THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

MACHINE LEARNING TECHNIQUES FOR EARLY DISEASE DETECTION IN POTATO PLANTS

PHILIP JOHN RYAN JR.
FALL 2017

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Zhiwen Liu
Professor of Electrical Engineering
Thesis Supervisor

Julio Urbina
Associate Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

The standard method of diagnosing early blight disease in potato plants is a visual inspection. This system is insufficient because potato plants can become contagious before they show visual symptoms of disease. Optical spectroscopy may prove to be a superior alternative; however, computer programming is needed to determine plant health from measurements. This thesis proposes a software solution to be used with mobile optical sensing equipment from ATOPTIX, Inc. The solution uses principal component analysis (PCA) to reduce the dimensions of the data. The two principal components that best model the data are plotted and the centroid of the resulting data cluster is computed. For this experiment, specific rows of potato plants were directly inoculated with early blight; these were the spreader rows. The disease then spread to the remaining rows, which were marked by orange flags. In just one day after inoculation, the percent difference between the healthy cluster of data and the diseased cluster of data exceeded 250% for the spreaders. For the remaining plants, the percent difference between the healthy and diseased clusters exceeded 100% three days after the spreaders were inoculated. Based on these results, PCA could be an effective way of determining plant health, especially if used in conjunction with other machine learning techniques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Studying Electrical Engineering and writing this thesis have been the most ambitious, challenging, and rewarding academic goals I have ever set for myself. I could not have been successful in completing these goals without guidance and support from all of my mentors. I would like to thank the following incredible people for all of their help along the way:

Professor Zhiwen Liu, my thesis supervisor, for providing guidance and wisdom from start to finish. Thank you for being an intellectual and professional role model, and for having a vested interest in my success.

Professor Julio Urbina, my honors advisor, for reviewing my thesis and providing feedback.

Perry Edwards, Ph.D., and Victor Bucklew, Ph.D., for helping me develop an idea for my thesis and supporting me through the process. Thank you for allowing me to use the proprietary equipment you developed at ATOPTIX, Inc. and for giving me feedback and advice.

Chen Zhou for helping me find accessible literature and guidance.

David Salvia and Svetla Jivkova for inspiring me to study Electrical Engineering and helping me discover my technical interests. You both provided me with excellent guidance and are easily the best instructors I have ever had.

My swim coaches, especially Todd Bauer and Caity Fisher, who taught me to believe in myself, to set goals for myself, and to hold myself accountable for achieving those goals.

My teachers, especially Chris Luck, Christine Jackson, and Carolyn Scott, who helped me realize my academic capabilities and encouraged me to pursue a career in engineering.

My family, especially my parents, for always supporting me, for helping me maintain a positive perspective, and for providing constructive criticism. I would also like to thank the engineers in my family, Bill Sickles, P.E., Stephen Ryan, P.E., Lorraine Ryan, and Jessica Shaw, Ph.D., for their professional guidance.

My friends for their support and encouragement, and for sticking by me during the inevitable rough patches. Thank you for helping me enjoy every aspect of my Penn State experience.

Chapter 1

Introduction

Agriculture is important to human health and global economies. Humans depend on farmed crops for nourishment. In addition, the agricultural industry provides sources of livelihood and economic opportunity – especially for innovative entrepreneurs [1]. Take, for example, the potato plant. In the United States alone, the production of potato crops was valued at $3.923 billion in 2016 [2]. Plant diseases, such as early blight and late blight for potato crops, pose significant health and economic risks. Widespread disease among a crop is particularly devastating in developing nations, where locally grown foods are the primary source of nourishment [3]. These diseases can also disturb the economy and result in loss of agricultural revenue.

The current standard for identifying the health of plants is visual inspection [4]. This method is insufficient because plants may begin to spread the disease to the rest of the crop before showing visual evidence of infection [5]. Optical sensing has the potential to be a superior alternative to traditional visual inspections; however, a method to classify the measurements to indicate plant health or disease is needed. The objective of this thesis is to correlate the optical spectra from a potato plant with its health. To accomplish this, a software code that uses unsupervised machine learning algorithms is needed. Once this is accomplished, farmers can use this correlative information for early disease detection.

The proposed solution utilizes a mobile optical sensor created by ATOPTIX, Inc. The sensing data is output in the form of an array containing received intensity values at different

wavelengths. The solution begins by processing this data – removing any observations that are not within one standard deviation of the mean. Next, principal component analysis (PCA) is used to reduce the dimensions of the processed data. The first two PCA scores are plotted to visualize the data. Just one day after the plants become infected with early blight, one can differentiate between the cluster of healthy data points and unhealthy data points. After three days, the difference becomes more apparent, with percent differences between healthy and unhealthy cluster centroids exceeding 100%.

Chapter 2

Literature Review

Much of the theory discussed in the literature review was presented by Andrew Ng as a part of Stanford University's Open Classroom. I would like to acknowledge him for making the content accessible to students without a background in software engineering.

2.1 | Overview of Machine Learning

Machine learning is becoming an increasingly popular method for data analysis, largely due to the ease with which modern technology can collect information. The idea of machine learning has existed since the mid-Twentieth century, but experts have not agreed on an official definition. Arthur Samuel, a software engineer known for artificial intelligence and computer gaming, describes it as the field of study that "gives computers the ability to learn without being explicitly programmed" [6].

Many machine learning algorithms have been developed; however, all of these algorithms belong to one of three categories: supervised learning, unsupervised learning, and reinforcement learning [7]. Supervised learning utilizes a "training" data set consisting of inputs and desired outputs. The algorithm uses this data to map the inputs to the outputs, and then applies the mapping to new sets of data for which the correct answers are unknown. In unsupervised learning, the training set consists only of inputs – the algorithm must assign a structure to the data. Reinforcement learning occurs in a dynamic environment in which the

algorithm receives feedback as it attempts to achieve a pre-determined goal [7]. Reinforcement learning is beyond the scope of this thesis.

I will now cite some common examples of supervised and unsupervised learning to help differentiate between the two. Oncologists use supervised machine learning to help determine the probability a given tumor is benign or malignant [8]. This type of machine learning algorithm would use a training set of data consisting of tumor features as inputs (such as size) and a binary output (malignant or benign). In this case, the algorithm receives a set of "correct answers" and creates a mapping between the features and the binary output. Oncologists can use this mapping to compute the probability that a tumor (whose malignancy is unknown) is malignant [8]. Using machine learning to identify segments within a market is an example of unsupervised learning. This type of algorithm would assign a structure (often clusters) to input features and output this structure to the user for interpretation [9].

In addition to supervised and unsupervised learning, there are also, in general, two types of problems which machine learning algorithms seek to solve: regression and classification. The output the learning algorithm produces for regression problems is continuous [10]. An example of this would be predicting the sale price of a house given the size of the property and the number of bedrooms [10]. For classification problems, the outputs of the learning algorithm are discrete in value. In the previous oncology example, the training set outputs, which are given to the learning algorithm, have two discrete values: malignant and benign [8].

2.2 | Linear Regression

Linear regression is a method of statistical analysis that linearly relates a dependent variable (outcome) with one or more independent variables. The linear function that maps the independent variable(s) to the dependent variable is created from initially unknown parameters. The parameters that create the mapping are determined from the data by minimizing the cost function [11]. The cost function is the difference between the hypothesis and the actual value.

Figure 2.1 shows a high-level block diagram of how a linear regression algorithm is implemented to solve a regression problem. The training set of "correct answers" is passed to the learning algorithm. The learning algorithm develops a hypothesis that will map the inputs from the training set to the desired outputs.

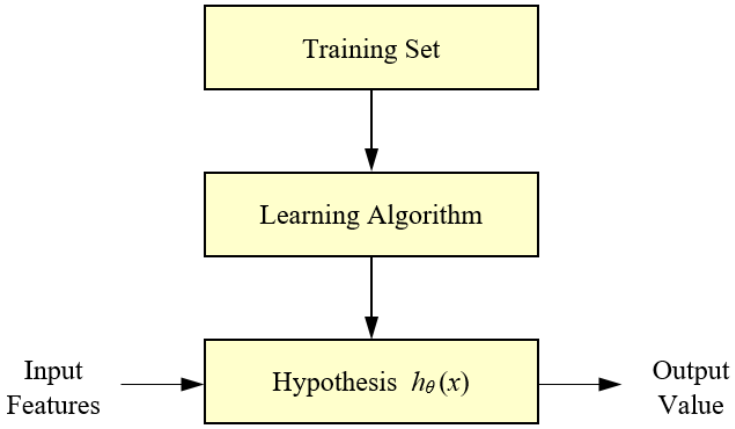

Figure 2.1: Block Diagram of Linear Regression Algorithm

Once this is complete, new inputs are passed to the hypothesis. The outputs are assigned on the basis of the hypothesis [12]. Table 2.1 establishes common notation that will be used for the remainder of the thesis.

Table 2.1: Literature Review Notation

Variable	Definition
m	number of training examples
x	input variables
y	output/target variables
θ	algorithm parameter
(x, y)	training example (one instance)
$(x^{(i)}, y^{(i)})$	i^{th} training example
θ_j	j^{th} algorithm parameter

As the name suggests, a linear regression algorithm seeks to define a linear hypothesis that maps the inputs to the outputs. Equation (2.1) shows this for a training set with a single feature. The hypothesis for a training set with *m* features is given by Equation (2.2) [12].

$$h(x) = \theta_0 + \theta_1 x_1 \tag{2.1}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m \tag{2.2}$$

A fundamental algorithm for implementing linear regression is gradient descent. Gradient descent begins by initializing values for $\vec{\theta}$ and computing an initial linear hypothesis $h_\theta(x)$. The algorithm computes the average square error for each measurement in the training set, denoted by the cost function $J(\theta)$ given by Equation (2.3) [13].

$$J(\theta) = \frac{1}{2} \cdot \frac{1}{m} \sum_{i=1}^{m} \left[h_\theta\left(x^i\right) - y^i \right]^2 \tag{2.3}$$

Gradient descent computes new values for $\vec{\theta}$ based on Equation (2.4), where $:=$ denotes assignment and α is the learning rate [13].

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \text{ for all } j \tag{2.4}$$

Equation (2.4) iterates until convergence – that is until values for $\vec{\theta}$ are obtained such that the cost function is minimized. The value of $\vec{\theta}$ that gives the local minima will produce the hypothesis that best maps the inputs to the outputs [12].

It is an accepted practice to scale the data in a training set with multiple features before attempting to implement gradient descent. Consider the earlier example of predicting the sale price of a house based on the property size and number of bedrooms. The first feature, property size, is likely reported in square feet, and would be several orders of magnitude greater than the number of bedrooms. If this training data were passed into the gradient descent algorithm without scaling, the contour plot of the cost function would be extremely skewed [12]. This makes it difficult (perhaps impossible) for the algorithm to converge. A popular method for scaling the data is normalization, in which the difference between each data point and the mean is computed, then divided by the standard deviation, as shown in Equation (2.5):

$$x_1 = \frac{x_1^i - \mu_1}{s_1} \tag{2.5}$$

where x_1 is the set of measurements of the first feature, μ_1 is the mean of the set, and s_1 is the scaling factor of the set. The scaling factor is usually the range or standard deviation of the measurements in the set [13].

2.3 | Polynomial Regression

Sometimes the training data is best modeled with a polynomial expression, rather than a linear hypothesis [14]. Equation (2.6) gives the expression for a cubic hypothesis for a training set with one feature, *x*:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \quad (2.6)$$

where $x_1 = x$, $x_2 = x^2$, and $x_3 = x^3$. This will create a hypothesis that resembles a cubic polynomial, but since the equation itself is a linear combination of $x_1, x_2,$ and x_3, the gradient descent algorithm can still be used [12].

2.4 | Logistic Regression

Classification problems can present a challenge to linear regression algorithms. Sometimes, classification problems can be accurately modeled using linear regression. There are many instances, however, where linear regression is not capable of modeling the data, especially if the range in the training set is comparatively large [15]. Logistic regression is a learning algorithm used to solve classification problems. We often want to determine the whether a data point should be classified in a specific category. For example, email spam filtering uses features of the email and computes the probability that the given message is spam [16].

To construct the hypothesis using logistic regression, the Sigmoid/Logistic function is used. The Sigmoid function is appropriate for the hypothesis because its value ranges from 0 to 1 from negative to positive infinity. This is particularly useful for classification because we want to determine the probability that unknown data belongs to a certain category/classification. The Sigmoid function can be written mathematically as shown in Equation (2.7) on the next page:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.7}$$

where *z* has been used so as not to be confused with the observations *x*. Equations (2.8) and (2.9) compare the hypotheses of Linear Regression and Logistic Regression, respectively:

$$h_\theta(x) = \theta^{\mathrm{T}} x \tag{2.8}$$

$$h_\theta(x) = g(\theta^{\mathrm{T}} x) \tag{2.9}$$

Substituting the hypothesis for logistic regression into the sigmoid function gives Equation (2.10):

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^{\mathrm{T}} x}} \in [0,1] \tag{2.10}$$

The hypothesis now estimates the probability that the output is 1 for a given input *x*. The cost function can be computed using Equation (2.11):

$$\text{cost} = \begin{cases} -\log\big(h_\theta(x)\big), & y = 1 \\ -\log\big(1 - h_\theta(x)\big), & y = 0 \end{cases} \tag{2.11}$$

2.5 | Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction algorithm that maps *n*-dimensional data to another *n*-dimensional coordinate system. The new coordinate system maximizes variation among observations [17]. First, a geometric description of PCA is presented, followed by mathematical evidence. The algorithm begins by centering and scaling the data in *n*-dimensional space. A line of best fit is constructed that minimizes the error from projecting each point in *n*-dimensional space onto the line. This maximizes the variance along the new axis, which is referred to as the first principal component [17]. This can be repeated for

k dimensions, where $k \leq n$. Figure 2.2 gives a geometric representation of PCA for a three-dimensional set of observations.

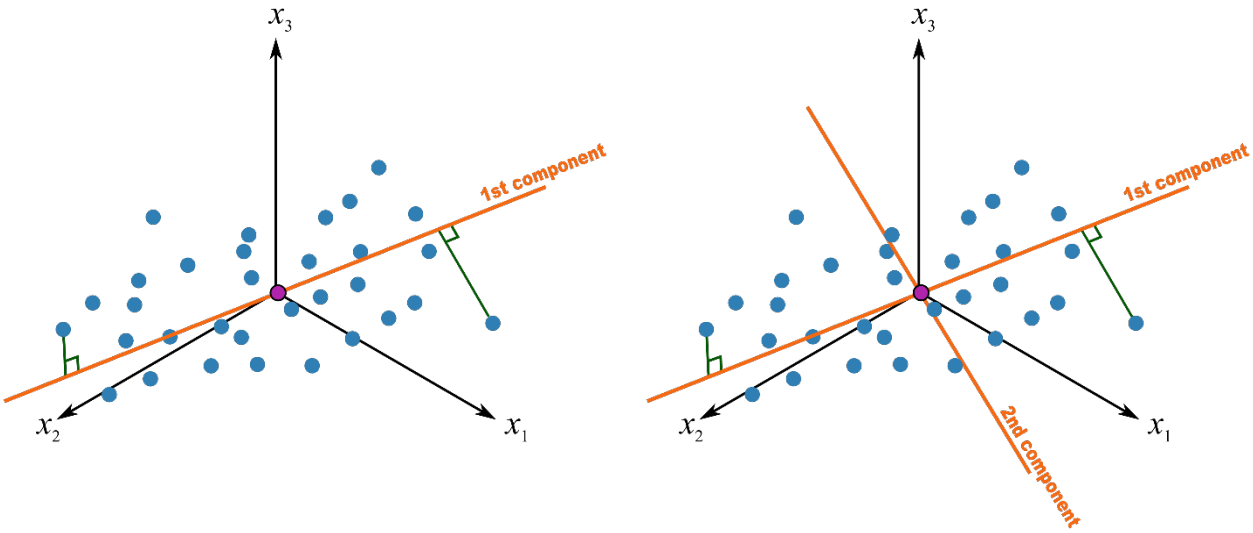

Figure 2.2: Geometric Representation of PCA in Three Dimensions

Image Source: K. Dunn, "Geometric explanation of PCA," Process Improvement Using Data, 4 August 2017. See Reference [17] for URL.

When performing PCA, it is often useful to compute the score for each principal component. The score is the distance from the origin to the data point's projection onto the principal component. For the purposes of data visualization, it is common to plot the first two or three scores in the plane [17].

Before PCA is applied to the set of observations, the data must be pre-processed. First the mean of the data must be zeroed. This is accomplished by computing the mean of the data set, then subtracting off the mean from each measurement, as shown in Equations (2.12) and (2.13) [18].

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \tag{2.12}$$

$$x^{(i)} := x^{(i)} - \mu \tag{2.13}$$

Next, rescale each coordinate to have a consistent unit variance, as shown in Equations (2.14) and (2.15). This will ensure that different attributes about the observations will be analyzed on the same scale [18].

$$\sigma_j^2 = \frac{1}{m}\sum_i \left(x_j^{(i)}\right)^2 \tag{2.14}$$

$$x_j^{(i)} := \frac{x_j^{(i)}}{\sigma_j} \tag{2.15}$$

If unit vector u represents a principal component of the observations x, the length of the projection of x onto u is given by $x^{\mathrm{T}}u$. Due to the zeroing of the mean, if $x^{(i)}$ is an observation, its projection onto u is the distance $x^{\mathrm{T}}u$ from the origin. To maximize variation among observations, choose u to maximize Equation (2.16), [18].

$$\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)\mathrm{T}}u\right)^2 = \frac{1}{m}\sum_{i=1}^{m} u^{\mathrm{T}}x^{(i)}x^{(i)\mathrm{T}}u = u^{\mathrm{T}}\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)\mathrm{T}}\right)u \tag{2.16}$$

Maximizing gives the principal eigenvector of $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)\mathrm{T}}$. Therefore, to find a one-dimensional modeling subspace, select *u* to be the principal eigenvector of Σ. To project the observations onto a *k*-dimensional subspace, select $u_1, u_2, \ldots, u_k$ to be the top *k* eigenvectors of Σ. The mapping is given by Equation (2.17), where $y^{(i)}$ is the data from the first *k* principal components [18].

$$y^{(i)} = \begin{bmatrix} u_1^{\mathrm{T}}x^{(i)} \\ \vdots \\ u_k^{\mathrm{T}}x^{(i)} \end{bmatrix} \epsilon\, \mathbb{R}^k \tag{2.17}$$

2.6 | Unsupervised Learning

In supervised learning algorithms, the software uses a set of labeled training data to map the inputs to the correct outputs. In unsupervised learning, the training set is *unlabeled*, and the software must assign a structure to the data. Equations (2.18) and (2.19) illustrate the difference between these two types of training sets. In Equation (2.18), the training set contains *x* and *y* values, where *x* is the feature and *y* is the desired output. In Equation (2.19), only the features are given [19].

$$\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}\} \tag{2.18}$$

$$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\} \tag{2.19}$$

Clustering is a method for organizing the unlabeled data in the training set. One of the most popular clustering algorithms is the *k*-means clustering algorithm. This method requires the user to define the number of clusters *k* to which the data may be assigned. There is not necessarily an effective way for computer software to select the value of *k* – it is often determined from the nature of the problem. Consider a set of sizing information collected by a clothier. If this unlabeled training set consists of customer waist and chest sizes, the clothier may want to group the measurements into three clusters that will represent shirt sizes (small, medium, and large) [19].

Similar to gradient descent, the *k*-means algorithm is an iterative algorithm. Figure 2.3 on the following page gives a visual representation of the *k*-means algorithm.

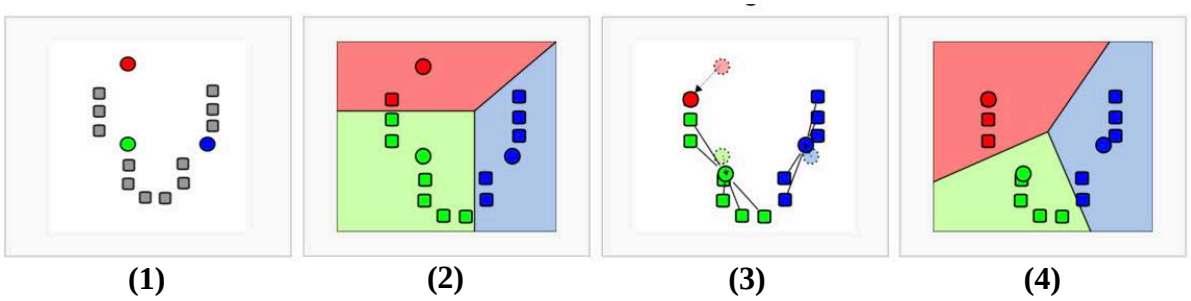

Figure 2.3: A visual representation of the *k*-means algorithm

Image Source: N. Landman et al. "k-Means Clustering," Brilliant.org.
See Reference [26] for URL.

The algorithm begins by randomly initializing values for *k* and plotting these values – the cluster centroids – in the plane [20]. This is shown in the first pane of Figure 2.3. The second step is to compute the distance between each data point and each centroid cluster. The data point is assigned to the cluster of the nearest centroid. This is known as the cluster assignment step and is shown in the second pane. The third step is the "move centroid" step. For each of the *k* existing centroids, a new centroid is computed using all of the data points belonging to that specific cluster, as shown in the third pane. Finally, the spatial boundaries of each cluster are adjusted to reflect the locations of the new centroids and the data points belonging to those centroids. This is shown in the fourth pane. The process iterates until convergence, when the change in location of the centroids ceases or becomes negligible [19], [20].

The *k*-means algorithm requires two inputs: *k*, the number of clusters, and the training set. The variables that mathematically describe the *k*-means algorithm are listed in Table 2.2 on the following page.

Table 2.2: *k*-means algorithm notation

Variable	Definition
$c^{(i)}$	index of the cluster to which example $x^{(i)}$ is currently assigned
μ_k	cluster centroid *k*
$\mu_c(i)$	centroid of cluster to which example $x^{(i)}$ has been assigned

The optimization objective of the algorithm is to minimize the square of the distance between each data point $x^{(i)}$ and the centroid of the cluster to which it has been assigned [19]. As with gradient descent, a distortion function $J(c, \mu)$ computes the square of the distance, given by Equation (2.20):

$$J\left(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu_c(i)\right\|^2 \qquad (2.20)$$

Some special consideration must be given to the random initialization of the *k* cluster centroids. Depending on the values that are assigned, the algorithm may organize the data into *k* clusters with a relatively small distortion, or it can force the organization of data into *k* clusters with a relatively large distortion. For the best results, both the initialization process and the *k*-means algorithm should be executed many times (typically 50 – 1000 times) [19]. The iteration that converges with the smallest distortion function is likely the best clustering of the data. If the number of clusters *k* is small, it is especially important to iterate through the random initialization and *k*-means algorithm process [19].

To verify that the correct value of *k* has been selected, repeat the process for varying values of *k*. Plot the convergence value of *J* as a function of the number of clusters *k*. If an "elbow" appears on the curve, as shown in Figure 2.4, the value selected for *k* is satisfactory.

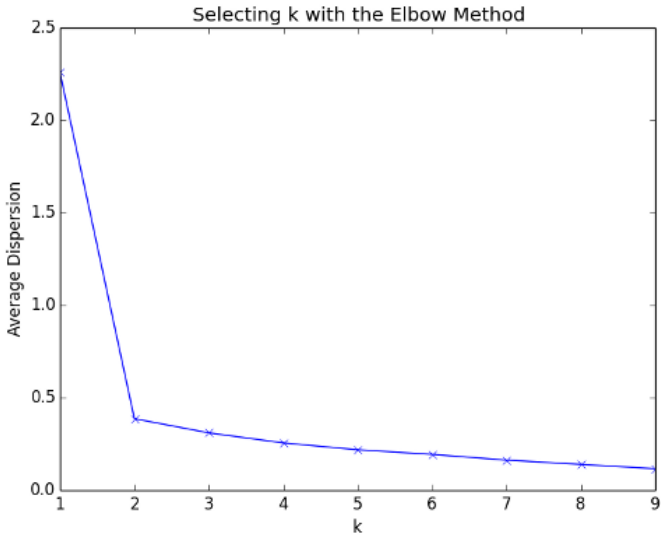

Figure 2.4: Selecting *k* using the "Elbow" method

Image Source: A. Ng, "CS229 Lecture Notes: Unsupervised Learning," Stanford Open Classroom, 2017. See Reference [19] for URL.

In Figure 2.4, the "Elbow" occurs for $k = 2$, indicating an optimal value for k. Sometimes, no elbow appears on the curve. In this case, one must use his/her best judgment in determining the value for k based on the nature of the problem [19].

2.7 | Early Blight

Early blight, or *Alternaria solani*, is a fungus that affects potato plants, especially those grown in humid environments. When a potato plant becomes infected with early blight, small dark spots begin to form on leaves (usually the larger ones that are closer to the soil). These spots will become larger over time and eventually form lesions that are have a dark brown or black pigment. They may even develop in concentric rings and have a target-like appearance [4]. See Figure 2.5 for examples of early blight lesions.

Figure 2.5: Examples of Early Blight

Image Source: B.J. Christ, "Identifying Potato Diseases in Pennsylvania," The Pennsylvania State University, 1999.

A mature lesion will form spores that can be transmitted by wind, rain, irrigation water, or machinery. When a spore lands on the surface of a wet leaf, it produces many zoospores that cover the surface of the leaf. The spores settle and grow into the tissues of the leaf and infect the plant. In warm, humid conditions, small dark spots can form in as little as three days, visually indicating disease. Early blight is less harmful if it occurs later in the growing season, after the tubers have developed. If the disease infects a crop early in the season – before the tubers can form – the crop will fail [5].

Chapter 3

Methods

3.1 | Overview

There are two major phases to the thesis experimentation: data collection and computational analysis. Measurements were collected from different potato plants for 16 days, using a mobile optical sensor created by ATOPTIX, Inc. After the data was collected, a separate program processed the measurements by normalizing the values and removing outliers. Finally, the machine learning script reduced the dimensions of the data using principal component analysis and plotted the first two scores for visualization purposes.

3.2 | Mobile Optical Spectroscopy

A smartphone-based optical spectrometer developed by ATOPTIX, Inc. was used to collect data from potato plants. The spectrometer utilized diffusive reflectance spectroscopy (DRS), which is capable of penetrating into the plant tissue [21]. The light will reflect back to the surface of the leaf after interactions with the plant tissue, including scattering from cellular structures. The smartphone spectrometer had a resolution of 5 nm and a wavelength measurement range between 400 nm and 1000 nm. The device utilized a transmission grating and Fresnel lens (G-Fresnel) fabricated using soft lithography [22]. The resulting component contained a Fresnel lens pattern on one side and a grating pattern on the other. The Fresnel pattern focused the impinging light and the grating pattern dispersed the received light into

different wavelengths. Finally, the dispersed light was focused onto a complementary metal oxide semiconductor (CMOS) image sensor [22].

The optical spectrometer was connected to an Android handset using a micro USB connection. In addition to data transfer, the micro USB connection provided power to the CMOS camera. Measurements were collected using an Android application developed by ATOPTIX, Inc. The device was calibrated to map pixels of data to specific wavelengths. This was accomplished using a calibration lamp. The peaks on the spectra from the calibration lamp occurred at known wavelengths, and the relationship between the pixel number and wavelength could be modeled using a linear trendline. The parameters for this linear fit were stored in the application, allowing results to be presented in terms of wavelengths, not pixel numbers. The application allowed the user to specify additional parameters (such as the integration time) and initiate measurements. After each capture, the app stored the data to internal memory and plotted the received spectrum.

3.3 | Configuration of Testing Site

Measurements were collected between July 7, 2016 and August 9, 2016 at the Pennsylvania State University Agricultural Research fields. The address of the site used is 1883 W. Pinegrove Rd, Pennsylvania Furnace, PA 16865. The field was organized into different rows of potato plants. Specific rows were designated as “spreader” rows – these rows would be inoculated with early blight on July 14, 2016. The points of inoculation were marked with orange flags. Upon inoculation, the early blight would spread from the spreader rows to the other rows of potato plants. Three varieties of potato plants were tested, given by the markers

"Flag 4", "Flag 17", and Flag 19." In addition to the three varieties that were tested, the spreader rows were also measured at flags marking the point of inoculation, or "Spreader Origin," and halfway between two inoculation points, or "Spreader Mid." Measurements taken on July 7 and July 9, 2016 correspond to healthy plants.

3.4 | Physical Data Collection

Except when prohibited by inclement weather and fertilization, measurements were taken daily. Each of the following plants was measured approximately one hundred times, daily: Flag 4, Flag 17, and Flag 19. Each Flag corresponds to one specific plant; there were not multiple plants with the same flag number. Twenty random leaves were selected from the plant, and five measurements were taken at different locations on each leaf. For the Spreader Mid and Spreader Origin plants, the process was repeated on ten leaves, for a total of fifty measurements each. For each measurement, the mobile optical sensor made physical contact with the leaf of the plant. This was achieved by "clipping" the sensor in place on the leaf. DRS light was transmitted by the mobile optical sensor to the leaf. The received image was converted from pixels to an array of intensity values, as described in the literature review. After each day of measurements, this data was transferred to a computer for processing.

3.5 | Extraction Script

The extraction script maps the dimensions of the image (in pixels) to wavelengths based on calibration data. Once the received intensities have been mapped to the corresponding wavelengths, the background noise (measured daily), is subtracted from each measurement. The

data is then scaled and resized. Finally, the new measurement matrices are saved to the directory. This program was heavily modified by Perry Edwards on October 24, 2016.

3.6 | Processing Script

The processing script starts by organizing measurements based on the corresponding plant (flag number). It utilizes a for-loop that loads each measurement into an array, then normalizes the data. The matrix now contains all the intensity values at each wavelength for that specific variety. To eliminate outliers, the matrix is duplicated and any infinite values are removed. Next, any measurement (row of data) that is more than one standard deviation away from the mean is removed. The plots of the original data matrix and the refined data matrix are also generated for comparison purposes. See Figure 3.2 on the following page for a flow chart of the processing script.

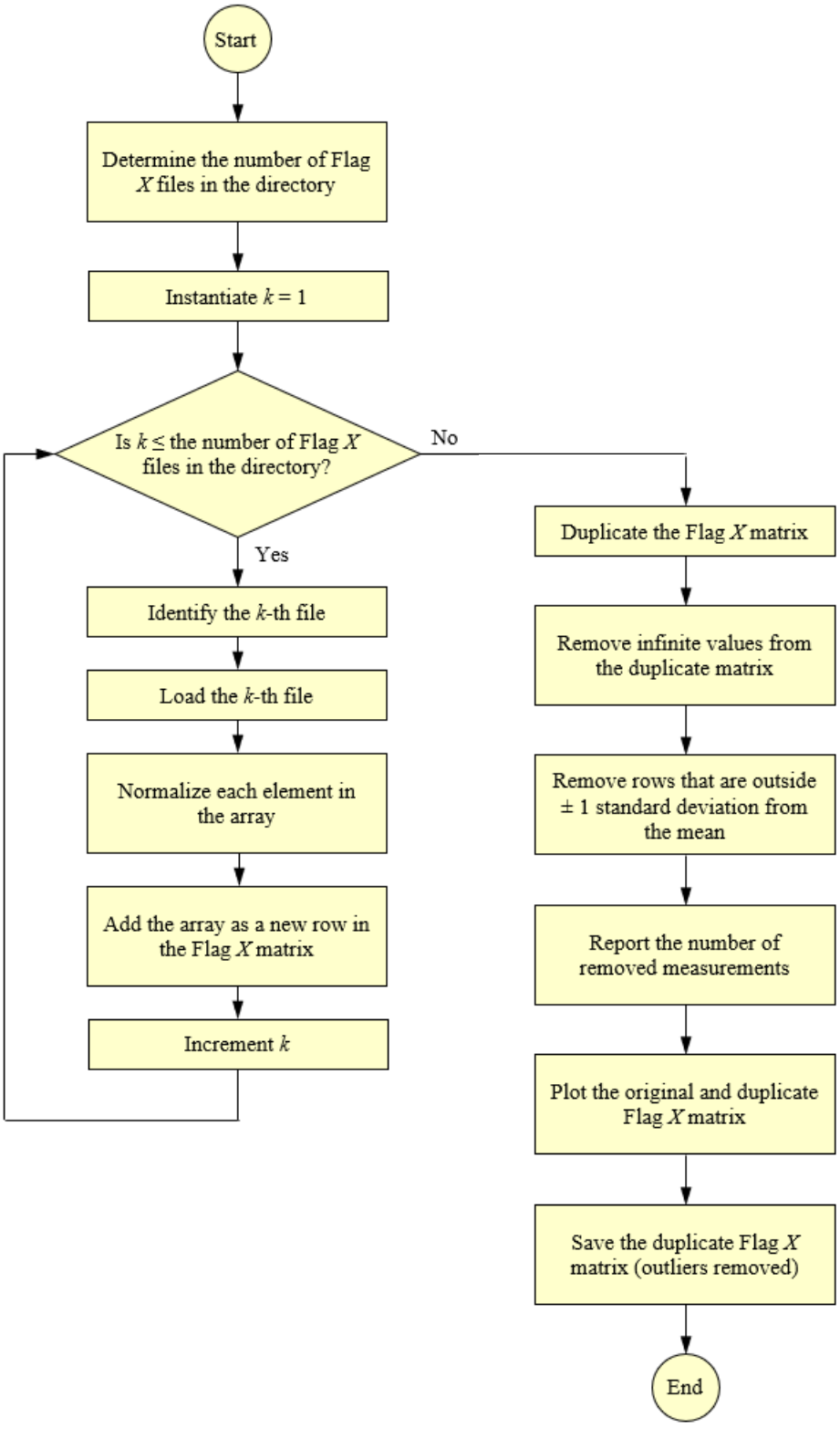

Figure 3.1: Flow Chart of Processing Script

To process all the data, the processing script was placed in every folder of measurements and executed. See Appendix A for the complete preprocessing MATLAB script.

3.7 | Combining the Measurements

In order to compare how the early blight affected the plants over time, the measurements were combined. For each plant, the processed data matrices for each trial were moved to a common folder. The individual matrices were opened in MATLAB. The matrices were combined in Excel, then saved as a combined spreadsheet. A note was made of the number of measurements taken before the spreader rows were inoculated. These measurements represent healthy data, and were differentiated later in the experiment. Table 3.1 summarizes the number of healthy preprocessed measurements collected for each type of plant.

Table 3.1: Number of measurements taken before inoculation

Plant Type	No. of Healthy Measurements
Flag 4	104
Flag 17	86
Flag 19	34
Spreader Mid	88
Spreader Origin	88

3.8 | PCA Script

MATLAB's standard PCA function was used to perform the principal component analysis on the preprocessed data. The command has been reproduced in Equation (3.1) for further explanation.

```
[coeff, score] = pca(flag)
```
(3.1)

Here, `flag` is defined as the processed and combined data matrix that was loaded into MATLAB. The data matrix `flag` is the input to the function `pca`, which outputs two matrices. The `coeff` matrix contains the coefficients that uniquely describe each principal component. The rows correspond to each observation, and the columns give the PCA coefficients for each observation [20]. For example, `coeff(1,2)` gives the second principal component coefficient (column 2) for the first measurement (row 1). The score matrix reports the principal component scores of each data point. Again, the rows correspond to the observation number and the columns give principal component scores for each observation. For example, `score(4,1)` gives the score of the first principal component (column 1) for the fourth observation (row 4).

3.9 | Plotting the Principal Components

The scores for the first two principal components of the measurements were used in the data visualization. A two-dimensional coordinate system was selected for reasonable execution time of the *k*-means algorithm. The first PCA score was plotted on the horizontal axis and the second score on the vertical axis. The observations that were known to be healthy were plotted in green. This was accomplished by executing the plotting command twice – once for the first *n* measurements that were known to be healthy, as summarized in Table 3.1, and again for the remaining measurements for the specific plant. The data points that corresponded to measurements on plants that were not infected with early blight were plotted in green. The remaining data points were plotted in black. The plots were visually inspected to determine how

many clusters of points appeared to form. The number of clusters gives the value of *k* for the *k*-means learning algorithm. The *k*-values for each type of plant are reported in the Results section.

3.10 | *k*-Means Algorithm

MATLAB's standard `kmeans` function was used to execute the *k*-means unsupervised learning algorithm on the preprocessed data. The command has been reproduced in Equation (3.2) for further explanation.

```
[idx, c] = kmeans(Y, k)
```
(3.2)

The preprocessed data matrix `Y` and the number of clusters `k` are input to the `kmeans` function. The preprocessed data in `Y` are the first two columns from the principal component scores matrix. `k` is the number of clusters that appear to be present. The function `kmeans` minimizes the distance between points in a cluster and moves the cluster centroids to optimized positions. The theory behind the *k*-means algorithm was described in detail in the literature review. The function outputs two matrices, `idx` and `c`. `idx` is a column vector that gives the cluster indices of each observation. For example, if the fourth observation belonged to the third cluster, the fourth row in the column vector would have a value of 3. `c` is a `k` by 2 matrix, since there are `k` clusters and there are two dimensions (score 1 and score 2) to the preprocessed data `Y`. `c` gives the coordinates of the centroid for each cluster [24]. To graphically represent the regions of the plane associated with each cluster, a mesh grid was created in MATLAB. Each node in the gird was associated with a cluster, and a point was colored at each node to create

colored regions. The spacing between nodes was selected to be small enough such that the "dots" blended together to form shaded regions [25].

3.11 | Programming Solution

The machine learning script combined principal component analysis and the *k*-means algorithm to reduce the dimensions of the data, maximize the variance among each dimension visualized, and cluster the processed data to help determine the health of a plant. Principal component analysis was applied to the preprocessed observations. The first two scores were plotted for the healthy measurements, then one, three, and five days after inoculation. The locations of the centroids were computed. Lastly, the *k*-means algorithm was applied on the entire data set. A flow chart of the programming solution is shown on the following page in Figure 3.3. The MATLAB code is available in Appendix B.

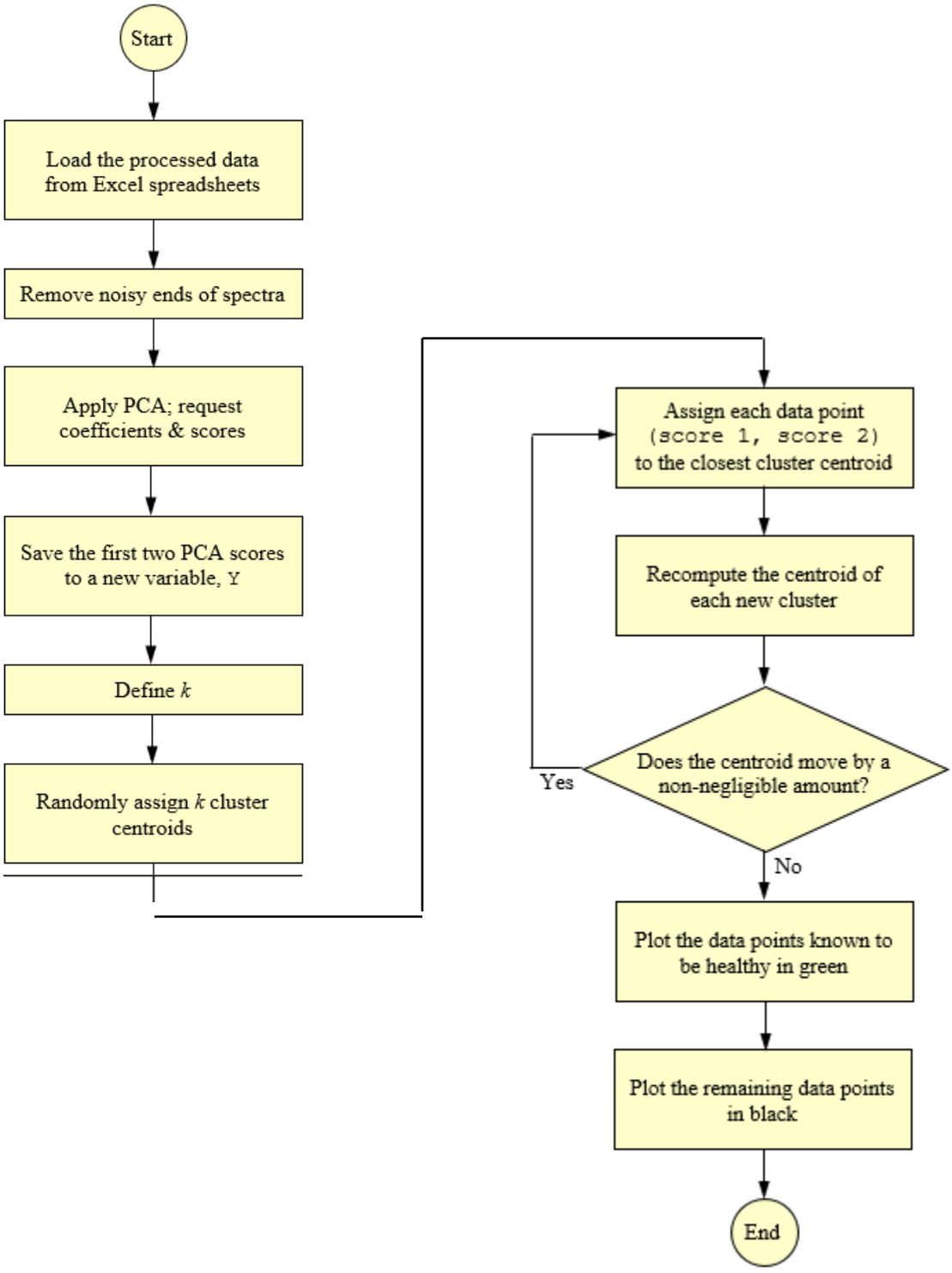

Figure 3.2: Flow Chart of Programming Solution

Chapter 4

Results

The outputs from PCA and the *k*-means algorithm are presented in the following section. First, the results from the preprocessing algorithm are presented. The PCA scores for each type of plant are plotted after one, three, and five days, and compared with the PCA scores from the dates before inoculation with early blight. The PCA scores are also plotted over the duration of the experiment. Finally, the results of the *k*-means clustering algorithm are presented.

4.1 | Preprocessing Results

The number of measurements to be taken on Flag 4, Flag 17, and Flag 19 was 100 per day. For Spreader Mid and Spreader origin, the number of measurements to be collected was 50 per day. After the data was preprocessed, outliers were removed as described in the Methods section. Table 4.1 on the following page gives the number of preprocessed measurements remaining after the outliers had been removed. Measurements taken on July 7, 2016 and July 9, 2016 were collected before the plants had been inoculated with early blight. Therefore, these measurements give PCA scores that correspond to healthy plants. In the plots that follow, green points represent PCA scores that correspond to healthy plants; black points represent PCA scores from observations taken after the spreader rows were inoculated. For the Spreader Mid and Spreader Origin plants, measurements ended on July 29, 2016. After this date, the leaves had wilted and prevented further data collection.

Table 4.1: Number of Preprocessed Measurements for Each Plant Type, per Day

Date	Number of Preprocessed Measurements				
	Flag 4	Flag 17	Flag 19	Spr. Mid	Spr. Origin
7/7/2016	37	43	0	29	29
7/9/2016	67	43	34	59	59
7/15/2016	54	82	66	33	37
7/16/2016	97	99	80	82	37
7/17/2016	93	83	84	42	54
7/18/2016	64	22	74	47	41
7/19/2016	74	85	62	44	34
7/20/2016	78	61	71	37	36
7/23/2016	81	87	95	41	44
7/25/2016	121	109	109	48	58
7/26/2016	88	81	83	46	44
7/27/2016	68	73	31	45	42
7/28/2016	97	82	86	39	55
7/29/2016	93	91	84	43	1
8/1/2016	94	105	100	0	0
8/4/2016	94	86	87	0	0
8/5/2016	95	92	86	0	0
8/9/2016	87	73	87	0	0

4.2 | PCA Scores One Day After Inoculation

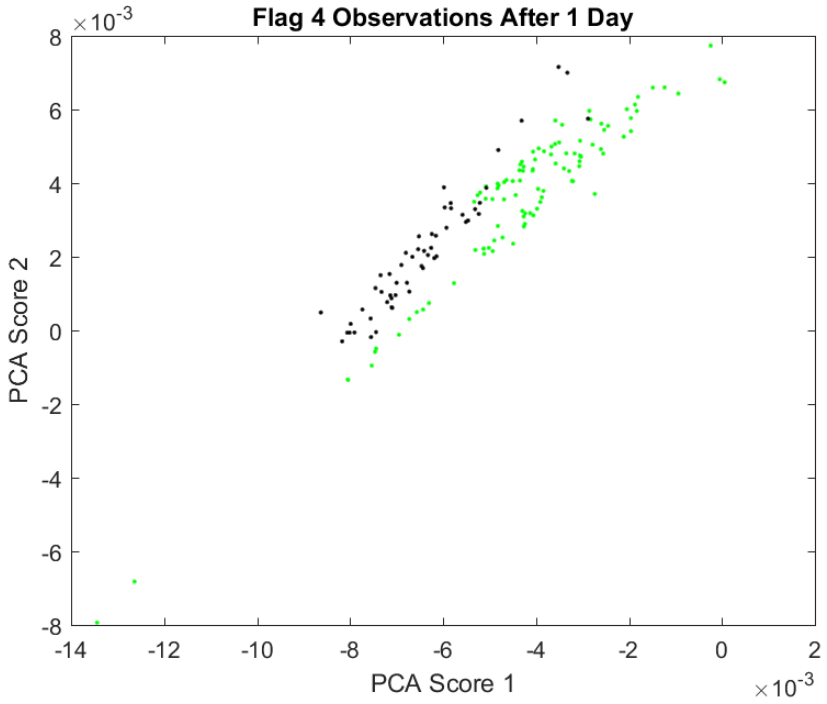

Figure 4.1: Flag 4 PCA Scores One Day After Inoculation

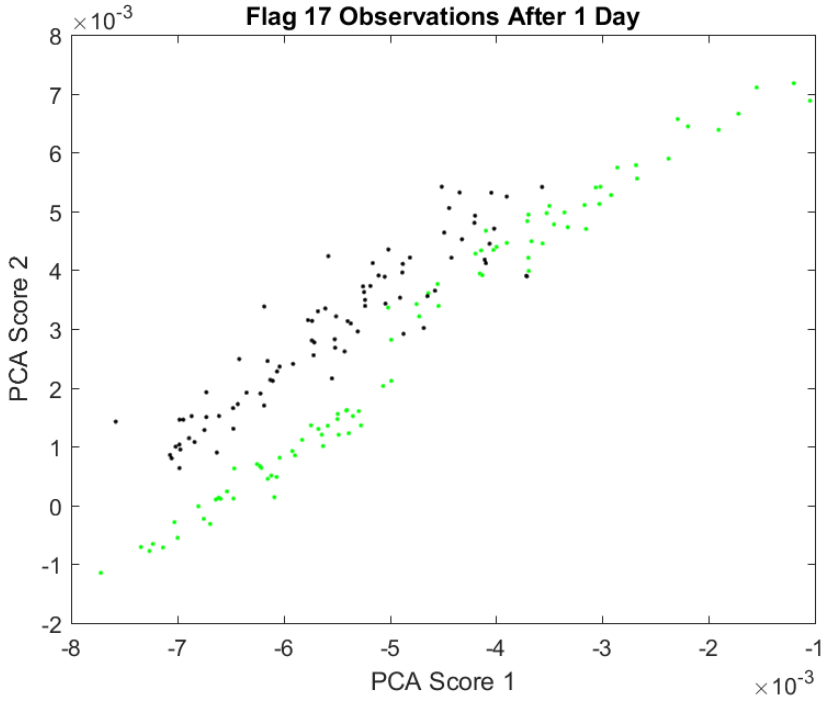

Figure 4.2: Flag 17 PCA Scores One Day After Inoculation

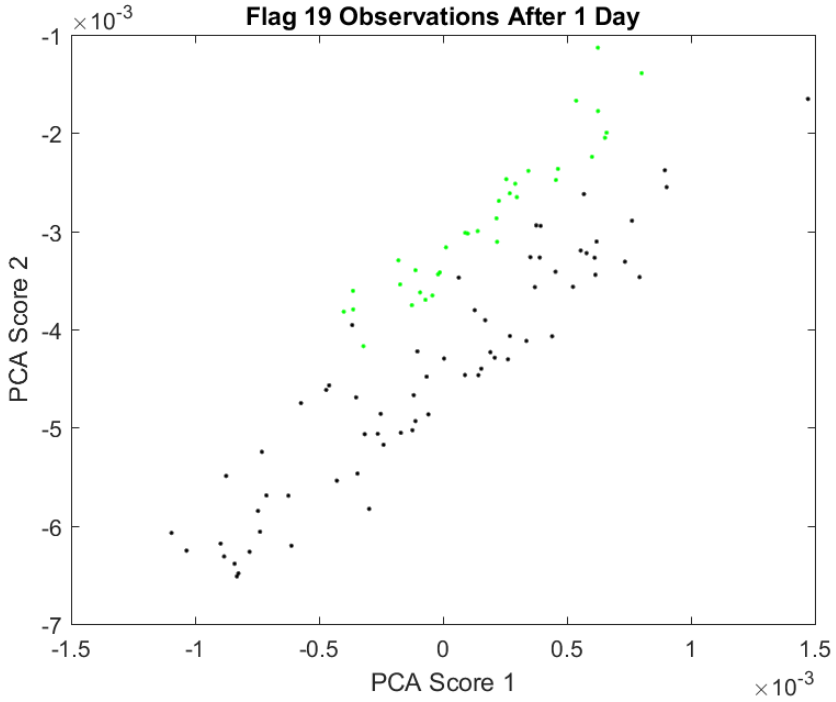

Figure 4.1: Flag 19 PCA Scores One Day After Inoculation

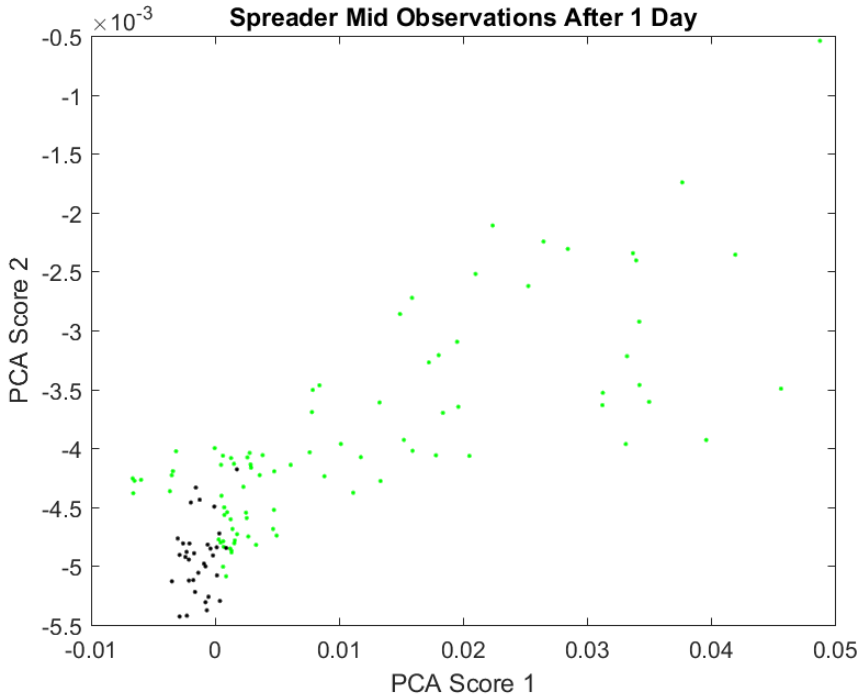

Figure 4.2: Spreader Mid PCA Scores One Day After Inoculation

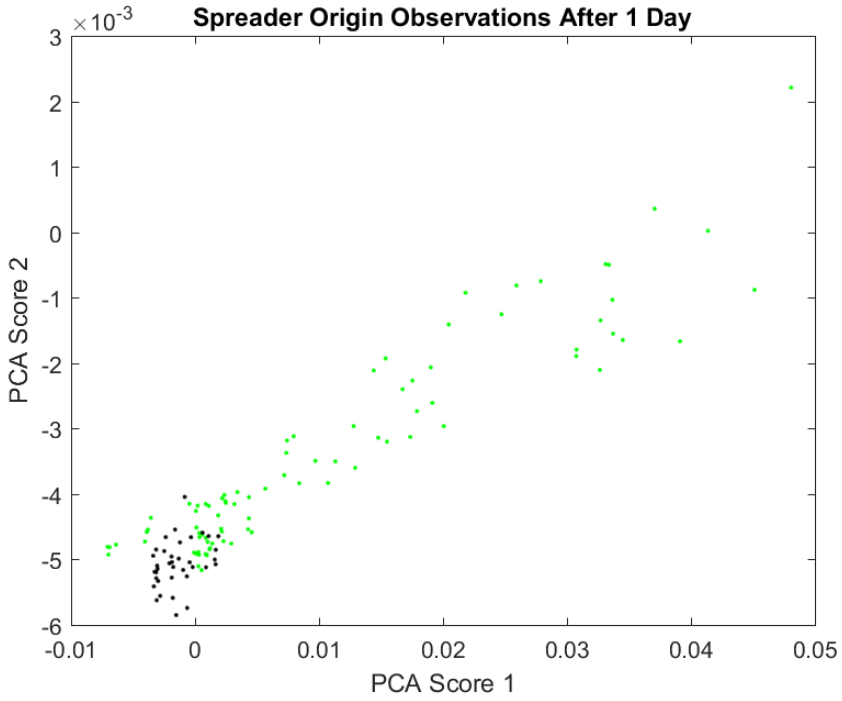

Figure 4.3: Spreader Origin PCA Scores One Day After Inoculation

4.3 | PCA Scores Three Days After Inoculation

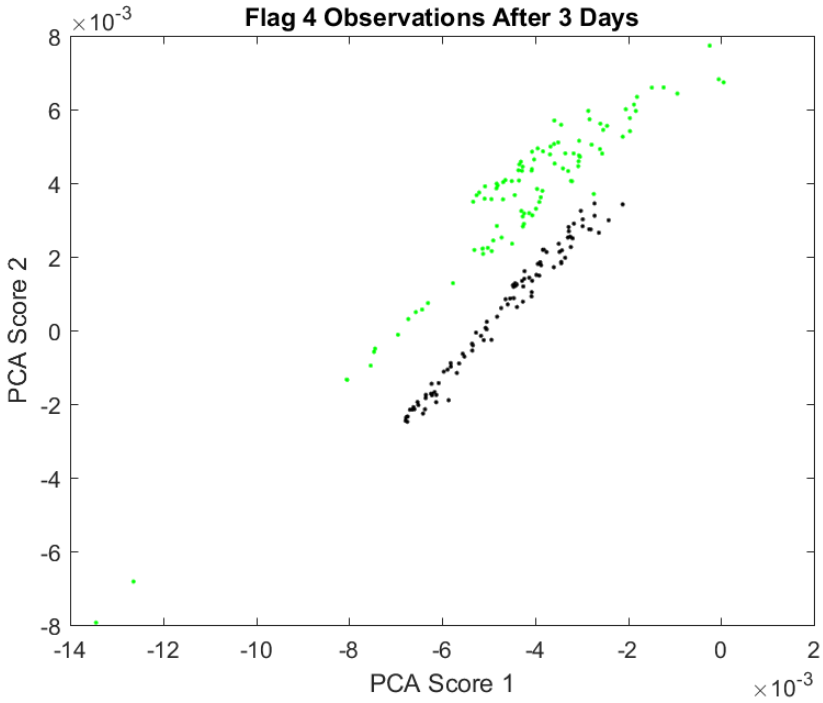

Figure 4.4: Flag 4 PCA Scores Three Days After Inoculation

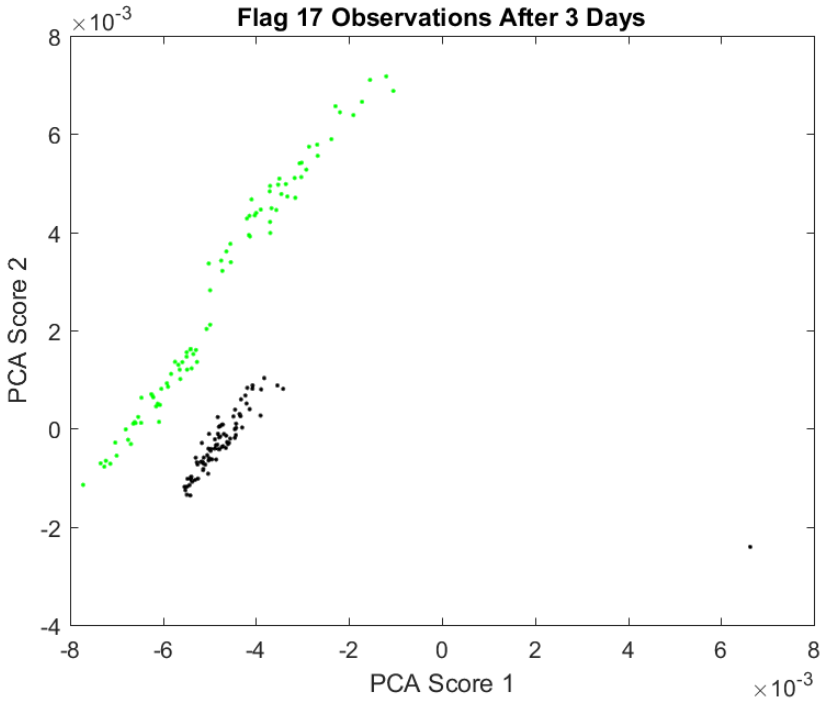

Figure 4.5: Flag 17 PCA Scores Three Days After Inoculation

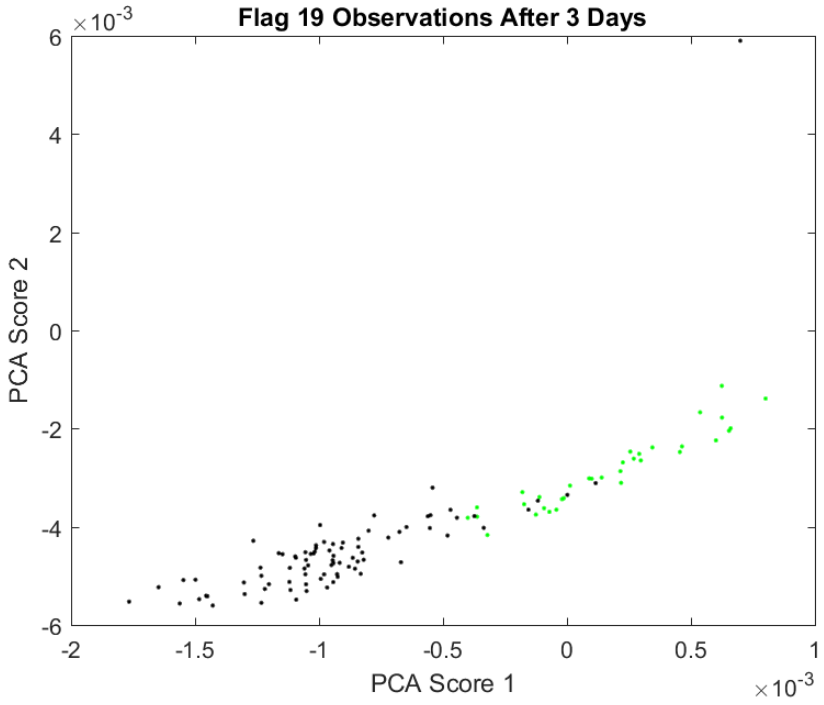

Figure 4.6: Flag 19 PCA Scores Three Days After Inoculation

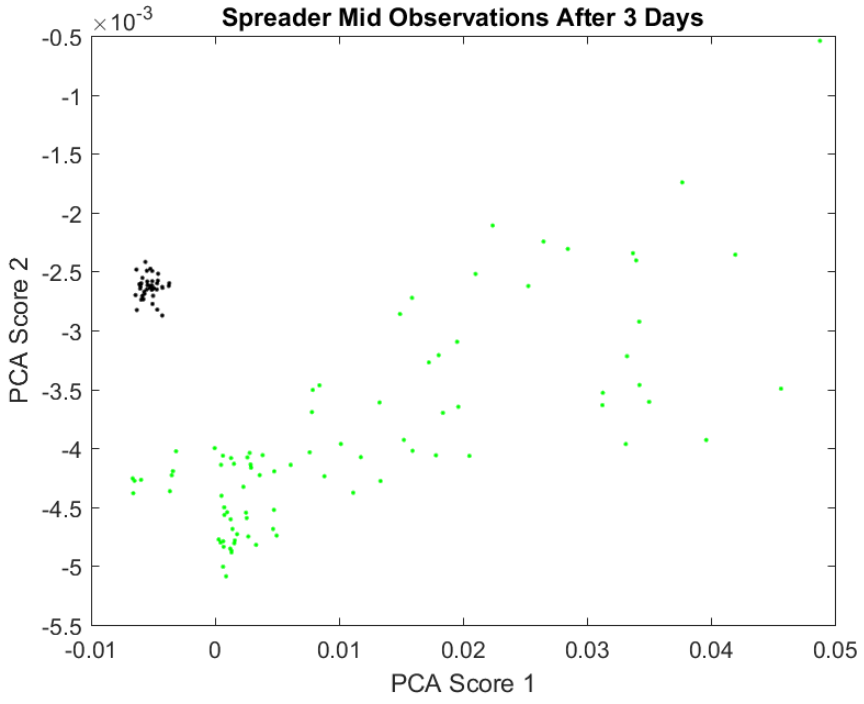

Figure 4.7: Spreader Mid PCA Scores Three Days After Inoculation

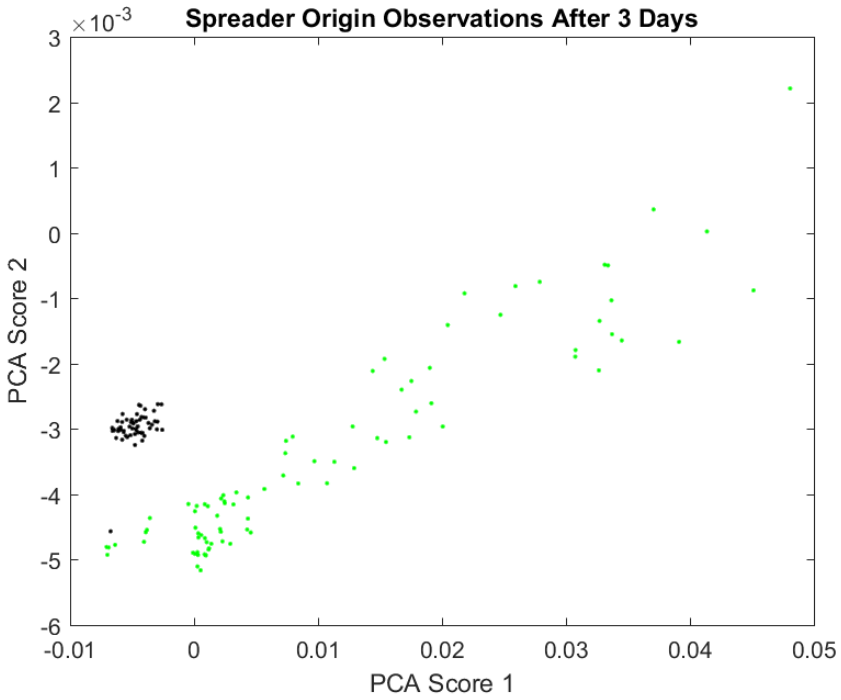

Figure 4.8: Spreader Origin PCA Scores Three Days After Inoculation

4.4 | PCA Scores Five Days After Inoculation

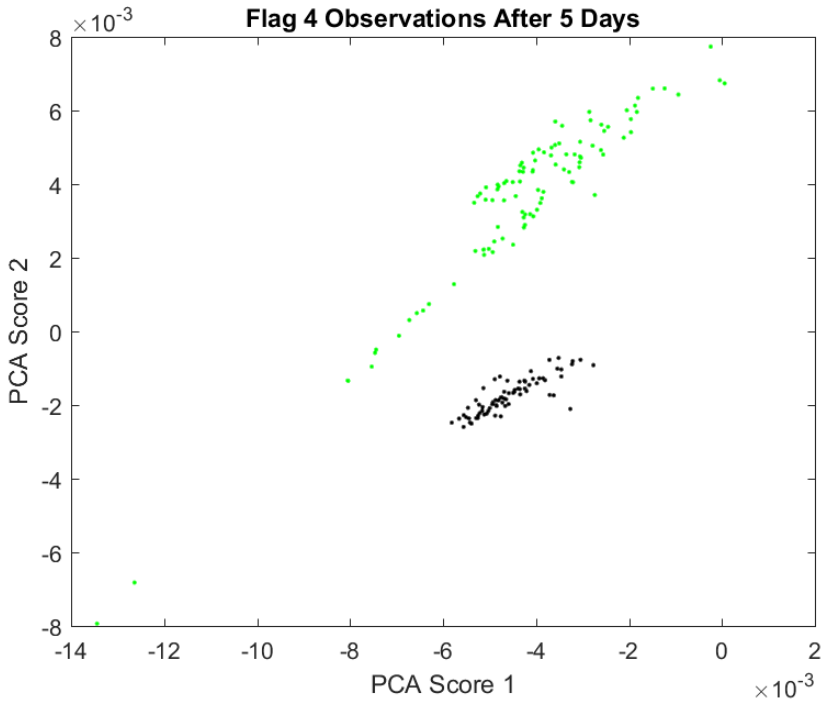

Figure 4.9: Flag 4 PCA Scores Five Days After Inoculation

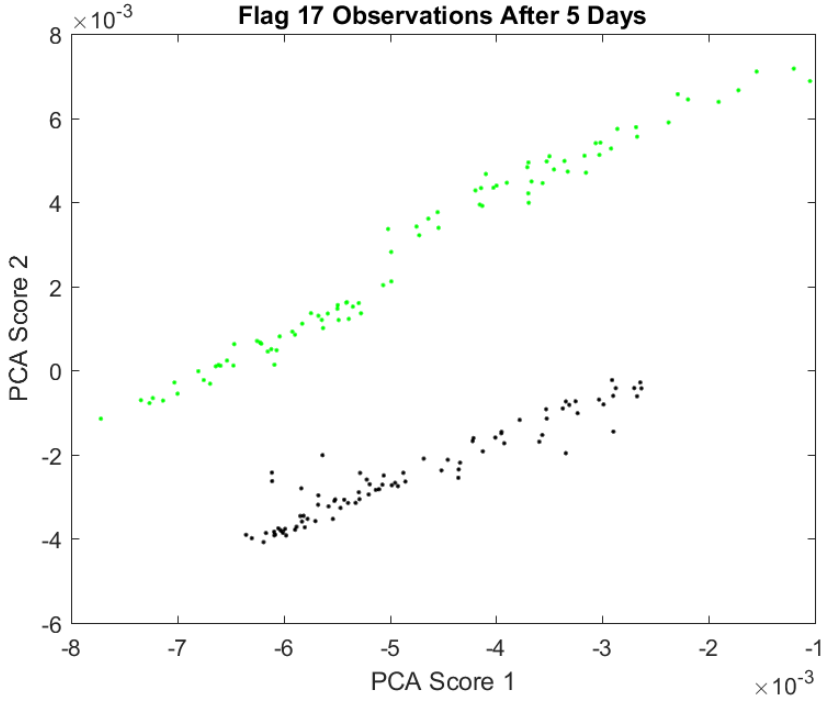

Figure 4.10: Flag 17 PCA Scores Five Days After Inoculation

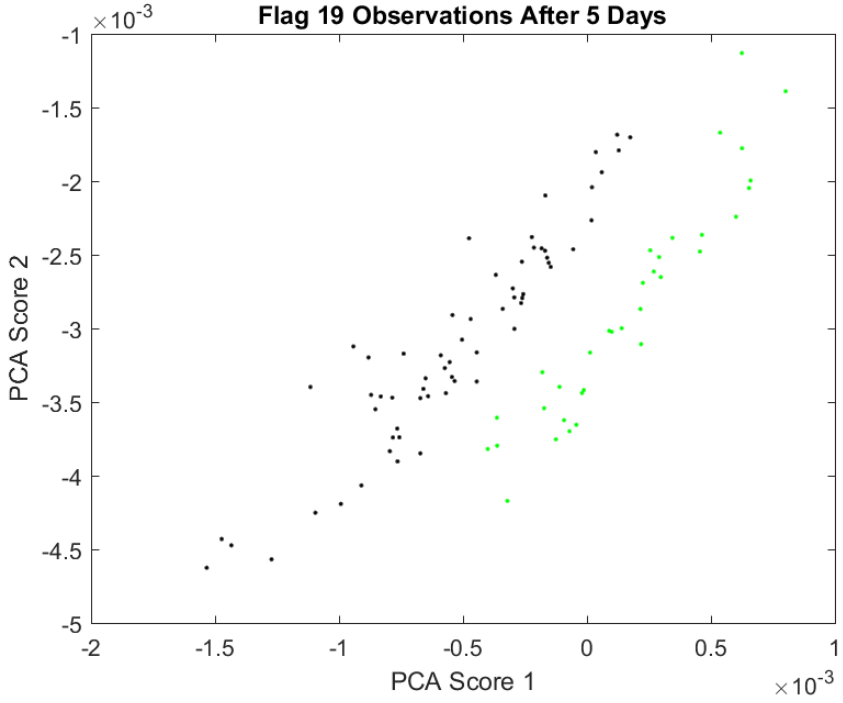

Figure 4.11: Flag 19 PCA Scores Five Days After Inoculation

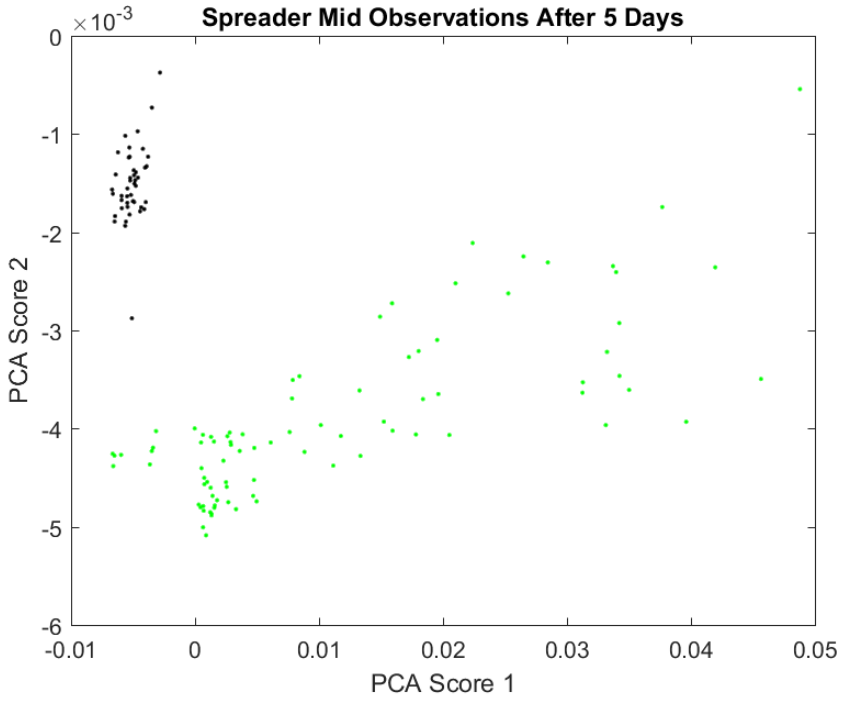

Figure 4.12: Spreader Mid PCA Scores Five Days After Inoculation

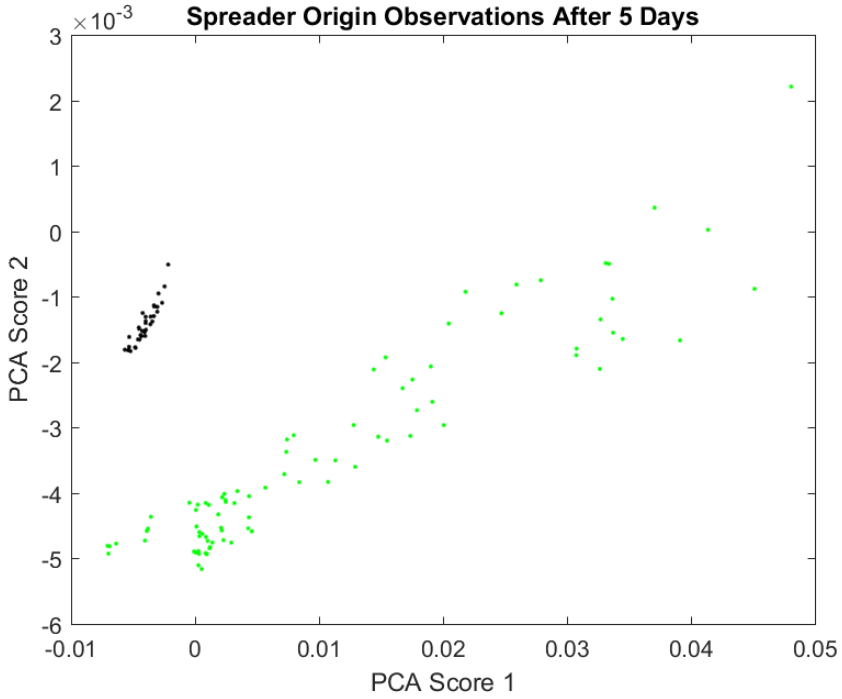

Figure 4.13: Spreader Origin PCA Scores Five Days After Inoculation

4.5 | PCA Scores for the Duration of the Experiment

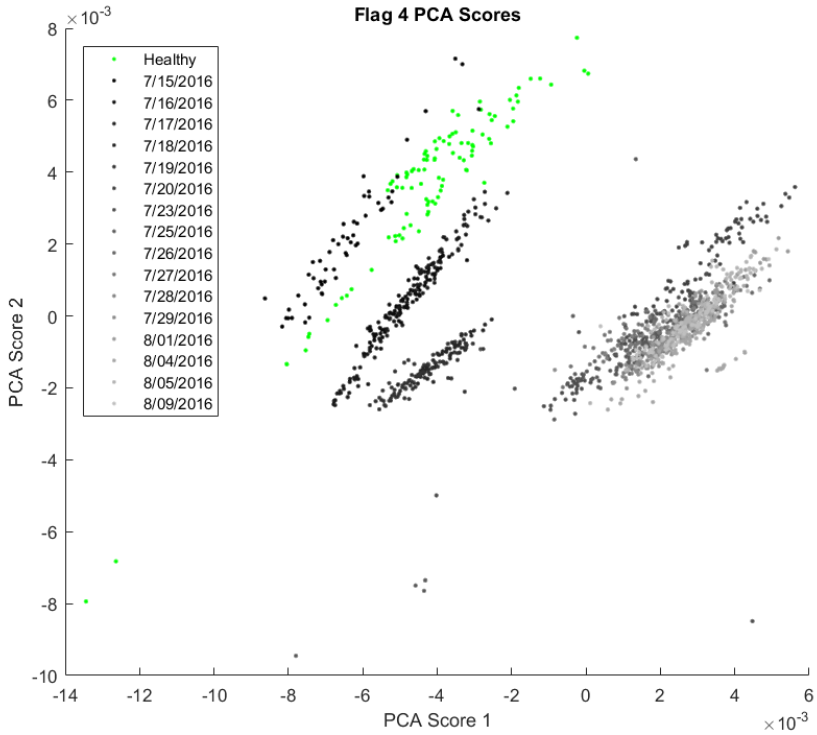

Figure 4.14: Flag 4 PCA Scores for Entire Experiment

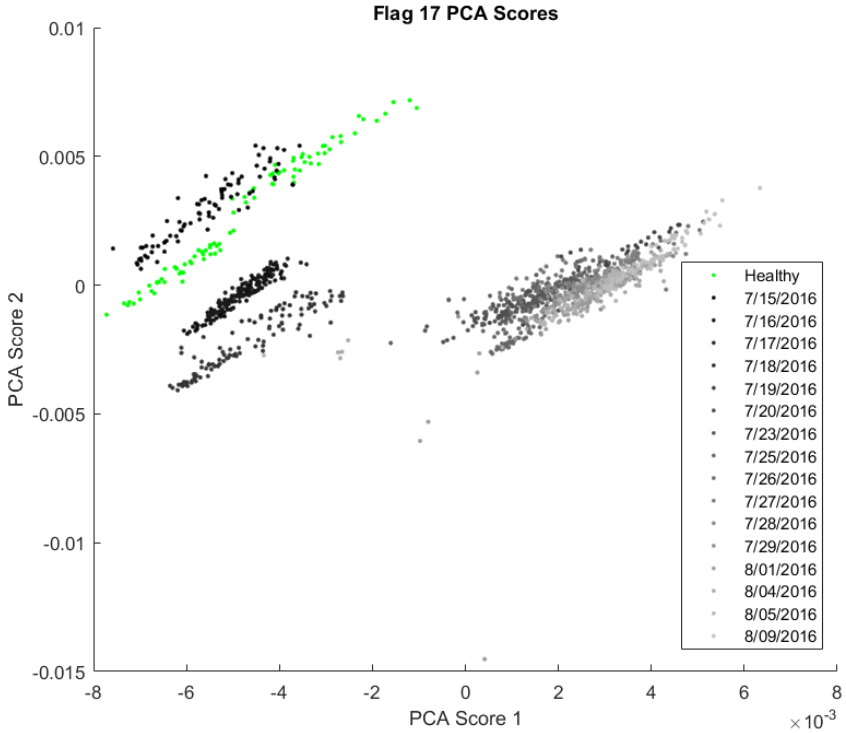

Figure 4.15: Flag 17 PCA Scores for Entire Experiment

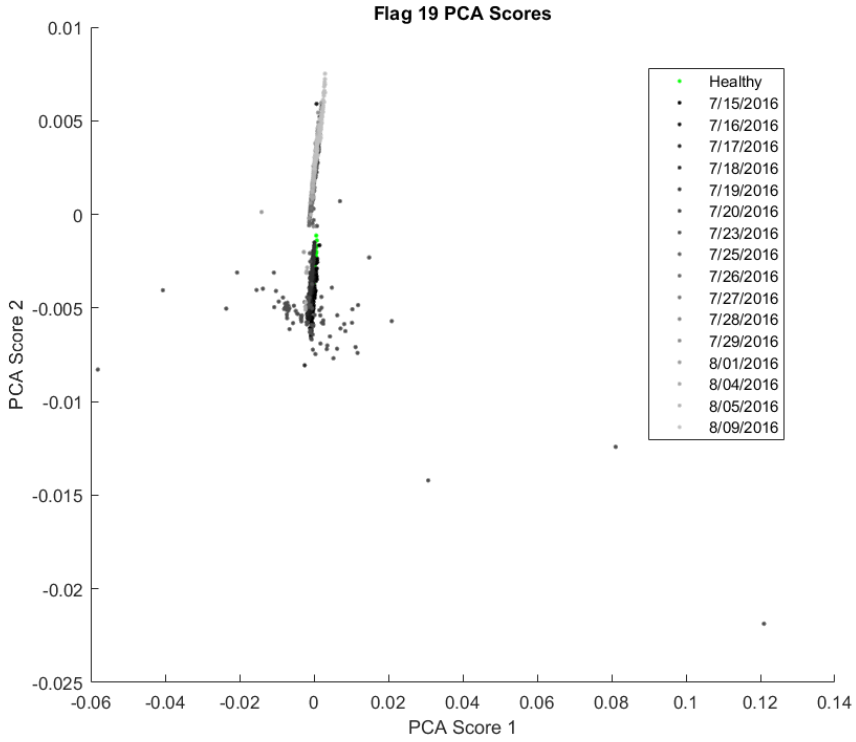

Figure 4.16: Flag 19 PCA Scores for Entire Experiment

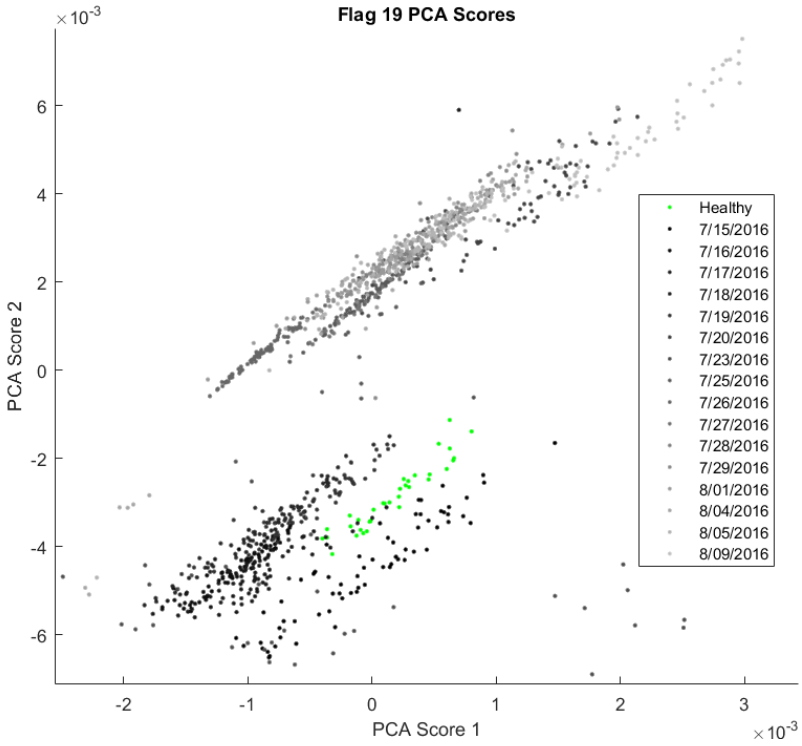

Figure 4.17: Flag 19 PCA Scores for Entire Experiment, Zoomed In

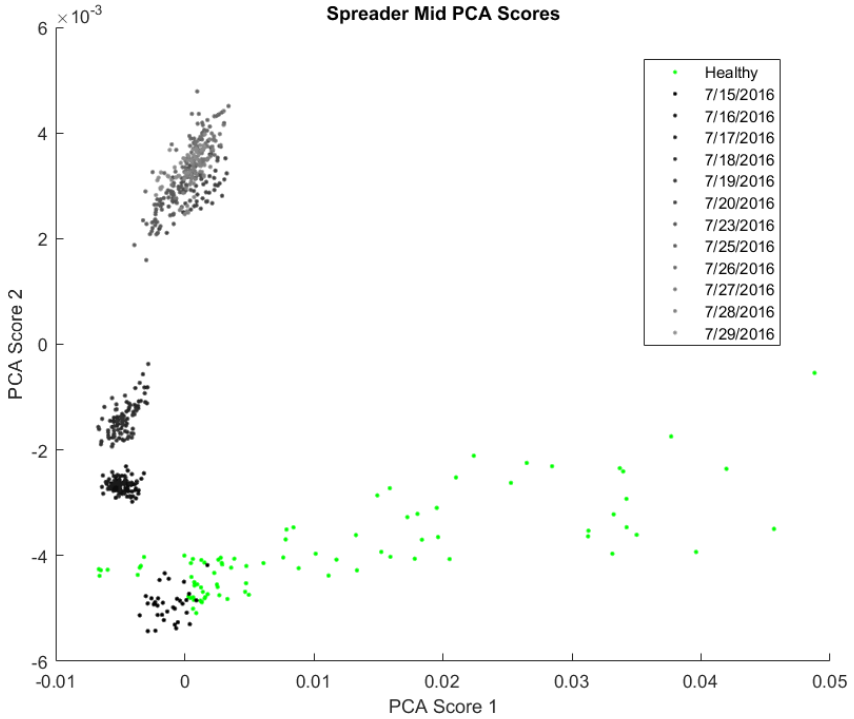

Figure 4.18: Spreader Mid PCA Scores for Entire Experiment

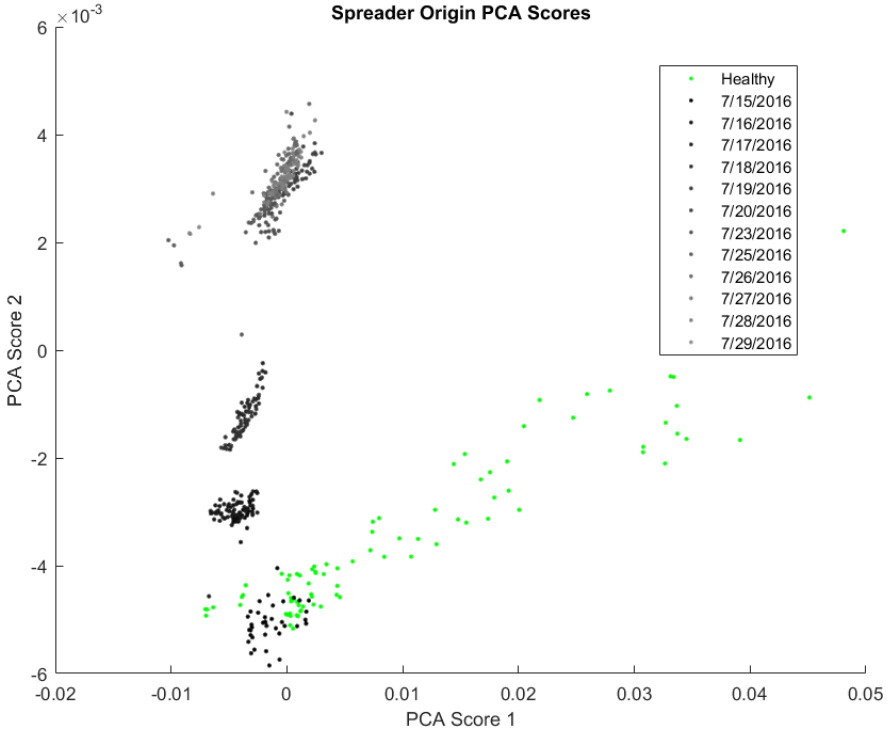

Figure 4.19: Spreader Origin PCA Scores for Entire Experiment

4.6 | *k*-means Confusion Matrices One Day After Inoculation

Table 4.2: Flag 4 Confusion Matrix One Day After Inoculation

FLAG 4 – Day 1		
	Predicted NO	Predicted YES
Actual NO	88	16
Actual YES	17	53

Table 4.3: Flag 17 Confusion Matrix One Day After Inoculation

FLAG 17 – Day 1		
	Predicted NO	Predicted YES
Actual NO	43	43
Actual YES	40	42

Table 4.4: Flag 19 Confusion Matrix One Day After Inoculation

FLAG 19 – Day 1		
	Predicted NO	Predicted YES
Actual NO	33	1
Actual YES	30	36

Table 4.5: Spreader Mid Confusion Matrix One Day After Inoculation

SP. MID – Day 1		
	Predicted NO	Predicted YES
Actual NO	30	58
Actual YES	0	33

Table 4.6: Spreader Origin Confusion Matrix One Day After Inoculation

SP. ORG. – Day 1		
	Predicted NO	Predicted YES
Actual NO	30	58
Actual YES	0	37

4.7 | *k*-means Confusion Matrices Three Days After Inoculation

Table 4.7: Flag 4 Confusion Matrix Three Days After Inoculation

FLAG 4 – Day 3		
	Predicted NO	Predicted YES
Actual NO	5	99
Actual YES	0	92

Table 4.8: Flag 17 Confusion Matrix Three Days After Inoculation

FLAG 17 – Day 3		
	Predicted NO	Predicted YES
Actual NO	43	43
Actual YES	0	83

Table 4.9: Flag 19 Confusion Matrix Three Days After Inoculation

FLAG 19 – Day 3		
	Predicted NO	Predicted YES
Actual NO	34	0
Actual YES	0	84

Table 4.10: Spreader Mid Confusion Matrix Three Days After Inoculation

SP. MID – Day 3		
	Predicted NO	Predicted YES
Actual NO	32	56
Actual YES	0	42

Table 4.11: Spreader Origin Confusion Matrix Three Days After Inoculation

SP. ORG. – Day 3		
	Predicted NO	Predicted YES
Actual NO	30	58
Actual YES	0	54

4.8 | *k*-means Confusion Matrices Five Days After Inoculation

Table 4.12: Flag 4 Confusion Matrix Five Days After Inoculation

FLAG 4 – Day 5		
	Predicted NO	Predicted YES
Actual NO	88	16
Actual YES	0	73

Table 4.13: Flag 17 Confusion Matrix Five Days After Inoculation

FLAG 17 – Day 5		
	Predicted NO	Predicted YES
Actual NO	45	41
Actual YES	0	85

Table 4.14: Flag 19 Confusion Matrix Five Days After Inoculation

FLAG 19 – Day 5		
	Predicted NO	Predicted YES
Actual NO	34	0
Actual YES	0	62

Table 4.15: Spreader Mid Confusion Matrix Five Days After Inoculation

SP. MID – Day 5		
	Predicted NO	Predicted YES
Actual NO	32	56
Actual YES	0	44

Table 4.16: Spreader Origin Confusion Matrix Five Days After Inoculation

SP. ORG. – Day 5		
	Predicted NO	Predicted YES
Actual NO	30	58
Actual YES	0	34

Chapter 5

Analysis

Upon plotting the PCA scores for the duration of the experiment, it is evident that clusters of data form. The focus of this thesis, however, is on early disease detection. Therefore, most of the discussion will focus on the plots created 1, 3, and 5 days after inoculation. The cluster centroids of the data after inoculation noticeably shift during this period of time. This was quantified by computing the percent difference of the centroids. The *k*-Means algorithm could not successfully classify the measurements as healthy or diseased for every plant tested.

5.1 | PCA Centroid Analysis

The quantitative analysis began by computing the centroid for each of the following clusters: all known healthy data, after one day, after three days, and after five days. The components of the centroid locations are summarized in Tables 5.1 and 5.2 and, together, represent points (Score 1, Score 2) in the plane.

Table 5.1: Score 1 Component of Centroid Locations

Score 1 Component of Centroid Locations				
	Healthy	After 1 Day	After 3 Days	After 5 Days
Flag 4	-0.004416	-0.0064291	-0.0046822	-0.00457
Flag 17	-0.004765	-0.0055333	-0.004657	-0.0048
Flag 19	0.000165	-4.395E-05	-0.000934	-0.00053
Spreader Mid	0.0110638	-0.0012196	-0.0052666	-0.0051
Spreader Origin	0.0106151	-0.0013702	-0.0047474	-0.00406

Table 5.2: Score 2 Component of Centroid Locations

Score 2 Component of Centroid Locations				
	Healthy	After 1 Day	After 3 Days	After 5 Days
Flag 4	0.003347	0.002118	0.0005	-0.00175
Flag 17	0.002759	0.002987	-0.00032	-0.00246
Flag 19	-0.00288	-0.00448	-0.00451	-0.00309
Spreader Mid	-0.00392	-0.00493	-0.00263	-0.00151
Spreader Origin	-0.0034	-0.00506	-0.00298	-0.00143

The percent difference was computed using the following formula:

$$\frac{(\text{Score}_{\text{day}} - \text{Score}_{\text{healthy}})}{\left(\frac{\text{Score}_{\text{day}} + \text{Score}_{\text{healthy}}}{2}\right)} \quad (5.1)$$

where $\text{Score}_{\text{day}}$ is the average of one of the component scores for a given number of days after the plant was inoculated and $\text{Score}_{\text{healthy}}$ is the average component of the respective PCA score for the cluster of data known to be healthy. The percent differences of the change in the Score 1 component and the change in the Score 2 component of the centroids is summarized in Tables 5.3 and 5.4.

Table 5.3: Percent Difference in Score 1 Component of Cluster Centroids

Percent difference in Score 1 Component of Cluster Centroids			
	After 1 Day	After 3 Days	After 5 Days
Flag 4	37.13%	5.86%	3.49%
Flag 17	14.92%	-2.30%	0.79%
Flag 19	-345.28%	285.79%	382.20%
Spreader Mid	-249.56%	-563.38%	-542.49%
Spreader Origin	-259.28%	-523.63%	-447.73%

Table 5.4: Percent Difference in Score 2 Component of Cluster Centroids

Percent difference in Score 2 Location			
	After 1 Day	After 3 Days	After 5 Days
Flag 4	-45.01%	-147.99%	-638.44%
Flag 17	7.93%	-252.35%	-3517.50%
Flag 19	43.53%	44.19%	7.19%
Spreader Mid	22.80%	-39.25%	-88.72%
Spreader Origin	39.36%	-13.18%	-81.43%

The total change in the centroid location was computed by finding the magnitude of the percent difference from the Score 1 and Score 2 components. This was computed using Equation (5.2).

$$(\%\ \text{diff.})_{\text{total}} = \sqrt{[(\%\ \text{diff})_{\text{score 1}}]^2 + [(\%\ \text{diff})_{\text{score 2}}]^2} \tag{5.2}$$

The magnitude of the change in centroid locations is summarized in Table 5.5:

Table 5.5: Percent Difference in Centroid Location

Percent difference in Centroid Location			
	After 1 Day	After 3 Days	After 5 Days
Flag 4	58.35%	148.11%	638.45%
Flag 17	16.89%	252.36%	3517.50%
Flag 19	348.01%	289.19%	382.26%
Spreader Mid	250.60%	564.75%	549.70%
Spreader Origin	262.25%	523.80%	455.07%

5.2 | Analysis After One Day

The clusters of data collected for Flags 4, 17, and 19 retain the same general shape as their respective healthy clusters. The location of the Flag 4 cluster appears to have shifted along the PCA Score 1 and PCA Score 2 axes. This is supported by the percent difference

calculations, which indicate that the centroid of the Flag 4 cluster has a percent difference of 58.35% when compared to the cluster of healthy data. The Flag 17 cluster appears to have shifted primarily in along the PCA Score 1 axis. This is also supported by the percent difference calculation, which indicates a change of 14.92% compared to the healthy measurements. The location of the Flag 19 centroid also appears to have shifted in the PCA score plane; the percent difference of the centroid location is 348.01%.

The clusters for both the Spreader Mid and Spreader Origin data appear to have changed their shapes. The domain and range of the clusters has been significantly reduced. The cluster centroids have experienced percent differences of 250.60% and 262.25% for Spreader Mid and Spreader Origin, respectively. The spreader plants were the ones directly inoculated with the early blight. This means that after one day of exposure to the fungus, the centroids of the clusters for both spreaders differ by more than 200% from their original healthy locations.

5.3 | Analysis After Three Days

The clusters of data collected for Flags 4 and 17 retain the same general shape as their healthy clusters; however, they appear to have shifted further away from the healthy clusters. The percent difference between the post- and pre-inoculation centroids is 148.11% and 252.36% for Flags 4 and 17 respectively. The Flag 19 cluster also appears to have shifted further away from the healthy centroid location. The separation is not as apparent as for Flag 19; however, the difference becomes evident when looking at the change in the centroid location. The percent difference in the Flag 19 centroid location is 289.19%.

These are also noteworthy results because the plants corresponding to Flags 4, 17, and 19 were not directly inoculated with the disease. Three days after nearby rows were infected with the fungus, the centroid clusters from these plants were differentiable by percent differences all exceeding 100%. The PCA cluster centroids of the Spreader Mid and Spreader Origin plants also continued to deviate from the location of the healthy centroids; both with percent differences exceeding 500%.

5.4 | Analysis After Five Days

The data in the Flag 4 cluster has become more compact, and the location of the centroid appears to have moved further away. The Flag 17 and Flag 19 clusters appear to have moved further away as well. The Spreader Mid and Spreader Origin clusters have shifted further away from the healthy clusters. Their shapes also appear to be extending more vertically than before. After five days, the percent difference for all of the centroids exceeds 300%. Figures 4.16 for 4.21 show that the clusters continue to shift away from the healthy cluster location as time passes.

5.5 | Analysis of *k*-Means Clustering

The *k*-Means clustering algorithm could not successfully classify the PCA scores as healthy or diseased for all of the plants in the experiment. In order to evaluate the accuracy of the cluster assignments, confusion matrices were constructed for each plant after 1, 3, and 5 days. The meaning of each cell in the confusion matrix is shown in Figure 5.1 on the following page.

	Predicted NO	Predicted YES
Actual NO	True Negative	False Positive
Actual YES	False Negative	True Positive

Figure 5.1: Confusion Matrix Explanation

For Flags 4 and 17, the algorithm correctly classified all of the unhealthy measurements three days after inoculation; however, it incorrectly classified some of the healthy measurements as unhealthy. The k-Means algorithm successfully clustered the Flag 19 measurements within three days after inoculation. For both Spreader Mid and Spreader Origin, the k-means algorithm correctly classified all of the diseased measurements one day after inoculation. The algorithm returned an unacceptable number of false positives for all trials. Although the k-Means algorithm implemented in these experiments correctly classified diseased plants, it reported a non-negligible number of false positives that would result in the removal of healthy plants.

Chapter 6

Conclusion

The objective of this thesis was to create software that can be used to determine if a plant is healthy or unhealthy based on processed optical sensing data. MATLAB scripts were created that collectively accomplish this goal. First, outliers are removed from the data set and principal component analysis is used to reduce the dimensions of the data. The scores of the two principal components that best model the data are plotted and the clusters that form are analyzed.

The percent difference in centroid location for both Spreader Mid and Spreader Origin exceed 250% one day after inoculation. The percent difference in centroid location for all plants tested exceeds 100% within three days of exposure to early blight fungus. This shows that principal component analysis could be used to help determine the health state of a plant, given its optical spectra. The *k*-Means clustering algorithm used in this thesis was ineffective at determining plant health. Future work could investigate the effectiveness of the *k*-Means algorithm using more than the first two PCA scores. Alternatively, a supervised learning algorithm could be tested.

Appendix A

Preprocessing Script

11/3/17 7:11 PM E:\...\01 General Extraction Script v1.m 1 of 8

```
% Prepreocessing Script
% Philip Ryan Senior Thesis - FA 17
%
% Input: Plant data files
%
% Processing:
% 1. Load each observation into a matrix
% 2. Remove any infinite values from the data
% 3. Remove any observations that are more than +/- 1 standard
%    deviation from the mean
% 4. Save the processed data matrix
%
% Output: Data matrix for a single day with outliers removed

%% PROCESS FLAG 4 MEASUREMENTS
% ------------------------------------------------------------------------

clear;
clc;
close all;

% Organize measurements based on associated plant number
flag4_files = dir('*flag4*.mat');

% Determine how many flag4 files there are
num_flag4_files = length(flag4_files);
datafinal_flag4=[];

% Create a loop that loads each flag4 file into an array
for k = 1:num_flag4_files

    % Identify the k-th file in the structure 'flag4_files'
    filename=flag4_files(k).name;

    % Load the k-th file
    [pathstr,name,ext]=fileparts(filename);
    load(filename)

    % Normalize the data matrix by dividing each element by the sum of the
    % data points
    data=data/sum(data);

    % Save the normalized data points to the 'datafinal_flag4' matrix
    datafinal_flag4=[datafinal_flag4;data];

end

% Create a duplicate of the datafinal_flag4 matrix, datafinal_flag4_DUP
datafinal_flag4_DUP=datafinal_flag4;
```

```
% Remove all infinite values from the datafinal_flag4 duplicate matrix
indexnan_flag4=isnan(sum(datafinal_flag4_DUP,2));
datafinal_flag4_DUP(indexnan_flag4,:)=[];

% Remove all data sets that fall outside of the constraints outlined below
out1=mean(datafinal_flag4_DUP(:,200));
index_flag4 = datafinal_flag4_DUP(:,200) < (out1-1*std(datafinal_flag4_DUP(:,200))) | ↙
datafinal_flag4_DUP(:,200) > (out1+1*std(datafinal_flag4_DUP(:,200)));
datafinal_flag4_DUP(index_flag4,:) = [];
% datafinal_cocoa_h=datafinal1;


% Report statistics about dropped data ----------------------------------
nanCount_flag4 = sum(indexnan_flag4);

% Display the original size of the data matrix
originalSize_flag4 = size(datafinal_flag4)

constraintDropped_flag4 = sum(index_flag4);

% Display the total number of data sets dropped
totalDropped_flag4 = constraintDropped_flag4 + nanCount_flag4

% Plot the refined data set, with outliers removed
figure(1)
plot(wavelength,datafinal_flag4_DUP)

% Save the refined data set, with outliers removed
save('flag4_FINAL.mat', 'datafinal_flag4_DUP');

% Plot the original data set for reference
figure(2)
plot(wavelength,datafinal_flag4)

% Insert breakpoint



% ------------------------------------------------------------------------
%% PROCESS FLAG 17 MEASUREMENTS
% ------------------------------------------------------------------------

clear;
clc;
close all;

% Organize measurements based on associated plant number
flag17_files = dir('*flag17*.mat');
```

```
% Determine how many flag17 files there are
num_flag17_files = length(flag17_files);
datafinal_flag17=[];

% Create a loop that loads each flag17 file into an array
for k = 1:num_flag17_files

    % Identify the k-th file in the structure 'flag17_files'
    filename=flag17_files(k).name;

    % Load the k-th file
    [pathstr,name,ext]=fileparts(filename);
    load(filename)

    % Normalize the data matrix by dividing each element by the sum of the
    % data points
    data=data/sum(data);

    % Save the normalized data points to the 'datafinal_flag17' matrix
    datafinal_flag17=[datafinal_flag17;data];

end

% Create a duplicate of the datafinal_flag17 matrix, datafinal_flag17_DUP
datafinal_flag17_DUP=datafinal_flag17;

% Remove all infinite values from the datafinal duplicate matrix
indexnan_flag17=isnan(sum(datafinal_flag17_DUP,2));
datafinal_flag17_DUP(indexnan_flag17,:)=[];

% Remove all data sets that fall outside of the constraints outlined below
out2=mean(datafinal_flag17_DUP(:,200));
index_flag17 = datafinal_flag17_DUP(:,200) < (out2-1*std(datafinal_flag17_DUP(:,
200))) | datafinal_flag17_DUP(:,200) > (out2+1*std(datafinal_flag17_DUP(:,200)));
datafinal_flag17_DUP(index_flag17,:) = [];

% Report statistics about dropped data ----------------------------------
nanCount_flag17 = sum(indexnan_flag17);

% Display the original size of the data matrix
originalSize_flag17 = size(datafinal_flag17)

constraintDropped_flag17 = sum(index_flag17);

% Display the total number of data sets dropped
totalDropped_flag17 = constraintDropped_flag17 + nanCount_flag17

% Plot the refined data set, with outliers removed
```

```
figure(3)
plot(wavelength,datafinal_flag17_DUP)

% Save the refined data set, with outliers removed
save('flag17_FINAL.mat', 'datafinal_flag17_DUP');

% Plot the original data set for reference
figure(4)
plot(wavelength,datafinal_flag17)

%Insert breakpoint


% --------------------------------------------------------------------
%% PROCESS FLAG 19 MEASUREMENTS
% --------------------------------------------------------------------

clear;
clc;
close all;

% Organize measurements based on associated plant number
flag19_files = dir('*flag19*.mat');

% Determine how many flag19 files there are
num_flag19_files = length(flag19_files);
datafinal_flag19=[];

% Create a loop that loads each flag19 file into an array
for k = 1:num_flag19_files

    % Identify the k-th file in the structure 'flag19_files'
    filename=flag19_files(k).name;

    % Load the k-th file
    [pathstr,name,ext]=fileparts(filename);
    load(filename)

    % Normalize the data matrix by dividing each element by the sum of the
    % data points
    data=data/sum(data);

    % Save the normalized data points to the 'datafinal_flag19' matrix
    datafinal_flag19=[datafinal_flag19;data];

end

% Create a duplicate of the datafinal_flag19 matrix, datafinal_flag19_DUP
datafinal_flag19_DUP=datafinal_flag19;
```

```
% Remove all infinite values from the datafinal duplicate matrix
indexnan_flag19=isnan(sum(datafinal_flag19_DUP,2));
datafinal_flag19_DUP(indexnan_flag19,:)=[];

% Remove all data sets that fall outside of the constraints outlined below
out3=mean(datafinal_flag19_DUP(:,200));
index_flag19 = datafinal_flag19_DUP(:,200) < (out3-1*std(datafinal_flag19_DUP(:,
200))) | datafinal_flag19_DUP(:,200) > (out3+1*std(datafinal_flag19_DUP(:,200)));
datafinal_flag19_DUP(index_flag19,:) = [];

% Report statistics about dropped data ----------------------------------
nanCount_flag19 = sum(indexnan_flag19);

% Display the original size of the data matrix
originalSize_flag19 = size(datafinal_flag19)

constraintDropped_flag19 = sum(index_flag19);

% Display the total number of data sets dropped
totalDropped_flag19 = constraintDropped_flag19 + nanCount_flag19

% Plot the refined data set, with outliers removed
figure(5)
plot(wavelength,datafinal_flag19_DUP)

% Save the refined data set, with outliers removed
save('flag19_FINAL.mat', 'datafinal_flag19_DUP');

% Plot the original data set for reference
figure(6)
plot(wavelength,datafinal_flag19)

%Insert breakpoint


% ------------------------------------------------------------------------
%% PROCESS SPREADER ORIGIN MEASUREMENTS
% ------------------------------------------------------------------------

clear;
clc;
close all;

% Organize measurements based on associated plant number
spreaderOrigin_files = dir('*spreader_origin*.mat');

% Determine how many spreader_origin files there are
num_spreaderOrigin_files = length(spreaderOrigin_files);
```

```
datafinal_spreaderOrigin=[];

% Create a loop that loads each spreader_origin file into an array
for k = 1:num_spreaderOrigin_files

    % Identify the k-th file in the structure 'spreaderOrigin_files'
    filename=spreaderOrigin_files(k).name;

    % Load the k-th file
    [pathstr,name,ext]=fileparts(filename);
    load(filename)

    % Normalize the data matrix by dividing each element by the sum of the
    % data points
    data=data/sum(data);

    % Save the normalized data points to the 'datafinal_spreaderOrigin' matrix
    datafinal_spreaderOrigin=[datafinal_spreaderOrigin;data];

end

% Create a duplicate of the datafinal_spreaderOrigin matrix,
datafinal_spreaderOrigin_DUP
datafinal_spreaderOrigin_DUP=datafinal_spreaderOrigin;

% Remove all infinite values from the datafinal duplicate matrix
indexnan_spreaderOrigin=isnan(sum(datafinal_spreaderOrigin_DUP,2));
datafinal_spreaderOrigin_DUP(indexnan_spreaderOrigin,:)=[];

% Remove all data sets that fall outside of the constraints outlined below
out4=mean(datafinal_spreaderOrigin_DUP(:,200));
index_spreaderOrigin = datafinal_spreaderOrigin_DUP(:,200) < (out4-1*std
(datafinal_spreaderOrigin_DUP(:,200))) | datafinal_spreaderOrigin_DUP(:,200) >
(out4+1*std(datafinal_spreaderOrigin_DUP(:,200)));
datafinal_spreaderOrigin_DUP(index_spreaderOrigin,:) = [];

% Report statistics about dropped data ----------------------------------
nanCount_spreaderOrigin = sum(indexnan_spreaderOrigin);

% Display the original size of the data matrix
originalSize_spreaderOrigin = size(datafinal_spreaderOrigin)

constraintDropped_spreaderOrigin = sum(index_spreaderOrigin);

% Display the total number of data sets dropped
totalDropped_spreaderOrigin = constraintDropped_spreaderOrigin +
nanCount_spreaderOrigin

% Plot the refined data set, with outliers removed
```

```
figure(7)
plot(wavelength,datafinal_spreaderOrigin_DUP)

% Save the refined data set, with outliers removed
save('spreaderOrigin_FINAL.mat', 'datafinal_spreaderOrigin_DUP');

% Plot the original data set for reference
figure(8)
plot(wavelength,datafinal_spreaderOrigin)

%Insert breakpoint


% --------------------------------------------------------------------------
%% PROCESS SPREADER MID MEASUREMENTS
% --------------------------------------------------------------------------

clear;
clc;
close all;

% Organize measurements based on associated plant number
spreaderMid_files = dir('*spreader_mid*.mat');

% Determine how many spreader_mid files there are
num_spreaderMid_files = length(spreaderMid_files);
datafinal_spreaderMid=[];

% Create a loop that loads each spreader_mid file into an array
for k = 1:num_spreaderMid_files

    % Identify the k-th file in the structure 'spreaderMid_files'
    filename=spreaderMid_files(k).name;

    % Load the k-th file
    [pathstr,name,ext]=fileparts(filename);
    load(filename)

    % Normalize the data matrix by dividing each element by the sum of the
    % data points
    data=data/sum(data);

    % Save the normalized data points to the 'datafinal_spreaderMid' matrix
    datafinal_spreaderMid=[datafinal_spreaderMid;data];

end

% Create a duplicate of the datafinal_spreaderMid matrix, datafinal_spreaderMid_DUP
datafinal_spreaderMid_DUP=datafinal_spreaderMid;
```

```
% Remove all infinite values from the datafinal duplicate matrix
indexnan_spreaderMid=isnan(sum(datafinal_spreaderMid_DUP,2));
datafinal_spreaderMid_DUP(indexnan_spreaderMid,:)=[];

% Remove all data sets that fall outside of the constraints outlined below
out5=mean(datafinal_spreaderMid_DUP(:,200));
index_spreaderMid = datafinal_spreaderMid_DUP(:,200) < (out5-1*std
(datafinal_spreaderMid_DUP(:,200))) | datafinal_spreaderMid_DUP(:,200) > (out5+1*std
(datafinal_spreaderMid_DUP(:,200)));
datafinal_spreaderMid_DUP(index_spreaderMid,:) = [];

% Report statistics about dropped data ----------------------------------
nanCount_spreaderMid = sum(indexnan_spreaderMid);

% Display the original size of the data matrix
originalSize_spreaderMid = size(datafinal_spreaderMid)

constraintDropped_spreaderMid = sum(index_spreaderMid);

% Display the total number of data sets dropped
totalDropped_spreaderMid = constraintDropped_spreaderMid + nanCount_spreaderMid

% Plot the refined data set, with outliers removed
figure(9)
plot(wavelength,datafinal_spreaderMid_DUP)

% Save the refined data set, with outliers removed
save('spreaderMid_FINAL.mat', 'datafinal_spreaderMid_DUP');

% Plot the original data set for reference
figure(10)
plot(wavelength,datafinal_spreaderMid)
```

Appendix B

Programming Solution

B.1 | PCA Script

11/15/17 6:06 PM E:\Penn State\2-Th...\PCA Only Script.m 1 of 2

```
% PCA Script
% Philip Ryan
% Senior Thesis - FA 17

% Input: Preprocessed data for each plant in separate MS Excel
% spreradsheets

% Processing:
% 1. Load the Excel files
% 2. Remove the noisy edges of the spectra
% 3. Apply PCA; request coefficients and scores

% Output: PCA scores for each plant variety

tic

clear
clc
close all

% Assign the directory to the current folder
mainFolder = cd('E:\Penn State\2-Thesis\2-Code\Final Code');


%% Load the Processed Data ------------------------------------------------

mainFolder;

% Combine all of the Flag 4 measurements in an Excel spreadsheet.  Load the
% excel spreadsheet into MATLAB.
flag4_Import = xlsread('Flag4_Prelims.xlsx');

% Repeat for remaining Flag 17, Flag 19, Spreader Mid, Spreader Origin data
flag17_Import = xlsread('Flag17_Prelims.xlsx');
flag19_Import = xlsread('Flag19_Prelims.xlsx');
spreaderMid_Import = xlsread('SpreaderMid_Prelims.xlsx');
spreaderOrigin_Import = xlsread('SpreaderOrigin_Prelims.xlsx');

% Select the data corresponding to the middle of the spectra.  The received
% values at the extrema will not be indicative of disease.
flag4 = flag4_Import(:,182:366);
flag17 = flag17_Import(:,182:366);
flag19 = flag19_Import(:,182:366);
spreaderMid = spreaderMid_Import(:,182:366);
spreaderOrigin = spreaderOrigin_Import(:,182:366);
```

```
%% Use Principal Component Analysis for Dimensionality Reduction
% Use principal component analysis to map the data from a higher dimension
% to a lower dimension of principal components.

[coeff_flag4, score_flag4] = pca(flag4);
[coeff_flag17, score_flag17] = pca(flag17);
[coeff_flag19, score_flag19] = pca(flag19);
[coeff_spreaderOrigin, score_spreaderOrigin] = pca(spreaderOrigin);
[coeff_spreaderMid, score_spreaderMid] = pca(spreaderMid);

% Keep the first two principal component scores; save them to Y_flagX
% matrices
Y_flag4 = score_flag4(:,1:2);
Y_flag17 = score_flag17(:,1:2);
Y_flag19 = score_flag19(:,1:2);
Y_spreaderOrigin = score_spreaderOrigin(:,1:2);
Y_spreaderMid = score_spreaderMid(:,1:2);
```

B.2 | *k*-Means Clustering and Plotting Script

11/15/17 6:07 PM ...\K Means One at a Time Flag4 FINAL.m 1 of 4

```
% K-means Clustering Flag 4
% Philip Ryan
% Senior Thesis - FA 17

% Input: PCA scores after 1, 3, and 5 days

% Processing:
% 1. Define k = 2 (healthy v. unhealthy) and mesh grid interval
% 2. Define k-means algorithm parameters
% 3. Execute the k-means algorithm
% 4. Plot the results
% 5. Manually construct the confusion matrices

% Output: Color-coded scatter plots, confusion matrices

tic

clear
clc
close all

% Assign the directory to the current folder
mainFolder = cd('E:\Penn State\2-Thesis\2-Code\Final Code\PCA Data - All at Once\Flag 4');


%% Load the PCA Scores

% Read in the Excel files
Y_Day1 = xlsread('Flag4_1Day.xlsx');
Y_Day3 = xlsread('Flag4_3Days.xlsx');
Y_Day5 = xlsread('Flag4_5Days.xlsx');


%% K-Means Parameters

% Assign appropriate interval spacing for the mesh grids
meshInterval = 1.0e-05;

% Assign value for k
k = 2;


%% K-Means After Day 1

% Define algorithm parameters
[idx_Day1, C_Day1] = kmeans(Y_Day1, k);

% Compute the difference between the maximum and minimum values.  Subdivide
```

```
% the range to create a mesh grid.  The intervals between the max and the
% min should be a couple of orders of magnitude smaller than the difference
% of the two values
x1_Day1 = (min(Y_Day1(:,1))):(meshInterval):(max(Y_Day1(:,1)));
x2_Day1 = (min(Y_Day1(:,2))):(meshInterval):(max(Y_Day1(:,2)));

% Create a matrix for the mesh grid
[x1_Day1_GRD, x2_Day1_GRD] = meshgrid(x1_Day1, x2_Day1);

% Define a grid on the mesh plot
XGrid_Day1 = [x1_Day1_GRD(:), x2_Day1_GRD(:)];

% Assign each node in the mesh to the closest centroid
idx_Day1_Region = kmeans(XGrid_Day1,k,'MaxIter',1000,'Start',C_Day1);

% Plot the cluster regions
figure(1)
gscatter(XGrid_Day1(:,1), XGrid_Day1(:,2), idx_Day1_Region,[],'..');
hold on;

% Plot Y 1 and Y 2 on the clustered plane
plot(Y_Day1(1:104,1), Y_Day1(1:104,2),'g.','MarkerSize',5);
plot(Y_Day1(105:158,1), Y_Day1(105:158,2),'k.','MarkerSize',5);
plot(C_Day1(1,1),C_Day1(1,2),'bl*','MarkerSize',10);
plot(C_Day1(2,1),C_Day1(2,2),'bl*','MarkerSize',10);
title('Clustered Flag 4 Observations After 1 Day');
legend('Region 1','Region 2','Data','Location','SouthEast');
xlabel('PCA Y 1');
ylabel('PCA Y 2');
hold off;


%% K-Means After Day 3

% Define algorithm parameters
[idx_Day3, C_Day3] = kmeans(Y_Day3, k);

% Compute the difference between the maximum and minimum values.  Subdivide
% the range to create a mesh grid.  The intervals between the max and the
% min should be a couple of orders of magnitude smaller than the difference
% of the two values
x1_Day3 = (min(Y_Day3(:,1))):(meshInterval):(max(Y_Day3(:,1)));
x2_Day3 = (min(Y_Day3(:,2))):(meshInterval):(max(Y_Day3(:,2)));

% Create a matrix for the mesh grid
[x1_Day3_GRD, x2_Day3_GRD] = meshgrid(x1_Day3, x2_Day3);

% Define a grid on the mesh plot
XGrid_Day3 = [x1_Day3_GRD(:), x2_Day3_GRD(:)];
```

```
% Assign each node in the mesh to the closest centroid
idx_Day3_Region = kmeans(XGrid_Day3,k,'MaxIter',1000,'Start',C_Day3);

% Plot the cluster regions
figure(2)
gscatter(XGrid_Day3(:,1), XGrid_Day3(:,2), idx_Day3_Region,[],'..');
hold on;

% Plot Y 1 and Y 2 on the clustered plane
plot(Y_Day3(1:104,1), Y_Day3(1:104,2),'g.','MarkerSize',5);
plot(Y_Day3(105:197,1), Y_Day3(105:197,2),'k.','MarkerSize',5);
plot(C_Day3(1,1),C_Day3(1,2),'bl*','MarkerSize',10);
plot(C_Day3(2,1),C_Day3(2,2),'bl*','MarkerSize',10);
title('Clustered Flag 4 Observations After 3 Days');
legend('Region 1','Region 2','Data','Location','SouthEast');
xlabel('PCA Y 1');
ylabel('PCA Y 2');
hold off;


%% K-Means After Day 5

% Define algorithm parameters
[idx_Day5, C_Day5] = kmeans(Y_Day5, k);

% Compute the difference between the maximum and minimum values.  Subdivide
% the range to create a mesh grid.  The intervals between the max and the
% min should be a couple of orders of magnitude smaller than the difference
% of the two values
x1_Day5 = (min(Y_Day5(:,1))):(meshInterval):(max(Y_Day5(:,1)));
x2_Day5 = (min(Y_Day5(:,2))):(meshInterval):(max(Y_Day5(:,2)));

% Create a matrix for the mesh grid
[x1_Day5_GRD, x2_Day5_GRD] = meshgrid(x1_Day5, x2_Day5);

% Define a grid on the mesh plot
XGrid_Day5 = [x1_Day5_GRD(:), x2_Day5_GRD(:)];

% Assign each node in the mesh to the closest centroid
idx_Day5_Region = kmeans(XGrid_Day5,k,'MaxIter',1000,'Start',C_Day5);

% Plot the cluster regions
figure(3)
gscatter(XGrid_Day5(:,1), XGrid_Day5(:,2), idx_Day5_Region,[],'..');
hold on;

% Plot Y 1 and Y 2 on the clustered plane
plot(Y_Day5(1:104,1), Y_Day5(1:104,2),'g.','MarkerSize',5);
```

11/15/17 6:07 PM ...\K Means One at a Time Flag4 FINAL.m 4 of 4

```
plot(Y_Day5(105:178,1), Y_Day5(105:178,2),'k.','MarkerSize',5);
plot(C_Day5(1,1),C_Day5(1,2),'bl*','MarkerSize',10);
plot(C_Day5(2,1),C_Day5(2,2),'bl*','MarkerSize',10);
title('Clustered Flag 4 Observations After 5 Days');
legend('Region 1','Region 2','Data','Location','SouthEast');
xlabel('PCA Y 1');
ylabel('PCA Y 2');
hold off;
```

Note: A very similar script is used for Flags 17 & 19, as well as Spreader Mid and Spreader Origin. The only modifications to these scripts are the variable names.

REFERENCES

[1] U. S. D. o. Agriculture, "About the U.S. Department of Agriculture," U.S. Department of Agriculture, [Online]. Available: https://www.usda.gov/our-agency/about-usda. [Accessed 1 September 2017].

[2] National Potato Council, "All Potatoes Price per Cwt and Value of Production - States and United States: 2014-2016," National Potato Council, Washington, DC, 2016.

[3] Food and Agriculture Organization of the United Nations, Technical Cooperation Department, "Farming Systems and Poverty," Food and Agriculture Organization of the United Nations, Technical Cooperation Department, 2000. [Online]. Available: http://www.fao.org/docrep/004/ac349e/ac349e03.htm. [Accessed 24 August 2017].

[4] B. J. Christ, "Identifying Potato Diseases in Pennsylvania," The Pennsylvania State University, University Park, PA, 1999.

[5] P. Council, "What is potato blight?," Agriculture and Horticulture Development Board, Stoneleigh Park, Kenilworth, Warwickshire, 2013.

[6] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development,* vol. 3, no. 3, pp. 210-229, 1959.

[7] S. Russell, Artificial Intelligence: A Modern Approach, Upper Saddle River, NJ: Prentice Hall, 2003.

[8] P. G. A. B. Jean-Emmanuel Bibault, "Big Data and machine learning in radiation oncology: State of the art and future prospects," *Cancer Letters,* vol. 382, no. 1, pp. 110-117, 2016.

[9] R. K. M. Indranil Bose, "Business data mining - a machine learning perspective," *Information & Management,* vol. 39, no. 3, pp. 211-225, 2001.

[10] A. Ng, "Introduction to Machine Learning," Stanford University, Stanford, CA, 2012.

[11] D. o. S. O. Programs, "What is Simple Linear Regression?," The Pennsylvania State University Eberly College of Science, University Park, PA, 2017.

[12] A. Ng, "Linear regression with one variable: Model representation," Stanford University, Stanford, CA, 2017.

[13] A. Ng, "CS229 Lecture Notes: Supervised Learning," Stanford University, Stanford, CA, 2017.

[14] C.-J. H. K.-W. C. M. R. C.-J. L. Yin-Wen Chang, "Training and Testing Low-degree Polynomial Data Mappings via Linear SVM," *Journal of Machine Learning Research,* vol. 11, pp. 1471-1490, 2010.

[15] A. Ng, "Regularization: The problem of overfitting," Stanford University, Stanford, CA, 2017.

[16] K. Tretyakov, "Machine Learning Techniques in Spam Filtering," in *Data Mining Problem-oriented Seminar, MTAT.03.177,* 2004.

[17] K. Dunn, "Geometric explanation of PCA," Process Improvement Using Data, 4 August 2017. [Online]. Available: https://learnche.org/pid/latent-variable-modelling/principal-component-analysis/geometric-explanation-of-pca. [Accessed 31 October 2017].

[18] A. Ng, "CS229 Lecture Notes: Principal Component Analysis," Stanford University, Stanford, CA, 2017.

[19] A. Ng, "CS229 Lecture Notes: Unsupervised Learning," Stanford University, Stanford, CA, 2017.

[20] MathWorks Documentation, "kmeans," [Online]. Available: https://www.mathworks.com/help/stats/kmeans.html. [Accessed 31 October 2017].

[21] e. a. Perry Edwards, "Smartphone based optical spectrometer for diffusive reflectance spectroscopic measurement of hemoglobin," Macmillan Publishers Limited, London, England, 2017.

[22] G. C. P. E. M.-D. Z. S. Z. a. Z. L. Chenji Zhang, "G-Fresnel smartphone spectrometer," The Royal Society of Chemistry, London, England, 2015.

[23] MathWorks, "Principal component analysis of raw data," 21 September 2017. [Online]. Available: https://www.mathworks.com/help/stats/pca.html. [Accessed 31 October 2017].

[24] MathWorks, "k-means Clustering," September 2017. [Online]. Available: https://www.mathworks.com/help/stats/kmeans.html. [Accessed 31 October 2017].

[25] MathWorks, "Mesh grid 2-D and 3-D grids," September 2017. [Online]. Available: https://www.mathworks.com/help/matlab/ref/meshgrid.html. [Accessed 31 October 2017].

[26] H. P. E. R. C. W. Nathan Landman, "k-Means Clustering," Brilliant.org, [Online]. Available: https://brilliant.org/wiki/k-means-clustering/. [Accessed 31 October 2017].

ACADEMIC VITA

Philip John Ryan Jr.

Contact pjr5219@gmail.com
+1.215.272.5197 (mobile)
51 Rittenhouse Rd.
Harleysville, PA 19438

Education

B.S. Electrical Engineering — Dec. 2017
with honors in Electrical Engineering
The Pennsylvania State University, University Park, PA
College of Engineering, Schreyer Honors College

Thesis

"Machine Learning Techniques for Early Disease Detection in Potato Plants" — Nov. 2017
Supervisor: Zhiwen Liu, Professor of Electrical Engineering

- Created a MATLAB script to visualize high-dimensional data using principal component analysis (PCA)
- Analyzed movement of cluster centroids to determine plant health
- Processed raw data by creating a MATLAB script to remove outliers

Relevant Coursework

Antenna Engineering (EE 438)
E.M. Field Theory (EE 430)
Applied E.M. (EE 330)
Engr. Mech. (E MCH 211, 212, 213)
Educational Psych. (EDPSY 014)
Electronic Design I, II (EE 310, 311)
Energy Conversion (EE 387)
Electric Machinery (EE 487)
Power Systems (EE 488)
Teaching Intern Seminar (ENGR 497)

Work Experience

Corporate Development Intern — May 2017 – Aug. 2017
MACOM Technology Solutions, Lowell, MA

- Communicated technical information about MACOM's products to coworkers with non-technical backgrounds
- Created graphical representations of revenue data to assist the executive management team with strategic planning
- Collaborated with other departments to help complete a vertical Acquisition; maintained confidentiality of sensitive information
- Developed emotional intelligence by shadowing members of the executive management team and attending a quarterly business review

Electrical Engineering Intern — May 2016 – May 2017
ATOPTIX, LLC, State College, PA

- Improved existing MATLAB scripts to analyze large sets of data faster
- Soldered surface-mount components to printed circuit boards
- Modeled and produced prototypes using SolidWorks and a 3D printer
- Learned how to calibrate and test optical sensing equipment

Teaching Experience

Undergraduate Teaching Intern Aug. 2016 – May 2017
Department of Electrical Engineering, University Park, PA

- Developed quizzes to assess student understanding of electrical network analysis and learning objectives
- Presented six electromagnetism lectures to crowds of over 100 undergraduate junior students

Calculus Learning Assistant Aug. 2014 – Dec. 2017
Department of Mathematics, University Park, PA

- Guided groups of 6-8 students through practice problems
- Facilitated learning by encouraging teamwork and problem-solving

Awards

Pennsylvania Technical Assistance Program Scholarship
Eta Kappa Nu, Electrical Engineering Honor Society
Tau Beta Pi, Engineering Honor Society

Activities

Coach and Athlete Aug. 2013 – Dec. 2017
Penn State Club Swim Team

- Motivated teammates to set and achieve academic, athletic, and professional goals for themselves
- Influenced the culture of the team by maintaining a positive training environment

Event Logistics Director and Mentor Jan. 2016 – Nov. 2017
Penn State Engineering Orientation Network

- Coordinated director and mentor availability to schedule and conduct over 100 interviews
- Created an itinerary that rotated over 800 students to different orientation day activities

Dancer, Committee Member, and Volunteer Sept. 2013 – Dec. 2017
Penn State Dance Marathon (THON)

- Danced for 46 hours to raise money & awareness for pediatric cancer treatment and research
- Developed and presented weekly cancer education lessons to an audience of 40 volunteers
- Provided emotional support to a family affected by pediatric cancer through a community outreach program

Assistant Manager and Lifeguard May 2011 – Aug. 2015
Harleysville Community Pool

- Managed 25 lifeguards to maintain a safe recreational facility
- Operated chemical control system to maintain pool water chemistry
- Hold American Red Cross Lifeguarding, First Aid, CPR certifications