
Computer Science Theses

Computer Science

Spring 5-1-2015

The Random Forest Algorithm with Application to Multispectral Image Analysis

Barrett E. Lowe

Follow this and additional works at: https://scholarworks.uttyler.edu/compsci_grad



Part of the [Computer Sciences Commons](#)

Recommended Citation

Lowe, Barrett E., "The Random Forest Algorithm with Application to Multispectral Image Analysis" (2015).
Computer Science Theses. Paper 5.

<http://hdl.handle.net/10950/260>

This Thesis is brought to you for free and open access by the Computer Science at Scholar Works at UT Tyler. It has been accepted for inclusion in Computer Science Theses by an authorized administrator of Scholar Works at UT Tyler. For more information, please contact tgullings@uttyler.edu.

THE RANDOM FOREST ALGORITHM
WITH APPLICATION TO MULTISPECTRAL IMAGE ANALYSIS

by

BARRETT E. LOWE

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science

Arun D. Kulkarni, Ph.D., Committee Chair

College of Business and Technology

The University of Texas at Tyler
May 2015

The University of Texas at Tyler
Tyler, Texas

This is to certify that the Master's Thesis of

BARRETT E. LOWE

has been approved for the thesis requirement on
26 March, 2015
for the Master of Science in Computer Science

Approvals:




Thesis/Dissertation Chair: Arun D. Kulkarni, Ph.D.



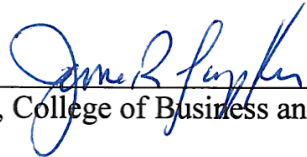
Member: Leonard Brown, Ph.D.



Member: Stephen Rainwater, Ed.D.



Chair, Department of Computer Science



Dean, College of Business and Technology

© Copyright 2015 by Barrett Lowe
All rights reserved.

Dedication

For Sawyer

Acknowledgements

I would like to thank Dr. Kulkarni, committee members, and the Department of Computer Science. Without encouragement and support from them, this document would have far fewer pages and, although I would've gotten more sleep over the past two years, I would not have the knowledge, forethought, understanding, or drive that were behind the writing of this thesis. Thank you.

Table of Contents

List of Tables	iii
List of Figures	iv
Abstract	vi
Chapter 1 Introduction	1
Chapter 2 Literature Review	4
2.1 Remote Sensing	4
2.2 Supervised Classifiers	8
2.2.1 Minimum Distance	9
2.2.2 K-Nearest Neighbor	10
2.2.3 Naïve Bayes	11
2.2.4 Neural Network	12
2.2.5 Support Vector Machine	15
Chapter 3 Methodology	20
3.1 Tree Classifiers	20
3.2 ID3 Tree	27
3.3 Random Forest	28
3.3.1 Supervised	28
3.3.2 Unsupervised	30
3.3.3 Synthetic Data	31
3.3.4 Variable Importance	32
3.4 Accuracy Assessment	32
Chapter 4 Implementation and Results	35
4.1 Iris dataset	35
4.2 Remotely Sensed Data	37
4.2.1 Yellowstone Scene	37
4.2.2 Mississippi Scene	42
Chapter 5 Conclusions and Future Work	46
References	50
Appendix A: Neural Network for Iris Data in R	56

Appendix B: Iris Data Classifiers	57
Appendix C: Random Forest and ID3 for Yellowstone Scene in R	58
Appendix D: SVM for Yellowstone Scene in R.....	60
Appendix E: Neural Network for Yellowstone Scene in R	61
Appendix F: Processing and Classification in ERDAS 2014	63

List of Tables

Table 1. Landsat 8 Band Descriptions	7
Table 2. Example of a Confusion Matrix.....	33
Table 3. RF Confusion Matrix - Yellowstone Scene.....	41
Table 4. RF Confusion Matrix - Mississippi Scene.....	43
Table 5. Classifier Statistics – Iris dataset	47
Table 6. Classifier Statistics – Yellowstone Scene.....	47
Table 7. Classifier Statistics – Mississippi Scene.....	47

List of Figures

Figure 1. Earth Observation Process.....	6
Figure 2. Linearly Separable Data	8
Figure 3. Linearly Inseparable Data.....	9
Figure 4. Data Separable by a Spiral Pattern	9
Figure 5. K-Nearest Neighbor Problem	11
Figure 6. Biological Neuron	13
Figure 7. Artificial Neuron	13
Figure 8. Basic Three-Layer Neural Network	14
Figure 9. SVM Hyperplane (Duda et al., 2001).....	16
Figure 10. Decision Tree (Duda et al., 2001)	21
Figure 11. Iris Data Clustered using Random Forest.....	36
Figure 12. Iris ID3 Tree	36
Figure 13. Iris Neural Network.....	37
Figure 14. Yellowstone Scene Composite	38
Figure 15. Spectral Signatures - Yellowstone Scene	39
Figure 16. Yellowstone ID3 Tree	39
Figure 17. OOB Error as a Function of m	40
Figure 18. Output Images – Yellowstone Scene.....	41
Figure 19. Mississippi Scene Composite	42
Figure 20. Mississippi Spectral Signatures	43

Figure 21. Output Images - Mississippi Scene	44
Figure 22. ID3 Tree - Mississippi Scene	45
Figure 23. ERDAS Inquire Box.....	63
Figure 24. ERDAS Subset Image	63
Figure 25. ERDAS Subset Image Window.....	64
Figure 26. ERDAS Signature Editor.....	64
Figure 27. ERDAS Selecting Test Points	64
Figure 28. ERDAS Accuracy Assessment.....	64

Abstract

THE RANDOM FOREST ALGORITHM WITH APPLICATION TO MULTISPECTRAL IMAGE ANALYSIS

Barrett E. Lowe

Thesis Chair: Arun D. Kulkarni, Ph.D.

The University of Texas at Tyler

May 2015

The need for computers to make educated decisions is growing. Various methods have been developed for decision making using observation vectors. Among these are supervised and unsupervised classifiers. Recently, there has been increased attention to ensemble learning – methods that generate many classifiers and aggregate their results. Breiman (2001) proposed Random Forests for classification and clustering. The Random Forest algorithm is ensemble learning using the decision tree principle. Input vectors are used to grow decision trees and build a forest. A classification decision is reached by sending an unknown input vector down each tree in the forest and taking the majority vote among all trees. The main focus of this research is to evaluate the effectiveness of Random Forest in classifying pixels in multispectral image data acquired using satellites. In this paper the effectiveness and accuracy of Random Forest, neural networks, support vector machines, and nearest neighbor classifiers are assessed by classifying multispectral images and comparing each classifier's results. As unsupervised classifiers are also widely used, this research compares the accuracy of an unsupervised Random

Forest classifier with the Mahalanobis distance classifier, maximum likelihood classifier, and minimum distance classifier with respect to multispectral satellite data.

Chapter One

Introduction

Multispectral image data are collected and analyzed for applications such as geo-computation, land management, potential mapping, forecast analysis, and soil assessment to name a few (Ghose et al., 2010). The process of analyzing images of objects taken at great distances is called remote sensing. For the earth's surface, this process requires an image to be taken from either an aircraft or spacecraft. This research considered scenes taken from the satellite spacecraft Landsat 8. It relied on the interaction of electromagnetic radiation with objects on earth's surface. The signal acquired by Landsat is expressed as a function of wavelength. This measurement is referred to as spectral signature. Each object on the ground will react differently to light resulting in each class of objects having a unique spectral signature. The first remote sensing satellite was launched in 1972 and has since become outdated. Today Landsat 8 is equipped with the Operational Land Imager (OLI) sensor which collects images in nine spectral bands. Each band detects different wavelengths of the visible or non-visible spectrum.

Each pixel in a Landsat image can be classified by using the vector of band values as input to a classifier. Conventional statistical methods have been used for classification including neural networks, maximum likelihood classifiers (Huang and Lippman, 1988), support vector machines (Mitra et al., 2004), and decision trees (Pal and Mather, 2001). Supervised methods, such as these, require prior knowledge of each class. Data must be collected to first train the classifier before classifying unknown cases. Clustering

algorithms have also been utilized such as split-merge (Laprade, 1988), fuzzy k-means (Hathaway and Bezdek, 1988) for remote sensing. These methods involve algorithms that group cases together based on input variables having no prior or explicit knowledge of the classes. Clustering is useful when groups of data need to be found that are presently unknown.

Recently research around some of these algorithms has increased with respect to classifier aggregation. Breiman (1996) found that producing multiple versions of a single classifier resulted in higher classification accuracy. He theorized a Random Forest algorithm in 2001, and although it has been used in many data mining applications, has not been fully explored in remote sensing or multispectral image analysis. Random Forest is based on decision tree aggregation where each tree resembles Quinlan's (1986) ID3 tree. Each classifier aggregate must be trained with different data to prevent identical classifiers. During classification, each tree in the forest votes on the class of an input vector. The forest returns the majority vote as the classification result (Breiman, 2001). Random Forest presents unexcelled accuracy among current algorithms, efficient implementation on large data sets, and an easily saved structure for future use of pre-generated trees (Ghose et al., 2010).

This research uses this method for land use classification and is presented as follows. Chapter 2, Literature Review, examines various classification methods against which Random Forest has been tested and explores formulas and theories related to those classifiers. Chapter 3 discusses decision tree classification and Random Forest construction and operation. Chapter 4 provides illustrated examples of classification

using Random Forest and analyzes two scenes acquired with Landsat 8. We then analyzed the same scenes with the same training data using a neural network, support vector machine, nearest neighbor classifier, maximum likelihood classifier, minimum distance classifier, Mahalanobis distance classifier, and spectral correlation classifier. Chapter 5 presents classification accuracies among classifiers, provides a final analysis of the tests, concludes, and discusses future research opportunities.

Chapter Two

Literature Review

2.1 Remote Sensing

Even though remote sensing today usually implies data sensed from space, it consisted of aerial photographs before the space age. Aircraft would fly over the earth taking pictures of interest areas in multiple bands. The photos were analyzed to achieve some level of knowledge about the area. Groundwork for remote sensing was laid at Purdue University in 1965 when the Department of Botany and Plant Pathology suggested that they and the School of Electrical Engineering work together toward better agricultural management solutions using aerospace platforms. Since the launch of SPUTNIK in 1957 followed by the National Air and Space Administration's (NASA) launch of TIROS 1 in 1960, the first satellite designed for earth observation, the potential for remote sensing had grown. TIROS 1 was initially designed as a weather satellite to observe cloud cover but in 1966 NASA asked the National Research Council (NRC) to take on a study regarding the "probable future usefulness of satellites in practical Earth-oriented applications." In 1967 a group of researchers was brought together at the NRC Woods Hole, Massachusetts study center to discuss the use of remote sensing in thirteen fields: forestry-agriculture-geography, geology, hydrology, meteorology, oceanography, broadcasting, point-to-point communication, point-to-point communication, navigation and traffic control, sensors and data systems, geodesy and cartography, economic analysis, and systems for remote sensing information and distribution (Landgrebe, 2003).

At the Woods Hole meeting, three relevant aspects became apparent if remote sensing was going to be used effectively. There was a need for timely information to better manage forest, agricultural, and urban environments as well as other land resources; Earth's cover surface is of a complex and dynamic nature; The problem requires well-coordinated, fundamental, and practical knowledge in multiple fields thus requiring input from multiple scientific disciplines. Even with these challenges, the group concluded that there was significant potential for the use of remote sensing in society. In 1970, a situation arose that tested the conclusion at Woods Hole under an agricultural lens.

During the latter part of the growing season, the U.S. nation's corn crop began seeing buildup of a dangerous pathogen on its leaves looking like brown lesions on the lower part of the canopy. The pathogen was called Southern Corn Leaf Blight and developed from airborne spores. Beginning at the bottom of the plant it would spread upward until the entire plant was destroyed. By the time the danger was discovered, seed for the following year had already been produced. If the spores could survive the winter, the outbreak for the following year would be massive and devastating. Since remote sensing had not yet been tested, this presented an excellent opportunity to evaluate its usefulness.

Efforts were made throughout the winter to track the possible epidemic using aerial photographs taken from a C-47 aircraft. After the winter was over, several possibilities of remote sensing were realized. It could not only be used to discriminate between corn and other types of ground cover, it could also accurately provide data

indicating several degrees of the blight infestation even before it was apparent from imagery created from the data (Landgrebe, 2005).

Since 1970, the basics of remote sensing and observing Earth have not changed. Utilizing spatial, spectral, and temporal data we can observe variations in the data and relate them to desired information. An observational system of the earth has three parts: the scene, the sensor, and the processing system. The entire process can be visualized as

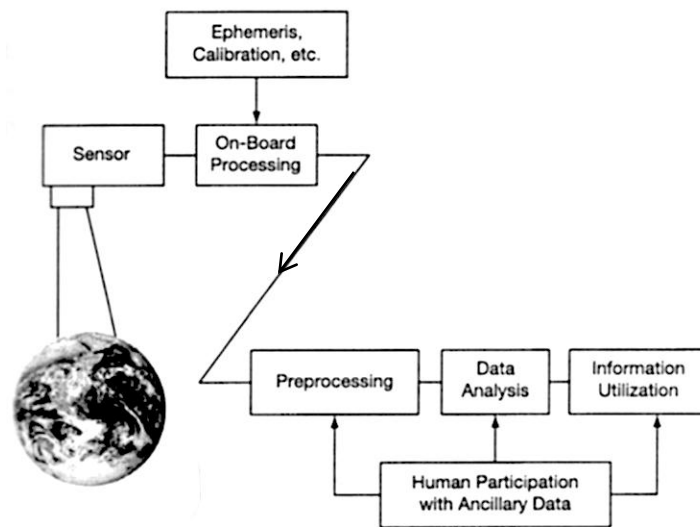


Figure 1. Earth Observation Process

in Figure 1. The scene is the electromagnetic spectrum as it bounces off the area of interest the sensor is the piece of technology that collects the electromagnetic spectrum measurements, and the processing system is what outputs useful information where an analyst specifically indicates what type of information is desired as output (Landgrebe, 2003).

Although remote sensing has been used for decades now, the accuracy of new and novel processing systems seems to not be increasing. Wilkinson (2005) analyzed research

from 1989 to 2004 and found that although more classification methods were being tested and researched with respect to remote sensing, they were not becoming more accurate. Depending on how the data are viewed, the methods may even be less accurate overall. It is noted that the approaches have become more “adventurous” and the change in nature or quality of data over the time period may be counterbalancing algorithmic improvements.

At its first launch, TIROS 1 was not able to capture many spectral bands. By 1968 NASA was developing a new satellite – the Earth Resources Technology Satellite (ERTS). The name was later changed to Landsat. Landsat 8 is orbiting earth today (Landgrebe, 2005). It is equipped with sensors, as previously noted, that are capable of capturing 11 spectral bands. Bands 1 through 9 are captured with the Operational Land Imager (OLI) sensor while the last two are captured with the Thermal Infrared Sensor (TIRS). This research uses bands captured with the OLI. Band descriptions and wavelengths are shown in Table 1 (Landsat 8, 2014).

Table 1. Landsat 8 Band Descriptions

Bands	Wavelength (micrometers)
Band 1 - Coastal aerosol	0.43 - 0.45
Band 2 - Blue	0.45 - 0.51
Band 3 - Green	0.53 - 0.59
Band 4 - Red	0.64 - 0.67
Band 5 - Near Infrared (NIR)	0.85 - 0.88
Band 6 - SWIR 1	1.57 - 1.65
Band 7 - SWIR 2	2.11 - 2.29
Band 8 - Panchromatic	0.5 - 0.68
Band 9 - Cirrus	1.36 - 1.38

2.2 Supervised Classifiers

A classification problem exists when various input values are to imply some output measure or class within a given range or dataset. If we were to classify some fruit as strawberries or oranges, weight and volume might be used as input to our classifier. Plotting the training data in two dimensions where the x-axis represents weight and the y-axis represents volume might yield a plot such as Figure 2. By examining the plot we can infer that the dots are strawberries and the crosses are oranges because oranges will have higher weights and volumes than strawberries. We can also see that the data is linearly separable meaning we can draw a straight line to separate the two classes from each other. Classifier construction is concerned with finding the decision boundary between the classes. This becomes more difficult as data becomes less separable as in Figure 3 and Figure 4. This chapter reviews various methods of finding the separation between data in

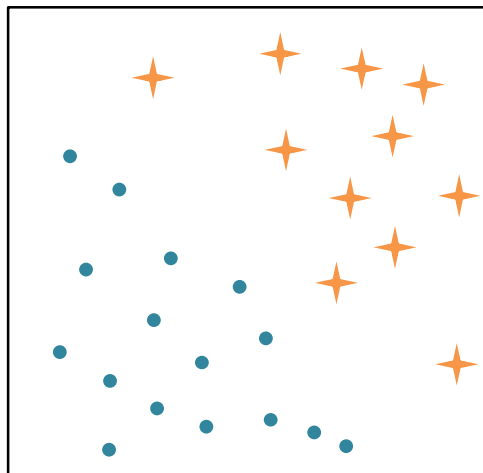


Figure 2. Linearly Separable Data

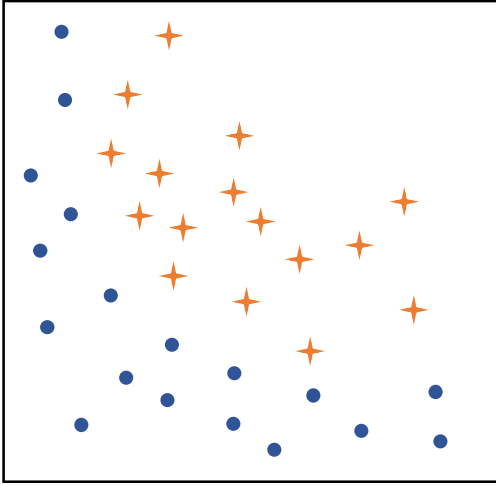


Figure 3. Linearly Inseparable Data

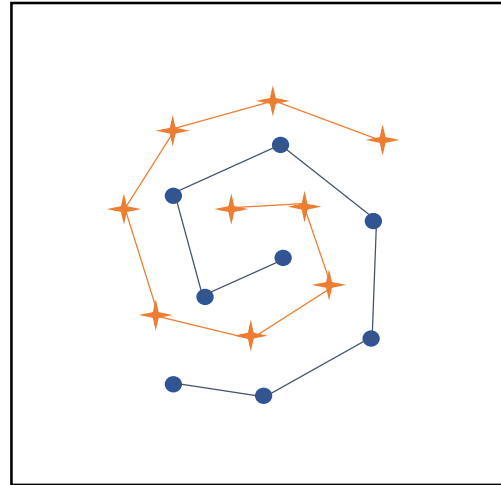


Figure 4. Data Separable by a Spiral Pattern

multi-dimensional spaces. Classifiers can be either supervised or unsupervised.

Supervised classifiers require a training dataset, where input variables and desired output are known. Supervised classifiers are susceptible to the phenomenon known as

overfitting. This occurs when the classifier performs better on training data than on test data. When this happens it indicates that the classifier does not generalize well and

unknown samples will not be as accurate as the training data (Forsyth and Ponce, 2012).

Unsupervised classifiers do not require any prior knowledge of the data and use statistical properties to group the data into clusters. Once clustered, a user can label each cluster as an appropriate class.

2.2.1 Minimum Distance Possibly the simplest classifier to understand and describe is the minimum distance classifier. This is a supervised classifier meaning prior knowledge about the dataset is required. From the known dataset, calculate a mean feature vector for each class. From that, assign an unknown tuple to the closest class

when plotted in a Euclidean space. Let $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \dots, \mathbf{r}_m$ represent mean reference vectors for m classes. Given an input tuple x , the classifier calculates

$$\|x - r_i\| = \sqrt{(x - r_i)^T (x - r_i)} \quad (1)$$

for $i=1, 2, \dots, m$ and assigns x to the class ω_i where the distance $\|x - r_i\|$ is minimum (Kulkarni, 2001).

Due to the algorithm's simplicity and less expensive calculations, it can be implemented on basic hardware. It has been used to detect road signs on a field-programmable gate array and achieved similar results to other more computationally complex algorithms (Zhao et al., 2012). The basic principle of the classifier has been altered and adapted in many ways to improve efficiency and accuracy among different applications. Lin and Venetsanopoulos (1993) created a weighted minimum distance classifier for pattern recognition and proved it had similar results to neural networks with far less computation. It has also been used for character recognition (Senda et al., 1995).

2.2.2 K-Nearest Neighbor The k-nearest neighbor classifier is similar to the minimum distance classifier. Instead of considering the means of each class, it considers the k nearest points to an unknown tuple and assigns the tuple to the majority of its neighbors. Choosing the correct value for k greatly affects the accuracy of the classifier. Choosing a value too large will encompass all of the training data and assign the tuple to the class with more training examples. Choosing a value too small creates a problem as in Figure 5. Clearly the data are linearly separable and the new point, represented as a red diamond, falls within the class on the bottom of the figure. However if we choose a k

value of 1, this tuple would be classified as part of the wrong class. Increasing k to just 3 or 5 would solve the problem.

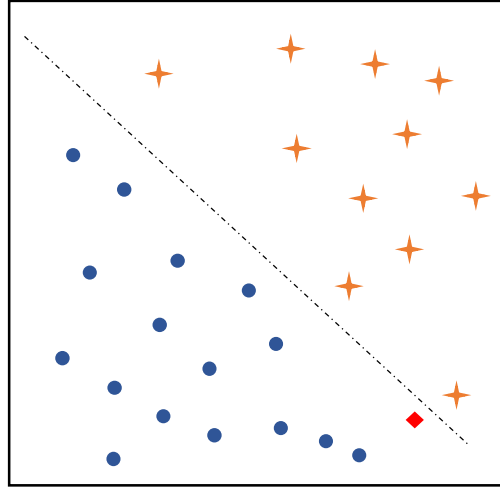


Figure 5. K-Nearest Neighbor Problem

2.2.3 Naïve Bayes Bayesian Decision Theory, named after Thomas Bayes, is a statistical method of classification that considers probability of a class given some set of data. The decision rule is simple. Suppose a goal is to classify tuples into classes ω_1 and ω_2 . Let the probability of a tuple belonging to class ω_1 be denoted by $P(\omega_1)$. We assign the tuple to class ω_1 if $P(\omega_1) > P(\omega_2)$, otherwise we assign it to ω_2 . With no given data and assuming normal distribution, both probabilities are equal and a classifier with this rule would always give the same classification resulting in a 50% accuracy rate. To increase this, let's use some feature measurement x . Thus $P(x/\omega)$ indicates the probability of measurement x given that the tuple belongs to class ω . This means that the difference in two classes is denoted as the absolute value of $P(x_1/\omega_1) - P(x_2/\omega_2)$. Now given a measurement x with unknown class ω_j , we can calculate

$$P(\omega_j | x) = \frac{P(x | \omega_j)P(\omega_j)}{P(x)} \quad (2)$$

based on the probability that the tuple both belongs to class ω_j and has measurement x as in Equation (3) (Duda et al., 2001).

$$P(x | \omega_j) = \prod_{k=1}^n P(x_k | \omega_j) \quad (3)$$

$$P(\omega_j | x)P(x) = P(x | \omega_j)P(\omega_j) \quad (4)$$

The class j for which $P(x|\omega_j)P(\omega_j)$ is maximized is the maximum posteriori hypothesis. Since $P(x)$ is constant for all classes, we must maximize $P(x|\omega_j)P(\omega_j)$ from Equation (2). If $P(\omega_j)$ is not known, we assume all class priors are equal and we must maximize $P(x|\omega_j)$. Computing this would be extremely computationally expensive if x became a vector X or set of measurements. However, assuming that each measurement is independent of the others, we can simplify the calculations as in Equation (4). Finally, to predict a class label for input vector X , the classifier chooses class ω_1 if

$$P(X | \omega_1)P(\omega_1) > P(X | \omega_2)P(\omega_2) \quad (5)$$

or ω_2 otherwise (Han et al., 2012).

This basic premise has been altered in various ways to produce more accurate classifiers across multiple applications as the simple classifier can only create linear frontiers (Duda and Hart, 1973). Domingos and Pazzani (1997) used the basic classifier for text classification learning conjunctions and disjunctions. They found that violation of the independence assumption has little effect on the accuracy of the classifier in textual applications. Rennie et al. (2003) also used the classifier for text classification removing assumptions at the cost of efficiency but gaining high accuracy rates surpassing a support vector machine.

2.2.4 Neural Network Perhaps the most well established classifier is the neural network. Neural networks have been used in classification since the 1950s (Rosenblatt,

1958). They are very powerful and can accurately classify linearly inseparable data (Huang and Lippman, 1988). Neural networks resemble the biological networking of the brain. A biological neuron is shown in Figure 6. Dendrites receive some signal which is interpreted by the cell body and nucleus. A reaction signal is sent from the cell body down the axon to the synapse where it is used as input to the dendrites of another biological neuron.

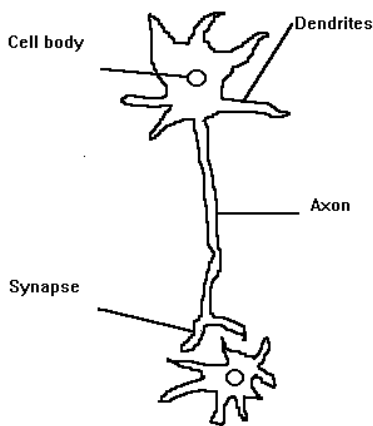


Figure 6. Biological Neuron

Neural networks are made up of multiple artificial neurons that can be seen in Figure 7. Input vector \mathbf{x} represents input of the neuron and is processed using some activation function F . The output of the neuron is then used as input to another artificial

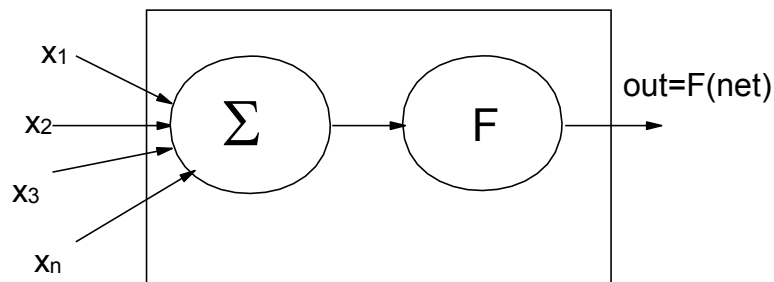


Figure 7. Artificial Neuron

neuron. A neural network can be made up of multiple layers. Figure 8 shows a basic three-layer network.

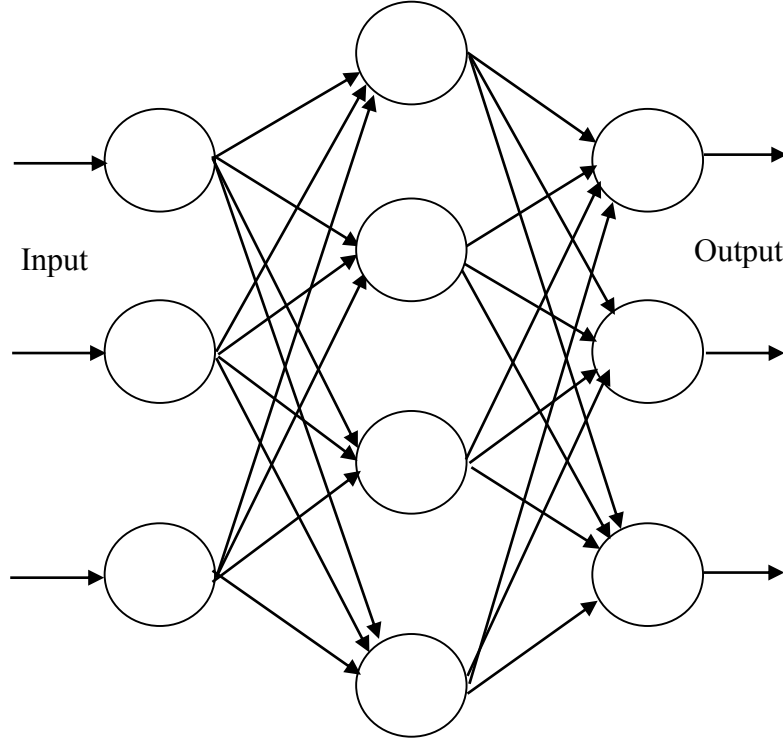


Figure 8. Basic Three-Layer Neural Network

Neurons are connected from one layer to another by way of some weight w_{ij} . Before training, a weight matrix \mathbf{W} is initialized with random numbers. The network learns by iteratively updating the weight matrix to obtain output values closer to a desired

$$y_i = F\left(\sum_{j=1}^m w_{ij}x_j\right) \quad (6)$$

target vector \mathbf{y} . Actual output is computed via Equation (6) where $F(\cdot)$ represents the activation function, m is the number of inputs, w_{ij} is the weight for the given input, and x_j is the input value. The change in weight can be computed as

$$\Delta w_{ij} = \alpha(y'_i - y_i)x_j \quad (7)$$

using a linear activation function where α is some constant defining the learning rate of the network and y_i' is the actual output. Weights are updated by Equation (8) where l is

$$w_{ij}(l+1) = w_{ij}(l) + \Delta w_{ij} \quad (8)$$

the previous iteration for input j . These steps are repeated until the error reaches some value ϵ_{min} or the number of iterations reaches some maximum value N_{max} . For data that are linearly separable, a single layer network can learn and classify with 100% accuracy. If data are not linearly separable more layers need to be used (Kulkarni, 2001).

This basic method of construction was one of the most powerful classifiers of its time. Paul Werbos, in his PhD thesis, formulated a method of backpropagation through the various layers of the network and updating weights for more accurate classification (Werbos, 1974). His thesis was not concentrated on neural networks. Rumelhard and McClelland (1986) applied backpropagation specifically to neural networks and found much higher accuracy rates for nonlinearly separable data.

2.2.5 Support Vector Machine Support vector machine (SVM) is a method of classification with an output resembling the neural network. They have been used to solve classification problems in handwritten digits (Cortes and Vapnik, 1995), multispectral images (Mitra et al., 2003), gene selection in cancerous tissues (Guyon et al., 2002), among others. SVMs were first introduced by Boser et al. (1992); however the high level mathematics to support its operation and theory can be traced back to literature concerning hyperplane decision boundaries by Vapnik and Chervonekis (1968). A support vector machine, at its most basic level as a two class linearly separable problem,

plots the input training data and finds the optimal plane separating the two classes. The plane is established as the decision boundary and any case put through the SVM is

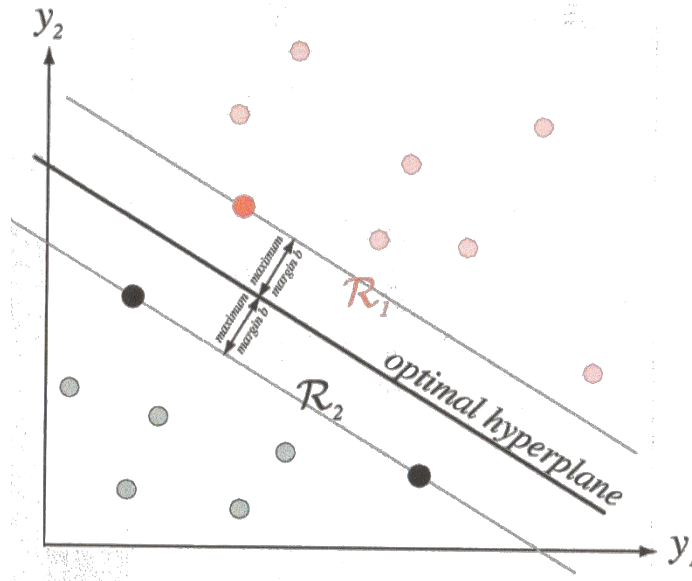


Figure 9. SVM Hyperplane (Duda et al., 2001)

classified based on what side of the decision boundary it falls. The distance from the plane to the nearest training tuple is referred to as margin and SVM will maximize the margin for both classes thus placing the plane equidistant from the closest training tuple in each class. These training tuples are called support vectors.

Suppose the same two class problem is no longer linearly separable. The SVM finds multiple planes that accurately separate the data. This becomes much more complex and challenging as the data becomes less separable thus requiring more planes for accurate separation. Consider a classification problem where data are plotted as in Figure 4. These data are clearly not linearly separable and require a great number of separating planes. It is possible that, in such a case, every training tuple might be a support vector requiring expensive calculation. SVMs tackle this obstacle by increasing the

dimensionality of the training data so that a single hyperplane can accurately separate the data. Simply, it creates a nonlinear decision boundary represented by multiple linear segments. Although training time for SVMs can be extremely slow, they are highly accurate and less prone to overfitting than other classification methods. They can be used for continuous and discontinuous data and the support vectors themselves can provide a good deal of knowledge for rule extraction problems (Han et al., 2012).

The method of finding a decision function for pattern vectors \mathbf{x} of n dimensions belonging to one of two classes A or B is as follows. If the training set consisting of p examples x_i with labels y_i , as in Equation (9), the algorithm finds the decision function $D(x)$ and after training, unknown patterns can be classified according to Equation (10).

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_p, y_p)$$

$$where \begin{cases} y_k=1 & \text{if } x_k \in \text{class A} \\ y_k=-1 & \text{if } x_k \in \text{class B} \end{cases} \quad (9)$$

$$\begin{aligned} x \in A & \text{ if } D(x) > 0 \\ x \in B & \text{ otherwise.} \end{aligned} \quad (10)$$

$$D(x) = \sum_{i=1}^N \omega_i \varphi_i(x) + b \quad (11)$$

The linear function expressed in n dimensions (the feature space) can be expressed as Equation (11). This is identical to the perceptron decision function of neural networks where φ_i is a training input tuple, ω_i is the weight vector of N attributes, and b is some scalar or bias. In our two dimensional two class example, we now need to maximize the margin in the feature space. Let us denote the distance from the decision

boundary to pattern x as $D(x)/\|w\|$. Assuming our training data is linearly separable and a margin M exists, then all training patterns will fulfill Equation (12).

$$\frac{y_k D(x_k)}{\|w\|} \geq M \quad (12)$$

The goal now is to find a weight vector w that maximizes the margin M . Imposing the constraint $M/\|w\|=1$ implies that Equation (12) minimizes $\|w\|^2$. The support vectors are the training patterns that satisfy Equation (13).

$$y_k D(x_k) = 1, \quad k = 1, 2, \dots, p \quad (13)$$

Informally speaking, we search all patterns for the worst classified pattern to use as a support vector. This can be computationally expensive for even small training sets (Duda et al., 2001). Transforming the data into a space with higher dimensionality actually creates a computationally less expensive problem. Call this space the dual space.

The transformation from the feature space to the dual space takes place by way of the Lagrangian, a high level mathematical function. Its operation is beyond this paper's scope. Consider, though, an example. To map our 2-D input vector into a 4-D dual space, we could use mappings $\phi_1(X)=x_1$, $\phi_2(X)=x_2$, $\phi_3(X)=x_1^2$, and $\phi_4(X)=x_1x_2$. Increasing dimensionality of the data results in a decision function in the form of Equation (14)

$$D(x) = \sum_{k=1}^p a_k K(x_k, x) + b \quad (14)$$

where a_k is the weight vector and K is some kernel function (Boser et al., 1992).

Depending on the application of the SVM, K could be a polynomial of some degree, a Gaussian radial basis function, or a sigmoid function. These kernels create decision boundaries similar to that of neural networks. For example, a SVM with a sigmoid kernel is equivalent to a two-layer neural network with no hidden layers. The decision

corresponds to a non-linear second order polynomial in the original feature space. Using the Lagrangian, Equation (14) can be rewritten as Equation (15)

$$d(\mathbf{x}^T) = \sum_{i=1}^l y_i a_i \mathbf{x}_i \mathbf{x}^T + b_0 \quad (15)$$

where y_i is the class label of the support vector \mathbf{x}_i , \mathbf{x}^T is a test tuple, a_i are Lagrangian multipliers, b_0 is the parameter figured during training, and l is the number of support vectors. Using Equation (15), we can classify test tuple \mathbf{x}^T by checking the sign of the result. A negative sign indicates it falls below the margin while a positive sign indicates it falls above it (Han et al., 2012). This indicates that the classifier is binary and only able to handle two classes. Hastie and Tibshirani (1998) created multiple binary classifiers and using a “max-wins” rule were able to implement binary classifiers to solve a multi-class problem. Applying the “max-wins” concept to SVMs allows their use in multi-class problems as well.

Chapter 3

Methodology

This chapter deals with tree classifiers and the Random Forest algorithm. Their implementations are explained and reviewed with respect to multispectral image analysis.

3.1 Tree Classifiers

Another method of classification is a decision tree classifier. The first use of a tree based decision system can be dated back to the 1960s where decision trees were first used in artificial intelligence in 1966 (Li et al., 2001; Hunt et al., 1966). They are supervised classifiers and are more efficient than single stage classifiers because decisions are made at multiple levels. Tree classifiers are represented as acyclic graphs with a root node and successive child nodes connected by directional branches. Decision trees use this structure to make decisions. Each node makes a decision based on an attribute value of the data. Binary decision tree classifiers make only true/false decisions but it is possible to make a more complex decision based on attribute values such as “yellow”, “red”, or “green” if “color” is the attribute in question. In binary trees, a case traversing to the left child is true while a case traversing to the right is false. Nodes in non-binary trees have one child for each possible attribute value. A case traverses down the tree based upon its data until reaching a leaf node at which point the tree can return a final decision or classification for that case. For example, if a decision tree is to classify fruit, each node may make a decision based on attributes of fruit such as color, shape, size, and taste. An apple and a banana might have a sweet taste and a medium size but will differ in shape and color so a node that makes a decision based on the color being red might send apples

to the left child and bananas to the right child. This allows a hierarchical decision scheme that seemingly breaks the problem down into smaller simpler questions. Figure 10 shows a basic decision tree for classifying fruit based on the features color, shape, size, and taste. The output classes are watermelon, apple, grape, grapefruit, lemon, banana, and cherry (Duda et al., 2001).

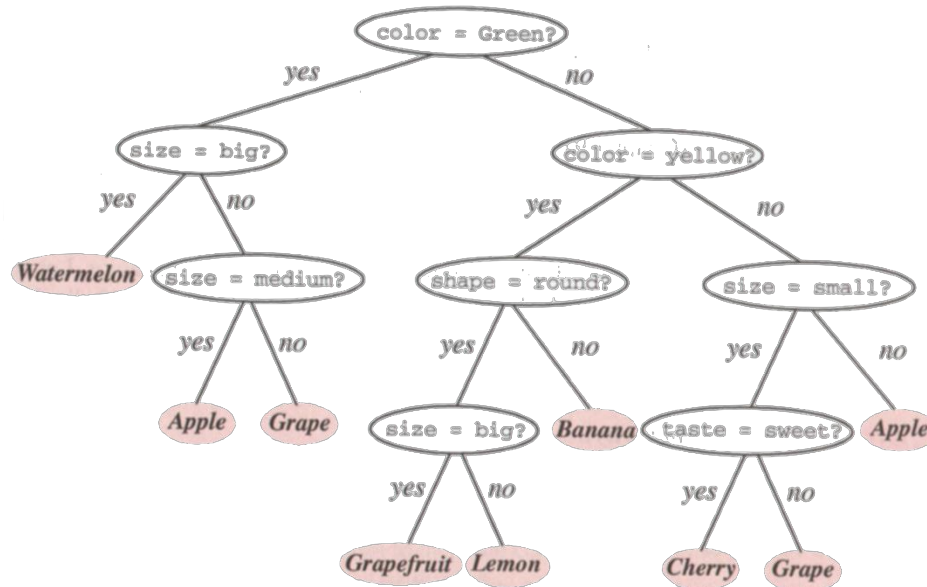


Figure 10. Decision Tree (Duda et al., 2001)

For such a small data set, how to build the tree might seem obvious. When more features are introduced, however, the problem becomes much more complex. The difficulty in utilizing decision trees lies in their construction. Many methods have been proposed; however, the construction of the tree heavily relies on the tree's application. One of the heuristic approaches to construct a tree is to utilize feature vectors and recursively partition the data at each node, based on a well-chosen feature, to obtain the highest level of information gain from node to node (Pal and Mather, 2001). Three major questions become apparent in construction: How data should be split at each node, the

splitting-condition, and when splitting should stop and a final decision be returned, the stopping-condition. It is desirable to construct an optimum tree so as to achieve the highest possible accuracy with the fewest calculations (Kulkarni, 2001).

In constructing the tree, every test case is presented at the root node where the tree starts. The goal is assigning to the node a set of features that splits the data or cases most efficiently. As more nodes are created, data are further separated until every case remaining at a node is of the same class. This indicates a terminal node. Appropriate splitting conditions vary across applications but among popular methods are entropy, gini, and twoing. Each method has its advantages and splits the data differently. The expected information needed to classify an observation vector D is given by:

$$entropy(D) = -\sum_{i=1}^m p_i \log(p_i) \quad (16)$$

where p_i is the non-zero probability that an arbitrary observation vector in D belongs to class C_i and m is the number of possible classes. Entropy is a basic measure of amount of information. This is the most widely-used splitting condition as it attempts to divide the data as evenly as possible giving the most information gain from parent to child nodes. It can be seen that if all cases belong to the same class, $entropy(D)$ will be 0 because there is no information gain. This indicates that the current node is terminal and a classification decision has been reached (Han et al., 2012). Entropy splits the data as evenly as possible from parent to child node. It is also possible to split data by extracting cases into the

largest possible homogeneous group (Apte and Weiss, 1997). This is obtained similarly by calculating the gini information gain given by:

$$gini(D) = 1 - \sum_{i=1}^m p_i^2 \quad (17)$$

Gini information gain can also be interpreted as the expected error rate if a classification decision is selected randomly from the class distribution present at the current node (Duda et al., 2001). Twoing makes use of a very different idea when splitting. It gives strategic splits by, at the top of the tree, grouping together cases that are largely similar in some characteristic. The bottom of the tree identifies individual classes. When twoing, classes are grouped into two super classes containing an as-equal-as-possible number of cases. The best split of the super classes is found and used as the split at the current node. This results in a reduction of class possibilities among cases at each child node and a reduction in impurity. For example denote all classes at a node by C where $C = \{1, \dots, J\}$. At each node, divide C into two classes, C_1 and C_2 where $C - C_1 = C_2$. The idea is to treat the problem as a two-class problem. For any given split s , at the node, compute the change in impurity $\Delta_i(s, D, C_1)$. Take the split $s^*(C_1)$ giving the highest change in impurity and find the superclass C_1^* maximizing $\Delta_i(s^*(C_1), D, C_1)$. Use $s^*(C_1)$ as the split at the current node. The twoing criterion function is defined by:

$$twoing(D) = \frac{P_L P_R}{4} \left[\sum_{i=1}^m |p(i | D_L) - p(i | D_R)| \right]^2 \quad (18)$$

Other splitting methods are noted by Duda et al. (2001) and Venables and Ripley (2002).

An ideal node is one that contains only records of the same class. In practice, reaching this node may require an excessive number of costly splits. Splitting too much

results in nothing more than a lookup table and will perform quite poorly for noisy data while splitting too little results in a small number of terminal nodes containing an even probability distribution among classes increasing the error rate of the decision tree (Duda et al., 2001). Often splitting will cease when a node is considered pure using one of the previously mentioned impurity rules. Node depth, that is length from root to node t , can also be used to identify leaf nodes. Similarly a stopping condition could also be satisfied by thresholding the depth of children of node t . Another common method is to threshold the number of existing cases at node t . If there are fewer cases than some threshold, splitting does not occur (Ghose et al., 2010). The basic algorithm for generating a decision tree is shown below (Han et al., 2012).

Generate decision tree(*data partition, attribute list*)

- 1) create a node N ;
- 2) IF tuples in D are all of the same class, C , THEN
- 3) return N as a leaf node labeled with the class C ;
- 4) IF *attribute_list* is empty THEN
- 5) return N as a leaf node labeled with the majority class in D ;
- 6) apply *attribute_selection_method*(D , *attribute_list*) to find the best *splitting_criterion*;
- 7) label node N with *splitting_criterion*;
- 8) IF *splitting_attribute* is discrete-valued AND multiway splits are allowed THEN
- 9) *attribute_list* \leftarrow *attribute_list* – *splitting_attribute*;
- 10) FOR EACH outcome j of *splitting_criterion*
- 11) let D_j be the set of data tuples in D satisfying outcome j ;
- 12) IF D_j is empty THEN
- 13) attach a leaf labeled with the majority class in D to node N ;
- 14) ELSE attach the node returned by **Generate_decision_tree**(D_j , *attribute_list*) to node N ;
- 15) END FOR
- 16) return N ;
- 17) return N ;

In the previous example the taste of fruit could be sweet or sour and can be represented by a single bit of data. Fruit color might use more than a single bit but is a discrete value. Measurements, however, are not discrete. When training a classifier with a data set of continuous values, say in the range $\{-1, 1\}$, the classifier learns only a certain number of values in the range because it views the values as discrete. Making decisions on attributes with continuous values is difficult and has been attempted in various ways. In order to handle continuous values, an algorithm creates interval bins where feature values can be organized in some well-chosen manner. If the training data contains v values for some feature F , k intervals can be created where $k=v$. Now every v in F can be organized into a bin and the bin number can be used as the value of F in operation (Jearanaitanakij, 2005).

Splitting and stopping conditions directly impact the accuracy and error rates of decision tree classifiers and their overall efficiency. Research has shown that growing a tree to its fullest extent and then strategically reducing, or pruning, the tree results in higher efficiency and accuracy rates for the final classifier than growing with strategic stopping and splitting conditions. Many methods of pruning have been proposed such as minimal-cost complexity pruning, reduced error pruning, pessimistic-error pruning and others (Breiman et al., 1984; Quinlan, 1986). As the Random Forest algorithm does not use pruned trees, this research does not explore effects of pruning on the classifier but methods are worth mentioning and considering for future research.

Breiman et al. (1984) proposed one of the first methods of pruning called minimal-cost complexity pruning. The idea is to measure the average error produced per

leaf and prune the leaves with the smallest errors. For each node, find some α as a complexity parameter. Then at that node calculate the resubstitution error estimate including a penalty of α for each terminal node of the subtree. Pruning nodes with the smallest α and finding subtrees with the smallest error estimates will create a more accurate classifier and prevent overfitting of the data. Reduced error pruning proposed by Quinlan (1987) involves using a validation set, usually a subset of the training set withheld from the initial training of the classifier. After the tree is grown, the validation set is put down the tree and errors are calculated. For each non-terminal node, errors of the entire subtree are summed and established as the error of that subtree. The error is calculated again, but this time with the subtree converted to a single leaf with a majority class label. In the overall tree, the node with the highest reduction in error is pruned. This process continues until error no longer reduces.

Pessimistic pruning can be performed without a validation set. If S is a subtree of the fully grown tree T , let $L(S)$ be the number of leaves in S . Let K be the total number of cases reaching S and J be the number of cases S misclassifies taking into account all leaves of S . Since S is built with training cases, we assume that after unseen cases are passed through S , it will misclassify $J + L(S)/2$ out of K cases. Now if we replace S by the most accurate leaf in S and let E be the number of cases it misclassifies, we replace S with that leaf when $E + \frac{1}{2}$ is within one standard error of $J + L(S)/2$. This method adjusts the error pessimistically assuming that more errors will be encountered with non-training data. This method is faster than minimal cost pruning and reduced error pruning because it evaluates each subtree in T only once.

3.2 ID3 Tree

A variation on the basic decision tree, the ID3 tree, has been found to be not only efficient but extremely accurate for large datasets with many features. The idea behind ID3 trees is that given a large training set, only a portion is used to grow a decision tree often referred to as the bootstrap portion. The remaining training cases are then put down the tree and classified. Misclassified results are used to grow the tree further and the process repeats. When all remaining cases in the training set are accurately classified, the tree is complete. Evidence suggests that this method will grow an accurate tree much more quickly than growing a tree using the entire training set; however it should be noted that this method cannot guarantee convergence on a final tree (Quinlan, 1986).

In Quinlan's original ID3 representation, entropy was used as a splitting condition and total node purity was used as a stopping condition. The information gain was found by calculating the total amount of information needed for the tree and subtracting the

$$Gain(A) = Info(D) - Info_A(D) \quad (20)$$

$$Info_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \times Info(D_j) \quad (19)$$

information needed by a tree with a root node N after being split with attribute A as in Equation (20). $Info(D)$ can be computed using one of Equations (16-18) depending on the desired method of splitting. $Info_A(D)$ is computed using Equation (19) where $|D_j|/|D|$ represents the weight of the j^{th} split and v is the number of discrete values of attribute A. The largest information gain using A determined the attribute on which to split at node N. The process is recursive. Using this, Quinlan was able to build efficient and accurate trees

very quickly without using the entirety of large training sets reducing construction time and cost.

Breiman (1996) introduced the idea of bagging which is short for “bootstrap aggregating”. The idea is to use multiple versions of a predictor or classifier to make an ultimate decision by taking a plurality vote among the versions. Twenty-five regression trees constructed from bootstrap samples of the training set gave a median error decrease of 40% from a single tree predictor over five datasets. In bagging, it has been proven that as the number of predictors increases, accuracy also increases until a certain point at which it drops off. Finding the optimal number of predictors to generate will yield the highest accuracy.

3.3 Random Forest

3.3.1 Supervised Random Forests are grown using a collaboration of the bagging and ID3 principles. Each tree in the forest is grown in the following manner. Given a training set, a random subset is sampled (with replacement) and used to construct a tree which resembles the ID3 idea. However, every case in this bootstrap sample is not used to grow the tree. About one third of the bootstrap is left out and considered to be out-of-bag (OOB) data. Also, not every feature is used to construct the tree. A random selection of features is evaluated in each tree. The OOB data is used to get a classification error rate as trees are added to the forest and to measure input variable (feature) importance. After the forest is completed, a sample can be classified by taking a majority vote among all trees in the forest resembling the bootstrap aggregating idea.

For example, in judicial court a robber may be on trial for stealing. The jury members will classify the robber as guilty or innocent. They have been trained by a random subset of every account of robbery in history. This subset contains all robberies of which a single juror has heard or seen. Having been trained by other accounts of robbery, each jury member will take different variables of the trial into consideration. One member might take into account the value of the object stolen, the victim of the robbery, and the robbers age while another jury member might classify the robber's guilt based on gender, his/her religion, and the robber's age. At the end of the trial, all jury members vote on the classification of the robber. In this limited example, the jury is the forest, each member is a tree, the robber is the case to be classified, and the pieces of evidence of the case are the features to be used for classification.

The error rates of the forest are measured by two different values. A quick measurement can be made using the OOB data but, of course, a set of test cases can be put through to forest to get an error rate as well. Given the same test cases, the error rate depends on two calculations: correlation between any two trees in the forest and the strength, or error rate, of each tree. Returning to our jury metaphor, if every member of the jury took only the features of age, gender, and race into account for classification, thus showing high-correlation between jurors, the jury would come to a correct conclusion about half the time (randomly) as age, gender, and race have almost nothing to do with theft. The goals are to establish a jury that considers every piece of evidence of the trial and to select jurors who, on their own, are usually right about the final outcome. If jury members are trees, how do we grow trees with low correlation to one another and

high in strength? The answer lies in how many variables each tree must consider. If we have M input variables and select m of them at random to grow a tree. As m increases correlation and individual tree accuracy also increases; thus an optimal m will give the lowest error rate. Each tree will be grown by splitting on m variables; m stays constant throughout the forest.

3.3.2 Unsupervised Random Forest can be run unsupervised; that is, without any prior information about the input data. Many unsupervised classifiers plot each input case in a multidimensional space based on feature values. If four features are evaluated in classification, each case is plotted in a four dimensional feature space and clustered using Euclidean distance. Random Forest works similarly unsupervised by calculating a variable dissimilarity measure and plotting each case in a space according to dissimilarity measure and clustering using Euclidean distance (Shi and Horvath, 2006).

Dissimilarity measures can be found in supervised learning by putting the training data down each tree. If observations i and j fall in the same terminal node, their similarity is increased by one. These values can be stored in matrix form. After the forest has been constructed and similarities have been found, the similarity matrix is divided by the number of trees. The dissimilarity is defined as:

$$D_{ij} = \sqrt{1 - S_{ij}} \quad (21)$$

where S is the similarity matrix. Using multidimensional scaling with dissimilarities as inputs, it is possible to plot points in a Euclidean space where the distance between the points is equal to the dissimilarities. The points can then be clustered.

3.3.3 Synthetic Data When classes are completely unknown, it is necessary to make up or synthesize classes. By smartly creating synthetic data and adding it to the original dataset, we then have two classes: original data, and synthetic data. We can solve this two class problem using Random Forest and evaluate dissimilarity measures. Plotting based off the dissimilarity measures results in data that is able to be clustered appropriately.

Using synthetic data to create an accurate classifier may seem inaccurate; however Nonnemaker and Baird (2009) have proved that an accurate unsupervised classifier can be created for character recognition using synthetic data. Similarly we can use Random Forest. Synthetic data is created by sampling at random from the original data. For example, given 100 original records, one would create 100 synthetic records. The feature values are drawn from the reference distribution. If each record has 20 individual features, we could standardize some number of those features among all synthetic records and see if those features give good indication of original versus synthesized data. In our example of 100 original records and 100 synthetic records, let us say we standardize the first feature and put all data through the forest. If the OOB error rate is, say, greater than 40%, it implies that the standardized feature looks too much like an independent feature and is not playing a significant enough roll in classifying the data. We could try switching the standardized feature in the synthesized data or using two features and reclassifying. Once an appropriate OOB error is reached, say 30% or less, the standardized features are playing an important roll in classification and a dissimilarity matrix can be constructed in the same way as mentioned above. Using the dissimilarities

as input, a multidimensional scale plot can be constructed and the data can be clustered (Shi and Horvath, 2006), (Breiman and Cutler, 2007).

3.3.4 Variable Importance Random Forest can measure variable importance. This is useful for data mining purposes and can assist in unsupervised classification. If we change a single feature's input value and reclassify the record, we can determine that the feature's importance is based on the new classification. This is done using OOB data. Each variable m is randomly permuted and the permuted OOB cases are sent through the forest again. Subtracting the number of correctly classified cases using permuted data from the number of correctly classified cases using non-permuted data gives the importance value of variable m . These values are different for each tree, but the average of each value over all trees in the forest gives a raw importance score for each variable (Breiman and Cutler, 2007). This research implemented Random Forest using a software package in R and analyzes Landsat images. Implementation and results from the analysis are in the next chapter.

3.4 Accuracy Assessment

After a classifier is trained, test data can be put through it and error rates can be represented in an $m \times m$ matrix where m is the number of classes in the dataset called a confusion matrix. Columns represent reference, or correct, classes and rows represent the classifier decision. The upper left to bottom right diagonal indicates the number of correct classifications where reference data and classifier agree. Table 2 is an example of an error matrix, also referred to as a confusion matrix. This matrix is a starting point for many statistical and descriptive techniques for assessing accuracy.

Table 2. Example of a Confusion Matrix

	Class 1	Class 2	Class 3	Class 4	Row Total
Class 1	65	4	22	24	115
Class 2	6	81	5	8	100
Class 3	0	11	85	19	115
Class 4	4	7	3	90	104
Column Total	75	103	115	141	434

Overall accuracy, the most common statistic of accuracy, is calculated by dividing the total number of correct classifications by the number of attempted classifications. For years, this was the only method of reporting classifier accuracy. Congalton (1991) remarked that other calculations of accuracy are pertinent to assess a classifiers accuracy. Assessing the classifier's accuracy with respect to a certain class can provide valuable data for classifier improvement. This can be found in two different ways depending on the expected result. The number of correctly classified records can be divided by the total number of records in the corresponding row or the corresponding column of the confusion matrix. Dividing by the column total, or the total number of actual classes, gives producer's accuracy of the class in question. Producer's accuracy is the probability of a record in a certain class being classified correctly – a measure of omission error. Alternatively dividing by the row total gives user's accuracy. This is the probability that a record classified as a certain class is correct – commission error. User's and producer's accuracies may seem similar but give very different statistics. In Table 2, Class 1's producer's accuracy is $65/75=87\%$ and its user's accuracy is $65/115=57\%$. If a record is classified as Class1, it has a 57% chance of being correctly classified, but all records that belong to Class 1 have an 87% chance of being correctly classified.

Most probabilities do not take into account random chance. Cohen (1960) formulated KAPPA, a statistic that removes chance as a factor of probability. It considers the rate of agreement between the actual class and the classifier's decision. Since agreements are on the diagonal, agreement is the same as overall accuracy but this factors in chance. Class 1 represents $75/434=17.3\%$ of the reference classes but $115/434=26.5\%$ of the classifier's decisions. This indicates that $17.3\% \times 26.5\%=4.6\%$ of the decisions are due to chance. Similarly Class 2 is 5.5%, Class 3 is 7%, and Class 4 is 7.8% chance.

KAPPA is defined by:

$$K = \frac{n_a - n_s}{n - n_s} \quad (22)$$

where n is the number of records, n_a is the number of agreements, and n_s is the number of agreements due to chance yielding 72.4% indicating that the classifier accuracy without chance. A KAPPA analysis can also include another statistic, KHAT. KHAT is another accuracy measure that assumes multinominal sampling and that variance is derived using the Delta method. It is defined in Equation (23) where r is the number of rows in the

$$\hat{K} = \frac{N \sum_{i=1}^r x_{ii} - \sum_{i=1}^r (x_{i+} * x_{+i})}{N^2 - \sum_{i=1}^r (x_{i+} * x_{+i})} \quad (23)$$

confusion matrix, x_{ii} is the number of observations in row i and column i , x_{i+} is the marginal total of row i , x_{+i} is the marginal total of column i and N is the total number of records classified.

Chapter 4

Implementation and Results

In this project we developed the code to simulate Random Forest. The simulation was tested with three datasets: the well-known Iris dataset from the University of California at Irving (Bache and Lichman, 2013), a Landsat scene of the Mississippi River Bottomland, and a Landsat scene from Yellowstone National Park. The same datasets were classified with a single ID3 tree, neural network, support vector machine, minimum distance classifier, maximum likelihood classifier, Mahalanobis distance classifier, and spectral angle and correlation classifiers. The results of all classifiers are provided in this chapter.

4.1 Iris dataset

The well-known Iris dataset was run through the forest. It consisted of 150 records and the dataset has four features: sepal length and width and petal length and width. The dataset describes three classes of iris, each having 50 records. For this analysis, the randomForest package of the Comprehensive R Archive Network (CRAN) was utilized. The package is written by Liaw and Wiener (2002) and mimics the original Fortran code by Brieman. Other classifiers were applied to the dataset as well. The code developed is shown in Appendix A and Appendix B. The random forest classifier was trained with half of the dataset and is shown in Figure 11 as run in unsupervised mode. The ID3 tree generated is shown in Figure 12, and the neural network is shown in Figure

13. Random Forest was able to classify data with 97.3% accuracy with a KAPPA coefficient of 0.96.

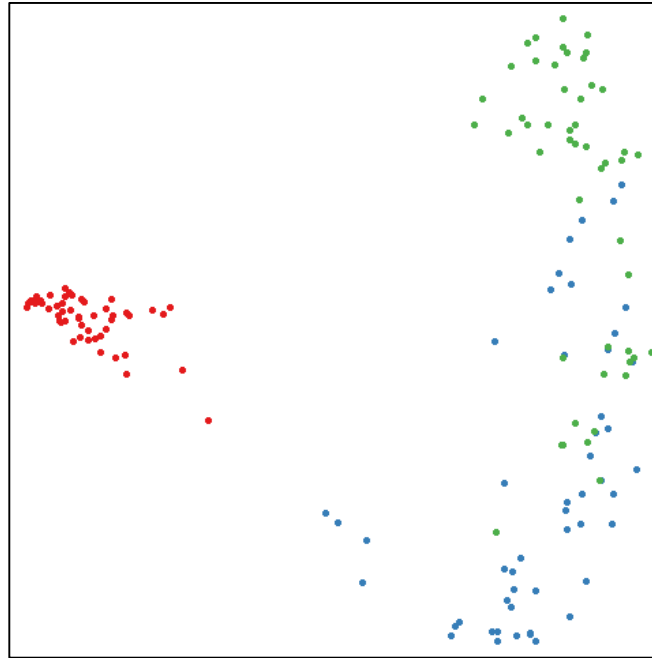


Figure 11. Iris Data Clustered using Random Forest

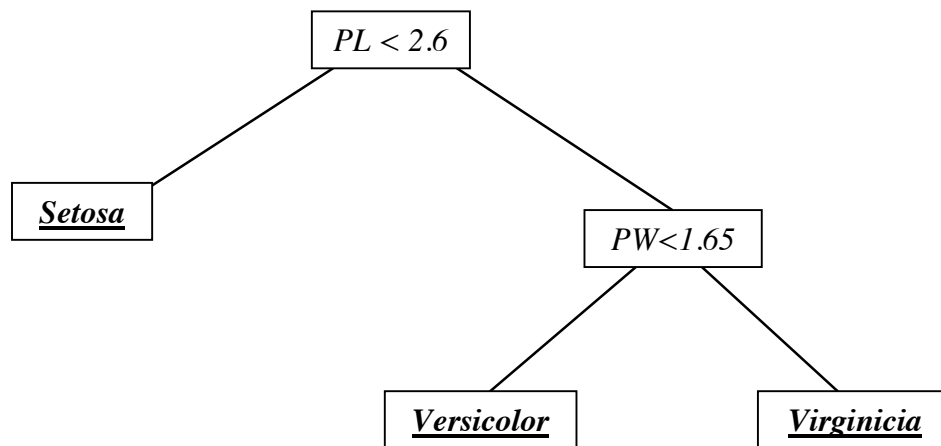


Figure 12. Iris ID3 Tree

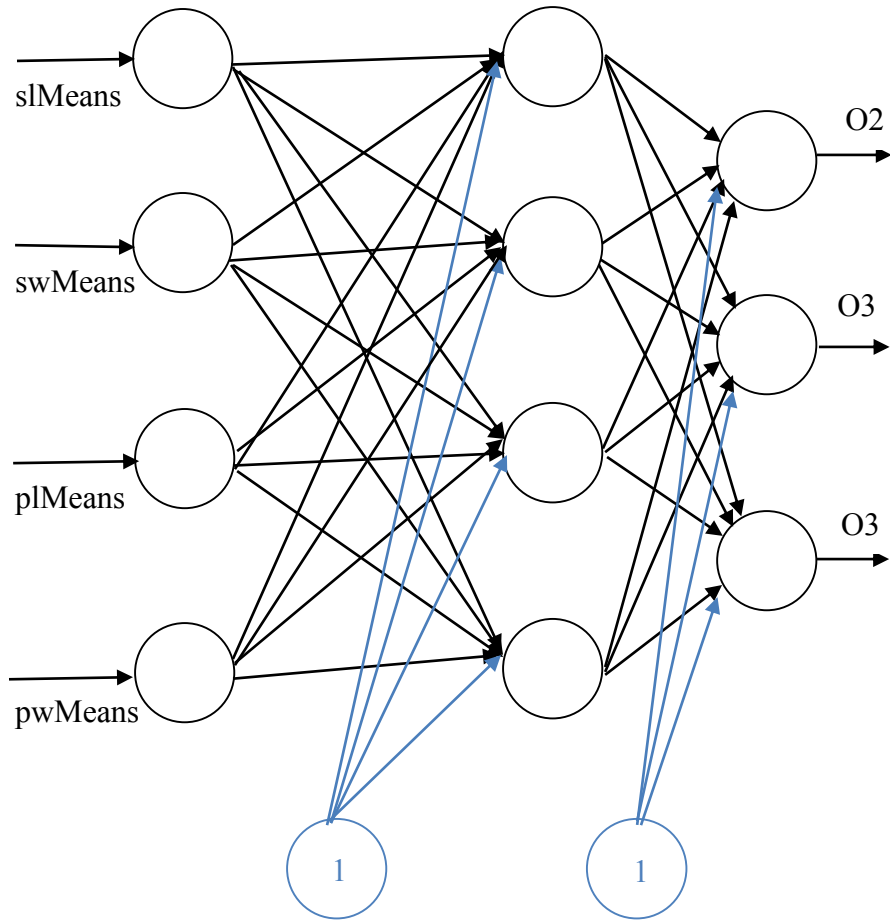


Figure 13. Iris Neural Network

4.2 Remotely Sensed Data

4.2.1 Yellowstone Scene As Random Forest has not been used frequently in remote sensing, the object of this research is to provide some of the first data for assessing its effectiveness with respect to Landsat scenes. Two scenes were considered. The first scene is of Yellowstone National Park acquired on 18 October 2014. The scene is centered around 44 34 5.4761 N latitude and 110 27 36.1818 W longitude and the image is 512 rows by 512 columns. The scene is shown as a color composite of bands 5, 6, and 7 in Figure 14. Forest, Water, Field, and Fire Damage were chosen as classes for

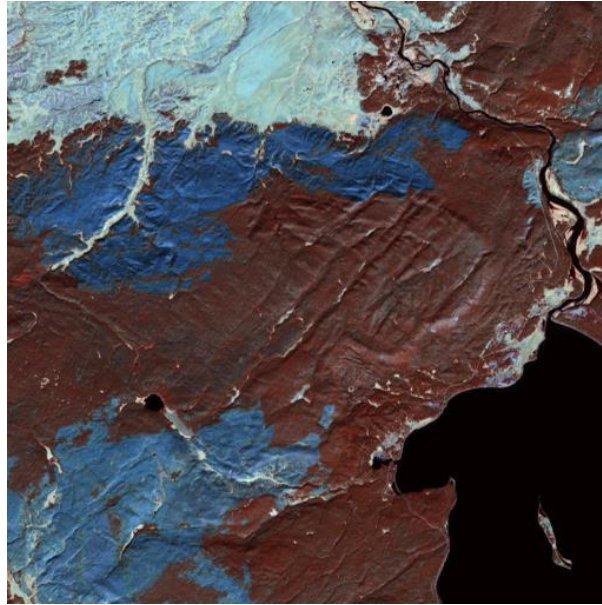


Figure 14. Yellowstone Scene Composite

this scene. We cross-referenced the satellite image with forest fire history from the Yellowstone National Park website confirming that damage from fires named Alum, Dewdrop, and Beach, occurring in 2013, 2012, and 2010 respectively, can all be seen in the figure as deep blue (Wildland Fire Activity in the Park, 2014). It can also be seen that, over time, the reflectance of the fire damage area changes slightly. When training Random Forest for this scene, 200 samples were taken from the Alum fire and 200 samples from the Dewdrop and Beach fires combined to represent the Fire Damage class.

A Random Forest classifier was trained with 200 samples from the field, forest, and water classes and 400 samples from the fire damage class. Bands 1 through 7 were used and spectral signatures were found by taking the band means of each class and are shown in Figure 15. The random forest classifier contained 500 trees. By plotting results

from tests with different m values, six was found to be the optimum m value as shown in Figure 17 and the forest was grown accordingly.

The R code used to train and grow the forest is shown in Appendix C. For comparison, a neural network and support vector machine were created in R using CRAN

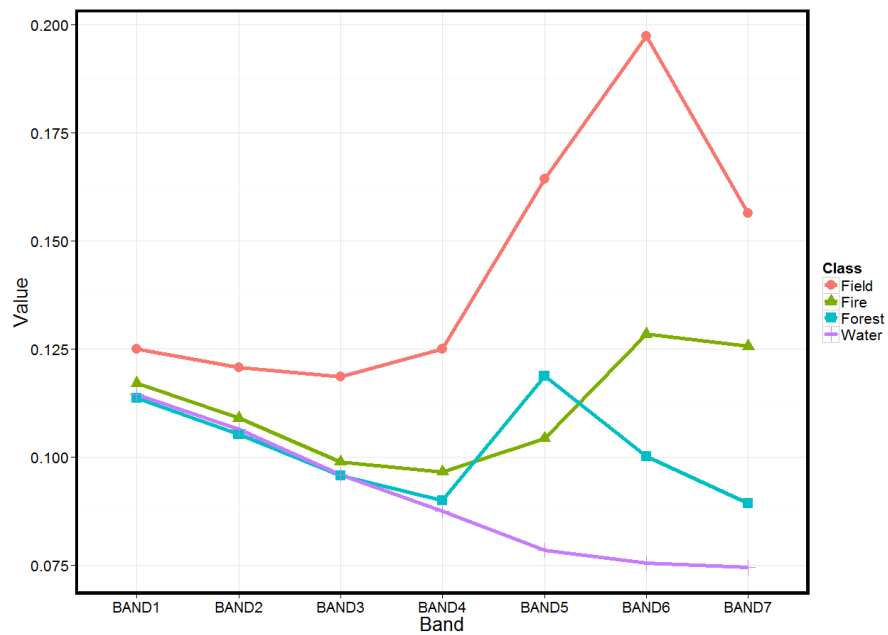


Figure 15. Spectral Signatures - Yellowstone Scene

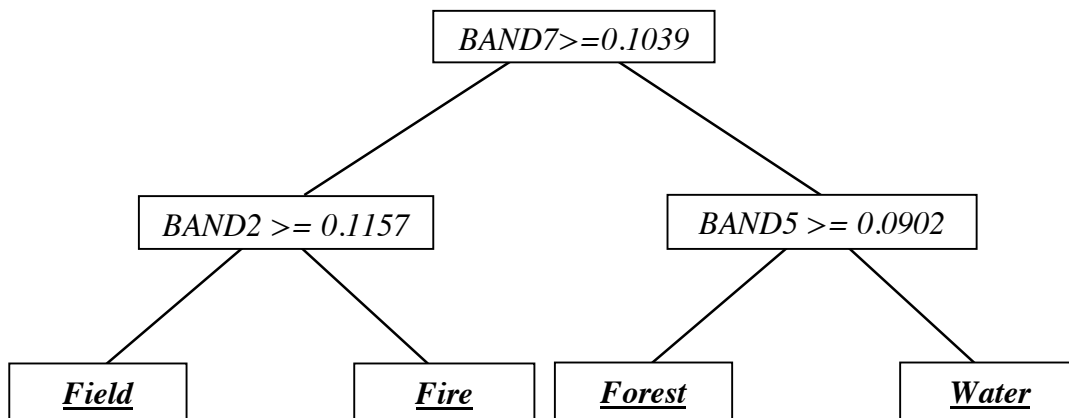


Figure 16. Yellowstone ID3 Tree

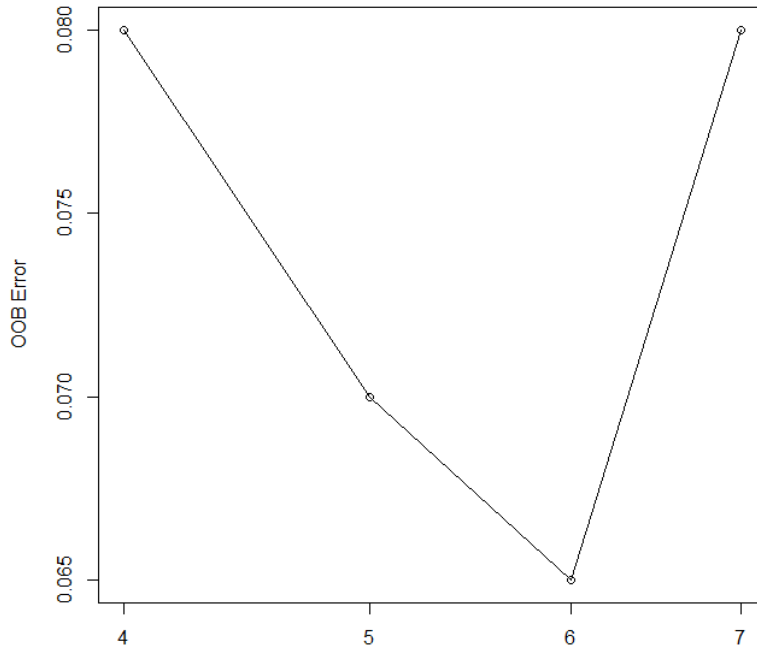


Figure 17. OOB Error as a Function of m

packages (Fritsch et al., 2012, Meyer et al., 2014). The R code used is shown in Appendix D and Appendix E. A minimum distance, maximum likelihood, spectral correlation, spectral angle, and Mahalanobis distance classifiers were all constructed using ERDAS Imagine Version 14 software. A single ID3 tree was also constructed in R to compare the bagged versus non-bagged classifier and is shown in Figure 16. To train the classifiers, the composite image for the scene was displayed on a computer terminal and 200 points from the fire damage class and 100 points from each other class were manually chosen as training data. To assess the accuracy of the classifiers, points were manually chosen from each class. Eighty points were chosen for the fire damage class and forty points were chosen for other classes resulting in a test dataset containing 200

records. Each record was put through each classifier to calculate an error rate. The error matrix for Random Forest is shown in Table 3.

Table 3. RF Confusion Matrix - Yellowstone Scene

	<i>Field</i>	<i>Fire</i>	<i>Forest</i>	<i>Water</i>
Field	40	1	0	0
Fire	0	75	1	0
Forest	0	4	39	2
Water	0	0	0	38

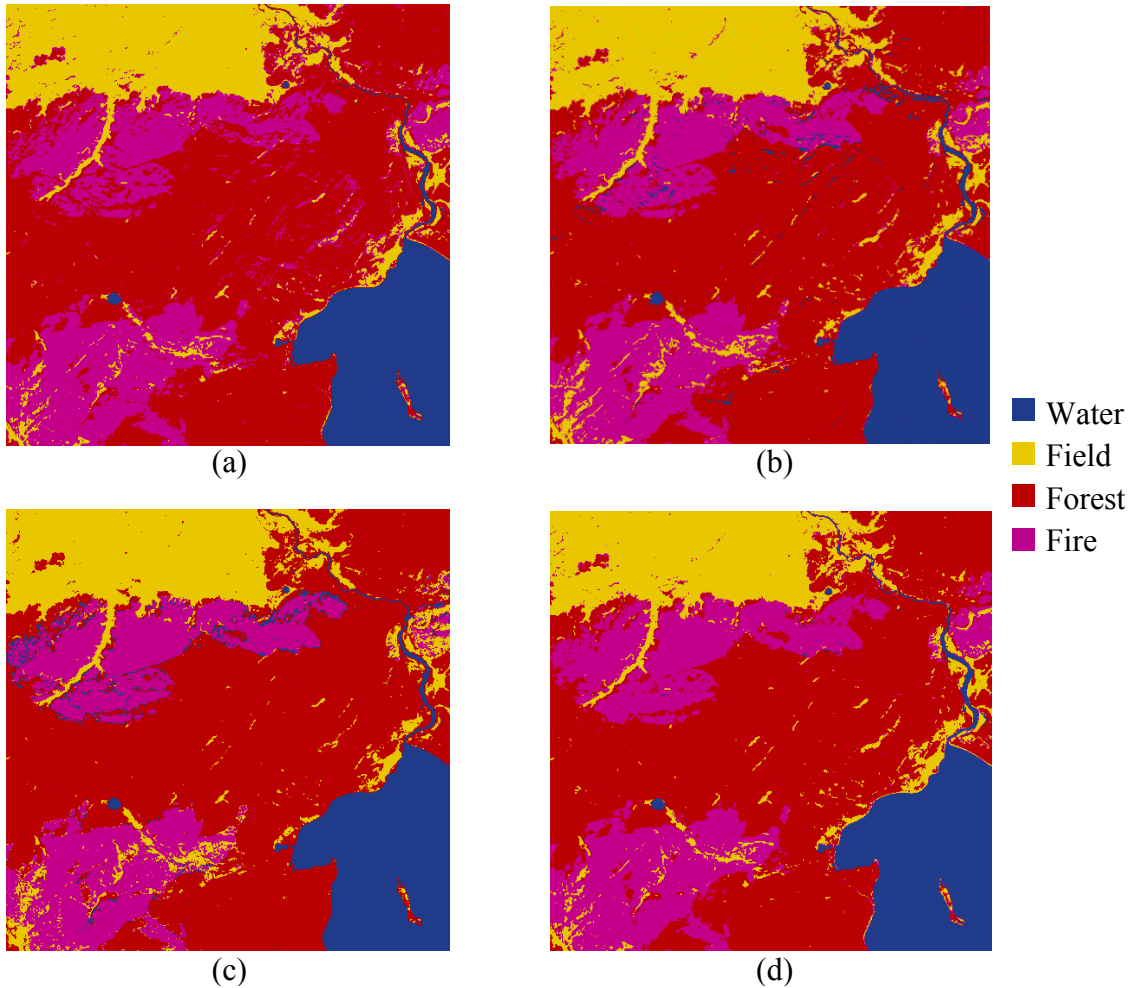


Figure 18. Output Images – Yellowstone Scene. a) Random Forest, b) Minimum Distance, c) Neural Network, d) Support Vector Machine

In this test, Random Forest was outperformed by many of the classifiers. The classified output images of Random Forest and the top three classifiers, minimum distance, support vector machine, and neural network are shown in Figure 18. Breiman and Cutler (2007) note that Random forest is unexcelled in accuracy among current algorithms. Since this was not shown in the Yellowstone scene, another scene was tested.

4.2.2 Mississippi Scene The second scene is of the Mississippi bottomland at 34 19 33.7518 N latitude and 90 45 27.0024 W longitude on 23 September 2014. The Mississippi scene is shown similarly in bands 5, 6, and 7 in Figure 19. Training and test data were acquired in the same manner as the Yellowstone scene. Classes of water, soil, forest, and agriculture were chosen and spectral signatures are shown in Figure 20.



Figure 19. Mississippi Scene Composite

This analysis shows that Random Forest outperforms other classifiers. Details about its performance are described in the confusion matrix in Table 4. The minimum distance, spectral correlation, and spectral angle classifiers performed next best to Random Forest respectively.

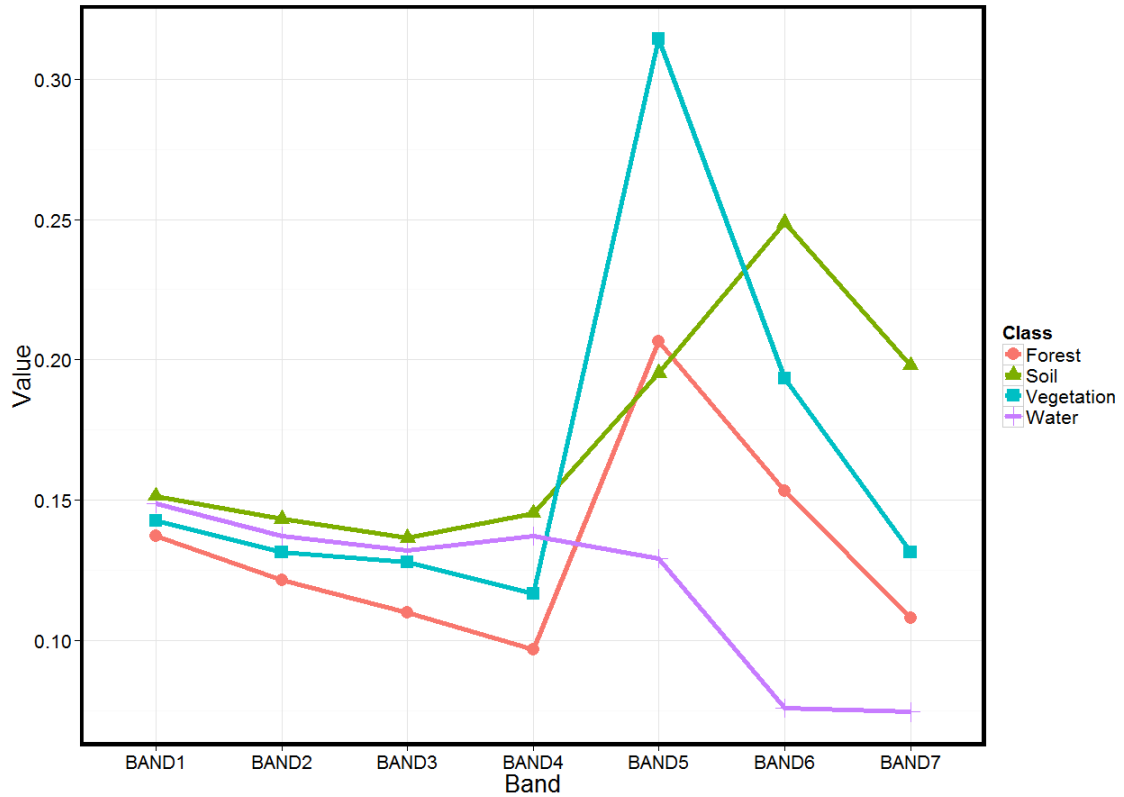


Figure 20. Mississippi Spectral Signatures

Table 4. RF Confusion Matrix - Mississippi Scene

	<i>Forest</i>	<i>Soil</i>	<i>Vegetation</i>	<i>Water</i>
Forest	39	0	5	0
Soil	0	40	0	0
Vegetation	1	0	35	0
Water	0	0	0	40

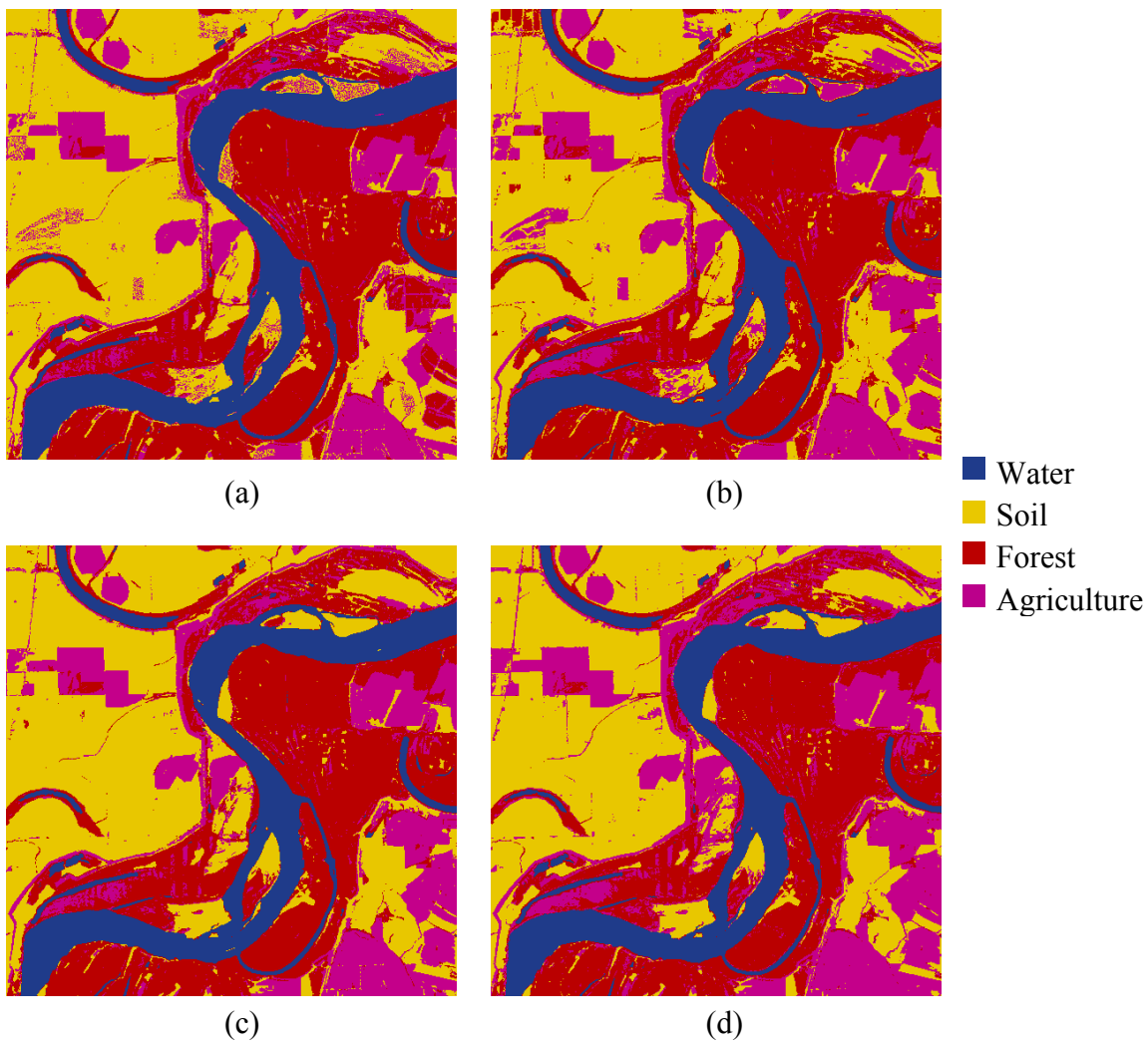


Figure 21. Output Images - Mississippi Scene a) Random Forest, b) ID3 Tree, c) Minimum Distance, d) Spectral Correlation

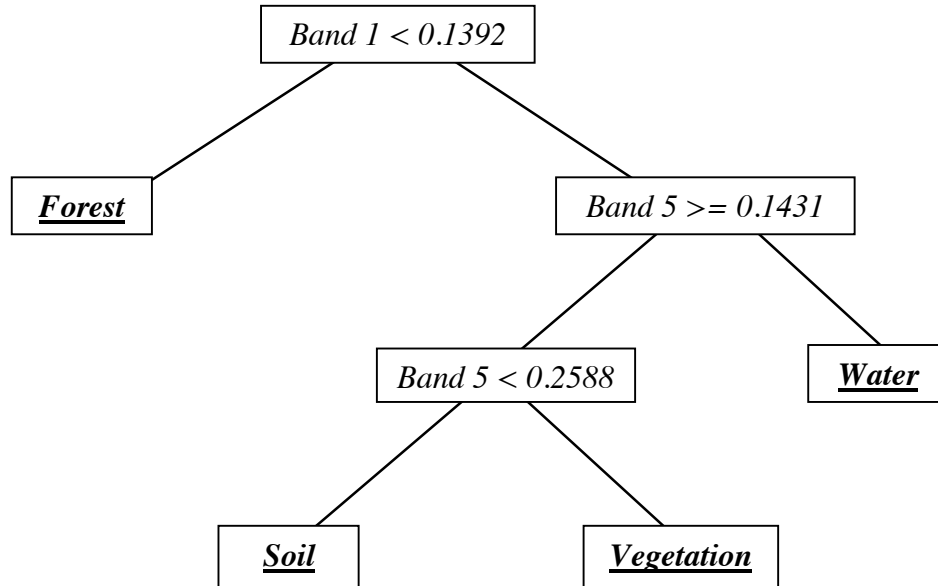


Figure 22. ID3 Tree - Mississippi Scene

In the analysis of the Mississippi scene Random Forest greatly outperformed a single ID3 tree. For reference, the generated ID3 tree is shown in Figure 22 and a comparison between Random Forest and ID3 output images is shown in Figure 21 along with the next top performing classifiers, minimum distance and spectral correlation.

Chapter 5

Conclusions and Future Work

As discussed by the research group at Woods Hole, the use of remote sensing is much broader than land use classification as discussed in this research (Landgrebe, 2003). As research progresses, new ideas are formulated and applied in hopes of improving the process in some way either through efficiency, accuracy, or both. In this project we have shown that the Random Forest algorithm falls under the accuracy category in remote sensing land use classification. Efficiency was not considered in this thesis though could be the primary topic in future research.

We implemented classifiers to compare the accuracy of Random Forest and classified multispectral images with each. This is a new approach for analyzing remotely sensed data as the potential of Random Forest in remote sensing has not yet been fully explored. Classifiers implemented included a neural network, support vector machine, ID3 tree, and Random Forest. We considered three datasets. The UCI Iris dataset was used for initial comparison. After positive results, shown in Table 5, two scenes acquired with Landsat 8 were analyzed – Yellowstone National Park and Mississippi River Bottomland. The scenes were each 512 by 512 pixels and we considered 7 spectral bands. Training and test areas were selected manually using graphics on a computer terminal and the results are documented in Table 6 and Table 7. Typically the field of remote sensing uses other classifiers like those built into ERDAS 2014. For this reason, we also included results from minimum distance, Mahalanobis distance, maximum likelihood, spectral angle, and spectral correlation classifiers for both Landsat scene analyses.

Table 5. Classifier Statistics – Iris dataset

CLASSIFIER	ACCURACY	KAPPA
Random Forest	97.3%	0.96
ID3 Tree	96%	0.94
Neural Network	76.7%	0.65
Support Vector Machine	97.3%	0.96

Table 6. Classifier Statistics – Yellowstone Scene

CLASSIFIER	ACCURACY	KAPPA
Random Forest	96%	0.9448
ID3 Tree	92.5%	0.8953
Neural Network	98.5%	0.9792
Support Vector Machine	99%	0.9861
Spectral Angle	98.5%	0.9792
Spectral Correlation	97.5%	0.9655
Minimum Distance	100%	1.0
Mahalanobis Distance	91.5%	0.8813
Maximum Likelihood	92.5%	0.8954

Table 7. Classifier Statistics – Mississippi Scene

CLASSIFIER	ACCURACY	KAPPA
Random Rorest	96.25%	0.95
ID3 Tree	81.25%	0.75
Neural Network	76.88%	0.6917
Support Vector Machine	86.88%	0.825
Spectral Angle	92.5%	0.9
Spectral Correlation	93.13%	0.9083
Minimum Distance	95%	0.9333
Mahalanobis Distance	83.13%	0.7750
Maximum Likelihood	83.13%	0.7750

We found that Random Forest did well in two Landsat scenes but did not outperform all other classifiers in both. The random forest generated for the Mississippi scene outperformed all other classifiers while the random forest generated for the Yellowstone scene did not. The two scenes have very different terrain. The Mississippi Bottomland area is relatively flat but the Yellowstone National Park area is quite mountainous. As the mountains cast shadows, intensity of pixels of the same class will vary based on the amount light that physically reaches that area. Thus the random forest for the Yellowstone scene was trained with areas in light and shadows making the training data non-homogeneous. This has created errors in the confusion matrix and supports Brieman's noted importance on training with homogeneous data. The random forest for the Mississippi scene was trained with homogeneous data, or no shadows, and results in higher accuracy rates among all classes.

Data mining is a continuously growing field in an age of technology and digital transactional trails. The need for mining of data is increasing and the necessity of efficiency is always present. Random Forest can be applied to many data mining classification applications. It handles large numbers of features very well and, given homogeneous training data, has been shown to outperform many classifiers. Our implementation of Random Forest in R contained a relatively small number of training cases. Consideration must be made if the number of cases and data required exceeds the capacity of memory. Although tree structures are easily saved, a large implementation of Random Forest may require further development to efficiently handle the memory. Other potential future research includes rule extraction. Knowledge extraction from trees has

been researched in detail. Extracting rules from each tree in a random forest is possible based off single tree knowledge extraction. Analysis of rules from every tree to obtain overall rules for the entire forest remains as potential area of research.

References

- C. Apté and S. Weiss, "Data Mining with Decision Trees and Decision Rules," *Futur. Gener. Comput. Syst.*, vol. 13, no. 2–3, pp. 197–210, Nov. 1997.
- K. Bache and M. Lichman, "{UCI} Machine Learning Repository." 2013.
- B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*, 1992, pp. 144–152.
- L. Breiman, "Bagging Predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.
- L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.
- L. Breiman and A. Cutler, "Random Forests," 2007. [Online]. Available: <https://www.stat.berkeley.edu/~breiman/RandomForests/>.
- L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.
- J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educ. Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, 1960.
- R. G. Congalton, "A Review of Assessing the Accuracy of Classifications of Remotely Sensed Data," *Remote Sens. Environ.*, vol. 37, no. 1, pp. 35–46, Jul. 1991.
- C. Cortes and V. Vapnik, "Support-Vector Networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.
- P. Domingos and M. Pazzani, "On the Optimality of the Simple Bayesian Classifier under Zero-One Loss," *Mach. Learn.*, vol. 29, no. 2–3, pp. 103–130, 1997.

- R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, vol. 7. 1973, p. 482.
- R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York, NY: John Wiley & Sons, Inc., 2001, pp. 394–434.
- Stefan Fritsch, Frauke Guenther and following earlier work by Marc Suling (2012).
neuralnet: Training of neural networks. R package version 1.32. <http://CRAN.R-project.org/package=neuralnet>.
- M. Ghose, R. Pradhan, and S. Ghose, “Decision Tree Classification of Remotely Sensed Satellite Data using Spectral Separability Matrix,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 1, no. 5, pp. 93–101, 2010.
- I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene Selection for Cancer Classification using Support Vector Machines,” *Mach. Learn.*, vol. 46, no. 1–3, pp. 389–422, 2002.
- J. Han, M. Kamber, and J. Pei, *Data Mining: concepts and techniques*, 3rd ed. Waltham, MA: Morgan Kaufmann, 2012.
- T. Hastie and R. Tibshirani, “Classification by Pairwise Coupling,” *Ann. Stat.*, vol. 26, no. 2, pp. 451–471, Apr. 1998.
- R. J. Hathaway and J. C. Bezdek, “Recent Convergence Results for the Fuzzy C-Means Clustering Algorithms,” *J. Classif.*, vol. 5, no. 2, pp. 237–247, Sep. 1988.
- W. Y. Huang and R. P. Lippmann, “Neural Net and Traditional Classifiers,” in *Neural Information Processing Systems*, 1988, pp. 387–396.
- E. B. Hunt, J. Marin, and P. J. Stone, *Experiments in Induction*. New York, NY: Academic Press, 1966.

- K. Jearanaitanakij, "Classifying Continuous Data Set by ID3 Algorithm," in *2005 5th International Conference on Information Communications & Signal Processing*, 2005, pp. 1048–1051.
- A. D. Kulkarni, *Computer Vision and Fuzzy Neural Systems*. Upper Saddle River, NJ: Prentice Hall, 2001.
- D. A. Landgrebe, "Multispectral Land Sensing: Where from, where to?," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 414–421, 2005.
- D. A. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*. West Lafayette, IN: Wiley, 2003.
- R. H. Laprade, "Split-and-Merge Segmentation of Aerial Photographs," *Comput. Vision, Graph. Image Process.*, vol. 44, no. 1, pp. 77–86, Oct. 1988.
- X.-B. Li, J. Sweigart, J. Teng, J. Donohue, and L. Thombs, "A Dynamic Programming Based Pruning Method for Decision Trees," *INFORMS J. Comput.*, vol. 13, no. 4, pp. 332–344, Nov. 2001.
- A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. *R News* 2(3), 18--22.
- H. Lin and A. N. Venetsanopoulos, "A Weighted Minimum Distance Classifier for Pattern Recognition," in *Proceedings of Canadian Conference on Electrical and Computer Engineering*, 1993, pp. 904–907.
- L. Luo, X. Zhang, H. Peng, W. Lv, and Y. Zhang, "A New Pruning Method for Decision Tree Based on Structural Risk of Leaf Node," *Neural Comput. Appl.*, vol. 22, no. S1, pp. 17–26, Jul. 2012.

- Mahesh Pal and P. M. Mather, "Decision Tree Based Classification of Remotely Sensed Data," *22nd Asian Conf. Remote Sens.*, 2001.
- David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel and Friedrich Leisch (2014). e1071: Misc Functions of the Department of Statistics (e1071), TU Wien. R package version 1.6-4. <http://CRAN.R-project.org/package=e1071P>.
- Mitra, B. Uma Shankar, and S. K. Pal, "Segmentation of Multispectral Remote Sensing Images Using Active Support Vector Machines," *Pattern Recognit. Lett.*, vol. 25, no. 9, pp. 1067–1074, Jul. 2004.
- J. Nonnemaker and H. S. Baird, "Using Synthetic Data Safely in Classification," in *IS&T/SPIE Electronic Imaging*, 2009, p. 72470G–72470G–11.
- S. K. Pal, R. K. De, and J. Basak, "Unsupervised Feature Evaluation: a Neuro-Fuzzy Approach," *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 366–76, Jan. 2000.
- J. R. Quinlan, "Induction of Decision Trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.
- J. R. Quinlan, "Simplifying Decision Trees," *Int. J. Man. Mach. Stud.*, vol. 27, no. 3, pp. 221–234, Sep. 1987.
- J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the Poor Assumptions of Naive Bayes Text Classifiers," *Proc. Twent. Int. Conf. Mach. Learn.*, vol. 20, no. 1973, pp. 616–623, 2003.
- J. A. Richards, "Analysis of Remotely Sensed Data: The Formative Decades and the Future," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 422–432, 2005.
- F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

- D. E. Rumelhart, J. L. McClelland, and R. J. Williams, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA: The MIT Press, 1986, pp. 318–362.
- S. Senda, M. Minoh, and I. Katsuo, “A Fast Algorithm for the Minimum Distance Classifier and its Application to Kanji Character Recognition,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995, vol. 1, pp. 283–286.
- T. Shi and S. Horvath, “Unsupervised Learning With Random Forest Predictors,” *Journal of Computational and Graphical Statistics*, vol. 15. pp. 118–138, 2006.
- V. N. Vapnik and A. Y. Chervonekis, “Algorithms with Complete Memory and Recurrent Algorithms in the Problem of Learning Pattern Recognition,” *Autom. Remote Control*, vol. 29, no. 4, pp. 606–616, 1968.
- V. N. Vapnik and A. Y. Chervonekis, “On the Uniform Convergence of Relative Frequencies of Event to Their Probabilities,” *Theory Probab. its Appl.*, vol. 16, no. 2, pp. 264–280, 1971.
- W. N. Venables and B. D. Ripley, “Modern Applied Statistics with S Fourth edition by,” *World*, vol. 53, p. 86, 2002.
- P. J. Werbos, *The Roots of Backpropagation: from Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley, 1994.
- G. G. Wilkinson, “Results and Implications of a Study of Fifteen Years of Satellite Image Classification Experiments,” *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 433–440, 2005.
- J. Zhao, B. Thornberg, Y. Shi, and A. Hashemi, “Color Segmentation on FPGA Using Minimum Distance Classifier for Automatic Road Sign Detection,” in *2012 IEEE*

International Conference on Imaging Systems and Techniques Proceedings, 2012,
pp. 516–521.

“Wildland Fire Activity in the Park,” 2014. [Online]. Available:
<http://www.nps.gov/yell/parkmgmt/firemanagement.htm>.

Appendix A. Neural Network for Iris Data in R

```
> irisData <- read.csv("Iris_Data.csv")
> slMeans<-
  c(mean(irisData[1:50,3]),mean(irisData[51:100,3]),mean(iris
    Data[101:150,3]))
> swMeans<-
  c(mean(irisData[1:50,4]),mean(irisData[51:100,4]),mean(iris
    Data[101:150,4]))
> plMeans<-
  c(mean(irisData[1:50,5]),mean(irisData[51:100,5]),mean(iris
    Data[101:150,5]))
> pwMean<-
  c(mean(irisData[1:50,6]),mean(irisData[51:100,6]),mean(iris
    Data[101:150,6]))
> correctoutput <- rbind(c(0,0,1),c(0,1,1),c(1,0,0))
> nnData <- cbind(slMeans, swMeans,plMeans,pwMean,correctoutput)
> colnames(nnData)<-
  c("slMeans","swMeans","plMeans","pwMean","O1","O2","O3")
> formula <- O1+O2+O3~slMeans+swMeans+plMeans+pwMean

> library(neuralnet)
> net <- neuralnet(formula, nnData, hidden=2)
> initialoutput <- compute(net,irisData[,3:6])
> picks<-(c(3,2,1))[apply(initialoutput$net.result,1,which.max)]

> results<-
  data.frame(cbind(Correct=irisData[,2],Prediction=picks))
> results[results==1]<-"Setosa"
> results[results==2]<-"Versicolor"
> results[results==3]<-"Virginicia"
> library(caret)
> confusionMatrix(results$Prediction,results$Correct)
```

Appendix B. Iris Data Classifiers

```
> #IMPORT NEEDED LIBRARIES
> library(randomForest)
> library(caret)

> #IMPORT IRIS DATA FROM CSV FILE
> irisData <- read.csv("Iris_Data.csv")

> #USE HALF OF EACH CLASS FOR BUILDING THE CLASSIFIER
> trainingData<-
  rbind(irisData[1:25,],irisData[51:75,],irisData[101:125,])

> #SETUP AND CONSTRUCT THE RANDOM FOREST WITH 50 TREES
> formula <- Species~SL+SW+PL+PW
> rf<-randomForest(formula, trainingData,mtry=4, ntree=50)

> #PUT ALL DATA THROUGH THE FOREST AND OBTAIN CONFUSION MATRIX
> testResults <- predict(rf,irisData)
> results <- data.frame(Correct=irisData$Species,
  Prediction=testResults)
> confusionMatrix(results$Prediction, results$Correct)

> #USE THE SAME DATA TO BUILD SVM AND GIVE RESULTS
> mySVM<-svm(formula, trainingData)
> testResults <- predict(mySVM,irisData)
> results <- data.frame(Correct=irisData$Species,
  Prediction=testResults)
> confusionMatrix(results$Prediction, results$Correct)

> #USE THE SAME DATA TO BUILD AN ID3 TREE
> library(party)
> tree<-ctree(formula, trainingData)
> testResults <- predict(tree,irisData)
> results <- data.frame(Correct=irisData$Species,
  Prediction=testResults)
> confusionMatrix(results$Prediction, results$Correct)
```

Appendix C. Random Forest and ID3 for Yellowstone Scene in R

```
#IMPORT LIBRARIES NEEDED FOR RANDOM FOREST AND
#CONFUSION MATRIX
> library(caret)
> library(randomForest)

#IMPORT TRAINING DATA AND REFERENCE DATA
#AND CONSTRUCT THE FOREST
> setwd("C:/Users/barrettlowe/Dropbox/ThesisFiles/Yellowstone")
> trainingData<-read.csv("200Samples.csv")
> x<-trainingData[,1:7]
> y<-trainingData[,9]
> rf<-randomForest(x,y, mtry=6, ntree=500)

#IMPORT TEST DATA AND
#PUT IT THROUGH THE FOREST
> testData <- read.csv("HardBandValues.csv")
> testResults <- predict(rf,testData[,1:7])
> results <- data.frame(correct =
+ testData[,9],prediction=testResults)

#GENERATE CONFUSION MATRIX
> confusionMatrix(results$prediction,results$correct)

#Read pgm files in to generate fully classified image
> library(pixmap)
> band1<-read.pnm("Band1.pgm")
> band2<-read.pnm("Band2.pgm")
> band3<-read.pnm("Band3.pgm")
> band4<-read.pnm("Band4.pgm")
> band5<-read.pnm("Band5.pgm")
> band6<-read.pnm("Band6.pgm")
> band7<-read.pnm("Band7.pgm")

> band1<-getChannels(band1)
> band2<-getChannels(band2)
> band3<-getChannels(band3)
> band4<-getChannels(band4)
> band5<-getChannels(band5)
> band6<-getChannels(band6)
> band7<-getChannels(band7)

#Function to extract band values for a given point
> bandValues <- function(xc,yc) {
> + c(band1[xc,yc], band2[xc,yc], band3[xc,yc], band4[xc,yc],
+     band5[xc,yc], band6[xc,yc], band7[xc,yc])
> + }
```

Appendix C. (Continued)

```
> final<-array(,c(512,512))

> for (ix in 1:512){
> + for (jy in 1:512) {
> + val<-bandValues(ix,jy)
> + cl<-predict(rf,val)
> + final[ix,jy]<-cl
> + }
> + }

> classifiedImage<-pixmapGrey(data=final)
> write.pnm(classifiedImage, file="RRandomForestClassified.pgm",
  forceplain=TRUE, type="pgm",maxval=255)

> #GENERATE ID3 TREE TREE
> formula<-CLASS ~ BAND1+BAND2+BAND3+BAND4+BAND5+BAND6+BAND7
> tree<-rpart(formula,trainingData)

> testData <- read.csv("HardBandValues.csv")
> testResults <- predict(tree,testData[,1:7])
> picks <- (c(1,2,3,4))[apply(testResults,1,which.max)]
> picks[picks==1]<-"Field"
> picks[picks==2]<-"Fire"
> picks[picks==3]<-"Forest"
> picks[picks==4]<-"Water"
> results <- data.frame(correct = testData[,8], prediction=picks)
> confusionMatrix(results$prediction,results$correct)
> plot(tree)
```


Appendix D. SVM for Yellowstone Scene in R

```
> library(e1071)
> library(caret)

> trainingData<-read.csv("200Samples.csv")
> x<-trainingData[,1:7]
> y<-trainingData[,9]
> mySVM<-svm(x,y)

> testData <- read.csv("HardBandValues.csv")
> testResults <- predict(mySVM,testData[,1:7])
> results <- data.frame(correct = testData[,9],
  prediction=testResults)
> confusionMatrix(results$prediction,results$correct)
> #Function to extract band values for a given point
> bandValues <- function(xc,yc) {
> + c(band1[xc,yc], band2[xc,yc], band3[xc,yc], band4[xc,yc],
  band5[xc,yc], band6[xc,yc], band7[xc,yc])
> + }

> final<-array(),c(512,512))

> for (ix in 1:512){
> + for (jy in 1:512) {
> + val<-bandValues(ix,jy)
> + cl<-predict(mySVM,rbind(val))
> + final[ix,jy]<-cl
> + }
> + }

> classifiedImage<-pixmapGrey(data=final)
> write.pnm(classifiedImage, file="SVMClassified.pgm",
  forceplain=TRUE, type="pgm",maxval=255)
```

Appendix E. Neural Network for Yellowstone Scene in R

```
> library(neuralnet)
> library(caret)

#training
> ysData<- read.csv("MeanValues200.csv",header=TRUE)
> output<-c(1,0,0,0)
> output<-rbind(output,c(0,1,0,0))
> output<-rbind(output,c(0,0,1,0))
> output<-rbind(output,c(0,0,0,1))
> colnames(output)<-c("Field","Fire","Forest","Water")
> output.names<-colnames(output)
> input.names<-colnames(ysData[,1:7])
> ysData<-cbind(ysData[,1:7],output)
> formula<-Field+Fire+Forest+Water ~
  BAND1+BAND2+BAND3+BAND4+BAND5+BAND6+BAND7
> net <- neuralnet(formula, ysData, hidden=2)

#testing
> ysTest<- read.csv("HardBandValues.csv")
> ysResults <- compute(net,ysTest[,1:7])
> picks<-(c(1,2,3,4))[apply(ysResults$net.result,1,which.max)]
> ysResults<-
  data.frame(cbind(Correct=ysTest[,9],Prediction=picks))
> confusionMatrix(ysResults$Prediction,ysResults$Correct)

#Read pgm files in to generate fully classified image
> library(pixmap)
> band1<-read.pnm("BAND1.pgm")
> band2<-read.pnm("BAND2.pgm")
> band3<-read.pnm("BAND3.pgm")
> band4<-read.pnm("BAND4.pgm")
> band5<-read.pnm("BAND5.pgm")
> band6<-read.pnm("BAND6.pgm")
> band7<-read.pnm("BAND7.pgm")

> band1<-getChannels(band1)
> band2<-getChannels(band2)
> band3<-getChannels(band3)
> band4<-getChannels(band4)
> band5<-getChannels(band5)
> band6<-getChannels(band6)
> band7<-getChannels(band7)
```

Appendix E. (Continued)

```
#Function to extract band values for a given point
> bandValues <- function(xc,yc) {
> + c(band1[xc,yc], band2[xc,yc], band3[xc,yc], band4[xc,yc],
> +   band5[xc,yc], band6[xc,yc], band7[xc,yc])
> + }

> final<-array(c(512,512))

> + for (ix in 1:512){
> + for (jy in 1:512) {
> + #use rbind to create a multi dimensional array as the network
> +   expects
> + val<-rbind(bandValues(ix,jy))
> + cl<-compute(net,val)
> + cl<-(c(1,2,3,4))[apply(cl$net.result,1,which.max)]
> + final[ix,jy]<-cl
> + }
> + }

> classifiedImage<-pixmapGrey(data=final)
> write.pnm(classifiedImage, file="NNetClassified.pgm",
> forceplain=TRUE, type="pgm",maxval=255)
```

Appendix F. Processing and Classification in ERDAS 2014

1. Open .img file in ERDAS
2. Select Home > Inquire > Inquire Box

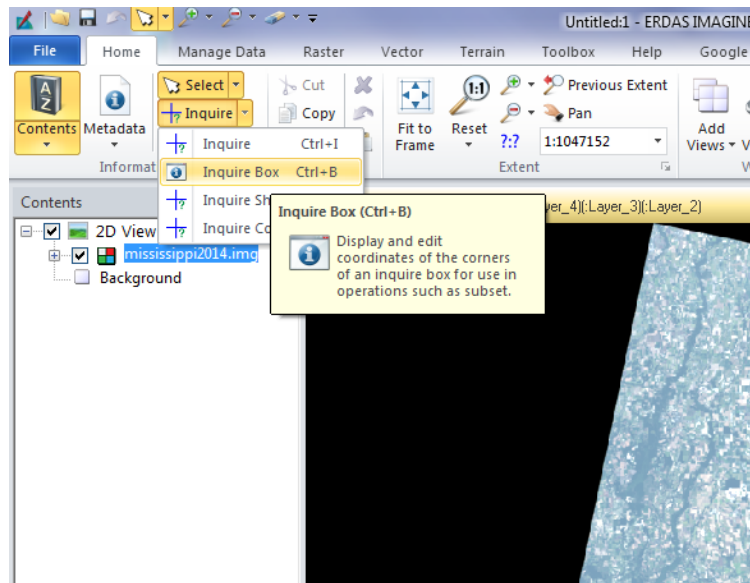


Figure 23. ERDAS Inquire Box

3. Use Inquire Box to select subset – do not close the inquire box window
4. Select Raster > Subset & Chip > Create Subset Image

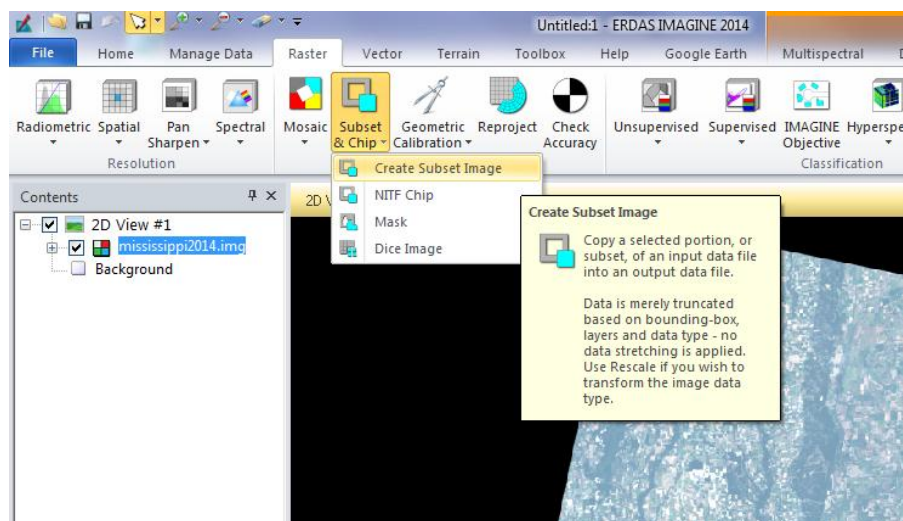


Figure 24. ERDAS Subset Image

5. In the subset window select an output file location, click From Inquire Box, specify layers desired in subset image and click OK. Wait for process to complete
6. Close the original .img file. The new .img file may now be opened for processing

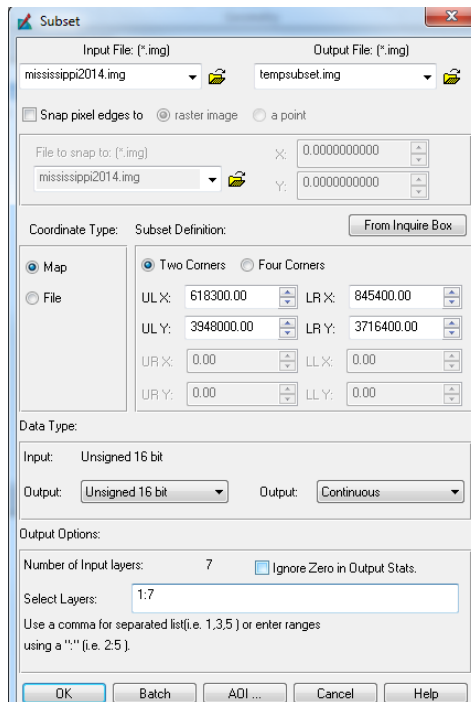


Figure 25. ERDAS Subset Image Window

7. To begin supervised classification select Supervised > Signature Editor
8. Select Raster > Drawing > Insert Geometry and select a desired tool. Use the tool to select training points for one class

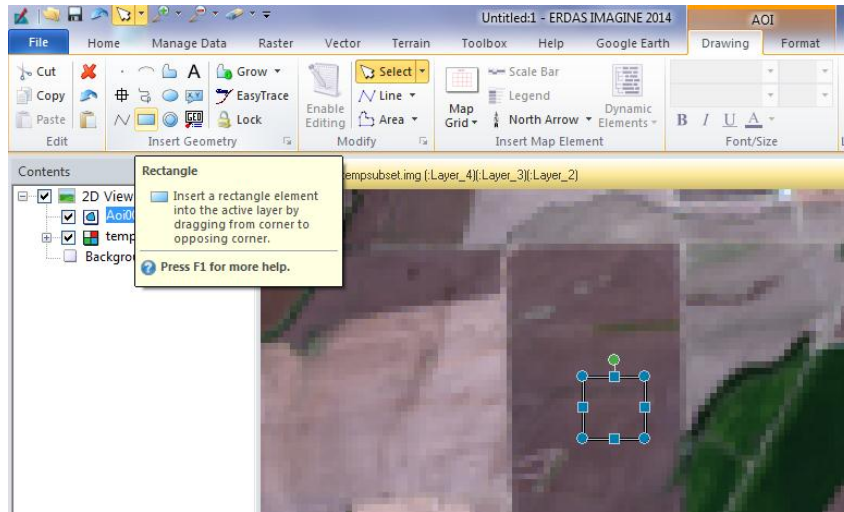


Figure 27. ERDAS Selecting Test Points

9. In the Signature Editor window, select Create New Signature(s) from AOI and specify a class name

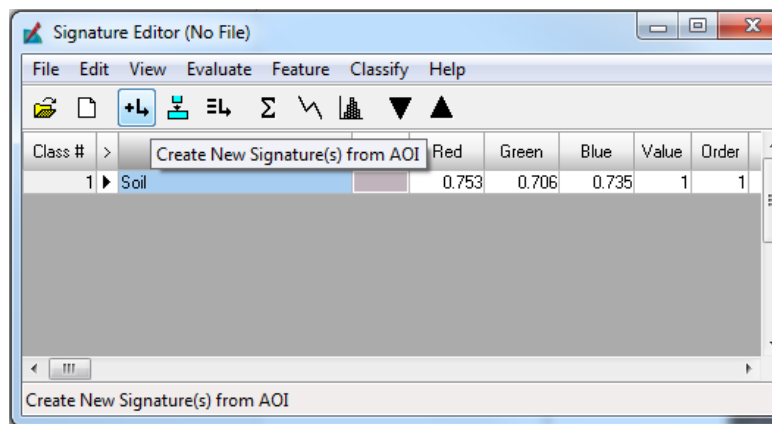
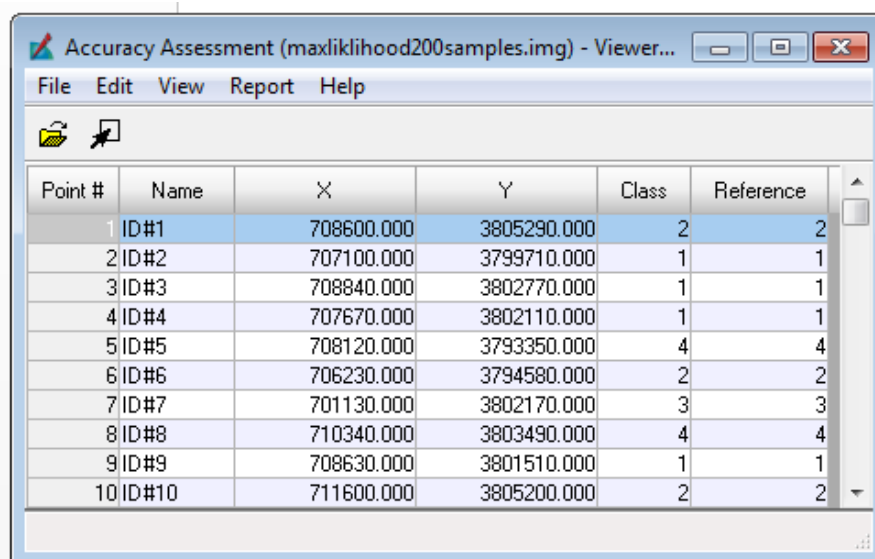


Figure 26. ERDAS Signature Editor

10. Repeat for all other classes, select desired color for each class, and save signature file
11. Select Raster > Supervised > Supervised Classification

12. In the Supervised Classification window, choose the saved signature file, select a name and destination for the classified file, choose a parametric rule (classification technique) and select OK
13. Open the classified .img file
14. Select Raster > Supervised > Accuracy Assessment and open the classified .img file from the Accuracy Assessment window
15. Under Edit select whether you want to select random points for testing or import custom points from a .txt file
16. Using the unclassified file, manually determine the reference class for each point



The screenshot shows the 'Accuracy Assessment (maxliklihood200samples.img) - Viewer...' window. It contains a table with the following data:

Point #	Name	X	Y	Class	Reference
1	ID#1	708600.000	3805290.000	2	2
2	ID#2	707100.000	3799710.000	1	1
3	ID#3	708840.000	3802770.000	1	1
4	ID#4	707670.000	3802110.000	1	1
5	ID#5	708120.000	3793350.000	4	4
6	ID#6	706230.000	3794580.000	2	2
7	ID#7	701130.000	3802170.000	3	3
8	ID#8	710340.000	3803490.000	4	4
9	ID#9	708630.000	3801510.000	1	1
10	ID#10	711600.000	3805200.000	2	2

Figure 28. ERDAS Accuracy Assessment

17. Select Report > Accuracy Report to view the accuracy report, error matrix, and KAPPA statistics. These may then be saved to a file