Introduction to Signals Processing and Machine Learning with a Focus on Biomedical Sciences

Joshua Pickard

Contents

Part 1

Signals Processing

LECTURE I

Signals

1. Signals as Information

The purpose of signals processing is to analyze data in order to gain insight into how a system operates. This analysis can be in them form or predicative analytics, modification, or synthesis of information from the system.

Definition 1. A signal is a function or ordered set of data that contains information about a process. The signal can be ordered in or a function of time, space, or other domains.

It is critical that a signal has order to it. Often, we are not concerned with the values of a signal as much as we are how the values are changing. For example, knowing the position of a car does not provide nearly as much information as the position of a car at specific times, which allows us to calculate velocity, acceleration, and possibly make other predictions on where the car is headed. Similarly, the price of a stock or financial security does not alone determine if it is a good investment, but understanding how it's price has changed over time gives more insight.

Types of Signals. Signals can be classified as based on if the amplitude of the data and the frequency the data is collected, relative to the order, is continuous or discrete.

Continuous Signals. A continuous signal, $X(t)$, collects data over a continuous amplitude at continuous points with respect to the order. In figure 1, the top left signal is body temperature taken on a Mercury thermometer over a period of time. At any time period in this interval, a temperature reading was taken and over the course of the measurements being taken, any temperature between the maximum and minimum values could have been read.

Discrete Signals. A discrete signal, $X(n)$, is similar to a continuous signal in that any amplitude of measurement can be recorded, but these measurements are not continuously taken. Instead, the measurements are taken at discrete time periods. For example, the top right signal in figure 1 is a discrete signal representing the same data as in the continuous signal it is next to, but the measurements are not recorded continuously.

Digital Signals. A digital signal, $X(n)$ or $X[n]$, contains measurements taken at discrete time periods and that can only contain discrete values. These are motivated by the fact that computers can only contain certain values depending on how many decimal places they can hold. For example, a digital thermometer may only have 1 decimal, in which case it could not record the value 99.12 and would store it as 99.1. The bottom signal in figure 1 is a digital signal representing the continuous and discrete signals above it.

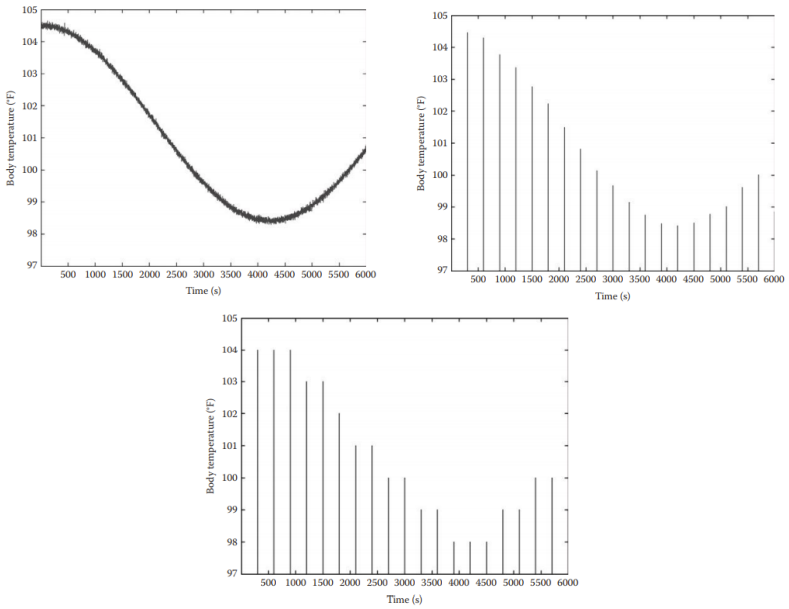

FIGURE 1. The top left: a continuous signal. The top right: a discrete signal. The bottom: a digital signal

In practice, however, modern computers can store digital signals with enough precision that it is convenient to consider digital signals processing the same as discrete signals processing.

2. Nyquist–Shannon Sampling Theorem

Since a signal is a function that carries information, it is important that the conversion of a continuous signal to a digital signal minimizes the loss of information. Computers do not store a signal as an analytical function, but instead they store a sample of points on the signal. As the sampling frequency is increased, the conversion from a continuous to digital signal looses less information.

THEOREM 2 (Nyquist–Shannon Sampling Theorem). *It is possible to sample a continuous signal frequently enough such that no information is lost in the conversion from a continuous signal to a discrete signal.*

This theorem guarantees the conversion from a continuous signal to a digital signal without the loss of information contained in the original signal. Let $f(t)$ be a continuous signal. From 2, there exist a method of lossless conversion of $f(t)$ to a discrete signal $f(n)$. This is done by sampling $f(t + k\delta)$ where $k \in \mathbb{Z}$ and δ is the sampling interval. δ must be small enough to ensure that there is no information loss. If δ is the interval between samples, then $1/\delta$ is the rate of sampling.

DEFINITION 3. Nyquist Rate is the minimum rate at which a continuous signal must be sampled such that no information is lost while converting it to a digital signal. Given a

signal $f(t)$, which contains information on different frequencies, the Nyquist Rate of $f(t)$ is equal to twice the maximum frequency of $f(t)$.

In practice, we do not know the maximum frequency of $f(t)$, so we do not use the minimum Nyquist Rate. Instead, sampling is preformed at rates greater than Nyquist Rate for a signal. By sampleing more often than the Nyquist Rate requires, we ensure that no information is lost.

3. Features of Signals

Often, signal processing is used as a precursor to a machine learning model. In these cases, and even when a signal is not directly used for this, we are interested in extracting features from the signal for further analysis.

Domain Features. Some features we extract are domain specific. For example, heart rate is an example of a domain signal that can be obtained from an electrocardiogram. Heart rate represents a value that has intuitive meaning to a clinician and most people when talking about health. Domain signals are typically the more obvious signals to first extract and process. Domain knowledge should not be overlooked, which is why it is important to be an expert in the field you are using signals processing in.

Signal Features. By contrast, a signal feature is a value that can be extracted from a signal but doesn't have as clear an interpretation in the domain space. Some examples of these include standard deviation, mean, maximum, minimum and so forth. Often, the fractal dimension is a signal feature that is measured during an electroencephalography because it has been shown to be a reliable measure of brain activity and health. Signal features are less intuitive to understand their meaning in the context of the domain, and not all signal features are always useful. For these reasons it is always important to do feature selection method when using signal features, which will be discussed later.

LECTURE II

Biomedical Signals

Domain knowledge is critical to successful signals processing, so this section provides an overview of a few common biomedical signals. Specifically, there are sections on electrocardiograms, electroencephalograms, and blood pressure, which are 3 commonly used clinical signals, and there is also a list of other commonly seen clinical and bioinformatics signals.

4. Electrocardiogram

This signal records the electrical activity of the heard. The standard ECG is taken with 12 leads or 12 sensors on the body taking these measurements; however, sometimes only a subset of the leads is actually used. Generally, it can be helpful to have an understanding of the anatomy of the heart and how the electrical signal propagates throughout the heart to better understand the need for each of the 12 leads, but abstracting away from the biology, an ECG remains an interesting signal to collect.

This electric signal is the result of the polarization and depolarization of the heart cells, which result in the heart contractions creating a beating heart. Because the heart is not the only muscle in the body, a substantial amount of noise from breathing and other physiological processes is collected.

Figure 2 shows a healthy ECG signal, with each part of the signal highlighted. Occasionally, there is an additional U wave that occurs between the P and T waves.

An ECG signal can be used to determine:

(1) Heart rate: R-R interval
(2) Heart rate variability: the derivative of the heart rate. Generally, a higher fractal dimension of heart rate variability corresponds to a healthier patient.
(3) The S-T interval is used to detect, far in advanced (often at least 12 hours), potential issues with or failure of a heart.

5. Electroencephalogram

Electroencephalogram (EEG) is a clinical test used to measure brain activity. Brain waves collected during an EEG are classified into different categories based on their frequencies.

- Delta: 0.5-4 Hz
- Theta: 4-8 Hz

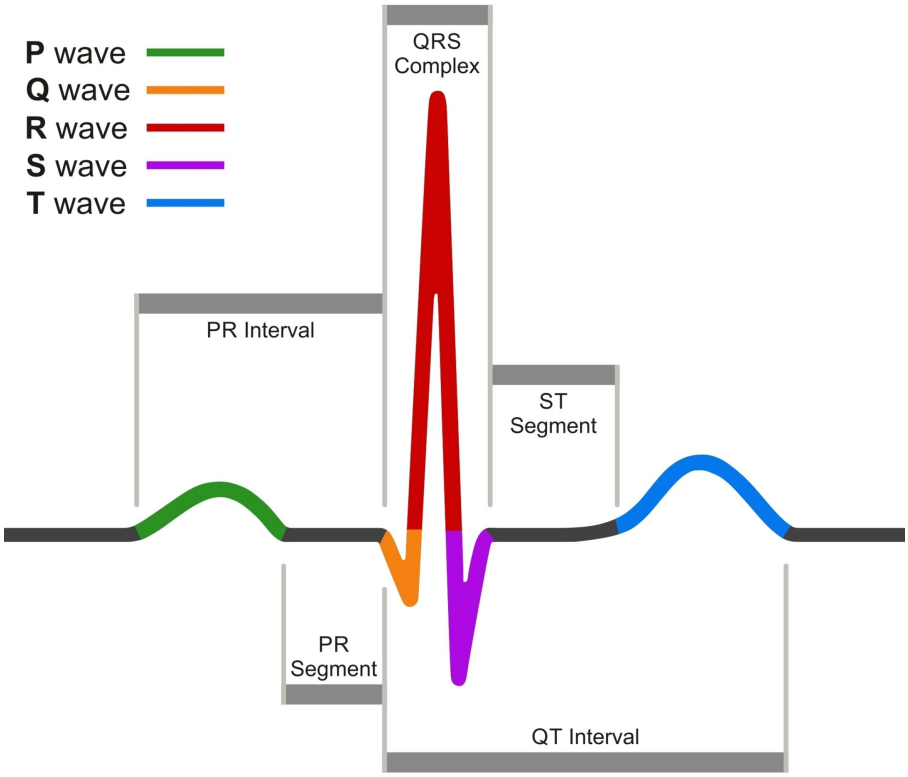

FIGURE 2. Healthy ECG signal

- Alpha: 8-14 Hz
- Beta: 14-31 Hz

Higher frequencies are associated with active thinking, and lower frequencies are associated with sleep or rest.

Often an EEG signal contains components in all frequencies, so it is important to be able to isolate and separate them out; however, it is possible to look at the signal and determine which frequency range is most active, which is a powerful diagnostic tool.

Additionally, the fractal dimension of an EEG is a very powerful diagnostic tool. As patients age, there should be a decrease in the fractal dimension of an EEG, but a sudden or sharp decrease in the fractal dimension is often associated with disease. The fractal dimension of a signal is an example of a feature signal, and the math for this will be discussed further in the section on Mother Wavelets.

6. Blood Pressure

Blood pressure describes the pressure of the blood in the arteries, and it is often measured arterially (arterial blood pressure or ABP). This signal is used in numerous applications, but getting an accurate measurement, more accurate than using an arm cuff and stethoscope, is invasive and requires inserting a catheter into the patient.

Although blood pressure is a continuous signal, typically 2 values are computed.

- Systolic blood pressure is the maximum pressure in the arteries.
- Diastolic blood pressure is the minimum pressure in the arteries.

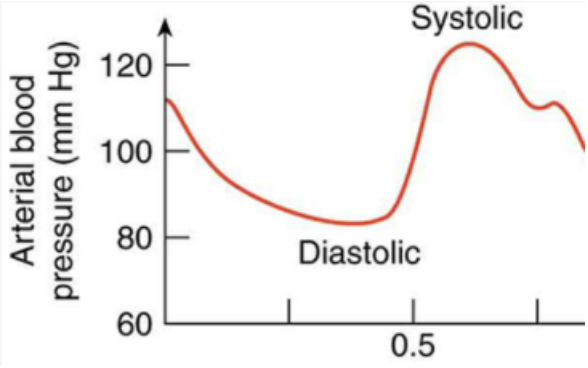

FIGURE 3. Blood Pressure Signal

Looking at figure 3, it can be seen that the area of systolic blood pressure takes up almost half as much as time as the diastolic section, so the mean arterial pressure (MAP) is calculated

$$\text{MAP} = \frac{2\text{Diastolic} + \text{Systolic}}{3}.$$

If there is a continuous way of calculating blood pressure, taking the integral or area under the curve is another way to determine the map. This is a more accurate way of calculating the MAP, but acquiring the continuous signal is an invasive procedure for the patient.

Another important property in blood pressure is the Dicrotic Notch, which is seen just to the right of the systolic peak in figure 3. Additionally, the interval between the systolic peaks of this signal is another method for calculating the heart rate.

7. Additional Clinical Signals

Photoplethysmogram (PPG). This is a noninvasive, optical measurement of blood's oxygen saturation. It has a signal structure similar to blood pressure.

Electromyogram (EMG). This signal is used to measure neuromuscular response, and show how active a muscle is. The raw data from these tests, which are performed by applying voltage over a muscle, create a very noisy signal.

8. Bioinformatics Signals

The following is a list of signals that are less common in clinical or diagnostic medicine, but are very popular areas of research.

- Amino-acid sequences

- DNA sequences
- Protein Folding

9. Multidimensional Signals

We have primarily considered signals of 2 dimension. For example, ECGs and PPGs are signals that have both amplitude and time. However, signals and signal processing techniques can be generalized to n dimensions. Techniques for image processing are based in the fundamentals of signals processing. This is a list of a few common multidimensional clinical signals.

- X-Rays or any other 2D medical imaging
- MRIs are a 3D imaging technique
- A coronary angiogram is a 3D signal
- Radiotherapy for treating cancer is processed as a 4D signal

LECTURE III

Mathematical Foundations

10. Complex Numbers

We are grocery shopping at Meijer, and we buy a 2 tubs of ice cream and 1 jar of hot fudge. Our goal is to represent what we bought in the simplest possible terms. We could use cartesian coordinates or vectors, but these solutions represent the data as a pair of 2 numbers. Then when we do operations on this, we will need to do more operations representing the data. Instead, we can represent the data as a complex number $z = 2+i1$, where the real component is the amount of ice cream and the imaginary term is the amount of ice cream we bought. This is a convenient approach because then the entire purchase is represented as 1 complex number rather than 2 real numbers.

Eulers Formula. Given a a complex number $z = x + iy$, we can represent this number in a cartesian plane where the y axis is imaginary and the x axis is the real part. Given this representation, we can have a magnitude A for this point such that $A = \sqrt{x^2 + y^2}$. Furthermore, there is an angle φ such that $\varphi = \tan^{-1}(y/x)$. From this, we can represent the value $z = Ae^{i\varphi}$.

THEOREM 4 (Euler's Formula).

$$e^{i\varphi} = \cos\varphi + i\sin\varphi$$

From this, we see a complex number $z = Ae^{i\varphi}$ can be represented as a series of sinosoidal waves.

11. Dirac Delta Function $\delta(t)$

The Dirac Delta Function or impulse function $\delta(t)$ was motivated for theoretical physics as a function that is equal to zero everywhere but when integrated is equal to 1. From this, it follows that

$$F\{\delta(t)\} = 1.$$

Although it was motivated to solve problems in physics, there are no physical systems that behave like this.

While no physical systems have behavior of $\delta(t)$, there are many instances where systems approach it. For example, in volleyball, a player is not allowed to hold the ball, but they must put enough force into the ball to hit it over the net. The goal of having minimal contact time and maximizing the force applied to the ball is similar to $\delta(t)$.

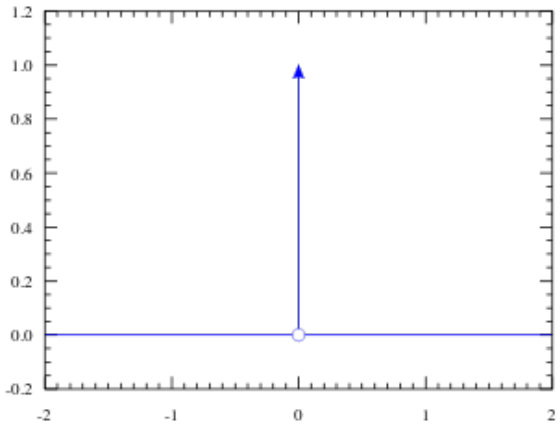

FIGURE 4. $\delta(t)$

12. Transformations

Given a credit card statement, we can answer a number of questions about an individual shopper. The questions

- When did they go to the grocery store?
- When did they but X last?
- When did ...?

It is easy to answer these questions given this data. However, answering how often ... questions is less intuitive given this data. Given data labeled in time, we can answer questions relating to frequency, but simply plotting out the data and taking a glance at it won't help. Instead, we need to transform our data from the time domain to the frequency domain.

The goal of a transformation is to take data recorded in one domain and represent the data in a new domain. Time and frequency are very intuitive units, but there are infinitely many domains to represent this data in. A transformation is helpful when it brings data into a domain that is suitable for answering our questions. Because the same data can be represented in any domain, all transformations must preserve the data and be invertible.

LECTURE IV

Fourier Transformation

The Fourier domain is the frequency domain, and the Fourier Transformation between time and frequency domains[1]. This section discusses the Fourier transform, its properties, and a Matlab implementation of it.

13. Continuous and Discrete Transform

The Fourier Transformation can be defined continuously or discretely.

Continuous Transformation. Given a continuous function $x(t)$ in the time domain, we define a $X(f)$ using the Fourier Transform as

$$X(f) = \int x(t)e^{-i2\pi ft}dt = \int x(t)\cos(2\pi ft)dt - i\int x(t)\sin(2\pi ft)dt$$

This is the analysis equation from time to frequency.

The amplitudes of $X(f)$ represent the dominate frequencies in the signal $x(t)$. The phase of $X(f)$ represents the frequency or periods of the frequencies of $x(t)$, so it has little meaning. Often, it is most helpful to graph the data by amplitude at each frequency, so visualize the dominate frequencies of a signal.

The Fourier transformation is invertible, so there is also the synthesis equation to compute $x(t)$ given $X(f)$.

$$x(t) = \int X(f)e^{i2\pi ft}df$$

Discrete Transformation. The analysis and synthesis equations for the discrete Fourier transform can be found by taking the summation analogue of their respective continuous equations. Given a discrete signal $x(n)$, $X(f)$ is computed

$$X(f) = \sum_{n=1}^{N} x(n)e^{-\frac{i2\pi}{N}fn}dt = \sum_{n=1}^{N} x(n)\cos\left(\frac{2\pi}{N}fn\right) - i\sum_{n=1}^{N} x(n)\sin\left(\frac{2\pi}{N}fn\right)$$

Similarly, given $X(f)$, $x(n)$ is computed

$$x(n) = \frac{1}{N}\sum_{n}^{N} X(f)e^{\frac{2\pi}{N}fn}$$

[1]A great video for intuition on the transform: https://www.youtube.com/watch?v=spUNpyF58BY

Signal Dimensions. Since signals can be more than 1 dimension, so it is important to generalize the transform to higher dimensions. This is commonly used in computed tomography and other medical imaging techniques. The Fourier Transformation analysis equation in 2 dimensions is written

$$X(f_1, f_2) = \int \int x(s,t) e^{-i2\pi(f_1 t + f_2 s)} ds dt.$$

This can be further generalized to $n-$dimensions.

14. Properties of the Fourier Transformation

There are a few key properties of the Fourier Transformation:

(1) The Fourier Transformation is a linear operator.
(2) Shifting Invariant:
$$F\{x(t-a) = X(f)e^{i2\pi af}\}$$
Because the amplitude of $e^{i2\pi af}$ is 1, it follows that $|F\{x(t-a)| = |F\{x(t)|$.
(3) Proportionality:
$$F\{x(at)\} = \frac{1}{a} X\left(\frac{f}{a}\right)$$
As $x(t)$ is scaled to $x(at)$, for some constant a, the change in phase will be accompanied by a change in frequency. This limits the amount of information that can be simultaneously obtained from a signal and its Fourier Transform[2].
(4) Convolution: Consider the convolution of 2 signals defined
$$x_1(t) * x_2(t) = x_2(t) * x_1(t) = \int x_1(\tau) x_2(t-\tau) d\tau$$
This is a big mess to integrate, but applying the Fourier Transform give the result
$$F\{x_1(t) * x_2(t)\} = F\{x_1(t)\} * F\{x_2(t)\} = X_1(f) \times F_2(f)$$
Taking the inverse Fourier Transformation of this, it is much faster to compute the convolution $x_1(t) * x_2(t)$.
(5) Derivative:
$$\frac{dx(t)}{dt} = (i2\pi f) X(f)$$
(6) Because the time signal and Fourier signal contain the same amount of information,
$$\int |x(t)|^2 dt = \int |X(f)|^2 df$$

[2]This related to the Heisenberg Uncertainty Principle of quantum mechanics, which states that it is impossible to have precise information on both the position and momentum of a particle at the same time. Similarly, we can have precise information in the time domain or the frequency domain, but there is a limit on the amount of information that can be obtained simultaneously from both domains. See: https://www-users.cse.umn.edu/ garrett/m/fun/uncertainty.pdf or https://math.uchicago.edu/ may/REU2013/REUPapers/Hill.pdf

15. Applications of the Fourier Transformation

The Fourier Transformation has applications in a variety of fields.

Signals Processing. Generally, the Fourier Transformation is used in signals processing to remove noise, smooth data, and filter signals, among other things.

Medical Imaging. Ultrasounds, computed tomography (CT) scans, MRIs, and other tests rely on the Fourier Transformation to create a 3D map of the inside of an object[3].

Other. It is commonly used in speech processing, music processing and generation, finance, the analysis of differential equations, and physics, among other things.

16. Short Term Fourier Transform

The time shift invariance of the Fourier Transform prevents the transformation from identifying the order of events and distinguishing the Fourier spectrum of each event in $x(t)$. One potential solution to this problem is to take the Fourier Transform for a window of time in $x(t)$. The Short Term Fourier Transform (STFT), defined in equation 1, uses $g(t-\tau)$ to focus select a window.

$$STFT(\tau, f) = \int g(t-\tau)x(t)e^{-i2\pi ft}dt \tag{1}$$

In the $STFT$, $g(t)$ is a gate in time or a function that has a value of 1 for a period of time and is zero everywhere else. τ is a parameter that allows us to place $g(t-\tau)$ over the area of the signal we wish to focus on. The advantage of the STFT is that is allows us to accurately describe a time section of the signal in both time and frequency. However, the STFT requires we select a window of interest, and selecting the appropriate window length is not a trivial task.

A disadvantage of STFT and the Fourier transform is that they both decompose functions or time signals into sinusoidal functions. This is problematic because the majority of signals of interest only change their values for a short period of time (i.e. they do not oscillate). A person has a healthy heart rate and then a heart attack. A person is healthy and then has a seizure before returning to normal. It is counterintuitive to decompose signals that exist for a short period of time into functions that continue to oscillate. It makes more sense to decompose these time signals into functions that exist for a short window of time and are 0 everywhere else. This is the motivating idea behind the Wavelet Transform.

[3]See: https://www.ajronline.org/doi/pdf/10.2214/AJR.07.2874

17. Fast Fourier Transform

Directly evaluating the analysis function of the discrete Fourier Transformation is $O(N^2)$ for a signal of length N. The Fast Fourier Transform (FFT) reduces this complexity to $O(N \log N)$. The most popular algorithm for FFT uses a Discrete Fourier Transform Matrix, W, which can be expressed as

$$W = \left(\frac{w^{i,j}}{\sqrt{N}}\right)_{i,j=0,\dots N-1} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & w & \dots & w^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & \dots & w^{(N-1)(N-1)} \end{bmatrix}$$

The algorithm is a divide and conquer algorithm that recursively splits W into smaller matrices to do the transformation. The algorithm was published by Cooley and Tukey in 1965[4], and it was cited by IEEE as one of the top 10 algorithms of the 20th century.

Matlab FFT. Matlab has an implementation of FFT. The following code constructs a signal x from high and low frequency components x1 and x2, applies FFT to x, and plots the components of x, the signal x, and its frequency spectrum. The plotting commands are not shown in the code, but can be found in the Matlab script at the end of the notes.

```
%% Fast Fourier Transform
Fs = 100;          % sampling frequency
L = 4;             % length of the signal in seconds
T = 1/Fs;          % the time between each sample
t = 0:T:L-T;       % time vector

% Construct a signal
x1 = 4*sin(2*pi*4*t);
x2 = 2*sin(2*pi*20*t);
x = x1 + x2;

% Uncomment these lines to add noise to your signal
% noise = 10;
% x = x + noise * rand(size(x)) - (noise / 2); % Add mean 0
    noise to the signal

% Fast Fourier Transform and shift the Fourier spectrum
Y = fft(x);
yshift = fftshift(Y);
magnitude_yshift = abs(yshift);
fshift = (-length(Y)/2:length(Y)/2-1)*Fs/(length(Y));
```

[4]See: Cooley and Tukey in 1965, An algorithm for the machine calculation of complex Fourier series

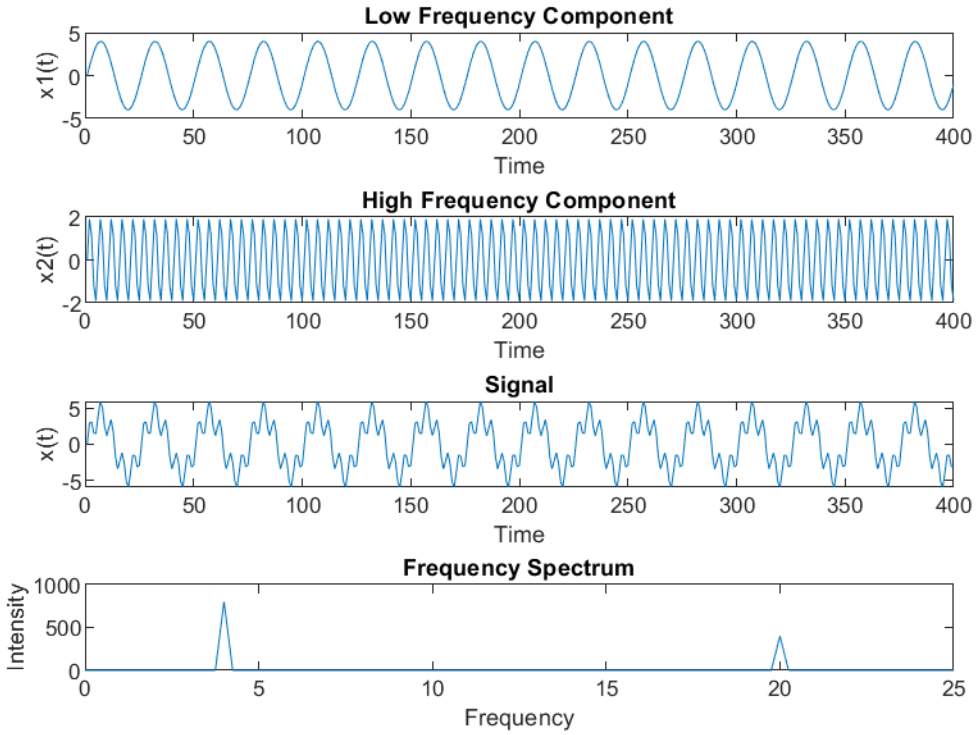

FIGURE 5. Signal x, its components, and its frequency spectrum

The first few lines of the code define attributes of the signal we are constructing (i.e. sampling frequency, length and a time vector). Lines 8 and 9 construct low and high frequency components of the signal, x1 and x2 respectively, which are added together to form x. Lines 13 and 14 can be uncommented to add noise to the signal. The FFT transformation is done on line 17 and outputs the variable Y, which is a vector of complex numbers. Lines 18-20 are used to convert the complex Fourier spectrum to a single sided real frequency spectrum, which we can plot.

The plots of the signal, its components, and its frequency spectrum are shown in figure 5. The top 2 plots show the low and high frequency components of the signal. The 3rd plot shows the entire signal x, and its frequency spectrum is shown in the bottom plot. The frequency spectrum has peaks at 4 and 20, which are the values we used to define x1 and x2 in the sinusoidal terms on lines 8 and 9.

This frequency spectrum is very clean, and has 0 values everywhere except these 2 peaks. However, most signals are corrupted with noise and measurements of other data. A noisy frequency spectrum and signal can be seen by uncommenting lines 13 and 14 and changing the intensity of the noise. There are a many techniques for signal denoising. As we will see, filtering is a method that allows us to isolate the components of a signal at different frequencies.

LECTURE V

Filters

Signal denoising for the purpose of extracting the relevant parts of a signal is a very important process and it is often acheived with filtering a signal. This section discusses ideal and real filters as well as their Matlab implementations.

18. Frequencies and Filtering

Given knowledge of which frequencies of a signal contain relevant information, removing noise, all intensity in the signal at other frequencies, can be achieved with a simple filter. The Fourier Transformation provides the relevant frequencies of a signal, and the relevant frequencies can often be identified using domain knowledge of the system you are working with.

Once the relevant frequencies are identified, extracting the relevant components of a signal can be accomplished with a simple filter.

The 4 major types of filters are, each of which filters, each of which maintains a different range of frequencies after filtering a signal.

(1) Lowpass Filter: low frequencies
(2) Bandpass Filter: middle frequencies
(3) Highpass Filter: high frequencies
(4) Notch Filter: low and high frequencies

19. Filtering Operation

Each type of filter can be represented as a function. Similar to the function $G(\tau)$ we considered in STFT, the filter functions are defined as $h(t)$ or $H(f)$, the later represents a window of view in the frequency domain and can be seen in figure 6. These filters have a value of 1 at frequency ranges that the filter keeps and 0 where the filter is removing components of the signal. The area where the filter has a high amplitude depends on the filter type.

Given a signal $x(t)$ and a system $h(t)$, the result of filtering the signal is computed with the convolution

$$y(t) = h(t) * x(t).$$

This convolution is difficult to compute, but the convolution property of the Fourier Transformation simplifies the calculations.

$$Y(f) = H(f) \times X(f)$$

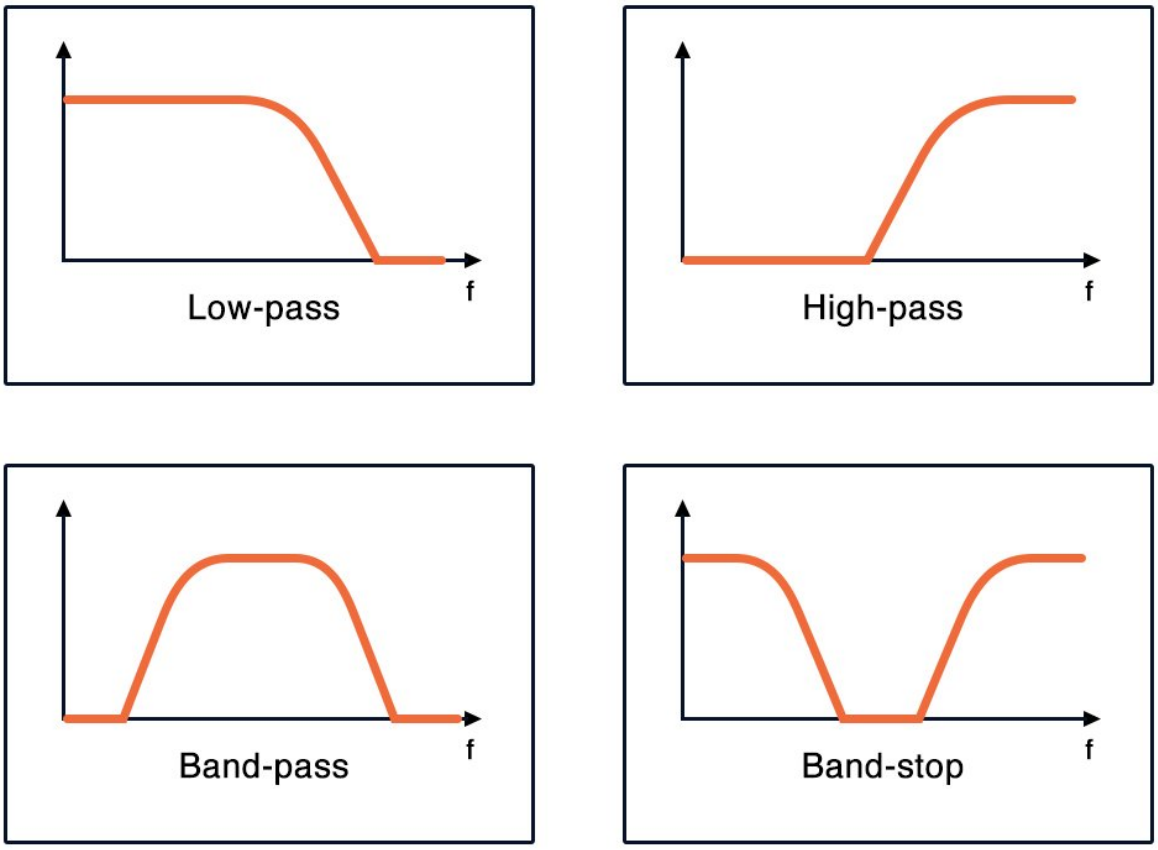

FIGURE 6. Types of Filters

20. Ideal Filters

Ideal filters can be described with a function containing a pass band and a stop band of the general form

$$H(f) = \begin{cases} 1 & \text{correct frequency range (pass band)} \\ 0 & \text{otherwise (stop band)} \end{cases}$$

The condition for the correct frequency range depends on the type of filter being used. For example, a lowpass filter that maintains all frequencies below f' will have the condition: $f < f'$, and a bandpass filter designed to maintain all frequencies between f_L and f_H will have the condition: $f_L \leq f \leq f_H$. Regardless of the frequency condition, $H(f)$ is a discontinuous function, which is the defining characteristic of an ideal filter.

While these filters are ideal, in practice it is not possible to construct an ideal filter.

PROOF. The output of an ideal filter $H(f)$ is the inverse Fourier Transform of $H(f)$. Generally, this takes the form

$$h(t) = \int H(f)e^{i2\pi ft}df.$$

In the case where $H(f)$ is a lowpass filter, this integral is evaluated

$$h(t) = \int H(f)e^{i2\pi ft}df = \int_{-f'}^{f'} e^{i2\pi ft}df = f' \operatorname{sinc}(f'f) \tag{2}$$

The output of an ideal filter, $h(t)$ is a function of sinc, where $\operatorname{sinc}(x) = \frac{\sin(x)}{x}$. However, sinc has nonzero values over all of $(-\infty, \infty)$. This suggest that the output of an ideal filter must have an infinite response time. This means it will take infinitely long to get the output from an ideal filter, which makes it non-realizable. □

The above proof shows lowpass filters are non-realizable, and this proof can be extended to bandpass, highpass, and notch filters by reevaluating the integral in equation 2 for the different frequency ranges associated with each filter type.

21. Real Filters

Since ideal filters are not practical to work with, we turn our attention to real filters. A real filter is an approximation of an ideal filter. Generally, these approximations are made by making the filter function continuous, which can be accomplished by creating a transfer function between the pass band and stop band.

Matlab Butter Filter. The Butterworth filter[1] is a popular filter that is implemented in Matlab. The code below uses the butter fiter to extract the high and low frequency components of the signal x that was constructed in the section on the FFT. The butter function accepts 3 arguments:

(1) order
(2) frequency ranges
(3) filter type

Order is an argument that specifies how steep the transition between the pass band and stop band is. The frequency ranges specify which components of the signal we are interested in. The Filter type specifies if it is a high pass, low pass, etc. type of filter. To separate the components of the signal, we construct high and low pass filters.

```
order = 3;
[b_low,a_low] = butter(order,10/Fs, 'low');

order = 5;
[b_high,a_high] = butter(order,20/Fs,'high');
```

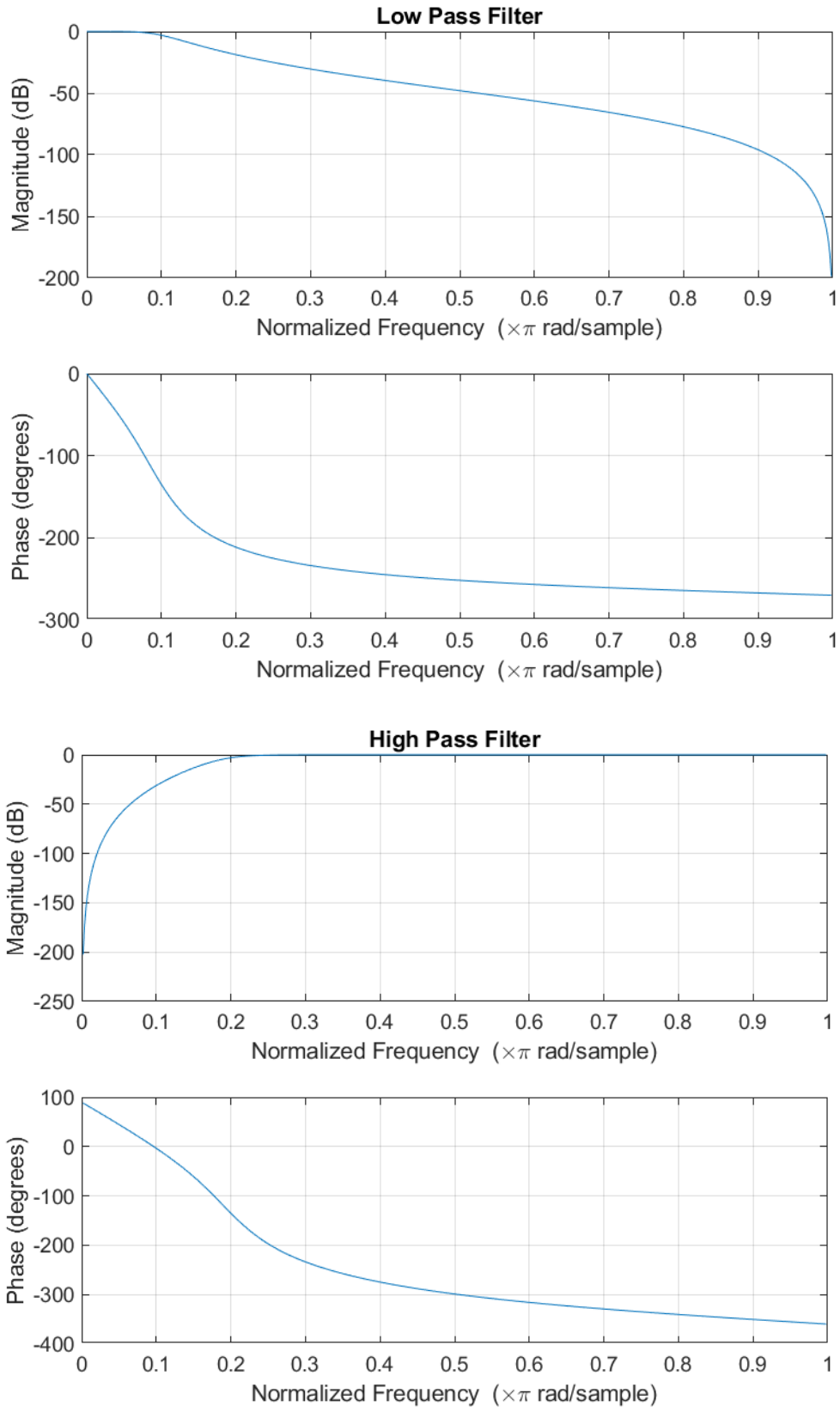

FIGURE 7. Matlab High and Low Pass Filters

Figure 7 shows the frequency and phases responses of the high and low pass filters constructed in the above code. These filters are used to separate the components of the signal in the following code

```
1 x_low = filter(b_low,a_low,x);
2 x_high = filter(b_high,a_high,x);
```

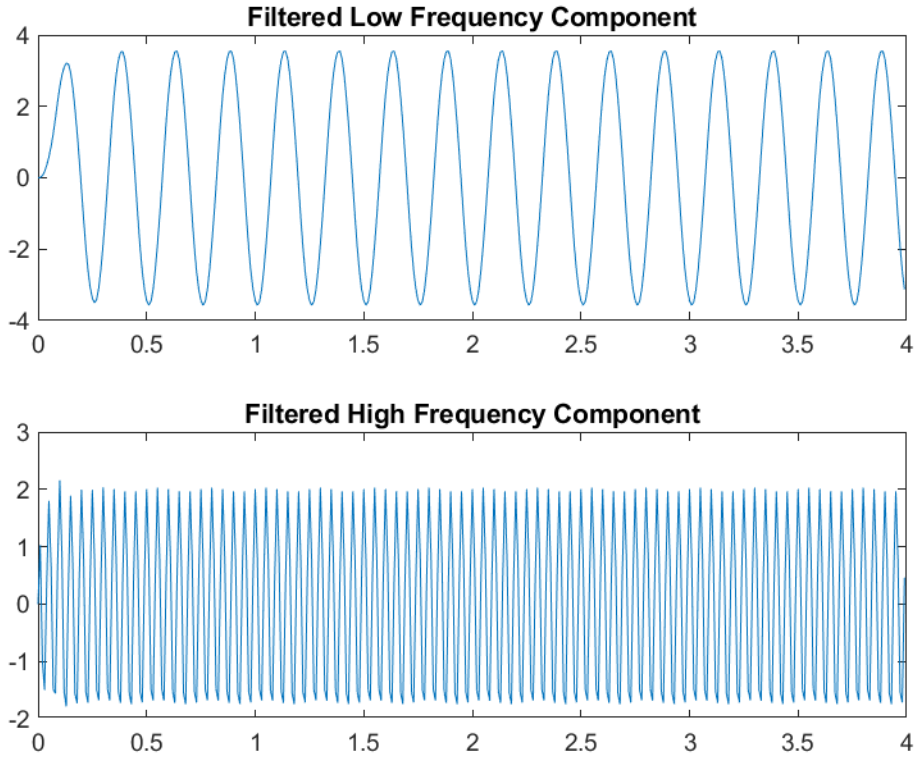

FIGURE 8. Separated Signal Components

Comparing the output of each filter to the initial components x1 and x2, which are shown in figure 5, the output of the filters is very close to what we would expect. Especially in the high frequency component plot, we notice that there is still a slight low frequency component remaining. This can be further removed by modifying the filter settings on line 4 and 5.

In the next section, we move away from frequency analysis and consider a more general decomposition method.

[1]See: Butterworth, On the Theorey of Filter Amplifiers

LECTURE VI

Wavelet Transformation

22. Fourier to Wavelet

The motivation of the STFT is that viewing a window of a signal allows for locality of the frequencies. This is helpful because the Fourier Transform is invariant to phase shifts. However, when τ is used to select a window of a signal $x(t)$ to apply STFT to, the windowed signal will be decomposed into sinusoidal functions. This is problematic because the signal in the window only occurs for the period of time τ is described by a set of basis functions that are infinite in length.

The Wavelet Transform is a generalization of the Fourier Transform that decomposes signals into a basis defined by a Mother Wavelet function Ψ[1]. The notion of frequency, associated with the Fourier domain, disappears for the Wavelet Transform because the basis functions are will be time limited and not periodic to address the issues discussed above. However, these time limited functions will be represented as multiresolutions scalings and compressions of one another. This allows the Wavelet Tansform to contain information about both time and frequency of a signal, something that was not originally possible with only the signal or its Fourier transform.

23. Wavelet Transform

Given a Mother Wavelet function Ψ, the Wavelet Transform is defined

$$W_{\Psi,x(t)}(a,b) = \frac{1}{\sqrt{a}} \int x(t)\Psi\left(\frac{t-b}{a}\right) dt$$

The synthesis equation of the Wavelet Transform is given

$$x(t) = \frac{C_\Psi^{-1}}{a^2} \int \int W_{\Psi,x(t)}(a,b)\Psi\left(\frac{t-b}{a}\right) da\, db$$

In the synthesis equation, a, b, and C_Ψ^{-1} are constants. a and b represent the scaling and shifting parameters respectively ($a \neq 0$). These allow for multi-resolution in both time and frequency.

[1]The Wavelet and Fourier Transforms are equivalent when $\Psi(t) = e^{-2i\pi t}$.

24. Mother Wavelet

The key feature of a Mother Wavelet function is that it exist for a short period of time but has a value of 0 everywhere else. This solves the problem we experienced with the Fourier and Short Term Fourier Transforms.

A dominate heuristic used for choosing these functions is that they should qualitatively resemble the main variation in the signal being transformed. For example, the Mother Wavelet function for the Wavelet Transform of an ECG, which has 1 main beat at the QRS complex, should be a simpler function than the Mother Wavelet used to transform the audio of a symphony, which is a more complicated than an ECG. Because the Mother Wavelet should resemble the underlying signal, domain knowledge is essential for using the Wavelet Transform.

Some common Mother Wavelet functions include:

- Daubecies (dbX)
- Harr
- Mexican Hat (Figure 9)
- Coiflets
- Symlets
- Morlet
- Meyer

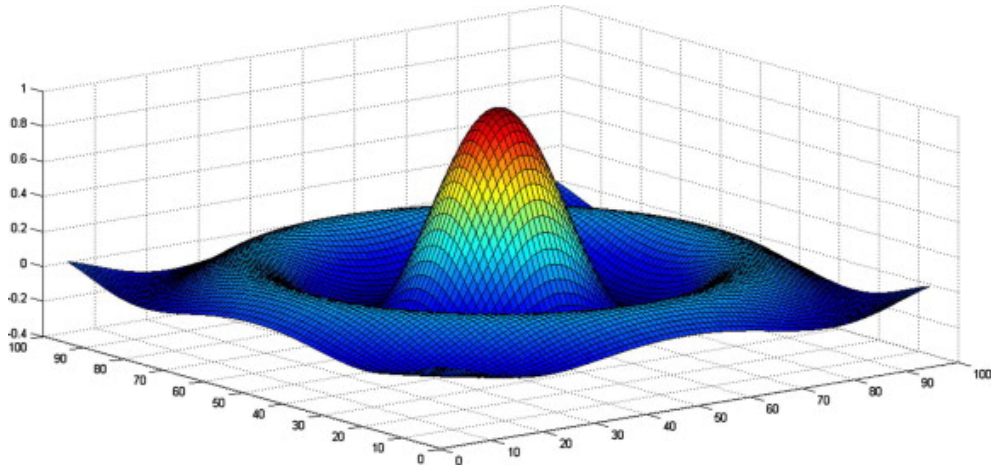

FIGURE 9. This is the Mexican Hat Mother Wavelet in 3D. The same function can be given in 2D, but it really looks like a hat in 3D.

25. Fractals

The Wavelet Transform gives insight into signals, such as fractals, that are not clearly possible to detect using the Fourier or time domains. A fractal is a pattern created with the same shape by repeatedly scaling, shifting, or rotating it. This is similar to how the Wavelet transform represents complex signals with many scalings and shiftings of the Mother Wavelet function.

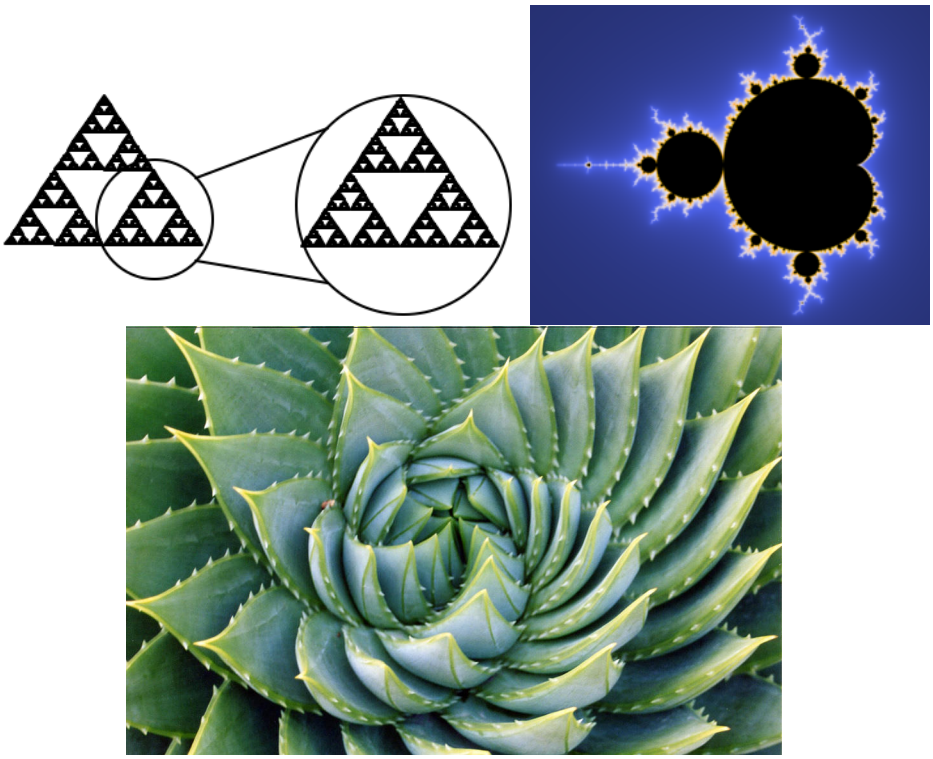

FIGURE 10. Examples of Fractals

In figure 10, some examples of fractals are given. In the bottom picture, a real plant takes the shape of a fractal. Fractals are common throughout nature and biology, both in terms of physical shape and different signals. For example, the shape lungs form into and the growth of a tree can both be represented accurately with fractals. Intuitively, this makes sense because as a tree grows a branch, it splits off into more branches, which grow and split off into more branches and so on. The vasculature of lungs grow similarly.

Less intuitively, the fractal dimension of an EEG, which measures brain waves, can be used diagnostically. Generally, the fractal dimension of an EEG increases as a brain develops and then gradually decreases over the remainder of the life. In humans the peak fractal dimension of an EEG occurs around the age of 20. However, the EEG of a patient having a seizure will have a significantly lower than expected fractal dimension. Fractal dimensions also correlate with stroke[2] and breast cancer patients[3], as well as a number of other diseases.

Matlab Fractal from an ECG. The code below demonstrates how a fractal can be constructed from an ECG signal using the continuous Wavelet transformation (CWT).

[2]Fractal Dimension of EEG Activity Senses Neuronal Impairment in Acute Stroke: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4072666/

[3]The Goodness-of-fit of the Fractal Dimension as a Diagnostic Factor in Breast Cancer: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6349609/

The ECG signal used in this code can be loaded into Matlab using the following line of code.

```
load('data\ecg.mat')
```

The entire signal represents 5 and a half minutes of an ECG, and figure 11 shows 4 seconds of the ECG signal. These 4 seconds were selected because the signal is relatively consistent during this window and limiting the window makes it easier to identify features in the signal from the plot. It is easy to identify the QRS complex and other major features of this ECG.

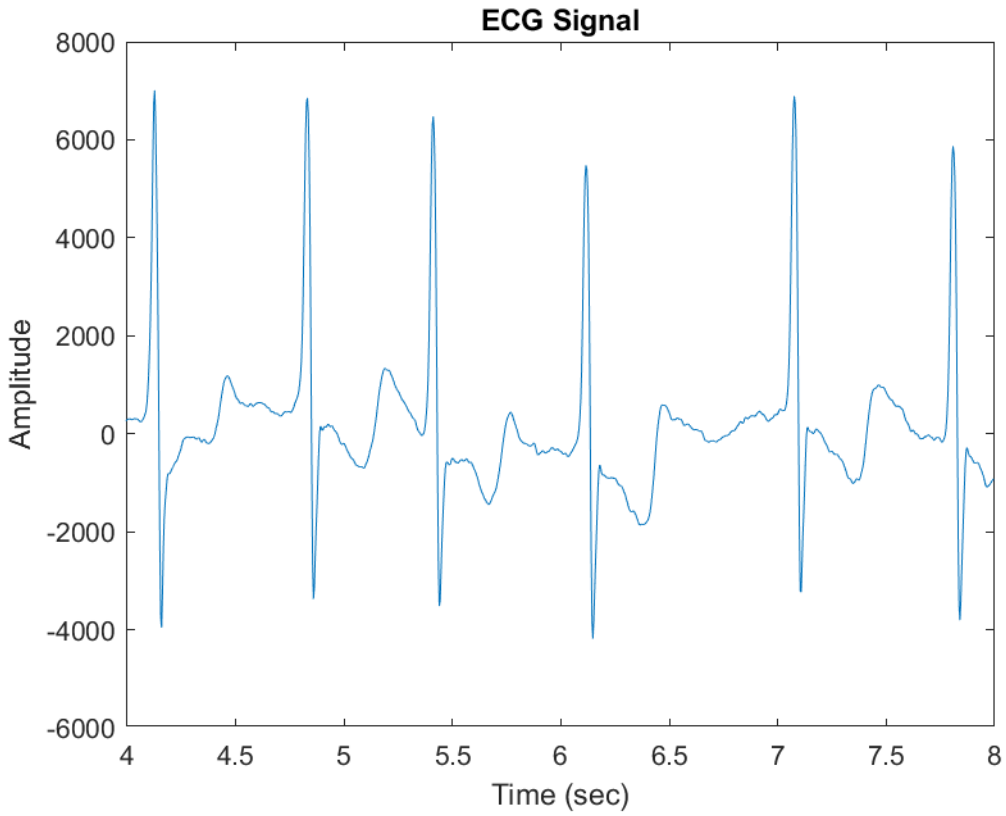

FIGURE 11. ECG Signal

The CWT is applied to the ECG using the following line of code.

```
cwt(ecg(4*Fs:8*Fs),1:50,'db4','plot');
```

This applies the transform to the same 4 second window that is displayed in figure 11. The function cwt with no output generates the plot in figure 12. Figure 12 is a scalogram, which is a plot that shows the absolute value of the CWT as a function of time (b) and frequency (a), where frequency is plotted on a log scale. The fractal that is generated in this scalogram represents the ECG signal's self similarity when viewed from multiple scales.

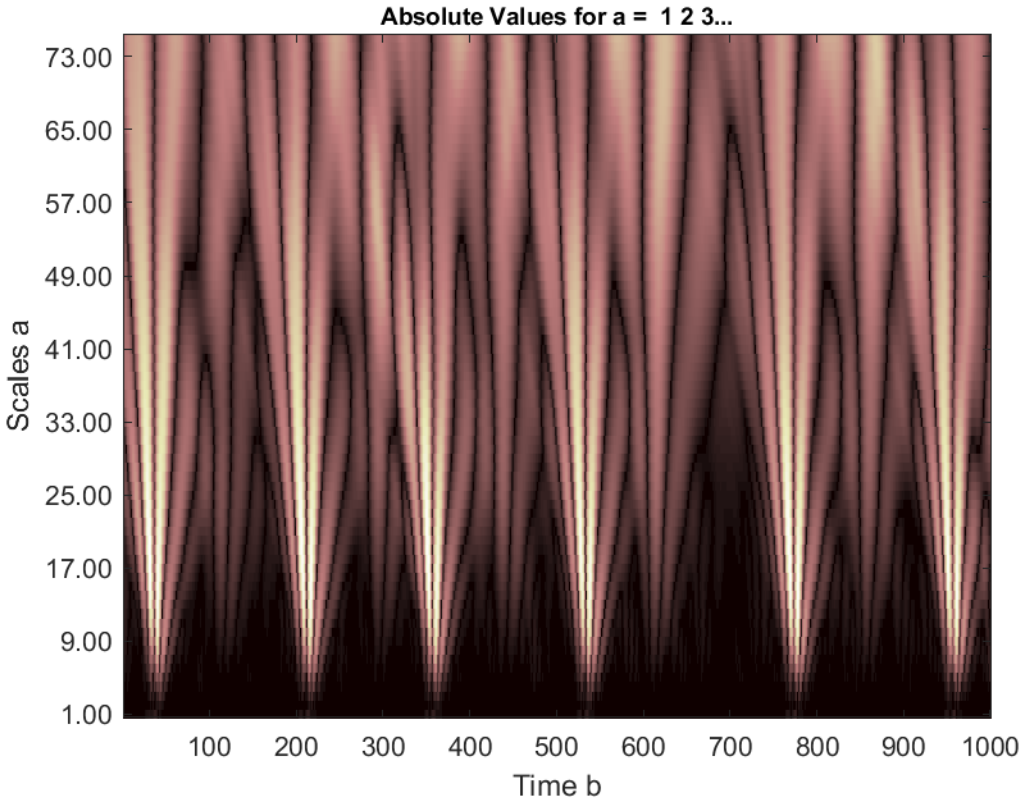

FIGURE 12. ECG Scalogram Fractal

26. Fractal Dimension

An important property of fractals is that they can have fractional dimensions, hence the name. Intuitively, any fractal drawn on the plane must have a dimension between 1 and 2 because it must have at least the dimension of a straight line and less than the dimension of a plane. Intuitively, a fractal the looks more like a straight line has a dimension closer to 1, and a fractal that is jagged and all over the plane has a dimension closer to 2.

To understand what the dimension of a fractal means, we can consider the motivation for fractals. The original motivating question for fractals was to measure the coast line of England. A simple solution is to take a map and a ruler and measure around, but a more accurate measurement could be taken with a more precise map and smaller ruler. Fractals were designed as an object to accommodate the fact that the further we zoom in the more accurately the coast line of England can be measured, as seen in figure 13.

The relationship between the number of segments used and the change in the calculated value defines the dimension of a fractal. Consider the fractal shown in figure 14, which shows the same Koch fractal in all 3 grids.

- In the top 3×9 grid, a total of 14 squares are filled.
- In the middle 6×18 grid, a total of 36 squares are filled.

FIGURE 13. As more and smaller line segments are used, we can calculate a more accurate value for the coast line.

- In the bottom 12×36 grid, a total of 86 squares are filled.

As smaller squares are used, more squares are requires to surround the fractal. The length of the line segments used to create the grid defines ε, the reciprocal of the number of segments used. Letting N be equal to the squares filled, we define the dimension of the fractal D with the equation

$$N = \varepsilon^{-D} \text{ or equivalently } D = -\frac{\log(N)}{\log(\varepsilon)}.$$

For the values above from figure 14, these values are computed as follows:

$$D = -\frac{\log(14)}{\log(\frac{1}{9})} \approx 1.20108\ldots$$
$$D = -\frac{\log(36)}{\log(\frac{1}{18})} \approx 1.23981\ldots$$
$$D = -\frac{\log(86)}{\log(\frac{1}{36})} \approx 1.24300\ldots$$

As the number of line segments used increases, a more accurate dimension for the fractal is calculated. The dimension of the perimeter of the Koch curve is approximately 1.26186.

27. Quadrature Mirror Filter

The Wavelet transformation is commonly implemented with Quadrature Mirror Filter (QMF) or Mallat Pyramidal Algorithm[4]. This algorithm preforms the transformation using a combination of high and low pass filters. The algorithm works to create a multi-resolution representation of the original signal. This means the algorithm works be examining the signal at different frequencies.

To begin the process, either a highpass filter $h(n)$ or lowpass filter $g(n)$ is defined. Only one filter must be chosen, as the choice for a highpass filter will define the lowpass filter

[4]See: A Theory for Multiresolution Signal Decomposition: The Wavelet Representation

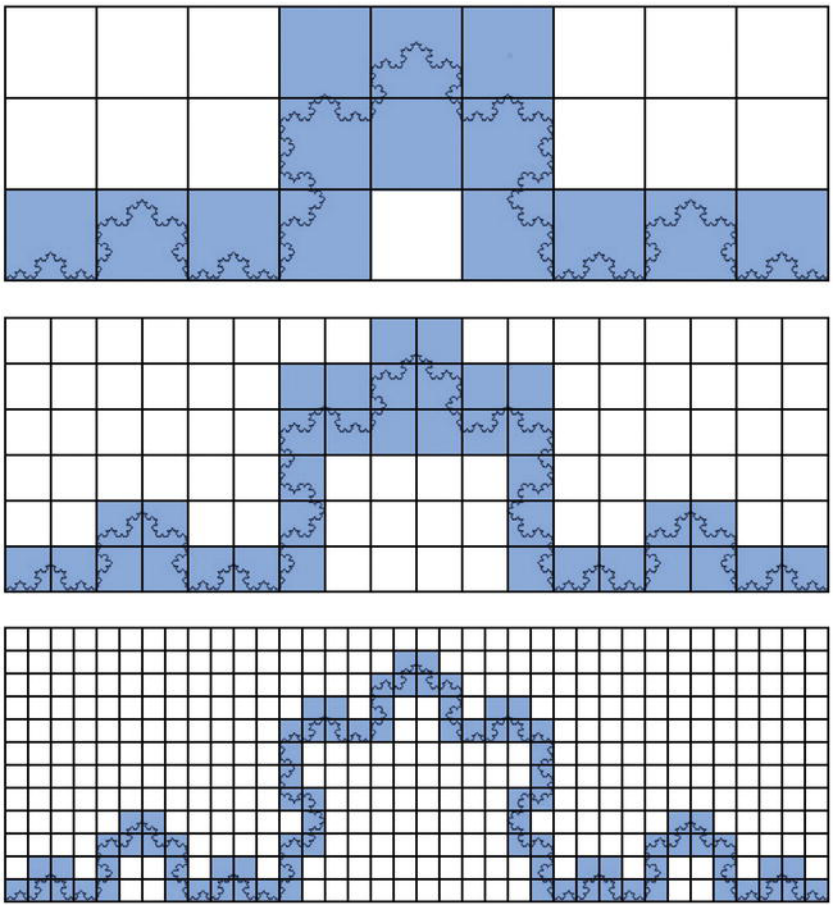

FIGURE 14. The log log relationship between the number of grid squares used and the area measured defines the dimension of this fractal.

and visa versa.

$$g(n) = h(2N - 1 - n).$$

Down Sampling. Given $h(k)$, $g(k)$, and an input signal, the Wavelet Transform is performed based on figure 15. Beginning with the input signal in the top diagram, it is first passed through both $h(x)$ and $g(x)$, separately. The input signal contained N data points, but after preforming this step, there are 2 separate signals containing N points each. To maintain a constant number of data point throughout this filter, the output signals from both filters are both down sampled by a factor of 2, represented by the circles with the downward arrow and 2 in them. This removes every other point from the signal, but because both outputs are down sampled, after this step the signals represented by d_1c and a_1c contain a total of N points, and no information has been lost. This information in this paragraph contain the fundamental structure of the first half of filtering (although the second half is very similar), and these steps are performed recursively.

Still looking at figure 15, the output of the low pass filter from the first step, a_1c, is passed through 2 new filters: $h_1(k)$ and $g_1(k)$. Again, these represent high and low pass

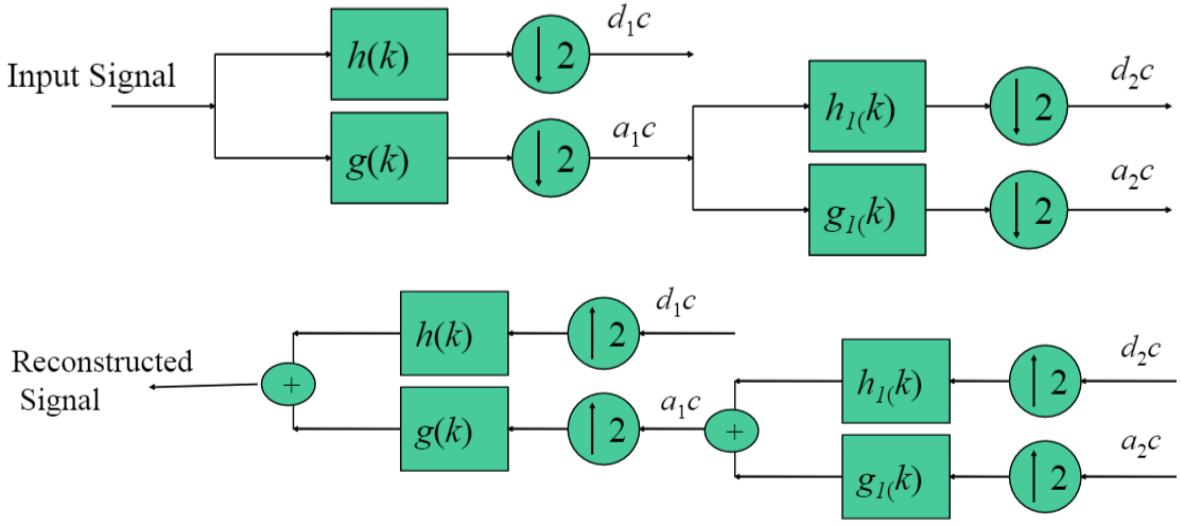

FIGURE 15. This figure represents the Wavelet Transform of an input signal given high and lowpass filters $h(k)$ and $g(k)$.

filters respectively; however, these filters are not the same as the filters from the previous step. They are defined with the equations:

$$h_1(k) = (-1)^{1-k} h(1-k) \text{ and } g_1 = h_1(2N-1-k)$$

Given these 2 filters, the fundamental process described in the above paragraph is repeated. Note, at the end of the top diagram in figure 15, signal d_1c contains $N/2$ points, and d_2c and a_2c both contain $N/4$ points, so the total size of the signal is preserved. Typically, this iterative process of passing the output from the low pass filter into a new set of filters is done 3 or 4 times, (myograms are an exception, and are often passed through 5 sets of filters).

Up Sampling. The second half of the transform is seen in the bottom diagram of figure 15. The reverse process of down sampling, up sampling, is performed to maintain number of data points as the samples are filtered again and added together. At each layer of the up sampling step, the high pass output from the corresponding layer in the down sampling process is up sampled and used as input to the high pass filter and added with the other part of the signal, which is directly input using the recursion.

Matlab Wavelet Transform. Matlab implements wavelet decomposition using the Mallat Pyramidal Algorithm. In the below code, the Wavelet transform is applied to the ECG signal from the previous section. The db4 Mother Wavelet is applied, and the code below is used to generate figure 16, a plot of the db4 function.

```
1 [phi,psi,xval] = wavefun('db4',10);
```

The transform is applied to the ECG using the below code.

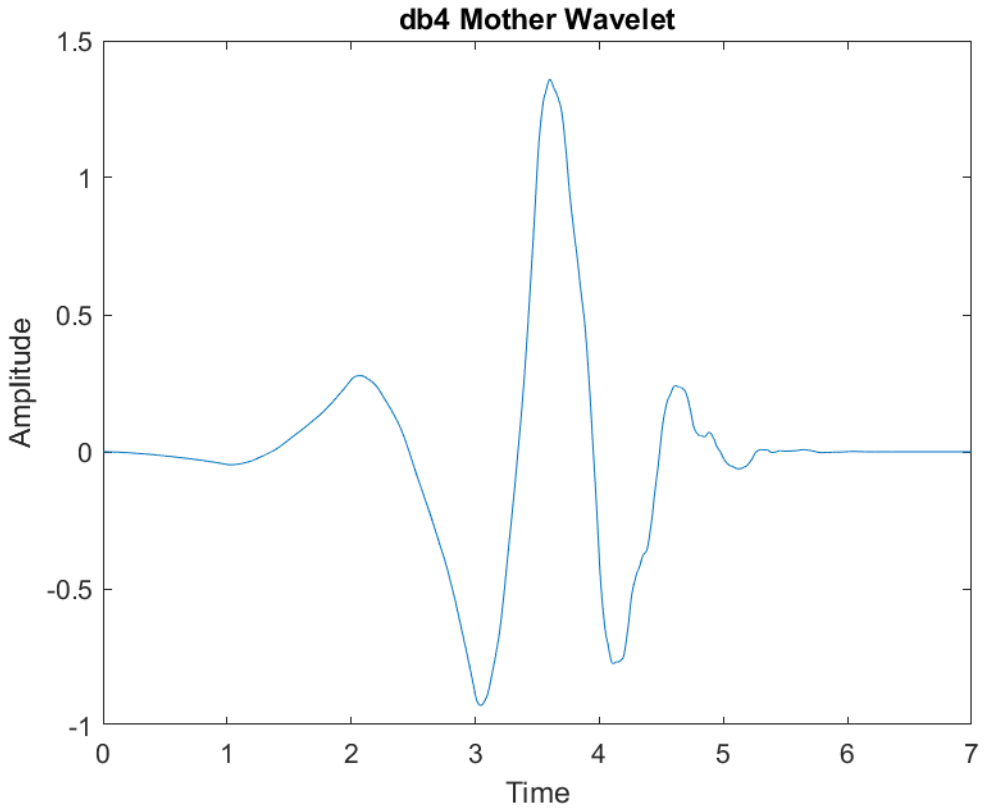

FIGURE 16. db4 Mother Wavelet

```
1  %% Wavelet Transform
2  % Wavelet Decomposition
3  [C,L] = wavedec(ecg, 4, 'db4');
4
5  % Reconstructed signal
6  rec = waverec(C,L,'db4');
7
8  % Remove different levels of the wavelet (QMF) decomposition by
9  % setting their coefficients to 0
10 C(1:L(1)+L(2)+L(3)+L(4)) = 0;
11 C(L(1)+L(2)+L(3)+L(4)+L(5)+1:end) = 0;
12
13 % Reconstructed signal
14 level_5 = waverec(C,L,'db2');
```

In this code, line 3 applied the Wavelet transformation using the QMF filter with 4 levels. This outputs C, which represents the Wavelet decomposition, and L, which represents the coefficients at each level. Using these outputs, a signal rec is constructed on line 6. Lines 10 and 11 are used to set the coefficients for every level, except 5, of the QMF filter

to 0. Using only level 5 of the QMF filter, a signal, level_5 is reconstructed on line 14. The original signal, reconstructed signal, and level 5 of the QMF are shown in figure 17.

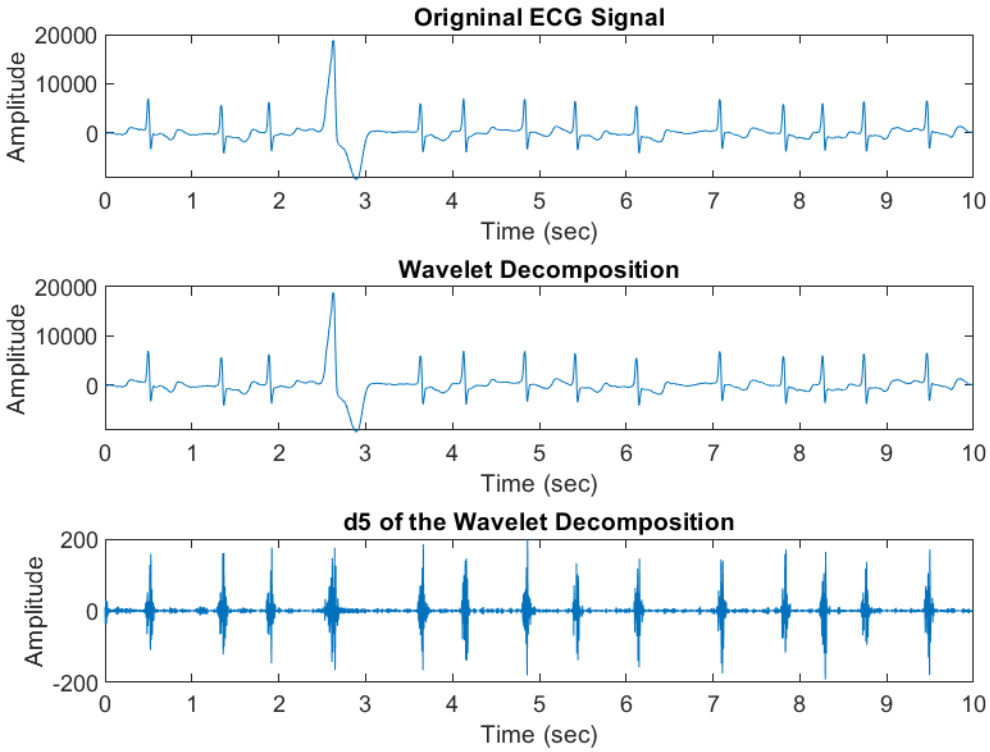

FIGURE 17. ECG Wavelet Decomposition

The top and middle plots are nearly identical, which shows how well the Wavelet decomposition is able to represent the original signal. The bottom plot doesn't represent the ECG signal well; however, it does identify the QRS complex. For each heart beat, there is a significant spike in the level 5 output of the QMF filter. This shows how levels of the filter are able to identify different features of a signal.

Part 2

Information Theory

LECTURE VII

Information and Entropy

28. Information

Information theory is the study of quantifying, storing, and usage of digital signals. The foundations of information theory come from ideas in math and physics, and the field has applications in machine learning, signals processing, cryptography, and a number of other fields.

The following sections consider a number of different methods of quantifying information. Generally, information of the information of a signal can be thought of as the level of surprise in a signal, data, or other communication. Consider the following 2 statements:

- "this section is about information theory"
- "my birthday is December 28"

The first statement is not surprising at all, since the reader already knew that this section is about information theory. The second statement is more surprising because the reader didn't know my birthday prior to reading the statement. As a result, the second statement contains more information than the first.

29. Shannon Entropy

Claude Shannon, a Michigan graduate and pioneer of information theory, developed the field based on his understanding of thermodynamics. In physics, entropy is the level of disorder and uncertainty in a system, and the second law of thermodynamics is that entropy is always increasing in nature. In information theory, entropy is used to quantify how much information is in a signal.

Measuring the Significance of a Signal. Any signal comes with a probability that it contains new information, and this is one way to define how much information a signal contains. Let p_i be the probability of signal i occurring.

$$\text{information in "}i\text{"} = p_i \log_2 \left(\frac{1}{p_i}\right).$$

When $p_i = 1$, i is a certainty, so the signal contains no information. This is consistent with the calculation $1 \log(1) = 0$. Similarly, when $p_i = 0$, indicating that it is not possible to see "i", then $0 \log(\infty) = 0$, so this this definition also shows an impossible signal (a signal which can't occur) contains no information.

Entropy of an Alphabet. In practice, a signal contains multiple possible values to receive, so in order to understand the total amount of information in a signal the sum is taken over all possible characters in the signal. This is the entropy or Shannon Entropy of a signal.

Definition 5. Shannon Entropy of a signal describes the net information of a signal, H, transmitted from a source with the equation

$$H = \sum p_i \log_2 \left(\frac{1}{p_i}\right)$$

30. Huffman Codes

Another perspective in information theory is to ask the question of what is the minimum number of bits that must be sent to convey a set amount of information. The answer to this question is the result of the specific alphabet used; however, the value H for Shannon Entropy, calculated above, represents the minimum number of bits (only using a binary alphabet) that must be used to send a signal. In that sense, entropy is the shortest possible length of the code.

Hoffman codes is a technique used to encode a message in a number of bits that approaches the minimum number of bits required to send the message, defined by Shannon Entropy. The best way to do this is to construct an alphabet in which all characters are equally likely to occur.

Consider the above definition of the information of a character in a signal:

$$\text{information of symbol "}i\text{"} = p_i \log_2 \left(\frac{1}{p_i}\right).$$

In this definition $\log_2$ is used with the assumption that a binary signal is used; however, this idea can be generalized to $\log_n$, when the alphabet of the signal can contain n distinct characters.

Note: when $\log_2$ is used, the units for H are given by bits.

The fundamental relation between Shannon Entropy and Huffman Codes is that Shannon proved there exist a minimum length of a signal required to convey a certain amount of information, and Huffman found a method of determining what the minimum length signal is.

LECTURE VIII

Measures of Information

31. Joint Entropy

In the previous section, we consider the entropy and information of one signal at a time. However, often multiple signals are received from different sources simultaneously (ex. heart rate and blood pressure), so it can be helpful to consider the information contained in receiving both of these signals together.

DEFINITION 6. The joint entropy of 2 alphabets X and Y is defined

$$H(X,Y) = \sum_{x\in X, y\in Y} p_{x,y} \log\left(\frac{1}{p_{x,y}}\right)$$

where $p_{x,y} = p(x,y)$.

The summation in definition 6 could be split into 2 summations, each over X or Y, but that is only a choice of notation.

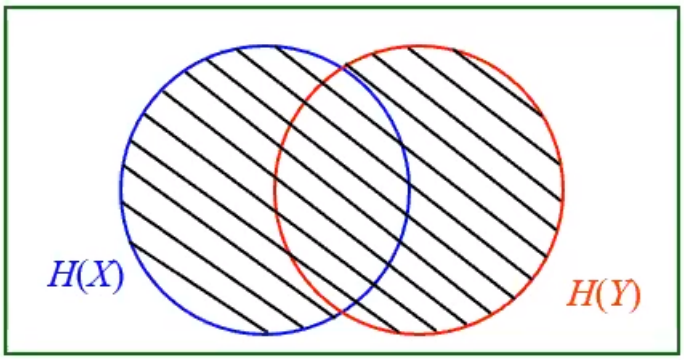

FIGURE 18. The entire shaded region represents $H(X,Y)$.

The notions of joint entropy and conditional entropy are defined similar to their corresponding probability terms and is similarly understood using ven diagrams, as seen in figure 18. This stems from the fact that the values $p_{x,y}$ actually represent probabilities. From this, it follows that $H(X,Y) = H(X) + H(Y)$ if and only if X and Y are independent.

32. Conditional Entropy

Joint entropy can be refined to consider the entropy of 1 signal conditional on another signal. Using the probability representation above, $p_{x|y}$ represents the probability of observing $x \in X$ given $y \in Y$, where X and Y are both alphabets used to transmit signals.

DEFINITION 7. The conditional entropy of 2 alphabets X and Y is defined

$$H(X|Y=y) = \sum_{x \in X} p_{x|y} \log\left(\frac{1}{p_{x|y}}\right).$$

From the ven diagram perspective, conditional entropy can be thought of as the circle representing the nonconditional set, less the intersection of the 2 circles. One way to see this is that $H(X) \geq H(X|Y)$. If X and Y are independent, then $H(X,Y) = H(X) + H(Y)$, so the intersection between the circles is 0, then the conditional entropy $H(X|Y) = H(X) = H(Y)$.

33. Mutual Information

Mutual information is the amount of information shared between 2 signals.

DEFINITION 8. Let X and Y be alphabets. The mutual information of X and Y, written $I(X,Y)$ is defined

$$I(X,Y) = H(Y)H(Y|X) = H(X)H(X|Y).$$

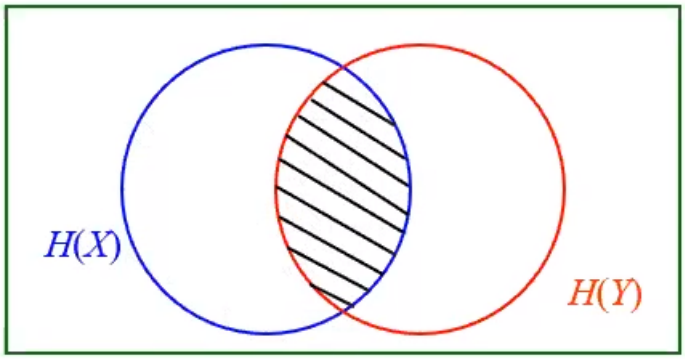

FIGURE 19. The shaded region represents $I(X,Y)$.

Based on the mathematical definitions as well as figures 18 and 19, it is always true that

$$I(X,Y) \leq H(X,Y).$$

It is a strict equality if and only if $X = Y$.

Distance between Signals. Given joint entropy and mutual information, the distance between 2 signals can be defined with the following equation.

$$D(X,Y) = H(X,Y) = I(X,Y)$$

In the ven diagram, this represents the union of the 2 circles minus the intersection. As the size the intersection (mutual information) increases, the distance between the signals goes to 0.

34. Kullback-Leibler Divergence

Kullback-Leibler Divergence, also known as relative entropy, is a metric designed to measure the difference between 2 probability distributions.

DEFINITION 9. The Kullback-Leibler Divergence between 2 probability distributions p and q, where the divergence is being measured with respect to p, is defined

$$D_{KL}(p,q) = \sum_{i=1}^{M} p_i \log\left(\frac{p_i}{q_i}\right)$$

It is important to note that $D_{KL}(p,q) \neq D_{KL}(q,p)$. For this reason, we will see later that Kullback-Leibler Divergence is not a valid distance metric, although it has many similar properties.

Cross Entropy. Cross entropy, a popular loss function, is defined from this definition of D_{KL}.

DEFINITION 10. The cross entropy of X and Y is defined

$$H(X,Y) = H(X) + D_{KL}(X,Y) = -\sum p_i \log(q_i).$$

First, note that $H(X,Y)$ now has 2 definitions because that is the current convention. Secondly, cross entropy is also not a distance function, but it is a divergence function.

35. Renyi Entropy

Renyi entropy is an entropy measure that can be parameterized to represent many different definitions of entropy.

DEFINITION 11. Hartley entropy is an entropy for an alphabet X, where X has n symbols, defined with the equation

$$H = \log(n)$$

DEFINITION 12. Min entropy is an entropy for an alphabet X is defined with the equation

$$H = -\log(\max_i p_i)$$

DEFINITION 13. Renyi entropy is parameterized with $0 \leq \alpha < 0$ defined

$$H_\alpha = \frac{1}{1-\alpha} \log\left(\sum_i p_i^\alpha\right)$$

As α varies, Renyi entropy represents different. When $\alpha \to 1$,

$$\lim_{\alpha \to 1} H_\alpha = -\sum_i p_i \log(p_i).$$

This shows as α goes to 1, Renyi entropy becomes equivalent to Shannon entropy. However, when $\alpha = 0$,

$$H_0 = \log(n),$$

where n is the number of symbols in the alphabet. This shows when $\alpha = 0$, Renyi entropy is Hartley entropy.

Renyi Divergence. Kullback-Leibler divergence can be generalized in terms of Renyi entropy.

DEFINITION 14. Renyi divergence is defined with the following equation

$$D(p,q)_\alpha = \frac{1}{1-\alpha} \log\left(\sum_i \frac{p_i^\alpha}{q^{\alpha-1}}\right)$$

As α goes to 1, Renyi entropy becomes equivalent to Kullback-Leibler divergence.

Part 3

Fundamentals of Machine Learning

LECTURE IX

From Biomedical Signals to Clinical Decision Making

36. The Role of Machine Learning in Biology

The goal of retrieving and processing biomedical data is to use this information to make decisions and uncover new patterns in the data. This process is diagrammed in figure 20. This decision making process based on the gathered information is the subject of machine learning.

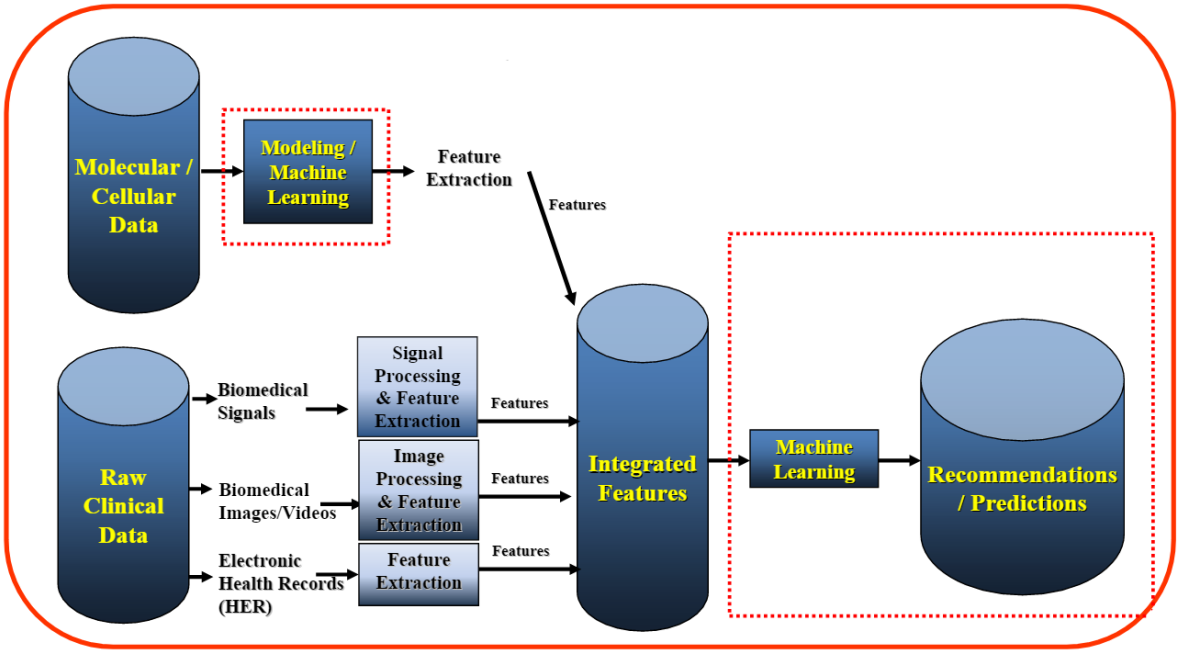

FIGURE 20. Clinical Decision Making Process

Mathematical Modeling. While machine learning is a popular topic today, there are many methods of mathematical modeling. The 2 main classes of these methods are:

- Statistical models
- Deterministic models

Statistical models are advantageous because they require very little domain specific knowledge. In this way, powerful tools can be used to model a variety of systems. Most

machine learning algorithms discussed here fall into this class. Powerful tools such as neural networks and random forest have been successfully used in many different domains.

The advantage of deterministic models is that they require domain specific information. Describing systems in terms of differential equations is a very powerful tool. While these models are very powerful, they are less flexible and tougher to fit to other systems. It requires an in depth understanding of the system being modeled in order to write and correctly parameterize a useful differential equation.

There are other methods of modeling, such as stochastic differential equations, the statistical and deterministic models represent the 2 main categories. Machine learning algorithms are primarily statistical models, but the use of domain knowledge is highly important in feature selection, model parameterization, and model evaluation. Machine learning is a set of tools that attempts to bridge the gap between these classes.

37. Overview of Machine Learning

Machine learning can be categorized into 3 main classes:

(1) Supervised Learning
(2) Unsupervised Learning
(3) Reinforcement Learning

In these notes, we will primarily focus on the first 2 classes of machine learning. While each class solves a different type of problem, there are many similarities between them. The following definitions are relevant to all 3 classes.

Definition 15. The feature space $\mathcal{X}$ is the space of all feature vectors x_i that a model will be trained and could be tested on.

Often, $\mathcal{X} = \mathbb{R}^n$ for some $n \in \mathbb{Z}$, but this space could be restricted in a number of ways. For example, $\mathcal{X}$ could be limited to vectors of only positive or binary values, or all vectors in $\mathcal{X}$ could be normalized. Categorical data, which doesn't fit into $\mathbb{R}^n$ nicely, can be inserted into $\mathcal{X}$ as one hot encodings.

Definition 16. The label space $\mathcal{Y}$ is the space of all labels that a model will be trained on and able to predict.

Labels are typically categorical variables in classification problems or continuous values in regression problems.

Definition 17. Training data S_n is the set of all data that the model will be trained on.

Definition 18. Testing data S_n' is the set of all data that a machine learning model can be tested on.

S_n and S_n' will include data from $\mathcal{X}$, and depending on the class of problem, they may include data from $\mathcal{Y}$ as well.

Supervised Learning. Supervised learning, also known as classification, is a form of machine learning where a model is given a training set of labeled data and must predict the labels of data in a test set. Mathematically, this means the training data is defined

$$S_n = \{x_i, y_i\},$$

where each feature vector is associated with a label. The test set is also given as

$$S'_n = \{x_i, y_i\}.$$

Classification models are trained on S_n, then predict labels for all $x_i \in S'_n$. The goal is to have the predicted labels match the labels associated with each $x_i \in S_n$.

Unsupervised Learning. Unsupervised learning, also known as clustering, is a machine learning technique used to group feature vectors into classes, which could be assigned labels. In unsupervised learning,

$$S_n = \{x_i\},$$

and the goal is to find classes y_i based on similarities between feature vectors. Conceptually, this can be a bit harder to understand, but figure 21 shows where this can be applied.

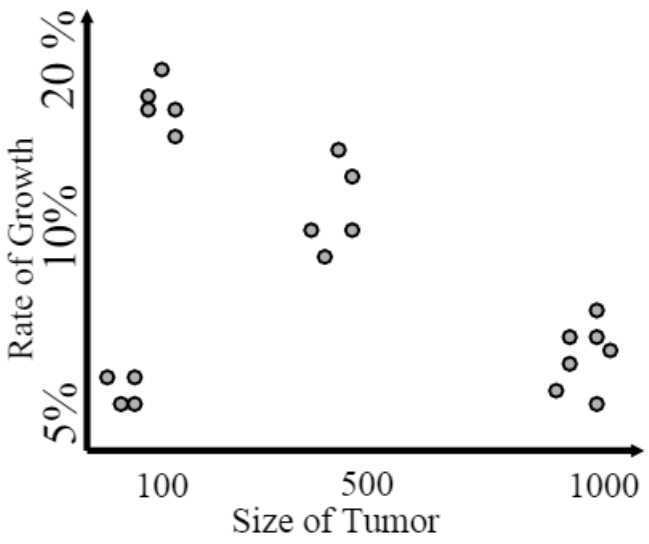

FIGURE 21. Unsupervised Learning for Tumors

Looking at the graph in figure 21, we clearly see there are 4 clusters of tumors, and once these clusters are identified, we can label each cluster as being a different type of tumor. Knowing which cluster a tumor belongs to can be relevant in clinical decision making. Often, clustering or unsupervised learning is an important precursor to a classification task as it allows us to label the data, which is a necessary step for forming S_n for a supervised classification problem.

The testing data S'_n used in unsupervised learning depends on the problem. It is possible that

$$S'_n = \{x_i, y_i\}.$$

In this case, evaluation of the clustering can be done by comparing how well the test data fits into the clusters generated by the model. However, it is also possible that there is no testing data in a clustering problem. In this case, the clusters can be evaluated using a number of different internal and external separation metrics.

LECTURE X

K-Means and K Nearest Neighbor

In order to understand the importance of some of the next sections, it is helpful to have an understanding of the algorithms we will be using. This section presents K-Means, a clustering algorithm, and K Nearest Neighbors (KNN).

38. K-Means

The K-Means is the simplest clustering algorithm, and many other clustering algorithms are adaptations of the simple version presented here. The algorithm partitions the feature vectors in S_n into k clusters. This is done by:

(1) Randomly initializing k feature vectors to be the centers of the k cluster.
(2) Assigning every feature vector to the cluster with the closest center.
(3) Update each cluster center to be the feature vectors that have the minimum distance from all other vectors in the cluster.
(4) Repeat steps 2 and 3 until the cluster centers no longer change.

The below algorithm gives the pseudo code for K-Means.

Algorithm 1 K-Mean

1: Randomly set k feature vectors $x_i \in S_n$ to be cluster centers $c_1, \ldots, c_k$
2: **do**
3: **for** $x_i \in S_n$ **do**
4: assign x_i to c_j, where c_j is the closest cluster center to x_i
5: **end for**
6: **for** $j \in 1, \ldots, k$ **do**
7: avg = the average of all x_i assigned to c_j
8: $c_j = x_i$ assigned to c_j such that x_i is closest to avg
9: **end for**
10: **while** any c_j updated or x_i was assigned to a new cluster in the last iteration

Two of the main steps of the algorithm, on lines 4 and 8, rely on choosing points that are "close." The notion of distance is important in machine learning and will be discussed in further sections.

K-Means is a simple to implement and fast to run algorithm. However, there are a few disadvantages of this algorithm:

- You must know the correct number of clusters before running the algorithm.

- The algorithm doesn't necessarily produce consistent results because the output depends on randomized initial conditions.
- It does not cluster complicated data well.

Using Euclidean distance, K-Means is unable to construct the red and blue clusters seen in figure 22. Both the red and blue clusters have centers inside of the red cluster, at the origin, so using K-Means won't assign the data points to the centers in a way that will identify the red and blue clusters. This demonstrates why K-means struggles to identify complicated clusters.

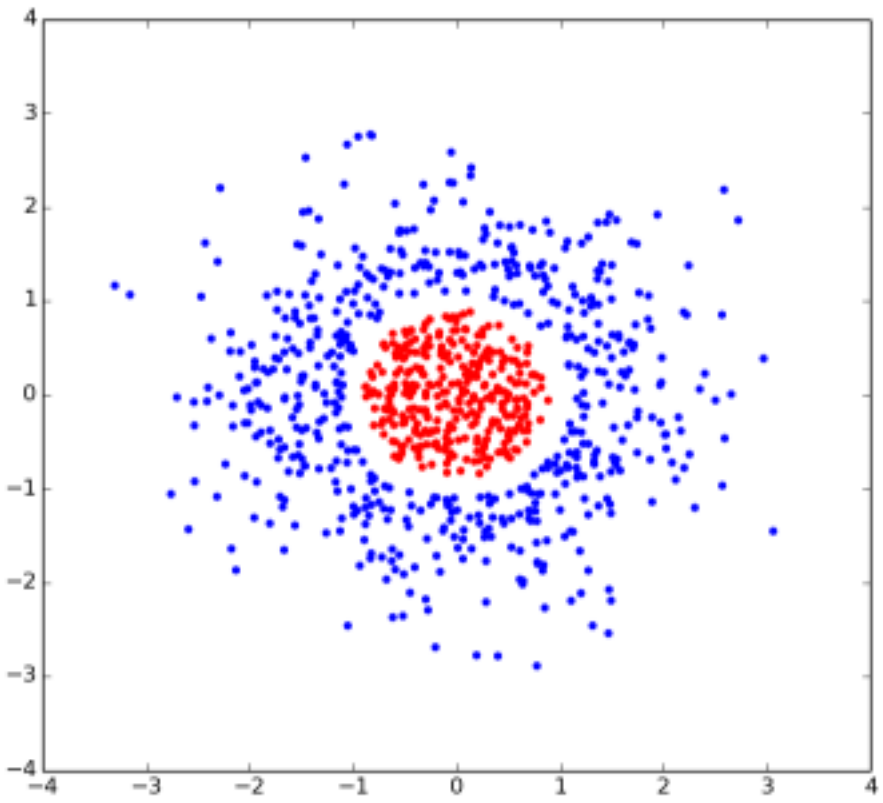

FIGURE 22. KNN can't cluster this dataset

Example. Figure 23 does a good job depicting a run of K-Means on 2 dimensional data. It shows 4 plots of the same data points, clustered by color. On this data, it takes 2 iterations to converge (top to bottom):

(1) 3 data points are randomly assigned to be the cluster centers, shown as the bold circles (initialization, line 1 of the algorithm).
(2) All data points are assigned to the cluster with the closest center (first iteration, lines 3-5).

(3) The cluster centers are updated (first iteration, lines 6-9),
(4) All data points are assigned to the cluster with the closest center (second iteration, lines 3-5).

After the cluster centers are assigned in the second iteration, no more data points change clusters and the cluster centers do not change, so the algorithm terminates.

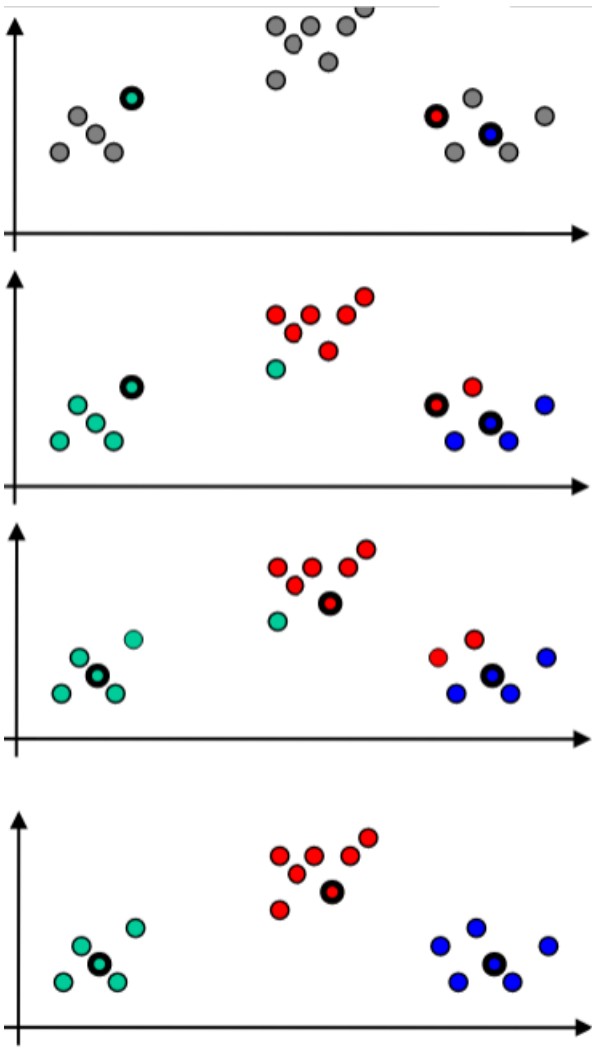

FIGURE 23. K-Means Sample Run

39. K Nearest Neighbors

K Nearest Neighbor (KNN) is a classification algorithm. It works to assign a label to $x_i' \in S_n'$ using the following steps:

(1) Compute the distance of x_i' to all $x_i \in S_n$.
(2) Select the $x_i \in S_n$ with the k lowest distances.

(3) Get the y_i associated with the x_i selected in the previous step.
(4) Assign x_i' the most common value of the y_is selected in the previous step.

The below algorithm gives the pseudo code for K Nearest Neighbors.

Algorithm 2 K Nearest Neighbor

1: **for** $x_i \in S_n$ **do**
2: Computer the distance between x_i' and x_i
3: **end for**
4: Select the k x_i with the smallest distances
5: Select the y_is associated with the x_i selected above
6: Assign x_i' the most common label of the y_is collected above

Similar to K-Means, KNN heavily depends on how the distance between feature vectors is measured, specifically on lines 2-4.

This algorithm doesn't require the model learning any parameters, which is advantageous. However, each prediction can be slow, depending on the size of S_n, because it must compute the distance between the feature vector being predicted and every point in S_n.

Example. Figure 24 shows an example of how KNN works. In the figure, the algorithm is classifying the green data point at the center.

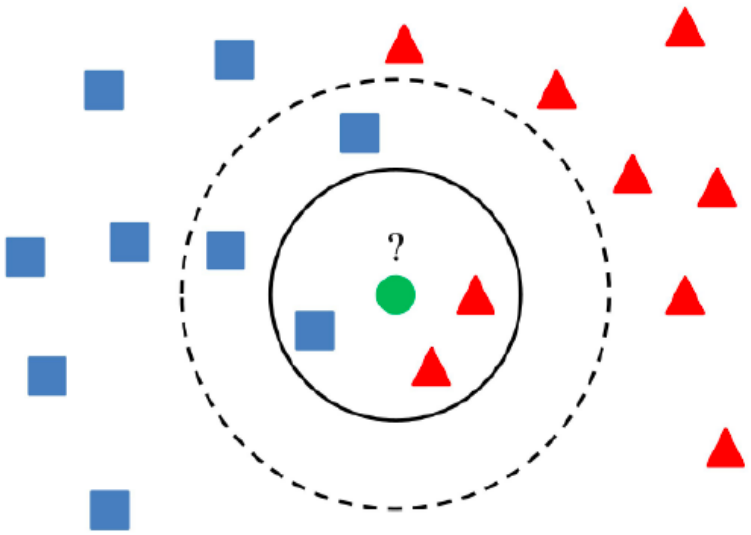

FIGURE 24. K Nearest Neighbor

The first step is to compute the distance between the green dot and every data point. While we are able to see which points are closest to the green dot at the center, the algorithm needs to compute the distances from every point. After that, we can select the k closest points to the green dot. If $k = 3$, the inner circle shows the 3 closest data points are 1 blue and 2 red, so red would be predicted. However, if $k = 5$, then there

are more blue than red, so blue would be predicted. The prediction depends on which k value is used.

LECTURE XI

Distances

As we have seen with with K-Means and KNN, distance is a fundamental concept to machine learning. In this section the notion of distance is formalized and some common distances are given.

40. Distance and Metric Spaces

In both supervised and unsupervised learning problems, $x_i \in S_n$, so it is important to understand the properties of $\mathcal{X}$. This motivates the following 2 definitions.

DEFINITION 19. A distance function d is a function that satisfies the following 4 properties:

(1) $d(x, y) \geq 0$
(2) $d(x, y) = d(y, x)$
(3) $d(x, y) = 0 \iff x = y$
(4) $d(x, y) \leq d(x, z) + d(z, y)$ (Donkey inequality)

DEFINITION 20. A metric space is a set X and a function d, where d is a valid distance function on the elements of X.

The feature space, $\mathcal{X}$, is a set that can be combined with a distance function to form a metric space. There are many functions that satisfy the 4 properties to be a valid distance, so every problem can construct a different metric space. The same algorithm will preform differently on different metric spaces, so it is important to have an understanding of distance functions.

41. Norms and Distances

A norm is similar to a distance function applied to only 1 vector.

DEFINITION 21. Let $p \in \mathbb{Z}_{\geq 0}$. A norm function $\|\cdot\|_p$ has the follow 4 properties:

(1) $0 \leq \|x\|_p$
(2) $\|x\|_p = 0 \iff x = 0$
(3) $\|kx\|_p = |k|\|x\|_p$
(4) $\|x + y\|_p \leq \|x\|_p + \|y\|_p$ (Donkey inequality)

Note that a norm function has 4 very similar properties to a distance function except it is only applied to 1 vector.

DEFINITION 22. A normed vector space $(V, \|\cdot\|)$ is a vector space V with a norm function that can be applied to V.

Given the similarities between norms and distances, we can redefined a metric space in terms of normed vector spaces.

DEFINITION 23. Given a normed vector space $(V, \|\cdot\|)$, the space (V, d) is a metric space where

$$d(x, y) = \|x - y\|$$

Note that definitions 20 and 23 are equivalent to one another. Since norms and distances are similar enough to define each other, we often discuss the norm or distance of a space interchangeably.

Commonly Used Norms. Given a vector $x \in \mathbb{R}^n$, these are some commonly used norms:

- $\|x\|_1 = x_1 + x_2 + \cdots + x_n$
- $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$
- $\|x\|_p = \sqrt[p]{x_1^p + x_2^p + \cdots + x_n^p}$

These norms, $\|x\|_p$, are also called the $L-p$ norm. In Matlab, the $\|x\|_p$ norm of a vector x can be computed with the following line of code:

```
n = norm(x,p)
```

These norms can be interpreted as the following distances.

Commonly Used Distances.

- City Block, Taxicab, Rectilinear and Manhattan Distance are the names of the distance defined in terms of $\|\cdot\|_1$. The distance defined with $\|\cdot\|_1$ on binary vectors is known as the Hamming Distance.
- Euclidean Distance is defined $d(x, y) = \|x - y\|_2 = \sqrt{(x \cdot y)^T (x \cdot y)}$. This is the distance associated with Euclidean Geometry.
- Maximum Metric, Chebyshev, Supremum, Uniform, and Chessboard Distance are names of the distance defined in terms of $\|\cdot\|_\infty$. It is calculated

  $$d(x, y) = \max(|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_1 - y_1|).$$

  This distance is popular in branches of controls and signals processing.

42. Locust of a Distance

Returning to the idea of a metric space, distance is used as a function to represent the relationship between elements in a data set. Consider figure 25.

Figure 25 shows a 2 dimensional vector space. Each curve plotted in the figure represents a set of data points that are all equidistant from the origin when using the norm denoted by the number next to the curve in quadrant 4. For example, the red line with a 2 next

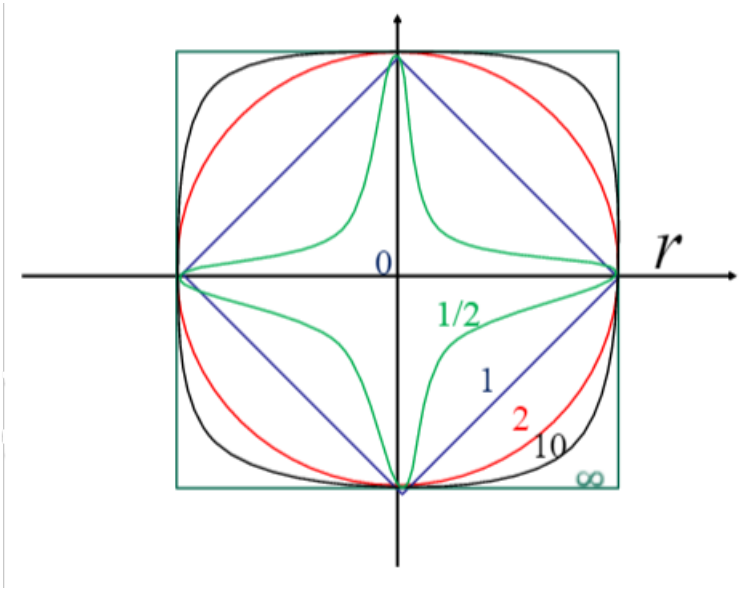

FIGURE 25. Distance Locuses

to it is a perfect circle. Using Euclidean Distance, which is defined in terms of $\|\cdot\|_2$, all points on a circle are equally far away from the center. The blue, rotated square is defined in terms of $\|\cdot\|_1$, and any 2 points in the square are equally far from the origin when measuring with this norm.

43. Weighted Distances

Consider the case of predicting which patients are at risk for heart disease. The information we have on the patients includes if there is a history of heart disease in the patients family and favorite color of each patient. Genetic history of heart disease seems a more important feature than favorite color in this classification problem, so when computing the distance the relative importance of these features should be taken into account. In weighted distances, this is done by assigning features different weights.

Let $x, y \in \mathcal{X}$, then the Euclidean Distance between x and y is given

$$d(x,y) = \sqrt{(x-y)^T \cdot (x-y)}.$$

To give weights to different features, consider the diagonal matrix

$$W = \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_n \end{pmatrix},$$

where the terms w_i is the weight of feature i. The weighted Euclidean Distance is calculated

$$d(x, y) = \sqrt{(x - y)^T W \cdot (x - y)}.$$

This function satisfies all 4 properties to be a valid distance function.

Letting $z = x - y$, the normed expression must satisfy $\sqrt{z^T W z} \geq 0$. It follows that $z^T W Z \geq 0$. From this, it follows that a weight matrix can be any positive semidefinite matrix[1].

44. Mahalanobis Distance

Consider the case of measuring the distance from a point to a cluster, in the shape of an ellipse, of other points. Since the cluster is elliptical and lacks complete symmetry, it is simpler to measure the distance from a point to a circle. Let A be the matrix of feature vectors represented in the cluster. The covariance matrix of the ellipse is defined

$$\Sigma = A^T A.$$

Geometrically, the covariance matrix represents the deviation (elongation, stretch, shrink, etc.) from the symmetric state. If we use the inverse of the covariance matrix to map all points in the cluster, then the cluster in this new space will be circular, and measuring the distance from a point to the cluster will be simple. This is the motivation behind Mahalanobis distance. Let $M = \Sigma^{-1}$ and m be the mean of all feature vectors in the cluster, the Mahalanobis distance is defined with the equation

$$d(x, m) = (x - y)^T M (x - m).$$

This is a very important, popular distance metric because it provides a metric for measuring the distance to an entire cluster and not any individual point. It is very effective for linearly separable data, but it doesn't work with not linearly separable data.

45. Distances in Practice

While this section is math heavy, it is critical to the remaining sections. To demonstrate why selecting the correct distance metric is important and how the same algorithm gives different results when applied to data in different metric spaces, the following code has an applies the K-Means and KNN algorithms to 2 different datasets using multiple distance metrics.

KNN. The Fisher Iris data set is a popular for introductory classification problems. It has 150 samples of 3 different flower types, each of which has 4 measurements recorded: sepal width, sepal length, petal width, and petal length[2]. In the following code, we only consider 2 dimensions of the data, sepal width and sepal length, because it makes

[1]For more on positive semidefinate matrices, see Chapter 10, Bilinear Forms, of Linear Algebra by Hoffman and Kunze.

[2]This data set was introduced by Fisher in the paper: The use of multiple measurements in taxonomic problems

visualizing the results easier. The goal of this section is to demonstrate how changing the distance function will change the predictions of a simple classifier model.

The first step is to load the data into Matlab.

```
%% Load the Iris data set
load fisheriris
X = meas(:,1:2);
y = categorical(species);
labels = categories(y);
```

The above code loads in the data, projects the 4 dimensional data into X, which only contains 2 dimensions, and extracts the labels. This projection can be seen in figure 26.

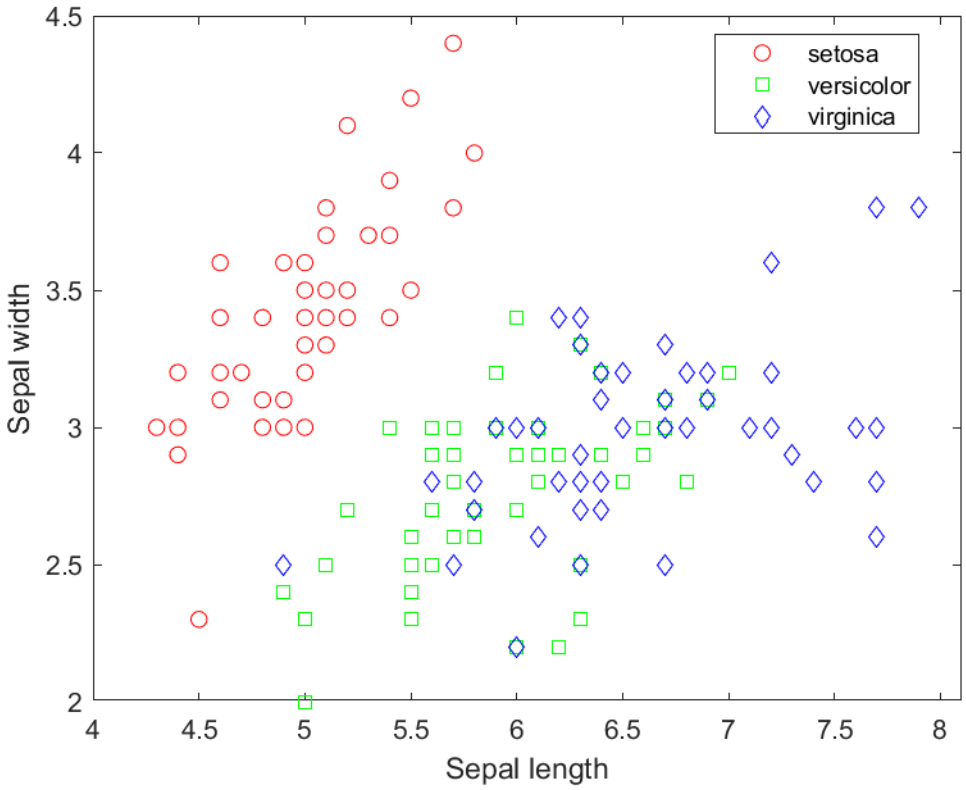

FIGURE 26. Caption

From figure 26, we see the Setosa flowers are easier to distinguish from the Versicolor or Virginica flowers, which have a fair amount of overlap in this projection. To demonstrate how a distance metric impacts a models predictions, the decision boundary can be plotted. A decision boundary is the border between regions in the metric space where everything inside of the regions have the same label predicted. To plot the decision boundaries, it is helpful to predict labels for every point in the space we are considering. To do this, a mesh grid is made over the region of interest, using the following code.

```
%% Generate a mesh grid
x1range = min(X(:,1)):.01:max(X(:,1));
x2range = min(X(:,2)):.01:max(X(:,2));
[xx1, xx2] = meshgrid(x1range,x2range);
XGrid = [xx1(:) xx2(:)];
```

A KNN model can be trained on the feature vectors and labels with a specific distance, and then make predictions on every point in the mesh grid. In the following code, this is done inside a loop, where the loop constructs models with different distance functions and the results are plotted in figure 27

```
%% Apply KNN with different distance functions
dists = ["euclidean", "cityblock", "chebychev", "mahalanobis"];
count = 1;
for d=dists
    % Create a KNN model with distance function d
    model = fitcknn(X,y,'NumNeighbors', 3,'Distance',d);
    % Make preditions using the model on the mesh grid
    predictions = predict(model,XGrid);
    % Plot the results
    subplot(2,2,count);
    gscatter(xx1(:), xx2(:), predictions,'rgb');
    title(d)
    count = count + 1;
    legend off, axis tight
end
```

In figure 27, we see the decision boundaries produced by KNN using 4 different distance functions. While most of the regions are similar, the top left is always red, the right is always blue, etc., the areas where they meet vary significantly depending on which distance function is used.

It is often impossible to visualize how these regions change, specifically when working with high dimensional data, but understanding that these regions change as a result of varying the hyperparameter k used in KNN, the distance function, or even the model type is a primary focus of classification problem.

K-Means. In this section, a similar experiment is preformed to demonstrate how distance impacts the K-Means algorithm. This experiment doesn't use a published data set, so instead the first step is to generate data to cluster. This is done with the following code.

```
%% Generate Cluster Data
num_points = 1000;
X = [randn(num_points,1)*0.1, randn(num_points,1)*1 + 3;
    randn(num_points,1)*1-0.5, randn(num_points,1)*0.1];
```

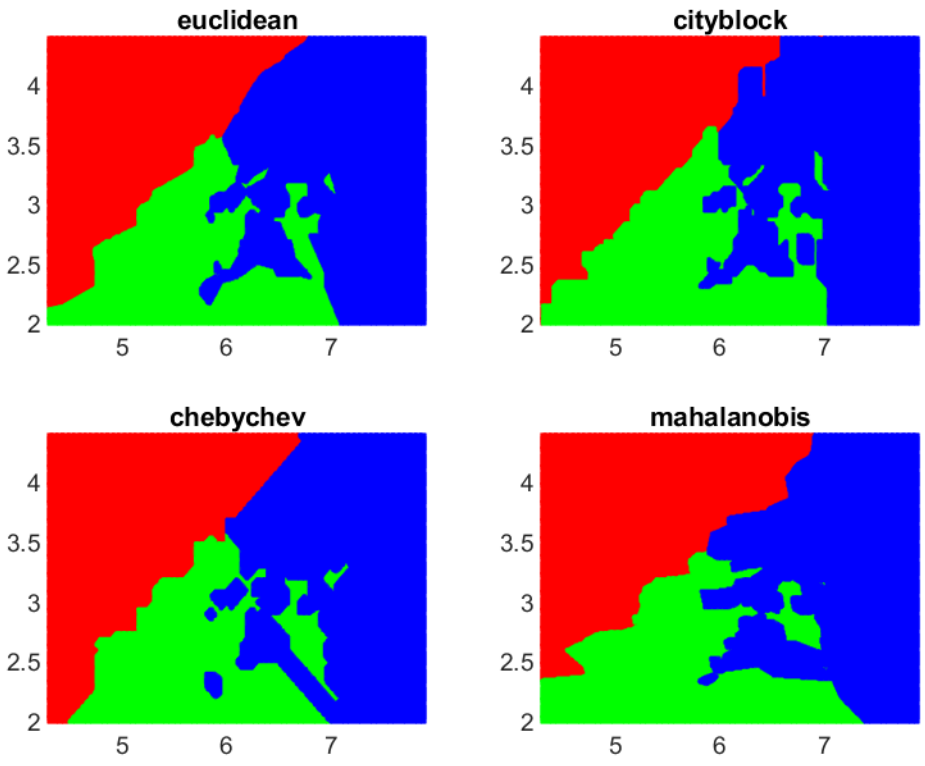

FIGURE 27. KNN with multiple distances

This generates 2 elliptical clusters that are perpendicular to one another.

Matlab has a built in K-Means function which is used in the below code, and this function works with multiple distance metrics. The next code block applies K-Means to the generated data with $k = 2$ and different distance metrics. The results are shown in figure 28.

```
%% Apply K-Means with different distance functions
figure;
dists = ["cityblock", "sqeuclidean", "cosine", "correlation"];
count = 1;
for d=dists
    [idx,C] = kmeans(X,2,'Distance',d);
    subplot(2,2,count);
    hold on
    plot(X(idx==1,1),X(idx==1,2),'r.','MarkerSize',12)
    plot(X(idx==2,1),X(idx==2,2),'b.','MarkerSize',12)
    plot(C(:,1), C(:,2), 'kx', 'MarkerSize',15,'LineWidth',3)
    title(d)
    count = count + 1;
```

```
14 end
```

Line 6 is the main line in the above code, where the K-Means algorithm is actually applied. In this line, pass the number of clusters, $k = 2$, and the distance metric to use, which come from the array declared in line 3. Information on these metrics can be found at the line below[3]. The remainder of this code is plotting commands that allow us to visualize the clusters generated for each distance function.

The clusters can be seen in figure 28. The citiblock and sqeuclidian (squared euclidean) distance functions generated similar clusters, as did the cosine and correlation distance functions. It appears that the top 2 plots generated better clusters on this data set, but even the top 2 plots did not generate the exact same clusters. Further, this data set is relatively simple and only in 2 dimensions. As the dimension increases and the data become more complicated, the results produced by any algorithm will diverge further when different distance metrics are used. This goes to show the importance of selecting the correct distance function.

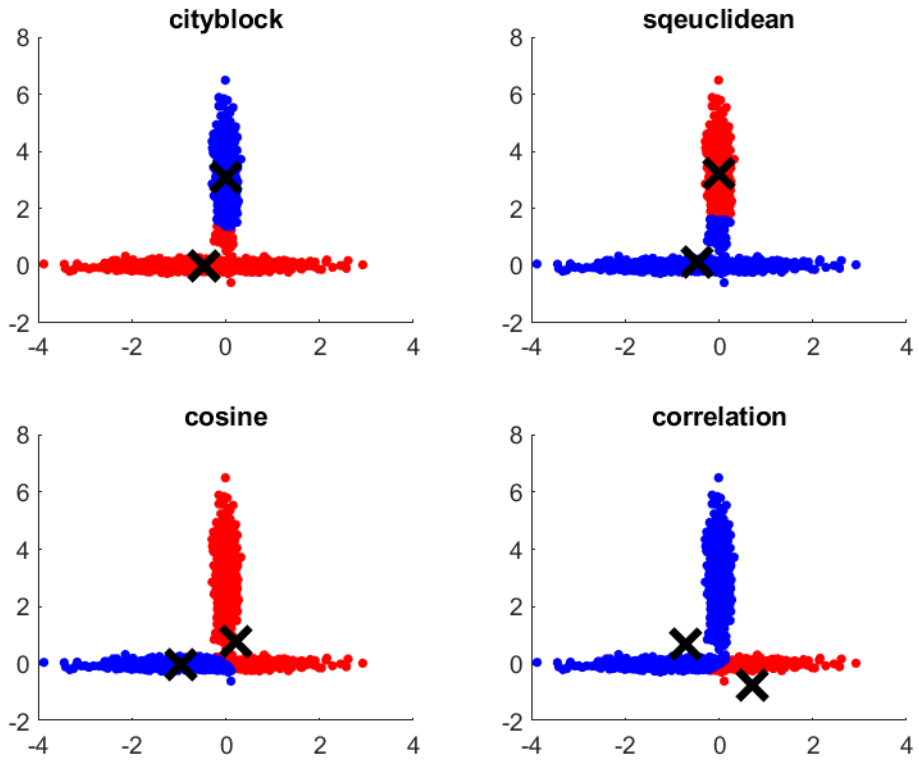

FIGURE 28. K-Means with different distance functions.

[3]For information on distances in the Matlab implementation of M-Means see: https://www.mathworks.com/help/stats/kmeans.html#buefs04-Distance

LECTURE XII

Kernels

The previous section considered how to combine the feature space $\mathcal{X}$ with a distance metric to form a metric space to solve the problem in. This section considers methods of modifying $\mathcal{X}$ to a better feature space $\mathcal{X}'$. There are many reasons this is useful, feature space reduction or data that isn't linearly separable in $\mathcal{X}$, and mapping between feature spaces is a powerful technique that can be applied to many machine learning problems and models.

46. Feature Maps

In the previous section on distances, the Iris data set, which is 4 dimensional, was used for classification. As we saw in our 2D projection, 1 class of flowers is easily, linearly separable, but it was not possible to separate out the other 2 classes with a simple decision boundary. In this 4 dimensional space, this makes it very difficult to correctly classify these 2 types of flowers, even by modifying the distance metric.

Feature maps is a method where data that is difficult to separate in $\mathcal{X}$ is transformed to $\mathcal{X}'$ using an operator φ, which can be any function. The goal of φ is to move data that is difficult for an algorithm to separate in $\mathcal{X}$ into a space that the same algorithms will preform better in. The feature map can be any map, linear or nonlinear, where

$$\varphi : \mathcal{X} \to \mathcal{X}'.$$

There is not always a clear way to define a feature map. In the case of the Iris data, there is not an intuitive feature space where the data would be more separable than it currently is in $\mathbb{R}^4$. As a result, in the remainder of this section, we will see a few general methods for selecting feature maps.

Distance in $\mathcal{X}'$. Given φ, we can define a distance measure of vectors $a, b \in \mathcal{X}$ using their mappings in $\mathcal{X}'$.

$$d_1(a, b) = d_2(\varphi(a), \varphi(b)) \tag{3}$$

In this example, d_1 is a distance function in $\mathcal{X}$, and d_2 is a distance function in $\mathcal{X}'$. It is useful to measure the distance of points a, b in terms of their representations in $\mathcal{X}'$, and this is the main idea behind kernels. The remainder of this section discusses a method for improving the efficiency of calculating this distance as well as the algebra associated with the trick.

Matlab Feature Map. When K-Means was introduced, figure 22 was given to display a failure mode of the algorithm. Since the inner and outter cluster have centers that both are in the inner cluster, K-Means doesn't preform well on this data. However, when a feature map is applied, K-Means can again be successful.

In the below code, a distribution similar to 22 is generated. This distribution is in the space $\mathbb{R}^2$.

```
num_points = 1000;
% Inner circle
X = [randn(num_points,1)*2, randn(num_points,1)*2];
% Outter circle
[x,y] = pol2cart(rand(num_points,1)*2*pi, 10 + rand(num_points
    ,1));
X = [X; x, y];
```

The next line of code applies a feature map from $\mathbb{R}^2$ to $\mathbb{R}^3$. The feature map can be described as

$$\varphi : \begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x \\ y \\ (|x| + |y|)^2 \end{bmatrix}.$$

```
Z = [X(:,1), X(:,2), (abs(X(:,1))+abs(X(:,2))).^2];
```

In the code, the variable Z represents $\mathcal{X}'$. Conveniently, Z also represents the data in $\mathbb{R}^3$ or (x, y, z) coordinates. The data in $\mathcal{X}$ and $\mathcal{X}'$ are shown in figure 29.

When the data is in $\mathcal{X} = \mathbb{R}^2$, the data is not linearly separable, which is problematic for classification, and the center of the outer and inner clusters are both in the inner cluster, which is problematic for clustering. However, in $\mathcal{X}' = \mathbb{R}^3$, the data can be linearly separated by inserting a plane parallel to the $X - Y$ plane at around $Z = 75$, which is beneficial for classification, and the inner and outer clusters have centers separated in the Z dimension, which is beneficial for clustering.

The feature map selected here is a simple feature map and doesn't take long to compute. However, many feature maps are more computationally expensive and less intuitive to define. This motivates the idea of kernels and the kernel trick.

47. Kernel Trick

A distance function that relies on a feature map φ can be decomposed to a function of the feature mappings. Take the case from equation 3 where d_2 is defined in terms of the $L - 2$ norm. This can be rewritten as follows.

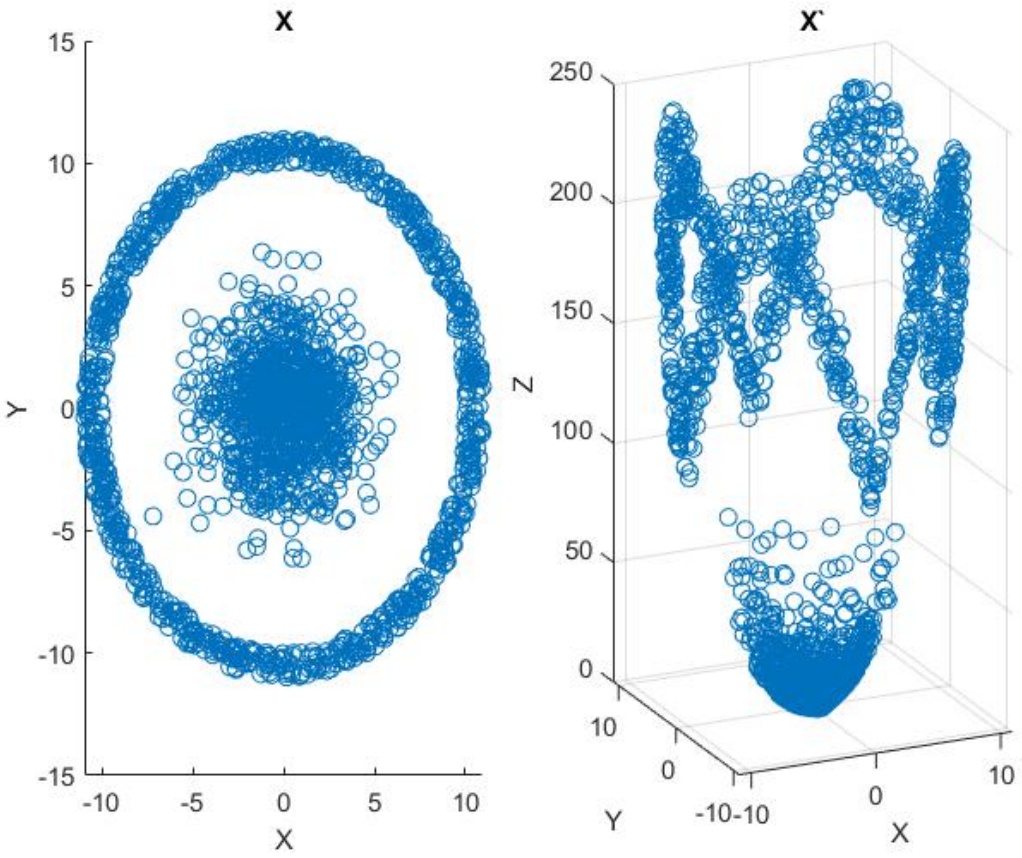

FIGURE 29. Feature Mapped Data

$$\begin{aligned} d_1(a,b) = d_2(a,b) &= \|\varphi(a) - \varphi(b)\|_2 \\ &= (\varphi(a) - \varphi(b)) \cdot (\varphi(a) - \varphi(b)) \\ &= \varphi(a) \cdot \varphi(a) - 2\varphi(a) \cdot \varphi(b) + \varphi(b) \cdot \varphi(b) \end{aligned}$$

From this, we see that the distance between a and b in $\mathcal{X}$ can be written as a function of inner products of $\varphi(a), \varphi(b) \in \mathcal{X}$. Knowing this, it is possible to calculate the inner product without explicitly calculating $\varphi(a)$ or $\varphi(b)$, which is beneficial because computing a feature map can be computationally expensive. This is the idea behind the Kernel trick. Let the Kernel $K(a,b) = \varphi(a) \cdot \varphi(b)$. Then the above function can be written

$$d_1(a,b) = d_2(a,b) = K(a,a) - 2K(a,b) + K(b,b).$$

When $K(a,b)$ is cheap to compute, the distance between a, b is very cheap to compute using the above equation. Often, computing $K(a,b)$ is much faster than computing $\varphi(a)$ and $\varphi(b)$, especially when $\mathcal{X}'$ is a high dimensional space. As a result, it becomes easy

for the new feature space $\mathcal{X}'$ to be very high dimensional, as long as feature vectors are never explicitly mapped into the space. Because $\mathcal{X}'$ is not restricted, there is a greater probability that the data will be linearly separable in large $\mathcal{X}'$.

Popular Kernels. These are some popular Kernels to use. Note, it is not necessary to know φ or $\mathcal{X}'$ to use any of these.

- Gaussian Kernel
$$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$
- Polynomial Kernel
$$K(x,y) = (x \cdot y + r)^d$$
- Sigmoid Kernel
$$K(x,y) = \tanh(\alpha(x \cdot y) + \beta)$$

48. Kernel Algebra

Given that $\mathcal{X} = \mathbb{R}^d$ for some d, the following gives a mathematical definition of a kernel function.

Definition 24 (Mercer's Theorem). A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \cup \{0\}$ is a valid kernel if and only if for $x_1, \ldots, x_n \in \mathcal{X}$ there exist a positive matrix $G \in \mathbb{R}^{n \times n}$ such that $G_{ij} = K(x_i, x_j)$.

From this definition, the following properties can be derived. A kernel algebra closely resembles a ring over a field, with the properties given below. For 2 kernels $K_1(x,y)$ and $K_2(x,y)$, the following 3 combinations represent valid new kernels

(1) $K_1(x,y) + K_2(x,y)$
(2) $K_1(x,y)K_2(x,y)$
(3) $z \times K_1(x,y)$ for $z \in \mathbb{R}$

It is important to note that Kernels have very similar properties to distance functions because they are used in computing distances. Namely, by definition Kernels are positive semidefinate, and Kernels are symmetric. However, note that the distance of a point to itself equals 0 does not apply to Kernels.

Part 4

Machine Learning Models

LECTURE XIII

Mixture Models for Clustering

49. Mixture Models

The most basic clustering algorithm, K-Means, works by assigning points to the cluster with the nearest center. This is problematic when a point is nearly equidistant from the center of multiple clusters. To address this problem, mixture models classify points as having certain probabilities of belonging to a given cluster rather than absolute assignment.

Mathematically, this problem can be formulated with training data $S_n = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$ and k clusters, where each cluster has a probability density function f_k. Given each distribution, the probability of each x_i is a sum of these distributions:

$$P(x_i) = \sum_{j=1}^{K} a_j f_j(x_i|\theta_j).$$

The parameters a_j represent the contribution of each cluster to the data set such that $\sum a_j = 1$. Additionally, f_k represents the distribution of a cluster with parameters θ_k. Generally, it is not a requirement that all clusters have the same distribution (Poisson, Gaussian, etc.), but it does simplify the problem.

50. Gaussian Mixture Models

If all the probability density functions in a mixture model are Gaussian distributions, then this is a Gaussian Mixture Model. In an $n-$dimensional space, a Gaussian distribution is defined

$$\phi_k(x_i|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_k|^{\frac{1}{2}}} \exp\left(\frac{-1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k)\right),$$

where μ_k and Σ_k are the mean and covariance matrix of the kth cluster. A Gaussian Mixture Model has the form

$$P(x_i) = \sum_{j=1}^{K} a_j \phi_k(x_i|\mu_j, \Sigma_j).$$

The model must find the parameters a_i, μ_i, and Σ_i. In order to find these parameters, we define an objective function to optimize.

$$L(a_i, \mu_i, \Sigma_i) = \log \prod_{i=1}^{n} p_i = \log \prod_{i=1}^{n} \left(\sum_{k=1}^{K} a_k \phi_k(x_i | \mu_k, \Sigma_k) \right)$$
$$= \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} a_k \phi_k(x_i | \mu_k, \Sigma_k) \right)$$

Expectation Maximization. Expectation maximization is a technique that can be used to maximize the above optimization function. This technique is often used with complex maximum likelihood optimization, which is what we have in the equation above since it has multiple distributions.

Let $\gamma_{i,k}$ represent the "responsibility" of the kth cluster for the ith data point such that

$$\gamma_{i,k} = \frac{a_k \phi_k(x_i | \mu_k, \Sigma_k)}{p_i}.$$

The responsibility variables are similar to what k means does, γ is a fractional value representing a fractional amount of belonging or contributing to a cluster rather than a binary representation of this, as in k means. If we can estimate $\gamma_{i,k}$ well, then we can estimate the remaining variables well.

Given $\gamma_{i,k}$, which is defined in terms of a_k, μ_k, and Σ_k, it is possible to write definitions for these 3 variables, which we initially wanted to estimated, in terms of $\gamma_{i,k}$.

$$a_k = \frac{1}{n} \sum_{i=1}^{n} \gamma_{i,k}$$
$$\mu_k = \frac{\sum_{i=1}^{n} \gamma_{i,k} x_i}{\sum_{i=1}^{n} \gamma_{i,k}}$$
$$\Sigma_k = \frac{\sum_{i=1}^{n} \gamma_{i,k} (\mu_k - x_i) \cdot (\mu_k - x_i)}{\sum_{i=1}^{n} \gamma_{i,k}}$$

The last definition, for Σ_k, relies on μ_k as well as $\gamma_{i,k}$, but regardless it is possible to redefine all 3 variables in terms of $\gamma_{i,k}$. Given these definition, the algorithm to optimize these variable works by randomly initializing the variables and then redefining them based on each other. This is similar to the K-Means algorithm, except rather than including or excluding a data point from a cluster discretely this is done using continuous distributions. Rather than the assignment step, this method computes responsibilities, and instead of the averaging step, this method computes a_k, μ_k, and Σ_k. The below algorithm gives pseudo code for this optimization.

51. Matlab Implementation

Using the Iris data, the code in this section generates Gaussian mixture models for clustering. In all models, $k = 3$. The Matlab function fitgmdist accepts the arguments

Algorithm 3 Optimization for Gaussian Mixture Model

Randomly initialize $\gamma_{i,k}, a_k, \mu_k$, and Σ_k
while convergence criteria are not met **do**
    for $i \in 1 \dots n$ **do**
        Update $\gamma_{i,k}$ based on a_k, μ_k, and Σ_k
    end for
    for $i \in 1 \dots n$ **do**
        Update a_k, μ_k, and Σ_k based on $\gamma_{i,k}$
    end for
end while

'CovarianceType,' which defines the Σ matrix as either diagonal or full, and 'SharedCovariance,' which determines if all Gaussian distributions must have the same covariance matrix.

```
count = 1;
for i = 1:length(sigma)
    for j = 1:length(sharedCovariance)
        % Fit the model
        model = fitgmdist(X,k,'CovarianceType',sigma{i},'
            SharedCovariance', sharedCovariance{j});
        % Cluster data
        clusters = cluster(model,X);
        % Calculate distances of all points on the grid
        mahalDist = mahal(model,grid);
        subplot(2,2,count);
        % Plot clustered data
        h1 = gscatter(X(:,1),X(:,2),clusters);
        hold on
        % Draw ellipsoids over each GMM component and show
            clustering result.
        for m = 1:k
            idx = mahalDist(:,m)<=threshold;
            Color = h1(m).Color*0.75 - 0.5*(h1(m).Color - 1);
            h2 = plot(X0(idx,1),X0(idx,2),'.','Color',Color,'
                MarkerSize',1);
            uistack(h2,'bottom');
        end
        plot(model.mu(:,1), model.mu(:,2),'kx','LineWidth',2,'
            MarkerSize',10)
        title(sprintf('Sigma is %s\nSharedCovariance = %s',sigma
            {i},scText{j}))
        legend off
```

```
24            count = count + 1;
25        end
26    end
```

The output of this code can be seen in figure 30.

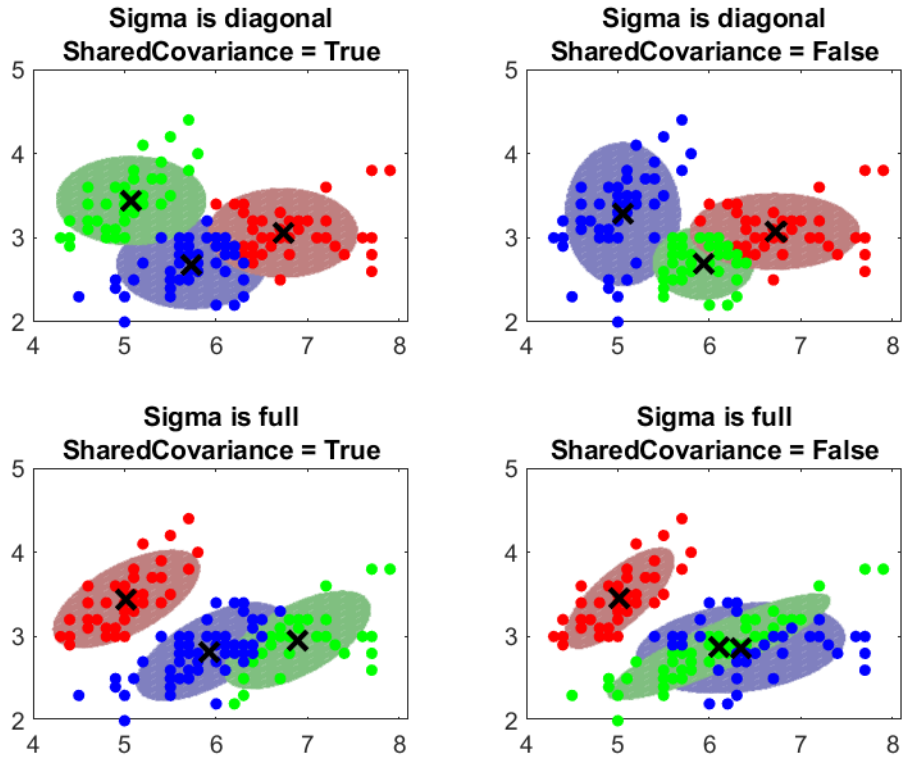

FIGURE 30. Gaussian Mixture Models of Iris Data

The covariance matrix defines the Gaussian distribution. When it is full, the optimization model must solve for more parameters. Additionally, when each distribution has its own covariance matrix, the optimization problem is larger. The impact of having each distribution have the same or different covariance matrices can be seen between the left and right plots. In the plots on the left, all 3 distributions have the same shape. However, in the plots on the right, all 3 distributions can have different shapes.

LECTURE XIV

Naive Bayes

Naive Bayes is a supervised learning technique for classification problems. Given training data $S_n = \{(x_i, y_i)\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, the idea behind Naive Bayes is to estimate the posterior probability $P(y_i|x)$ to classify a data point x into the correct class. This probability can be rewritten a number of ways.

$$P(y_i|x) = \frac{P(y_i)P(x|y_i)}{P(x)} = \frac{P(x, y_i)}{P(x)}$$

The goal is to maximize with respect to i the expression above. Since $P(x)$ has no impact on this maximization problem, the following is an equivalent problem.

$$y = \arg\max_i P(i)P(y_i|x) = \arg\max_i P(x, y_i).$$

52. Bayesian Probability

It is not simple to compute $P(x, y)$. However, assuming conditional independence in $\mathcal{X}$ reformulates the probability expression as follows.

$$P(x, y) = P(x_1|y)P(x_2|y)\dots P(x_n|y)P(y),$$

where x_i is the ith element in the feature vector x. In this form, each term $P(x_i|y)$ can be calculated by examining the data, so it is possible to calculate $P(x, y)$. This reduces the initial Naive Bayes problem to the form

$$y = \arg\max_i P(x_1|y_i)P(x_2|y_i)\dots P(x_n|y_i)P(y_i).$$

In this final form, the Naive Bayes expression is both simple and fast to compute, making it a very powerful machine learning technique.

Assumption of Conditional Independence. While the assumption of conditional independence is necessary, it is a major drawback of Naive Bayes. Naive Bayes doesn't work well for complex classification problems where there is a high degree of correlation between variables in the feature vector or there is excessive noise. For this reason, the performance of Naive Bayes significantly increases when coupled with effective feature selection and reduction methods.

53. Matlab Implementation

Matlab has preimplemented Naive Bayes classifier. In this section, the model is trained on 80% of the Iris data. The decision boundary is plotted, and predictions are made on the remaining 20%. This code is used to partition the data.

```
c = cvpartition(length(X),'Holdout',0.2);
idxTrain = c.training;
X_train = X(idxTrain,:);
y_train = y(idxTrain,:);
X_test = X(~idxTrain,:);
y_test = y(~idxTrain,:);
```

The model is trained using the following line of code.

```
model = fitcnb(X_train, y_train,'ClassNames',{'setosa','
    versicolor','virginica'});
```

The decision boundary and the predictions on the held out data are shown in figure 31.

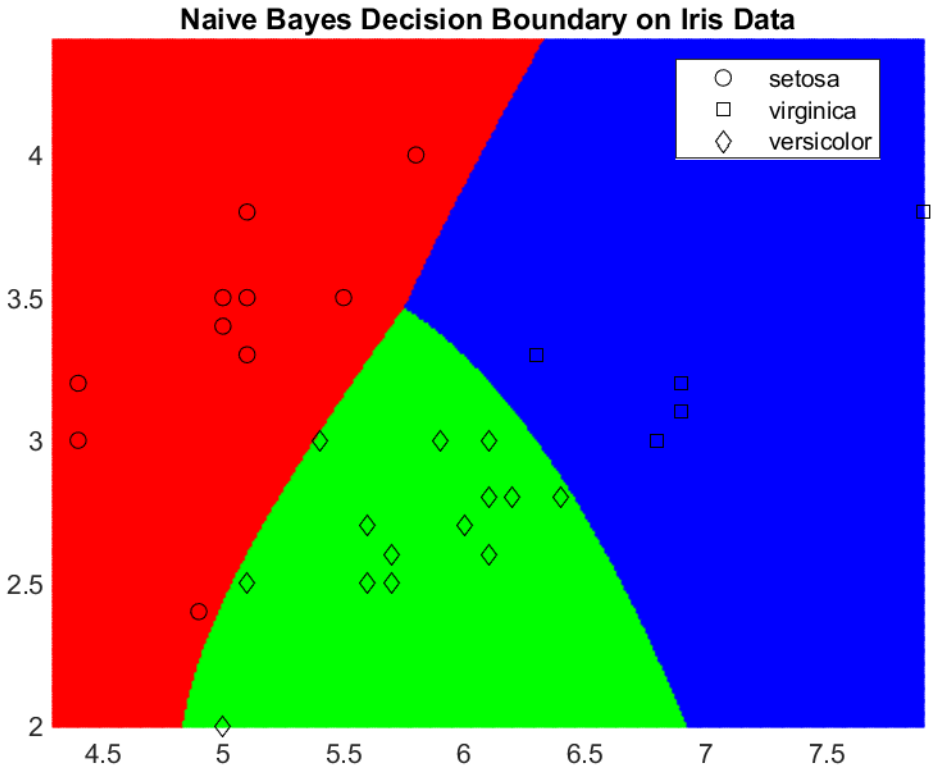

FIGURE 31. Naive Bayes Classifier decision boundary on Iris data

LECTURE XV

Support Vector Machines

54. Linear Classifiers and the Perceptron Algorithm

Consider $S_n = \{x_i, y_i\}$ where $\mathcal{Y}$ is a binary value of -1 or 1. S_n is linearly separable if there exist a hyper plane $h \in \mathcal{X}$ such that all feature vectors with label -1 are on one side of the plane and all feature vectors with label 1 are on the other side of the plane.

Let $\mathcal{H}$ be the set of all hyperplanes in $\mathcal{X}$. If $\mathcal{X} = \mathbb{R}^n$, then $\mathcal{H} = \mathbb{R}^{n-1}$, if we restrict that all planes in $\mathcal{H}$ must pass through the origin. If the data is linearly separable with $h \in \mathcal{H}$, then S_n is linearly separable through the origin.

For S_n linearly separated through the origin by h, let θ be the normal vector of h. Then the label of each $x_i \in S_n$ is given with the sign of $x_i \cdot \theta$. From this, it follows that the sign of $y_i(x_i \cdot \theta)$ is positive. Using this fact, the perceptron algorithm is able to search for a separating plane in $\mathcal{X}$.

The perceptron algorithm, which has pseudo code below, searches for the separating plane using the following process. The outer loop of the algorithm, beginning on line 2, loops while there exist $x_i \in S_n$ that is misclassified. The inner loop, beginning on line 3, iterates over each labeled feature vector in S_n. The condition on line 4 determines if the current θ correctly classifies the labeled feature vector. If the feature vector is misclassified, θ is updated in the following line by adding the term $y_i \times x_i$. This increment of θ "pushes" the hyperplane in the right direction to correctly classify x_i in the next iteration of the algorithm.

Algorithm 4 Perceptron Algorithm

```
1: Initialize θ = 0.
2: while there exist a misclassified feature vector do
3:     for i = 1, ..., n do
4:         if y_i × (θ · x_i) ≤ 0 then        ▷ This is true when x_i is misclassified by θ
5:             θ = θ + y_i × x_i               ▷ θ moves in the "right direction"
6:         end if
7:     end for
8: end while
```

The perceptron algorithm is very simple, but it is a fundamental building block of models such as support vector machines and neural networks.

Perceptron Update. To show that the update $\theta = \theta + y_i \times x_i$ pushes θ close to correctly classifying x_i, consider the following.

$$\begin{aligned} y_i \times (\theta_{k+1} \cdot x_i) &= y_i \times ((\theta_k + y_i \times x_i) \cdot x_i) \\ &= y_i \times ((\theta_k \cdot x_i) + (y_i \times x_i \cdot x_i)) \\ &= y_i \times (\theta_k \cdot x_i) + |x_i|_2^2 \\ y_i \times (\theta_{k+1} \cdot x_i) &\geq y_i \times (\theta_k \cdot x_i) \end{aligned}$$

This shows that even if the classifier incorrectly classifies x_i after the updated θ, it will still be closer to 0 than it was before because the term $|x_i||_2^2$ was added. $|x_i||_2^2$ will be positive for all x_i. This makes the answer "less negative," which is the "right direction" since $\theta \cdot x_i$ should be positive.

The Perceptron with Offset. The pseudo code above only considers $\mathcal{H}$ as being hyperplanes through the origin, but this can be extended to all hyperplanes in $\mathcal{X}$ by adding a term b, the offset from the origin. This means a vector is correctly classified when $y_i(\theta \cdot x_i + b) > 0$. For a misclassified point in this case, the update to θ remains the same, and b is updated as $b_{k+1} = b_k + y_i$.

Often the offset term can be incorporated into the algorithm without explicitly adding in a term b. Rather, let

$$\theta' = \begin{bmatrix} \theta \\ b \end{bmatrix} \text{ and } x_i' = \begin{bmatrix} x_i \\ 1 \end{bmatrix} \forall i.$$

In this case, $y_i(\theta' \cdot x_i') > 0$ is equivalent to $y_i(\theta \cdot x_i + b) > 0$.

Limitation. Another limitation of the Perceptron Algorithm is that it can only works with linearly separable data. If the data is not linearly separable, then the algorithm will not converge. This can be accounted for with a few different schemes. Two common solutions would be to have the algorithm could terminate after a set number of updates to θ or have it terminate when the updates to θ are sufficiently small. Another modification to the algorithm is using kernels to map S_n into a space where the data is linearly separable.

Finally, the size of θ is unbounded. In this case, the size of θ is representative of the complexity of the model. It can be helpful to limit the complexity of the model so that it does not over fit the data. Each of these limitations to the Perceptron Algorithm is also a limitation of Support Vector Machines.

55. Support Vector Machine

The set of the Support Vector Machine problem is very similar to the Perceptron Algorithm. Given binary labeled S_n the goal is to find a separating plane $h \in \mathcal{X}$ that separates the 2 classes of data.

Consider figure 32. In this case the data are in $\mathbb{R}^2$, and the data points, represented as + or - are linearly separable. The thick black line is h, the hyper plane, and the 2 circled data points on the dashed lines are support vectors. The distance between the

support vectors and θ is $\frac{2}{\|\theta\|}$ and called the margin. The goal of support vector machines is the maximize this margin while correctly classifying all points. Maximizing the margin allows the classifier to generalize to new data well.

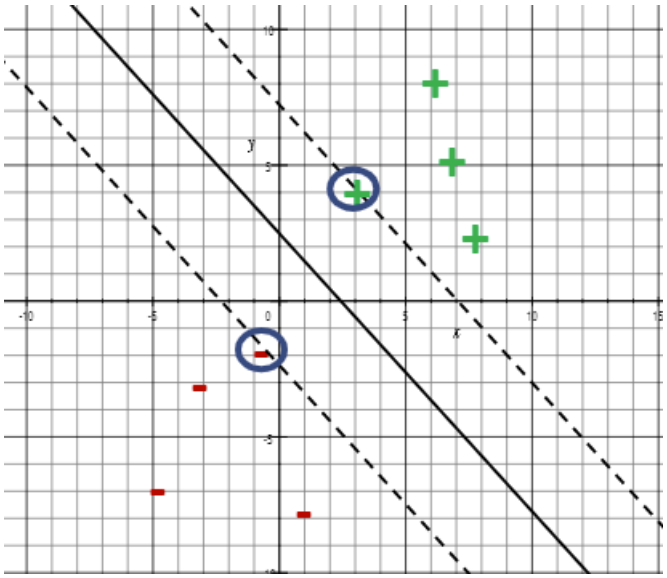

FIGURE 32. Support Vector Machine

Often, it is not possible to correctly classify all data points, so Support Vector Machines allow for some misclassification by accepting some error if on the whole the classifier does well.

56. Primal Formulation

Since support vector machines are used for binary classification, all labels y_i can be thought of as ± 1. Additionally, rather than including the offset of the hyper plane[1] in θ, it can be included as an additional term b, so that θ is the coefficients of the variables in the plane equation. Since the problem has a same set up, similar to the Perceptron Algorithm a point is correctly classified when

$$y_i(\theta \cdot x_i + b) \geq 0.$$

DEFINITION 25. The primal formulation of support vector machines is the optimization problem

$$\min_{\theta,b} \|\theta\| \text{ such that } y_i(\theta \cdot x_i + b) \geq 0.$$

This is the most basic interpretation of a hard margin Support Vector Machine. It is a hard margin because and it requires that the all data points are correctly classified. An

[1]The offset of the hyper plane for a support vector machine is similar to the bias of a neuron in a neural network. If the offset were 0, then the hyper plane must pass through the origin. By allowing the offset to be nonzero, $\mathcal{H}$ is the space of all hyper planes in $\mathbb{R}^n$

important feature of Support Vector Machines over the Perceptron Algorithm is that it minimizes θ. A limitation of the Perceptron Algorithm is that allowing θ to increase in size with out constraining it increase the model's complexity, which limits its ability to generalize. Support Vector Machines prevent this by minimizing θ.

Soft Margin Support Vector Machines. The above formulation works well for data that can be entirely separated, but if the data isn't separable, we can redefine the problem by allowing each x_i to be misclassified by the amount ξ_i, which are called slack variables. When this is done, the optimization problem becomes

$$\min_{\theta,\xi_i} \left(\frac{\|\theta\|^2}{2} + C \sum \xi_i \right) \text{ such that } y_i \times (\theta \cdot x_i) \geq m - \xi_i.$$

Using the above equation, the support vector machine can classify data that is not linearly separable. The hyperparameter C reflects the importance of a misclassified point. As C goes to infinity, the minimization equation for a soft margin support vector machine becomes equivalent to a hard margin Support Vector Machine.

57. Dual Formulation

The primal formulation gives an intuitive definition of Support Vector Machines in terms of finding a separating plane defined by θ. However, there are many equivalent definitions of. The dual formulation is significant because it can be solved using Lagrange multipliers, an important concept in optimization.

Definition 26. The dual formulation of support vector machines is the optimization probeml

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \dot{x}_j)$$

LECTURE XVI

Neural Networks

Phineas Gage was an American construction worker famous for surviving a railroad spike removing 1/3 of his brain. After a few weeks of recovery, he was able to return to work. He did, however, have significant behavioral changes, including leaving his wife and family. His survival and the fact that his brain could function after a chunk was removed is the motivating idea behind connectionist methods of machine learning and caused people to rethink how the brain works. Rather than thinking of the brain like a CPU, Phineas' survival suggest that the brain is composed of a series of smaller, simple neurons that make up the function of the brain.

58. Biological vs. Artificial Neurons

Figure 33 shows on top a diagram of a neuron in our brain and on the bottom gives a crude mathematical formulation for a neuron. Biologically, the neuron on the left receives electrochemical inputs from its dendrites (the branches that extend outward from the main cell). If the electrochemical gradient from all dendrites is great enough, then the neuron will send its own electrochemical signal on its axon. This is seen in the figure traveling from the neuron on the left over the center axon towards the output branches, which become the input signals for other neurons.

The mathematical model below works in a similar way. The neuron, which can be represented by the red dot, accepts inputs $x_i w_i$ from n input sources. This step is similar to the electrocheical signals being received by the dendrites. These inputs are summed up, and put through an activation function. A biological neuron has a nondifferentiable activation function (it sends a signal if the input is great enough but doesn't send any signal if it is not great enough, so it is discontinuous). Although deviating from the biological foundations, it is helpful to a differentiable activation function. Common activation functions include

- Binary Sigmoid (exponential sigmoid)

$$f(x) = \frac{1}{1 + \exp(-\sigma x)}$$

- Bipolar Sigmoid (atan)

$$f(x) = \frac{1 - \exp(-\sigma x)}{1 + \exp(-\sigma x)}$$

Both activation functions are given a hyper parameter σ. The exponential sigmoid function has a range of (0,1) and the atan activation function has a range of (-1,1).

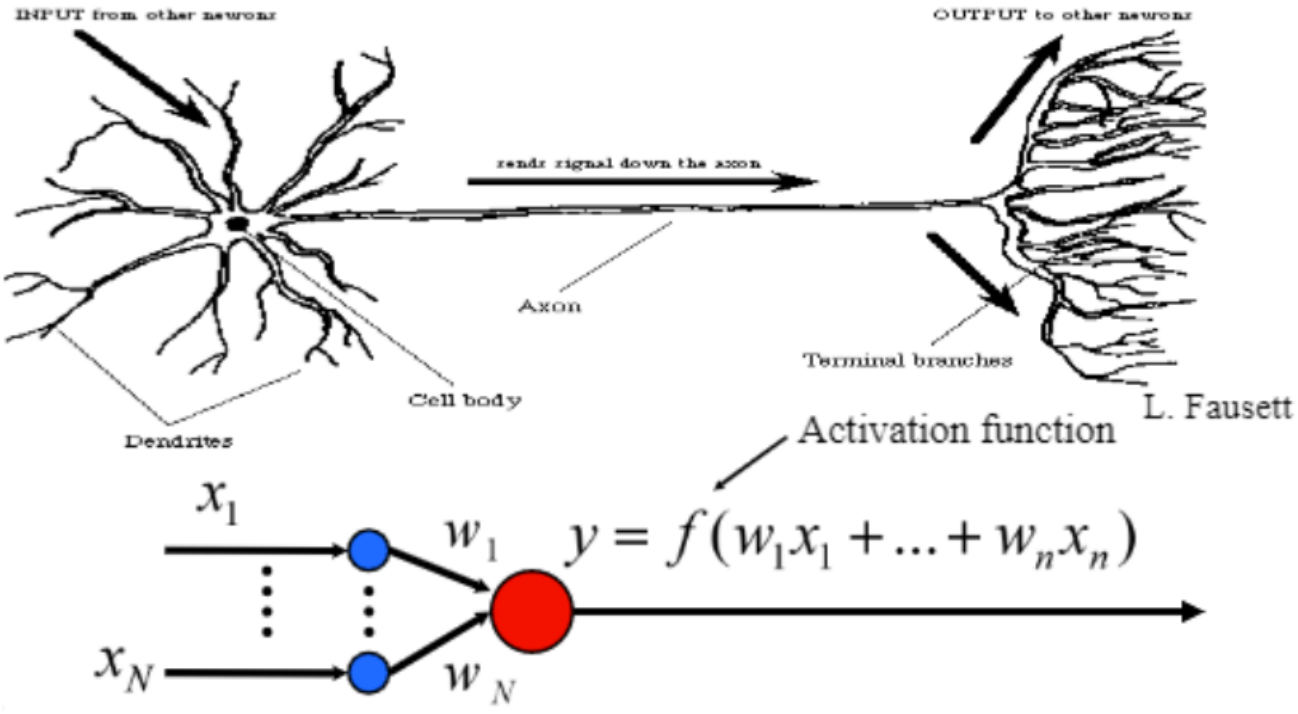

FIGURE 33. Biological and Mathematical Neurons

59. A Mathematical Formulation of the Neuron

While the above section describes the biological intuition behind neurons, it is important to have a rigorous mathematical definition as well.

DEFINITION 27. A neuron is a function $f : \mathbb{R}^n \to \mathbb{R}$ parameterized with a weight vector $w \in \mathbb{R}^{n+1}$ and an activation function $a : \mathbb{R} \to \mathbb{R}$. For $x \in \mathbb{R}^n$, the output of a neuron is defined

$$f(x) = a\left(\begin{pmatrix} x \\ 1 \end{pmatrix} \cdot w\right)$$

In this formulation, the neuron begins by computing the dot product of w and $\begin{pmatrix} x \\ 1 \end{pmatrix}$. Note that 1 is added to the bottom of this vector because $x \in \mathbb{R}^n$ and $w \in \mathbb{R}^{n+1}$. The term in w that is multiplied with 1 when taking the dot product is referred to as the bias of the neuron. This bias is similar to adding an additional input term to the neuron, and it is important because it allows the neuron to define a linear decision boundary that is not required to pass through the origin. In essence, a neuron defines a linear decision boundary. If the decision boundary is only a function of the inputs, then the hyper plane decision boundary must pass thought the origin. This is the case for all $y \cdot x = 0$ hyper planes, where $x, y \in \mathbb{R}^n$. However, if we allow this additional input, then the hyper plane

is defined $y \cdot x = b$, where b is the bias. Adding in a bias allows the hyper plane to be any possible plane in the space $\mathbb{R}^n$, even those that do not pass through the origin.

60. From a Neuron to a Brain

Since a whole chunk of Phineas' brain was removed, we know that a signal neuron isn't that important to the functionality of our brain, but our brain and connectionist machine learning models rely on passing the output from 1 neuron into another neuron. Figure 34 shows how it is possible to pass inputs and outputs between neurons to create a neural network. The user is able to define the number of inputs, the number of hidden layers, and the number of outputs. The goal of this neural network is to create a map from the input space to the output space.

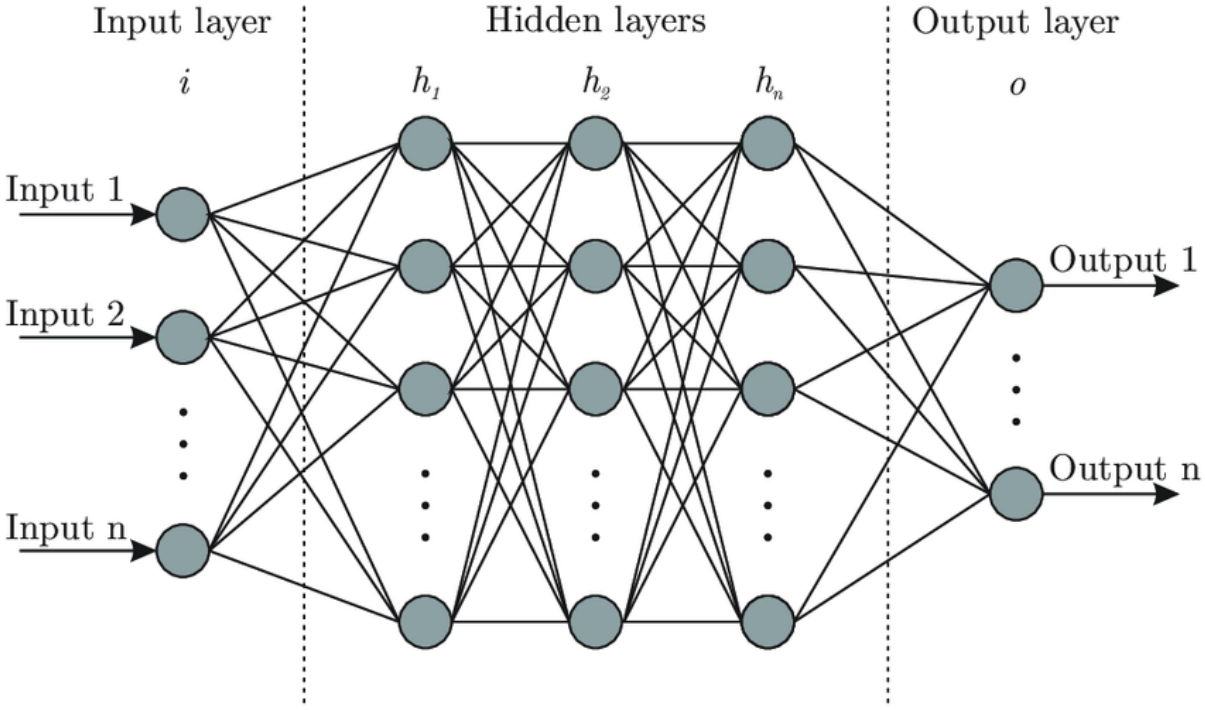

FIGURE 34. Neural Network

Parameterizing a neural network is not trivial. Defining the architecture along (choosing the number of inputs, outputs, hidden layers, neurons per hidden layer, etc.) can be done an infinite number of ways. As the number of neurons or layers increases, the number of parameters in the neural network increases. In order to have a simple model and reduce over fitting, it is beneficial to minimize the number of parameters in the network. As a rule of thumb, it is optimal to have at least 10X the number of examples in the training data as there are parameters in the model.

Conveniently, the computationally expensive part of the output of a neuron is taking the dot product between 2 neurons. In the transition from a single neuron to a neural

network, as layers are formed, the passing the output between layers changes this dot product to a matrix multiplications, which can run in $O(n^3)$.

61. Training Neural Networks

As we have seen, neural network is no more then a set of neurons, each of which is parameterized with a weight vector. The goal of training a neural network is to optimize all weights in the network. Generally, given a weight w

$$w_{new} = w_{old} + \Delta w_y$$

The term Δw_y represents the amount that y, the output in the label space, depends on the given weight w.

$$\Delta w_y = \alpha \times \text{Error Rate}$$

The Error Rate is the dependency of the output on w, calculated as a partial derivative, and α is the learning rate that is used to control the amount of adjustment of w at each iteration. Generally, α should be a monotonically decreasing function because the updates to each weight should become smaller as they approach optimal values.

Rather than updating the weights after passing through each training example, which is commonly done with other machine learning models, the weights of a neural network are updated after each epoch. An epoch represents a group of training examples that are passed through the neural network.

Backpropagation Algorithm. This is the most common algorithm for training a neural network. The idea behind this algorithm is to first calculate the error of the output nodes and then calculate the error in each previous layer assuming the error of the following layer is known. In essence, this algorithm relies on the chain rule to compute the partial derivatives of the output with respect to each weight in the network.

Consider figure 34. for example, since each training input has an associated output, it is possible to compute the errors in the output layer (the partial derivatives for each weight). Then since the output layer values are now known, it is possible to compute the partial derivatives of the weights associated with layer h_n. Once h_n is known, it can be done for h_2 and so on.

This algorithm is a gradient descent algorithm being performed on the neural network function, which is a complex function with many, many parameters. This leads to 2 main problems:

- Like all gradient descent algorithms, backpropagation is not guaranteed to converge to a global optimum. Instead, the algorithm may converge to a local optimum, which there will be many of since the network contains so many parameters.

- If the network has too many layers, then estimating the error at the first few networks in the layer will be worse off. The accuracy of the estimation decreases with each layer.

Similar to KNN, the many versions of the backpropagation algorithm exist, all with subtle differences designed to overcome these potential issues.

62. Uses of Neural Networks

Time Series Analysis. Neural networks have been used in time series problems including weather forcasts, market predictions, and biomedical applications. Time series signals are especially powerful when looking at the time series of multiple signals together.

Clustering. A major problem with KNN and K-Means is that they both require knowledge of k the number of clusters prior to running the algorithms. The most popular type of unsupervised neural network is Kohonen Self-Organizing Maps (K Maps)

LECTURE XVII

Decision Trees

A major disadvantage of support vector machines and previous classifiers that have been discussed is the lack of intuition humans have behind a trained model. As we have seen, it is easy to describe why a classifier should work and the mathematical formulation for tackling the problem. However, once the classifier is trained, there is no simple mechanism for understanding the values in θ. This motivates decision trees, a type of classifier designed to be easy for both computers and humans to understand.

63. Decision Trees

Decision trees work by recursively splitting data based on elements of the feature vectors that contain the most information. Based on Information Theory, we can define a simple model which is readily understandable by both humans and machines, even after it is done training.

Definition 28. A decision tree is a tree structure built from training data where each node denotes a test condition, the branches are defined based on the results of the tests, and the terminal nodes of the tree predict labels for data.

The test condition is defined using information theoretic criteria. Since information theory deals with both ordinal and categorical data, this allows decision trees to use data categorical or binary data, something that is not possible with connectionist methods.

64. Example

Consider the problem of predicting if a person is healthy based on:

- Age
- If they eat pizza
- If they exercise

In this problem, age can be any real number, if they eat pizza and if they exercise are binary variables, True/False.

We could use the decision tree in figure 35 as a model for making these predictions. To illustrate how the tree works, consider a person described by variables A, P, E, corresponding to age, pizza consumption, and exercise. Beginning at the root node, if $A < 30$, then the decision tree follows the left branch to the next node. The new test asks if $P =$True. Since this is a terminal node, the output of this test results in a prediction rather than a new test node. If $P =$True, then the model predicts the person is unfit.

Otherwise, the model predicts the person is fit. However, returning to the root node, if $A \not< 30$, then the decision tree follows the right branch. The next node asks if $E =$True. Similarly, this is a terminal node that will make a prediction. If $E =$True, then the model predicts the person is fit. Otherwise, the model predicts the person is unfit.

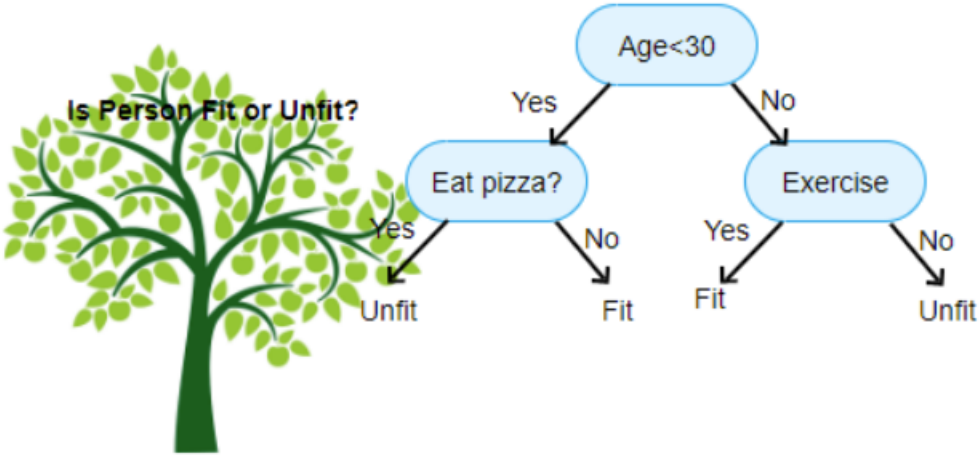

FIGURE 35. Health Predictor Decision Tree

In all cases the value of A is used by the model, but note that not all variables are considered for every person. If $A < 30$, then the value of E is not considered. Similarly, if $A \not< 30$, then the value of P is not considered. There is no general rule for decision trees to say which variables are always relevant and which are not. This is the result of how the decision tree is built, which is considered in the next sections.

A key feature of this decision tree is that we understand how the classification process works. θ for a SVM and the weights and biases for a neural network do not have a meaningful interpretations. In contrast, a decision tree provides insight into how predictions are made.

65. Iterative Dichotomiser 3 (ID3)

For decision trees, training is the process of building the tree based on training data. This requires recursively calculating the optimal test at each node in the tree. This is done by selecting the feature that has the largest information gain.

Since building trees is a recursive process, it is important to consider the base cases of the recursion. These base cases construct terminal nodes of the tree, so they predict the labels of the data. These are the 3 base cases:

(1) If all feature vectors have the same label, then the node predicts the common label.
(2) If there are no remaining attributes to split the data on, then the node predicts the most common label.

(3) If there are no feature vectors in the node, then the node predicts the most common label seen by the node's parent.

The ID3 algorithm, which has pseudo code below executes the described process. An interesting point about this algorithm is that only 2 of the recursive cases are checked at the onset of the algorithm, and the 3rd base case is actually checked prior to the algorithm having a recursive call to it.

Algorithm 5 ID3

Require: feature_vectors, labels, attributes
R is a tree node
if all labels are the same **then** ▷ Base case 1
    $R - label$ =labels(1)
else if length(attributes)== 0 **then** ▷ Base case 2
    $R - label$ =mode(labels)
else
    A =attribute in the feature vector that maximizes information gain
    Remove A from attributes
    $R - attribute = A$
    for each value v of A **do**
        Add branch B to R that corresponds with $A = v$.
        split_feature_vectors = subset of feature_vectors where $A = v$
        if length(split_feature_vectors)== 0 **then** ▷ Base case 3
            $Child - label$ =mode(labels) ▷ create a child node
        else
            split_labels are the labels corresponding with split_feature_vectors
            $Child = ID3$(split_feature_vectors, split_labels, attributes)
        end if
        add $Child$ as the node at the end of B.
    end for
end if
Return R

The ID3 is the simplest algorithm for training decision trees. C4.5 and C5 are extensions of the ID3 algorithm that are also commonly used.

66. Pruning

Decision trees are prone to over fitting. The complexity of the decision tree model is a function of the depth of the tree, and similar to all models, as the complexity of a model increases model over fit the training data.

To prevent this, decision trees are pruned to limit the size of the tree. Pruning can occur based on the following heuristics.

Maximum Depth. Setting a maximum depth for decision trees prevents them from growing endlessly. Since the complexity of decision trees is a function of the depth of the tree, this directly limits the complexity of the model. However, this method can be detrimental by not allowing the model to be complex enough. While setting a maximum depth may prevent over fitting, it may also prevent the model from learning all significant aspects of the data.

Minimum Sample Size. A decision tree can be pruned so that every node is based on at least n many samples in the training data. By preventing nodes in the decision tree from representing small subsets of the training data (subsets with less than n samples), the model should only capture aspects of the data that generalize beyond the training data.

Significance Test. Significance pruning prevents a node in a decision tree from growing when the information criteria doesn't reach a threshold value. Each node in the decision tree represents a test. Returning to the example, the first node's test is if the person is under the age of 30. These tests are formalized using information theoretic criteria. ID3 uses information gain, but Gini Index[1] and other measures can be used as well. When the maximum information gain from splitting on a feature at a given node is small, the model will not learn more from building a subtree at that node, so it makes sense for the node to be terminal.

[1]Gini Index is a measure of the probability of a random sample being incorrectly classified. It is defined

$$Gini = 1 - \sum p_i^2,$$

where p_i is the probability of being classified into class i.

LECTURE XVIII

Random Forests

The goal of a Random Forest is to improve Decision Trees by decreasing the variance with out increasing the bias. Generally, this is the focus of ensemble methods, which combine multiple base models to make predictions. Decision Trees are the base model for Random Forests.

67. Bootstrapping

Bootstrapping is a sampling technique used to generate more data to train models on from our initial data. It is based on

68. Bootstrap Sampling and Bagging

Given that we only have 1 set of training data, we need to devise a scheme in for creating multiple different classifiers from the same data. (If the classifiers are the same, then averaging them does nothing). Bootstrap sampling is a method of sampling the data set S_n without replacement to form a new set $S'_n \subseteq S_n$.

Bagging is the method of bootstrap aggregating. We can bootstrap S_n k times, creating $S'_{n_1}, \ldots, S'_{n_k}$. From these new data sets, we can train classifiers $C_1, \ldots, C_k$. Given a feature vector x in the test set, we can run $C_1, \ldots, C_k$ on x and then take the most seen result from all k classifiers.

This methods relies on 2 major assumptions:

- If the classifiers $C_1, \ldots, C_k$ are not independent of one another, then there is no use in creating k of them an taking the average. This need for the independence drives the bootstrapping sampling methodology, which is designed to create random subsets of S_n so that the classifiers will be independent of one another.
- In order for bagging to be effective the average "correctness" of $C_1, \ldots, C_k$ must be greater than randomly guessing. For the binary case, if on average a classifier is correct at a rate of 0.6 and incorrect at a rate of 0.4, then having k similar yet independent classifiers gives a probability of being incorrect at a rate of 0.4^k. As $k \to \infty$, the probability of being wrong goes to 0.

In practice, it is not always possible to have k perfectly independent classifiers, but there are numerous methods for increasing the randomness that is used to create these k classifiers. Decision trees are one such method.

69. Random Forests

The ideas of bootstrapping and bagging can be applied throughout machine learning, but a random forest is designed specifically to use these ideas with decision trees. As discussed in the previous section, having independent classifiers $C_1, \ldots, C_k$ is one of the key limitations in bagging. To further the independence between decision trees in a random forest, in addition to selecting different subsets S_{n_i} to train each C_i, a random set of m features is selected that the classifier can split data on.

Appendices

LECTURE A

Code

1. Signals

```
%% Start
close all;
clear all;
clc;

%% Fast Fourier Transform
Fs = 100;          % sampling frequency
L = 4;             % length of the signal in seconds
T = 1/Fs;          % the time between each sample
t = 0:T:L-T;       % time vector

% Construct a signal
x1 = 4*sin(2*pi*4*t);
x2 = 2*sin(2*pi*20*t);
x = x1 + x2;

% Uncomment these lines to add noise to your signal
% noise = 10;
% x = x + noise * rand(size(x)) - (noise / 2); % Add mean 0
    noise to the signal

% Fast Fourier Transform and shift the Fourier spectrum
Y = fft(x);
yshift = fftshift(Y);
magnitude_yshift = abs(yshift);
fshift = (-length(Y)/2:length(Y)/2-1)*Fs/(length(Y));

% Plots
subplot(4,1,1);
plot(x1(1:4*Fs));
xlabel('Time');
ylabel('x1(t)');
```

```
title('Low Frequency Component');

subplot(4,1,2);
plot(x2(1:4*Fs));
xlabel('Time');
ylabel('x2(t)');
title('High Frequency Component');

subplot(4,1,3);
plot(x(1:4*Fs));
xlabel('Time');
ylabel('x(t)');
title('Signal');

subplot(4,1,4);
plot(fshift,magnitude_yshift)
axis([0, 25, 0, 1000])
xlabel('Frequency');
ylabel('Intensity');
title('Frequency Spectrum');
%close all

%% Filtering
order = 3;
[b_low,a_low] = butter(order,10/Fs, 'low');

order = 5;
[b_high,a_high] = butter(order,20/Fs,'high');

x_low = filter(b_low,a_low,x);
x_high = filter(b_high,a_high,x);

figure;
freqz(b_low,a_low);
title("Low Pass Filter")

figure;
freqz(b_high,a_high);
title("High Pass Filter")

figure;
subplot(2,1,1);
```

```
74 plot(t,x_low)
75 title("Filtered Low Frequency Component")
76 subplot(2,1,2);
77 plot(t,x_high)
78 title("Filtered High Frequency Component")
79
80 % close all;
81 clear all;
82
83 %% Fractals
84 load('data\ecg.mat')
85
86 T = 1/Fs;                    % the time between each sample
87 t = 0:T:length(ecg)-T;       % time vector
88 plot(t(4*Fs:8*Fs),ecg(4*Fs:8*Fs));
89 xlabel("Time (sec)")
90 ylabel("Amplitude");
91 title("ECG Signal");
92
93
94 figure;
95 cwt(ecg(4*Fs:8*Fs),1:75,'db4','plot');
96 %% Mother Wavelet
97 [phi,psi,xval] = wavefun('db4',10);
98
99 figure
100 plot(xval,psi)
101 title("db4 Mother Wavelet")
102 xlabel("Time")
103 ylabel("Amplitude")
104
105 %% Wavelet Transform
106 % Wavelet Decomposition
107 [C,L] = wavedec(ecg, 4, 'db4');
108
109 % Reconstructed signal
110 rec = waverec(C,L,'db4');
111
112 % Remove different levels of the wavelet (QMF) decomposition by
113 % setting their coefficients to 0
114 C(1:L(1)+L(2)+L(3)+L(4)) = 0;
115 C(L(1)+L(2)+L(3)+L(4)+L(5)+1:end) = 0;
```

```
116
117 % Reconstructed signal
118 level_5 = waverec(C,L,'db4');
119
120 figure;
121 subplot(3,1,1)
122 plot(t(1:10*Fs),ecg(1:10*Fs));
123 xlabel('Time (sec)');
124 ylabel('Amplitude');
125 title('Origninal ECG Signal')
126 subplot(3,1,2)
127 plot(t(1:10*Fs),rec(1:10*Fs));
128 xlabel('Time (sec)');
129 ylabel('Amplitude');
130 title('Wavelet Decomposition')
131 subplot(3,1,3)
132 plot(t(1:10*Fs),level_5(1:10*Fs));
133 xlabel('Time (sec)');
134 ylabel('Amplitude');
135 title('d5 of the Wavelet Decomposition')
```

2. Fundamentals of Machine Learning

```
1 %% KNN Classification
2 % Reference: https://www.mathworks.com/help/stats/visualize-
     decision-surfaces-for-different-classifiers.html
3 %% Load the Iris data set
4 load fisheriris
5 X = meas(:,1:2);
6 y = categorical(species);
7 labels = categories(y);
8
9 gscatter(X(:,1),X(:,2),species,'rgb','osd');
10 xlabel('Sepal length');
11 ylabel('Sepal width');
12
13 %% Generate a mesh grid
14 x1range = min(X(:,1)):.01:max(X(:,1));
15 x2range = min(X(:,2)):.01:max(X(:,2));
16 [xx1, xx2] = meshgrid(x1range,x2range);
17 XGrid = [xx1(:) xx2(:)];
18
```

```
%% Apply KNN with different distance functions
dists = ["euclidean", "cityblock", "chebychev", "mahalanobis"];
count = 1;
for d=dists
    % Create a KNN model with distance function d
    model = fitcknn(X,y,'NumNeighbors', 3,'Distance',d);

    % Make preditions using the model on the mesh grid
    predictions = predict(model,XGrid);

    % Plot the results
    subplot(2,2,count);
    gscatter(xx1(:), xx2(:), predictions,'rgb');
    title(d)
    count = count + 1;
    legend off, axis tight
end

%% K-Means Clustering
%% Generate Cluster Data
num_points = 300;
X = [randn(num_points,1)*0.1, randn(num_points,1)*1 + 3;
    randn(num_points,1)*1-0.5, randn(num_points,1)*0.1];

plot(X(:,1),X(:,2),'.');

%% Apply K-Means with different distance functions
figure;
dists = ["cityblock", "sqeuclidean", "cosine", "correlation"];
count = 1;
for d=dists
    opts = statset('Display','final');
    [idx,C] = kmeans(X,2,'Distance',d);
    subplot(2,2,count);
    hold on
    plot(X(idx==1,1),X(idx==1,2),'r.','MarkerSize',12)
    plot(X(idx==2,1),X(idx==2,2),'b.','MarkerSize',12)
    plot(X(idx==3,1),X(idx==3,2),'g.','MarkerSize',12)
    plot(C(:,1), C(:,2), 'kx', 'MarkerSize',15,'LineWidth',3)
    title(d)
    count = count + 1;
end
```

```
61
62 %% Feature Maps
63 %% Generate Data
64 num_points = 1000;
65 % Inner circle
66 X = [randn(num_points,1)*2, randn(num_points,1)*2];
67 % Outter circle
68 [x,y] = pol2cart(rand(num_points,1)*2*pi, 10 + rand(num_points
       ,1));
69 X = [X; x, y];
70
71 subplot(1,2,1);
72 scatter(X(:,1),X(:,2))
73 xlabel("X")
74 ylabel("Y")
75 title("X")
76
77 %% Plot Z (X')
78 Z = [X(:,1), X(:,2), (abs(X(:,1))+abs(X(:,2))).^2];
79 subplot(1,2,2);
80 scatter3(Z(:,1),Z(:,2), Z(:,3))
81 xlabel("X")
82 ylabel("Y")
83 zlabel("Z")
84 title("X'")
```