Classifying Digits

Paul Abboud

***Abstract*—In this paper we analyze the MNIST data set of images by applying Singular Value Decomposition and projecting into Principal Components. Then we performed Linear Discriminant Analysis to construct a linear classifier that can identify digits. Finally, we compare the constructed LDA to Support Vector Machine and Decision Tree classifiers.**

I. Introduction

The MNIST data set consists of images of digits. In this paper we will perform SVD on the data set to project into Principal Components. Using this information we can determine the number of modes necessary for sufficient image recognition. We will then build Linear Discriminant Analysis (LDA) classifier and determine the accuracy at each digit the model yields when compared to the true values. We then implement Support Vector Machine (SVM) and Decision Tree classifiers to classify the training data. Finally, we examine the identification performance of all three classifiers when compared to the true values to determine the most accurate method.

II. Theoretical Background

To understand the issue at hand, the following will discuss the key ideas behind Singular Value Decomposition, Principal Component Analysis, Linear Discriminant Analysis, and Support Vector Machines and Decision Trees.

A. Singular Value Decomposition

A Singular Value Decomposition decomposes a matrix into three components such that

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

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{m\times n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal. U is a matrix with orthonormal columns, V is an orthonormal matrix and S has non-negative singular values in decreasing order along the diagonal. Their product $A\hat{x}$ represents an orthogonal scaling by $V^T\hat{x}$ then an axis-aligned scaling by $S(V^T\hat{x})$ and finally applying resulting coefficients in the orthonormal basis by $U(S(V^T\hat{x}))$. In this application, we will subtract the row-wise mean from the data set before computing the SVD in order to center the data. This will make the average value of a pixel 0 across all the images.

B. Principle Component Analysis

Given a vector X, we can compute the covariance of the rows with the following formula

$$C_X = \frac{1}{n-1}XX^T \tag{2}$$

This is called the covariance matrix which is a very useful tool in minimizing redundancies in the dataset which is the purpose of PCA. Applying SVD to PCA with the $1/(n-1)$ factor we have

$$C_X = AA^T = U\Sigma^2U^T \tag{3}$$

Multiplying by U^T we have

$$Y = U^TX \tag{4}$$

The covariance of Y can be described in the same fashion as Eq.(2) and Eq.(3) where

$$C_Y = \frac{1}{n-1}YY^T = U^TAA^TU = \Sigma^2 \tag{5}$$

Thus the elements of Y aren't correlated since the only non-zero elements of Σ are along the diagonal. Now that we have uncorrelated information in Y, we have essentially reduced the dimension of the data by eliminating redundancies in the original data set X.

C. Linear Discriminant Analysis

The purpose of LDA is to find a projection of the data which maximizes inter-class data distance but also minimizes intra-class data. Essentially, for this application, we are maximizing the separation between different digits while minimizing the separation between data points of the same digit. Which can be found by computing the vector w where

$$w = argmax\frac{w^TS_Bw}{w^TS_ww} \tag{6}$$

where S_B is the inter-class scatter matrix (measure of variance between classes) and S_w is the intra-class scatter matrix (measure of variance within classes). The following two sums describe S_B and S_w in this application for digits 0 to 9 (10 classifications)

$$S_B = \sum_{j=0}^{9}(\mu_j-\mu)(\mu_j-\mu)^T \tag{7}$$

$$S_w = \sum_{j=0}^{9}\sum^{x}(x-\mu_j)(x-\mu_j)^T \tag{8}$$

where μ is the overall mean and μ_j is the mean of an individual class. Finally, we can compute w by solving the eigenvalue problem

$$S_Bw = \lambda S_ww \tag{9}$$

Now we have computed a basis for the projection which minimizes intra-class distance and maximizes inter-class distance which will help us classify images of digits.

D. Support Vector Machines and Decision Trees

An SVM maps training data to maximize the distance between classifications and assigns new data to the training data group which it is nearest to. It does this by constructing multiple hyperplanes which separate groups of data. The goal is to maximize the distance of the nearest points to the hyperplane. A decision tree evaluates the data point at each node of the tree and determines which branch to follow. Since the classification is discrete in this application (0 to 9), the branches lead to a class label which is a digit in this case.

III. Algorithm Implementation

To begin, we load the test and training data from the MNIST data set using a MNIST parser and the following

```
[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');
```

After loading the data and reshaping it, we can then take the row-wise mean and subtract it from the data set. We can then compute the SVD for Principal Component Analysis.

```
mu = mean(train_data,2);
train_data = train_data - repmat(mu,1,
    length(train_data));

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

The projection is visualized by plotting a 3D scatter plot of the digits depending on three different modes.

```
figure(3);
colormap jet
for i=0:1:9
    index = find(train_labels == i);
    scatter3(V(index,2),V(index,3),V(
        index,5),20,train_labels(index)
        ,'.')
    hold on
end
xlabel('Column 2 of V')
ylabel('Column 3 of V')
zlabel('Column 5 of V')
legend({'0','1','2','3','4','5','6','7
    ','8','9'});
```

After computing the SVD, we can develop an LDA classifier to classify the digits from the training data.

```
train = proj(:,2:10);
test_t = (U'*test_data)';
test = test_t(:,2:10);

class = classify(test,train,
    train_labels);

err = sum(abs(test_labels-class)>0);
acc_lda = 1-err/length(test_labels)
```

Using this method we compare pair-wise digits to determine the least and most accurate digit classification pairs of LDA. Then we implement SVM and Decision Tree classifiers using matlab commands to compare their accuracy in digit classification with LDA. The algorithm for building the SVM can be seen in Appendix B. The following builds the Decision Tree and determines the accuracy.

```
train = proj(:,2:10);
test_t = (U'*test_data)';
test = test_t(:,2:10);
Mdl = fitctree(train,train_labels,'
    OptimizeHyperparameters','auto');

class = predict(Mdl,test);

err = sum(abs(test_labels-class)>0);
accuracy_ct = 1-err/length(test_labels
    )
```

IV. Computational Results

In this application, the SVD has a very intuitive interpretation. U represents a basis for the coordinates of images since the original data set is made up of image coordinates. Σ helps us determine which components are significant by applying a scalar to V^T which is an orthonormal matrix of eigenvectors. Thus $U\Sigma V^T$ is computed by scaling eigenvectors in V^T by their significance in Σ and these weighted components are transformed by U into the original coordinate system where the digits are defined. We found that after applying

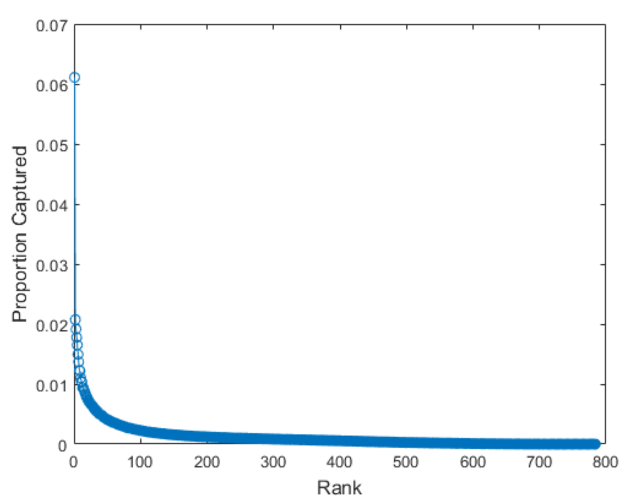

Fig. 1. Energy Captured at Each Dimension in the Image Data

SVD, 90% of the data is captured in 350 modes. This is a significant reduction in dimension when compared to the original data set. We can also see that 60% of the data is captured in the first rank alone in Figure 1. However, Figure 2 illustrates the reconstruction of digit 5 under different ranks. We can see that it only takes about 100 modes to produce a sufficiently clear image. We can then plot the projection of

Fig. 2. Reconstruction of Digit 5 under various Ranks

three V-modes. Figure 3 shows the projection of the 2nd, 3rd and 5th modes. Each digit is colored and we can see digits clustering which is promising for designing our classifiers on training data. However, there are data points mixing together which could present difficulties and reduce the accuracy of our classifiers. We then implemented LDA and tested the number

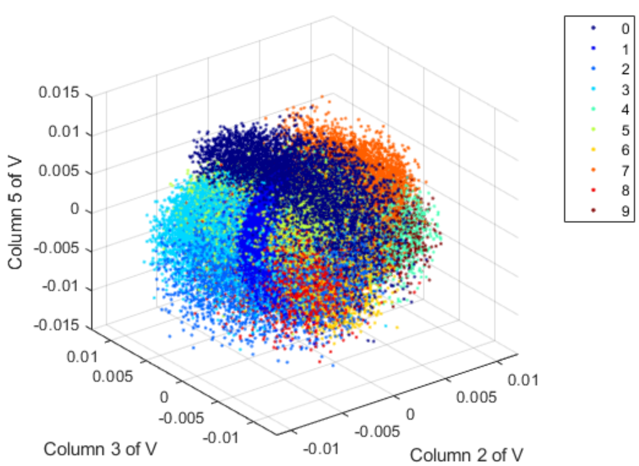

Fig. 3. Projection of modes 2, 3, and 5 and the Resulting Digit Clusters

of V-modes on the classification accuracy. Using digits 0 and 4, we train the LDA classifier and determine the accuracy presented in Figure 4. We can see that at 9 V-modes we have an accuracy of 99.13% which is sufficient. Then we built an LDA for the digits 0, 4 and 7. Figure 5 illustrates the predictions of the constructed LDA. This LDA had an accuracy

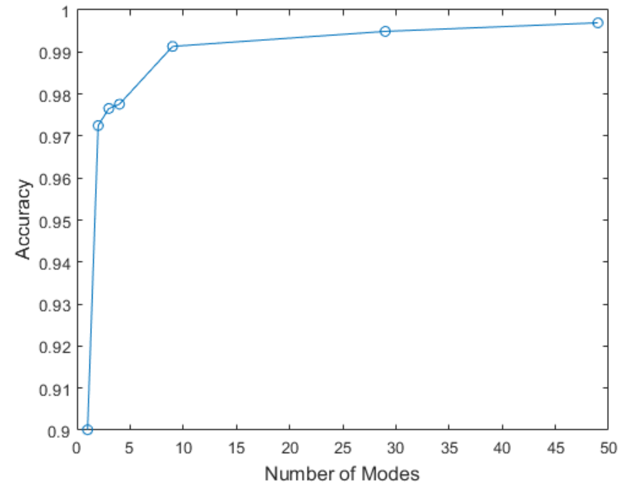

Fig. 4. Two Digit LDA Accuracy

of 96.02% which is highly accurate, but still less accurate when compared to the two digit model. We can identify some inaccuracies from the isolated bars but the majority of digits were classified accurately. Using the trained model, we

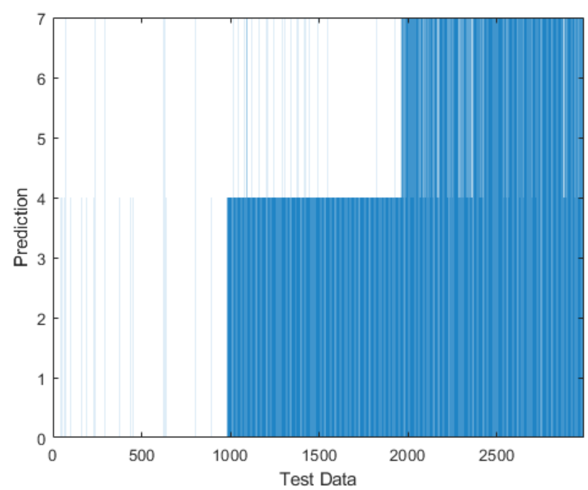

Fig. 5. Three Digit LDA Predictions

determined the accuracy of different digit pairs which is listed in Table 1 and Table 2. Table 1 lists the combinations of digits 0 to 4 with digits 1 to 5. Table 2 lists the combinations of digits 0 to 8 with digits 6 to 9. The digits 4 and 9 had the lowest accuracy of 81.62% meaning 4 and 9 were most difficult to classify. The digits 0 and 1 were easiest to classify with an accuracy of 99.81%. Finally, we built decision tree and SVM classifiers using matlab commands. The decision tree yielded an accuracy of 83.98% while the SVM had an accuracy of 93.32% on the test data. Building an LDA to indentify all digits yielded an accuracy of 75.59%, much lower when compared to the two digit classification. Thus the SVM had the highest accuracy when classifying all digits while the

Digit Class	1	2	3	4	5
0	0.9981	0.9732	0.9794	0.9913	0.9498
1		0.9723	0.9828	0.9934	0.9901
2			0.9589	0.9712	0.9574
3				0.9864	0.8985
4					0.9707

TABLE I
DIGIT PAIR ACCURACY IN PREDICTIONS

Digit Class	6	7	8	9
0	0.9623	0.9895	0.9795	0.9814
1	0.9804	0.9736	0.9445	0.9865
2	0.9266	0.9626	0.9327	0.9691
3	0.9817	0.9637	0.8987	0.9629
4	0.9814	0.9636	0.9693	0.8162
5	0.9492	0.9724	0.8966	0.9548
6		0.9864	0.9731	0.9863
7			0.9401	0.9018
8				0.9319

TABLE II
DIGIT PAIR ACCURACY IN PREDICTIONS

LDA performed the worst. Finally, we compared the LDA, SVM and decision tree models on the hardest and easiest pair found in the LDA classification. The most difficult pair to classify for LDA was 4 and 9. The least difficult pair to classify was 0 and 1. Table 3 lists the accuracy of each model for the specified pairs. We can see that LDA, SVM and decision tree classifiers were all similarly and significantly accurate in classifying 0 and 1 digits. For the more difficult pair, 4 and 9, we see that SVM performed the best, followed by the decision tree and finally LDA was the least accurate in classifying these digits.

V. Summary of Results

In this paper, we applied SVD to significantly reduce the dimension of our data and found that as few as 100 modes could produce identifiable images. We then built an LDA classifier to classify first pairs of digits and then three digits. We found that increasing the number of digit classes lowered the accuracy of the classifier making it inaccurate when classifying all 10 digits with an accuracy of only 75.59%. We then compared the 10 digit LDA classifier to an SVM and decision tree classifier and found that the SVM had the highest accuracy with 93.32%. Finally, we compared the two digit LDA with SVM and decision tree classifiers. We found that all three classifiers had similarly high accuracy when classifying the 0 and 1 digit pair. However, when classifying the 4 and 9 digit pair, we found that the SVM was significantly more accurate than the LDA. Overall, the combination of SVD, PCA and LDA significantly reduced the dimension of data and sufficiently classified images of digits, however, the SVM produced the most accurate data.

	Classifier Accuracy		
Digit Pairs	LDA	SVM	Decision Tree
0 & 1	0.9981	0.9976	0.9948
4 & 9	0.8162	0.9387	0.8769

TABLE III
CLASSIFIERS AND THEIR ACCURACY IN DIGIT PAIRS

Appendix A

MATLAB Functions

classify: Classifies the rows of the sample data into groups depending the training data.

fitctree: Returns a binary decision tree bases on the input and output matrices. The tree nodes and branches are determined by the data.

scatter3: Draws a 3D scatter plot of the data with specified colors.

fitcsvm: Returns an SVM classifier trained using training data and labels.

predict: Returns a vector of predicted class labels for the predicted data.

Appendix B

MATLAB Code

```
close all; clear; clc

[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');


[a,b,c] = size(train_images);
[a2,b2,c2] = size(test_images);
train_data = zeros(a*b, c);
test_data = zeros(a2*b2, c2);
for i=1:1:c
    train_data(:,i)=reshape(
        train_images(:,:,i),a*b, 1);
end
for i=1:1:c2
    test_data(:,i)=reshape(test_images
        (:,:,i),a2*b2, 1);
end

mu = mean(train_data,2);
train_data = train_data - repmat(mu,1,
    length(train_data));

[U,S,V]=svd(train_data,'econ');
proj=(S*V')';
```

```
plot(diag(S)/sum(diag(S)), '-o')
xlabel('Rank','Fontsize',12)
ylabel('Proportion Captured','Fontsize',12)

figure(3);
colormap jet
for i=0:1:9
    index = find(train_labels == i);
    scatter3(V(index,2),V(index,3),V(index,5),20,train_labels(index),'.')
    hold on
end
xlabel('Column 2 of V')
ylabel('Column 3 of V')
zlabel('Column 5 of V')
legend({'0','1','2','3','4','5','6','7','8','9'});

col_v = [2,3,4,5,10,30,50];
acc = zeros(1,length(col_v));
for i=1:1:length(col_v)
    train_1 = proj(train_labels == 0, 2:col_v(i));
    train_2 = proj(train_labels == 4, 2:col_v(i));
    [res_1,y] = size(train_1);
    [res_2,y] = size(train_2);
    train = [train_1; train_2];
    adj_train = [0*ones(res_1,1); 4*ones(res_2,1)];

    test_t = (U'*test_data)';
    test_1 = test_t(test_labels == 0, 2:col_v(i));
    test_2 = test_t(test_labels == 4, 2:col_v(i));
    [res_1,y] = size(test_1);
    [res_2,y] = size(test_2);
    test = [test_1; test_2];
    adj_test = [0*ones(res_1,1); 4*ones(res_2,1)];

    class = classify(test, train, adj_train);

    err = sum(abs(adj_test - class)>0);
    acc(i)= 1-err/length(adj_test);
end
figure(4)
plot(col_v-1, acc, '-o')
xlabel('Number of Modes','Fontsize',12)
ylabel('Accuracy','Fontsize',12)


acc_lda = zeros(10,10);
for i=0:1:8
    for j=i+1:1:9
        train_1 = proj(train_labels == i,2:10);
        train_2 = proj(train_labels == j,2:10);
        [res_1,y] = size(train_1);
        [res_2,y] = size(train_2);
        train = [train_1; train_2];
        adj_train = [i*ones(res_1,1); j*ones(res_2,1)];

        test_t = (U'*test_data)';
        test_1 = test_t(test_labels==i,2:10);
        test_2 = test_t(test_labels==j,2:10);
        [res_1,y] = size(test_1);
        [res_2,y] = size(test_2);
        test = [test_1; test_2];
        adj_test = [i*ones(res_1,1); j*ones(res_2,1)];

        class = classify(test,train,adj_train);

        err = sum(abs(adj_test-class)>0);
        acc_lda(i+1,j+1)=1-err/length(adj_test);
    end
end

train_1 = proj(train_labels == 0,2:10);
train_2 = proj(train_labels == 4,2:10);
train_3 = proj(train_labels == 7,2:10);
[res_1,y] = size(train_1);
[res_2,y] = size(train_2);
[res_3,y] = size(train_3);
train=[train_1; train_2; train_3];
adj_train = [0*ones(res_1,1); 4*ones(res_2,1); 7*ones(res_3,1)];

test_t = (U'*test_data)';
test_1 = test_t(test_labels == 0,2:10);
test_2 = test_t(test_labels == 4,2:10);
```

```
test_3 = test_t(test_labels == 7,2:10);
[res_1,y] = size(test_1);
[res_2,y] = size(test_2);
[res_3,y] = size(test_3);
test = [test_1; test_2; test_3];
adj_test = [0*ones(res_1,1); 4*ones(res_2,1); 7*ones(res_3,1)];

class = classify(test,train,adj_train);

err = sum(abs(adj_test-class)>0);
accuracy_047 = 1-err/length(adj_test)

figure(5)
bar(class)
xlabel('Test Data')
ylabel('Prediction')


train = proj(:,2:10);
test_t = (U'*test_data)';
test = test_t(:,2:10);

class = classify(test,train,train_labels);

err = sum(abs(test_labels-class)>0);
acc_lda = 1-err/length(test_labels)


train = proj(:,2:10);
test_t = (U'*test_data)';
test = test_t(:,2:10);
Mdl = fitctree(train,train_labels,'OptimizeHyperparameters','auto');

class = predict(Mdl,test);

err = sum(abs(test_labels-class)>0);
accuracy_ct = 1-err/length(test_labels)


train = proj(:,2:10)/max(max(S));
test = test_t(:,2:10)/max(max(S));

SVM = cell(10,1);
group = 0:1:9;
rng(1);
for j = 1:numel(group)
    indx = train_labels==group(j);
    SVM{j} = fitcsvm(train,indx,'ClassNames',[false true],'Standardize',true,...
        'KernelFunction','rbf','BoxConstraint',1);
end
for j = 1:numel(group)
    [~,score] = predict(SVM{j},test);
    Scores(:,j) = score(:,2);
end

[~,maxi] = max(Scores,[],2);
err = sum(abs(test_labels+1-maxi)>0);
accuracy_svm = 1-err/length(test_labels)

digits = [4,9];

train_1 = proj(train_labels == digits(1),2:10);
train_2 = proj(train_labels == digits(2),2:10);
[res_1,y] = size(train_1);
[res_2,y] = size(train_2);
train = [train_1; train_2];
adj_train = [digits(1)*ones(res_1,1); digits(2)*ones(res_2,1)];

test_t = (U'*test_data)';
test_1 = test_t(test_labels == digits(1),2:10);
test_2 = test_t(test_labels == digits(2),2:10);
[res_1,y] = size(test_1);
[res_2,y] = size(test_2);
test = [test_1; test_2];
adj_test = [digits(1)*ones(res_1,1); digits(2)*ones(res_2,1)];

rng default
Md2 = fitcsvm(train,adj_train,'Standardize',true,...
    'KernelFunction','rbf','BoxConstraint',1);

class = predict(Md2,test);

err = sum(abs(adj_test-class)>0);
acc_svm_2 = 1-err/length(adj_test)

Md3 = fitctree(train,adj_train,'OptimizeHyperparameters','auto');

class = predict(Md3,test);

err = sum(abs(adj_test-class)>0);
accuracy_ct_2 = 1-err/length(adj_test)
```