# Contents



# Contents

UNIVERSITY OF CALIFORNIA, DAVIS

# MAT 167 Programming Project Writeup

Sneha Patrachari 999217441

May 29, 2016

# 1 INTRODUCTION

Often times, we find it difficult to automate machines to behave like humans. Over the years, developments have been made to advance these efforts. Machine learning was derived from these interests, whose primary use is data analysis. It utilizes algorithms that dynamically learn to better predict the generated data. This field allows a great deal of flexibility, since computer functionality is no longer dependent upon static program instructions (<www.sas.com>).

One practical application of machine learning is pattern recognition–an example we implemented in our code. Pattern recognition seeks to perform computations to yield higher recognition rates for the visual patterns of interest. We wish to do the same in our own example. Our primary goal is to provide instructions for the computer to iteratively learn and classify handwritten digits. Many approaches exist in formulating solutions for this classification. In this paper, we will discuss two specific approaches we took, and go into further analysis and applications of this algorithm (Elden).

# 2 DATA SET

Our algorithm uses a data set of handwritten digits from the U.S. Postal Service Database. This file we are using consists of four arrays, two of which will comprise the training data and the other two of the test data. Both the arrays trainpatterns and testpatterns are image reconstructions represented by 16x16 pixel intensities, which we will manipulate and perform our operation on. The true information about the images will be in the two arrays labeled trainlabels and testlabels, which we will use to compare our computations and predictions against.

# 3 METHODS

## 3.1 STEP 1

We creates a matrix whose values are all zeros.Matrix dimensions are used as the parameter to create matrix of size 256x1. We then transform the given trainpatterns data into one vector. Let us continue by parsing through sixteen images in the given trainpatterns array and set these values to variable v. We may then display the first sixteen images of the trainpatterns array. However, we must reshape the into a square matrix with dimensions 16x16,followed by taking the transpose, to ensure the cluster is well-formed.

## 3.2 STEP 2

Here in step 2, we create a matrix whose values are all zeros. The matrix dimensions are used as the parameter to create matrix of size 256x10. Once we begin looping through all the image, we fill the matrix trainaves with the computed centroids(means) for each of the ten pooled images. We continue by displaying the 10 mean digit images from the trainaves matrix that was recently computed. Similar to step 1, we must reshape and transpose array to a 16x16 matrix, before displaying the images, so that the dimensions are aligned to produced undistorted images. Now, we finish by displaying this matrix to produce the 10 mean images.

### 3.3 STEP 3

Similar to the other steps, we begin by creating a matrix of size 10x4649, whose values are all zeros. Let us create an additional matrix again with all zero values, but this time of size 1x4649. Now, we compute the distances to the j cluster set for all the test images at once. We use the repmap command to get the matrices to be of the same dimension before we are able to perform operations on them. We create two separate for loops, first by looping over all the cluster centers, and a second loop to cover all the digit images. Here, we will perform a comparison on which distance in each column has the smallest distance to the cluster center. The ind variable we use indicates which row contains the smallest distance, and is offset by 1 because of MATLAB's default indexing. Now, we form to nested for loops. The outer loop will be used to extract the guesses for every zero truth value. The inner for loop will be used for the zeros in the truth column, as obtained by comparing against our predicted corresponding value. The algorithm continues by counting the occurrences of each digit appearing in tmp and fills every position in the confusion matrix by placing the correct predictions on the main diagonal of the confusion matrix.

### 3.4 STEP 4

We begin by creating matrices to hold both two dimensional and three dimensional arrays. Our first step would be to iterate over each kth digit of trainpatterns to compute the rank 17 SVD of a set of images. For each iteration (kth image), these left singular vectors placed into the array trainu. We proceed by using this matrix to compute the expansion coefficients of each test digit image with respect to the 17 singular vectors of each train digit image set. The rank 17 approximations of test images using the 17 left singular vectors are then computed. We further continue by computing the error between each original test image and its rank 17 approximation.

We continue our algorithm by looping over all the test images and again perform comparisons on which distance in each column has the smallest distance to the cluster center, similar to step 3. Again, we perform a comparison against the predicted corresponding values. Now, we compute the confusion matrix using the SVD classification method. The confusion matrix is created to enable us to make comparisons between our predicted and actual results, in this case between our training and test data. The correct predictions are place on the main diagonal, just as the values off the main diagonal are incorrect predictions. This method yields larger numbers on the main diagonal, as compared to the confusion matrix computed using our first method. Hence, why the SVD is more accurate.

## 4 ANALYSIS

We will conduct an an analysis of the code. The subsections are broken down by the analysis required in Step 5 of the Programming Project.

### 4.1 PART A

By representing the digit of the images as matrices, we use a two dimensional array as our primary data structure. One dimension arrays are also used, although not as common. They are used to store the information as vectors, allowing for easy search and find access. As we begin to advance our computations, we observe the use of three dimensional arrays as well, which allows us to traverse the columns and rows of an image, and which digit that is currently of interest.

Our algorithm uses two data sets, the training data and the test data. Both data sources serve different purposes. The training data will be used to perform our operations upon. We will compute the predictions using various approaches, and use the test data to compare the accuracy of these results.

## 4.2 PART B

Within this step, we compute the means of the digit images. Our algorithm begins by performing this computation, since this calculation is important in showing whether a cluster of points within an image is well-formed and well separated. If these conditions remain unsatisfied, then the resulting images would no longer be as easily distinguishable to make predictions on. By parsing every image and pooling them together, we are able to compute the centroids of the images within one command.

$$trainaves(:,k) = mean(trainpatterns(:,trainlabels(k,:) == 1),2); \tag{4.1}$$

The images displayed from our computed means show well formed images, which allows for a feasible classification using our algorithm.

## 4.3 PART C

The original approach we used was computing the means and comparing the distances to the cluster centers of each digit image. This classified the digits and make our predictions. Unfortunately, with this method the variation within each digit image is not taken into account, which is what is responsible for generating a wide range of correct and incorrect predictions. The percentage of each identified correctly reflects this, because it yields a 84.6634%.

The second approach we used was the SVD. The flaws from the original approach are addressed in this approach, since the variation for each digit image is taken into account. This algorithm performs with a high success rate of 95.5474%. By finding an orthogonal basis within one of the digit images, we are able to use this to represent the variation present within that image.

The image that is easiest to classify, considering both instances of confusion matrices, is the digit image 0. In both cases, this value had the highest number of true positive predictions.
From Steps 4a-d we use the Singular Value Decomposition bases as a different classification method. It models around the variation that is present within each digit image. The theory involves computing the orthogonal basis of an image to represent its variation. By assuming that the images are mostly well formed, we can continue by calculating the rank 17 SVD for each digit image. This stacks all the images the represent a version of the image seen in the image of a digit Now, to significantly reduce the error of our data, we compute the approximation and then proceed by using this to compute our confusion matrix.

## 4.4 PART D

Step 4 yield better predictions. By using the SVD, we assume we are working under additional conditions that make the generated more precise and closer to the actual value.Since the Singular Value decompositions identically handles the rows and columns of a matrix, the relevant information of a matrix is typically generated. Often times, the approaches we use to solve the least squares problem reflect neglecting either a row space or column space. This approach preserves and yields more information after performing operations upon the matrix, which is why it is more likely to generate better predictions.

On a more concrete basis, we see that the percentages between steps 3 and 4 reflect this difference as well. The accuracy in step 3 defined by 84.6634% cannot out perform 95.5474% percent from step 4.

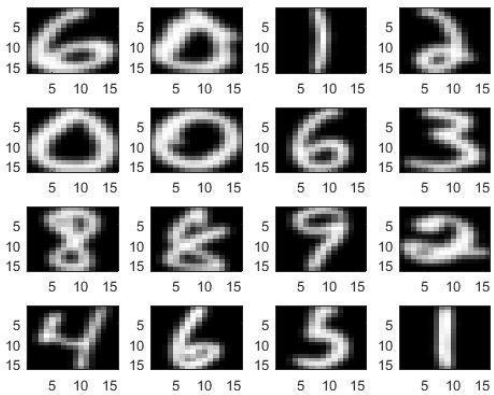
## 5 Figures and Matrices

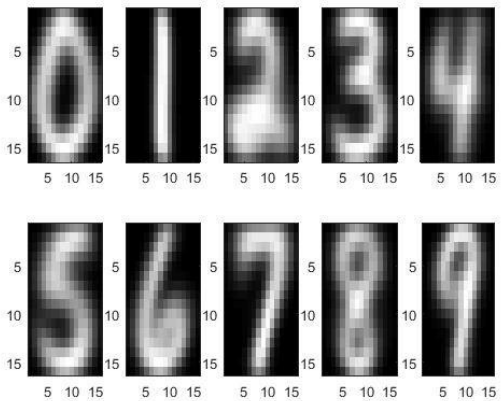
$testconfusion =$

$$\begin{bmatrix} 656 & 1 & 3 & 4 & 10 & 19 & 73 & 2 & 17 & 1 \\ 0 & 644 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 14 & 4 & 362 & 13 & 25 & 5 & 4 & 9 & 18 & 0 \\ 1 & 3 & 4 & 368 & 1 & 17 & 0 & 3 & 14 & 7 \\ 3 & 16 & 6 & 0 & 363 & 1 & 8 & 1 & 5 & 40 \\ 13 & 3 & 3 & 20 & 14 & 271 & 9 & 0 & 16 & 6 \\ 23 & 11 & 13 & 0 & 9 & 3 & 354 & 0 & 1 & 0 \\ 0 & 5 & 1 & 0 & 7 & 1 & 0 & 351 & 3 & 34 \\ 9 & 19 & 5 & 12 & 6 & 6 & 0 & 1 & 253 & 20 \\ 1 & 15 & 0 & 1 & 39 & 2 & 0 & 24 & 3 & 314 \end{bmatrix}$$

$testsvd17confusion =$

$$\begin{bmatrix} 772 & 0 & 3 & 1 & 2 & 2 & 6 & 0 & 2 & 0 \\ 2 & 646 & 6 & 1 & 8 & 0 & 4 & 2 & 9 & 5 \\ 1 & 0 & 431 & 4 & 1 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 6 & 401 & 0 & 5 & 0 & 0 & 5 & 1 \\ 1 & 0 & 0 & 0 & 424 & 2 & 2 & 2 & 1 & 0 \\ 1 & 0 & 3 & 7 & 1 & 335 & 3 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 & 7 & 399 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 5 & 1 & 0 & 387 & 0 & 4 \\ 3 & 0 & 2 & 4 & 0 & 1 & 0 & 0 & 309 & 1 \\ 0 & 1 & 0 & 0 & 1 & 2 & 0 & 11 & 3 & 388 \end{bmatrix}$$





# 6 CODE

```
v = zeros(256,1);
v = trainpatterns(:,1)
for k=1:16 do
    v = trainpatterns(:,k);
    subplot(4,4,k)
    u = reshape(v,[16,16])'
    imagesc(u)
    colormap(gray);
end
trainaves = zeros(256,10);
for k = 1:10 do
    trainaves(:,k) = mean(trainpatterns(:, trainlabels(k,:)==1),2);
    subplot(2,5,k)
    s = trainaves(:,k);
    temp = reshape(s,[16,16])'
    imagesc(temp)
end
testclassif = zeros(10,4649);
testclassifres = zeros(1,4649);
for j = 1:10 do
    testclassif(j,:) = sum((testpatterns - repmat(trainaves(:,j),[1 4649])).^2);
end
for i=1:4649 do
    [tmp, ind] = min(testclassif(:,i));
    testclassifres(:,i) = ind-1;
end
testconfusion = zeros(10,10);
for k =1:10 do
    tmp= testclassifres(testlabels(k,:)==1);
    for m = 1:10 do
        for n = 1:length(tmp) do
            if tmp(n) == m - 1 then
                testconfusion(k,m) = testconfusion(k,m)+1
            end
        end
    end
end
```

```
trainu = zeros(256,17,10);
for k = 1:10 do
    [trainu(:,:,k),tmp,tmp2] = svds(trainpatterns(:,trainlabels(k,:)==1),17);
end
testsvd17 = zeros(17,4649,10);
for k = 1:10 do
    testsvd17(:,:,k) = trainu(:,:,k)' * testpatterns;
end
approxmatrix = zeros(256,4649,10);
testsvd17res = zeros(10,4649);
for k = 1:10 do
    approxmatrix(:,:,k) = trainu(:,:,k) * testsvd17(:,:,k);
    testsvd17res(k,:) = sum(testpatterns - approxmatrix(:,:,k).^2);
end
testclassifsvd = zeros(1,4649);
for i=1:4649 do
    [tmp, ind] = min(testsvd17res(:,i));
    testclassifsvd(:,i) = ind;
end
testsvd17confusion = zeros(10,10);
for k do
    =
end
1:10
tmp= testclassifsvd(testlabels(k,:)==1);
for m = 1:length(tmp) do
    temptwo = tmp(1,m);
    testsvd17confusion(temptwo,k) = testsvd17confusion(temptwo,k)+1
end
```