2017

# Pattern Recognition in Stock Data

Kathryn Dover
*Harvey Mudd College*

# Pattern Recognition in Stock Data

**Kathryn Dover**

Weiqing Gu, Advisor

Dagan Karp, Reader

HARVEY MUDD COLLEGE

**Department of Mathematics**

May, 2017

# Abstract

Finding patterns in high dimensional data can be difficult because it cannot be easily visualized. Many different machine learning methods are able to fit this high dimensional data in order to predict and classify future data but there is typically a large expense on having the machine learn the fit for a certain part of the dataset. This thesis proposes a geometric way of defining different patterns in data that is invariant under size and rotation so it is not so dependent on the input data. Using a Gaussian Process, the pattern is found within stock market data and predictions are made from it.

# Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to thank my advisor, Weiqing Gu, for helping me through this project and teaching me how to take a geometric perspective and put it into mathematical terms. I would also like to thank my second reader, Dagan Karp, for taking the time to offer advice and thoughts on this project. Finally, I'd like to thank my family and friends for all of their support throughout my time at Harvey Mudd.

# Chapter 1

# Introduction

When data is visually represented, an expert in the field can often identify and have knowledge about certain patterns in the data. The goal of this research is being able to apply specific expert knowledge in a field and have a computer program learn to identify these patterns within the data. This is a form of supervised learning and is different than other methods because the pattern definitions will be defined by geometry that is based on how the pattern looks to the expert. Therefore, the algorithm will be learning shapes rather than numbers. For a proof of concept, I will be looking at stock prices and finding certain patterns that have been identified and understood by experts in this field. In Chapter 2, I discuss the previous methods used to perform pattern recognition in stock data and the different stock patterns I will be trying to recognize. In Chapter 3, I will define the stock patterns in a geometric way and in Chapters 4 and 5, I will discuss how to apply these definitions in different ways and use them to find patterns in stock market data. Chapter 6 describes how the algorithm was implemented using a Gaussian Process. The results of this algorithm are in Chapter 7 and I discuss what my algorithm was able to find in different stock market data. Chapter 8 concludes with a reflection of my work and possible future work for my thesis.

# Chapter 2

# Background

Pattern recognition is the study within machine learning that is dedicated to finding different numerical methods to find patterns within a dataset. The ability to find patterns within data can also be used to classify data into different categories or predict behaviors on future datasets (Bishop (2006)). Because patterns can exist in many forms of data, pattern recognition has been used in to diagonose disorders with MRIs (Schiffmann and van der Knaap (2009)), classifying fingerprints (Kawagoe and Tojo (1984)) and facial recognition (Lawrence et al. (1997)). This section is a discussion on some of the current methods in pattern recognition, how it relates to understanding stock market data, and where the work in this paper builds upon the current field.

## 2.1 Methods for Pattern Recognition in Stock Data

This section gives an overview of a few common methods used for pattern recognition in stock data and the mathematics behind them. There are a variety of methods that are used in machine learning for stock data; for example, support vector machines, time series analysis, hidden markov model, bayesian linear regression, and neural networks all are well documented methods used in this field. While time series analysis and bayesian linear regression models are very important in stock market analysis (Halls-Moore (2016)), the hidden markov model, support vector machines and neural networks are closely looked at in this section because they are the most closely related to modeling stock market data using pattern recognition.

### 2.1.1 Support Vector Machines

A common method used in pattern recognition is a Support Vector Machine (SVM). SVMs are used specifically in classification of different patterns within data by separating data points using a hyperplane. A visualization of a SVM is shown in Figure 2.1. Support Vector Machines work to find a

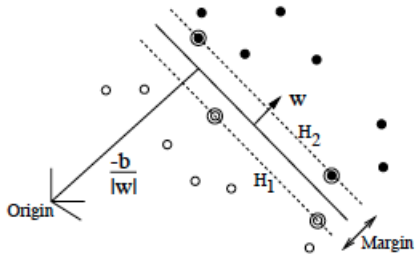


**Figure 2.1** Linear hyperplane separating data points (Burges (1998))

hyperplane that best separates the data into different classifications. In order to find this hyperplane, an SVM has a loss function given below (Murphy (2012)).

$$L_\epsilon(y, \hat{y}) = \begin{cases} 0 & \texttt{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}$$

where $y$ a known data point and the $\hat{y}$ is the predicted data point. Therefore, the loss function creates an $\epsilon$ tube around the data and penalizes the proposed hyperplane if too many points are far away from it, visualized in Figure 2.2. To use this loss function, one can add it to the objective function used to model the data. If $w$ is a vector of estimated solutions found by the algorithm and $C$ is a regularization constant, then an SVM tries to optimize the following equation.

$$J = C \sum_{i=1}^{N} L_\epsilon(y_i, \hat{y}_i) + \frac{1}{2}||w||^2$$

Murphy (2012) shows that the $w$ that optimizes this function, or creates a hyperplane that best separates the data, is a linear combination of the inputs.

$$w = \sum_i \alpha_i x_i$$

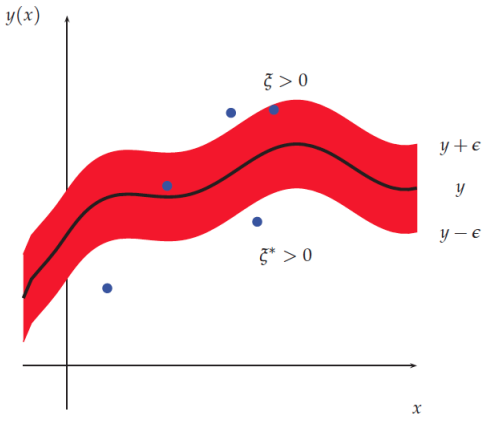


**Figure 2.2** A visualization of the $\epsilon$-tube created by the loss function (Murphy (2012))

In the above equation, the $\alpha_i$ are the constants that the machine changes to optimize the function. Burges (1998) found that this method of classification is limited by its speed and size for both training and testing the dataset as training the SVM takes two passes through the dataset.

### 2.1.2 Recurrent Neural Network

Another way to handle pattern recognition is with an artificial neural network. An illustration of a basic artificial neural network is shown in Figure 2.3. A neural network is typically structured into three different layers. The input layer puts different variables of the data into nodes. For example, consider a neural network being trained to figure out which students at the Claremont Colleges are Harvey Mudd students based off of their hometown, major, and age. The hometown would be an input for one node, the major for another node, and so on. The inputs are then passed to the hidden layer whose nodes typically start with randomized weights. Based on these weights, the inputs are scaled and then sent to the output layer. The output layer classifies the input based off of how they were scaled by the hidden layer. The classification is then compared with the real data and the weights are adjusted based on whether the classification is correct. This process is done for each data point in the training set so that the neural network converges to certain weights. Neural networks are popular with stock market analysis. Kamijo and Tanigawa (1990) used an neural network to find a

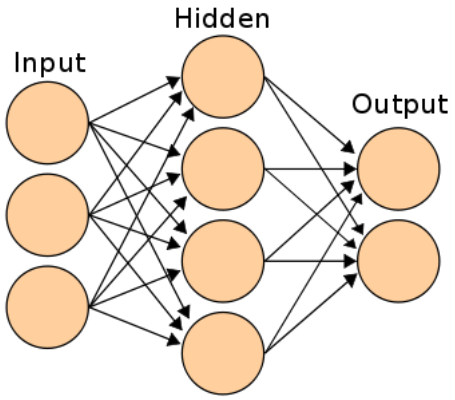


**Figure 2.3** A simple feed-forward neural nework (Mihail (2016))

triangle pattern in stock data. After training the neural network, the triangle pattern was accurately recognized in 15 out of 16 experiments. A limitation of this method is that patterns are numerically defined and the ANN is highly dependent on these numerical values.

### 2.1.3 Hidden Markov Model

Another common way to predict stock market data is a Hidden Markov Model (HMM), which is described in detail by Hassan and Nath (2005). An HMM is a finite state machine that uses probability to model a time series of multivariate observations. To understand the model, we define the following terms as outlined in [Hassan and Nath (2005)].

1. N is the number of states in the model.
2. T is the length of the observation sequence being modeled.
3. O represents all the observations of a sequence.
4. M is the number of observations, or outputs, for the system being modeled. In this case, it is the stock price of an index for each time step.
5. Q represents all the states in our Markov model.
6. $\pi$ is the the probability of being in a certain state of the model, e.g. $\pi_i$ is the probability of being in state $i$ at $t = 1$. Since $\pi$ is all the probabilities of this nature, $\sum_i \pi_i = 1$.

7. $A = \{a_{ij}\}$ represents the probability of a transition from state $i$ to $j$. Since $A$ is the collection of all these probabilities, $\sum_j a_{ij} = 1$.

8. $B = \{b_j(O_t)\}$ represents the probability of observing the observation sequence $O_t$ while in state $j$. Since $B$ is the set of all probabilities, $\sum_t b_i(O_t) = 1$.

The goal of this method is to use the Markov Model so that it most closely represents each of the observations at every state, or for each $b_j(O)$. In Hassan and Nath (2005), they used a Gaussian distribution to represent the observations. The probability for each observation at each state is the following sum.

$$b_j(O) = \sum c_{jm} \mathbb{N}[O, \mu_{jm}, U_{jm}] \quad 1 \leq j \leq N$$

$c_{jm}$ is the weighted coefficient in state $j$, where for each state $j$, $\sum_{m=1}^{M} c_{jm} = 1$. $\mu_{jm}$ is just the mean for each $m$th component in state $j$ and $U_{jm}$ is the covariance for the $m$th component in state $j$. $\mathbb{N}$ represents the Gaussian density, related to the Gaussian distribution chosen for the variables. In order to find a good model for the stock market data, the Markov Model is optimized by finding the optimal $c_{jm}$ such that the error between $b_j(O)$ and the original observation are as close as possible. Hassan and Nath (2005) found that Hidden Markov Models were just as successful as Artificial Neural Networks in modeling the stock market data and were able to fit the patterns of stock price well. However, this kind of algorithm needs to be trained on many data points because it is more exploratory as it requires the algorithm to find optimal coefficients for the model. Because of this, it is highly subjective to the dataset it is trained on and it's results cannot be applied to another stock index without being trained again.

## 2.2 Stock Patterns

In the following subsections, we explore the different patterns found in stock volume data. We look at the double-bottom pattern, the double-top pattern and the head shoulder pattern that are similar to the shapes $W$, $M$ and some combination of the two, respectively. The shapes that we define with vectors will be based off of these definitions we state here.

### 2.2.1 Double-bottom Pattern

There is a term in stock analysis call double-bottom pattern that has a *W* shape. It is defined as having the height of the second maximum be a 10-20% of the first minimum and the two minimum points should be within 3 – 4% of each other (Investopedia (2016)). An image of the pattern is shown in Figure 2.4. From this basic shape we can calculate the ratio of how the

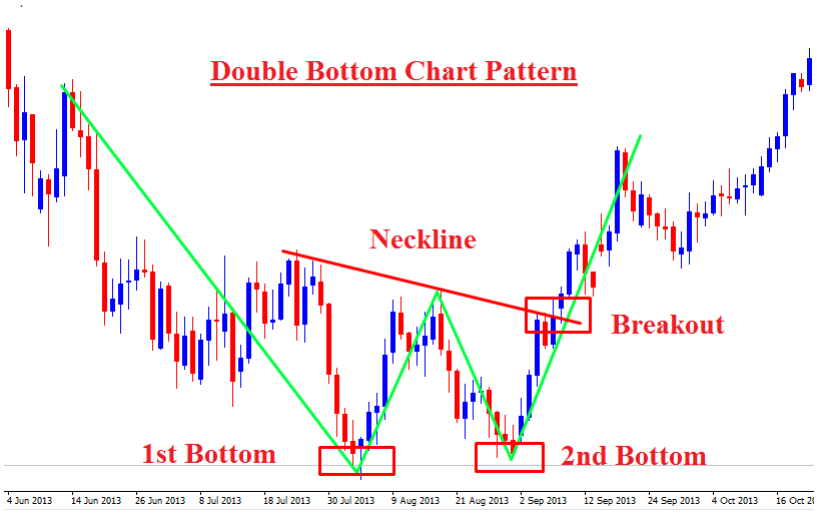


**Figure 2.4** An example of a double-bottom pattern in stock volume data (Forex (2016))

vectors making up the double-bottom pattern should be. This kind of shape is important in stock analysis as a trader can estimate when to sell the stock holdings towards the end of the double-bottom pattern. Additionally, traders can estimate how valuable it is to keep a stock holding while in one of the minimums of the double-bottom pattern (Investopedia (2016)).

### 2.2.2 Double-top Pattern

Another term in stock analysis is called a double-top pattern and it has a *M* shape rather than the *W* shape of the double-bottom pattern. It is defined to be a shape with two peaks with a trough in the middle, where the trough is a decline of about 10-20% of volume of the first peak, and the second peak is within 3% of the first peak (Investopedia (2016)). An illustration of a double-top pattern is shown in Figure 2.5. From this basic shape we can calculuate the ratio of how the vectors making up the double-top pattern

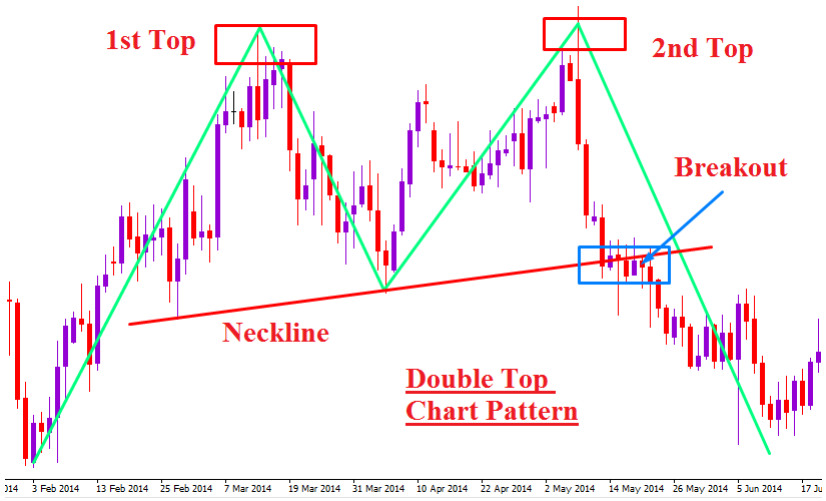


**Figure 2.5** An example of a double-top pattern in stock volume data (Forex (2016))

should be. This shape is important for stock traders to know because it can be used to understand what state a certain stock holding is in and when during the double-top pattern a holder should buy or sell (Investopedia (2016)).

### 2.2.3 Head and Shoulder Pattern

The head and shoulder pattern is a stock pattern that is almost an extension of the double bottom pattern. In the head and shoulder pattern, there are three peaks, the left shoulder the right shoulder and the head. The left and right shoulder are the outer peaks and the head is the middle peak. Preferably, the two troughs between the peaks lie approximately on the same line, which is called the neckline. Typically, the head peak is higher than the left and right shoulder and the left and right shoulder are approximately at the same height. An example of the pattern is shown in Figure 2.6. From this basic shape we can calculuate the ratio of how the vectors making up the head and shoulder pattern should be. These patterns discussed above are good to use for analysis on stock data because they represent different trends and if an algorithm is able to recognize them within data, an algorithm can be made to learn from them and be used as a predictive model on financial data.

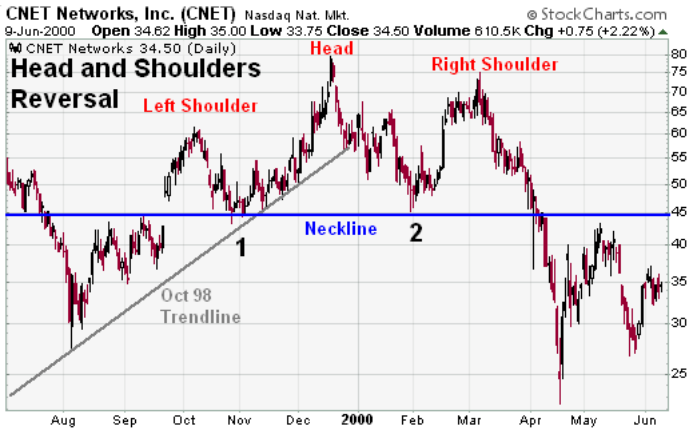


**Figure 2.6** An example of a head and shoulder pattern in stock volume data (StockCharts (2016))

## 2.3 Proposed Change in Approach

For my thesis, I propose to use a different approach in pattern recognition. Instead of using a support vector machine or creating an neural network, I will define shapes using geometry such that the definition is invariant under scaling and rotation. The ultimate goal will be to define a multidimensional shape using differential geometry and use that definition to identify those shapes in the data. The machine can then learn different shapes and predict the data using those shapes. For a proof of concept, I will start with 2-dimensional data so that I can visually check whether my algorithm is working correctly. I chose to work with stock data for different companies because it is readily available on the internet in large quantities. I plan to define the double-bottom, double-top and head and shoulder pattern as base cases for my 2-dimensional data and have an algorithm use these definitions to find them in the data. The algorithm will then learn from those shapes and use that as a predictive model for stock market data.

# Chapter 3

# Approach: Geometric Definition of Patterns

Based off of the patterns we explored above, we will be defining three different shapes:

1. Standard Double-Bottom Pattern (A Standard $W$)
2. Standard Double-Top Pattern (A Standard $M$)
3. Standard Head and Shoulder Pattern

Once the standard shapes are defined, we will explore how to approximate these shapes and create a metric for a pattern in the data with respect these definitions. For the rest of the paper, the double-bottom pattern will be referred to as a $W$ shape and the double-top pattern will be referred to as an $M$ shape because those geometrically best describe those patterns and are a better shorthand.

## 3.1 The Standard W

A $W$ shape can exist in a section of the data only if there are 3 local maxima and 2 local minima. The 3 local maxima are labeled in succession as the points $\{Q_1, Q_2, Q_3\}$ and label the 2 local minima as $\{P_1, P_2\}$. An example of such a labeling is shown in Figure 3.1. Now that we have labeled the points, we can make a basis. We do this by first finding the midpoint of $P_1$ and $P_2$, which we denote as $O$. In the standard $W$ shape, the vector $\overrightarrow{OQ_2}$ is perpindicular to the vector $P_1P_2$. From this, we can see that $\overrightarrow{P_2Q_2} = \overrightarrow{OQ_2} - \overrightarrow{OP_2}$ and that

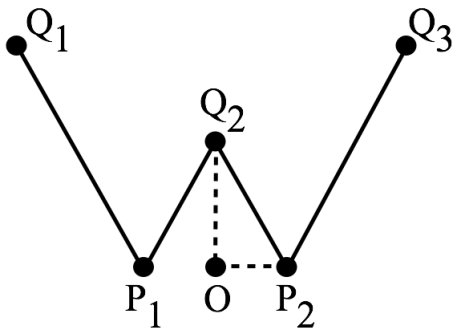


**Figure 3.1** A visualization of how to construct a basis for a standard $W$ shape using the maximum and minimum points

$\overrightarrow{P_1Q_2} = \overrightarrow{OQ_2} + \overrightarrow{OP_2}$. If we apply this to some standard $W$, then we can also come up with the following coefficients $c_1, c_2$:

$$\overrightarrow{P_1Q_1} = \overrightarrow{OP_1} + c_1\overrightarrow{P_2Q_2}$$

$$\overrightarrow{P_2Q_3} = \overrightarrow{OP_2} + c_2\overrightarrow{P_1Q_2}$$

In the most ideal $W$, we would find that the height of $Q_2$ is about 10% to 20% of the height of $Q_1$ and $Q_3$, which would mean that $c_1$ and $c_2$ would be greater than 1. Once we have found $c_1, c_2$, one way we can compare other $W$ shapes to the standard $W$ by comparing how scaled their egde vectors are compared to the standard $c_1, c_2$. The basis we will use will be $\{\vec{v_1}, \vec{v_2}\}$ where $\vec{v_1}$ and $\vec{v_2}$ are defined as the following:

$$\vec{v_1} = \overrightarrow{OP_2}$$

$$\vec{v_2} = \overrightarrow{OQ_2}$$

This basis will be useful when we approximate shapes in the next chapter.

## 3.2 The Standard M

Now that we have defined a $W$ shape, we will use the same process to define the other shapes. Figure 3.2 shows a visualization of an $M$ shape. Our representation of an $M$ is very similar to $W$. An $M$ is first identified by having 3 local mins and 2 local max's over an interval, identified as $\{P_1, P_2, P_3\}$ and $\{Q_1, Q_2\}$ respectively. We find the midpoint $O$ between $Q_1$ and $Q_2$ and in a standard $M$ shape, the vector $\overrightarrow{OP_2}$ should be perpindicular

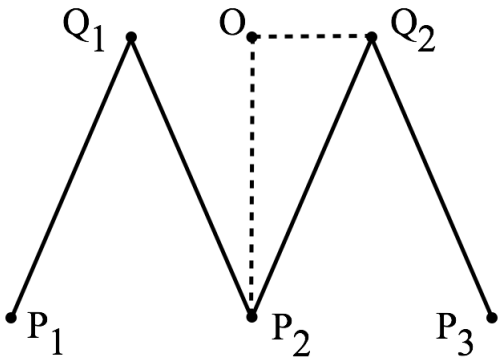


**Figure 3.2** A visualization of how to construct a basis for a standard $M$ shape using the maximum and minimum points

to the vector $\overrightarrow{Q_1Q_2}$. We define the basis of the shape to be $\{\overrightarrow{OP_2}, \overrightarrow{OQ_2}\}$. Like $W$, we want to find the constants $c_1, c_2$ such that the following is true:

$$\overrightarrow{Q_1P_1} = \overrightarrow{OQ_1} + c_1\overrightarrow{Q_2P_2}$$
$$\overrightarrow{Q_2P_3} = \overrightarrow{OQ_2} + c_2\overrightarrow{Q_1P_2}$$

$c_1$ and $c_2$ help define how the edges of $M$ are scaled compared to the vectors that are parallel to them. Again, the height of $P_2$ is supposed to be within 10% of the heights of $Q_1$ and $Q_2$, so we would expect our constants $c_1$ and $c_2$ to be greater than 1. Once we have found $c_1, c_2$ we can compare other $M$ shapes to the standard $M$ to see if the scaling of edge vectors is similar or not.

## 3.3 The Standard Head and Shoulder

Another common shape in economics is the head and shoulders pattern, shown in Figure 3.3. Again, we find the midpoint $O$ between $P_2$ and $P_3$ and let the basis of the shape be the vectors $\{\overrightarrow{OQ_2}, \overrightarrow{OP_3}\}$. Thus, we want to compare the scaled vectors to each other so we calculate the coefficients $c_1, c_2, c_3, c_4$ such that

$$\overrightarrow{P_1Q_1} = \overrightarrow{OP_3} + c_1\overrightarrow{P_3Q_3}$$
$$\overrightarrow{P_4Q_3} = \overrightarrow{OP_2} + c_2\overrightarrow{P_2Q_1}$$
$$\overrightarrow{P_2Q_2} = \overrightarrow{OP_3} + c_3\overrightarrow{P_3Q_3}$$

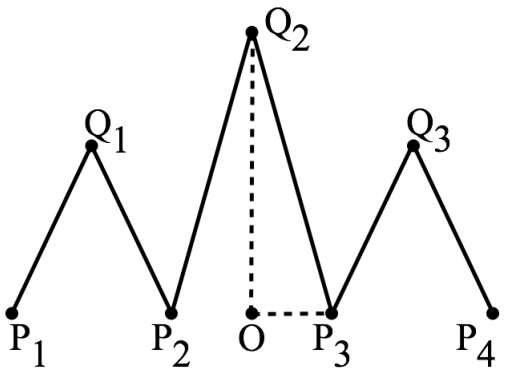


**Figure 3.3** A visualization of how to construct a basis for a standard head and shoulder shape using maximum and minimum points

$$\overrightarrow{P_3Q_2} = \overrightarrow{OP_2} + c_4\overrightarrow{P_2Q_1}$$

We would hope to find that $c_1 \approx c_2$ and that $c_3 \approx c_4$ since the lengths of these vectors should be about the same given the shape of the head and should pattern. The coefficients, like the ones we found for $M$ and $W$ are helpful when comparing other shapes to the standard head and shoulder shape.

# Chapter 4

# Approach: Geometric Definition of Fuzzy Shapes

When we actually apply our definition of shapes to the data, we will find that there is no perfect match to our standard definition. Therefore, we will want to define a "fuzzy shape" that will be considered an approximate form of the standard shape. These fuzzy shapes will not only allow us to approximate how far away a shape is from the standard shape but also how far fuzzy shapes are from each other. The way that we define our fuzzy shapes is discussed below.

## 4.1 The Fuzzy W

In the previous chapter we defined a standard $W$ that we will compare all other $W$ shapes to. Real $W$'s in the data will be skewed smaller or bigger than the standard $W$ and we want our program to still be able to identify these shapes correctly. In order to do these comparisons, we will need to define two different terms: almost parallel and almost perpindicular, denoted $\dot{\parallel}$ and $\dot{\perp}$ respectively. Given the standard $W$ in Figure 3.1, we assumed that the edges were parallel to each other and that $\overrightarrow{OQ_2}$ was perpendicular to $\overrightarrow{P_1P_2}$. But what if our $W$ does not quite meet those standards? Thus we are motivated to calculate the angle between the two vectors.

$$\cos(\theta_1) = \frac{\langle \overrightarrow{P_1P_2}, \overrightarrow{Q_1Q_3} \rangle}{||\overrightarrow{P_1P_2}||||\overrightarrow{Q_1Q_3}||}$$

If $\cos(\theta_1) = 1$, then $\theta_1 = 0$ and the vectors are parallel. $\theta_1 = 0$ is too much to hope for in real data, so we loosen our bounds. Thus we say that if $\cos(\theta_1) \approx 1$, then $\theta \approx 0$ or for some threshold value $\epsilon_1$, $|\theta_1 - 0| < \epsilon_1$. This means that the vectors from the local maxima values that define our shape are almost parallel to each other. Whether these important vectors are parrallel or perpindicular help determine if something is a $W$ shape. Therefore we would say that if $\cos(\theta_1) \approx 1$, then $\overrightarrow{P_1P_2}$ is almost parallel to $\overrightarrow{Q_1Q_3}$, denoted $\overrightarrow{P_1P_2} \dot{\parallel} \overrightarrow{Q_1Q_3}$. We also want vectors in the $W$ shape to be perpindicular, like $\overrightarrow{Q_2O}$ and $\overrightarrow{P_1P_2}$ to consider the following angle $\theta_2$.

$$\cos(\theta_2) = \frac{\langle \overrightarrow{Q_2O}, \overrightarrow{P_1P_2} \rangle}{||\overrightarrow{Q_2O}||||\overrightarrow{P_1P_2}||}$$

If $\cos(\theta_2) \approx 0$, then this means that $\theta \approx \pi/2$. We can regulate this with another threshold value $\epsilon_2$ such that $|\theta_2 - \pi/2| < \epsilon_2$. Therefore, we have that $\overrightarrow{Q_2O}$ and $\overrightarrow{P_1P_2}$ are almost perpidicular to each other, so we denote this by $\overrightarrow{Q_2O} \dot{\perp} \overrightarrow{P_1P_2}$. If we have our desired $\theta_1$ and $\theta_2$ such that $\overrightarrow{P_1P_2} \dot{\parallel} \overrightarrow{Q_1Q_3}$ and $\overrightarrow{Q_2O} \dot{\perp} \overrightarrow{P_1P_2}$, then our shape is a $W$. Alternatively, if $\theta_1$ and $\theta_2$ do not meet our pre-designated threshold, the figure is considered not a $W$.

In conclusion, a pattern is a $W$ shape if it satisfies the following properties.

1. $\overrightarrow{P_1P_2} \dot{\parallel} \overrightarrow{Q_1Q_2}$
2. $\overrightarrow{OQ_2} \dot{\perp} \overrightarrow{P_1P_2}$
3. $\overrightarrow{P_1Q_1} \dot{\parallel} \overrightarrow{P_2Q_2}$
4. $\overrightarrow{P_1Q_2} \dot{\parallel} \overrightarrow{P_2Q_3}$
5. $Q_1$, $Q_3$ have approximately the same height
6. $P_1$, $P_2$ have approximately the same height
7. $Q_2$ has a lower height than $Q_1$ and $Q_3$

## 4.2 The Fuzzy M

In order for a shape to be a $M$ based off of our standard $M$ shown in Figure 3.2, it must have the following qualities.

1. $\overrightarrow{Q_1Q_2} \dot{\parallel} \overrightarrow{P_1P_3}$

2. $\overrightarrow{OP_2} \dot{\perp} \overrightarrow{Q_1Q_2}$
3. $\overrightarrow{P_1Q_1} \dot{\parallel} \overrightarrow{P_2Q_2}$
4. $\overrightarrow{P_2Q_1} \dot{\parallel} \overrightarrow{P_3Q_2}$
5. $P_1$, $P_3$ have approximately the same height
6. $Q_1$, $Q_2$ have approximately the same height
7. $P_2$ has a lower height than $P_1$ and $P_2$

## 4.3 The Fuzzy Head and Shoulder

We want to check for the following conditions in order for a pattern to be a head and shoulders shape based off of our standard head and shoulders shape shown in Figure 3.3.

1. $\overrightarrow{Q_1Q_3} \dot{\parallel} \overrightarrow{P_1P_4}$
2. $\overrightarrow{OQ_2} \dot{\perp} \overrightarrow{P_2P_3}$
3. $\overrightarrow{P_1Q_1} \dot{\parallel} \overrightarrow{P_2Q_2} \dot{\parallel} \overrightarrow{P_3Q_3}$
4. $\overrightarrow{P_2Q_1} \dot{\parallel} \overrightarrow{P_3Q_2} \dot{\parallel} \overrightarrow{P_4Q_3}$
5. $P_1, P_2, P_3, P_4$ are at approximately the same height
6. $Q_1, Q_3$ are at approximately the same height
7. $Q_2$ height is greater than heights $Q_1, Q_2$

Something notable about these definitions is that they are entirely dependent on how the vectors are defined. We are checking whether vectors are parallel or perpindicular, not their actual numerical values. Because of this, the definitions should be invariant under size and rotation. Each of these vectors is defined by their basis, so a basis for a small shape should be similar for a basis for a larger shape. In this way we have applied the definitions so that approximate shapes can be found in the data via the fuzzy shape definitions. Once fuzzy shapes are identified in the data, the definitions allow us to compare the shapes to each other and how they change in the data. The next section discusses how the fuzzy shapes can be handled so predictions can be made.

# Chapter 5

# Results: New Approach on Handling the Shapes

Now that we have defined both our standard and fuzzy shapes, we want to be able to define some actions we can take on these shapes. We discuss the change of basis and flipping a shape and how they can be used to compare different shapes to each other. We also discuss how the slopes and lengths of the vectors can be used to categorize the shape that can be used for prediction.

## 5.1 Change of Basis

Given the standard $W$ in Figure 3.1, our basis for the shape is defined with the following vectors:

$$\vec{v_1} = \overrightarrow{OP_2}$$
$$\vec{v_2} = \overrightarrow{OQ_2}$$

To compare this $W$ to a skewed $W^{'}$ we need to change any vectors defined on $W^{'}$ and write them in terms of the basis of $W$. Let the maximum points of $W^{'}$ be $\{Q_1^{'}, Q_2^{'}, Q_3^{'}\}$ and let the consecutive mininum points be $\{P_1^{'}, P_2^{'}\}$. The basis for $W^{'}$ is $\{\overrightarrow{O^{'}P_2^{'}}, \overrightarrow{O^{'}Q_2^{'}}\}$. We can find the coefficients $a, b, c, d$ such that

$$\overrightarrow{O^{'}P_2^{'}} = a\overrightarrow{OP_2} + b\overrightarrow{OQ_2}$$
$$\overrightarrow{O^{'}Q_2^{'}} = c\overrightarrow{OP_2} + d\overrightarrow{OQ_2}$$

Which means that the change of basis matrix from $W^{'}$ to $W$ is

$$P = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

With this change of basis matrix, we can compare different vectors that make up $W^{'}$ compared to $W$. For example, if $W^{'}$ is almost like $W$, then $P[\overrightarrow{Q_1^{'}P_1^{'}}]$ will almost equal $\overrightarrow{Q_1P_1}$.

## 5.2 Flipping a Shape

Finding a change of basis for a shape is not the only way to compare patterns to each other. Since some of the shapes we defined are reflections of each other, it could be useful to define what flipping a shape across an axis would mean. To see the idea of this, we show an illustration of flipping a $W$ into an $M$ in Figure 5.1. The idea is to reflect all the vectors making up the shape

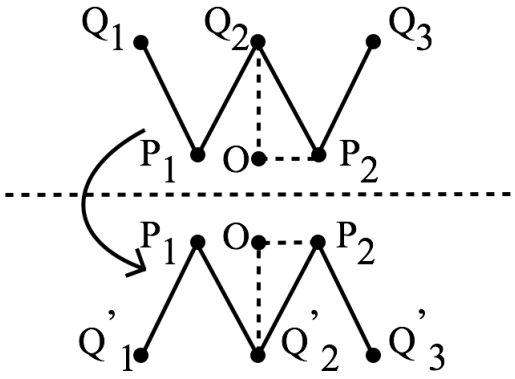


**Figure 5.1** Flipping a $W$ shape into an $M$ shape

across a certain axis, in this case the axis is the vector $\overrightarrow{OP_2}$. Because we are reflecting the shape across the $x$ axis, we have that the reflection of a vector $\vec{v}$ across $\overrightarrow{OP_2}$.

$$\text{reflection}_{\overrightarrow{OP_2}}(\vec{v}) = \frac{\langle \overrightarrow{OP_2}, \vec{v} \rangle}{||\overrightarrow{OP_2}||^2} \overrightarrow{OP_2} - \vec{v}$$

So in order to flip the entire shape $W$, we flip all of our defined vectors across the vector $\overrightarrow{OP_2}$, which is considered our basis $x$ axis.

## 5.3 Symmetric Representation

Flipping an entire shape across an axis is one way to use reflection. We could also use this idea of reflection with an application to symmetry when defining a shape. Let us go back to the original $W$ shape shown in Figure 5.2. We define the axis of reflection to be the vector $\overrightarrow{OQ_2}$. Therefore, we

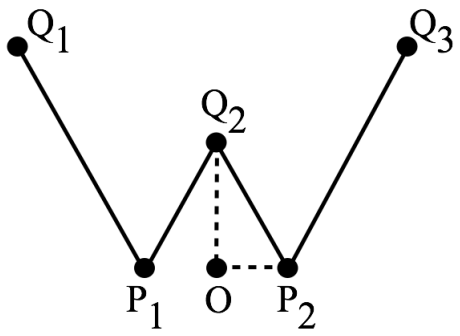


**Figure 5.2** A standard $W$

want to see if the reflection of the vectors $\overrightarrow{P_1Q_1}$ and $\overrightarrow{P_1Q_2}$ reflected across the axis gives us $\overrightarrow{P_2Q_3}$ and $\overrightarrow{P_2Q_2}$ respectively. Since we consider $O$ to be the origin, to flip the left hand vectors to the other side we must multiply them by the matrix $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$. To check if the reflected vectors are the same as the vectors in the shape, we take the projection of the reflected vector onto the actual vector. If the projection is approximately the same as the original vectors, the shape would be considered very symmetrical. Let us consider the vector $\overrightarrow{P_1Q_1}$, and reflect it across $\overrightarrow{OQ_2}$, giving us the vector $\vec{V}$. We get that

$$\text{proj}_{\overrightarrow{P_2Q_3}}(\vec{V}) = \frac{\langle \overrightarrow{P_2Q_3}, \vec{V} \rangle}{||\overrightarrow{P_2Q_3}||^2} \overrightarrow{P_2Q_3}.$$

If this shape is symmetric, then the flipped vector $\vec{V}$ should be $\overrightarrow{P_2Q_3}$, which means that the projection of $\vec{V}$ onto $\overrightarrow{P_2Q_3}$ should equal $\overrightarrow{P_2Q_3}$.

## 5.4 Fuzzy Symmetry

Now that we have defined a symmetric representation, we can define a fuzzy symmetry for shapes that are almost symmetric. Let us again consider the

case with the shape $W$ and the vector $\overrightarrow{Q_1P_1}$ and check if it is symmetric to $\overrightarrow{P_2Q_3}$. So, we want to flip $\overrightarrow{P_1Q_1}$ over the basis vector $\overrightarrow{OQ_2}$.

$$\vec{V} = \frac{\langle \overrightarrow{OQ_2}, \overrightarrow{P_1Q_1} \rangle}{||\overrightarrow{OQ_2}||^2} \overrightarrow{OQ_2} - \overrightarrow{P_1Q_1}$$

Now that we have the reflected vector $\vec{V}$, we can compare it to $\overrightarrow{P_2Q_3}$. In order to do so, we want to project $\vec{V}$ onto $\overrightarrow{P_2Q_3}$ and see if we get approximately $\overrightarrow{P_2Q_3}$. To do so, we do the following projection.

$$\text{proj}_{\overrightarrow{P_2Q_3}}(\vec{V}) = \frac{\langle \overrightarrow{P_2Q_3}, \vec{V} \rangle}{||\overrightarrow{P_2Q_3}||^2} \overrightarrow{P_2Q_3}$$

If $\text{proj}_{\overrightarrow{P_2Q_3}}(\vec{V})$ is approximately $\overrightarrow{P_2Q_3}$, then

$$\frac{\langle \overrightarrow{P_2Q_3}, \vec{V} \rangle}{||\overrightarrow{P_2Q_3}||^2} \approx 1.$$

So we say that $W$ is fuzzy symmetric if $|\frac{\langle \overrightarrow{P_2Q_3}, \vec{V} \rangle}{||\overrightarrow{P_2Q_3}||^2} - 1| \leq \epsilon$ for some error factor $\epsilon$. If the shape is not symmetric enough, then it is not a $W$ shape. This same process can be used for the $M$ shape and head and shoulder shape by reflecting all vectors on one half of the axis. If the projections are close to the other vectors, the shape is symmetric.

## 5.5 Categorizing Shapes Using Slopes and Lengths

Once the shapes have been found in the data, we can categorize these shapes using their slopes and lengths. By categorizing certain shapes by their slopes and lengths, we can group similar shapes together and run traditional machine learning techniques on these specific groups. This is to determine indicators of these types of shapes, which can ultimately be used for predictive purposes. We discuss how shapes can be categorized using slopes and lengths and how we can use those categories to do rough predictions of stock market data.

### 5.5.1 Saving the Slopes and Lengths

When the algorithm identifies a shape, it will calculate the set of lengths $\{\ell_i\}$ and slopes $\{s_i\}$ of each segment of the shape, as illustrated in Figure 5.3.

For prediction purposes, the algorithm also calculates and stores the length and slope of the segment that follows the shape. Now that these lengths and slopes are stored, we can calculate a constant $k$ that will categorize this shape.

Let us consider for this case the predetermined $W$ shape in figure in 3.1. Let

$$W_\ell = \{\ell_1, \ell_2, \ell_3, \ell_4\}$$

$$W_s = \{s_1, s_2, s_3, s_4\}$$

be the lengths and slopes of $W$, respectively. Then, for the $i$ith $W$ found in the data by the algorithm, we define the following

$$W_{i_\ell} = \{\ell_1^i, \ell_2^i, \ell_3^i, \ell_4^i\}$$

$$W_{i_s} = \{s_1^i, s_2^i, s_3^i, s_4^i\}$$

to be the slopes and lengths of the $i$th $W$ found by the data. Thus, we can define a $k_i$ to be the constant that minimizes the difference between the slopes and lengths of $W$ with the slopes and lengths of $W_i$.

$$(\ell_1, \ell_2, \ell_3, \ell_4) = k_\ell^i \left(\ell_1^i, \ell_2^i, \ell_3^i, \ell_4^i\right)$$

$$(s_1, s_2, s_3, s_4) = k_s^i (s_1^i, s_2^i, s_3^i, s_4^i)$$

Thus, $k_\ell^i$ and $k_s^i$ are the optimal solutions for the following equations.

$$(\ell_1, \ell_2, \ell_3, \ell_4) - k_\ell^i \left(\ell_1^i, \ell_2^i, \ell_3^i, \ell_4^i\right) = 0$$

$$(s_1, s_2, s_3, s_4) - k_s^i (s_1^i, s_2^i, s_3^i, s_4^i) = 0$$

In the next section, we discuss how we can use these constants to do a rough prediction on the data based on the shape found.

## 5.6 Rough Predictions

Now that we have defined how we can categorize the shapes, there are many different ways we can actually use these categories to do rough predictions with the data. Below, two different methods for predictions are described and how they relate to our geometric definitions. These are more rough predictions, or averages, rather than rigorous machine learning techniques used for prediction.

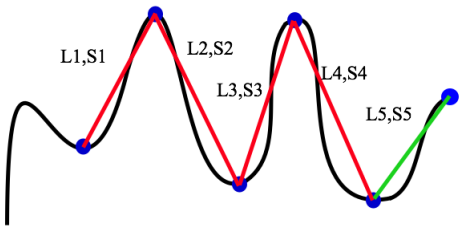


**Figure 5.3** Identifying a shape in the data and saving the following segment's length and slope

### 5.6.1 Predictions with Slopes and Lengths

Once the algorithm has stored and calculated the constants $k_i$'s for all the shapes, we can do a rough calculation to determine what the expected slope and length. We do this by averaging the sets of $k_\ell^i$'s and $k_s^i$'s with the segment that follows each of the shapes. If $\ell_5^i$ and $s_5^i$ are the length and slope of the following segment for each shape, then our estimated length and slope for the $W$ shape are the following values.

$$\ell^{'} = \frac{k_\ell^1 \ell_5^1 + k_\ell^2 \ell_5^2 + \ldots + k_\ell^n \ell_5^n}{n}$$

$$s^{'} = \frac{k_s^1 s_5^1 + k_s^2 s_5^2 + \ldots + k_s^n s_5^n}{n}$$

We scale the each of the segments by the the associated constant so that extra large and extra small shapes will not skew the average. The prediction is just measuring the average of the following slope and lengths for each shape relative to the size of the original.

This method of prediction can be used to identify what kinds of trends follow a specific shape. Additionally, we can make this prediction more specific by just doing a prediction over shapes that have similar $k_i$'s for either lengths or slopes. In this case, all of the $k_i$'s would be similar, which would mean that a general average would not reveal much for a prediction. However, we could use these $k_i$'s to group out shapes and then do other machine learning algorithms on this specific part of the data set.

### 5.6.2 Prediction with Direction

Another way to consider prediction is tracking the direction of the stock after each shape. Again, we consider the segment that follows any shape, in this case let us consider the ith $W$ found in the data. The last point of $W_i$ is the $Q_3^i$ point, and let $P^i$ be the next local maxima in the data. Then, we can define the following vector.

$$\overrightarrow{P^iQ_3^i} = P^i - Q_3^i$$

We can then normalize this vector, creating a unitary vector $\mu_i$.

$$\mu_i = \frac{\overrightarrow{P^iQ_3^i}}{||\overrightarrow{P^iQ_3^i}||} = \frac{P^i - Q_3^i}{||P^i - Q_3^i||}$$

Therefore, for each shape found in the dataset, we can store all the directional vectors that follow the shapes. We can have a better understanding of these directional vectors by taking the average of all the vectors we find $\mu$.

$$\vec{\mu} = \frac{\overrightarrow{\mu_1} + \overrightarrow{\mu_2} + \ldots + \overrightarrow{\mu_n}}{n}$$

In the prediction part of our algorithm, if we find some directional vector $\overrightarrow{\mu_j}$, then we can calculate how far it is from the average by calculating the angle between the two vectors.

$$\cos(\theta) = \frac{\langle \vec{\mu}, \overrightarrow{\mu_j} \rangle}{||\vec{\mu}||||\overrightarrow{\mu_j}||}$$

$$\theta = \arccos\left(\frac{\langle \vec{\mu}, \overrightarrow{\mu_j} \rangle}{||\vec{\mu}||||\overrightarrow{\mu_j}||}\right)$$

The smaller $\theta$ is, the closer the vectors are to each other and the closer the prediction is to the actual value. Additionally, since we normalize the vectors, these values should not be skewed by very large or very small shapes.

# Chapter 6

# Implementation: Creating an Algorithm to Find Patterns

Now that we have defined what our shapes are and how to approximate shapes, we will be discussing the algorithm that finds these shapes and how it handles the data. I used a Gaussian Process to fit the data with a function that the local minimums and maximums could be extracted from. These local extrema are used to define the vectors of the shapes and subsequently the algorithm checks these vectors using the definitions from the previous sections.

## 6.1 Gaussian Process

One of the issues with supervised learning is that data can often be corrupted by noise so it is better to infer a distribution over different functions of the data to make predictions with new imputs. A way to do this is to use a Gaussian Process (GP). A GP for a regression model gives a variety of fits to the same data based off of what the noise in the data is estimated to be. An example of the different fits are shown in Figure 6.1. Assume we have some finite data set of inputs $\{x_i\}$ for $1 \leq i \leq N$ which corresponding outputs $\{y_i\}$. To use a GP, we assume that there exists some function $f(x_i) = y_i$ such that $p(f(x_1), f(x_2), \ldots, f(x_N))$ (the probability of these points) is jointly Gaussian where $\mu(x)$ is the mean function and the covariance function is $\sigma(x)$. The covariance function is defined by a kernel function $k(x_i, x_j)$ where $\sigma_{ij} = k(x_i, x_j)$. This basicially gaurantees that if $x_i$ and $x_j$ are similar by the kernel's standards, then their outputs should be similar as well. When

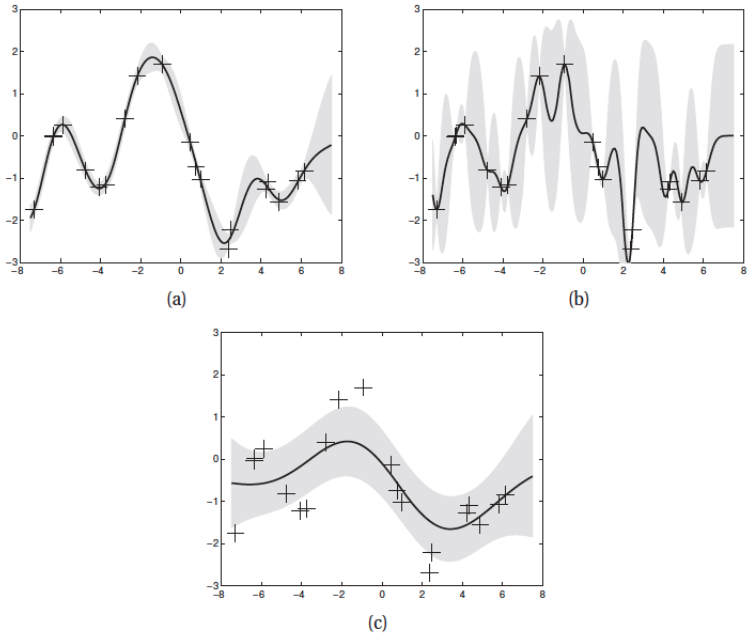


**Figure 6.1** Various Gaussian Fits

GPs are used, it is often assumed that $\mu(x) = 0$ so that the GP is completely defined by the covariance function. The result of this is that if the covariance function allows for little variance between points, then the fit the GP returns is very tight, or overfitted. This occurs because it is assuming there is very little noise in the data. However, if the GP has a kernel that allows for extreme variation, there will be a looser fit in the data as it is assuming a lot of noise is present in the data. We can see this with 6.1, where plot (a) shows the tightest fit possible with the data, and plot (c) shows the loosest fit for the data points.

I used a Gaussian Process to fit my stock data so I could find different sizes of the same shape. The plan was to overfit and underfit the data with different models so that I could find both small and large shapes. I used the function gausspr that is an available R package.

## 6.2 Finding Local Extrema

After a Gaussian Process is fitted to the data, the next step is for the algorithm to find the local minima and maxima of the data. To do this, the GP of the data predicts what the data will look like given the same time interval. These predicted values should be close to the original data. Using these predicted values, a function in R checks each point in the fit and determines if it is a local maxima. It does this by determining if its neighbors have a greater or smaller value. Once the algorithm has gone through the entire GP fit, it saves the local extrema in a single list where the maxima are ordered by their appearance in the data.

## 6.3 Creating Vectors Using End Points

Once the algorithm stores all the minimum and maximum points, it runs through them to determine whether a certain shape is possible. For example, consider our standard $W$ shape in Figure 3.1. In order for a $W$ to exists, the sequence of points would need to be the following list.

maximum, minimum, maximum, minimum, maximum

If we are looking specifically a $W$ shape, the algorithm will look a list of 5 maxima (in sequential order) to see if they could possibly be a $W$. If they could possibly be the shape, each maxima is made an endpoint and the rest of the checking follows the definitions defined in Chapter 4.

## 6.4 Storing Information for the Prediction

If a sequence of extrema is determined to be a valid shape, the algorithm stores the slopes and lengths of the vectors in the shape and the segment that follows the shape. From these values, it can calculate the constants described in the previous chapter. Once the algorithm has found every shape in the data, it will use these properties of the shape to do a rough prediction.

## 6.5 Running the Algorithm on the Data

The overview of the algorithm is given in the following steps.

1. Given a certain variance, a Gaussian Process is used to fit the data.

2. From this fit of the data, the algorithm identifies all the local extrema in the data.

3. The algorithm goes through the local extrema in the data sequentially and identifies sequences that could possibly be a predefined shape. For example, a *W* has a following sequence of minium and maximum: max/min/max/min/max.

4. If a sequence is valid for a certain shape, then the algorithm uses the definitions of the shapes defined earlier in order to figure out if they qualify.

5. If a shape is found, the lengths and slopes of the vectors in the shape are stored (as well as the following segment) for predictive purposes.

6. The categorizing constants $k_\ell^i$ and $k_s^i$ are calculated and stored.

7. Once the algorithm has gone through all the extrema, it uses the stored slopes, lengths, and categorizing constants and uses them to calculate the rough prediction from the weighted average.

In Figure 6.2, we can see the steps of the algorithm at work. The image in the top left corner shows the data in it's original form. The red line shows the fit the Gaussian Process makes when it is given a tight variance. The blue points indicate the completion of the next step of the process where the local maxima and minima are found on the fit from the GP. Finally, the algorithm runs through all the local extrema to find which combinations of them make a valid shape, which is highlighted in green.

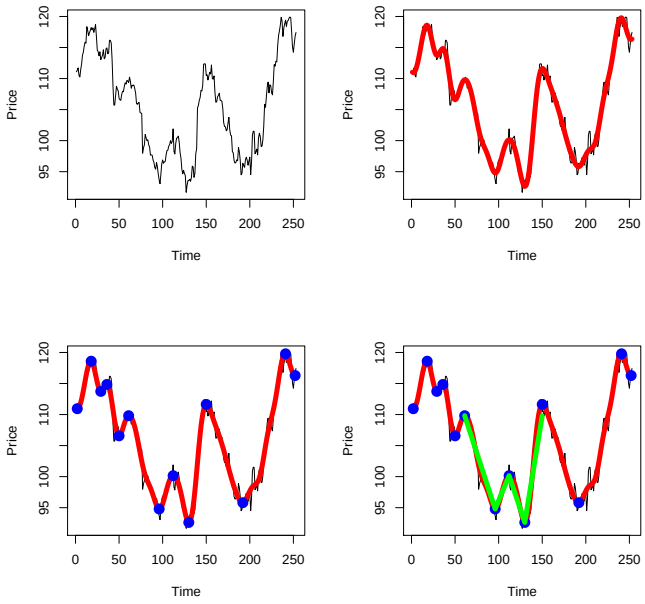


**Figure 6.2** The original data (upper left), the fit of the data when the variance is 0.001 (upper right), the algorithm identifying the local extrema (lower left) and the $W$ found by the algorithm (lower right)

# Chapter 7

# Results: Running the Algorithm on Real Data

In this section, I describe the results of the algorithm I developed in the last chapter when it was run on stock price history for different companies. The algorithm was run for variances 0.001 and 4 over the entire stock price history for the companies Apple, Disney, Microsoft, Walmart and the NASDAQ index. For testing the prediction part of the algorithm, the entire stock price history for the Dow Jones index was used. The reason for choosing this dataset was twofold. First, there was between 20-30 years worth of data for each company, which would mean I could potentially find a variety of shapes in different sizes. Second, I chose these specific companies by looking through the stock history of companies and choose those that had many turbulent years that offer enough fluctuation that can create the shapes I am looking for. All stock data was found at yahoo.finance.com. The below subsections discuss how I fit my data, found the shapes, and calculated predictions for different shapes.

## 7.1 Results

For my data, I used a Gaussian Process with variances 0.001 for the tightest fit and 4 for the largest fit. Since the dataset had 70,430 lines of data, the Gaussian Process function in R was not able to fit the data all at once. Instead, I broke up the data into steps of 500 data points and fit each group of points separately in order to make the function run. The results from these fits are discussed below.

### 7.1.1 Finding Different Sized Shapes

Figures 7.1 and 7.2 show the steps the algorithm takes to find the shapes and how the same definition can find different sized shapes. Figure 7.1 shows the original data and the fit made to the data when the variance is 0.001. This fit leads to a $W$ shape being found in the data. Similarly, Figure 7.2 shows the same steps but with a larger variance of 4. Even though this is the same exact dataset, the fit is looser and leads to a larger $W$ being found. The definition for the shape stayed the same and the algorithm was successful in finding both shapes in the data just using different fits from the GP.

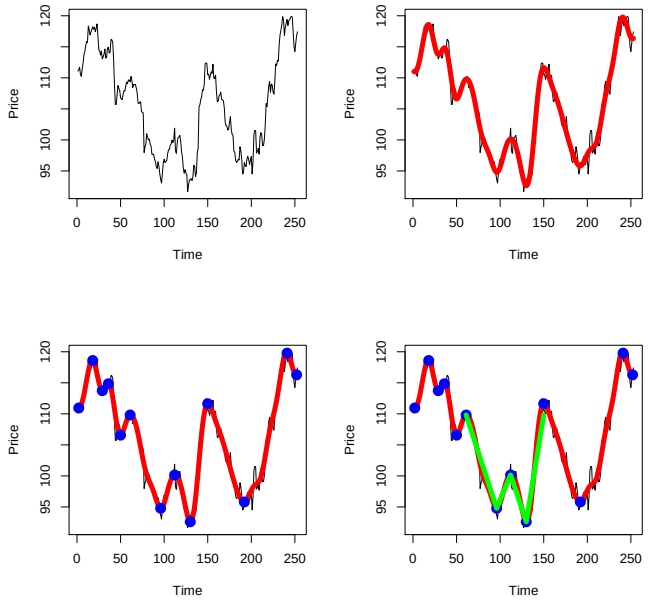


**Figure 7.1** The original data (upper left), the fit of the data when the variance is 0.001 (upper right), the algorithm identifying the local extrema (lower left), the $W$ found by the algorithm (lower right)

### 7.1.2 Finding Multiple Shapes

Figures 7.3, 7.4 and 7.5 each show multiple instances of $W$, $M$ and head and shoulder shapes being found in the data when the fit had a variance of 0.001. In Figure 7.3, we can see two different examples of when a $W$ shape was

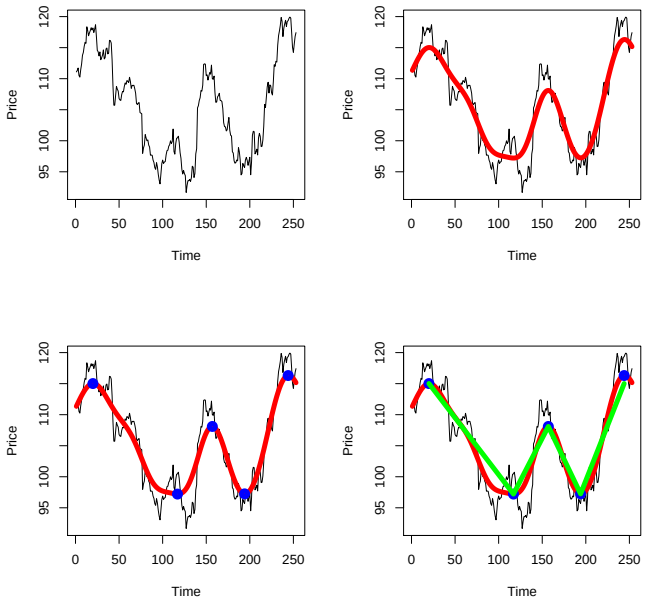


**Figure 7.2** The original data (upper left), the fit of the data when the variance is 4 (upper right), the algorithm identifying the local extrema (lower left), the $W$ found by the algorithm (lower right)

found in the data. What was significant about these two shapes was that the algorithm was able to identify that they were two types of $W$'s, or basically assigning categorizing constants to them. We can see that while the $W$ on the left has a following segment that has a negative slope, the $W$s on the right both have a following segment with a positive slope. Therefore, if the algorithm were able to group these $W$'s based on their constants, we would hope to find that these trends are consistent with other similar $W$s. Similar comments can be make about Figures 7.4 and 7.5. Multiple types of these shapes were found and were labeled as different by the algorithm.

The fact that the algorithm is able to find the different varieties of a shapes using the same definition illustrates how the geometric definition allows the algorithm to be more flexible. We do not have to worry about the data not being perfect or shapes being slightly different from each other because the algorithm will still be able to identify them. Additionally, the algorithm allows for an error when calculating whether something is almost

perpindicular or almost parallel. The higher we allowed the error to be, the less restrictive we were with the condition of a shape. This means that we can concretely mesure how flexible we will allow the definitions to be.

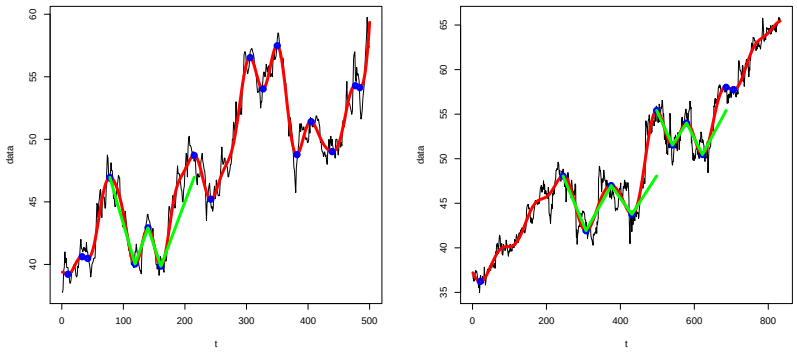


**Figure 7.3** A fit with variance = 0.001 where multiple $W$s were found

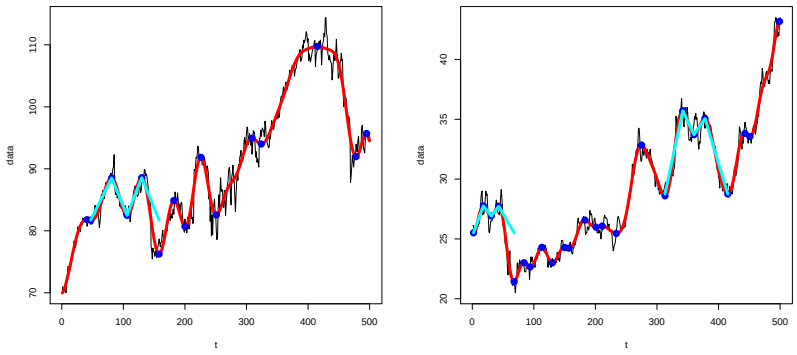


**Figure 7.4** A fit with variance = 0.001 where multiple $M$s were found

## 7.2 Predictions

I used the data from Apple, Disney, Microsoft, Walmart and the NASDAQ as my learning set for both of my prediction methods and I used the Dow Jones index as my test data. The results from each of the methods is given below.

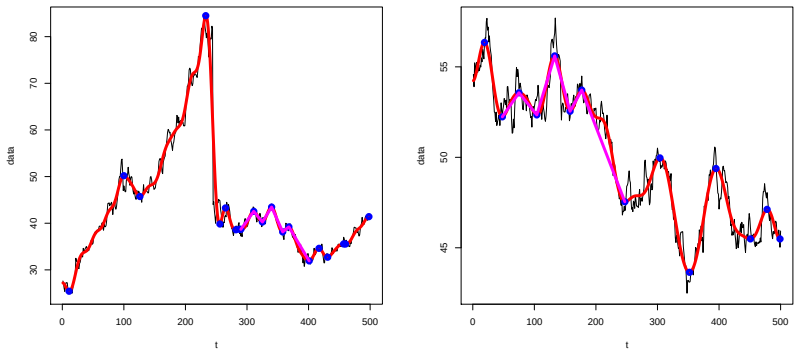


**Figure 7.5** A fit with variance = 0.001 where multiple head and shoulder shapes were found

### 7.2.1 Lengths and Slopes

Once the algorithm was able to recognize shapes, I had it perform the weighted averaging using the lengths and slopes of the segments that followed each shape. In Table 7.1, we can see what each of the expected lengths and slopes for each shape were. The *W* and *M* shapes had roughly the similar expected lengths while the head and shoulder shape had a much longer length expected. All the shapes had a positive slope expected, but at varying degrees. Using these expected values, I wanted to compare them

| | *W* | *M* | Head and Shoulder |
|---|---|---|---|
| Expected Length | 21.304 | 22.301 | 31.779 |
| Expected Slope | 0.246 | 0.124 | 0.032 |

**Table 7.1** The expected lengths and slopes for each of the shapes as calculated by the algorithm

to what was found in the experimental data. The results from each of the shapes are given in Tables 7.2, 7.3, and 7.4. To compare the slopes and lengths to the expected, the percentage of the found was taken out of the expected value. We can see that there are some large percentages in all of the data, especially the lengths. If the percentage of the observed slope out of the expected slope was negative, that means the that these two slopes had opposite directions. Likewise, if the percentage of the two is positive, the

observed slope and expected slope are the same direction. Just looking at Table 7.2, we can see that some percentages of the slopes are as large as 615 and as small as -917, implying that this method was not very accurate or helpful in trying to understand the lengths of the segments that follow the shapes.

Another issue with this method is trying to understand the difference in slopes. What a difference between slopes means is not easily quantifiable. The signs of the percentage value indicate if a slope was in the opposite direction, but it is difficult to measure just how far away it is. Understanding

| $W$ Shape | | | |
|---|---|---|---|
| Slope | Percentage of Expected | Length | Percentage of Expected |
| -0.055 | -22.358 | 24.036 | 112.824 |
| 0.056 | 22.764 | 13.020 | 61.115 |
| -0.412 | -167.480 | 44.348 | 208.167 |
| -0.229 | -93.089 | 45.135 | 211.862 |
| 1.515 | 615.854 | 61.711 | 289.669 |
| -0.539 | -219.106 | 21.583 | 101.310 |
| -0.159 | -64.634 | 14.176 | 66.541 |
| 1.491 | 606.098 | 46.682 | 219.123 |
| 1.427 | 580.081 | 20.913 | 98.165 |
| -0.628 | -255.285 | 18.898 | 88.706 |
| 2.258 | -917.886 | 46.916 | 220.222 |

**Table 7.2** The slopes and lengths calculated for the $W$ shapes and how they compare to the expected values in Table 7.1

that this is not an easy thing to quantify, we rely more on the next predicative method with a unit vector is much easier to measure.

### 7.2.2 Direction Vector

Unlike the previous method, using the directional vector made it much easier to quantify difference. A found directional vector was measured against the expected by finding the angle between them. This measurement is also very easily put into a range in order to understand how far away these vectors are from each other. Since all the vectors are going forward in time, none of them will be in the second or third quadrant. Therefore, the furthest that

| $M$ Shape | | | |
|---|---|---|---|
| Slope | Percentage of Expected | Length | Percentage of Expected |
| -0.058 | -46.774 | 23.039 | 103.309 |
| 0.074 | 59.677 | 18.050 | 80.938 |
| -0.062 | -50.000 | 17.032 | 76.373 |
| -0.238 | -191.935 | 21.585 | 96.789 |
| 0.019 | 15.323 | 10.002 | 44.850 |
| -0.009 | -7.258 | 6.000 | 26.905 |
| -0.288 | -232.258 | 26.014 | 116.649 |
| -0.107 | -86.290 | 26.148 | 117.250 |
| 1.282 | 1033.871 | 65.027 | 291.588 |
| -0.545 | -439.516 | 21.638 | 97.027 |
| -1.119 | -902.419 | 31.515 | 141.317 |
| -2.113 | -1704.032 | 60.784 | 272.562 |
| 1.967 | 1586.290 | 46.339 | 207.789 |
| 0.842 | 679.032 | 36.608 | 164.154 |
| 0.120 | 96.774 | 15.108 | 67.746 |

**Table 7.3** The slopes and lengths calculated for the $M$ shapes and how they compare to the expected values in Table 7.1

| Head and Shoulders Shape | | | |
|---|---|---|---|
| Slope | Percentage of Expected | Length | Percentage of Expected |
| 0.127 | 396.875 | 21.169 | 66.613 |
| 0.175 | 546.875 | 22.335 | 70.282 |
| -0.144 | -450.000 | 23.238 | 73.123 |
| -0.747 | -2334.375 | 23.715 | 74.625 |
| -0.025 | -78.125 | 5.002 | 15.740 |
| 2.350 | 7343.750 | 204.301 | 642.881 |
| -1.888 | -5900.000 | 53.408 | 168.061 |
| 0.156 | 487.500 | 8.096 | 25.476 |
| 3.985 | 12453.125 | 110.940 | 349.098 |

**Table 7.4** The slopes and lengths calculated for the head and shoulders shapes and how they compare to the expected values in Table 7.1

two vectors can be from each other is an angle of $\pi$. However, it would be

safe to say that any vectors that are more than $\pi/2$ away from each other are too far.

| $W$ | $M$ | Head and Shoulder |
|---|---|---|
| (0.866, -0.203) | (0.859,-0.113) | (0.499,0.554) |

**Table 7.5** The expected directional vectors calculated for each shape

We can see the expected vectors for each of the shapes in Table 7.5. It is worth noting that both the $M$ and the $W$ shape are expected to have some sort of negative slope follow them, a contradiction from the previous prediction. Additionally, we can see that the head and shoulder will have a positive direction vector following it. The results from the observed vectors in the data are shown in Tables 7.6, 7.7, and 7.8. $\pi/2 \approx 1.57$ and most of the angles found are less than that value, a good sign. We can see that for the $W$ shapes, the vectors that we found were on average 0.502 radians away from the expected. For the $M$ shape, the difference was on average 0.393 radians and for the head and shoulders shape, the average difference was about 0.39. This is a promising result as these vectors we found are anywhere between 20 and 28 degrees from the expected value, which is significantly better than what we found with the previous predictive algorithm. Some ways to improve the accuracy of this method are discussed in the next section.

## 7.3 Potential Issues

While the algorithm was successful in finding the shapes and doing a rough prediction, there were still some issues with it. For example, finding a good tolerance for finding a shape was not exact. I had to run the algorithm many times to determine at what point the algorithm would find identify a pattern was not the correct shape. This usually happened because the tolerance was too high and the patterns found did not look like the shape. Likewise, when the tolerance was too low, it would identify no shape. Finding an appropriate tolerance (somewhere between 0.7 and 1) took a while and the tolerance for each shape was different as the $W$s had a lower tolerance while the head and shoulder pattern needed a higher tolerance.

Another issue with this algorithm was its dependency on the Gaussian Process. Even though I was using the same data, code and variance, sometimes

| $\vec{\mu}$ found for $W$ | |
|---|---|
| Found | Difference from Expected |
| (0.999,-0.049) | 0.182 |
| (0.100,-0.002) | 0.228 |
| (0.976,-0.218) | 0.010 |
| (0.991,-0.133) | 0.097 |
| (0.564,0.826) | 1.202 |
| (0.973,-0.229) | 0.001 |
| (0.981, -0.192) | 0.038 |
| (0.935,-0.356) | 0.133 |
| (0.493,0.870) | 1.285 |
| (0.764,0.645) | 0.931 |
| (0.373,0.928) | 1.412 |

**Table 7.6** The directional vectors calculated for each $W$ shape and how they compare to the expected values in Table 7.5. The mean of the differences is 0.502 with a standard deviation of 0.575

| $\vec{\mu}$ found for $M$ | |
|---|---|
| Found | Difference from Expected |
| (0.999,-0.054) | 0.077 |
| (0.100,-0.003) | 0.128 |
| (0.987,-0.159) | 0.029 |
| (0.100,-0.016) | 0.115 |
| 0.918,-0.397) | 0.278 |
| (0.995,-0.102) | 0.029 |
| (0.917,-0.399) | 0.279 |
| (0.982,-0.188) | 0.058 |
| (0.528,0.849) | 1.146 |
| (0.449,0.894) | 1.236 |
| (0.899,-0.437) | 0.321 |
| (0.411,-0.912) | 1.017 |

**Table 7.7** The directional vectors calculated for each $M$ shape and how they compare to the expected values in Table 7.5. The mean of the differences is 0.393 with a standard deviation of 0.460

| $\vec{\mu}$ found for Head and Shoulders | |
|---|---|
| Found | Difference from Expected |
| (-2.309 $\times 10^{-6}$,1.00) | 0.733 |
| (-8.329 $\times 10^{-4}$,1.00) | 0.734 |
| (7.071 $\times 10^{-1}$,7.071 $\times 10^{-1}$) | 0.052 |
| (7.091 $\times 10^{-1}$, 7.051 $\times 10^{-1}$) | 0.055 |
| (7.071 $\times 10^{-1}$, 7.071 $\times 10^{-1}$) | 0.052 |
| (9.924 $\times 10^{-1}$, 1.230 $\times 10^{-1}$) | 0.714 |

**Table 7.8** The directional vectors calculated for each head and shoulders shape and how they compare to the expected values in Table 7.5. The mean of the differences is 0.39 with a standard deviation of 0.370

the Gaussian Process would do a slightly different fit which would then lead to having different local extrema which would end in a shape not being found. An example of this would be Figure 7.6, in which the same code was run on the same data (both with variance 4) but two different sized $W$s were found. The ability of the GP to fit the data appropriately really restricted

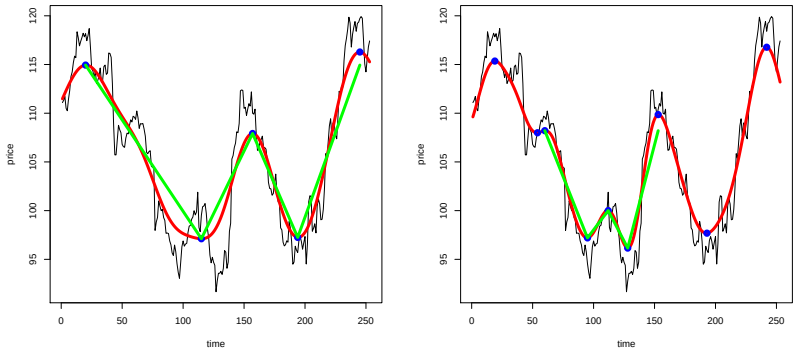


**Figure 7.6** Both of these trials were given a variance of 4 but the GP defined two different fits with different local extrema which gave very different $W$ patterns

how well the rest of the algorithm worked. We can overcome this problem by using the same variance, getting different fits, and unioning the sets for the different shapes that are found.

Finally, the predictive step of the algorithm has a long way to go. Using the slopes and lengths method did not work well which might be explained by having no good way of measuring the difference between slopes. Additionally, the weighted average method might not be enough to get a good understanding unless I had a lot more data. I had about 55 shapes for each type of shape when I ran the algorithm on the data. If I want to get a better average, I would need to run my code on a lot more data in order for the average to not be swayed as heavily by outliers. The directional vector method worked much better than the slopes and lengths method, but there was still a error of about 20-28 degrees. We could potentially decrease this error by simply getting more data. This prediction also indicated that there actually might be a correlation between the shapes and the directional vector, so perhaps we could use more traditional machine learning methods on these directional vector values to see if we can learn anything extra from it. For example, is there some sort of correlation between the $k$ values we found in the slope/lengths predictions and the directional vector we calculated? A question like this might be answered by using a neural network or a support vector machine.

# Chapter 8

# Conclusion

The goal of this project was to develop a geometric definition that would be invariant under size and rotation. From the results in the previous chapter, it is clear that the definition was successfully used to find the same shape using the same definition without limiting the shape to certain sizes. This invariance came from the fact that the shapes were not defined by the numerical values of the inputs but instead the basis that could be made from those inputs. The geometric property of the definitions allowed us to create a metric to measure how different shapes were without worrying about the difference of their sizes or rotations. The only limitation of the use of this definition came from the fact that the Gaussian Process was not consistent about giving a fit. While the predictive part of the algorithm is lacking, there is a lot of room for growth, perhaps with more traditional machine learning methods.

## 8.1 Future Work

There are many different ways that this work can be extended. The algorithm that I have already developed has many different ways it can be improved upon so that the prediction works better. This project was also a two-dimensional test case of how geometry can be used with machine learning. There are pleanty of datasets that are in multiple dimensions, so using differential geometry to define certain patterns is an extension of the work I have already done. More specifics on these two different topics are given in the following subsections, including some of the work I've already done to extend my thesis.

### 8.1.1 Improving Shape Recognition

One of the biggest flaws with this algorithm is that is so heavily depends on how the Gaussian Process fits the data. While the GP allows for flexibility on how tight or loose the data is fit, it is not consistent on how it fits the data even if the variance is not changed. An improvement to this algorithm might be finding a different way to fit the data and find the local extrema such that it is more consistent. If that improvement were made to the algorithm, it would be much more consistent on finding shapes of a certain size in the data.

My algorithm is currently able to find the different patterns within the data. I want to be able to add on to my algorithm so that it will be able to make preditions based off of the shape and find out what patterns tend to follow other patterns. Since I currently lack shapes within the dataset for a few companies, it might be helpful to look at shapes within even more companies as well and combine them together to get a large dataset of shapes so that we could find more than 50 shapes at a time. The current prediction that I propose is just a weighted average and it might be helpful to use a other machine learning methods (SVM, ANN) on the shapes I find.

### 8.1.2 Extending into Multiple Dimensions

In addition to this, I want to take this type of algorithm and try to apply it to mutli-dimensional data rather than 2-dimensional data. I have begun extending my work into this area by trying various different methods of fitting multi-dimensional data. I decided to work with the price, time, and volume of the stock as my 3-dimensional case. Figure 8.1 has the scatter plot of the time, price and volume for Apple stock data as well as the parametric curve that shows the fitted values of price and volume over time. Some of the ways that this can be extended could be to fit a surface to the scatterplot and try to have certain surface geometric features be found by the algorithm. Some issues I ran into while attempting this was trying to fit a curve to the data. Because the data is relatively clustered, most of the surfaces fit to the data are planes and are not a good representation. There is a lot of research done with how to fit a surface to data, so I would probably try to look into other kinds of methods.

Another way to handle the 3-dimensional data is to represent it as a para-

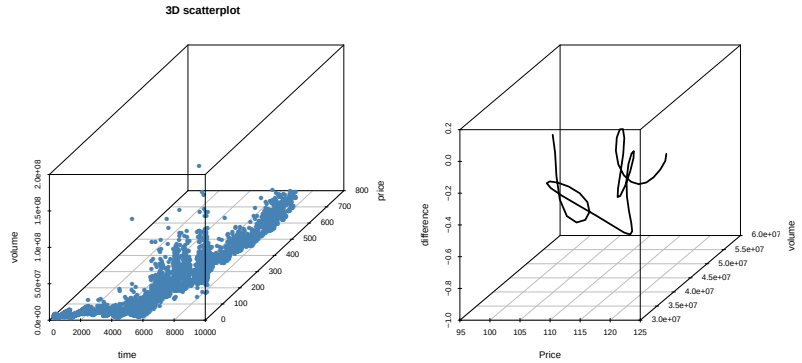


**Figure 8.1** A scatterplot and parametric curve representation of the time, price and volume for Apple stock data

metric curve and have the algorithm find definitions and properties from differential geometry. There might be certain aspects of curves we can look for in the data and possibly use it as a predictive measure. One such property to consider would be different distributions at each point on a curve. For example, I considered the curve of the price of the stock over time. At every 10 prices, I would calculate the estimated Gaussian distribution of those prices against the volume over the same time period. The plots of the means and standard deviations for each of those found distributions are shown in Figure 8.2. This shows that at around the year 2008, the mean and standard

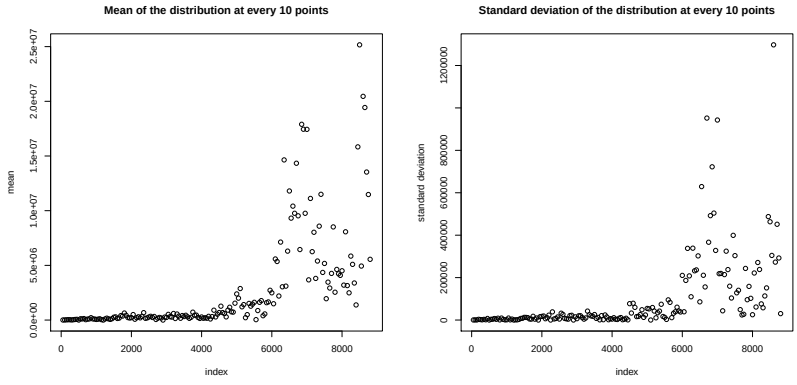


**Figure 8.2** The mean and standard deviation for each distribution found at every 10 points

deviation for the found distributions have a very high variance. Using traditional machine learning techniques, it might be interesting to model this data and see why these distributions change so much over time. Another way to extend this research would be to take the differences between the different distributions found and find the rate at which they change.

## 8.2 Closing Thoughts

Above are the different ways that this research can be extended into other areas of differential geometry. In my work this year I have shown that creating geometric definitions can be helpful in the area of machine learning. Instead of an algorithm being entirely dependent on the numerical entries of data, it can learn from the geometric shape of a pattern, making it invariant under size and rotation. While this year I exclusively worked with geometric shapes that are in two dimensions, above I have shown that these kinds of definitions and approaches can be easily extended into multiple dimensions, hopefully with just as promising of results.

# Bibliography


Bishop, Christopher M. 2006. Pattern recognition. *Machine Learning* 128.

Burges, Christopher JC. 1998. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery* 2(2):121–167.

Forex, Staff. 2016. Forex chart pattern trading on double top and double bottom. Available online at forexfunction.com. URL http://www.forexfunction.com/forex-chart-pattern-trading-on-double-top-and-double-bottom.

Halls-Moore, Michael L. 2016. *Advanced Algorithmic Trading*.

Hassan, Md Rafiul, and Baikunth Nath. 2005. Stock market forecasting using hidden markov model: a new approach. In *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, 192–196. IEEE.

Investopedia, Staff. 2016. Investopedia. Available online at investopedia.com. URL http://www.investopedia.com/terms/d/doublebottom.asp.

Kamijo, Ken-ichi, and Tetsuji Tanigawa. 1990. Stock price pattern recognition-a recurrent neural network approach. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, 215–221. IEEE.

Kawagoe, Masahiro, and Akio Tojo. 1984. Fingerprint pattern classification. *Pattern recognition* 17(3):295–303.

Lawrence, Steve, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks* 8(1):98–113.

Mihail, Morosan. 2016. Simple js neural network. Available online at morosanmihail.com. URL http://www.morosanmihail.com/home/project/simple-js-neural-network.

Murphy, Kevin P. 2012. *Machine Learning: A probabilistic Perspective*. The MIT Press.

Schiffmann, Raphael, and Marjo S van der Knaap. 2009. Invited article: an mri-based approach to the diagnosis of white matter disorders. *Neurology* 72(8):750–759.

StockCharts, Staff. 2016. Head and shoulders top (reversal). Available online at stockcharts.com. URL http://stockcharts.com/school/doku.php?id=chart_school:chart_analysis:chart_patterns:head_and_shoulders_top_reversal.

Wood, Jeffrey. 1996. Invariant pattern recognition: a review. *Pattern recognition* 29(1):496–498.