Classifying Digits

Ming Lou

Abstract:

In this paper, I will mainly apply the Linear classifier (LDA) to build a classifier to identify individual digits in a training set. This data-driven method is commonly using in image recognition and classification. With some training image and label, we are able to do the machine learning. Also, we will compare the performance with decision trees and SVM.

Section I. Introduction and Overview :

We will use the MNIST data from http://yann.lecun.com/exdb/mnist to train the digital images and create a linear classifier. Before using LDA, we need to use methods like wavelet transforms and SVD to clean the raw data. Once I have my data projected into PCA space, I will build a linear classifier both for two digits and three digits and quantify the accuracy of the separation with LDA on the test data.

Section II. Theoretical Background:

1. Singular Value Decomposition (SVD)

   Singular Value Decomposition is a matrix method for reducing a matrix to its constituent part in order to make certain subsequent matrix calculations simpler. It can be represented as

$$A = USV^T$$

where $U \in R^{m \times m}$and $V \in R^{n \times n}$are unitary matrices, and $s \in 12^{m \times n}$ is diagonal matrix with non-negative real numbers on the diagonal.

The diagonal entries σ_i of S are known as the singular values of S. The number of non-zero singular values is equal to the rank of S. The columns of U and the columns of V are called the left-singular vectors and right-singular vectors of M. In this project, we use SVD to make the raw data simpler by lower the rank. With the data we kept, we can still see the main information we want.

2. Linear Discriminant Analysis (LDA)

   The main goal for Linear Discriminant Analysis is to find a suitable projection that maximize the distance between the inter-class data while minimizing the intra-class data. For 2 datasets, we define the between-class scatter matrix

$$s_B = (\mu_2 \cdot \mu_1)(\mu_2 - \mu_1)^T$$

   which is a measure of the variance within each group; While the with-class scatter matrix is defined as

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

Then we want to find $w = argmax \frac{w^T s_B w}{w^T s_w w}$, which is hard to solve. However, we can solve it easily because vector w that maximize the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem

$$s_B w = \lambda s_u w$$

LDS for more groups has very similar logic. We just changes to the scatter matrices:

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

Where μ is the overall mean and μ_j is the mean of each of the N > 3groups/classes.

The within-class scatter matrix becomes

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

Section III. Algorithm Implementation and Development

Main steps:

1. Using the reshape and wavelet transformation to clean the datasets
2. Applying SVD to project the data onto the principle component basis
3. Using LDA to separates different categories
4. Verifying how the classify works by quantifying the accuracy

5. Compare the performance with SVM and decision tree

After loading the data, I firstly reshape each image by reshape function in MATLAB into a column vector and each column represents a different image. Then I use the built-in function : [cA, cH, cV, cD] = dwt2(X, 'name') to do the wavelet transformation. By doing so, I add horizontal details and vertical details together to represent data. With the data cleaned, I applied the SVD function to do the singular value analysis. With different principle components shown, I found 60 modes is good for image recognition. Thus, I choose to use the first 60 modes instead of 196. This makes the matrix smaller. After projecting onto the selected PCA modes I started to di the Linear Discrimination Analysis. I firstly randomly pick three number which are one two and three. I use the logic in LDA to compute the between-class scatter matrix and the within-class scatter matrix and find w. To project the data onto w, I multiplied w with the original data. After that, I verify the performance by calculating the success rate.

Section IV. Computational Results

Figure 1 below the first nine principle components. By observing the image, we can see that the first principle components captures most information which is corresponding to the points in Figure2.

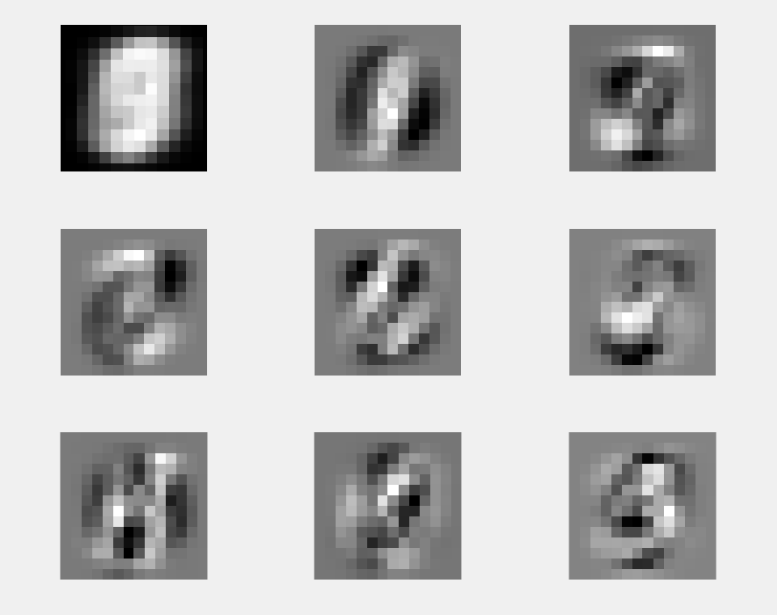


Figure 1: First Nine Principle Components

Figure2 shows that the first mode is dominant, which is far larger than any other modes. While several modes are also lager than 200, it means they have captures some weights of information

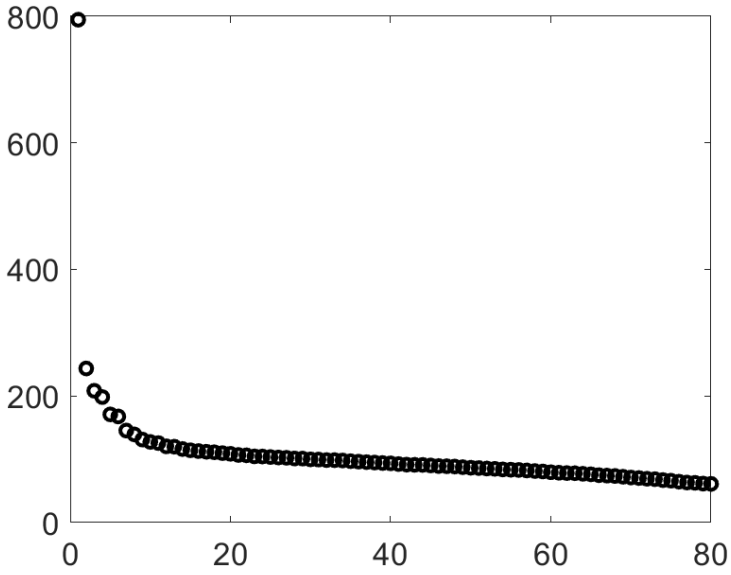


Figure2: Singular Values

Figure 3 and Figure 4 shows the projection for images labeled 1 and 2.

We found that the success rate is 99.59% which is relatively high. The SVM and decision tree gave very similar.

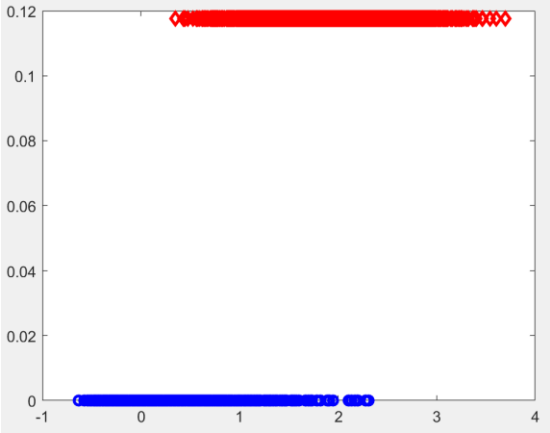
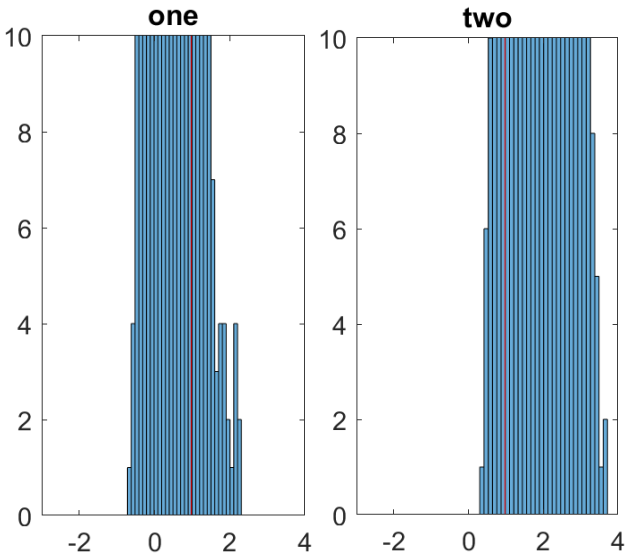


Figure 3: projection of ones and twos



Fugure 4: histograms of the projected values

Section V. Summary and Conclusions

The Linear Discriminant analysis is a very efficient method to classify datasets since we only need to train the main information for datasets thanks to the SVD. While data are trained, the success rate is pretty high when we using the test data to verify.

Appendix A. MATLAB functions used and brief implementation explanation

`B = reshape(A,sz)` reshapes `A` using the size vector, `sz`, to define `size(B)`.

`[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of `m`-by-`n` matrix `A`:

diag(V) returns a square diagonal matrix with the elements of vector V on the main diagonal.

`L = kfoldLoss(obj,Name,Value)` calculates loss with additional options specified by one or more `Name,Value` pair arguments.

Appendix B. MATLAB codes

```
%%
[train_images, train_labels] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
[test_images, test_labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');
train_images = reshape(train_images,[784 60000]);
test_images = reshape(test_images,[784 10000]);

train_wave = dcwavelet(train_images);
test_wave = dcwavelet(test_images);

%%
[U,S,V] = svd(train_wave,'econ');
```

```
%%
for k = 1:9
    subplot(3,3,k)
    ut1 = reshape(U(:,k),14,14);
    ut2 = rescale(ut1);
    imshow(ut2)
end
%%
plot(diag(S),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 80])

%%
PCA=S*V';
%%
ones = []; twos=[]; threes=[];
feature=60;
for j=1:60000
    if train_labels(j) == 1
       ones(:,end+1) = PCA(1:feature,j);
    elseif train_labels(j) == 2
       twos(:,end+1) = PCA(1:feature,j);
    elseif train_labels(j) == 3
        threes(:,end+1) = PCA(1:feature,j);
    end
end
sz = 5958;
%% 1 2
ones = ones(1:feature,1:sz);
twos = twos(1:feature,1:sz);
mo = mean(ones,2);
mt = mean(twos,2);
sw=0;
for k = 1:sz
    sw = sw + (ones(:,k) - mo)*(ones(:,k) - mo)';
end

for k=2:sz
    sw = sw + (twos(:,k) - mt)*(twos(:,k) - mt)';
end
Sb = (mo-mt)*(mo-mt)';
[V2, D] = eig(Sb,sw);
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);

vone = w'*ones;
vtwo = w'*twos;
```

```
if mean(vone)>mean(vtwo)
   w = -w;
   vone = -vone;
   vtwo = -vtwo;
end

%%
plot(vone,zeros(5958),'ob','Linewidth',2)
hold on
plot(vtwo,ones(5958),'dr','Linewidth',2)
%%
sortone = sort(vone);
sorttwo = sort(vtwo);
t1 = length(sortone);
t2 = 1;
while sortone(t1)>sorttwo(t2)
      t1 = t1 - 1;
       t2 = t2 + 1;
end
threshold = (sortone(t1) + sorttwo(t2))/2;

%%
subplot(1,2,1)
histogram(sortone,30); hold on, plot([threshold threshold], [0
10],'r')
set(gca,'Xlim',[-3 4],'Ylim',[0 10],'Fontsize',14)
title('one')
subplot(1,2,2)
histogram(sorttwo,30); hold on, plot([threshold threshold], [0
10],'r')
set(gca,'Xlim',[-3 4],'Ylim',[0 10],'Fontsize',14)
title('two')
%%
U = U(:,1:feature);
test_proj = U'*test_wave;

test = [];
label = [];
for i = 1:1000
if test_labels(i) == 1 || test_labels(i) == 2
test(:,end+1) = test_proj(:,i);
label(:,end+1) = test_labels(i);
end
end
val = w'*test;
ResVec = (val>threshold);
```

```
err = abs(ResVec - label);
errNum = sum(err);
sucRate = 1 - errNum/60000

%%
one_two = [ones twos]
one_two_label=[ones([1 5958]) twos([1 5959])]
tree=fitctree(one_two', one_two_label,'CrossVal','on');
%%
SVM = fitcsvm(one_two',one_two_label);
CV = crossval(SVM);
SVMLoss = kfoldLoss(CV)
```