AMATH 482 Homework 4:
Classifying Digits

Malikah Nathani
March 10th, 2021

Abstract

The objective of this paper is twofold: first, to analyze a dataset with images of unique digits from zero to nine by projecting it into the PCA space; and second, examine the practical implementation of Linear Discriminant Analysis by building a system to identify different digits. We use our analyses to determine which digits are easier and harder for the computer to make out. Furthermore, we compare the LDA method to other commonly known machine learning algorithms such as decision trees and support vector machines (SVM) to understand the true importance of the LDA.

1 Introduction and Overview

Linear Discriminant Analysis (LDA) is vast area of study with numerous applications ranging from biomedical studies to marketing. In the context of this paper, we will be applying this analytical method to the realm of object recognition, specifically a digit. Teaching a machine to recognize what a digit is can serve as a basis for more complicated problems, enhancing the ability of machine learning through LDA.

In this paper, we are given two datasets, a training and test set, containing images of digits ranging from zero to nine. First, we are tasked with analyzing this dataset by performing the PCA algorithm to understand linear combinations that best explain the dataset. Then, using that information we are asked to use the LDA to reasonably identify two digits, then three. We also examine which two digits appear to be easier and harder for the LDA to differentiate. Finally, we are asked to compare this method to other well-known ML algorithms to garner a better understanding of this analysis as well as learn its limitations and strengths.

In Section 2, I will go about explaining the LDA in more detail while section 3 will go over the implementation of this method and the algorithm used to solve the problem. Sections 4 and 5 will go over the computed results and my conclusions respectively. Finally, I will also present the MATLAB functions used and provide my code.

2 Theoretical Background

2.1 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a commonly used data analytics tool to extract linear combinations that describe specific features from a dataset to ultimately separate objects. The goal of the LDA is to minimizes the intra-object data while maximizing the inter-object data thus clearly identifying what is each element.

In this problem, before using the LDA we want to reduce our dimensionality thus saving our computations. We use the SVD to achieve this. In the SVD, the U matrix represents the basis vectors making up the image while the Σ and V matrices represent how much of each basis is actually represented in the data. Once reduced, we can define the between-class scatter matrix as:

$$S_B = \sum_{j=1}^{N} (\mu_j - \mu)(\mu_j - \mu)^T$$

Where μ is the overall mean and μ_j is the mean of each of the groups. It is important to note that μ is a column matrix as it is the mean across each row. The within-class scatter matrix measures the variance within each group.

$$S_W = \sum_{j=1}^{N} \sum_{x} (x - \mu)(x - \mu)^T$$

Afterwards, we find a vector w such that:

$$w = argmax \frac{w^T S_B w}{w^T S_W w}$$

This can also be done by finding the eigenvector corresponding to the largest eigenvalue of the following:

$$S_B w = \lambda S_W w$$

With the w vector, we are now able to project our data onto w by multiplying each group by w' and thus determine a threshold using the mean of our new projected classes for our decision making.

2.2 Support Vector Machine (SVM)

The goal of SVM is to find a hyperplane with X-dimensions, where X is the number of features, that best separates the data points. Unlike the LDA, which solely uses a linear relationship to classify data, SVM can be done linearly and non-linearly.

2.3 Decision Tree Classifier

A decision tree classifier is a tree-like structure where each node in the tree is a test on the attribute of the data. Whether it does or does not meet that check decides what check will be next and ultimately going through a specific path leads to one decision of the outcome. Therefore, the decision tree is similar to the LDA where is uses data of the features as a whole to understand the outcome, unlike LDA however, it is non-linear.

3 Algorithm Implementation and Development

This section details the implementations of the methods above and any other steps taken to solve this problem in MATLAB. For the actual code used, please see Appendix B.

Our first step is to load in our images and its labels for the training and test data using our created function. As our images are in a 3-dimensional matrix, we reshape each image into a column vector and add it to matrix containing a column vector for each image. Since the images and labels are not sorted in order in numerical order, we do so using the `sort` function to be able to easily call on specific digits for analysis later. We repeat this process for both the training and the test images and labels.

Afterwards, we calculate the SVD of our matrix containing our training pictures. To determine the necessary modes for good image recreation, we determine the cumulative energies contained in the singular values. Then, we visualize the photo recreation at those different energy values to determine at which rank there is not much difference to the original image. In our calculations, 95% energy is reached at the 103rd mode.

To project our data onto selected U-modes for each digit, we first use our labels vector to determine the start and end index for each digit. We can then multiply our U' matrix calculated in the SVD with our data matrix cut at those indices to project the data of that digit onto V. We choose columns 2,3, and 4 of U to project our data onto. We can then use the plot3 command to visualize, this can also show which digits have more or less similar features which may allow us to determine which digits are easier or harder for our LDA to separate between.

We then initialize a `sample` and `training` matrix as well as a `group` vector that contains information about each of the digits we want to differentiate between. Both of the matrices will have columns containing the number of features we will use (103) and the rows will be of the digits we are differentiating between. For example, in our two-digit LDA for our `training` matrix, the first 6000ish rows will be of one number and the next 6000ish rows will be of the other digit. I use the form 6000ish because in our dataset each digit does not have exactly 6000 pictures, therefore sorting the labels in order helps to see the index of the first and last instance of that digit to only use those values. Our `sample` matrix is of our test data while our `training` matrix is of our training data. It is important to note that when we are determining the values for our matrices, we are not simply using the matrices themselves, we are first transforming the datasets to onto a different basis (principal components) by multiplying U' (from SVD calculation) to our data matrix. The `group` vector contains information about which digit is represented in each row in the `training` matrix.

To calculate our LDA, we use the `classify` command to expedite our process (syntax for this method is in more detail in Appendix A). For our two-digit classification, I chose the values 1 and 8 as I felt the shape of these digits are unique, while in our three-digit classification I used the same reasoning and chose the numbers 1, 8, and 5. As we found that 95% of the energy in the data is captured in mode 103, we use this as the number of features we differentiate our numbers with as the first 103 will capture the most variance. We use our function to generate the digit that each row in `sample` represents, it also returns the error from our calculations. Since we also want to see the LDA performance on our training dataset, we simply replace the `sample` matrix with `training` to achieve our goal. We note that we keep updating `sample`, `training`, and `group` depending on which and how many digits we are differentiating between.

To understand which digits may be the hardest and easiest to separate, we look to the plot we generated before. Digits that have points further apart, have features that are dissimilar therefore it is easier for the LDA to distinguish between them, and vice versa. Therefore, looking at our plot, we notice that digits 7 & 9, 4 & 9, and 4 & 7 seem extremely close on the plot, therefore we find the error of the LDA on the training and test dataset for each pair to quantify our separation abilities to determine which is the hardest to separate. We also notice that digits 0 & 1, 2 & 6, and 0 & 3 do not seem to overlap and are pretty far apart, therefore we run and LDA on these two digits and calculate the error to find the easiest to separate.

Another goal of this assignment is to compare the LDAs performance to other state-of-the-art classifiers such as SVM and decision tree. We want to compare the three on being able to separate all the ten digits as well as the two easiest and hardest to separate found in the LDA.

For our decision tree, we use the `fitctree` command and determine the error on our tree using `kfoldloss`. We specify a maximum number of splits as 10 to not overfit the tree. For our SVM, we determine our model using the `fitcecoc` command and determine our digits using the `predict` method. To determine the error using SVM, we can add an if statement checking if the labels generated match the actual labels for each row in our `sample` matrix, and then calculate how many were not correct.

4 Computational Results

Type of Classifier	**Digits Chosen**	**Error**	
		Training	**Test**
2-Digit LDA	1 & 8	0.0348	0.0388
3-Digit LDA	1 & 8 & 5	0.0541	0.0553

Table 1: Error for training and test dataset for 2 and 3-digit LDA

Digits Chosen	**Error**	
	Training	**Test**
7 & 9	0.0428	0.0429
4 & 9	0.0433	0.0442
4 & 7	0.0151	0.0147

Table 2: Error for training and test datasets for digits that seemed the hardest to separate

Digits Chosen	**Error**	
	Training	**Test**
0 & 1	0.0041	0.0043
2 & 6	0.0079	0.0078
0 & 3	0.0083	0.0088

Table 3: Error for training and test datasets for digits that seemed the easiest to separate

Type of Classifier	Digits Chosen	Error	
		Training	Test
SVM	All 10 digits	0.0531	0.0597
SVM - Hardest	4 & 9	0.0363	0.0378
SVM - Easiest	0 & 1	9.4563e-4	9.7523e-4
Decision Tree	All 10 digits	0.2600	0.3210
Decision Tree - Hardest	4 & 9	0.0769	0.0773
Decision Tree - Easiest	0 & 1	0.0041	0.0048

Table 4: Error for SVM and Decision Tree for all digits and the easiest/hardest pairs of digits to separate with LDA

5 Summary and Conclusions

When discovering the test data errors for the two and three-digit LDA we found 0.0388 and 0.0553 respectively (Table 1). Understanding these errors shows that the LDA was over 90% accurate for both of these cases, which seems quite successful. Therefore, showing that the LDA can reasonably identify these values. It is also expected that the error for the three-digit LDA would be higher than the two-digit LDA as there are more objects to differentiate between and has a higher chance for crossover in features.

It was interesting to see that digits 4 & 9 were the hardest for the LDA to separate; these digits had a training and test data error of 0.0433 and 0.0442 respectively (Table 2). It is interesting to note that digits 4 & 9 had an error that was not much higher than 7 & 9 but, digits 4 & 7 had a much lower error. Although the features for 4 & 7 seemed close, it was easy for the LDA to distinguish between them, but not for the others. When looking at the image shape of digits 4 & 9, they both have a hole in the top. Although they are of different shapes, 4 has a triangle and 9 has a circle, it is understandable why these digits could get mixed up. It is also important to note, that the LDA was still over 95% accurate when separating these digits, therefore showing how powerful the LDA is.

After observing the data found in Table 3, we are able to conclude that the LDA found digits 0 & 1 the easiest to separate as it had a training data error of 0.0041 and a testing data error of 0.0043. This seems reasonable as they have quite distinct shapes, one being round and one a line. However, it is interesting to see that digits 1 & 8 also have very different shapes but had a much higher error (Table 1). Digits 2 & 6 and 0 & 3 both had very low error values, which is interesting has their shapes are both have many round portions, while the hardest to separate digits (7 & 9 and 4 & 9) had more angular edges. All of the easiest to separate digits had an accuracy of over 99% showing that the performance of the LDA is very strong.

When comparing the performance of the LDA to the SVM and Decision Tree classifier in Table 4, it was interesting to see some of the results we got. When distinguishing between all 10 digits for the SVM we received an error of 0.0531 for the training data and 0.0597 for our test data, which shows that the SVM was over 94% accurate. Similarly, these errors are very similar to the LDA distinguishing between 3-digits (Table 1), thus suggesting that the SVM may be more powerful than the LDA. When looking at the SVM's errors for the hardest and easiest to separate digits, we find a testing error of 0.0378 and 9.7523e-4. Both of these values are significantly lower than the errors the LDA had, further supporting the robust algorithm of the SVM.

The Decision Tree classifier had a much higher error than the SVM when trying to classify all 10 digits, with errors of 0.2600 and 0.3210 (Table 4). This value seems incredibly high; therefore, it may be interesting to look over the leaf nodes in the tree to understand perhaps why the classifications were so off. When looking at the errors for the easiest to separate pair found in the LDA, the training error for both the LDA and Decision Tree is 0.0041. However, the training and test dataset error for the Decision Tree for the hardest to separate digits is much higher than the LDA. This data supports the idea that the LDA performs more accurately than the Decision Tree classifier.

Understanding the reasoning behind the decisions the LDA makes is similar to looking at a black box, because it is not possible for us to know. Therefore, we do not understand why some digits get misclassified. Our understanding is based on the features in our projected data and how close or far apart the means of these objects are. Therefore, we again realize that there is much work and study that can be done in the machine learning space. Similarly, by testing other well-known classifiers, we can understand the strengths and limitations presented in our algorithms. Ultimately, allowing us to garner a better understanding and improve upon machine learning methods.

Appendix A. MATLAB functions used

- `B = sort(A)`**:** sorts the elements of `A` in ascending order.
- `B = reshape(A, sz)`**:** reshapes `A` using the size vector, `sz`, to define `size(B)`.
- `[U,S,V] = svd(A, 'econ')`**:** performs a singular value economy-size decomposition of matrix `A`, such that A = U*S*V' and removes extra rows or columns of zeros from the diagonal matrix of singular values.
- `E = eig(A)`**:** returns a column vector containing the eigenvalues of square matrix `A`.
- `[class, error] = classify(sample,training,group,'linear')`**:** classifies each row of the data in `sample` into one of the groups in `training`, `group` is a grouping variable for training. The output `class` indicates the group to which each row of `sample` has been assigned and is of the same type as `group`. Command also returns the error from the calculations.
- `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`.
- `L = kfoldloss(obj)`**:** returns loss obtained by cross-validated classification model `obj`.
- `Mdl = fitecoc(Tbl,Y)`**:** returns an ECOC model using the predictors in table `Tbl` and the class labels in vector `Y`. `fitecoc` uses $K(K-1)/2$ binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels).
- `ypred = predict(mdl,Xnew)`**:** returns the predicted response values of the linear regression model `mdl` to the points in `Xnew`.

Appendix B. MATLAB codes

```
%%
[images, labels] = mnist_parse('train-images-idx3-ubyte', 'train-labels-idx1-
ubyte');
[timages, tlabels] = mnist_parse('t10k-images-idx3-ubyte', 't10k-labels-idx1-
ubyte');
%%
images = im2double(images);
[labels,I] = sort(labels);

A = zeros(784,60000);
for i = 1:60000
    A(:,i) = reshape(images(:,:,i), 1, []);
end

A = A(:,I);
[U,S,V] = svd(A, 'econ');
sig = diag(S);
energies = zeros(1, length(sig));
for j = 1:length(sig)
    energies(j) = sig(j)^2/sum(sig.^2);
end
%%
timages = im2double(timages);
[tlabels, I] = sort(tlabels);

X = zeros(784,10000);
for i = 1:10000
    X(:,i) = reshape(timages(:,:,i), 1, []);
end

X = X(:, I);
%%
for i = 1:length(sig)
    cum(i) = sum(energies(1:i));
end
figure(1)
x= U(:,1:53)*S(1:53,1:53)*V(:,1:53)';
imshow(reshape(x(:,1), 28, 28))
figure(2)
x2= U(:,1:103)*S(1:103,1:103)*V(:,1:103)';
imshow(reshape(x2(:,1), 28, 28))
%%
plot3(U(:,2)'*A(:, 1:5923), U(:,3)'*A(:, 1:5923), U(:,4)'*A(:, 1:5923),'go');
hold on;
dR = [0.5, 0, 0];
dG = [0, 0.5, 0];
dB = [0, 0, 0.5];
plot3(U(:,2)'*A(:, 5924:12665), U(:,3)'*A(:, 5924:12665), U(:,4)'*A(:,
5924:12665),'mo');
plot3(U(:,2)'*A(:, 12666:18623), U(:,3)'*A(:, 12666:18623), U(:,4)'*A(:,
12666:18623),'co');
plot3(U(:,2)'*A(:, 18624:24754), U(:,3)'*A(:, 18624:24754), U(:,4)'*A(:,
18624:24754),'ro');
plot3(U(:,2)'*A(:, 24755:30596), U(:,3)'*A(:, 24755:30596), U(:,4)'*A(:,
24755:30596),'yo');
```

```
plot3(U(:,2)'*A(:, 30597:36017), U(:,3)'*A(:, 30597:36017), U(:,4)'*A(:,
30597:36017),'bo');
plot3(U(:,2)'*A(:, 36018:41935), U(:,3)'*A(:, 36018:41935), U(:,4)'*A(:,
36018:41935),'o', 'Color', dB);
plot3(U(:,2)'*A(:, 41936:48200), U(:,3)'*A(:, 41936:48200), U(:,4)'*A(:,
41936:48200),'ko');
plot3(U(:,2)'*A(:, 48201:54051), U(:,3)'*A(:, 48201:54051), U(:,4)'*A(:,
48201:54051),'o', 'Color', dR);
plot3(U(:,2)'*A(:, 54052:60000), U(:,3)'*A(:, 54052:60000), U(:,4)'*A(:,
54052:60000),'o', 'Color', dG);
xlabel('Projection onto 2nd Singular Vector');
ylabel('Projection onto 3rd Singular Vector');
zlabel('Projection onto 4th Singular Vector');
legend('digit 0', 'digit 1','digit 2', 'digit 3','digit 4','digit 5','digit
6','digit 7','digit 8','digit 9');
%% 2 Digit LDA
feature = 103;

o = A(:, 5924:12665);
e = A(:, 48201:54051);

n1 = size(o,2);
n8 = size(e, 2);

nums = U'*A;
os = nums(1:feature, 5924:12665);
es = nums(1:feature, 48201:54051);

m1 = mean(os,2);
m8 = mean(es, 2);

Sw = 0;
for k = 1:n1
    Sw = Sw + (os(:,k) - m1)*(os(:,k) - m1)';
end

for k = 1:n8
   Sw = Sw + (es(:,k) - m8)*(es(:,k) - m8)';
end

Sb = (m1-m8)*(m1-m8)';

[V2, D] = eig(Sb, Sw);
[lambda, ind] = max(abs(diag(D)));
w = V2(:, ind);
w = w/norm(w,2);

vone = w'*os;
veight = w'*es;

if mean(vone) > mean(veight)
    w = -w;
    vone = -vone;
    veight = -veight;
end
%%
```

```
plot(vone, zeros(n1), 'ob', 'Linewidth', 2)
hold on
plot(veight, ones(n8), 'dr', 'Linewidth', 2)
%%
sortones = sort(vone);
sorteights = sort(veight);

t1 = length(sortones);
t2 = 1;
while sortones(t1) > sorteights(t2)
    t1 = t1 - 1;
    t2 = t2 + 1;
end

threshold = (sortones(t1) + sorteights(t2))/2;
%%
subplot(1,2,1)
histogram(sortones,30);
hold on, plot([threshold threshold], [0 1500],'r')
set(gca,'Xlim',[-3 4],'Ylim',[0 1500],'Fontsize',14)
title('ones')
subplot(1,2,2)
histogram(sorteights,30); hold on, plot([threshold threshold], [0 600],'r')
set(gca,'Xlim',[-3 4],'Ylim',[0 600],'Fontsize',14)
title('eights')
%% 2 DIGIT TEST DATA
feature = 103;
training = zeros(12593,feature);
ma = U'*A;
training(1:6742, :) = ma(1:feature, 5924:12665)';
training(6743:12593, :) = ma(1:feature, 48201:54051)';

group = zeros(12593, 1);
group(1:6742, 1) = labels(5924:12665,1);
group(6743:12593, 1) = labels(48201:54051,1);

nums = U'*X;
sample = zeros(2109, feature);
sample(1:1135,:) = nums(1:feature, 981:2115)';
sample(1136:2109,:) = nums(1:feature, 8018:8991)';

[class2,err2] = classify(sample, training, group, 'linear');

checks = zeros(2109,1);
checks(1:1135, 1) = 1;
checks(1136:2109, 1) = 8;

error = 0;
for i = 1:2109
    if class2(i) ~= checks(i)
        error = error + 1;
    end
end
error = error/2109;
%% 3 DIGIT LDA
feature = 103;
training = zeros(18014,feature);
```

```
ma = U'*A;
training(1:6742, :) = ma(1:feature, 5924:12665)';
training(6743:12593, :) = ma(1:feature, 48201:54051)';
training(12594:18014, :) = ma(1:feature, 30597:36017)';

group = zeros(18014, 1);
group(1:6742, 1) = labels(5924:12665,1);
group(6743:12593, 1) = labels(48201:54051,1);
group(12594:18014, 1) = labels(30597:36017,1);

nums = U'*X;
sample = zeros(3001, feature);
sample(1:1135,:) = nums(1:feature, 981:2115)';
sample(1136:2109,:) = nums(1:feature, 8018:8991)';
sample(2110:3001,:) = nums(1:feature, 5140:6031)';

[class1,err1] = classify(training, training, group, 'linear');
[class2, err2] = classify(sample, training, group, 'linear');

checks = zeros(3001,1);
checks(1:1135, 1) = 1;
checks(1136:2109, 1) = 8;
checks(2110:3001, 1) = 5;
error = 0;
for i = 1:3001
    if class2(i) ~= checks(i)
        error = error + 1;
    end
end
error = error/3001;
%% 2 DIGIT LDA - Bullet 3 (7/9)
feature = 103;
training = zeros(12214,feature);
ma = U'*A;
training(1:6265,:) = ma(1:feature, 41936:48200)';
training(6266:12214,:) = ma(1:feature, 54052:60000)';

group = zeros(12214, 1);
group(1:6265, 1) = labels(41936:48200,1);
group(6266:12214, 1) = labels(54052:60000,1);

[class1, err1] = classify(training, training, group, 'linear');

sample = zeros(2037, feature);
nums = U'*X;
sample(1:1028,:) = nums(1:feature, 6990:8017)';
sample(1029:2037,:) = nums(1:feature, 8992:10000)';

[class2, err2] = classify(sample, training, group, 'linear');
%%
feature = 103;
training = zeros(12214,feature);
ma = U'*A;
training(1:6265,:) = ma(1:feature, 24755:30596)';
training(6266:12214,:) = ma(1:feature, 54052:60000)';
```

```
group = zeros(12214, 1);
group(1:6265, 1) = labels(24755:30596,1);
group(6266:12214, 1) = labels(54052:60000,1);

[class1, err1] = classify(training, training, group, 'linear');

sample = zeros(2037, feature);
nums = U'*X;
sample(1:1028,:) = nums(1:feature, 4158:5139)';
sample(1029:2037,:) = nums(1:feature, 8992:10000)';

[class2, err2] = classify(sample, training, group, 'linear');
%% 2 DIGIT LDA - Bullet 4 (0/1)
feature = 103;

training = zeros(12665,feature);
ma = U'*A;
training(1:5923,:) = ma(1:feature, 1:5923)';
training(5924:12665,:) = ma(1:feature, 5924:12665)';

group = zeros(12665, 1);
group(1:5923, 1) = labels(1:5923,1);
group(5924:12665, 1) = labels(5924:12665,1);

[class1, err1] = classify(training, training, group, 'linear');

sample = zeros(2115, feature);
nums = U'*X;
sample(1:980,:) = nums(1:feature, 1:980)';
sample(981:2115,:) = nums(1:feature, 981:2115)';

[class2, err2] = classify(sample, training, group, 'linear');
%% BULLET 5 UGHHHHHHHHHH
feature = 103;
ma = U'*A;
training = ma(1:feature, :)';
nums = U'*X;
test = nums(1:feature, :)';

tree = fitctree(training, labels,'MaxNumSplits', 10, 'CrossVal', 'on');
view(tree.Trained{1}, 'Mode', 'graph');
classError = kfoldLoss(tree);
%%
mdl = fitcecoc(training, labels);
test_labels = predict(mdl,test);
error = 0;
for i = 1:10000
    if test_labels(i) ~= tlabels(i)
        error = error + 1;
    end
end
error = error/10000;
%% BULLET 6 !!!!!1
% hardest two digits
feature = 103;
training = zeros(12214,feature);
```

```
ma = U'*A;
training(1:6265,:) = ma(1:feature, 41936:48200)';
training(6266:12214,:) = ma(1:feature, 54052:60000)';

group = zeros(12214, 1);
group(1:6265, 1) = labels(41936:48200,1);
group(6266:12214, 1) = labels(54052:60000,1);

tree = fitctree(training, group, 'MaxNumSplits', 10, 'CrossVal', 'on');
view(tree.Trained{1}, 'Mode', 'graph');
classError = kfoldLoss(tree);

sample = zeros(2037, feature);
nums = U'*X;
sample(1:1028,:) = nums(1:feature, 6990:8017)';
sample(1029:2037,:) = nums(1:feature, 8992:10000)';

checks = zeros(2037,1);
checks(1:1028, 1) = 7;
checks(1029:2037, 1) = 9;

mdl = fitcecoc(training, group);
test_labels = predict(mdl,sample);
error = 0;
for i = 1:2037
    if test_labels(i) ~= checks(i)
        error = error + 1;
    end
end
error = error/2037;
%% Easiest two (Bullet 6)
feature = 103;

training = zeros(12665,feature);
ma = U'*A;
training(1:5923,:) = ma(1:feature, 1:5923)';
training(5924:12665,:) = ma(1:feature, 5924:12665)';

group = zeros(12665, 1);
group(1:5923, 1) = labels(1:5923,1);
group(5924:12665, 1) = labels(5924:12665,1);

tree = fitctree(training, group, 'MaxNumSplits', 10, 'CrossVal', 'on');
view(tree.Trained{1}, 'Mode', 'graph');
classError = kfoldLoss(tree);

sample = zeros(2115, feature);
nums = U'*X;
sample(1:980,:) = nums(1:feature, 1:980)';
sample(981:2115,:) = nums(1:feature, 981:2115)';

checks = zeros(2115,1);
checks(1:980, 1) = 0;
checks(981:2115, 1) = 1;

mdl = fitcecoc(training, group);
```

```
test_labels = predict(mdl,sample);
error = 0;
for i = 1:2115
    if test_labels(i) ~= checks(i)
        error = error + 1;
    end
end
error = error/2115;
```