12-7-2020

Clustered Hyperspectral Target Detection

Sean Onufer Stalley
Portland State University

Clustered Hyperspectral Target Detection

by

Sean Onufer Stalley

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
John Lipor, Chair
James McNames
Christof Teuscher

Portland State University
2020

Abstract

Aerial target detection is often used to search for relatively small things over large areas of land. Depending on the size and signature of the target, detection can be a very easy or very difficult task. By capturing images with several hundred color channels, hyperspectral sensors provide a new way of looking at this task, both literally and figuratively. Hyperspectral sensors can be used in many aerial target detection tasks such as identifying unhealthy trees in a forest, searching for minerals at a mining site, or finding the sources of chemical leaks at a factory. The high spectral resolution of hyperspectral imagery makes it well suited for these tasks, but the inherent high dimensionality of these images poses a unique set of challenges.

The motivation of this work is to investigate the use of data clustering to improve our ability to detect targets within hyperspectral images. Target detection algorithms operate by identifying locations that are likely to contain a target when compared with the background. We propose a new clustering-based target detection method that allows multiple background models to be used. This new method pairs a clustering algorithm with an array of spectral matched filters. We then analyze the performance of various clustering algorithms when used with this method to detect targets in aerial hyperspectral images.

We evaluate the performance of our clustered target detector on several aerial hyperspectral images when using clusters generated by several popular algorithms, specifically k-means, spectral clustering, Gaussian mixture models, and two variants of subspace clustering. We show empirically that clusters generated by Gaussian mixture models provide the best performance, obtaining a pAUC score of 0.192 in the true positive detection rate on the RIT Radiance image for false positive rates of 1% or less, providing over a 12-fold increase when compared to the pAUC score of 0.0148 obtained for target detection without clustering. We then tune a Laplacian-regularized Gaussian mixture model (LapGMM) algorithm specifically for the task of aerial hyperspectral target detection. We show empirically that our tuned algorithm outperforms all others when used for this task, outpacing the traditional Gaussian mixture model with a pAUC score of 0.219 for the same case above, thereby offering over a 14-fold improvement in performance. We offer several hypotheses to explain these results. We then discuss some of the features, most notably the versatility provided by the regularizer, that make make the tuned LapGMM algorithm well suited for this application.

Considering future work, we propose a number of potential applications for our tuned LapGMM algorithm, as well as several potential improvements or modifications to the clustered target detector that may be worth further investigation. The contributions of this thesis are a detailed investigation and analysis of the use of clustering algorithms when used for target detection, and an analysis of the performance of several clustering algorithms when used in an aerial hyperspectral image application. Additionally, we contribute an algorithm tuned specifically for clustering aerial hyperspectral images, which to the best of our knowledge is state of the art.

to LP and CJ

Acknowledgements

First I would like to thank several former coworkers who either convinced or encouraged me (mostly unknowingly) to return to school full time. Specifically, I would like to thank David Bouse, Carolyn Duran, Dan Froelich, Sybil Grace, Dave Harriman, John Howard, Ivan Lobachev, Ackshay Pethe, Sage Sharp, Daniel Shattuck, Kyle Sheahan, Rupin Vakharwala, Stephanie Wallick, Marc Wells, and Ted Willke.

I would also like to thank my accademic advisor John Lipor for all of his help both technically and professionally, as well as Stanley Rotman, who was a visiting scholar to PSU in 2019. Their influence and guidance on this research was immeasurable, and I certainly would not have been able to complete this thesis without their guidance and expertise.

I would also like to thank the staff and faculty of PSU's Electrical Engineering Department. Their passion for both the material and the students is everpresent, which has personally made my graduate school experience almost as inspirational as it was educational. In particular I would like to thank Melisa Cehajic, Yuchen Huang, James McNames, Mark Martin, Christof Teuscher, Eric Wan, and Ted Willke.

I would also like to thank my peers in PSU's ECE program. I feel fortunate to have had the opportunity to work alongside such bright individuals. Specfically I would like to thank Bruno Duregon, Alejandro Espinoza, Daniel Frister, Cody Henderson, Alex Higgins, Christine Jigau, Ethan Lew, Christopher Neighbor, Phillipe Proctor, Wubin Sheng, Zachery Stamler, and Alex Ruhland.

Finally, I would like to thank my wife Leah, who has been not only supportive but incredibly encouraging of me and my decision to return to school. There have been times when I myself have questioned this decision, but her belief in me was constant and unwavering, which was essential to me overcoming my self-doubt and tackling the challenges I faced while working towards this degree. In short, I would not have been able to do it without her.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Aerial target detection is often used to search for relatively small things over large areas of land. This task can be quite the difficult if the target is small and its color is similar to its surroundings. Hyperspectral sensors offer an alternative way of looking for targets, both literally and figuratively.

New innovations are often discovered when techniques and methods from one field of study are applied to problems in another. A desire to find such innovations was the primary motivation of this research. In this thesis we investigate data clustering techniques from the field of machine learning and see how they can be applied to the signal processing task of target detection.

Hyperspectral sensors can be used in aerial applications to gather a great deal of information about an area, but the inherent high-dimensionality of this information often makes it difficult to effectively utilize. Additionally, hyperspectral sensors do not see things in the same way we do, making their output unintuitive and challenging to interpret. The goal of our work is to investigate the application of data clustering techniques to hyperspectral target detection problems and find out how these techniques can be used to better interpret the information and improve target detection performance.

1.2 Overview of this Document

This thesis focuses on the task of target detection within hyperspectral images. Specifically, it focus on the effect data clustering algorithms can have on point target detection in aerial hyperspectral images.

This chapter provides some background on hyperspectral imaging and lists the contributions of this document. The next chapter covers several target and anomaly detection algorithms, common enhancements to these algorithms, and introduces methods for analyzing and quantifying the performance of a target detector. Chapter 3 covers several popular clustering algorithms and introduces the clustered target detector, a target detection algorithm that utilizes clustering to improve performance. In Chapter 4 we show the performance of our clustered target detector when using different clustering algorithms, and tune a clustering algorithm specifically for our hyperspectral use case. In Chapter 5 we discuss the implications of our findings, and highlight some potential avenues for future work.

1.2.1 Hyperspectral Imaging Background

Hyperspectral Imaging is an exciting technology that allows us to capture images with details beyond what our eyes can see. Before we discuss hyperspectral imaging in detail, it is important to understand some concepts behind light, photography and digital imaging.

Light is the range of electromagnetic radiation that is visible to humans. Any photon with a wavelength between 400 and 700 nanometers (nm) is light. At its core, photography is just the practice of capturing and recording light. A camera is simply a sensor that measures the amount of photons received. What sets different cameras apart is how they subdivide and categorize the photons.

Most cameras subdivide the photons in two ways: propagation direction and wavelength. Digital cameras capture images by measuring the number of photons from each direction and storing these values in units known as pixels. Pixels can be plotted on a two dimensional graph to create a projection of whatever the camera was pointed at (the subject). In that sense a photograph can be considered a two dimensional graph of photon intensity (i.e., a photon graph). The quality of an image is often measured by the number of pixels it contains, as images with greater pixel count can capture more visual details of the original subject.

Pixels are further subdivided into color channels, with each channel corresponding to the number of photons received that fall within a particular range (or 'band') of wavelengths. Pixels in greyscale images contain a single color channel that measures all the photons received within the band of visible light, whereas pixels in color images contain three channels (red, green, and blue), corresponding to the three bands of light that are differentiable by humans [7].

Three channels are used in color images because the intended viewer (trichromat humans) can only see three color bands. If the intended viewer of the image is human, then there is no benefit to capturing additional channels of light. That said, if the image is to be viewed by a non-human (such as a target detection algorithm), then additional channels can be used to capture additional information about the subject. There are three categories of sensors that capture more than the three traditional color channels, specifically:

1. *Multispectral sensors* — These sensors capture application-specific bands in addition to the three human-differentiable bands of light. One example would be the sensors on LANDSAT satellites [8], which (depending on the specific satellite) capture between 1 and 8 channels in addition to red, green and blue. Of the three categories of spectral sensors, Multispectral sensors tend to have the highest pixel resolution.

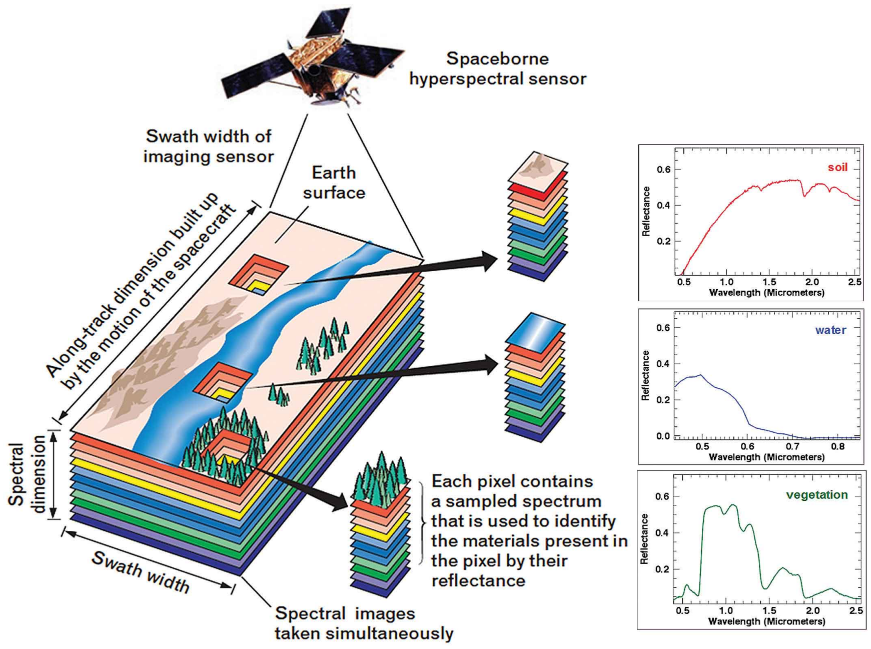
Spaceborne hyperspectral sensor
Swath width of imaging sensor
Earth surface
Along-track dimension built up by the motion of the spacecraft
Spectral dimension
Swath width
Spectral images taken simultaneously
Each pixel contains a sampled spectrum that is used to identify the materials present in the pixel by their reflectance
soil
water
vegetation
Reflectance
Wavelength (Micrometers)

Figure 1.1: Using hyperspectral imaging to identify the materials and ground covering present in a pixel. Each type of ground cover can be identified by its unique spectral signature. Taken from [1].

2. *Hyperspectral sensors* — These sensors capture a large number (100+) of channels with narrow (≈ 1 to 20 nm) contiguous bands. Hyperspectral sensors usually cover the full range of visible light, but do not capture the 3 human-visible bands of light as independent channels. In contrast to multispectral sensors, hyperspectral sensor bands are not chosen for a particular application. Instead, the bands are chosen so that the channels of each pixel can be combined to form a spectrograph. An example of a hyperspectral sensor would be the OCI-F Hyperspectral Imager [9], which captures 240 evenly-spaced contiguous bands between 400 and 1000 nm.

3. *Ultraspectral sensors* — Ultraspectral sensors have a very fine spectral resolution (< 0.1 nm) but very low pixel resolution (often only one pixel). An example would be an interferometer such as the The MK II Fraunhofer Line Discriminator (FLD-II) [10].

The major trade-off between these three types of sensors is spatial vs. spectral resolution. Multispectral sensors offer the greatest pixel resolution whereas ultraspectral sensors offer the greatest spectral resolution. The factor that limits both of these types of resolution is the number of photons received by the sensor. In order for a sensor to operate correctly, each color channel within each pixel must receive enough photons to make an accurate measurement. Smaller pixels receive less photons, so they must sense over a wider spectral band of light to make an accurate measurement. Similarly, narrower spectral bands of light contain less photons, so a larger pixel is needed to make an accurate measurement. As an example of this trade off is the Panchromatic senor on LANDSAT 8. Being monochromatic, it has a very wide spectral band, covering most of the visible spectrum, but also offers 4× as much spatial resolution when compared to the other sensors on the satellite. In contrast, hyperspectral sensors offer have over one hundred narrow-banded channels throughout the visible spectrum, offering high spectral resolution but with lower spatial resolution. In aerial hyperspectral images a single pixel is typically over one square meter. While this low spatial resolution makes hyperspectral imaging unsuitable for some applications, the high spectral resolution makes it especially suited for a variety of applications. Figure 1.1 shows an example use case for hyperspectral imaging, identifying ground covering from aerial images.

In the next chapter, we introduce several target detection algorithms that can be used to find things in hyperspectral images. In Chapter 3 we show how data clustering can be used to classify ground coverings in aerial hyperspectral images and improve the performance of a target detection algorithm.

1.3 Contributions

Most of the target detection problems that were overcome in this thesis were done so by implementing advanced clustering methods. Many of these problems have admittedly already been solved through other techniques. That said, when you have a lot of experience with a hammer, every problem looks like a nail, and when you are developing a clustering algorithm, every problem looks like a potential application for clustering to overcome. As such, most of the contributions of this thesis are application-specific clustering considerations.

1.3.1 Clustering Considerations for Hyperspectral Images

In this thesis we provide empirical results showing the performance of various clustering algorithms when used in a hyperspectral target detection application. Our results show that many algorithms work well for clustering aerial hyperspectral images, and many do not. In addition to showing the empirical results, we provide hypotheses to potentially explain why some algorithms performed the way they did, and in Section 5.2 list some potential ways poorly performing algorithms could be better utilized. To the best of our knowledge, this is the first paper to investigate Gaussian mixture model clustering in any Hyperspectral application, and (as covered in Chapter 3) was listed as future work by several sources.

1.3.2 Clustering Considerations for Spectral Matched Filters

In Chapter 3 we discuss some of the properties of GMM and LapGMM that make them especially well suited for clustering data for spectral matched filters (SMFs). In Section 5.2 we discuss additional filters we believe these algorithms would pair well with, extending the use of these algorithms to tasks such as anomaly detection.

1.3.3 A Clustering Algorithm Tuned Specifically for Aerial Hyperspectral Images

In Section 4.5 we use spectral initialization and a Laplacian regularizer to tune the LapGMM algorithm specifically for the task of aerial hyperspectral target detection. To the best of our knowledge, this algorithm is the state of the art method for clustering aerial hyperspectral images. In Section 5.2 we discuss additional applications we believe this algorithm would be well suited for.

Chapter 2

Target Detection Algorithms

2.1 Overview

Target detection is the ability to find a specified target within an image. Some example applications are given in Section 1.3.

The goal of all target detection algorithms is to distinguish between two hypotheses

$$\begin{aligned} &H_0 : \text{Target Absent} \\ &H_1 : \text{Target Present.} \end{aligned} \tag{2.1}$$

This thesis focuses on algorithms for signature-based point target detection in hyperspectral image analysis. For a more comprehensive overview of other hyperspectral target detection algorithms, such as subspace- and support-vector-based approaches, see [11, 2].

Point target detection is the ability to find a target signature within a single pixel. For point target detection, the hypotheses in (2.1) can be formulated as:

$$\begin{aligned} &H_0 : \mathbf{x} = \mathbf{w} && \text{Target Absent} \\ &H_1 : \mathbf{x} = \alpha\mathbf{s} + (1-\alpha)\mathbf{w} && \text{Target Present,} \end{aligned} \tag{2.2}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the pixel vector, $\mathbf{s} \in \mathbb{R}^d$ is the target signature vector, α is the target strength and $\mathbf{w} \in \mathbb{R}^d$ is the background noise.

The difficulty of finding a target in a given image depends largely on the other things in the image. For example, it would be much easier to find Waldo if nobody else was in the image (and much more difficult if everyone was dressed like him [12]). In order to accurately determine when a target is present in an image we must accurately model the other things in the image, commonly known as the background. All of the algorithms discussed in this chapter model the background as a multivariate Gaussian distribution, i.e., $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{B}}, \mathbf{C}_{\mathbf{B}})$, where $\boldsymbol{\mu}_{\mathbf{B}}$ is the mean of the background and $\mathbf{C}_{\mathbf{B}}$ is the associated $d \times d$ covariance matrix. In Chapter 3 we use clustering to enhance our background model to come from of multiple multivariate Gaussian distributions.

Multivariate Gaussian distributions have several properties that make them effective for modeling the values of pixels within an image. They allow us to characterize the values of each color channel in statistical terms like the mean and variance, and allow us to model and quantify any correlations between color channels. From the Gaussian model we can identify pixels that likely contain a target, as well as anomalous pixels that differ from the background in statistically significant ways [13].

Modeling the background as a Gaussian distribution also allows us to take advantage of the extensive set of mathematical tools developed for Gaussian distributions, such as data whitening [14] and the Mahalanobis distance [15]. Data whitening is the process of linearly transforming a dataset such that every dimension becomes uncorrelated and has unit variance, i.e., $X_{\text{whitened}} = WX$, where W is a $d \times d$ transformation matrix resulting in $\mathbf{C}_{\text{whitened}} = \mathbf{I}$. As shown in [14], $\mathbf{C}_{\mathbf{B}}^{-\frac{1}{2}}$ can be used as a whitening filter, where $\mathbf{C}_{\mathbf{B}}^{-\frac{1}{2}}(\mathbf{C}_{\mathbf{B}}^{-\frac{1}{2}})^T = \mathbf{C}_{\mathbf{B}}^{-1}$, and $\mathbf{C}_{\mathbf{B}}^{-1}$ is any matrix that satisfies $\mathbf{C}_{\mathbf{B}}^{-1}\mathbf{C}_{\mathbf{B}} = \mathbf{C}_{\mathbf{B}}\mathbf{C}_{\mathbf{B}}^{-1} = \mathbf{I}$.

As described in [15], the Mahalanobis distance is a measurement between two points of a given Gaussian distribution. It measures the difference between two points while accounting for the difference in variance between dimensions of the distribution. It is defined as

$$\mathbf{dist}_M(\mathbf{x}, \mathbf{s}) = \sqrt{(\mathbf{x} - \mathbf{s})^T \mathbf{C}_{\mathbf{B}}^{-1} (\mathbf{x} - \mathbf{s})}. \tag{2.3}$$

The Mahalanobis distance between a point and the mean of a given distribution measures the number of standard deviations that point is away from the mean.

In hyperspectral images the background variance varies widely by dimension. For example, in aerial hyperspectral images some wavelengths are attenuated heavily by the atmosphere, while others are not. The dimensions corresponding to the attenuated wavelengths have very low variance per channel, while the unattenuated wavelengths can vary widely. We want our target detectors to be more sensitive in low variance dimensions and less sensitive in high variance dimensions, and using the Mahalanobis gives us this scaled sensitivity. Equation (2.3) can be rearranged to show that the Mahalanobis distance is simply a 'whitened' Euclidean distance

$$\begin{aligned}\mathbf{dist}_M(\mathbf{x},\mathbf{s}) &= \sqrt{\left(\mathbf{C}_\mathbf{B}^{-\frac{1}{2}}(\mathbf{x}-\mathbf{s})\right)^T \left(\mathbf{C}_\mathbf{B}^{-\frac{1}{2}}(\mathbf{x}-\mathbf{s})\right)} \\ &= \sqrt{(W(\mathbf{x}-\mathbf{s}))^T (W(\mathbf{x}-\mathbf{s}))} \\ &= \|W\mathbf{x} - W\mathbf{s}\|_2 .\end{aligned} \tag{2.4}$$

2.2 Spectral Matched Filter

The Spectral Matched Filter (SMF) [16] is a simple but powerful target detection algorithm, and is the optimal linear filter in terms of the signal to noise ratio (SNR) for distinguishing between hypotheses in (2.2) [17]. These properties make it the SMF the de facto target detection algorithm for hyperspectral images, and therefore is the detector this thesis primarily focuses on. The SMF is a constant false-alarm rate (CFAR) detector that is derived from a generalized-likelihood ratio test (GLRT) [13]. The SMF operates by constructing a simple model of the background, then seeing if the pixel in question $\mathbf{x}$ stands out from the background $\mathbf{B}$ in the same way as the target $\mathbf{s}$ would. To construct a matched filter, all that is required is the target signature $\mathbf{s}$, as well as the first- and second- order statistics of the background. This yields the decision rule

$$D_{\mathrm{MF}}(\mathbf{x}) = \frac{(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}})^T \mathbf{C}_{\mathbf{B}}^{-1}(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})}{\sqrt{(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})^T \mathbf{C}_{\mathbf{B}}^{-1}(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})}} \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\mathrm{MF}}, \tag{2.5}$$

where $\boldsymbol{\mu}_{\mathbf{B}}$ is the mean of the background, $\mathbf{C}_{\mathbf{B}}^{-1}$ is the inverse of the background covariance matrix, and η_{MF} is the chosen detection threshold. We discuss choosing a detection threshold in Section 2.8.

In practice we do not know the true mean or covariance of the background, so we estimate them using the following formulas [18]:

$$\hat{\boldsymbol{\mu}_{\mathbf{B}}} = \frac{1}{N}\mathbf{1}_N^T\mathbf{X} = \textstyle\sum_{i=1}^N \mathbf{x}_i, \tag{2.6}$$

$$\hat{\mathbf{C}_{\mathbf{B}}} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X} - \hat{\boldsymbol{\mu}_{\mathbf{B}}}\hat{\boldsymbol{\mu}_{\mathbf{B}}}^T, \tag{2.7}$$

where N is the number of pixels, $\mathbf{1}_N$ is a vector of all 1s, and $\mathbf{X}$ is an $N \times d$ matrix where each row is a single pixel, and d is the number of channels per pixel (also known as the dimensionality).

Since we are using estimates for our background statistics, (2.5) implicitly imposes the requirement that the number of pixels N is greater than or equal to the dimensionality d, (i.e., $N \geq d$). In order to invert the covariance matrix $\mathbf{C}_{\mathbf{B}}$ it must be full rank, (i.e., rank d), and in order for a covariance matrix to be rank d, $\mathbf{X}$ must have a rank equal-to or greater-than d, which is only possible if $N \geq d$.

In the cases where $N < d$, the covariance matrix is not natively invertible and thus must be regularized before it can be inverted. Tikov regularization can be applied to the matrix to make it invertible, yielding the estimate

$$\hat{\mathbf{C}_{\mathbf{B}}}^{-1} = (\mathbf{C}_{\mathbf{B}} + \lambda\mathbf{I}_d)^{-1}, \tag{2.8}$$

where λ is the regularization scaling factor and $\mathbf{I}_d$ is a $d \times d$ identity matrix. If a covariance matrix is an identity matrix, then there is no correlation between the dimensions. By adding a portion of an identity matrix into the covariance matrix, we make it full rank (and therefore invertible), but also reduce the amount of correlation between dimensions, reducing the performance of our detector in cases where there is correlation between dimensions.

The SMF calculates the inner product between the demeaned and whitened pixel signature and a normalized, demeaned and whitened target signature. This inner product is used to measure the similarity between the pixel and the target. Equation (2.5) can be rearranged to show this relationship

$$D_{\mathrm{MF}}(\mathbf{x}) = \left\langle \mathbf{C}_{\mathbf{B}}^{-\frac{1}{2}}(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}}), \frac{\mathbf{C}_{\mathbf{B}}^{-\frac{1}{2}}(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})}{\left\|\mathbf{C}_{\mathbf{B}}^{-\frac{1}{2}}(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})\right\|} \right\rangle \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\mathrm{MF}}. \tag{2.9}$$

We use the fact that the SMF is an inner product in Section 2.6 to build a kernelized SMF.

2.3 Adaptive Cosine Estimator

The Adaptive Cosine Estimator (ACE) is a target detection algorithm that can be viewed as a non-linear scale-invariant extension of the SMF. the ACE detector has been shown to have impressive performance for hyperspectral target detection, as noted in ‘The Remarkable Success of Adaptive Cosine Estimator in Hyperspectral Target Detection’ [19], as well as in the less bombastically-named ‘Evaluating Subpixel Target Detection Algorithms in Hyperspectral Imagery’ [20].

The ACE detector is defined as follows

$$D_{\mathrm{ACE}}(\mathbf{x}) = \frac{(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}})^T\mathbf{C}_{\mathbf{B}}^{-1}(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})}{\sqrt{(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})^T\mathbf{C}_{\mathbf{B}}^{-1}(\mathbf{s}-\boldsymbol{\mu}_{\mathbf{B}})}\sqrt{(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}})^T\mathbf{C}_{\mathbf{B}}^{-1}(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}})}} \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\mathrm{ACE}}. \tag{2.10}$$

Equation (2.10) can be rewritten to highlight the relationship between the ACE and the SMF detectors

$$D_{\mathrm{ACE}}(\mathbf{x}) = \frac{D_{\mathrm{MF}}(\mathbf{x})}{\sqrt{(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}})^T \mathbf{C}_{\mathbf{B}}^{-1}(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{B}})}} \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\mathrm{ACE}}. \tag{2.11}$$

The only difference between the two is that the output of the ACE detector is scaled by the demeaned and whitened magnitude of $\mathbf{x}$. This regularizer makes the ACE detector insensitive to the magnitude of the target, making the output of the ACE detector completely dependent on the angle between the target signature and the pixel. This property allows the ACE algorithm to detect weak target signatures that the SMF may miss, and also avoid false positives from pixels that are only somewhat resemble the target signature $\mathbf{s}$, but have large enough magnitude to reach the SMF detection threshold [21].

This thesis does not investigate the performance of the ACE detector, but we do incorporate scale insensitivity into our detector through clustering. See Section 3.5.1 for implementation details. In Chapter 4 we show that incorporating scale-insensitivity into our clusters yields improved target detection performance.

2.4 RX Anomaly Detector

So far we have only considered the case where we are searching for a known target, but in some cases we want to find anomalies, pixels that do not look like the background. Target detectors search for pixels that deviate from the background in a particular direction, whereas anomaly detectors search for pixels that deviate from the background in any direction. Example hyperspectral applications for anomaly detectors include searching for artificial materials in a natural background (such as structures in a forest), or finding defects and imperfections on a product in an assembly line.

The most frequently used anomaly detector for hyperspectral imaging is the Reed-Xiaoli (RX) detector [22]. Like the SMF, the RX detector is also a CFAR detector that is derived from a GLRT [13]. For each pixel the RX detector is specified by

$$D_{\mathrm{RX}}(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_{\mathbf{B}})^T \mathbf{C}_{\mathbf{B}}^{-1} (\mathbf{x} - \boldsymbol{\mu}_{\mathbf{B}}) \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\mathrm{RX}}. \tag{2.12}$$

The output of the RX detector is simply a whitened distance measurement between a given pixel and the mean of the background. Thus, the further a pixel is away from the mean, the greater value the RX detector will output. One may note that (2.12) is simply the squared Mahalanobis distance (2.3) between $\boldsymbol{\mu}_{\mathbf{B}}$ and $\mathbf{x}$.

2.5 Windowing

Windowing is a method that allows us to incorporate the location of the pixel within the image when constructing our detector. Windowing involves constructing an outer sliding window region that contains the pixels used to calculate the sample mean and covariance when constructing the detector. In contrast with detectors that use global statistics, windowed detectors take the pixel location into account by only considering nearby pixels as part of the background. In Sections 3.6 and 3.5 we discuss alternative methods for incorporating pixel location into target detection via clustering.

While this thesis only considers point-target detection, when attempting to detect targets larger than a single pixel an additional inner window and guard band are used. Pixels within the inner window or guard band are excluded when calculating the statistics for the detector. The inner window and guard band are used to prevent nearby pixels, which may also contain the target, from being used to compute the background mean and covariance. To avoid needing to regularize the covariance matrix (as discussed in 2.2) a large enough window must be used. Figure 2.1 shows some example sliding windows designed for detecting a 7×7 target.

When using windowing a unique matched filter is constructed for each pixel based on the window surrounding that particular pixel. This generally makes windowing

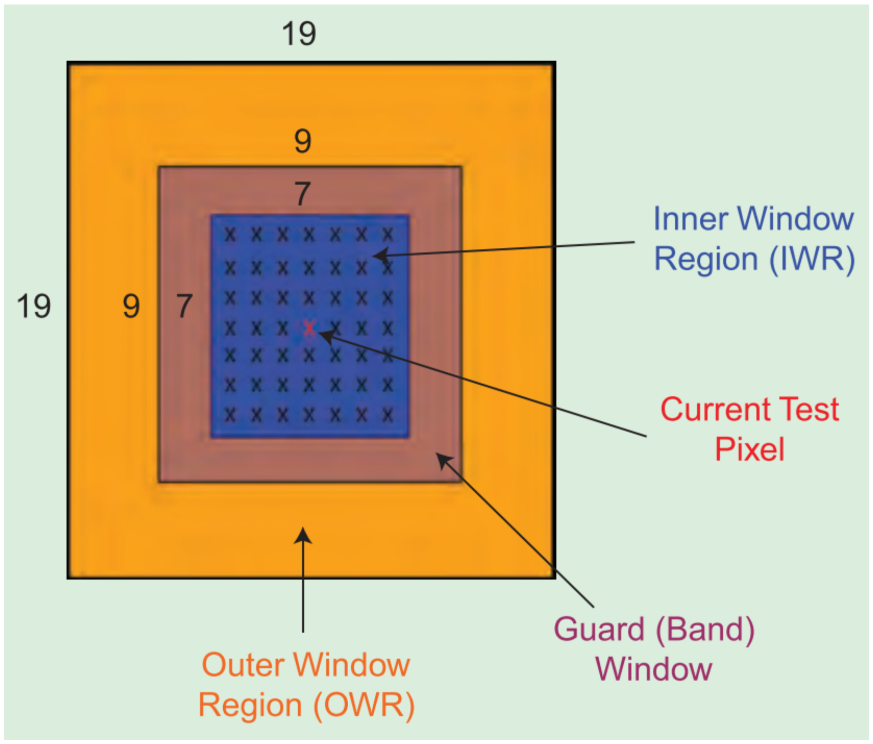

Figure 2.1: An example sliding window for non-point targets. The inner window prevents any nearby target-containing pixels from biasing the background mean and covariance calculations. Taken from [2].

more computationally intensive than when using the same statistics for the entire image. That said, the computation time needed to detect targets within and image when using windowing scales linearly with the number of pixels in the image. For algorithms that do not scale well to large images (such as the detectors introduced in the next section, which scale polynomially), windowing can offer a computational savings.

2.6 The Kernel Method

As shown in (2.9), the SMF detectors can be viewed as an inner product, and thus can utilize the kernel method, more commonly known as 'the kernel trick' [23].

The kernel method is a technique that allows one to convert a linear algorithm into a nonlinear algorithm by projecting the data into a high-(possibly infinite-)dimensional feature space. This conversion allows us to solve more complicated and nuanced problems in a tractable way. Through use of clever mathematics we can perform operations in a high-dimensional feature space without having to actually project the data into the high-dimensional space.

Suppose there exists a projection function $\phi(\mathbf{x})$ that maps $\mathbf{x}$ from a native d-dimensional input space into another, possibly higher-dimensional feature space of dimension p (i.e., $\phi(\,) : \mathbb{R}^d \to \mathbb{R}^p$ for some $p \in \mathbb{N}$). Further, suppose that there exists an easily-computable function $k(\mathbf{x}_1, \mathbf{x}_2)$ that outputs the inner product of the of the projected inputs $\mathbf{x}_1$ and $\mathbf{x}_2$, i.e.

$$k(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle \,. \tag{2.13}$$

A given algorithm $a(\mathbf{x}_1, \mathbf{x}_2)$ can be kernelized by substituting the projected values into the equation, $a(\phi(\mathbf{x}_1), \phi(\mathbf{x}_2))$. If all the projected values can be expressed as kernels (i.e. $a(\phi(\mathbf{x}_1), \phi(\mathbf{x}_2))$ can be expressed in terms of $k(\mathbf{x}_1, \mathbf{x}_2)$ with no remaining $\phi(\mathbf{x})$ terms), then the given algorithm can be kernelized.

Let us investigate the homogeneous quadratic kernel as a simple example. The homogeneous quadratic kernel is defined as

$$\begin{aligned} k(\mathbf{x}, \mathbf{s}) = (\mathbf{x}^T\mathbf{s})^2 = \left(\sum_{i=1}^{d} \mathbf{x}_i\mathbf{s}_i\right)^2 = \sum_{i=1}^{d}\sum_{j=1}^{d}(\mathbf{x}_i\mathbf{x}_j)(\mathbf{s}_i\mathbf{s}_j) \\ = \sum_{i=1}^{d} \mathbf{x}_i^2\mathbf{x}_j^2 + \sum_{i=2}^{d}\sum_{j=1}^{i-1}\left(\sqrt{2}\mathbf{x}_i\mathbf{x}_j\right)\left(\sqrt{2}\mathbf{s}_i\mathbf{s}_j\right) = \phi(\mathbf{x})^T \phi(\mathbf{s}) \,. \end{aligned} \tag{2.14}$$

Through inspection we can see that this kernel maps each input vector into a feature vector containing each original input value squared, as well as every possible 2-term product of the individual values. Thus, the feature map generated by this kernel is

$$\phi\left(\mathbf{x}\right)=\left\{\mathbf{x}_1^2,\ldots,\mathbf{x}_d^2,\sqrt{2}\mathbf{x}_2\mathbf{x}_1,\sqrt{2}\mathbf{x}_3\mathbf{x}_1,\sqrt{2}\mathbf{x}_3\mathbf{x}_2,\sqrt{2}\mathbf{x}_4\mathbf{x}_1,\ldots,\sqrt{2}\mathbf{x}_d\mathbf{x}_{d-1}\right\}. \qquad (2.15)$$

Using this kernel in an algorithm allows us to run calculations on vectors of size d, but get results as if those vectors were projected into a feature vector of size $\frac{1}{2}d(d-1)$. This example illustrates the power of the kernel method — it allows us to use simple algebraic manipulations (in the homogeneous quadratic case: replacing the value of each inner product in an algorithm with that of its square) to greatly increase the dimensionality of the solution space.

Another popular kernel to use in machine learning is the Radial Basis Function (RBF) kernel. It maps the input space into an infinite dimensional feature space. The RBF Kernel is defined as

$$k_{RBF}(\mathbf{x},\mathbf{s})=\exp\left(-\gamma\left\|\mathbf{x}-\mathbf{s}\right\|_2^2\right), \qquad (2.16)$$

where γ is a tunable parameter that determines the ‘spread’ of the kernel.

As shown in [24, 25], the feature map generated by this kernel is

$$\phi\left(\mathbf{x}\right)=\left(\exp\left(-\gamma\left\|\mathbf{x}\right\|_2^2\right)\frac{x_1^{n_1}\ldots x_d^{n_d}}{\sqrt{n_1!\ldots n_d!}}\right),\ \forall j\in\{0,1,\ldots,\infty\},\sum_{i=1}^{d}n_i=j. \qquad (2.17)$$

One interesting application of the RBF kernel is the use of the function as a similarity measurement. The function itself has several properties that make it well fitted for use measuring similarity in machine learning algorithms. In Section 3.5.1 we discuss some of these properties and use the function to create spectral clusters.

Choosing an appropriate projection and associated kernel function for a given problem is an important decision. While there are many kernels that perform well over a wide range of algorithms, often kernels are designed and tuned for a specific application to further improve performance. For example, [26] developed a kernel specifically for hyperspectral target detection and showed that it can provide a 5% improvement in target detection over more general kernels, such as the RBF and quadratic kernel described above.

2.7 Kernelized Detectors

Kernelized versions of both the SMF and the RX detectors have been developed. The inverse covariance matrix in these detectors complicate the use of the kernel method, but it still can be applied using kernelized principal component analysis [27]. Kernelized principal component analysis (also known as kernelized PCA) is a mathematical tool set that allows the use of size $N \times N$ gram matrices in place of $p \times p$ matrices in the projected dimension. The details of kernelized PCA are beyond the scope of this thesis, but the resulting kernelized detectors are described below.

The Kernel RX Algorithm, developed in [28], follows the decision rule

$$D_{\mathrm{KRX}}(\mathbf{x}) = \left(K_{\mathbf{x}}^T - K_{\boldsymbol{\mu}_{\mathbf{B}}}^T\right)^T K_{\mathbf{B}}^{-2} \left(K_{\mathbf{x}}^T - K_{\boldsymbol{\mu}_{\mathbf{B}}}^T\right)^T \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\mathrm{KRX}}, \tag{2.18}$$

where

$$K_{\mathbf{x}} = \left[f(\mathbf{x}_1, \mathbf{x}), \ldots, f(\mathbf{x}_N, \mathbf{x})\right]^T, \quad f(\mathbf{x}_i, \mathbf{x}) = k(\mathbf{x}_i, \mathbf{x}) - \frac{1}{N}\sum_{j=1}^{N} k(\mathbf{x}_j, \mathbf{x}), \tag{2.19}$$

$$K_{\boldsymbol{\mu}_{\mathbf{B}}} = \left[g(\mathbf{x}_1), \ldots, g(\mathbf{x}_N)\right]^T, \quad g(\mathbf{x}_i) = \frac{1}{N}\sum_{j=1}^{N} k(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{N^2}\sum_{j=1}^{N}\sum_{k=1}^{N} k(\mathbf{x}_j, \mathbf{x}_k), \tag{2.20}$$

and $K_{\mathbf{B}}$ is a centered $N \times N$-dimensional gram matrix.

The Kernel SMF [11] follows the decision rule:

$$D_{\text{KSMF}}(\mathbf{x}) = \frac{K(\mathbf{X},\mathbf{x})^T K_{\mathbf{B}}^{-2} K(\mathbf{X},\mathbf{s})}{K(\mathbf{X},\mathbf{s})^T K_{\mathbf{B}}^{-2} K(\mathbf{X},\mathbf{s})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta_{\text{KSMF}}, \tag{2.21}$$

where

$$K(\mathbf{X},\mathbf{s})^T = \left[k(\mathbf{x}_1,\mathbf{s}),\ldots,k(\mathbf{x}_N,\mathbf{s})\right]^T. \tag{2.22}$$

A major disadvantage to the kernelized detectors is their computational inefficiency. The kernelized detectors require constructing, inverting, and squaring an $N \times N$ gram matrix. In contrast, the non-kernelized equivalents only require a constructing and inverting a $d \times d$ covariance matrix. One way to alleviate this computational expense is to lower the value of N, i.e., limit the number of pixels that compose the background, either through windowing (see Section 2.5) or through clustering (see Chapter 3).

2.8 Detector Performance Analysis

As discussed in Section 2.1, a detector is just an algorithm that decides between two potential hypothesis for each pixel. The performance of an algorithm for a given detection problem can be summarized with 4 statistics:

1. True Positives (TP) – the number of target pixels that were correctly identified as target pixels
2. True Negatives (TN) – the number of background pixels that were correctly identified as background pixels
3. False Positives (FP) – the number of background pixels that were incorrectly identified as target pixels
4. False Negatives (FN) – the number of target pixels that were incorrectly identified as background pixels.

Calculating these statistics requires a priori knowledge of which pixels actually contain a target and which do not. An ideal target detector would correctly identify all of the pixels, and thus have $FP = FN = 0$.

Two secondary statistics are commonly used to evaluate a detectors performance, the True Positive Rate (TPR) and the False Positive Rate (FPR), which are defined as

$$TPR = \frac{TP}{TP + FN}, \tag{2.23}$$

and

$$FPR = \frac{FP}{TN + FP}. \tag{2.24}$$

An ideal target detector would have a TPR of 1 and a FPR of 0. In practice, however, the TPR and FPR are both highly dependent on the sensitivity of the detector. The sensitivity of a detector can be adjusted by changing the detection threshold η. A low detection threshold will identify more targets, whereas a high detection threshold will identify fewer. When the detection threshold is set low enough, all pixels in the image will be identified as targets, meaning the detector will have a TPR and FPR of one. Similarly, when the detection threshold is set high enough, none of the pixels in the image will be identified as targets, meaning the detector will have a TPR and FPR of zero.

The threshold η is generally chosen based on what the target detector is being used for. Depending on the application it may be desirable to have a more or less sensitive detector. As such, when evaluating and comparing the performance of target detection algorithms it is important to look at a wide range of detection thresholds.

2.8.1 Detection Histograms

All the target detectors we have discussed output a value corresponding to the likelihood that a given pixel contains a target. Under normal use this value is compared with the threshold to decide between the two hypothesis, but for performance evaluation it is often beneficial to ignore the threshold and work with the output of the detector directly.

One common method to evaluate, visualize, and tune the performance of a detector is to generate two histograms from the detector output values: one for the values from the pixels that contain a target and one for the values from the pixels with no target [29]. Figure 2.2 contains three such histograms. In such histograms the horizontal can be viewed as a range of possible detection thresholds. If a vertical line were placed on the graph at a given detection threshold, the values to the left of the line correspond to the pixels classified as H_0 (no target present) by the detector, whereas the values to the right correspond to the pixels classified as H_1 (containing a target). Figure 2.3 shows how the four detection statistics can be derived from the histogram. In order for a detector to have perfect performance it must be able to correctly classify every pixel without error. To do so, there must exist a threshold value that perfectly divides the two distributions. Overlap between the two histograms indicates that there are pixels that the detector will be unable to classify, regardless of the threshold chosen.

2.8.2 Receiver Operating Characteristic (ROC) Curves

Receiver Operating Characteristic (ROC) curves are another way to visualize the performance of detectors. They are generated by sweeping over the full range of threshold values and plotting the TPR versus the FPR at each step. Figure 2.3 shows how an ROC curve can be generated from target detection histograms.

All ROC curves go between $(0,0)$ and $(1,1)$, regardless of the algorithm, target signature, or image used. The ROC curve of an ideal detector is a step function, indicating that all the true targets will be detected before any false targets. The worst ROC curve is a line of slope one, indicating that the detector is just as likely to detect false targets as it is to detect true targets. The worst ROC is generated if the two histograms are completely overlapping, indicating that the detector cannot distinguish between the two hypothesis.

Note that while all ROC curves go between the same values, the curve can vary based on algorithm, target signature, and image. In order to compare the performance of multiple algorithms, the target signature and image must be identical.

2.8.3 Area Under Curve (AUC) Metrics

While ROC curves are an excellent tool for comparing target detection performance, sometimes it is useful to summarize the performance with a single number. A simple but effective metric is the Area Under Curve (AUC) value, which simply measures the area underneath the ROC curve. While it does not provide as much nuance as an ROC curve, it does give us a general idea how well an algorithm is performing. AUC can be calculated with a single integral

$$AUC = \int_0^1 TPR(FPR) \partial FPR, \tag{2.25}$$

where FPR is the False Positive Rate and $TPR(FPR)$ is the True Positive Rate for a given False Positive Rate. AUC values range from 0.5 to 1, with 0.5 being the performance of the worst possible detector (corresponding to the area under a line with slope 1 in the range $(0, 1)$) and 1 being the performance of an ideal detector (corresponding to the area under a step function over the same range).

In many target detection problems it is desirable to have a low false positive rate. For example, if we are attempting to find a potential target to destroy, we want to have a high confidence that the target is genuine and not a false positive. One property of the AUC metric is that it incorporates the performance over the entire range of possible FPR values, while this is good in the general case, it makes it a poor rubric for identifying algorithms that perform well at low FPR values. In such cases it makes sense to use a partial AUC that only sweeps over the range of low FPR values, i.e.

$$pAUC(\theta) = \frac{1}{\theta}\int_0^\theta TPR(FPR)\partial FPR, \tag{2.26}$$

where $pAUC(\theta)$ is the partial AUC and θ is the largest FPR you wish to consider.

AUC metrics, being derived from ROC curves, are specific to a given algorithm on a specific target within a specific image. When comparing the AUC of multiple algorithms, the target signature and image must be identical.

2.9 Target Detection Summary

In this section we investigated several target detection algorithms and looked at a few methods to analyze and compare their performance. The remainder of this thesis almost exclusively focuses on the target detection performance of the SMF target detector (discussed in Section 2.2) on clustered hyperspectral images. In chapter 3 we investigate a variety of clustering algorithms, and in chapter 4 we compare the target detection performance of the SMF on the clusters generated by those algorithms using the metrics we discussed in Section 2.8. The focus of our work is on how to improve the performance of the SMF using clustering, and while we do not investigate the target detection performance of the other algorithms discussed in this chapter, we do discuss how to use clustering to gain some of same the benefits and features as the more complicated target detection algorithms.

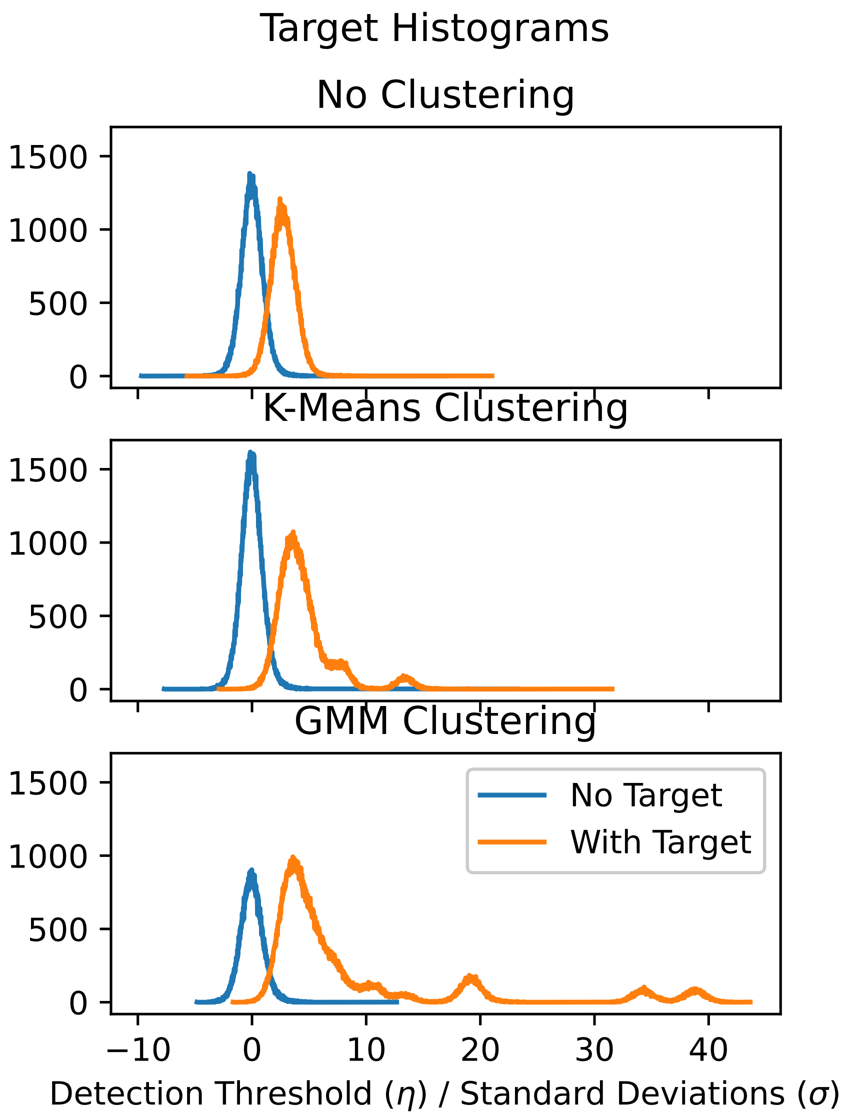

Figure 2.2: Histograms of target detection with and without target embedding with no Clustering, K-Means Clustering, and Gaussian Mixture Model (GMM) Clustering. Clustering can be used to further separate pixels in the two histograms, providing improved target detection performance.

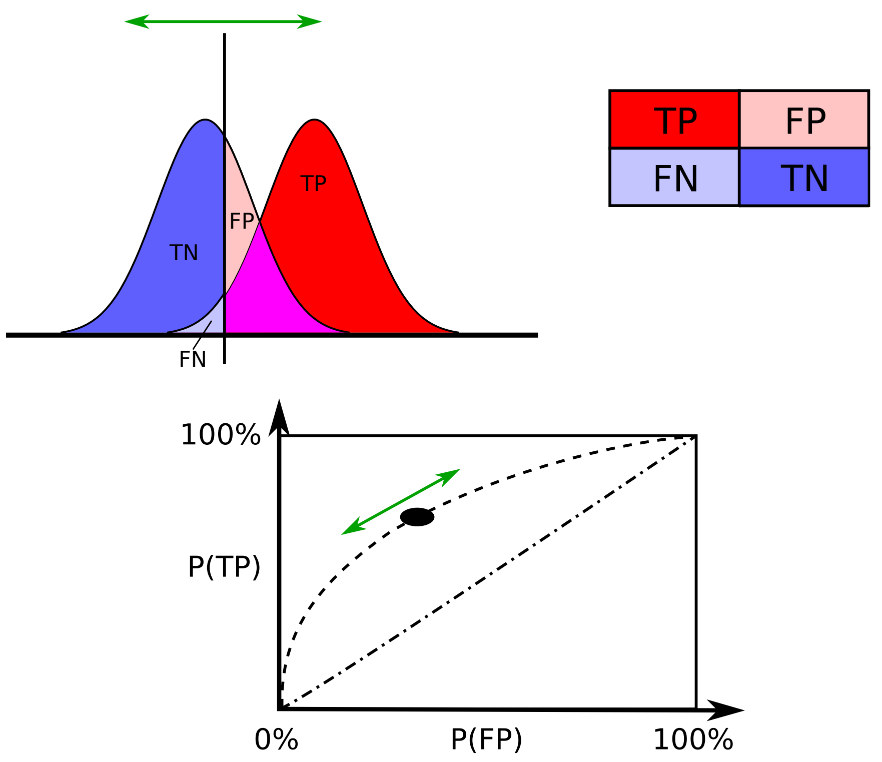

Figure 2.3: Constructing a ROC curve from Target Detection Histograms, from [3]. An ROC curve shows the performance of the detector at all possible threshold values.

Chapter 3

Data Clustering Algorithms

3.1 Clustering for Target Detection

Data clustering is the process of grouping together similar data points into disjoint collections known as clusters. Clustering can be used to group together similar data points to help make generalizations about portions of the data. Clustering can also be used to break apart a complicated problem into a collection of simpler problems. In our case of clustering hyperspectral data, the data points are the d-dimensional pixel vectors, and the task we are simplifying is target detection.

Figure 3.1 shows how a SMF is used to detect targets in a hyperspectral image. The SMF uses the hyperspectral image and the target signature to determine the likelihood that each pixel contains a target. The likelihood of every pixel is placed into a target strength map and passed through a thresholding function that decides between the two possible hypothesis for each pixel. The output of the thresholding function is a map indicating the pixels that contain the target (H_1) and those that do not (H_0).

Figure 3.2 shows how clustering can be used with multiple SMFs to solve the target detection problem. Instead of using a single SMF as a target detector for all the pixels in the image, we now subdivide the image into clusters and use a different SMF to perform target detection on each cluster with the hope of improving the overall detection performance. The output of each SMF is aggregated together to create a target strength map. From there the process is identical to the process shown in

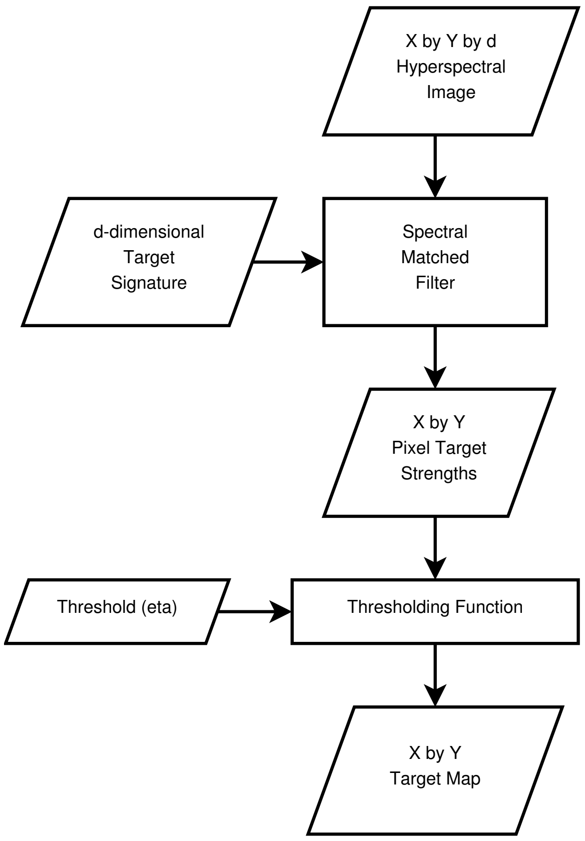

Figure 3.1: Data Flow for Hyperspectral Target Detection without Clustering. The SMF is used to construct a target map.

Figure 3.1. The target strength map is passed through a thresholding function that decides between the two possible hypothesis, and outputs a map indicating which pixels contain the target and which do not. While the inner workings of the clustered target detection algorithm are significantly more complex, it is important to note that the inputs and outputs of this process are the same as the detector in Figure 3.1. In fact, the detector in Figure 3.1 can be seen as a special case of the detector in Figure 3.2 where the number of clusters is one.

Target detection algorithms work by finding pixels that stand out from the background of the image in the same way that a target would. As discussed in Section 2.1, in order to find the pixels that stand out, a target detection algorithm must first accurately model the background. Clustering allows us to more accurately model the background, giving us better performance. When a single SMF is used to detect targets on an entire hyperspectral image it does not perform well. This is because the SMF models the entire background as a single Gaussian distribution, which is not accurate. In contrast, the use multiple SMFs when using clustering makes the background model of our detector effectively a mixture of Gaussians, with each SMF modeling a single Gaussian. This improves the accuracy of the background model, thereby improving detection performance.

As shown in Figure 3.3, different clustering algorithms find different relationships in the data and can lead to different clusterings. In Chapter 4 we will see that the algorithms that perform well for target detection tend to be the ones that divide the pixels based on the type of ground cover within each pixel, i.e., grass, water, pavement, forests, roofing, etc. When an image is clustered in this way, each individual SMF only has to detect targets in pixels containing a specific type of ground cover.

Our investigation shows that clustering hyperspectral data leads to a significant improvement in detection performance. Previous investigations into clustering came to a variety of conclusions. Funk et al. [17] only investigated clustering via k-means but found that it offered improved detection performance when applied to highly correlated data. For future work, Funk et al. suggested using clusterings that incorporate second-order statistical information to further improve performance, which we investigate in Section 3.3. Pieper et al. [30] concluded that the added complexity of clustering did not outweigh the gains in detection performance. This is in stark contrast to the conclusions we draw in Chapter 5. The differences are likely due to their use of a windowed target detector, which we discuss in Section 2.5.

In this thesis, we consider four popular clustering algorithms:

1. k-Means

2. Subspace Clustering

3. Spectral Clustering

4. Gaussian Mixture Model (GMM),

as well as a variation on GMM known as Laplacian-Regularized GMM (LapGMM). All of these algorithms are described later in this chapter. In Chapter 4 we will investigate the performance of the clustering algorithms when used in the clustered target detector and fine tune the LapGMM algorithm specifically for hyperspectral target detection.

3.2 k-means Clustering

The k-means algorithm is a straightforward algorithm for data clustering. The k-means algorithm treats clustering as an optimization problem and attempts to find a clustering within the data that minimizes a given objective function [4]. k-means operates by attempting to minimize the k-means objective function, defined as

$$G_{k\text{-means}}(\mathcal{X}, \mathbf{dist}, (\mathcal{C}_1, \ldots, \mathcal{C}_k)) = \min_{\mu_1,\ldots,\mu_k \in \mathbb{R}^d} \sum_{i=1}^{k} \sum_{x \in \mathcal{C}_i} \mathbf{dist}(x, \mu_i)^2, \tag{3.1}$$

where $\mathcal{X}$ is the data set, $\mathcal{C}_i$ are the clusters, $\mathbf{dist}()$ is a distance function (typically Euclidian), and μ_i are the centers of the clusters $\mathcal{C}_i$, also known as 'centroids'.

In other words, k-means seeks to find a set of k clusters that minimize the total squared distance between the points in the data set and the centroids of the clusters they are placed in.

Unfortunately, task of finding the clustering that globally minimizes (3.1) is an NP-hard problem, making it unfeasible to calculate in most cases. That said, an approximate solution to (3.1) can be computed fairly efficiently. The algorithm used to heuristically optimize the k-means objective function, known as the k-means algorithm, is an iterative algorithm with two steps described in Algorithm 1.

The first step in the algorithm is to construct clusters based on the distance between the data and the current centroids. The second step is to recalculate the centroids

Algorithm 1: k-means [4]

```
Input      : $\mathbf{X} \subset \mathbb{R}^d$, number of clusters $k$
Initalize  : Randomly choose initial centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
Until convergence:
    $\forall i \in [k]$ set $C_i = \left\{\mathbf{x} \in \mathbf{X} : i = \arg\min_j \left\|\mathbf{x} - \boldsymbol{\mu}_j\right\|_2\right\}$
    (break ties in some arbitrary manner)
    $\forall i \in [k]$ update $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$
Repeat
Output     : Clusters $C_1, \ldots, C_k$
```

based data contained within the current clusters. These steps are repeated until the algorithm has converged, i.e., subsequent iterations do not change the elements within each cluster.

In [4] it is proven that the clusters found by Algorithm 1 are monotonically-non-increasing with respect to the loss function (3.1), meaning that the objective will never get worse with subsequent iterations. That said, there is no guarantee that this algorithm finds the clustering corresponding to the global minimum of (3.1), and may converge to a local minimum instead. The minimum that the algorithm converges to is largely dependent on the initial placement of the centroids, and while the original k-means algorithm used a random centroid initialization, other centroid initialization methods, such as $k$means++ described in [31], have been shown to provide better theoretical convergence guarantees. We took advantage of the theoretical advantages of $k$means++ initialization, and all of the k-means results shown in Chapter 4 were initialized with $k$means++.

k-means effectively divides the solution space into a set of Voronoi partitions. While this partitioning is capable of clustering many data sets successfully, it has its limitations. Figure 3.3 shows that while k-means is very capable of clustering simple sets of data, it struggles to cluster data sets that have different distributions for each cluster or for each dimension. For example, the second row of the Figure shows that k-means struggles with data sets where the variance of the clusters differs by dimension. k-means also has difficulty clustering the data set in the third row, where there is a different amount of variance in each underlying cluster. In the next section we will introduce Gaussian mixture model (GMM) clustering, a method that has no difficulty clustering the data in the cases described above. GMM is very similar to k-means but utilizes a more sophisticated model that offers more flexibility in terms of the shapes of clusters. k-means can be viewed as a limited, or 'hard' thresholded Gaussian mixture model. We will explain this relationship in Section 3.3.

3.3 Gaussian Mixture Models (GMM)

Gaussian Mixture Model (GMM) clustering is a clustering algorithm similar to k-means. k-means searches for a collection of k centroids, whereas GMM searches for a mixture of k Gaussian distributions. The use of distributions instead of centroids offers much more flexibility in terms of the shapes of data GMM is capable of clustering. First a mixture of k Gaussian distributions is found that matches the data, then the data points are divided into clusters based on the Gaussian distribution they most likely came from.

GMM clusterings aim to maximize the log-likelihood function

$$\mathcal{L}(\Theta) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{K} \alpha_j p_j(\mathbf{x}_i \mid \Theta_j) \right), \tag{3.2}$$

where Θ_j contains the mean $\boldsymbol{\mu}_j$ and covariance Σ_j of the jth Gaussian distribution in the mixture. The heuristic algorithm used to find the GMM clustering is quite similar to the Algorithm 1 used to optimize k-means. The model it searches for is more complicated, and thus the computation is more complicated, but the process is effectively the same. It is an Expectation-Maximization (EM) algorithm that seeks to maximize the log-likelihood equation above.

Expectation-Maximization algorithms operate by alternating between two steps.

Algorithm 2: Gaussian Mixture Model (GMM) [5]

```
Input      : $\mathbf{X} \subset \mathbb{R}^d$, number of clusters $K$, termination condition value $\delta$
Initalize  : $\Theta^0$ by using $k$-means, $t = 1$
Until convergence ($|\mathcal{L}(\Theta^t) - \mathcal{L}(\Theta^{t-1})| \leq \delta$):
    t = t + 1
    E-step:
    Compute posterior probabilities:
    $\mathbb{P}(k \mid \mathbf{x}_i, \Theta^{t-1}) = \frac{\alpha_k^{t-1} p_k(\mathbf{x}_i \mid \Theta_k^{t-1})}{\sum_{j=1}^K \alpha_j^{t-1} p_j(\mathbf{x}_i \mid \Theta_j^{t-1})}$, $k = 1, \ldots, n$
    M-step:
    Compute the GMM estimates $\alpha_i^t$, $\boldsymbol{\mu}_i^t$, and $\Sigma_i^t$:
    $\alpha_i^t = \frac{1}{n} \sum_{j=1}^n \mathbb{P}(i \mid \mathbf{x}_j)$,
    $\boldsymbol{\mu}_i^t = \frac{\sum_{j=1}^n \mathbf{x}_j \, \mathbb{P}(i \mid \mathbf{x}_j)}{\sum_{j=1}^n \mathbb{P}(i \mid \mathbf{x}_j)}$,
    $\Sigma_i^t = \frac{\sum_{j=1}^n \mathbb{P}(i \mid \mathbf{x}_j)(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^n \mathbb{P}(i \mid \mathbf{x}_j)}$.
    Evaluate the regularized log likelihood:
    $\mathcal{L}(\Theta^t) = \sum_{i=1}^n \log\left(\sum_{j=1}^K \alpha_j p_j(\mathbf{x}_i \mid \Theta_j)\right)$.
Repeat
Output : $\Theta^t$
```

In the expectation step (or E-step), the probability (or 'expectation') of each data point belonging to each Gaussian distribution is calculated. In the maximization step (or M-step), the Gaussian distributions are then updated based on the data points they are expected to contain. The GMM algorithm is guaranteed to be monotonically non-decreasing in the objective function, which means subsequent iterations will never produce clusters with lower log-likelihood values. Unfortunately, the GMM algorithm suffers the same local optimality problem as the k-means algorithm, and while it can easily find a local optimum, there is no guarantee that the optimum is global.

Although implemented differently, conceptually k-means can be viewed as a 'hard' thresholded Gaussian mixture model. If the covariance matrices of each cluster are forced to $\mathbf{0}$, then the k-means objective function (3.1) and the GMM objective function (3.2) are equivalent.

Out of all the purely model-based clusterings we tried using in our clustered target detector, GMM performed the best. GMM is also the basis for the algorithm that yielded the best performance in our clustered target detector, discussed in Section 3.6. We believe that this is partially due to its ability to accurately model the image, but primarily because it pairs well with our chosen target detection algorithm. The SMF is the optimal target detection algorithm for Gaussian-distributed data in terms of SNR, and the GMM algorithm finds Gaussian distributed clusters. Intuitively it makes sense then that our clustered target detector would perform well when given Gaussian clusters.

In the next section we discuss another model-based clustering algorithm. While the next algorithm was not able to provide the same level of performance with our clustered target detector as GMM, we believe that it is worth further investigation, and discuss possible future work in Section 5.2.

3.4 Subspace Clustering

Subspace clustering is another model-based clustering method. Instead of modeling the data as a collection of centroids or a mixture of Gaussian distributions, it models the data as a collection of low-dimensional subspaces within a high-dimensional input space. When compared to other algorithms, subspace clustering has been shown to perform well on high-dimensional data [32].

Figure 3.3 shows subspace clustering algorithm being used to cluster a variety of two dimensional synthetic data sets. The algorithm does not cluster any of these sets well. This is because subspace clustering was designed specifically to work with high-dimensional data, and as such, struggles to cluster low dimensional data. *k*-means models each cluster as a two dimensional point and GMM models each cluster as a two dimensional Gaussian distribution. In contrast, on two dimensional data subspace clustering is limited to modeling each cluster as a line through the origin. Higher dimensions inherently contain more subspaces, giving more flexibility to the clustering algorithm, but in low dimensions the utility of subspace clustering is admittedly limited.

Subspace clustering is an active research topic, and many recent algorithms have been published for finding subspace clusterings. We chose to test two algorithms, the Sparse Subspace Clustering by Orthogonal Matching Pursuit (SSC-OMP) algorithm [33] and the Elastic Net Subspace Clustering (EnSC) algorithm [34]. Since hyperspectral images are naturally high-dimensional, we believed that subspace clustering would would give us good performance when used in our clustered target detector. The clusters generated by EnSC provided target detection performance comparable to the k-means, while the clusters generated by SSC-OMP provided the worst target detection performance of all the clustering algorithms we tested. See Chapter 4 for details. Although the performance from using either of these algorithms was much less than when using GMM, we still believe that the use of subspace clustering in hyperspectral target detection is worth further investigation, as we discuss in Section 5.2.

3.5 Spectral Clustering

All of the algorithms we have discussed thus far cluster the data by constructing a simplified model based on the values of the data points. k-means constructs a model of centroids, GMM constructs is a model of Gaussian distributions, and subspace clustering constructs a model of subspaces. In stark contrast, spectral clustering does not consider the location of the data, and constructs no equivalent model. Instead, spectral clustering finds clusters based on similarities between the data points. One inherent advantage to spectral clustering is that it makes no assumptions about the shape of the cluster, which allows it to find useful clusters in cases where other algorithms may have difficulty. Figure 3.3 shows that spectral clustering is capable of correctly clustering concentric circles and half moon-shaped datasets, a task where all the other algorithms have difficulty.

Spectral clustering attempts to minimize the following equation [4]

$$G_{\text{Spectral}}(C_1, \ldots, C_k) = \sum_{i-1}^{k} \frac{1}{|C_i|} \sum_{r \in C_i, s \notin C_i} W_{r,s}, \tag{3.3}$$

where $W \in \mathbb{R}^{N \times N}$ is an affinity matrix. An affinity matrix, also known as a similarity matrix, is a matrix that contains similarity measurements between all the points within the set, e.g. $W_{i,j}$ contains the similarity measurement between the ith and jth data points. The spectral objective function is simply a scaled sum of the similarity measurements between every data point and data points in other clusters. Minimizing (3.3) maximizes the similarity between points belonging to the same cluster. The $\frac{1}{|C_i|}$ term in the equation can be seen as a cluster size regularizer, preventing the number of points any cluster from becoming to small. Without this term (3.3) can be minimized simply by placing a majority of the data points into a single cluster.

Algorithm 3 attempts to minimize (3.3). It operates by constructing and eigendecomposing a Laplacian matrix L, constructing a set of vectors $\mathbf{V}$ from the eigendecomposition, then using the k-means algorithm (described in 3.2) to cluster the vectors. The resulting vectors clusters from k-means correspond to the spectral clusters of the input data. Unfortunately, because this algorithm is dependent on the k-means algorithm no guarantees about global optimality can be made. A proof, available in [4], shows that if we could find an optimal k-means clustering of the eigen-derived vectors, then this algorithm is optimal. Additionally, it has been proven that under certain conditions the Spectral Clustering algorithm can be used to find an optimal Gaussian Mixture Model [35].

Algorithm 3 requires the computation of a graph Laplacian. Graph Laplacians, like affinity matrices, are simply matrix representations of similarities between points in a data set. A graph Laplacian can be generated from an affinity matrix by subtracting

Algorithm 3: Spectral Clustering [4]

```
Input     : $W \in \mathbb{R}^{n \times n}$, number of clusters $k$
Initalize : Compute the graph Laplacian $L$ from $W$
Let $U \in \mathbb{R}^{n,k}$ be the matrix whose columns are the eigenvectors of $L$
  corresponding to the $k$ smallest eigenvalues
Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the rows of $U$
Cluster the points $\mathbf{v}_1, \ldots, \mathbf{v}_n$ using $k$-means
Output    : Clusters $C_1, \ldots, C_k$ from the $k$-means algorithm
```

the affinity matrix from a diagonal matrix consisting of the sum of each row (or column) of itself, or more compactly

$$L = \mathbf{diag}(W1_N) - W, \tag{3.4}$$

where 1_N is a vector of all 1s and $\mathbf{diag}(x)$ is a function that returns a matrix with diagonal elements equal to x. Like affinity matrices, the nth row (or column) of the Laplacian contains information about the nth element in the data set. While graph Laplacians contains the same information as affinity matrices, the information is stored in a different way that allows the spectral relationships of the data to be discoverable through eigendecomposition. Note that if the data set contains perfectly separateable clusters, then the eigenvectors will be piecewise consistent vectors indicating the members of that cluster and the objective function shown in Equation (3.3) will equal zero.

3.5.1 Similarity Measurements

In order to implement spectral clustering, we must compute the graph Laplacian L. In order to compute L, we must construct an affinity matrix W to measure the 'similarity' between points. In order to construct W, we must define a measurement of 'similarity'. Defining a 'similarity' measurement for hyperspectral pixels is not a straightforward task. For example, all of the target detectors in Chapter 2 attempt to find pixels that are 'similar' to the target, but do so using completely different notions of similarity. In the following subsections we cover several common metrics of similarity and discuss how they can be used to improve hyperspectral target detection. In Chapter 4 we compare the performance of various similarity measurements when used for hyperspectral target detection.

Euclidian-based Similarities

Euclidian distance is a common way to measure the distance between two data points, and it also can be used to measure similarity between two points. Euclidian distance is defined as

$$\mathbf{dist}_{EUC}(\mathbf{x}, \mathbf{s}) = \|\mathbf{x}, \mathbf{s}\|_2 = \sqrt{\sum_{i=1}^{d} (\mathbf{x}_i - \mathbf{s}_i)^2}, \tag{3.5}$$

where $\mathbf{x}, \mathbf{s} \in \mathbb{R}^d$.

Distance measurements grow larger as the inputs become less similar. In contrast, similarity measurements should grow larger as the inputs become more similar. A simple way to convert a distance measurement into a similarity measurement is to subtract the distance from a constant value, i.e.,

$$w_{EUC}(\mathbf{x}_1, \mathbf{x}_2) = \max_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}} \mathbf{dist}_{EUC}(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{dist}_{EUC}(\mathbf{x}_1, \mathbf{x}_2), \tag{3.6}$$

but other methods of conversion exist, such as the RBF kernel.

The RBF kernel, described in Section 2.6, calculates the Euclidean distance between the points, and uses it to generate a value that grows larger as the inputs vectors become more similar. Unlike the similarity measurement in (3.6), the RBF kernel has a term γ that can be used to tune the sensitivity of the algorithm.

We investigated the use of both (2.16) and (3.6) as a similarity measurement in our tuned clustered target detector, but neither resulted in significant performance improvements. This leads us to believe that Euclidean similarity measurements are not particularly useful for clustering hyperspectral images. As discussed in 2.1, the variance of hyperspectral images can be quite different from channel to channel. This causes some channels to have a larger influence on Euclidean distance than others. With Euclidean distance, strong similarities in a channel with low variance could be masked by other channels with larger variance. While the Euclidean distance measurement suffers from this weakness, the Mahalanobis distance, defined in (2.3), accounts for the variance, giving each channel equal influence on the measurement, regardless of scale. While our tuned clustering algorithm did not incorporate Euclidean distances, it did indirectly use the Mahalanobis distance through use of the GMM algorithm. See Chapter 4 for details.

Cosine Similarity

Cosine similarity is a scale-insensitive similarity metric that compares the spectral angle between two data points in the set. Scale insensitivity can help improve the identification of certain types of ground cover in aerial hyperspectral images. Specifically, it makes our clusters less sensitive to shade and shadows. Most of the light captured by aerial images is reflected sunlight, so pixels capturing shaded regions will have a much lower magnitude than those that capture regions in direct sunlight. Pixels that contain ground coverings with a lot of natural shade (such as forests) tend to vary widely in magnitude. This is because some pixels will capture the shaded regions while others will capture the regions in direct sunlight (providing aforementioned shade). While the amount of light reflected in these regions varies widely from pixel to pixel, the specific wavelengths and proportions of light that are reflected tends to stay the same. The cosine similarity measurement, being scale-insensitive, completely ignores large differences in magnitude such as this and identifies the shaded and unshaded pixels as similar.

Cosine similarity is defined as

$$w_{COS}(\mathbf{x}_1, \mathbf{x}_2) = \cos\left(\langle \mathbf{x}_1, \mathbf{x}_2 \rangle\right) = \frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\|_2 \, \|\mathbf{x}_2\|_2}, \tag{3.7}$$

where $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ is the spectral angle between the two pixels.

In Section 2.3 we looked at the ACE target detector and its scale insensitive properties. We can use the cosine similarity measurement to add a similar amount of scale insensitivity through clustering. The SMFs used in our clustered target detector are inherently a scale-sensitive but also adapt to the scale of the data they are given. If the SMFs construct their background model based on pixels that vary widely in magnitude, then the resulting detector is less likely to misidentify similarly-angled pixels as targets because of magnitude.

In [36] a scale-insensitive filter is used to improve multispectral image clustering. In Chapter 4 we incorporate cosine similarity into the LapGMM algorithm to improve the performance of our clustered target detector.

Location Similarity

Until now we have only considered the value contained within the pixel when clustering. We have not considered the location of the pixel within the image. Location similarity is a measurement of how close together two pixels are within the image. Clusters generated by location similarity only consider the location of the pixel, completely ignoring the data within the pixel, and divide the image into Voronoi cells. We define location similarity as the Euclidean similarity of the pixel locations, or more formally as

$$w_{loc}(\mathbf{x}_1, \mathbf{x}_2) = \max_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}} \left\|\mathcal{L}\left(\mathbf{x}_i\right) - \mathcal{L}\left(\mathbf{x}_j\right)\right\|_2 - \left\|\mathcal{L}\left(\mathbf{x}_1\right) - \mathcal{L}\left(\mathbf{x}_2\right)\right\|_2, \tag{3.8}$$

where $\mathcal{L}\left(\mathbf{x}\right)$ is a function that returns the horizontal and vertical coordinates of pixel $\mathbf{x}$ within the hyperspectral image.

Spectral Clusters generated from location similarity alone are have very little utility by themselves, as far simpler techniques can be used to find groups of similarly located pixels, such as windowing (which is discussed in Section 2.5). That said, the location similarity measurement was an important part of our tuned clustering algorithm. While the location similarity measurement is of little use by itself, it can be very useful when combined with other similarity measurements. In the next section we will discuss how to generate spectral clusters using multiple similarity measurements, allowing the pixel location to influence (but not dominate) the resulting clusters.

3.5.2 Blending Similarity Measurements

Each of the similarity measurements discussed in the previous sections have their own unique properties, and when used to construct an affinity matrix for spectral clustering, will yield entirely different clusters. Using Euclidean similarity will produce clusters of pixels that have similar values. Using cosine similarity will produce clusters of pixels with similar spectral angle. Using location similarity will produce clusters of pixels that are located close to one another.

While each of these clusterings can be useful by themselves, oftentimes we desire a clustering that has many of these properties instead of just one. For example, in our application we desire a clustering that can improve the performance of our target detector. We believe that such a clustering would cluster the image based on ground cover. As such, it would be nice if this clustering had some scale insensitivity, so it doesn't mistake shadows for different kinds of ground cover. It would also be nice if the clustering considered the location of the pixel, as ground cover generally covers a contiguous region of pixels. Cosine similarity gives us scale insensitivity, but doesn't consider the location. Location similarity only considers the location of the pixel, but ignores the pixel contents. Neither of these similarity measurements will generate the clustering we desire, but if there were a way to combine them together, a clustering that has both of these properties could be produced.

Fortunately there is a very straightforward method for constructing such a similarity measurement. Equation (3.9) shows how two similarity measurements can be combined to construct a new blended similarity measurement. A blended similarity measurement can be formed using a convex combination of two similarities

$$w_C(\mathbf{x}_1, \mathbf{x}_2) = \alpha w_A(\mathbf{x}_1, \mathbf{x}_2) + (1 - \alpha) w_B(\mathbf{x}_1, \mathbf{x}_2), \tag{3.9}$$

where $\alpha \in [0, 1]$ determines how much each similarity measurement influences the combined value. Performing spectral clustering with an affinity matrix that was constructed using the similarity measurement in (3.9) will produce clusters of data that were similar according to either w_A or w_B (or according to both). Equivalently, the same combined affinity matrix can be constructed by combining two affinity matrices, i.e.,

$$W_C = \alpha W_A + (1 - \alpha) W_B, \tag{3.10}$$

where W_A and W_B are the affinity matrices constructed with w_A and w_B, respectively. It is fairly straightforward to see how this method could be repeated to combine any number of similarity measurements.

The α term highlights the major trade-off faced when using this blending method. While it give us the ability to generate clusters that incorporate multiple desirable properties, the resulting clusters are not the ‘best of both worlds,’ but rather a compromise between the two. Combining multiple similarity measurements effectively tasks the spectral clustering algorithm with multiple objectives to optimize for, and the resulting clusters are effectively ‘jacks-of-all-trades,’ partially optimizing each objective, but effectively optimizing few. This is not a limitation of the method, but rather a fundamental property of clustering itself — a cluster that is optimized for one objective is unlikely to be optimal for another.

That said, we were able to achieve better target detection performance using this method than we could without it. The best target detection performance we were able to achieve required use of this this blending method to construct an affinity matrix, but the performance gain from using a blended affinity over an unblended one was minimal.

In the next section we introduce a clustering algorithm that can generate clusterings that are a blend of spectral clustering and a Gaussian mixture. It is the algorithm that we found to have the best performance when used in the clustered target detector. The resulting clusters are partially Gaussian, but also retain some of the desirable spectral properties.

3.6 Laplacian-Regularized GMM (LapGMM)

In Section 3.3 we discussed GMM clustering and its ability to find Gaussian-shaped clusters. In Section 3.5 we discussed spectral clustering and its ability to find clusters based on similarities. In this section, we introduce an algorithm that finds Gaussian-shaped clusters out of spectrally-similar data.

The Laplacian-Regularized GMM (LapGMM) algorithm can be used to incorporate additional relational information into a GMM model. The LapGMM algorithm, shown in Algorithm 4, is based off of the Expectation Maximization GMM Algorithm (2), but introduces Laplacian regularization terms into the M-step to make the clusters maximize for spectral similarity in addition to log-likelihood. When clustering hyperspectral pixels, LapGMM lets us generate clusters that are primarily Gaussian, but also somewhat similar when measured by location or cosine similarity. LapGMM allows us to generate clusters that incorporate all the desirable properties of GMM and the desirable properties from spectral clustering, allowing us to produce Gaussian clusters out of locally-similar pixels. Combining LapGMM with the spectral blending technique discussed in Section 3.5.2 allows us to generate Gaussian-shaped clusters that incorporate any number of relationships.

Zeng et al. [5] showed that a combination of Gaussian Mixture Models and Spectral Clustering can be used to construct accurate image models. He et al. [6] introduced an Expectation Maximization (EM) algorithm for finding LapGMM clusterings and showed the improved performance it can offer over GMM on synthetic datasets, as well as on classification of the USPS handwritten digits data set [37]. Gan et al. [38] applied this algorithm to additional datasets, and also developed a way to incorporate labeled data into the clustering results.

LapGMM seeks to maximize the Laplacian-regularized log likelihood objective function, and is given as

$$\mathcal{L}(\Theta^t) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{K} \alpha_j p_j(\mathbf{x}_i \mid \Theta_j) \right) - \lambda \sum_{k=1}^{K} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(p(k \mid \mathbf{x}_i, \Theta) - p(k \mid \mathbf{x}_j, \Theta) \right) S_{i,j}, \tag{3.11}$$

where Θ_k are the properties of the kth Gaussian distribution, S is the affinity matrix, and α_i is a per-cluster regularization term defined in Algorithm 4.

The two regularization terms in the M-step work together to preserve spectral similarities within the clusters. The regularizer in the log-likelihood computation modifies the objective function to penalize clusters that are not considered spectrally similar. The λ term in Equation 3.11 controls the amount of penalty that is applied to the objective function, and can be used to control the amount of spectral similarity that is present in the final cluster. The log-likelihood regularizer does not change how the clusters are computed, but simply ensures our objective function considers spectral similarity. The regularizer used to smooth the posterior probabilities ensures that the spectral similarity is considered when determining the probability that each pixel belongs to each cluster. This regularizer changes the EM computation to ensure that spectral similarity between data points is preserved.

Unfortunately, the smoothing regularizer can also prevent the EM algorithm from converging to a locally-optimal clustering. When the γ smoothing term in Algorithm 4 is set too high, the smoothing regularizer can preserve too much spectral similarity, preventing the algorithm from finding Gaussian distributions in the data. To ensure that the smoothing regularizer does not prevent the discovery of Gaussian-shaped clusters, the γ term is incrementally reduced. We know that the EM algorithm used to compute GMM generates clusters with monotonically non-decreasing log-likelihood values. In other words, the traditional GMM algorithm will only find better clusters over time. Therefore, if our algorithm produces a clustering with an incrementally worse log-likelihood value, then it must be due to the regularizers, not the Gaussian-derived terms. We reduce the γ term whenever an iteration of the EM algorithm makes our regularized log-likelihood worse. This reduces the amount of spectral similarity that is considered in the EM calculation, allowing the EM algorithm to focus on finding a suitable Gaussian model. It is important to note that reducing γ does not reduce the effect of the log-likelihood regularizer, and therefore does not reduce the total importance placed on the final cluster's spectral affinity.

The log-likelihood regularizer highlights the major trade-off associated with the clusters from LapGMM. While the regularizer ensures spectral similarity is preserved in the clusters, it also can prevent the Gaussian model from finding distributions that fit the data. Clusters from LapGMM suffer from the same fundamental problem as the blended spectral clusters discussed in Section 3.5.2, that ultimately a clustering that is optimized for two things is truly optimized for neither.

That said, using the LapGMM algorithm in the clustered target detector yielded the best performance we were able to obtain, outperforming all the other algorithms we discussed in this chapter. This leads us to believe that the clusterings generated by the LapGMM algorithm most accurately modeled the hyperspectral images. The Laplacian regularizer in the LapGMM algorithm gives us a very powerful tool for tuning. It allows us to express spectral similarities to our algorithm allows us to incorporate application-specific knowledge into our otherwise completely Gaussian clustering. This regularizer gives us a way to to numerically express concepts concepts like shade and location (through similarity measurements), and incorporate those concepts into our algorithm to improve clustering performance.

3.7 Algorithm Complexity

It is important to consider the computational requirements imposed by each of these clustering algorithms. In general, the algorithms that utilize a simpler model generally have lower computational requirements, whereas the algorithms that utilize more nuanced models are considerably more expensive. The k-means algorithm utilizes the simplest model and scales the best out of all of the algorithms we have discussed, with a per-iteration complexity of $\mathcal{O}(NKd)$. The GMM algorithm has a per-iteration complexity of $\mathcal{O}(NKd^3)$, scaling worse than k-means on high-dimensional data, but still scaling relatively well to large data sets. Spectral clustering utilizes the k-means algorithm, and when used in this way the k-means algorithm has a low per-iteration complexity of $\mathcal{O}(NK^2)$. Spectral clustering is not generally limited by this constraint, and is instead typically limited by the computational expense of the Eigendecomposition during initialization, having a complexity of $\mathcal{O}(N^3)$. The LapGMM algorithm has a per-iteration complexity of $\mathcal{O}\left(\max(NKd^3, N^2K)\right)$, making its complexity greater than GMM but generally less than spectral clustering.

Spectral clustering and LapGMM have the highest computational requirements because of the use of affinity matrices. In the next chapter we will show that these algorithms offer improved performance over the simpler algorithms, making the added complexity potentially worthwhile. Additionally, in Section 4.2.1 we discuss how to use sparse matrices to reduce the computational requirements of these algorithms.

3.8 Initialization Considerations

All of the clustering algorithms we have discussed begin with a non-optimal clustering and then iteratively step towards a optimal solution. A problem with all of these clustering algorithms is that they converge to a local optimum, but there is no guarantee that the optimum found is the global optimum (i.e., the best possible clustering). [1] This makes the initialization of these algorithms very important, as the starting clustering will determine which optimal clustering the algorithm arrives at.

LapGMM, being based on GMM, is quite effective at using EM to iteratively to improve the Gaussian portion of its clusters, but the algorithm struggles at improving the spectral portion of the clusterings. This makes initialization especially important, as it is unlikely that the spectral portion of our clusters will iteratively improve. Previous work with the LapGMM algorithm initialized the clusters using either the k-means algorithm [6] or a priori knowledge [38]. We have found that initializing LapGMM with clusters generated via spectral clustering leads to better clusters in terms of both target detection performance and lower Laplacian-regularization penalties. As such, all of the LapGMM results in Chapter 4 are from LapGMM clusters initialized using spectral clustering. Using spectral initialization ensures that the initial clusters have a strong spectral affinity, and allows the LapGMM algorithm to converge to a local-optimum with the desired spectral affinity.

Using spectral initialization relaxes the role of the LapGMM algorithm and allows the algorithm to do what it excels at. With k-means initialization, the LapGMM algorithm starts with a clusters that are somewhat Gaussian and is expected to iteratively discover a clustering that is both more Gaussian and has a stronger spectral affinity. This requires LapGMM to both improve the log-likelihood of the Gaussian distribution and minimize the Laplacian-regularization penalties. With

[1]This makes the results in [35] especially exciting, as they have found a globally optimal clustering solution, albeit for a specific case.

spectral initialization, the LapGMM algorithm starts with spectral clusters and is tasked with finding a Gaussian clustering while maintaining the already low Laplacian-regularization penalty. As before, LapGMM is expected to improve the log-likelihood of the Gaussian distribution, but unlike before, it does not need to improve the Laplacian-regularization penalty, as the initialization ensures that it is already minimized. Instead of tasking the algorithm with improving both terms, we are tasking it with improving one term while preserving the other.

Figure 3.4 shows the influence spectral initialization can have on the resulting clustering. The Left column shows the clusters generated using spectral clustering when using a similarity that is a blend between the distance and cosine similarity measurements. The middle column shows the clusters generated by LapGMM when initialized with the spectral clusters. The right column shows the clusters generated by GMM when initialized by the spectral clusters. What is interesting to note is that the clusters from the GMM algorithm are noticeably different when initialized with different clusters, despite the fact that the algorithm seeks to minimize the same objective function in all cases. This shows the importance of properly initializing both GMM and LapGMM algorithms.

Spectral clustering is admittedly much more computationally expensive than k-means, requiring the construction and subsequent eigendecomposition of the Laplacian matrix. That said, the Laplacian is required by the LapGMM algorithm anyway, so its construction can be seen as a sunk cost, making the only additional computation required over k-means a single Eigendecomposition. Assuming $d > k$, then the use of k-means when spectral clustering is of lower dimensionality than when using k-means directly on the data, offering additional computational savings. In summary, the added computational cost of using spectral clustering to initialize LapGMM is the expense of calculating the Eigendecomposition of the graph Laplacian minus the savings from running k-means at a lower dimension.

3.9 Data Clustering Summary

In this section we investigated several clustering algorithms and discussed the trade-offs associated with each. We discussed several model-based algorithms, specifically the k-means, GMM, and subspace clustering algorithms, as well as the similarity-based spectral clustering. We also looked at a few techniques that can be used to combine and blend these algorithms together, such as spectral blending in Section 3.5.2 and LapGMM algorithm in Section 3.6. In the next chapter, we will test the performance of these algorithms when used on hyperspectral images with our clustered target detector.

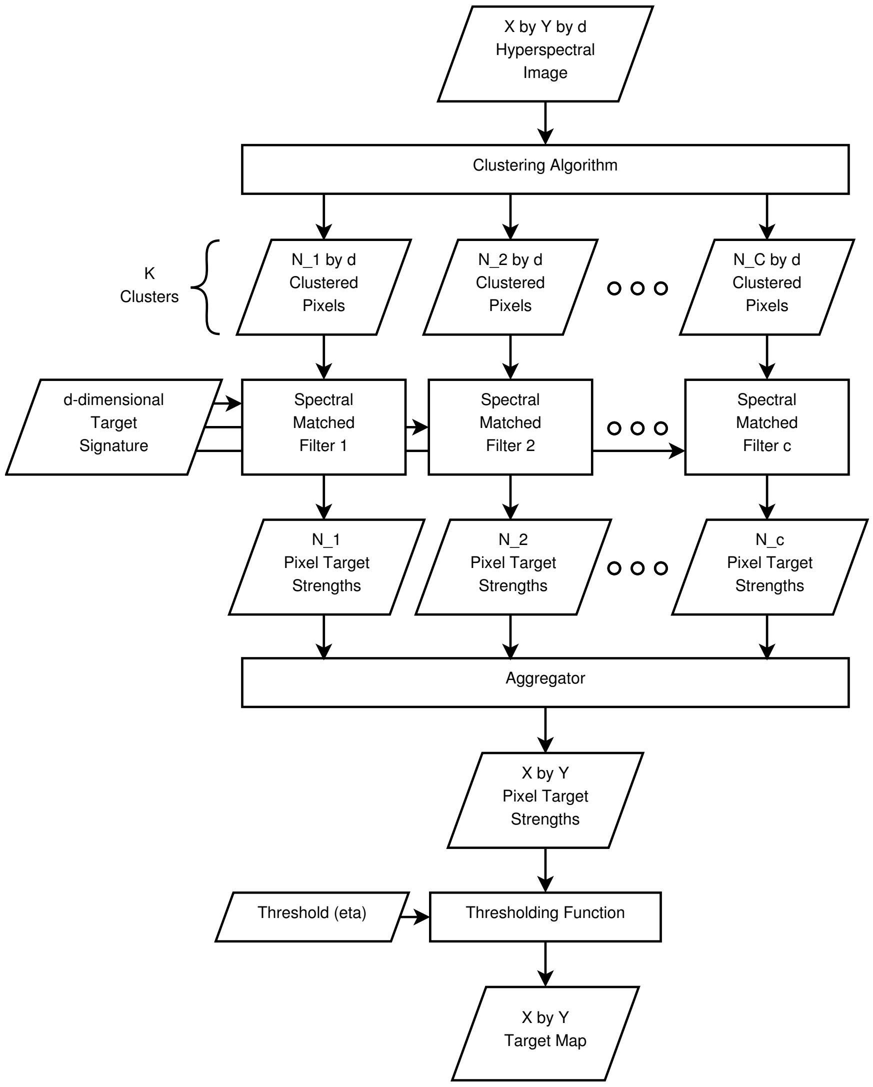
X by Y by d Hyperspectral Image
Clustering Algorithm
K Clusters
N_1 by d Clustered Pixels
N_2 by d Clustered Pixels
N_C by d Clustered Pixels
d-dimensional Target Signature
Spectral Matched Filter 1
Spectral Matched Filter 2
Spectral Matched Filter c
N_1 Pixel Target Strengths
N_2 Pixel Target Strengths
N_c Pixel Target Strengths
Aggregator
X by Y Pixel Target Strengths
Threshold (eta)
Thresholding Function
X by Y Target Map

Figure 3.2: Data Flow for Hyperspectral Target Detection with Clustering. Despite the increased complexity, the inputs and resulting outputs of this method are identical to the method shown in Figure 3.1.

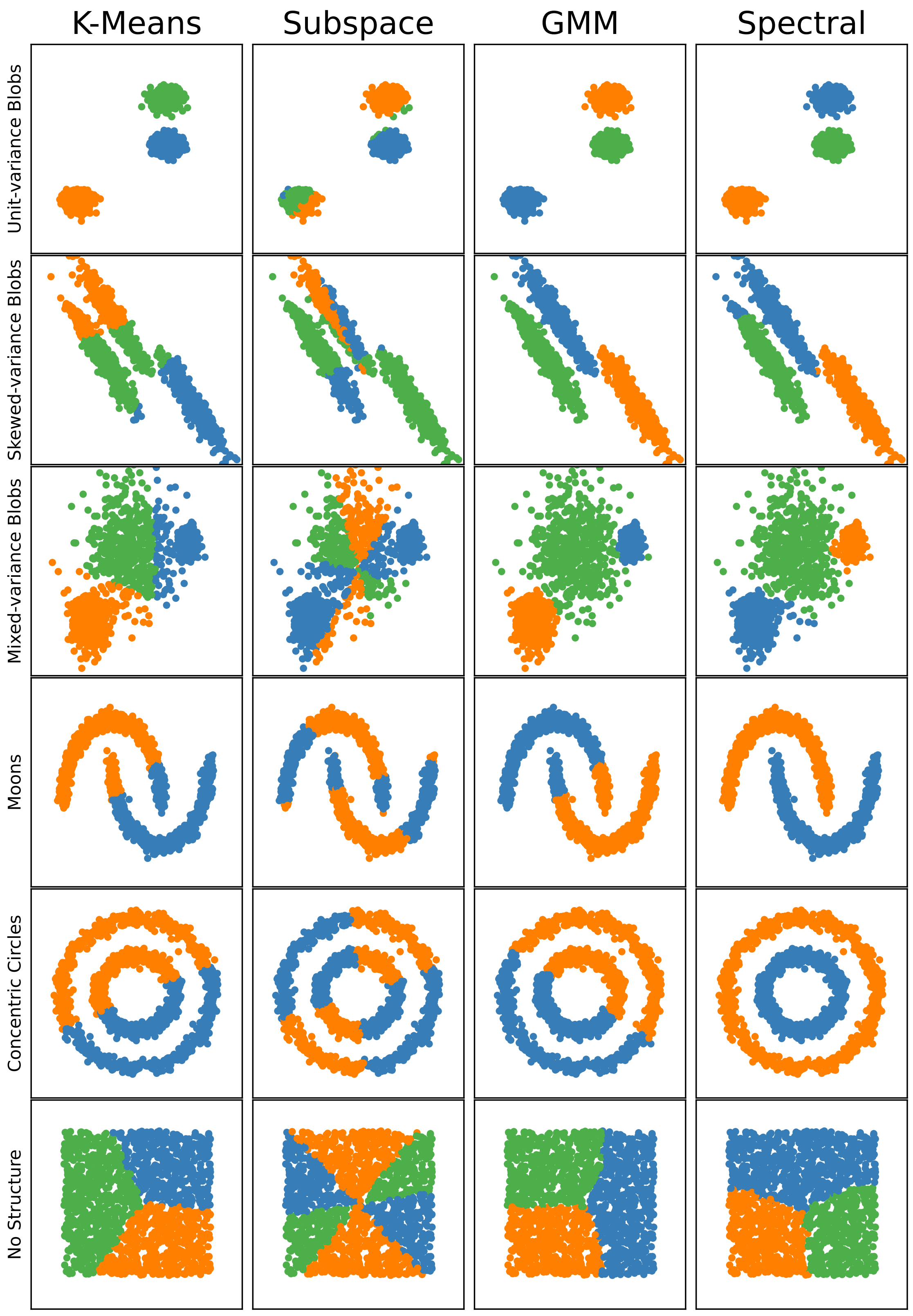

Figure 3.3: A comparison of four clustering algorithms on synthetic datasets. The shape of the data within the set can affect the clustering performance of some algorithms.

Algorithm 4: LapGMM [6]

```
Input     : $\mathbf{X} \subset \mathbb{R}^d$, Number of clusters $K$, regularization parameter $\lambda$,
            termination condition value $\delta$, graph Laplacian $L$
Initalize : $\gamma = 0.9$, $\Theta^0$ by using $k$-means, $t = 1$
Construct an affinity matrix $S$ from graph Laplacian $L$.
Until convergence ($\mathcal{L}(\Theta^t) - \mathcal{L}(\Theta^{t-1}) \leq \delta$):
    E-step:
    Compute posterior probabilities:
    $\mathbb{P}(k \mid \mathbf{x}_i) = \frac{\alpha_k^{t-1} p_k(\mathbf{x}_i \mid \Theta_k^{t-1})}{\sum_{j=1}^{K} \alpha_j^{t-1} p_j(\mathbf{x}_i \mid \Theta_j^{t-1})}$
    M-step:
    Do
        Smooth the posterior probabilities until convergence:
        $\mathbb{P}(k \mid \mathbf{x}_i) = (1-\gamma)\,\mathbb{P}(k \mid \mathbf{x}_i) + \gamma \frac{\sum_{j=1}^{n} S_{ij}\,\mathbb{P}(k \mid \mathbf{x}_j)}{\sum_{j=1}^{n} S_{ij}}$, $(i = 1, \dots, n;$
         $k = 1, \dots, K)$.
        Compute the LapGMM estimates $\alpha_i^t$, $\boldsymbol{\mu}_i^t$, and $\Sigma_i^t$:
        $\alpha_i^t = \frac{1}{n} \sum_{j=1}^{n} \mathbb{P}(i \mid \mathbf{x}_j)$,
        $\boldsymbol{\mu}_i^t = \frac{\sum_{j=1}^{n} \mathbf{x}_j\,\mathbb{P}(i \mid \mathbf{x}_j)}{\sum_{j=1}^{n} \mathbb{P}(i \mid \mathbf{x}_j)}$,
        $\Sigma_i^t = \frac{\sum_{j=1}^{n} \mathbb{P}(i \mid \mathbf{x}_j)(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} \mathbb{P}(i \mid \mathbf{x}_j)}$.
        Evaluate the regularized log likelihood:
        $\mathcal{L}(\Theta^t) = \sum_{i=1}^{n} \log\left(\sum_{j=1}^{K} \alpha_j p_j(\mathbf{x}_i \mid \Theta_j)\right) - \lambda \sum_{j=1}^{K} \mathcal{R}_j$,
        where $\mathcal{R}_k = \sum_{i=1}^{n} \sum_{j=1}^{n} \left(p(k \mid \mathbf{x}_i, \Theta) - p(k \mid \mathbf{x}_j, \Theta)\right) S_{i,j}$
        if $\mathcal{L}(\Theta^t) < \mathcal{L}(\Theta^{t-1})$ then
            $\gamma = 0.9\gamma$
        end
        t = t + 1
    While $\mathcal{L}(\Theta^t) < \mathcal{L}(\Theta^{t-1})$;
Repeat
Output : $\Theta^t$
```

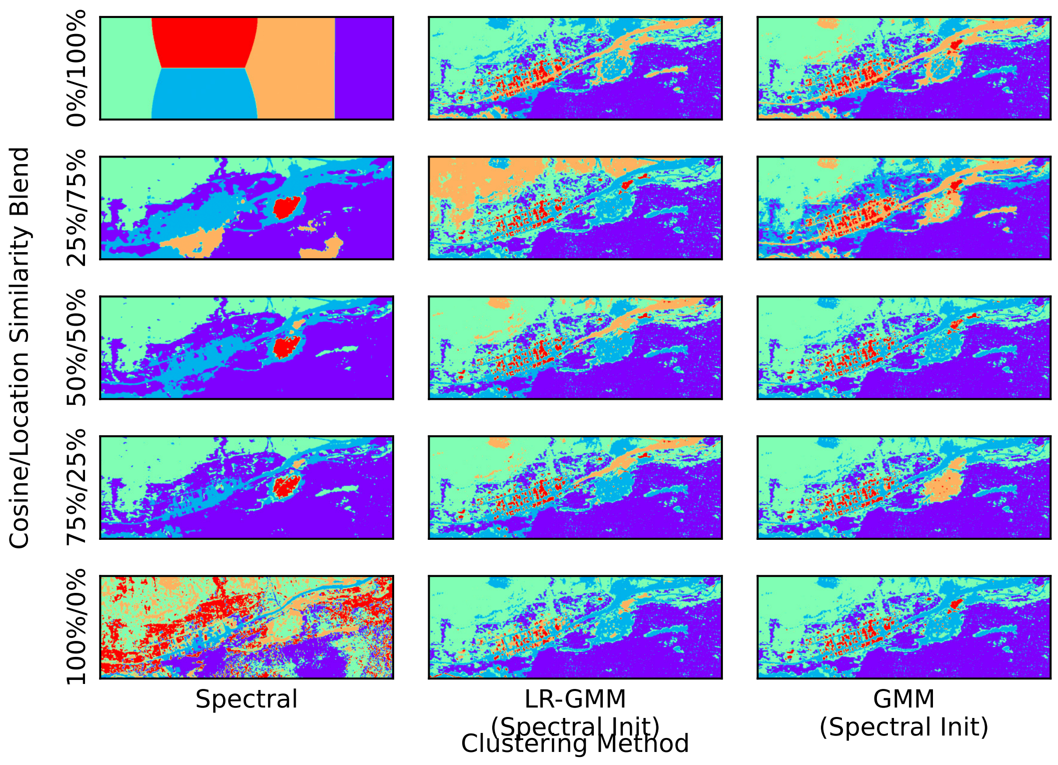

Figure 3.4: Clusters formed by Spectral, GMM, and LapGMM algorithms on the RIT Radiance image when using different measurements of similarity. Notice the differences between the clusters generated by each method, even for GMM where the objective function is the same.

Chapter 4

Results

4.1 Experiment Setup

We ran three sets of experiments to evaluate the performance of our clustered target detector and find a suitable clustering algorithm. First we wanted to evaluate a wide variety of different clustering algorithms to discover which methods seemed most promising in terms of performance. Once we had learned which algorithms offered the best performance, we went to work looking for a way to tune and further improve them for the task of hyperspectral target detection, investigating the use of different similarity measurements with both spectral clustering and LapGMM.

We chose to test the clustered target detector using several hyperspectral images from two data sets. The images in [39] were collected as part of a field experiment in July 2006. The images were captured using a HyMap sensor and include two 126-channel aerial images of Cooke City, Montana and the surrounding area. The images in [40] were collected as part of a field experiment conducted over two days in May 2013. The images were captured using a SIM.GA sensor and include three 511-channel aerial images of a suburban area in Viareggio, Italy. The results in this chapter show the detection performance in terms of *pAUC* scores when tested on one image from [39] and two from [40]. These images were chosen because they offer different levels of target detection difficulty, allowing us to show the performance of these algorithms under a wide variety of conditions.

We chose to run all the tests of our detector with clustering algorithms that find five clusters within each image (i.e., $k = 5$). The choice to use five clusters per image was made as a trade-off between computational expense and target detection performance. Five is a large enough number of clusters to show the benefits of the clustered target detector, and a small enough number for the clustering algorithms to converge quickly. In Section 5.2 we discuss varying the number of clusters as possible future work.

We wanted our results to be representative of our clustered target detectors performance when attempting to detect a variety of targets. To accomplish this we use the aggregated results from 9 different targets embedded into each hyperspectral image in every result shown.

Every image captured in both data sets includes several actual test targets (vehicles and colored tarps) that were placed in the captured area. While the actual targets can be useful for some applications, their use for evaluating target detection performance is admittedly limited. When testing the target detection performance of an algorithm, it is a good idea to consider as many cases as possible, but when working with real hyperspectral data, the limited number of test cases in each image makes that difficult. Ideally we would like to know the performance of the target detector if any target were placed in any location on the image. In the next section, we discuss the computationally-efficient technique to simulate cases where targets are in different locations.

4.2 Simulated Target Embedding

In order to test our clusterings on a wide variety of cases, a simulated target embedding method described in [20] was used. This method allowed us to simulate targets and place them anywhere on the image, and can be used to evaluate the performance of any CFAR detector on a given target. Using this method greatly increased the number of target detection cases we considered when evaluating each clusterings performance.

The first step in this embedding process is to generate two hyperspectral images, One where the target is present in all pixels $(\mathbf{X}_{wt})$, and one where the target is present in none $(\mathbf{X}_{nt})$.

To 'embed' the target signature into a pixel, we use the following equation

$$\mathbf{x}_{wt} = \alpha\mathbf{s} + (1-\alpha)\mathbf{x}, \tag{4.1}$$

which is derived from the hypothesis given in (2.2), where $\mathbf{x}_{wt}$ is the pixel with the target signature embedded and α is a chosen target strength. In all the results shown in this chapter we used a target strength of 5%, i.e., $\alpha = 0.05$.

Once the two images are constructed, the image with no targets $\mathbf{X}_{nt}$ is clustered and used with the target signature to configure the SMFs in our clustered target detector. The clustered target detector is then used to detect targets in $\mathbf{X}_{nt}$ and construct a pixel target strength map. These steps match the process shown in Figure 3.2 for performing clustered target detection. Because $\mathbf{X}_{nt}$ contained no targets, the results from this target strength map will be used to calculate the true negatives and false positives from our detector. The next step is to have the clustered target detector attempt to detect targets in $\mathbf{X}_{wt}$ using the same SMFs and the same clusterings as before and construct a pixel target strength map. Because $\mathbf{X}_{wt}$ only contained target-embedded pixels, the results from this target strength map will be used to calculate the true positive and false negatives from our detector.

The strength map constructed from $\mathbf{X}_{nt}$ shows how the target detector responds to each pixel when there is no target, whereas the strength map constructed from $\mathbf{X}_{wt}$ shows how the target detector responds to each pixel when it contains a target. We can determine the performance of the detector by comparing these two results. Specifically, we can construct two histograms from the strength maps generated by our detector on $\mathbf{X}_{nt}$ and $\mathbf{X}_{wt}$, and use those histograms with all of the techniques described in Section 2.8 to characterize a detector's performance. All of the results in this Thesis were calculated using this method.

By comparing the value of every pixel with an embedded target to the value of every pixel without an embedded target, we can construct a histogram, generate an ROC curve, and compute $pAUC$ values that evaluate the performance of the detector for targets located anywhere in the image.

To use simulated target embedding, we must make several assumptions about the data. First, we must make the assumption that the presence of a point target within a pixel will not alter its signature enough to change the cluster it belongs to. Secondly, we must assume that the addition of the point target to a single pixel would not significantly alter the estimates of the mean and covariance used to construct the matched filter. Because we are specifically considering the point target detection case, where the signature of a single pixel is simulated to change by only 5%, we believe that these are safe assumptions to make. We feel that the impact of this small change to a single data point on both the clustering and the resulting SMF detectors is most likely insignificant.

4.2.1 Sparse Matrices

The LapGMM and Spectral Clustering algorithms highlight both the power and the flexibility of similarity-based clustering. As discussed in Sections 3.5 and 3.6, these algorithms allow their users to numerically define 'similarity', and thus can generate clusters based on any quantifiable measurement of similarity. Unfortunately, this versatility comes at a significant computational cost. While the affinity matrices and graph laplacians that are used by these clustering algorithms are very useful tools, their inherently large size introduces computational challenges. These challenges are similar to the challenges faced by kernelized detectors (discussed in Section 2.7) because they use gram matrices. Like gram matrices, both affinity matrices and graph laplacians are $N \times N$ matrices, and their relatively large size makes them both time-intensive to calculate and resource-intensive to store and manipulate.

To overcome this steep computational expense we approximate all of our affinity matrices and graph laplacians as sparse matrices. Sparse matrices are matrices where only a small number of elements are non-zero. The exact percentage of zero elements needed to consider a matrix truly 'sparse' is debatable, but for our purposes we will consider any matrix where more than half of the elements are zero as 'sparse'. Storing $N \times N$ matrices usually requires enough memory to store N^2 elements, but if a matrix contains mostly zero elements, then we can store the matrix as a table containing the locations and values of all non-zero elements, and can assume the elements not listed in the table are zero-valued. Storing matrices in this way greatly reduces the memory footprint needed to store a sparse matrix, but requires additional overhead as the locations of the non-zero elements must now be stored. That said, when the matrix contains mostly zero elements, using this method can greatly reduce the memory needed to store it. Affinity matrices and graph laplacians are generally not sparse, but can often be closely approximated with a sparse matrix.

We construct our sparse affinity matrix approximations by keeping the M largest values in each row of the matrix and setting the rest to zero, where M is a natural number and $M < N$. For our calculations we chose $M = int(\sqrt{N})$ to ensure the matrices had substantial sparsity. Using the sparse approximations is computationally equivalent to only considering the M most similar points to a given data point when clustering, ignoring all the weaker relationships. Matrix approximations generated using this method are significantly smaller and contain significantly less information than the full matrices, but still work well for clustering in practice because the most important information (i.e., the strongest similarities between points) is still present in the matrices.

4.3 Comparison of Baseline Clustering Algorithms

The results of our clustered target detector when used with several clustering algorithms are shown in Table 4.1. The ROC curves of the clustered target detector on the RIT Radiance image are shown in Figure 4.1, with the spectral results from Table 4.1 shown in Figure 4.2.

GMM clusters provided the best target detection performance on the RIT Radiance image. Based on the poor target detection results from all the target detectors, we believe this image poses a relatively difficult target detection problem. Other algorithms that performed well on this image were the cosine spectral clustering, with a total AUC value slightly below GMM, and distance spectral clustering, which performs well at low max FPR (θ) values. The SSC-OMP clusters performed the worst on this image out of all the clustering algorithms, providing performance that was only slightly better than the non-clustered SMF target detector.

Surprisingly, the k-means provided the best target detection performance on the Viareggio Day 1 image. That said, virtually every detector performed well on this image, including the non-clustered SMF target detector, leading us to believe that this image poses the least challenging target detection problem. SSC-OMP again has the worst performance, but still outperforms the unclustered SMF target detector.

The Viareggio Day 2 image appears to pose a target detection challenge that is somewhere between the Day 1 image and the RIT Radiance image, with pAUC values landing between these two extremes. With this image the target detector performed the best when using the cosine spectral clustering. This was the only image where we saw a comparable target detection performance from the SMF target detector and a clustered target detector, specifically when when the SSC-OMP algorithm was used for clustering cluster.

Multiple conclusions can be drawn from these results. While the results show that

Image	RIT Radiance			Viareggio Day 1			Viareggio Day 2		
Max FPR (θ)	0.01	0.1	1.0	0.01	0.1	1.0	0.01	0.1	1.0
No Clustering	0.0148	0.156	0.698	0.317	0.776	0.971	0.0663	0.331	0.826
k-means	0.0437	0.28	0.783	**0.58**	**0.9**	**0.989**	0.15	0.507	0.895
Subspace (EnSC)	0.0762	0.301	0.772	0.567	0.885	0.986	0.169	**0.533**	**0.897**
Subspace (SSC-OMP)	0.0279	0.186	0.719	0.495	0.866	0.984	0.0663	0.331	0.826
Spectral (Cosine)	0.0879	**0.339**	**0.805**	0.562	**0.895**	**0.988**	**0.275**	**0.584**	**0.909**
Spectral (Location)	**0.1**	0.302	0.773	0.559	0.88	0.985	**0.177**	0.483	0.879
GMM	**0.192**	**0.415**	**0.831**	**0.568**	0.891	0.987	0.153	0.509	0.895

Table 4.1: The pAUC values of several common clustering algorithms when used with the clustered target detector.

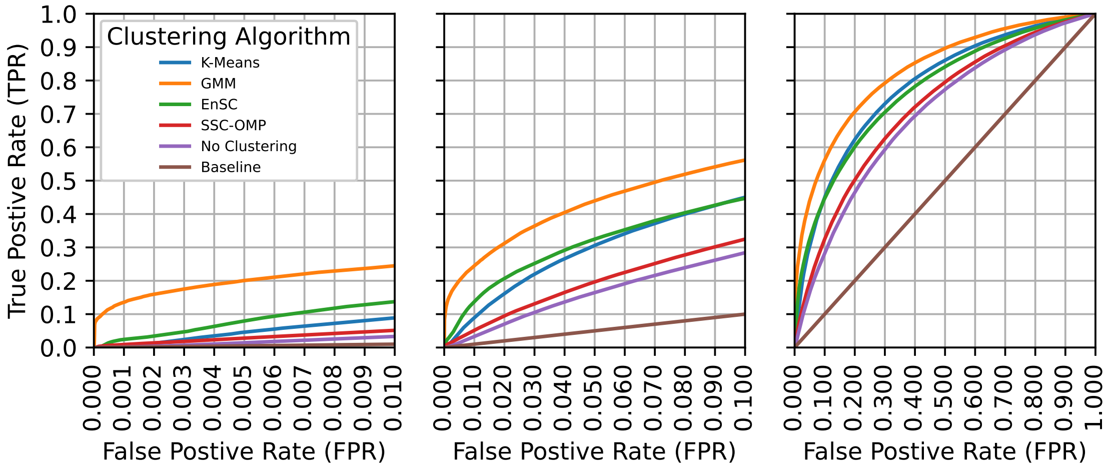

Figure 4.1: A comparison of the performance of the clustered target detector when using several popular clustering algorithms on the RIT Radiance image. GMM outperforms all other algorithms in all cases, and all algorithms outperform the non-clustered results.

different clustering algorithms offer different levels of detection performance, they also show that our target detector can outperform (or at least perform just as well as) a non-clustered SMF target detector when used with virtually any clustering algorithm. It also shows that even when used to perform the same task, the best performing clustering algorithm can vary depending on the image.

4.4 Comparison of Spectral Clustering Results

Based on the results from our first set of experiments described in Section 4.3, we felt that both GMM and spectral clustering were methods worth further investigation. In this section we show the performance of our target detector when using spectral clustering with a variety of similarity measurements. In the next section we show the performance of the LapGMM algorithm using these clusterings for initialization, and regularizing with the same graph Laplacian.

Table 4.2 and Figure 4.2 show the performance of the clustered target detector when using the spectral clustering algorithm with a variety of similarity measurements. The results make it difficult to say what measurement works best with our detector — when comparing the performance from the spectral clusterings generated from the cosine, location, and Euclidian-based RBF similarity measurements, no clustering yields better performance than the others in all cases. Each notion of similarity outperforms the others on one of the images, with cosine performing best on the Viareggio Day 2 image, location performing best on the RIT Radiance image, and RBF performing the best on the Viareggio Day 1 image. That said, all of the algorithms perform fairly well on the Viareggio Day 1 image, leading us to believe that that the benefits of RBF similarity on truly challenging problems are likely limited.

The last row of Table 4.2 shows the detection performance when using a similarity measurement that was a blend of 40% cosine similarity (Equation (3.7)) and 60% location similarity (Equation (3.8)). We blended the similarities using the process described in Section 3.5.2. As we discuss in the next section, this is the blend of similarities that we have found to offer the best performance when used to regularize and spectrally initialize the LapGMM algorithm. We show these results in the next section. These results help explain why the use of this blended similarity measurement with LapGMM offers improved performance. The results in Table 4.2 and Figure 4.2 clearly show that the spectral clusters generated from this blend lead to better results at low pAUC values. When attempting to use an EM clustering algorithm to improve performance, A good place to start the search is with a clustering that already performs quite well, which spectral initialization allows us to do.

Image	RIT Radiance			Viareggio Day 1			Viareggio Day 2		
Max FPR (θ)	0.01	0.1	1.0	0.01	0.1	1.0	0.01	0.1	1.0
No Clustering	0.0148	0.156	0.698	0.317	0.776	0.971	0.0663	0.331	0.826
Spectral (RBF)	0.0543	0.241	0.75	**0.631**	**0.922**	**0.991**	0.17	**0.53**	**0.902**
Spectral (Cosine)	0.0879	**0.339**	**0.805**	**0.562**	**0.895**	**0.988**	**0.275**	**0.584**	**0.909**
Spectral (Location)	**0.1**	0.302	0.773	0.559	0.88	0.985	0.177	0.483	0.879
Spectral (40 Cos 60 Loc)	**0.119**	**0.339**	**0.795**	0.552	0.889	0.987	**0.282**	0.524	0.888

Table 4.2: The pAUC values of spectral clustering when using various similarity metrics with the clustered target detector.

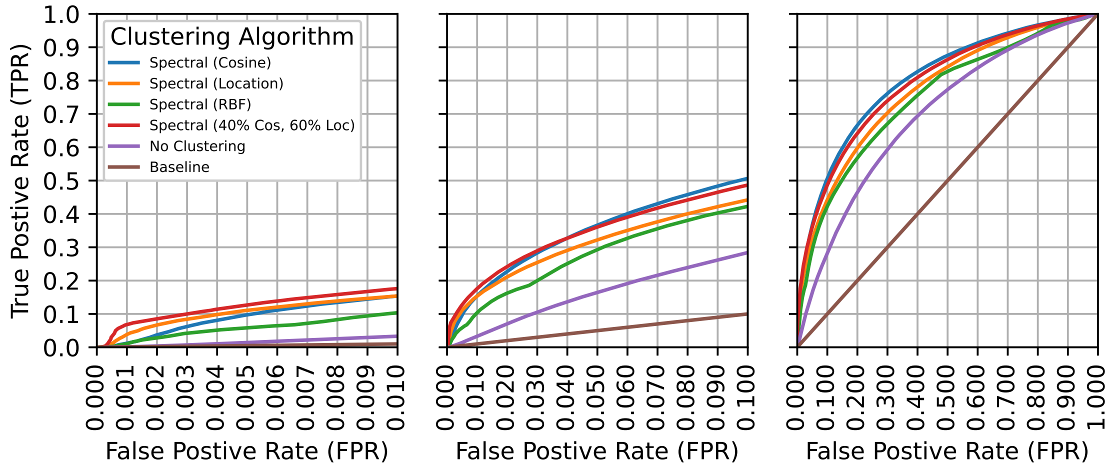

Figure 4.2: A comparison of the performance of the clustered target detector when using spectral clusters on the RIT Radiance image. The clusters constructed using the cosine and distance similarity measurements performed well, and the cluster constructed using a blend of these similarities outperformed all others when the FPR is low.

4.5 Comparison of LapGMM Results

This section shows the performance of our clustered target detector when using clusters generated by the LapGMM algorithm. Clusters generated by the LapGMM algorithm provided the best results that we were able to achieve with our clustered target detector. Table 4.3 compares the results of one of the best performing algorithms in Section 4.3, GMM, with the results we were able to achieve with LapGMM.

As discussed in Section 3.8, we construct a graph Laplacian and use it in the LapGMM algorithm for both spectral initialization and Laplacian Regularization. When comparing the performance effects of using the RBF, cosine, and location similarity measurements the results align with the results in Section 4.4, with cosine

Image	RIT Radiance			Viareggio Day 1			Viareggio Day 2		
Max FPR (θ)	0.01	0.1	1.0	0.01	0.1	1.0	0.01	0.1	1.0
GMM	0.192	0.415	0.831	0.568	0.891	0.987	0.153	0.509	0.895
LapGMM (RBF)	0.167	0.389	0.804	**0.637**	**0.923**	**0.991**	0.178	0.535	**0.902**
LapGMM (Cosine)	0.199	0.418	0.834	0.572	0.898	**0.988**	**0.291**	**0.605**	**0.914**
LapGMM (Location)	**0.214**	**0.435**	**0.835**	0.607	0.899	0.988	0.209	0.545	0.901
LapGMM (40 Cos 60 Loc)	**0.219**	**0.435**	**0.837**	**0.609**	**0.9**	0.988	**0.305**	**0.565**	0.9

Table 4.3: The pAUC values of the clustered target detector when using LapGMM that is regularized and spectrally initialized with various similarity metrics.

performing best on the Viareggio Day 2 image, location performing best on the RIT Radiance image, and RBF performing the best on the Viareggio Day 1 image.

Armed with the knowledge that different notions of similarity offer better performance in different situations, we set out to find a blended notion of similarities that offered improved performance in all situations. We investigated several blends of two to three similarity measurements, and through trial and error we discovered a blend that performed well on all of the images, in many cases outperforming all unblended similarities. The blend we found to be most effective over all cases consisted of 40% cosine similarity and 60% location similarity. While the RBF similarity was able to improve target detection performance more than any other similarity on the Viareggio images, The use of it on the RIT Radiance image resulted in decreased performance. For this reason we left it out of our final blend. In contrast, the cosine and location similarities improved performance over GMM in all cases, leading us to believe that their inclusion is likely beneficial in most cases.

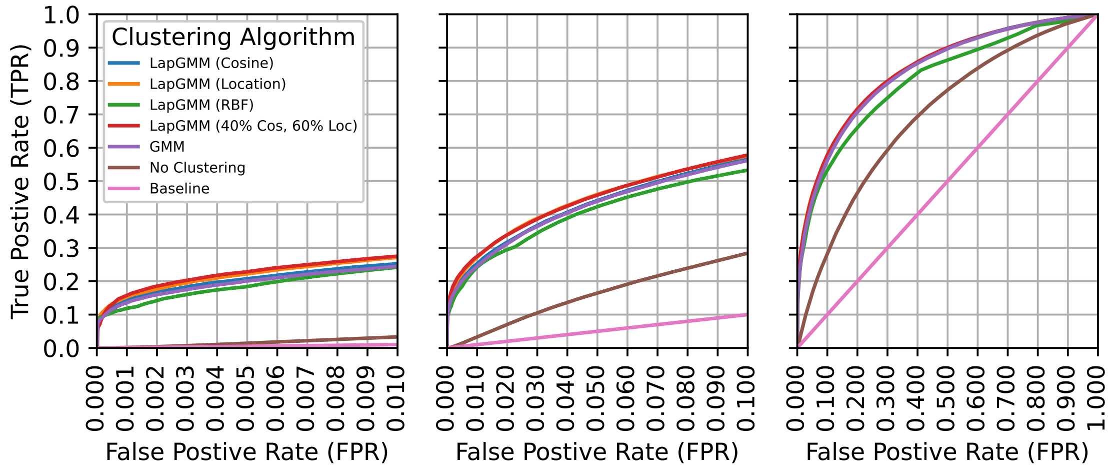

Figure 4.3: Clustered target detector performance when using LapGMM on the RIT Radiance image. LapGMM outperforms GMM in most cases, with the RBF case being the notable exception.

Chapter 5

Conclusion & Future Work

5.1 Conclusions

Our results conclude that clustering can indeed be used to improve target detection performance. In Chapter 4 we showed empirically that our SMF-based clustered target detector outperformed the non-clustered SMF target detector in every case we investigated. That said, the performance of the clustered target detector varied widely depending on the algorithm used to cluster the pixels, making the choice of clustering algorithm important.

Additionally, we conclude that initialization is very important for most clustering algorithms, and that the de facto or default initialization method of an algorithm may not yield the most performant clustering. As discussed in Section 3.8, we have found spectral initialization to be especially valuable (and fairly computationally inexpensive) when used with the LapGMM algorithm.

Another conclusion we came to is that in order for a clustering algorithm to perform well with our clustered detector, it must recognize multiple types of similarities in the data. Table 5.1 compares the results from several of the tests conducted in Chapter 4. Specifically it shows that the detection performance of a clustering algorithm can be improved by incorporating additional relationships. Our clustered detector outperforms the unclustered detector by a significant margin when using the GMM algorithm. The GMM algorithm finds Gaussian mixtures and effectively incorporates the Mahalanobis distance into our clustering. Table 4.3 showed that when the LapGMM algorithm is used with certain similarity measurements, it can outperform the unregularized GMM algorithm in all cases. Table 5.1 shows one such example, where LapGMM utilizes location similarity to improve target detection across the board. The final row of Table 5.1 shows that a blended similarity measurement can be used to further improve the performance of our clustered detector with the LapGMM algorithm. Overall this table shows that incorporating additional measurements of similarity can lead to incremental improvements in target detection performance.

Table 5.1 also highlights the diminishing returns when attempting to optimize for multiple objectives, as discussed in Section 3.5.2. The performance gained by using GMM clustering over no clustering at all is quite large. In contrast, the performance gained from incorporating location similarity into GMM is fairly small, and the gain from incorporating both location and cosine similarity is even smaller.

One of the most important conclusions we came to was that there is no one-size-fits-all clustering algorithm, and the 'best' clustering algorithm depends on both the dataset as well as the application. As shown in Chapter 4, different clustering algorithms perform differently on different data sets, even when used for the same application. That said, some algorithms (specifically GMM and LapGMM) appeared to perform well in all cases, and in Section 3.3 we proposed some possible hypotheses as to why. In the next section we discuss additional applications where GMM and LapGMM may offer performance similar to the impressive performance shown in our results, and discuss some possible applications and explanations for algorithms that did not perform well in our clustered detector.

Image	RIT Radiance			Viareggio Day 1			Viareggio Day 2		
Max FPR (θ)	0.01	0.1	1.0	0.01	0.1	1.0	0.01	0.1	1.0
No Clustering	0.0148	0.156	0.698	0.317	0.776	0.971	0.0663	0.331	0.826
GMM	0.192	0.415	0.831	0.568	0.891	0.987	0.153	0.509	0.895
LapGMM (Location)	**0.214**	**0.435**	**0.835**	**0.607**	**0.899**	**0.988**	**0.209**	**0.545**	**0.901**
LapGMM (40 Cos 60 Loc)	**0.219**	**0.435**	**0.837**	**0.609**	**0.9**	**0.988**	**0.305**	**0.565**	**0.9**

Table 5.1: The pAUC values of several algorithms we have discussed sorted by complexity. Additional complexity offers additional performance, but with diminishing returns.

5.2 Future Work

As shown in Chapter 4, the use of subspace algorithms in our SMF-based clustered target detector did not produce noteworthy results. We believe that this was due at least in part to a mismatch between the clustering algorithm and the SMF filter. We assume that our SMF-based target detector would perform best when given Gaussian-shaped clusters, since that is how the SMFs model the image background. This would explain why the algorithms that yield the best performance with the SMF-based clustered target detector are the ones that find clusters based on Gaussian models. In contrast, subspace algorithms find clusters based on subspace models, and as such we believe that these clusters may offer good performance if paired with a target detection algorithm that models the background as a subspace, such as the 'Matched Subspace Detector' or the 'Adaptive Subspace Detector' (both described in [2]).

The RX anomaly detection algorithm is closely related to the SMF and models the background in the exact same way. As such, we suspect that the clustering algorithms that yield good results when paired with our SMF-based clustered target detector would also yield good results when paired with an equivalent RX-based clustered anomaly detector.

We speculate that the tuned LapGMM algorithm we developed performs well because of its ability to accurately model the background. If true, then this clustering algorithm could be applied to other hyperspectral tasks, such as classification. It would be interesting to compare the classification performance of our tuned LapGMM algorithm and the *k*-means-based trilateral filtering method described in [36].

This investigation did not look into the effect varying the number of clusters has on the performance of our detector. As such, the best number of clusters used to divide the image is still an open question. We suspect that the ideal number of clusters to use for a given problem is directly related to how well that number allows the model to fit the data. For GMM and LapGMM, this would involve finding the number of Gaussian distributions needed to accurately represent the contents of the image. In [5], an algorithm is developed for finding this value in a 3-channel image that could potentially be applied to hyperspectral images as well.

Bibliography

[1] "Hyperspectral Imaging: An Emerging Tool for Mission Readiness | L3Harris." [Online]. Available: https://www.harris.com/perspectives/global-situational-awareness/hyperspectral-imaging-an-emerging-tool-for-mission

[2] N. M. Nasrabadi, "Hyperspectral target detection : An overview of current and future challenges," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 34–44, 2014.

[3] "Receiver operating characteristic," sep 2020. [Online]. Available: https://en.wikipedia.org/wiki/Receiver{_}operating{_}characteristic

[4] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*, 2013, vol. 9781107057. [Online]. Available: http://www.cs.huji.ac.il/{~}shais/UnderstandingMachineLearning

[5] S. Zeng, R. Huang, Z. Kang, and N. Sang, "Image segmentation using spectral clustering of Gaussian mixture models," *Neurocomputing*, vol. 144, pp. 346–356, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2014.04.037

[6] X. He, D. Cai, Y. Shao, H. Bao, and J. Han, "Laplacian regularized Gaussian mixture model for data clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 9, pp. 1406–1418, 2011.

[7] A. Stockman and L. Sharpe, "The spectral sensitivities of the middle- and long-wavelength-sensitive cones derived from measurements in observers of known genotype," *Vision research*, vol. 40, pp. 1711–1737, 2000.

[8] "Technical Details « Landsat Science." [Online]. Available: https://landsat.gsfc.nasa.gov/about/technical-information/

[9] "OCI™-F Hyperspectral Imager (VIS-NIR, SWIR) - BaySpec." [Online]. Available: https://www.bayspec.com/spectroscopy/oci-f-hyperspectral-imager/

[10] J. A. Plascyk, "The MK II Fraunhofer Line Discriminator (FLD-II) for Airborne and Orbital Remote Sensing of Solar-Stimulated Luminescence," *Optical Engineering*, vol. 14, no. 4, 1975. [Online]. Available: https://doi.org/10.1117/12.7971842

[11] H. Kwon and N. M. Nasrabadi, "A comparative analysis of kernel subspace target detectors for hyperspectral imagery," *Eurasip Journal on Advances in Signal Processing*, vol. 2007, p. 13, 2007.

[12] M. Handford, *The Great Waldo Search.* Boston: Little, Brown and Company, 1989.

[13] S. Buganim and S. R. Rotman, "Matched filters for multispectral point target detection," *Imaging Spectrometry XI*, vol. 6302, no. 2, p. 63020Z, 2006.

[14] A. C. Koivunen and A. B. Kostinski, "The feasibility of data whitening to improve performance of weather radar," *Journal of Applied Meteorology*, vol. 38, no. 6, pp. 741–749, 1999.

[15] P. C. Mahalanobis, "On the generalized distance in statistics." National Institute of Science of India, 1936.

[16] F. C. Robey, D. R. Fuhrmann, E. J. Kelly, and R. Nitzberg, "A CFAR adaptive matched filter detector," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 28, no. 1, pp. 208–216, 1992.

[17] C. C. Funk, J. Theiler, D. A. Roberts, and C. C. Borel, "Clustering to improve matched filter detection of weak gas plumes in hyperspectral thermal imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 7, pp. 1410–1420, 2001.

[18] S. M. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice Hall, 1997.

[19] D. Manolakis, M. Pieper, E. Truslow, T. Cooley, M. Brueggeman, and S. Lipson, "The remarkable success of adaptive cosine estimator in hyperspectral target detection," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XIX*, S. S. Shen and P. E. Lewis, Eds., vol. 8743, International Society for Optics and Photonics. SPIE, 2013, pp. 1–13. [Online]. Available: https://doi.org/10.1117/12.2015392

[20] Y. Cohen, Y. August, D. G. Blumberg, and S. R. Rotman, "Evaluating subpixel target detection algorithms in hyperspectral imagery," *Journal of Electrical and Computer Engineering*, 2012.

[21] R. Li and S. Latifi, "Improving hyperspectral subpixel target detection using hybrid detection space," *Journal of Applied Remote Sensing*, vol. 12, no. 01, p. 1, 2018.

[22] I. S. Reed and X. Yu, "Adaptive Multiple-Band CFAR Detection of an Optical Pattern with Unknown Spectral Distribution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 10, pp. 1760–1770, 1990.

[23] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[24] A. Shashua, "Introduction to Machine Learning 67577-Fall, 2008," Tech. Rep., 2009. [Online]. Available: https://arxiv.org/abs/0904.3664

[25] Matthew N. Bernstein, "The Radial Basis Function Kernel," pp. 1–4, 2006. [Online]. Available: http://pages.cs.wisc.edu/{~}matthewb/pages/notes/pdf/svms/RBFKernel.pdf

[26] X. Wu, B. Guo, C. Chen, and H. Shen, "A RX-based hyperspectral target detection method by fusing two kernels," *Proceedings - 2014 7th International Congress on Image and Signal Processing, CISP 2014*, pp. 536–540, 2014.

[27] B. Schölkopf, A. Smola, and K.-R. Muller, "Kernel Principal Component Analysis," in *Gerstner W., Germond A., Hasler M., Nicoud JD. (eds) Artificial Neural Networks — ICANN'97. ICANN 1997. Lecture Notes in Computer Science*, 2011, no. 3, pp. 1–6.

[28] H. Kwon and N. M. Nasrabadi, "Kernel RX-algorithm: A nonlinear anomaly detector for hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 2, pp. 388–397, 2005.

[29] S. M. Kay, *Fundamentals of Statistical Signal Processing: Detection theory*, ser. Fundamentals of Statistical Signal Processing. PTR Prentice-Hall, 1993. [Online]. Available: https://books.google.com/books?id=XF6xxAEACAAJ

[30] M. Pieper, D. Manolakis, E. Truslow, T. Cooley, and S. Lipson, "Performance evaluation of cluster-based hyperspectral target detection algorithms," in *2012 19th IEEE International Conference on Image Processing*. IEEE, 2012, pp. 2669–2672.

[31] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. January 9, 2007.

[32] P. S. Bradley and O. L. Mangasarian, "k-Plane Clustering," Tech. Rep., 1999.

[33] C. You, D. P. Robinson, and R. Vidal, "Scalable Sparse Subspace Clustering by Orthogonal Matching Pursuit," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. December, 2016, pp. 3918–3927.

[34] C. You, C. G. Li, D. P. Robinson, and R. Vidal, "Oracle Based Active Set Algorithm for Scalable Elastic Net Subspace Clustering," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. December, pp. 3928–3937, 2016.

[35] M. Löffler, A. Y. Zhang, and H. H. Zhou, "Optimality of Spectral Clustering for Gaussian Mixture Model," 2019. [Online]. Available: http://arxiv.org/abs/1911.00538

[36] W. Sun and D. W. Messinger, "Trilateral filter on multispectral imagery for classification and segmentation," *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVII*, vol. 8048, no. May 2011, p. 80480Y, 2011.

[37] J. J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, may 1994.

[38] H. Gan, N. Sang, R. Huang, and X. Chen, "Manifold regularized gaussian mixture model for semi-supervised clustering," in *Proceedings - 2nd IAPR Asian Conference on Pattern Recognition, ACPR 2013*, 2013, pp. 361–365.

[39] D. Snyder, J. Kerekes, I. Fairweather, R. Crabtree, J. Shive, and S. Hager, "Development of a web-based application to evaluate target finding algorithms," *International Geoscience and Remote Sensing Symposium (IGARSS)*, vol. 2, no. 1, pp. 915–918, 2008.

[40] N. Acito, S. Matteoli, A. Rossi, M. Diani, and G. Corsini, "Hyperspectral Airborne 'Viareggio 2013 Trial' Data Collection for Detection Algorithm Assessment," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 6, pp. 2365–2376, 2016.