# ECE 759 Project 2

Shamim Samadi, Chris Jenkins

April 2016

## 1 Introduction

In this second project, we build upon the analysis already completed in Project 1. Working with the same data set, which is composed of a set of RGB images of natural scenes with some depicting flooded areas. Our goal here is to develop algorithms capable of detecting the presence or absence of water in the images. In Project 1, the main focus was on Bayesian classifiers, where we performed our classification using a QDA (Quadratic Discriminant Analysis) classifier which is a general purpose Bayesian classifier supporting Gaussian distributions. We reported an error of 14% on the training and 23% on the testing data sets. Now we seek to improve our classification results through 1)exploitation of different classification approaches, and 2)extraction of more effective features from the dataset, and further use efficient assessment tools to analyze our results.
This report has been organized as follows: section 2 provides some background on dimension reduction techniques. The rest of the report extensively studies two major types of learning on the dataset: supervised learning (classification) in section 3 and unsupervised learning (clustering) in section 4. The final section will be conclusion and future work.

## 2 Background

### 2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a dimension-reduction technique used in machine learning. This technique reduces the dimension by making a linear transformation to a new orthogonal basis. The change of basis is chosen such that the coordinate projections order the variance of the data. Meaning that projecting onto the first $n$ coordinate axis provides the $n$ dimensional representation of the data with the highest variance. This property is ideal for dimension-reduction. Consider a set of $m$-dimensional centered signals, PCA provides a change of basis to best represent those signals in an $n$ dimensional space with $n \leq m$.

## 2.2 Isomap

Isometric maping (Isomap) is a nonlinear technique for dimension reduction in data. In isomap, data points are assumed as vertices in a graph. The algorithm then performs dimension reduction by manifold learning through seeking a lower-dimensional embedding that preserves geodesic distance between points. Geodesic distance is the number of edges in the shortest path between two nodes in a graph. The algorithm can be summarized in three steps:

1. Find the k nearest neighbors of each data point in the high-dimensional data space we wish to reduce (k is a user-defined parameter).
2. Compute the geodesic pairwise distances between all points.
3. Create an embedding of data via MDS [1] to preserve distances between points.

Isomap will be effective as a means of reducing the dimensions in case of presence of non-linearity in data.

## 2.3 LLE: Locally-linear Embedding

Locally linear embedding (LLE) [2], like isomap, is a nonlinear dimension reduction technique that assumes that each data point and its neighbors lie on or close to a *locally linear* patch of the manifold. Then it stitches together small linear neighborhoods by finding a set of weights that perform local linear interpolations. The LLE algorithm is summarized as follows: (1) Select neighbors of each data point $x_i$, $N(i)$ (e.g. $k$-nearest neighbors or $\epsilon-$neighbors). (2) Reconstruct each point $x_i$ as a weighted, convex combination of its neighbors. Find weights $W_{ij}$ to minimize the cost $\epsilon(W) = \sum_{i=1}^{m} ||x_i - \sum_{j=1}^{m} W_{ij} x_j||$ subject to (1) $W_{ij} = 0$ if $x_j$ is not in $N(x_i)$, and $\sum_j W_{ij} = 1$. (3) Map to embedded coordinates: The weights $W_{ij}$ reflect intrinsic geometric properties of the data that are invariant to rotations, scaling, and translations, and are used to reconstruct its embedded manifold coordinatesin lower dimensions, by choosing coordinates $y_i$ that minimize the cost: $\Phi(y) = \sum_{i=1}^{m} ||y_i - \sum_{j=1}^{m} w_{ij} y_j||$, which be solved by $d+1$ bottom eigenvectors of the sparse matrix $M = (I-W)^T(I-W)$.

The advantage of nonlinear dimension reduction techniques over linear methods is that they can possibly extract nonlinear intrinsic structure existing in complex and high dimensional data sets. Such structures, cannot always be presented as a linear parametric form and cannot be extracted efficiently by linear projections. From another point of view, since the PCA tries to project the data along uncorrelated components, it might perform better in Bayesian framework.

# 3 Supervised Learning

## 3.1 General Framework

### 3.1.1 Feature Extraction

In order to further improve the classification accuracy we got for the existing imagery dataset in project 1, we came up with a more relevant set of features for our particular application of detecting water from non-water. Based on our results in part 1 of the project, *sky* regions and *green* regions (grass, trees, etc) were identified as problematic regions in the classification process. To learn a new set of features, we manually extracted patches in images which had sky or green regions in them, and labelled them differently. All the remaining sub-images (water patches and non-water patches) were then used to derive the set of features for our problem. Smaller patches of size $28 \times 28$ with steps of size 8 across vertical and horizontal axes were extracted from these sub-images leading to a total of $28 \times 28 \times 3 = 2352$ features.

### 3.1.2 Dimension Reduction

To reduce dimensions, PCA has been used to extract $k$ number of principal components from the feature space. This stage reduces dimensions from 2352 to $k$, which makes the problem computationally much easier to tackle.

Thus the pre-processing steps we've taken can be summarized as:

1. Manually extract patches with the following labels from images: "water", "non-water", "sky" and "vegetation"
2. Pick "water" and "non-water" patches
3. Transform to HSV Color Space
4. Dimension Reduction

Given reduced features, different classification approaches can now be applied to the dataset. Next, we need to split data for the purpose of training and testing. In the following parts of this section, results for each approach (Bayes, neural networks and SVM) have been presented, as well as comparisons between different techniques and corresponding results.

### 3.1.3 Segmentation Strategy

The next objective is to segment the image into water and non-water regions. To segment the image we estimate the likelihood that each pixel belongs to the water class. Our likelihood estimate is the ratio of the number of times the pixel was used in a water classification to the number of image patches containing that pixel. The resulting counts for each pixel are then normalized and assigned

to the appropriate class based on a threshold level allowing the image to be segmented into "water" and "non-water" regions.

## 3.2 Bayesian Quadratic Discriminant Analysis

### 3.2.1 Bayesian Classifier

The classifier that we have used is based on quadratic discriminate analysis (QDA). Assuming preprocessed feature vector $X$, QDA performs based upon the Bayes' Rule and decides on class membership based on maximization of the conditional probabilities:

$$P(Y = k|X) = \frac{P(X|Y = k)P(Y = k)}{P(X)} \tag{1}$$

where $P(Y = k)$ is the prior for a particular class.

In this particular application, we are trying to distinguish between "water" and "non-water" classes for every patch considered in an input image of interest, and thus deal with two conditional probabilities, i.e:

- If: $P(Y = \omega_1|X) > P(Y = \omega_2|X)$, classify as class 1.
- If: $P(Y = \omega_2|X) > P(Y = \omega_1|X)$, classify as class 2.

Using Bayes, we get:

- If: $P(X|\omega_1)P(\omega_1) > P(X|\omega_2)P(\omega_2)$, classify as class 1.
- If: $P(X|\omega_2)P(\omega_2) > P(X|\omega_1)P(\omega_1)$, classify as class 2.

In QDA, we assume Gaussian multivariate distributions for classes:

$$P(X|Y = k) = \frac{1}{(2\pi)^n|\Sigma_k|^{1/2}} e^{\frac{-1}{2}((X-\mu_k)^T{\Sigma_k}^{-1}(X-\mu_k))} \tag{2}$$

where $n$ is the dimensions, and $\Sigma_k$ and $\mu_k$ are class covariance matrix and empirical mean, respectively, for class $k$.

The classifier then classifies $X$ as $\omega_1$ "water" class if:

$$(2\pi)^{-n}|\Sigma_1|^{-1/2} e^{\frac{-1}{2}((X-\mu_1)^T{\Sigma_1}^{-1}(X-\mu_1))} P(\omega_1) > (2\pi)^{-n}|\Sigma_2|^{-1/2} e^{\frac{-1}{2}((X-\mu_2)^T{\Sigma_2}^{-1}(X-\mu_2))} P(\omega_2). \tag{3}$$

which allows us to write the likelihood ratio test as:

$$\frac{|\Sigma_2|^{-1/2} e^{\frac{-1}{2}((X-\mu_2)^T{\Sigma_2}^{-1}(X-\mu_2))}}{|\Sigma_1|^{-1/2} e^{\frac{-1}{2}((X-\mu_1)^T{\Sigma_1}^{-1}(X-\mu_1))}} < \frac{P(\omega_1)}{P(\omega_2)} = \eta \tag{4}$$

Taking the log of the two sides results in the classifier simplified equation:

$$\frac{-1}{2}X^T{\Sigma_2}^{-1}X+\frac{1}{2}X^T{\Sigma_2}^{-1}\mu_2-\frac{1}{2}{\mu_2}^T{\Sigma_2}^{-1}\mu_2+\frac{1}{2}{\mu_2}^T{\Sigma_2}^{-1}X+ln(P(\omega_2))-\frac{1}{2}ln(|\Sigma_2|)$$
$$+\frac{1}{2}X^T{\Sigma_1}^{-1}X-\frac{1}{2}X^T{\Sigma_1}^{-1}\mu_1+\frac{1}{2}{\mu_1}^T{\Sigma_1}^{-1}\mu_1-\frac{1}{2}{\mu_1}^T{\Sigma_1}^{-1}X-ln(P(\omega_1))+\frac{1}{2}ln(|\Sigma_1|)$$
$$< 0 \quad (5)$$

for the feature to be classified as "water" $\omega_1$.

Since the set of features that we're working with differ from those in project 1, our class priors have changed. Assuming $\omega_1$ to be "water" class and $\omega_2$ to be "non-water" class, we estimated appropriate priors from the training and test data (by the proportion of instances of occurrence of each class in the total number of extracted patches) as:

Train: $P(\omega_1) = 0.0.495852$, $P(\omega_2) = 0.504148$
Test: $P(\omega_1) = 0.487144$, $P(\omega_2) = 0.512856$

It can be inferred from the calculated priors that, with the new set of derived features, the two classes are roughly equiprobable.

### 3.2.2 Classification Results

We split the data into training and testing samples, leaving us with a total of 16753 training and 8372 testing samples. Classification performance summary for Bayesian approach has been shown in table 1. Looking at the results, one can observe that in case of no dimension reduction, Bayesian classifier resulted in overfitting. After reducing the dimension using PCA, overfitting issue was resolved and classifier improved in performance. We ran the classification with different values of $k$ (the resulting number of dimension after PCA). Values $k = 16, 32$ show consistently good results on both the training and testing samples.

Table 1: Bayesian Classifier Performance

| Dimension Reduction | Training Error (%) | Test Error (%) |
|---|---|---|
| — | 8.30 | 17.31 |
| PCA, $k = 8$ | 13.27 | 15.24 |
| PCA, $k = 16$ | 12.24 | 14.27 |
| PCA, $k = 32$ | 12.69 | 14.43 |
| PCA, $k = 64$ | 12.42 | 17.24 |
| PCA, $k = 128$ | 11.96 | 17.22 |

### 3.2.3 Segmentation Results Using Bayesian Classifier

In segmenting our images following the guideline described in section 3.1.3, we have used PCA-reduced set of features. Figure 1 shows the results for segmenting using Bayesian approach. Results for different values of $k$ used for PCA look very similar, but slightly different. (see figure 2)

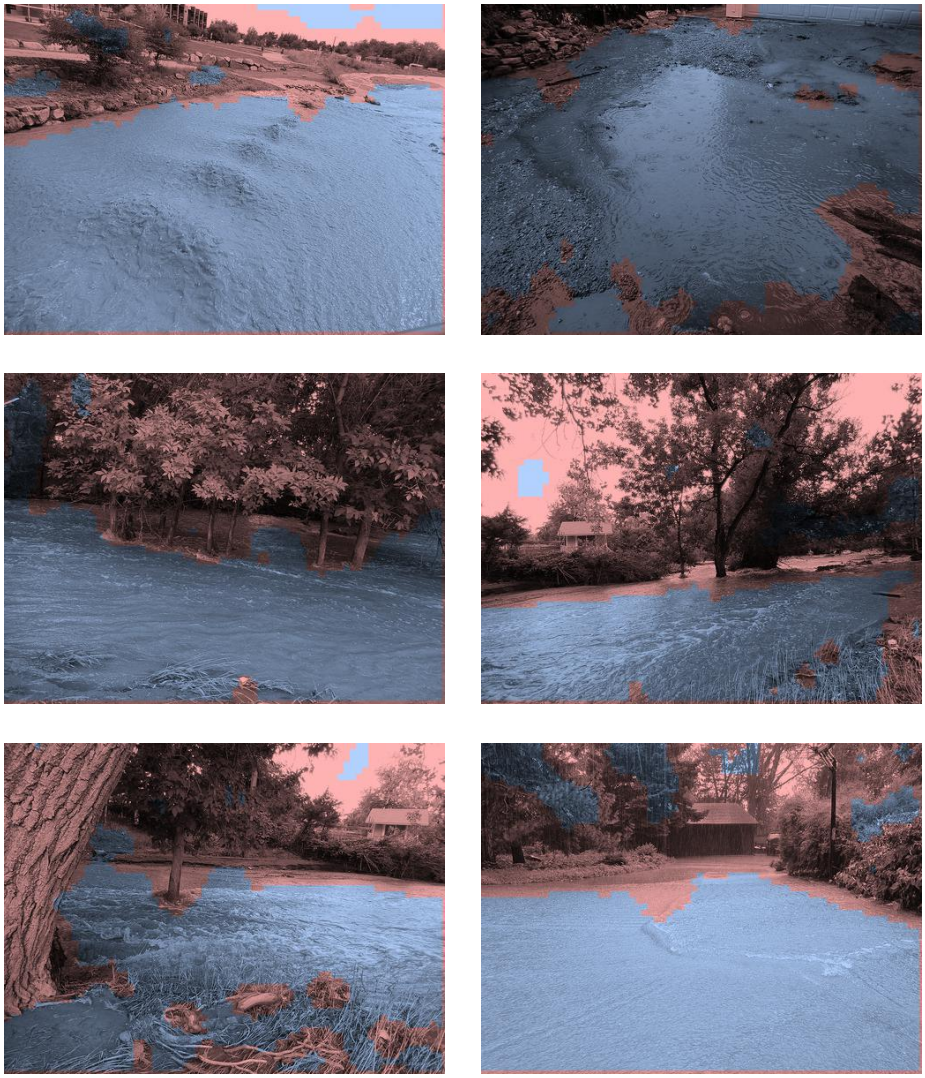
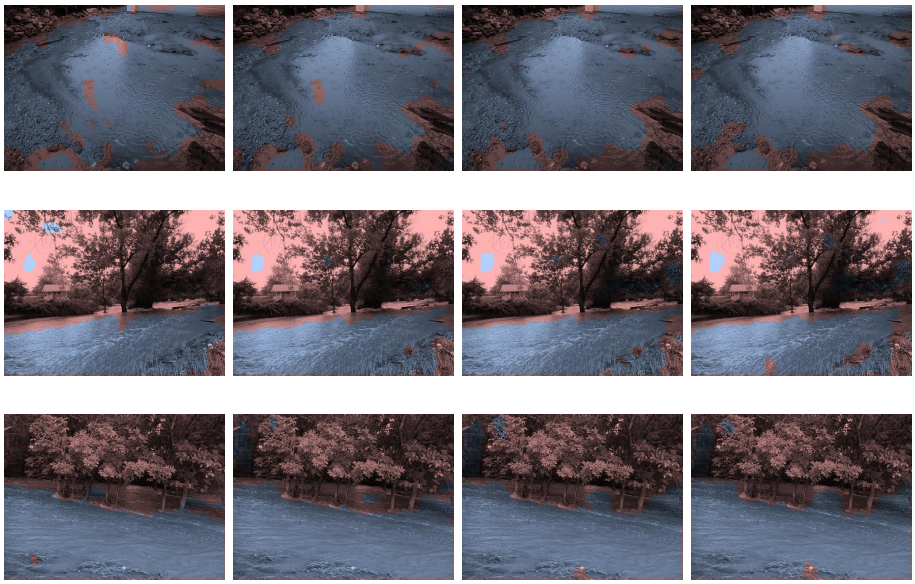


Figure 1: Bayesian classifier - segmentation results, The blue tinted regions are predicted to belong to the water class while the red tinted regions, the non-water class.



(a) $k$=8 (b) $k$=16 (c) $k$=32 (d) $k$=64

Figure 2: Bayesian classifier - segmentation results for different values of $k$ (PCA destination dimension)

## 3.3 Neural Networks

Neural networks (NN's) are a common and potentially effective tool when applied to classification tasks. In this section, we discuss the application of neural networks to the classification of the "water" vs. "non-water" images.

### 3.3.1 Architecture

A neural network in general realizes a non-linear transformation of the feature space by applying non-linear activation functions to the outputs of the hidden layers. The major challenge in implementing neural networks for a classification task is with the difficulty of learning the connection weights between the neurons of the hidden layers through the back propagation algorithm.

Table 2: Neural Network Architectures

| Architecture | Dimension Reduction | Input Layer | Hidden Layer 1 | Hidden Layer 2 | Output Layer |
|---|---|---|---|---|---|
| 1 | — | 2352 | 100 | 100 | 2 |
| 2 | PCA, $k = 32$ | 32 | 100 | 100 | 2 |
| 3 | PCA, $k = 32$ | 32 | 100 | — | 2 |
| 4 | PCA, $k = 32$ | 32 | 50 | — | 2 |

A neural network is compromised of three layers of neurons: input layer,

hidden layer(s) and output layer. When performing classification, the number of neurons in the input layer is equal to the number of features present in data set and the number of output neurons will typically be equal to the number of classes (depending on the structure). In our implementation, we have used $Softmax$ activation function at the final layer of the network, and thus the output layer of our network has 2 neurons (since we have two classes). The $Softmax$ function returns probabilities of a set of input features belonging to a particular class, and then classifies data by picking the class with the greater probability. The art of choosing 1) the appropriate number of hidden layers, and 2) the appropriate number of neurons in each hidden layer is highly data dependent and plays a determining role in how well the classifier performs.

Table 2 shows the different architectures that have been implemented and trained on our data set.

### 3.3.2 Classification Using Neural Networks

Table 3 shows the classification errors corresponding to the various neural network architectures that were observed in our experiments. The mean squared error (MSE) which was produced by the back propagation training algorithm over a total of 20 epochs for each network has been plotted in figure 3. For additional insight, we have also plotted the weights learned by the neural network in the first hidden layer (see figure 4).

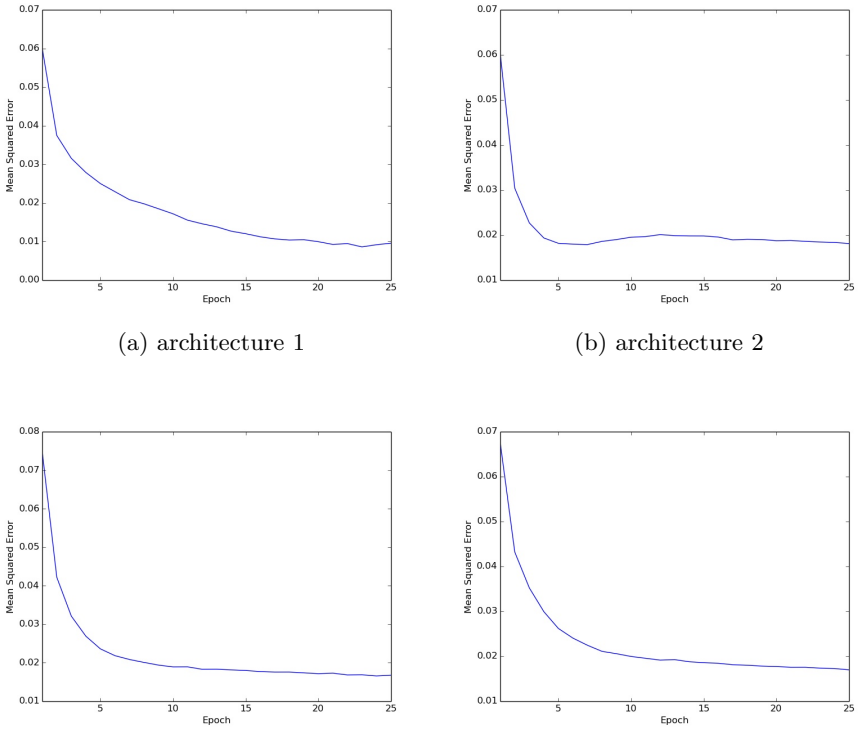
Based on table 3, architecture 3 showed the best performance out of the 4 trained networks, i.e. the network using PCA with 32 principal components, an input layer of size 32, a single hidden layer of size 100 and an output layer.

Table 3: Neural Network Classifier Performance

| Architecture | Training Error (%) | Test Error (%) |
|---|---|---|
| 1 | 5.21 | 12.61 |
| 2 | 4.13 | 9.27 |
| 3 | 3.81 | 8.90 |
| 4 | 4.11 | 9.39 |

### 3.3.3 Segmentation Results Using Neural Networks

Architecture 3 showed the best classification performance and appears to have also achieved superior segmentation performance. We have provided a representative set of segmented images for all four architectures in figure 5. Both architectures with two hidden layers appear to possibly suffer from over fitting as their segmentation performance does not compare well to that of architecture 3 which contains a single hidden layer. The results for architecture 4 again do not compare favorably to architecture 3. Since the only difference is that architecture 4 contains a hidden layer of half the size of architecture 3 and thus we assume that the neural network does not have sufficient parameters to fully learn




(a) architecture 1

(b) architecture 2

(c) architecture 3

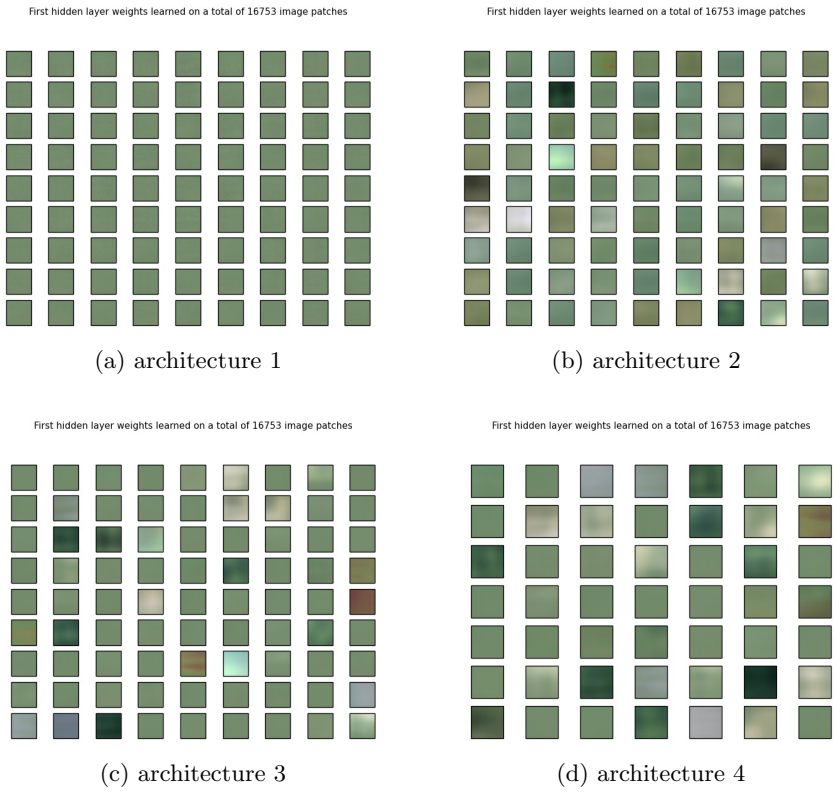
(d) architecture 4

Figure 3: Training Error over epochs

the features of our data set. Figure 6 shows a representative set of segmented images generated by the architecture 3 neural network.

### 3.3.4 Comparison of Neural Networks & Bayesian Classifier

Comparing the error statistics presented in tables 1 & 3 along with a comparison of segmented images for the Bayesian and neural network classifiers presented in figure 7, we can safely conclude that with the current data set, the optimal neural network classifier performs better than the Bayes classifier. One reason behind the superior performance of the neural network architecture is possibly due to the non-linear transformation applied to the input features set. This would potentially render a more separable set of features presented to the classification layer of of the neural network.




(a) architecture 1 (b) architecture 2

(c) architecture 3 (d) architecture 4

Figure 4: Weights of the Neural Network, first hidden layer.

## 3.4 Support Vector Machines

### 3.4.1 Classification Using SVM

We have used SVM with radial basis function kernel as our classifier. Data was split into 10890 training and 5532 testing samples. Performance summary for classification of raw features as well as reduced features with different values of $k$ is presented in table 4.

Table 4: SVM Classifier Performance

| Dimension Reduction | Training Error (%) | Test Error (%) |
|---|---|---|
| — | 3.07 | 10.18 |
| PCA, $k = 16$ | 2.52 | 6.74 |
| PCA, $k = 32$ | 2.26 | 5.98 |
| PCA, $k = 64$ | 2.08 | 7.70 |

(a) architecture 1

(b) architecture 2

(c) architecture 3

(d) architecture 4

Figure 5: Segmentation results using various neural network architectures

#### 3.4.2 Segmentation Results Using SVM

Segmentation results for SVM approach have been shown in figure 8. Segmented images look pretty good and accurate.

#### 3.4.3 SVM Comparison with Neural Networks & Bayesian Classifier

Comparing the segmentation performance of Bayes and SVM, by looking at segmented images from the two classifiers, SVM results stand out. Figure 9 shows a visual comparison between the results, focusing on scenarios where Bayes failed to perform a consistently good segmentation and SVM clearly outperformed Bayes.

### 3.5 Feature Separability Measures

This section analyzes overall performance of classifiers through feature separability measures. Picking a good set of features for the particular problem one deals with is a key factor in classification results. A good set of features are features that are dense within the class they represent, and are wide apart between different classes. To determine how efficient extracted features for each task are, we have used the scatter matrix analysis to derive the class separability measures, as suggested by section 5.6.3 of the textbook.[3] These measures are defined as:

$$J_1 = \frac{trace(S_m)}{trace(S_w)} \tag{6}$$

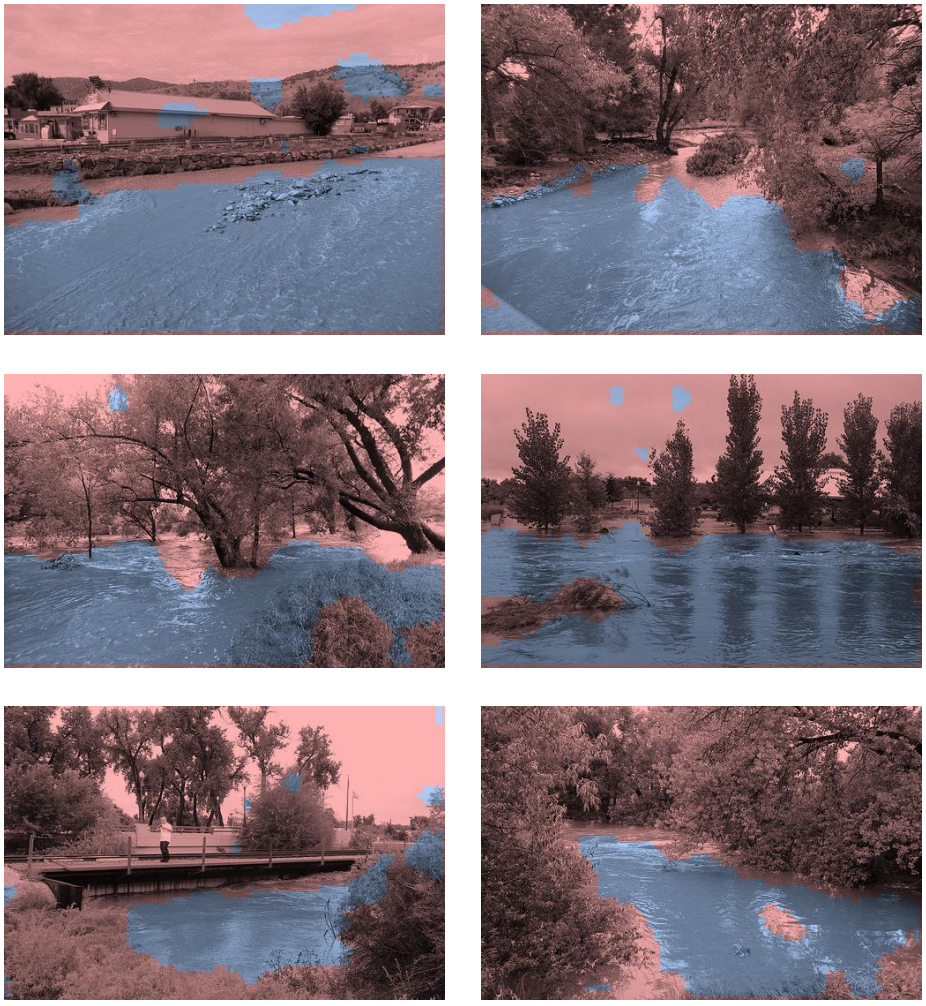


Figure 6: Neural Network classifier - segmentation results. The blue tinted regions are predicted to belong to the water class while the red tinted regions, the non-water class.

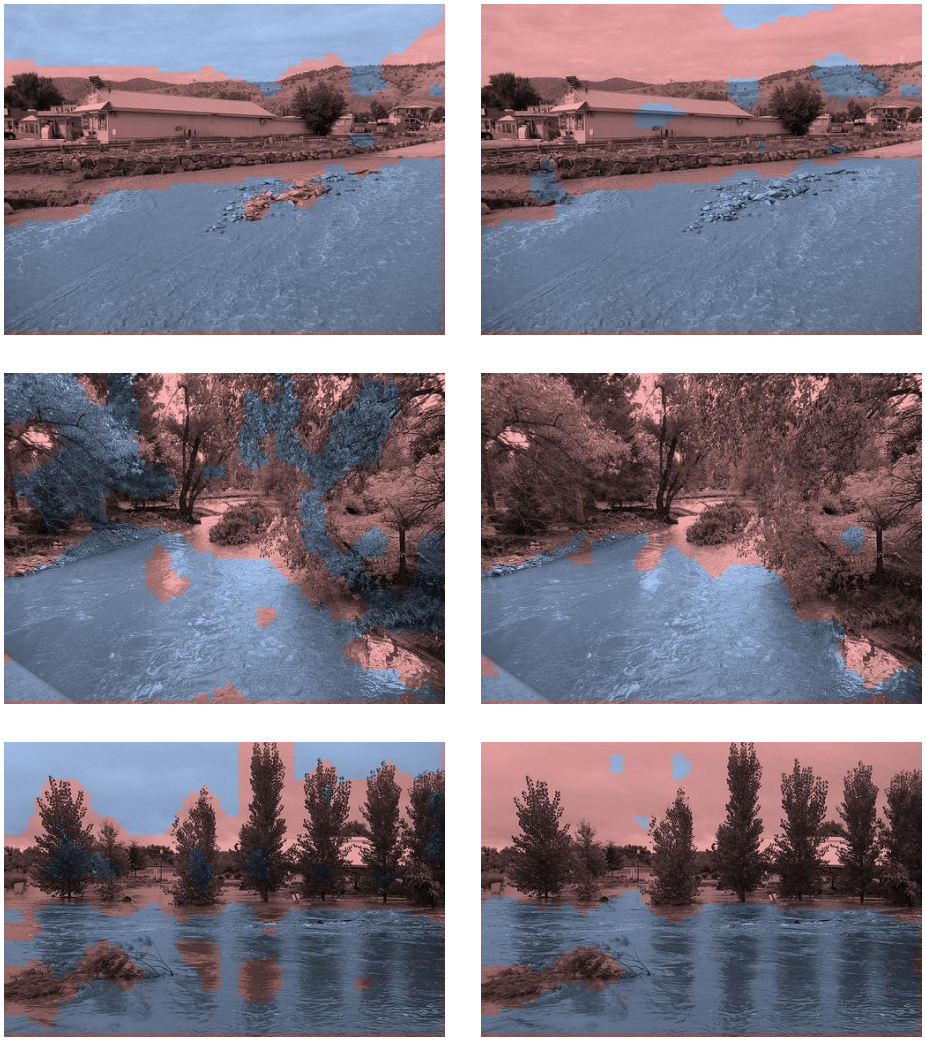


Figure 7: Comparison of segmentation results for Bayesian classifier (left) and neural network (right).

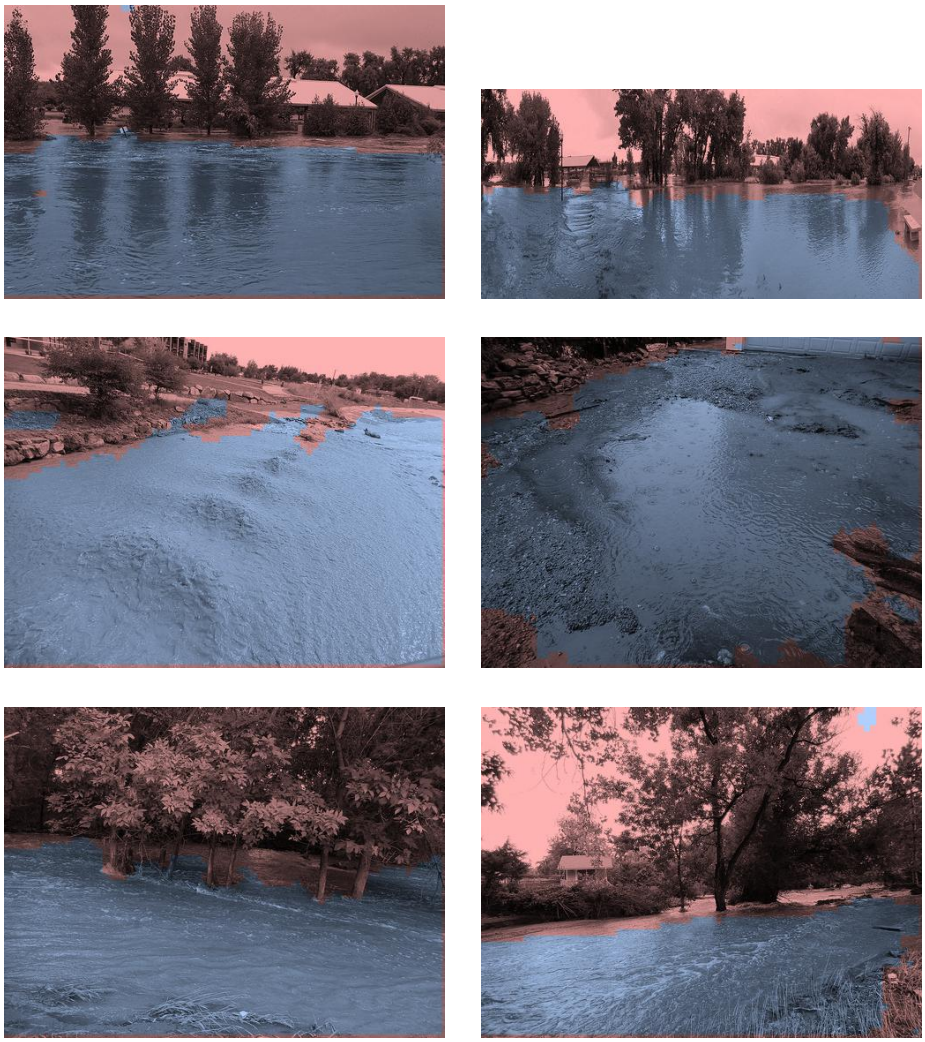


Figure 8: SVM classifier - segmentation results, The blue tinted regions are predicted to belong to the water class while the red tinted regions, the non-water class.

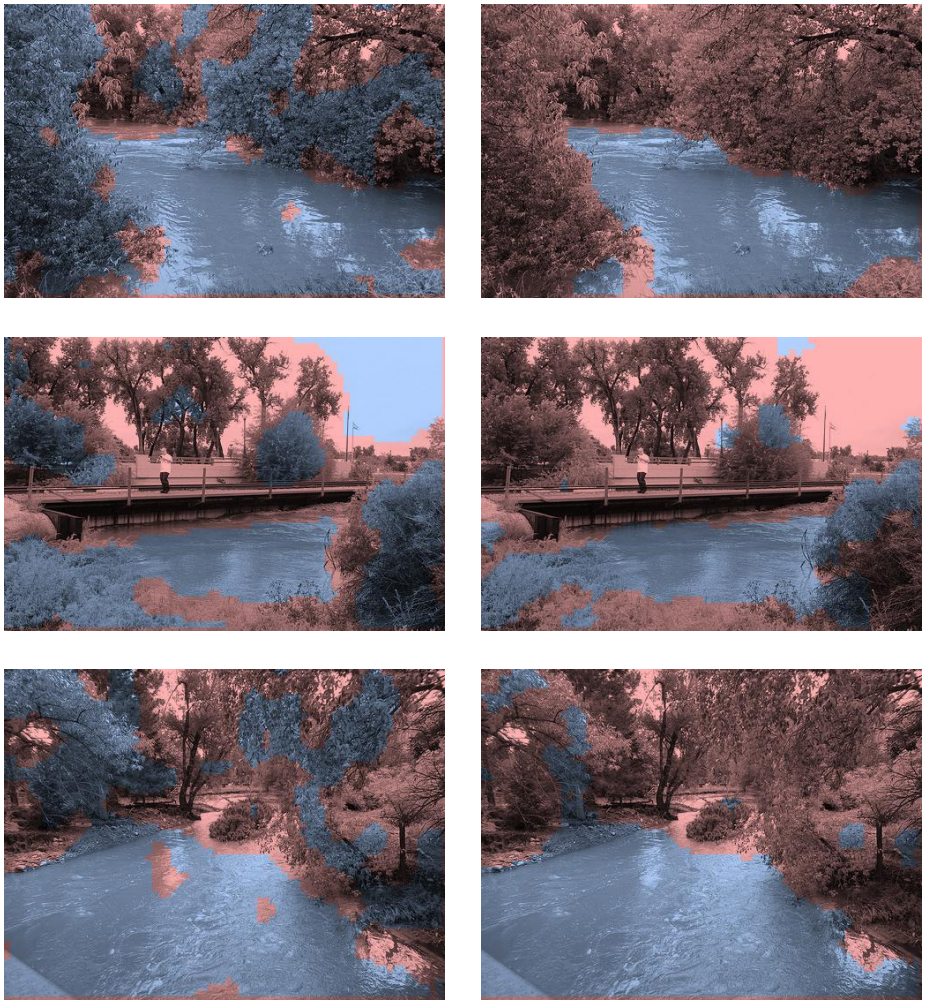


Figure 9: segmentation results - left: Bayesian classifier, right: SVM

$$J_2 = \frac{|S_m|}{|S_w|} = |{S_w}^{-1} S_m| \tag{7}$$

$$J_3 = trace({S_w}^{-1} S_m) \tag{8}$$

where $S_m$ and $S_w$ are *mixture* and *within class* scatter matrices, respectively. The larger the values for these class separability measures, the more well-clustered individual classes are within each class, and the more well-separated different classes will be. We have particularly studied $J_1$ and $J_3$ measures and have used these as criteria telling us how good our features are. It is worth mentioning that we have normalized these separability measures by the dimension at which that measure was assessed. This is because the *trace* increases as the dimension (and thus the matrix size) goes up while we want our measures to be representative of feature effectiveness irrespective of the dimensions.

Table 5 includes calculated normalized values for $J_1$ and $J_3$ before and after applying PCA to data, and also for different values of $k$ used in PCA. Looking at the values from the table, PCA with $k = 1$ has the highest $J_1$ and $J_3$ out of all for Bayesian approach. Although this could to some extent be the effect of normalization (this measure was normalized by a smaller value compared to other values of $k$), it also shows that classes resulting from classification of PCA-reduced features with a value of $k = 1$ are have less variance within classes, and classes are well far apart (despite the errors the classifier has made).

Table 5: Bayesian Classifier - Class Separability Measures Comparison

| Feature Reduction | normalized $J_1$ | normalized $J_3$ | Training Error (%) | Test Error (%) |
|---|---|---|---|---|
| — | 0.000456 | 1.000343 | 8.30 | 17.31 |
| PCA, $k = 8$ | 0.140103 | 1.065781 | 13.27 | 15.24 |
| PCA, $k = 16$ | 0.069561 | 1.033647 | 12.24 | 14.27 |
| PCA, $k = 32$ | 0.034585 | 1.017216 | 12.69 | 14.43 |
| PCA, $k = 64$ | 0.017204 | 1.008702 | 12.42 | 17.24 |
| PCA, $k = 128$ | 0.008557 | 1.004425 | 11.96 | 17.22 |

# 4 Unsupervised Learning

## 4.1 K-means Clustering

### 4.1.1 Linear Dimension Reduction

To apply linear dimension reduction, we used PCA to reduce dimensions to $k = 3$ to make the clustering problem computationally more feasible and also for the sake of visualization. In running the K-means algorithm, we set the desired number of clusters to $n = 2$. Table 6 provides some insight into the clustering performance. Accuracy in assigning labels to samples has been reported. From the results, it can be found that we were able to bring the dimensions down to $k = 3$ while losing little accuracy in label assignment.

As part of our clustering evaluation, we have used the confusion matrix, i.e. a matrix $A$ whose $(i, j)$ element is the number of points that originally belong to the $i^{th}$ class and have been assigned to the $j^{th}$ cluster. We have also reported the homogeneity and completeness measures of clusters (we have used true labels in deriving these measures). Homogeneity refers to the quality that clusters contain only data points which are members of a single class. On the other hand, a clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster. Both scores have positive values between 0 and 1, and larger values are desirable.

Table 6: K-means Clustering Using PCA

| Dimension Reduction | Accuracy | Normalized Confusion Matrix | homogeneity | completeness |
|---|---|---|---|---|
| — | 66.955 | $\begin{bmatrix} 0.51 & 0.49 \\ 0.17 & 0.83 \end{bmatrix}$ | 0.0987 | 0.1067 |
| PCA, $k = 3$ | 63.755 | $\begin{bmatrix} 0.47 & 0.53 \\ 0.19 & 0.81 \end{bmatrix}$ | 0.0651 | 0.0708 |

Figure 10 visualizes clustering results. We have also visualized the derived confusion matrix in figure 11.

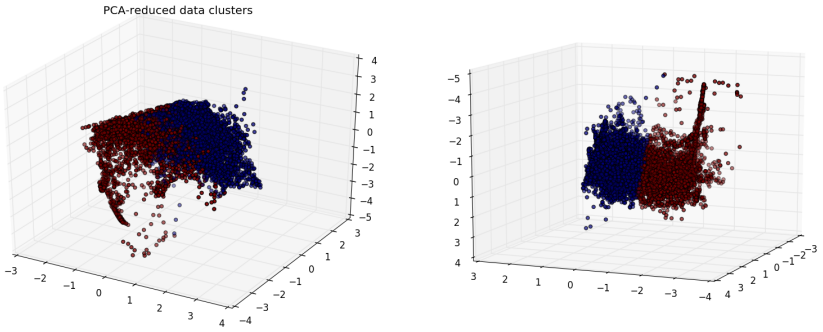



Figure 10: Clustering performance Using PCA - different views

### 4.1.2 Nonlinear Dimension Reduction

The nonlinear dimension reduction approaches discussed in section 2 (LLE and isomap) were also applied to high-dimensional features and clustering performances of each was assessed. In case of both techniques, dimension was reduced to $k = 3$. See table 7 for performance summary. Please note that, to make isomap computationally feasible to carry out of such high dimensions, we initially reduced a very large dimension of 2352 by the first 100 principal components using PCA and then ran the isomap algorithm on data with dimension 100 to then further reduce the dimensions to 3. Figure 12 shows clusters after isomap dimension reduction. For visualization purposes, a 2-D mapping of isomap was also done and has been plotted in figure 13.

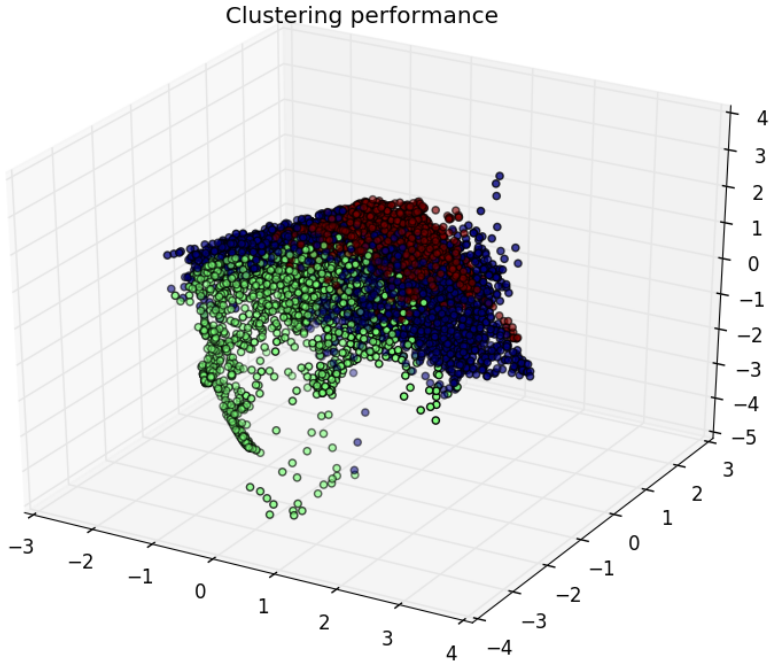


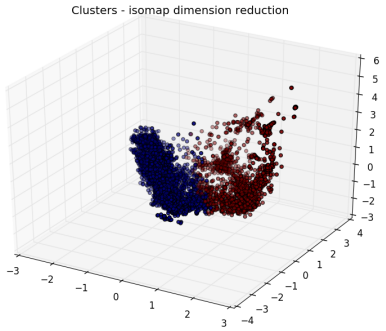

Figure 11: Confusion Matrix Visualization of PCA-reduced Clustering - "blue" shows wrongly clustered data points (summation of $A_{12}$ and $A_{21}$), "red" corresponds to the $A_{11}$ while "green" corresponds to $A_{22}$.

As expected, we get the best homogeneity and completeness in case of no dimension reduction. If we can process the high-dimensional data, we can build more robust clusters. As we reduce dimension of data, we face the trade-off between accuracy and computational cost. We can run clustering more easily on data with smaller dimensions, but our clustering results will accordingly be less accurate.

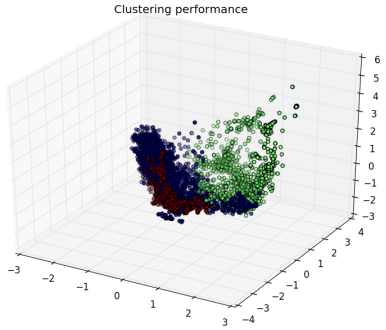
Table 7: K-means Clustering Using Nonlinear Dimension Reduction

| Dimension Reduction | Accuracy | Normalized Confusion Matrix | homogeneity | completeness |
|---|---|---|---|---|
| — | 66.955 | $\begin{bmatrix} 0.51 & 0.49 \\ 0.17 & 0.83 \end{bmatrix}$ | 0.0987 | 0.1067 |
| LLE, $k = 3$ | 57.073 | $\begin{bmatrix} 0.32 & 0.68 \\ 0.17 & 0.83 \end{bmatrix}$ | 0.0218 | 0.0272 |
| Isomap, $k = 3$ | 62.491 | $\begin{bmatrix} 0.42 & 0.58 \\ 0.16 & 0.84 \end{bmatrix}$ | 0.0587 | 0.0673 |

In summary, out of all the dimension reduction techniques for the K-means algorithm, PCA performed the best, but isomap showed a performance of almost as good as PCA. LLE mapping resulted in larger errors in label assignment by

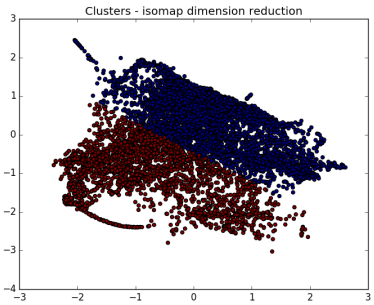



(a) Two Clusters

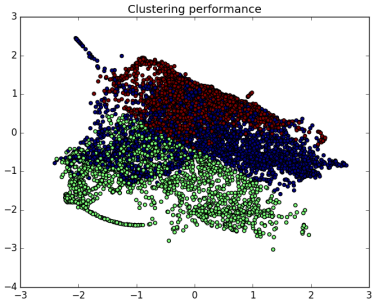



(b) Confusion matrix "A" visualization- "blue":$A_{12}$+$A_{21}$, "red":$A_{11}$& "green":$A_{22}$

Figure 12: Clustering performance Using Isomap with 3-D Mapping




(a) Two Clusters




(b) Confusion matrix "A" visualization- "blue":$A_{12}$+$A_{21}$, "red":$A_{11}$& "green":$A_{22}$

Figure 13: Clustering performance Using Isomap with 2-D Mapping

K-means.

## 4.2 Mini Batch K-means Clustering

### 4.2.1 Results

Mini Batch K-means is a faster slightly different version of K-means. Unlike K-means, where we set the number of clusters to 2 prior to running the algorithm, we investigated different number of clusters and clustering error associated with each. We also analyzed the Mini Batch K-means performance under various values of $k$ for PCA dimension reduction. Table 8 represents a summary of our results with Mini Batch K-means.

Table 8: Mini Batch K-means - Clustering Results

| Dimension Reduction | Number of clusters | Error (%) |
|---|---|---|
| PCA, $k = 16$ | 2 | 35.92 |
| — | 3 | 22.61 |
| PCA, $k = 16$ | 3 | 24.44 |
| — | 4 | 25.88 |
| PCA, $k = 16$ | 4 | 23.39 |
| PCA, $k = 32$ | 2 | 31.71 |
| PCA, $k = 64$ | 2 | 33.59 |

#### 4.2.2 Analysis of Number of Clusters

Based on the information from table 8, the smallest error in assigning clusters to points (err=22.61%) corresponds to clustering of data into 3 clusters, where feature dimension is not reduced beforehand. Considering the computational complexity of efficient clustering in high-dimensional feature spaces, one could consider reducing dimensions *a priori* to 16 using PCA and then cluster.

## 5 Conclusion

The major challenge in this project was to obtain a set of features that would provide for the classification of water and non-water regions in natural images. The approach taken here was to use PCA to reduce the dimensionality of the data set and then implement a neural network classifier to apply a non-linear transformation to the data set producing a potentially more separable set of features that could then be classified by standard linear classifiers. The trained classifiers in our experiments applied to the problem of image segmentation were able to successfully identify the general morphology of natural images containing both water and non-water regions. Future work would definitely benefit from a more sophisticated framework for constructing the neural networks. Adding a SVM as the classifier on the output layer of the network along with pre-training of the hidden layers with an unsupervised approach (such as sparse coding with a learned dictionary) could potentially provide a higher performance classifier.

## References


[1] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. CRC press, 2000.

[2] Lawrence K Saul and Sam T Roweis. An introduction to locally linear embedding. *unpublished. Available at: http://www. cs. toronto. edu/˜ roweis/lle/publications. html*, 2000.

[3] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition.* Academic Press, 4th edition, 2008.