X	X	X	X	X	X	X	X	96.4	90.3
8	X	X	X	X	X	X	X	X	X	94.1

Table 1: Accuracy of 2-digit LDA Classifiers

hardest: [3, 5], 87.645% accuracy
easiest: [0,1], 99.196% accuracy

3-digit LDA classification of 0,1, and 8:
0 classification accuracy: 82.959%
1 classification accuracy: 97.621%
8 classification accuracy: 77.105%
Overall accuracy: 86.501%

Decision tree classification accuracy on all 10 digits: 87.770%
Decision tree classification accuracy on [0,1]: 99.622%
Decision tree classification accuracy on [3,5]: 95.847%

SVM classification accuracy on all 10 digits: 94.380%
SVM classification accuracy on [0,1]: 99.905%
SVM classification accuracy on [3,5]: 96.688%

5 Summary and Conclusions

We can see from table 1 that our LDA classifiers were able to classify any pair of two digits with reasonably high accuracy, with the lowest being 87.6% for digits 3 and 5 and the highest being 99.2% for 0 and 1. Both of these values are worse than that of decision trees and SVM classifiers, with SVM classifiers being the most accurate on both [0,1] and [3,5]. Also, if we compare the overall accuracy of our 3-digit LDA classifier with the 10-digit accuracy of both the decision tree and the SVM model, we can see that LDA did worse at classifying just three digits than the other models did at classifying all 10. This means that decision trees and SVM models are significantly better at classifying data into many classes, with the decision tree classifying with 87.8% accuracy and the SVM model classifying with 94.4% accuracy.

Appendix A MATLAB Functions

- `gunzip(fileName)` loads in data from a .gz file
- `array2table(matrix)` converts a matrix into a table and returns the resultant table
- `fitctree(data, class)` trains a decision tree using the given data table and uses all class columns in the table to predict the given class.
- `fitcecoc(data, class)` trains an SVM model to classify data into multiple categories.
- `fitcsvm(data, class)` trains an SVM model to classify data into two categories.

Appendix B MATLAB Code

File 1 of 3

```
clear all; close all;

gunzip('train-images-idx3-ubyte.gz');
gunzip('train-labels-idx1-ubyte.gz');
gunzip('t10k-images-idx3-ubyte.gz');
gunzip('t10k-labels-idx1-ubyte.gz');
%% parse data

[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');

[testImages, testLabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');

%% reshape
reshapedImages = im2double(reshape(images, [28*28, length(images)]));
reshapedTestImages = im2double(reshape(testImages, [28*28, length(testImages)]));

%% Create Singular Value Plot
sVals = diag(S);
plot([1:50],sVals(1:50), '.', 'MarkerSize', 10);
xlabel('Index')
ylabel('Value')
title('First 50 Singular Values')

%% Determine easiest to classify and hardest to classify
easiest = [];
hardest = [];
lowestAccuracy = 100;
highestAccuracy = 0;
allAccuracy = [];

for j = 0:9
    for k = (j+1):9
        [U,S,V,threshold,w] = digit_trainer(j,k,images,labels);
        [digit1, digit2] = extract_data(j,k,testImages, testLabels);
        Test_wave = dc_wavelet([digit1, digit2]);
        TestMat = U'*Test_wave; % PCA projection
        pval = w'*TestMat;
        ResVec = (pval > threshold);
        solutions = [zeros(1, length(digit1)) ones(1, length(digit2))];
        incorrect = abs(ResVec - solutions);
        accuracy = 100 - sum(incorrect)/length(solutions)*100;
        if (accuracy > highestAccuracy)
            highestAccuracy = accuracy;
            easiest = [j,k];
        end
        if (accuracy < lowestAccuracy)
            lowestAccuracy = accuracy;
            hardest = [j,k];
        end
        allAccuracy = [allAccuracy; [j, k, accuracy]]
    end
```

```
end

%% Train Model to distinguish between 1, 0, and 8
[U,S,V,w,mFirst,mSecond,mThird] = three_digit_trainer(0, 1, 8, images, labels);
%% Classify test data
[Zeros, Ones, eights] = extract_data_3(0, 1,8,testImages, testLabels);
Test_wave = dc_wavelet([Zeros, Ones, eights]);
TestMat = U'*Test_wave; % PCA projection
pval = w'*TestMat;
ResVec = zeros(1, length(pval));
solutions = [zeros(1, length(Zeros)) ones(1, length(Ones)) 2*ones(1,length(eights))];
for j=1:length(pval)
    diff1 = abs(pval(j)-mFirst);
    diff2 = abs(pval(j)-mSecond);
    diff3 = abs(pval(j)-mThird);
    minVal = min([diff1,diff2,diff3]);
    if (minVal == diff1)
        ResVec(j) = 0;
    elseif (minVal == diff2)
        ResVec(j) = 1;
    else
        ResVec(j) = 2;
    end
end
incorrectZeros = (ResVec(1:length(Zeros)) ~= solutions(1:length(Zeros)));
incorrectOnes = (ResVec(length(Zeros)+1:length(Zeros)+length(Ones)) ~= solutions(length(Zeros)+1:length
incorrectEights = (ResVec(length(Zeros)+length(Ones)+1:length(Zeros)+length(Ones)+length(eights)) ~= so
zerosAccuracy = 100 - sum(incorrectZeros)/length(Zeros)*100;
onesAccuracy = 100 - sum(incorrectOnes)/length(Ones)*100;
eightsAccuracy = 100 - sum(incorrectEights)/length(eights)*100;
overallAccuracy = 100 - (sum(incorrectZeros) + sum(incorrectOnes) + sum(incorrectEights))/(length(Zeros

%% Create Binary Classification Tree
data = array2table([reshapedImages' labels]);
tree=fitctree(data, 'Var785');

%% Use binary tree to predict test images
testData = array2table(reshapedTestImages');
predicted = predict(tree, testData);
incorrect = (predicted ~= testLabels);
accuracy = 100 - sum(incorrect)/length(testLabels)*100;

%% Create and use binary tree to classify easiest and hardest digit pairs
[trainZeros, trainOnes] = extract_data(0,1,images, labels);
data = array2table([[trainZeros, trainOnes]; [zeros(1,length(trainZeros)), ones(1,length(trainOnes))]]'
BTree01 = fitctree(data, 'Var785');
%%
[trainThrees, trainFives] = extract_data(3,5,images, labels);
data = array2table([[trainThrees, trainFives]; [3*ones(1,length(trainThrees)), 5*ones(1,length(trainFiv
BTree35 = fitctree(data, 'Var785');
%%
[Zeros, Ones] = extract_data(0,1,testImages, testLabels);
testData = array2table([Zeros Ones]');
predicted = predict(BTree01, testData);
```

```
zeroOneLabels = [zeros(length(Zeros), 1); ones(length(Ones), 1)];
incorrect = (predicted ~= zeroOneLabels);
accuracyEasy = 100 - sum(incorrect)/length(incorrect)*100
%%
[threes, fives] = extract_data(3,5,testImages, testLabels);
testData = array2table([threes fives]');
predicted = predict(BTree35, testData);
threeFiveLabels = [3*ones(length(threes), 1); 5*ones(length(fives), 1)];
incorrect = (predicted ~= threeFiveLabels);
accuracyHard = 100 - sum(incorrect)/length(incorrect)*100

%% Create SVM classifier
data = array2table([reshapedImages' labels]);
model = fitcecoc(data, 'Var785');

%% Use SVM classifier to predict test images
testData = array2table(reshapedTestImages');
predicted = predict(model, testData);
incorrect = (predicted ~= testLabels);
accuracy = 100 - sum(incorrect)/length(testLabels)*100;

%% Create and use SVM classifier to classify easiest and hardest digit pairs
[trainZeros, trainOnes] = extract_data(0,1,images, labels);
data = array2table([[trainZeros, trainOnes]; [zeros(1,length(trainZeros)), ones(1,length(trainOnes))]]'
SVM01 = fitcsvm(data, 'Var785');
%%
[trainThrees, trainFives] = extract_data(3,5,images, labels);
data = array2table([[trainThrees, trainFives]; [3*ones(1,length(trainThrees)), 5*ones(1,length(trainFiv
SVM35 = fitcsvm(data, 'Var785');
%%
[Zeros, Ones] = extract_data(0,1,testImages, testLabels);
testData = array2table([Zeros Ones]');
predicted = predict(SVM01, testData);
zeroOneLabels = [zeros(length(Zeros), 1); ones(length(Ones), 1)];
incorrect = (predicted ~= zeroOneLabels);
accuracyEasy = 100 - sum(incorrect)/length(incorrect)*100
%%
[threes, fives] = extract_data(3,5,testImages, testLabels);
testData = array2table([threes fives]');
predicted = predict(SVM35, testData);
threeFiveLabels = [3*ones(length(threes), 1); 5*ones(length(fives), 1)];
incorrect = (predicted ~= threeFiveLabels);
accuracyHard = 100 - sum(incorrect)/length(incorrect)*100

%% Trainer function
function [U,S,V,threshold,w] = digit_trainer(dOne, dTwo, images, labels)
[firstDigit, secondDigit] = extract_data(dOne, dTwo, images, labels);
digits = dc_wavelet([firstDigit, secondDigit]);
features = 30;
[U,S,V] = svd(digits, 'econ');
U = U(:,1:features);
numFirst = size(firstDigit, 2);
numSecond = size(secondDigit, 2);
digitsProj = S*V';
```

```
firstProj = digitsProj(1:features, 1:numFirst);
secondProj = digitsProj(1:features, numFirst+1:numFirst + numSecond);
mFirst = mean(firstProj,2);
mSecond = mean(secondProj,2);
Sw = 0; % within class variances
for k = 1:numFirst
    Sw = Sw + (firstProj(:,k) - mFirst)*(firstProj(:,k) - mFirst)';
end
for k = 1:numSecond
    Sw =  Sw + (secondProj(:,k) - mSecond)*(secondProj(:,k) - mSecond)';
end
Sb = (mFirst-mSecond)*(mFirst-mSecond)'; % between class
[V2,D] = eig(Sb,Sw);
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
vFirst = w'*firstProj;
vSecond = w'*secondProj;
if mean(vFirst) > mean(vSecond)
    w = -w;
    vFirst = -vFirst;
    vSecond = -vSecond;
end
sortFirst = sort(vFirst);
sortSecond = sort(vSecond);
t1 = length(sortFirst);
t2 = 1;
while sortFirst(t1) > sortSecond(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold = (sortFirst(t1) + sortSecond(t2))/2;
end

%% Trainer function for 3 digits
function [U,S,V,w,mFirst,mSecond,mThird] = three_digit_trainer(dOne, dTwo, dThree, images, labels)
[firstDigit, secondDigit, thirdDigit] = extract_data_3(dOne, dTwo, dThree, images, labels);
digits = dc_wavelet([firstDigit, secondDigit, thirdDigit]);
features = 30;
[U,S,V] = svd(digits, 'econ');
U = U(:,1:features);
numFirst = size(firstDigit, 2);
numSecond = size(secondDigit, 2);
numThird = size(thirdDigit, 2);
digitsProj = S*V';
firstProj = digitsProj(1:features, 1:numFirst);
secondProj = digitsProj(1:features, numFirst+1:numFirst + numSecond);
thirdProj = digitsProj(1:features, numFirst + numSecond+1:numFirst + numSecond + numThird);
mFirst = mean(firstProj,2);
mSecond = mean(secondProj,2);
mThird = mean(thirdProj, 2);
Sw = 0; % within class variances
for k = 1:numFirst
    Sw = Sw + (firstProj(:,k) - mFirst)*(firstProj(:,k) - mFirst)';
```

```
end
for k = 1:numSecond
    Sw =  Sw + (secondProj(:,k) - mSecond)*(secondProj(:,k) - mSecond)';
end
for k = 1:numThird
    Sw =  Sw + (thirdProj(:,k) - mThird)*(thirdProj(:,k) - mThird)';
end
Sb = (mFirst-mSecond)*(mFirst-mSecond)' + (mFirst-mThird)*(mFirst-mThird)' + (mSecond-mThird)*(mSecond-m
[V2,D] = eig(Sb,Sw);
[lambda,ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
vFirst = w'*firstProj;
vSecond = w'*secondProj;
vThird = w'*thirdProj;
mFirst = mean(vFirst);
mSecond = mean(vSecond);
mThird = mean(vThird);
end
%% data extraction function

function [mat1, mat2] = extract_data(val1, val2, mat, labels)
    reshapedMat = reshape(mat, 28*28, length(mat));
    mat1 = [];
    mat2 = [];
    for j = 1: length(reshapedMat)
        if (labels(j) == val1)
            mat1 = [mat1, reshapedMat(:,j)];
        elseif (labels(j) == val2)
            mat2 = [mat2, reshapedMat(:,j)];
        end
    end
    mat1 = im2double(mat1);
    mat2 = im2double(mat2);
end

%% 3 digit data extraction function

function [mat1, mat2, mat3] = extract_data_3(val1, val2, val3, mat, labels)
    reshapedMat = reshape(mat, 28*28, length(mat));
    mat1 = [];
    mat2 = [];
    mat3 = [];
    for j = 1: length(reshapedMat)
        if (labels(j) == val1)
            mat1 = [mat1, reshapedMat(:,j)];
        elseif (labels(j) == val2)
            mat2 = [mat2, reshapedMat(:,j)];
        elseif (labels(j) == val3)
            mat3 = [mat3, reshapedMat(:,j)];
        end
    end
    mat1 = im2double(mat1);
    mat2 = im2double(mat2);
```

```
    mat3 = im2double(mat3);
end
```

File 2 of 3

```
function dcData = dc_wavelet(dcfile)

    [m,n] = size(dcfile); % 4096 x 80
    pxl = sqrt(m);
    nw = m/4; % wavelet resolution
    dcData = zeros(nw,n);

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

File 3 of 3

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

% 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-

% 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
fprintf('Magic Number - Images: %d\n', magicNumber1);

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

% 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
fprintf('Total number of images: %d\n', totalImages);

% 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));

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

% 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
```