Water Detection and Segmentation using Pattern Recognition Algorithms

ECE 759: Pattern Recognition (Spring 2016)

Department of Electrical and Computer Engineering

Siddharth Roheda
North Carolina State University
sroheda@ncsu.edu

Prathamesh Prabhudesai
North Carolina State University
ppprabhu@ncsu.edu

Amit Rao
North Carolina State University
arao4@ncsu.edu

Contents

1 Introduction

This project involves implementation of pattern recognition techniques in order to detect and segment water regions in images. This project can be considered to be a preliminary step towards a bigger picture of being able to detect flooded areas in an image feed from social networking websites and being able to recognize such geographical areas and putting out alerts.

The first phase of the project is to create a dataset of water and non-water areas, cropped from larger images. We also choose the features for the classification of data, and discuss the reasons for the selection in this phase.

The second phase involves learning the characteristics of 'water' and 'non-water' areas in an image using certain features. We try to implement the Bayesian Classifier to learn the characteristics of the 'water' and 'non-water' areas and test the classifier using some test images. We also evaluate the performance of a linear classifier (Perceptron) and compare it with that of the Bayesian Classifier.

The idea behind a Bayesian classifier is that, if an agent knows the class, it can predict the values of the other features. If it does not know the class, Bayes' rule can be used to predict the class given (some of) the feature values. In a Bayesian classifier, the learning agent builds a probabilistic model of the features and uses that model to predict the classification of a new example. The Bayesian Classifier will provide the minimum error probability and gives the best performance. But, it also requires the knowledge of priors, which might not be available in many situations. In such a case, we go for the linear classifier.

In the third phase of this project we try to implement the Bayesian Classifier in the segmentation of the water from the image. We also use the Perceptron to do the same, and compare the results.

The method used in [1] for segmentation of SAR images involving areas of forest and grass has been implemented here for areas considered to be 'water' or 'non-water'. This is a multi-scale segmentation method. A neighborhood of size 32x32 is chosen at a time, and decision is made based on the Bayesian distribution. If a boundary is present in the chosen neighborhood, it is then divided into quadrants and then a decision is made for these quadrants separately. The quadrants are further divided if a decision cannot be made. This is done until the size of the quadrant reaches 2x2.

The rest of the report is divided into three sections, namely, the Approach, Results and Conclusion Sections. The Approach section describes the problem in detail and explains how the authors have implemented the solution. The next section presents and discusses the results obtained from the simulations which were run using MATLAB. The final section, Conclusion, talks about what was learnt from the project and concludes it.

2 Approach

2.1 Pre-processing and Selection of Features

Pre-processing involves the creation of data sets and labeling of images. The first step was to crop out 'water' and 'non-water' areas from the images. These cropped images can then be used to learn the statistics of the water regions and the non-water regions. Out of the fifty images that were provided, thirty images were randomly selected for training purposes and the rest twenty images were used for testing the classifier.

Features that were selected for the classification of the regions as 'water' and 'non-water' are hue and saturation. This means, the image was first converted from the RGB model to the HSV (Hue-Saturation-Value) model. The hue (H) of a color refers to which pure color it resembles. Hues are described by a number that specifies the position of the corresponding pure color on the color wheel, as a fraction between 0 and 1. Value 0 refers to red; 1/6 is yellow; 1/3 is green; and so forth around the color wheel. All tints, tones and shades of red have the same hue. So, The water, which will usually be muddy, will have the same hue whether there are shadows, or different illumination due to images being taken at different times of the day. The saturation (S) of a color describes how white the color is. A pure red is fully saturated, with a saturation of 1; tints of red have saturations less than 1; and white has a saturation of 0. It was observed that, in areas where there was reflection of the sky in the flood water, the saturation was closer to zero. The value (V) of a color, also called its lightness, describes how dark the color is. A value of 0 is black, with increasing lightness moving away from black. The single-hexcone model of color space can be seen in figure 1. It can help you visualize the meaning of the H, S, and V parameters.

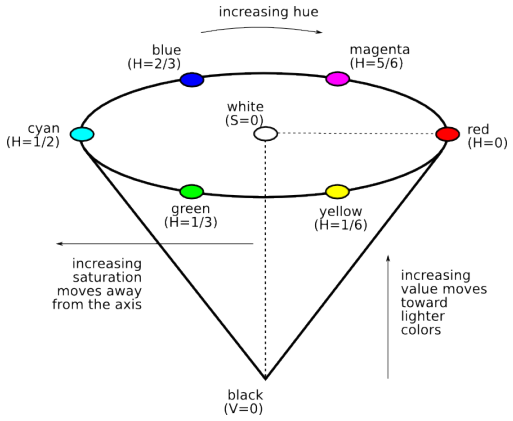


Figure 1: Single-hexcone Model of Color Space [1]

The hue of the 'water' regions is expected to be significantly different from the 'non-water' regions except for areas with mud, Since flood water is usually brownish in color. On using saturation as a second feature, better results were obtained. Saturation of 'water' regions was observed to be lower in general when compared to 'non-water' regions, except when compared to the sky.

Using hue and saturation as features could produce pretty good classification results, except for the fact that there are many reflections of the sky present in the water, and hence the sky could

be mistaken for water in the image. The solution to this problem is discussed in section 2.5, which talks about segmentation of water from the image.

2.2 Bayesian Classifier

In this section we discuss the learning of the features for the two classes, namely, 'water' and 'non-water'.

Figure 2 shows the distribution of two classes, considering just one feature. On selecting x_0 as the decision plane, optimum results can be obtained. For this classifier to work, it is required to have prior information. When $p(\boldsymbol{x}/w1) > p(\boldsymbol{x}/w2)$, the feature vector, $\boldsymbol{X}$ most likely lies toward the left of x_0 in figure 1. This means that $\boldsymbol{x}$ belongs to class 1. In our case, two features have been used, i.e. $\boldsymbol{x} = [x_1 \;\; x_2]^T$

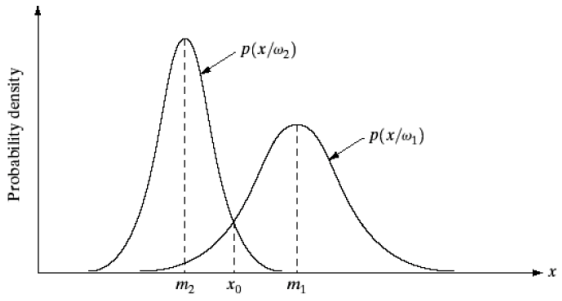


Figure 2: Bayesian Classifier[2]

The first step was to convert the RGB images into HSV images. Next, the the hue channel for each image is selected and a mean is calculated. Similarly, the mean for the saturation channel of each image is also calculated.

So, now we have values of means of the hue and saturation for each of the training images. A total of 30 values for each feature is available for each class. Now, we need to determine the probability distributions for these features. Since, the number of features available are very less, plotting a histogram is difficult. So, instead, we make the assumption that the probability distributions are Gaussian and fit them using the mean and variances of the obtained features. Since we are considering two features, we will get 3-D Gaussian curves.

Once, the distributions are obtained, likelihood ratio was used to discriminate between the two classes. Consider two classes w1 ('water') and w2 ('non-water'), then the incoming features, $\boldsymbol{x}$, can be classified as w1(w2) if:

$$l_{12} = \frac{p(\boldsymbol{x}/w1)}{p(\boldsymbol{x}/w2)} > (<) \frac{P(w2)}{P(w1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}} \tag{1}$$

It is assumed that both events are equally likely to occur, i.e. P(w1) = P(w2) = 1/2, and the risk factors are also assumed to be equal, i.e., $\lambda_{12} = \lambda_{21}$ and $\lambda_{11} = \lambda_{22} = 0$

Hence we have,

$$l_{12} = \frac{p(\boldsymbol{x}/w1)}{p(\boldsymbol{x}/w2)} > (<)1 \tag{2}$$

So, using the values of $p(\boldsymbol{x}/w1)$ and $p(\boldsymbol{x}/w2)$ from the distributions we formed in the beginning of this section, and equation (2), we can determine whether an incoming image is from the 'water' class or the 'non-water' class.

Note that these are at present cropped images which are being classified. The significance of such a classification will be much clearer when we reach the segmentation section. These distributions will be used to classify small neighborhoods in the image as 'water' or 'non-water'.

2.3 Linear Classifier (Perceptron)

The next step is to implement a linear classifier and compare the performance with the Bayesian classifier. Theoretically, it is expected that the Bayesian classifier will give the minimum error probability, so, the linear classifier, at the most, is expected to give results as good as the Bayesian classifier, i.e. it definitely cannot do better than the Bayesian classifier. The advantage of using a linear classifier though, is that no prior information is required. It is not required to know the distributions P(w1), P(w2), p($\boldsymbol{x}$/w1), and p($\boldsymbol{x}$/w2) as we did in the Bayesian classifier. All we require to implement linear classifiers is the training data. This can come in very handy when we do not have enough data to plot an accurate histogram or do not know the priors at all.

We selected Perceptron as the linear classifier in our case. On comparison with other linear classifiers, the Perceptron gave the best performance, and hence, we selected the Perceptron as our linear classifier.

To implement the perceptron, we implement the reward and punishment scheme. The N training vectors enter the algorithm cyclically, one after the other. If the algorithm has not converged after the presentation of all the samples once, then the procedure keeps repeating until convergence. A maximum iteration limit is set in order to avoid entering into an infinite loop, in case convergence is not reached. This is likely to happen when the data is not linearly separable, as is the case with our training data. This is further discussed in section 3.2.

The reward and punishment algorithm can be stated as follows:

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) + \rho\boldsymbol{x}(t) \text{ , When } \boldsymbol{x}(t) \epsilon \text{ class 1 and } \boldsymbol{W}^T(t)\boldsymbol{x}(t) <= 0 \tag{3}$$

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) - \rho\boldsymbol{x} \text{ , When } \boldsymbol{x}(t) \epsilon \text{ class 2 and } \boldsymbol{W}^T(t)\boldsymbol{x}(t) > 0 \tag{4}$$

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) \text{ , otherwise} \tag{5}$$

2.4 Segmentation

This section involves the detection of the entire water region in the flood images that were provided. We implement a multi-scale approach for the segmentation of water in the image. This method has been previously used to segment out forest/grassy areas from images in [3]. The method performs satisfactorily in our case, though there are certain problems which were faced and can be possibly resolved by further processing, or using different methods.

We also tried using the decision plane obtained from the perceptron, and simple thresholding methods in place of the modified likelihood ratio which will be discussed below. We found that the performance of the modified likelihood ratio was superior compared to the other two. Examples of the other two methods can be seen in Appendix 1.

2.4.1 Segmentation Procedure

The idea is to classify each pixel into class w1 ('water') or w2 ('non-water') based on the distribution of its neighborhood. The feature vector, $\boldsymbol{x}$, is obtained for each neighborhood by calculating the means of hue and saturation, and the likelihood ratio test is performed to determine which class that block of pixels belong to. If the block belongs to, say w1 ('water'), the center pixel is then classified as w1 ('water').

The size of this window must be chosen judiciously. A larger window provides a more accurate classification of homogeneous regions, but increases the likelihood of the window containing a clutter boundary (boundary between the 'water' and 'non-water' classes in our case). On the other hand, a smaller window leads to much more computational complexity. So, a trade off between the two must be made [3]. We select a window of 32x32 as it gives satisfactory performance.

Whenever there is a boundary present in the test window, the validity of the decision made using this method is questionable. This effect may lead to a classification bias near boundaries. To address this, the boundary proximity is determined via a simple modification of the decision made

based on the likelihood ratio. Now, instead of comparing the likelihood ratio to a single threshold, we compare it to two thresholds, a and b. Representing the likelihood ratio as l, we have,

$$l > a \text{ , Classify as 'water'} \tag{6}$$

$$a > l > b \text{ , Defer Decision (Possible boundary presence)} \tag{7}$$

$$l < b \text{ , Classify as 'non-water'} \tag{8}$$

Below we discuss a procedure used to classify the deferred pixel.

2.4.2 Deferred Pixel Classification

It is required to determine the classification of the pixels that have been deferred according the procedure discussed above. To do this we use multi-scale likelihood calculations, which allows us to perform the classification of deferred pixels as a replication of the classification procedure discussed previously. What we need to do here, is to determine on which side of the boundary the center pixel lies. If we know that, we can easily determine the class of the deferred pixel. Assuming that only one boundary may lie within a test window, the pixel may be classified by merely determining which of the two classes occupies the majority of the window.

This is done in a recursive manner, with the recurrence relation of $T(n) = T(n/4)+n^2$. The original test window is now divided into four quadrants. These four quadrants are now independently analyzed using a similar classification procedure from 2.4.1. This is repeated recursively unless, either a decision is reached, or the window becomes so small that a significant decision cannot be made. In the latter case, the decision is made based on the regions that have been classified.

2.4.3 Sky Removal

As mentioned before in section 2.1, in some cases, the sky was mistaken for water. This lead to a false detection of water even if water was actually not present. To combat this problem, we formed a novel adaptive algorithm which successfully removes majority portion of the sky from the image.

This algorithm starts checking the image from the top left corner and checks the hue of the first 'm' pixels. It makes sense to check only in the top region of the image as if sky is present in the image, it will most likely be at the top, unless the image has been rotated. Since the hue of the sky (Assuming day-time images) will usually lie between 0.5 to 0.7, pixels lying in that range are

counted. If the number of pixels with the hue in range of sky blue color is more than or equal to 'm/2', that area is deemed to be sky. The algorithm now checks the presence of sky starting from where it left off, i.e. the 'mth' pixel. The algorithm is of a recursive nature having the recurrence relation $T(n) = T(n-500) + n^2$, except for the thresholds which decrease proportionally with each iteration.

This algorithm successfully removes majority area of the sky, and makes the segmentation algorithm described above much more effective. The results of the algorithm can be observed in the Results section.

3 Results

3.1 Bayesian Classifier

Figure 3 shows the resulting distributions for 'water' and 'non-water' on using the mean of Hue(x-axis) and Saturation(y-axis) as the two features. It can be seen that there is a good separation between the two classes, i.e. they are not overlapping and could be a good selection as features.

Once the distributions for the classes have been formed, they can be used to determine the class for the incoming images using the likelihood ratio from equation (2). Assuming that that the features obtained from the incoming image are represented by the vector, $\boldsymbol{x}$, we calculate p($\boldsymbol{x}$/w1) and p($\boldsymbol{x}$/w2) from the distributions shown in Figure 3.

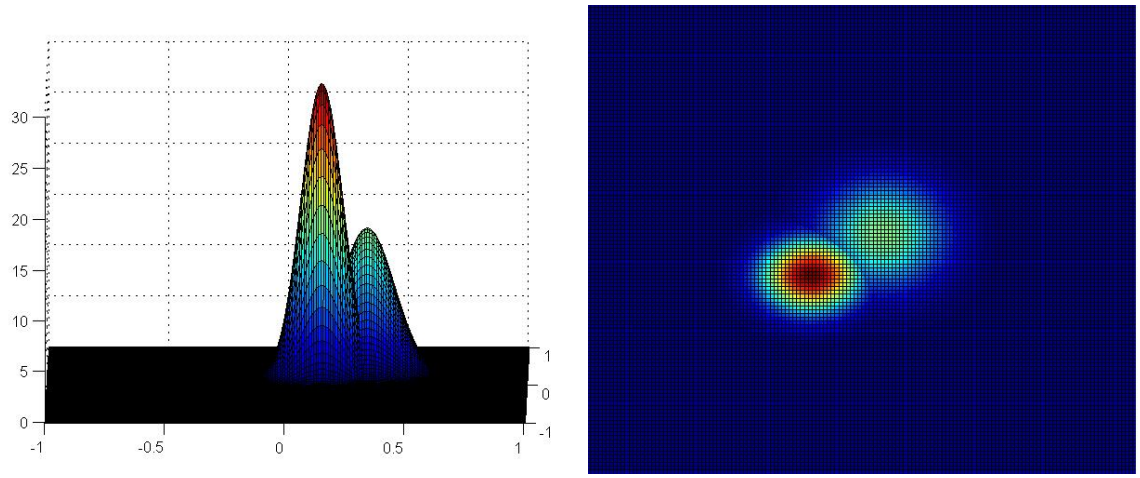


Figure 3: Distribution of 'non-water' (distribution on the left) and 'water' (distribution on the right)

A couple of successful results of using the Bayesian Classifier for classification can be seen in figure 4 and figure 5. Eleven images of each class, namely, 'water and 'non-water' were used to test the classifier. Of the twenty images, eighteen were successfully classified, yielding an accuracy of 90%

Figure 4: Images correctly classified as 'Non-Water' by the Bayesian Classifier

Figure 5: Images correctly classified as 'Water' by the Bayesian Classifier

3.2 Perceptron

In this section we show the results of using a linear classifier, perceptron in our case, for classification of the data. The distribution of the training data can be seen on the left, in figure 6. Looking at the distribution it can be concluded that while it may be possible to implement a linear classifier, the data samples would be extremely close to the decision plane, hence leading to an increased probability of error during the testing phase. This means, it is expected that the accuracy of the perceptron would be lower compared to the Bayesian Classifier.

On the right, in figure 6, the result of training the perceptron can be seen. It is seen that the it was not possible to train the perceptron for perfect classification, since the separation between the classes is non-linear.

In spite of the non-linearity issue, the perceptron gives satisfactory results in classification. We found that seventeen of the 20 images were classified successfully, yielding a success rate of 85%.

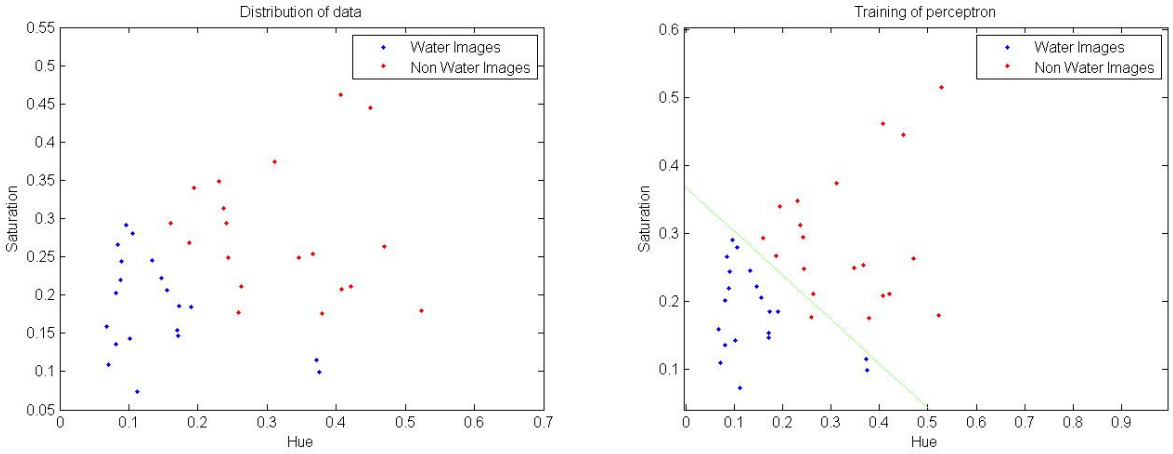


Figure 6: The image on the left shows the Data Distribution, and the one on the right shows the trained perceptron decision plane

3.3 Segmentation

This section will discuss the results of implementing the segmentation algorithm. First we go for sky removal in order to avoid confusion of sky and water. This is followed by the application of the multi-scale segmentation algorithm.

3.3.1 Sky Removal

It is observed that the sky removal algorithm is extremely effective in detecting the presence of sky in an image, and removing majority of it. In figure 7, we can see instances of sky removal which work very effectively. In figure 8, on the left, there was no sky in the original image. The algorithm recognizes this and the image stays as it is. On the left, a non-water image is displayed, which has a large chunk of sky, which was again successfully removed.

Figure 7: Images showing the removal of sky in order to avoid confusion with water

Figure 8: On the left, there was no sky in the original image, hence, it stays unchanged. On the right, the entire chunk of sky is removed successfully.

3.3.2 Detection and Segmentation of water areas

The modified form of the likelihood ratio, as discussed in section 3.3, was used to detect and segment out the water regions. On the left in figure 9, it is seen that the majority of the water area is successfully segmented out by the algorithm. The algorithm also successfully avoids the grassy areas in the lower part of the image. On the right, the segmentation algorithm's performance is displayed on another image.

In figure 10, On the left a pretty good segmentation of water is observed. On the right, it can be seen that no water regions have been detected in the 'non-water' image. There are very small regions which have been detected as water, but these areas can be rejected by simply considering a threshold for the size of connected components.

In figure 11, we can again see instances of the successful segmentation of water areas from the image. Detected areas which are very small can be rejected since flood areas would usually be large.

Figure 9: Images showing successful segmentation of water

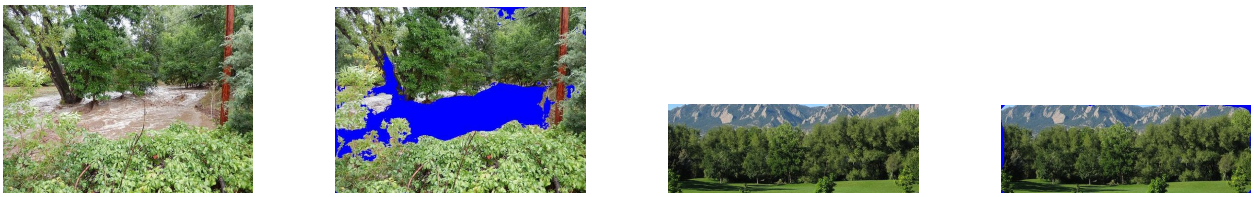


Figure 10: On the right, water is successfully segmented out from the image. On the left, since this is a non-water image, almost nothing was detected

Figure 11: Images showing successful segmentation of water

3.4 Erroneous Detection

Figure 12 shows erroneous detection of water and non-water images when using the Bayes Classifier and the Perceptron. An example of an error in the Bayes Classifier can be seen in the left most image. The next two images are examples of errors in the classification when the perceptron was used.

Figure 12: The left most image shows an erroneous detection for the Bayesian Classifier, the other two images show an erroneous detection for the linear classifier (perceptron)

In figure 13, we can see that certain areas of water have not been detected. These are mainly reflections of trees. This issue might be due to the lack of training data. It is expected that these regions would be detected successfully given a larger data set of training images.

Figure 13: Segmentation image with undetected reflections

Figure 14 is actually a decent segmentation of the water area, but along with that, the muddy area in the right top corner of the image is also detected as water. This can again be solved by using a larger training data set, since this will allow us to form a histogram and find a closer fit to the actual distribution, rather than assuming a Gaussian distribution.

Figure 14: The Segmentation images with undetected water areas

4 Conclusion

In this project we learned how to select features, implementation of Bayesian and Linear classifiers, and also implemented the segmentation algorithm described in [3]. It is observed that the segmentation algorithm gives fairly satisfactory results. Better results are expected on increasing the size of the training data set. The accuracy for the Bayesian classifier and the linear classifier was found to be 90% and 85% respectively.

5 References

[1] 4.3. The hue-saturation-value (HSV) color model. [Online]. Available: http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html. [Accessed: 10-Mar-2016].

[2] A question regarding conditional probabilities in Bayesian Classifier theory - Math Help Forum. [Online]. Available: http://mathhelpforum.com /advanced-statistics/181962-question-regarding-conditional-probabilities- bayesian-classifier-theory.html. [Accessed: 10-Mar-2016].

[3] Charles H. Fosgate, Hamid Krim, William W. Irving, William C. Karl, and Alan S. Willsky, Multiscale Segmentation and Anomaly Enhancement of SAR Imagery.

[4] A. L. Rankin, L. H. Matthies, and A. Huertas, DAYTIME WATER DETECTION BY FUSING MULTIPLE CUES FOR AUTONOMOUS OFF-ROAD NAVIGATION.

[5] S. Theodoridis and K. Koutroumbas, Pattern Recognition. .

6 Appendix

Other Methods

The decision plane obtained from the perceptron can also be used to segment the water regions in a similar fashion as we used the modified likelihood ratio. Another option would be to use simple thresholding of hue and saturation. The results are compared with the modified likelihood ratio method in figure 15, and it can be observed that the method involving modified likelihood ratio is superior. The results from the perceptron method are highly disconnected, and as expected, thresholding results are not satisfactory.

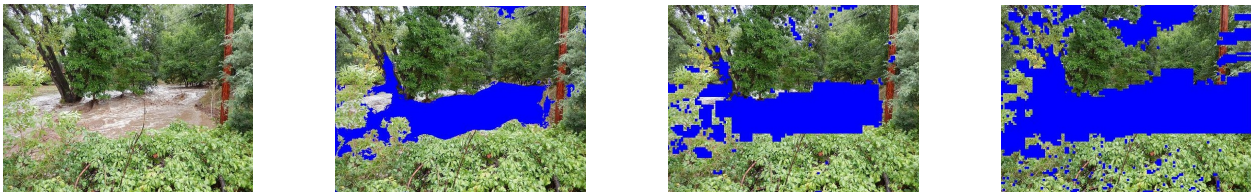


Figure 15: The Segmentation images with undetected water areas

Codes

Program for training and testing the Bayesian Classifier:

```
clc;
clear all;
close all;

% Training
train_bayesian();

% Test

Dir = 'C:\Users\Sid Roheda\Testing images';
% Test Directory Path

images = dir(fullfile(Dir, '*.jpg'));

for im = 1:1:length(images)
image = imread(fullfile(Dir, images(im).name));
%image  = imread(img_input);
test_bayesian(image);

end
```

Function - train-bayesian():

```
function [] = train_bayesian()

Dir ='C:\Users\Sid_Roheda\ImagesWithWater\Cropped_water\train';

%Train directory path, water images

directory = 'C:\Users\Sid_Roheda\NoWater';

%Train directory path, non-water images

[F1, F2] = feature_extraction(Dir, directory);
csvwrite('waterDistribution.csv',F1);
csvwrite('nonWaterDistribution.csv',F2);
end
```

Function - Feature-extraction:

```
function [F1, F2] = feature_extraction(~, ~) % Future use input parameters

% This function extracts features required for the implementation of
% bayesian classifier.

Dir ='C:\Users\Sid_Roheda\ImagesWithWater\Cropped_water\train';

% Training directory path (water images)

[a,b,c,d] = mean_var(Dir); %Mean & Var values of Hue and Sat components

images = dir(fullfile (Dir, '*.jpg'));

directory = 'C:\Users\Sid_Roheda\NoWater';

% Training directory path (non-water images)

[e,f,g,h] = mean_var(directory);

images = dir(fullfile (directory, '*.jpg'));

mu1 = [a c]; % Mean Vector
sigma1 = [b 0; 0 d]; % Covarience Mat

x1 = -1:0.01:1;
```

```
x2 = -1:0.01:1;
[X1,X2] = meshgrid(x1,x2);
F1 = mvnpdf([X1(:) X2(:)],mu1,sigma1); %pdf
F1 = reshape(F1,length(x2),length(x1));
figure; surf(x1,x2,F1); hold on;

mu2 = [e g]; % Mean Vector
sigma2 = [f 0; 0 h]; % Covariance Mat

x3 = -1:.01:1;
x4 = -1:.01:1;
[X3,X4] = meshgrid(x3,x4);
F2 = mvnpdf([X3(:) X4(:)],mu2,sigma2); %pdf
F2 = reshape(F2,length(x4),length(x3));
surf(x3,x4,F2);

end
```

Function - Mean-var:

```
function [meanhue_m, meanhue_v, meansat_m, meansat_v] = mean_var(Dir)

% This function calcualtes the mean and variance of hue and saturation
% component of images present in a directory.

images = dir(fullfile (Dir, '*.jpg'));
vargray = double(zeros(length(images),1));

% Loop for accessing all images
for j = 1:length(images)

    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
    I = K(:,:,1);
    L = K(:,:,2);

    meanhue(j) = mean(mean(I));
    meansat(j) = mean(mean(L));
end

% Retrun Parameters
meanhue_m = mean(meanhue);
meanhue_v = var(meanhue);
meansat_m = mean(meansat);
meansat_v = var(meansat);
```

end

Function - test-bayesian():

```
function [   ] = test_bayesian( image )

F1 = csvread('waterDistribution.csv');
F2 = csvread('nonWaterDistribution.csv');

K = rgb2hsv(image);

X(1) = mean(mean(K(:,:,1)));
X(2) = mean(mean(K(:,:,2)));

X(1) = X(1)*100;
X(1) = floor(X(1));
X(1) = X(1)/100;

X(2) = X(2)*100;
X(2) = floor(X(2));
X(2) = X(2)/100;

if X(1)>0
    X(1) = (X(1)/0.01) +101;
else
    X(1) = (-1*X(1)/0.01) +1;
end

if X(2)>0
     X(2) = (X(2)/0.01) +101;
else
     X(2) = (-1*X(2)/0.01) +1;
end
p1 = F1(X(1), X(2));
p2 = F2(X(1), X(2));

l12 = p1/p2;

if l12>1
    position = [10  10];
    str = 'Water Image'
    RGB = insertText(image,position,str, 'FontSize',8);

    figure; imshow(RGB);
```

```
else
    str = 'Non-water Image'
    position = [10 10];
    RGB = insertText(image,position,str, 'FontSize',8);
    figure;imshow(RGB);

end

end
```

Program for linear classifier (perceptron):

```
tic;
% Linear Classifier (Perceptron)

clear all;
close all;
clc;

Dir ='C:\Users\Sid Roheda\ImagesWithWater\Cropped_water\train';

%Training data path water

images = dir(fullfile (Dir, '*.jpg'));

for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
    I = K(:,:,1);
    L = K(:,:,2);

    meanhue(j) = mean(mean(I));
    meansat(j) = mean(mean(L));
end


Dir ='C:\Users\Sid Roheda\NoWater';

%Training data path non water

images = dir(fullfile (Dir, '*.jpg'));
vargray = double(zeros(length(images),1));
for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
```

```
    I = K(:,:,1);
    L = K(:,:,2);

    meanhuen(j) = mean(mean(I));
    meansatn(j) = mean(mean(L));
end

% Data Visualization

figure; plot (meanhue(:), meansat(:), '.b'); hold on;
plot (meanhuen(:), meansatn(:), '.r');

legend('Water Images', 'Non Water Images');
title('Distribution of data');
xlabel('Hue');
ylabel('Saturation');

X1 = [meanhue' meansat'];
X_1 = [meanhuen' meansatn'];

data = [X1; X_1];

delta_x = [ -1*ones(20,1); +1*ones(20,1) ];
label = [ones(20,1); -ones(20,1)];

%% Perceptron Training

rho = 0.7;
maxIters = 10000; % Maximum no of iterations

data = [ data, ones(size(data,1),1) ];
l_extended = size(data,2);

w_i = randn(size(data,2),1); % the initial weight vector

Niters = 0;
while( 1 ) % we assume the data is linearly seperable

  % Find the set J (the missclassified samples) with this weight vector:
  predicted_class = data * w_i;
  predicted_class(21:40) = -predicted_class(21:40);
  % negate the sign of class 2 objects
  Y = find( predicted_class < 0 );
  % find the indices of misclassified vectors
  if( isempty(Y) ) % we are done ... everything is classified correctly!
```

```
    break;
  end

  delta_w = sum( data( Y, : ) .* repmat( delta_x( Y ), 1, l_extended ), 1 ).';

  w_i = w_i - rho * delta_w;

  Niters = Niters + 1;
  if( Niters > maxIters )
    fprintf('max number of iterations= %10d exceeded\n',maxIters);
    break
  end
end

fprintf('the weight vector for perceptron algorithm is: [%d, %d, %d]',
w_i(1), w_i(2), w_i(2));

fprintf('\n');

syms x1 x2

f(x1, x2) = w_i(1)*x1 + w_i(2)*x2 + w_i(3);
ezplot(f);
title('perceptron algorithm result');
toc
fprintf('\n \n');

%% Testing

Dir = 'C:\Users\Sid Roheda\Testing images';
images = dir(fullfile (Dir, '*.jpg'));

for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    J = rgb2hsv(K);
    I = J(:,:,1);
    L = J(:,:,2);

    X_1(j) = mean(mean(I));
    X_2(j) = mean(mean(L));

    g(j) = w_i(1)*X_1(j) + w_i(2)*X_2(j) + w_i(3);

if g(j)<0
    position = [10 10];
```

```
    str = 'Non-Water Image'
    RGB = insertText(K,position,str, 'FontSize',8);

    figure; imshow(RGB);

else
   str = 'Water Image'
    position = [10 10];
    RGB = insertText(K,position,str, 'FontSize',8);
    figure;imshow(RGB);
end

end
```

Program for Segmentation of water in an image:

```
tic;

% This code is the main code for Segmentation.

clear all;
clc;
close all;

Dir = 'C:\Users\Sid Roheda\Segmentation testing images';

% Test Directory Path

images = dir(fullfile(Dir, '*.jpg'));

for im = 1:1:length(images)
original_image = imread(fullfile(Dir, images(im).name)); % read the image
[processed_image] = issky(original_image,1,1,10000,5000); % sky detection
removed_sky = removesky(processed_image); % sky is removed from the image.
figure
% subplot(2,1,1)
imshow(removed_sky); title('Sky Removed Image');

h = rgb2hsv(removed_sky); % rgb to hsv conversion

% We are going to use likelihood ratio based thresholding
f1 = csvread('C:\Users\Sid Roheda\Documents\GitHub\water_dection_classification\
Project_Codes\Segmentation\waterDistribution.csv');
f2 = csvread('C:\Users\Sid Roheda\Documents\GitHub\
water_dection_classification\Project_Codes\Segmentation\nonWaterDistribution.csv');
```

```
[r,c,f] = size(h);
him = zeros(r+32,c+32);
sim = zeros(r+32,c+32);
indexmatx = []; indexmaty = [];
mat = sim;
for i = 17:1:r+16
    for j = 17:1:c+16
        him(i,j) = h(i-16,j-16,1);
        sim(i,j) = h(i-16,j-16,2);
    end
end
for str = 1:1:r
    fprintf('Processing....\n');
    for stc = 1:1:c
        rows =  str:str+31;
        cols = stc:stc+31;
[indexmatx,indexmaty] = pixellhr(him,sim,rows,cols,indexmatx,indexmaty,f1,f2);

        for i = 1:1:length(indexmatx)
            mat(indexmatx(i),indexmaty(i)) = 255;
        end
        indexmatx = [];
        indexmaty = [];
    end

end

for i = 17:1:r+16
    for j = 17:1:c+16
        if (mat(i,j) == 255)
            removed_sky(i-16,j-16,1) = 0;
            removed_sky(i-16,j-16,2) = 0;
            removed_sky(i-16,j-16,3) = 255;
        end
    end
end

figure; %subplot(2,1,2)
imshow(removed_sky); title('Segmented Water');
end
toc;
```

Function - remove-sky:

```
function [ b] = removesky( a )
[row,col] = size(a(:,:,1));
for i = 1:1:row
    for j = 1:1:col
        if(a(i,j,1) ~= 0 && a(i,j,2) ~= 0 && a(i,j,3) ~= 0)
            break;
        end
    end
    if(a(i,j,1) ~= 0 && a(i,j,2) ~= 0 && a(i,j,3) ~= 0)
            break;
    end
end
start = i;
b(1:row-start+1,1:col,1:3) = a(start:row,1:col,1:3);

end
```

Function - pixellhr:

```
function[indexmatx,indexmaty]=pixellhr(hue,sat,rows,cols,indexmatx,indexmaty,f1,f2)

% This fucntion is for pixel based segmentation
% The likelihood ratio is calculated on a predefined neighborhood.
% Based on the decided thresholds, it goes through recursion for finding
% water pixels.

% Recursion considerably reduces the time complexity.

if(length(rows)>=4 && length(cols)>=4) % Recursion limitting condition

    rowmax = length(rows); colmax = length(cols);
    huemean = 0; satmean = 0;

     for i = rows(1):1:rows(rowmax)
        for j = cols(1):1:cols(colmax)
            huemean = huemean + hue(i,j);
            satmean = satmean + sat(i,j);
        end
    end
    huemean = huemean / (rowmax*colmax);
    satmean = satmean / (rowmax*colmax);

    huemean = (floor(huemean * 100))/100;
    satmean = (floor(satmean * 100))/100;
```

```
    % Rounding off to two decimals
    if huemean > 0
        huemean = (huemean/0.01) + 101;
    else
        huemean = (-1*huemean/0.01) + 1;
    end

    if satmean > 0
        satmean = (satmean/0.01) + 101;
    else
        satmean = (-1*satmean/0.01) + 1;
    end

    % Probabilities
    p1 = f1(huemean,satmean);
    p2 = f2(huemean,satmean);

    % Likelihood ration
    l12 = p1/p2;

    % Conditions
    if l12 > 1.8
        indexmatx = [indexmatx floor((max(rows)+min(rows))/2)];
        indexmaty = [indexmaty floor((max(cols)+min(cols))/2)];
    elseif l12 >= 0.8 && l12 <= 1.8 % Recursion
[indexmatx,indexmaty] = pixellhr(hue,sat,rows(1:round(rowmax/2)),
cols(1:round(colmax/2)),indexmatx,indexmaty,f1,f2);
[indexmatx,indexmaty] = pixellhr(hue,sat,rows(1:round(rowmax/2)),...
...cols(round(colmax/2):colmax),indexmatx,indexmaty,f1,f2);
[indexmatx,indexmaty] = pixellhr(hue,sat,rows(round(rowmax/2):rowmax),...
...cols(1:round(colmax/2)),indexmatx,indexmaty,f1,f2);
[indexmatx,indexmaty] = pixellhr(hue,sat,rows(round(rowmax/2):rowmax),...
...cols(round(colmax/2):colmax),indexmatx,indexmaty,f1,f2);
    else
    end
end
end
```

WATER DETECTION AND SEGMENTATION USING PATTERN RECOGNITION ALGORITHMS (ANN,SVM,PCA,Clustering)

ECE 759: PATTERN RECOGNITION
SPRING 2016
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Siddharth Roheda
North Carolina State University
sroheda@ncsu.edu

Prathamesh Prabhudesai
North Carolina State University
ppprabhu@ncsu.edu

Amit Rao
North Carolina State University
arao4@ncsu.edu

1. Introduction

This project involves implementation of pattern recognition techniques in order to detect and segment water regions in images. This project can be considered to be a preliminary step towards a bigger picture of being able to detect flooded areas in an image feed from social networking websites and being able to recognize such geographical areas and putting out alerts.

The first phase of the project is to create a dataset of water and non-water areas, cropped from larger images. We also choose the features for the classification of data, and discuss the reasons for the selection in the succeeding section.

Preprocessing and Selection of Features

Pre-processing involves the creation of data sets and labeling of images. The first step was to crop out 'water' and 'non-water' areas from the images. These cropped images can then be used to learn the statistics of the water regions and the non-water regions. Out of the images that were provided, two hundred images (100 Water and 100 Non-Water) were randomly selected for training purposes and around fifty images were used for testing the classifier.

Features that were selected for the classification of the regions as 'water' and 'non-water' are hue and saturation. This means, the image was first converted from the RGB model to the HSV (Hue-Saturation-Value) model. The hue (H) of a color refers to which pure color it resembles. Hues are described by a number that specifies the position of the corresponding pure color on the color wheel, as a fraction between 0 and 1. Value 0 refers to red; 1/6 is yellow; 1/3 is green; and so forth around the color wheel. All tints, tones and shades of red have the same hue. So, the water, which will usually be muddy, will have the same hue whether there are shadows, or different illumination due to images being taken at different times of the day. The saturation (S) of a color describes how white the color is. A pure red is fully saturated, with a saturation of 1; tints of red have saturations less than 1; and white has a saturation of 0. It was observed that, in areas where there was reflection of the sky in the flood water, the saturation was closer to zero. The value (V) of a color, also called its lightness, describes how dark the color is. A value of 0 is black, with increasing lightness moving away from black. The single-hexcone model of color space can be seen in figure 1. It can help you visualize the meaning of the H, S, and V parameters.

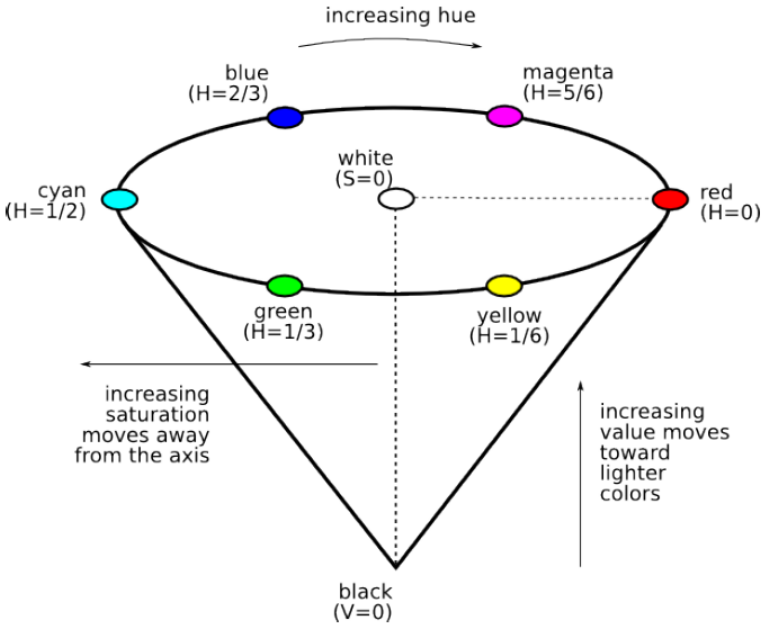


Figure 1. Single Hex cone model of Color Space

The hue of the 'water' regions is expected to be significantly different from the 'non water' regions except for areas with mud, since flood water is usually brownish in color.
On using saturation as a second feature, better results were obtained. Saturation of 'water' regions was observed to be lower in general when compared to 'non-water' regions, except when compared to the sky.

The second phase involves learning the characteristics of 'water' and 'non-water' areas in an image using Hue and Saturation as features. We implement different Classifiers to learn the characteristics of the 'water' and 'non-water' regions and test the classifier using test images. We also evaluate and compare the performance of the classifiers implemented in this project with that of the Bayesian classifier from the previous project. Using hue and saturation as features produced good classification results, except for the fact that there are many reflections of the sky present in the water, and hence the sky could be mistaken for water in the image. The solution to this problem is discussed in sections below, which talks about segmentation of water from the image.

Water Segmentation

After the classifiers are trained and tested to classify the cropped image as water or non-water, we proceed to segment water regions from the original dataset of images. The segmentation is done by reading the entire image and then dividing it into parts of 32X32 blocks, followed by, classification of each block using our classifier network. If the classifier classifies the block as water, then we segment that block else if there is ambiguity in deciding the class, the block is divided and checked for appropriate classification. The modified Likelihood ratio is used to do this, and this was discussed in detail in Project 1. This is recursively done until the block size reaches 4X4. The following figure illustrates the level of recursion.

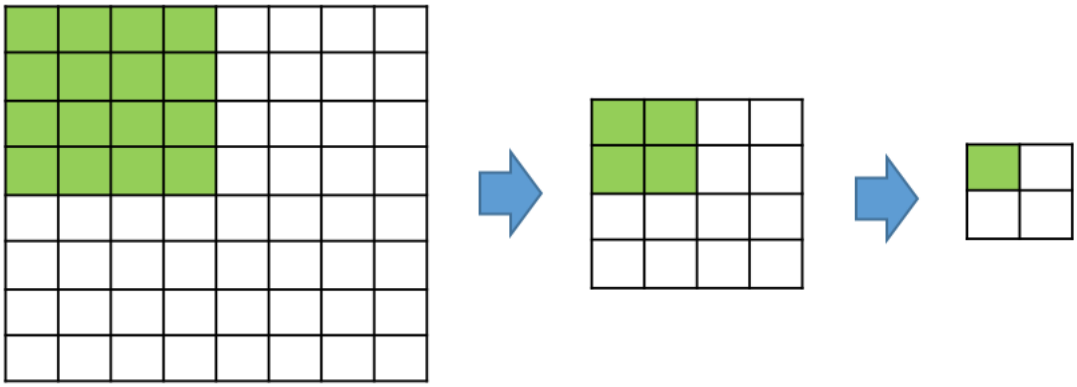


Figure 2. shows the level of recursion used to evaluate a block of image when the decision by the classifier is ambiguous

For the minimum error in water segmentation first sky is removed. **Sky removal** is an important part as the sky texture resembles with the texture of water in many aspects. To remove sky, the first few rows of the image are checked for the presence of specific hue values (mostly lying in the region of 0.667 to 1) and if some threshold is achieved these rows are removed.

Sky Removal

As mentioned earlier, in some cases, the sky was mistaken for water. This lead to a false detection of water even if water was actually not present. To combat this problem, we formed a novel adaptive algorithm which successfully removes majority portion of the sky from the image. This algorithm starts checking the image from the top left corner and checks the hue of the first 'm' pixels. It makes sense to check only in the top region of the image as if sky is present in the image, it will most likely be at the top, unless the image has been rotated. Since the hue of the sky (Assuming day-time images) will usually lie between 0.5 to 0.7, pixels

lying in that range are counted. If the number of pixels with the hue in range of sky blue color is more than or equal to 'm/2', that area is deemed to be sky. The algorithm now checks the presence of sky starting from where it left of, i.e. the 'mth' pixel. The algorithm is of a recursive nature, except for the thresholds which decrease proportionally with each iteration.

Each classifier is divided into two sections, namely, the Approach and Results. The Approach section describes the problem in detail and explains how the authors have implemented the solution. The next section presents and discusses the results obtained from the simulations which were run using MATLAB.

2. Artificial Neural Network

Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time and learn by comparing the classification of the record with the known actual classification of the record. The errors from the initial classification of the first record is fed back into the network and used to modify the networks algorithm for further iterations.

A neuron in an artificial neural network is

- A set of input values and associated weights
- A function that sums the weights and maps the results to an output

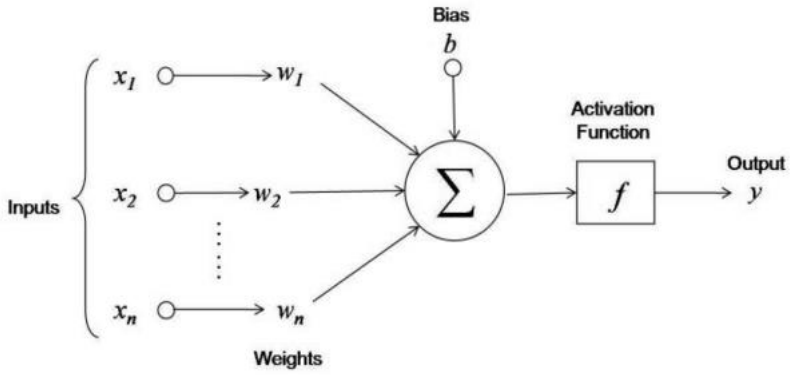


Figure 3. A model of Neuron in an Artificial Neural Network

Neurons are organized into layers: input, hidden and output. The input layer is composed not of full neurons but rather consists simply of values that are inputs to the next layer of neurons. The next layer is the hidden layer. Several hidden layers can exist in one neural network. The final layer is the output layer, where there is one node for each class. A single sweep forward through the network results in the assignment of value to each output node and the record is assigned to the class node with the highest value.

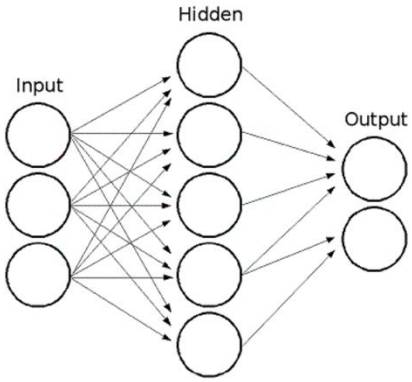


Figure 4. Model of an Artificial Neural Network

The number of layers and the number of processing elements per layer are important decisions. There is no quantifiable answer to the layout of the network for any particular application. There are only general rules (given below) picked up over time and followed by most researchers and engineers to solve the problems.

- As the complexity in the relationship between the input data and the desired output increases, the number of the processing elements in the hidden layer also increase.

- If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization of the training set, and not a true general solution.

- The amount of Training Set available sets an upper bound for the number of processing elements in the hidden layer(s). To calculate the upper bound, the number of cases in the Training Set are used and divided by the sum of the number of nodes in the input and output layers in the network. The result is then divided again by a scaling factor between five and ten. Larger scaling factors are used for relatively less noisy data. If too many artificial neurons are used the Training Set will be memorized, not generalized, and the network will be useless on new data sets. [2]

2.1 Approach

I. Training the Artificial Neural Network

In the training phase, the correct class for each record is known, and the output nodes are assigned correct values: 1 for the node corresponding to the correct class, and 0 for the others. It is thus possible to compare the network's calculated values for the output nodes to these correct values, and calculate an error term for each node (the Delta rule). These error terms are then used to adjust the weights in the hidden layers so that, hopefully, during the next iteration the output values will be closer to the correct values.

A key feature of neural networks is an iterative learning process in which records are presented to the network one at a time, and the weights associated with the input values are adjusted each time. After all cases are presented, the process is often repeated. During this learning phase, the network trains by adjusting the weights to predict the correct class label of input samples. Once a network has been structured for a particular application, that network is ready to be trained. To start the training process, the initial weights are chosen randomly. Then the training (learning) begins.

The network processes the records in the Training Set one at a time, using the weights and functions in the hidden layers, then compares the resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record. This process occurs repeatedly as the weights are corrected. During the training of a network, the same set of data is processed many

times as the connection weights are continually refined. Note that some networks never learn. This could be because the input data does not contain the specific information from which the desired output is derived. Networks also will not converge if there is not enough data to enable complete learning. Ideally, there should be enough data available to create a Validation Set. [2]

II. Feedforward and Back-Propagation

The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there are just one or two. Some studies have shown that the total number of layers needed to solve problems of any complexity is five (one input layer, three hidden layers and an output layer). Each layer is fully connected to the succeeding layer.

The training process normally uses some variant of the Delta Rule, which starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times, which are a scaling factor for global accuracy. This means that the inputs, the output, and the desired output all must be present at the same processing element. The most complex part of this algorithm is determining which input contributed the most to an incorrect output and how must the input be modified to correct the error. (An inactive node would not contribute to the error and would have no need to change its weights.) To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is computed by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function. The connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached. [2]

We continue with the algorithm till all the data in the dataset are used to train the network to classify the image as water or non-water. Once the network is trained a random set of test image are used to check for the accuracy of the network.

2.2 Results

I. Network Training

100 images of water areas and 100 images of non-water areas are used for training the classifier. Figure below shows some of the cropped images used for Training the Neural Network

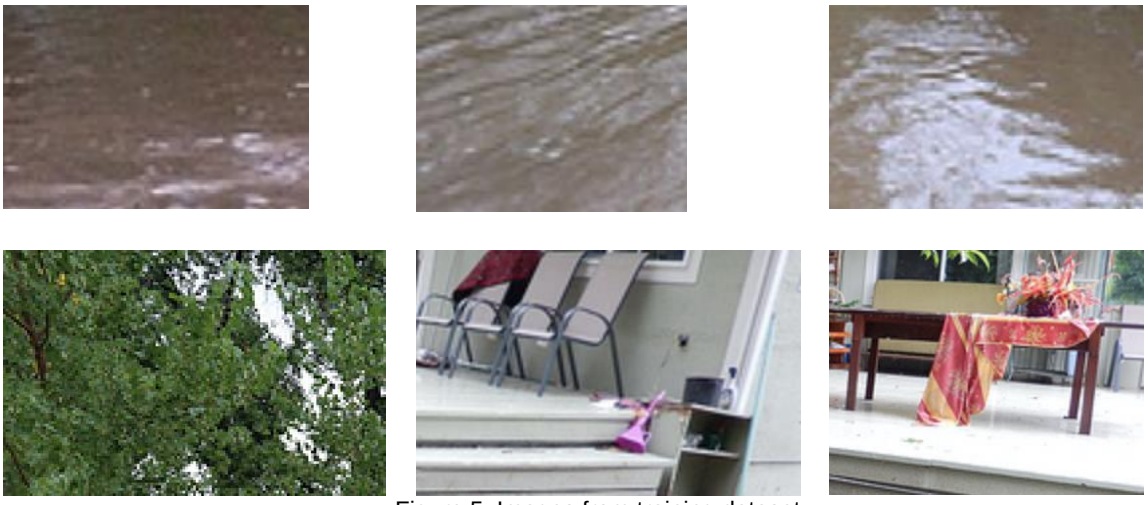


Figure 5. Images from training dataset

II. Network Testing

For testing the neural network, 30 images were chosen at random from the Test Image dataset. We observed an accuracy of 86.66% in the classification of classes. Some of the correct classification are shown in the Figure below.

Figure 6. Images classified correctly as Water by the Network

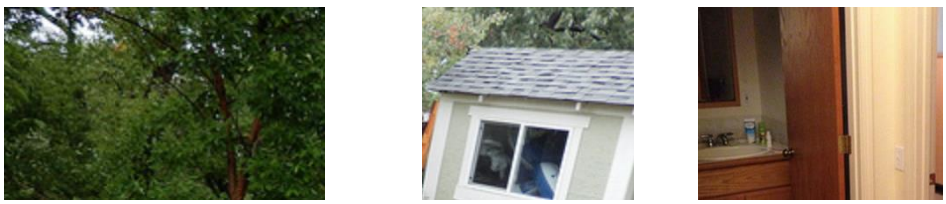


Figure 7. Images classified correctly as Non-Water by the Network

There were errors in classification due to the reflection present in the water as it changes the texture, and color of the water in that area. Some of the misclassified results are shown below

Figure 8. Images classified inaccurately as Water by the Network

We observe that all of the images have huge portions of reflections which drastically deviates the feature parameters and thus leading to wrong classification of images.

III. Water Segmentation

Figure shows the output when water detection algorithm is applied to the original image.

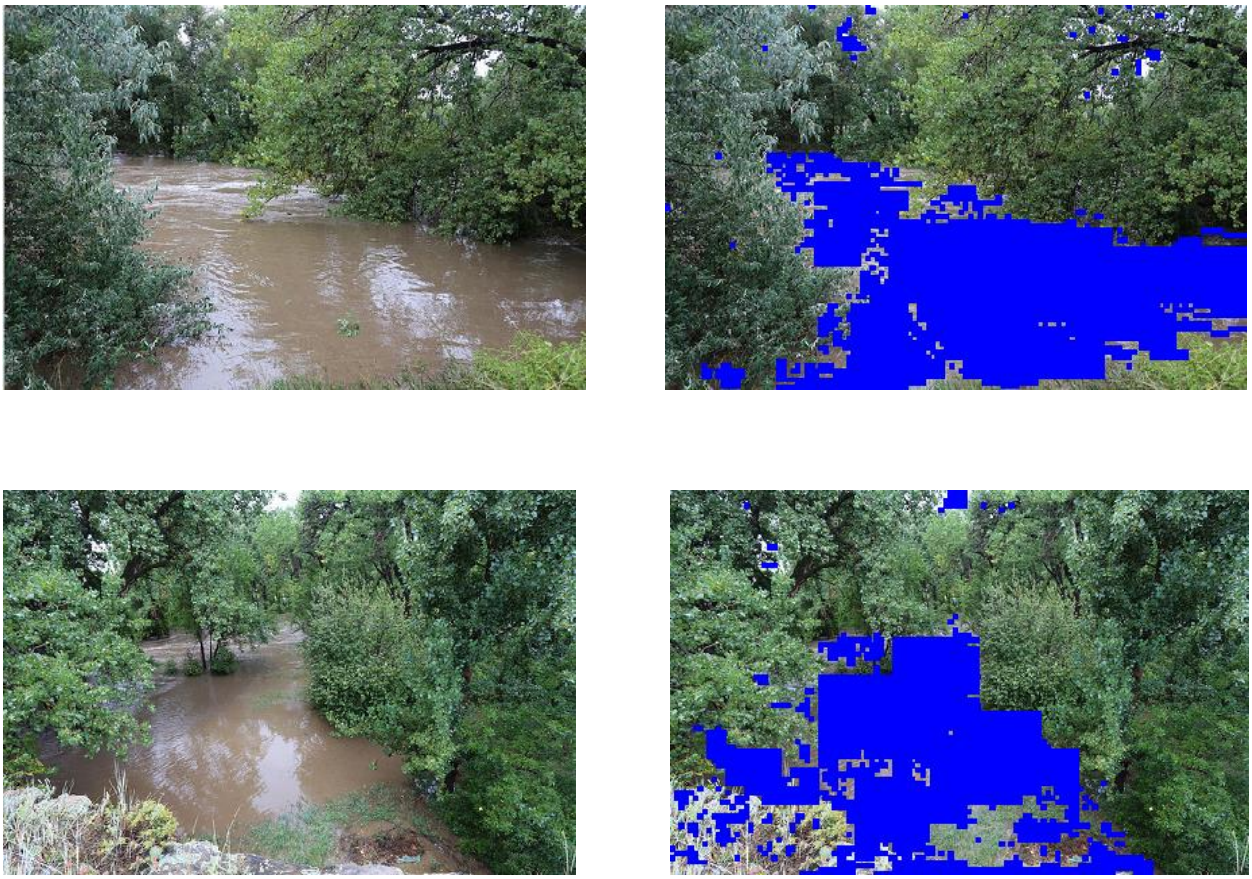


Figure 9. (left to right) Original Image, Water detected regions from the original image

2.3 Comparison With Bayesian Classifiers

The Bayesian classifier used in Project 1 had an accuracy of 90%. Bayesian Classifiers are expected to give the best classification results, and this is observed on comparing them with the implementation of Neural Networks above, where we got an accuracy of 86.66 %. The performance of neural networks is a bit lower than what we expect, and same goes for Bayesian Classifiers. The major reason behind this is the low number of images used for training. Usually Neural Networks are associated with deep learning where training data is much larger (order of thousands).

3. Support Vector Machines

SVMs are considered to be among the best supervised learning algorithm. We will first discuss the idea of margins. Consider a logistic regression, where the probability $p(y = 1|x; \theta)$ is modeled by $h_\theta(x) = g(\theta^T x)$. We would then predict "1" on an input x if $h_\theta(x) \geq 0.5$, or equivalently, if and only if $\theta^T x \geq 0$. The larger the $\theta^T x$ is, the larger also is $h_\theta(x) = p(y = 1|x; w, b)$, and thus also the higher our degree of confidence that the label is 1.[3]

Consider the following figure in which x's represent positive training samples and o's denote the negative training samples.

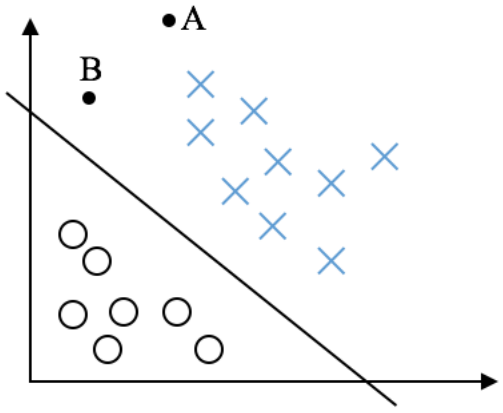


Figure 10. Illustrating linearly separable classes

Notice that the point A is very far from the decision boundary. If we are asked to make the prediction for y at A, we are quite confident that y = 1 there. Conversely, the point B is very close to the decision boundary, and while it's on the side of the decision boundary on which we would predict y = 1, it seems likely that just a small change to the decision boundary could easily have caused our prediction to be y = 0. Hence we are much more confident about our prediction at A than at B.[3]

Informally, it would be nice to make all correct and confident predictions on the training samples, so the margin (distance between the sample and hyper-plane) should be maximized.

We can now define margin as the "distance" form example to decision boundary. The margin is positive if the example is on the correct side of the decision boundary, otherwise it's negative. $Margin(x_i) = y_i f(x_i)$.[4]

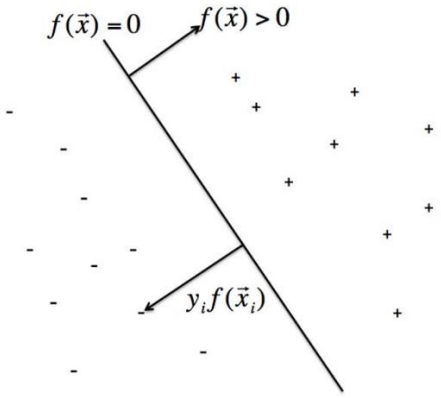


Figure 11. Illustration of margin

So we want all examples to have large margins, want them to be as far from the decision boundary as possible and that way, the decision boundary is more "stable", we are confident in all decisions. Most other algorithms (logistic regression, decision trees, perceptron) don't generally produce large margins.

3.1 Approach

I. Separable Classes

Consider the following figure.

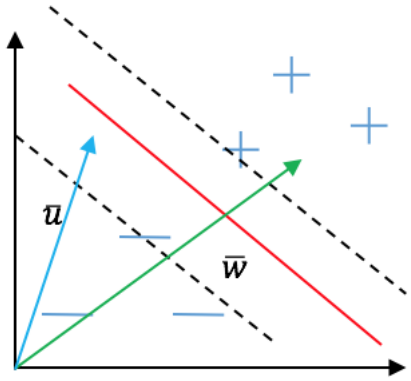


Figure 12. Illustration of separable classes used for mathematical approach

To formulate this problem, we will use the widest street approach. The red line in the above figure is the separating hyper-plane (called "street" in this approach) and the dotted lines are nothing but the gutters.
We know that any sample lying above the red line belongs to positive class and the sample lying below belongs to negative class.
Consider the vector $\bar{w}$ normal to the decision hyper-plane and $\bar{u}$ be the sample vector. If we take the projection of $\bar{u}$ on $\bar{w}$ then we can say that,

$$\bar{w}.\bar{u} \geq C \ldots (\text{some constant C depends on the length of } \bar{w})$$

Then for a positive sample, we can state a decision rule as

$$\boldsymbol{\bar{w}.\bar{u} + b \geq 0} \quad \textbf{(1)}$$

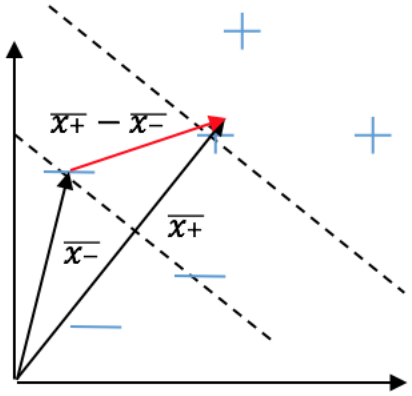


Figure 13. shows Margin as a distance vector

Now consider the samples lying on gutters (margin lines). Let vector $\overline{x_-}$ be the vector pointing to negative sample and $\overline{x_+}$ be the vector pointing to a positive class sample. So the distance between these two vectors i.e. $\overline{x_+} - \overline{x_-}$ will give us the width of this street i.e. total margin.

So using equation (1) and considering the fact that these x vectors lie on the boundary of the margin, we can write

$$\bar{w}.\overline{x_+} + b \geq 1 \quad (2)$$
$$\bar{w}.\overline{x_-} + b \leq -1 \quad (3)$$

Let us assume y_i such that $y_i = +1$ for positive samples and $y_i = -1$ for negative samples.
Multiplying y_i with equations (2) and (3), we get

$$\overline{(w}.\overline{x_+} + b)y_i \geq 1 \quad (4)$$
$$\overline{(w}.\overline{x_-} + b)y_i \geq 1 \quad (5)$$

Since after the multiplication of y_i we get answer 1 for both positive and negative samples we can write for every instance of x_i

$$y_i(\bar{x}_i.\overline{w_i} + b\ ) - 1 \geq 0 \quad (6)$$

The points we are considering are on the gutters we can write,

$$\boldsymbol{y_i(\bar{x}_i.\overline{w_i} + b\ ) - 1 = 0} \quad \textbf{(7)}$$

The width of the street = margin = $(\overline{x_+} - \overline{x_-}).\frac{\bar{w}}{||\bar{w}||}$ (8)

$\therefore width = \frac{2}{||\bar{w}||}$ (9)

So basically we want to maximize this width i.e. equation 10.

This maximization problem can be converted to a minimization problem for mathematical convenience. So we will minimize $\frac{1}{2}\left|\left|\bar{w}\right|\right|^2$.

To find extremum of the function with constraints, we have to use the Lagrange Multipliers.

$L = \frac{1}{2}\left|\left|\bar{w}\right|\right|^2 - \sum\alpha_i[y_i(\bar{w}.\overline{x_i} + b) - 1]$ (10)

We will now differentiate this function with $\bar{w}$.

$\frac{\partial L}{\partial w} = \bar{w} - \sum\alpha_i y_i x_i = 0$ (11)

$\boldsymbol{\bar{w} = \sum_i \overline{x_i}\,\alpha_i y_i}$ **(12)**

$\frac{\partial L}{\partial b} = -\sum\alpha_i y_i = 0$ (13)

$\boldsymbol{\sum\alpha_i y_i = 0}$ **(14)**

Substituting equation (12) and (14) in equation (10), we get

$L = \frac{1}{2}(\sum\alpha_i y_i x_i)(\sum\alpha_j y_j x_j) - (\sum\alpha_i y_i x_i)(\sum\alpha_j y_j x_j) - b\sum\alpha_i y_i + \sum\alpha_i$ (15)

$L = -\frac{1}{2}(\sum\alpha_i y_i x_i)(\sum\alpha_j y_j x_j) + \sum\alpha_i$ (16)

$\boldsymbol{L = \sum\alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i x_j}$ **(17)**

So we can see that the maximization of L mainly depends on the $\boldsymbol{x_i.x_j}$ $(dot\ product\ of\ samples)$. These are nothing but the **support vectors**.

Therefore, the decision rule can also be stated as

$$\boldsymbol{\sum\alpha_i y_i x_i.\bar{u} + b \geq 0}. \quad (18)$$

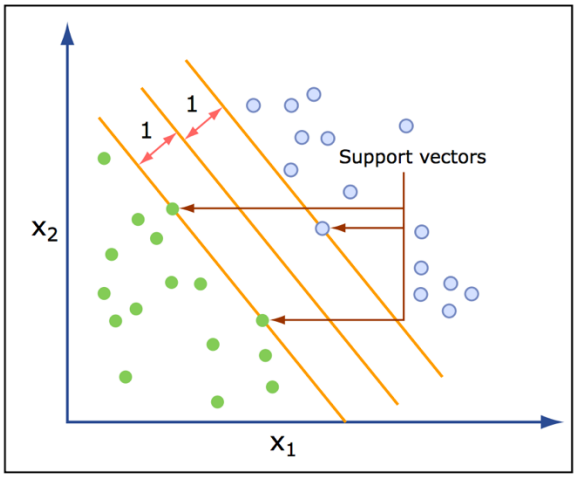


Figure 14. Illustration of Support Vectors [4]

II. Non-separable classes and Kernels

Whenever our input features are non-separable by a hyper-plane, we may want to use some features $\phi(x)$ instead of original input features x. To do so, we may want to just change the inputs $x \rightarrow \phi(x)$ everywhere.

Since the algorithm can be written entirely in terms of the inner products $< x, z >$, we would just replace all those inner products with $< \phi(x), \phi(z) >$. Specifically, given a feature mapping ϕ, we define the corresponding Kernel to be

$$\boldsymbol{K(x,z) = \phi(x)^T \phi(z)} \quad \textbf{(19)}$$

We would now be using the features ϕ for learning. Then by using the algorithm by calculating $K(x,z)$, we can get SVMs to learn in the high dimensional feature space given by ϕ.

For the purpose of this project, authors have used the radial basis function kernel or Gaussian kernel which is given by

$$\boldsymbol{K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)} \quad \textbf{(20)}$$

This is a reasonable measure of x and z's similarity, and is close to 1 when x and z are close, and near 0 when x and z are far apart.

So while mapping data to a high dimensional feature space via ϕ does generally increase the likelihood that the data is separable but we can't guarantee that it will always be so. Also, in some cases it is not, clear that finding a separating hyper-plane is exactly what we would want to do, since that might be susceptible to outliers. Consider the figures below. Left figure shows an optimal margin classifier, and when a single outlier is added in the upper-left region (right figure), it causes the decision boundary to make a dramatic swing and the resulting classifier has a much smaller region. [3]

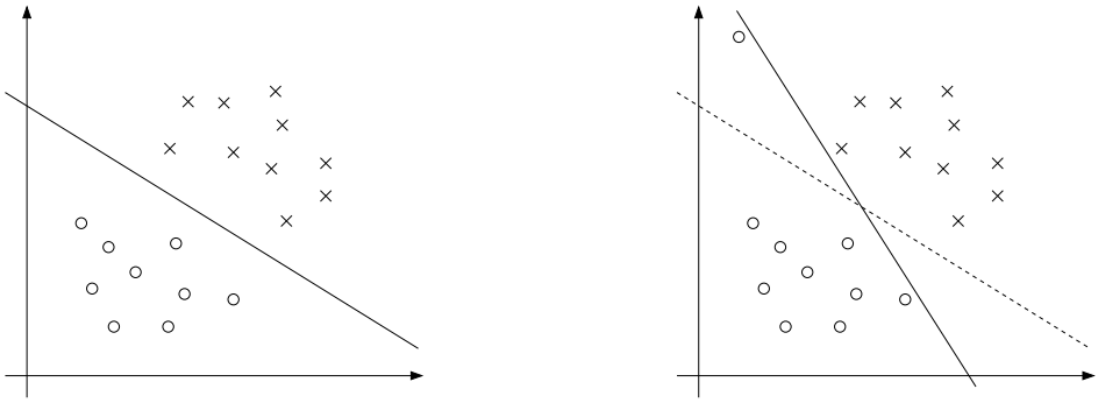


Figure 15. Shows changes in the separating hyperplane

To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers, we reformulate our optimization using l_1 regularization as follows:

$$\boldsymbol{\min_{\gamma,w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i} \quad \textbf{(21)}$$

such that $y^i(w^T x^i + b) \geq 1 - \xi_i, \qquad i = 1, \ldots\ldots, m$
$\xi_i \geq 0, i = 1, \ldots\ldots, m$

The examples are now permitted to have function margin less than 1, and if an example has function margin $1 - \xi_i$, we would pay a cost of the objective function being increased by $C\xi_i$. The parameter C controls the relative weighting between the twin goals of making the $||w||^2$ small and of ensuring that most examples have function margin at least 1.

III. Sequential Minimal Optimization

SMO is a type of coordinate ascent algorithm, but adapted to SVM so that the solution always stays within the feasible region.

Consider equation (15). Let's say we keep $\alpha_2, \ldots\ldots, \alpha_m$ fixed and take a coordinate step in the first direction i.e. change $\alpha_1$to maximize the objective in equation (15). Looking at constrain in equation (15) we cannot have a feasible solution and it will yield that $\alpha_1$is also fixed. So if we want to update any of the $\alpha_i's$, we need to update at least 2 of them simultaneously to keep the solution feasible. We will go on optimizing the objective over $\alpha_i's$ and the value will be used in the next iteration of optimization. [4]

3.2 SVM Classifier Training

For training the classifier, authors have chosen the data set of 200 images, (100 images having water and 100 without water). Gaussian (Radial Basis Function) kernel of standard deviation 1 and for optimization we have used the Sequential Minimal Optimization. SMO provides the values of alpha and bias. Classifier creates a structure with fields such as support vectors and their respective indices, alpha, bias and the kernel function used.

3.3 SVM Classifier Testing

Image to be tested is chosen and its features are calculated. The features are provided to the classifier along with the structure generated during classification. The radial basis function is evaluated on these parameters keeping the parameters obtained from classifier constant.

3.4 Results

Test Images used for the training of an SVM classifier (cropped regions of water and other areas)



Figure 16. Images from training dataset

The classifier is trained to deal with the reflections too. 100 images of water areas and 100 images of non-water areas are used for training the classifier.

I. Classifier Training

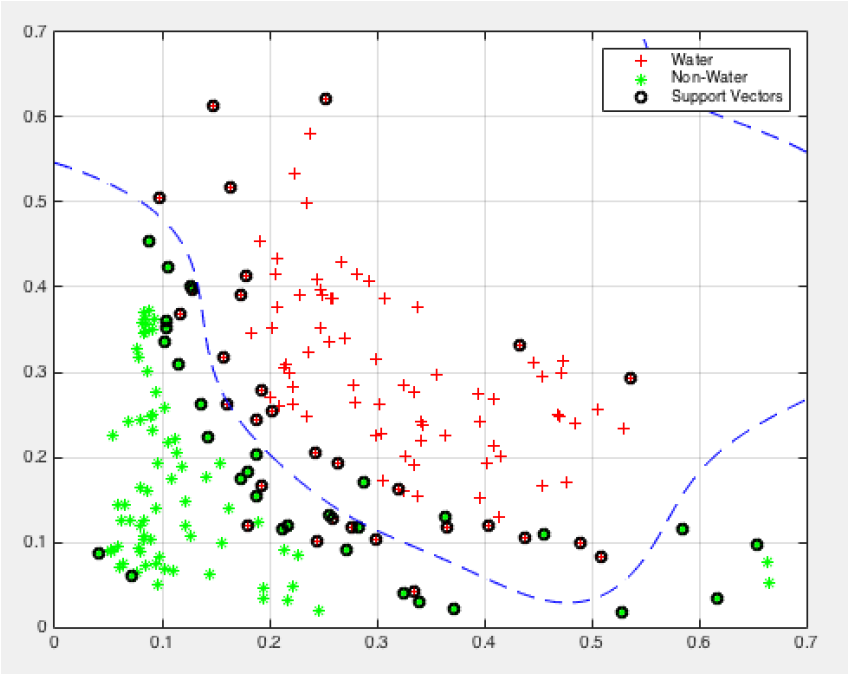


Figure 17. SVM training output. Blue line shows the separation hyperplane $\sigma = 1$

II. Classifier Parameters:

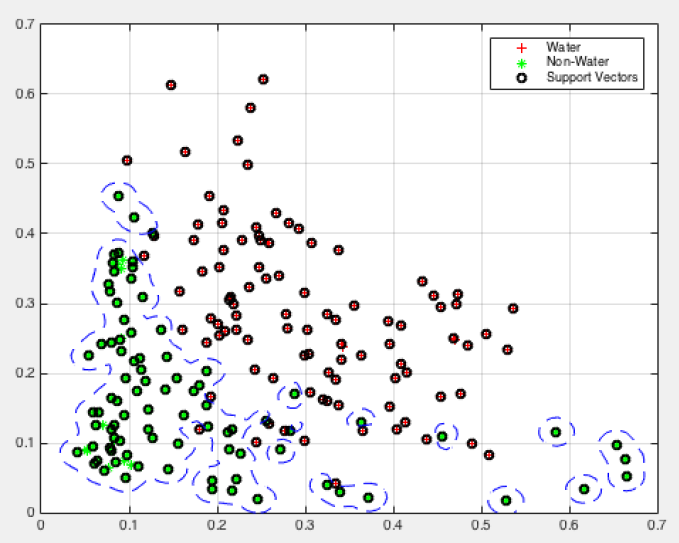
SupportVectors	*61x2 double*
Alpha	*61x1 double*
Bias	-0.0067
KernelFunction	@rbf_kernel
KernelFunctionArgs	*1x1 cell*
GroupNames	*200x1 double*
SupportVectorIndices	*61x1 double*
FigureHandles	*[ ]*

Figure 18. SVM parameters

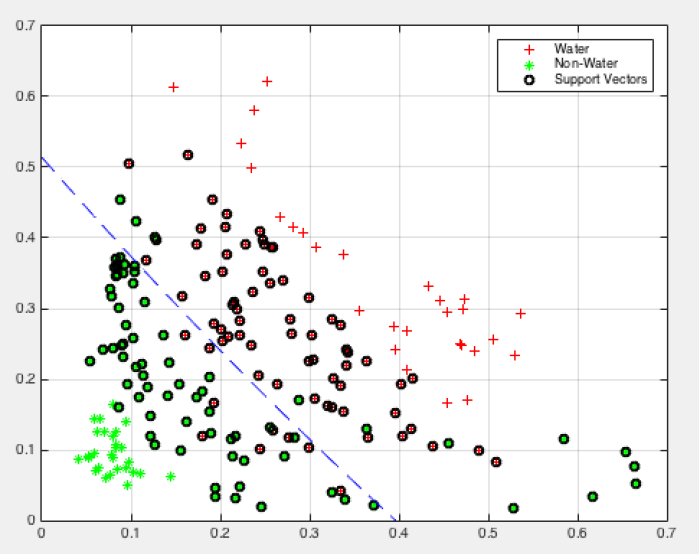
These parameters are then used for the testing purpose.

Effect of change in standard deviation of Gaussian kernel:

$$\sigma = 0.1$$



$$\sigma = 10$$



III. Classifier Testing

50 images of water and 50 images of non-water are tested. We achieved 94% accuracy in the classification. Some of the accurately classified images are shown in the figure below

Figure 19. Images classified correctly as Water by the classifier

Figure 20. Images classified correctly as Non Water by the classifier

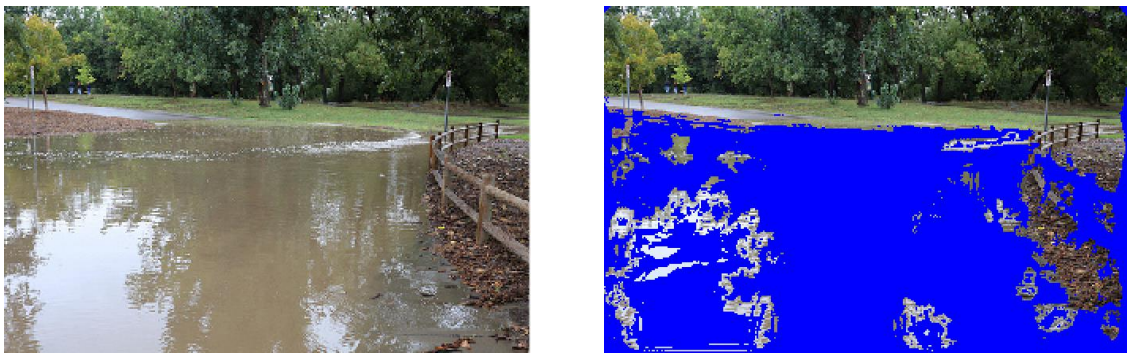
There are errors in classification which usually happens because of the reflection present in the water as it changes the texture, and color of the water in that area.

Figure 21. Images misclassified as Non Water by the classifier

All of these images have huge portions of reflections which drastically deviates the feature parameters thus leading to misclassification.

IV. Water Segmentation

It was observed that the segmentation works better than the linear classification algorithms. The Water segmented regions are shown in the figure below.



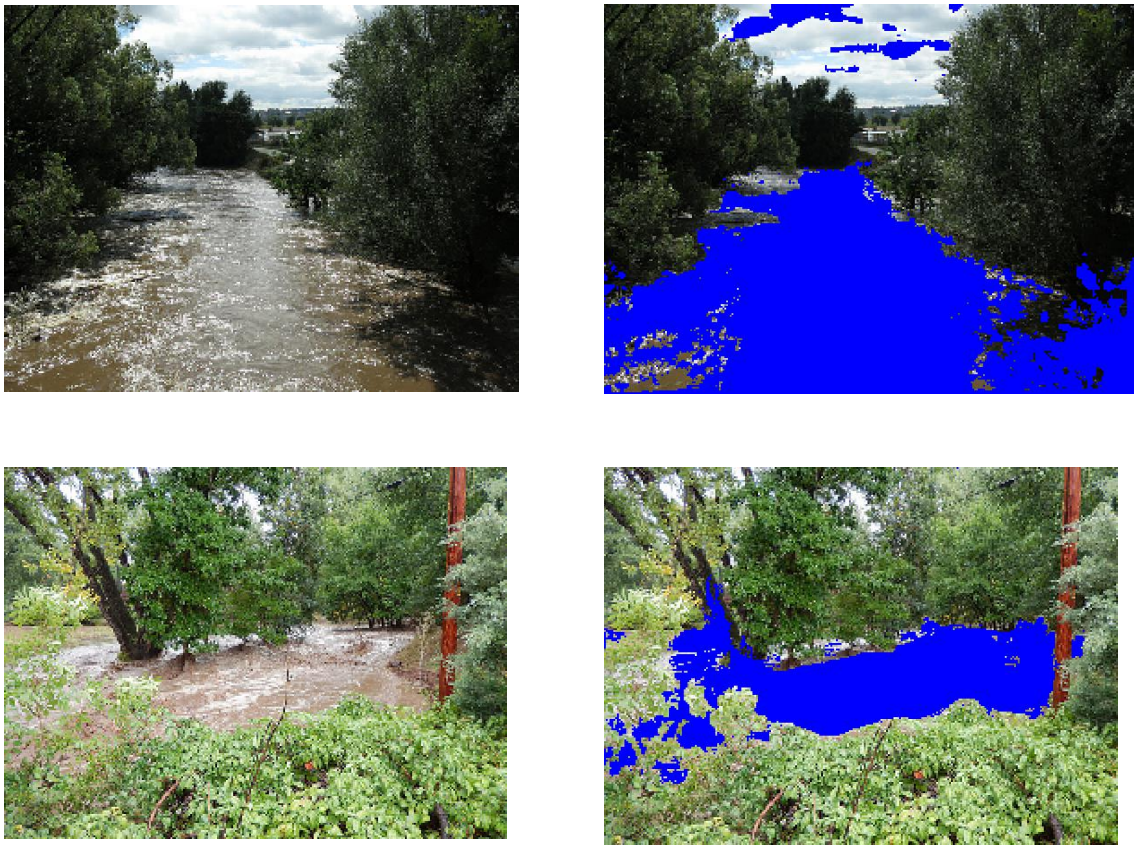


Figure 22. (left to right) Original Image, Water detected regions from the original image

3.5 Comparison of SVM and Neural Networks

The performance of SVM is found to be much better compared to Neural Networks. The Neural Networks provided classification accuracy of 86.66 %, while SVM provided an accuracy of 94%. In cases like ours, where the training data is less, it is expected that Neural Networks will not be able to outperform SVM, which works pretty well even with low training data.

4. Principal Component Analysis

4.1 Feature Selection and Motivation

The features selected originally, during Project 1, were the mean values of hue and saturation. These gave satisfactory results in classification as well as segmentation. In this case we have training and testing data which is 2 dimensional. Now, in this part, we aim to try and reduce the dimensions. Reducing to a single dimension can help reduce the computational complexity to ½ .

Here, we use Principal Component Analysis to reduce the dimensions of the data. This process is discussed in the next section. This is then compared with results from the Bayesian classifier from project 1 (part 3).

4.2 Dimension Reduction Using PCA

Principal component analysis (**PCA**) is a statistical procedure that uses the results from the Karhunen-Loeve transform to reduce the dimensions of a given data set by making use of variables called **principal components**. These principal components are nothing but the Eigen vectors of the covariance matrix of the training data. In our case, we originally used two features, so we will get a covariance matrix of 2x2. After performing Eigen Value Decomposition (myPCA.m) we get two Eigen vectors, as is expected. We have a major Eigen vector and a minor Eigen vector. The major Eigen vector defines the direction of the best fitting line, and is expected to give best separation in 1-D space. This is not necessarily true, as illustrated in figure below.

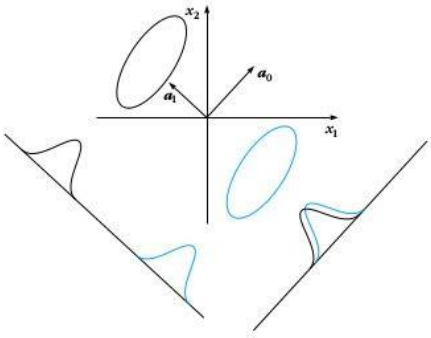


Figure 23: Better Separation on the minor Eigen vector rather than the major Eigen vector [1]

So, if we project the data onto one of the Eigen vectors, we can convert the 2D data into 1D data. This is difficult to do for the more practical cases where the data is non-linear. This is the case with our data too. The results of performing PCA are discussed in the results section.

4.3 Classification using Bayesian Approach

In this section we discuss the learning of the features for the two classes, namely, 'water' and 'non-water'. In order to classify data, we adopt a Bayesian Classification technique. This is similar to the technique used in Project 1, and we will later compare the results with that of Project 1. The results in Project 1 involved the use of 2D data, while here we are considering the data in the reduced dimensionality.

The figure below shows the distribution of two classes, considering just one feature (which is the case after the reduction of dimensions) . On selecting x_0 as the decision plane, optimum results can be obtained. For this classifier to work, it is required to have prior information. When p(**x**/w1) > p(**x**/w2), the feature vector, **X** most likely lies toward the left of x_0 in the figure. This means that **x** belongs to class 1.

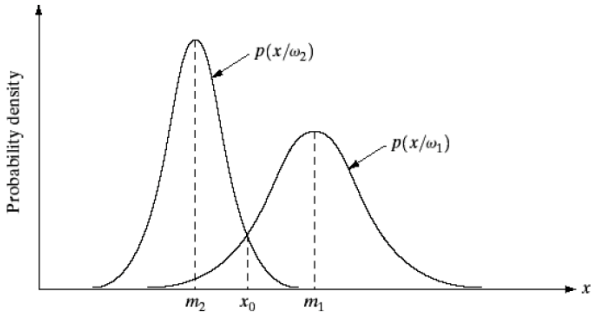


Figure 24: Bayesian Classifier [5]

The first step was to convert the RGB images into HSV images. Next, the hue channel for each image is selected and a mean is calculated. Similarly, the mean for the saturation channel of each image is also calculated. This gives us the data distribution as shown in figure 25. The data is then reduced to a single dimension using PCA. The data in 1 dimension is then used to fit Gaussian curves for each class. Likelihood ratio is further used in order to classify the test images as water or non-water. The results for this process are seen in the next section.

4.4 Results

I. Dimension Reduction using PCA

Data Distribution in two dimensions:

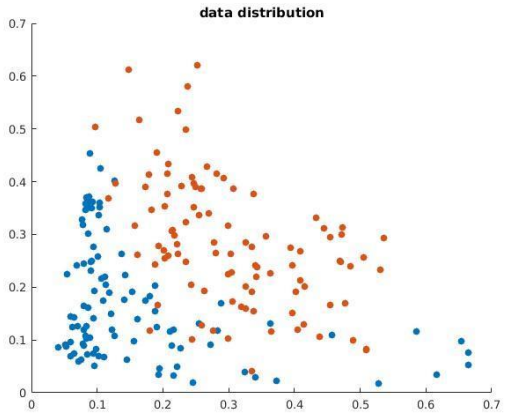


Figure 25: Data Distribution in 2D

KL transform was performed on the training data, and the data was projected onto the minor Eigen vector, as the separation in this case were found to be better than using the major Eigen vector. The figure below displays the original data in two space, and the data projected onto the Eigen vector. The separation isn't found to be so good. The major reason for this is that the data is non-linear. A kernel approach, similar to that in SVM, could be used if further separation is required.

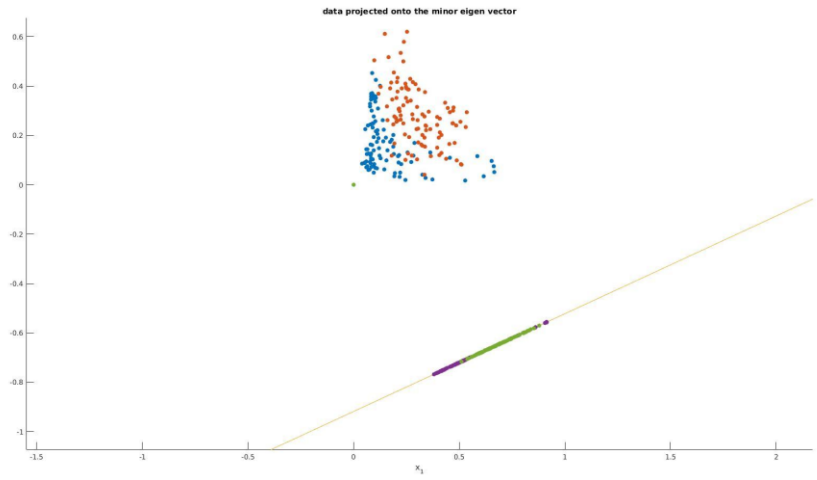


Figure 26: Projection of data onto the minor Eigen Vector

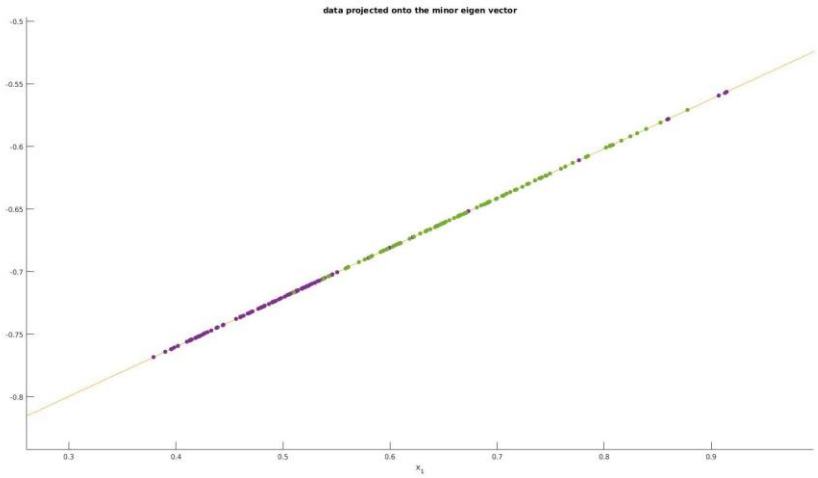


Figure 27: Zoomed in view of the projection to make the separation clear. Purple - Water and Green - Non-water

As can be seen from the figures above, there is overlapping of the training data, and the results are not expected to be good. A perfect separation is difficult to obtain in cases involving data which is non-linear in the original space. But, in spite of that, we can get correct classification to some extent.

II. Classification of data using Bayesian Approach

As discussed earlier, the 1D data obtained from PCA is used to form the Gaussian distributions for each class. This can be done by taking the mean and variance of all the vectors in each class and modelling a Gaussian with those parameters. The figure below shows the modeled Gaussian curves for each class.

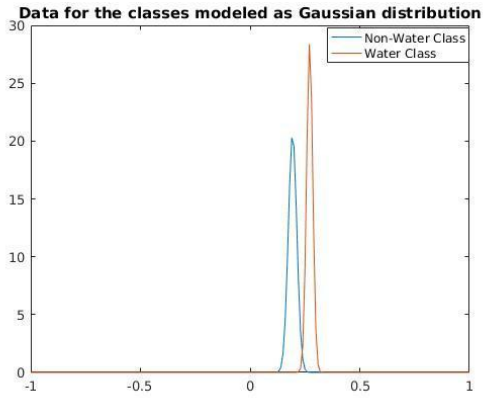


Figure 28: Data for Classes modeled as Gaussian Distribution

The Gaussian Curves in the figure above have some separation, but the performance is not expected to be as good as the case with two features. On using this distribution to classify the test images, using the likelihood ratio as discussed in the Bayesian Classifier section, we get an accuracy of 76.66 %. This is not a satisfactory result. This result is further compared to the Bayesian results from Project 1 in section 5.2 - II.

Some of the successful classifications were:

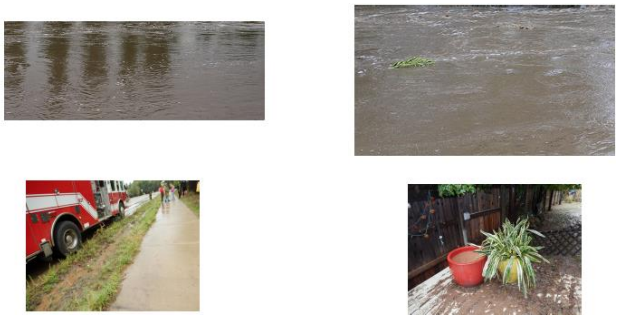


Figure 29: Succesful Classifications

5. Scatter Matrices

The scatter matrices provide information about how the data in the l-dimensional space (2D here) is distributed. There are two kinds of scatter matrices used here to find a discrimination measure, which defines the discrimination properties of the selected features.

The scatter matrices are defined as below:

- Within Class Scatter Matrix:

$$S_w = \sum_{i=1}^{M} P_i \Sigma_i$$

Where Σ_i is the covariance matrix for class ω_i.

$$\Sigma_i = E\left[(x-\mu_i)(x-\mu_i)^T\right]$$

P_i is the a-priori probability of class ω_i, that is, $P_i = n_i/N$, where n_i is the number of samples in class ω_i, out of a total of N samples.

Trace{S_w} is a measure of the average, over all classes, variance of the features.

- Between Class Scatter Matrix:

$$S_b = \sum_{i=1}^{M} P_i(\mu_i-\mu_o)(\mu_i-\mu_o)^T$$

Where μ_o is the global mean vector.

$$\mu_o = \sum_{i}^{M} P_i\mu_i$$

Trace{S_b} is a measure of the average, over all classes, distance of the mean of each individual class from the respective global value.

- Mixture Scatter Matrix:

The mixture scatter matrix is defined as:

$$S_m = S_w + S_b \quad S_m = S_w + S_b$$

S_m is defined as the covariance matrix of the feature vector with respect to the global mean.

5.1 Discrimination measure

The measure used to quantify the the discrimination provided by the selected features is the J3 measure. This measure can be defined as:

$$J_3 = trace\left(S_w^{-1}S_m\right)$$

For good separation between classes, we want trace{S_b} to be high, and for compact clusters we want trace{S_w} to be low. This means, J_3 is expected to be high if features yielding high discrimination between classes are selected. So we can say, higher the J_3 measure, easier (computationally less complex) it would be to get good classification results. This means, we favor higher values of the J_3 measure for the given data.

5.2 Results

I. Discrimination Measure using Scatter Matrices

The following results were obtained for the 2D data:

```
Sw =
    0.0156     -0.0061
   -0.0061      0.0135

Sb =
    0.0591      0.0568
    0.0568      0.0547

Sm =
    0.0747      0.0507
    0.0507      0.0682

J3 = 15.6020
```

The value of the J_3 measure clearly indicates that the data is not very well separated, and this is true, since our data is non-linear. A practical problem like this one is expected to have a non-linear data distribution, and increasing the number of features will not necessarily make it better separated, in fact, it may worsen the condition.

After reduction of dimensions, the 1D data was analyzed using scatter matrices and the J_3 measure. The results obtained were as follows:

```
Sw = 0.0167
Sb = 0.0556
Sm = 0.0723

J3 = 4.3393
```

As can be seen, the J_3 measure has drastically reduced. This is again expected since in the lower dimension, the overlap between the classes was much more and hence, the discrimination provided by the selected features is naturally less.

This suggests that it may not be wise to reduce the dimensions of the data. And in our case, since the originally selected dimensions are two, the cost of misclassification on using 1D distribution of data is much higher than using a higher number of dimensions. This is apparent from the results of Bayesian classifiers compared in the next subsection.

II. Comparison of Bayes Approach from Project 1 with Approach involving PCA

As was observed in the results above, using 2D data provides much better discrimination between classes (higher J_3 measure), compared to using 1D data obtained after implementing PCA. And as expected, the classification results from using PCA followed by Bayesian classifier, are worse compared to the results obtained from implementing Bayesian approach with 2D data. Accuracy obtained for Bayesian classification in Project 1 was as high as 90%, compared to a much lower result of 76.66 % in case of implementation involving PCA.

The segmentation results from both the approaches were also compared, and as can be seen in the figures below, using 2D data gives far superior results. The time taken to perform segmentation on the image below in case of the Bayesian approach on 2D data was 308.12 seconds, while the time taken to run the PCA approach was 203.66 seconds. Hence, on comparing these results, we can conclude that the cost of reducing the number of dimensions, in terms of accuracy, is much higher than than the computational advantage obtained from the reduction of dimensions using PCA.

Figure 30: The image on the left is the original image, the one in the center is the result of using the PCA approach to reduce dimensions, and the one on the right is the result obtained from Bayesian approach from project 1

6. Higher Number of Dimensions

In this project, a higher number of classes can be easily defined. For example, we could define three classes as 'water', 'sky', and 'trees'. But since, the bigger picture is to be able to detect floods in various geographical areas, it does not make sense to detect sky or trees. We wish to detect the 'water' areas, and the rest are clutter for us, and hence, have been correctly included into a single class, that is 'non-water'.

Also, selection of a higher number of features may be required in order to achieve satisfactory classification in case of higher number of classes. For example, if we consider 'sky' as one of the classes, there will be large overlap between the 'water' and 'sky' classes with the current features. This problem has been dealt with sky removal during pre-processing, which was discussed briefly earlier in this report, and in detail in the Project 1 report. This procedure gets rid of the sky since we do not really require that area. But, this will not be true if we consider 'sky' as a class. Hence, we may want to consider more features in such a case.

7. Clustering

Cluster analysis divides data into groups or clusters that are meaningful and useful. Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. This is an unsupervised method for classification where we only have data, belonging to all classes clubbed together and we have to find the classes analyzing the data only [6].

The goal behind clustering is that the data points within a same cluster should be similar or related to each other and data points from different groups should be unrelated. The greater the similarity or homogeneity

within a group and the greater the difference between groups, the better the clustering is.

The definition of the cluster is actually imprecise and it hugely depends on the nature of the data and required or desired results.

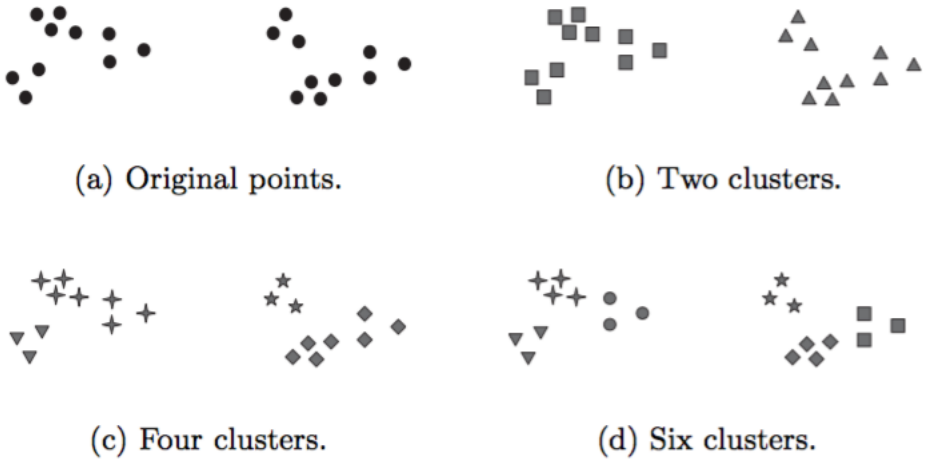


Figure 31: Showing different ways of clustering with the same data [5]

We are going to use the K-means algorithm for clustering.

7.1 K-means Clustering Algorithm:

This algorithm is a prototype-based, partition clustering technique that attempts to find already decided K clusters, which are represented by their centroids.

In this algorithm, we first choose K initial centroids (randomly or k-median problem), where K is already defined which is number of clusters desired. Each data point is then assigned to the closest centroid and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes cluster or equivalently, until the centroids remain same.[5]

This assignment of the data points to the respective centroid is done on the basis of the distance between the data point and centroid. The distance from a point which needs to be assigned to a centroid (cluster) is calculated from all the K centroids and the centroid having the minimum distance is chosen.

Now there are two types of distances we can use to calculate the distance between centroids and data points.

- **Euclidean Distance**: It is the straight line distance between two points. The Euclidean distance between two points p and q can be given as

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{(\sum_i (q_i - p_i)^2}$$

- **Mahalanobis Distance**: It is a measure of the distance between a point and some distribution D. Consider the set of observations x = $(x_1, x_2, x_3, \ldots, x_N)^T$with mean μ = $(\mu_1, \mu_2, \mu_3, \ldots, \mu_N)$ and covariance matrix S then, the Mahalanobis distance is given as

$$d_m(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}$$

So the Mahalanobis distance between the two random vectors, x and y can be defined as

$$d_m(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)}$$

Number of Clusters:

Since we expect that there will be two classes “water” and “non-water”, we will keep k = 2.

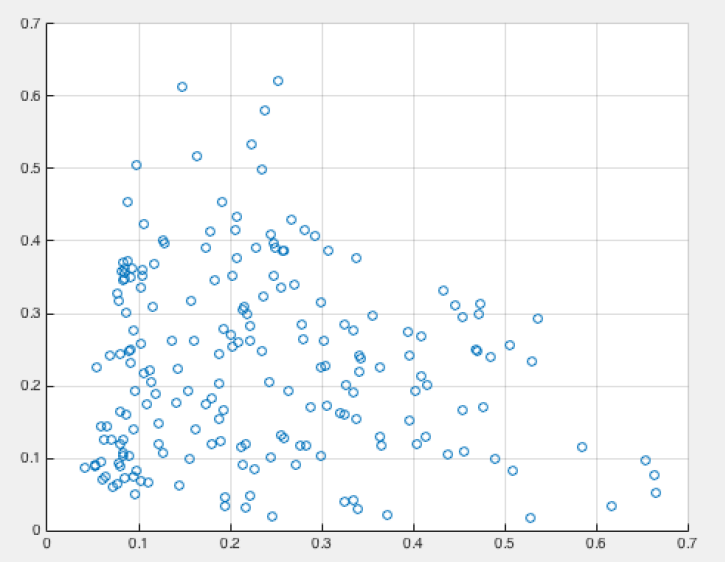


Figure 32: Initial Data

Initial Centroids:

First the data is sorted feature wise. Then the median is calculated and the data is subdivided into two parts at median. Then the median of each part is calculated. These two medians are taken as the initial centroids for the clusters.

K-means algorithm using Euclidean Distance:

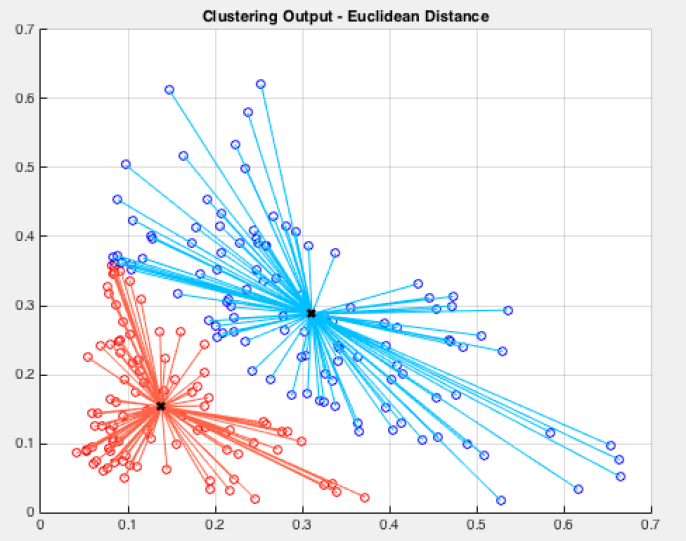


Figure 33: Clusters created using Euclidean Distance

The black points are the centroids of the clusters. Orange represents cluster 1 and Blue represents the second cluster.

Comparing this output with the original data, we find that error = $\frac{27}{200} * 100 = \mathbf{13.5}\%$

K-means algorithm using Mahalanobis Distance:

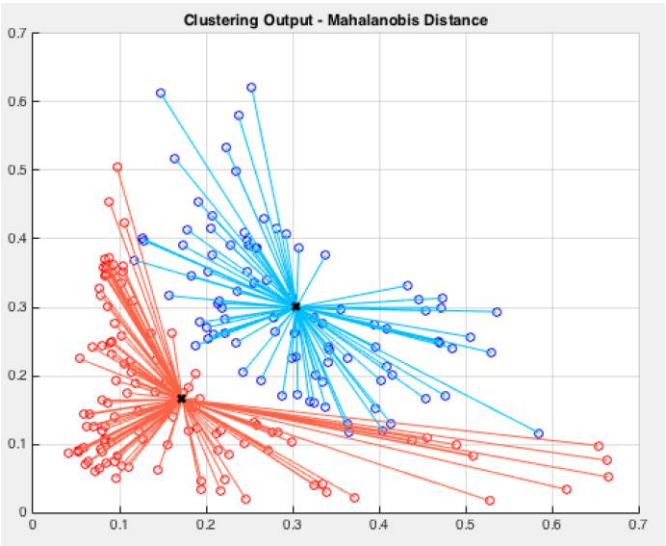


Figure 34: Clusters created using Mahalanobis Distance

The black points are nothing but the centroids of the clusters. Orange represents cluster 1 and Blue represents the second cluster.

Comparing this output with the original data, we find that error = $\frac{17}{200} * 100 = \mathbf{8.5}\%$

We can see that the error is less when the Mahalanobis Distance is used as it is calculated using the data distribution and covariance matrix.

Clustering can be used as the classification method when the group of points are separated in the space.

References

[1] S. Theodoridis and K. Koutroumbas, Pattern Recognition.

[2] “NEURAL NETWORK CLASSIFICATION – FrontLine Solvers”, [Online]. Available: http://www.solver.com/xlminer/help/neural-networks-classification-intro. [Accessed: 22-April-2016]

[3] Stanford University CS229, Andrew Ng, - Lecture Notes on Support Vector Machines

[4] Massachusets Institute of Technology 15.097, Cynthia Rudin - Suppport Vector Machines

[5] “A question regarding conditional probabilities in Bayesian Classifier theory- Math Help Forum.” [Online]. Available:http://mathhelpforum.com/advanced-statistics/181962-question-regarding-conditional-probabilities-bayesian-classifier-theory.html. [Accessed: 10-Mar-2016].

[6] University of Michigan, Cluster Analysis: Basic Concepts and Algorithms

[7] “A Step by Step Backpropogation Example”, Matt Mazur, [Online]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/. [Accessed: 22-April-2016]

[8] “Crab Classification - Mathworks”, [Online]. Available: http://www.mathworks.com/help/nnet/examples/crab-classification.html?s_tid=gn_loc_drop&refresh=true. [Accessed: 22-April-2016]

[9] “SVD implementation - mySVD”, Deng Cai, Available: http://www.cad.zju.edu.cn/home/dengcai/Data/code/mySVD.m. [Accessed: 22-April-2016]

APPENDIX A (MATLAB CODES)

Code for training and testing the Neural Network:

```
clear; close all; clc;

TrainWater ='/Users/Amit/Documents/MATLAB/Data/TrainWater';
images = dir(fullfile (TrainWater, '*.png'));

for i = 1:length(images)
    HSV = imread(fullfile(TrainWater, images(i).name));
    HSV = rgb2hsv(HSV);
    H = HSV(:,:,1);
    S = HSV(:,:,2);

    waterMeanHue(i) = mean(mean(H));
    waterMeanSat(i) = mean(mean(S));
end


TrainNonWater = '/Users/Amit/Documents/MATLAB/Data/TrainNonWater';
images = dir(fullfile (TrainNonWater, '*.png'));

for i = 1:length(images)
    HSV = imread(fullfile(TrainNonWater, images(i).name));
    HSV = rgb2hsv(HSV);
    H = HSV(:,:,1);
    S = HSV(:,:,2);

    NonwaterMeanHue(i) = mean(mean(H));
    NonwaterMeanSat(i) = mean(mean(S));
end

Data = [waterMeanHue NonwaterMeanHue; waterMeanSat NonwaterMeanSat];
target = [ones(1,100) zeros(1,100); zeros(1,100) ones(1,100)];

net = patternnet(10);
% view(net)

[net,tr] = train(net,Data,target);
% nntraintool

% Test ='/Users/Amit/Documents/MATLAB/Data/Test';
% images = dir(fullfile (Test, '*.png'));
%
% for i = 1:length(images)
%     HSV = imread(fullfile(Test, images(i).name));
%     HSV = rgb2hsv(HSV);
%     H = HSV(:,:,1);
%     S = HSV(:,:,2);
%
%     MeanHue(i) = mean(mean(H));
%     MeanSat(i) = mean(mean(S));
% end

% testX = [MeanHue; MeanSat];
% testT = [ones(1,25) zeros(1,5); zeros(1,25) ones(1,5)];

testX = Data(:,tr.testInd);
testT = target(:,tr.testInd);

testY = net(testX);
testIndices = vec2ind(testY);

plotconfusion(testT,testY)
[c,cm] = confusion(testT,testY)

fprintf('Percentage Correct Classification   : %f%%\n', 100*(1-c));
fprintf('Percentage Incorrect Classification : %f%%\n', 100*c);
```

```
Dir = '/Users/Amit/Documents/Pattern Recognition/Pattern/Pattern/Water Detection
Dataset/database'; % Test Directory Path
Images = dir(fullfile(Dir, '*.jpg'));

for im = 1:length(Images)
original_image = imread(fullfile(Dir, Images(im).name)); % read the image
[processed_image] = issky(original_image,1,1,10000,5000); % sky detection
removed_sky = removesky(processed_image); % sky is removed from the image.
figure
subplot(2,1,1)
imshow(removed_sky); title('Sky Removed Image');

[row,col,frames] = size(removed_sky); % Size of the image without sky
rowso = 1:row; % vector of all the rows
colso = 1:col; % vector of all the columns
index = zeros(row,col); % this will keep track of the water presence during recursion
mat = index; % co-ordinates of water pixels

h = rgb2hsv(removed_sky); % rgb to hsv conversion

% Loop for working on images for segmentation
% Sarts from the 16 x 16 Kernel and goes down to 4 X 4 recursively.
for m = 1:16:rowso(length(rowso))
    for n = 1:16:colso(length(colso))
        rows = m:m+16;
        if(rows(16) > rowso(length(rowso)))
            rows = m:rowso(length(rowso));
        end
        cols = n:n+16;
         if(cols(16) > colso(length(colso)))
            cols = m:colso(length(colso));
        end
        mat = segment(h(:,:,1),h(:,:,2),rows,cols,index,net); % preceptron
segmentation function
        for i=1:1:row
            for j = 1:1:col
                if(mat(i,j) == 255)
                    removed_sky(i,j,1) = 0;
                    removed_sky(i,j,2) = 0;
                    removed_sky(i,j,3) = 255; % Marking water pixels blue
                end
            end
        end
    end
end
subplot(2,1,2)
imshow(removed_sky);title('Segmented Water');

end
```

Code for segmentation using Neural Network:

```
Function - segment()

function [ index ] = segment( image1,image2,rows,cols,index,net)

% This fucntion recursively detects the water areas in the image.
% It uses the decision we have found in the perceptron linear classifier
% training.

% Recursive approach considerably reduces the time complexity.

%fprintf('Processing...\n');

if(length(rows)>=4 && length(cols)>=4) % Limitting condition for recursion
huemean = 0;satmean = 0;
row = length(rows); col = length(cols);
hmat = zeros(row,col); smat = zeros(row,col);

for i = rows(1):1:rows(row)
    for j = cols(1):1:cols(col)
```

```
        hmat(i,j) = image1(i,j);
        smat(i,j) = image2(i,j);
    end
end

for i = rows(1):1:rows(row)
    for j = cols(1):1:cols(col)
        huemean = huemean + hmat(i,j);
    end
end
huemean = huemean / (row*col); %Hue Mean in the neighborhood

for i = rows(1):1:rows(row)
    for j = cols(1):1:cols(col)
        satmean = satmean + smat(i,j);
    end
end
satmean = satmean / (row*col); %Saturation Mean in the neighborhood

X = [huemean; satmean];
Y = net(X);
Index = vec2ind(Y);

if (Index == 2) % Recursion
    index =
segment(image1,image2,rows(1:round(row/2)),cols(1:round(col/2)),index,net);
    index =
segment(image1,image2,rows(1:round(row/2)),cols(round(col/2):col),index,net);
    index =
segment(image1,image2,rows(round(row/2):row),cols(1:round(col/2)),index,net);
    index =
segment(image1,image2,rows(round(row/2):row),cols(round(col/2):col),index,net);
else
    for p=rows(1):1:rows(row)
        for q=cols(1):1:cols(col)
            index(p,q) = 255;
        end
    end
end
end
```

Codes for Training and Testing the SVM Classifier:

```
% Support Vector Machines
% This is the main program which includes feature extraction, creation of
% data matrix, labels and calls to various functions such training and
% testing the classifier and finally the water segmentation

tic;
clc;
clear all;
close all;

Dir ='/Users/prathameshprabhudesai/Desktop/Pattern Recognition/Project
2/ann_svm_water_segmentation/Database/TrainWater'; %Training data path water
images = dir(fullfile (Dir, '*.png'));

for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
    I = K(:,:,1);
    L = K(:,:,2);

    meanhue(j) = mean(mean(I));
    meansat(j) = mean(mean(L));
end
l1 = length(images);

Dir ='/Users/prathameshprabhudesai/Desktop/Pattern Recognition/Project
2/ann_svm_water_segmentation/Database/TrainNonWater'; %Training data path non water
images = dir(fullfile (Dir, '*.png'));
```

```
vargray = double(zeros(length(images),1));
for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
    I = K(:,:,1);
    L = K(:,:,2);

    meanhuen(j) = mean(mean(I));
    meansatn(j) = mean(mean(L));
end
l2 = length(images);
X1 = [meanhue' meansat'];
X_1 = [meanhuen' meansatn'];

data = [X1; X_1]; % final data matrix
label = [ones(l1,1); -ones(l2,1)]; % label matrix

[svm_struct, svIndex] = training_svm(data,label); % training svm

plotsvm( data,label,svm_struct ); % plotting the data and separating hyperplane

question = input('Do you want to test the data now? [y=1/n=0]: '); % Testing
if(question == 1)
    Xnew = zeros(1,2);
    Dir = '/Users/prathameshprabhudesai/Desktop/Pattern Recognition/Project
2/ann_svm_water_segmentation/Database/TestWater';
    images = dir(fullfile (Dir, '*.jpg'));
    for j = 1:length(images)
        K = imread(fullfile(Dir, images(j).name));
        K = rgb2hsv(K);
        I = K(:,:,1);
        L = K(:,:,2);

        Xnew(1) = mean(mean(I));
        Xnew(2) = mean(mean(L));
        decision = classify_svm(svm_struct,Xnew); %test function call
        if (decision == -1)
            fprintf('\nWater');
        else
            fprintf('\nNon-water');
        end
    end
else
    disp('OK');
end

% Segmentation of Water
 Dir = '/Users/prathameshprabhudesai/Desktop/Pattern Recognition/Project
2/ann_svm_water_segmentation/Database';
images = dir(fullfile (Dir, '*.jpg'));

for j = 1:1
    disp(fullfile(Dir, images(j).name));
    K = imread(fullfile(Dir, images(j).name));
    water_segmentation_svm(svm_struct,K);
end
toc;

function [svm_struct, svIndex] = training_svm(training, groupnames)

% This function is for training the SVM classifier
% Sequential Minimal Optimization is used as an optimization technique
% Gaussian kernel is used the classes are non-separable
% seqmin and rbf_kernel are the inbuilt function in matlab

[groupIndex, ~] = grp2idx(groupnames);
nPoints = length(groupIndex);
groupIndex = 1 - (2* (groupIndex-1));
boxC = 1;
kfun = @rbf_kernel;
sigma = 0.1;
smo_opts = statset('Display','off','MaxIter',15000);
```

```
tolkkt = 1e-3;
kerCL = 5000;
kktvl = 0;
smo_opts =
svmsmoset(smo_opts,'tolkkt',tolkkt,'KernelCacheLimit',kerCL,'KKTViolationLevel',kktvl)
;
kfunargs = {sigma};

boxconstraint = ones(nPoints, 1);
n1 = length(find(groupIndex==1));
n2 = length(find(groupIndex==-1));
c1 = 0.5 * boxC * nPoints / n1;
c2 = 0.5 * boxC * nPoints / n2;
boxconstraint(groupIndex==1) = c1;
boxconstraint(groupIndex==-1) = c2;

boxconstraint =
min(boxconstraint,repmat(1/sqrt(eps(class(boxconstraint))),size(boxconstraint)));
tmp_kfun = @(x,y) feval(kfun, x,y, kfunargs{:});

[alpha, bias] = seqmin(training, groupIndex,boxconstraint, tmp_kfun, smo_opts);

svIndex = find(alpha > sqrt(eps));
sv = training(svIndex,:);
alphaHat = groupIndex(svIndex).*alpha(svIndex);

% Data Structure for keeping track of the classifier parameters
svm_struct.SupportVectors = sv;
svm_struct.Alpha = alphaHat;
svm_struct.Bias = bias;
svm_struct.KernelFunction = kfun;
svm_struct.KernelFunctionArgs = kfunargs;
svm_struct.GroupNames = groupnames;
svm_struct.SupportVectorIndices = svIndex;
svm_struct.FigureHandles = [];
end

function out = classify_svm(svm_struct,Xnew)

% This fucntion is used in the classification of images as water or
% non-water
% The output is signum as we need the 1 or -1 decision (binary)

sv = svm_struct.SupportVectors;
alphaHat = svm_struct.Alpha;
bias = svm_struct.Bias;
kfun = svm_struct.KernelFunction;
kfunargs = svm_struct.KernelFunctionArgs;

f = (feval(kfun,sv,Xnew,kfunargs{:})'*alphaHat(:)) + bias;
disp(f);
out = sign(f);
% points on the boundary are assigned to class 1
out(out==0) = 1;end


function [  ] = plotsvm( data,label,svm_struct )

% This function is used for plotting the data and separating hyperplane

class1 = 'r+';
class2 = 'g*';

X1 = data(label==-1,:);
h1 = plot(X1(:,1),X1(:,2),class1);
hAxis = get(h1,'parent');
hold on
Xn =  data(label==1,:);
h2 = plot(Xn(:,1),Xn(:,2),class2);
if isempty(hAxis)
    hAxis = get(h2,'parent');
end
```

```
drawnow
hold on

sv = svm_struct.SupportVectors;
plot(sv(:,1),sv(:,2),'ko','linewidth',2);
lims = axis(hAxis);
[X,Y] = meshgrid(linspace(lims(1),lims(2)),linspace(lims(3),lims(4)));
Xorig = X; Yorig = Y;

sv = svm_struct.SupportVectors;
alphaHat = svm_struct.Alpha;
bias = svm_struct.Bias;
kfun = svm_struct.KernelFunction;
kfunargs = svm_struct.KernelFunctionArgs;

f = (feval(kfun,sv,[X(:),Y(:)],kfunargs{:})'*alphaHat(:)) + bias;
contour(Xorig,Yorig,reshape(f,size(X)),[0 0],'--bs');
grid on;

legend('Water','Non-Water','Support Vectors');

end
```

Code for Segmentation using SVM:

```
function [ ] = water_segmentation_svm(svm_struct,image)

% This fucntion deals with the water segmentation
% It keeps track of the water pixels in the images
% All the preporcessing are done in this function

processed_image = issky(image,1,1,10000,5000);
sky_removed = removesky(processed_image);

figure
subplot(2,1,1)
imshow(sky_removed); title('Sky Removed Image');

h = rgb2hsv(sky_removed);
[r,c,~] = size(h);
him = zeros(r+32,c+32);
sim = zeros(r+32,c+32);
indexmatx = [];indexmaty = [];
mat = sim;
for i = 17:1:r+16
    for j = 17:1:c+16
        him(i,j) = h(i-16,j-16,1);
        sim(i,j) = h(i-16,j-16,2);
    end
end
for str = 1:1:r
    fprintf('Processing....\n');
    for stc = 1:1:c
        rows =  str:str+31;
        cols = stc:stc+31;
        [indexmatx,indexmaty] =
pixellhr(him,sim,rows,cols,indexmatx,indexmaty,svm_struct);

        for i = 1:1:length(indexmatx)
            mat(indexmatx(i),indexmaty(i)) = 255;
        end
        indexmatx = [];
        indexmaty = [];
    end

end

for i = 17:1:r+16
    for j = 17:1:c+16
        if (mat(i,j) == 255)
```

```
            sky_removed(i-16,j-16,1) = 0;
            sky_removed(i-16,j-16,2) = 0;
            sky_removed(i-16,j-16,3) = 255;
        end
    end
end

subplot(2,1,2)
imshow(sky_removed); title('Segmented Water');

end


function [ processed_image ] = issky( original_image,newi,newj,pixelth,countth )

% This function detects whether the sky is present or not in the image.
% SKy forms a smooth area which resembles to water in some cases.
% Hence its removal seems unavoidable.

% This is a recursive approach.


processed_image = original_image;
[row,col] = size(original_image(:,:,1));
pixel = 0;
count = 0;
for i=newi:1:row
    for j = newj:1:col
        pixel = pixel + 1;
        if (pixel > pixelth)
            break;
        end
        if(original_image(i,j,3) > 150)
            count  = count + 1;
        end
    end
    if (pixel > pixelth)
        break;
    end
end

if(count > countth)
    [processed_image] = issky(original_image,i,j,pixelth-500,countth-250);
    for p = 1:1:i
        for q = 1:1:col
            processed_image(p,q,:) = 0;
        end
    end
end

end

function [ b] = removesky( a )
[row,col] = size(a(:,:,1));
for i = 1:1:row
    for j = 1:1:col
        if(a(i,j,1) ~= 0 && a(i,j,2) ~= 0 && a(i,j,3) ~= 0)
            break;
        end
    end
    if(a(i,j,1) ~= 0 && a(i,j,2) ~= 0 && a(i,j,3) ~= 0)
            break;
    end
end
start = i;
b(1:row-start+1,1:col,1:3) = a(start:row,1:col,1:3);

end

function out = svm_classify_segmentation(svm_struct,Xnew)

% This function is used during the segmentation since we need the decimal
```

```
% outputs for the output fot our classification
% These outputs are thresholded for taking water non-water decision

sv = svm_struct.SupportVectors;
alphaHat = svm_struct.Alpha;
bias = svm_struct.Bias;
kfun = svm_struct.KernelFunction;
kfunargs = svm_struct.KernelFunctionArgs;

f = (feval(kfun,sv,Xnew,kfunargs{:})'*alphaHat(:)) + bias;
out = f;
end
```

Code for Dimension reduction using PCA:

```
clear all; clc;

load('data.mat');
load('label.mat');

options = [];
options.ReducedDim = 1;
[evec, eval] = myPCA(data, 1);
Y = data*evec;
figure; scatter(data(1:100,1), data(1:100,2), 'filled'); hold on;
scatter(data(101:200,1), data(101:200,2), 'filled'); hold on;

%scatter(Y(1:100,1), Y(1:100,2), 'filled'); hold on;
%scatter(Y(101:200,1), Y(101:200,2), 'filled'); hold on;

a = -1*(1/evec(2,1));

syms x1 x2 y1 y2
f1(x1) = evec(2,1)*x1 + evec(2,2);
f2(x2) = a*x2 + evec(2,2);
ezplot(f1); hold on;
%ezplot(f2); hold on;



for i = 1:100

s = abs([evec(1,1), evec(1,2)]*[data(i,1); data(i,2)]);
SOL = solve([s == sqrt((data(i,1) - y1)^2 + (data(i,2) - y2)^2), y2 == evec(2,1)*y1 +
evec(2,2)],[y1, y2]);
% solve for y1 and y2
newx(i) = vpa(SOL.y1(1)); % save new numerical values of x
newy(i) = vpa(SOL.y2(1)); % and y
end
scatter(newx(:), newy(:), 'filled'); hold on;


for i = 101:200

s = abs([evec(1,1), evec(1,2)]*[data(i,1); data(i,2)]);
SOL = solve([s == sqrt((data(i,1) - y1)^2 + (data(i,2) - y2)^2), y2 == evec(2,1)*y1 +
evec(2,2)],[y1, y2]);
% solve for y1 and y2
newwx(i) = vpa(SOL.y1(1)); % save new numerical values of x
newwy(i) = vpa(SOL.y2(1)); % and y
end
scatter(newwx(:), newwy(:), 'filled');

title('data projected onto the minor eigen vector');
```

Function for creating 1D Gaussian distributions (PCA):

```
clear all; clc;

load('data.mat');

[residuals,reconstructed] = pcares(data,1);

m1 = mean(reconstructed(1:100,1));
v1 = var(reconstructed(1:100,1));

X1 = -1:0.01:1;

f1 = normpdf(X1,m1,v1);
figure; plot(X1,f1);
hold on;

m2 = mean(reconstructed(101:200, 1));
v2 = var(reconstructed(101:200, 1));

X2 = -1:0.01:1;
f2 = normpdf(X2, m2, v2);
%figure;
plot(X1,f2);

title('Data for the classes modeled as Gaussian distribution');
legend('Non-Water Class','Water Class' );

csvwrite('pca_waterdist.csv', f2);
csvwrite('pca_nonwaterdist.csv', f1);
```

Function for testing using PCA:

```
function [] = water_segmentation_pca(image)
I = imread(image);
tic;
processed_image = issky(I,1,1,10000,5000);
sky_removed = removesky(processed_image);

figure;
subplot(2,1,1)
imshow(sky_removed); title('Sky Removed Image');

h = rgb2hsv(sky_removed);
[r,c,~] = size(h);
him = zeros(r+32,c+32);
sim = zeros(r+32,c+32);
indexmatx = [];indexmaty = [];
mat = sim;
for i = 17:1:r+16
    for j = 17:1:c+16
        him(i,j) = h(i-16,j-16,1);
        sim(i,j) = h(i-16,j-16,2);
    end
end
for str = 1:1:r
    fprintf('Processing....\n');
    for stc = 1:1:c
        rows =  str:str+31;
        cols = stc:stc+31;
        [indexmatx,indexmaty] = pca_pixellhr(him,sim,rows,cols,indexmatx,indexmaty);

        for i = 1:1:length(indexmatx)
            mat(indexmatx(i),indexmaty(i)) = 255;
        end
        indexmatx = [];
        indexmaty = [];
    end

end

for i = 17:1:r+16
```

```
    for j = 17:1:c+16
        if (mat(i,j) == 255)
            sky_removed(i-16,j-16,1) = 0;
            sky_removed(i-16,j-16,2) = 0;
            sky_removed(i-16,j-16,3) = 255;
        end
    end
end

%subplot(2,1,2)
figure;imshow(sky_removed); title('Segmented Water');
toc;
end
```

Code for Segmentation using PCA:

```
function [] = water_segmentation_pca(image)
I = imread(image);
tic;
processed_image = issky(I,1,1,10000,5000);
sky_removed = removesky(processed_image);

figure;
subplot(2,1,1)
imshow(sky_removed); title('Sky Removed Image');

h = rgb2hsv(sky_removed);
[r,c,~] = size(h);
him = zeros(r+32,c+32);
sim = zeros(r+32,c+32);
indexmatx = [];indexmaty = [];
mat = sim;
for i = 17:1:r+16
    for j = 17:1:c+16
        him(i,j) = h(i-16,j-16,1);
        sim(i,j) = h(i-16,j-16,2);
    end
end
for str = 1:1:r
    fprintf('Processing....\n');
    for stc = 1:1:c
        rows =  str:str+31;
        cols = stc:stc+31;
        [indexmatx,indexmaty] = pca_pixellhr(him,sim,rows,cols,indexmatx,indexmaty);

        for i = 1:1:length(indexmatx)
            mat(indexmatx(i),indexmaty(i)) = 255;
        end
        indexmatx = [];
        indexmaty = [];
    end

end

for i = 17:1:r+16
    for j = 17:1:c+16
        if (mat(i,j) == 255)
            sky_removed(i-16,j-16,1) = 0;
            sky_removed(i-16,j-16,2) = 0;
            sky_removed(i-16,j-16,3) = 255;
        end
    end
end

%subplot(2,1,2)
figure;imshow(sky_removed); title('Segmented Water');
toc;
end
```

Function for Classification using PCA (pca_classify):

```
function[l12] = pca_classify(X)
    %X = [hue_mean sat_mean];
    f1 = csvread('pca_waterdist.csv');
f2 = csvread('pca_nonwaterdist.csv');
    [residual, reconstructed] = pcares(X,1);

    red_dim = reconstructed(1,1);

    red_dim = red_dim*100;
    red_dim = floor(red_dim);
    red_dim = red_dim/100;

if red_dim>0
    red_dim = (red_dim/0.01) +101;
else
    red_dim = (-1*red_dim/0.01) +1;
end

%disp(red_dim);

p1 = f1(red_dim);
p2 = f2(red_dim);

l12 = p2/p1;

end
```

Code for Clustering:

```
% K means Clustering

tic;
clc;
clear all;
close all;

Dir ='/Users/prathameshprabhudesai/Desktop/Pattern Recognition/Project
2/ann_svm_water_segmentation/Database/TrainWater'; %Training data path water
images = dir(fullfile (Dir, '*.png'));

for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
    I = K(:,:,1);
    L = K(:,:,2);

    meanhue(j) = mean(mean(I));
    meansat(j) = mean(mean(L));
end
l1 = length(images);

Dir ='/Users/prathameshprabhudesai/Desktop/Pattern Recognition/Project
2/ann_svm_water_segmentation/Database/TrainNonWater'; %Training data path non water
images = dir(fullfile (Dir, '*.png'));
vargray = double(zeros(length(images),1));
for j = 1:length(images)
    K = imread(fullfile(Dir, images(j).name));
    K = rgb2hsv(K);
    I = K(:,:,1);
    L = K(:,:,2);

    meanhuen(j) = mean(mean(I));
    meansatn(j) = mean(mean(L));
end
l2 = length(images);
X1 = [meanhue' meansat'];
X_1 = [meanhuen' meansatn'];
```

```
data = [X1; X_1];
sort_X = sort(data(:,1));
sort_Y = sort(data(:,2));

mean1 = [sort_X(50) sort_Y(50)];
mean2 = [sort_X(150) sort_Y(150)];

kdata = [data, ones(200,1)];

S = data'*data;

%[mean1,mean2,kdata] = kmeans(mean1,mean2,kdata); %euclidean distance

[mean1,mean2,kdata] = kmeans_maha(mean1,mean2,kdata,S); %Mahalanobis Distance

plot_class = ['r','b'];
plot_line_2 = [0 0.7490 1.0000];
plot_line_1 = [1.0000,0.3882,0.2784];
figure

for i=1:1:200
    scatter(data(i,1),data(i,2),plot_class(kdata(i,3)));
    hold on;
end

grid on;

for i=1:1:200
    if(kdata(i,3) == 1)
        x = [data(i,1),mean1(1)];
        y = [data(i,2),mean1(2)];
        l = line(x,y);
        set(l,'Color',plot_line_1);
    elseif(kdata(i,3) == 2)
        x = [data(i,1),mean2(1)];
        y = [data(i,2),mean2(2)];
        l = line(x,y);
        set(l,'Color',plot_line_2);
    end
end
hold on;
scatter(mean1(1),mean1(2),'kx','linewidth',3);
hold on;
scatter(mean2(1),mean2(2),'kx','linewidth',3);
title('Clustering Output - Mahalanobis Distance');


label = [ones(100,1);2*ones(100,1)];
wrong  = 0;
for i = 1:1:200
    if(kdata(i,3) ~= label(i,1))
        wrong = wrong + 1;
    end
end
```

Code for K-means clustering using Euclidean Distance:

```
function [ mean1,mean2,kdata ] = kmeans( mean1,mean2,kdata )

% This function is used for calculating the centroids of the clusters
% recursively
% Euclidean Distance is used for the calculation and deciding the nearest
% cetroid of the cluster

[data_length,~] = size(kdata);
distance_label_mat = zeros(data_length,3);

for i=1:1:data_length
    distance_label_mat(i,1) = ((kdata(i,1) - mean1(1))^2 + (kdata(i,2) - mean1(2))^2
)^0.5;
    distance_label_mat(i,2) = ((kdata(i,1) - mean2(1))^2 + (kdata(i,2) - mean2(2))^2
```

```
)^0.5;
    if(distance_label_mat(i,1) <= distance_label_mat(i,2))
        distance_label_mat(i,3) = 1;
    else
        distance_label_mat(i,3) = 2;
    end
end
mean1sum_x = 0;
mean1sum_y = 0;
mean2sum_x = 0;
mean2sum_y = 0;
count1 = 0;
count2 = 0;
for i=1:1:data_length
    if(distance_label_mat(i,3) == 1)
        mean1sum_x = mean1sum_x + kdata(i,1);
        mean1sum_y = mean1sum_y + kdata(i,2);
        count1 = count1 + 1;
    elseif(distance_label_mat(i,3) == 2)
        mean2sum_x = mean2sum_x + kdata(i,1);
        mean2sum_y = mean2sum_y + kdata(i,2);
        count2 = count2 + 1;
    end
end

newmean1 = [mean1sum_x mean1sum_y]/count1;
newmean2 = [mean2sum_x mean2sum_y]/count2;

if((abs(mean1(1) - newmean1(1)) > 0.1) || (abs(mean1(2) - newmean1(2)) > 0.1))
    [mean1,mean2,kdata] = kmeans(newmean1,newmean2,kdata);
else
    mean1 = newmean1;
    mean2 = newmean2;
    for i=1:1:data_length
        kdata(i,3) = distance_label_mat(i,3);
    end
end
end
```

Code for K-means clustering using Mahalanobis Distance:

```
function [ mean1,mean2,kdata ] = kmeans_maha( mean1,mean2,kdata,S )

% This function is used for calculating the centroids of the clusters
% recursively
% Mahalanobis Distance is used for the calculation and deciding the nearest
% cetroid of the cluster

[data_length,~] = size(kdata);
distance_label_mat = zeros(data_length,3);

for i=1:1:data_length
    distance_label_mat(i,1) = ((kdata(i,1) - mean1)*S^-1*(kdata(i,2) - mean1)')^0.5;
    distance_label_mat(i,2) = ((kdata(i,1) - mean2)*S^-1*(kdata(i,2) - mean2)')^0.5;
    if(distance_label_mat(i,1) <= distance_label_mat(i,2))
        distance_label_mat(i,3) = 1;
    else
        distance_label_mat(i,3) = 2;
    end
end
mean1sum_x = 0;
mean1sum_y = 0;
mean2sum_x = 0;
mean2sum_y = 0;
count1 = 0;
count2 = 0;
for i=1:1:data_length
    if(distance_label_mat(i,3) == 1)
        mean1sum_x = mean1sum_x + kdata(i,1);
        mean1sum_y = mean1sum_y + kdata(i,2);
        count1 = count1 + 1;
    elseif(distance_label_mat(i,3) == 2)
```

```
        mean2sum_x = mean2sum_x + kdata(i,1);
        mean2sum_y = mean2sum_y + kdata(i,2);
        count2 = count2 + 1;
    end
end

newmean1 = [mean1sum_x mean1sum_y]/count1;
newmean2 = [mean2sum_x mean2sum_y]/count2;

if((abs(mean1(1) - newmean1(1)) > 0.1) || (abs(mean1(2) - newmean1(2)) > 0.1))
    [mean1,mean2,kdata] = kmeans_maha(newmean1,newmean2,kdata,S);
    disp('Gone Recursion');
else
    mean1 = newmean1;
    mean2 = newmean2;
    for i=1:1:data_length
        kdata(i,3) = distance_label_mat(i,3);
    end
end
end
```