Math 444: Final

Levent Batakci

May 17, 2021

In this project, we leveraged gray-level co-occurrence matrices (GLCM), principal component analysis (PCA), linear discriminant analysis (LDA), and tree classifiers to analyze and classify black and white texture images. All of the mentioned implementations can be found on the public github repo.

Texture Images Background

The data set we're working with consists of 350 distinct 128×128 pixel black and white images. Each image is represented by a $128^2 = 16834$ element vector in which each entry represents the grayscale value of a pixel. As such, the data is stored in a matrix $X \in \mathbb{R}^{16834\times 350}$. This data set is available for download on the public github repository (see intro).

Moreover, each image in the set corresponds to one of 5 texture types. Bloew, **Figure 1** displays examples of each of the texture types.

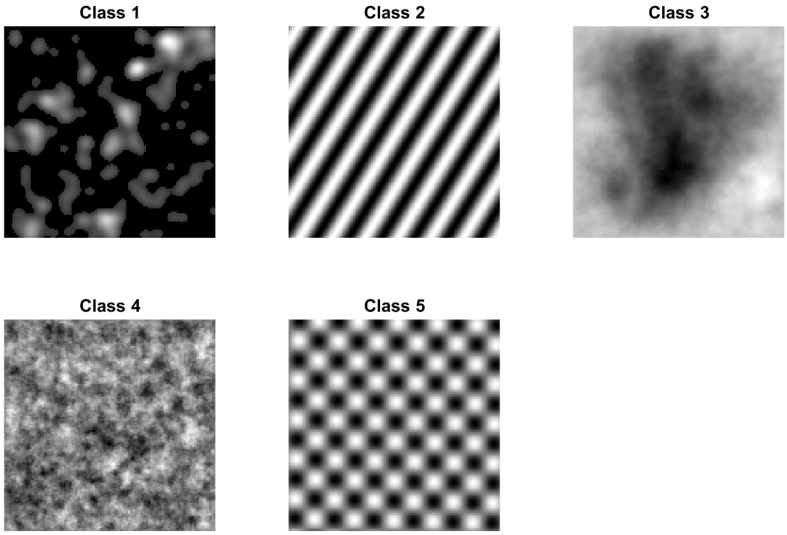

Figure 1: Examples of the 5 Texture Classes

An important aspect of the data to note is that rotation is not uniform. That is, images in class 2 contain many different rotations of stripes - some vertical, some horizontal, and some in-between. This knowledge will help us later when it comes to computing gray level co-occurrence matrices.

Gray Level Discretization

The raw images are encoded with a high level of detail – that is, there is a gradient of gray values that pixels can take on. In practice, much of this detail is not crucial when it comes to texture classification. So, to simplify the data, we coarsen the gray levels.

Specifically, we simplify the encoding to only account for k discrete gray levels. Generally speaking, k will be a relatively small number – we've chosen to work with $k = 6$. To coarsen the gray levels, we partition the interval of valid gray levels into k equal subintervals. Naturally, each gray level from before corresponds to one of these subintervals, and each subinterval corresponds to one of the discrete gray levels. We take the center of the subinterval to be the gray level it corresponds to.

We find it instructive to visualize the coarsification in order to really understand it. Below, **Figure 2** shows coarsified versions of the 5 images from **Figure 1**. The effects of the coarsification here are quite apparent – especially in the examples of classes 1 and 3.

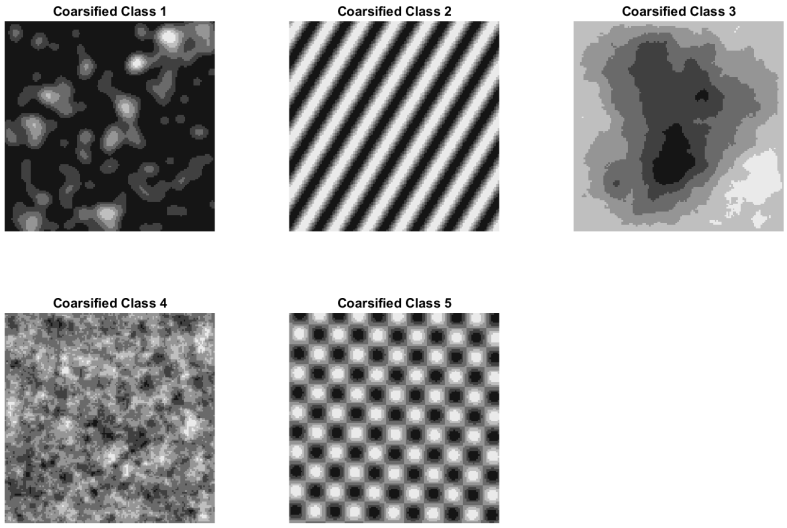

Figure 2: Coarsified ($k = 6$) Examples of the 5 Texture Classes

Gray Level Co-occurrence Matrices

Much of our analysis will be driven by gray level co-occurrence matrices (GLCM). A crucial factor in a GLCM is the offset parameter. The offset is a pair (u, v). Given that we are working with k discrete colors, the gray level co-occurrence matrix $G = [g_{p,q}] \in \mathbb{R}^{k\times k}$ of an image x is defined by

$$g_{p,q} = \text{The } \text{ of times that } x_{i,j} = p \text{ and } x_{i+u,j+p} = q$$

where $x_{i,j}$ represents the color of the pixel of x located at (i, j). Moreover, the matrix is usually normalized so that the entries sum to 1. This promotes an interpretation of entries as relative frequencies. An immediate concern that arises is boundary conditions. Indeed, how do we count when $x_{i+u,j+p}$ is not in bounds of the image?

There are two main ways to address this issue. One route is to ignore these points – that is, to exclude them from the counting altogether. Another possible solution is to use periodic boundary conditions. That is, to virtually tile the image so that out-of-bounds pixels loop back around to known parts.

Since repetition is inherent to texture, this solution is viable. We implemented both of these boundary conditions and found exclusion boundary conditions to be marginally better.

The big idea here is that these matrices are capable of capturing patterns in how color (gray levels) vary. In some ways, this is really what defines a texture. However, in order to capture the essence of a texture, we cannot simply rely on one offset.

Indeed, it is unlikely that we can find a single offset capable of distinguishing many different textures. The solution to this problem is not so tough. We simply choose a variety of offsets and compute multiple GLCMs for each image. Specifically, with the non-uniformity of texture rotation in mind, we chose to use the offsets $(2, 0), (0, 2)$, and $(2, 2)$. We believe these to encapsulate horizontal, vertical, and diagonal patterns.

So, putting it all together, we computed 3 6-by-6 GLCMs for each image. Already, this is a drastic reduction in dimensionality. Indeed, the images went down from being described by 16834 values to being described by just $3 \cdot 6 \times 6 = 108$. To be clear, we turned each GLCM into a vector by stacking the columns, then we stacked those on top of one another. So the images are now described by a 108 element vector.

Principal Component Analysis and Linear Discriminant Analysis

Next, we decided to perform a principal component analysis on the data to see if any kind of clustering was retained through the drastic dimension reduction. We decided to just look at the first 4 principal components. Below, **Figure 3** shows scatterplots of pairings of the first 4 principal components.

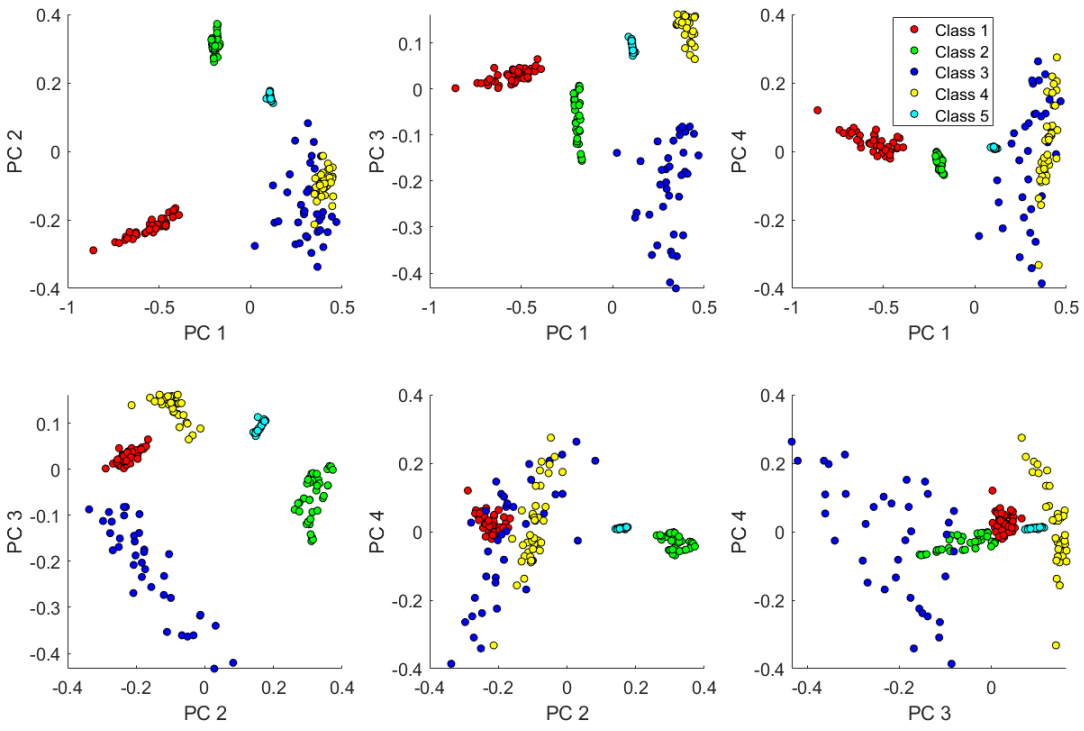

Figure 3: Principal Component (PC) Scatterplots

Immediately, we notice that the images are already separating by class! To see how far we could push the separation, we performed a linear discriminant analysis on the first 4 principal components. The projections onto the first two separators can be seen below in **Figure 4**.

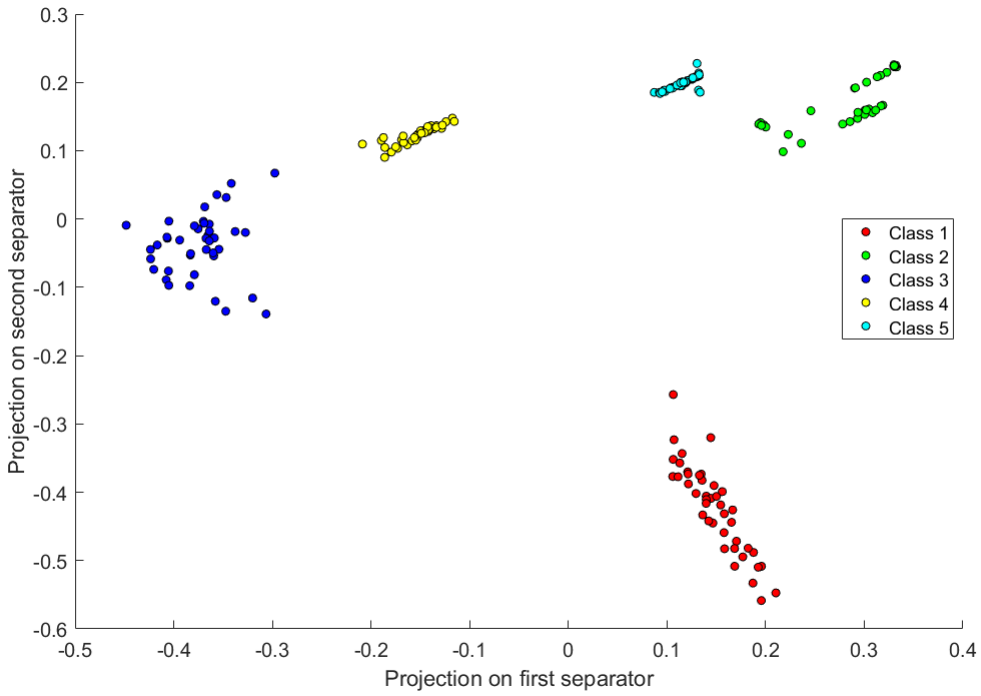

Figure 4: LDA on First 4 PCs of GLCMs

According to us, the separation looks great. With this in mind, we were pretty sure that we could classify the images with only the projections of their GLCMs on the first two separators. That is, we're down to describing the images with only two dimensions!

Building and Testing a Tree Classifier

Next, we decided to build a classifier upon this seemingly separable data. We chose to use a tree classifier and to build it on the projections on the LDA separators. One important thing to note is that we can't use the entire data-set to build a tree classifier. Indeed, that would leave us with no test set!

To account for this, we randomly sampled 200 images from the data set and built the tree classifier off of them. In over 50 runs of the code, the number of leaves in the maximal (unpruned) tree never exceeded 6. Many times, the maximal tree had only 5 leaves. Considering that there are only 5 classes of texture, this is quite incredible.

With this in mind, we decided that it makes no sense to prune such a small tree. Indeed, pruning a tree with 5 leaves is guaranteed to result in disaster - not every class will even be accounted for! Moreover, 6 leaves truly is a minuscule amount in any context and is in no ways space of search inefficient. For these reasons, we did not even attempt to prune the maximal tree.

Then, we evaluated the classifier on the unused 150 images. On just these images, we got a minmum accuracy of 99.33% and a common maximum of perfect (100%) accuracy. Naturally, we were delighted with these results.

To see if we could get even better results, we also built trees on the principal components. These trees performed very similarly, having the same maximum and minimum accuracy. For this reason, we opted to stick with the trees built on the LDA – these describe images with just two dimensions as opposed to four.

Conclusion

Our analysis and classification of this data set resulted in a glorious success. Indeed, we were able to reduce the dimensionality of the data from 16834 to 2 AND build a perfectly accurate classifier from just 57% of the data. We are led to conclude that black and white texture images - at least ones in these 5 classes - are easily reducible and classifiable via gray level co-occurrence matrices.

The implications of this kind of technology are quite important. One can imagine the bearings such capabilities have on computer vision and autonomous systems. Indeed, classifying textures can enable computers to identify objects without human input. Moreover, since the GLCMs record relative frequencies, classification can be quite robust - comparing different-sized images. We could see this application of data science being invaluable to exploratory robots such as the Mars Rover.

Appendix

As always, feel free to access the code at the public github repo. The Node data class and associated algorithms are not included because there are many helper functions and it would not be very readable on an LaTeX document.

1 Main Code

```
%Levent Batakci
%MATH444 - Final
%Classifies black and white textures.

clear all

%Load the images
%The data consists of 409 128x128 grayscale images.
%The columns represent the images, so the data is stored in a 16384x409
%matrix
%The entries are between 0 and 255, inclusive
load TestImages.mat
X = X* 255;

%Select num random images
p = size(X,2);
num = 200;
sample = sort(randsample(1:p, num));
Xsample = X(:, sample);
Isample = I(sample);

%Parameters
k = 6; % # of gray levels
n=128; % image dimension
offsets(:,1) = [2 0]';
```

```
offsets(:,2) = [0 2]';
offsets(:,3) = [2 2]';
r=4; %PCs

%Process the images
[G, discretizedImages] = ProcessImages(Xsample, k, n, offsets);

%TEST WHAT THE DISCRETIZATION LOOKS LIKE
Xdisc = DiscreteGrayScale(discretizedImages,k);
clf(figure(1))
figure(1)
show(Xdisc(:,2), n, n);
clf(figure(2))
figure(2)
show(Xsample(:,2), n, n);
%Works well.

%Compute the principal components!
[Z, features] = PCA_r(G, r);

colors = [1 0 0; 0 1 0; 0 0 1; 1 1 0; 0 1 1; 1 0 1; 1 0.5 0.2; 0 0 0];
clf(figure(3))
figure(3)
hold on
ct = 1;
for i = 1:r
    for j = i+1:r
        subplot(2,3, ct);
        ct = ct + 1;
        hold on
        for c = unique(I)
            Z_ = Z(:, Isample==c);
            scatter(Z_(i,:), Z_(j,:), "MarkerEdgeColor", "k", "MarkerFaceColor", colors(c,:));
        end
        xlabel("PC " + i);
        ylabel("PC " + j);
        set(gca,"FontSize", 15);
    end
end
legend("Class 1", "Class 2", "Class 3", "Class 4", "Class 5");

%LDA
[Q, Zlda] = LDA(Z, Isample);
clf(figure(4))
figure(4)
for i = unique(I)
    Z_ = Zlda(:, Isample==i);
    s = scatter(Z_(1,:), Z_(2,:),"MarkerEdgeColor", "k", "MarkerFaceColor", colors(i,:));
    hold on
end

legend("Class 1", "Class 2", "Class 3", "Class 4", "Class 5");
xlabel("Projection on first separator");
```

```
set(gca,'FontSize',15);
ylabel("Projection on second separator");

disp("MAKING MAXIMAL TREE"); %ON PCA!
[root, Nodes, Leaves] = MaximalTree(Z, Isample);
[LDAroot,~,LDAleaves] = MaximalTree(Zlda, Isample);
disp("MAXIMAL TREE COMPLETED");

%Test maximal tree on the remaining data points
%We have to process the data again..
G = ProcessImages(X, k, n, offsets);
Gc = G - sum(G,2) / size(G,2);
Zfull = features' * Gc;
Zlda_full = Q' * Zfull;

%Test on untested set
percentCorrect = 0;
percentCorrectLDA = 0;
for i = 1:p
    if(nnz(sample-i) == num) %not in sample
        atr = Zfull(:,i);
        LDAatr = Zlda_full(:,i);
        c = I(i);
        percentCorrect = percentCorrect + ((classify(root, atr) == c)/(p-num));
        percentCorrectLDA = percentCorrectLDA + ((classify(LDAroot, LDAatr) == c)/(p-num));
    end
end
percentCorrect
percentCorrectLDA
```

2 GLCM

```
function GLCMs = GLCM(images, r, c, offsets, k)
GLCMs = cell(3);

for j = 1:size(offsets,2)
    GLCMs{j} = [];
end

for i = 1:size(images,2)
        im = reshape(images(:,i),r,c);
    for j = 1:size(offsets,2)
        GLCMs{j} =  [GLCMs{j} GHelper(im, offsets(:,j),k)];
    end
end

end

function G = GHelper(im, offset, k)
%Ignores points past the boundary, which is logical for textures
%Textures repeat!
```

```
G = zeros(k,k);

shifted = CircularShift(im, offset);
for i = 1:k
    for j = 1:k
        G(i,j) = nnz(im == i & shifted == j);
    end
end

%Stack the results.
G = reshape(G, k*k, 1);

%Scale so the entries sum to 1;
G = G / sum(G);
end

function shifted = CircularShift(X, offset)

x = offset(1);
y = offset(2);

xStart = 1+x;
if(xStart > size(X,2))
    xStart = xStart - size(X,2);
end

yStart = 1+y;
if(xStart > size(X,1))
    yStart = yStart - size(X,1);
end

shifted = X;

if(x ~= 0)
    shifted = [X(xStart:end, :);  -1 * ones(size(X(1:xStart-1, :)))];
    %shifted = [X(xStart:end, :);  X(1:xStart-1, :)]; %PERIODIC
end

if(y ~= 0)
    shifted = [shifted(:, yStart:end)  -1 * ones(size(shifted(:,1:yStart-1)))];
    %shifted = [shifted(:, yStart:end)  shifted(:,1:yStart-1)]; %PERIODIC
end


end
```

3 Maximal Tree

```
function [root, Nodes, Leaves] = MaximalTree(attr, I)
root = Node(attr, I);
Nodes = cell(1);
Nodes{1} = root;
```

```
%GROW THE TREE !
Lpure = [];
Lmixed = [1];
count=1;

Leaves = cell(1);
i = 1;
while numel(Lmixed) ~= 0
    index = Lmixed(1);
    chosen = Nodes{index}; %Choose mixed node

    %Adjust Lmixed
    if(numel(Lmixed) == 1)
        Lmixed = [];
    else
        Lmixed = Lmixed(2:end);
    end

    %Optimally split the node
    chosen = SplitNode(chosen);

    %Check the children
    L = chosen.left;
    count=count+1;
    if(getErr(L) ~= 0)
        Lmixed(end+1) = count;
    else
        Lpure(end+1) = count;
        Leaves{i} = L;
        i=i+1;
    end
    Nodes{count} = L;

    R = chosen.right;
    count=count+1;
    if(getErr(R) ~= 0)
        Lmixed(end+1) = count;
    else
        Lpure(end+1) = count;
        Leaves{i} = R;
        i=i+1;
    end
    Nodes{count} = R;

    if(mod(count+1,100) == 0)
        disp(count)
    end
end

root = Nodes{1};
end
```