Data Mining for Hyperspectral Images

GRADUATE PROJECT PROJECT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University - Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Xinyi Wang
Summer 2014

Committee Members

Dr. Longzhuang Li _____
Committee Chairperson

Dr. Scott A. King _____
Committee Member

Dr. _____
Committee Member

ABSTRACT

Hyperspectral images are a series of image frames in different wavelength bands which form a cube of data with both spatial and spectral information. Data mining is to mine the useful data from the huge datasets. Since the pixels of hyperspectral images are all in very high dimensional space, data mining for hyperspectral images, or also called hyperspectral image classification, is quite challenging. A different approach with more process steps is introduced for better accuracy. The system contains hyperspectral data understanding, data preparation, vegetation detection algorithm, the k-means algorithm, and support vector machine. Data understanding is the first step to let people know about the datasets. Data preparation is the basic step for input of data mining algorithms because some of the algorithms need to have a specified format. Vegetation detection algorithm is to mine the useful data for the k-means algorithms. The k-means algorithm can generate some feature for support vector machine (SVM). SVM is to apply the features from the previous step and to mine the data, and finally some accuracies are generated according to those processes of hyperspectral imaging dataset, which helps us test the model.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Information age is a popular term to describe the world we are living in. However, what is information made up of? Actually it is a wide range of data. Data can be huge, but not all of them are useful for us. How do we abstract the useful data from it? A correct data mining technique is therefore important for numerous practical purposes, especially for the huge dataset. Data mining is currently being exploited in many applications, such as games, businesses and sciences.

In this study, a hyperspectral imaging system is used to obtain the hyperspectral images of some mud. To mine the significant objects from these mud samples is the main purpose here. The general steps in this project are hyperspectral data understanding, pre-processing, unsupervised learning, feature selection, supervised learning (post-processing), and rules or decision. Data understanding is the first step to let people know about the datasets. Pre-processing is the basic step for data cleaning because the experimental datasets contain lots of noise or even missing data. Unsupervised learning is the next step to mine the data from the cleaned data by using K-means. Feature selection is the step right after unsupervised learning which can generate some feature after data mining algorithm. Supervised learning, also known as post-processing is to apply the features from the previous step and to mine the data, and finally some rules or decision is generated according to those processes of hyperspectral imaging dataset. These data mining algorithms are presented for hybrid techniques. The final goal of this paper is to test more data mining models for hyperspectral image data, including more processing in order to improve the accuracy, and finally provides some conclusions and recommendations regarding these methods.

1.1 Statement of the Problem

It is important for researchers to abstract the useful information from the huge dataset in any field. The question can be summarized as follows:

Given a dataset that contains information in the form of hyperspectral variables that are believed to significantly influence the objects, what are the most appropriate data mining techniques to determine useful information with a reasonable margin of error?

To answer this question, this study explores several data mining methods that might be used to determine the significant objects by using data of hyperspectral images on mud samples. The main goal of this work focuses on studying the currently available applicable data mining tools (such as the k-means algorithm and support vector machine) for finding out the significant objects.

Hyperspectral image classification is the method used to abstract information from images. Finding the optimal approach for hyperspectral image classification for a given application becomes a popular research problem for computer scientists.

Hyperspectral images are a series of image frames in different wavelength bands which form a cube of data with both spatial and spectral information. In each pixel of the hyperspectral image, wavelength bands versus raw intensity counts (arbitrary units) can be obtained and graphed. Different objects have different spectral information in certain wavelength bands. Therefore, the goal is to understand the microdistribution of organisms, by analyzing the plots. Figure 1.1 gives you the idea of what a hyperspectral image look like.

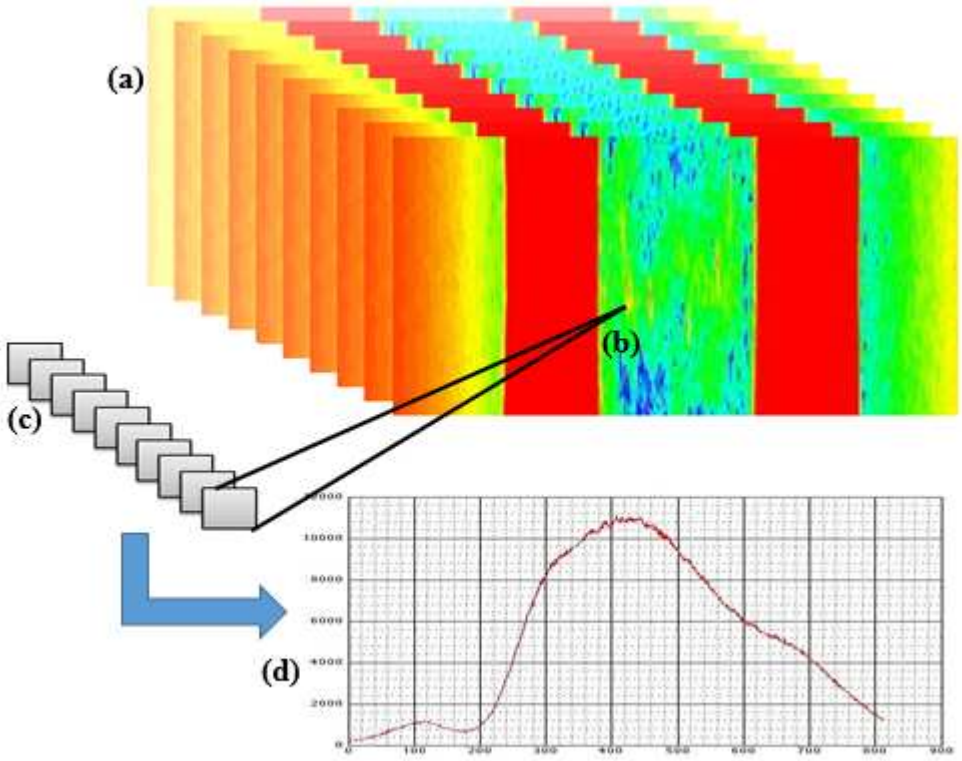

Fig. 1.1. (a) A hyperspectral image of gold nanoparticles. (b) A single pixel location from the hyperspectral image. (c) The pixel at different frames with the same location. (d) The spectrum of the single pixel location, corresponding to a series of image frames at different wavelength.

The essence of hyperspectral image classification using spectral analysis in this project involves processing data through data mining. Data mining refers to finding patterns in large data sets, such as the hyperspetral images, which can be as large as 20 Giga Bytes (GB) each. Several algorithms have been defined in the data mining domain such as decision trees, artificial neural networks, K-means, and support vector machine (SVM) algorithm. Determining and building an efficient and accurate classifier based on the optimal data mining algorithm is one of the objectives of this mud hyperspectral image classification project.

In this study, the hybrid of pre-processing, unsupervised learning and supervised

learning is presented for hyperspectral image analysis and surface classification from mud samples. In this system, the classifier has three layers: the first layer is for pre-processing. The medial layer is unsupervised learning, the k-means algorithm. The last layer represents supervised learning, support vector machine.

1.2 Review of the Literature

Since entering in the space age in 1957, people keep documenting the image data of Earth from outer space [1]. The image information can be analyzed for different purposes, such as military uses, agriculture, and urban planning [2]. Hyperspectral images classification is more challenging because of the very high dimensionality of the pixels and the small number of labeled examples typically available for learning [3].

1.2.1 Hyperspectral Image

Because human eyes only can see the light under bands from 390nm to 750 nm, there are requirements for detecting visions, whose bands are below 390 nm or beyond 750nm [4]. For example, the gold nanoparticles are actually spherical organic coated particles, and dissolved in toluene. Then the old nanoparticles dissolved in toluene were further diluted in different concentration of 14%, 28% and 50% in toluene. Human eyes are mostly sensitive at around band of 555 nm.

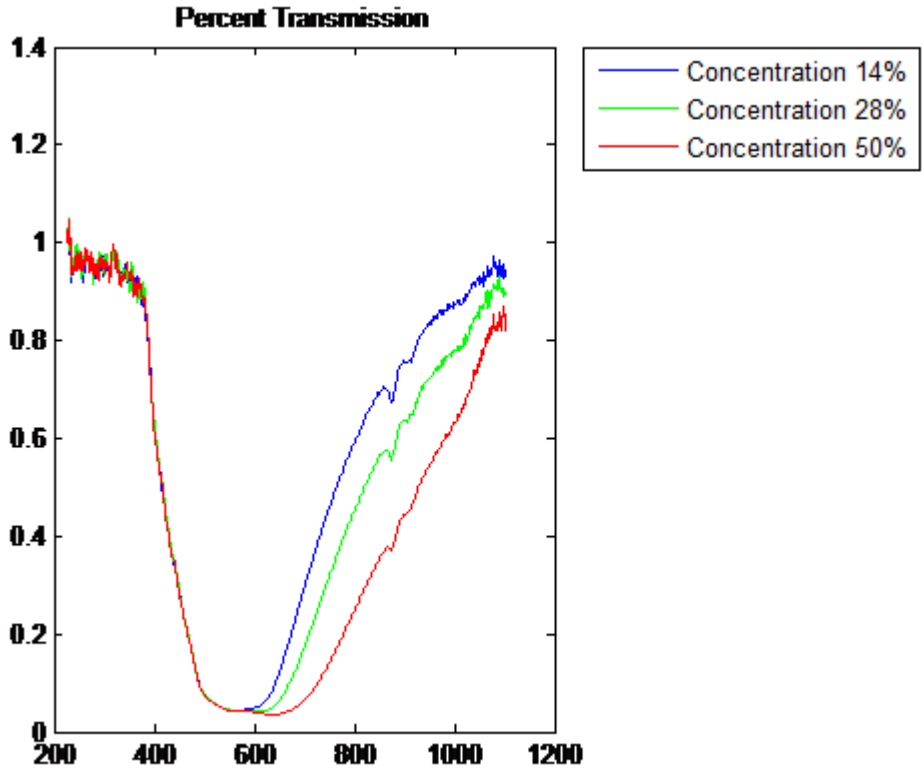

Fig. 1.2. Percent Transmission of three different gold nanoparticle concentrations.

From Figure 1.2 mentioned, it shows that the figures of three samples overlap each other at around band of 555 nm but split up at band of 600 nm or above. It means that it is hard to identify the concentration of three samples by human eyes. If the research is extended beyond the visible, there are some significant differences between the samples.

1.2.2 The k-means Algorithm: A Centroid-Based Technique

A centroid-based partitioning technique uses the centroid of a cluster, C_i, to represent that cluster. Conceptually, the centroid of a cluster is its center point. The centroid can be defined in various ways such as by the mean or medoid of the objects or points assigned to the cluster.

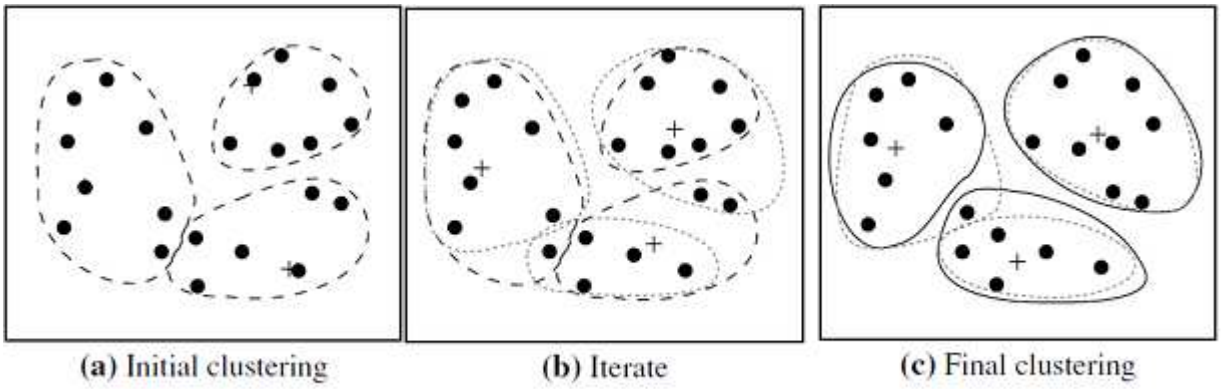

Fig. 1.3. Clustering of a set of objects using the k-means algorithm.

The k-means algorithm is for partitioning, where each clusters center is represented by the mean value of the objects in the cluster [5]. Figure 1.3 shows how k-means algorithm works.

1.2.3 Support Vector Machine (SVM)

SVM is a method for the classification of both linear and nonlinear data. It uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane. With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors and margins. The data mining algorithm SVM is capable of finding the maximum margin for image pattern recognition with high classification accuracy and limited architecture design [6].

1.3 Methodology

Going back to the original research question: What data mining techniques can be used to find out the significant objects? The literature review in the previous section

indicated different alternatives to answer this question. This study selects the techniques that were most successfully used in the literature. A practical example with real data from mud sample solves for the objects using these different techniques and their results are compared.

A general data mining is used to find out the significant objects for mud samples. Chapter Two shows the technique of identifying how differences between the visible red and near-infrared bands of the mud sample image, which we can use this technique to identify the areas having the significant objects; k-means, a data mining technique, is an unsupervised learning, which means we can find out the classes or groups, and label them for the final data mining process, SVM; and the SVM technique that takes into account the classes or groups in training the unknown dataset, providing a prediction of each groups. Chapter Three shows how the approaches work and the system architecture. Chapter Four focuses on experimental results from testing a hyperspectral image. Finally, Chapter Five discusses these data mining techniques and provides conclusions and recommendations regarding these methods.

1.4 The sample

The hyperspectral image of the mud sample is taken from Hyperspectral Optical Property Instrumentation (HOPI) Lab at Texas A&M University —Corpus Christi. Figure 1.4 shows the photo of the mud sample.

Fig. 1.4. The photo of the mud sample.

Figure 1.5 shows the hyperspectral image of the mud sample in MATLAB with gray scale and original axis.

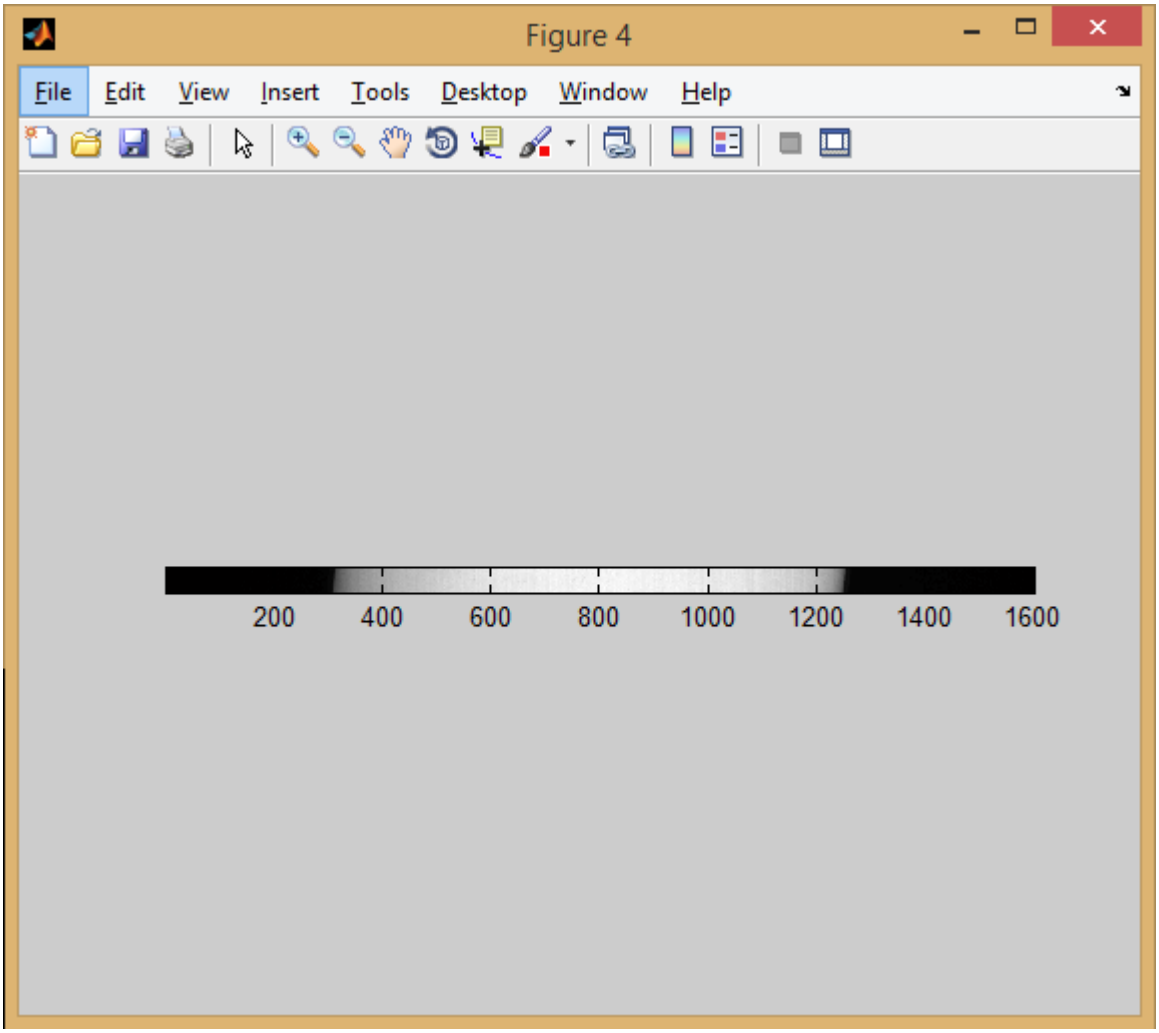

Fig. 1.5. A hyperspectral image of a mud sample.

1.5 Exploring the Dataset

Each hyperspectral image has its own header file, which records the date of taking the image, samples, lines and bands. The important information is saved here for setting up the parameter when the image is processed. Figure 1.6 shows what the header file includes.

```
ENVI
description = {[Thu, Dec 06, 2012, 2:49:48 PM]}
samples = 1600
lines   = 50
bands   = 811
header offset = 0
file type = ENVI Standard
data type = 4
interleave = bil
sensor type = 1003A-10145, SN:G4-195; Slit Width 25 um
byte order = 0
wavelength units = nm
wavelength = {
399.538849, 400.279535, 401.020221, 401.760907, 402.501592, 403.242278,
403.982964, 404.723650, 405.464336, 406.205022, 406.945708, 407.686394,
408.427079, 409.167765, 409.908451, 410.649137, 411.389823, 412.130509,
412.871195, 413.611881, 414.352566, 415.093252, 415.833938, 416.574624,
417.315310, 418.055996, 418.796682, 419.537368, 420.278053, 421.018739,
421.759425, 422.500111, 423.240797, 423.981483, 424.722169, 425.462855,
```

Fig. 1.6. The header file of a hyperspectral image of a mud sample.

From Figure 1.5, it shows that a hyperspectral image can be imported into MATLAB through the code below:

```
im = multibandread(fn, [lines,samples,bands],'single',0,'bil','ieee-le');
```

Once processed, the saved hyperspectral image is shown in the workspace of MATLAB:

```
im
50x1600x811 double
Cannot display summaries of variables with more than 524288 elements.
```

The size of the hyperspectral image of mud sample is 50-by-1600 with 811 bands in depth. Once the im is opened, its said that:

```
im
50x1600x811 double
Cannot display summaries of variables with more than 524288 elements.
```

To process this hyperspectral image, data preparation in each chapter is always

the first and important step to deal with. The Figure 1.7 is the hyerpsectral image of mud sample in MATLAB with color and square axis. Next, all the related algorithm or techniques is shown in Chapter Two.

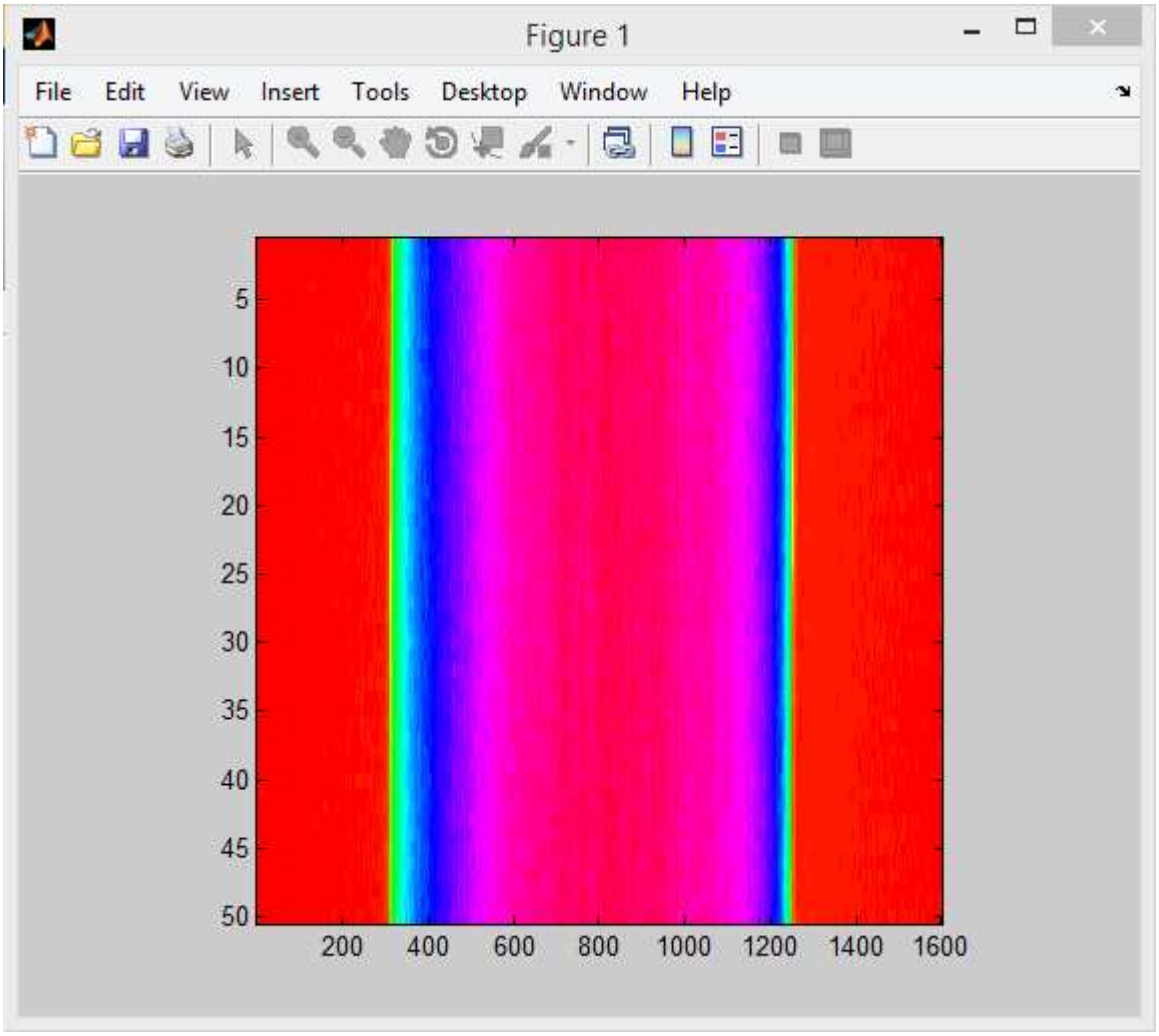

Fig. 1.7. The hyperspectral image of mud sample in MATLAB with color and square axis.

2. RELATED ALGORITHMS

The related techniques or algorithms are vegetation detection algorithm, the k-means algorithm and support vector machine. Algorithm for vegetation detection can detect the hyperspectral image for the purpose of predicting the percentage of significant objects. The k-means algorithm, a centroid-based technique, will help us find out the clusters. The k-means algorithm is a cluster analysis algorithm, which mainly calculates data aggregation by constantly computing the average distances between each points and mean point. Machine learning is essentially an approximation of the true model of problems, which means the best approximate mode that is used for the true model is called a hypothesis. So there is no doubt that the true model is certainly unknown. Otherwise, the true model can just be used and there is no point to have machine learning. Since the true model is unknown, how much differences between the hypothesis and the real solution are also unknown. However, if some attributes or instances are known, they can be used to approach to the real model. Support Vector Machine is used to find the errors, approach the true model and predict the unknown model.

2.1 Vegetation Detection Algorithm

This algorithm tells us how to find out the differences between the visible red and near-infrared bands of a Landsat image and the percentage of the Vegetation. To predict the percentage, we need to find out the differences between the visible red and near-infrared bands of the hyperspectral image. This technique identifies the pixels or area that contains the significant objects, such as algae.

To achieve this result, there are six steps in this original algorithm [7], shown in Figure 2.8:

Step 1: Import CIR bands from a BIL image file
Step 2: Enhance the CIR composite with a de-correlation stretch
Step 3: Construct an NIR-red spectral scatter plot
Step 4: Compute vegetation index via MATLAB array arithmetic
Step 5: Locate vegetation – threshold the NDVI image
Step 6: link spectral and spatial content

Fig. 2.8. Vegetation detection algorithm.

2.2 The k-means algorithm

K-means algorithm is a typical clustering algorithm based on distances, using distance as the similarity index for evaluation, which also considers the closer the objects, the greater the similarity. The clusters in this algorithm are determined by nearby objects, so the ultimate goal of this algorithm is to get compact and independent clusters. Figure 2.9 shows the k-means algorithm [5].

Input of k-means:
1) k is the number of clusters;
2) D is a data set containing n objects
Output of k-means: A set of k clusters
Method:
1) Randomly choose k objects from D as the initial cluster centers;
2) Repeat;
3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
5) until no change.

Fig. 2.9. The k-means algorithm.

The input of the k-means algorithm, k, is very important, but the question is how

the value of k is determined. In the k-means algorithm, k cluster centers are chosen at first, where k is a user-specified parameter, which is the desired number of clusters. Therefore, its known the number of clusters contained in the dataset. However in many cases, the distribution of the dataset is unknown. In fact, clustering is one way to find the distribution, which leads us into the contradiction of which came first, the chicken or the egg. Some methods of how to effectively determine the value of k will be introduced.

The first method is to find the stabilization of a dataset. A dataset can be resampled twice to generate two subsets, and then the two subsets can be clustered by using the same algorithm, which generates two sets of k clusters. Next, the similarity of these two sets are calculated. If the degree of similarity is high, it means this k-cluster is stable and can be used for the entire dataset [8].

The second method is starting the k-means algorithm with k=1, and going through the process of splitting and merging. The structure will evolve its stable equilibrium state k_i. Its very useful for a slightly overlapping dataset [8].

The next question is how to choose the appropriate initial cluster centers. One method is to choose the initial centers randomly, but this may cause poor quality of the clusters. However, the initial cluster centers can be chosen randomly multiple times, and then run each set with different random centers. Finally, the cluster with the least sum of squared errors (SSE) is chosen [5].

The second effective method is to take a sample and use hierarchical clustering techniques. k clusters can be extracted from the hierarchical clustering, and used as the initial cluster centers. This method is effective, but only for the following two situations: first, the sample is relatively small; second, the value of k is relatively smaller than the size of the sample [5].

Another method is to choose the first cluster center randomly or take the first

points as their cluster centers. For each subsequent initial center, the point that is farthest away from the initial center is chosen. Its certain that the initial cluster center is not only random but also spread out by using this method, but beware of the outliers. In addition, the cost of finding out points farthest from the current set of initial centers is also very large. One way to overcome this problem is to use the samples, which can also reduce the amount of computation [5].

Which distance measure is better becomes the next question. Euclidean distance and cosine similarity are different in mathematical logic. Euclidean distance measure is affected by different scale indicators. Therefore, generally it need to be standardized, and the greater the distance is, the greater the differences between individuals are. Cosine similarity measure in vector space is not affected by the scale indicators. The value of cosine lies in the interval [-1,1], and the larger the value is, the smaller the differences are. However, for specific applications, under what circumstances are better for using Euclidean distance or under what circumstances are better for the use of cosine similarity?

In the geometric meaning, a triangle is formed by using a segment in an n-dimensional vector space as a base and the origin as a apex, but the angle of the apex is unknown. It means that even though the distance between two vectors are certain, the value of cosine of the angle between them is still uncertain. If two vectors have the same trend, the cosine similarity tends to have better performance than Euclidean distance [8]. For example, two vectors are (9,9) and (6,6). Obviously, these two vectors have the same trend, but the solution of Euclidean distance is way worse than cosine similarity.

If there are no points assigned to some cluster, it can form an empty cluster. A strategy is required to choose a substitute cluster center. Otherwise, the sum of squared errors will be large. One way is to choose a point farthest away from the

current cluster center, which can eliminate the most impact on sum of squared errors. Another method is to choose a substitute from the cluster center with the largest sum of squared errors, which can split the clusters and reduce the sum of squared errors. If there is more than one empty cluster, the process should be repeated several times. In addition, the empty cluster may cause bugs during the programming.

2.3 Support vector machine

Data classification is one of the most important subjects in data mining. Data classification is based on some existing training data, and then generates a trained classifier according to some theory. Next, the trained classifier is used to test the unknown data to find out a new classification.

One of these theories is Support Vector Machine (SVM). SVM is first proposed by Cortes and Vapnik in 1995. SVM has its advantage of solving the problem of nonlinear and high dimensional pattern recognition, and be able to extend the use to other machine learning problems, such as the function fitting [5].

Support Vector Machine is a method that is based on the boundary classification. The basic principle of SVM can be defined as follows: For the better understanding, two-dimensional data is taken as an example. If all the points of a training data is distributed on a two-dimensional plane, they are gathered in different areas according to its own classification. The goal of those classification algorithm based on the boundary classification is to find the boundary between the classifications through training. If the boundary is a straight line, this case is linear; if the boundary is a curve, this case is considered as non-linear. For the multi-dimensional data, such as N-dimensional, they can be regarded as the points in the N-dimensional space, and the boundary in such space is (N-1)-dimensional plane. Thus, linear classifier in multi-dimensional data is a hyper-plane, and non-linear classifier in multi-dimensional

data is a hyper-curve.

Linear classifier is shown in the following Figure 2.10. According to the position of the new data point against the boundary, its classification can be determined. Here is the example of only two classifications, but it can be easily extended to multi-classification.

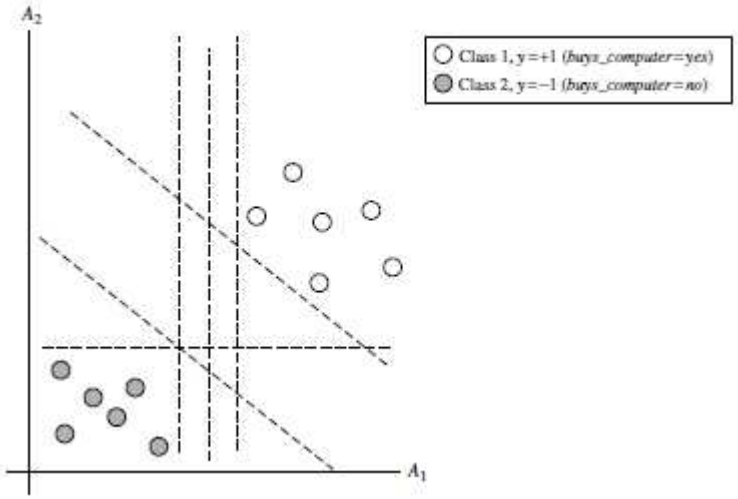

Fig. 2.10. Linearly separable data.

2.3.1 Linearly Separable

From Figure 2.10, the white points are class 1, the black points are class 2, and there are several straight lines that separate these two classifications are represented as $f(x) = k \cdot x + b$, where both k and b are vectors. Therefore, they can also be written in the form of $f(x) = k_1 \cdot x_1 + k_2 \cdot x_2 + \cdots + k_n \cdot x_n + b$. When the dimension of vector x is 2, $f(x)$ is a straight line in a two-dimensional plane; when the dimension of vector x is 3, $f(x)$ is a plane in a two-dimensional space; when the dimension of vector x is greater than 3, $f(x)$ is an (n-1)-dimensional hyper-plane in an n-dimensional space. Here, sign function will used to determine which classification the points belong to. When $f(x)$ is greater than zero, $sign(f(x))$ is +1; when $f(x)$ is less than zero, $sign(f(x))$

is -1. As shown in the Figure 2.10, the y value of the white points is +1, and the y value of the black points is -1.

However, there are several lines that can separate these two classifications, so how to determine the optimal one is the key to SVM. In Figure 2.11, there are two possible separating hyper-planes and their margins.

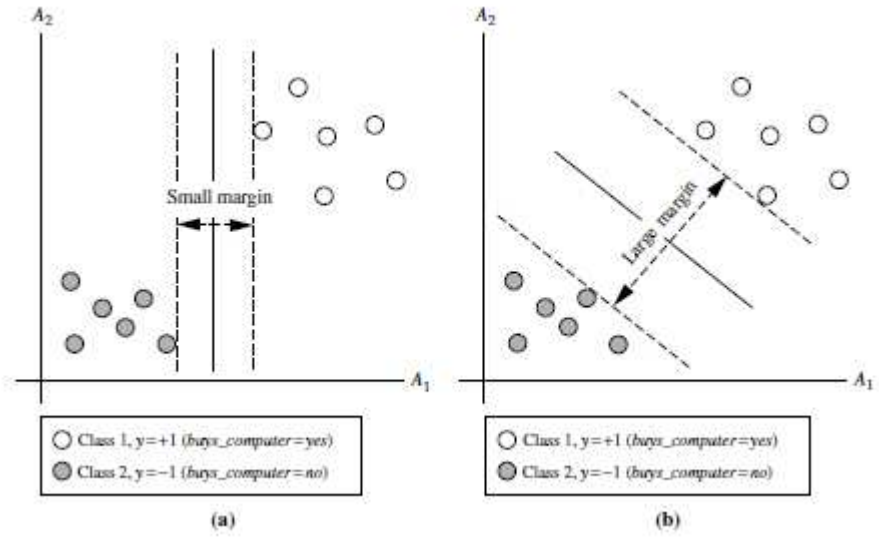

Fig. 2.11. Two possible linearly separable cases.

Intuitively speaking, the larger the margin is, the better the case is. So, maximum marginal hyper-plane is one of the most important conditions in the SVM theory. It makes more sense that the linear classifier with the largest margin is chosen. From a probabilistic point of view, it makes the points with the least confidence have the most confidence. The points on the dash lines are called support vectors, and the straight line in the middle is called classifier boundary $sign(f(x))$. If M is the margin width, M can be defined as

$$M = \frac{2}{\sqrt{k \cdot k}} \tag{2.1}$$

Therefore, the problem is transferred into finding out the optimization:

$$\max \frac{1}{\|k\|} \rightarrow \min \frac{1}{2}\|k\|^2 \tag{2.2}$$

where $\|k\|$ is the Euclidean norm of k, which also equal to $\sqrt{k \cdot k}$. Therefore, the original SVM problem is also the optimization problem:

$$\begin{cases} \min \frac{1}{2}\|k\|^2 \\ s.t.\ y_i(k^T x_i + b) \geq 1,\ i = 1, 2, \ldots, n \end{cases} \tag{2.3}$$

2.3.2 Linearly Inseparable

Since there are too many limitations of linearly separable cases in reality, linearly inseparable cases are mentioned here in Figure 2.12.

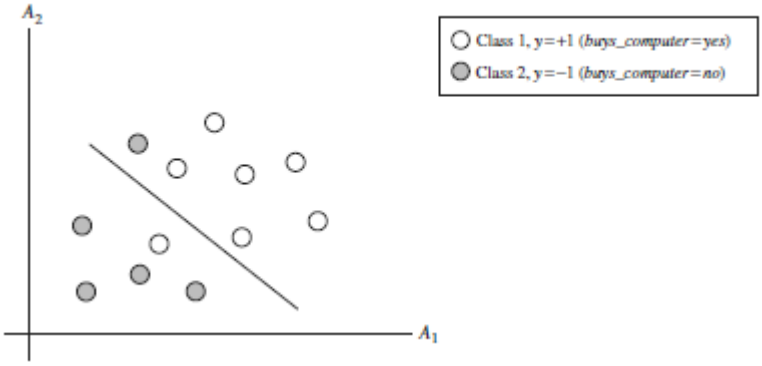

Fig. 2.12. Linearly inseparable data.

In order to do the classification in this case, there are two ways: one is to use the straight line or hyper-plane, but it is not guaranteed that all the points should be separated according to the classification. However, the error is not guaranteed also; the other is to use the curve to completely separate all the points, and its associated with the kernel function.

Since the error of the first method is not guaranteed, here the second method

is detailed. In the Figure 2.12, it isnt possible to separate all the points by using a straight line, so the original input data has to be transferred into a higher dimensional space for linearly separation. However, the nonlinear mapping to a higher dimensional space is uncertain, and the computation involved may be costly. There are three admissible kernel functions (shown in Figure 2.13) with which can be replaced anywhere $\phi(X_i) \cdot \phi(X_j)$shows in the algorithm.

Polynomial kernel of degree *h*:	$K(X_i, X_j) = (X_i \cdot X_j + 1)^h$
Gaussian radial basis function kernel:	$K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$
Sigmoid kernel:	$K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$

Fig. 2.13. Three kernel functions.

2.3.3 Multiple classification (N-classification)

In the case of multiple classification, there are two basic methods: one is 1 vs (N-1) method; the other is 1 vs 1 method.

In the first method, N classifiers are trained, and whether i-th classifier belongs to classification i or other classifications is checked. To identify an x, a classification with the largest $g^j(x)$ is where i belongs to. $g^j(x)$ is defined as

$$g^j(x) = \sum_{i=1}^{j} a_i^j y_i K(x, x_i) + b^j \tag{2.4}$$

In the second method, $N * (N-1)/2$ classifiers are trained. To identify an x, a vote method is used. The classification with the most votes finally gets x.

2.4 Discussion

All the algorithms are used to form a hybrid system. Algorithm for vegetation detection is the first step to detect percentage of significant objects of the hyperspectral

image. The k-means algorithm, a centroid-based technique, is considered as the second step to help us identify the location of clusters. Support Vector Machine is used to be the last step to find the errors, approach the true model and predict the unknown model.

3. THE HYBRID SYSTEM DESIGN

Data preparation is the very first step of the algorithm. Figure 3.14 shows a diagram of the system architecture. The hyperspectral dataset is trained by using algorithm for vegetation detection, the k-means algorithm and support vector machine, and an unknown dataset is tested to get the accuracy by using the test mode, 10-fold cross-validation.

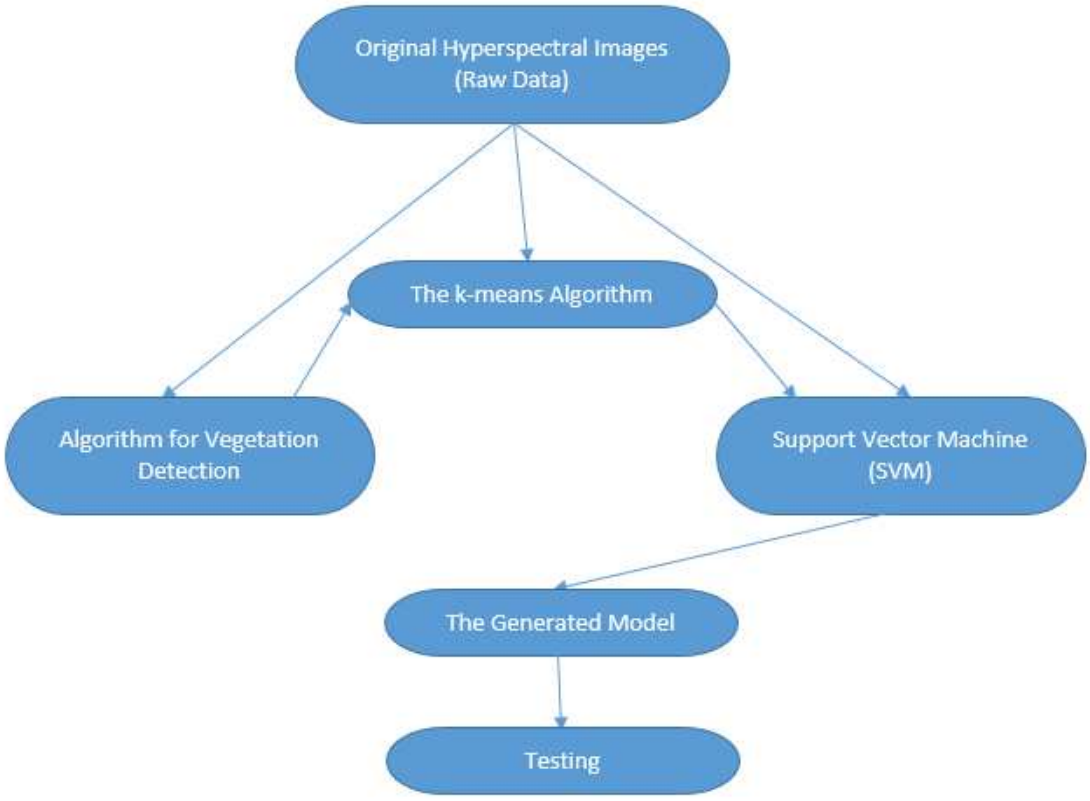

Fig. 3.14. The system architecture.

3.1 Data Preparation

The dataset is an 811-band 50-by-1600 hyperspectral image of a mud sample, whose pixel values are band interleaved by line (BIL) in order of increasing band number.

BIL is one of three primary methods for encoding multiband image data, which has the advantage of accessing the spatial and spectral data easily.

Since the hyperspectral image is a three dimensional dataset, it isnt mathematically convenient to operate for the further data mining. Therefore, the three dimensional dataset is reformatted into two dimensional dataset by having each column display the entire pixel values of each layer or band, so we make a new 80000-by-811 dataset.

3.2 Vegetation Detection Algorithm

The very first step is to select the three most informative bands from the mud sample file. To determine these three bands, we need to find out how many 0s are in each band. Therefore, by figuring out the three least 0s bands, we can know the three bands we want. With the three bands, we can continue to our algorithm to find out the percentage of the significant objects.

The color-infrared (CIR) composite is the result when the near infrared (NIR), the visible red and the visible green are mapped into the red, green and blue planes of an RGB image. To read the hyperspectral image, the MATLAB function multibandread can help us handle it in a special form that we can recognize. Therefore, when we use the MATLAB function multibandread from the three determined bands, we have an RGB image, which red means the NIR band, green means the visible red band and blue means the visible green band. However, still the contrast is not as good as we want.

Next, we will construct an NIR-Red scatter plot, which helps us compare the NIR band (displayed as red) and the visible red band (displayed as green). As we said above, we can easily extract the NIR and red bands from the CIR composite. So we can see the difference between these two bands in grayscale images.

To find out the significant objects, we have to compute the index of each pixel:

$$\text{Index} = (\text{NIR - red}) \,./\, (\text{NIR + red}) \tag{3.1}$$

In order to identify pixels most likely to contain significant objects, we apply a simple threshold to the image. We just need to create the significant objects image that have index greater than threshold. Percentage of significant objects is calculated by ratio of number of while pixels in binary image to total number of pixels in NIR image.

3.3 The k-means Algorithm

To compute the distance on numeric data, distance measures are commonly used, which includes Minkowski distance, Euclidean distance and Manhattan distance.

The Minkowski distance between sample point i and sample point j is defined as

$$d(i,j) = \sqrt[h]{|x_{i1} - y_{j1}|^h + |x_{i2} - y_{j2}|^h + \cdots + |x_{ip} - y_{jp}|^h} \tag{3.2}$$

where $i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ and $i = (x_{j1}, x_{j2}, \ldots, x_{jp})$ are two p-dimensional data objects, and h is the order (the distance so defined is also called $L - h$ norm). Here, h can be positive, negative, or even infinite. Therefore, here comes some special case of Minkowski distance when h is specified.

While $h = 1$, here comes Manhattan distance which also called as city block distance ($L_1 norm$). It is defined as

$$d(i,j) = \sqrt{|x_{i1} - y_{j1}|^2 + |x_{i2} - y_{j2}|^2 + \cdots + |x_{ip} - y_{jp}|^2} \tag{3.3}$$

Whats more, the supremum distance is generated from the Minkowsi distance for $h \to \infty$. It is defined as

$$d(i,j) = \lim_{h\to\infty} (\sum_{f=1}^{p} |x_{if} - x_{jf}|)^{\frac{1}{h}} = \max_f |x_{if} - x_{jf}| \tag{3.4}$$

There are also some important properties for these distance measures as follows:

$$d(i,j) > 0 if i \neq j, and d(i,i) = 0 (Positive definiteness) \tag{3.5}$$

$$d(i,j) = d(j,i) (Symmetry) \tag{3.6}$$

$$d(i,j) \leq d(i,k) + d(k,j) (Triangel Inequality) \tag{3.7}$$

Last but not the least, cosine similarity can be also used for distance measure. It is defined as

$$\cos(d_1, d_2) = \frac{(d_1 \cdot d_2)}{\|d_1\| \cdot \|d_2\|} \tag{3.8}$$

where d_1, d_2 are two vectors, $\cdot$ indicates vector dot product and $\|d\|$ is the length of vector d.

The purpose of using k-means is to divide the sample into k clusters. In cluster analysis, a training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ is given, where each $x^{(i)} \in \mathbb{R}^n$. First of all, k cluster centers will be chosen randomly as $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$.

Secondly, for every sample i, each cluster that the sample belongs to should be calculated as:

$$c^{(i)} := arg\min_j \|x^{(i)} - \mu_j\|^2 \tag{3.9}$$

And for every cluster j, the cluster centers (or mean points) of each cluster will be recalculated as:

$$\mu_j = \frac{\sum_{i=1}^{m} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)} = j\}} \tag{3.10}$$

Here, k is the given number of clusters, and $c^{(i)}$ represents the cluster that has the shortest distance between sample i and k clusters, where the value of $c^{(i)}$ is ranged from 1 to k. The cluster centers (or mean points) μ_j represent the guess of the center of the samples that belongs to the same cluster.

For example, in the universe, all the stars can be divided into k constellations.

To do that, first of all, k points (or k stars) in the universe are randomly chosen as cluster centers. Then the first step is to calculate each distance between each stars and k cluster centers, and choose the nearest constellations as $c^{(i)}$. Therefore, after the first step, all the stars will have their own constellations. The next step is, for each constellations, to re-calculate their new cluster centers μ_j, which means find out the average of all the stars in each constellations. Then, repeat step one and step two until there is no change for cluster centers or the change can be ignored. Figure 3.15 shows how k-means works for n sample points, here $k = 3$.

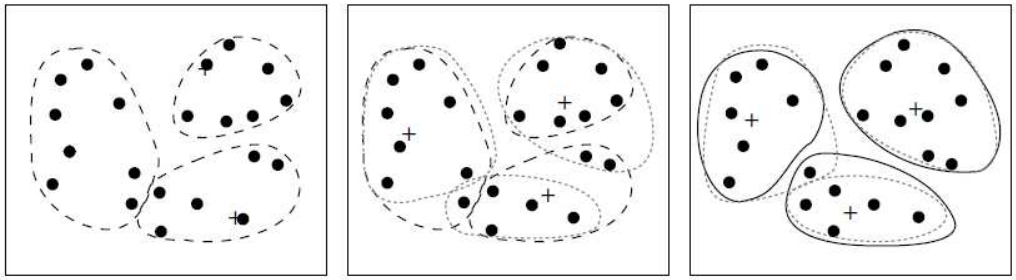

Fig. 3.15. Clustering the sample points by using the k-means algorithm.

The iteration steps of the k-means algorithm wont end until there is no change for cluster centers or the change can be ignored. In mathematical logic, it means that the algorithm will converge. As it was mentioned above, the end condition of the algorithm in the previous section is convergence. Therefore, it can prove that the k-means algorithm can ensure the convergence. Next, for figuring out the convergence, distortion function will be defined as follows:

$$J(c, \mu) = \sum_{i=1}^{m} \|x(i) - \mu_{c^{(i)}}\|^2 \tag{3.11}$$

The function J means the sum of squares of the distances between each sample points

and their cluster centers, so the purpose of the k-means algorithm is to minimize the function J. Assume that before the function J reaches its minimum, the cluster center μ_j of each cluster cant be changed and the cluster $c^{(i)}$ that the sample points belong to can adjusted for the reduction of the function J. In the same manner, the cluster $c^{(i)}$ that the sample points belong to can be fixed and the cluster center μ_j of each cluster can be adjusted for the reduction of the function J. These two processes can make sure that the function J will monotonically decrease within the loop. When the function J achieve to its minimum, μ and c also converge. Theoretically, there are many different pairs of μ and c that make the function J reach its minimum, but they are rare practically.

Since the distortion function J is a non-convex function, it means that it isnt guaranteed that the obtained minimum value is the global minimum. Therefore, it is really picky and tricky how the initial cluster centers of the k-means algorithm are chosen, but generally the local minimum, achieved by the k-means algorithm, is enough to meet the requirement. If the local minimum doesnt meet the requirement, the different initial value of the k-means algorithm can be chosen and find out the corresponding μ and c of the minimum J among the different cases.

3.4 Support Vector Machine (SVM)

All the support vector machine are based on a linear classifier, but not all the data can be linearly separated. For example, in a two-dimensional plane, all the points that belongs to two classifications can be separated by a curve. Therefore, all the points in a low-dimensional space can be mapped into high-dimensional space by using SVM, and in the high-dimensional space, all the points are now linearly separated. The boundary of two classification can be determined by a linear classifier now. In a high-dimensional space, it is a linear classifier; in the original dimensional space, it is

a non-linear classifier.

The algorithm of mapping from a low-dimensional space to a high-dimensional space is not the key to SVM, and these algorithms are given the kernel functions, which includes Polynomial kernel of degree h, Gaussian radial basis function kernel and Sigmold kernel. SVM is considered as an optimization problem, which means to find out the optimal solution.

Assume that the problem is considered as a function $f(x)$. To find out the optimal solution is to find out the minimum value of the function $f(x)$. Since $f(x)$ is not always continuously differentiable, the solution cant be found by finding the points where derivative equals to zero. The optimization problems can be divided into two categories: unconstrained optimization and constrained optimization. Unconstrained optimization can be expressed as

$$\min_{x} f(x) \tag{3.12}$$

, and constrained optimization can be expressed as

$$\begin{cases} min_x f(x) & x \in E^n \\ s.t.\ \varphi_i(x) \geq 0 & i \in \{1, 2, \ldots, m\} \end{cases} \tag{3.13}$$

3.5 Discussion

The procedure of this hybrid system is data preparation for each algorithm; determine the percentage of the significant objects by using algorithm for vegetation detection; find out the clusters or groups of the hyperspectral image by using the k-means algorithm; and finally generate the training model and find out the accuracy by using support vector machine. Next, some experimental results are shown according to this system architecture.

4. EXPERIMENTAL RESULTS

The hyperspectral image of the mud sample mentioned in Chapter 1 is used as the input data (raw data) of the hybrid system. The dataset is an 811-band 50-by-1600 hyperspectral image of a mud sample, whose pixel values are band interleaved by line (BIL) in order of increasing band number. BIL is one of three primary methods for encoding multiband image data, which has the advantage of accessing the spatial and spectral data easily.

4.1 Apply Vegetation Detection Algorithm

To find out the three most informative bands is the first step. The MATLAB function multibandread is one of the most important functions to read in the hyperspectral image. Therefore, an RGB image is imported into MATLAB, which red means the NIR band, green means the visible red band and blue means the visible green band. Figure 4.16 and 4.17 show the color infrared image.

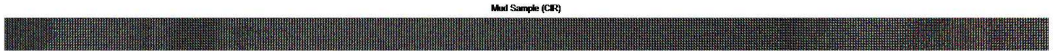

Fig. 4.16. Entire color infrared (CIR).

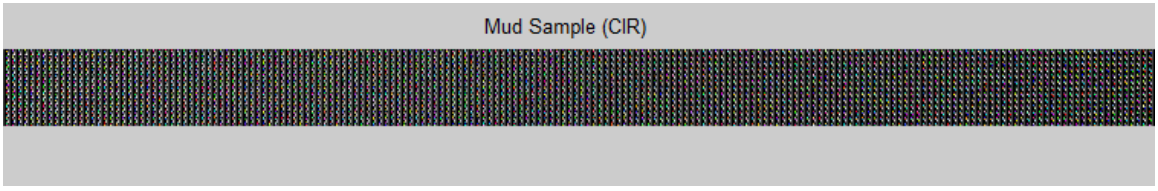

Fig. 4.17. Zoom in of CIR.

Decorrelation stretch is applied in the previous CIR image for better display. Decorrelation stretch can highlight the elements in a multichannel image by exaggerating the color differences. It is pretty wise to use option Tol in decrrelation stretch,

which can linearly contrast stretch. Tol = [LOW_FRACT HIGH_FRACT] specifies the fraction of the image to saturate at low and high intensities. Therefore, Tol is specified as 0.01, saturating at low and high intensities by one percent. Figure 4.18 and 4.19 show the color infrared image with decorrelation stretch.

Fig. 4.18. Entire color infrared (CIR) with decorrelation stretch.

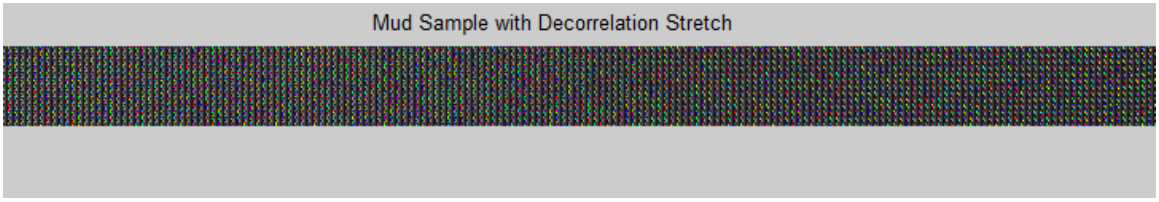

Fig. 4.19. Zoom in of CIR with decorrelation stretch.

An NIR-Red scatter plot is constructed to help us compare the NIR band (displayed as red) and the visible red band (displayed as green). As we said above, we can easily extract the NIR and red bands from the CIR composite. So we can see the difference between these two bands in grayscale images. Figure 4.20, 4.21, 4.22 and 4.23 show the near infrared band image and the visible red band image.

Fig. 4.20. Entire near infrared (NIR) band.

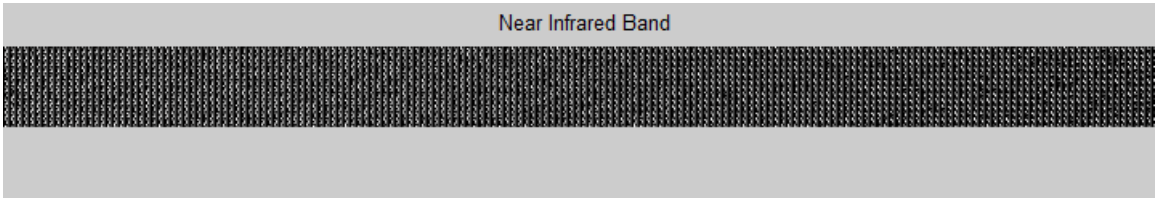

Fig. 4.21. Zoom in of NIR band.

Fig. 4.22. Entire visible red band.

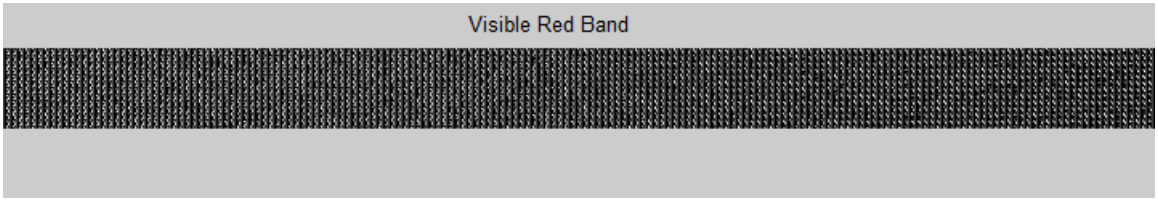

Fig. 4.23. Zoom in of visible red band.

Using the MATLAB function plot, we can easily have the scatter plot of each pixel with red level vs NIR level. For the purpose of the further computation, we scale the red level and NIR level values from 0 to 1. Figure 4.24 shows the scatter plot of visible red band and NIR band.

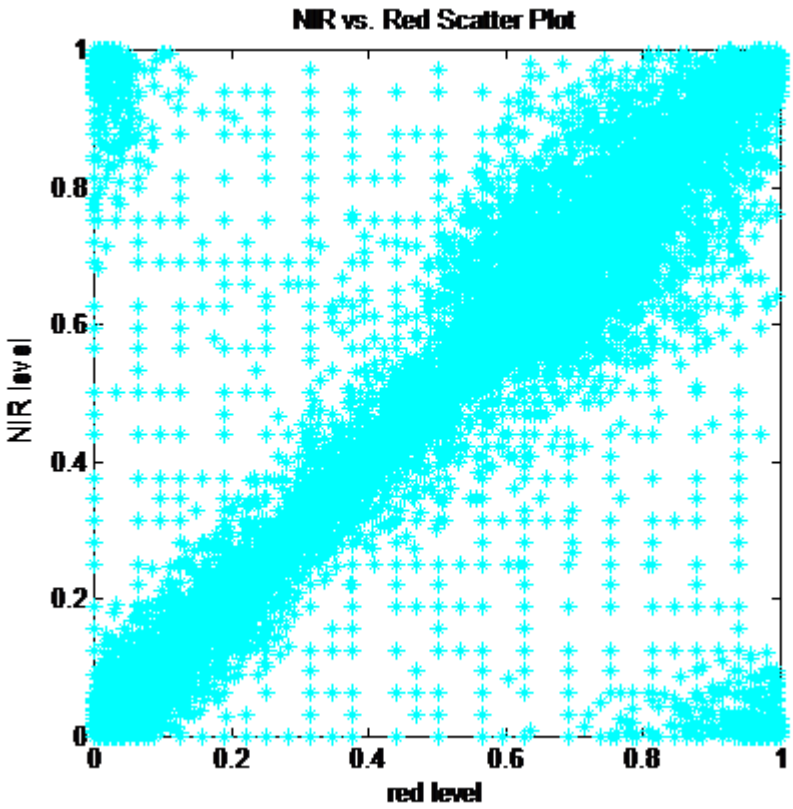

Fig. 4.24. NIR band vs visible red band.

From the scatter plot, most of pixels are along the diagonal, which means the NIR and red values are nearly the same. Also, we have some pixels at corners, which are the significant objects or pixels we are looking for.

To find out the significant objects, we have to compute the index of each pixel:

$$\text{Index} = (\text{NIR - red}) \,./\, (\text{NIR + red}) \tag{4.1}$$

From this equation, if index approaches to 0, pixels are along the diagonal; if index approaches to 1, pixels are at top left corner; if index approaches to -1, pixels are at bottom right corner. Figure 4.25 and 4.26 show the normalized difference image by using the index equation.

Fig. 4.25. Normalized difference image by using the index equation.

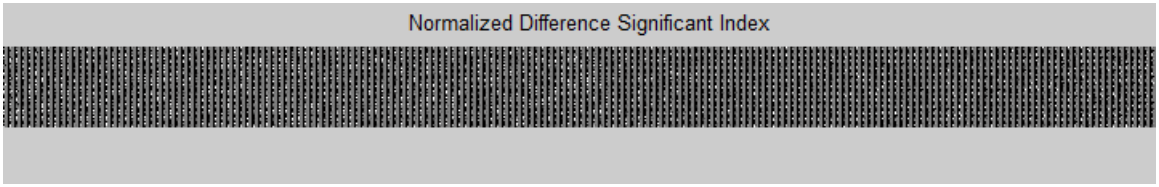

Fig. 4.26. Zoom in of normalized difference image by using the index equation.

In order to identify pixels most likely to contain significant objects, we apply a simple threshold to the previous image. We just need to create the significant objects image that have index greater than threshold. Figure 4.27 and 4.28 show the index image with threshold applied.

Fig. 4.27. Index image with threshold applied.

Fig. 4.28. Zoom in of index image with threshold applied.

Finally, we re-draw the useful scatter plot and the index image with threshold applied for better display. Figure 4.29 and 4.30 show the colored scatter plot and index image with threshold applied.

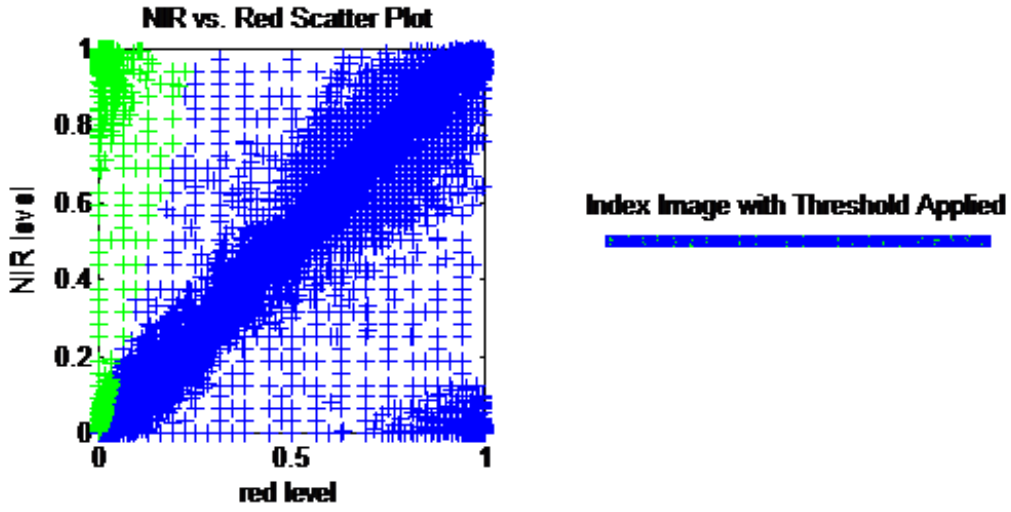

Fig. 4.29. Scatter Plot of NIR vs Red band and Index image with threshold applied with color. The green part is the significant objects; the blue part is the mud.

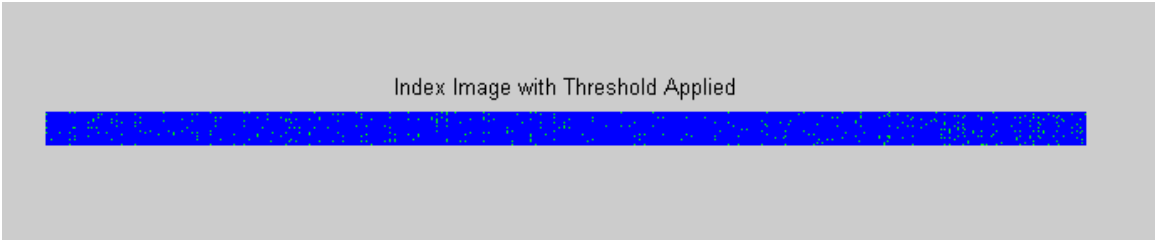

Fig. 4.30. Zoom in of Index image with threshold applied with color.

Percentage of significant objects is calculated by ratio of number of while pixels in binary image to total number of pixels in NIR image. It was found to be 3.5438%.

4.2 Apply the k-means Algorithm

The dataset here is the same as the one before. However, the hyperspectral image is read in single precision, and mathematically, the dataset is stored in two dimensional by having each column display the entire pixel values of each layer or band. Since

the second method of how to determine k and the percentage of significant objects is used, here is the result of the k-means algorithm as follows: (start from $k = 2$)

If $k = 2$, it means that the result only has two clusters. The measured time for running the model is 187.12 seconds. The hyperspectral image is divided into two clusters, one includes 34640 (43%) and the other includes 45360 (57%). If we see the original photo, we can see the tapes along the sides of the picture. Its reasonable that the 43% is the tape and that the 57% is the mud sample. Since the sum of squared errors of $k = 2$ is 731482.1304914309, the mud sample definitely can be divided into more clusters. Figure 4.31 shows the k-means algorithm with $k = 2$.

Fig. 4.31. The k-means algorithm with k=2.

If $k = 3$, it means that the result only has three clusters. The measured time for running the model is 703.78 seconds, and it is significantly increased from $k = 2$. The hyperspectral image is divided into three clusters, 33441 (42%), 11029 (14%)

and 35530 (44%). Its still reasonable that the 42% is the tape and that the rest 58% is the mud sample with one more cluster. Figure 4.32 shows the k-means algorithm with $k = 3$.

Fig. 4.32. The k-means algorithm with k=3.

If $k = 4$, it means that the result only has four clusters. The measured time for running the model is 723.3 seconds. The hyperspectral image is divided into four clusters, 33132 (41%), 11105 (14%), 31009 (39%) and 4754 (6%). Its still reasonable that the 41% is the tape and that the rest 59% is the mud sample with two more clusters. Its noticed that the 6% cluster may be have some association with the significant objects and that the sum of squared errors of $k = 4$ is 138644.72818111547. Although the sum of squared errors is significantly reduced compared to $k = 2$, it isnt stable yet. Figure 4.34 shows the k-means algorithm with $k = 4$.

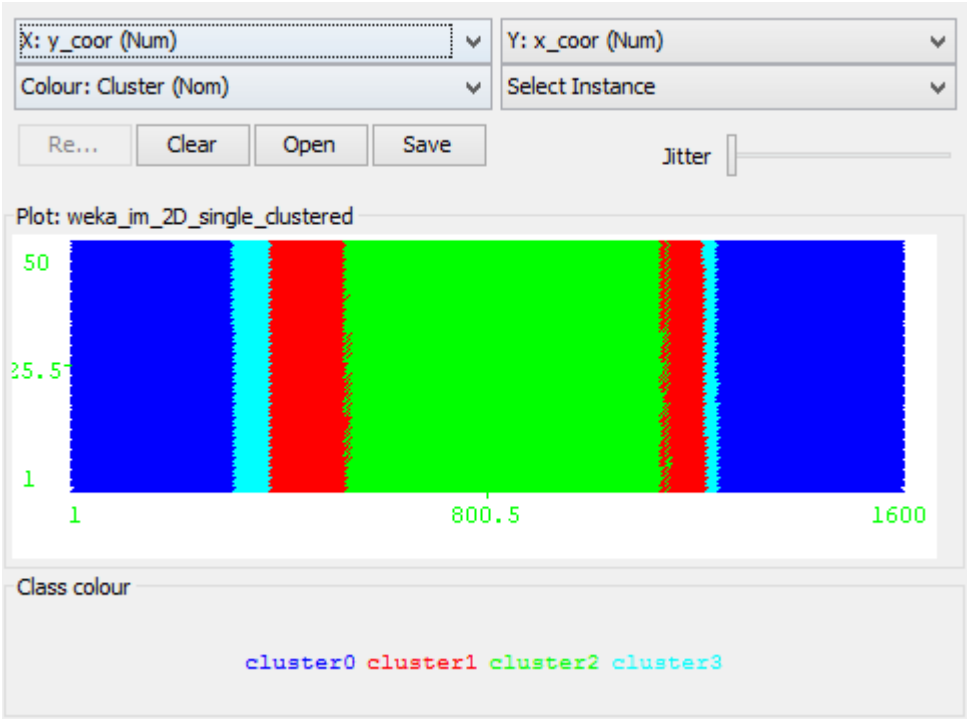

Fig. 4.33. The k-means algorithm with k=4.

The measured time for running the model of $k = 8$ is 1053.1 seconds. The hyperspectral image is divided into eight clusters, 12882 (16%), 4537 (6%), 6740 (8%), 5238 (7%), 9517 (12%), 23895 (30%), 2237 (3%) and 14954(19%). Its noticed that the 3% cluster is below the percentage of the significant objects we got from the last section and that may be have some association with the significant objects. But the sum of squared errors of $k = 8$ is 61049.33552500921, and it isnt stable yet. Figure **??** shows the k-means algorithm with $k = 8$.

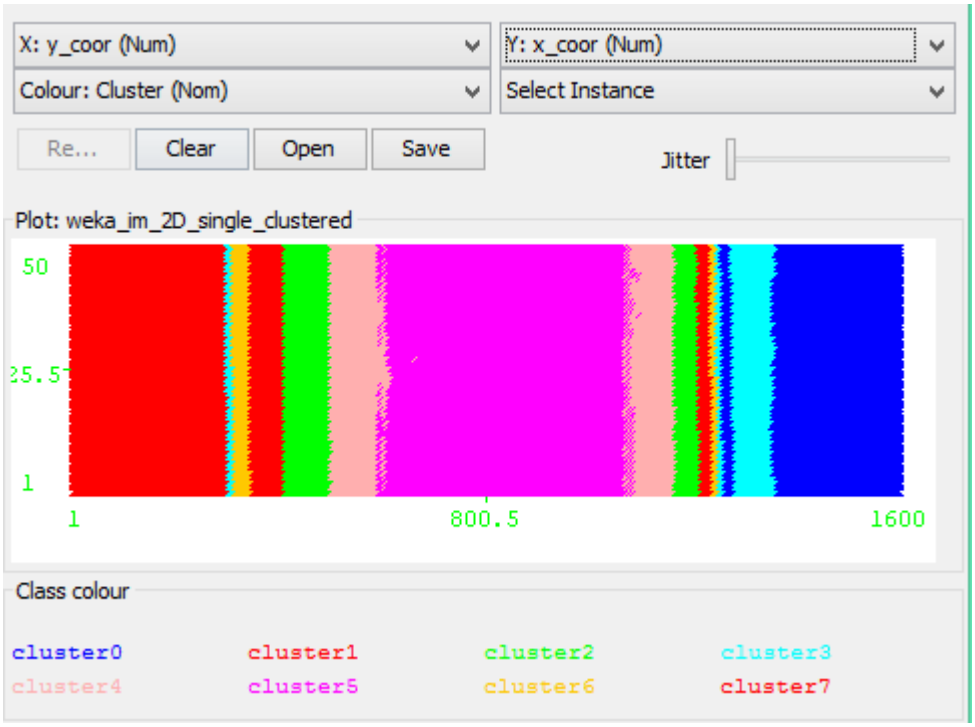

Fig. 4.34. The k-means algorithm with k=8.

The measured time for running the model of $k = 12$ is 1066.19 seconds. The hyperspectral image is divided into eight clusters, 6491 (8%), 5978 (7%), 6910 (9%), 5114 (6%), 10693 (13%), 15891 (20%), 1202 (2%), 7449 (9%), 2445 (3%), 3861(5%), 7509 (9%) and 6457 (8%). Its noticed that some clusters are below the percentage of the significant objects we got from the last section and that may be have some association with the significant objects. But the sum of squared errors of $k = 12$ is 40940.46539080349, and still it isnt stable yet. Figure **??** shows the k-means algorithm with $k = 12$.

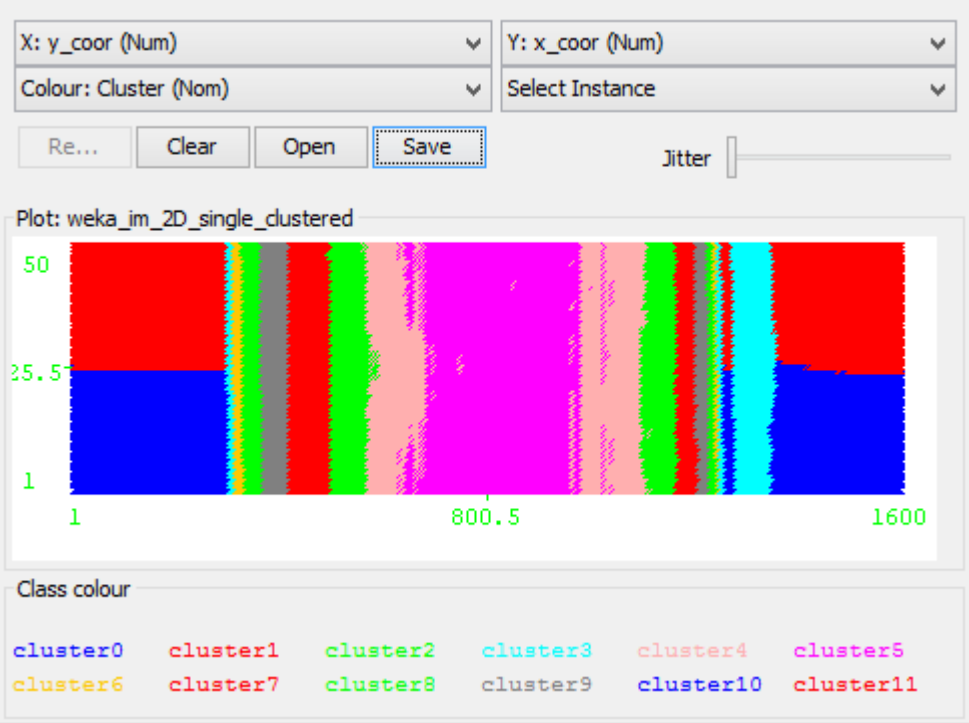

Fig. 4.35. The k-means algorithm with k=12.

The measured time for running the model of $k = 16$ is 1454.78 seconds. The hyperspectral image is divided into eight clusters, 6370 (8%), 3707 (5%), 4405 (6%), 712 (1%), 9822 (12%), 14089 (18%), 1339 (2%), 2784 (3%), 1954 (2%), 2856(4%), 5352 (6%), 5169 (6%), 6457 (8%), 5055 (6%), 6855 (9%) and 3074 (4%). Its noticed that some clusters are below the percentage of the significant objects we got from the last section and that may be have some association with the significant objects. The sum of Squared Error of 16 clusters: 30656.364971050734, and still it is stable. Figure 4.36 shows the k-means algorithm with $k = 16$.

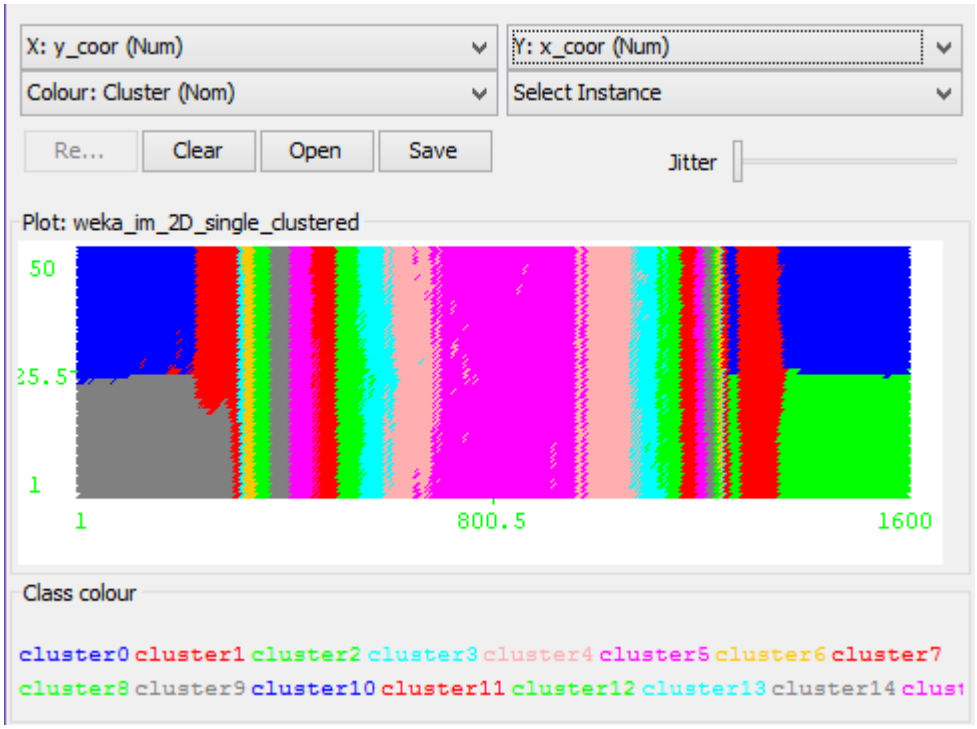

Fig. 4.36. The k-means algorithm with k=16.

The sum of squared errors of $k = 20$ is 28912.487844715703, and the sum of squared error of 24 clusters is 26249.48621819265. Figure 4.37 shows curve fitting of the number of clusters versus the sum of squared error. According to the curve fitting, $k = 16$ has the least sum of squared error.

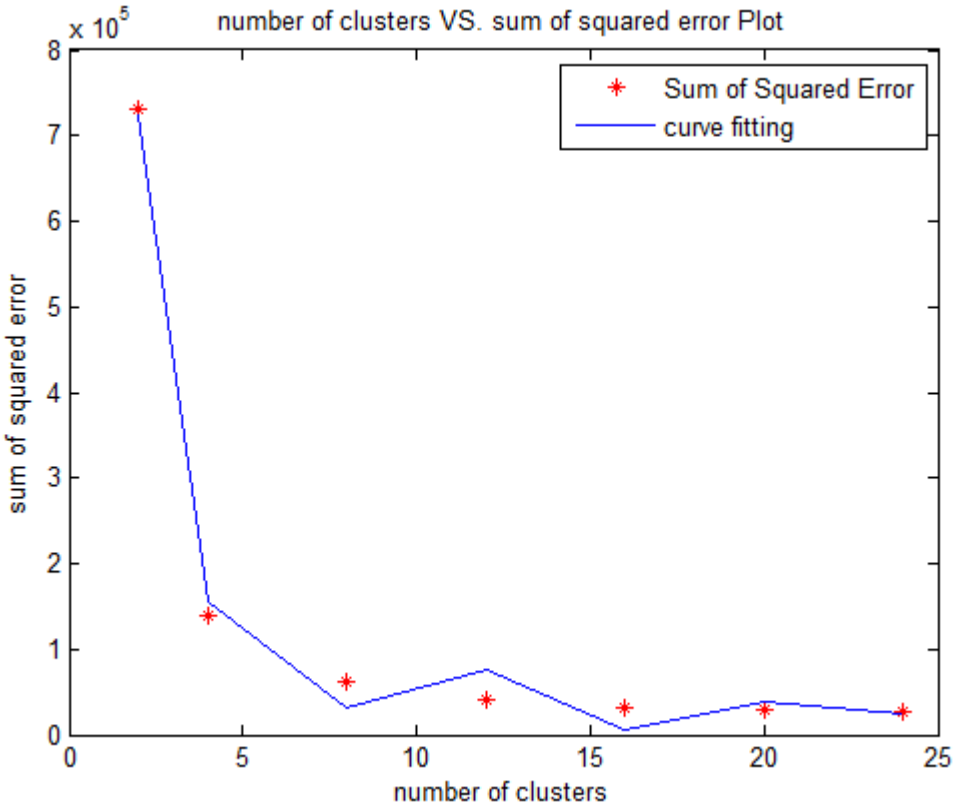

Fig. 4.37. The curve fitting for the sum of squared error.

4.3 Apply Support Vector Machine

According to the result of the previous section, since the hyperspectral image is partitioned into 16 classifications, this problem is considered as multi-class classification. The original SVM from MATLAB can only support two classification, so LIBSVM must be imported. LIBSVM is a library for support vector machines [9], and it supports regression, distribution estimation and even multi-class classification. LIBSVM is available in many programming languages, such as Python, R and MATLAB etc.

The output dataset is generated from the k-means algorithm, and its now labelled with the class number. Two hyperspectral images are classified into 16 groups

or classes, and tested by using 10-fold cross-validation. It means that total 160000 sample points are randomly divided into 10 sub-samples. The 9 out of 10 sub-samples are randomly chosen as the training dataset, and the rest is chosen as our testing dataset. Therefore, that 9 out of 10 sub-samples are now used as the input dataset for LIBSVM. Figure 4.38 and 4.39 are the two hyperspectral images with 16 classifications.

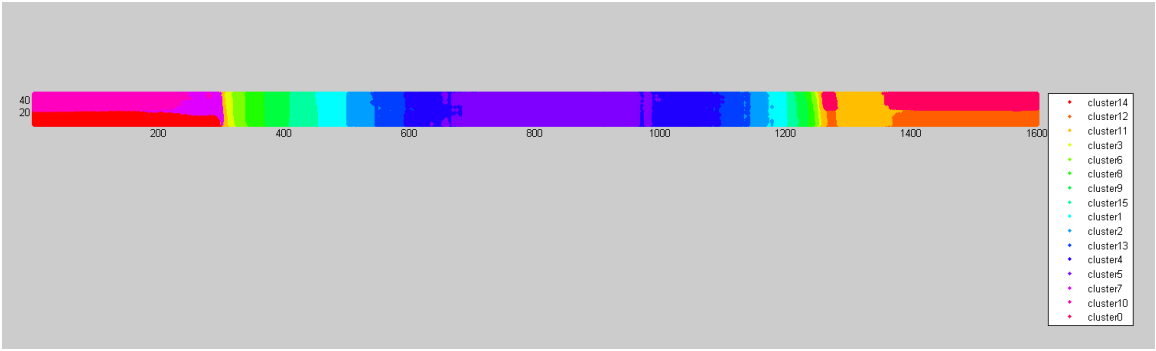

Fig. 4.38. 16 classifications of Sample 1.

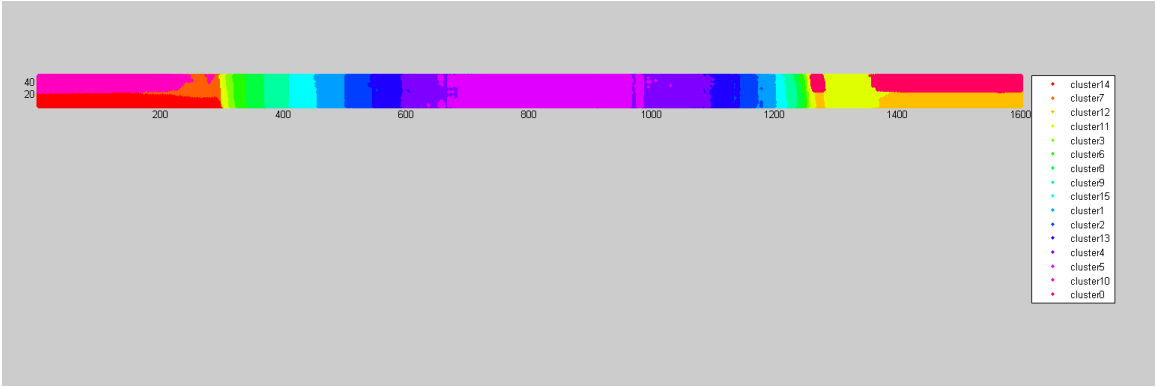

Fig. 4.39. 16 classifications of Sample 2.

It only takes two steps to get the accuracy of the testing data [10]. The first step is to train your prepared dataset which is the random 9 out of 10 sub-samples. The next step is to test the rest sub-sample. Since the test mode is 10-fold cross-validation,

there are total 10 sub-samples can be used as the test dataset. The average running time of training the sub-samples is about 35 hours, and the average running time of testing the sub-sample is about 30 mins. The three accuracies so far are 96.8875%, 96.3188% and 97.5875%, so the average accuracy is 96.9313%.

5. DISCUSSION AND CONCLUSIONS

This Chapter discusses the findings in previous chapters and suggests the best strategies to do the unsupervised learning of hyperspectral images. Also, the limitations of the study and possible directions for future research are discussed.

5.1 Discussion of algorithms

The vegetation detection algorithm is adapted to the hyperspectral image of mud samples to determine the percentage of significant objects. The percentage of significant objects is easy and quick to get by using this algorithm. Since the three most informative bands are chosen for process, the computation is reduced even for large data sets. However, this can also be the disadvantage of this algorithm if there are more than three bands having the same information. If this happens, the three bands should be randomly chosen from those bands having the same information.

The k-means algorithm is used for partitioning the hyperspectral image with the pre-defined k. If a dataset is dense and clear, the result of the k-mean algorithm is better. According to the running time in the previous chapter, this algorithm is relatively efficient and scalable for large data sets. The sum of squared error is a very good standard to determine the value of k. Since k is user-defined and must be determined at first, finding out the value of k may take more time than just processing the dataset by the k-means algorithm. Iterations is processed according to the initial clustering, so the computation is costly comparing to the vegetation detection algorithm.

Support vector machine (SVM) is used to train the data and predict another data for better accuracy by using 10-fold cross-validation. SVM can use the three kernel function instead of non-linear mapping, which can save amount of running time. The problem can be treated as finding the optimal solution in SVM, so Non-linear

data can be solved by finding the optimal hyper-plane in a high dimensional space. However, since SVM belongs to supervised learning, the dataset must be labelled and the computation is more costly than the k-means algorithm for large data sets.

5.2 Conclustions

Since we determine the percentage of the significant objects by using algorithm for vegetation detection, we have set the bar for the upcoming data mining technique, the k-means algorithm. When we find the groups whose percentage is less than 3.5

$k = 16$ meets the requirements of having the least sum of squared errors and stable structure. Therefore, $k = 16$ is chosen for the k-means algorithm for labeling each pixel in the hyperspectral image. After the dataset is stuffed with cluster label, a supervised learning technique - support venter machine (SVM) will be used for training and testing the dataset with high accuracy.

The two kernel functions of support vector machine we used are the Gaussian radial basis function and linear kernel. The Gaussian radial basis function kernel runs faster than the linear kernel, but the accuracy of the Gaussian radial basis function kernel is way less than the one of the linear kernel.

By combining these methods above, results of accuracy are showed in Chapter Four. We are going to compare our result with the results in other articles [11][12]. The first article is to do a fast support vector machine classification of very large dataset by decomposing the original problem into linear subproblems. The second article is to improve the accuracy by manually tuning the parameters of SVM. Comparing with these results, the accuracy of the linear kernel in our system has its advantage, shown in Table 5.1.

Kernel Functions	Our Result	Result of Article 1	Result of Article 2
The linear kernel	96.9313%	95.69%	96.4%

Table 5.1. Comparing results with other articles

5.3 Limitations

Since we only have two hyperspectral images, the number of samples may not be enough. Therefore, 10-fold cross-validation is used as the test mode. More samples of hyperspectral images will improve the accuracy. The running time is more concerned as this system runs. A 64-bit operating system is a must, and there is no way you can run this kind of large datasets in a 32-bit OS because the minimum heap size of processing the dataset for memory is 6 GB.

5.4 Future Works

It will be desirable to include in future research some additional models, such as Hierarchical Methods, Bayesian Belief Networks and Probabilistic Model-Based Clustering. Adding more pre-processing or post-processing step is better to improve the accuracy, but more costly. Also, the hyperspectral images can be expanded to include additional factors that may be able to identify the objects with single pixel. The mud sample in this study is just a tool to demonstrate the hybrid system can have a better accuracy. The mud sample can be replaced by any other kinds of hyperspectral images, and even this hybrid system can be extended into other fields required for clustering or classifications.

BIBLIOGRAPHY AND REFERENCES

[1] Landarebe, D. Hyperspectral image data analysis. *IEEE Signal Processing Magazine*(2002), 17-28.

[2] Li, Z., Li, L., Zhang, R., and Ma, J. An improved classification method for hyperspectral data based on spectral and morphological information. *International Journal of Remote Sensing*(2011), vol.32, 2919-2929.

[3] Gamps-Valls, G., Tuia, D., Bruzzone, L., and Atli Benediktsson, J. Advances in Hyperspectral Image Classification: Earth monitoring with statistical learning methods. *Signal Processing Magazein, IEEE Journal* (2014), 1007-1011.

[4] Starr, C. *Biology: Concepts and Application*, Thomson Brooks/Cole, 2005.

[5] Han, J., Kamber, M., and Pei, J. *Data Mining: Concepts and Techniques.*, Morgan Kaufmann Publishers, 2011.

[6] Mercier, G., and Lennon, M. Support vector machines for hyperspectral image classification with spectral-based kernels *Geoscience and Remote Sensing Symposium*(2003), vol.1, 288-290.

[7] Chouhan, R., and Rao, N. Vegetation Detection in Multi Spectral Remote Sensing Images: Protective Role-analysis of Vegetation in 2004 Indian Ocean Tsunami. PDPM Indian Institute of Information Technology, 2011.

[8] Witten, I., Frank, E., and Hall, M. *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Publishers, 2011.

[9] Chang, C., and Lin, C. LIVSVM: a library for support vector machines *ACM Transactions on Intelligent Systems and Technology*(2011), vol.2, 1-27.

[10] Hsu, C., Chang, C., and Lin, C. A paractical guide to support vector classification *PDF Online*, May, 2009.

[11] Fehr, J., Arreola, K., and Burkhardt, H. Fast Support Vector Machine Classification of Very Large Datasets *Studies in Classification, Data Analysis, and Knowledge Organization*(2008), 11-18.

[12] McCue, R. A Comparison of the Accuracy of Support Vector Machine and Na?ve Bayes Algorithms in Spam Classification, University of California at Santa Cruz, Nov,2009.

APPENDIX A – CODE FOR VEGETATION DETECTION ALGORITHM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  First of all, read the hyperspectral image that you want to plot with
%  unin8. Converter the three dimensional dataset into a two
%  dimensional dataset with the band attribute and the pixel instance.
%  Finally, write the two dimensional dataset into a csv format file.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clc
close all
clear all

%% read the hyperspectral image
% Input the image that you want to plot
% fn = 'Reflec_Seagrass3_C_FA3_SB1_SP100_SL7_SS100_IT500_1';
fn = '121206_MudSample_EndOfFall2012/Reflec_FA3_SB1_SP100_SL5_SS100_IT500mS_Spectral

%Check the header file and locate the following information. In this case,
samples = 1600;
lines = 50;
bands = 811;
% read the multi-band image
im = multibandread(fn, [lines,samples,bands],'uint8=>uint8',0,'bil','ieee-le');

%% re-write the 3D matrix into the 2D matrix
% define the new index of 2D matrix as count
count = 1;
% define the 2D matrix
im_2D = zeros(80000,813);
% convert the 3D matrix into 2D matrix with x and y coordinates
for k = 1:bands
    for i = 1:lines
        for j = 1:samples
            im_2D(count,k) = im(i,j,k);
            im_2D(count,812) = i;
            im_2D(count,813) = j;
            count = count + 1;
        end
    end
    count = 1;
end

%% write a attribute name matrix for header
%attribute name matrix
attribute_name = cell(10,1);
% attribute name matrix in 'bandX' format
for i = 1:bands
    con_i = num2str(i);
```

```
    attribute_name{i} = ['band' con_i];
end

%% write the content into the csv file for weka
%write into csv format
fid = fopen('im_2D.csv','w');
for row = 1:bands
    fprintf(fid, repmat('%s\t',size(attribute_name,2)-1,1), attribute_name{row,1:enc
    fprintf(fid, '%s,', attribute_name{row,end});
end
fclose(fid);

% start to write the content into csv file
csvwrite('im_2D.csv',im_2D,1,0);
% show content in matlab
%type im_2D.csv;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  compute 0s in each column in im_2D.csv to find the index of 3 bands
%  has the least 0s
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create a zero matrix to store how many 0s in each column
zero_info = zeros(1,bands);

%use for loop to start computer 0s
for i = 1:samples*lines
    for j = 1:bands
        if (im_2D(i,j) == 0)
            zero_info(j) = zero_info(j)+1;
        end
    end
end

%sort the zero_info to find the minimum 0s
sorted_zero = sort(zero_info);

%find the index of 3 bands has the least 0s
a = find(zero_info < 28000)



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Apply the algorithm for our case
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clc
close all
```

```
clear all

%%
% Input the image that you want to plot
% fn = 'Reflec_Seagrass3_C_FA3_SB1_SP100_SL7_SS100_IT500_1';
fn = '121206_MudSample_EndOfFall2012/Reflec_FA3_SB1_SP100_SL5_SS100_IT500mS_Spectral

%Check the header file and locate the following information. In this case,
samples = 1600;
lines = 50;
bands = 811;

%% Step 1: Import CIR Bands from a BIL Image File
% read the hyperspectral image
im_read = multibandread(fn, [lines,samples,bands],'uint8=>uint8',0,'bil','ieee-le',

figure;
imshow(im_read);
%axis image;
title('Mud Sample (CIR)')

%% Step 2: Enhance the CIR Composite with a Decorrelation Stretch
% one percent
decorr_im = decorrstretch(im_read, 'Tol', 0.01);
figure;
imshow(decorr_im);
title('Mud Sample with Decorrelation Stretch')

%% Step 3: Construct an NIR-Red Spectral Scatter Plot
% find NIR and red
NIR = im2single(im_read(:,:,1));
red = im2single(im_read(:,:,2));

% compare in grayscale image
figure
imshow(red)
title('Visible Red Band')
figure
imshow(NIR)
title('Near Infrared Band')

% scatter plot in [0,1]
figure
plot(red, NIR, 'c*')
set(gca, 'XLim', [0 1], 'XTick', 0:0.2:1,...
         'YLim', [0 1], 'YTick', 0:0.2:1);
axis square
xlabel('red level')
ylabel('NIR level')
title('NIR vs. Red Scatter Plot')
```

```
%% Step 4: Compute algae Index via MATLAB Array Arithmetic
indexx = (NIR - red) ./ (NIR + red);

figure
imshow(indexx,'DisplayRange',[-1 1])
title('Normalized Difference Significant Index')

%% Step 5: Locate algae -- Threshold the index Image
% threshold set up
threshold = 0.6;
q = (indexx > threshold);

% precentage calculation
100 * numel(NIR(q(:))) / numel(NIR)

figure
imshow(q)
title('Index Image with Threshold Applied')

%% Step 6: Link Spectral and Spatial Content
%Create a figure with a 1-by-2 aspect ratio
h = figure;
p = get(h,'Position');
set(h,'Position',[p(1,1:3),p(3)/2])
subplot(1,2,1)
% Create the scatter plot
plot(red, NIR, '+b')
hold on
plot(red(q(:)), NIR(q(:)), 'g+')
set(gca, 'XLim', [0 1], 'YLim', [0 1])
axis square
xlabel('red level')
ylabel('NIR level')
title('NIR vs. Red Scatter Plot')
% Display the thresholded index image
subplot(1,2,2)
imshow(q)
set(h,'Colormap',[0 0 1; 0 1 0])
title('Index Image with Threshold Applied')
```

APPENDIX B – CODE FOR THE K-MEANS ALGORITHM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  First of all, read the hyperspectral image that you want to plot with
%  single percision. Converter the three dimensional dataset into a two
%  dimensional dataset with the band attribute and the pixel instance.
%  Finally, write the two dimensional dataset into a csv format file.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clc
close all
clear all

%% read the hyperspectral image
% Input the image that you want to plot
% fn = 'Reflec_Seagrass3_C_FA3_SB1_SP100_SL7_SS100_IT500_1';
fn = '121206_MudSample_EndOfFall2012/Reflec_FA3_SB1_SP100_SL5_SS100_IT500mS_Spectral

% Check the header file and locate the following information. In this case,
samples = 1600;
lines = 50;
bands = 811;
% read the multi-band image
im = multibandread(fn, [lines,samples,bands],'single',0,'bil','ieee-le');

%% re-write the 3D matrix into the 2D matrix
% define the 2D matrix
im_2D = zeros(80000,813);
% define the new index of 2D matrix as count
count = 1;
% convert the 3D matrix into 2D matrix with x and y coordinates
for k = 1:bands
    for i = 1:lines
        for j = 1:samples
            im_2D(count,k) = im(i,j,k);
            im_2D(count,812) = i;
            im_2D(count,813) = j;
            count = count + 1;
        end
    end
    count = 1;
end


%% write a attribute name matrix for header
% define the attribute name matrix
attribute_name = cell(10,1);
% attribute name matrix in 'bandX' format
for i = 1:bands
```

```
    con_i = num2str(i);
    attribute_name{i} = ['band' con_i];
end

%% write the content into the csv file for weka
% open the csv file for writing
fid = fopen('im_2D_single.csv','w');
% print the each row into file
for row = 1:bands
    fprintf(fid, repmat('%s\t',size(attribute_name,2)-1,1), attribute_name{row,1:enc
    fprintf(fid, '%s,', attribute_name{row,end});
end
fclose(fid);
% start to write the content into csv file
csvwrite('im_2D_single.csv',im_2D,1,0);
% show content in matlab
%type im_2D_single.csv;


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// convert csv file into arff file for weka in java
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.CSVLoader;

import java.io.File;

public class csvTOarff {

  public static void main(String[] args) throws Exception {

// check whether we have 2 args
    if (args.length != 2) {
      System.out.println("\nUsage: csvTOarff <input.csv> <output.arff>\n");
      System.exit(1);
    }

    // load CSV
    CSVLoader loader = new CSVLoader();
    loader.setSource(new File(args[0]));
    Instances data = loader.getDataSet();

    // save ARFF
    ArffSaver saver = new ArffSaver();
    saver.setInstances(data);
    saver.setFile(new File(args[1]));
    //saver.setDestination(new File(args[1]));
    saver.writeBatch();
  }
}
```

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// check files permission in java
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
import java.io.File;
import java.io.IOException;

public class file_permission {

    public static void main( String[] args )
    {
     try {

      File file = new File("D:/University/TAMUCC/COMPUTER/DataMining_Thesis/CODE/wek

      if(file.exists()){
       System.out.println("Is Execute allow : " + file.canExecute());
       System.out.println("Is Write allow : " + file.canWrite());
       System.out.println("Is Read allow : " + file.canRead());
      }

      file.setExecutable(true);
      file.setReadable(true);
      file.setWritable(true);

      System.out.println("Is Execute allow : " + file.canExecute());
      System.out.println("Is Write allow : " + file.canWrite());
      System.out.println("Is Read allow : " + file.canRead());

      if (file.createNewFile()){
        System.out.println("File is created!");
      }else{
        System.out.println("File already exists.");
      }

     } catch (IOException ex) {
      ex.printStackTrace();
    }
    }
}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// the k-means algorithm in java
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.FileReader;
```

```
import java.io.FileWriter;
import java.io.PrintStream;
import java.text.DecimalFormat;
import java.text.NumberFormat;

import weka.clusterers.ClusterEvaluation;
import weka.clusterers.Clusterer;
import weka.clusterers.SimpleKMeans;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.EuclideanDistance;


public class Kmeans {

public static void main(String[] args) throws Exception {
// start time of program
long startTime = System.currentTimeMillis();
// Redirect output to file
System.setOut(new PrintStream(new FileOutputStream("output")));

// check whether we have 2 args
    if (args.length != 2) {
      System.out.println("\nUsage: Kmeans <input.arff> <number of clusters>\n");
      System.exit(1);
    }

// read the arff dataset
BufferedReader reader = null;
reader = new BufferedReader(new FileReader(args[0]));

Instances data = new Instances(reader);
reader.close();

// set the options for k means
String[] options = new String[2];
options[0] = "-N";
options[1] = args[1];

// build up the clusterer
SimpleKMeans kMeans = new SimpleKMeans();
kMeans.setOptions(options);
kMeans.setSeed(10);

// The important parameter to set
kMeans.setDisplayStdDevs(true);
kMeans.setPreserveInstancesOrder(true);
kMeans.getMaxIterations();
//kMeans.setNumClusters(2);
kMeans.buildClusterer(data);
```

```
Instances stand_dev = kMeans.getClusterStandardDevs();
Instances clus_centr = kMeans.getClusterCentroids();
int[] clus_size = kMeans.getClusterSizes();
double squared_error = kMeans.getSquaredError();


// Evaluation
ClusterEvaluation eval = new ClusterEvaluation();
//Clusterer clusterer = new SimpleKMeans();
//clusterer.setOptions(options);
//clusterer.buildClusterer(data);
eval.setClusterer(kMeans);
eval.evaluateClusterer(data);
System.out.println("# of clusters: " + eval.getNumClusters());

for (int i = 0; i < clus_centr.numInstances(); i++) {
System.out.println("Cluster#"+ (i +1) + ": " + clus_size[i] + " (" + 100*clus_size[i
System.out.println("Centride:"+ clus_centr.instance(i));
System.out.println("STDDEV:" + stand_dev.instance(i));
System.out.println("Cluster Evaluation:"+eval.clusterResultsToString());
}


// create a file to write
FileWriter outputstream = new FileWriter("clus_group.csv");
BufferedWriter output = new BufferedWriter(outputstream);

// create a file to write
FileWriter outputstream_num = new FileWriter("clus_group_num.csv");
BufferedWriter output_num = new BufferedWriter(outputstream_num);

// write attribute name to csv file
output.write("Cluster_Group");
    output.newLine();

    // write attribute name to csv file
  output_num.write("Cluster_Group");
  output_num.newLine();

// instance -> cluster
int[] assignments = kMeans.getAssignments();
int i=0;
for(int clusterNum : assignments) {
    System.out.printf("Instance %d -> Cluster %d \n", i+1, clusterNum);
    output.write("cluster"+clusterNum);
    output_num.write(""+clusterNum);
    output.newLine();
    output_num.newLine();
    i++;
}
```

```
    // close the output stream
  output.close();
output_num.close();

System.out.println("Squared Error:" + squared_error + "\n");

// end time of program
long endTime   = System.currentTimeMillis();
// format the time
NumberFormat formatter = new DecimalFormat("#0.00000");
// system out the total time
System.out.print("\nExecution time is " + formatter.format((endTime - startTime) / 1
}
}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Scatter plot of number of clusters and sum of squared error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y = [731482.1304914309, 138644.72818111547, 61049.33552500921, 40940.46539080349,306
x = [2,4,8,12,16,20,24];
A=polyfit(x,y,5);
z=polyval(A,x);
plot(x,y,'r*',x,z,'b')
xlabel('number of clusters');
ylabel('sum of squared error');
legend('Sum of Squared Error', 'curve fitting', 'Location','NorthEast');
title('number of clusters VS. sum of squared error Plot')
```

APPENDIX C – CODE FOR SUPPORT VECTOR MACHINE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  First of all, import the two dimensional dataset and the result of
%  cluster group from weka.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% running the profile
profile on

clear all
clc
close all

%% read the csv file of original 2D file
% define the files
filename1 = 'weka_im_2D_single_2.csv';
% set up the parameters for loading the dataset
delimiter = ',';
headerlines = 1;
% import the data of each pixel into M1
M1 = importdata(filename1, delimiter, headerlines);

%% read the csv file of the cluster group
% define the files
filename2 = 'clus_group_2.csv';
% open the input file
fileID = fopen(filename2);
% scan the header for cluster set
M2_head = textscan(fileID,'%s', 1, 'Delimiter', '\n', 'MultipleDelimsAsOne', true);
% calculate the number of columns in the cluster set
cols = numel(regexp(M2_head{1}{1}, '\s*\w+'));
% load the cluster set into M2
[M2] = textread(filename2, '%s');
% close the input file
fclose(fileID);

%% use gscatter to plot the clustered image
x = M1.data(:,812:813);
group = M2(2:end,1);
gscatter(x(:,2),x(:,1),group);
axis image;
legend('Location','NorthEastOutside');

%% train the dataset by svm (got an error here but dont worry)
% command out for running smoothly
%svmStruct = svmtrain(x,group,'showplot',true, 'kernel_function', 'rbf', 'rbf_sigma

%% read the csv file of the NEW cluster group
% define the file
```

```
filename3 = 'clus_group_num_2.csv';
% set up the parameters for loading the dataset
delimiter = ',';
headerlines = 1;
% load the cluster set into M3
M3 = importdata(filename3, delimiter, headerlines);

%% Parameters and randomly divide the sample
samples = 1600;
lines = 50;
bands = 811;

numofset = 5;
datanum = 1:80000;
newdata=datanum(randperm(80000));
newdataset=reshape(newdata,numel(datanum)/numofset,numofset);

%% write out to afterKmean with libsvm format
%% 1st random dataset
fid1 = fopen('afterKmeans_21', 'w');
for i = 1 : samples*lines/numofset
    fprintf(fid1, '%d ', M3.data(newdataset(i,1),1));
    for j  = 1: bands+2
        fprintf(fid1, '%d:%f ', j, M1.data(newdataset(i,1),j));
    end
    fprintf(fid1, '\n');
end

fclose(fid1);

%% 2nd random dataset
fid2 = fopen('afterKmeans_22', 'w');
for i = 1 : samples*lines/numofset
    fprintf(fid2, '%d ', M3.data(newdataset(i,2),1));
    for j  = 1: bands+2
        fprintf(fid2, '%d:%f ', j, M1.data(newdataset(i,2),j));
    end
    fprintf(fid2, '\n');
end

fclose(fid2);

%% 3rd random dataset
fid3 = fopen('afterKmeans_23', 'w');
for i = 1 : samples*lines/numofset
    fprintf(fid3, '%d ', M3.data(newdataset(i,3),1));
    for j  = 1: bands+2
        fprintf(fid3, '%d:%f ', j, M1.data(newdataset(i,3),j));
    end
    fprintf(fid3, '\n');
end
```

```
fclose(fid3);

%% 4th random dataset
fid4 = fopen('afterKmeans_24', 'w');
for i = 1 : samples*lines/numofset
    fprintf(fid4, '%d ', M3.data(newdataset(i,4),1));
    for j  = 1: bands+2
        fprintf(fid4, '%d:%f ', j, M1.data(newdataset(i,4),j));
    end
    fprintf(fid4, '\n');
end

fclose(fid4);

%% 5th random dataset
fid5 = fopen('afterKmeans_25', 'w');
for i = 1 : samples*lines/numofset
    fprintf(fid5, '%d ', M3.data(newdataset(i,5),1));
    for j  = 1: bands+2
        fprintf(fid5, '%d:%f ', j, M1.data(newdataset(i,5),j));
    end
    fprintf(fid5, '\n');
end

fclose(fid5);
%% turn off the profile and save it in html format
profile off
profsave(profile('info'),'myprofile_results')



clc
close all
clear all

profile on
% read in the dataset for training
[labels, dataset]  = libsvmread('afterKmeans');
% svm train with the RBF kernel
model = svmtrain(labels, dataset, '-c 1 -g 2');
% svm train with the linear kernel
model1 = svmtrain(labels, dataset, '-t 0 -c 1');

profile off
profsave(profile('info'),'myprofile')



profile on
```

```
% read in the dataset for training
[test_label, test_data]  = libsvmread('afterKmeans_test');

[predict_label, accuracy, dec_values] = svmpredict(test_label, test_data, model); %

profile off
profsave(profile('info'),'myprofile_test')
```