Automatic Modulation Recognition in the 868 MHz Wireless Network using Tree-Based Methods

by

Ken Lau

B.Sc., The University of British Columbia, 2013

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Statistics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

June 19, 2015

Abstract

Wireless communication systems enable the transfer of data between devices through the transmission of radio signals. These wireless devices often transmit a variety of waveform types called modulation formats. Misidentification of modulation formats between two communicating devices could lead to undesirable delays in data transfer and inefficient energy consumption. The role of automatic modulation recognition (AMR) is the identification of the different modulations of transmitted signals. In this project, we explore tree-based AMR methods in the 868 MHz frequency band. An existing work implements a tree-based classifier that is constructed manually by feature inspection. We extend the recent approach by implementing classification tree (CT) and random forest (RF) classifiers, as well as introducing an expanded list of features. Performance is verified via a simulation at different signal to noise ratios (SNR). Signal data is initially preprocessed, and features are extracted and used to train each classifier. Improvements of 14% and 3% success rates are found for RF and CT, respectively using all features and at a SNR of 1 dB.

Preface

This project is submitted in partial fulfillment of the requirements for a Master of Science in Statistics. It contains work from January 2014 to December 2014 under the supervision of Lutz Lampe and Matías Salibián-Berrera from the Electrical and Computer Engineering and Statistics Departments at UBC.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Wireless communication has become very important in our daily lives as Wifi is pervasive at home, workplace, airports, and street corners. Wifi is aimed at long-distance communication, and used for applications that require a higher bandwidth and data transfer rate compared to most other wireless networks. This project is concerned with a Zigbee network which is similar to Wifi in the sense that Zigbee is also used for wireless communication. The difference is that Zigbee is targeted at short-distance communication and involves lower-powered and lower-data-rate applications. Examples of these applications are home temperature and light sensors, security devices, and wall-mounted switches. These applications do not require a high bandwidth, thus have lower costs to maintain. Devices using Zigbee operate on the 868 MHz frequency band.

We give a simple example of how two devices in a network interact. For simplicity, let us assume that data transfer is unidirectional, that is, signal is strictly sent from one device to another and not in the reverse direction. The bidirectional case is similar, but requires more work to explain. For example, Figure 1.1 shows a home temperature device transmitting a signal to a heater device. When the signal reaches the heater, data is extracted from the signal. For instance, the extracted data might contain information about the current temperature of the room, and so the heater adjusts the amount of electrical energy converted to heat accordingly. In case the reader is interested in wireless networks in general, chapter six of [1] provides detailed explanation of networks.

The home temperature and heater devices are examples of devices that use the 868 MHz frequency band. The devices generally transmit signals of six different waveforms, which are often referred to as modulation types in the Institute of Electronics Engineers (IEEE) literature. The job of the receiving device is to process the signal and identify which of the six waveforms the signal belongs to, so that data can be extracted from the signal. The general name for classifying modulations is automatic modulation recognition (AMR). Classifiers take a set of feature variables and a single class variable, and predict the class variable on future data. Say there is no classifier at

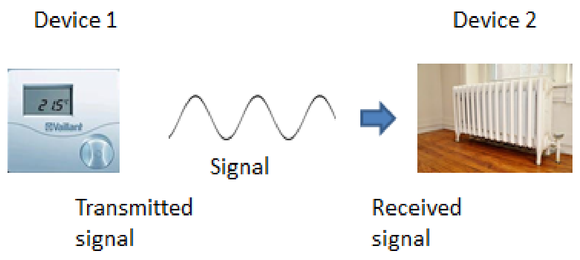

Figure 1.1: Example of signal transfer between two devices. Device 1 is a home temperature device which transmit signals to a heater device which is labeled as device 2.

the receiving device, an arbitrarily poor classifier can be constructed such that it always identify the signals as one out of the six modulations. We implement three different classifiers described in Section 3.

We provide a simple example involving a classifier to predict tomorrow's weather given some atmospheric attributes of today's weather. The purpose of this example is to introduce the idea of machine learning and to ease us into AMR. The data is presented in Table 1.1. The first three columns correspond to the features and each of them describe atmospheric attributes of today's weather. The features are the amount of rain, humidity level, and percentage of sunshine for a particular day. The class variable is the weather condition for the next day given a particular day. A decision tree classifier can be constructed based on observing the pattern between the features and class variable. For example, based on the data, we might predict that there is rain tomorrow given the average amount of rain today was greater than 2mm. An example of a decision tree classifier might look like one given in Figure 1.2. Evidently, the tree appears to be constructed in a fairly ad hoc manner.

In the context of our study, the class variable corresponds to the modulation type. The set of feature variables is extracted from the transmitted signals. The modulation type and feature variables are further described in Sections 2.1 and 2.3 respectively. Figure 1.3 shows that the classifier receives the signal before being processed by the heater device. As the transmitted device can transmit more than one type of modulation, the classifier's job is to identify the correct modulation. In fact, the classifier is integrated within the receiving device. Classifiers are described further in Section 3.1.

A feature-based tree classifier using strategically engineered features is

Rain	Humidity	Sunshine	Weather Tomorrow
1mm	82%	9%	Cloudy
3mm	85%	40%	Rainy
0mm	80%	80%	Sunny
1mm	84%	5%	Cloudy
..	..	..	..

Table 1.1: Weather prediction example. The feature variables are amount of rain, humidity level, and percentage of sunshine today. The class prediction variable is the weather for tomorrow consisting of rainy, cloudy, or sunny.

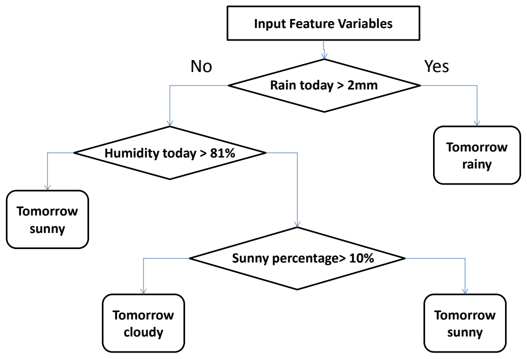

Figure 1.2: A classifier for the weather prediction data. Avg stands for average.

implemented in [2]. This method is explained in 3.2. Similar to the classifier used in the weather prediction example, the features are chosen in a fairly ad hoc manner. Therefore, in this project, we extend the approach by implementing a classification tree that considers many more combinations of features and thresholds in constructing the tree. We further consider the random forest classifier that takes a consensus of predictions from numerous classification trees. The classification tree and random forest classifiers are described in Sections 3.3 and 3.4 respectively. We also implement an expanded list of features that improve the predictive performances of these classifiers described in Section 2.3.3. The papers in [3, 4] provide an overview of common feature variables and classification algorithms applied to AMR tasks. The two papers also summarize the type of features that improve different classifiers in AMR. For example, [4] presents a compact overview of available features and classifiers used in AMR which assists in choosing the appropriate algorithms for intended applications. In this project, we implement classifiers to predict six different waveforms as well as identify signal from noise as described in Section 3.5.

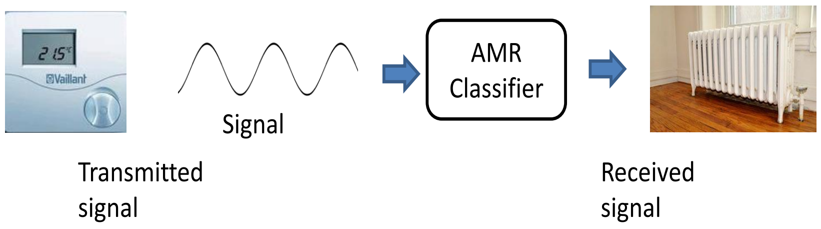

Figure 1.3: Example of signal transfer between two devices with classifier installed.

1.1 Other Related Work

Although this project involves only tree-based methods, we now describe other non tree-based methods that are also often used in AMR tasks. An artificial neural network (ANN) is implemented in [5]. In fact, three ANNs are used in total to classify six types of digital modulation signals. For simplicity, let us identify these modulations using integers from 1 through 6. The method classifies modulations 1 and 2 separately, and modulations 3 and 4 as well as 5 and 6 together as shown in Figure 1.4.

A comparison between a feature-based tree (more description in Section 3.2) and ANN is also described in [5]. A disadvantage of ANN is the interpretation of the model. For example, it is difficult to determine how the prediction would change given we modify the values of some features.

Support vector machines are also used as described in [6]. The method identified 16 different modulations. In addition, [4] provides an overview of many methods in automatic modulation recognition.

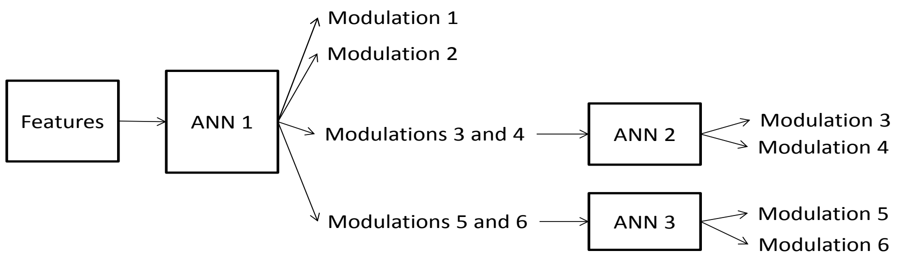

Figure 1.4: The visualization of using three ANNs to classify six modulations.

Chapter 2

Data Simulation

We first describe the transmission of the signals for six different waveforms and corrupting the signals at the receiver. Simulations are performed in the Communications Toolbox of MATLAB [7]. Next, we show how the signals are preprocessed. Finally, we provide details about the features that are extracted from the signals.

2.1 Transmitting Signals

Recall that we are interested in the devices that use the 868 MHz frequency band which are found in the Zigbee network. These devices generally transmit six different modulation types. Three of the modulations include on-off keying (OOK), binary phase-shift keying (BPSK) and offset quaternary phase-shift keying (OQPSK). Each of these have a carrier frequency of 868.3 MHz which is just the frequency of the radio signal as it propagates from the transmitter to receiver device, for example in the introduction section, from the temperature to heater device. The other three modulations are the binary frequency-shift keying (BFSK) operating at carrier frequencies of 868.3 MHz, 868.95 MHz, and $868.03 + b \cdot 0.06$ MHz, where $0 \leq b \leq 9$, respectively. These are used in the wireless meter-bus specification which is a section of the device, and according to this specification, we denote the three modulation types as BFSK-A, BFSK-B, and BFSK-R2, respectively. Figure 2.1 provides a visualization of the differences between three general types of modulations, and where each of the six waveforms in our study resides on. OOK belongs to amplitude shift keying (ASK) where the signal has zero amplitude when the input is zero. Evident from the figure, frequency shift keying (FSK) affects the frequency of the signal for different inputs, whereas phase shift keying (PSK) affects the phase of the signal for different inputs.

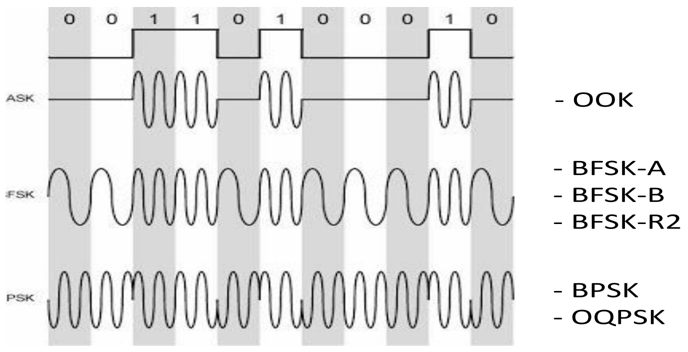

Figure 2.1: Behaviors of the modulations. OOK belongs to amplitude shift keying (ASK), BFSK-A, BFSK-B, and BFKS-R2 belong to frequency shift keying (FSK), and BPSK and OQPSK belong to phase shift keying.

2.1.1 How to Transmit the BFSK-A Modulation

We describe the procedure in transmitting the BFSK-A modulation. The other modulations are transmitted similarly (more details can be found in [2]). We first describe the specification of the BFSK-A modulation. The data rate is 16,384 bits per second which represents the amount of data that can be transferred from one device to another in seconds as data is represented as a stream of bits. Furthermore, Manchester encoding is applied to each bit as shown in Figure 2.2. As shown in (a), a 1 bit becomes symbols of 0 and 1. Note that we are referring the 0 and 1 values as symbols rather than bits, because technically the symbols are generated from the bits. In (b), we see the symbols generated from applying Manchester encoding on a bit sequence of 1, 0, and 1. Applying Manchester encoding, we obtain a symbol rate (R_s) of 32,768 symbols per second (by multiplying the data rate by 2). Moreover, the sampling frequency (f_s) is 6.25 MHz, so the number of samples per symbol (sps) is

$$sps = \left\lceil \frac{f_s}{R_s} \right\rceil = 191 \text{ samples per symbol} \tag{2.1}$$

The samples per symbol is important and determines length of the bit stream. In addition, the observation window size is 512 samples which signifies the number of samples per transmitted signal. Also, the binary frequency shift keying modulates at two different frequencies depending on the symbol. The frequency separation (f_d) defines the difference in the frequency as follows:

$$\begin{aligned} f_1 &= f_c - f_d \\ f_2 &= f_c + f_d \end{aligned} \tag{2.2}$$

where f_1 and f_2 correspond to the two different frequencies with a difference of $2f_d$.

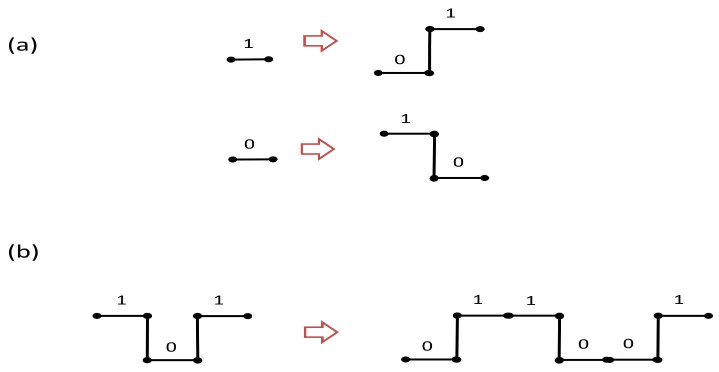

Figure 2.2: Depiction of Manchester encoding on bits of data. In (a), the figure shows that a 1 bit become a 0 and 1 symbol.

Now that we have described the specifications of the BFSK-A modulation, we show how a signal is transmitted. Remember that a single bit is spread into two symbols by the Manchester encoding. Focusing on a single symbol, the digitized transmitted signal ($s[k]$) consists of two forms:

- If the symbol is 0, then

$$s[k] = cos\left(2\pi f_1 \frac{k}{f_s}\right) + jsin\left(2\pi f_1 \frac{k}{f_s}\right). \tag{2.3}$$

- If the symbol is 1, then

$$s[k] = cos\left(2\pi f_2 \frac{k}{f_s}\right) + jsin\left(2\pi f_2 \frac{k}{f_s}\right). \tag{2.4}$$

where $k = 0, 1, 2, \ldots, 190$ for a length of 191 because the samples per symbol is 191. Since the observation window size is 512 samples, we only need 2 bits of data for a total of 4 symbols to transmit one signal.

To transmit a signal, we randomly generate two bits of data as in 0 or 1, and then apply Manchester encoding to obtain 4 symbols. For each symbol we apply the equations 2.3 and 2.4 to obtain 764 samples (4 symbols by 191 samples per symbol). We then take the first 512 samples as described by our window size of 512 samples. We re-use the $s[k]$ notation and refer to it as a single the transmitted signal for samples $k = 1, \ldots, 512$. See Figure 2.3 for an example of a transmitted BFSK-A signal with data bits 0, 0 and symbols 0, 1, 0, 1.

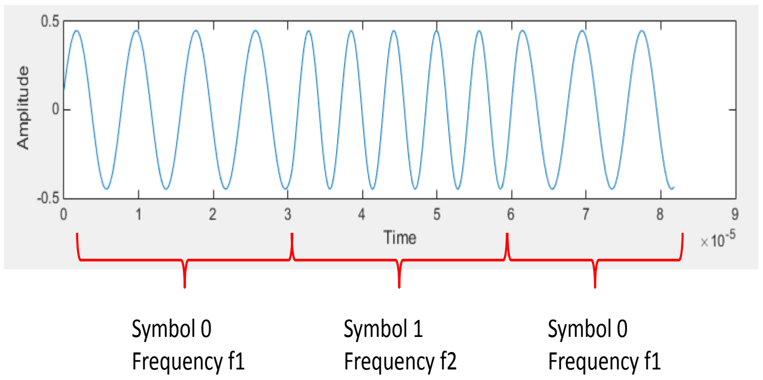

Figure 2.3: Example of transmitted BFSK-A signal where the data bits are 0, 0 and the symbols are 0, 1, 0, 1.

2.2 Obtaining the Received Signal

At the receiver, the signals are mixed to a low intermediate frequency. One reason in using an intermediate frequency is to improve the performance in which a radio receiver identifies a signal. In other words, it is useful in the encoding and decoding of the radio signals in the stages of amplifiers, filters, and detectors. It does not directly affect the automatic modulation recognition (AMR) task we are studying, but we should be consistent with how the rest of the device is integrated, and use the same intermediate frequency throughout. Therefore, for our numerical results presented in Section 4, we apply down-conversion on the carrier frequency (f_c) by 867.3 MHz, i.e., the intermediate frequency for OOK modulation would be 1 MHz. Down-conversion means subtracting the frequency by 867.3 MHz. Furthermore, if we denote the intermediate frequency by f_{if}, then $f_{if} = 1$ replaces f_c in Equation 2.2. The signal at intermediate frequency is sampled with sampling frequency f_s, resulting in the discrete-time signal $r[k]$, where k is the discrete-time index. As in [2], we apply a sampling frequency $f_s = 6.25$ MHz. The received signal is expressed as

$$r[k] = gs[k] + n[k] \ , \tag{2.5}$$

where $s[k]$ is the subtracted and sampled transmitted signal, $n[k]$ is white Gaussian noise with variance σ_n^2, and g is a fading channel gain. An example of the fading channel gain arises from objects obstructing the direct signal path between the transmitter and receiver, and is expressed as follows:

$$g = |c_1|e^{i2\pi c_2} \tag{2.6}$$

where c_1 is a value sampled from a standard normal distribution and c_2 is sampled from a uniform distribution with parameters 0 and 1. Denoting the variance of $s[k]$ by σ_s^2, we define the signal to noise ratio (SNR) as σ_s^2/σ_n^2. Further details regarding the six communication signals are provided in [2].

The task of AMR is to determine from the received signal $r[k]$ which of the six modulation formats has been used in $s[k]$. To this end, features are computed from $r[k]$ and then used in a classifier. The features and classifiers applied in previous and our AMR approaches will be introduced next.

2.3 Feature Extraction

This section describes the preprocessing of the received signals, followed by how features are computed.

2.3.1 Preprocessing

The task of AMR begins by preprocessing N samples of the sampled received signal in 2.5. Following [2], we adopt $N = 512$ and generate via discrete Fourier transform (DFT) the spectral representations

$$R[k] = \sum_{i=0}^{N-1} \frac{r[i]}{\mu_r} e^{-j2\pi\frac{ik}{N}} \ , \quad k = 0, \ldots, N-1 \tag{2.7}$$

and

$$A[k] = \sum_{i=0}^{N-1} \frac{|a[i]|}{\mu_a} e^{-j2\pi\frac{ik}{N}} \ , \quad k = 0, \dots, N-1 \ , \tag{2.8}$$

where ($\mathcal{H}\{\cdot\}$ denotes Hilbert transformation and j is an imaginary number.)

$$a[i] = r[i] + j\mathcal{H}\{r[i]\} \ , \quad i = 0, \dots, N-1 \tag{2.9}$$

is the analytic received signal and

$$\mu_a = \frac{1}{N}\sum_{i=0}^{N-1} |a[i]|, \ \ \mu_r = \frac{1}{N}\sum_{i=0}^{N-1} |r[i]|. \tag{2.10}$$

According to the Matlab documentation [8], the Hilbert transformation returns the original sequence with a 90° phase shift. This simply means, for example, that sines are transformed into cosines. The equation, $A[k]$, is the DFT of the normalized absolute analytic signal. The analytic signal is required in computing some of the features such as the transformation based features in Section 2.3.3.

2.3.2 Features From [2]

Let the features required in the feature-based binary tree in [2] be denoted as m_1, m_2, m_3, m_4, and m_5. Feature m_1 is computed as follows,

$$m_1 = max(|A[k1]|) + max(|A[k2]|) \tag{2.11}$$

where $1 \leq k_1 \leq 7$ and $25 \leq k_2 \leq 27$. Next, we compute m_2 and m_3:

$$m_2 = \frac{A[k_2]}{max(A[k_1])} \tag{2.12}$$

$$m_3 = \frac{|R[k_4]|}{R[k_3]} \tag{2.13}$$

where k_3 is defined as the index of the maximum value of $|R[k]|$ and k_4 is the second largest value with at least four indices away from k_3. Finally,

$$m_4 = \left\lfloor \frac{k_3 + k_4 + 1}{2} \right\rfloor , \tag{2.14}$$

and

$$m_5 = \frac{\sum_{k=m_4-B_2/2}^{m_4+B_2/2} |R[k]|}{\sum_{k=m_4-B_1/2}^{m_4+B_1/2} |R[k]|} \tag{2.15}$$

where $B_1 = 36$ and $B_2 = 18$.

2.3.3 Additional Features

In this section, we implement additional features for the AMR. The type of features we describe are commonly used in AMR problems, e.g. cf. [4, 9–11].

Transformation Based Features

The first set of proposed features is related to the spectral representation of the analytical signal. More specifically, we compute the maximal squared magnitude frequency component

$$\gamma_{\max} = \max_{k=0,\ldots,N-1} \frac{1}{N} |A[k]|^2. \tag{2.16}$$

Since $\gamma_{\max}$ corresponds to amplitude variation of signals [4], this feature is useful in discriminating between OOK and the PSK/FSK modulations, because OOK is the only amplitude shift keying modulation. Therefore, the only modulation we expect amplitude variation is of course, OOK. Furthermore, the maximum value of the DFT of the 2^{nd} and 4^{th} power of the analytical form is computed, i.e.,

$$\Gamma_n = \max_{k=0,\ldots,N-1} \frac{1}{N} \sum_{i=0}^{N-1} |a[i]|^n e^{-j2\pi \frac{ik}{N}} , \tag{2.17}$$

for $n = 2$ and $n = 4$. The features, Γ_2 and Γ_4 are useful in classification of PSK signals. In particular, these features discriminate BPSK and OQPSK modulations from the remaining modulations. Similar to the amplitude variation case, we expect that the only modulations to obtain phase variation are BPSK and OQPSK.

Higher-Order Cumulant Features

Each modulation has a different in-phase and quadrature (I-Q) component which is often represented by a constellation diagram. For example, Figure 2.4 depicts constellation diagrams for BPSK and OQPSK. The x and y axes represent the in-phase and quadrature components of the modulation respectively. For instance, the two circles labeled with 0 and 1 for BPSK represent the symbols, and indicate the two phases with a shift separation of $180\,^\circ$. The four circles of the constellation diagram for OQPSK represent its four symbols and the $90\,^\circ$ shift separation. In fact, the I-Q components are related to the real and imaginary components of the signal [9]. The in-phase component is shown on the x-axis which also resembles the real component of

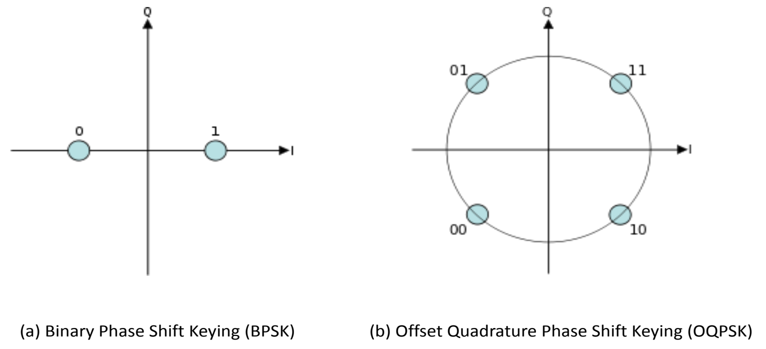
Q
I
0
1
(a) Binary Phase Shift Keying (BPSK)
Q
I
01
11
00
10
(b) Offset Quadrature Phase Shift Keying (OQPSK)

Figure 2.4: Constellation diagrams for BPSK and OQPSK.

the modulation. Similarly, the quadrature component is shown on the y-axis which also resembles the complex component of the modulation. This leads us to why we are computing the higher-order cumulant (HOC) features. The HOC features are based on the non-linear combinations of the real and imaginary components of the analytical signal as shown in Equation 2.18 below. By computing the higher-order cumulant (HOC) features, we can capture the real and imaginary parts of the signal, which in turn discriminates different modulations based on the I-Q components and constellation diagrams.

At this point, we calculate the real and imaginary parts of the analytical signal from Equation 2.9, and denote either parts as $x_i[0], x_i[1], \ldots, x_i[N-1]$. A signal vector is represented as $X_i = \{x_i[0], x_i[1], \ldots, x_i[N-1]\}$ for some i. For all combinations of $X_i, i = 1, \ldots, 4$ based on the real or imaginary parts of the analytical signal, we compute the following equations:

$$\begin{aligned} C_{X_1X_2} &= \frac{1}{N}\sum_{k=0}^{N-1} x_1[k]x_2[k] \\ C_{X_1X_2X_3} &= \frac{1}{N}\sum_{k=0}^{N-1} x_1[k]x_2[k]x_3[k] \\ C_{X_1X_2X_3X_4} &= \frac{1}{N}\sum_{k=0}^{N-1} x_1[k]x_2[k]x_3[k]x_4[k] - C_{X_1X_2}C_{X_3X_4} \\ &\quad - C_{X_1X_3}C_{X_2X_4} - C_{X_1X_4}C_{X_2X_3} \end{aligned} \tag{2.18}$$

$X_i, i = 1, \ldots, 4$ can be either the real or imaginary signal vector of the normalized analytical signal $a[k]/\mu_a$. As an example, we can compute the real part of the analytical signal and denote it as a signal vector X_1, and imaginary part as X_2. Using X_1 and X_2 we compute $C_{X_1X_2}$. The cumulants are also called the time-averaging approximations of the second, third, and fourth order cumulants.

Magnitude Spectrum

The last set of proposed features consists of the magnitude spectrum, $|R[k]|$ for frequency indexes of $k = 45$ to $k = 160$. These features are particularly useful in detecting the BFSK-B modulation, as it is modulated at a higher carrier frequency than the remaining modulations. Figure 2.5 depicts the magnitude spectrum for each of the modulations at a SNR of 11 dB. At the best of our knowledge, the collection of these features has not been used in any AMR tasks, therefore we did not expect a major impact in terms of performance on the classifiers. However, our analysis in Section 4.2 begs differently. It turns out that many of the magnitude spectrum features contributed towards the random forest classifier based on Figure 4.6. We summarized all the features in the Table 2.1.

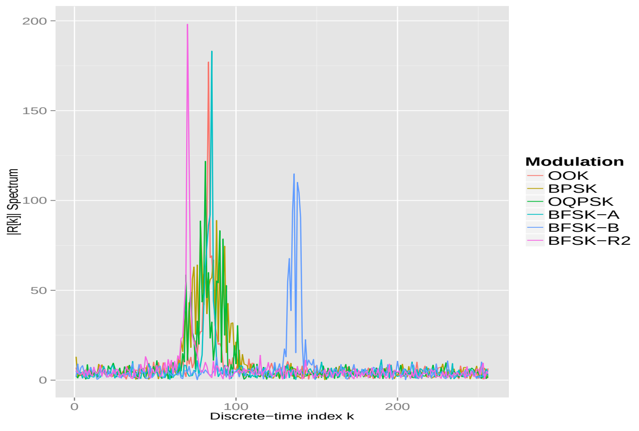

Figure 2.5: Magnitude spectrum, $|R[k]|$ at SNR of 11 dB. The x-axis corresponds to the discrete-time index.

Table 2.1: Summary of feature variables.

Description	**Features**
Features from [2]	m_1, m_2, $\ldots$, m_5
Transformation-based	$\gamma_{\max}$, Γ_2, Γ_4
Higher-order cumulant	C_{II}, C_{IQ}, C_{QQ}, C_{III}, C_{IIQ}, C_{IQQ}, C_{QQQ}, C_{IIII}, C_{IIIQ}, C_{IIQQ}, C_{IQQQ}, C_{QQQQ}
Magnitude spectrum	$\{\lvert R[k]\rvert \,\vert\, k = 45, \ldots, 160\}$

Chapter 3

Implementation

We now turn to the use of the features in the classification methods. We first briefly discuss the choice of a training data set and review the implementation of the feature-based tree (FBT) from [2]. Then we present the proposed classification tree (CT) and random forest (RF) classifiers. Finally, we describe the modifications required in order to detect pure white noise signals.

3.1 Training Data

In the introduction, we mentioned that the class variable is the modulation type, while the feature variables are just the ones summarized in Table 2.1. Let us recall the data set from the simple weather prediction example shown in Table 1.1. This data set represents the training data, as the decision tree classifier was constructed on the basis of this data set. In Section 4, we compare the three classifiers: FBT, CT, and RF using a training data set generated from a simulation, obtaining P values for each feature variable per modulation and SNR value. We provided a visualization example of the process of generating a single observation in Figure 3.1. The signal is first transmitted from the device, and then noise is added to the signal to get our received signal, $r[k]$. At this point, $r[k]$ is preprocessed, and features are extracted from the signal. So, in the example, the received signal is first obtained by transmitting a signal using an OOK modulation, and then noise is added to the signal in order to obtain a SNR of 10 dB. Using the received signal and some pre-processing steps as described in Section 2.3.1, all the features are computed. This represents a single observation. This same process is performed P times for every modulation and SNR. Section 4 describes the specification of P and range of SNR considered. We provided an example of the data in Table 3.1. The modulation column corresponds to the class variable, while the columns to the right of the modulation column correspond to the feature variables.

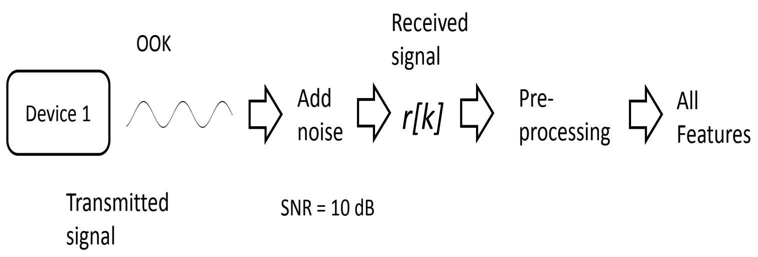

Figure 3.1: Process of generating one observation of the training data.

Table 3.1: Example of training data. The modulation corresponds to the class variable, and the columns to the right of modulation from m_1 to R[160] correspond to the feature variables.

Modulation	$\boldsymbol{m_1}$	$\boldsymbol{m_2}$	$\boldsymbol{m_3}$	**...**	**R[160]**
OOK	190	0.05	0.3	...	2.5
OOK	186	0.12	0.21	...	1.8
OOK	172	0.08	0.25	...	3.1
..	..	..	..	..	..
BFSK-R2	15.3	0.64	0.03	...	5.4
BFSK-R2	14.4	0.54	0.08	...	3.3
BFSK-R2	89.7	0.35	0.12	...	2.48

3.2 Feature-Based Binary Tree

The feature-based binary tree (FBT) classifier utilizes features from [2], and selects thresholds that discern different modulation types. The construction of the FBT is very similar to the decision tree classifier for the weather prediction example described in the introduction section. The first split of the tree is chosen based on the feature m_1. Next, a threshold value of m_1 must be determined. Figure 3.3 shows the distribution of values of m_1 using a boxplot, where the red lines indicate the minimum and maximum values of BPSK and BFSK-A, and the purple line shows the gap between the min and max of BPSK and BFSK-A respectively. It is easy to see that m_1 values above a certain threshold discern OOK and BPSK from the remaining modulations. Therefore, it makes sense to choose a value that is within the gap formed by the minimum m_1 value of OOK or BPSK and the maximum m_1 value of OQPSK, BFSK-A, BFSK-B, and BFSK-R2 as shown by the

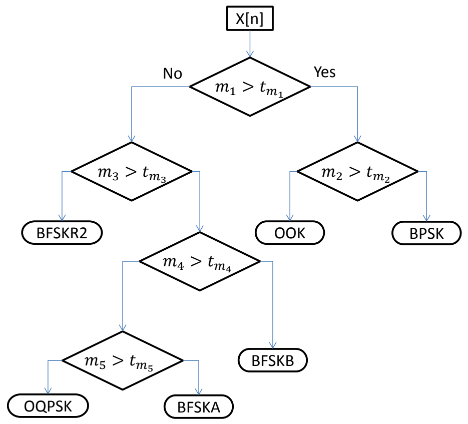

Figure 3.2: Flowchart of feature-based tree depicting features and decision thresholds at interior nodes. Modulation types reside at leaf nodes.

purple line. In [2], not all the modulations were used to determine this gap, and only BPSK and BFSK-A were chosen for m_1.

In order to determine the threshold, a worst-case analysis between m_1 values of BPSK and BFSK-A is performed. Figure 3.4 presents the maximum and minimum value of m_1 for BPSK and BFSK-A respectively. Similar to the boxplot visualization, the gap between the min value of m_1 of BPSK is compared with the max value for BFSK-A using a purple line. As we can see from the figure, as SNR increases, the gap shrinks. The threshold is chosen as the value of m_1 where the gap reaches zero. In [2], not all the modulations were used to determine this gap, and only BPSK and BFSK-A were chosen for m_1. Figure 3.4 depicts the worst-case analysis between m_1 values of BPSK and BFSK-A. The gap reaches zero when the two lines in the plot intersects. The intersection appears approximately at 4.7 dB with a threshold of $t_{m_1} = 47$. Consequently, signals with $m_1 > 47$ are classified as either OOK or BPSK, while signals with $m_1 \leq 47$ are classified as one of the remaining modulations. The thresholds for $m_2, \ldots, m_5$ are determined similarly, see [2] and Table 3.2. Additional details for the threshold determination can be found in [2]. The tree is presented in Figure 3.2. Finally, a prediction is made by traversing down the tree until a leaf node is reached.

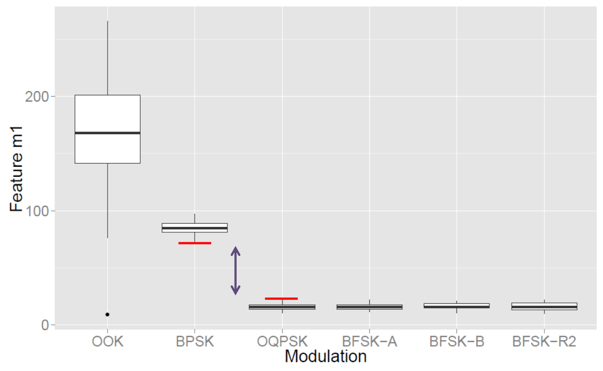

Figure 3.3: Boxplot visualization of m_1 for each modulation.

	Feature	**Threshold**
1	m_1	47.23
2	m_2	0.42
3	m_3	0.23
4	m_4	89.68
5	m_5	0.71

Table 3.2: Thresholds determined by FBT.

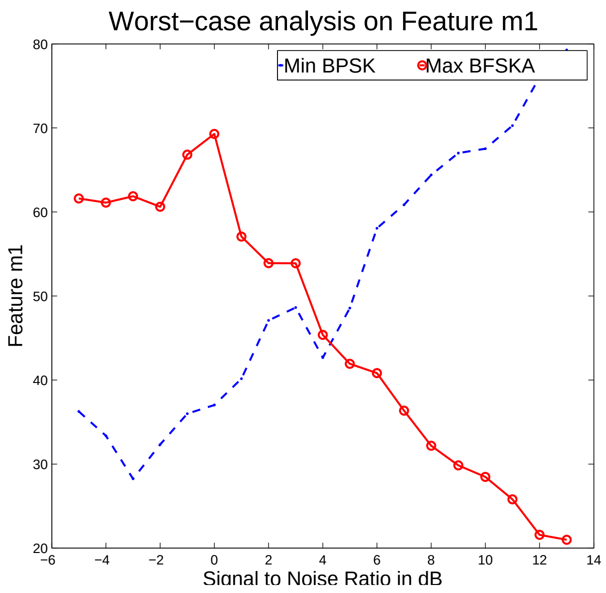

Figure 3.4: Worst-case analysis plot of m_1. Solid line corresponds to the maximumim of m_1 of BFSK-A. Dashed line corresponds to the minimum of m_1 of BPSK.

3.3 Classification Tree

The first classifier we propose is a classification tree (CT). Similar to the FBT, it also selects thresholds using a set of features. However, features and thresholds are selected automatically based on maximizing homogeneity in modulation types at nodes. Recall that a prediction using FBT is made by traversing down the tree until a leaf node is reached. Intuitively, increasing homogeneity at leaf nodes also increases predictive accuracy as we show in the following example. Consider Figure 3.5 where the parent node of a tree consists of blue and red balls. Moreover, assume that a split can only be made by a vertical line. Indeed, the split shown by the dashed line splits the balls into two groups so that each child node only contains one of each color. If we draw a ball from the left child, we could predict correctly that the ball is blue 100% of the time. Similarly, for the right child. However, without splitting into the child nodes, and by randomly picking a ball from the parent node, we are unsure whether it is a red or blue ball. This example illustrates the importance of splitting by homogeneity and optimizing predictive performance.

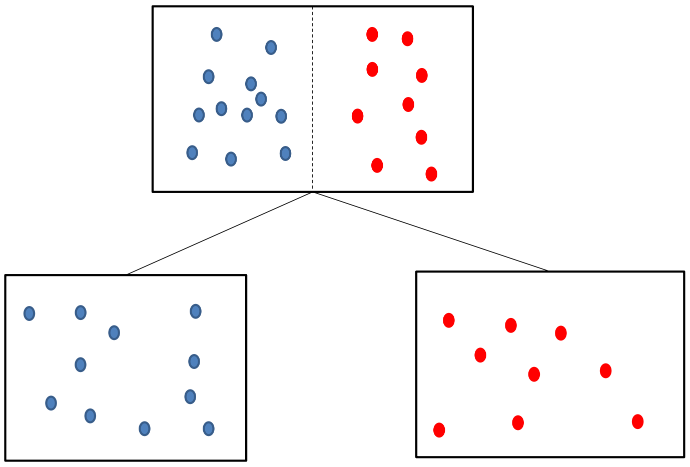

Figure 3.5: Intuitive example of tree split. There are two groups: red and blue balls.

3.3.1 Classification Tree Implementation

Let us denote the training data set as $\mathscr{D} := \{(y_i, \mathbf{x_i})\}_{i=1}^T$ where y_i is a modulation outcome, $\mathbf{x_i} = (x_{i,1}, x_{i,2}, \ldots, x_{i,M})^T$ are the features, and T is the total number of observations of the training data. Let us denote Y as the modulation from randomly drawing one outcome from $\mathscr{D}$. As described in Section 3.1, the modulations of the training data set only consists of 6 different modulations corresponding to OOK, BPSK, ..., BFSKR2, where we denote as 1, 2, ..., 6 respectively for simplicity. The probability mass function (PMF) of Y is:

$$P(Y = l) = p_l = \frac{|\{i : y_i = l\}|}{T}.$$

where the numerator resembles the number of observations corresponding to modulation l. We seek splits that maximize homogeneity as described in the example related to Figure 3.5. Therefore, we quantify homogeneity through an impurity function $i(Y)$ on the random variable Y:

$$i(Y) = -\sum_l p_l log(p_l) \tag{3.1}$$

This impurity function has two very important properties which describes homogeneity as described in Definition 2.5 of [12]. Based on the context of our problem, the first property ensures that $i(Y)$ is maximum only when $p_l = \frac{1}{6}$ for all l. The second property ensures that $i(Y)$ is minimum only when $p_l = 1$ and $p_{l'} = 0$ for $l' \neq l$ for all l. In other words, minimizing $i(Y)$ achieves maximal homogeneity and maximizing $i(Y)$ achieves minimal homogeneity.

Of course, the goal of CT as described by Figure 3.5 is to determine a split that maximizes homogeneity of modulation types. Let us define the feature variable X_u for $u = 1, 2, \ldots, M$ similar to Y taking on values $x_{i,u}$ with probability $\frac{1}{T}$ for all i, u. Let us define $Y|X_u < t$ as a conditional random variable describing the left split using feature u and threshold t. The PMF of $Y|X_u < t$ is:

$$P(Y = l|X_u < t) = \frac{|\{i : y_i = l \bigwedge x_{i,u} < t\}|}{|\{i : x_{i,u} < t\}|}$$

where the numerator corresponds to the number of observations with modulation l and values of feature u less than t. The denominator is the number of observations of values of feature u less than t. The impurity can be similarly computed as in Equation 3.1 $i(Y|X_u < t)$. The same computation applies for the right split so that we obtain impurity $i(Y|X_u \geq t)$ for the same split for the right.

Now that we have the impurity of the left and right splits, we have a good idea of how to achieve good splits. Clearly, it makes sense to minimize a combination of $i(Y|X_u < t)$ and $i(Y|X_u \geq t)$, because that way we maximize homogeneity in terms of modulations. We might be inclined to just use the sum of $i(Y|X_u < t)$ and $i(Y|X_u \geq t)$ as a splitting criteria to choose the optimal split. However, Figure 3.6 shows that this criteria is not good. The figure presents two trees with the same data but different splits. Similar to Figure 3.5, a split can be made by a vertical line in a node. So, if we were to minimize the sum of $i(Y|X_u < t)$ and $i(Y|X_u \geq t)$, then clearly, the right tree is better than the left tree based on the impurity scores computed in the figure beside each node. However, the left tree seems to have a better split, because there are more blue balls grouped in the left child as opposed to the right child. Another problem is that the sum of $i(Y|X_u < t)$ and $i(Y|X_u \geq t)$ which is 0.693 is greater than $i(Y) = 0.474$, and so we are inclined to not split at all. However, it is clear that at least one of these splits improves our objective of increasing homogeneity since it partitions the majority of blue balls to the left child. Therefore, we use the expectation of the impurity as a result of splitting as the criteria which takes a weighted average of $i(Y|X_u < t)$ and $i(Y|X_u \geq t)$ instead of the sum. We describe this splitting criteria next.

Let us define a random variable that captures the impurity from splitting by t at X_u using:

$$I = \begin{cases} i(Y|X_u < t), & p(X_u < t) \\ i(Y|X_u \geq t), & 1 - p(X_u < t) \end{cases} \tag{3.2}$$

where $p(X_u < t) = \frac{|\{i : x_{i,u} < t\}|}{T}$ representing the proportion of observations with values of feature u less than t. Furthermore, the expectation of I denoted as $E[I]$ is:

$$E[I] = p(X_u < t)i(Y|X_u < t) + p(X_u \geq t)i(Y|X_u \geq t) \tag{3.3}$$

Minimizing the expectation of I, i.e. $E[I]$ maximizes the homogeneity of the modulations of the groups produced by the split, because $E[I]$ is proportional to the impurity, and minimizing the impurity maximizes homogeneity as we have discussed. Furthermore, in Figure 3.6, we obtain $E[I] = 0.252$ and $E[I] = 0.35$ for the left and right trees, respectively. Therefore, this criteria correctly chooses the left tree, and $E[I] < i(Y)$ implies that splitting

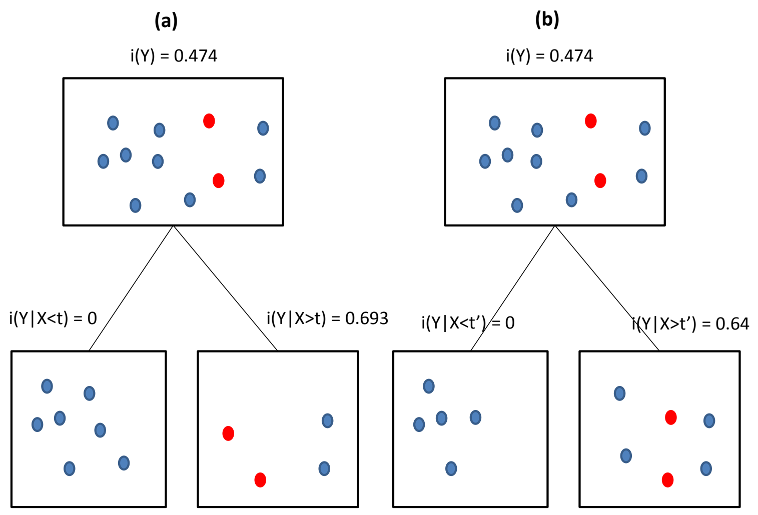

Figure 3.6: Conditional expectation of impurity example.

indeed decreases the impurity solving both the problems discussed related to Figure 3.6. Therefore, the feature u and threshold t that minimizes $E[I]$ is the feature and threshold chosen for each split of the tree.

After the first split, the same procedure of finding a split u and t is recursively applied to each of the left and right child nodes replacing $\mathscr{D}$ with the left and right child region of the split respectively. The algorithm stops splitting when the minimum number of observations at nodes which we specify at the start of the algorithm is reached or all the modulations in a node are the same. Typically, the minimum number of observations at a node is 5 to 20. A smaller number results in a larger tree and increases the complexity of the tree, whereas a larger number results in a smaller tree and decreases the complexity. We use a minimum number of observations in a node of 10 in the CTs we built. We refer to these nodes as the leaf nodes.

Predictions are made at the leaf nodes based on the majority vote of modulations in that node. We provide an example of the execution of the CT algorithm in the following. Additional details on CTs can be found in [13].

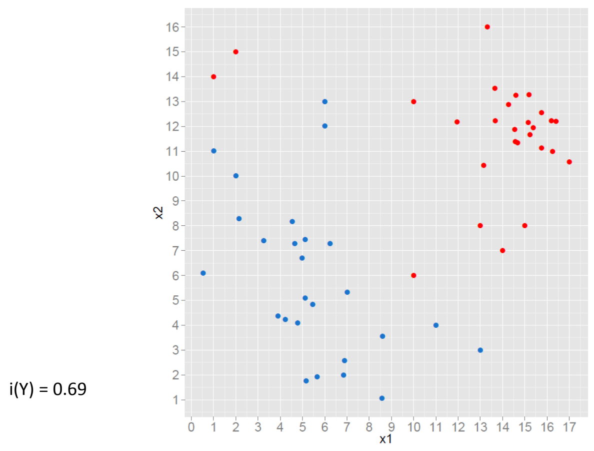

Figure 3.7: Example of data with two feature variables denoted as x1 and x2. The class variable takes on two values of either blue or red. There are 53 observations in total, where 26 are labeled as blue and 27 as red.

3.3.2 Example

A simple example to illustrate the algorithm used to construct a CT as shown in the sequence of Figures 3.7 to 3.10. Figure 3.7 depicts the example data set. There are two feature variables denoted as X1 and X2 in the figure, and one class variable with two possible values: red or blue. There are 53 observations in total, where 26 of them are labeled as blue and 27 of them red. Let us re-use notations and define the training data set for this example as $\mathscr{D} = \{(y_i, \mathbf{x_i})\}_{i=1}^{53}$ contains all the points in the figure. The random variable Y for the class variable takes on two values: blue and red. Based on $\mathscr{D}$, the PMF of Y has $p_{blue} = \frac{26}{53}$ and $p_{red} = \frac{27}{53}$ for blue and red points respectively. The impurity $i(Y)$ is computed to be 0.69.

The optimal split occurs at x_1 anywhere from 8.7 to 10, since this split results in both regions with highest proportion of each color (blue on the left region and red on the right). Therefore, we arbitrarily choose $x_1 = 9.5$. We compute $i(Y|X_1 < 9.5) = 0.271$ and $i(Y|X_1 >= 9.5) = 0.264$. We also compute $E[I] = 0.268$ for this split. We next determine the split on the left and right regions as we recursively apply the same method of finding the splits on the two regions.

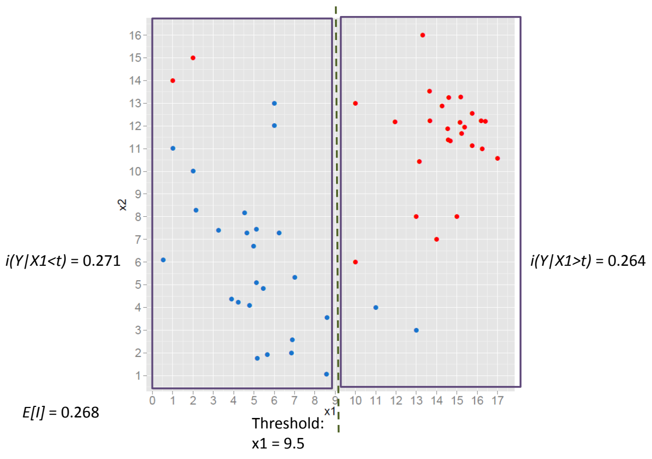

Figure 3.8: The optimal split is depicted by the dashed dark green line obtained by choosing variable x1 = 9.5. The entropy of the left and right region indicated by the purple rectangle windows are computed to be 0.271 and 0.264 respectively.

Consider the left region of the split as shown by Figure 3.9. For the sake of brevity, let us re-use notations to denote the current left region of data as $\mathscr{D} \leftarrow \mathscr{D}_{lower}$ and treat this region as the current training data. Likewise we define random variables Y, $X_u < t$ for $u \in \{1, 2\}$ and some threshold t for this new data set $\mathscr{D}$ as before.

It is clear that a value of x_2 between 13 and 14 would be optimal since the two regions would only contain either blue or red points. The impurity of the region was computed previously to be 0.271. However, we need to determine the PMF of $Y|X_2 < 13.5$ and $Y|X_2 \geq 13.5$. The impurity $i(Y|X_2 < t) = 0$ is computed for the lower region. Likewise, the impurity $i(Y|X_2 > t) = 0$ for the upper region is computed. Thus, $E[I] = 0$. Since the proportions of blue/red points are 1 in each region, the splitting stops as per algorithm instructions.

The same process occurs on the right side as shown in Figure 3.10. We found the split $x_2 = 5$ as the split and designate it as the right child split of $x_1 = 9.5$. Similar to the left side, the algorithm stops. Ultimately, the tree of this simple example is displayed in Figure 3.11.

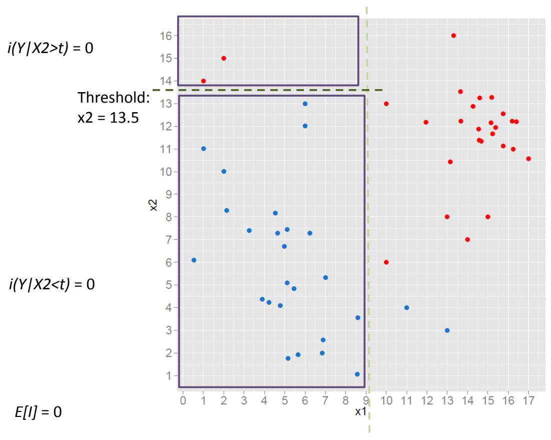

Figure 3.9: The optimal split is depicted by the dashed dark green line obtained by choosing the variable x2 = 13.5. The entropy of the bottom and upper region indicated by the purple rectangle windows are computed to 0 for both regions.

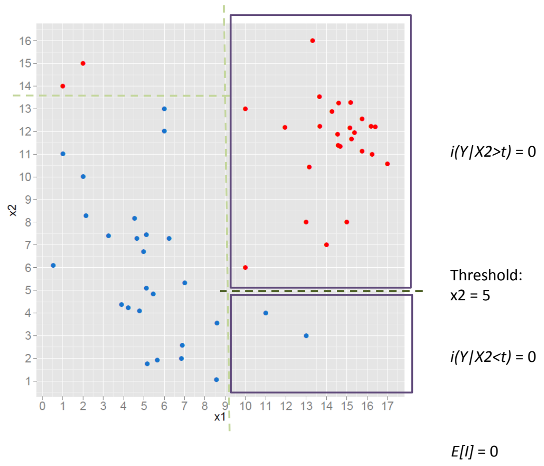

Figure 3.10: The optimal split is depicted by the dashed dark green line obtained by choosing the variable x2 = 5. The entropy of the bottom and upper region indicated by the purple rectangle windows are computed to 0 for both regions.

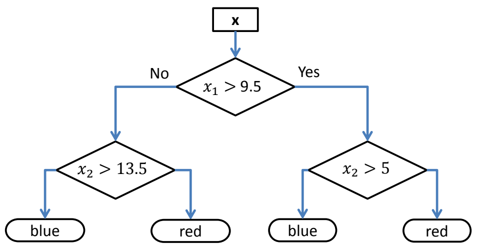

Figure 3.11: The tree structure of the simple classification tree algorithm shown in the sequence of figures 3.7 to 3.10.

The CT classifier algorithm performs a more extensive search of the features and thresholds in constructing the tree than FBT. In particular, the CT allows the features to be selected any number of times, and it considered all variables and thresholds at each split. It also used a loss function, for which contributed towards constructing the tree. Classifications are made by traversing down the tree until a leaf node is reached. A new observation x_i (using the same notation as we defined $\mathscr{D}$) is classified to one of the six possible modulations using the tree we constructed in Figure 3.12 for the simple CT example we described.

3.3.3 Pruning

The complete tree is typically quite large which means the classifier is very adaptive towards the fitted data and has a high complexity. For complexity restriction of a model, a regular behavior in small neighborhood of the input space such that for all input points x sufficiently close to each other in some metric, $\hat{C}(\mathbf{x})$ groups together the leaf nodes of the tree [13]. The complexity of a model generally induces a bias-variance trade-off of estimating the true observation $C(\mathbf{x})$ by $\hat{C}(\mathbf{x})$ such that a high complexity tends to increase the variance (see Figure 2.11 of [13]). The reason for pruning is to reduce the complexity of the tree. We describe the procedure next.

Pruning is performed by selecting the subtree with minimal splits while minimizing the entropy of each split. The strategy is to grow the tree entirely as we described in Section 3.3, and then prune the tree using a cost-complexity pruning parameter. The implementation involves computing subsets of the tree and taking the subset of tree with the lowest misclassification error and penalizing larger trees through a 10-fold cross-validation. We are not going to go over cross-validation as there are many resources describing the method such as in [14]. The pruned version of the simple classification tree example might look like the tree of Figure 3.12. For more information about pruning classification trees, we refer the readers to [13].

We apply the CT using features from Section 2.3.2 without and with the additional features introduced in Section 2.3.3. Figure 3.13 illustrates an example of a CT based on the features from [2]. Evidently, the CT is considerably larger than the FBT, and features can appear more than once for the CT. Training was carried out through the rpart package in R [15].

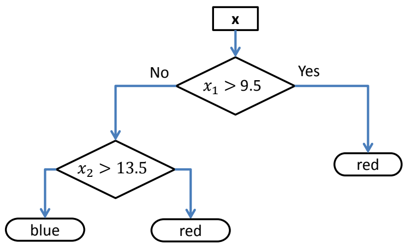

Figure 3.12: Pruned example of the simple classification tree example.

3.4 Random Forest

Random forest (RF) utilizes a method known as bagging to average over numerous classification trees obtained by bootstrapping the original data [13]. By doing so, the RF leverages from the predictions acquired from all CTs so that the consensus achieves a more accurate prediction. We describe the implementation in the next paragraph. In addition, we show later that the RF improves the success rate over the CT in Figures 4.1 and 4.4 in Section 4.

The CTs are trained on bootstraps of the original training data set ($\mathscr{D}$) where the bootstrap sample is the same size as $\mathscr{D}$. The implementation of RF begins by first drawing a bootstrap sample from $\mathscr{D}$. Let us call the bootstrap sample, $\mathscr{D}*$. A CT is trained using $\mathscr{D}*$. Another difference is that at each split of the CT, the classifier only uses a subset of the original M features for that particular split. In other words, the node defined by that split can only be one of the features in the subset. In addition, no pruning is performed on the trees. We repeat the process n_{tree} times where n_{tree} is a parameter, and we explain this later in this section. At the end, we have n_{tree} number of CTs which together forms a RF.

Classification of a new observation x_i is performed similarly to the CT, except we have n_{tree} CTs. Therefore, we have n_{tree} classifications. Let us refer to the b^{th} classification made by the b^{th} tree to be $\hat{C}_b(x)$. In statistics, the mode is the value that appears most often in a set of data. The classification of the RF is $\hat{C}_{rf}(\mathbf{x}) = mode\{\hat{C}_b(\mathbf{x})\}_1^{n_{tree}}$ which is the majority vote prediction of the n_{tree} trees.

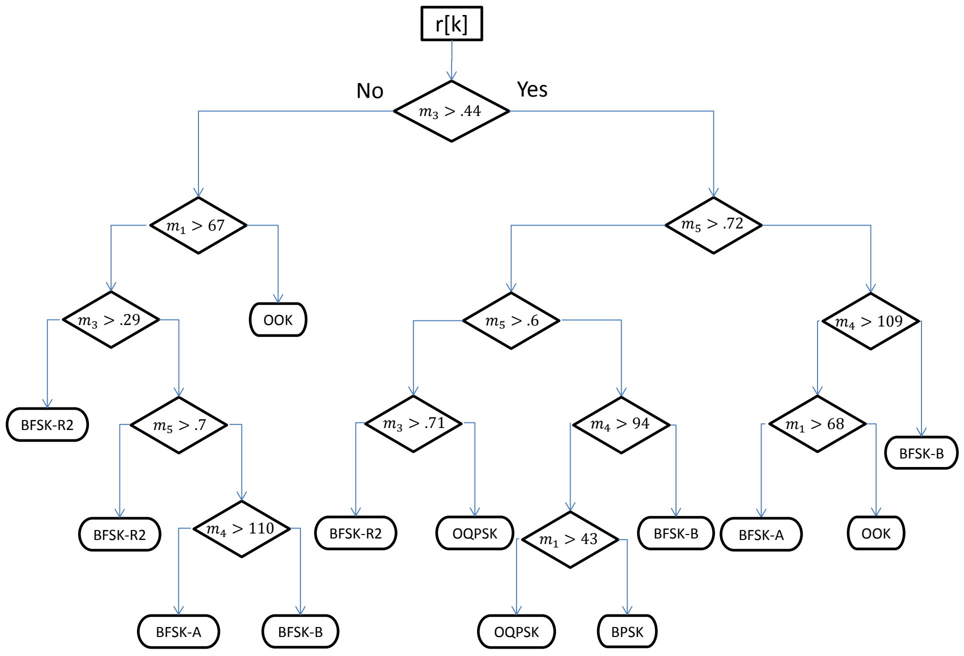

Figure 3.13: Flowchart of classification tree depicting features and decision thresholds at interior nodes. Modulation types reside at leaf nodes.

3.4.1 Advantages of Random Forest

We have mentioned earlier that bagging involves the average over the CTs, but $(\hat{C}_{rf})$ above is based on the majority vote of the prediction estimators of the CTs $(\hat{C}_b)$ instead of the average. Intuitively, the idea of utilizing a collection of trees to improve the predictive accuracies are similar in both the regression and classification case. The analysis underlying the majority vote and the discrete prediction case is more complex and we revert the reader to [13] for an in-depth discussion of the topic, and we analyze the continuous case based on regression instead in the following.

Using the same notation for classification, the estimator of the classification is $\hat{C}_b(\mathbf{x})$ which is random in terms of bootstrapping the training data. The main advantages of the RF comes from reducing the variance and instability from fitting a single CT.

The $\hat{C}_b(\mathbf{x})$'s from constructing each CT of a RF are not independent, because we are using the same algorithm in constructing each CT in the RF which also means that the pairwise correlations between the CTs are positive. Say, the variance of $\hat{C}_b(\mathbf{x})$ is σ^2, in other words,

$$Var(\hat{C}_b(\mathbf{x})) = \sigma^2.$$

Clearly, if the trees are independent, then $Var(\hat{C}_{rf}(\mathbf{x})) = \frac{1}{n_{tree}}\sigma^2$. However, the trees are positively correlated as mentioned. Let us use ρ to refer to the pairwise correlation between two trees. In other words, $Cor(\hat{C}_i(\mathbf{X}), \hat{C}_j(\mathbf{X})) = \rho$ for some $i = 1, 2, \ldots, n_{tree}$ and $j \neq i$. The variance of the prediction estimator of RF is:

$$\begin{aligned} Var(\hat{C}_{rf}(\mathbf{x})) &= \frac{1}{n_{tree}^2}(n_{tree}\sigma^2 + n(n-1)\sigma^2\rho) \\ &= \frac{\sigma^2}{n_{tree}} + \frac{(n_{tree}-1)}{n_{tree}}\sigma^2\rho \\ &= \sigma^2\rho + \frac{\sigma^2}{n_{tree}} - \frac{\sigma^2\rho}{n_{tree}} \\ &= \sigma^2\rho + \frac{1-\rho}{n_{tree}}\sigma^2 \end{aligned} \tag{3.4}$$

As n_{tree} increases the second term becomes small, so the variance is roughly $\sigma^2\rho$. This is an improvement since $-1 \leq \rho \leq 1$. Moreover, the less correlated the trees are the lower the variance, so at the end we are limited by the correlation of the trees which ultimately limit the advantages of bagging. This issue brings us back to why the RF algorithm involves randomly selecting $m < M$ features at each split rather than all M features at each split. Intuitively, selecting only subsets of the features allow the trees to be different which leads the trees to be less correlated. If we use all features at each split, then the trees would be very similar, hence, the prediction estimators would be very correlated.

3.4.2 Random Forest Implementation

Based on [13], fully growing the trees seldom cost much, and as a result, we have one less parameter to optimize. Therefore, we allow the trees to be grown to maximal depth. Model fitting was carried out using the randomForest package in R [16].

The parameters that require tuning in RF are the number of trees, *ntree*, and the number of randomly selected features, *mtry*. The number of trees should be large enough so that predictions are made multiple times for each observation [17]. Recall that each bootstrap sample retains roughly 63% of the original data, since the procedure involves randomly sampling from the same data set with replacement. In the worst-case scenario of not having enough trees is that not all the data have been used, which is bad since we always want to consider using all the training data.

A suitable value is around 500-1000 according to [13]. A large value of *ntree* often does not overfit or hinder the predictive performance of the model. However, as we have mentioned, *ntree* needs to be large enough so that predictions are made multiple times for each observation. In our experiments, we used $ntree = 850$ trees. The inventors of RF suggest $mtry$ to be approximately equal to the square-root of the number of features [13]. Hence, if we consider only $m_1, \ldots, m_5$, then $mtry = 2$, while the use of all the features from Table 2.1 leads to $mtry = 12$.

3.4.3 Random Forest Feature Selection

A feature selection step is initially carried out on all the features by training an RF, and then selecting the features with the highest importance. The importance of a feature is computed by the sum of the information gain at decision thresholds subject to all the trees [13].

Information gain is defined as:

$$info = i(Y) - E[I] \tag{3.5}$$

where $i(Y)$ and $E[I]$ are described in Equations 3.1 and 3.3. The importance measure for a single classification tree is expressed [13] as

$$Imp_u(T) = \sum_{t=1}^{J-1} info_t \mathbb{1}(v(t) = l) \tag{3.6}$$

where $info_t$ represents information gain at a particular node t, and u represents the feature chosen at that split. Furthermore, $J - 1$ is the number of interior nodes in the tree. The variable importance for a random forest model takes the average of importance for the N classification trees.

$$Imp_u = \frac{1}{N} \sum_{n=1}^{N} Imp_u(T_n) \tag{3.7}$$

The importance of each feature is ranked, and the optimal features are selected. This selection step filters out irrelevant features that could lead to overfitting.

We can select the top 40-70% of features based on the importance. We used a 10-fold cross validation to verify the fraction of features that seem to give the lowest misclassification error. We tried 40%, 50%, 60%, and 70%, and we found that 40% provides the optimal performance in terms of misclassification error. For feature selection, it is sufficient to use 300-500 trees, because the most relevant features often appear numerous times across all the trees. It also speeds up the selection step. We used 400 trees for feature selection.

3.5 Modifications for Noise Detection

In addition to the six modulation types, the classifiers are also trained to detect noise from signal. It is important to detect noise so that signals identified as noise are not processed at the receiving device. Processing a signal that is noise is a waste of resource for the receiver. Referring to Equation 2.5 of Section 2.1, the additional noise signal is computed using $r[k] = n[k]$. The training data described in Section 3.1 should also include the noise signal data under the column header 'Modulation'. In other words, the noise signal is included as a seventh class variable for the classifiers.

To predict the additional class, we include a feature representing the energy of the signal denoted by *en* to the training data set. The noise variance, σ_n^2 is defined earlier in Section 2.1.

$$\begin{aligned} e &= \frac{1}{N}\sum_{i=1}^{N} |x[i]|^2 \\ en &= \frac{e}{\sigma_n^2} \end{aligned} \tag{3.8}$$

The training data set is similar to the data set shown in Table 3.1, except there is one more class class variable and one more feature. The new training data set is shown in Table 3.3.

For FBT, we use *en* to add a new split in the tree. From Figure 3.14, we can see that values of *en* for the noise signal is consistently lower than the six modulations. This indicates that *en* is a useful feature at recognizing noise from the modulations. We performed a worstcase analysis based on *en*. For example, if *en* is less than some threshold, then we classify the signal as noise. If the signal is greater than some threshold, then we classify the signal as one of the six modulations. After this initial classification, the tree is constructed the same as before as in Figure 3.3.

Table 3.3: Example of training data with noise class. The modulation corresponds to the class variable, and the columns to the right of modulation from m_1 to en correspond to the feature variables.

Modulation	$\boldsymbol{m_1}$	$\boldsymbol{m_2}$	$\boldsymbol{m_3}$	**...**	$R[k]$	en
OOK	190	0.05	0.3	...	2.5	34.9
OOK	186	0.12	0.21	...	1.8	32.3
OOK	172	0.08	0.25	...	3.1	29.6
..	..	..	..	..	..	..
BFSK-R2	15.3	0.64	0.03	...	5.4	13.5
BFSK-R2	14.4	0.54	0.08	...	3.3	15.2
BFSK-R2	89.7	0.35	0.12	...	2.48	16.1
..	..	..	..	..	..	..
Noise	56.2	1.3	0.8	...	3.5	34.5
Noise	40.1	1.2	1.1	...	3.2	32.7
Noise	38.5	0.8	0.9	...	1.8	33.3

From the worstcase analysis plot of Figure 3.15, we observe that at SNR of approximately -8 dB, the maximum value of en intersects with the minimum value of OOK. The threshold value of en is calculated to be 1.0257. Figure 3.16 shows the structure of the tree when the noise signal is included.

For the CT and RF classifiers, we simply input the new training data set as in Table 3.3 into the same algorithms we described in Sections 3.3 and 3.4.

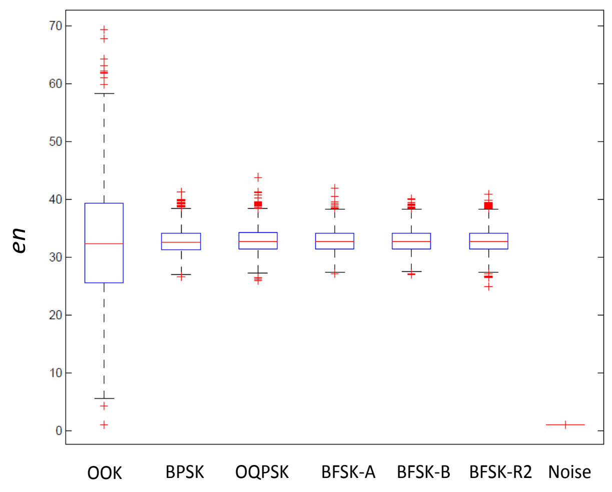

Figure 3.14: Boxplot visualization of *en* for each modulation.

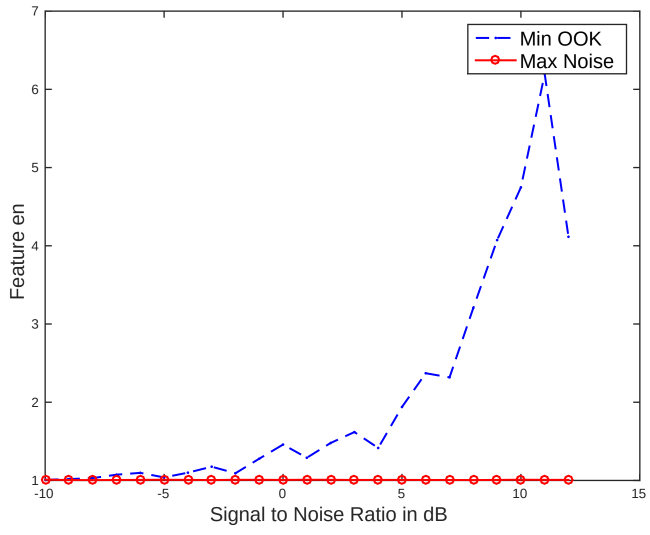

Figure 3.15: Worstcase plot of *en* where minimum value of OOK is compared to maximum value of the noise signal for each SNR.

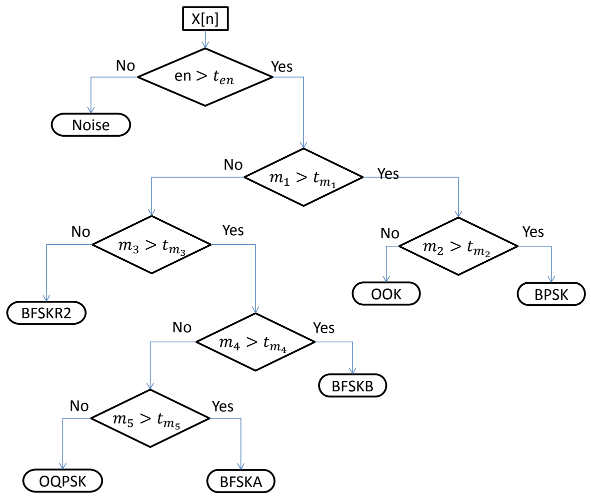

Figure 3.16: Tree structure of FBT with noise signal included.

Chapter 4

Analysis and Results

In this section, we compare the predictive performance of the FBT, CT, and RF classifiers using different sets of features. We also show the difference in performance when including the noise signal as an additional class. To this end, we measure the success rate (SR) of each classifier by applying it on a new data set that was not used for training, and then recording the proportion of correct predictions obtained by the different classifiers.

We generate a training data set based on $P = 200$ values for each feature per modulation between SNR from -32 dB to 16 dB in steps of 2 dB. The testing data set is generated similarly to training, except $P = 1100$ is used.

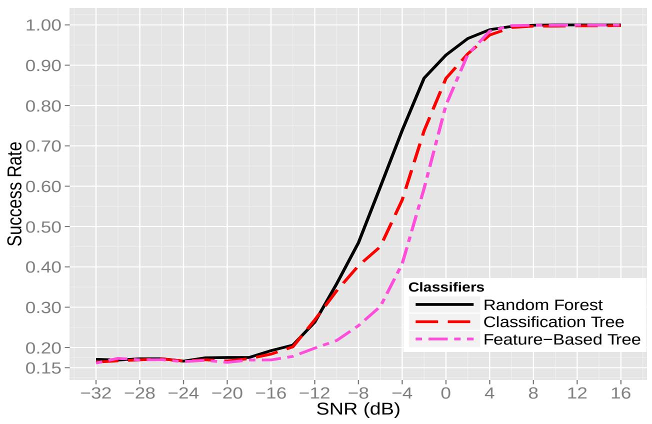

Figure 4.1: Success rates of models based on features from [2] as a function of SNR generated in ggplot2 [18].

4.1 Analysis on Features From [2]

In the first experiment, we only use the features from [2] and six modulation types. Figure 4.1 shows the SR of each classifier. We observe notable gains for both RF and CT over FBT for SNRs ranging from about -15 dB to 1 dB. RF shows the overall best performance with improvements in SR over FBT of, for example, 32% and 13% (where all improvements are expressed in absolute value) at -5 dB and 0 dB, respectively. Likewise, CT obtained improvements of 20% and 8% at -5 dB and 0 dB, respectively. All three classifiers obtain a SR of approximately one for SNR > 3 dB.

In the second experiment, we use the same features as the first experiment, but the noise signal is included as a seventh class for prediction. As shown in Figure 4.2, the results are similar to the first experiment involving six classes. The RF classifier outperforms the other two classifiers from SNR of -15 dB to 3 dB, and CT outperforms the FBT for the same range of SNR.

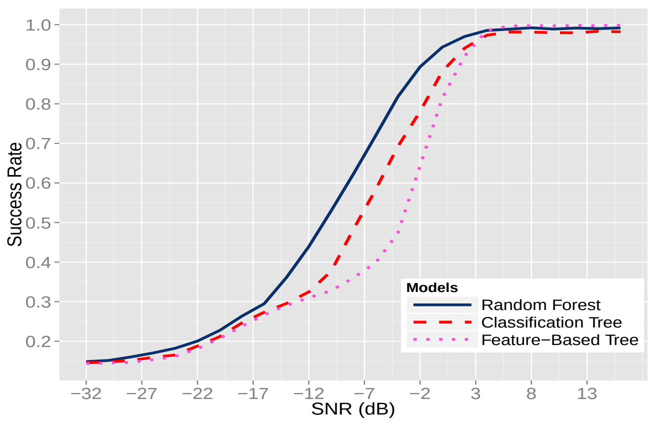

Figure 4.2: SRs based on the features from [2] and prediction for seven classes.

Although the structure of the CT is quite different from the FBT as shown in Figures 3.2 and 3.13, if we fit the CT with only high SNR data of > 1 dB, then the structure is almost equivalent to the FBT. The CT fit with SNR > 1 dB and FBT are shown side-by-side in Figure 4.3. The structure of the tree at depth two is identical with only minor differences. The nodes highlighted in orange emphasize that m_5 is chosen before m_4 for the CT. Based on the left subtree after splitting by m_3, small values of m_5 predict OQPSK for both trees. Similarly, small values of m_4 predict BFSK-A and large values of m_5 predict BFSK-B for both trees. From this analysis, it appears that the FBT classifier is similar to the CT if the training data set includes only high SNR signal data. If we recall from Section 3.2

and Figure 3.4, as SNR increases, it becomes more difficult to separate the modulations as shown by how the two lines in the worst-case plot intersect. As a result, using only high SNR data, it is much quicker to achieve maximal homogeneity at the leafs of the tree leading to a smaller tree.

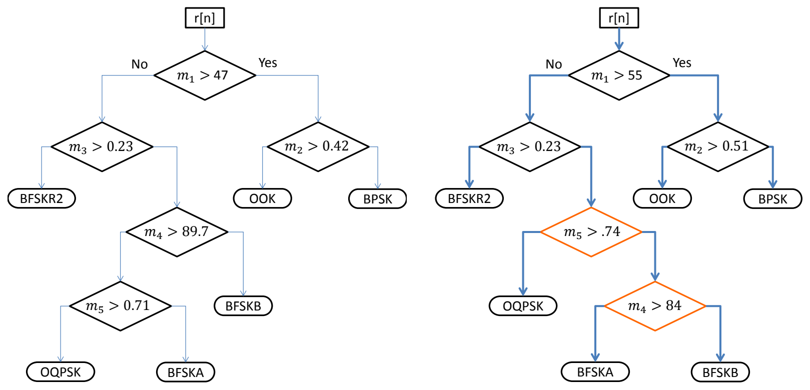

Figure 4.3: The left flow chart represents the FBT with computed threshold values. The right chart corresponds to the CT fit at a high SNR. The main difference is m_5 is used before m_4 with the distinction highlighted in orange.

4.2 Analysis on All Features

In the third experiment, we apply RF and CT based on all the features described in Section 2.3 and summarized in Table 2.1 for six modulations. The corresponding SRs are shown in Figure 4.4. RF obtained a SR of approximately one for SNR > -2 dB. Comparing the SR curves for RF and CT from Figures 4.1 and 4.4, we note the significant additional gain due to the inclusion of the proposed features. The RF and CT classifiers outperform the FBT classifier by a large margin based on a wide SNR range from about -25 dB to 3 dB. Furthermore, the RF classifier achieves best SRs with the gap to CT widened compared to Figure 4.1.

Similar to the third experiment, the fourth experiment includes the noise signal as the seventh class to the data, and we found the results to be very similar. The predictive performance for each classifier is shown in Figure 4.5.

The variable importance plot based on computing the importance of each feature as in Sections 3.4 and [13] is presented in Figure 4.6. Vari-

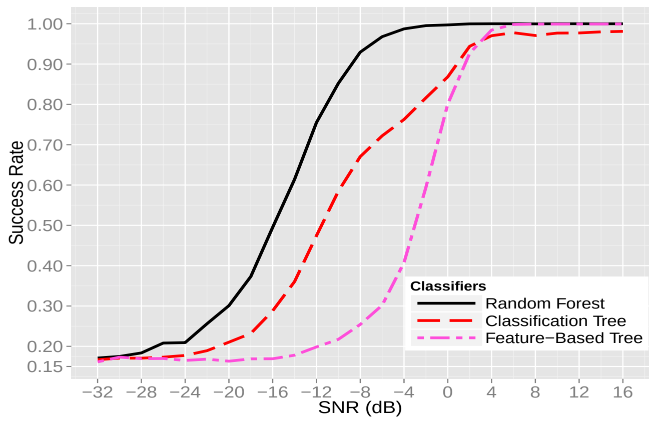

Figure 4.4: Success rates of each model proportional to SNR using all features.

able importance is described in Section 3.4.3. The features are ordered by variable importance. From the figure, features with labels beginning with Rk correspond to the magnitude spectrum $|R[k]|$. In fact, $|R[k]|$ at $k = 83, 84, 85 \text{and}, 136$ are within the top 7 features selected which may be because BFSK-B is modulated at a higher intermediate frequency than the remaining modulations or the fact that only subsets of features are used at each split causes one of these features to be chosen if one is not present. The magnitude spectrum of each modulation type is displayed in Figure 2.5. The figure shows several peaks occurring at discrete-time index k of 83, 84, 85, and 136. Finally, from the variable importance figure, we notice that transformation based features γ_{max} and γ_2 also have high importance. Similarly, all features from [2] have high importance.

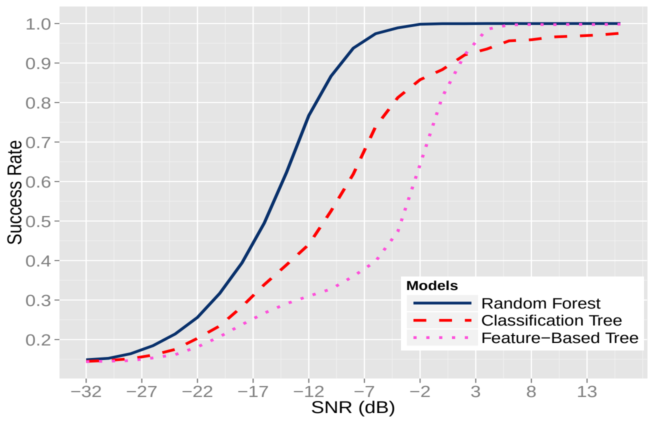

Figure 4.5: SRs based on the all features and prediction for seven classes.

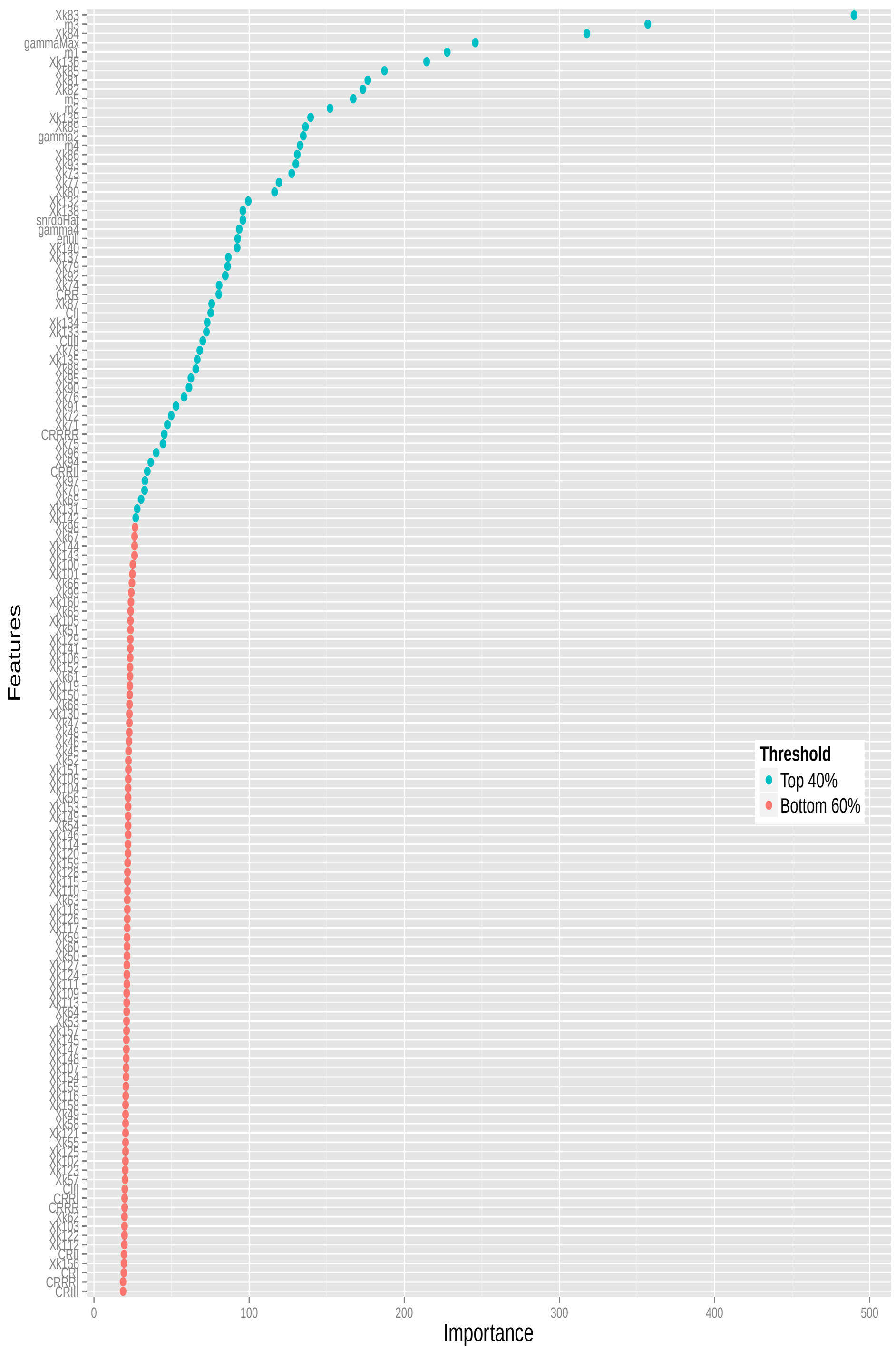

Figure 4.6: Variable importance based on the RF using all the features.

4.3 Further Discussion

Note that the CT classifier does not reach an SR of one even for very high SNRs. One possible explanation is that its thresholds have been influenced by highly noisy training data with low SNR. This observation is supported by the notable improvements that CT achieves over FBT at low SNRs ($\lesssim -2$ dB). Finally, note that, as expected, all classifiers perform almost perfectly well for large SNR levels (SNR $\gtrsim 3$ dB). Hence, the advantage of our proposal is most evident for the more challenging situations of moderate to low SNRs.

The price to be paid for this performance gain is an increase in computation time and storage requirement. The computational complexity of CT and RF classifiers largely depends on the structure of the data, much like many sorting algorithms depend on the initial distribution of the data. That being said, it is still worthwhile to describe the complexity in general. The computational complexity of training a basic implementation of a CT classifier for T training samples and M features is $O(MT\log(T))$ as described on page 199 of [19]. In fact the most basic brute-force approach is $O(MT^2\log(T))$, but is easier to explain first, and then we will explain how the implementation can be improved to be $O(MT\log(T))$.

For each feature variable m, we must compute the entropy using each observation which in total has T observations. Since the observations are not ordered, we must check each remaining $T-1$ observations. Therefore, it takes $T(T-1)$ operations to compute the minimum entropy possible for feature m. Therefore, one split takes roughly MT^2 operations. If we assume that the tree is bushy (in other words it doesn't degenerate to a few long branches) as in the average case of CT, then the depth of the tree is roughly $log(T)$. Thus, in total the number of operations to construct the CT is $MT^2log(T)$ in this brute-force approach. However, if we sort each feature by its observations prior to constructing the tree we can achieve $O(MTlog(T))$.

The average sorting of the observations of one feature is $O(Tlog(T))$ (as is obvious using a standard approach such as quicksort or mergesort). In determining the optimal split, for each feature m, we only need to scan through the observations once in order to find the minimum entropy since the observations are already sorted and we do not need to compare with the remaining operations. So, each split takes roughly MT operations, and assuming a bushy tree the process of constructing the tree is $O(MTlog(T))$ on average.

The computational cost of training a RF classifier is intuitively $O(n_{tree}MT\log(T))$

since the algorithm involves constructing n_{tree} CTs. Training a FBT classifier only costs $O(T\log(T))$, because the features are selected prior to threshold determination (and sorting a single feature variable takes $O(T\log(T))$ on average). Therefore, the CT and RF classifiers require $O(M)$ and $O(n_{tree}M)$ times more operations to train than a FBT, respectively.

The computational cost of a single prediction for the FBT is approximately $O(\log(M))$ evaluations (where $\log(M)$ corresponds to a binary search). The CT classifier with pruning requires $O(\log(M))$ and without pruning requires $O(\log(N)\log(M))$ operations. Therefore, a prediction for RF requires $O(n_{tree}\log(N)\log(M))$ operations.

Since the number of thresholds is fixed for the FBT, the storage cost is $O(1)$. Likewise, the CT with pruning often results in a near constant depth tree regardless of the size of the data set. So, the storage cost of CT with pruning is $O(1)$, and therefore the additional storage requirement of CT is relatively modest. However, the RF requires $O(n_{tree}\log(N))$ in storage requirement as the trees are not pruned. In order to facilitate the implementation of RF, considerable memory allocation may be required.

Chapter 5

Conclusion

In this paper, the classification tree and random forest classifiers are introduced for modulation recognition in the 868 MHz band. The classification tree allows features to appear more than once in the tree leading to an improvement in prediction performance over the feature-based binary tree. The random forest classifier leverages the predictions from a set of classification trees which further improved performance over a single classification tree. Moreover, an additional set of features commonly used in modulation recognition is experimented on and found to improve upon the proposed methods. Based on our findings, both classification tree and random forest attained higher success rates in modulation prediction relative to the feature-based binary tree method when the signal is corrupted with white noise.

Future studies include the addition of more training data for the classifiers, along with an analysis on success rate as a function of the number of observations trained on. This analysis could lead to the implementation of more scalable methods. For example, we plan to implement a map-reduce version of random forest. It would allow data from a distributed file system to be streamed into memory where individual classification trees are fitted and stored in temporary disk until the algorithm is complete.

Bibliography

[1] J. F. Kurose, *Computer networking: a top-down approach featuring the Internet.* Pearson Education India, 2005.

[2] M. Kuba, K. Ronge, and R. Weigel, "Development and implementation of a feature-based automatic classification algorithm for communication standards in the 868 mhz band," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, pp. 3104–3109, IEEE, 2012.

[3] O. A. Dobre, A. Abdi, Y. Bar-Ness, and W. Su, "Survey of automatic modulation classification techniques: classical approaches and new trends," *Communications, IET*, vol. 1, no. 2, pp. 137–156, 2007.

[4] A. Hazza, M. Shoaib, S. Alshebeili, and A. Fahad, "An overview of feature-based methods for digital modulation classification," in *Communications, Signal Processing, and their Applications (ICCSPA), 2013 1st International Conference on*, pp. 1–6, IEEE, 2013.

[5] A. K. Nandi and E. E. Azzouz, "Algorithms for automatic modulation recognition of communication signals," *Communications, IEEE Transactions on*, vol. 46, no. 4, pp. 431–436, 1998.

[6] M. H. Valipour, M. M. Homayounpour, and M. A. Mehralian, "Automatic digital modulation recognition in presence of noise using svm and pso," in *Telecommunications (IST), 2012 Sixth International Symposium on*, pp. 378–382, IEEE, 2012.

[7] I. The Mathworks, *MATLAB and Communications System Toolbox Release 2014b.* Natick, Massachusetts, United States, 2014.

[8] M. U. Guide, "The mathworks," *Inc., Natick, MA*, vol. 5, p. 333, 1998.

[9] M. D. Wong and A. K. Nandi, "Automatic digital modulation recognition using artificial neural network and genetic algorithm," *Signal Processing*, vol. 84, no. 2, pp. 351–365, 2004.

[10] Z. Wu, X. Wang, Z. Gao, and G. Ren, “Automatic digital modulation recognition based on support vector machines,” in *Neural Networks and Brain, 2005. ICNN&B’05. International Conference on*, vol. 2, pp. 1025–1028, IEEE, 2005.

[11] F. Xie, C. Li, and G. Wan, “An efficient and simple method of mpsk modulation classification,” in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–3, 2008.

[12] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[13] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning*, vol. 2. Springer, 2009.

[14] R. Kohavi *et al.*, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Ijcai*, vol. 14, pp. 1137–1145, 1995.

[15] T. M. Therneau, B. Atkinson, and B. Ripley, “rpart: Recursive partitioning,” *R package version*, vol. 3, no. 3.8, 2010.

[16] A. Liaw and M. Wiener, “Classification and regression by randomforest,” *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[17] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] H. Wickham, *ggplot2: elegant graphics for data analysis.* Springer New York, 2009.

[19] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.