AN ABSTRACT OF THE THESIS OF

Ryan Frier for the Master of Science in Mathematics presented on April 5, 2017

Title: Wavelet-Based Acoustic Classification of Bird Species

Thesis Chair: Qiang Shi

Abstract approved:____

ABSTRACT

Identifying birds based off their calls is extremely useful in the realm of avian biology, especially ecology. In this paper we consider the calls of the Whip-poor-will (*Antrostomus vociferus*), the Northern Bobwhite (*Colinus virginianus*), the Barred Owl (*Strix varia*), the Eastern Kingbird (*Tyrannus tyrannus*), and the Common Raven (*Corvus corax*), and ways of using automated classifiers to identify the bird species based off these calls. In this study, we segment the bird calls into syllables. Then we apply wavelet decomposition to decompose the recordings of the bird calls and extract certain parameters from the syllables. All of the instances and the parameters were placed in an Excel file and uploaded into WEKA, a software for classification. We used various classifiers to classify the different syllables, but Random Tree and Random Forest were the most successful in our study. Both of the classifiers achieved over 70% accuracy when classifying species on the data set that contained the various species of birds. This thesis shows that birds can be classified into their species based off recordings of their calls with relative accuracy.

Keywords: Bird calls, wavelet transform, Random Forest, Random Tree, classifier.

Wavelet-Based Acoustic Classification of Bird Species

A Thesis

Presented to

The Department of Mathematics and Economics

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mathematics

By

Ryan Frier

April 2017

Approved by the Department Chair

Committee Member

Committee Member

Committee Member

Committee Member

Approved by the Dean of the Graduate School and Distance Education

Acknowledgements

I would like to thank Dr. Qiang Shi for his guidance in writing this paper, as well as his time and effort. His help will forever be appreciated. I want to thank the Macaulay Library at Cornell University for providing me with the recordings which I used in my study. I also thank Emporia State University for its support during this endeavor. I would also like to thank my graduate committee, Dr. Hollenbeck, Dr. Jensen, Dr. Shi, and Dr. Scott, for devoting time and effort into this project.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In the world of biology, knowing which species are present in a given ecosystem can be extremely beneficial, and in some cases crucial. Knowing the bird species present in a given habitat can tell us many important things about the area. First and foremost, knowing the bird species helps us track the extinction of birds. According to the National Audubon Society [18], roughly 27,000 plant and animal species go extinct each year. For many of these species, humans are the main factor that drive them to extinction. Knowing how many species are present in an ecosystem also tells us about the function of the system. Birds are also “winged sentinels” [18, para 34] of a sort. That is, birds are bioindicators in the ecosystem. This study will attempt to identify bird species based off their calls. This study will attempt to identify individual birds based off their calls, also.

However, current methods of determining which birds are present have their limitations. Current methods of surveying what birds are present involves biologists identifying the size, shape, posture, the field marks, the habitat, and flight patterns [4]. The reason these may not be especially precise is because multiple birds may have similar field marks. Also, many birds may live in the same ecosystem.

There have been different mathematical methods used to classify bird sounds. Selin, Turunen, and Tanttu [16] used algorithms to classify bird species. The general approach used can be seen in Figure 1.1.1.

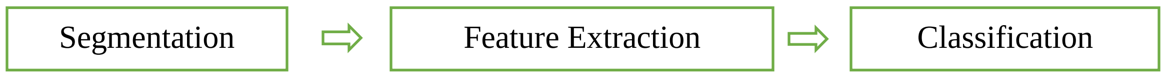


Figure 1.1.1: General method.

To begin we took the sound files and segmented them into different files and stored them. We then transformed the data and extracted various features from the different sound files. We then stored the features of each sound file in an Excel spreadsheet and uploaded the spreadsheet into classifiers. We used different classifiers to help us classify the different bird species.

Traditional analysis utilizes Fourier transforms. However, since the late 20th century a new field of mathematics, called wavelets, has been blossoming and may provide some assistance in that area. Wavelet theory was discovered predominately by Ingrid Daubechies of Duke University, Stéphane Mallat of Ecole Polytechnique, and Jean Morlet of Centre International de Rencontres Mathématiques [6].

Selin, Turunen, and Tanttu [16] use Wavelet Packet Decomposition to classify inharmonic and transient bird sounds efficiently. Learned [9] also discusses the use of Wavelet Decomposition Packets to detect and classify the sounds generated by the snapping shrimp and sperm whale clicks.

The aim of this project is to use wavelet theory in conjunction with modern computer technology to provide evidence that one may classify birds based off their calls. The birds used in the study are the Eastern Whip-poor-will (*Antrostomus vociferus*), the Northern Bobwhite (*Colinus virginianus*), the Barred Owl (*Strix varia*), the Eastern Kingbird (*Tyrannus tyrannus*), and the Common Raven (*Corvus corax*). There were five of each species used, and their calls were provided by the Macaulay Library at the Cornell Lab of Ornithology [10]. These species were chosen because they represent a wide range of the various types of vocalizations that birds can produce.

Since this work includes the analysis of digital recordings, a general introduction to the digital representation of sound will be provided in Chapter 2. Wavelet theory relies heavily on the work of Joseph Fourier known as Fourier transforms and Fourier series.

Fourier analysis and spectrum analysis will be addressed in Chapter 3. Also impulses, linear shift-invariant systems, convolution, the Sampling Theorem, aliasing, and how to use the sampled data will be covered in Chapter 3. Fourier series are composed of complex numbers. For the reader unfamiliar with complex variables, there is also a short review of complex variables at the beginning of Chapter 3.

Since this study primarily uses wavelet theory, wavelets will be covered in Chapter 4, including the definition of lowpass and highpass filters, the integral and discrete wavelet transform, orthogonal wavelet bases, and how to construct the Daubechies wavelet filters.

The classification methods will be addressed in Chapter 5. In order to fully appreciate these methods, an introduction to data mining and classifiers will be provided. The credibility of these classifiers will also be discussed.

In Chapter 6 the methods and results will be discussed. The methods involved include the preprocessing and segmentation, wavelet decomposition, texture extraction, and classification. The results of the different classifiers, as well as the conclusion and any further discussion of the topic, will be the final topic covered. The classifiers had varied results when trying to classify species. Only a portion of the possible classifiers were used in the study. The results had a minimum accuracy of 64.794% and a maximum of 74.25%. When classifying the bird species, we found that Random Tree and Random Forest using cross validation were the most successful, which were more than 70% accurate. When classifying four individuals per species, we conclude that our method can separate the four Barred Owl individuals into three groups, the Bobwhite individuals into one group, the Common Raven individuals into three groups, the Eastern

Kingbird individuals into four groups, and the Whip-poor-will individuals into three groups. That is, when given four individuals of each species, the classifiers were able to find three Barred Owls, one Bobwhite, three Common Ravens, four Eastern Kingbirds, and three Whip-poor-wills. It should be noted that because of the segmentation part of the process, there is only one bird call recognizable in each syllable. Seeing as the process separated four out of the five bird species into at least three groups, the results, over all, are considered successful. The results will be covered in full in Chapter 6.

Chapter 2 Digital Representation of Sound

Sound is created by a vibrating object that sends waves through some elastic medium. Normally this medium is either air or water. Sound is recognized because it causes a change in pressure relative to the ambient pressure. Since it creates a pressure, sound can be measured based on its pressure (in Pascals) or based on the wavelengths emitted from the vibrating object [3]. In this study, a focus will be given to the wavelengths associated with sound as opposed to the pressure.

It should be noted that sound is a continuous signal, also called an analog signal (See Figure 2.1.1). In order for a computer to process the sound, a digital representation of sound is required (See Figure 2.1.2). That is, we need discrete points along the continuous signal. "The digital representation … consists of a sequence of numeric values representing the amplitude of the original waveform at discrete, evenly spaced points in time" [3, p 319].

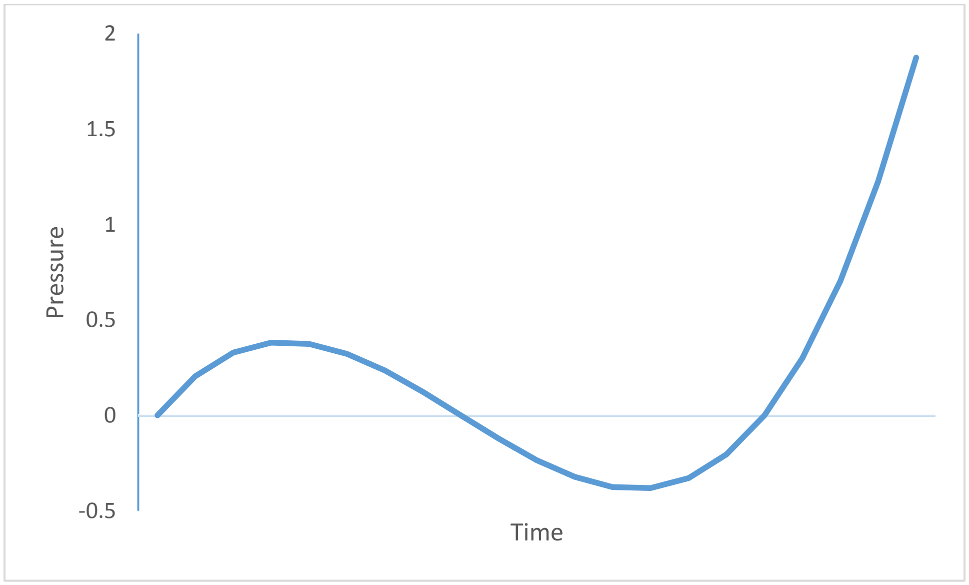


Figure 2.1.1: A continuous representation of a sound wave

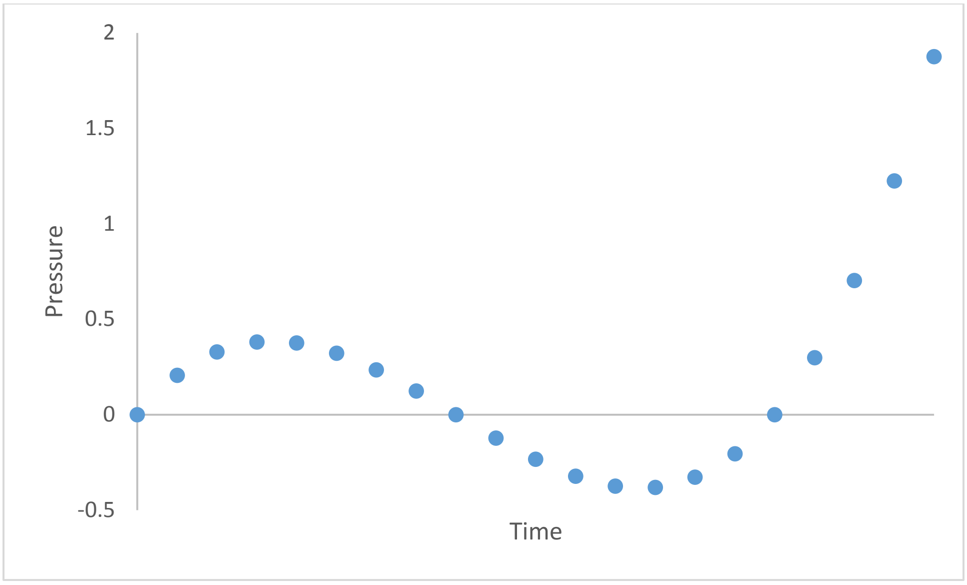


Figure 2.1.2: Discrete representation of Graph 2.1

Digital signals may be obtained by sampling continuous signals. A sample rate is how often a sample is taken from the signal. For this study, a sampling rate of 44.1 kHz is used. That means that there are 44,100 samples taken per second. The accuracy of the digital representation of the sound wave depends on the number of samples taken from the wave. In order to obtain a fairly accurate digital representation, we must have a sample rate greater than the Nyquist frequency in order to avoid aliasing, where aliasing is defined as “the appearance of phantom frequencies as an artifact of inadequate sample rate” [3, p 321]. According to the Cornell Lab of Ornithology, the Nyquist frequency is “the highest frequency that can be represented in a digitized signal without aliasing” [3, p 321]. In other words, the Nyquist frequency is the threshold or critical limit that a frequency can be and still be represented as a digital signal without aliasing. For instance, if a wave had a frequency of 10 Hz, then the sampling rate must be greater than 20 discrete samples per second. If, in this example, the sampling rate was, say, 5 discrete samples per second, then there would be aliasing, or “frequencies represented in it that were not actually present in the original at all" [3, p 321]. A mathematical explanation of sampling and aliasing can be found in Section 3.8.

In order to acquire a digital signal, the amplitude values also need to be quantized into discrete values. The accuracy is dependent on the number of bits used in the binary representation of the data or its bit depth, or "the sample size or number of bits" [3, p 323]. Similar to frequency, more bits in representation mean more discrete levels, which gives better accuracy. If too few bits are used, noise, known as quantization noise, can occur [3, p 323]. Quantization noise is typically manifested in a low hiss. By using an appropriate bit depth, we may avoid some if not all audible quantization noise. In our study, each sample has a 32-bit depth.

Having a digital representation of sound allows us to apply different discrete transformations (such as discrete Fourier transform or a wavelet transform) to the sound and extract features for classification. We will use these features to classify birds based off these calls. We will introduce Fourier transforms and wave transforms in the next chapter.

Chapter 3 Fourier Analysis and Spectrum Analysis

Fourier and spectrum analysis are done in the complex plane. Thus, we first briefly cover the complex numbers and the complex plane. For a complete discussion on the complex numbers, please refer to [2]. We will then extend the idea of complex variables and discuss a certain type of series in the complex plane known as the Fourier series. Afterwards we will discuss impulse, Fourier transform, convolution, the Fourier transform of sampled functions, the sampling theorem, and how we can reconstruct the original signal from the sampled data. Gonzales and Wood provide more details on these ideas in their book [7]. At the end of this chapter a study of the windowed short-time Fourier transform is covered.

Fourier analysis and spectrum analysis are crucial in the study of wavelets. Wavelet filters can be characterized in the Fourier domain. In the next chapter, we will use the ideas of Chapter 3 to help build the theory of wavelets.

Section 3.1 The Complex Numbers and Complex Plane

In order to understand how wavelet theory and sound analysis work, we must have a decent understanding of how complex variables work.

The complex numbers are defined as a set of coordinates in the complex plane. We say that a number is a complex number if it can be written as

$$z = x + iy,$$

where x and y are real numbers and $i = \sqrt{-1}$. We can add, subtract, multiply, and divide complex numbers in a similar manner to how we would with real numbers. If we say that $z = x + iy$ and say that $w = a + ib$, then the following properties hold

$$z \pm w = (x \pm a) + i(y \pm b),$$

$$z \cdot w = ax - by + i(xb + ay).$$

If $z = x + iy$, then the conjugate is defined as

$$\bar{z} = x - iy.$$

The idea of division is expressed by the multiplicative inverse. We define the multiplicative inverse, denoted by z^{-1}, as

$$z^{-1} = \frac{1}{z} = \frac{1}{x + iy} = \frac{1}{x + iy} \cdot \frac{x - iy}{x - iy} = \frac{x - iy}{x^2 + y^2}.$$

In general, we try to avoid leaving an *i* in the denominator of the expression, so we use the conjugate to rationalize the expression. This will leave a real value in the denominator.

Complex numbers can also be represented in exponential form. The exponential form, or polar form, is a very useful tool in the world of complex variables. To express a complex number $z = x + iy$ in polar form, we should let $x = rcos(\theta)$ and $y = rsin(\theta)$. Then *z* becomes

$$z = r\big(\cos(\theta) + i\sin(\theta)\big),$$

where *r* is the distance from the origin to the point and *θ* is the angle from the positive real axis to the point. Through Euler’s formula it can be shown that *z* becomes

$$z = re^{i\theta}.$$

This identity is very useful throughout complex variables, especially in Fourier series. An important quality about complex variables is that the rules of exponents used for real-valued numbers transfer quite seamlessly to complex-valued numbers. That is

$$e^{i\theta_1}e^{i\theta_2} = e^{i(\theta_1 + \theta_2)}.$$

Using this idea, it can be seen that

$$z^n = \left(re^{i\theta}\right)^n = r^n e^{in\theta},$$

where *n* is an integer.

The general rules for differential calculus still apply for complex variables. Differentiation leads to the idea of integration. We will use the ideas of differentiation to help define the idea of integration. To begin, let $w(t)$ be a complex-valued function consisting of the two real valued functions $u(t)$ and $v(t)$ where $w(t) = u(t) + iv(t)$. Then the definite integral over $t \in [a, b]$ of $w(t)$ is

$$\int_a^b w(t)dt = \int_a^b u(t)dt + i\int_a^b v(t)dt\,,$$

assuming that $\int_a^b u(t)dt$ and $\int_a^b v(t)dt$ exist.

Major properties of integration transfer over quite smoothly to complex functions. A more in-depth discussion of complex variables can be found in [2]. These ideas help with the manipulation of Fourier series, which are discussed in the following section.

Section 3.2 Fourier Series

The principle idea behind Fourier series is that any function of any continuous variable *t* that is periodic and has a period *T* can be written as the sum of cosines and sines. More precisely, the Fourier series of a function $f(t)$ with period *T* is defined as

$$F(t) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{i2\pi n}{T}t}\,,$$

where

$$c_n = \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)e^{-i2\pi n/T}dt\,.$$

When that f is a 2π-periodic function, the Fourier series $F(\omega)$ is

$$F(\omega) = \sum_{k=-\infty}^{\infty} c_k e^{ik\omega},$$

where c_n is defined as

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(\omega)e^{-ik\omega}d\omega, \qquad k \in \mathbb{Z}.$$

Recall Euler's Formula $e^{i\theta} = \cos(\theta) + isin(\theta)$. We can see $f(\omega)$ can be recognized as a series of sine and cosine functions. Fourier series give an approximation of the original function.

For example, consider the piecewise function

$$f(x) = \begin{cases} 1, & 0 < x < \pi \\ -1, & -\pi < x < 0 \end{cases}.$$

Using Maple [11] we see that our truncated Fourier series is

$$F(u) = \sum_{k=1}^{u} c_k \sin(kx),$$

where

$$c_k = -\frac{-1 + \cos(\pi k)}{\pi k}.$$

The graph of the truncated Fourier series as compared with the original function is seen in Figure 3.2.1.

In a similar manner, we can calculate the Fourier transform of the identity function

$$f(x) = x, \qquad -\pi \le x \le \pi.$$

Here we see that the truncated Fourier transform is

$$F(u) = 2\sum_{k=1}^{u} c_k \sin(kx),$$

where

$$c_k = -\frac{-\sin(\pi k) + \cos(\pi k)\,\pi k}{\pi k^2}.$$

To view the graph of the Fourier transform as compared with the original function, view Figure 3.2.2.

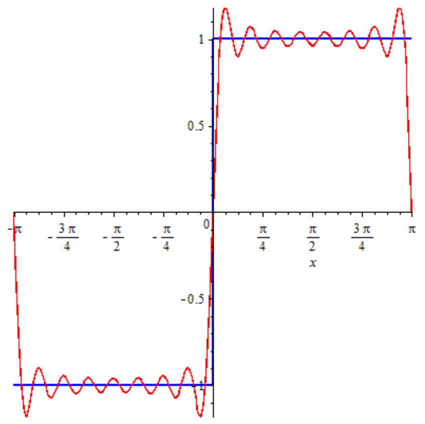


Figure 3.2.1: Fourier transform of a piecewise function

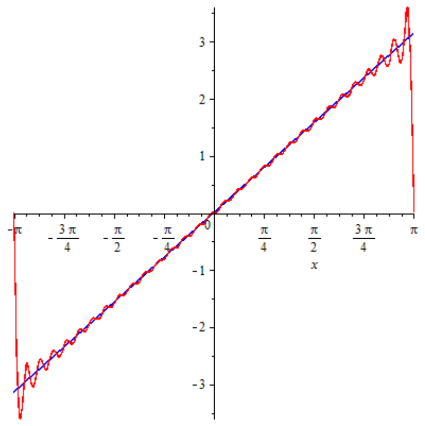


Figure 3.2.2: Fourier transform of $f(x) = x$

Section 3.3 Impulses

When processing a signal, we sometimes need to catch the impulse of the signal at a particular time. For this purpose, we define the impulse function $\delta(t)$. The unit impulse, also known as the delta function, is defined as

$$\delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases}.$$

Note that this is a generalized function, or a distribution. This definition also has the constraint that

$$\int_{-\infty}^{\infty} \delta(t)dt = 1.$$

A way to interpret this is by letting *t* be time. Then we have one point where the amplitude is infinity (at $t = 0$), that has a duration of 0 units of time (seconds, milliseconds, etc.), and the area of this function is 1. The impulse function has a special property known as the sifting property. The sifting property means that the function has the property

$$\int_{-\infty}^{\infty} f(t)\delta(t)dt = f(0),$$

where $f(t)$ is continuous at $t = 0$. A more generalized formula for the sifting property allows for the impulse to be located at any $t = t_0$, as opposed to just at $t = 0$. This is denoted by $\delta(t - t_0)$ and the sifting property is now expressed by

$$\int_{-\infty}^{\infty} f(t)\delta(t - t_0)dt = f(t_0).$$

For example, if $f(t) = \tan(t)$ and $t_0 = \frac{\pi}{4}$, then

$$\int_{-\infty}^{\infty} f(t)\delta\left(t - \frac{\pi}{4}\right)dt = f\left(\frac{\pi}{4}\right) = \tan\left(\frac{\pi}{4}\right) = 1.$$

The delta function and the sifting property also apply when a discrete variable is being used in place of a continuous function.

The unit discrete impulse $\delta(x)$, where *x* is a discrete variable, works in the same manner as $\delta(t)$ does where *t* is a continuous variable. The discrete impulse is defined in a very similar manner to the continuous impulse. That is

$$\delta(x) = \begin{cases} 1, & x = 0 \\ 0, & x \neq 0 \end{cases}.$$

Since the continuous variable satisfies

$$\int_{-\infty}^{\infty} \delta(t)dt = 1,$$

the discrete case must satisfy the discrete equivalent to this, which gives us the equation

$$\sum_{x=-\infty}^{\infty} \delta(x) = 1.$$

$\delta(x)$ also satisfies the sifting property

$$\sum_{x=-\infty}^{\infty} f(x)\delta(x) = f(0).$$

The generalized sifting property for a discrete variable can be derived in a very similar manner to that of the generalized sifting property for a continuous variable. That is

$$\sum_{x=-\infty}^{\infty} f(x)\delta(x - x_0) = f(x_0).$$

In a similar manner to the continuous variable, the sifting property gives the value of the function at the impulse location. However, there is a significant difference between the continuous variable and the discrete variable. That is, a discrete variable can be an ordinary function, while a continuous variable cannot. One other consequence of the impulse and sifting property is the idea of the impulse train. An impulse train is the sum of infinitely many periodic impulses ΔT units apart and is defined as

$$s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta T).$$

These impulses on the impulse train can be either continuous or discrete. The idea of the impulse and the impulse train also have certain implications when dealing with Fourier transform and their properties.

Section 3.4 Fourier Transforms and Properties

The Fourier transform is one of the most important tools in signal processing. The Fourier transform $\mathcal{F}\{f(t)\}$ of a continuous function $f(t)$ of the continuous variable *t* is defined as

$$\mathcal{F}\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-i2\pi\mu t}dt\,,$$

where μ is a continuous variable as well. Since the Fourier transform is an integral with respect to *t*, then the resulting function is a continuous function of μ. The variable μ is often referred to as the variable of frequency. Hence, we also denote $\mathcal{F}\{f(t)\} = F(\mu)$. The inverse transform, denoted by $\mathcal{F}^{-1}\{F(\mu)\}$, is

$$f(t) = \mathcal{F}^{-1}\{F(\mu)\} = \int_{-\infty}^{\infty} F(\mu)e^{i2\pi\mu t}d\mu\,.$$

The equations for $f(t)$ and $F(\mu)$ make what are known as a Fourier transform pair. This is an important consequence because this means that one may recover the original function from its transform.

For example, if we let $f(t)$ be a window function

$$f(t) = \begin{cases} A, & \text{if} -\dfrac{W}{2} \le t \le \dfrac{W}{2} \\ 0, & \text{Otherwise} \end{cases},$$

then we can see that the Fourier transform of $f(t)$ is

$$F(\mu) = \int_{-\infty}^{\infty} f(t)e^{-i2\pi\mu t}dt = \int_{-W/2}^{W/2} Ae^{-i2\pi\mu t}dt = \frac{-A}{2i\pi\mu}\left(e^{-Wi\pi\mu} - e^{Wi\pi\mu}\right),$$

where $A \in \mathbb{R}$, W is the width, and μ is a continuous variable. Recall that μ is a frequency variable. Thus the Fourier transform is in the frequency domain. It should be noted that the units of μ are dependent upon the units of the continuous variable t. In general, μ is in terms of cycles per unit of t.

When one takes the Fourier Transform of the unit impulse located at the origin, the output is

$$F(\mu) = \int_{-\infty}^{\infty} \delta(t) e^{-i2\pi\mu t} dt = e^{-i2\pi\mu 0} = e^0 = 1.$$

For a Fourier transform of an impulse where $t = t_0$ the output is

$$F(\mu) = \int_{-\infty}^{\infty} \delta(t - t_0) e^{-i2\pi\mu t} dt$$

$$= e^{-i2\pi\mu t_0}$$

$$= \cos(2\pi\mu t_0) - i\sin(2\pi\mu t_0).$$

Another useful property of the Fourier transform involves the modulus or the magnitude of the Fourier transform. Using the previous example of the window function, we get

$$|F(\mu)| = AW \left|\frac{\sin(\pi\mu W)}{\pi\mu W}\right|.$$

$|F(\mu)|$ is called the Fourier spectrum or the frequency spectrum. Some important observations of this example are that the zeros of $F(\mu)$ and $|F(\mu)|$ are inversely proportional to the width W. The heights of the waves decrease the further the function gets from the origin.

To show an example that will be used later, we will find the Fourier transform of an impulse train. We will first prove the following property of the Fourier transform (this proof follows from [7]).

Proposition 3.4.1: If $f(t)$ has the Fourier transform $F(\mu)$, then $F(t)$ must have the Fourier transform $f(-\mu)$.

Proof: To prove this, we start by stating that, by definition, the Fourier transform of $F(t)$ is

$$F\{F(t)\} = \int_{-\infty}^{\infty} F(t)e^{-i2\pi\mu t}dt. \tag{3.4.1}$$

We also know from definition of the inverse Fourier transform that

$$f(t) = \int_{-\infty}^{\infty} F(\mu)e^{i2\pi\mu t}d\mu\,,$$

which implies that by replacing μ by t' we get

$$f(t) = \int_{-\infty}^{\infty} F(t')e^{i2\pi t' t}dt'\,.$$

Therefore,

$$f(-\mu) = \int_{-\infty}^{\infty} F(t')e^{-i2\pi t'\mu}dt'. \tag{3.4.2}$$

By comparing equation (3.4.1) and equation (3.4.2) we see that the Fourier transform of $F(t)$ is equal to $f(-\mu)$. ■

The next property that we will prove is as follows (this proof follows from [7]):

Proposition 3.4.2: The Fourier transform of $e^{i2\pi at}$ is $\delta(u-a)$.

Proof: To prove this, we start by stating that it is known that the Fourier transform of $\delta(t-t_0)$ is $e^{-i2\pi\mu t_0}$. By Proposition 3.4.1 we can see that the Fourier transform of $e^{-i2\pi t_0 t}$ is $\delta(-\mu - t_0)$. Let $-t_0 = a$. We can see that the Fourier transform of $e^{i2\pi at}$ is

$\delta(-\mu + a)$. Since $\delta(-\mu + a)$ is nonzero only when $\mu = a$, then

$$\delta(-\mu + a) = \delta(\mu - a).$$

Therefore,

$$F\{e^{i2\pi at}\} = \delta(\mu - a). \qquad \blacksquare$$

A particular example of Proposition 3.4.2 is

$$F\left\{e^{i\frac{2\pi n}{\Delta T}t}\right\} = \delta\left(\mu - \frac{n}{\Delta T}\right).$$

Now that we have properties 3.4.1 and 3.4.2, we can prove the Fourier transform of an impulse train is also an impulse train.

To begin the proof we start by letting

$$S_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta T).$$

Then by the definition of the Fourier series we see that

$$S_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi n}{\Delta T}t},$$

where

$$c_n = \frac{1}{\Delta T}\int_{-\Delta T/2}^{\Delta T/2} S_{\Delta T}(t) e^{-i\frac{2\pi n}{\Delta T}t} dt.$$

The only nonzero delta function in $S_{\Delta \mathrm{T}}(t)$ over $\left[-\frac{\Delta T}{2}, \frac{\Delta T}{2}\right]$ is $\delta(t)$. Therefore

$$c_n = \frac{1}{\Delta T}\int_{-\Delta T/2}^{\Delta T/2} \delta(t) e^{-i\frac{2\pi n}{\Delta T}t} dt.$$

Thus

$$c_n = \frac{1}{\Delta T} e^0 = \frac{1}{\Delta T}.$$

By substituting $C_n = \frac{1}{\Delta T}$ into Fourier series of $S_{\Delta T}(t)$, we get

$$S_{\Delta T}(t) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} e^{i\frac{2\pi n}{\Delta T}t}.$$

Hence

$$F\{S_{\Delta T}(t)\} = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left\{e^{i\frac{2\pi n}{\Delta T}t}\right\}.$$

Therefore

$$F\{S_{\Delta T}\} = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \delta\left(\mu - \frac{n}{\Delta T}\right).$$

Thus it can be seen that the Fourier transform of an impulse train is also an impulse train. If the original impulse train has a period of ΔT, then the Fourier transform of the impulse train has a period of $\frac{1}{\Delta T}$. The ideas of the Fourier transform, combined with the idea of the convolution, discussed in Section 3.6, are needed to build the Sampling Theorem in Section 3.7.

Section 3.5 Linear Time-Invariant Systems

We will discuss the idea of linear time-invariant (LTI) systems in this section. We will see that a LTI system will naturally lead to the concept of convolution.

To discuss linear time-invariant systems, we will first define what a system is. We say a device is a system if it takes some sort of input signal $x(n)$, processes it, and provides an output $y(n)$. $y(n)$ at any time may depend on all inputs of $x(n)$. That is, it depends on $\dots x(n-2), x(n-1), x(n), x(n+1), x(n+2), \dots$.

A generic way showing a diagram of a system is

$$x(n) \rightarrow [SYSTEM] \rightarrow y(n).$$

This does not mean that the output $y(n)$ is dependent solely on $x(n)$. A system is a beneficial way of displaying real world devices such as population models or business models.

Expanding upon the idea of a system is the idea of a linear system. If $x_1(n) \to [SYSTEM] \to y_1(n)$ and $x_2(n) \to [SYSTEM] \to y_2(n)$, then a linear system is a system that has the property $\big(x_1(n) + x_2(n)\big) \to [SYSTEM] \to \big(y_1(n) + y_2(n)\big)$. This is known as superposition, and is not limited to two inputs. A linear system also has the scaling property. That is, $ax(n) \to [SYSTEM] \to ay(n)$, where a is a real number.

We define a time-invariant (TI) system by saying that if we delay the input by any constant *T*, then the output is delayed by *T* as well. Consider the system $y(n) = \sin\big(x(n)\big)$. By shifting the value of *n*, we shift the value of $y(n)$. That is, by adjusting $x(n)$, we adjust $y(n)$.

A linear time-invariant system (LTI) is one that is both a linear system and a time-invariant system. Suppose $h(n)$ is the response to an impulse $\delta(n)$ in a LTI system. That is

$$\delta(n) \to [LTI] \to h(n). \tag{3.5.1}$$

Since the LTI system is time-invariant, then using equation (3.5.1) we see that

$$\delta(n-a) \to [LTI] \to h(n-a), \tag{3.5.2}$$

for any constant a. Since the LTI system is also linear, using equation (3.5.2) we see that

$$x(a)\delta(n-a) \to [LTI] \to x(a)h(n-a), \tag{3.5.3}$$

for any constant $x(a)$. Since the LTI system is linear, we know that the LTI system also has the property of superposition. Thus using the property of superposition we see that

$$\sum_{a=-\infty}^{\infty} x(a)\delta(n-a) \to [LTI] \to \sum_{a=-\infty}^{\infty} x(a)h(n-a). \tag{3.5.4}$$

In other words, a LTI system is completely characterized by its unit response $h(n)$. See Section 1.2 of [13] for more discussion on linear time-invariant systems. Also see [13] for more discussion on linear time-invariant systems. We want to point out the last expression in (3.5.4) is the convolution, which we will continue to discuss in the next section.

Section 3.6 Convolution

The convolution is an operation that can be defined between either discrete or continuous functions. The discussion in this section was derived from [6].

In the discrete case, the convolution is an operation that happens between two bi-infinite series. On page 128 of [6], the convolution is defined as follows: Let $\boldsymbol{h}$ and $\boldsymbol{x}$ be two bi-infinite sequences. Then the convolution product $\boldsymbol{y}$ of $\boldsymbol{h}$ and $\boldsymbol{x}$, denoted by $\boldsymbol{h} * \boldsymbol{x}$, is the bi-infinite sequence $\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}$, whose nth component is given by

$$y_n = \sum_{k=-\infty}^{\infty} h_k x_{n-k}\,.$$

In a manner of speaking, the convolution involves reversing the order of the second bi-infinite and shifting it past the first bi-infinite sequence. At each point in the shift a computation is performed which is the sum of the products. This gives the convolution of two discrete bi-infinite sequences. However, the convolution of two continuous functions can also be performed. If $f(t)$ and $g(t)$ are two continuous functions of a single continuous variable *t*, then the convolution is defined by

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau\,.$$

where τ is a dummy variable.

When taking the Fourier transform of the convolution, there are two different cases that amount to the same idea. That is, in both the discrete case and the continuous case, the results of the Fourier transform of the convolution are equal to the product of the Fourier transform of the two functions. In other words, let $F(\mu)$ the Fourier transform of $f(t)$ and let $G(\mu)$ be the Fourier transform of $g(t)$. Then

$$\mathcal{F}\{f(t) * g(t)\} = F(\mu)G(\mu).$$

Since t is generally referred to as the spatial domain and μ is generally referred to as the frequency domain, then it can be seen that Fourier transform of the convolution of two functions gives the product of the Fourier transform of the same two functions. The converse of this is true as well. That is, the product of two Fourier transforms implies that one can find the convolution of the two base functions by applying the inverse Fourier transform. This shows that $f(t) * g(t)$ and $F(\mu)G(\mu)$ form a Fourier transform pair.

The ideas of the convolution discussed in this section, along with the idea of the Fourier transform and its properties discussed in Section 3.4, help us build a process which will be used to sample a function.

Section 3.7 The Fourier Transform of Sampled Functions

Sampling is a process used to transform continuous functions into discrete values so that a computer can process them. Sampling takes samples at some uniform interval ΔT of a continuous function $f(t)$ of a continuous variable t.

In order to sample a function we can use a sampling function to take a uniform sampling of a continuous function $f(t)$. To do this, let the sampling function be an impulse train ΔT units apart. Hence the sampled function $\tilde{f}(t)$ is

$$\begin{aligned}\tilde{f}(t) &= f(t)s_{\Delta T}(t)\\ &= \sum_{n=-\infty}^{\infty} f(t)\delta(t-n\Delta T)\,.\end{aligned}$$

It can be shown that the *k*th value in the sampled sequence can be determined by

$$\begin{aligned}f_k &= \int_{-\infty}^{\infty} f(t)\delta(t-k\Delta T)dt\,, \qquad k \in \mathbb{Z}\\ &= f(k\Delta T), \qquad k \in \mathbb{Z}.\end{aligned}$$

Doing this gives equally spaced samples of the function $f(t)$ that are spaced ΔT units apart.

Letting $F(\mu)$ be the Fourier transform of $f(t)$, the convolution theorem tells us that the Fourier transform $\tilde{F}(\mu)$ of the sampling function $\tilde{f}(t)$ is

$$\tilde{F}(\mu) = F(\mu) * S(\mu),$$

where, from Section 3.4,

$$S(\mu) = \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty} \delta\left(\mu - \frac{n}{\Delta T}\right)$$

is the Fourier transform of the impulse train $s_{\Delta T}(t)$. Using the convolution defined in Section 3.6 we see that

$$\begin{aligned}\tilde{F}(\mu) &= F(\mu) * S(\mu)\\ &= \int_{-\infty}^{\infty} F(\tau)S(\mu-\tau)d\tau\\ &= \frac{1}{\Delta T}\int_{-\infty}^{\infty} F(\tau)\sum_{n=-\infty}^{\infty} \delta\left(\mu-\tau-\frac{n}{\Delta T}\right)d\tau\\ &= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty}\int_{-\infty}^{\infty} F(\tau)\delta\left(\mu-\tau-\frac{n}{\Delta T}\right)d\tau\end{aligned}$$

$$= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left(\mu - \frac{n}{\Delta T}\right).$$

Hence we find that

$$\tilde{F}(\mu) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left(\mu - \frac{n}{\Delta T}\right).$$

It can be seen from this that $\tilde{F}(\mu)$ is periodic and infinite (since $\tilde{F}(\mu)$ is built from a periodic infinite function) as well as translations of the Fourier transform of the original function $f(t)$. The distance between copies is $\frac{1}{\Delta T}$. It should be noted that the accuracy of $\tilde{F}(\mu)$ depends on $\frac{1}{\Delta T}$. If $\frac{1}{\Delta T}$ is too small, just right, or too large, then the function is over-sampled, critically-sampled, or under-sampled, respectively. Refer to [7] for more discussion on this topic. This observation leads to the discussion of the Sampling Theorem in the next section.

Section 3.8 The Sampling Theorem and Aliasing

This section will be devoted to building the Sampling Theorem and ideas regarding aliasing.

A band-limited function $f(t)$ is a function whose Fourier transform has a value of zero outside of a finite interval $[-\mu_{max}, \mu_{max}]$. If a proper sample is taken, then we can recover the original function $f(t)$. This requires that a copy of $F(\mu)$ be isolated from the periodic sequence of copies contained in $\tilde{F}(\mu)$. Recall that from Section 3.7 we have

$$\tilde{F}(\mu) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left(\mu - \frac{n}{\Delta T}\right).$$

Therefore $\tilde{F}(\mu)$ is a continuous, periodic function with a period of $\frac{1}{\Delta T}$. If we can isolate a complete copy of $F(\mu)$ from $\tilde{F}(\mu)$, we can recover $f(t)$ by applying inverse Fourier transform on the copy of $F(\mu)$.

Suppose that *f* is a band-limited function whose Fourier transform is nonzero between $-\mu_{max}$ and μ_{max}. In order for us to extract a copy of $F(\mu)$ from $\tilde{F}(\mu)$, the separation between each period must be sufficient. That is, for a given period, a sufficient separation is assured if

$$\frac{1}{\Delta T} > 2\mu_{max}.$$

Thus we have the following theorem, the Sampling Theorem:

Theorem 3.8.1. A continuous, band-limited function can be recovered completely from a set of its samples, if the samples are acquired at a rate exceeding twice the highest frequency content of the function [7].

That is, if the sample rate is greater than twice the highest frequency content of the function, then there is no information lost. In other terms, the maximum frequency that can be acquired is $\mu_{max} = \frac{1}{2\Delta T}$. This is known as the Nyquist rate. It is generally advised via the sampling theorem that one use a sampling rate greater than the Nyquist rate. If the sampling rate is below the Nyquist rate, it is called under-sampling.

In order to recover the original function, we can create a function

$$H(\mu) = \begin{cases} \Delta T, & -\mu_{max} \le \mu \le \mu_{max} \\ 0, & \text{otherwise} \end{cases}.$$

If we multiply $\tilde{F}(\mu)$ by $H(\mu)$ we get

$$F(\mu) = H(\mu)\tilde{F}(\mu).$$

Thus we can obtain $f(t)$ by using the inverse Fourier transform discussed in Section 3.4.

Aliasing corresponds with a function that is under-sampled. Not only does under-sampling cause the periods to overlap, but it is now impossible to isolate a single period of the Fourier transform, and thus impossible to find the original function. If we were to try to find the original function using an under-sampled function, then the function returned would be corrupted. This is known as aliasing. See [7] for more on this topic.

Section 3.9 Reconstruction from Sampled Data

To reconstruct the original function from sets of samples, we only need to interpolate the data. First we need to prove a property. The proof of this follows from the discussion in [7].

Proposition 3.9.1. If

$$H(\mu) = \begin{cases} \Delta T, & -\frac{1}{2\Delta T} \leq \mu \leq \frac{1}{2\Delta T}, \\ 0, & \text{Otherwise} \end{cases}$$

then

$$h(t) = \operatorname{sinc}\left(\frac{t}{\Delta T}\right),$$

where

$$\operatorname{sinc}(\phi) = \frac{\sin(\phi)}{\phi}.$$

Proof: We start by stating that

$$h(t) = F^{-1}\{H(\mu)\} = \int_{-\infty}^{\infty} H(\mu) e^{i2\pi\mu t} d\mu\,.$$

By replacing $H(\mu)$ with the conditions of proposition 3.9.1, we see that

$$h(t) = \int_{-\frac{1}{2\Delta T}}^{\frac{1}{2\Delta T}} \Delta T e^{i2\pi\mu t} d\mu$$

$$= \left[\frac{\Delta T}{i2\pi t} e^{i2\pi\mu t}\right]_{-\frac{1}{2\Delta T}}^{\frac{1}{2\Delta T}}$$

$$= \frac{\Delta T}{i2\pi t}\left(e^{\frac{i\pi t}{\Delta T}} - e^{-\frac{i\pi t}{\Delta T}}\right)$$

$$= \frac{\Delta T}{i2\pi t}\left(2it\sin\left(\frac{i\pi t}{\Delta T}\right)\right)$$

$$= \frac{\Delta T}{\pi t}\sin\left(\frac{\pi t}{\Delta T}\right)$$

$$= \frac{\sin\left(\frac{\pi t}{\Delta T}\right)}{\frac{\pi t}{\Delta T}}$$

$$= \operatorname{sinc}\left(\frac{\pi t}{\Delta T}\right).$$

To reconstruct the function, the use of the convolution theorem from Section 3.6 will be needed. That is

$$f(t) = \mathcal{F}^{-1}\{F(\mu)\}$$

$$= \mathcal{F}^{-1}\{H(\mu)\tilde{F}(\mu)\}$$

$$= h(t) * \tilde{f}(t).$$

Hence

$$f(t) = h(t) * \left(f(t)s_{\Delta T}(t)\right)$$

$$= h(t) * \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)$$

$$= \sum_{n=-\infty}^{\infty} h(t) * \left(f(t)\delta(t - n\Delta T)\right)$$

$$= \sum_{n=-\infty}^{\infty} f(\tau)\delta(\tau - n\Delta T)h(t - \tau)$$

$$= \sum_{n=-\infty}^{\infty} f(n\Delta T)h(t - n\Delta T)$$

$$= \sum_{n=-\infty}^{\infty} f(n\Delta T)\text{sinc}\left(\frac{(t - n\Delta T)}{\Delta T}\right),$$

where the last step is a consequence of proposition 3.9.1. That is, the values of $f(t)$ that are between the sample points are interpolations that are created by the sum of the sinc functions. Since these require an infinite sum, an approximation is generally used in practice.

Section 3.10 The Discrete Fourier Transform

The discrete Fourier transform is crucial for signal processing. This section will cover the discrete Fourier transform.

Suppose that $\boldsymbol{f} = (f_0, f_1, \dots, f_{N-1})$ is a discrete signal. Define the discrete Fourier transform as

$$F_m = \sum_{n=0}^{N-1} f_n e^{-\frac{2i\pi mn}{M}}, \qquad m = 0, 1, \dots, M - 1.$$

This is the discrete Fourier transform. Also, if we have the set $\{F_m\}$, we can find f_n by using the inverse discrete Fourier transform

$$f_n = \frac{1}{M} \sum_{m=0}^{M-1} F_m e^{\frac{i2\pi mn}{M}}, \qquad n = 0, 1, \dots, N - 1.$$

By substituting f_n into the equation for F_m and vice versa, the equations simplify down to identities, which means that the equations for f_n and F_m form a discrete Fourier transform pair.

An important implication from these equations is that the inverse Fourier transform exists for any set of samples with a finite cardinality. Note that neither f_n nor

F_m depend on the interval of the sample ΔT. Hence, if the sample is finite, discrete, and taken uniformly, then the discrete Fourier transform pair can be applied.

For example, consider $\boldsymbol{f} = (f_0, f_1, f_2, f_3) = (1, 2, 3, 4)$. Then we see that

$$F(0) = \sum_{x=0}^{3} f(x)$$

$$= f(0) + f(1) + f(2) + f(3)$$

$$= 1 + 2 + 3 + 4$$

$$= 10$$

$$F(1) = \sum_{n=0}^{3} f(n)e^{-\frac{i2\pi(1)n}{4}}$$

$$= f(0) + f(1)e^{-\frac{i\pi}{2}} + f(2)e^{-i\pi} + f(3)e^{-\frac{3i\pi}{2}}$$

$$= 1 + 2(-i) + 3(-1) + 4(i)$$

$$= -2 + 2i.$$

Likewise, we can show that $F(2) = -2$ and $F(3) = -2 - 2i$.

If we were, instead, given F and we wanted the inverse, then we would follow a similar method. For example:

$$f(0) = \frac{1}{4}\sum_{\mu=1}^{3} F(\mu)e^{i2\pi\mu(0)}$$

$$= \frac{1}{4}\sum_{\mu=1}^{3} F(\mu)$$

$$= \frac{1}{4}(10 - 2 + 2i - 2 - 2 - 2i)$$

$$= 1.$$

Even though these processes are easy to understand, they can be very cumbersome when *M* and *N* are large. However, the fast Fourier transform tackles that problem for us. For more on the Fourier transform, refer to [6].

Section 3.11 The Fast Fourier Transform

Implementing the discrete Fourier transform and the inverse discrete Fourier transform can be extremely time consuming and take an extreme amount of processor power. There are roughly $(MN)^2$ summations and additions that must be done in these equations. However, a discovery known as fast Fourier transforms (often denoted FFT) has allowed for a remarkable reduction in these operations. The fast Fourier transform allows for a reduction to $MNlog_2(MN)$ summations and additions. This section is derived from [7].

The successive-doubling method is the algorithm that allows the fast Fourier transform to work. One of the requirements for this method is that the number of samples is a power of 2. For simplicity, when dealing with the fast Fourier transform we normally express $F(u)$ by

$$F(u) = \sum_{x=0}^{M-1} f(x)W_M^{ux}, \qquad u = 0, 1, \dots, M-1$$

and

$$W_M = e^{-\frac{i2\pi}{M}},$$

where

$$M = 2^n, \qquad n = 1, 2, \dots .$$

When *M* is even, M can be written as $M = 2k$, where *k* is an integer. Recalling that

$$F(u) = \sum_{x=0}^{2k-1} f(x) W_{2k}^{ux}$$

and the fact that $W_{2k}^{2ux} = W_k^{ux}$, it can be seen that

$$F(u) = \sum_{x=0}^{k-1} f(2x) W_k^{ux} + \sum_{x=0}^{k-1} f(2x+1) W_k^{ux} W_{2k}^{u}.$$

Define $F_{even}(u)$ and $F_{odd}(u)$ as

$$F_{even}(u) = \sum_{x=0}^{k-1} f(2x) W_k^{ux}$$

and

$$F_{odd}(u) = \sum_{x=0}^{k-1} f(2x+1) W_k^{ux},$$

then one can see that

$$F(u) = F_{even}(u) + F_{odd}(u) W_{2k}^{u}. \qquad (3.11.1)$$

Since $W_M^{u+M} = W_M^u$ and $W_{2M}^{u+M} = -W_{2M}^u$, it can also be shown that

$$F(u+k) = F_{even}(u) - F_{odd}(u) W_{2k}^{u}. \qquad (3.11.2)$$

The Equations (3.11.1) and (3.11.2) give the possibilities of constructing a recursion tool algorithm to compute the discrete Fourier transform.

We will use the following example to explain the notations and the process of FFT. Define *a* and *c* as

$$a = [a_0\ a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7]^T \quad c = [c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7]^T,$$

where *a* is the input and *c* is the output. When using the discrete Fourier transform we get:

$$c_0 = a_0 + a_1 e^{-i2\pi\frac{0\cdot 1}{8}} + a_2 e^{-i2\pi\frac{0\cdot 2}{8}} + \cdots + a_7 e^{-i2\pi\frac{0\cdot 7}{8}}$$

$$= a_0 + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$$

$$= (a_0 + a_2 + a_4 + a_6) + (a_1 + a_3 + a_5 + a_7).$$

In a similar manner, we can find c_1:

$$c_1 = a_0 + a_1 e^{-i2\pi\frac{1\cdot 1}{8}} + a_2 e^{-2i\pi\frac{1\cdot 2}{8}} + \cdots + a_7 e^{-2i\pi\frac{1\cdot 7}{8}}$$

$$= \left[a_0 + a_4 e^{-i2\pi\frac{4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{4}{8}}\right] e^{-i2\pi\frac{1}{8}}$$

$$+ \left\{\left[a_1 + a_5 e^{-i2\pi\frac{4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{4}{8}}\right] e^{-i2\pi\frac{2}{8}}\right\} e^{-i2\pi\frac{1}{8}}.$$

Likewise, c_2 is:

$$c_2 = a_0 + a_1 e^{-i2\pi\frac{2\cdot 1}{8}} + a_2 e^{-2i\pi\frac{2\cdot 2}{8}} + \cdots + a_7 e^{-2i\pi\frac{2\cdot 7}{8}}$$

$$= \left[a_0 + a_4 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}$$

$$+ \left\{\left[a_1 + a_5 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}\right\} e^{-i2\pi\frac{2}{8}}.$$

We find c_3 and c_4 the same way:

$$c_3 = a_0 + a_1 e^{-i2\pi\frac{3\cdot 1}{8}} + a_2 e^{-2i\pi\frac{3\cdot 2}{8}} + \cdots + a_7 e^{-2i\pi\frac{3\cdot 7}{8}}$$

$$= \left[a_0 + a_4 e^{-i2\pi\frac{3\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{3\cdot 4}{8}}\right] e^{-i2\pi\frac{3\cdot 2}{8}}$$

$$+ \left\{\left[a_1 + a_5 e^{-i2\pi\frac{3\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{3\cdot 4}{8}}\right] e^{-i2\pi\frac{3\cdot 2}{8}}\right\} e^{-i2\pi\frac{3}{8}}$$

$$c_4 = a_0 + a_1 e^{-i2\pi\frac{4\cdot 1}{8}} + a_2 e^{-2i\pi\frac{4\cdot 2}{8}} + \cdots + a_7 e^{-2i\pi\frac{4\cdot 7}{8}}$$

$$= [a_0 + a_2 + a_4 + a_6] - [a_1 + a_3 + a_5 + a_7].$$

Notice that c_4 and c_0 are very similar. The only difference is that the sign in the center is the opposite. We see notice a similar a familiarity when finding c_5:

$$c_5 = a_0 + a_1 e^{-i2\pi\frac{5\cdot 1}{8}} + a_2 e^{-2i\pi\frac{5\cdot 2}{8}} + \cdots + a_7 e^{-2i\pi\frac{5\cdot 7}{8}}$$

$$= \left[a_0 + a_4 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}$$

$$- \left\{\left[a_1 + a_5 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}\right\} e^{-i2\pi\frac{2}{8}}.$$

Notice that c_5 is nearly identical to c_1. A similar pattern arises for c_6 and c_7. That is, c_6 and c_7 are nearly identical to c_2 and c_3. Hence

$$c = [c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7]^T$$

$$= \begin{bmatrix} [a_0 + a_2 + a_4 + a_6] + [a_1 + a_3 + a_5 + a_7] \\ \left[a_0 + a_4 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}} + \left\{\left[a_1 + a_5 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}\right\} e^{-i2\pi\frac{2}{8}} \\ \left[a_0 + a_4 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}} + \left\{\left[a_1 + a_5 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}\right\} e^{-i2\pi\frac{2}{8}} \\ \left[a_0 + a_4 e^{-i2\pi\frac{3\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{3\cdot 4}{8}}\right] e^{-i2\pi\frac{3\cdot 2}{8}} + \left\{\left[a_1 + a_5 e^{-i2\pi\frac{3\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{3\cdot 4}{8}}\right] e^{-i2\pi\frac{3\cdot 2}{8}}\right\} e^{-i2\pi\frac{3}{8}} \\ [a_0 + a_2 + a_4 + a_6] - [a_1 + a_3 + a_5 + a_7] \\ \left[a_0 + a_4 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}} - \left\{\left[a_1 + a_5 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}\right\} e^{-i2\pi\frac{2}{8}} \\ \left[a_0 + a_4 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}} - \left\{\left[a_1 + a_5 e^{-i2\pi\frac{2\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{2\cdot 4}{8}}\right] e^{-i2\pi\frac{2\cdot 2}{8}}\right\} e^{-i2\pi\frac{2}{8}} \\ \left[a_0 + a_4 e^{-i2\pi\frac{3\cdot 4}{8}}\right] + \left[a_2 + a_6 e^{-i2\pi\frac{3\cdot 4}{8}}\right] e^{-i2\pi\frac{3\cdot 2}{8}} - \left\{\left[a_1 + a_5 e^{-i2\pi\frac{3\cdot 4}{8}}\right] + \left[a_3 + a_7 e^{-i2\pi\frac{3\cdot 4}{8}}\right] e^{-i2\pi\frac{3\cdot 2}{8}}\right\} e^{-i2\pi\frac{3}{8}} \end{bmatrix}$$

$$= \begin{bmatrix} [a_0 + a_4] + [a_2 + a_6] + [a_1 + a_5] + [a_3 + a_7] \\ [a_0 - a_4] + [a_2 - a_6] e^{-i\frac{2\pi}{4}} + \left\{[a_1 - a_5] + [a_3 - a_7] e^{-i\frac{2\pi}{4}}\right\} e^{-i\frac{2\pi}{8}} \\ [a_0 + a_4] - [a_2 + a_6] + \{[a_1 + a_5] - [a_3 + a_7]\} e^{-i\frac{2\pi}{4}} \\ [a_0 - a_4] - [a_2 - a_6] e^{-i\frac{2\pi}{4}} + \left\{[a_1 - a_5] - [a_3 - a_7] e^{-i\frac{2\pi}{4}}\right\} e^{-i\frac{2\pi}{4}} \\ [a_0 + a_4] + [a_2 + a_6] - \{[a_1 + a_5] + [a_3 + a_7]\} \\ [a_0 - a_4] + [a_2 - a_6] e^{-i\frac{2\pi}{4}} - \left\{[a_1 - a_5] + [a_3 - a_7] e^{-i\frac{2\pi}{4}}\right\} e^{-i\frac{2\pi}{8}} \\ [a_0 - a_4] - [a_2 - a_6] - \{[a_1 - a_5] - [a_3 - a_7]\} e^{-i\frac{2\pi}{4}} \\ [a_0 - a_4] - [a_2 - a_6] e^{-i\frac{2\pi}{4}} - \left\{[a_1 - a_5] - [a_3 - a_7] e^{-i\frac{2\pi}{4}}\right\} e^{-i\frac{2\pi}{4}} \end{bmatrix}.$$

Let $w_k = e^{\frac{-2\pi i}{k}}$. Then

$$w_8^0 = 1, \qquad w_8^1 = e^{-\frac{2\pi i}{8}}, \qquad w_8^2 = e^{-\frac{2\pi i}{8}(2)}, \qquad w_8^3 = e^{-\frac{2\pi i}{8}(3)}$$

$$w_4^0 = 1, \qquad w_4^1 = e^{\frac{-2\pi i}{4}}$$

$$w_2^0 = 1.$$

Let c_{ee}, c_{eo}, c_{oe}, and c_{oo} be defined as

$$c_{ee} = \begin{bmatrix} a_0 + a_4 w_2^0 \\ a_0 - a_4 w_2^0 \end{bmatrix}, \quad c_{eo} = \begin{bmatrix} a_2 + a_6 w_2^0 \\ a_2 - a_6 w_2^0 \end{bmatrix}, \quad c_{oe} = \begin{bmatrix} a_1 + a_5 w_2^0 \\ a_1 - a_5 w_2^0 \end{bmatrix}, \quad c_{oo} = \begin{bmatrix} a_3 + a_7 w_2^0 \\ a_3 - a_7 w_2^0 \end{bmatrix}.$$

Then we can calculate c_e and c_o

$$c_e = \begin{bmatrix} c_{ee} + w_4 \cdot c_{eo} \\ c_{ee} - w_4 \cdot c_{eo} \end{bmatrix}$$

$$= \begin{bmatrix} a_0 + w_2^0 a_4 + w_4^0(a_2 + w_2^0 a_6) \\ a_0 - w_2^0 a_4 + w_4^0(a_2 - w_2^0 a_6) \\ a_0 + w_2^0 a_4 - w_4^0(a_2 + w_2^0 a_6) \\ a_0 - w_2^0 a_4 - w_4^0(a_2 - w_2^0 a_6) \end{bmatrix}$$

$$c_o = \begin{bmatrix} c_{oe} + w_4 \cdot c_{oo} \\ c_{oe} - w_4 \cdot c_{oo} \end{bmatrix}$$

$$= \begin{bmatrix} a_1 + w_2^0 a_3 + w_4^0(a_3 + w_2^0 a_7) \\ a_1 - w_2^0 a_3 + w_4^0(a_3 - w_2^0 a_7) \\ a_1 + w_2^0 a_3 - w_4^0(a_3 + w_2^0 a_7) \\ a_1 - w_2^0 a_3 - w_4^0(a_3 - w_2^0 a_7) \end{bmatrix}.$$

From this we can calculate *c* as

$$c = \begin{bmatrix} c_e + w_8 \cdot c_o \\ c_e - w_8 \cdot c_o \end{bmatrix} = [c_0\ c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7]^T.$$

Hence we can see that

$$c = \begin{bmatrix} a_0 + w_2^0 a_4 + w_4^0(a_2 + w_2^0 a_6) + w_8^0[a_1 + w_2^0 a_5 + w_4^0(a_3 + w_2^0 a_7)] \\ a_0 - w_2^0 a_4 + w_4^1(a_2 - w_2^0 a_6) + w_8^1[a_1 - w_2^0 a_5 + w_4^1(a_3 - w_2^0 a_7)] \\ a_0 + w_2^0 a_4 - w_4^0(a_2 + w_2^0 a_6) + w_8^2[a_1 + w_2^0 a_5 - w_4^0(a_3 + w_2^0 a_7)] \\ a_0 - w_2^0 a_4 - w_4^1(a_2 - w_2^0 a_6) + w_8^3[a_1 - w_2^0 a_5 - w_4^1(a_3 - w_2^0 a_7)] \\ a_0 + w_2^0 a_4 + w_4^0(a_2 + w_2^0 a_6) - w_8^0[a_1 + w_2^0 a_5 + w_4^0(a_3 + w_2^0 a_7)] \\ a_0 - w_2^0 a_4 + w_4^1(a_2 - w_2^0 a_6) - w_8^1[a_1 - w_2^0 a_5 + w_4^1(a_3 - w_2^0 a_7)] \\ a_0 + w_2^0 a_4 - w_4^0(a_2 + w_2^0 a_6) - w_8^2[a_1 + w_2^0 a_5 - w_4^0(a_3 + w_2^0 a_7)] \\ a_0 - w_2^0 a_4 - w_4^1(a_2 - w_2^0 a_6) - w_8^3[a_1 - w_2^0 a_5 - w_4^1(a_3 - w_2^0 a_7)] \end{bmatrix}.$$

This is an example of the fast Fourier transform. Notice that the discrete Fourier transform has 120 calculations while the fast Fourier transform only has 36 calculations.

As the size of the transformed matrix increases, so does the benefit of using the fast Fourier transform over the discrete Fourier transform. As we can see from the example, the fast Fourier transform has a computational advantage over the discrete Fourier transform.

Section 3.12 Windowed Short-Time Fourier Transforms

In many applications of Fourier transforms, we want to know the information about the frequency **and** the time. This is why a windowed short-time Fourier transform is needed. These Fourier transforms are similar to Fourier transforms discussed in Section 3.4. The only difference is that the interval in which they are studied is reduced. Refer to [14] to find more on this subject.

We want to obtain information about the frequency and the time. However, since the Fourier transform requires the entire set of data, we cannot obtain both of these at an instant. Thus we want our data to be contained in a certain window.

To start the windowed short-time Fourier transform (STFT), we will segment the signal into smaller parts called windows. Let $g(u)$ be a window if it is a function that dissipates outside of a finite interval $-T \leq u \leq 0$. Note that $g(u)$ can be a complex-valued function. We also define $f_t(u)$ as

$$f_t(u) \equiv \bar{g}(u-t)f(u).$$

We see that the support of the function, i.e. the interval over which the function is nonzero, is $[t-T, t]$. That implies that *f* depends only on the values of $f(u)$ on the interval $t - T \leq u \leq t$.

From here we define the windowed short-time Fourier transform as follows:

$$\tilde{f}(\omega, t) = \int_{-\infty}^{\infty} e^{-2\pi i \omega u} f_t(u)du = \int_{-\infty}^{\infty} e^{-2\pi i \omega u} \bar{g}(u-t)f(u)du.$$

We also know that in the case when $g(u) = 1$, then this becomes the ordinary Fourier transform.

Even though the STFT gives the information of both time and frequency, we can see it has to use a fixed window length.

Chapter 4 Introducing Filters and Wavelet Transformations

In this chapter we will explain the connection between convolution and matrix multiplication, focus on how to construct Haar and Daubechies filters, and Wavelet Packet Decomposition. An introduction to continuous transforms and a comparison between Fourier and wavelet transforms are also introduced. These ideas are intertwined with complex variables and Fourier transforms. The information in this chapter is derived from [6].

Section 4.1 An Introduction to Filters

We use the ideas of complex variables and linear algebra to build what are known as highpass and lowpass filters. In general, there are a few different obstacles that need to be addressed when building wavelet filters, such as orthogonality and invertibility.

To begin, we will start by defining a filter: To filter a signal $\boldsymbol{x}$ through a filter $\boldsymbol{h}$ we compute $\boldsymbol{y}$ by

$$\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}.$$

That is,

$$y_n = \sum_{k=-\infty}^{\infty} h_k x_{n-k}\,.$$

It should be noted that the way we build our filter $\boldsymbol{h}$ affects how $\boldsymbol{x}$ is influenced, and vice versa. A basic example of a filter is an averaging filter. Let

$$\boldsymbol{x} = (\dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots)$$

be a sequence of numbers. Let

$$\boldsymbol{h} = (h_0, h_1) = \left(\frac{1}{2}, \frac{1}{2}\right). \tag{4.1.1}$$

Then

$$\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}$$

with

$$y_n = \frac{x_n + x_{n-1}}{2}.$$

Then $\boldsymbol{h}$ is an averaging filter, which is sometimes also called a Haar filter. Now we will define some special types of filters that have certain attributes.

One special kind of filter is known as a causal filter. If $\boldsymbol{h}$ is a filter and $h_k = 0$ for all $k < 0$, then $\boldsymbol{h}$ is a causal filter. The averaging filter is a causal filter.

A second kind of special filter is a type of causal filter known as a finite impulse response (FIR) filter. To define a FIR filter $\boldsymbol{h}$, we start by letting $\boldsymbol{h}$ be a causal filter and assuming that L is a positive integer. If $h_k = 0$ for all $k > L$ where $h_0 \neq 0$ and $h_L \neq 0$, then $\boldsymbol{h}$ is a FIR filter and $\boldsymbol{h}$ can be written as

$$\boldsymbol{h} = (h_0, h_1, \dots, h_L).$$

There are two kinds of FIR filters that will be discussed here, a lowpass filter and a highpass filter.

Given a filter $\boldsymbol{h} = (h_0, \dots, h_L)$, consider the Fourier series

$$H(\omega) = \sum_{k=0}^{L} h_k e^{ik\omega}.$$

We call $\boldsymbol{h}$ a lowpass filter if we obtain the following properties:

$$|H(0)| = 1, \qquad |H(\pi)| = 0. \tag{4.1.2}$$

When a lowpass filter is applied on a signal, the lower end of the frequency is preserved while the upper end of the wave is annihilated.

It is easy to verify that $|H(0)| = 1$ if and only if

$$\sum_{k=0}^{L} h_k = \pm 1. \tag{4.1.3}$$

Similarly, $|H(\pi)| = 0$ if and only if

$$\sum_{k=0}^{L} (-1)^k h_k = 0. \tag{4.1.4}$$

The filter in equation (4.1.1) is a lowpass filter. To see this we can consider the properties in equations (4.1.2):

$$|H(0)| = \frac{1}{2} + \frac{1}{2} = 1,$$

$$|H(\pi)| = \frac{1}{2} - \frac{1}{2} = 0.$$

A highpass filter will preserve the high frequency of a signal and annihilate the lower frequency.

Given a filter $\boldsymbol{g} = (g_0, \dots, g_L)$, consider the Fourier series

$$G(\omega) = \sum_{k=0}^{L} g_k e^{ik\omega} .$$

Then $\boldsymbol{g}$ is a highpass filter if the filter $\boldsymbol{g}$ has the following properties:

$$|G(0)| = 0, \qquad |G(\pi)| = 1. \tag{4.1.5}$$

Similar to the lowpass filter, a consequence of these properties is that $|G(0)| = 0$ if and only if

$$\sum_{k=0}^{L} g_k = 0 \tag{4.1.6}$$

and $|G(\pi)| = 1$ if and only if

$$\sum_{k=0}^{L} (-1)^k g_k = \pm 1. \tag{4.1.7}$$

An example of a highpass filter is

$$\boldsymbol{g} = (g_0, g_1) = \left(\frac{1}{2}, -\frac{1}{2}\right), \tag{4.1.8}$$

because

$$|G(0)| = \frac{1}{2} - \frac{1}{2} = 0,$$

$$|G(\pi)| = \frac{1}{2} + \frac{1}{2} = 1.$$

As has been shown, wavelet filters can be expressed as a string of numbers. In the next section, we will see the convolution can be equivalently represented as a matrix multiplication. Hence, a filter will correspond to the rows in the matrix.

Section 4.2 Expressing Convolution as Matrix Multiplication

In Section 4.1 we showed that to apply a filter we only need to convolve the signal with it. However, there are ways in which we can represent the convolution as a matrix product. That is, we are attempting to rewrite $\boldsymbol{y} = \boldsymbol{h} * \boldsymbol{x}$, where $\boldsymbol{h}$ is a filter and $\boldsymbol{x}$ is a signal, as $\boldsymbol{y} = H\boldsymbol{x}$, where *H* is an infinite matrix.

To begin we will only consider FIR filters. Let $h = (h_0, \dots, h_L)$. Recall that a convolution is a linear operator. If we look at the definition of convolution (Section 3.6), we see that if $\boldsymbol{x}$ is the signal, $\boldsymbol{h}$ is the filter, and $\boldsymbol{y}$ is the result, then

$$y_n = \sum_{k=0}^{L} h_k x_{n-k} \, .$$

From this we can recognize y_n as the dot product of the nth row of matrix *H* and the vector $\boldsymbol{x}$. Let us consider the entry y_0, which is computed as

$$y_0 = \sum_{k=0}^{L} h_k x_{-k} \, .$$

Similarly,

$$y_1 = \sum_{k=0}^{L} h_k x_{1-k} \, .$$

In other words:

$$\begin{bmatrix} \vdots \\ y_0 \\ y_1 \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & h_L & \cdots & h_2 & h_1 & h_0 & 0 & \cdots \\ \cdots & 0 & 0 & h_L & \cdots & h_2 & h_1 & h_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x_{-L} \\ \vdots \\ x_{-2} \\ x_{-1} \\ x_0 \\ x_1 \\ \vdots \end{bmatrix}$$

If we continue the pattern and stack the rows of $\boldsymbol{h}$, we can build a matrix. That is

$$H = \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots\ddots \\ \cdots & h_L & h_{L-1} & \cdots & h_0 & 0 & 0 & \cdots \\ \cdots & 0 & h_L & \cdots & h_1 & h_0 & 0 & \cdots \\ \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots \\ \cdots & 0 & 0 & \cdots & 0 & h_L & h_{L-1} & \cdots \\ \cdots & 0 & 0 & \cdots & 0 & 0 & h_L & \cdots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

with h_0 down the main diagonal. Notice that *Hx* is equivalent to $\boldsymbol{h} * \boldsymbol{x}$.

Let us consider, for example, the filter in (4.1.1). This matrix would look like

$$H = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

As can be seen, this matrix will average the two consecutive values of the vector by which it is multiplied. Now consider the filter in equality (4.1.8). If we were to put this in a matrix, it would be represented as

$$G = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Matrix *H* is known as the average matrix, while matrix *G* is known as the difference matrix. By combining these two matrices together, we get

$$\begin{bmatrix} H \\ G \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} H\boldsymbol{x} \\ G\boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix}, \tag{4.2.1}$$

where $\boldsymbol{y}$ is the average of two consecutive terms and $\boldsymbol{z}$ is the half difference of two consecutive terms. Notice that *H* is contructed from a lowpass filter and *G* is constructed from a highpass filter. Having these two matrices combined will help us when trying to recover the original signal.

In theory, this matrix should be an infinite dimensional matrix. However, this is not very practical. Since our input signals are finite, we must truncate this matrix and do what is known as downsampling.

First we consider (4.2.1) where $\boldsymbol{x} = (\dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots)$. Then

$$\begin{bmatrix} H \\ G \end{bmatrix} \begin{bmatrix} \vdots \\ x_{-2} \\ x_{-1} \\ x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \\ \vdots \\ z_{-2} \\ z_{-1} \\ z_0 \\ z_1 \\ z_2 \\ \vdots \end{bmatrix},$$

where ***y*** is the average filter and ***z*** is the difference filter. That is

$$y_0 = \frac{x_1 + x_0}{2}$$

$$y_1 = \frac{x_2 + x_1}{2}$$

$$\vdots$$

$$z_0 = \frac{x_1 - x_0}{2}$$

$$z_1 = \frac{x_2 - x_1}{2}$$

$$\vdots$$

We are building a matrix that has two different filters that are related. The top portion of the result, ***y***, gives a rough approximation of ***x*** and the bottom portion, ***z***, tells us how far the value of ***x*** is from the approximation. This means that given the output vector, we can reconstruct the original input vector. For example:

$$x_0 = \frac{x_1 + x_0}{2} - \frac{x_1 - x_0}{2} = y_1 - z_1$$

$$x_1 = \frac{x_1 + x_0}{2} + \frac{x_1 - x_0}{2} = y_1 + z_1$$

$$x_2 = \frac{x_3 + x_2}{2} - \frac{x_3 - x_2}{2} = y_3 - z_3$$

$$x_3 = \frac{x_3 + x_2}{2} + \frac{x_3 - x_2}{2} = y_3 + z_3.$$

Note that y_0, z_0, y_2, and z_2 were not used. Hence we only need every other element of the output vector (the right-hand side) of (4.2.1). Thus we can shrink, or downsample, the matrix on the left-hand side of (4.2.1) to

$$H = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

and

$$G = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Hence our new matrix $\begin{bmatrix} H \\ G \end{bmatrix}$ is

$$\begin{bmatrix} H \\ G \end{bmatrix} = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \cdots \\ \cdots & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & \cdots \\ \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \qquad (4.2.2)$$

An important factor of this matrix is that the original input vector can easily be retrieved from the output vector. We will denote $\begin{bmatrix} H \\ G \end{bmatrix}$ by $\widetilde{W}_N$.

To finish off this matrix, we need to add one more property to it. A key element is that our matrix needs to be orthogonal.

Section 4.3 Orthogonality

Now that we have seen how a lowpass and a highpass filter can form a wavelet matrix, we will discuss how to characterize the orthogonality of the filters in this section.

A matrix U is an orthogonal matrix if $UU^T = I$, where I is the identity matrix.

To start, note that

$$\widetilde{W}_N \widetilde{W}_N^T = \frac{1}{2} I_N,$$

where I_N is the $N \times N$ identity matrix. In order to make $\widetilde{W}_N$ orthogonal, we must multiply $\widetilde{W}_N$ by $\sqrt{2}$. We label this matrix W_N and it is equal to

$$W_N = \sqrt{2}\widetilde{W}_N. \qquad (4.3.1)$$

This specific wavelet transform is known as the Haar wavelet transform and is a filter of length 2. A useful property is that

$$W_N^T = W_N^{-1}. \tag{4.3.2}$$

Generally, if we use two even length filters $\boldsymbol{h} = (h_0, \dots, h_L)$ and $\boldsymbol{g} = (g_0, \dots, g_L)$ to form a square matrix W, the orthogonality requirement on W will give

$$WW^T = I,$$

where $W = \begin{bmatrix} H \\ G \end{bmatrix}$. Hence we see that

$$\begin{bmatrix} H \\ G \end{bmatrix} \begin{bmatrix} H^T & G^T \end{bmatrix} = \begin{bmatrix} HH^T & HG^T \\ GH^T & GG^T \end{bmatrix} = I.$$

Thus

$$HH^T = GG^T = I, \qquad GH^T = HG^T = \boldsymbol{0},$$

where I is the identity matrix and $\boldsymbol{0}$ is the zero matrix. $HH^T = I$ and $HG^T = 0$ imply the following properties:

$$\sum_{k=0}^{L} h_k^2 = 1, \tag{4.3.3}$$

$$\sum_{k=2m}^{L} h_k h_{k-2m} = 0, \qquad m = 1, 2, \dots, \frac{L-1}{2} \tag{4.3.4}$$

Note that if we define g_k as

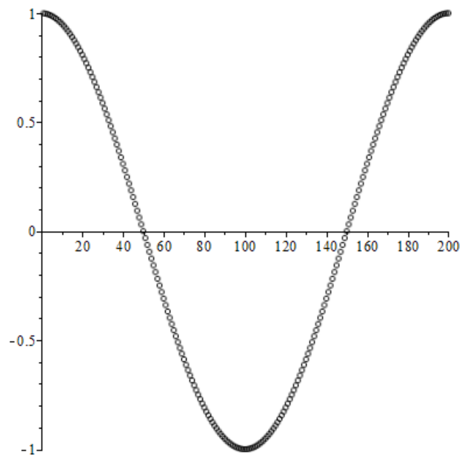
$$g_k = (-1)^k h_{L-k}, \tag{4.3.5}$$

it is easy to verify that $GG^T = I, HG^T = 0$, and $GH^T = 0$. $\boldsymbol{g}$ is also a highpass filter when $\boldsymbol{h}$ is a lowpass filter.

To see an example of the Haar wavelet transform, consider the function

$$f(t) = \cos(2\pi t). \tag{4.3.6}$$

If we plot 200 discrete points of this wave, it looks like Figure 4.3.1.



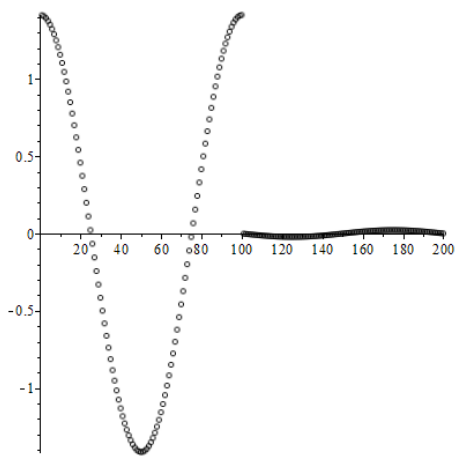


Figure 4.3.1: $f(t) = \cos(2\pi t)$ Figure 4.3.2: The Haar transform

When we apply the Haar wavelet transform we obtain a graph that looks like Figure 4.3.2.

As can be seen, the first half of the Figure 4.3.2 maintains the shape of the original function, while the second half of the graph is the difference. From this information, the original image can be reconstructed. The transformed data also provides us the ability to send the same data but with less entropy required.

It should be noted that this wavelet transform is built for a discrete case. In Section 4.5 the idea of the continuous case will be discussed.

Section 4.4 Daubechies Wavelet Transformations

Daubechies Wavelet Transformations are named for the groundbreaking mathematician, Ingrid Daubechies, who in her 1988 paper [5] discussed a special family of orthogonal lowpass filters. Daubechies filters are typically used in applications. In our work, we used a Daubechies filter of length 6 (D6). For the sake of simplifying the computation, we will explain the process of building a Daubechies filter of length 4 (D4). The D6 filter is obtained in a similar manner. The method presented here will follow the method in [6].

Let $\boldsymbol{h} = (h_0, h_1, h_2, h_3)$ be a lowpass filter and $\boldsymbol{g} = (g_0, g_1, g_2, g_3)$ be a highpass filter. Similar to the design of matrix 4.2.1, $\boldsymbol{h}$ and $\boldsymbol{g}$ can form a square matrix W_N where N is even. An example of W_8 using $\boldsymbol{h}$ and $\boldsymbol{g}$ is

$$W_8 = \begin{bmatrix} h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & h_3 & h_2 \\ g_3 & g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_3 & g_2 & g_1 & g_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_3 & g_2 & g_1 & g_0 \\ g_1 & g_0 & 0 & 0 & 0 & 0 & g_3 & g_2 \end{bmatrix}.$$

In order to simplify things, we want to replace the terms in $\boldsymbol{g}$ with terms of $\boldsymbol{h}$ by using the expression $g_k = (-1)^k h_{3-k}$. The above matrix can be written as

$$W_8 = \begin{bmatrix} h_3 & h_2 & h_1 & h_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_3 & h_2 & h_1 & h_0 \\ h_1 & h_0 & 0 & 0 & 0 & 0 & h_3 & h_2 \\ -h_0 & h_1 & -h_2 & h_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & -h_0 & h_1 & -h_2 & h_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -h_0 & h_1 & -h_2 & h_3 \\ -h_2 & h_3 & 0 & 0 & 0 & 0 & -h_0 & h_1 \end{bmatrix}.$$

We would like W_8 to be an orthogonal matrix. That is

$$W_8 W_8^T = I_8.$$

Thus

$$HH^T = I_4,$$

which implies that

$$h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1, \tag{4.4.1}$$

$$h_0 h_2 + h_1 h_3 = 0. \tag{4.4.2}$$

We can also use the condition that $\boldsymbol{h}$ is a lowpass filter to give us more parameters. We can use equation (4.1.2) for these parameters. However, in order to maintain orthogonality, we need to adjust the equations. Thus in order to maintain orthogonality we need to adjust (4.1.2). Specifically we need to set $H(0)$ equal to $\sqrt{2}$. To show this, we start by squaring the sum $h_0 + \cdots + h_3$

$$\left(\sum_{k=0}^{3} h_k\right)^2 = h_0^2 + h_1^2 + h_2^2 + h_3^2 + 2(h_0h_1 + h_0h_2 + h_0h_3 + h_1h_2 + h_1h_3 + h_2h_3).$$

However, from equation (4.4.1) we see that

$$\left(\sum_{k=0}^{3} h_k\right)^2 = 1 + 2(h_0h_1 + h_0h_2 + h_0h_3 + h_1h_2 + h_1h_3 + h_2h_3).$$

We can also use equation (4.4.2) to see that $h_0h_2 + h_1h_3 = 0$. Hence

$$\left(\sum_{k=0}^{3} h_k\right)^2 = 1 + 2(h_0h_1 + h_0h_3 + h_1h_2 + h_2h_3). \tag{4.4.3}$$

We also know for equation (4.1.4) that

$$h_0 - h_1 + h_2 - h_3 = 0. \tag{4.4.4}$$

Squaring (4.4.4) gives

$$h_0^2 + h_1^2 + h_2^2 + h_3^2 - 2(h_0h_1 - h_0h_2 + h_0h_3 + h_1h_2 - h_1h_3 + h_2h_3) = 0. \tag{4.4.5}$$

Using equations (4.4.1) and (4.4.2) again, we see that

$$1 - 2(h_0h_1 + h_0h_3 + h_1h_2 + h_2h_3) = 0. \tag{4.4.6}$$

If we add equation (4.4.3) and equation (4.4.6) we get

$$\left(\sum_{k=0}^{3} h_k\right)^2 = 2,$$

which implies that

$$\sum_{k=0}^{3} h_k = \sqrt{2}.$$

Using the fact that $h_0h_2 + h_1h_3 = 0$, we can say that the vectors $[h_0, h_1]^T$ and $[h_2, h_3]^T$ are orthogonal and thus

$$[h_2, h_3]^T = c[-h_1, h_0]^T. \tag{4.4.7}$$

where c is an arbitrary nonzero real number. (4.4.7) gives us $h_2 = -ch_1$ and $h_3 = ch_0$. If we use $h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1$ and $h_2 = -ch_1$ and $h_3 = ch_0$, we see that

$$h_0^2 + h_1^2 = \frac{1}{1+c^2}. \tag{4.4.8}$$

Also, recall

$$h_0 - h_1 + h_2 - h_3 = 0.$$

By substituting h_2 and h_3 we get

$$h_0 - h_1 - ch_1 - ch_0 = 0.$$

Combining like terms we see that

$$h_0(1-c) - h_1(1+c) = 0.$$

Solving down for h_1 we get that

$$h_1 = \left(\frac{1-c}{1+c}\right)h_0, \tag{4.4.9}$$

where $c \neq -1$. Combining (4.4.9) with (4.4.8) we get

$$h_0^2 + \left(\frac{1-c}{1+c}\right)^2 h_0^2 = \frac{1}{1+c^2}.$$

By combining like terms we see that

$$h_0^2\left(1 + \left(\frac{1-c}{1+c}\right)^2\right) = \frac{1}{1+c^2}.$$

In simplifying we get

$$2h_0^2\left(\frac{1+c^2}{(1+c)^2}\right) = \frac{1}{1+c^2}.$$

In solving for h_0 in terms of the constant c we get

$$h_0 = \pm\frac{\sqrt{2}(1+c)}{2(1+c^2)}. \tag{4.4.10}$$

Note that we have two different values for h_0. At this point we are able to arbitrarily pick either the positive or negative root. For now, we will choose the positive root. Using equation (4.4.9) we find that

$$h_1 = \left(\frac{1-c}{1+c}\right)\left(\frac{\sqrt{2}(1+c)}{2(1+c^2)}\right).$$

By combining like terms we find that

$$h_1 = \frac{\sqrt{2}(1-c)}{2(1+c^2)}. \tag{4.4.11}$$

Using equation (4.4.7) we can find h_2 and h_3:

$$h_2 = -\frac{c\sqrt{2}(1-c)}{2(1+c^2)}. \tag{4.4.12}$$

$$h_3 = \frac{c\sqrt{2}(1+c)}{2(1+c^2)}. \tag{4.4.13}$$

Before we can finish finding the values of the filter $\boldsymbol{h}$, we need to add an additional smoothness condition. We ask that

$$H'(\pi) = 0. \tag{4.4.14}$$

Differentiating

$$H(\omega) = h_0 + h_1e^{i\omega} + h_2e^{2i\omega} + h_3e^{3i\omega}$$

gives

$$H'(\omega) = ih_1e^{i\omega} + 2ih_2e^{2i\omega} + 3ih_3e^{3i\omega}.$$

Hence $H'(\pi) = 0$ gives us

$$-ih_1 + 2ih_2 - 3ih_3 = -i(h_1 - 2h_2 + 3h_3) = 0,$$

which can be simplified to

$$h_1 - 2h_2 + 3h_3 = 0. \qquad (4.4.15)$$

Equations (4.4.15) and (4.4.7) give

$$h_1 - 2h_2 + 3h_3 = h_1 + 2ch_1 + 3ch_0 = (1 + 2c)h_1 + 3ch_0 = 0.$$

Hence

$$h_1 = \frac{-3c}{1 + 2c} h_0.$$

Comparing with equation (4.4.7) we can see that

$$\frac{1 - c}{1 + c} = \frac{-3c}{1 + 2c}.$$

Now we can finally solve for *c*:

$$c = -2 \pm \sqrt{3}. \qquad (4.4.16)$$

Once again, we get to select which value of *c* we use. If we use the value $c = -2 + \sqrt{3}$, we can see that

$$h_1 = \left(\frac{1 - c}{1 + c}\right) h_0.$$

Hence

$$h_1 = \left(\frac{3 - \sqrt{3}}{-1 + \sqrt{3}}\right) h_0 = \sqrt{3}\mathrm{h}_0,$$

which gives

$$h_0^2 + h_1^2 = h_0^2 + 3h_0^2 = 4h_0^2.$$

Now (4.4.8) and (4.4.16) give us

$$4h_0^2 = \frac{2 + \sqrt{3}}{4},$$

$$h_0 = \pm \frac{1 + \sqrt{3}}{4\sqrt{2}}. \qquad (4.4.17)$$

We will assume the positive value of h_0. Using this, we can find the remaining values in the filter $\boldsymbol{h}$:

$$h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, \qquad h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}, \qquad h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}. \tag{4.4.18}$$

Thus our filter $\boldsymbol{h}$ is

$$\boldsymbol{h} = \left(\frac{1+\sqrt{3}}{4\sqrt{2}}, \frac{3+\sqrt{3}}{4\sqrt{2}}, \frac{3-\sqrt{3}}{4\sqrt{2}}, \frac{1-\sqrt{3}}{4\sqrt{2}}\right). \tag{4.4.19}$$

This set is known as the Daubechies four-term orthogonal filter. This means that our matrix is

$$\frac{1}{4\sqrt{2}}\begin{bmatrix} A & B & C & D & 0 & 0 & 0 & 0 \\ 0 & 0 & A & B & C & D & 0 & 0 \\ 0 & 0 & 0 & 0 & A & B & C & D \\ C & D & 0 & 0 & 0 & 0 & A & B \\ -D & C & -B & A & 0 & 0 & 0 & 0 \\ 0 & 0 & -D & C & -B & A & 0 & 0 \\ 0 & 0 & 0 & 0 & -D & C & -B & A \\ -B & A & 0 & 0 & 0 & 0 & -D & C \end{bmatrix},$$

where

$$A = 1-\sqrt{3}, \qquad \mathrm{B} = 3-\sqrt{3}, \qquad \mathrm{C} = 3+\sqrt{3}, \qquad \mathrm{D} = 1+\sqrt{3}.$$

Using this same process, we can find any even-length filter that fits these criteria. However, the longer the filter becomes, the more difficult the solutions are to find. More discussion on filters and convolution can be found in [6].

Section 4.5 The Integral Wavelet Transform

We start by defining the continuous wavelet transform. The idea of this process is to transform a continuous function into a function of two continuous variables that is highly redundant. This helps us interpret time-frequency analysis. We define the continuous wavelet transform of a continuous, square-integrable function, $f(x)$, relative to a real-valued wavelet, $\psi(x)$, as

$$W_{\psi}(s,\tau) = \int_{-\infty}^{\infty} f(x)\,\psi_{s,\tau}(x)dx,$$

where

$$\psi_{s,\tau}(x) = \frac{1}{\sqrt{s}}\psi\left(\frac{x-\tau}{s}\right),$$

s is the scale parameter, and τ is the translation parameter.

We define the inverse continuous wavelet transform $f(x)$

$$f(x) = \frac{1}{C_{\psi}}\int_{0}^{\infty}\int_{-\infty}^{\infty} W_{\psi}(s,\tau)\,\frac{\psi_{s,\tau}(x)}{s^2}\,d\tau ds\,.$$

Here,

$$C_{\psi} = \int_{-\infty}^{\infty} \frac{|\Psi(\mu)|^2}{|\mu|}\,d\mu\,,$$

and $\Psi(\mu)$ is the Fourier transform of $\psi(x)$. These equations are reversible so long as $C_{\psi} < \infty$. Roughly speaking, $1/s$ provides the frequency bandwidth and τ provides information regarding spatial/temporal localization, which are useful in many applications. See [7] for more discussion on continuous wavelet transforms.

Section 4.6 Wavelet Packet Decomposition

The process of a Wavelet Packet Decomposition can be described as a coherent processing step. Here we will follow the approach in [9].

The Wavelet Packet Decomposition is a level-by-level transformation of a signal. As was discussed in Chapter 3, the transformation is from the time domain to the frequency domain. Let $\boldsymbol{h}$ and $\boldsymbol{g}$ be a lowpass and a highpass filter, respectively. Define $x = (x_1, \dots, x_N)$ as the original signal. If we let x_h denote the signal after being passed through a lowpass filter $\boldsymbol{h}$, and if we let x_g denote the signal after being passed through a highpass filter $\boldsymbol{g}$, then we can define x_h and x_g as

$$x_h(n) = \sum_{k=1}^{N} x_k h_{2n-k}\,, \qquad x_g(n) = \sum_{k=1}^{N} x_k g_{2n-k}.$$

We apply these filters recursively to the signal. At each step or level of the process we create what is known as a bin vector. Since the vectors are typically chosen to be powers of 2, we can continue this process until there is 1 element in each bin vector.

To see a visual representation of this, we will start by defining $\boldsymbol{x}$ as the signal $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$. Let $\boldsymbol{h}$ be the lowpass filter and let $\boldsymbol{g}$ be the highpass filter. Also define $x_h[n]$ as the nth component of the lowpass portion and $x_g[n]$ as the nth component of the highpass portion. We also define x_{hh} as h being applied to x_h, x_{gh} as g being applied to x_h, and so on. We continue this process as needed. Thus our signal now looks like Figure 4.6.1.

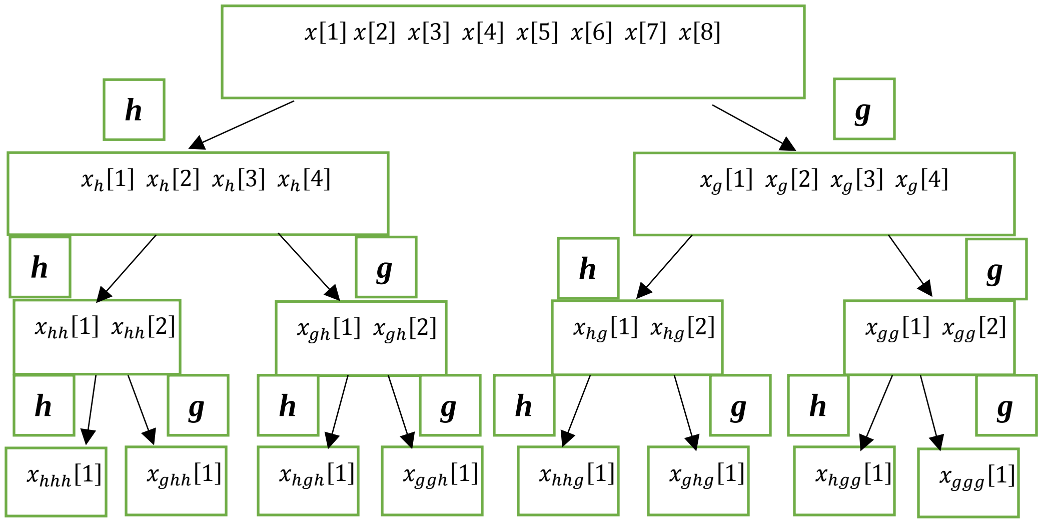


Figure 4.6.1: A representation of the WPD process

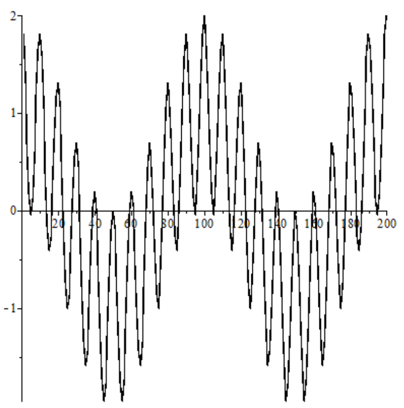
Section 4.7 Comparing the Fourier Transform and Wavelet Transform

The Fourier transform and the wavelet transform have similarities and differences.

To begin, if the signal being obtained is in the time domain, then it will be in the frequency domain after the Fourier transform has been applied. When we graph the transformed signal, there is an amplitude at each frequency. For example: if a signal was sent out that has a frequency of 100 Hz, then the graph would only show one spike at 100 Hz. It should also be noted that there will be a second amplitude, but only because of symmetry.

We have already mentioned that an inverse of the Fourier transform already exists. That is, we can have the raw data (in our example the time data) or the transformed data (the frequency data). However, it is limited in the fact that it cannot produce both at the same time. If someone wanted both time and frequency information, then a Fourier transform may not be the transform that he should use. In our example the Fourier transform would tell us what the frequency was but not when it happened. This is not a problem if the signal is stationary (that is, if the frequency exists at all times). However, this is not always the case.

For example: the signal $f(t) = \cos(2\pi t \cdot 2) + \cos(2\pi t \cdot 20)$, as seen in Figure 4.7.1, is a stationary signal and always has the frequencies of 2 Hz and 20 Hz at any given moment.



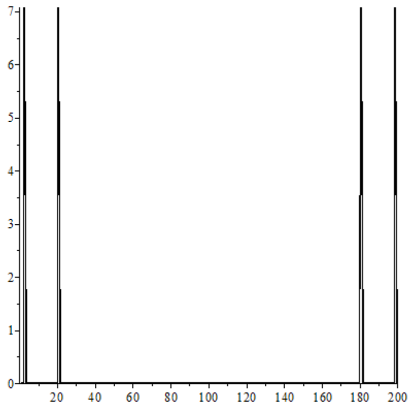


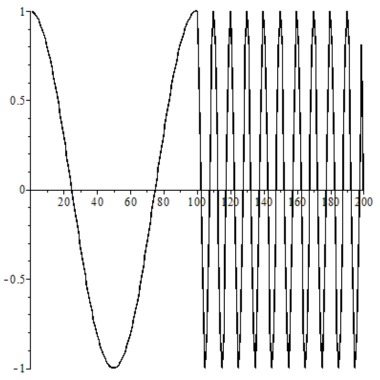
Figure 4.7.1: The signal $f(t)$ Figure 4.7.2: The Fourier transform of $f(t)$

If the Fourier transform of this were to be plotted, then there would be an amplitude at 2 and 20, respectively, on the frequency axis as seen in Figure 4.7.2.

Now consider the piecewise continuous signal

$$g(t) = \begin{cases} \cos(2\pi t \cdot 2), & 0 \le t < 1/2 \\ \cos(2\pi t \cdot 20), & 1/2 \le t < 1 \end{cases}$$

with a graph that can be seen in Figure 4.7.3.



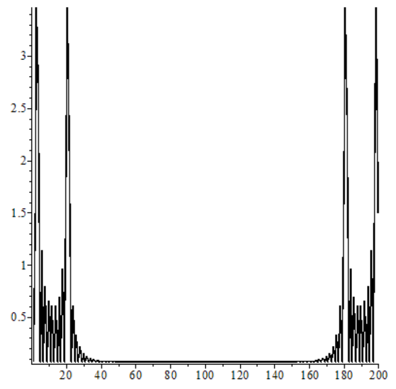


Figure 4.7.3: The signal $g(t)$ Figure 4.7.4: The transform of $g(t)$

If the Fourier transform were applied to $g(t)$, then its output would look very similar to the Fourier transform of $f(t)$, as can be seen in Figure 4.7.4. However, the frequency of 2 Hz only exists for the time $0 \le t < 1/2$ for $g(t)$, but it exists everywhere for $f(t)$.

Likewise, the frequency of 20 Hz only exists for $1/2 \leq t < 1$, but it exists everywhere for $f(t)$. Hence it would be impossible for the standard Fourier transform to decipher between the two different signals.

However, a wavelet transform is capable of this. The wavelet transformation provides its output in time and frequency simultaneously. It should be noted that the short-time Fourier transform can partially resemble this pattern. The graph will be in three dimensions with time on one axis, frequency on another, and amplitude on the last. Thus we can find the time, frequency, and the amplitude simultaneously.

However, the wavelet transform also has a benefit over the short-time Fourier transform. The short-time Fourier transform has a fixed resolution, while the wavelet transform has a variable resolution, which is why the wavelet transform is a good tool for multi-resolution analysis.

Thus we can see that in signal processing, the wavelet transform has a certain advantage over not only the Fourier transform but also the short-time Fourier transform. In cases where time and frequency are needed, such as the classification of sound, the wavelet transform will be more beneficial. [14] has further discussion and more examples on this topic.

Chapter 5 Machine Learning and Data Mining

In this Chapter we will be discussing data mining and machine learning. We will introduce the basic concepts in the first section. Then we will explain the input and output of a learning scheme [17]. In Section 5.4 we will look at decision trees and Random Forest [1]. Then we will discuss combination methods and how to validate the data [17].

Section 5.1 Introduction to Data Mining and Machine Learning

Every day we try to make informed decisions based off available information. For example: while at lunch we might choose our meal based off price, how it tastes, how healthy it is, and how filling it is. We use this information to make an informed decision. The more information that we have, the better decision we might be able to make.

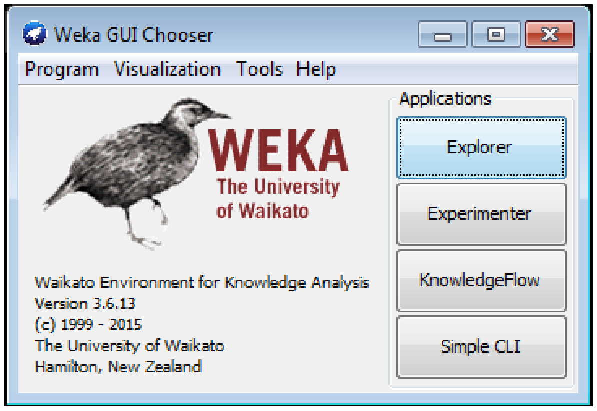
In data mining this process is performed by a computer. We define data mining as the process of discovering patterns in data. However, this process must be either automatic or semi-automatic. We want the findings to be meaningful and to lead to a decision. With the proper amount of data, we can make non-trivial predictions on a new set of data.

We see that data mining involves having our computer learn and make informed decisions. We define machine learning as a machine changing its behavior in a way that makes it perform better at a given task in the future.

To start, we let the input be a set of data and the output will be a set of rules. In this study, the input is a set of attributes attributed to bird calls. In the beginning of the study we had four attributes that we used: the species of the bird; the maximum energy (or “the largest average energy value” [16, p 3]); the position (or “the number of the bin *r*, in which the maximum energy was located” [16, p 3]); and the spread (or the “sum of the average energies of those coefficients whose energy exceeded [a] threshold value” [16, p 3]). Excluding the species of the bird, the other attributes are obtained using wavelet transforms. However, after collecting data, we found that the position was fairly consistent throughout almost all of the bird calls tested. Thus we decided that using the position (as defined earlier) was unnecessary and we only used the maximum energy, the species, and the spread. We use the maximum energy and the spread to determine the species of the bird.

The machine learning software used in our study is known as Waikato Environment for Knowledge Analysis (WEKA) [8]. (The name WEKA is interesting on two levels because it is not only an acronym, it is the name of a flightless bird (*Gallirallus australis*) that is found in New Zealand, the same country that Waikato is found.) WEKA has many applications, but for our purposes we only used it as a learning machine. To use WEKA, we stored all of the data in Excel files that were saved in CSV format.

When running WEKA, a graphical user interface (GUI) appears and presents the user with four different options: Explorer; Experimenter; KnowledgeFlow; and Simple CLI. For our studies we only utilized the Explorer. Figure 5.1.1 shows the GUI of the software.



Waikato Environment for Knowledge Analysis
Version 3.6.13
(c) 1999 - 2015
The University of Waikato
Hamilton, New Zealand

Figure 5.1.1: WEKA Interface

Section 5.2 Input

Under the Explorer is where we can input our data. In data mining, there are a few different kinds of input that are possible. The basic forms that the input can be is in the form of concepts, instances, and attributes. We prepared our input as instances.

Each instance in our study is a bird call and its associated attributes. Every instance in the data set must have attributes; otherwise, a relationship cannot be discovered. Attributes are characterizations of instances. For example, let a data set contain people and the kind of coffee they preferred out of black coffee, cappuccino, Americano, and decaffeinated coffee. Each person and his/her preferred kind of coffee is an instance in the study. Sometimes in data mining, the attributes may not apply to all instances. If one person did not drink coffee, then he/she would not have a preferred kind of coffee. Hence, the subject would not be able to correctly select between black coffee, cappuccino, Americano, and decaffeinated coffee. In the coffee example we use what is known as a nominal value, or a value that represents a category. In this same scenario we could ask people to rate three different kinds of coffee on a scale of 1 to 3, where 1 is the best, 2 is the second best, and 3 is the one least preferred. Then what we have will be ordinal data. For learning machines, ordinal and nominal data is typically used.

We used these instances to try to learn a concept. To learn the concept, data mining machines have four different styles of learning that it can utilize: classification learning, association learning, clustering, and numeric prediction. Classification learning is a learning scheme that takes a set of classified examples, and it uses this set to classify examples it has not seen. Association learning tries to find any kind of association that exists, and not only the associations that help make predictions for a particular class value. Clustering tries to group instances that belong together. Numeric predictions give us a numeric quantity as opposed to a discrete class. In our study we utilized several classifiers. More on the process and results of our study will be discussed in Chapter 6.

Classification learning is used primarily when there are distinct different options. For instance: if someone wanted to buy a car and their options were a truck, an SUV, and a sedan, then he could use classification learning to help him make his decision.

We also call classification learning supervised learning. This is because classification learning is given a training set to learn from and the training set already has the correct classifications. For the car example, the training set may have information such as “if the person had children, then he did not purchase a truck.” Here the outcome of the learning algorithm is the class that each instance belongs to. Thus we can see that classification learning has specified classes.

When there are not specified classes, association learning can be used. A main difference between association learning and classification learning is that association learning can predict any attribute, while classification learning is only used to predict the class. Association learning can also predict more than one attribute at a time. However, a drawback is that association learning requires more rules than classification learning.

Another learning method that works well when there is no specified class is clustering. Clustering tries to group instances that fall naturally together. For example: say everyone in a city was asked what their favorite color is, what their favorite music genre is, and what their favorite kind of food is. We can then cluster people together with similar interest. This would be an example of clustering. However, an issue that arises is that one person may seem to belong to more than one cluster.

Numeric prediction is considered to be a variant of classification learning. The only difference is that for numeric prediction the outcome is a numeric value, whereas with classification learning the outcome is any kind of category. When the task is to predict numeric values, the classification becomes regression and the learned model is a regression model. If an economist were to use data to decide when to buy and sell stock, then that would be considered classification learning. If an economist were to predict the value of a certain stock, then that would be considered numeric prediction.

Section 5.3 Output

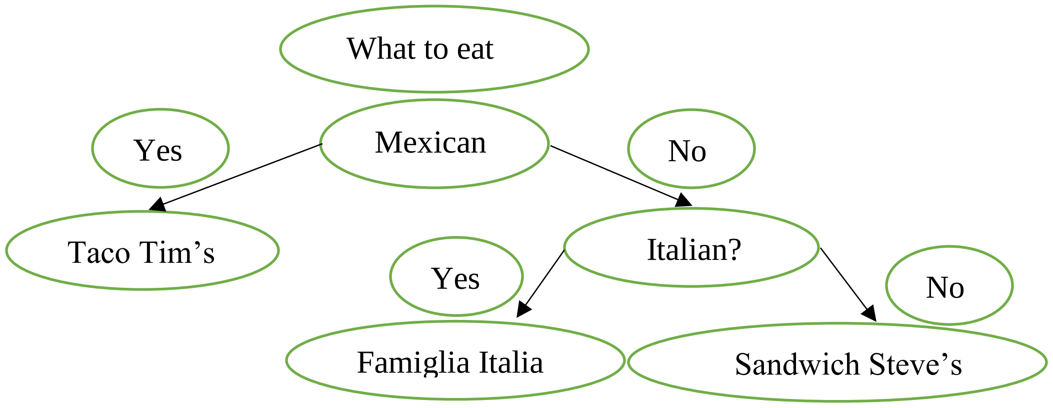
As for the output, WEKA can provide decision tables, decision trees, classification rules, association rules, trees for numeric prediction, and clusters.

Decision tables are the most rudimentary way of representing the output. A decision table may look something like

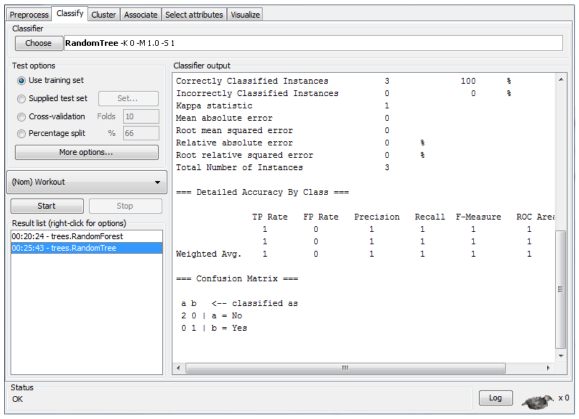
Tired?	Other obligations?	Workout
Yes	Yes	No
Yes	No	Yes
No	Yes	No
No	No	Yes

In this example, the decision table helps the user decide if he/she should workout or not.

A decision tree typically involves solutions that have a fixed number of possibilities. Someone may use a decision tree to decide where he/she wants to eat. For example, if a town has the three restaurants, Taco Tim's, Famiglia Italia, and Sandwich Steve's, then someone could use the following decision tree:



We can use WEKA to create a decision tree for us. Consider the gym example. By inputting the table into an Excel file, we can upload the file to WEKA and produce a decision tree. WEKA provides some base rules like in Figure 5.3.1. By right-clicking on the trees.RandomTree selection on the far left, we have the option to create a decision tree out of the given data. To see the decision tree for the gym example, refer to Figure 5.3.2. As we can see, the decision depended solely on whether or not there were other obligations. Decision trees will be further discussed in Section 5.4.



Classifier: RandomTree -K 0 -M 1.0 -S 1

Test options: (● Use training set) Supplied test set; Cross-validation Folds 10; Percentage split % 66

(Nom) Workout

Result list (right-click for options)
00:20:24 - trees.RandomForest
00:25:43 - trees.RandomTree

Classifier output

```
Correctly Classified Instances           3              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances                3

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area
                 1         0          1          1         1          1
                 1         0          1          1         1          1
Weighted Avg.    1         0          1          1         1          1

=== Confusion Matrix ===

 a b   <-- classified as
 2 0 | a = No
 0 1 | b = Yes
```

Status
OK

Figure 5.3.1: The WEKA Classification Interface

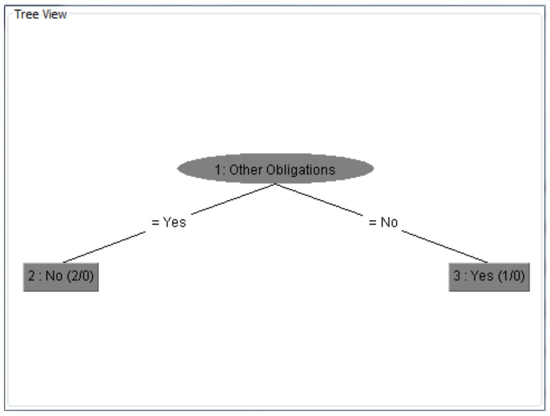


Figure 5.3.2: The Decision Tree Designed by WEKA

The output for classification learning is very similar to a decision tree. The main difference is that classification learning uses rules. Generally, classification learning can be expressed as a decision tree, but it typically has many more steps involved. For example, let *a* go to *x*, *b* go to *y*, and *c*, *d*, *e*, *f*, and *g* go to *z*. If we made a decision tree, it would be considerably longer than creating rules. The only rules we need to make are "if *a*, then *x*," "if *b*, then *y*," and "if *c*, *d*, *e*, *f*, or *g*, then *z*." Here we only had three rules to deal with. This will save computing time. Another reason why rules are more popular than building a decision tree is that rules can be added anywhere in the process. If we were to add a step to a tree, we may have to remake the entire tree. The only time that a rule cannot be added at any arbitrary point in the process is when that rule depends on the outcome of other rules.

A possibility that arises out of using classification learning is when different rules lead to different conclusions of the same instance. There is also the possibility of the instance not being classified at all. These issues cannot arise when using a decision tree. A possible way around these issues is to make the classes Boolean, or only having two possible outcomes. This is not always possible in a real world situation.

For numeric predictions we can use the same rules and trees that we have already discussed. If a tree is used for numeric predictions, then it is called a regression tree. However, since the numeric predictions involve numbers, we often use regression equalities to make predictions. It is also possible to combine the regression equations and the regression trees.

Another possible way to classify data is by using instance-based representation. That is, we are trying to group instances with similar attributes into the same class. This process is known as clustering. This method avoids using rules. The new instance is compared to the other classified instances and, using a distance metric, classifies it with the instance that is closest to it. This method is called the nearest neighbor method. This method works very well with numeric data; however, with nominal data it has its downfalls. For instance, if our set had different kinds of fruits, it would be hard to determine if an orange was closer to a cantaloupe or a star fruit without converting the data to some kind of numerical data. Another issue is determining weights for the distance. That is, determining which attribute is considered more important using this method.

When using clustering over a classifier, a diagram is used to show how the classes are divided. In some cases, a single instance can belong to more than one class, depending on the clustering method used. A common example of this would be a Venn diagram. In some clustering techniques, a probability that a certain instance is in a particular class is given instead. In some cases clusters are presented in a hierarchal structure, where instances in the higher levels are loosely related, while the instances in the lower levels are more closely related. These are typically referred to as dendrograms.

Section 5.4 Trees and Random Forests

A decision tree contains a set of decisions. These decisions split the results into different nodes, depending on the decision. This process is continued until a final decision is reached. The data can lead to one of two options: a leaf node or a non-leaf node. The non-leaf nodes are associated with a feature test, also known as a split. From here the data will split into different subsets based off the values of their feature test. When the data falls on a leaf node, then it is associated with a label. This label will be given to all instances that arrive at this node. The first node that the user comes to is known as the root node.

Consider the dinner example from Section 5.3. The nodes “Mexican?” and “Italian?” are considered to be non-leaf nodes. The remaining nodes, that is, “Taco Tim’s,” “Famiglia Italia,” and “Sandwich Steve’s”, are considered to be leaf nodes. Here the node “Mexican?” also happens to be the root node. In Figure 5.3.2 “Other Obligations” is considered a non-leaf node while “No” and “Yes” are considered leaf nodes. “Other Obligations” is considered to be the root node, as well.

We say that decision tree learning is a recursive process. At each step the data set is divided into subsets, depending on the node. The most important part of the decision tree is how the splits are decided. By having a properly devised decision tree, we can reduce the entropy of the set where the entropy is defined as

$$Ent(D) = -\sum_{y \in \Upsilon} P(y|D) \log P(y|D)$$

where D is the training set and Υ is the set of all predicted class labels y [19, p 5]. Entropy is a typical characteristic of how much information is in the signal. We also say that by reducing the entropy, we increase the information gain where the gain is defined as

$$G(D; D_1, \dots, D_k) = Ent(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} Ent(D_i),$$

where the training set D is divided into k subsets and $|\cdot|$ denotes the size of the data set [19, p 5]. Thus we want to create a split that reduces the entropy in order to increase the information gain. See J.R. Quinlan [15] for a specific example.

Some other algorithms use gain ratio [19, p 5] which is defined as

$$P(D; D_1, \dots, D_k) = G(D; D_1, \dots, D_k) \cdot \left(-\sum_{i=1}^{k} \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|} \right)^{-1}$$

Notice that this is variant of the information gain criterion. We select the split that has the highest gain ratio.

Another decision tree algorithm that is often used is called CART. This algorithm uses the Gini index [19, p 6] and is

$$G_{gini}(D; D_1, \dots, D_k) = I(D) - \sum_{i=1}^{k} \frac{|D_k|}{|D|} I(D_k)\,,$$

where $I(D)$ is defined [19, p 6] as

$$I(D) = 1 - \sum_{y \in \Upsilon} P(y|D)^2$$

There are cases where there are outliers in the training set. These outliers may cause the training set to find a “truth” for the set that does not exist. This is known as overfitting. To avoid this, we use the idea of pruning, or removing branches of the tree that are caused by outliers or noise in the data set. WEKA can prune the tree beforehand, known as pre-pruning, or we can prune the tree afterwards, known as post-pruning. WEKA can do this when we have a data set whose outcome is known.

The process of forming decision trees can help us further understand and utilize the different methods to ensemble data.

WEKA can increase the accuracy by creating a group of trees, known as forests, and allowing them to vote for the most popular class. We call using various single models to get a combined decision an ensemble method. We will introduce it in the next two sections.

Section 5.5 Combinations, Bagging, and Boosting

Bagging and boosting are different combination methods that can help you reach a decision based off the available information. They use multiple learners to make a decision. Bagging gives each data set equal weight, where boosting gives different data sets different weights.

We will begin with discussing bagging. We will introduce it with an example. Consider the decision tree in Section 5.3 that was used to decide what restaurant to go to. Also imagine that we have 9 other decision trees that are built to help us decide what restaurant to go to. Let each person who is deciding where he wants to eat be an instance. Allow each decision tree to decide what restaurant each person wants to go to. The restaurant that receives more votes than any other is the restaurant that is chosen. It is generally accepted that the more decision trees that are used, the more accurate the outcome will be, although there do exist theoretical conditions in which adding more decision trees does not increase the accuracy of the outcome. What has been described here is considered combining different decision trees.

In a perfect world, we could use an infinite number of training sets that are the same size to create an infinite number of classifiers. However, we begin to have issues with this when we put this into practice. That is, we cannot have an infinite number of test sets, nor can we have an infinite number of classifiers.

This is where bagging is useful (the term bagging is derived from bootstrap aggregating). Bagging takes a given training set and creates many training sets by deleting various instances and copying others (to keep the training sets the same size). We then apply the learning scheme to the given training set, and the classifier that is chosen votes for the class. Returning to our “which restaurant” decision tree, assume that we have a total of 10 Italian restaurants from which to choose. By using the bagging method we will create multiple training sets by sampling a given training set multiple times, with replacement. Whichever Italian restaurant receives the most votes will then be the restaurant that will be chosen if they select “yes” when asked if they want Italian or not. That is, if there are 5 votes for Famiglia Italia and 1 vote for Steve's Pizzeria, then the choice at that node will be Famiglia Italia.

Bagging is beneficial because it does not need multiple training sets to come to a strong conclusion. Bagging only requires one training set, and it samples this training set multiple times. Bagging tends to produce a model that is better than the models that are produced with a single training set. Bagging also rarely, if ever, produces a model that is significantly worse than what a single training set produces. We can also apply bagging to numerical data, with one slight difference. When applying bagging to numerical data, we average the outcomes, as opposed to voting on the outcomes. A benefit of bagging is that it uses the instability of a learning scheme to help make a stronger learning scheme. If by adding or removing an instance we run the risk of drastically changing the outcome, then we say that the learning scheme has a high cost. Bagging is a perfect candidate for learning schemes that have a high cost, or that are cost-sensitive.

The boosting algorithm starts by taking all of the instances in the training data and giving each one equal weight. It then uses the learning scheme to create a classifier for the training set, where it then reweights each instances based off the classifier so that each instance that is correctly classified sees a decrease in weight, and the instances that are incorrectly classified see an increase in weight. To create a prediction, the boosting algorithm uses a weighted vote to combine the output of each classifier. Those classifiers that have a relatively low error are given a higher weight, while those with a relatively high error are given a lower weight. Therefore, boosting is a sequential method.

Boosting has some similarities and some differences with bagging. Both methods use voting (or averaging in the case of numerical data) to combine learning models, and they both only combine models that are of the same type. However, a major difference is that boosting is iterative. As we described earlier, bagging creates new, independent models each time. Boosting, on the other hand, creates new models based on the performance of the previous model. Another major difference between boosting and bagging is that bagging gives everything an equal weight, whereas boosting weights a model based on its performance.

Section 5.6 Voting and Averaging

In this section, we will give an introduction on how to combine single learners to obtain better predictions. The two combination methods that are discussed are the averaging method and the voting method.

Suppose we obtain some base learners $h_1, \dots, h_T$ from one training data set and suppose that the output for the learner h_i of the instance $\boldsymbol{x}$ is $h_i(\boldsymbol{x})$. For simple averaging we combined $h_1(x), h_2(x), \dots, h_T(x)$ by defining

$$H(\boldsymbol{x}) = \frac{1}{T}\sum_{i=1}^{T} h_i(\boldsymbol{x}).$$

Simple averaging is often used in application due to its simplicity.

Weighted averaging gives certain learners more importance when averaging. That is, it weighs certain learners. We say that the weighted average is

$$G(\boldsymbol{x}) = \sum_{i=1}^{T} w_i h_i\,(x),$$

where w_i is the weight, or importance, assigned to h_i. We also limit our weights by

$$w_i \geq 0, \qquad \sum_{i=1}^{T} w_i = 1.$$

We can see that simple averaging is just a special case of weighted averaging where all the weights are equal. In practice weighted averaging may not always perform better than simple averaging.

Voting uses base learners' votes to determine the final output. For voting, we are going to assume that we have a set of *T* individual classifiers $\{h_1, \dots, h_T\}$. What we are trying to do is to combine all of our individual classifiers into a set of possible class labels $\{c_1, \dots, c_l\}$, where there are *l* possible labels. We also assume that the outputs h_i are given as an *l*-dimensional label vector

$$V = \left[h_i^1(x), \dots, h_i^l(x)\right]^T.$$

Here each element h_i^j is the output of h_i for the class label c_j.

The most popular voting method is known as majority voting. For example, assume that there are 10 classifiers working to identify a given bird call using arbitrary into one of the following species: Whip-poor-will; Bobwhite; Common Raven; Barred Owl; and Eastern Kingbird. Assume that for the first call, call it call A, 2 classifiers classify it as a Whip-poor-will, 2 classify it as a Common Raven, none classify it as a Barred Owl, 6 classify it as an Eastern Kingbird, and none classify it as a Bobwhite. Since the Eastern Kingbird received more than 50% of the votes, then the call is classified as the Eastern Kingbird. However, if no class (or for this example, no species) gets more than 50% of the votes, then the bird will be classified as the rejection option. Assume the rejection option is the Common Raven. Assume that for the second call, call it call B, 2 classifiers classify it as a Whip-poor-will, 2 classify it as a Common Raven, 2 classify it as a Barred Owl, 4 classify it as an Eastern Kingbird, and none classify it as a

Bobwhite. Notice that none of the classes obtained 5 votes (or 50% of the votes). That means that call A will be classified as the rejection option, the Common Raven.

Another method of voting that is similar to majority voting is known as plurality voting. Plurality voting selects the class that gets the most votes, regardless if it obtains more than 50% of the votes or not. Consider the example given in the previous paragraph that had 10 classifiers that attempted to classify call B. Again assume that 2 classifiers voted for the Whip-poor-will, 2 classifiers voted for the Common Raven, 2 classifiers voted for the Barred Owl, 4 classifiers voted for the Eastern Kingbird, and none classified it as the Bobwhite. Since the Eastern Kingbird obtained more votes than any other class, then the bird call will be classified as an Eastern Kingbird.

Although the methods of averaging and voting are used the most often, there are other ensemble methods such as stacking or dynamic classifier selection [19].

Section 5.7 Credibility

In order to know the performance of a classifier, we can look at its error rate. We say that if the classifier predicts the correct class, then it is a success, and if it does not, then it is an error. We say that the error rate is the total number of errors divided by the total number of instances. The error rate of a training set is generally not a good indicator of future performance. This is due to the fact that the classifier is learned from the training data, thus leading the results to be optimistic.

The solution comes from testing the classifier on another, independent set called the test set. An important aspect of the test set is that it cannot be used in any way to create the classifier. This can be done by creating a learning scheme that has two separate stages: the first stage creates a structure while the second stage optimizes the parameters involved. Again, two different data sets may be needed in the two different stages. This creates the need for a total of three data sets in the process: a training set, a validation set, and a test set. The training set is used to create the classifiers, the validation set is used to optimize the classifiers, and the test set is used to find the error rate of the classifiers. These three sets must be pairwise disjoint.

If we have a large data set, there is no problem in creating three sets that are pairwise disjoint. In general, larger training sets provide better classifiers. However, there is a limit. If the training set becomes too large, then the returns begin to diminish. The larger issue comes about when there is not much data from which to pull sets. Since we need to use some data to test the data, we start by pulling out what we need for a test set, and we can leave the remainder for training and validating (if need be). The issue is that in order to determine a good error estimate, we need a fairly large testing set. However, the larger our testing set is, the smaller our training set (and validating set, if required) must be.

Another process for smaller sets is known as stratification. We consider this process when we have a smaller set with which to work. In general, we like to reserve one-third of the data for testing, while reserving the rest for the training set. In order to make sure that every class is represented in both the training set and the test set, we use a procedure known as stratification. Stratification is the process of performing a random sampling, but ensuring that each class is represented in the random sample. For example, if a population was 80% A and 20% B, then a sample would be stratified of the sample also had 80% A and 20% B.

Another way to ensure that every class is represented is by repeating the process numerous times with different random samples. Other options include making the training set the test set, and making the test set the training set.

The last method that will be discussed that is used for smaller samples is known as cross-validation. To begin cross-validation, we start by deciding a number of partitions into which the sample will be divided. For example, assume we decide to divide the sample into five partitions. We then use four partitions for training, and one partition for testing. We then select a different partition for testing and repeat the process. We do this so that every partition is used exactly once for testing. We find the error estimate of each partition, and we then average the five error estimates giving us an overall error estimate. This is known as fivefold cross-validation, and, if it is stratified, it is known as stratified fivefold cross-validation. The standard number of partitions used to find the error rate is ten, thus using tenfold cross-validation.

Tests have shown that ten partitions yields the best results and that stratification slightly improves the results. To improve upon these results, it is common practice to repeat the tenfold cross-validation ten times, finding the mean of the results. That is, performing ten tenfold cross-validations and then finding their average.

Another method that is used is known as the leave-one-out method. It is essentially an *n*-fold cross-validation, where one instance is removed and then the classifiers are determined. Then the classifier is tested on the removed instance. This is repeated for each instance in the sample set. The error estimate is determined by assigning a value of 0 or 1 for a failure or a success, respectively, for each run of the *n*-fold cross-validation. Then the 0s and 1s are averaged to find the overall error estimate.

This procedure is used for two primary reasons. The first of these reasons is that the largest amount of data possible is used to train the classifiers, thus presumably increasing the accuracy of the classifier. The second of these reasons is that the procedure is deterministic. That is, there is no random sampling involved in this procedure. Drawbacks to this method are that it is extremely taxing, computationally speaking. However, for small data sets, this is not a problem. Another drawback is that it cannot be stratified. Since the training set has $n - 1$ instances and the test set only has 1 instance, then it guarantees that the test set will not contain every possible classification.

The last method discussed here is the bootstrap method. The bootstrap method involves sampling the data set with replacement to make a training set. We sample a data set with n instances n times. We then place these samples into a set, call it T. This makes the training set have n instances, possibly with some repeated elements. Let the original data set be O, then the test set is $E = T \setminus O$. We then repeat this numerous times and average the results to find the overall error estimate.

Chapter 6 Results, Conclusion, and Discussion

In Chapter 6 we will discuss the conclusions that we have reached based off our findings. In Section 6.1 we will discuss the methods of data representation, the Macaulay Library [10], and the program Audacity [12]. Section 6.2 discusses feature extraction using wavelets. Section 6.3 discusses how we classified species, while Section 6.4 discusses how we classified individuals of a given species. Section 6.5 discusses the overall results and conclusions of the study.

Section 6.1 Data Representation, Macaulay Library, and Audacity

In this section we refer to Audacity [12], which is a free audio editing software. We also refer to the Macaulay Library [10] that is run by Cornell University. To obtain recordings for our study, we began by searching the Macaulay Library that is run and maintained at the Cornell Lab of Ornithology in Ithaca, New York. This library is an online source that contains thousands of different bird calls that have been obtained over the past few decades. For our study, we considered five different species of birds: the Eastern Whip-poor-will (*Antrostomus vociferus*); the Northern Bobwhite (*Colinus virginianus*); the Barred Owl (*Strix varia*); the Eastern Kingbird (*Tyrannus tyrannus*); and the Common Raven (*Corvus corax*). The recordings of the Eastern Whip-poor-will were recorded by M. Robbins and M. Medler, the recordings of the Northern Bobwhite were recorded by M. Robbins, the recordings of the Barred Owl were recorded by N. Taylor, S. O'Brien, and B. McGuire, the recordings of the Eastern Kingbird were recorded by M. Robbins, and the recordings of the Common Raven were recorded by M. Andersen, G. Vyn, and B. McGuire (view Table 6.1.1 for the geographic information).

M. Robbins, M. Medler, and G. Vyn used the NAGRA ARES-BB+ recorder, M. Andersen used the SOUND DEVICES 744T, B. McGuire used the SOUND DEVICES 702, N. Taylor used the MARANTZ PMD 661 recorder, S. O'Brien used the FOSTEX FR-2 recorder, and M. Robbins also used the SONY TC-D5 PRO II recorder. In Table 6.1.1 the column labeled "Individuals" refers to the identification number given to the recording by the Macaulay Library. These calls were recorded in both stereo and mono, but they were all converted to mono so that Maple could handle the signals. All recordings were sent as a WAV file. That is, all files were digital signals. There were four birds of each species, giving us 20 birds in total. These birds were selected because each individual of each species was from the same general geographic area. This was to avoid possible regional dialects that the birds may or may not have. These species were selected so that each vocalization type was represented in the study. The bird recordings were of birds that were in the same general geographic location. This is to ensure that the vocalizations of each species were similar in nature. The Whip-poor-will recordings come from the Kansas/Missouri area, the Northern Bobwhite recordings come from Missouri, the Barred Owl recordings come from New York, the Eastern Kingbird recordings come from Missouri, and the Common Raven recordings come from Alaska. This study only separated birds on the species level, so subspecies were not accounted for when selecting the recordings. In total, we used 400 "syllables" from the bird recordings. (Here we use the term syllables to refer to a portion of a bird's call in a similar manner that we refer to syllables in human speech. For example: the Northern Bobwhite call would be divided into two syllables (the "Bob" and the "white"), the Barred Owl call would be one syllable (the "hoo"), etc.)

Bird Name	Individuals	Location	Recordings	Syllables
Whip-poor-will	146826	Missouri	1	21
	515817	Kansas	1	21
	145359	Missouri	1	21
	145378	Missouri	1	21
Northern Bobwhite	147549	Missouri	1	12
	516009	Missouri	1	20
	145412	Missouri	1	20
	146253	Missouri	1	20
Barred Owl	163907	New York	1	20
	173578	New York	1	20
	195738	New York	1	20
	188896	New York	1	20
Eastern Kingbird	144953	Missouri	1	21
	516001	Missouri	1	21
	145382	Missouri	1	21
	146239	Missouri	1	21
Common Raven	132161	Alaska	1	20
	207177	Alaska	1	20
	137574	Alaska	1	20
	132203	Alaska	1	20

Table 6.1.1: The geographic information where the recordings took place

Our goal is to find classifiers or clusters that can separate birds into species using only their calls. Our approach is summarized into three steps in Figure 6.1.1.

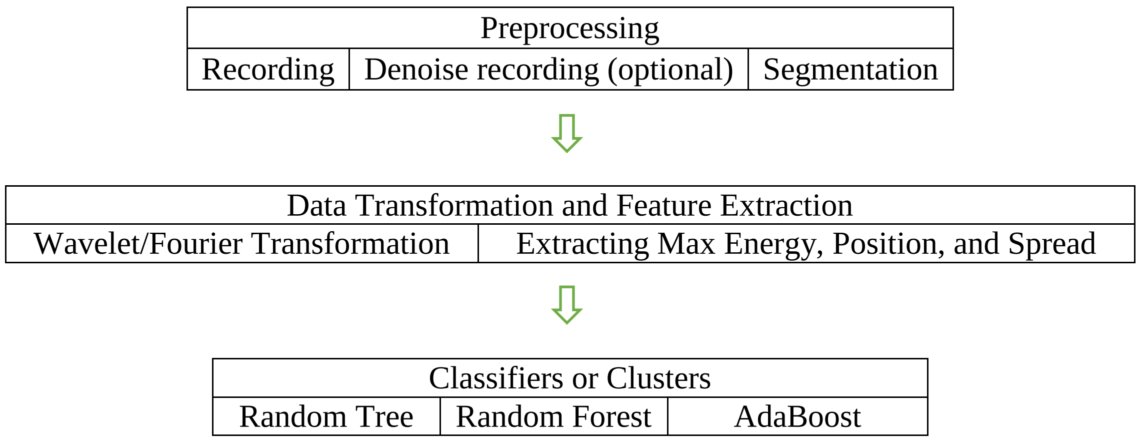

Preprocessing
Recording
Denoise recording (optional)
Segmentation
Data Transformation and Feature Extraction
Wavelet/Fourier Transformation
Extracting Max Energy, Position, and Spread
Classifiers or Clusters
Random Tree
Random Forest
AdaBoost

Figure 6.1.1: An overview of the process used

The preprocessing step involved recording the sounds, applying noise reduction, and the segmentation process. Since the Macaulay Library provided the recordings, it is unclear as to how the recordings were obtained. Note that if the recording and segmentation steps are done properly, then noise reduction may not be necessary. The recordings here were obtained via the Macaulay Library at Cornell University. We selected the audio recordings that contained minimum noise (i.e. the recordings where the bird that was being considered was the primary sound in the recording), which allow us to save work on noise reduction and focus on the second and third steps. These files were obtained in a WAV format to guarantee that as much information as possible was obtained. That is, the files were sent in a WAV format since the WAV format saves the most data out of the given data types.

We then imported the information into Audacity, a software that edits sounds. Using Audacity we were then able to segment each bird's calls into syllables. To begin, we imported the audio file into Audacity. We then were given the file in a sound wave (see Figure 6.1.2). Here we were presented with a number of options with which to work. With Audacity we were able to pause, play, stop, rewind, fast forward, and record new sounds. We highlight an area of the audio file that we want to work with and we cut and paste it into a new Audacity file. We then export the file with a new name.

When we started working with the recordings, it became apparent that some of the recordings were in mono, while the rest were in stereo. Using one of the functions of Audacity we were able to convert the stereo recordings into a mono recording. Not only did this help keep our data consistent, but the Maple [11] program that we used required that the sound files be in mono. Audacity allows users to select a part of the recording they need (see Figures 6.1.3 and 6.1.4). Figures 6.1.3 and 6.1.4 are from a Barred Owl. The *x* axis represents time, whereas the *y* axis represents pressure, or the volume. We use this feature to select the syllables, which are generally only a small part of the whole recording. We then saved the files. Another feature of Audacity allows us to decide at what frequency we would like to save our data. That is, we are able to decide how many pieces of information we obtain per second. We stored our data at the highest that Audacity allowed us, 44.1 kHz. That is, 44,100 pieces of information per second.

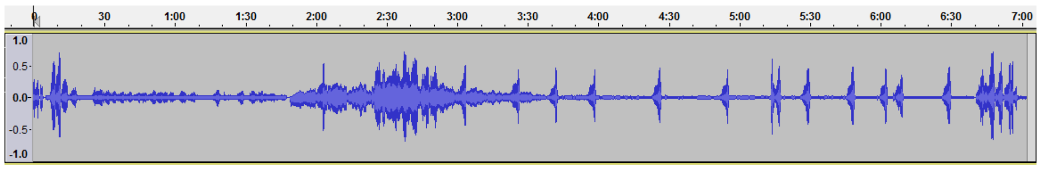


Figure 6.1.2: Roughly 7 minutes of data

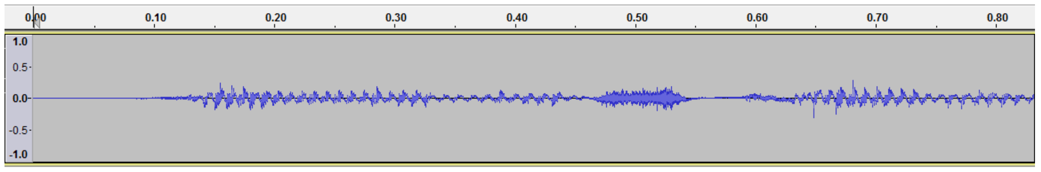


Figure 6.1.3: Roughly 0.80 seconds of data

We then saved each syllable individually. We then linked the location of the syllable into a Maple program. We used Maple for the data transformation and feature extraction. In our work, we used the Wavelet Packet Decomposition (WPD) algorithm. We first needed to make sure that our data set had the correct number of data points. Since we are going to use the 4th level of the WPD tree, each of our recordings need to have a number of sample points as a multiple of 16 (that is, 2^4). To do this we truncated the data to the nearest multiple of 16. Each audio file has thousands, if not tens of thousands of data points, so truncating at most 15 was not damaging to the study overall. During this step we also normalize our data. We use the Daubechies wavelet filter of length 6 (D6) (discussed in Chapter 5) to decompose the sound waves. Thus we extract the maximum energy, the position, and the spread. These will be further discussed in Section 6.2.

During the classifier step we consider Random Tree and Random Forest. See Section 5.4 for more details.

Section 6.2 Wavelet Decomposition and Feature Extractions

In this section we refer to Selin, Turunen, and Tanttu [16] to find what features are useful. The three features that we used were the maximum energy, the spread, and the position.

We begin by defining the bins. In Figure 6.2.1, we apply the wavelet transformation four times (refer to [9]). Each time we have a sum (S) vector and a difference (D) vector. In Figure 6.2.1 the bin on the bottom left will be referred to as bin 1, the bin directly to its right will be bin 2, and we continue this process until we reach the last bin, which in this example is bin 16.

Original															
S								D							
S				D				S				D			
S		D		S		D		S		D		S		D	
S	D	S	D	S	D	S	D	S	D	S	D	S	D	S	D

Figure 6.2.1: The bin example

The maximum energy depends on the bin energy. We define the bin energy with wavelet coefficients *c* of bin *r* as

$$E_B(r) = \sum_{n=1}^{n_c} c^2(n,r), \qquad r = 2, 3, \ldots, 16,$$

where n_c is the number of wavelet coefficients in each bin [16]. Note that we omit the bin $r = 1$. According to Selin, Turunen, and Tanttu [16], this bin contains the noise in the recording. Thus we do not use it in our study. We then define the average energy [16] of each bin *r* as

$$\widetilde{E_B}(r) = \frac{E_B(r)}{n_c}.$$

Hence the maximum energy [16] is defined as

$$E_M = \max_{2 \leq r \leq 16} \left(\widetilde{E_B}(r) \right).$$

This value finds the maximum energy produced by the sound.

Consider, for example, the vector $V = [1, 2, 3, \ldots, 64]$. After normalizing and running the WPD four times, we see that our first two bins are

$$bin_1 = [0.8277909461, 1.778373759, 3.184784592, 2.334050703]$$

$$bin_2 = [0.009657704743, 0.5267099362, -0.9221896581, 0.2652781042]$$

Note that these values have been rounded in order to fit. In all, we have 16 bins with varying values. After four iterations of the WPD, our n_c is 4. We only need to count the number of values in each bin to determine this number. The average bin energy of bins 1 and 2, respectively, are 4.85962416488393 and 0.299580716538431.

The next value found is known as the position. The position, *P*, refers the bin *r* in which the maximum energy was found [16]. According to our calculations, most instances have a maximum energy in the second bin. Thus we removed this value, given it would not provide any further insight into the problem.

Finally we calculated the spread. Let q be the number of the sample, *r* be the bin number, $T_{h1}(r) = \tilde{E}_B(r)/6$ be the threshold value, and *J* be the set of index pairs (q, r) such that $c^2(q, r) > T_{h1}(r)$. Also, let $\#J$ be the cardinality of the set *J*. Then we define the spread [16] to be

$$S = \frac{1}{\#J} \sum_{(q,r)\in J} c^2(q, r) \,.$$

Using the same example, we can calculate the spread to be 0.152948311471895.

We wrote a program using the Maple software in order to find the features mentioned above. After obtaining these features for each call of each bird, we then organized them into Excel files and saved them in the CSV format. In Table 6.2.1 we have an example of the Common Raven data. Each row is an instance, or a syllable. The first column is a name that is used solely to keep track of the individuals. The second column is the maximum energy. The third column is the position. The fourth column is the spread. The fifth column is the common name of the bird.

File Name	Maximum Energy	Position	Spread	Bird Name
Common Raven a 1	0.179094439	2	0.127335954	Common Raven
Common Raven a 2	0.187008524	2	0.142694617	Common Raven
Common Raven a 3	0.204325899	2	0.144938941	Common Raven
Common Raven a 4	0.157743768	2	0.098308357	Common Raven
Common Raven a 5	0.181794241	2	0.116743861	Common Raven
Common Raven a 6	0.120920788	2	0.074890981	Common Raven
Common Raven a 7	0.251314322	2	0.171162873	Common Raven
Common Raven a 8	0.255887332	2	0.14791002	Common Raven
Common Raven a 9	0.108557857	2	0.076480965	Common Raven
Common Raven a 10	0.24270076	2	0.134252454	Common Raven
Common Raven a 11	0.254711586	2	0.151094424	Common Raven
Common Raven a 12	0.130208515	2	0.108633567	Common Raven
Common Raven a 13	0.203880274	2	0.135682038	Common Raven
Common Raven a 14	0.212800543	2	0.132853468	Common Raven
Common Raven a 15	0.141669149	2	0.136983284	Common Raven

Table 6.2.1: Example of the Common Raven Data

Once the data had been collected from the audio files and put into spread sheets, we then were able to use machine learning software known as WEKA (discussed in Section 5.1) in order to determine classifiers and clusters.

Section 6.3 Classifying Species and Results with Random Forest Classifier

In this section and the next section we use the software WEKA (referenced in Chapter 5). Prior to running the test, we divided the spreadsheet into two different spreadsheets. In one spreadsheet we randomly put 133 instances (roughly one-third of the instances) to use for a training set. We placed the remaining 267 instances into another spreadsheet for testing purposes. We imported the training set in order to create a classifier, and then we imported the test set to see how proficiently the classifier could classify the data set.

The Random Forest classifier correctly classified 182 instances and incorrectly classified 85 instances. This means that the Random Forest classifier was correct roughly 68.15% of the time. Of the 48 Bobwhite syllables in the test set, 20 were classified as Bobwhites, 0 as Barred Owls, 5 as Common Ravens, 8 as Eastern Kingbirds, and 15 as Whip-poor-wills. Of the 53 Barred Owl syllables in the test set, 48 were classified as Barred Owls and 5 were classified as Whip-poor-wills. Of the 53 Common Raven syllables in the test set, 1 was classified as Barred Owl, 40 were classified as Common Ravens, 1 was classified as an Eastern Kingbird, and 11 were classified as Whip-poor-wills. Of the 56 Eastern Kingbird syllables in the test set, 5 were classified as Bobwhites, 0 were classified as Barred Owls, 2 were classified as Common Ravens, 38 were classified as Eastern Kingbirds, and 11 were classified as Whip-poor-wills. Of the 57 Whip-poor-will syllables in the test set, 4 were classified as Bobwhites, 8 were classified as Barred Owls, 3 were classified as Common Ravens, 6 were classified as Eastern Kingbirds, and 36 were classified as Whip-poor-wills. Figure 6.3.1 shows a confusion matrix that generalizes this data. The matrix presents the data such that the columns are what the birds were classified as and the rows refer to what the bird actually is. *a* refers to Bobwhites, *b* refers to Barred Owls, *c* refers to Common Ravens, *d* refers to Eastern Kingbirds, and *e* refers to Whip-poor-wills.

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 20 & 0 & 5 & 8 & 15 \\
b & 0 & 48 & 0 & 0 & 5 \\
c & 0 & 1 & 40 & 1 & 11 \\
d & 5 & 0 & 2 & 38 & 11 \\
e & 4 & 8 & 3 & 6 & 36
\end{array}$$

Figure 6.3.1: A confusion matrix for the Random Forest

The Random Tree classifier performed almost as well as the Random Forest classifier did. The Random Tree classifier correctly classified 173 instances and incorrectly classified 94 instances. That means the Random Tree classifier correctly identified one of the syllables roughly 64.79% of the time. That is, Having 48 Bobwhite syllables it classified 22 as Bobwhites, 0 as Barred Owls, 5 as Common Ravens, 9 as Eastern Kingbirds, and 12 as Whip-poor-wills. Out of 53 Barred Owl syllables it classified 0 as Bobwhites, Common Ravens, or Eastern Kingbirds, 46 as Barred Owls, and 7 as Whip-poor-wills. Of the 53 Common Raven syllables it classified 0 as Bobwhites or Barred Owls, 36 as Common Ravens, 3 as Eastern Kingbirds, and 14 as Whip-poor-wills. Out of the 56 Eastern Kingbird syllables 5 were classified as Bobwhites, 0 were classified as Common Ravens, 2 were classified as Barred Owls, 38 were classified as Eastern Kingbirds, and 11 were classified as Whip-poor-wills. Out of the 57 Whip-poor-will syllables, 11 were classified as Bobwhites, 5 were classified as Barred Owls, 4 were classified as Common Ravens, 6 were classified as Eastern Kingbirds, and 31 were classified as Whip-poor-wills. Figure 6.3.2 is a confusion matrix for the Random Tree classifier, similar to that in Figure 6.3.1. The columns are what the birds were classified as and the rows refer to what the bird actually is. The labels are the same as those in Figure 6.3.1.

$$
\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 22 & 0 & 5 & 9 & 12 \\
b & 0 & 46 & 0 & 0 & 7 \\
c & 0 & 0 & 36 & 3 & 14 \\
d & 5 & 0 & 2 & 38 & 11 \\
e & 11 & 5 & 4 & 6 & 31
\end{array}
$$

Figure 6.3.2: A confusion matrix for the Random Tree classifier

We also performed a cross-validation on the entire data set (refer to Section 5.7). With the cross-validation we ran both the Random Forest and the Random Tree classifiers. We tested these with 10 fold, 20 fold, and 40 fold. The percentage that was correctly classified by the cross-validation with Random Tree for the 10 fold, 20 fold, and 40 fold were 70.25% for the 10 fold and the 20 fold, and was 71.25% for the 40 fold. However, the confusion matrix for each was not the same. The confusion matrix for 10 fold, 20 fold, and 40 fold can be seen in Figures 6.3.3, 6.3.4, and 6.3.5, respectively. These Figures use the same labels as Figure 6.3.1 and 6.3.2.

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 40 & 0 & 7 & 10 & 15 \\
b & 0 & 73 & 2 & 0 & 5 \\
c & 5 & 0 & 62 & 3 & 10 \\
d & 5 & 0 & 4 & 64 & 11 \\
e & 19 & 6 & 8 & 9 & 42
\end{array}$$

Figure 6.3.3: Random Tree confusion matrix for 10-fold cross-validation

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 40 & 0 & 7 & 6 & 19 \\
b & 0 & 72 & 2 & 0 & 6 \\
c & 8 & 0 & 58 & 2 & 12 \\
d & 4 & 0 & 4 & 65 & 11 \\
e & 15 & 7 & 7 & 9 & 46
\end{array}$$

Figure 6.3.4: Random Tree confusion matrix for 20-fold cross-validation

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 40 & 0 & 7 & 7 & 18 \\
b & 0 & 72 & 2 & 0 & 6 \\
c & 6 & 0 & 60 & 3 & 11 \\
d & 4 & 0 & 4 & 65 & 11 \\
e & 14 & 5 & 8 & 9 & 48
\end{array}$$

Figure 6.3.5: Random Tree confusion matrix for 40-fold cross-validation

We also performed a cross-validation with Random Forest for 10 fold, 20 fold, and 40 fold. It should be noted that these computing times were notably longer than the computing times for any other method (these took anywhere from a few seconds to around 30 seconds, while the others were relatively instant). The percentage of instances that were correctly classified using cross-validation for the Random Forest classifier for 10 fold, 20 fold, and 40 fold were, respectively, 72.50%, 74.25%, and 73%. The confusion matrices can be seen in Figures 6.3.6, 6.3.7, and 6.3.8.

Notice that in these confusion matrices, there are relatively large values down the main diagonal, and relatively small numbers elsewhere. That means that the classifiers correctly classified the syllable more times than not. When the Bobwhite was misclassified, it was generally misclassified as a Whip-poor-will. Likewise, when the Whip-poor-will was misclassified, it was generally misclassified as a Bobwhite. These error makes sense, when the syllables of the calls are considered. That is, the “bob” of the Bobwhite’s call is quite similar to the “poor” of the Whip-poor-will’s call.

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 41 & 0 & 8 & 8 & 15 \\
b & 0 & 72 & 1 & 0 & 7 \\
c & 3 & 0 & 68 & 32 & 7 \\
d & 7 & 0 & 4 & 65 & 8 \\
e & 14 & 8 & 8 & 10 & 44
\end{array}$$

Figure 6.3.6: Random Forest confusion matrix for 10-fold cross-validation

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 41 & 0 & 8 & 6 & 17 \\
b & 0 & 73 & 0 & 0 & 7 \\
c & 2 & 0 & 67 & 3 & 8 \\
d & 5 & 0 & 4 & 66 & 9 \\
e & 11 & 7 & 7 & 9 & 50
\end{array}$$

Figure 6.3.7: Random Forest confusion matrix for 20-fold cross-validation

$$\begin{array}{cccccc}
 & a & b & c & d & e \\
a & 40 & 0 & 7 & 6 & 19 \\
b & 0 & 72 & 1 & 0 & 7 \\
c & 3 & 0 & 65 & 3 & 9 \\
d & 5 & 0 & 4 & 66 & 9 \\
e & 13 & 6 & 8 & 8 & 49
\end{array}$$

Figure 6.3.8: Random Forest confusion matrix, 40-fold cross-validation

Section 6.4 Classifying Individuals and Results with Random Forest Classifier

We also tested to see if using Random Tree, Random Forest, and cross-validation we could classify individuals of a given species (view Table 6.4.1), as opposed to in Section 6.3 where we classified the species. This time, instead of manually splitting the instances into training and testing sets, we used a function in WEKA known as percentage split. When using percentage split, the user decides what percentage will be used to train the classifier, and the remainder is used for testing. For classifying individuals, we will be using 67% of the data to train, and the remaining 33% to test. These percentages are used for every species. In general, we found that Random Forest performed as well, if not better than Random Tree, so only the results of the Random Forest classifier are discussed in here. The results for Random Tree, Random Forest, and percentage split are presented in Table 6.4.1.

Next we tested the Random Forest classifier on each species. For the Barred Owl, the classifier correctly classified 69.23% of the instances in the testing set, the same as the Random Tree classifier. For the Bobwhite, the classifier correctly classified 47.83% of the instances, the same as the Random Tree classifier. For the Common Raven, the classifier correctly classified 53.85% of the instances. For the Eastern Kingbird, the classifier correctly classified 51.85% of the instances in the training set. For the Whip-poor-will, the classifier correctly classified 51.85% of the instances.

We then tested cross-validation at 10-fold and 20-fold for Random Forest. For the Barred Owl, the Random Forest with cross-validation correctly classified 74.68% at 10-fold, and 74.68% at 20-fold. For the Bobwhite, 52.11% of the instances were correctly classified at 10-fold, and 46.48% of the instances were correctly classified at 20-fold. For the Common Raven, 56.96% of the instances were classified correctly at 10-fold, and 58.22% of the instances were classified correctly at 20-fold. For the Eastern Kingbird, the classifier correctly classified 61.45% of the instances at 10-fold, and 56.63% of the instances at 20-fold. For the Whip-poor-will, the classifier correctly classified 57.83% of the instances at 10-fold, and 54.22% of the instances at 20-fold.

	Validation	Barred Owl	Bobwhite	Common Raven	Eastern Kingbird	Whip-Poor-Will
Classifier Random Tree	10-fold	73.42%	42.25%	55.70%	54.22%	55.42%
	20-fold	73.42%	40.85%	55.70%	49.40%	53.01%
	Percentage Split	69.23%	47.83%	50%	48.15%	55.56%
Classifier Random Forest	10-fold	74.68%	52.11%	56.96%	61.45%	57.83%
	20-fold	74.68%	46.48%	58.23%	56.63%	54.22%
	Percentage Split	69.23%	47.83%	53.85%	51.85%	51.85%

Table 6.4.1: The percentage that each classifier classified correctly

Section 6.5 Discussions and Conclusions

Out of all the methods attempted to discern one species from another, the average amount of instances that were classified correctly was 70.24%, with a minimum of

64.79% and a maximum of 74.25%. Selin, Turunen, and Tanttu [16] found that the SOM network (self-organizing map, a classifying function) classified 78% of their data correctly, while the MLP (multilayer perceptron, a classifying function), classified 96% of the test sounds correctly. The main difference between using the SOM network or the MLP and our study is that our study focused on using trees, while Selin, Turunen, and Tanttu used functions. Selin, Turunen, and Tanttu used four features (the maximum energy, the position, the spread, and the width), and our study only used two features.

When considering classifying individuals, we had varied results. The results for the Barred Owl were relatively similar to our results for classifying species. However, the results for the remainder of the species was not as good. There are various explanations for this. To begin, some species may have several different types of calls that seem similar to us, but in fact are different calls. Since they seem similar, these different calls may have been incorrectly used instead of using the same calls. We know this issue is not an isolated incident, seeing as Selin, Turunen, and Tanttu encountered a similar issue with the Graylag goose.

Our method has a similar disadvantage to the method used by Selin, Turunen, and Tanttu. If we were to add a new species to our set, then we would need to retrain the classifiers to account for the new species.

Based off Selin, Turunen, and Tanttu's results and conclusions, we say that the proposed method of classification works well for classifying species. When compared to Selin, Turunen, and Tanttu, we see that reducing the number of features from four to two reduces the accuracy, but may save on the computational power needed to find and use those features. The results in the study are not perfectly accurate, but they are better than making a random selection.

This work can be applied to bird surveys, ecology, and conservation biology. Being able to tell what bird species are present in a given ecosystem can tell us about the extinction rates and the health of the ecosystem.

In the future, we could work to see if adding more parameters helped with the classification of species and individuals or not. We could also work to see if certain features work better for classifying species over individuals, or vice versa. Another future study could work to see if different filters or different decomposition levels help in the classification process. One final thing that future studies could find is a more effective way of utilizing clusters in these kinds of projects.

Bibliography

[1] Breiman, Leo. "Random forests." *Machine learning* 45.1 (2001): 5-32.

[2] Brown, J. W., & Churchill, R. V. (2009). *Complex Variables and Applications.* Boston: McGraw-Hill Higher Education.

[3] Charif, R. A. (2010). *Raven Pro 1.4 User's Manual.* Ithaca, NY: Cornell Lab of Ornithology.

[4] Cornell University. (n.d.). *How to ID Birds*. Retrieved from Cornell Lab of Ornithology: http://www.birds.cornell.edu/AllAboutBirds/birding123/identify/

[5] Daubechies, Ingrid. "Orthonormal bases of compactly supported wavelets." *Communications on pure and applied mathematics* 41.7 (1988): 909-996.

[6] Fleet, P. J. (2008). *Discrete Wavelet Transforms.* Hoboken, New Jersey: John Wiley & Sons.

[7] Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing.* Upper Saddle River, New Jersey: Pearson Prentice Hall.

[8] Hall, Mark, et al. "The WEKA data mining software: an update." *ACM SIGKDD explorations newsletter* 11.1 (2009): 10-18.

[9] Learned, R. (1992). *A Wavelet Packet Based Transient Signal Classification.* Cambridge, MA Master's Thesis: Massachusetts Institute of Technology.

[10] The Macaulay Library at the Cornell Lab of Ornithology, Ithaca, New York.

[11] Maple (2016). Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.

[12] Mazzoni, D., and R. Dannenberg. "Audacity [software]. Pittsburg." (2000).

[13] Oppenheim, A. V., & Schafer, R. W. (1975). *Digital Signal Processing.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

[14] Polikar, R. (2016, August 3). *A Wavelet Tutorial: Second Edition*. Retrieved from University of Nevada, Reno: https://www.cse.unr.edu/~bebis/CS474/Handouts/WaveletTutorial.pdf

[15] Quinlan, J. (1986). Induction of Decision Trees. *machine Learning*, 81-106.

[16] Selin, Arja, Jari Turunen, and Juha T. Tanttu. "Wavelets in recognition of bird sounds." *EURASIP Journal on Applied Signal Processing* 2007.1 (2007): 141-141.

[17] Witten, I. H., & Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* San Francisco, CA: Morgan Kaufmann Publishers.

[18] Yeoman, B. (2013, April 08). *What Do Birds Do For Us?* Retrieved from The National Audubon Society: http://www.audubon.org/news/what-do-birds-do-us

[19] Zhou, Zhi-Hua. *Ensemble methods: foundations and algorithms*. CRC press, 2012.