Digit Recognition Using Machine Learning Methods

Ben Francis,

March 10, 2021

Abstract

This paper applies three machine learning methods toward hand-written digit classification. Digit recognition is a canonical test for machine learning systems. This paper analyses two elements of machine learning: (1) Data preprocessing and feature selection, and (2) Selection of machine learning model. Principal component analysis is used for feature selection and linear discriminant analysis, support vector machines, and decision trees are used for machine learning models. Accuracies above 90% are achieved for each model while classifying all ten digits, and binary classification between select pairs of digits is explored.

1 Introduction and Overview

Intelligence is a general purpose adaptation that has enabled the continual success of human evolution. The field of machine learning hopes to capture this fundamentally important trait in computational systems. One of the hallmark characteristics of human intelligence is pattern recognition. As such, pattern recognition has become a benchmark problem to focus academic and industrial efforts toward. Classification of hand-written digits has become one of the canonical tasks to exemplify the capabilities of new machine learning systems, due to its relative ease and practical utility. In this paper, three simple yet effective machine learning models will be applied to the problem of hand-written digit recognition. This will exemplify the power of machine learning, a field that is rapidly revolutionizing civilization. The dataset being used for digit classification is the MNIST dataset, a widely used set of digitized handwritten digits already separated out into training and test sets. [3]

2 Theoretical Background

Practical machine learning usually has two major components: data preparation and the machine learning model. While the novel discoveries of machine learning are generally made within the domain of the model, preparing the data is equally vital for success and time consuming for the practitioner.

2.1 Preparing the Data

Data is often not presented in a form ready for input to the machine learning model. However, the MNIST dataset is already fairly well cultivated for machine learning applications, and the provided function `mnist_parse` transforms the given data into arrays for MatLab use. The MNIST data also comes randomized, which would otherwise be important to impose on the data to help ensure the data is independent and identically distributed (IID), an unstated assumption in many machine learning models. This data is then vectorized, such that each row is a feature and each column is a data point. Data is usually multi-dimensional, and so the choice of which features, which are the dimensions of the data, to be used in the machine learning model can be vital. In this case, 28x28 pixel images are provided. While each pixel could be used as a feature, it is often advantageous to perform transformations to the data to construct more informationally-rich features. One common such transformation is dimension reduction via principal component analysis (PCA). This has two benefits. First, the dimensionality of the data is reduced, which reduces computational overhead of the methods and reduces the size of the feature space, thus combating the curse of dimensionality. Second, the features with the highest informational density, with relation to variance of dimensions, can be easily selected out of the data. This will be shown later on in Figure 3, where one principal component clearly contains more information than a pixel's grayscale value.

When performing machine learning, two datasets are used. One is the training set, which represents the information the model is supposed to use to learn, and the second is the test set, which evaluates how well the model has learned. This leaves a slight ambiguity of how to use the PCA method. PCA works by taking the singular value decomposition (SVD) of data, after taking the normal steps such as setting the mean of each data point to zero, which results in three matrices, U, S, and V. U represents the principal components, S^2 represents the variance captured by each principal component, and V describes the combination of principal components that constitute each data point. In order to perform dimension reduction, only a subset of each matrix is used, corresponding to the first r elements of S, or the first r principal components. The strategy is to use the first r columns of V, thus representing the training data by its linear combination of principal components. The choice of how many columns is determined by analyzing the singular value spectrum given by S. The machine learning model is then trained on this transformed data. As such, when it is time to test, the test data must be projected onto these same principal components, achieved by multiplying $U' * test$. This is a common choice of feature selection, due to the advantageous mathematical properties of PCA.

2.2 Machine Learning Models

Machine learning methods come in a variety of types, two of the most common being supervised and unsupervised methods. Supervised methods require labeled data while unsupervised methods do not. While unsupervised methods are of great importance to machine learning, they will not be covered here. For supervised learning, the goal is to construct a model from labeled training data that can accurately label test data from a similar distribution. There are multiple pitfalls associated with supervised learning, some of which will be described here. First, it is always possible to have a model that works well for one particular training and test set but not others. Cross-validation is the process by which training and test sets are repeatedly randomly selected from one total dataset, and the modeling process is conducted for each random selection. These results can be analyzed to prevent the possibility of getting lucky. Closely related is the problem of overfitting, whereby the model only memorizes the pattern of the training set, instead of learning the underlying key patterns of the distribution that would allow for generalization to unseen data. The sign of this is a model that performs well on the training set but poorly on the test set. Remedying this problem is one of the major pillars of current machine learning work. The simplest solutions are to either limit the models capacity to memorize or to use more data, thus making it more difficult to memorize, and so to encourage learning patterns. Cross-validation will not be done here, as the purpose here is primarily to demonstrate the different models.

The goal of supervised learning is usually either classification or regression. In classification, test data is labeled by discrete classes, whereas in regression test data is assigned a continuous number. Only classification will be considered here, fitting with the nature of the digit recognition task. If the model chooses between two classes it is considered binary classification, while if the model chooses between more than two classes it is multiclass classification. [1]

2.2.1 Linear Discriminant Analysis

The goal of linear discriminant analysis (LDA) is to find a lower-dimensional embedding to project the data onto, such that the data is clearly separable afterward. Linear discriminant analysis, as opposed to quadratic discriminant analysis, imposes linearity on this embedding. For example, in two-dimensional data this amounts to choosing an embedding line such that the data projected onto the line are clustered by class. These clusters can then be separated by a new classification line, perpendicular to the projection line and midway between the two clusters. This classification line divides data into different classes, such that new data either falls into one class or another. This lower-dimensional example extends similarly to higher dimensions, and generalizes to multiclass classification. The LDA algorithm selects the projection embedding by optimizing to maximize the distance between class means and to minimize the distance within classes to their mean position. [2]

2.2.2 Support Vector Machines

Support vector machines (SVM) work by selecting a hyperplane that is furthest from the boundaries of each class. In this manner, significantly less data is needed since it is only the boundaries of the classes that determine the behavior of the classification. In non-linear SVMs, this separation boundary is not restricted to linearity. With kernel methods, this is expanded further. SVMs are extremely effective and a tool of choice in the absence of enough data to implement neural networks. [2]

2.2.3 Decision Tree Classification

Decision trees, along with SVMs, are among the best non-deep learning methods for machine learning. These operate by designing a flow chart to determine the class of each test data point, such that a series of binary "yes/no" choices are made. These choices are based upon inequalities along the dimenions of the feature space. Choosing the points where the higher-or-lower decisions should be made for each dimension is the basis of the algorithm. This method also lends itself naturally toward visualization, which can be important in many applied machine learning settings. The number of decisions allowed can be restricted to prevent overfitting.

3 Algorithm Implementation and Development

Algorithm 1: Data Pre-Processing

```
Data: Set of MNIST training images and test images
Result: Training data and test data
Reshape individual images to columns;
Square s values to get % variances;
Plot % variances to estimate how many modes needed;
Plot principal components of training data;
Plot projection of certain modes;
Use V matrix with indexed rows as new training data;
Project test data onto U' matrix as new test data;
for each train, test data point do
    Sort into digits
Create new train, test data and labels for each digit pair possibility;
```

Algorithm 2: Machine Learning Methods

```
Data: Total train, test data, labels, and all combinations of two digits
Result: Predictions and accuracy for each method
Create LDA model for each of 45 possible digit pairs;
Create LDA model for 0, 1, 4 digit triple;
Create LDA, SVM, and decision tree models for all 10 digits;
Create LDA model for best digit pair, 0 and 1, and worst, 7 and 9;
Create SVM model for best digit pair, 0 and 1, and worst, 7 and 9;
Create decision tree model for best digit pair, 0 and 1, and worst, 7 and 9;
Predict each of above models performance on corresponding test data;
Calculate accuracies of each above predictions;
```

Preprocessing

First, the training and test data is loaded in as 3D tensors using the provided `mnist_parse` function. Both are reshaped, converting each $28x28$ image into a $784x1$ vector. Both datasets are now 2D matrices, with each column representing a different image. The mean of each column is calculated and subtracted from each column, so that the data has zero mean. Additionally, the training data is divided by $\sqrt{n-1}$, where n is the length of each which is important for the SVD process. The SVD is then taken for the training data. The squared diagonal of the S matrix gives the percent variances of each principal component. This is plotted as a percentage of the sum of the diagonal, which gives its percentage of the total variance. The first three principal components for each image are then plotted and colored by digit to visualize how separable the images are by their highest-variance principal components. It is visually apparent that the images cluster by digit. The first four principal components, being the first four columns of the U matrix, are visualized as images.

It is desired to keep 90% of the variance, which is calculated to be captured by the first 87 modes, so these will be used as the principal components. This represents an 89% reduction of dimensionality. The V matrix determines the combination of principal components used for each data point, weighted by the singular values. As such, the first 87 modes represent a lower-dimensional version of the training data that still captures 90% of the variance. The test data is projected onto the transposed U matrix, to give its projection onto the principal components, with only the first 87 modes used as well.

In addition to being interested in the ability of classifiers to distinguish between all digits, it is of interest to determine how they work when only distinguishing between different pairs of digits. To do this, the training and test data is manually sorted through by digit type, using the fact their labels are given. These are then recombined into 45 new test and train datasets, each having images from the $\frac{10*9}{2}$ different pair permutations. The `classify` function is used to predict the values of the test data on the LDA model constructed from the training data and labels. The results for each pair can be compared with their corresponding labels to determine the accuracy. The pair with the best results was found to be 0 and 1 while the pair with the worst results was found to be 7 and 9. A dataset with three digits was also used, selecting 0, 1, and 4.

Machine Learning Methods

The LDA, SVM, and decision tree models were then constructed with the dataset comprised of all ten digits. This was done using the `fitcdiscr`, `fitcecoc`, and `fitctree` methods, respectively, which each take in the training data and labels. The `predict` method is then used, which takes in the model and the test data as parameters, to give the predicted digit labels. A confusion matrix was made for each model using `confusionchart`, which takes in the test labels and the prediction labels, to show its performance across categories, as well as showing what the errors were misclassified as. Then each method is used on the easiest pair of digits to separate and the hardest pair to separate. This uses the same functions, except now the datasets that are only comprised of two digits, except for the SVM model. Binary classification with SVM, as opposed to multiclass classification, uses the `fitcsvm` method, with the same parameters. These classifiers are then also tested with the training data, to compare the performance on training data versus test data.

4 Computational Results

Data Preprocessing

The squared singular values show the captured variance in each principal component. These are plotted as percentages of the total variance in Figure 1 to get a sense of how many modes are needed. As a heuristic, modes will be used such that they capture 90% of the variance. This was calculated to be the first 87 modes. There will always be a tradeoff between dimensionality reduction and performance quality. Each model used was done relativley quickly with relatively high performance, so it is considered that this selection was appropriate. The first four principal components, given by the first four columns of the U matrix, are reshaped and plotted in Figure 2 for visual intuition of the process and the results are similar to what is expected. The first three principal components for each image give a point in 3D space, and so plotting each of these points gives a heuristic visualization of the separability of the data by principal components. This is shown in Figure 3, where each digit is given a unique color. It is seen that they are well-separable by visual inspection, which gives good evidence that the principal components represent good features for the machine learning methods.

Machine Learning

Using the LDA method, all 45 pairs of digits are examined. The average accuracy was 94.8%. The pair with the highest accuracy was 0 and 1, and the pair with the lowest accuracy was 7 and 9. These represent the pairs with the best and worst separability. This is in alignment with intuitive expectation, since 0 and 1 are both simple shapes that share no similar components, while 7 and 9 are only differentiated by a small loop at the top. Additionally, a dataset with three digits, being 0, 1, and 4, was done, with an accuracy of 98.61%. Next all three methods were tried on all ten digits. These results can be summarized by confusion matrices, which has as its vertical axis the true classes and its horizontal axis the predicted classes. The values in each box represent the number of times that a member of the true class was predicated as the predicted class. Then the diagonal represents the accurate classifications. These are shown in Figure 4. From these matrices, it can be seen that the SVM performs best, the LDA second best, and the decision tree worst.
Each method was then tried on the most and least separable digits, being 0 and 1, and 7 and 9. For 0 and 1, the accuracy for LDA was 99.9%, for SVM was 99.8%, for the decision tree was 28.9%. For 7 and 9, the accuracy for LDA was 78.1%, for SVM was 91.7%, and for the decision tree was 43.8%. While the LDA and SVM methods achieved good scores, the decision tree achieved very poor results, even upon careful inspection that each method was being trained and tested identically. To examine this discrepancy, each method was examined for overfitting. This was done by feeding the training data into the prediction with each model, and seeing the disparity between this result and with the test data. For 0 and 1, this gave 99.6% for LDA, 98.4% for SVM, and 99.9% for the decision tree. For 7 and 9, this gave 95.5% for LDA, 86.3% for SVM, and 99.0% for the the decision tree. Then the problem with the decision tree is clearly overfitting. If the number of splits is limited to 3, then the new accuracies for the decision tree with 0 and 1 become 92.0% and 43.8%, which is a noticeable increase. Adjusting the number of splits further did not markably improve this result. This highlights the dangers of overfitting, and how to remedy them by restricting the capacity of the model.

5 Summary and Conclusions

Digit recognition was performed well by all three methods examined, with the order of success being: support vector machines, linear discriminant analysis, and decision trees. Binary classification was also examined for each pair of digits and one set of three digits. The success and difficulties of the methods followed along the lines of what human intelligence would intuitively expect, namely the distinguishability of 0 and 1 and the similarity of 7 and 9.

Instead of using raw pixel data as features, principal components were used to reduce the dimensionality of the problem to the components that contained the highest variance. The separability of the data in this space was clearly visualizable in the space of the first three principal components, and visual inspection of these components fell in line with what might be expected.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* URL: `https://www.deeplearningbook.org/`.

[2] Nathan Kutz and Steve Brunton. *Data-driven Science and Engineering.* Oxford University Press, 2019.

[3] *MNIST Dataset.* URL: `http://yann.lecun.com/exdb/mnist/`.

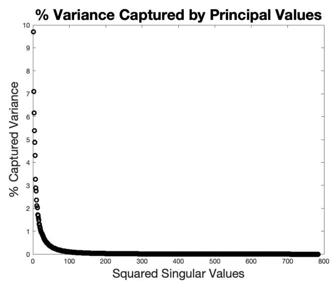


Figure 1: Percent variance captured by each principal component.

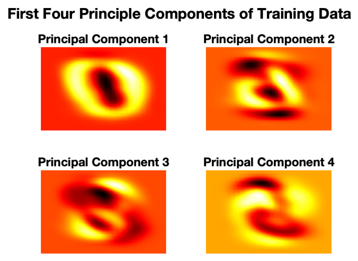


Figure 2: Visualization of first four principal components of training data.

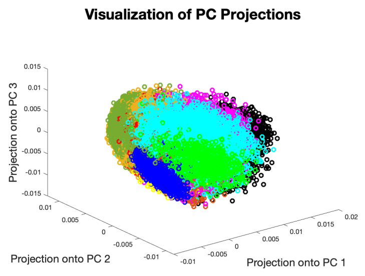
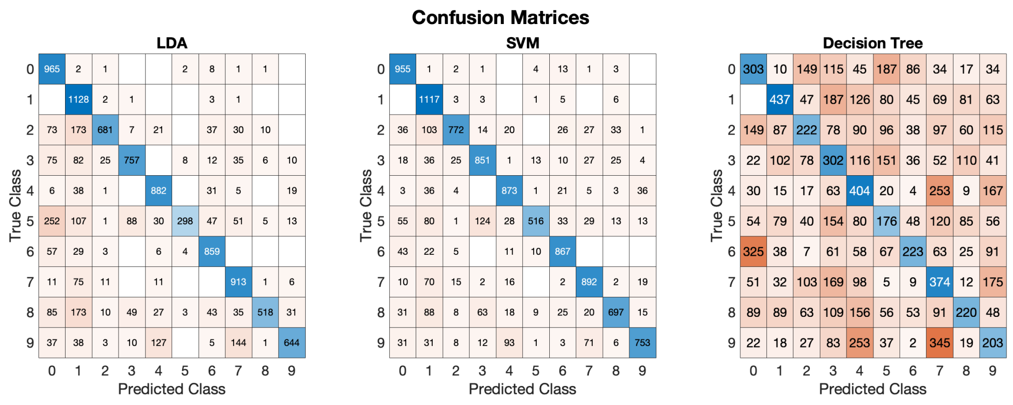


Figure 3: Visualization of image separability by principal components.



LDA

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	965	2	1			2	8	1	1	
1		1128	2	1			3	1		
2	73	173	681	7	21		37	30	10	
3	75	82	25	757		8	12	35	6	10
4	6	38	1		882		31	5		19
5	252	107	1	88	30	298	47	51	5	13
6	57	29	3		6	4	859			
7	11	75	11		11			913	1	6
8	85	173	10	49	27	3	43	35	518	31
9	37	38	3	10	127		5	144	1	644

SVM

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	955	1	2	1		4	13	1	3	
1		1117	3	3		1	5		6	
2	36	103	772	14	20		26	27	33	1
3	18	36	25	851	1	13	10	27	25	4
4	3	36	4		873	1	21	5	3	36
5	55	80	1	124	28	516	33	29	13	13
6	43	22	5		11	10	867			
7	10	70	15	2	16		2	892	2	19
8	31	88	8	63	18	9	25	20	697	15
9	31	31	8	12	93	1	3	71	6	753

Decision Tree

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	303	10	149	115	45	187	86	34	17	34
1		437	47	187	126	80	45	69	81	63
2	149	87	222	78	90	96	38	97	60	115
3	22	102	78	302	116	151	36	52	110	41
4	30	15	17	63	404	20	4	253	9	167
5	54	79	40	154	80	176	48	120	85	56
6	325	38	7	61	58	67	223	63	25	91
7	51	32	103	169	98	5	9	374	12	175
8	89	89	63	109	156	56	53	91	220	48
9	22	18	27	83	253	37	2	345	19	203

Figure 4: Confusion matrices for LDA, SVM and decision tree methods, evaluated on the test data.

Appendix A MATLAB Functions

- `pred = classify(test,train,train_labels)`: Makes predictions on the labels of test data based upon an LDA model made using the train data and corresponding labels.
- `lda_model = fitcdiscr(train,labels)`: Creates an LDA model using training data and corresponding labels.
- `svm_model = fitcecoc(train,labels)`: Creates a multi-class SVM model using training data and corresponding labels.
- `tree_model = fitctree(train,labels)`: Creates a decision tree model using training data and corresponding labels.
- `svm_model = fitcsvm(train,labels)`: Creates a binary-classification SVM model using training data and corresponding labels.
- `label = predict(model,test)`: Uses a model to predict labels for test data.
- `confusionchart(trueLabels,predictedLabels):` Creates a confusion matrix, which visually shows how the predicted labels compare to the true labels.

Appendix B MATLAB Code

See next page. Github: https://github.com/benfrancis314/AMATH582

```
[images_train, labels_train] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-ubyte');
[images_test, labels_test] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte');

train_vec = reshape(double(images_train), 28*28, []);
test_vec = reshape(double(images_test), 28*28, []);
[m_train, n_train] = size(train_vec); [m_test, n_test] = size(test_vec);
mn_train = mean(train_vec,2);  mn_test = mean(test_vec,2);
train_vec_mean = train_vec-repmat(mn_train,1,n_train);
test_vec_mean = test_vec-repmat(mn_test,1,n_test);
[u_train,s_train,v_train] = svd(train_vec_mean/sqrt(n_train-1),'econ');
lambda_train = diag(s_train).^2;

figure(1); plot(100*(lambda_train / sum(lambda_train)) ,'ko','LineWidth',2);
title("% Variance Captured by Principal Values","FontSize",22, "FontWeight", "bold");
ylabel("% Captured Variance","FontSize",18); xlabel("Squared Singular Values","FontSize",18);
for j=1:4
    sing_vals_of_interest = [1,2,3,4];  subplot(2,2,j);
    ef = reshape(u_train(:,sing_vals_of_interest(j)),28,28);
    pcolor(ef), axis off, shading interp, colormap(hot);
    title(["Principal Component "+ j],"FontSize",18);
end
sgtitle("First Four Principle Components of Training Data","FontSize",22, "FontWeight", "bold");
label_colors = [0 0 0; 0 0 1; 0 1 0; 0 1 1; 1 0 0; 1 0 1; 1 1 0; 0.4660 0.6740 0.1880; ...
    0.8500 0.3250 0.0980; 0.9290 0.6940 0.1250];
num_points = 60000; sing_vals = [1,2,3];
for i=1:num_points
    digit_color = label_colors(labels_train(i)+1,:);
    plot3(v_train(i,1),v_train(i,2),v_train(i,3),'o-','Color',digit_color,'LineWidth',2); hold on;
end
xlabel('Projection onto PC 1', 'FontSize', 16), ylabel('Projection onto PC 2', 'FontSize', 16), ...
    zlabel('Projection onto PC 3', 'FontSize', 16);
title('Visualization of PC Projections', 'FontSize', 22);
r = 87; sum(lambda_train(1:r))/sum(lambda_train) % Percent variance from 87 modes
train_proj = [v_train(:,1:r)]; test_proj_all = (u_train'*test_vec_mean).';
test_proj = [test_proj_all(:,1:r)];
train_0 = []; test_0 = []; ... train_9 = []; test_9 = [];
for i=1:length(labels_train)
    if labels_train(i) == 0
        train_0 = [train_0; train_proj(i,:)];
    end ...
end
for i=1:length(labels_test)
    if labels_test(i) == 0
        test_0 = [test_0; test_proj(i,:)];
    end ...
end

train_01 = [train_0; train_1]; test_01 = [test_0; test_1];
...
train_89 = [train_8; train_9]; test_89 = [test_8; test_9];
labels_train_01 = [0*ones(length(train_0),1); 1*ones(length(train_1),1)]; labels_test_01 = [0*ones(leng
...
labels_train_89 = [8*ones(length(train_8),1); 9*ones(length(train_9),1)]; labels_test_89 = [8*ones(leng
```

Listing 1: Code to preprocess data.

```
pred_01 = classify(test_01, train_01, labels_train_01);
...
pred_89 = classify(test_89, train_89, labels_train_89);

correct_01 = 0;
...
correct_89 = 0;

for i=1:length(pred_01)
    if pred_01(i) == labels_test_01(i)
        correct_01 = correct_01 + 1;
    end
end
...
for i=1:length(pred_89)
    if pred_89(i) == labels_test_89(i)
        correct_89 = correct_89 + 1;
    end
end

accuracy = zeros(45,1);

accuracy(1) = correct_01/length(labels_test_01);
...
accuracy(45) = correct_89/length(labels_test_89);

t = 1:45;
plot(t,accuracy);

[~, worst] = min(accuracy); % 7 and 9
[~, best] = max(accuracy); % 0 and 1
mean(accuracy) % Average accuracy
```

Listing 2: Code for LDA on all possible pairs.

```
train_014 = [train_0; train_1; train_4]; test_014 = [test_0; test_1; test_4];
labels_train_014 = [0*ones(length(train_0),1); 1*ones(length(train_1),1);4*ones(length(train_4),1)];
labels_test_014 = [0*ones(length(test_0),1); 1*ones(length(test_1),1);4*ones(length(test_4),1)];
pred_014 = classify(test_014, train_014, labels_train_014);

correct_014 = 0
for i=1:length(pred_014)
    if pred_014(i) == labels_test_014(i)
        correct_014 = correct_014 + 1;
    end
end

accuracy_014 = correct_014/length(labels_test_014);
confusionchart(labels_test_014,pred_014);
```

Listing 3: Code for LDA on three digits.

```
lda = fitcdiscr(train_proj,labels_train);
ldaClass_test = predict(lda,test_proj);
svm = fitcecoc(train_proj,labels_train);
svmClass_test = predict(svm,test_proj);
tree = fitctree(train_proj,labels_train);
treeClass_test = predict(tree,test_proj);

figure(1)
subplot(1,3,1), cm1 = confusionchart(labels_test,ldaClass_test,'FontSize',16);
cm1.Title = 'LDA';
subplot(1,3,2), cm2 = confusionchart(labels_test,svmClass_test,'FontSize',16);
cm2.Title = 'SVM';
subplot(1,3,3), cm3 = confusionchart(labels_test,treeClass_test,'FontSize',16);
cm3.Title = 'Decision Tree';
sgtitle("Confusion Matrices",'FontSize',22, 'FontWeight','bold');
```

Listing 4: Code for LDA, SVM, and decision tree on all digits.

```
%  Best
lda_01 = fitcdiscr(train_01,labels_train_01);
svm_01 = fitcsvm(train_01,labels_train_01);
tree_01 = fitctree(train_01,labels_train_01,'MaxNumSplits',3);
lda_pred_01 = predict(lda_01, test_01);
svm_pred_01 = predict(svm_01, test_01);
tree_pred_01 = predict(tree_01, test_01);

% Worst
lda_79 = fitcdiscr(train_79,labels_train_79);
svm_79 = fitcsvm(train_79,labels_train_79);
tree_79 = fitctree(train_79,labels_train_79);
lda_pred_79 = predict(lda_79, test_79);
svm_pred_79 = predict(svm_79, test_79);
tree_pred_79 = predict(tree_79, test_79);

correct_lda_01 = 0; correct_svm_01 = 0; correct_tree_01 = 0;
for i=1:length(lda_pred_01)
    if lda_pred_01(i) == labels_test_01(i)
        correct_lda_01 = correct_lda_01 + 1;
    end
    if svm_pred_01(i) == labels_test_01(i)
        correct_svm_01 = correct_svm_01 + 1;
    end
    if tree_pred_01(i) == labels_test_01(i)
        correct_tree_01 = correct_tree_01 + 1;
    end
end

accuracy_lda_01 = correct_lda_01/length(labels_test_01)
accuracy_svm_01 = correct_svm_01/length(labels_test_01)
accuracy_tree_01 = correct_tree_01/length(labels_test_01)

correct_lda_79 = 0; correct_svm_79 = 0; correct_tree_79 = 0;
for i=1:length(lda_pred_79)
    if lda_pred_79(i) == labels_test_79(i)
        correct_lda_79 = correct_lda_79 + 1;
    end
    if svm_pred_79(i) == labels_test_79(i)
        correct_svm_79 = correct_svm_79 + 1;
    end
    if tree_pred_79(i) == labels_test_79(i)
        correct_tree_79 = correct_tree_79 + 1;
    end
end

accuracy_lda_79 = correct_lda_79/length(labels_test_79)
accuracy_svm_79 = correct_svm_79/length(labels_test_79)
accuracy_tree_79 = correct_tree_79/length(labels_test_79)
```

Listing 5: Code for LDA, SVM, and decision trees on best and worst pairs.

```
function [images, labels] = mnist_parse(path_to_digits, path_to_labels)

fid1 = fopen(path_to_digits, 'r');

% The labels file
fid2 = fopen(path_to_labels, 'r');

A = fread(fid1, 1, 'uint32');
magicNumber1 = swapbytes(uint32(A)); % Should be 2051
fprintf('Magic Number - Images: %d\n', magicNumber1);

A = fread(fid2, 1, 'uint32');
magicNumber2 = swapbytes(uint32(A)); % Should be 2049
fprintf('Magic Number - Labels: %d\n', magicNumber2);

A = fread(fid1, 1, 'uint32');
totalImages = swapbytes(uint32(A));
A = fread(fid2, 1, 'uint32');
if totalImages ~= swapbytes(uint32(A))
    error('Total number of images read from images and labels files are not the same');
end
fprintf('Total number of images: %d\n', totalImages);

A = fread(fid1, 1, 'uint32');
numRows = swapbytes(uint32(A));

A = fread(fid1, 1, 'uint32');
numCols = swapbytes(uint32(A));

fprintf('Dimensions of each digit: %d x %d\n', numRows, numCols);

images = zeros(numRows, numCols, totalImages, 'uint8');
for k = 1 : totalImages
    A = fread(fid1, numRows*numCols, 'uint8');

    images(:,:,k) = reshape(uint8(A), numCols, numRows).';
end

labels = fread(fid2, totalImages, 'uint8');

fclose(fid1);
fclose(fid2);

end
```

Listing 6: Code to parse mnist data into an array for MatLab use.