Handwritten Digit Recognition Use Machine Learning

Zihe (Tyler) Deng
Electrical Engineering student
University of California, San Diego

Notice: This project is only an application of the Machine Learning theories and methods that I learned in an upper division course (ECE 175) at UC San Diego. Therefore, the results may not be the state-of-the-art results. Please let me know if anything has been mistyped, inaccurate, or unprofessional and need to be improved. Thank you!

About this project:

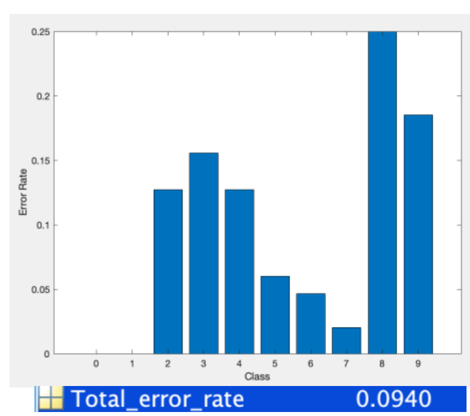
For this project, I've programmed different Machine Learning algorithms from scratch in MATLAB to minimized error rate and therefore to improve classification accuracy. Datasets include a **Training set**, which contains 5000 images of handwritten random digits, and a **Test set** which has 500 images of handwritten random digits that used to test the accuracies of the algorithms. The labels for both datasets are provided. Digits can be classified as ten classes represent digits from "0" to "9" respectively. For supervised learning, we used all 5000 training images for training and then used the 500 test images for testing and error rate analyses. Examples of handwritten samples are shown below.

<u>KNN Classifier:</u>

In this part, I implemented the NN classifier by finding the closest Euclidean distance between each test image and all training images, then label the test image as the same class with the training image which closest to it. The Euclidean distance can be described as **Eq. 1**.

$$\boldsymbol{d(X,Y) = \|X - Y\| = \sqrt{\sum_{l=allpixels} |X(l) - Y(l)|^2}} \qquad \boldsymbol{Eq.1}$$

After repeated the above step for all 500 test images, I obtained prediction classes for all test images. Compare the result with the actual label, I obtained the **error rate of 9.4%** and error rates for each class show in the plot to the right. This is an implementation of KNN where K = 1 using Euclidean distance as its matric. The result error rate is high specifically on handwritten digits eight and nine. One of the reason is that in some of the handwriting style, digits eight and nine are lot alike other digits, and this has misled the classifier to predicted the wrong class.



Also, when calculating the summation of pixel distances, it is possible that two different digits have the shortest distance because they share the most features (patterns) even may be two completely different digits.

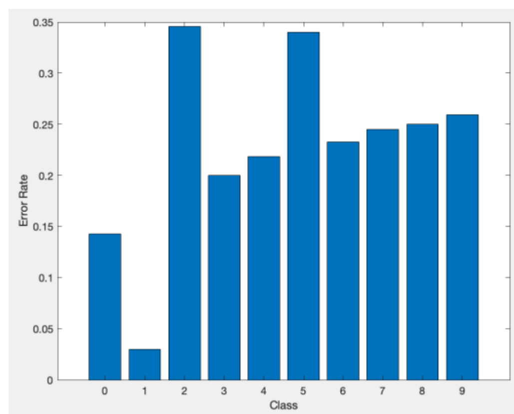
<u>Bayes Decision Rule (BDR):</u>

In this method, assuming Gaussian distribution for all classes, I followed the Bayes Decision Rule to obtain the optimal decision (classification) which minimized the lost function at the same time. After calculation, I obtain **Eq. 2** which is used to find the I (class, from 0 to 9) that maximum the argument inside the "{}".

$$\boldsymbol{i^*(x) = argmax_i \left\{ -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) - \frac{1}{2}\log(2\pi)^d\,|\Sigma_i| + \log P_Y(i) \right\}} \qquad \boldsymbol{Eq.2}$$

Where x is each of the training images, μ_i is the class mean, and I assume, for now, the Σ_i covariance is identity. I used all 5000 training images to calculate and obtain the classmeans of each class (shown in the plot to the left). Then I used **Eq. 2** to pick the class (i^*) that maximized the argument and used it as the target test image's prediction.

By used BDR only, I yield an **error rate of 22.2%** with the error rate for each class show in the plot to the right.

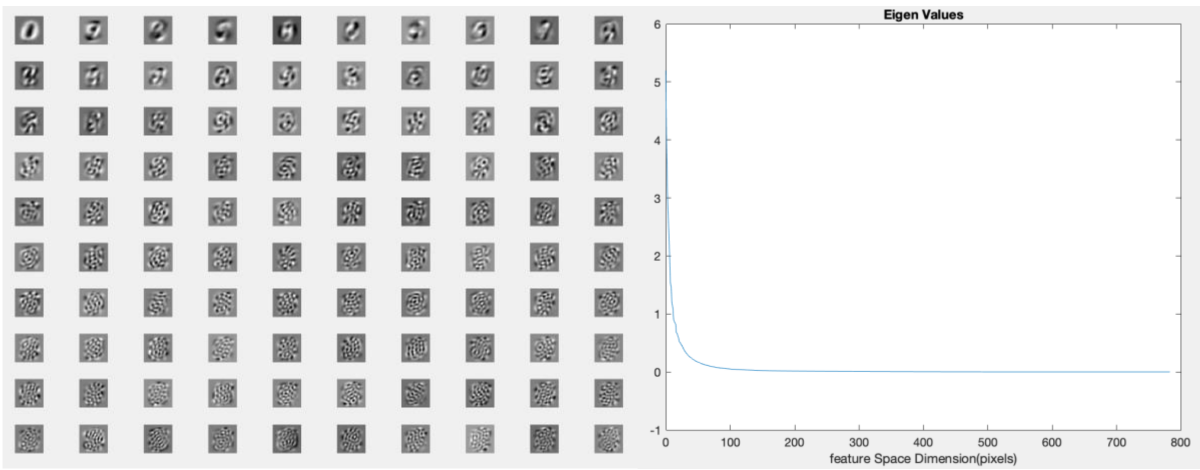


total_error_rate 0.2220

Principle Component Analysis (PCA):

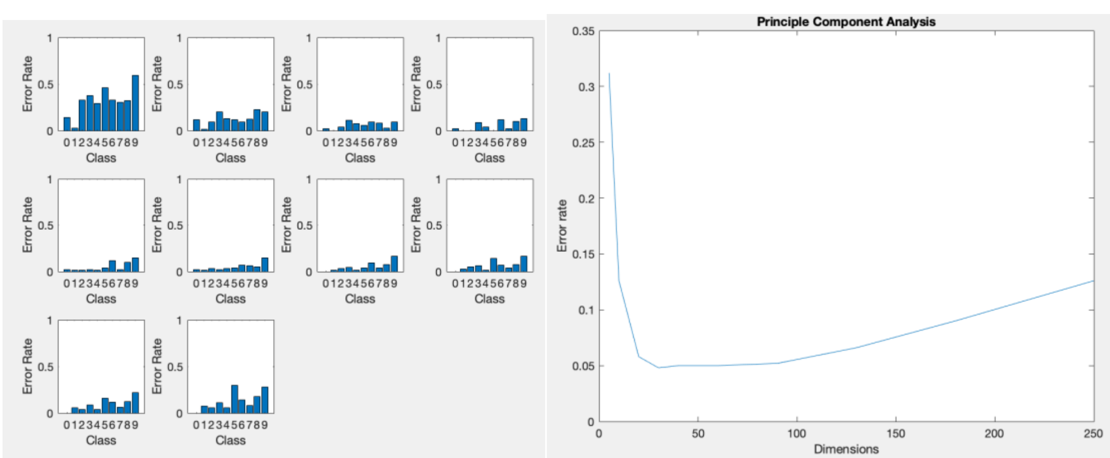
Things have always been weird in higher dimensions. Beyond certain point, higher dimensions will only bring in noise without providing any useful feathers. Also, for supervised learning, the amount of training sets needed to maintain classification accuracy growth exponentially with the dimension of the feature space. Therefore, it is very useful to apply dimensionality reduction techniques such as PCA to avoid unnecessary dimensions. This will reduce the need for training samples hugely while remains (or potentially improve) the accuracy.

The plots below are the first 100 feature space as 28x28 images (left plot), and the Eigen Values of each feature space (pixels) in descending order (right plot):



As we can see, the most useful feature spaces are roughly in the first 100 feature spaces out of 784 (28x28=784). So far, I've been assuming the covariance in BDR's argument is identity convince, for this part, however, I will take the actual covariance for each class into consideration.

Now, to see how many dimensions (Eigen Vectors) to use to obtain the best result, let's redo BDR again with all datasets projected into lower dimensions of 5, 10, 20, 30, 40, 60, 90, 130, 180, and 250. I yield the following result plots:



The plots on the above showed the error rate for each class and total error rate used dimensions of 5, 10, 20, 30, 40, 60, 90, 130, 180, and 250 respectively. As we can see, we obtained the lowest total **error rate of 4.8%** by using the **first 30 dimensions** (the table below represents the total error for each dimension, and it also approved this conclusion).

total_error_rate

1x10 double

	1	2	3	4	5	6	7	8	9	10
1	0.3120	0.1260	0.0580	0.0480	0.0500	0.0500	0.0520	0.0660	0.0900	0.1260

Conclusion:

In this project, we obtained the error rate as low as 4.8% by applied **Principle Component Analysis** to the whole datasets and reduced datasets dimension by projecting them into a lower subspace, then applied unsupervised learning and future analysis on this reduced dimension dataset. Also from the plots, we observed that when we increase the dimension of the dataset, the error rate dropped rapidly until certain dimension, and the model started to pick up noise once we pass that dimension. Therefore, it is critical to choose the subspace for dataset to be projected on.

Appendix

Codes for KNN:

```
clear all;
close all;
clc
load 'data.mat';
load 'label.mat';

% distance between TestData and TrainData:
EDistance = zeros(500,5003);
small = zeros(1,2); % value and index
r_rate = zeros(500, 10); % for error_rate calculation
for i = 1:500
    for j = 1:5000
        difference = (imageTrain(:,:,j) - imageTest(:,:,i)).^2;
        summation = sum(difference(:));
        EDistance(i,j) = sqrt(summation);
        if j == 1
            small(1) = EDistance(i,1);
            small(2) = 1;
        end
        if EDistance(i,j) < small(1)
            small(1) = EDistance(i,j);
            small(2) = j;
        end
    end
    EDistance(i,5001) = small(1); % predicted value
    EDistance(i,5002) = small(2); % predicted index
    EDistance(i,5003) = labelTrain(small(2)); % predicted class

    for k = 1:9
        if labelTest(i) == k
            if EDistance(i,5003) == labelTest(i)
                r_rate(i,k) = 1;
            end
        end
    end
    if labelTest(i) == 0
        if EDistance(i,5003) == labelTest(i)
            r_rate(i,10) = 1;
        end
    end
end
error_rate = zeros(10,3); %error rate, class_total#, and class_correct

% Error_rate per class:
for l = 1:9
    error_rate(l+1,1) = 1 - ((sum(r_rate(:,l) == 1)) / sum(labelTest(:) == l));
    error_rate(l+1,3) = sum(labelTest(:) == l);
    error_rate(l+1,2) = sum(r_rate(:,l) == 1);
end
error_rate(1,1) = 1 - ((sum(r_rate(:,10) == 1)) / sum(labelTest(:) == 0));
error_rate(1,3) = sum(labelTest(:) == 0);
error_rate(1,2) = sum(r_rate(:,10) == 1);

% Plot:
index = (0:1:9);
bar(index,error_rate(:,1));
xlabel('Class');
ylabel('Error Rate');

%Error_Table:
Class = {'0';'1';'2';'3';'4';'5';'6';'7';'8';'9'};
Correctly_classified = error_rate(:,2);
Incorrectly_classified = error_rate(:,3) - error_rate(:,2);
Total_num_of_images = error_rate(:,3);
Error_rate = error_rate(:,1);
T = table(Class,Total_num_of_images,Correctly_classified,Incorrectly_classified,Error_rate);

% Total Error_rate:
Total_error_rate = (sum(EDistance(:,5003) ~= labelTest(:)))/500;
```

Codes for BDR:

```matlab
clear all;
load('label.mat');
load('data.mat');
% data = load 'data.mat';
% train = data.imageTrain;

classmean = zeros(28,28,10);
summation = zeros(28,28,10);
for i = 1:10
    class = find(labelTrain==i-1);
    for j = 1:size(class,1)
        summation(:,:,i) = summation(:,:,i) + imageTrain(:,:,class(j));
    end
    classmean(:,:,i) = summation(:,:,i)./(size(class,1));
end

figure;
for i = 1:10
    subplot(2,5,i);
    imshow(classmean(:,:,i),[]);
end

% part2:
samplemean = reshape(classmean,[784,10]);
diff = zeros(784,10);
diff_trans = zeros(10,784);
i_x = zeros(10,1);

sampletest = reshape(imageTest,[784,500]);
prediction = zeros(500,1);

 for i = 1:500
    for j = 1:10
        diff(:,j) = sampletest(:,i) - samplemean(:,j);
        diff_trans(j,:) = diff(:,j).';
        i_x(j,1) = (-1/2)*(diff_trans(j,:) * diff(:,j));
    end
    [a,prediction(i,1)] = max(i_x);
 end

 for i = 1:500
     prediction(i,1) = prediction(i,1)-1;
 end

[total_error_num,col] = size(find(prediction~=labelTest));
total_error_rate = total_error_num/500;

error_num_for_each_class = zeros(10,1);
num_for_each_class = zeros(10,1);
for i = 1:10
    [num_for_each_class(i),dummyy] = size(find(labelTest==i-1));
    [error_num_for_each_class(i),dummy] = size(find(prediction(labelTest==i-1) ~= i-1));
end
error_rate = error_num_for_each_class./num_for_each_class;


%Error_plot:
figure;
index = (0:1:9);
bar(index,error_rate);
xlabel('Class');
ylabel('Error Rate');


%Error_Table:
Class = {'0';'1';'2';'3';'4';'5';'6';'7';'8';'9'};
Correctly_classified = num_for_each_class - error_num_for_each_class;
Incorrectly_classified = error_num_for_each_class;
Total_num_each_class = num_for_each_class;
Error_rate = error_rate;
T = table(Class,Total_num_each_class,Correctly_classified,Incorrectly_classified,Error_rate);
```

Codes for PCA:

```
clear all;
close all;
clc;
load('data.mat');
load('label.mat');
train = reshape(imageTrain,[784,5000])/255;
test = reshape(imageTest,[784,500])/255;

%% Part 1
% compute sample mean:
train_mean = mean(train,2); % get 784*1 mean of all 5000 imageTrain
% cov
train_cov = cov(train'); % rows are observations while columns are random variables
% eigens:
[eigen_vector,eigen_value] = eig(train_cov);
% sort:
eigen_value = diag(eigen_value);
[eigen_value_sort,I] = sort(eigen_value,'descend');
eigen_vactor_sort = eigen_vector(:,I(1:784)); % flip becasue it is already
% in ascending order just like it in eigen_value
eigen_vactor_sort2828 = reshape(eigen_vactor_sort,[28,28,784]);

figure;
title('Top 10 Principle Components');
for i = 1:100
    subplot(10,10,i);
    imshow(eigen_vactor_sort2828(:,:,i),[]);
end

figure;
index = 0:1:783;
plot(index,eigen_value_sort);
title('Eigen Values');
xlabel('feature Space Dimension(pixels)');

%% Part 2:feature space with top k's:
figure;

k_loop_counter = 1;
for k = [5,10,20,30,40,60,90,130,180,250]
train_mean = mean(train,2); % get 784*1 mean of all 5000 imageTrain
% cov
train_cov = cov(train'); % rows are observations while columns are random variables?5000*784
% eigens:
[eigen_vector,eigen_value] = eig(train_cov);

% sort:
eigen_value = diag(eigen_value);
[eigen_value_sort,I] = sort(eigen_value,'descend');

% subspace:
eigen_value_lower = eigen_value_sort(1:k,:);
eigen_vactor_lower = eigen_vector(:,I(1:k));

% project imageTest into lower dimension:
test = test - train_mean;
test_lower = eigen_vactor_lower' * test;
% project imageTrain into lower dimension:
train = train - train_mean;
train_lower = eigen_vactor_lower' * train;

% calculate cov:
%cov_lower = cov(train_lower'); %**************************************
% Apply BDR:
classmean_lower = zeros(k,10);
sum_lower = zeros(k,10);
for i = 1:10
    index_each_class_lower = find(labelTrain==i-1);
    for j = 1:length(index_each_class_lower)
        sum_lower(:,i) = sum_lower(:,i) + train_lower(:,index_each_class_lower(j));
    end
```

```
    classmean_lower(:,i) = sum_lower(:,i)./(length(index_each_class_lower));
end
diff = zeros(k,10);
i_x = zeros(10,1);
prediction = zeros(500,1);
for i = 1:500
    for j = 1:10
        index_each_class_lower = find(labelTrain==j-1);
        each_class_lower = train_lower(:,index_each_class_lower);
        cov_each_class_lower = cov(each_class_lower');

        diff(:,j) = test_lower(:,i) - classmean_lower(:,j);
        %i_x(j,1) = (-1/2)*(diff(:,j)' * diff(:,j)); % with cov = 1
        %i_x(j,1) = (-1/2)*(diff(:,j)'/cov_lower * diff(:,j));% with train-cov
        %i_x(j,1) = (-1/2)*(diff(:,j)'/cov_each_class_lower * diff(:,j));% with class-train-c
        i_x(j,1) = mvnpdf(test_lower(:,i),classmean_lower(:,j),cov_each_class_lower);
    end
    [a,prediction(i,1)] = max(i_x); % return the max value and its index
end
for i = 1:500
     prediction(i,1) = prediction(i,1)-1;
end
% Calculate total errors:
total_error_num = length(find(prediction~=labelTest));
total_error_rate(k_loop_counter) = total_error_num/500;
% Calculate errors for each class:
error_num_for_each_class = zeros(10,1);
num_for_each_class = zeros(10,1);
for i = 1:10
    num_for_each_class(i) = length(find(labelTest==i-1));
    error_num_for_each_class(i) = length(find(prediction(labelTest==i-1) ~= i-1));
end
error_rate_for_each_class = error_num_for_each_class./num_for_each_class;
% plots:
index = (0:1:9);
subplot(3,4,k_loop_counter);
bar(index,error_rate_for_each_class);
ylim([0 1]);
xlabel('Class');
ylabel('Error Rate');


k_loop_counter = k_loop_counter + 1;
end
%% total error plot for part 2:
figure;
plot([5,10,20,30,40,60,90,130,180,250],total_error_rate);
title('Principle Component Analysis');
ylabel('Error rate');
xlabel('Dimensions');

```