Assignment 4

Classifying Digits

Author: Sofia Chen

Abstract: This assignment is to perform an analysis of the MNIST data set and classify the digits, utilizing and comparing the methods of Singular Value Decomposition (SVD), Linear Discriminant Analysis (LDA), Support Vector Machines, and Decision Trees.

Section I. Introduction and Overview

This assignment is to perform an analysis of digit images in the MNIST data set, which has both the training data and the test data. We convert them into data matrix and reshape each image into a column vector, so each column of the data matrix is a different image. Then I do an SVD analysis of the digit images. I make the singular value spectrum to look for its rank, and project onto three selected V-modes.

After I perform the above, I build a linear classifier to identify individual digits in the training set and quantify the accuracy of the separation with LDA on the test data. And then I will do the same thing applying the SVM and decision tree by using the Matlab built-in functions. Lastly, I will compare the performance between LDA, SVM and decision tree.

Section II. Theoretical Background

The *singular value decomposition* of the matrix A is: $\boldsymbol{A} = \boldsymbol{U\Sigma V}^*$, where $U \in R^{m\times m}$ and $V \in R^{n\times n}$ are unitary matrices, and $\Sigma \in R^{m\times m}$ is diagonal.

The values σ_n on the diagonal of Σ are called the *singular values* of the matrix A. The vectors u_n which make up the columns of U are called the *left singular vectors* of A. The vectors v_n which make up the columns of V are called the *right singular vectors* of A. U and V are unitary matrices that simply lead to rotation and Σ scales an image as prescribed by the singular values

Linear Discriminant Analysis (LDA) is to find a suitable projection that maximizes the distance between the inter-class data while minimizing the intra-class data. We will use μ_1 and μ_2 to denote the column vectors which was the means across each row the data matrix.

We can then define the **between-class** scatter matrix: $S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$. This is a measure of the variance between the groups (between the means).

Then we can define the **within-class** scatter matrix: $S_w = \sum_{j=1}^{2}\sum_x (x - \mu_j)(x - \mu_j)^T$. This is a measure of the variance within each group.

Then we can find a vector w such that $w = argmax\frac{w^T S_B w}{w^T S_w w}$.

It turns out that the vector w that maximizes the above quotient is the eigenvector corresponding to the largest eigenvalue of the generalized eigenvalue problem $S_B w = \lambda S_w w$.

We can solve this use the eig command with Matlab. Once we have w using the above operations, we can choose a cutoff between the two groups of data to be used as a threshold for decision making.

A *support-vector machine* constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier. (Wikipedia)

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. Classification trees give responses that are nominal, such as 'true' or 'false'. Regression trees give numeric responses. (MATLAB Documentation)

Section III. Algorithm Implementation and Development

To start, we build a function *mnist_parse* to load both the training data and the test data in the MNIST data set, which contain the digits and labels. I reshape the data into matrix so that each column is a matrix and cast it into double. We also need to subtract mean of each row from the data. We do this to scale the variance.

Then, we can do the SVD to the training data. After plotting the singular value spectrum, we can find that approximately the first 25 of the singular values is dominating the image. So, we will use the feature (referring the modes) of 25 for the following demonstration.

Next, let us interpret the U, Σ, V matrix from the SVD. We recall that U and V are unitary matrices that simply lead to rotation and Σ scales an image as prescribed by the singular values. Thus, the mixed image can be thought of as first rotating the square via the matrix V, stretching it into a parallelogram with the diagonal matrix Σ, then rotating the parallelogram via the matrix U.

Then, we make a 3D plot by projecting onto three selected V-modes (columns) colored by their digit label. I use columns 2, 3, and 5 here. There are clusters in the plot. In my graph, since we can kind of separate the clusters by eyes, the modes I choose here is fine, which means we can use it to identify the digits.

After we perform the above, I write the LDA function to identify the digits. The algorithm of it is just what I mentioned in the above theoretical part. And then I pick 3 pairs of digits (0,7), (1,2), and (2,3) to implement the LDA. I also write the function *accuracy_rate* to calculate the accuracy rate of my linear classifer by comparing to the original test data. Notice that I use the projection onto principal components here as my testing matrix of the effectiveness of the classifying methods. By computing results, we can find that (0,7) is the pair of digits that easiest to separate, while (2,3) is the pair of digits that hardest to separate.

For the three digits linear classifier, we can separate the digits pair by pair. For example, for digits set (1,2,3), we can first find the threshold in between 1 and 2, and then find the threshold between 2 and 3, same for 1 and 3. Then we can compare the thresholds and then identify the digits.

Then, we perform the SVM and decision tree. By comparing the accuracy rates, we can know that the performance of SVM is almost the best, while the decision tree doing the worst job in contrast.

Section IV. Computational Results

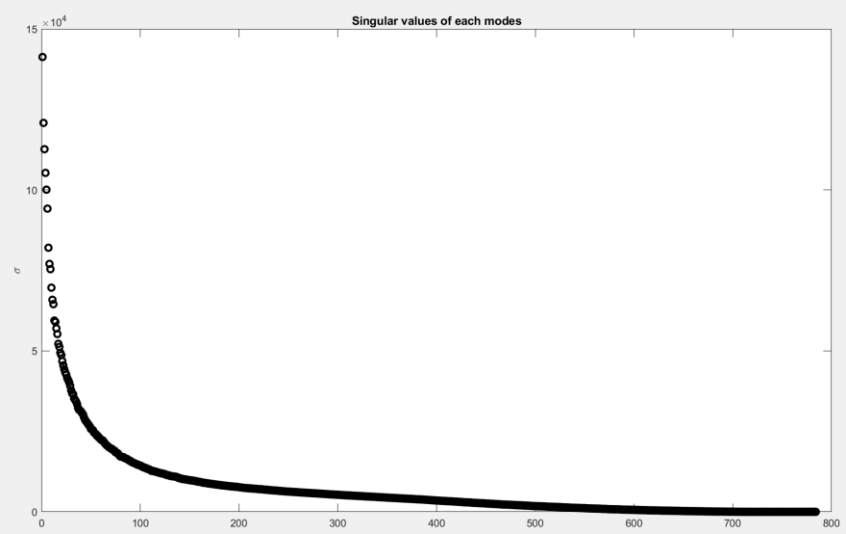

Figure 1: Singular value spectrum of the SVD of the training data.

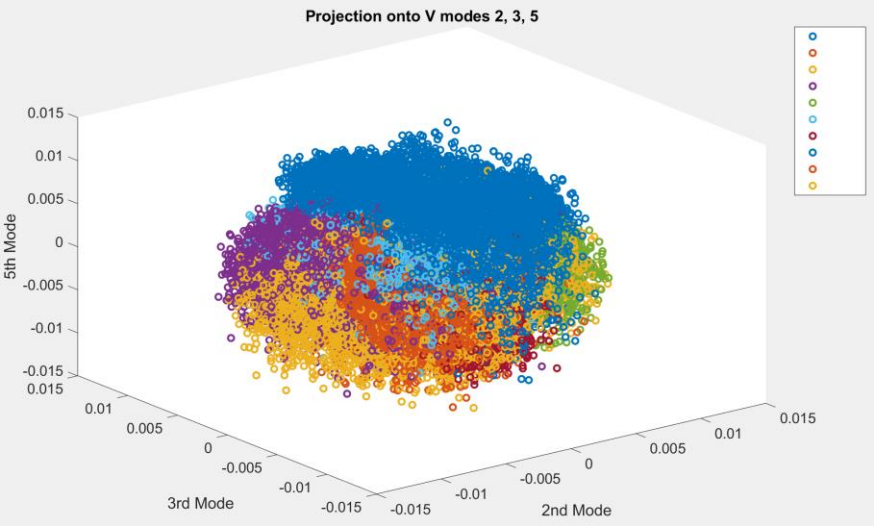

Figure 2: Projection onto three selected V-modes 2, 3, 5 colored by their digit label

Pairs of digits	Algorithm	Accuracy rate
(0,7)	LDA	0.9925
(1,2)	LDA	0.9857
(2,3)	LDA	0.9701
(0,7)	SVM	0.9987
(2,3)	SVM	0.9802
(0,7)	Decision Tree	0.8830
(2,3)	Decision Tree	0.8276

Section V. Summary and Conclusions

We first use the SVD to analysis the data. We get a rank of 25 by looking at the singular value spectrum. And we plot a 3D figure to discuss the relationship between the digit separation and the clusters in the graph. It tells us that V-modes 2, 3, 5 is kind of ok since we can separate the clusters by eyes. Then we build the LDA to identify the digits and compare its performance with SVM and decision tree. By comparing the accuracy rate, we can come to the conclusion that SVM doing the best job in identify the digits, while decision trees the worst.

Appendix A. MATLAB functions used and brief implementation explanation

- ***reshape*** - *Reshape array*: This MATLAB function reshapes A using the size vector, sz, to define size(B).
- ***find*** - *Find requirements in requirements set that have matching attribute values*: This MATLAB function finds and returns an slreq.Requirement object myReq in the requirements set rs specified by the properties matching PropertyName and PropertyValue.
- ***svd*** - *Singular value decomposition*: This MATLAB function returns the singular values of matrix A in descending order.
- ***repmat*** - *Repeat copies of array*: This MATLAB function returns an array containing n copies of A in the row and column dimensions.
- ***fitctree*** - *Fit binary decision tree for multiclass classification*: This MATLAB function 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 ResponseVarName
- ***fitcecoc*** - *Fit multiclass models for support vector machines or other classifiers*: This MATLAB function returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName.

Appendix B. MATLAB codes

Assign3.m:

```
[trainingdata, traingnd] = mnist_parse('train-images', 'train-labels');
numTr = size(trainingdata);
numTr = numTr(3);
trainingdata = double(reshape(trainingdata, size(trainingdata,1)*size(trainingdata,2), []));
traingnd = double(traingnd);

[testdata, testgnd] = mnist_parse('test-images', 'test-labels');
numTe = size(testdata);
numTe = numTe(3);
testdata = double(reshape(testdata, size(testdata,1)*size(testdata,2), []));
testgnd = double(testgnd);

tr_data = trainingdata - repmat(mean(trainingdata,  2), 1, numTr);
te_data = testdata - repmat(mean(testdata,  2), 1, numTe);

[U,S,V] = svd(tr_data, 'econ');
sig = diag(S);
```

```
figure(1)
plot(sig, 'ko', 'Linewidth', 2)
ylabel('\sigma')
title('Singular values of each modes')

figure(2)
for i=0:9
    label_indices = find(traingnd == i);
    plot3(V(label_indices, 2), V(label_indices, 3), V(label_indices, 5), 'o', 'DisplayName',
sprintf('%i',traingnd), 'Linewidth', 2);
    hold on
end
xlabel('2nd Mode'), ylabel('3rd Mode'), zlabel('5th Mode')
title('Projection onto V modes 2, 3, 5')
legend
set(gca,'Fontsize', 14)

UdX = S * V';
test = (U(:,1:rank)')*te_data;
rank = 25;
[threshold,w,sortA,sortB] = lda(UdX, traingnd, rank, 0,7);

accuracy1 = accuracy_rate(test, testgnd, threshold, w, 0, 7)

[threshold,w,sortA,sortB] = lda(UdX, traingnd, rank, 1,2);
accuracy2 = accuracy_rate(test, testgnd, threshold, w, 1, 2)

[threshold,w,sortA,sortB] = lda(UdX, traingnd, rank, 2,3);
accuracy3 = accuracy_rate(test, testgnd, threshold, w, 2, 3)

Mdl = fitcecoc(UdX(1:rank,:)', traingnd);
svm_predict = predict(Mdl,test');
accuracy4 = accuracy_rate2(svm_predict, testgnd, 0, 7)
accuracy5 = accuracy_rate2(svm_predict, testgnd, 2, 3)


tree=fitctree(UdX(1:rank,:)', traingnd);
tree_predict = predict(tree,test');
accuracy6 = accuracy_rate2(tree_predict, testgnd, 0, 7)
accuracy7 = accuracy_rate2(tree_predict, testgnd, 2, 3)
```

accuracy_rate.m:

```
function accuracy_rate = accuracy_rate(test, testgnd, threshold, w, digitA, digitB)
    testA_i = find(testgnd == digitA);
    la = length(testA_i);
    testB_i = find(testgnd == digitB);
    lb = length(testB_i);
    count_right = 0;
    test1 = w'*test;
    i = 0;
    for j = 1:la
        i = i+1;
        if test1(testA_i(i)) < threshold;
            count_right = count_right + 1;
        end
    end
    i = 0;
    for j = 1:lb
        i = i+1;
        if test1(testB_i(i)) > threshold;
            count_right = count_right + 1;
        end
    end
    accuracy_rate = count_right / (la + lb);
end
```

accuracy_rate2.m

```
function accuracy_rate2 = accuracy_rate2(predictgnd, testgnd, digitA, digitB)
    accu = testgnd(find(testgnd == digitA | testgnd == digitB));
    predicts = predictgnd(find(testgnd == digitA | testgnd == digitB));
```

```
    p_length = length(predicts);
    count_right = 0;
    for k = 1:p_length
        if accu(k) == predicts(k)
            count_right = count_right + 1;
        end
    end
    accuracy_rate2 = count_right / p_length;
end
```