Classification of Land Types Through Clustering

Kyle Colton
Thomas Kwak

June 4, 2015

1 Introduction

Hyperspectral imaging collects and processes information by taking images at varying frequencies within the EM spectrum. Because the imaging can divide the spectrum into more band than the human eye, hyperspectral imaging is used to find objects, identify materials, or detect certain processes. Therefore, there are multiple applications within the fields of astronomy, agriculture, biomedical imaging, physics, and geosciences.

One example of hyperspectral imaging is the Indian Pines test site in Northwestern Indiana. This dataset contains 145 by 145 pixel images taken at 224 different spectral reflectance bands in the wavelength ranging from 0.4 to 2.5 micrometers. Due to the region of water absorption, the number of bands had been reduced from 224 to 200.

The goals of this experiment are to use k-means clustering, soft-margin support vector machines, and linear models to separate and classify each pixel on the entire dataset set into one of the sixteen different crops found at the Indian Pines test site with accuracy as close as possible to the ground truth.

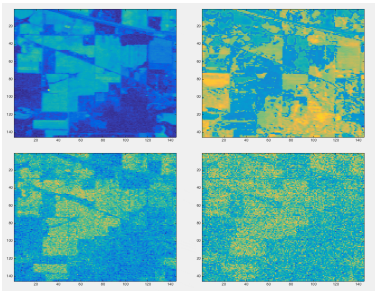
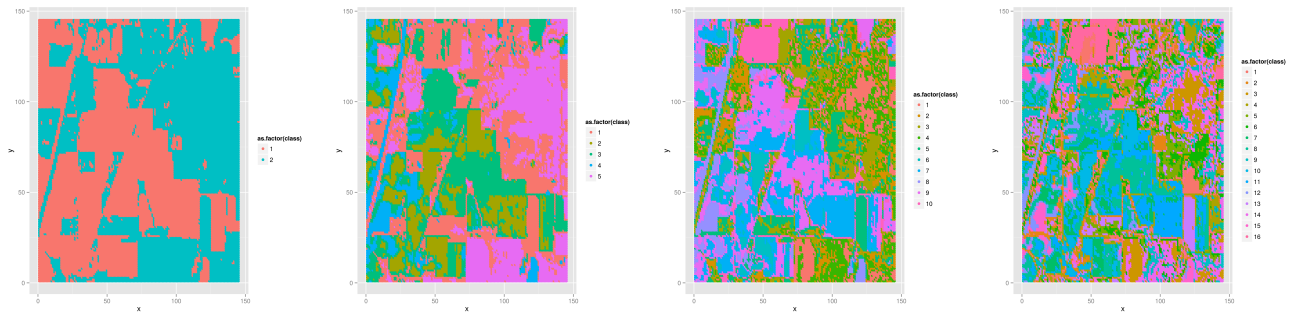


Figure 1: Sample captures taken from the hyperspectral image

2 Methods

2.1 K-Means Clustering

K-means clustering is used to group samples into k clusters such that the distances between the samples within each group is small compared to the distance between groups. This was used to group each of the pixels into either two, five, ten, or sixteen different groups.



(a) $k = 2$ (b) $k = 5$ (c) $k = 10$ (d) $k = 16$

Figure 2: K Means at various values of k clusters

2.2 Soft-Margin Support Vector Machines

Soft-margin support vector machines are supervised learning models used for classification through pattern recognition. Soft-margin support vector machines were used instead of hard-margin support vector machines because the dataset was found to be not linearly separable. A quarter of the ground truth dataset was used as training data for the soft-margin multiclass support vector machine.

2.3 Principle Component Analysis

To reduce the size of the dataset, principal component analysis was used on the data set after concatenating all bands into a single matrix. This method can then reduce the dataset into the necessary principal components, thus reducing the size of the overall dataset. By comparing the unweighted and weighted principal components, there is a reduction in preprocessing data from 21025 dimensions to 200 dimensions without much loss in accuracy.

3 Results

3.1 K-Means Clustering

As the number of classes increased, more landtypes were able to be distinguished. However, the level of noise also increased as the number of classes increased. While K-Means was able to get general shapes, the individual crop-detection was noisy.

3.2 Soft-Margin Support Vector machines

Initially, simple SVM was run to test the separability of the data into two classes. The data was found to be not linearly separable.

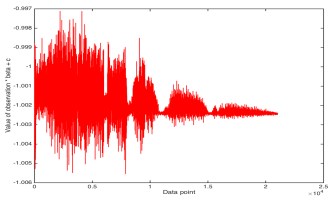
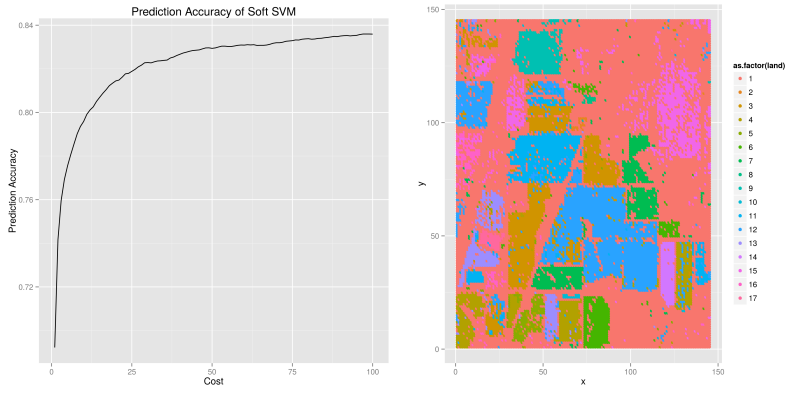


Figure 3: Testing linear separability using SVM

Multiple SVM was used from the R library `e1071`. While simple SVM allows for 2 class selection, the multiclass SVM allows a greater number of classes by running SVM multiple times. A cost analysis was run to see the performance of the model as a function of cost. The maximum accuracy found by the soft-margin SVM was approximately 83.5% at a constraint violation cost of 100. Generally, model accuracy increased as constraint violation cost increased.

Multiclass SVM was done using the gaussian radial basis kernel.



(a) SVM Accuracy vs Cost (b) SVM Classification Map

Figure 4: Multiple SVM

3.3 Principle Component Analysis

Using PCA, the dataset was reduced from a 200 x 21025 matrix to a 200 x 200 matrix. From the unweighted PCA, approximately 93.5% of the variance could be explained by just the first three principal components. From the weighted PCA, approximately 90.4% of the variance could be explained by the first three principal components.

4 Discussion

Overall, the ability to separate and classify each data point using every band was a success. The k-means clustering was able to roughly identify the different landtypes and shapes seen in the ground image. Noise was introduced because of the lack of spacial awareness in the K-means function. One possible noise reduction method would be to weight the spectra beforehand to rely more on the crisply defined images. As is, some data points were classified incorrectly simply because of their proximity to other high-noise points.

Given a training subset of 25%, the soft-margin multiple SVM could predict approximately 83.5% of the data accurately compared to the ground truth. This could be explained by the kernel which was used as well as the fact that the data was nonlinear.

For the principal component analysis, the data set was reduced and then examined either with weighting or without weighting. More than 90% of the variance in the data could be explained by the first three principal components. By reducing the data set down from 21025 variables to 200, there is only a loss of about 10% of information when only considering the first 3 of 200 variables. This disproportionate distribution of noise in the first few components allows for less noise and better dimensionality reduction.

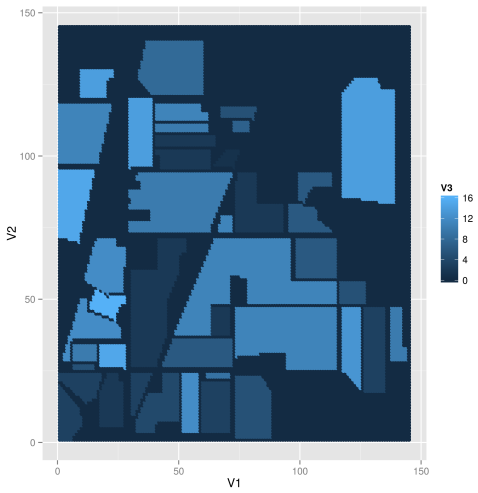


Figure 5: Ground truth data

5 Future Work

Using both linear and nonlinear Gaussian models after preprocessing to separate the data would be less computationally expensive in comparison to k-means. More analysis on multiclass-SVM could lead to a higher accuracy in the separation of landtypes. Further dimensionality reduction would also be another solution to faster computations, but this could lead to higher inaccuracies or more loss in data.

Post processing using mathematical morphology may also reduce noise in the predicted values. Since farmland is generally solid, inorganic shapes, closing (or opening) the predicted groups may reduce the noise.

Noise reduction could also be approached by attempting to identify noisy spectra in the data. Some spectra appeared to contribute little to the model while including large amounts of noise. By reducing the influence of these less important spectra, we may improve the model.

6 Appendix

6.1 GitHub

All code is available at `https://github.com/kycolton/drunken-secret-ninja`.

6.2 Data Investigation

6.2.1 GIF of all Spectra

```
% Iterates through the Indian Pines file and produces a gif for each band

fname = 'indian_pines.gif';
fps = 24;
fh = figure(1);

for i = 1:size(indian_pines_corrected, 3)
    imagesc(indian_pines_corrected(:,:,i));
    drawnow;
    f = getframe(fh);
    im = frame2im(f);
    [imind,cm] = rgb2ind(im,256);
    if i == 1;
        imwrite(imind,cm,fname,'gif','Loopcount',inf);
    else
        imwrite(imind,cm,fname,'gif','WriteMode','append','DelayTime',1/fps)
    end
end
```

6.3 K-Means

6.3.1 Matlab

```
%% K-Means
%%  Separate data points into K clusters with no other information.
%% Inputs:
%%  A - D-by-N matrix of N points in D dimensions.
%%  K - Integer number of clusters to detect.
%% Outputs:
%%  C - D-by-K matrix with the learned cluster centroids.
%%  labels - Length N vector with integer (1, 2, ..., K) class assignments.

function [C, labels] = km(A, K)
    % D = 2
    % N = 1000
    [D, N] = size(A);

    C = zeros(D, K);
    labels = zeros(N, 1);
    r = zeros(K);

% initially picks random data points as centroids
```

```
    % creates K random numbers
    rng('shuffle');
    for i = 1:K
        r(i) = 1 + floor(rand()*N);
    end

    % assigns centroids with labels 1, 2, ... , K
    for j = 1:K
        for i = 1:D
            C(i, j) = A(i, r(j));
            labels(r(j)) = j;
        end
    end

    disp('Starting k-means calculations.');

% iterates until classes no longer change
changes = N;
while(changes ~= 0)

% assign each observation to nearest centroid

    % calculates distance from each point to each centroid
    dist = zeros(K, N);
    for j = 1:K
        for i = 1:N
            dist(j, i) = norm(C(:, j) - A(:, i));
        end
    end

    % finds smallest index in the dist matrix (interested in minIndex
    [minValue, minIndex] = min(dist);

    % check to see how many indexes changed
    changes = 0;
    for i = 1:N
        if(minIndex(i) ~= labels(i))
            changes = changes + 1;
        end
    end
    disp(horzcat('Changes made: ', num2str(changes)));

    % redefines labels output
    labels = minIndex;

% relearn centroid based on average of assigned data points
    classSum = zeros(D, K);
    count = zeros(1, K);

    % find sum and total number in each class
    for j = 1:K
        for i = 1:N
            if(labels(i) == j)
                for m = 1:D
                    classSum(m, j) = classSum(m, j) + A(m, i);
                end
```

```
                count(j) = count(j) + 1;
            end
        end
    end

    % reassign centroids
    for j = 1:K
        for i = 1:D
            C(i, j) = classSum(i, j) / count(j);
        end
    end
end

disp('Finsihed k-means calculations. Process completed.');

end
```

6.3.2 R

```
library(R.matlab)
library(reshape)
library(animation)
library(ggplot2)

ip <- readMat('Data/Indian_pines_corrected.mat')
ip <- array(unlist(ip),dim=c(145,145,200))

n <- dim(ip)[1]; m <- dim(ip)[2]; z <- dim(ip)[3]

ip.long <- array(0,dim=c(n * m, 3, z))
for( i in 1:(z) )
{ ip.long[,,i] <- as.matrix(as.data.frame(melt(ip[,,i]))) }

for( i in 1:z )
{
    print(ggplot(as.data.frame(ip.long[,,i]),aes(V1,V2,color=V3))+geom_point())
    readline("Press <return> to continue")
}

ip.df <- array(dim=c(m*n,z))
for( i in 1:n)
{
    for( j in 1:m )
    { ip.df[(i-1)*n+j,] <- ip[i,j,] }
}

ip.km <- kmeans(ip.df,5)

classed <- matrix(ip.km$cluster,nrow=n,ncol=m,byrow=T)

ip.km.df <- as.data.frame(cbind(rep(1:n,each=m),rep(1:m,n),ip.km$cluster))
names(ip.km.df) <- c('x','y','class')

ggplot(ip.km.df,aes(x,y,color=as.factor(class))) + geom_point()
```

```
saveGIF({for(i in 1:z)
{ print(multiplot(
    ggplot(as.data.frame(melt(ip[,,i])),aes(value)) + geom_histogram(binwidth=1,aes(y=..density..)) + xlim
    ggplot(as.data.frame(ip.long[,,i]),aes(V1,V2,color=V3))+geom_point(), cols=2))
}}, interval=0.2, movie.name = 'together.gif')


#~ km3d <- function(x,k,v=F)
#~ {
#~     n <- dim(x)[1]; m <- dim(x)[2]; z <- dim(x)[3]
#~     entries <- sample(1:(n*m),k)
#~     means <- array(dim=c(z,k)); for(l in 1:length(entries)){ means[,l] <- x[ceiling(entries[l]/n),entri
#~     dists = array(dim=c(n,m,k))
#~     converge <- FALSE
#~     old.labs <- labs <- array(dim=c(n,m))
#~     while( !converge )
#~     {
#~         for( j in 1:k )
#~         { dist[,,j] <- apply(x,1:2,function(y) {dist(cbind(y,means[k]))}) }
#~         labs <- apply(dist,1:2,which.min)
#~         if(old.labs == labs){ converge <- TRUE }
#~         if(v){ print("Round") }
#~         old.labs <- labs
#~     }
#~     return(labs)
#~ }
```

6.4 SVM

```
library(R.matlab)
library(reshape)
library(animation)
library(ggplot2)
library(e1071)

ip <- readMat('Data/Indian_pines_corrected.mat')
ip <- array(unlist(ip),dim=c(145,145,200))
gt <- readMat('Data/Indian_pines_gt.mat')
gt <- array(unlist(gt),dim=c(145,145,200))

n <- dim(ip)[1]; m <- dim(ip)[2]; z <- dim(ip)[3]

ip.long <- array(0,dim=c(n * m, 3, z))
for( i in 1:(z) )
{ ip.long[,,i] <- as.matrix(as.data.frame(melt(ip[,,i]))) }

ip.df <- array(dim=c(m*n,z))
for( i in 1:n)
{
    for( j in 1:m )
    { ip.df[(i-1)*n+j,] <- ip[i,j,] }
}

# All GT Layers are identical
gt.long <- as.vector(t(gt[,,1]))
```

```
set.seed(48625)
train <- sample(1:length(gt.long),length(gt.long)/4)
test <- (1:length(gt.long))[-train]

ip.svm <- svm(x=ip.df[train,], y=as.factor(gt.long[train]), cost = 100)
ip.pred <- predict(ip.svm,newdata=ip.df)
svm.confusion <- table(pred = ip.pred, true = as.factor(gt.long))

ip.svm.df <- as.data.frame(cbind(rep(1:n,each=m),rep(1:m,n),ip.pred))

names(ip.svm.df) <- c('x','y','land')

ggplot(ip.svm.df,aes(x,y,color=as.factor(land))) + geom_point()
```

6.5 PCA

```
load '../Data/Indian_pines_corrected.mat'

[n, m, k] = size(indian_pines_corrected);
obs = reshape(indian_pines_corrected, [n*m, k]);

disp('Starting regular PCA');
[coeff,score,latent,tsquared,explained,mu] = pca(obs);
disp('Finished regular PCA');

disp('Starting weighted PCA');
[w_coeff,~,w_latent,~,w_explained] = pca(obs,'VariableWeights','variance');
disp('Finished weighted PCA');

disp('Processes completed');
```