Classifying Digits

Yumin Yin

March 10, 2021

Abstract

Our goal is to build a classifier that takes images and determines what digit it is. To accomplish this, we need to first perform an SVD analysis of the training data set and have our data projected onto PCA space. After successfully building our handmade linear classifier, we want to quantify its accuracy on the test data and compare its performance between suppor vector machines (SVM) and decision tree classifiers on both the training and test sets.

1 Introduction and Overview

We first do an SVD analysis of the reshaped training data matrix. Then we determine the number of modes that are necessary for good image reconstruction by looking at the singular value spectrum. To take advantage of the SVD that we have already computed by projecting onto 50 features (PCA modes), we start building our handmade linear classifier. After successfully building the classifier, we test its accuracy on the projected test data matrix. Later on, we feed support vector machines (SVM) and decision tree classifiers the same project training data. lastly we compare their performance with our handmade linear classifier on both training and test data sets.

2 Theoretical Background

2.1 Linear Discriminant Analysis (LDA)

The Fig. (1) and (2) illustrate the idea of LDA: two data sets are considered and projected onto new bases. On the left (1), the projection shows the data to be completely mixed, not allowing for any reasonable way to separate the data from each other. On the right (b), we can see the ideal caricature for LDA since the data is completely separated. In particular, the means μ_1 and μ_2 of the two data sets are well apart when projected onto the chosen subspace.

2.2 Goal of LDA

Find a suitable projections that maximizes the distance between the inter-class data while minimizing the intra-class data.

2.3 Implementing LDA for 2 Datasets

First, we calculate the means for each of our froups for each feature. As above, we will cal these μ_1 and μ_2. Note that these μ's are column vectors. since they are means across each row. We can then define the **between-class scatter matrix**, the Eq. (1):

$$\mathbf{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T. \tag{1}$$

This is a measure of the variance between the groups (between the means). Then we can define the **within-class scatter matrix**, the Eq. (2):

$$\mathbf{S}_w = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T. \tag{2}$$

This is a measure of the variance within each group. The goal is then to find a vector $\mathbf{s}$ such that, the Eq. (3):

$$\mathbf{w} = \text{argmax} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}. \tag{3}$$

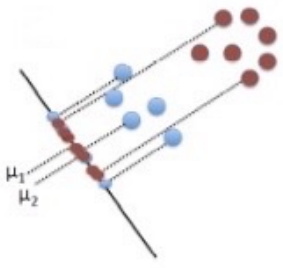

Figure 1: Bad Projection

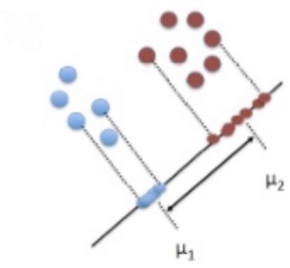

Figure 2: Good Projection

This is a tough problem to solve, but luckily for us it turns out that the vector $\mathbf{w}$ that maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem, the Eq. (4):

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_w\mathbf{w}. \tag{4}$$

This is something we can actually solve esaily with MATLAB!

2.4 LDA for More Groups

We can make the following changes to the scatter matrics, the Eq. (5):

$$\mathbf{S}_B = \sum_{j=1}^{N}(\mu_j - \mu)(\mu_j - \mu)^T, \tag{5}$$

where μ is the overall mean and μ_j is the mean of each of the $N \geq 3$ groups/classes. The within-class scatter matrix is the Eq. (6)

$$\mathbf{S}_w = \sum_{j=1}^{N}\sum_{\mathbf{x}}(\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T. \tag{6}$$

Then, `w` is found as it was above.

2.5 Decision trees

Decision trees, or classification trees and regression trees, predict responses to data. To predict a response, follow the decisions in the tree from the root (beginning) node down to a leaf node. The leaf node contains the response. Each step in a prediction involves checking the value of one predictor (variable). For example, the Fig. (3) is a simple classification tree. To predict, start at the top node, represented by a triangle (Δ). The first decision is whether x1 is smaller than 0.5. If so, follow the left branch, and see that the tree classifies the data as type 0. If, however, x1 exceeds 0.5, then follow the right branch to the lower-right triangle node. Here the tree asks if x2 is smaller than 0.5. If so, then follow the left branch to see that the tree classifies the data as type 0. If not, then follow the right branch to see that the tree classifies the data as type 1.

2.6 support vector machines (SVM)

support-vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

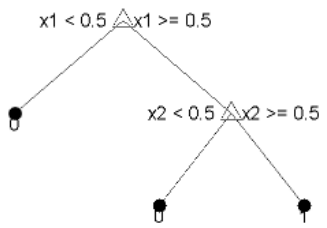

Figure 3: A Simple Classification Tree

3 Algorithm Implementation and Development

3.1 Perform an SVD analysis of the training data set

We first reshape each image in the training data set into a column vector and convert it to double precision. Each column of our training data matrix is now a different image represented by double precision. Then we subtract the mean of each row from the training data matrix so that each row has zero mean. We are now ready to compute SVD of our training data matrix. After that, we plot the singular value spectrum to see how many modes are necessary for good image reconstruction. It turns out that pick 50 modes is reasonable for good image reconstruction later.

3.2 Building a linear classifier (LDA) to identify individual digits

We start building our handmade linear classifier by taking advantage of the SVD that we have already computed by projecting onto 50 features (PCA modes). We first calculate the scatter matrices. Then We find $\mathbf{w}$ using the MATALAB eig() command with two arguments, our two scatter matrices. After that, to project onto $\mathbf{w}$, we multiply by $\mathbf{w}$'. Now we need to set a threshold for our classifier. It is easier for us to make decision if we have consistency for whether the digit X's are above or below the threshold, so we make digit X's always below. Lastly, we start with the last digit X value and smallest digit Y value. If the digit Y value is less than the digit X value, we move in one on each. We continue until the digit X value is less than the digit Y value, and we set out threshold between those values (the midpoint).

3.3 Predicting on the test date set

We also need to reshape each image in the test data set into a column vector and convert it to double precision. Then we demean the test data set by subtracting the mean of each row from the training data matrix. After that we want to project our test data onto the 50 modes of U, which is the left singular matrix we obtained from computing the SVD of the training data matrix. Now we are ready to use the classifier we built to classify digits on the test data set.

3.4 Compare the performances of different classifiers on both training and test sets

We compare the performances between our handmade linear, support vector machines (SVM) and decision tree classifiers on both training and test datasets. It is worth mentioning that when we are feeding SVM and decision tree classifiers, we want to use the projected training data matrix. Similarily, we want to use classifiers to predict projected test data matrix. This helps us compare the performances of these classifiers by controlling the variables.

4 Computational Results

4.1 SVD analysis

We can tell from the Fig. (4) that 50 modes are necessary for good image reconstruction, i.e., the rank of the digit space is 50. All three matrices, $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ from the SVD analysis provide valuable information. $\mathbf{U}$ matrix presents us the most important directions, i.e., features. $\mathbf{\Sigma}$ matrix tells us how relatively important each feature is. Columns of $\mathbf{V}$ matrix gives us the coefficients of linear combinations for columns of $\mathbf{U}$ matrix. The Fig. (5) is the projection onto the first three $\mathbf{V}$-modes. The digit 4 and 9 appear to be the most difficult to separate, as we can see in the Fig. (5), these two digits tend to clump together heavily. The digit 0 and 1 appear to be the easiest to separate, are they are far apart from each other in the Fig. (5).

4.2 Performances of different classifiers on both the training and test sets

The Table(1) shows the performances of different classifiers on different tasks on both the training and test sets. A linear classifier is not a great option for identifying three digits. Its performance on separating

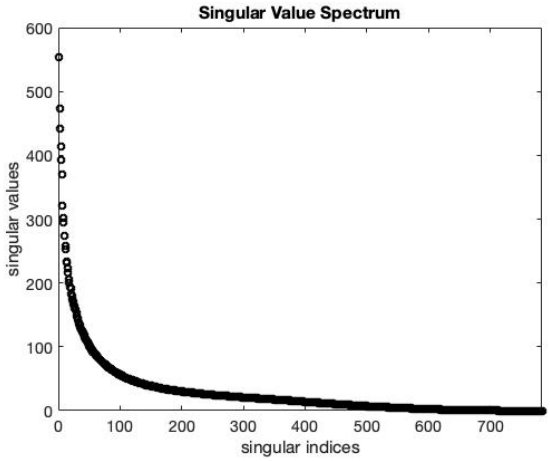

Figure 4: Singular Value Spectrum

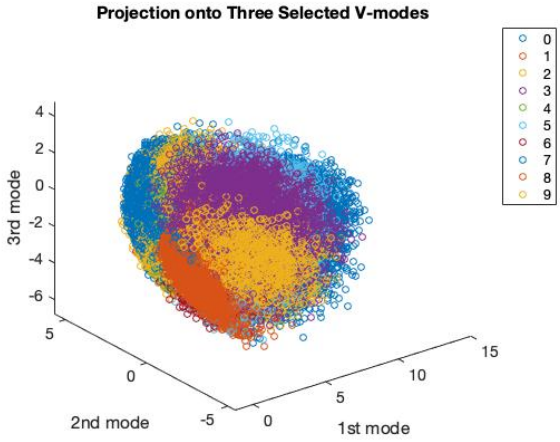

Figure 5: Projection onto Three Selected V-modes

	linear classifier	SVM classifier	decision tree classifier
(training) arbitrary two digits (6 & 8)	98.33%	/	/
(test) arbitrary two digits (6 & 8)	98.50%	/	/
(training) arbitrary three digits (0, 1 & 2)	34.89%	/	/
(test) arbitrary three digits (0, 1 & 2)	34.41%	/	/
(training) all ten digits	/	94.00%	96.11%
(test) all ten digits	/	94.12%	84.51%
(training) hardest pair (4 & 9)	30.85%	93.33%	95.01%
(test) hardest pair (4 & 9)	31.99%	93.87%	81.01%
(training) easiest pair (0 & 1)	99.72%	98.03%	98.44%
(test) easiest pair (0 & 1)	99.81%	98.63%	93.29%

Table 1: Performances of different classifiers on both the training and test sets

two digits varies depends on how difficult these two digits are to separate. SVM classifier and decision tree classifier both have the better accuracy than a linear classifier. Decision tree classifier is slightly more accurate on the pair of digits that is easy to separate, while SVM classifier is more accurate on the pair of digits that is hard to separate. On average, SVM is little more accurate than decision tree classifer. All classifier have close accuracy on the training and test sets, which is expected.

5 Summary and Conclusions

We have succeeded in building a classifier that takes images and determines what digit it is. An SVD analysis and PCA are great tools helps us projecting images while reducing the dimensionality. We build a linear classifier based upon the projected training data. We then test it on the test data. It turns out the accuracy of the linear classifier limits when the digits are hard to separate. Later on, we build SVM and decision tree classifies. We find both these two classifier have better accuracy than a linear classifier. On average SVM classifier is better than decision tree classifier.

Appendix A MATLAB Functions

- `B = reshape(A,sz1,...,szN)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `plot3(X,Y,Z,LineSpec)` creates the plot using the specified line style, marker, and color.

- `sz = size(A)` returns a row vector whose elements are the lengths of the corresponding dimensions of `A`. For example, if `A` is a 3-by-4 matrix, then `size(A)` returns the vector [3 4].
- `M = mean(A)` returns the mean of the elements of `A` along the first array dimension whose size does not equal 1.
- `B = repmat(A,r1,...,rN)` specifies a list of scalars, `r1,..,rN`, that describes how copies of `A` are arranged in each dimension.
- `[U,S,V] = svd(A,'econ')` produces an economy-size decomposition of m-by-n matrix `A`.
- `x = diag(A)` returns a column vector of the main diagonal elements of `A`.
- `[V,D] = eig(A)` returns diagonal matrix `D` of eigenvalues and matrix `V` whose columns are the corresponding right eigenvectors, so that `A*V = V*D`.
- `L = length(X)` returns the length of the largest array dimension in `X`.
- `find (X == Y)` returns the indices where `X = Y`.
- `class = classify(sample,training,group,'type')` classifies each row of the data in `sample` into one of the groups in `training`. `type` allows you to specify the type of discriminant function.
- `tree = fitctree(Tbl,ResponseVarName)` returns a fitted binary classification decision tree based on the input variables (also known as predictors, features, or attributes) contained in the table `Tbl` and output (response or labels) contained in `Tbl.ResponseVarName`. The returned binary tree splits branching nodes based on the values of a column of `Tbl`.
- `Mdl = fitcecoc(Tbl,ResponseVarName)` returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table `Tbl` and the class labels in `Tbl.ResponseVarName`.
- `label = predict(Mdl,X)` returns a vector of predicted class labels for the predictor data in the table or matrix `X`, based on the trained, full or compact classification tree `Mdl`.
- `Y = abs(X)` returns the absolute value of each element in array X.
- `[M,I] = max(A)` returns the linear index into A that corresponds to the maximum value in `A`.
- `X = zeros(sz1,...,szN)` returns an `sz1-by-...-by-szN` array of zeros where `sz1,...,szN` indicate the size of each dimension.
- `I2 = im2double(I)` converts the image `I` to double precision.

Appendix B MATLAB Code

```
clear all; close all; clc

[images, 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');

%% reshape each image and perform an SVD analysis
[~,~,num_of_images] = size(images);
images_reshape = reshapeimages(images);

% Subtract the mean of each row from the data
mean_of_row = mean(images_reshape, 2);
```

```
images_demean = images_reshape - repmat(mean_of_row, 1, num_of_images);

% demean the test data
test_images_reshape = reshapeimages(test_images);
[~,~,num_of_test_images] = size(test_images);
test_images_reshape_demean = test_images_reshape - repmat(mean_of_row, 1,
    num_of_test_images); % demean the test data

% perform an SVD analysis
[U,S,V] = svd(images_demean,'econ');

%% Plot the singular value spectrum

plot(diag(S),'ko','Linewidth',2)
title('Singular Value Spectrum')
set(gca,'Fontsize',14,'Xlim',[0 784])
xlabel('singular indices')
ylabel('singular values')

%% Project onto three selected V-modes

Projection = U(:,[1,2,3])' * images_reshape;
for i = 0:9
    Projection_digits = Projection(:, find(labels == i));
    plot3(Projection_digits(1,:), Projection_digits
        (3,:), 'o'); hold on
end
legend('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')
title('Projection onto Three Selected V-modes')
set(gca, 'fontsize', 14)
xlabel('1st mode')
ylabel('2nd mode')
zlabel('3rd mode')

%% Projection

feature = 50;
digits = S * V'; % projection onto principal components: X = USV' --> U'X = SV
    '
TestMat = U(:,1:feature)'*test_images_reshape_demean; % PCA projection

%% build a linear classifier to identify two arbitrary digits

[threshold_arbitrary, w_arbitrary, sortdigitXs, sortdigitYs] = handmade_LDA(
    digits, labels, feature, 6, 8);
[threshold_hard, w_hard, sortdigitXs, sortdigitYs] = handmade_LDA(digits,
    labels, feature, 3, 5);
[threshold_easy, w_easy, sortdigitXs, sortdigitYs] = handmade_LDA(digits,
    labels, feature, 0, 1);

%% Quantify the accuracy on the test data

accuracy_handmade_LDA_arbitrary_training = accuracy_rate_handmade_LDA(
    w_arbitrary, digits(1:feature,:), labels, 6, 8, threshold_arbitrary);
```

```
accuracy_handmade_LDA_arbitrary = accuracy_rate_handmade_LDA(w_arbitrary, TestMat, test_labels, 6, 8, threshold_arbitrary);
accuracy_handmade_LDA_hard_training = accuracy_rate_handmade_LDA(w_hard, digits(1:feature,:), labels, 4, 9, threshold_hard);
accuracy_handmade_LDA_hard = accuracy_rate_handmade_LDA(w_hard, TestMat, test_labels, 4, 9, threshold_hard);
accuracy_handmade_LDA_easy_training = accuracy_rate_handmade_LDA(w_easy, digits(1:feature,:), labels, 0, 1, threshold_easy);
accuracy_handmade_LDA_easy = accuracy_rate_handmade_LDA(w_easy, TestMat, test_labels, 0, 1, threshold_easy);

%% Build a linear classifier to identify three arbitrarily picked digits

training_three_digits = digits(1:feature,:);
training_three_digits = digits(find(labels == 0 | labels == 1 | labels == 2));
training_three_digits_labels = labels(find(labels == 0 | labels == 1 | labels == 2));
test_three_digits = TestMat(find(test_labels == 0 | test_labels == 1 | test_labels == 2));
test_three_digits_labels = test_labels(find(test_labels == 0 | test_labels == 1 | test_labels == 2));
class = classify(test_three_digits, training_three_digits, training_three_digits_labels, 'linear');
num_of_correct = 0;
for i = 1:length(test_three_digits)
    if class(i) == test_three_digits_labels(i)
        num_of_correct = num_of_correct + 1;
    end
end
accuracy_three_digits_test = num_of_correct / length(test_three_digits);
class = classify(training_three_digits, training_three_digits, training_three_digits_labels, 'linear');
num_of_correct = 0;
for i = 1:length(test_three_digits)
    if class(i) == test_three_digits_labels(i)
        num_of_correct = num_of_correct + 1;
    end
end
accuracy_three_digits_training = num_of_correct / length(test_three_digits);

%% decision tree classifier

tree = fitctree(digits(1:feature,:)', labels);

%% predict using decision tree classifier

predict_labels_dtreeclassifier = predict(tree, TestMat');
dtreeclassifier_training_labels = predict(tree, digits(1:feature,:)');
performance_dtreeclassifier_training = classifier_performance(dtreeclassifier_training_labels, labels);
performance_dtreeclassifier_test = classifier_performance(predict_labels_dtreeclassifier, test_labels);
hardest_pair_of_digits_dtreeclassifier = accuracy_rate(predict_labels_dtreeclassifier, test_labels, 4, 9);
```

```
hardest_pair_of_digits_dtreeclassifier_training = accuracy_rate(
    dtreeclassifier_training_labels, labels, 4, 9);
easiest_pair_of_digits_dtreeclassifier = accuracy_rate(
    predict_labels_dtreeclassifier, test_labels, 0, 1);
easiest_pair_of_digits_dtreeclassifier_training = accuracy_rate(
    dtreeclassifier_training_labels, labels, 0, 1);

%% SVM classifier

Mdl = fitcecoc(digits(1:feature,:)', labels);

%% predict using SVM classifier

predict_labels = predict(Mdl, TestMat');
SVM_training_labels = predict(Mdl, digits(1:feature,:)');
performance_SVM_training = classifier_performance(SVM_training_labels, labels)
    ;
performance_SVM_test = classifier_performance(predict_labels, test_labels);
hardest_pair_of_digits = accuracy_rate(predict_labels, test_labels, 4, 9);
hardest_pair_of_digits_training = accuracy_rate(SVM_training_labels, labels,
    4, 9);
easiest_pair_of_digits = accuracy_rate(predict_labels, test_labels, 0, 1);
easiest_pair_of_digits_training = accuracy_rate(SVM_training_labels, labels,
    0, 1);

%% functions

function performance = classifier_performance(predict_digits, correct_digits)
    number_of_correct = 0;
    for j = 1:length(predict_digits)
        if predict_digits(j) == correct_digits(j)
            number_of_correct = number_of_correct + 1;
        end
    end
    performance = number_of_correct / length(predict_digits);
end

function accuracy_handmade_LDA = accuracy_rate_handmade_LDA(w, TestMat,
    test_labels, digitX, digitY, threshold)
    pval = w'*TestMat;
    test_digitXs = find(test_labels == digitX);
    test_digitYs = find(test_labels == digitY);
    numCorrect = 0;
    index = 0;
    for k = 1:length(test_digitXs)
        index = index + 1;
        if pval(test_digitXs(index)) < threshold;
            numCorrect = numCorrect + 1;
        end
    end
    index = 0;
    for k = 1:length(test_digitYs)
        index = index + 1;
        if pval(test_digitYs(index)) > threshold;
```

```
147                 numCorrect = numCorrect + 1;
148             end
149         end
150         accuracy_handmade_LDA = numCorrect / (length(test_digitXs) + length(
                test_digitYs));
151 end
152
153
154 function [threshold,w,sortdigitXs,sortdigitYs] = handmade_LDA(digits, labels,
        feature, digitX, digitY)
155         ndigitXs = length(find(labels == digitX));
156         ndigitYs = length(find(labels == digitY));
157         digitXs = digits(1:feature,find(labels == digitX));
158         digitYs = digits(1:feature,find(labels == digitY));
159
160         % Calculate scatter matrices
161
162         mdigitXs = mean(digitXs,2);
163         mdigitYs = mean(digitYs,2);
164
165         Sw = 0; % within class variances
166         for k = 1:ndigitXs
167             Sw = Sw + (digitXs(:,k) - mdigitXs)*(digitXs(:,k) - mdigitXs)';
168         end
169         for k = 1:ndigitYs
170             Sw =  Sw + (digitYs(:,k) - mdigitYs)*(digitYs(:,k) - mdigitYs)';
171         end
172
173         Sb = (mdigitXs-mdigitYs)*(mdigitXs-mdigitYs)'; % between class
174
175         % Find the best projection line
176
177         [V2, D] = eig(Sb,Sw); % linear disciminant analysis
178         [lambda, ind] = max(abs(diag(D)));
179         w = V2(:,ind);
180         w = w/norm(w,2);
181
182         % Project onto w
183
184         vdigitXs = w'*digitXs;
185         vdigitYs = w'*digitYs;
186
187         % Make zeros below the threshold
188
189         if mean(vdigitXs) > mean(vdigitYs)
190             w = -w;
191             vdigitXs = -vdigitXs;
192             vdigitYs = -vdigitYs;
193         end
194
195         % Find the threshold value
196
197         sortdigitXs = sort(vdigitXs);
198         sortdigitYs = sort(vdigitYs);
```

```
199
200     t1 = length(sortdigitXs);
201     t2 = 1;
202     while sortdigitXs(t1) > sortdigitYs(t2)
203         t1 = t1 - 1;
204         t2 = t2 + 1;
205     end
206     threshold = (sortdigitXs(t1) + sortdigitYs(t2))/2;
207 end
208
209 function accuracy = accuracy_rate(predict_labels, test_labels, digitX, digitY)
210     predict_two_digits = predict_labels(find(test_labels == digitX |
            test_labels == digitY));
211     test_two_digits = test_labels(find(test_labels == digitX | test_labels ==
            digitY));
212     nCorrect = 0;
213     for i = 1:length(predict_two_digits)
214         if predict_two_digits(i) == test_two_digits(i);
215             nCorrect = nCorrect + 1;
216         end
217     end
218     accuracy = nCorrect / length(predict_two_digits);
219 end
220
221 function images_reshape = reshapeimages(images)
222     [m,~,num_of_images] = size(images);
223     images_reshape = zeros(m^2, num_of_images);
224     for k = 1:num_of_images
225         images_reshape(:,k) = im2double(reshape(images(:,:,k),m^2,1));
226     end
227 end
```