Katherine Olson

kcolson@ucdavis.edu

998992204

ECS 130 Final Project – Classification of Digits

Abstract

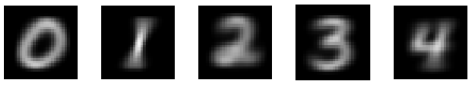
Comparing the centroid method and the PCA method for classifying digits, the PCA method does significantly better. As the number of singular vectors used in the PCA method increases, so does the classification rate, but so does the computation that is needed. Using 4 to 8 singular vectors has the best balance between the classification rate and how computationally expensive the method is.

Introduction

This report is tackling the issue of classifying images of handwritten digits through a computer without the use of the human eye. This issue is very important to the post office. The more accurately this problem can be solved, the easier and more efficiently they can get out mail. This report will be looking into two main methods and how their classification rates compare. Both methods use a training set and a test set. The test set contains images we want to make digit predictions for. The training set is a set of images of which we know the true digits for. This set is used to make predictions for the test set. Here are some example testing digits on the left:

Testing Digits

Average for Each Digit



The first method is the centroid method which uses the average of an image. This method computes the average for each digit's training set and predicts each test set's digit to be the digit whose average is closest to that image's average. The average for each digit from the training set can be seen above on the right.

The second method is the PCA or Principal Component Analysis method. This method models the variation of the training set using SVD or Singular Value Decomposition. For each digit's training set, the SVD is calculated and the first five (or any number, this is the basis length) orthogonal singular vectors are used for prediction. These five singular vectors are the five most dominant, so they characterize the data well. Then for prediction, a residual is calculated using a least squares approach for each test image comparing it to the ten sets of singular vectors for each digit. The digit from the training set with the smallest residual is chosen as the prediction. Here are the first five bases for the digit 4:

Related definitions, concepts and theory

A. 2-norm for a matrix M

   $\left|\left|M\right|\right|_2 = \max_{x\neq 0} \frac{\left|\left|Mx\right|\right|_2}{\left|\left|x\right|\right|_2}$ = the largest singular value of M.

   This norm allows a way to measure which training set is closest to the image we want to predict a digit for.

B. SVD

   If M is m by n with m ≥ n then $M = U\Sigma V^T$ where:
   U is an m by n orthogonal matrix where the columns are left singular vectors.
   V is an n by n orthogonal matrix where the columns are right singular vectors.
   Σ is diag(σ_1, ..., σ_n) where $\sigma_1 \geq ... \geq \sigma_n$ are the singular values.
   In PCA, using SVD allows us to form a basis for each digit containing the most dominant singular vectors. This basis represents the characteristics of each digit and takes into account the variation.

C. Least Squares

   In PCA, we are solving the least squares problem $\min_\beta \left|\left|z - U_{digit}\,\beta\right|\right|_2$ which has a solution of $\left|\left|z - U_{digit}\,U_{digit}^T\,z\right|\right|_2$ where the norm definition above comes in.

Algorithms

1. **Centroid** – Pseudocode:
   Calculate the mean of each training set and store it in a column of TestAvg.

For each digit:
    For each row from that digit's testing set:
        Computes its 2-norm with each training set using TestAvg.
        Find the smallest norm and use the digit of that training set as the prediction.

    End
    Find the classification rate for that digit.
End

2. **PCA** – Pseudocode:

Pick a basis length.
Create a 28^2 by basis length by 10 matrix of zeros called Us.
For each digit:
    Grab the training set for that digit and convert it to double precision, call it A.
    Get the (basis length) largest singular values of A^T and grab the U matrix from that decomposition that contains the left singular vectors in its columns. Put that U in the third dimension of the Us matrix at the index of digit + 1.
End

For each digit:
    For each column in the testing set for that digit:
        Convert it to double precision and take the transpose, call it z.
        For each digit:
            Take the norm of z minus that digit's matrix in Us times the product of that digits matrix in Us transposed and z.
        End
        Take the smallest norm and use the digit it corresponded to as the prediction.
    End
    Compute the classification rate for the predictions.
End

Functions Used

1. Built into MATLAB:
    a) [_, _, _] = svds(_, _) gives the singular value decomposition.
    b) double() converts to double precision.
    c) cell() to loop over the testing and training matrices.
    d) norm() calculates the 2-norm.
    e) sum(), length(), mean(), zeros(), and size().
2. Built by me:

a) PCA
   Input: an n-by-784 matrix containing n digits that we want to predict and a 784-by-5-by-10 matrix containing the first basis_len singular vectors for each of the training sets
   Output: an n-by-1 vector containing the predictions (0-9).
b) Centroid
   Input: a10-by-784 matrix containing the average for each digit from the training set and an n-by-784 matrix containing n digits that we want to predict.
   Output: an n-by-1 vector containing the predictions (0-9).

Experiment Results

The data used in this report is from the U.S. Postal Service database. The numbers were taken off of envelopes and they are from zip codes. Each of the ten test sets range from having 892 to 1,235 images. Each of the ten training sets range from having 5,421 to 6,742 images.

Table I: Comparing Classification Rates for Each Method

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
C	0.8959	0.9621	0.7568	0.8059	0.8259	0.6861	0.8633	0.8327	0.7372	0.8067
PCA	0.9786	0.9921	0.9021	0.9376	0.8982	0.9013	0.9624	0.8930	0.9004	0.8890

Note: C = Centroid and PCA has a basis length of 5

From table I we can see that PCA with a basis length of five does a much better job than the centroid method for all digits. PCA is more computationally expensive than the centroid method, but in this case it is worth it since PCA does significantly better. 1's have the best classification rate for both methods, which makes sense consider how it is such a simple number most often written with just one vertical line. 0's and 6's have the next best classification rates. This also makes sense considering how there is really only one way to write each digit. 3's are in the middle. There is one way to write a 3, but there is probably a lot of variation with angle and loop size. The rest of the digits come in last. For 2's, 4's and 7's, this is most likely because of the different ways each digit can be written. A two is commonly written as 2 or  , a four can be written like 4 or  , and a seven is often written as 7 or  . 5's, 8's, and 9's can have a lot of variation in angle, loop size and how closed each loop is.

Table II: Comparing Basis Lengths for PCA:

Basis#	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0.9204	0.9374	0.7471	0.8337	0.7974	0.6491	0.8841	0.8191	0.7639	0.8028
2	0.9347	0.9833	0.8450	0.8624	0.8014	0.7937	0.9301	0.8473	0.8326	0.8652
4	0.9755	0.9921	0.8847	0.9030	0.8809	0.8957	0.9551	0.8891	0.8871	0.8821
6	0.9796	0.9956	0.9118	0.9337	0.9318	0.9002	0.9666	0.9125	0.9107	0.8930
8	0.9857	0.9947	0.9244	0.9366	0.9440	0.9081	0.9656	0.9212	0.9168	0.9158
10	0.9878	0.9947	0.9273	0.9426	0.9562	0.9193	0.9656	0.9339	0.9179	0.9326

For table II, I picked the same basis lengths to look into as the book did. As the number of SVD bases used increases, so does the classification rate. The biggest increase seems to happen from 2 to 4 and then from 4 to 6. The increase from 6 to 8 to 10 is much smaller, so I would stick with using 4 to 6 bases, maybe 8. The more bases used, the more we are over fitting our data and the more computation is needed. Within each basis length, the order of digits with the highest classification rate stays about the same.

Conclusion

With the classifying digits problem, comparing the centroid method to the PCA method, the PCA method does a much better job. The classification rate varies by digit and both methods encounter the same challenges each digit has to offer with the common ways each one can be written. This result is not surprising considering how the centroid method only uses the average of the training set while the PCA method considers the variation of the training set. Neither method is perfect, but that is to be expected with how differently people write numbers. It is probably for the best that the better method is not perfect, because we would then be over fitting our prediction model to the training set and considering every possible way each digit can be written would be way too computationally expensive.

References

The textbook: Numerical Computing with MATLAB (revised reprint) by C. Moler, SIAM, 2008

The project description with code found on the class website titled Class final project: Classification of Handwritten Digits

Code Appendix

Main Script:

```
% ECS130 Winter 2017 Final Project: Classifying Digits
clear;
load mnistdata;

% Creating cell array of all training and testing sets to loop over later:
trains = cell(1, 10);
trains{1} = train0; trains{2} = train1; trains{3} = train2; trains{4} =
train3;
trains{5} = train4; trains{6} = train5; trains{7} = train6; trains{8} =
train7;
trains{9} = train8; trains{10} = train9;

tests = cell(1, 10);
tests{1} = test0; tests{2} = test1; tests{3} = test2; tests{4} = test3;
tests{5} = test4; tests{6} = test5; tests{7} = test6; tests{8} = test7;
tests{9} = test8; tests{10} = test9;

% Plot some example test digits:
```

```
figure(3)
for i = 1:10
    subplot(2,5,i)
    image(rot90(flipud(reshape(tests{i}(1, :),28,28)),-1));
    colormap(gray(256));
    axis square tight off;
end

%% The Centroid Method
% Get the avg for each digit from the training set:
for i = 1:10
    TrainAvg(i, :) = mean(trains{i});
end

% Get the classification rate for each
for i = 1:10
    predictions = Centroid(TrainAvg, tests{i});
    classRatesC(i) = sum(predictions == (i - 1)) / length(predictions);
end
% Plot the average for each digit:
figure(4)
for i = 1:10
    subplot(2,5,i)
    image(rot90(flipud(reshape(TrainAvg(i, :),28,28)),-1));
    colormap(gray(256));
    axis square tight off;
end

%% The PCA Method

basis_len = 5;
Us = zeros( 28*28, basis_len, 10);
for k = 1:10
    % go through each digit 0 to 9
    A = double(trains{k});

    % and get first 5 singular vector of A transposed
    [U,~,~] = svds( A', basis_len );
    Us(:,:,k)=U;
end

% Plot the bases for 4:
figure(5)
for i = 1:basis_len
    subplot(1,5,i)
    a1 = reshape(Us(:, i, 5),28,28)
    a1=(a1-min(min(a1))*ones(size(a1)));
    a1=(256/max(max(a1)))*a1;
    digitImage=256*ones(size(a1)) - a1;
    image(rot90(flipud(digitImage),-1));
    colormap(gray(256));
    axis square tight off;
end

% Get the classification rate for each
for i = 1:10
```

```
    predictions2 = PCA(tests{i}, Us);
    classRatesPCA(i) = sum(predictions2 == (i - 1)) / length(predictions2);
end
```

Centroid Function:

```
function [ predictions ] = Centroid( TrainAvg,  testing)
    % Inputs:
        % TrainAvg = a 10-by-784 matrix containing the average train digits
        % testing = an n-by-784 matrix containing n digits that we want to
      make predictions for
    % Output: an n-by-1 vector containing the labels (0-9) for digits in
        % Testing

    [n, ~] = size(testing);
    predictions = zeros(n, 1);% empty n by 1

    for i = 1:n % Each column in testing
        test = double(testing(i,:)); % Grab one to predict from test set
        compareAvg = zeros(10,1); % Empty 10 columns

        % Compare the avg of what we are testing to each avg of training per
        digit
        for digit = 1:10
            compareAvg(digit) = norm( test - TrainAvg(digit,:) );
        end

        % Want the one with the smallest distance as our prediciton ( - 1
        because
        % matlab indexing starts at 1)
        yhat = find(compareAvg == min(compareAvg)) - 1;
        predictions(i) = yhat;
    end

end
```

PCA Function:

```
function [ predictions ] = PCA( testing, Us )
    % Input:
        % testing = an n-by-784 matrix A containing n digits that we want to
        make predictions for
        % Us = a 784-by-5-by-10 matrix containing the first 5 singular
        vectors for each of the training sets (train0', train1', train2' ...,
        train9')

    % Output: an n-by-1 vector containing the labels (0-9) for digits in
        % Testing

    [n, ~] = size(testing);
    predictions = zeros(n, 1);% empty n by 1

    for i = 1:n % Each column in testing
        z = double(testing(i, :))';
```

```
        dist = zeros(10, 1);
        for k = 1:10
            Uk = Us(:, :, k);
            dist(k) = norm( z - Uk*(Uk'*z) );
        end
        yhat = find(dist == min(dist)) - 1;
        predictions(i) = yhat;
    end

end
```