AMATH 482 Homework 4

Maxime Didascalou

March 10, 2021

Abstract

The aim of this project is to train the computer to recognize what digit is in a picture. More specifically, I used PCA on a training set of images (all containing a number between 0 and 9) and then used several methods that use the principal components in order to try and classify what digits are in a test set of images. We get that the most accurate way of classifying the test set is SVM.

1 Introduction and Overview

There are a lot of different ways that one can implement supervised machine learning. In this paper, I will look at three of those methods, and how their performances in identifying what digit (0 through 9) is in an image compare. We will mostly focus on linear discriminant analysis but we will also look at support vector machine and decision tree classifiers.
We will first go through some theoretical background, and then explain the code (Appendix B) and algorithms that were used in order to compare these methods and their success rates.

2 Theoretical Background

2.1 SVD/PCA

As we learned from our textbook [2], using the singular value decomposition (SVD) it is possible to write any m by n matrix $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{1}$$

This comes from the fact that you can write

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma} \tag{2}$$

Where $\mathbf{U}$ is a matrix containing all the principle semiaxes (scaled to have a length of 1) that would appear if we applied $\mathbf{A}$ to the unit circle, $\mathbf{\Sigma}$ contains all the lengths of these semiaxes (from biggest to smallest), and $\mathbf{V}$ contains the vectors (from the unit circle) that give these principle semiaxes when multiplied by $\mathbf{A}$. This makes it so that $\mathbf{V}$ and $\mathbf{U}$ are both orthonormal (so their inverse equals their transpose) and $\mathbf{\Sigma}$ is a diagonal matrix where each of the scaling factors are arranged in decreasing order (they are called the singular values).
When the mean of the rows of $\mathbf{A}$ is set to 0 and we divide it by $\sqrt{n-1}$, the resulting columns of $\mathbf{U}$ from the SVD are called the principal components of $\mathbf{A}$, and it can be shown that the singular values are the standard deviations of these components. Therefore it is possible to only keep the principle components that contain a lot of information (i.e that have a big variance). You can do this by projecting the data onto the principle components:

$$\mathbf{Y} = \mathbf{U}^*\mathbf{A} \tag{3}$$

and keeping a certain number of rows.

2.2 Linear Discriminant Analysis (LDA)

Now that we have data projected onto its principal components, we want to be able to see patterns for the different categories of the data. For example if we have two categories (A and B) and we plot the first two principal components, all the A data might be far from all the B data on the plot. So if we get a new data point, and we don't know whether it belongs to A or B, we can project it onto the principal components, plot it on the graph and see where it lies.
LDA can do this with more than 2 principal components or groups. It projects everything on one line ($\mathbf{w}$), and establishes a threshold where for example, every new data point that is below it will be classified as A, and if it's above it will be classified as B (this example is for only 2 groups). To find this line you want to minimize the variance within the groups:

$$\mathbf{S}_w = \sum_{j=1}^{N} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T, \tag{4}$$

and maximize the variance between the groups:

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T, \tag{5}$$

for 2 groups, and:

$$\mathbf{S}_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T, \tag{6}$$

for more groups.
μ_j represents the mean for group j for each feature, μ is the overall mean, and N is the number of different groups.
$\mathbf{w}$ is the eigenvector that corresponds to the largest eigenvalue of the generalized eigenvalue problem

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_w\mathbf{w}. \tag{7}$$

2.3 SVM

Instead of projecting everything onto one line, SVM finds a hyperplane that best separates the data. It is easier to think about in low dimensions: For example if the data is in 2 dimensions, SVM will find lines that best separate the different groups, if it is in 3 dimensions, it will find planes.
Note that the hyperplanes are not "straight". They can curve around the data points and create boundaries for the different classes. [1]

2.4 Decision Tree

The decision tree method makes a binary tree. Say the training data has 3 features, the algorithm will look at the training data and establish several thresholds for each of the 3 features so that when a new data point follows the tree based on the value of it's own features, it will be labeled by where it ends up (no more splits).[3]

3 Algorithm Implementation and Development

Using a training set of 60000 images of digits (0-9), the goal was to write a program that could correctly identify the digits in a test set of 10000 images. My strategy was to:

1. Find the principal components of the training set digit images.
2. Project the test set images on these principal components.
3. Perform LDA to label each test set image.

4. Compare to performance of SVM and decision tree classifiers.

Algorithm 1: Setting up LDA

Import training data and training labels.
Import test data and test labels.
Reshape each training picture into a column vector, and make a matrix `A` with all the pictures in it
Subtract mean of each row
Follow the same procedure for the matrix `A2` (test data).
Compute the SVD of `A`
Project data onto Principal components: $\mathbf{\Sigma V}^T$ $(= \mathbf{U}^T\mathbf{A})$

Algorithm 2: LDA for 2 digits

`feature` = number of modes you want to use
Create a matrix `digit1` that corresponds to the columns labeled with the first digit of the projection from algorithm 1, only keep the first `feature` rows.
Do the same for `digit2`.
Compute `Sw` and `Sb` as described in Theoretical Background (more details in Appendix B Listing 2).
Compute `w` as described in Theoretical Background (more details in Appendix B Listing 2).
`v1` = $\texttt{w}^T$`digit1`
`v2` = $\texttt{w}^T$`digit2`
Find the threshold value using `v1` and `v2` (more details in Appendix B Listing 2)
Project the columns of `A2` (from Algorithm 1) that corresponds to the 2 digits in question onto the PCs (also found in Algorithm 1). Only keep the first `feature` columns
Project onto `w`
Use threshold value and true labels to figure out the success rate (more details in Appendix B Listing 2)

LDA for 3 digits is done almost the same way. The things that change are:

1. Create `digit3` on top of `digit1` and `digit2`.
2. The computation of `Sw` and `Sb` as described in the Theoretical background (more details in Appendix B Listing 3).
3. We now find 2 threshold values (more details in Appendix B Listing 4).

To find the max and min success rate for 2 digits, you loop over all the combinations of digits and calculate the success rate, set it as min if it's smaller than the current min, and set it as max if it's bigger than the current max. More details are available in Appendix B Listing 6, which also finds the average and does it for 3 digits.

Algorithm 3: Implementing SVM and decision tree classifiers (details in Appendix B Listing 7)

Use `fitcecoc` (SVM) or `fitctree` (decision tree) on the columns of your choice from the training matrix projected onto its principal components, along with their respective labels to build classifiers.
Using `predict` and the classifier, label a test set of your choice (also has to be projected on principal components from step 1)
Compare with the true labels from the test set that you chose
Find the error rate by adding all the errors together and dividing by the length of the labels vector
Success rate = 1 - error rate

4 Computational Results

4.1 Analysis

I first rearranged all the training pictures in one matrix as described in Algorithm Implementation, then subtracted the mean and performed SVD on that matrix. The resulting **U** matrix then contained the principal components (what makes up the pictures), and the columns of **V** contained the right singular

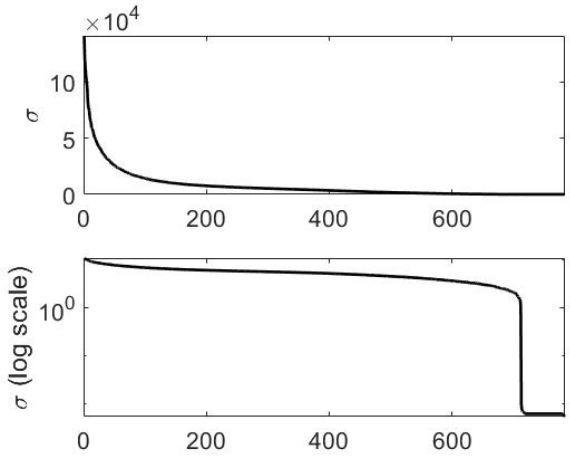

Figure 1: Singular values of training set

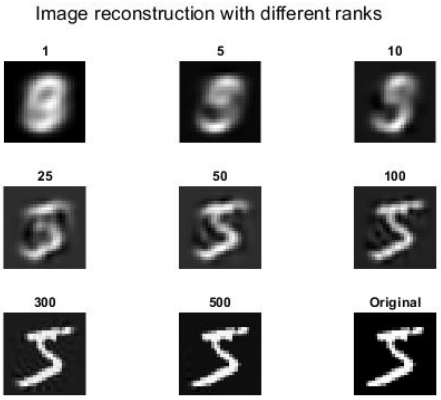

Figure 2: Rank r approximations of the first training image

vectors, meaning that they say how much of each principal component is represented in each picture. For example, in the first column of $\mathbf{V}$, the first index is how much the first picture uses the first component, the second index is how much the second picture uses the first component, and so on. It makes sense then that when we scale those columns by their corresponding singular value (this is equivalent to projecting on the principal components) and plot three of them as was done in Figure 3, we can get a sense of where the separations occur.

The singular values contained in $\mathbf{\Sigma}$ represent the importance of their respective principal component (not quite the standard deviation since I did not divide by $\sqrt{n-1}$). As we can see in Figure 1, there is a fast drop off for the value of the singular values at the start but it takes a while for them to be close to 0. Also, we can see in Figure 2 that the digits can already be recognized with rank 10-50 approximations. This made me try different numbers of modes for LDA, SVM and decision tree classifiers. To make it simpler, I will discuss my findings with 40 modes since it gave good results with all the methods.

4.2 LDA

Using LDA, the 2 digits that had the highest success rate of recognition in the test set were 0 and 1 with a success rate of 0.9986. For 3 digits they were 0, 1 and 7 with a success rate of 0.882.

The 2 digits with the least success were 4 and 9 with a success rate of 0.9488, and with 3 digits they were 2, 7 and 9 with a success rate of 0.5272. This makes sense when you look at Figure 3 because, for example,

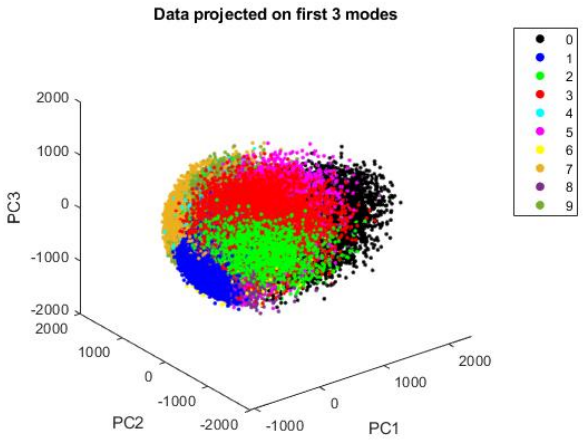

Figure 3: Training set projected on first 3 principal components

0 and 1 seem to be very well separated, and 4 and 9 have a lot of overlap.
When using the classifier on the training set, the success rate barely changes for all the digits: average 2 digit success rate for:

1. test set: 0.9824
2. training set: 0.9807

Average 3 digit success rate for:

1. test set: 0.7522
2. training set: 0.7450

4.3 SVM

Using SVM, we get that the 10 digit separation success for the test set is 0.9337 (0.9288 for training set), and the success rate for the best performing digits in LDA (0 and 1) is 0.9991 (0.9992 for training set), and for the worst performing (4 and 9) is 0.9623 (0.9611 for training set).

4.4 Decision Tree

Using a decision tree classifier, we get a 10 digit separation success rate of 0.8333 for the test set (0.9483 for training set), and the success rate for the best performing digits in LDA (0 and 1) is 0.9943 (0.9993 for training set), and for the worst performing (4 and 9) is 0.8895 (0.9750 for training set).

5 Summary and Conclusions

When classifying the test set, we get that the most successful classifier for 2 digit classification was SVM. It also was the most successful classifier for 10 digit separation. LDA was better than a decision tree for 2 digit separation. However, when classifying the training set, the decision tree was the most successful in both 2 digit and 10 digit separation since it was the only one that saw a true increase in performance. This seems to suggest that the decision tree did some over fitting, much like a regression model of too high degree.
The changes in accuracy in LDA going from 2 digits to 3 show some of the disadvantages of LDA, but all in all, it is still a pretty good result considering the simplicity of the algorithm.

In conclusion, this shows that these methods for supervised machine-learning work and that they could be useful for many applications.

References

[1] Mat Deris Ashanira. *Overview of Support Vector Machine in Modeling Machining Performances*. Nov. 2011. URL: `https://www.sciencedirect.com/science/article/pii/S1877705811054993`.

[2] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[3] Yan-Yan Song. *Decision tree methods: applications for classification and prediction*. Apr. 2015. URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/`.

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix `A`, such that `A = U*S*V'`.
- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1`-by-...-by-`szN` array where `sz1`,...,`szN` indicates the size of each dimension.
- `[V,D] = eig(A,B)` returns diagonal matrix `D` of generalized eigenvalues and full matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V = B*V*D`.
- `n = norm(v,p)` returns the generalized vector `p`-norm.
- `Mdl = fitcecoc(X,Y)` returns a trained ECOC model using the predictors `X` and the class labels `Y (SVM)`.
- `tree = fitctree(Tbl,Y)` returns a fitted binary classification decision tree based on the input variables contained in the table `Tbl` and output in vector `Y`.

Appendix B MATLAB Code

```
clear all; close all; clc

[images_t, labels_t] = mnist_parse('train-images.idx3-ubyte', 'train-labels.idx1-ubyte');
[images, labels] = mnist_parse('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte');

[m,n,N] = size(images_t);
A = zeros(m*n,N);
for i = 1:N
    A(:,i) = reshape(images_t(:,:,i),m*n,1);
end
num = sum(labels_t == 0:9);
M = mean(A,2);
A_mean0 = A - M*ones([1 N]);

[m2,n2,N2] = size(images);
A2 = zeros(m2*n2,N2);
for i = 1:N2
    A2(:,i) = reshape(images(:,:,i),m2*n2,1);
end
M2 = mean(A2,2);
A2_mean0 = A2 - M2*ones([1 N2]);

%% svd decomposition
[U,S,V] = svd(A_mean0,'econ');
digits = S*V';
```

Listing 1: Setting up data and performing SVD

```
function sucRate = LDA(U,digits,feature,labels_t,labels,dig1,dig2,A,num)
    digit1 = digits(1:feature,find(labels_t == dig1));
    digit2 = digits(1:feature,find(labels_t == dig2));
    %scatter matrices:
    m1 = mean(digit1,2);
    m2 = mean(digit2,2);
    Sw = 0; % within class variances
    for k = 1:num(dig1+1)
        Sw = Sw + (digit1(:,k) - m1)*(digit1(:,k) - m1)';
    end
    for k = 1:num(dig2+1)
        Sw =  Sw + (digit2(:,k) - m2)*(digit2(:,k) - m2)';
    end
    Sb = (m1-m2)*(m1-m2)'; % between class
    % Find the best projection line
    [V2, D] = eig(Sb,Sw); % linear disciminant analysis
    [lambda, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w,2);
    % Project onto w
    v1 = w'*digit1;
    v2 = w'*digit2;
    % Make 0 below the threshold
    if mean(v1) > mean(v2)
        w = -w;
        v1 = -v1;
        v2 = -v2;
    end
    % Find the threshold value
    sort1 = sort(v1);
    sort2 = sort(v2);
    t1 = length(sort1);
    t2 = 1;
    while sort1(t1) > sort2(t2)
        t1 = t1 - 1;
        t2 = t2 + 1;
    end
    threshold = (sort1(t1) + sort2(t2))/2;
    % classifying unknown digits
    labelsnew = labels(labels == dig1 | labels == dig2);
    labelsnew = labelsnew == dig2;
    Y = U'*A(:,labels == dig1 | labels == dig2); % PCA projection
    pval = w'*Y(1:feature,:);
    ResVec = (pval > threshold);
    err = abs(ResVec - labelsnew');
    errNum = sum(err);
    sucRate = 1 - errNum/size(labelsnew,1);
end
```

Listing 2: Performing LDA on 2 digits

```
% Project onto PCA modes
digit1 = digits(1:feature,find(labels_t == dig1));
digit2 = digits(1:feature,find(labels_t == dig2));
digit3 = digits(1:feature,find(labels_t == dig3));
%scatter matrices:
m1 = mean(digit1,2);
m2 = mean(digit2,2);
m3 = mean(digit3,2);
mo = (m1+m2+m3)/3;
Sw = 0; % within class variances
for k = 1:num(dig1+1)
    Sw = Sw + (digit1(:,k) - m1)*(digit1(:,k) - m1)';
end
for k = 1:num(dig2+1)
    Sw =  Sw + (digit2(:,k) - m2)*(digit2(:,k) - m2)';
end
for k = 1:num(dig3+1)
    Sw =  Sw + (digit3(:,k) - m3)*(digit3(:,k) - m3)';
end
Sb1 = (m1-mo)*(m1-mo)'; Sb2 = (m2-mo)*(m2-mo)'; Sb3 = (m3-mo)*(m3-mo)';
Sb = Sb1 + Sb2 + Sb3;
% Find the best projection line
[V2, D] = eig(Sb,Sw); % linear disciminant analysis
[lambda, ind] = max(abs(diag(D)));
w = V2(:,ind);
w = w/norm(w,2);
% Project onto w
v1 = w'*digit1; v2 = w'*digit2; v3 = w'*digit3;
```

Listing 3: Performing LDA on 3 digits (Part 1: Projection line)

```
% Find the threshold values
[M,Imin] = min([sum(v1(1,:)) sum(v2(1,:)) sum(v3(1,:))]);
[M,Imax] = max([sum(v1(1,:)) sum(v2(1,:)) sum(v3(1,:))]);
Imid = 6 - Imin - Imax;
sortt{1} = sort(v1(1,:)); sortt{2} = sort(v2(1,:)); sortt{3} = sort(v3(1,:));
t1 = length(sortt{Imin}); t2 = 1;
while sortt{Imin}(t1) > sortt{Imid}(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end
threshold1 = (sortt{Imin}(t1) + sortt{Imin}(t2))/2;
t2 = length(sortt{Imid}); t3 = 1;
while sortt{Imid}(t2) > sortt{Imax}(t3)
    t2 = t2 - 1;
    t3 = t3 + 1;
end
threshold2 = (sortt{Imid}(t2) + sortt{Imax}(t3))/2;
% classifying unknoum digits
digs = [dig1 dig2 dig3];
Y = U'*A(:,labels == dig1 | labels == dig2| labels == dig3); % PCA projection
pval = w'*Y(1:feature,:);
labelsnew = labels(labels == dig1 | labels == dig2 | labels == dig3);
err = zeros([1 length(pval)]);
for i = 1:length(pval)
    if pval(i) < threshold1
        if digs(Imin) ~= labelsnew(i)
            err(i) = 1;
        end
    elseif pval(i) > threshold2
        if digs(Imax) ~= labelsnew(i)
            err(i) = 1;
        end
    else
        if digs(Imid) ~= labelsnew(i)
            err(i) = 1;
        end
    end
end
errNum = sum(err);
sucRate = 1 - errNum/size(labelsnew,1);
```

Listing 4: Performing LDA on 3 digits (Part 2: Threshold and success rate)

```
%% approximations of images:
[U2,S2,V2] = svd(A,'econ');
r = [1 5 10 25 50 100 300 500];
figure(1)
for k = 1:8
   subplot(3,3,k)
   new = U2(:,1:r(k))*S2(1:r(k),1:r(k))*V2(:,1:r(k))';
   im = reshape(new(:,1),28,28);
   im = rescale(im);
   imshow(im)
   title(r(k))
end
subplot(3,3,9)
imshow(images_t(:,:,1))
title('Original')
sgtitle('Image reconstruction with different ranks')
%% plotting singular values
figure(2)
subplot(2,1,1)
plot(diag(S),'k','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 m*n])
ylabel("\sigma")
subplot(2,1,2)
semilogy(diag(S),'k','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 m*n])
ylabel("\sigma (log scale)")
%% plotting projected data
figure(3)
col = [0 0 0;0 0 1;0 1 0;1 0 0;0 1 1;1 0 1;1 1 0;
    0.9290 0.6940 0.1250;0.4940 0.1840 0.5560;0.4660 0.6740 0.1880];
for k = 0:9
    plot3(digits(1,find(labels_t == k)),digits(2,find(labels_t == k)),digits(3,find(labels_t == k)),'k.
    hold on
end
[h,icons] = legend('0','1','2','3','4','5','6','7','8','9');
icons = findobj(icons,'Type','line');
% Find lines that use a marker
icons = findobj(icons,'Marker','none','-xor');
% Resize the marker in the legend
set(icons,'MarkerSize',20);
xlabel("PC1")
ylabel("PC2")
zlabel("PC3")
title("Data projected on first 3 modes")
```

Listing 5: Plotting projection, singular values, and approximations

```
%% max and min succes rate
feature = 40;
min = 1;
max = 0;
min3 = 1;
max3 = 0;
sumrate = 0;
sumrate3 = 0;
for i = 0:9
   for j = i+1:9
        sucRate = LDA(U,digits,feature,labels_t,labels,i,j,A2_mean0,num);
        sumrate = sumrate + sucRate;
        if sucRate < min
            min = sucRate;
            mindigs = [i j];
        end
        if sucRate > max
            max = sucRate;
            maxdigs = [i j];
        end
        for k = j+1:9
            sucRate3 = LDA3(U,digits,feature,labels_t,labels,i,j,k,A2_mean0,num);
            sumrate3 = sumrate3 + sucRate3;
            if sucRate3 < min3
                min3 = sucRate3;
                mindigs3 = [i j k];
            end
            if sucRate3 > max3
                max3 = sucRate3;
                maxdigs3 = [i j k];
            end
        end
   end
end
sumrate = sumrate/45;
sumrate3 = sumrate3/120;
```

Listing 6: Finding max, min and average success rate

```
%% set up data
feature = 40;
X = digits(1:feature,:)';
Y = U'*A2_mean0;
Y = Y(1:feature,:)';

%% SVM for all 10:
Mdl = fitcecoc(X./max(X(:)),labels_t);
test_labels = predict(Mdl,Y./max(Y(:)));
% test accracy
err = sum(test_labels ~= labels);
succesrate = 1-err/N2

%% tree method for all 10:
tree=fitctree(X,labels_t,'CrossVal','on');
test_labels = predict(tree.Trained{1},Y);
% test it
err = sum(test_labels ~= labels);
succesrate = 1 - err/N2

%% set up data for 2 digits
dig1 = 4;
dig2 = 9;
Y2 = Y(labels == dig1 | labels == dig2,:);
X2 = X(labels_t == dig1 | labels_t == dig2,:);
newlab_t = labels_t(labels_t == dig1 | labels_t == dig2);
newlab = labels(labels ==dig1 | labels == dig2);

%% SVM for 2 digits
Mdl = fitcsvm(X2./max(X2(:)),newlab_t);
test_labels = predict(Mdl,Y2./max(Y2(:)));
% test accuracy
err = sum(test_labels ~= newlab);
succesrate = 1-err/length(test_labels);

%% tree for 2 digits:
tree=fitctree(X2,newlab_t,'CrossVal','on');
test_labels = predict(tree.Trained{1},Y2);
% test it
err = sum(test_labels ~= newlab);
succesrate = 1 - err/length(test_labels);
```

Listing 7: Using SVM and decision tree classifiers