ABSTRACT

BRAYFINDLEY, EVANGELINA. Automated Defect Detection in Spent Nuclear Fuels in Wet Storage Using Machine Learning and Image Analysis Techniques. (Under the direction of Dr. Ralph Smith.)

The International Atomic Energy Agency (IAEA) is the governing body of the world's nuclear programs, negotiating agreements, inspecting facilities and conducting research around the world. One widely implemented aspect of IAEA-negotiated agreements is spent fuel monitoring. Such monitoring is costly and repetitive. In any given cooling pond, there is a prohibitive number of spent fuel assemblies that require inspection. Even with sub-sampling, the number of assemblies inspected by the IAEA is remarkably large even within a single facility.

Spent fuel inspections, therefore, are an ideal application of automation. However, in a regime as critical as nuclear treaty verification, robustness is paramount. Anomalies must be detected with a high true positive rate (above 70%) and a low false positive rate (less than 10%).

Two currently implemented methodologies for spent fuel inspections and defect detection are optimal for such automation, as their instrumentation strengths and weaknesses are complementary and can thus be combined for increased accuracy. The first instrument is the Digital Cerenkov Viewing Device (DCVD) which gives a top-down view of the assembly. The second instrument is Gamma Emission Tomography (GET) which provides a reconstructed cross-section of the assembly at a particular height.

In this dissertation, we develop methods for robust automated defect detection using both Bayesian and non-Bayesian approaches. The non-Bayesian approaches are rooted in facial recognition-based algorithms and include principal component analysis with k-nearest neighbors, neural networks, and convolutional neural networks. These methods serve as proof-of-concept for combining DCVD and GET for automated defect detection in 17×17 PWR fuel. True positive defect identification rates are maximized with a convolutional neural net at 70%, with the corresponding false positive rate just under 1%.

However, to combine the two data types robustly, we detail alternate methodologies using Bayesian data fusion within metric learning that additionally quantifies uncertainty by estimating the classification distribution of a fuel rod. We thus develop, test and implement a pipeline Bayesian neighborhood component analysis (BNCA) with linear opinion pooling algorithm. While our true positive rates are not as high as in the non-Bayesian method, at 5%, the false positive rates are consistently below 5%. When burnup and cooling times are consistent between training and test sets, however, we are able to identify true defects correctly in nearly 60% of tests. By estimating distributions, we can also estimate confidence in the classifications of each rod in an assembly, which is essential for effective safeguards-applicable algorithms. Finally, the BNCA+LOP Pipeline results identified model weaknesses to address in future research, particularly the underlying binary nature of the classification framework.

The major contribution of this work is in the safeguards application. Both the developed Bayesian and non-Bayesian methodologies demonstrate the advantage of fusing DCVD and GET data to improve defect detection and quantify detection uncertainty. In addition, our methods avoid computationally expensive physics models for database pattern matching. Instead, our machine learning algorithms are able to learn features intrinsic to defects that enable detection with small training data sets. Mathematically, the major contributions of this work are as follows. The first is in the development of expert-informed weighting schemes within simple fusion frameworks and their demonstrated value when combining machine learning algorithms such as neural nets. The second contribution lies in the development of fused supervised algorithms applicable specifically to defect detection where there are few existing examples of defects for validation. The final major contributions relate to the development of a fusion algorithm within a metric learning framework, the BNCA+LOP Pipeline algorithm. Within this framework, novel developments include a Cholesky-based compression, the addition of the fusion pipeline framework, and the use of the BNCA+LOP Pipeline to identify model deficiencies.

Automated Defect Detection in Spent Nuclear Fuels in Wet Storage Using Machine Learning and Image Analysis Techniques

by
Evangelina Brayfindley

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Applied Mathematics

Raleigh, North Carolina

2019

APPROVED BY:

Dr. Mansoor Haider	Dr. Hien Tran
Dr. John Mattingly	Dr. Ralph Smith Chair of Advisory Committee

DEDICATION

To my husband, Nate Grayley, and my family, Rod, J.B. and Isak Brayfindley.
I couldn't have done this without the support of those around me. The constant cheerleading from my family kept me motivated, and the companionship of my dogs kept me sane. But most of all, I would like to thank my husband for supporting me in the entirety of my degree and in helping make sure I ate something besides coffee and licorice while writing and researching this thesis.

BIOGRAPHY

The author pursued degrees in both chemistry and mathematics at the University of San Francisco in CA before moving to North Carolina to pursue her graduate degrees. After receiving her Master's in Computational Applied Mathematics, the author then pivoted towards nonproliferation research and direct applications of mathematics in the operations of nuclear plant inspections and fuel storage, aiming to find practical solutions to long-term problems in nuclear engineering.

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor, Dr. Ralph Smith, for his technical expertise and help along the way. I'd also like to acknowledge my committee members, Drs. Mansoor Haider, John Mattingly, and Hien Tran for their guidance. A special thank you to Dr. Robert Brigantic from Pacific Northwest National Laboratory under whose guidance this project began at a summer internship, and without whom, would not have succeeded.

This research was funded by the U.S. Department of Energy's National Nuclear Security Administration (NNSA) through the Consortium for Nonproliferation Enabling Capabilities (CNEC) under award number DE-NA0002576.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Problem Statement and Application

The International Atomic Energy Agency (IAEA) is the international governing body for nuclear facilities, technology, and agreements. Although established in 1957, it was the 1970 Nonproliferation Treaty (NPT) that made the IAEA an integral part of nuclear facilities and operations across the world [29]. IAEA inspectors visit facilities around the world to conduct inspections and ensure the use of nuclear technology for peaceful purposes. One particular aspect of inspection, spent fuel monitoring, has been a pillar of safeguards agreements for several decades. One of the first Comprehensive Safeguards Agreements (CSA) between the IAEA and Canada, completed in 1972, includes required material accounting of spent fuel in storage [3]. At nuclear plants, spent fuel is stored in wet storage (cooling ponds) until it cools. It can then be transferred to dry storage in concrete casks or sent to permanent storage facilities. In France and Japan, spent fuel is reprocessed for future use in nuclear facilities. The primary motivation for spent fuel inspections is material accounting–the host country must be able to show where all potentially weapons-usable material is at a given time in order to confirm its usage for exclusively peaceful applications.

There are two kinds of defects that inspectors look for in spent fuel assemblies: gross defects, where an entire fuel assembly is a non-fuel item, or partial defects, where $\geq 50\%$ of the individual fuel rods in an assembly are non-fuel items. These non-fuel items cover both cases where the fuel item has been completely removed and cases where the fuel item was replaced with a non-fuel item, such as steel. Ideally, the IAEA would like to identify when even a single fuel rod within an assembly is a non-fuel item.

The problem faced by the IAEA for many years has been expanding agency responsibilities with the growth of the global nuclear industry, while being held to a nearly constant budget. Because of this, robust automation of any repeat processes of large data sets is of critical interest. Spent fuel monitoring is a repetitive task over a very large set of assemblies within a cooling pond. Whereas the automation of the measurement technology itself has been intensely investigated, this work primarily focuses on the automation of classification of the large streams

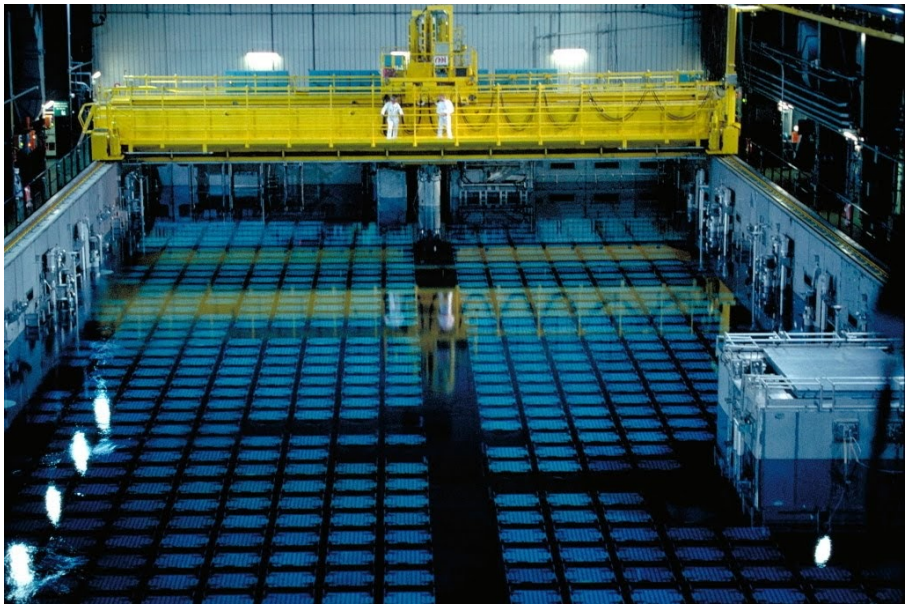


Figure 1.1: Spent fuel cooling pond [1].

of data taken during these measurements.

In particular, the Digital Cerenkov Viewing Device (DCVD), described in Section 1.2.1, has been used since the mid 1980s as a gross defect detection technique. In more recent years, gamma emission tomography (GET), in Section 1.2.1, has emerged with the potential for use as a single rod defect detection method. Past research has shown that for large assemblies–for example, a standard 17×17 assembly– and in varying burnup and cooling time scenarios, both DCVD and GET have weaknesses when determining defects. Whereas DCVD remains a gross defect method, it is a much cheaper option than GET which, as currently implemented, requires several detectors placed at multiple heights along an assembly for high accuracy.

The goal of our investigation is to combine the complementary strengths and weaknesses of DCVD and GET spent fuel measurement methods to develop a robust single fuel rod defect detection method. Using GET data from only a single height along a given assembly and DCVD data that provides a top-down view of that same assembly, we will incorporate the best of both methods while minimizing their weaknesses for a more robust and automated identification of defects in spent fuel ponds.

Prior work has focused on using a single method for gross and partial defect detection. Prior work has also relied on either pre-existing libraries or generating large theoretical databases by modeling detector physics for defect pattern matching. Our work is novel in that it proposes a combination of methods to develop an automated pipeline to directly identify defects with high accuracy, assuming the assembly geometry is known. Using multiple data sources and machine learning algorithms, our approaches circumvent expensive physics models and can extrapolate to new detector scenarios; e.g., burnup and cooling time modifications, without having to gen-

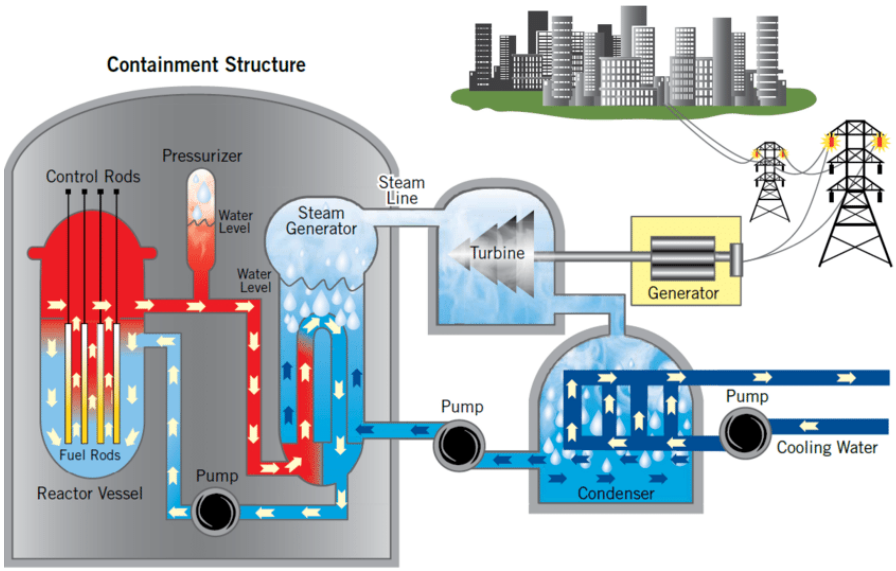


Figure 1.2: Pressurized Water Reactor (PWR) schematic [51].

erate comprehensive libraries. Additionally, the robust combination of data through Bayesian fusion methods will be unique in its nuclear safeguards application. Finally, the combination of Bayesian Neighborhood Component Analysis (BNCA) with Bayesian data fusion provides a unique and new mathematical formulation addressing the combination of data from varing detectors and of varying sizes in a supervised learning context.

1.2 Background and Prior Work

1.2.1 Nuclear engineering methods and background

There are two primary classes of nuclear reactors currently in operation: boiling water reactors (BWRs) and pressurized water reactors (PWRs). PWRs are the focus of this work, with a general schematic as shown in Figure 1.2. In that schematic, the fuel is shown as individual fuel rods. In fact, the fuel is an array or *assembly* of rods, with arrays varying in size. In our data set, we consider 17×17 PWR fuel assemblies shown in Figure 1.3. Thus, there is a consistent general geometry of the assembly, and each fuel rod in an assembly is numbered according to the pin map depicted in Figure 1.4. Once the fuel assemblies are removed from the reactor, they are stored in cooling ponds, as depicted in 1.1. The spent fuel, once in storage in cooling ponds, is the focus of our research.

Spent nuclear fuel is primarily described by two factors: burnup and cooling time. Burnup is a measurement of how many atoms undergo fission while in the reactor, or equivalently, the energy released per amount of starting material. Cooling time is the amount of time the spent fuel has been out of the reactor and cooled in some manner. Both are factors in the signal

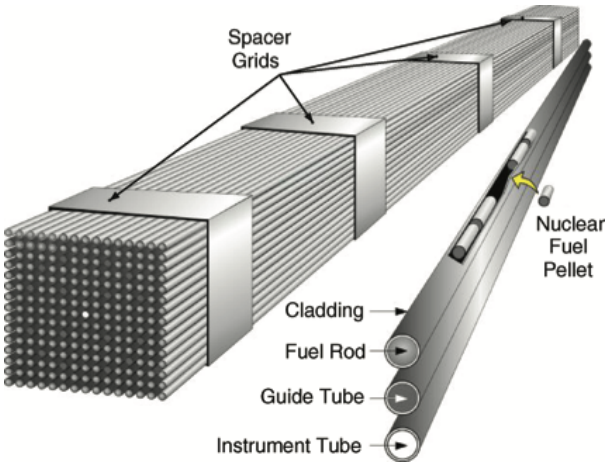


Figure 1.3: 17×17 PWR fuel schematic [15].

detected by spent fuel inspection tools.

Digital Cerenkov Viewing Device (DCVD)

Cerenkov light is light produced in the visible spectrum by charged particles moving faster than the speed of light in a given medium. The Digital Cerenkov Viewing Device (DCVD) is a specialized, stationary camera used to photograph the Cerenkov radiation emitted from spent fuel rods in wet storage; see Figure 1.5. Another version of this detector is the Improved Cerenkov Viewing Device (ICVD) [30], which, for simplicity, can be considered the video version of the DCVD. DCVDs are currently widely implemented in IAEA inspection sites but the handheld ICVDs are becoming more common in inspections [30]. Renewed interest in DCVDs to inspect large quantities of spent fuel before relocation to long-term storage motivates our focus on this method [9]. Additionally, the data gathered from ICVDs are videos that can be split into frames, so the analysis performed on still images remains applicable.

Due to the motion of water in the cooling ponds, assembly geometries, and the meters-long distance between the camera and measured assemblies, this method has limited accuracy [9]. The DCVD is most accurate at the center of its field of view, which negatively impacts much of the data at the edges of an assembly. Because the CVDs only detect light from the visible region, they can be easily blocked by hardware like handles and other assembly geometry features that cover regions of a given assembly. In particular, the edges of the fuel assembly are often covered by hardware. For partial defect detection, a DCVD is able to detect when 15% or more rods are missing, or when 50% or more of fuel rods in an assembly are substituted, but higher accuracy is limited [13]. This imposes a minimum detectable defect on spent nuclear fuel (SNF) inspections. An important detail for interpreting DCVD images is that due to the nature of Cerenkov radiation, a present fuel rod will yield a dark region, whereas water or missing rods (defects) will be bright. Another item of note is that DCVD is much more sensitive in the

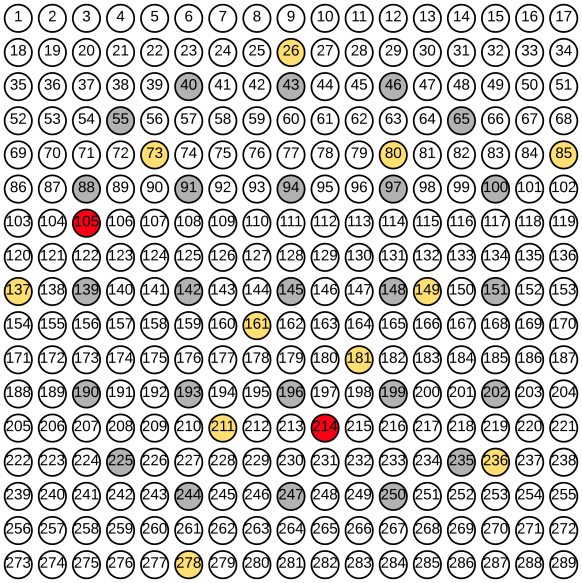


Figure 1.4: 17×17 Pin map: gray = fuel rod, white = guide tube, yellow = DCVD defect location, red = GET defect location [49].

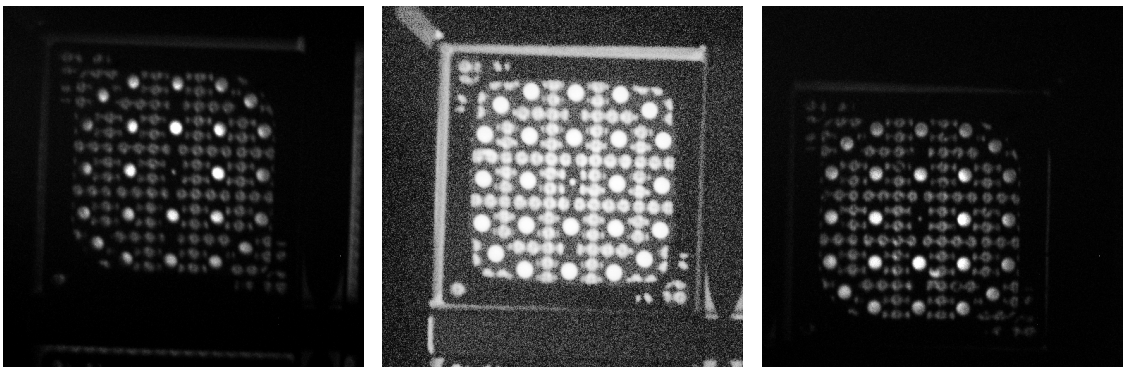
middle of the camera, and loses sensitivity moving towards the edge of the field of view due to collimation effects.

The Applied Nuclear Physics group at Uppsala University and their collaborators have been very influential in both the CVD and GET development. This group has worked on simulations, and various improvements of those simulations, for DCVD images [8], as well as template matching in the resulting image [9]. Within the scope of the simulations, they have also investigated various properties of spent fuel that impact the simulations and how to adjust for those impacts. Additionally, they have produced recommendations for improving the DCVD data collection process [9].

Overall, the DCVD detector is most accurate at the center of its field of view which, when properly calibrated, is the center of the assembly. Because of the distance between the assembly and the detector, environmental factors like water motion in the cooling pond have a large impact on the detected signal and corresponding defect detection accuracy. DCVD signals are derived from ultraviolet light in the water surrounding a fuel rod, with a non-defect producing a dark region–this allows for substitutions of a fuel rod with another material to go undetected, as long as that material blocks light in a fuel rod location [30]. When it comes to gross defects; i.e., when an entire assembly is missing, DCVD is highly accurate. On the partial defect level, however, previous testing has shown that DCVD defect detection is limited to cases where at least 15% of fuel rods are missing or substituted, which is equivalent to 40 fuel rods in the case of 17×17 PWR fuel [27]. While the 15% removed/substituted level is high, current IAEA requirements for partial defect accuracy are at the 50% removed/substituted level.

Figure 1.5: Digital Cerenkov Viewing Device (DCVD) [12].

The speed of measurement of DCVD, at 10-30 seconds per assembly, constitutes a significant advantage of the method [7]. Additionally, DCVD is accurate specifically in low burnup and long cooling time scenarios, where many other detection methods are more limited. Finally, the 15% removed/substituted requirement for accurate defect detection is heavily impacted by assembly hardware. Whereas assembly hardware cannot be modified, even a small amount of extra information from a detection method not limited by hardware and DCVDs other drawbacks could provide a large increase in defect detection accuracy.



(a) (b) (c)

Figure 1.6: DCVD data set [10]: (a) no defect, (b) defect in location 105, and (c) defect in location 214.

Gamma Emission Tomography (GET)

Another method currently under investigation as a single defect detection method for safeguards implementation is gamma emission tomography (GET), a prototype of which is shown in Figure 1.7 [52]. In this method, the detector is a collar that encircles a fuel assembly, taking measurements at a particular height along the assembly. This collar detector setup permits increased accuracy in defect detection around the outer edges of the assembly [49]. However, this same setup requires either that the detector be submerged near highly radioactive material, shortening the equipment lifetime, or a time- and labor-intensive process of moving the fuel assembly from the cooling pond to a hot cell for inspection. In addition, the GET measurement time is an order of magnitude larger than that of DCVD–hours rather than seconds for a measurement [49]. This makes the GET method significantly more expensive than DCVD. Finally, GET detection methods rely on the ability of emitted gammas to reach the detector without being absorbed within the assembly. However, in the larger pressurized water reactor (PWR) fuel assemblies, GET demonstrates limited sensitivity due to the large size of the assembly and small inter-rod distances [49]. For PWR fuel, depending on burnup and cooling time scenarios as well as where a defect is located within an assembly, GET has between a 20% and 95% probability of identifying a single defect [49]. The 95% probability is limited to high burnup/short cooling time scenarios [49]. For low burnup and long cooling times, GET has a 20-25% probability of identifying a single defect with a false positive rate at 10% [49]. Note that these rates are with simulated data with 20% rod-to-rod variation, but otherwise ideal measurement scenarios.

The GET data in this work comes from Pacific Northwest National Laboratory PNNL); for detailed simulation parameters, see [49]. Sinograms were simulated with Monte Carlo N-Particle Transport Code (MCNP) by calculating sinograms for single pins, summing them, and adding Poisson noise to account for pin-to-pin variation. The integration time, 3.89 seconds, is consistent with a one hour measurement. Reconstructions were then calculated using a filtered back projection algorithm. Sinograms and reconstructions for the data used in this work are plotted in Figure 1.8. There are two cases of burnup and cooling times considered in this work, and both are shown in Figure 1.8. Case 1 is a high burnup, short cooling time scenario, and Case 2 is a low burnup, long cooling time scenario. For details, see Table 5.1.

Gamma emission tomography has been a field of interest for many years, and has been difficult to perfect. Experimental information [31, 34] and simulated data [32, 33, 49, 50, 57] have both been produced through the years. Similar to DCVD research, prior related work has focused on template matching and fuel rod location identification [14]. Current tomography research has focused on multiple methods of cross-section recreation, but the data provided by Pacific Northwest National Laboratory and used for this investigation implements filtered back projection [49].

Overall, GET is primarily of interest because it overcomes the weaknesses of DCVD. It is better able to identify defects near the edge of the assembly, and can identify defects due to both

Figure 1.7: Gamma Emission Tomographer (GET) [16].

removal and substitution, whereas DCVD is primarily limited to removals. In the larger PWR assemblies, and in low burnup/long cooling time scenarios, the single defect detection is about 25%. The major drawback of GET is the cost and time required; a standard measurement takes an hour as opposed to DCVD's 30 seconds. Additionally, the GET detector has to be submerged with the assembly in the cooling pond or the assembly removed from the cooling pond into a hot cell. Finally, the GET detector also requires multiple detectors at multiple heights along the assembly for best accuracy.

The goal of our research is to investigate defect detection accuracy when DCVD measurements are combined with GET measurements taken at only a single height along the assembly.

1.3 Outline and Significant Contributions

In this dissertation, we discuss the methods we developed and implemented to solve the problem of defect detection using both DCVD and GET data. In Chapter 2, we discuss machine learning methods and their strengths and weaknesses, specifically in the defect detection problem. In Chapter 3, we discuss various optimization algorithms typically implemented in the training of machine learning algorithms from Chapter 2. The mathematical concepts of data fusion are introduced in Chapter 4, followed by non-Bayesian classification results in Chapter 5. Finally, Chapter 6 presents a novel pipeline algorithm that combines Bayesian classification and data fusion for probabilistic classifications.

In Chapter 5, we present non-Bayesian classification results. While the mathematical methods pre-date this work, their application to spent nuclear fuel defect detection is novel. Additionally, the fusion techniques we implement in Chapter 5, the combine-then-classify and classify-then-combine schemes first discussed in Sections 4.2.1 and 4.2.2, have not been well-documented in the literature.

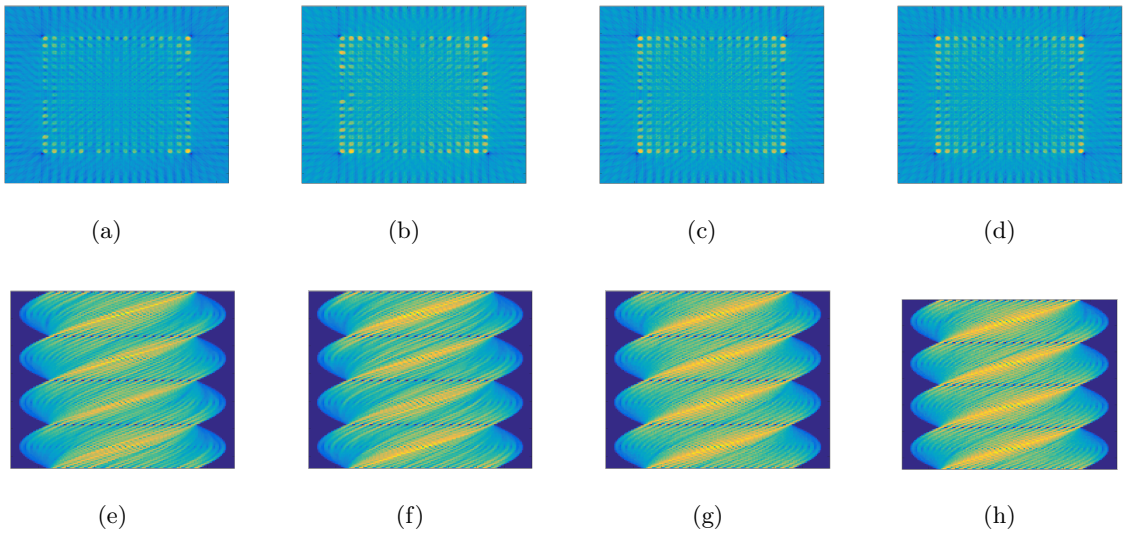


Figure 1.8: Tomographic reconstruction data set [49]: (a) Case 1, Image 1, (b) Case 1, Image 2, (c) Case 2, Image 1, (d) Case 2, Image 2, (e) Case 1, Sinogram 1, (f) Case 1, Sinogram 2, (g) Case 2, Sinogram 1, (h) Case 2, Sinogram 2.

In Chapter 6, we present Bayesian classification results. In particular, our contributions focus on compression, fusion, and again, our spent fuel application. Our proposed compression technique, using Cholesky factors within the BNCA framework, maintains the Mahalanobis nature of our learned metric, whereas the original BNCA formulation in [54] does not. The second main contribution in this chapter is the development of a fusion pipeline, the BNCA+LOP Pipeline algorithm, built around the BNCA algorithm. This pipeline aggregates data from multiple sources of differing sizes and quality, providing probabilistic classifications as output.

The over-arching contributions of this dissertation, however, are focused in the application space, providing the proof-of-concept algorithms for combining DCVD with GET to automate defect detection in spent fuel in wet storage. The machine learning methods we develop circumvent the computational expense of current industry standards by directly identifying underlying data structures in defects rather than generating large libraries of theoretical data for pattern matching. Overall, the methods and results contained in both Chapters 5 and 6 show potential for improved defect detection over current methodologies in the safeguards regime for 17×17 PWR fuel. The methods of Chapters 5 increase the true defect detection rate by nearly 50% in Case 2 burnup and cooling time. The methods of Chapter 6 do not perform as well, with true defect detection rates below 5% in some cases. When burnup and cooling times are consistent between training and test set data, however, we are able to identify true defects in nearer 60% of tests. However, methods developed in Chapter 6 identify weaknesses in our initial assumptions and provide a path forward for quantifying uncertainties in automated defect classification.

Chapter 2

Machine Learning Overview

2.1 Machine Learning Methods

To develop the machine learning methods used throughout this work, we start with a motivating example: the exclusive or function (XOR) [21]. The data points are shown in Table 2.1, with true and false as the classes (denoted by 1 and 0 respectively). From a plot of this data in Figure 2.1, we can see that the classes are not *linearly separable*; i.e., we cannot draw a single line between the two classes to separate them. Because the data is not linearly separable, we cannot easily separate the data into two classes and hence we need classification methods that don't rely on linearity.

The following sections detail several methods for solving this problem with machine learning. We detail the associated optimization methods in Chapter 3, including gradient descent, stochastic, and Newton's method-based algorithms.

2.1.1 Supervised versus Unsupervised Learning

There are two broad classes of machine learning algorithms: supervised and unsupervised algorithms. Supervised learning takes advantage of any *labels* that may come with the original data for which we're learning a classification algorithm. For example, in the XOR problem, the labels are the true and false categorizations of each data point. Supervised methods use these labels by penalizing incorrect classifications during the training stage for a final trained algorithm that

Table 2.1: XOR data points with true and false denoted by 1 and 0, respectively.

x_1	x_2	**Class**	**Graph legend**
1	1	0	●
0	0	0	●
0	1	1	♦
1	0	1	♦

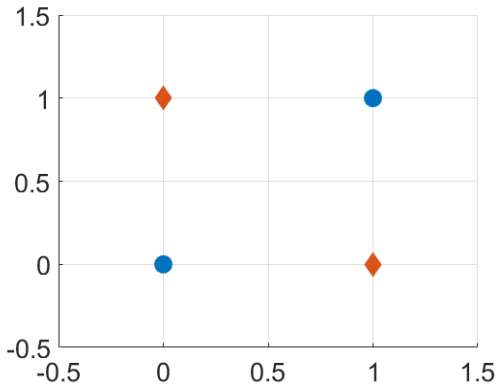


Figure 2.1: XOR graph.

best fits both the original data and their known labels. In the case of our nuclear application, the labels are defect/non-defect, or when extended beyond the simple binary problem, defect, water tube/guide channel, and present rod.

If the XOR example did not have any true/false labeling, then we would employ unsupervised learning. In unsupervised methods, there are no true labels by which we can penalize an incorrect classification. Instead, unsupervised methods rely on identifying some structure within the data based on similarities and differences between points according to a variety of heuristics. Generally, because there is no associated ground truth that the algorithm can compare its performance against, unsupervised methods require more data than supervised methods to be able to identify classes in a data set.

2.1.2 k-Nearest Neighbors

Intuitively, it makes sense to classify a data point based on the class of its neighbors; this is the principle behind k-Nearest Neighbors (kNN). We denote our data set as $\mathbf{X}$, with the columns as the individual data vectors, $\mathbf{x}$. Let k indicate the number of neighbors used to classify a particular $\mathbf{x}$, and $d(\mathbf{x}, \mathbf{y})$ be the distance between two points $\mathbf{x}$ and $\mathbf{y}$ in our set given some distance metric. Given a point $\mathbf{x}$, the kNN algorithm searches through all points in the data matrix $\mathbf{X}$ until it finds the k points closest to the point $\mathbf{x}$ based on a distance metric $d(\mathbf{x}, \mathbf{y})$. In the kNN context, both the number of neighbors k and the distance metric $d(\cdot, \cdot)$ are parameters that can be learned during a training stage. We provide an overview of the algorithm in Algorithm 1.

With the XOR example, we consider the point $\mathbf{x} = (1, 1)^T$. Our goal is to determine whether the classification for this point should be 0 or 1 based on its neighbors. One set of parameter choices for the XOR problem is shown in Figure 2.2. Based on the classes of the two nearest

Algorithm 1: kNN Algorithm from [4]

Result: kNN classification of data
Initialization: choose number of neighbors k and metric d;
for *all* x_i *data points in our set* **do**
  Calculate $d(x_i, x_j)$ for all j
  Select x_{ij}^k as the k values of x_j that maximize $d(x_i, x_j)$ for each x_i
  switch do
    To fit a new model: return the mean of the neighborhood labels to classify x_i
    To classify with an existing model: return the mode of the neighborhood labels to classify query point x_i
  end
end

neighbors, given a Euclidean distance metric with

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{N} (x_i - y_i)^2 \right)^{\frac{1}{2}},$$

we would classify the point $\mathbf{x} = (1, 1)^T$ as belonging to class 1.

Because we know that the true label of the point $\mathbf{x} = (1, 1)^T$ is in fact 0, clearly kNN with our choice of parameters is not the best classifier for this problem. In fact, there is no set of parameters using kNN on its own for this problem that will correctly classify the point $\mathbf{x} = (1, 1)^T$. Another issue with kNN is the following. Had we chosen only one nearest neighbor, this Euclidean distance metric would have yielded a tie between the two nearest points as they are each equidistant to $\mathbf{x}$. Ties can be addressed by the implementation of a tie-breaking criteria [4]. Examples of tie-breakers include randomly choosing between the tied points or simply choosing the point that appears first in the data set.

Another issue that may arise, but which we don't see in the XOR problem, is if there is no clear classification among the neighbors. In the case of two nearest neighbors, this manifests as both neighbors belonging to a different class, which raises the question of how the new point is classified. This can be solved by either increasing or decreasing k until the tie is broken, or weighting the neighbors under consideration inversely proportional to their distance from the point being classified [4].

In summary, we have two options if we want to use kNN for the XOR problem: either modify the parameters or pair kNN with another algorithm to map the data to a new space where kNN performs better. Because modifying the parameters does not provide a solution for XOR, we employ a second option.

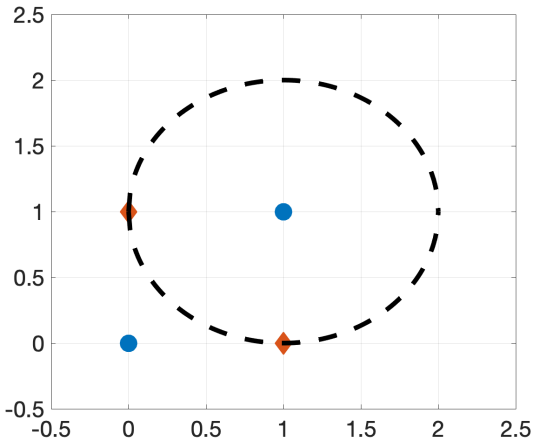


Figure 2.2: Two nearest neighbors, Euclidean distance.

2.1.3 Principal Component Analysis

In principal component analysis (PCA), the goal is to remap data to a new space that maximizes the variance in a data set [5]. In theory, maximizing the variance will permit identification of any trends in the data and improve the ability to separate classes.

Let $\{\mathbf{x}_n\}_{n=1}^N$ denote a set of N observations, where each $\mathbf{x}_n$ is a D-dimensional column vector. The idea of compression with principal component analysis is to project the data $\{\mathbf{x}_n\}_{n=1}^N$ from its initial D dimensions to a space with dimensionality $M << D$ [5]. The sample covariance matrix of the data in the original D-dimensional space is defined to be

$$\mathbf{S} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T.$$

For an introduction to the mathematics of PCA, we assume for now that $M = 1$, so we are projecting our data onto a 1-dimensional space [5]. The goal of PCA is to identify the mapping such that the variance in the new 1-dimensional space is maximized. Let this new 1-dimensional subspace be defined by some vector $\mathbf{u}_1$; if the data set variance in $\mathbf{u}_1$-space is maximized, it is the first principal component. For now, assume that $||\mathbf{u}_1|| = 1$, as a non-unit norm would appear only as a coefficient carried through this derivation. Each data point in $\{\mathbf{x}_n\}$ is projected into the space defined by $\mathbf{u}_1$ and can be expressed as $\mathbf{u}_1^T\mathbf{x}_i$ for $i \in 1, \ldots, M$. The variance of the projected data is $\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1$ [5], where $\mathbf{S}$ is the covariance matrix defined in the original D-dimensional space and $\bar{\mathbf{x}}$ is the sample average of the data set,

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n.$$

To find the direction of greatest variation, we maximize the projected variance with respect to

$\mathbf{u}_1$, constrained so $||\mathbf{u}_1||$ stays finite. We thus have a constrained optimization problem with a Lagrange multiplier, $\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T\mathbf{u}_1)$. Taking the gradient with respect to $\mathbf{u}_1$ and setting it equal to 0, we find that $\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1$, implying that $\mathbf{u}_1$ must be an eigenvector of $\mathbf{S}$ with the associated eigenvalue λ_1. The variance is maximized when $\mathbf{u}_1$ is the eigenvector with the largest eigenvalue λ_1 [5]. Thus, vector $\mathbf{u}_1$ is the first principal component.

The process continues similarly for any subsequent principal components $\mathbf{u}_2, \mathbf{u}_3, \ldots, \mathbf{u}_M$ when $M > 1$, with the constraint that each component is orthogonal to all the prior principal components. The final set of components is an orthogonal basis for the new space onto which the data is projected. This orthogonalization can be done via the standard Gram-Schmidt algorithm; see [56] for details, or, if data is centered, via singular value decomposition (SVD) as detailed later. When all the principal components have been defined, we can reconstruct the original data via the relation

$$\mathbf{x} \approx \bar{\mathbf{x}} + \sum_{i=1}^{M} \left(\mathbf{x}_n^T\mathbf{u_i} - \bar{\mathbf{x}}^T\mathbf{u_i}\right)\mathbf{u_i}. \tag{2.1}$$

If the data is centered; i.e., the mean is subtracted from all data vectors, then we can solve the eigenvalue/eigenvector problem using singular value decomposition (SVD) [53]. When the data is centered, the sampled covariance matrix $\mathbf{S}$ is $\mathbf{S} = \frac{1}{N-1}\mathbf{X}^T\mathbf{X}$, where $\mathbf{X}$ is the $N \times D$ data matrix with N observations of D variables. The covariance matrix is nonnegative and symmetric. Hence it is diagonalizable, and can be expressed $\mathbf{S} = \mathbf{V}\mathbf{L}\mathbf{V}^T$, where $\mathbf{V}$ is the matrix of eigenvectors of $\mathbf{S}$–and of $\mathbf{X}^T\mathbf{X}$ by our covariance definition in the centered case–and $\mathbf{L}$ is the diagonal matrix of eigenvalues of $\mathbf{S}$ [53]. This aligns with the previous derivation of principal components as eigenvectors of $\mathbf{S}$. However, if we take the singular value decomposition (SVD) of our original data matrix, we obtain

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where $\mathbf{U}$ is unitary, $\mathbf{\Sigma}$ is the diagonal matrix of singular values, and $\mathbf{V}$ is the matrix of eigenvectors of $\mathbf{X}^T\mathbf{X}$ [53]. The covariance matrix is now

$$\mathbf{S} = \frac{1}{N-1}\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \tag{2.2}$$

Because $\mathbf{U}$ is unitary, $\mathbf{U}^T\mathbf{U} = I$. The matrix $\mathbf{\Sigma}$ is diagonal so $\mathbf{\Sigma}^2$ is comprised of squared singular values along its diagonal. Equating our two definitions for $\mathbf{S}$, we obtain

$$\mathbf{S} = \mathbf{V}\mathbf{L}\mathbf{V}^T = \frac{1}{N-1}\mathbf{V}\mathbf{\Sigma^2}\mathbf{V}^T.$$

This implies that $\mathbf{L} = \frac{1}{N-1}\mathbf{\Sigma}^2$, and therefore that the eigenvectors of $\mathbf{X}^T\mathbf{X}$ found via SVD of the original data corresponds to the eigenvectors of $\mathbf{S}$, and ultimately, are equivalent to the principal components of the data [53]. Thus, if data is centered, the principal components are

Table 2.2: Centered XOR data points.

Index	x_1	x_2	Class
1	0.5	0.5	0
2	-0.5	-0.5	0
3	-0.5	0.5	1
4	0.5	-0.5	1

computed using an SVD of the original data.

In the XOR example, centering the data yields the points in Table 2.2. Since the data is centered, we can compute the principal component mapping using singular value decomposition. Often, compression is required due to computational limitations. However, we may be able to compress data by examining the decay of the eigenvalues. If there is a significant gap in eigenvalue magnitude, or if they fall below a user-defined threshold, then the principal components will be all components corresponding to eigenvalues above this magnitude change/threshold. In this particular case, we have no compression, and the principal components of the centered data are simply the standard $\mathbb{R}^2$ basis, $[1, 0]$ and $[0, 1]$. In this particular case, PCA does not change the current data mapping. This implies that the variance within our data is already maximized.

This example shows an important flaw of PCA for classification; maximizing the data variance does *not* imply that we have maximized the number of correct classifications. When a data set is unlabeled; i.e., we don't know the true classifications of any data points, maximizing the variance is often the best choice. In this case, PCA can identify patterns inherent in the data without requiring extra information. However, if we have labeled data, we would like to see a classifier maximize the number of correct classifications. This corresponds to a change in objective function, encapsulated by Neighborhood Component Analysis.

2.1.4 Neighborhood Component Analysis

Neighborhood Component Analysis (NCA) [22] is a supervised version of PCA. The principle is to construct a mapping into a space where the classifier of choice for a problem performs optimally on a labelled data set. PCA maximizes the variance within a data set and remaps to a space where that variation is best separated; NCA instead attempts to learn a mapping into a space where we maximize the number of correct classifications in a data set. To accomplish this, the classifications of the data must be known, thus making this a supervised learning method.

Data is classified by mapping a new data point using a Mahalanobis matrix transformation $\mathbf{A}$ that maps the data into a space where k-nearest neighbors (kNN) performs the classification task best, thus maximizing the probability that a given point is classified correctly [22]. A Mahalanobis distance is a generalized quadratic distance defined by a matrix $\mathbf{Q}$, measured

between two points $\mathbf{x}$ and $\mathbf{y}$ such that

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

Here $\mathbf{Q}$ is a symmetric non-negative definite matrix; i.e., $\mathbf{Q} = \mathbf{A}^T \mathbf{A}$ for some $\mathbf{A}$ [48, 59]. In the NCA context, this metric measures the dissimilarity between two points belonging to the same class. Because of the definition of $\mathbf{Q}$, NCA functionally learns a distance metric $d(\mathbf{x}, \mathbf{y}) = (\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{y})^T (\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{y})$ [22]. This $\mathbf{A}$ matrix is what we employ in the definitions of NCA and its variants.

We define our data set as $\{\mathbf{x}_i\}_{i=1}^N$, where N is the number of observations, and each $\mathbf{x}_i$ has a corresponding classification C_i, such as True/False or 0/1 in the XOR example. NCA maximizes the function

$$J(\mathbf{A}) = \sum_i^N p_i = \sum_i^N \sum_{j \in C_i} p_{ij}, \tag{2.3}$$

where the value p_{ij} is defined by

$$p_{ij} = \frac{\exp\left(-||\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_j||_2^2\right)}{\sum_{k \neq i} \exp\left(-||\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_k||_2^2\right)}. \tag{2.4}$$

The value of p_{ij} can be interpreted as the probability that point $\mathbf{x}_i$ will be classified using the point $\mathbf{x}_j$. The sum p_i is the sum over the probability that point $\mathbf{x}_j$ is used to classify point $\mathbf{x}_i$ for all $\mathbf{x}_j$ in the same class as $\mathbf{x}_i$; i.e., is the total probability that a neighbor in the correct class is used to classify $\mathbf{x}_i$. The value of p_i is always between zero and one, indicating that it is, indeed, a valid probability measure [22]. Then the total over all p_i is $J(\mathbf{A})$, the number of points classified correctly across the data set. The gradient, $\frac{\partial J}{\partial \mathbf{A}}$, is

$$\frac{\partial J}{\partial \mathbf{A}} = 2\mathbf{A} \sum_i \left(p_i \sum_k p_{ik} \mathbf{x}_{ik} \mathbf{x}_{ik}^T - \sum_{j \in C_i} p_{ij} \mathbf{x}_{ij} \mathbf{x}_{ij}^T \right), \tag{2.5}$$

where $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$.

The p_{ij} function is relatively smooth and both the function and derivative are defined almost everywhere. The cost functional $J(A)$ can thus be optimized using gradient descent methods with the analytic gradient in Equation (2.5) or with other optimization routines detailed in Chapter 3.

After $\mathbf{A}$ is computed via optimization methods using a set of training points, a new point can be classified by using the learned $\mathbf{A}$ to map into the transformed space, and subsequently using kNN in the $\mathbf{A}$-space to compare it with previously classified points.

Memory/storage can become an issue as the gradient matrix dimension grows. The gradients can quickly become unfeasible to calculate and store, even with data vectors as small as one hundred components. Additionally, the values of p_i take into account all other points in the

data set, and neighborhoods for each point are recalculated at each step. This adds significant computational expense with limited ability to parallelize the algorithm.

Using the example XOR problem, we start by calculating the p_{ij} values for each of the four points. In the NCA case, we consider all neighbors of a given point when calculating the p_{ij} values with Equation (2.4). To give a consistent comparison with the PCA example, we again center the data, although this is not required in NCA. We then have the data shown in Table 2.2. Centering does not affect the classifications.

For a simple example, let $\mathbf{A}$ be the identity for our first step, so we can use the points directly to calculate all relevant parts. Using NCA definitions, we calculate all d_{ij}^2 values,

$$\begin{aligned} d_{12}^2 &= ||\mathbf{x_1} - \mathbf{x_2}||_2^2 = 2, \\ d_{13}^2 &= ||\mathbf{x_1} - \mathbf{x_3}||_2^2 = 1, \\ d_{14}^2 &= ||\mathbf{x_1} - \mathbf{x_4}||_2^2 = 1, \end{aligned}$$

where the bold face indicates the vector $\mathbf{x}_i = (x_1, x_2)_i$ with index i. Once we have computed all d_{ij}^2 values, we can calculate all p_{ij} values. Again, we show only those for $i = 1$ for brevity and note that

$$\begin{aligned} p_{12} &= \frac{\exp(-d_{12}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} = \frac{\exp(-2)}{\exp(-2) + \exp(-1) + \exp(-1)} = 0.1554, \\ p_{13} &= \frac{\exp(-d_{13}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} = \frac{\exp(-1)}{\exp(-2) + \exp(-1) + \exp(-1)} = 0.4223, \\ p_{14} &= \frac{\exp(-d_{14}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} = \frac{\exp(-1)}{\exp(-2) + \exp(-1) + \exp(-1)} = 0.4223. \end{aligned}$$

The values of p_i are then

$$\begin{aligned} p_1 &= 0.1554, \\ p_2 &= 0.1554, \\ p_3 &= 0.1554, \\ p_4 &= 0.1554. \end{aligned}$$

Because of the symmetry in our original problem, all are equal at this first step. Here $J(\mathbf{A}) = 4 \times 0.1554 = 0.6216$. In the NCA formulation, this $J(\mathbf{A})$ quantity is the number of points that choose a point in their same class as a neighbor, or equivalently, the number of points that are classified correctly. Calculating the gradient, g for $\mathbf{A}$ as the identity yields

$$g(\mathbf{A}) = \begin{bmatrix} -0.525 & 0 \\ 0 & -0.525 \end{bmatrix}.$$

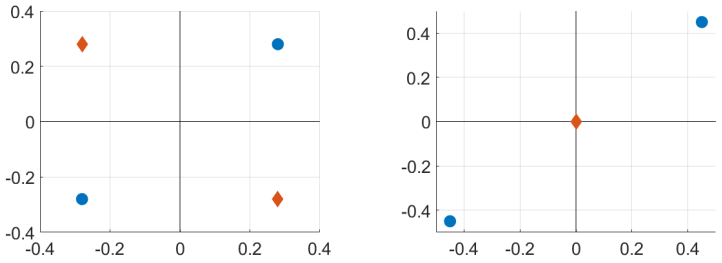


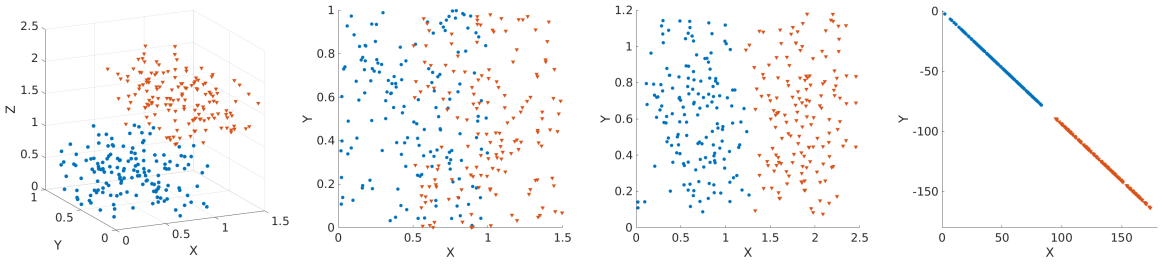
Figure 2.3: Final optimization mapping for two different initial guesses: (a) $\mathbf{A_0}$ = identity matrix and (b) $\mathbf{A_0}$ = PCA of the original data.

The step will vary based on the employed optimization method but, for simplicity, we employ steepest descent for this example, with $\lambda = 1$. For more information on steepest descent and other optimization routines, see Chapter 3. Our new matrix is then

$$\begin{aligned}\mathbf{A}_{i+1} &= \mathbf{A}_i - \nabla J(\mathbf{A}_i) \\ &= \begin{bmatrix} 1.525 & 0 \\ 0 & 1.525 \end{bmatrix},\end{aligned}$$

and we evaluate $J(\mathbf{A}_{i+1})$ and $J(\mathbf{A}_{i+1})$ again to determine the next step. Eventually, we will arrive at some $\mathbf{A}^*$ where $J(\mathbf{A})$ is maximized. We note that NCA is very sensitive to initial values as demonstrated by the results shown in Figure 2.3. The $\mathbf{A}^*$ that maximizes $J(\mathbf{A})$ for initial value of the identity is scaled down, but is otherwise the same as the original XOR mapping. The $\mathbf{A}^*$ mapping from an initial value of all ones, however, maps both $[0, 1]^T$ and $[1, 0]^T$ to the origin while keeping the other two points further away. These are clearly two very different final mappings and it is unclear which to choose as the final $\mathbf{A}^*$ choice. With $\mathbf{A}_0$ as the identity, NCA performs no better than PCA. With $\mathbf{A}_0$ as the matrix of all ones, the final mapping sends the two points with class 0 both to the origin, marginally improving our overall classification result as compared with PCA. This dependence on initial values remains a problem wherever this method is employed. Two common starting points are either the standard $\mathbb{R}^k$ basis for a $k-$dimensional subspace, or the PCA mapping of the data into a $k-$dimensional subspace.

We illustrate a different and more complex example problem in Figure 2.4 to demonstrate the power of NCA when separating larger data sets [17]. The goal is to map a 3-D set of data to 2-D. The original 3-D data is comprised of 300 points belonging to two different classes. The x, y, and z coordinates of the original data are individually sampled from uniform distributions. In the x direction, the points are sampled from the uniform distribution between 0.5 and 1.5, in the y direction, the distribution is between 0 and 1, and in the z direction, it's between 1.1 and 2.1. The original data, the projection into XY plane, and the learned projection are all shown in Figure 2.4. The PCA mapping for this data is also shown in Figure 2.4 for comparison between the two methods. Whereas both the NCA and PCA mappings separate the two classes



(a) (b) (c) (d)

Figure 2.4: Binary example problem for NCA where blue circles and orange triangles define membership in the two classes: (a) original data, (b) 2-D mapping, (c) PCA mapping, (d) NCA mapping.

within this example, they are notably not the same mapping. The scale of the NCA mapping is much different than that of PCA, it separates the two classes without overlap, and the data in the transformed space is linearly separable. The PCA mapping separates the classes, but there is still some minor overlap in the middle, and the data is still not linearly separable. This is not to say that NCA will always find a mapping that is linearly separable; however, the difference in objective function between PCA and NCA leads to noticeably improved classification here. In classification problems, therefore, the NCA-learned mapping is often the better choice when data labels are available.

2.1.5 Fast Neighborhood Component Analysis

The major difference between Fast Neighborhood Component Analysis (FNCA) [60] and traditional NCA is a restriction in the neighbors considered within the calculation of each p_i and p_{ij}. This enables computational speedup in the algorithm without sacrificing the NCA framework. The value p_{ij} is now calculated as

$$p_{ij} = \frac{\exp(-||\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_j||_2)}{\sum_{k \in M_i \bigcup H_i} \exp(-||\mathbf{A}\mathbf{x}_i - \mathbf{A}\mathbf{x}_k||_2)},$$

where M_i is the set of k neighbors in the same class as point x_i and H_i is the set of k nearest neighbors in all other classes. The definition for p_i is also changed by this formulation and is now

$$p_i = \sum_{j \in M_i} p_{ij}.$$

Note that there is no square on the norms in this formulation. This permits more equal weighting of the influence of the neighbor nearest a given point versus those farther away, and may help avoid overfitting [60].

In the simple 300 point classification problem from Figure 2.4, this minor change in formulation, and the similar change in p_i's definition to sum over only the $j \in M_i$, yielded a nearly 3-fold decrease in timing. On our sample problem with 300 3-dimensional points, it went from a 230 second optimization using traditional NCA to an 80 second optimization.

2.1.6 Bayesian Neighborhood Component Analysis

Bayesian Neighborhood Component Analysis (BNCA) [54] is formulated similarly to the original NCA, with the same distance definition. However, in BNCA, the goal is to learn a distribution corresponding to the mapping $\mathbf{A}$. By doing so, the BNCA framework enables the estimation not only of a new point's classification, but also the associated distribution.

The objective function to be maximized is now a posterior estimation distribution,

$$J(\mathbf{A}) = \prod_i P(\mathbf{Y}|\mathbf{X}, \mathbf{A}) p(\mathbf{A}), \tag{2.6}$$

where $P(\mathbf{Y}|\mathbf{X}, \mathbf{A})$ is the likelihood and $p(\mathbf{A})$ is the prior distribution. The prior incorporates any pre-existing knowledge about our data set, whereas the likelihood is the probability of a variable taking a certain value given observed data.

To perform this maximization, we need a formulation for each term within this objective function. BNCA learns an $\mathbf{A}$ under a set of assumptions that permits estimation of both the prior and likelihood, and thus the posterior. Note that this method will still achieve the same goal as NCA and FNCA; we're still maximizing the probability that a point $\mathbf{x}_i$ is closer to a point in its same class after transformation by the matrix $\mathbf{A}$. In the BNCA case, however, we are additionally fitting a distribution to the class memberships, providing probabilistic classifications for each data point.

To reformulate this problem in the already established NCA framework, define

$$P(y_i = k|\mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{A}) = \frac{\sum\limits_{j \in N_i} 1\{y_j = k\} \exp(-d^2_{\mathbf{A}^{ij}})}{\sum\limits_{t \in N_i} \exp(-d^2_{\mathbf{A}^{ij}})},$$

where $\{y_i = k\}$ is the equivalence function defined by

$$\{y_i = k\} = \begin{cases} 1, & y_i = \text{Class}(x_i) = k, \\ 0, & \text{otherwise} \end{cases}$$

and the distance $d_{A^{ij}}$ is defined by

$$d^2_{\mathbf{A}^{ij}} = (\mathbf{x_i} - \mathbf{x_j})^T \mathbf{A} (\mathbf{x_i} - \mathbf{x_j}).$$

Note that with this distance formulation, $\mathbf{A}$ is actually equal to the $\mathbf{Q}$ matrix learned in the

original NCA formulation, where $\mathbf{Q}$ is symmetric positive semi-definite. The likelihood is then

$$P(\mathbf{Y}|\mathbf{X},\mathbf{A}) = \frac{1}{Z(\mathbf{A})}\prod_i P(y_i|\mathbf{x}_i,\mathbf{Y}_{N_i},\mathbf{X}_{N_i},\mathbf{A}),$$

where $Z(\mathbf{A})$ is a normalizing constant.

As in NCA, the principle is to learn a Mahalanobis metric to maximize the number of points classified correctly, while estimating the associated distributions. The BNCA approach starts by approximating the matrix $\mathbf{A}$ as a linear combination of the eigenvectors of the data in a manner similar to PCA.

We define this approximation as

$$\mathbf{A} = \sum_{\ell=1}^{b} \gamma_\ell \mathbf{v}_\ell \mathbf{v}_\ell^T,$$

where $\mathbf{v}_\ell$ the ℓ^{th} eigenvector of our data matrix $\mathbf{X}\mathbf{X}^T$, γ_l are the coefficients in the linear combination, and b denotes the number of eigenvectors used in the approximation. Note that $\boldsymbol{\gamma}$ is a column vector of γ_ℓ values, but each $\mathbf{v}_\ell$ is a column vector. Our probability can then be expressed as

$$P(y_i = k|\mathbf{x}_i,\mathbf{Y}_{N_i},\mathbf{X}_{N_i},\mathbf{A}) = \frac{\sum_{j\in N_i} 1\{y_j = k\}\exp\left[-(\mathbf{x}_i-\mathbf{x}_j)^T\sum_{\ell=1}^{b}\gamma_\ell\mathbf{v}_\ell\mathbf{v}_\ell^T(\mathbf{x}_i-\mathbf{x}_j)\right]}{\sum_{t\in N_i}\exp\left[-(\mathbf{x}_i-\mathbf{x}_t)^T\sum_{\ell=1}^{b}\gamma_\ell\mathbf{v}_\ell\mathbf{v}_\ell^T(\mathbf{x}_i-\mathbf{x}_t)\right]} \tag{2.7}$$

$$= \frac{\sum_{j\in N_i} 1\{y_j = k\}\exp\left(-d^2_{\gamma^{ij}}\right)}{\sum_{t\in N_i}\exp\left(-d^2_{\gamma^{it}}\right)} \tag{2.8}$$

$$= P(y_i = k|\mathbf{x}_i,\mathbf{Y}_{N_i},\mathbf{X}_{N_i},\boldsymbol{\gamma}). \tag{2.9}$$

Here we define $d^2_{\gamma^{ij}} = d^2_{\mathbf{A}^{ij}} = \boldsymbol{\gamma}^T\mathbf{w}_{ij}$ with $\boldsymbol{\gamma} = [\gamma_1,\ldots,\gamma_b]^T, w^\ell_{ij} = (\mathbf{v}_\ell^T(\mathbf{x}_i-\mathbf{x}_j))^2$, and $\mathbf{w}_{ij} = [w^1_{ij},\ldots,w^b_{ij}]^T$. The distance function is now denoted $d^2_{\boldsymbol{\gamma}^{ij}}$ for any points $\mathbf{x_i},\mathbf{x_j}$, though this is purely notational and designed to indicate that we are estimating $\boldsymbol{\gamma}$ rather than $\mathbf{A}$ directly.

To compute the posterior distribution of $\mathbf{A}$, we need only compute the posterior distribution for $\boldsymbol{\gamma}$. Assume that $p(\boldsymbol{\gamma}) = N(\boldsymbol{\gamma}|\mathbf{m}_0,\mathbf{V}_0)$; i.e., that $\boldsymbol{\gamma}$ is normally distributed with mean $\mathbf{m}_0$ and covariance $\mathbf{V}_0$ [54]. The next step is to approximate the posterior distribution of $\boldsymbol{\gamma}$ using a variational approximation. Variational distributions for $\boldsymbol{\gamma}$ define a lower bound that is maximized to obtain the approximate estimate for the posterior [54]. Because we consider $Z(A)$ as a normalizing constant, it will be omitted from the following discussion. Taking into account

all N data points and all K classes, the log-likelihood, ℓ, is then

$$L = \sum_{i}^{N} \sum_{k}^{K} 1\{y_i = k\} \log\{p(y_i = k|\mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \boldsymbol{\gamma})\}.$$

The summations take all points and classes into account, while $\{y_i = k\}$ term ensures the inner sum over possible classes only considers one class designation k at a time.

Employing the definition of $p(y_i = k|\mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \boldsymbol{\gamma})$ and using the fact that $\log(a+b) > \log(a) + \log(b)$ if $0 < a, b < 1$, we obtain

$$\begin{aligned}
L &> \sum_{i} \sum_{k} 1\{y_i = k\} \left[\sum_{j \in N_i} \log\{y_j = k\} \exp\left(-d^2_{\gamma^{ij}}\right) - \log \sum_{t \in N_i} \exp\left(d^2_{\gamma^{it}}\right) \right] \\
&> \sum_{i} \sum_{j \in N_i} y_{ij} \log \left(\frac{\exp(-d^2_{\gamma^{ij}})}{\sum\limits_{t \in N_i} \exp(-d^2_{\gamma^{it}})} \right) \\
&> \sum_{i} \sum_{j \in N_i} y_{ij} \log \left(\frac{1}{1 + \sum\limits_{t \in N_i} \exp(d^2_{\gamma^{ij}} - d^2_{\gamma^{it}})} \right).
\end{aligned}$$

We then employ the definition of the log-sum-exp function, $lse(\eta_{ij}) = \log\left(1 + \sum\limits_{t \in N_i} \exp(\eta^t_{ij})\right)$, where we define $\eta^t_{ij} = (\mathbf{w}_{ij} - \mathbf{w}_{it})^T \boldsymbol{\gamma}$. It follows that

$$L > -\sum_{i} \sum_{j \in N_i} y_{ij} lse(\eta_{ij}),$$

where $y_{ij} = \delta_{ij}$ is the Kronecker delta [54].

Bohning's quadratic bound [6] applied to the lse function yields

$$lse(\eta) \leq \frac{1}{2} \eta^T \mathbf{A} \eta - b^T_{\psi} \eta + c_{\psi},$$

where

$$\begin{aligned}
A &= \frac{1}{2}(\mathbf{I}_M - \frac{1}{M+1} \mathbf{1}_M \mathbf{1}^T_M), \\
b_{\psi} &= A\psi - g(\psi), \\
c_{\psi} &= \frac{1}{2} \psi^T A \psi - g(\psi)^T \psi + lse(\psi).
\end{aligned}$$

Here M indicates the size of our matrices.

Let N_{i_κ} denote the κ^{th} nearest neighbor of point x_i, and let ψ_{ij} be the variational parameter to be updated during training. Applying this lse bound formula to L, we obtain

$$L > \sum_i \sum_{j \in N_i} y_{ij} \left(-\frac{1}{2} \boldsymbol{\gamma}^T \mathbf{W}_i^j \mathbf{H} (\mathbf{W}_i^j)^T \boldsymbol{\gamma} + b_{ij}^T (\mathbf{W}_i^j)^T \boldsymbol{\gamma} - c_{ij} \right),$$

where

$$\begin{aligned}
\mathbf{W}_i^j &= [\mathbf{w}_{ij} - \mathbf{w}_{iN_{i_1}}, \ldots, \mathbf{w}_{ij} - \mathbf{w}_{iN_{i_\kappa}}], \\
\mathbf{H} &= \frac{1}{2} \left[\mathbf{I}_\kappa - \frac{1}{\kappa + 1} \mathbf{1}_\kappa \mathbf{1}_\kappa^T \right], \\
b_{ij} &= \mathbf{H} \psi_{ij} - g(\psi_{ij}), \\
g(\psi_{ij}) &= \exp(\psi_{ij} - lse(\psi_{ij})).
\end{aligned}$$

Here c_{ij} is a constant, κ is the number of nearest neighbors, $\mathbf{I}_\kappa$ is the identity of size $\kappa \times \kappa$, and $\mathbf{1}_\kappa$ is a length κ vector of ones.

Using the prior distribution $\mathcal{N}(\gamma | \mathbf{m}_0, \mathbf{V}_0)$ and the likelihood estimate lower bound from the Bohning's quadratic bound step, we can directly compute the posterior distribution of $\boldsymbol{\gamma}$, $N(\boldsymbol{\gamma} | \mathbf{m}_T, \mathbf{V}_T)$ where

$$\begin{aligned}
\mathbf{V}_T &= \left[\mathbf{V}_0^{-1} + \sum_i \sum_{j \in N_i} y_{ij} \mathbf{W}_i^j \mathbf{H} (\mathbf{W}_i^j)^T \right]^{-1} \\
\mathbf{m}_T &= \mathbf{V}_T \left(\mathbf{V}_0^{-1} \mathbf{m}_0 + \sum_i \sum_{j \in N_i} y_{ij} \mathbf{W}_i^j \mathbf{b}_{ij} \right).
\end{aligned}$$

The variational parameter is updated by the computation

$$\psi_{ij} = (\mathbf{W}_i^j)^T \mathbf{m}_T. \tag{2.10}$$

This update process is repeated to self-consistency.

The prediction for a new point $\mathbf{x}_i$ can be found with the estimation of $p(y_i | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X})$ using Markov Chain Monte Carlo (MCMC) methods, summarized in Appendix A, such that the continuous distribution is approximated by

$$p(y_i | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}) = \int_\gamma p(y_i | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \boldsymbol{\gamma}) q(\boldsymbol{\gamma}) d\boldsymbol{\gamma} \approx \frac{1}{T} \sum_{t=1}^{T} p(y_i | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \gamma_t).$$

Here $\boldsymbol{\gamma}_t$ is a set of T points sampled from our constructed Gaussian distribution of $q(\boldsymbol{\gamma}) = \mathcal{N}(\boldsymbol{\gamma} | \mathbf{m}_T, \mathbf{V}_T)$ that approximates the continuous distribution. Note that whereas $q(\boldsymbol{\gamma})$ is a Gaussian, $p(y_i | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \boldsymbol{\gamma})$ is a multivariate normal distribution.

There is some capability to incorporate the FNCA speedup in the restriction of the neighborhoods N_i in the calculation of p_{ij} throughout the BNCA framework. The N_i restriction in both numerator and denominator allows for an implementation similar to FNCA; for any

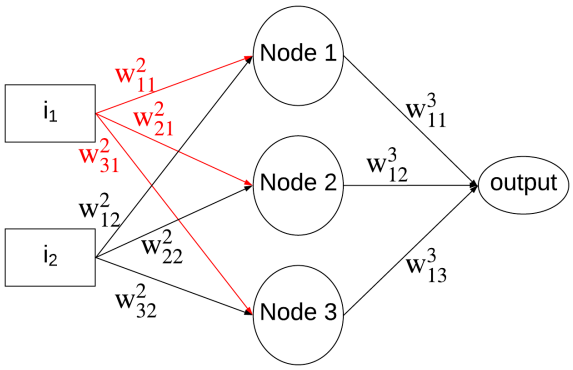


Figure 2.5: Neural network for the XOR example.

point $\mathbf{x}_i$, the corresponding calculation of p_i is limited to looking at points in a user-defined neighborhood.

2.1.7 Neural Networks

Neural networks (NN) can be motivated with our exclusive or (XOR) problem. A neural network does not require data to be linearly separable, making it ideal for this problem.

Consider the network depicted in Figure 2.5, comprised of 2 input nodes, one hidden layer with three nodes, and one output node. The input is one of the four possible combinations of x_1 and x_2, and the output is in the interval [0,1]. For a layer ℓ, the weight at node j acting on output from node k is denoted by w_{jk}^{ℓ}. In this network, $\ell = 1, 2$ and 3. The input is layer $\ell = 1$, the hidden layer is $\ell = 2$ and the output is $\ell = 3$. The weights are indexed accordingly.

The final output of the network can be expressed as

$$y = \sigma\left[w_{11}^3\sigma\left(i_1w_{11}^2 + i_2w_{12}^2\right) + w_{12}^3\sigma\left(i_1w_{21}^2 + i_2w_{22}^2\right) + w_{13}^3\sigma\left(i_1w_{31}^2 + i_2w_{32}^2\right)\right], \tag{2.11}$$

where i denotes the inputs component, σ denotes an activation function, and w denotes the weights, all as indexed in the image. In this example, we take the activation function to be the sigmoid function,

$$\sigma(v_i) = \frac{1}{1 + e^{-v_i}}. \tag{2.12}$$

Other choices include the the Rectified Linear Unit (ReLU),

$$\sigma(v_i) = \max(0, v_i) \tag{2.13}$$

and the softmax function,

$$\sigma(v_i) = \frac{e^{v_i}}{\sum\limits_{k=1} e^{v_k}}, \tag{2.14}$$

shown in Figure 2.6. In this problem, we want a value strictly between 0 and 1, with the output

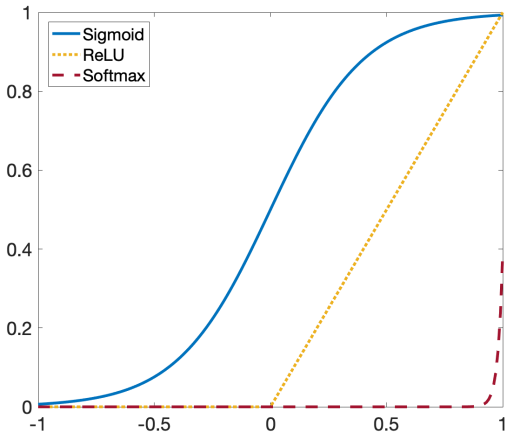


Figure 2.6: Common NN activation functions.

favoring 0 or 1 rather then the values in between, so we use the sigmoid function bounded to the $[0, 1]$ interval.

The randomly initialized network for the XOR example is shown in Figure 2.7 with the input (1,1), and the output calculated with Equation (2.11) is 0.6785. Because our data set is labeled, we know the output should be 0, making the error $e = d - y = 0.6785$, where d is the target output and y is the network output.

For simplicity, we demonstrate here a gradient descent update where the objective function to minimize is the $e = d - y$ function, with y as the network output and d defined as the target output. We provide a detailed discussion of gradient descent and other optimization methods in Chapter 3.

The weight update for gradient descent defines the update such that

$$w_{i+1} = w_i + \lambda \nabla e_{w_i},$$

where e_{w_i} is the function $e = d - y$ evaluated with the weight w_i.

Back propagation, covered in Section 2.2, is the process of propagating the error e back through the network so that the weight update can be defined for every node and weight in the network. Here, we implement the process outlined in Section 2.2 with a gradient descent step with objective function $e = d - y$. A single step yields the weights shown in Figure 2.8. The final trained network is shown in Figure 2.9.

If the input, (1,1), was the only input vector employed for training, the weights would be updated using $e = d - y$. If a second vector was used for training, the update would not directly involve this error, but would involve some combination of the errors across the output for all our training data, and this quantity is what the training algorithm would attempt to minimize.

For all considered binary classification, the cross-entropy cost functional,

$$J = \sum_{i=1}^{M} \left[-d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i) \right], \tag{2.15}$$

where M is the number of training vectors, replaces e when more than one training vector is present. Network weights are updated via back-propagation of this error until the performance reaches a desired goal; i.e., cost functional is minimized. Chapter 3 provides other cost functional examples as well as other optimization updates.

Note that this network has been trained only on the point (1,1), and has learned that classification very well, with an output of 3.5×10^{-4} instead of the initial value of 0.6785. However, if we test the other XOR points with this network, we obtain outputs of 0.0117 for (0,0), 0.0012 for (1,0) and 9.6×10^{-4} for (0,1). This is an example of *overfitting* in the sense that the network has been trained so specifically for the point (1,1) that it cannot generalize to any other data. There are many ways to avoid overfitting, but the technique most commonly implemented in machine learning is *cross-validation* [5]. This is the process involves removing a

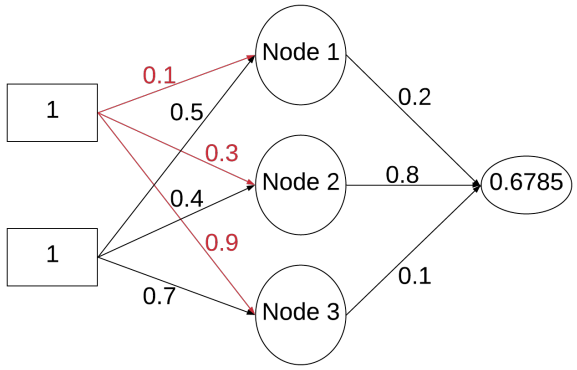


Figure 2.7: Initialized network.

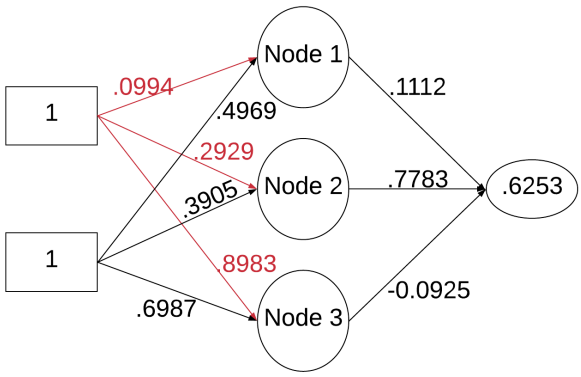


Figure 2.8: Updated network after one full training iteration.

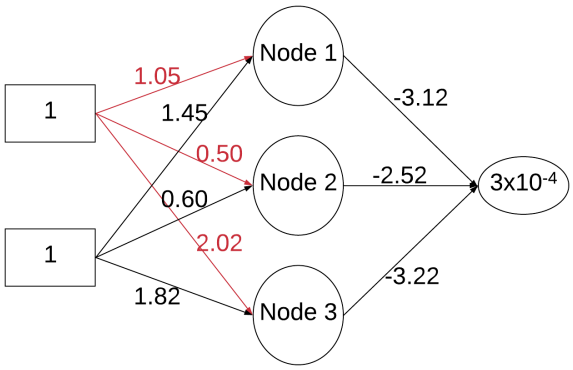


Figure 2.9: Final updated network after training.

subset of training data, called a validation set, for out-of-sample testing throughout the training process to ensure the network is still able to generalize to unseen data. In this simple example, the training data only contained one point, so there was no validation set, but cross-validation is extremely important in more realistic examples such as our motivating nuclear application.

2.1.8 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a variation of a neural network in which there are two stages preceding the actual neural network whose objective is to reduce the number of parameters learned in the training stage. As a definition, CNNs are neural networks that "use convolution in place of general matrix multiplication in at least one of their layers" [23]. Through convolution and pooling layers, CNNs can perform both feature identification and dimension reduction within the NN architecture.

Historically, CNNs derive much of their characteristic elements from neuroscience, specifically from studies of the primary visual cortex and data processing in mammal brains [23]. The efficiency and accuracy of this model was subsequently demonstrated repeatedly through many applications. CNNs were some of the first and best deep models; i.e., nets with more than one hidden layer, that were both commercially viable and performed well [23]. The dimension reduction of the convolution and pooling layers permits application in situations where a fully connected neural network would be prohibitively expensive, making it a standard method for large data problems in image processing/analysis and other applications [25]. In our application, we have a relatively small number of training points at just 289 fuel rod images, but each training point is an image of 23×23 pixels. These images are large enough that a fully connected neural net is prohibitive, but the NN architecture could still provide powerful results. Thus, we turn to CNNs for the reasons listed above.

The general structure of a CNN is as follows: first, we have a convolution layer, followed by an activation layer, and then a pooling layer. Finally, the output of the pooling layer is sent to a traditional fully connected neural net for final classification. An overview is depicted in

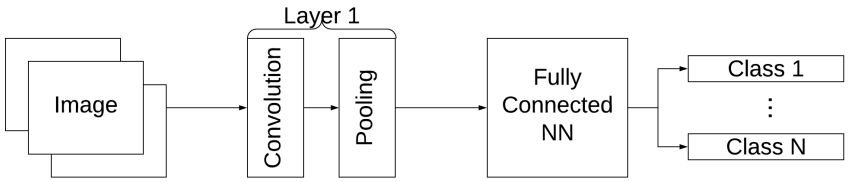


Figure 2.10: Convolutional network structure.

Figure 2.10.

The first step in a CNN is the convolution itself. A convolution is similar to a weighted average of some function, where the weight is also a function. As an example, if the most recent data in a measurement scheme is most relevant, the weighted average should weight the older data less heavily than the new. However, in this case, both the data–function output–and the weights in the average are functions of time. The smoothed weighted average can be expressed

$$s(t) = \int x(a)w(t-a)da,$$

where a is the age $a <$ time t of the measurement, $x(t)$ is the time-dependent function output, and $w(t)$ is the weight function [23]. This is the convolution of the functions x and w, expressed as

$$s(t) = (x * w)(t).$$

The *input*, which is our function x, is convolved with the *kernel* $w(t)$ to generate the *feature map* $s(t)$ [23]. In the discrete case, the convolution becomes

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

In the case of two dimensions, this takes the form

$$\mathbf{S}(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\mathbf{K}(i-m, j-n), \tag{2.16}$$

where $\mathbf{I}$ is the input matrix–often an image–and $\mathbf{K}$ is the kernel [23].

The next step sends the feature maps from the convolution through a nonlinear activation stage. In this stage, a particular neuron 'fires' based on the output from its activation function, similar to the neural network activation. Typically, the activation function used at this stage is a rectified linear activation function,

$$\sigma(x) = \max(0, x),$$

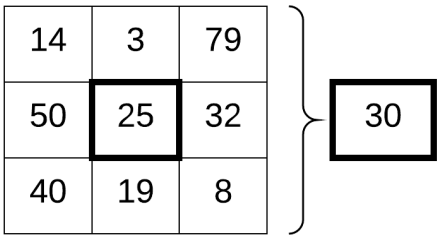


Figure 2.11: 3×3 mean pooling. The bold box in the center of the left indicates the pixel under consideration, the outer box indicates all the pixels pooled, and the bold box on the right hand side is the final result from the mean pooling.

which avoids vanishing gradient issues in gradient-based back-propagation methods [25].

The third step in the convolutional neural network is the pooling layer. In this layer, the output of the convolution/activation stages is modified again before being sent to a standard fully connected NN for final classification. Note that there can be several layers of convolution/activation/pooling before being sent to a standard NN. The pooling function replaces net output from a particular neuron with a summary of the outputs from a small region of the net [23, 25]. Max/Min pooling reports the max/min output for a rectangular neighborhood in the net. Mean pooling reports the mean of a rectangular neighborhood, and is shown in Figure 2.11. One advantage of pooling is that it makes the representation approximately invariant to small translations of the input [23]. This is particularly useful in the case of small amounts of noise in data.

Once the data is sent through the convolution and pooling layers, it is fed into a standard neural network. The network is trained according to the standard back propagation method outlined in the neural net section, Section 2.1.7, and more in depth in Section 2.2.

In our XOR example, there is no real advantage to convolution and pooling. To demonstrate the power of the two stages prior to the neural network, we turn to our chosen application for this work, spent fuel defect detection. This is an image classification problem, and image input vectors to neural networks can quickly get very large. In our spent fuel data sets, we consider 23 $\times$ 23 pixel images, making our input vector 529 components long. In a fully connected neural network, this implies at least one input node for each of the 529 components. To simplify the algebra, we assume a fully connected neural network with a 529 node input layer and a single hidden layer with 2160 nodes. Fully connected simply means that there is a connection between every node in one layer to the next layer. This means that we have $2160 \times 23 \times 23 = 1,142,640$ parameters [25]. Given our training data size of 289, the estimation problem is hugely underdetermined. Taking advantage of the CNN characteristics, we start by rearranging the nodes into equally sized blocks: 15 12 $\times$ 12 blocks [25]. Making the assumption that pixels are correlated only to nearby pixels, we can restrict the 'receptive field' of each neuron to a 5 $\times$ 5 region within the image [25]. The number of parameters needed to complete our network is now

$5 \times 5 \times 15 \times 12 \times 12 = 54,000$. While still large, this is a 95% reduction in the number of parameters as compared to the original fully connected neural network.

The parameters of the convolution and pooling layers themselves are user-defined. In our application, based on image size, we implement a 5×5 convolution with 3×3 pooling. To illustrate the machine learning aspect in the CNN setup, we formulate the convolution output using the block and receptive field sizes used in our application. The output from the convolution and activation layers for a given 12×12 block is defined as $f(p, q)$ such that

$$f(p,q) = \sigma\left(\sum_{i=0}^{4}\sum_{j=0}^{4} \mathbf{X}(p+i, q+j) w_{ij}\right) \quad \forall\, p, q \in [0, 11]. \tag{2.17}$$

Here $\mathbf{X}$ is the data matrix, p, q indicate the location of a neuron in the 12×12 block and σ is the activation function, indicating the activation step acting on the convolution result [25]. The weights, w_{ij}, are learned via back propagation outlined in Section 2.2), as are the weights in the traditional neural net that follows the convolution/activation/pooling stages.

2.2 Back Propagation

To go more in depth regarding the back propagation process, we first define the general setup of a neural network. With the XOR problem, we ignored bias, but for completeness, we define it for the general framework. Rather than simply weighting node inputs; e.g., $w_{11}^2 i_1 + w_{12}^2 i_2$ as input to Node 1 in the XOR example, we can both weight and shift the inputs; e.g., $w_{11}^2 i_1 + w_{12}^2 i_2 + b$. This shift b is the *bias*. The bias can be a constant added across all nodes equally or a vector. Here, we denote the bias for a given layer ℓ as $\mathbf{b}^\ell$.

For a general network framework, we follow the outline and derivations in [28]. Denote the layers of a network with $\ell = 1, \ldots, L$, where $\ell = 1$ is the input and $\ell = L$ is the output. Then, let n_ℓ be the number of neurons in layer ℓ. The input dimension is n_1 and the output dimension is n_L. The weights at layer ℓ are denoted $\mathbf{W}^\ell$, where w_{jk}^ℓ indicates the weight that neuron j in layer ℓ applies to output from neuron k. If our input data is denoted $\mathbf{x}$, then we have

$$\begin{aligned} \mathbf{a}^1 &= \mathbf{x} \\ \mathbf{a}^\ell &= \sigma(\mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell), \end{aligned}$$

where σ is the activation function and $\mathbf{a}^\ell$ indicates the output at layer ℓ.

Now, assume a set of N training points $\{\mathbf{x}^i\}_{i=1}^N$ with N known target outputs or known classifications $\{y^i\}_{i=1}^N$. The goal is to minimize the number of incorrect classifications. This requires the definition of a cost functional, which can vary according to application. For simplicity, we

will use a simple quadratic function

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} ||y^i - a^L(\mathbf{x}^i)||_2^2,$$

where $a^L(\mathbf{x}^i)$ indicates the network output at the final layer L for the input $\mathbf{x}^i$. The goal is to learn the parameters $\mathbf{W}$ and $\mathbf{b}$ for all layers that minimize the cost functional. For now, x^i will be fixed, so we drop the dependency in our notation.

To minimize J, we take the derivatives with respect to every w_{jk}^ℓ and b_j^ℓ. Let $\mathbf{z}^\ell = \mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell$ and a_j^ℓ be the output at layer ℓ at neuron j. Then $\mathbf{a}^\ell = \sigma(\mathbf{z}^\ell)$ and $a_j^\ell = \sigma(z_j^\ell)$. Note that the cost functional only explicitly contains the final layer, $\ell = L$. Using the chain rule and component-wise definitions, we can express the partial derivatives for the final layer as

$$\begin{aligned} \frac{\partial J}{\partial w_{jk}^L} &= \frac{\partial J}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} = \sigma'(z_j^L)(a_j^L - y)(a_k^{L-1}) \\ \frac{\partial J}{\partial b_j^L} &= (a_j^L - y)\sigma'(z_j^L), \end{aligned}$$

where σ' indicates the derivative of σ with respect to $\mathbf{z}$. Note that neither the target y nor a_L are vectors in our chosen binary classification problem. Thus, the subscript for neuron at the layer $\ell = L$ is not necessary, as there is only one output. However, for ease of notation as we move to the other layers, it is included.

In the final layer derivatives, previous layers start to appear as evidenced by the a_j^L and a_k^{L-1} terms in $\frac{\partial J}{\partial w_{jk}^L}$. The a_j^L term can be written as $\sigma(z_j^{L-1})$, and we can relate the quantity $z_{\ell+1}$ to z_ℓ by

$$z_k^{\ell+1} = \sum_{k=1}^{n_\ell} w_{ks}^{\ell+1} \sigma(z_s^\ell) + b_k^{\ell+1}.$$

Here n_ℓ indicates the number of neurons in layer ℓ [28]. This implies that

$$\frac{\partial z_k^{\ell+1}}{\partial z_j^\ell} = w_{kj}^{\ell+1} \sigma'(z_j^\ell).$$

For $\ell = L - 1$, the partial derivatives are then

$$\begin{aligned} \frac{\partial J}{\partial w_{jk}^\ell} &= \left(\sum_{i=1}^{n_{\ell+1}} \sigma'(z_i^L)(a_i^L - y_i) w_{ji}^{\ell+1} \sigma'(z_j^\ell) \right) a_k^{\ell-1} \\ \frac{\partial J}{\partial b_k^\ell} &= \sum_{i=1}^{n_{\ell+1}} \sigma'(z_i^L)(a_i^L - y_i) w_{ij}^L \sigma'(z_j^\ell). \end{aligned}$$

This pattern repeats for all $2 < \ell < L$. This is summarized in Lemma 1 in [28]. Let $\circ$ denote component-wise multiplication, or the Hadamard product, such that $(x \circ y)_i = x_i y_i$ for vectors

and $(\mathbf{A} \circ \mathbf{B})_{ij} = \mathbf{A_{ij}B_{ij}}$. It then follows that

$$\begin{aligned}
&\delta^L = \sigma'(\mathbf{z}^L) \circ (\mathbf{a}^L - \mathbf{y}) \\
&\delta^\ell = \sigma'(\mathbf{z}^\ell) \circ (\mathbf{W}^{\ell+1})^T \delta^{\ell+1} && \text{for } 2 \leq \ell \leq L-1 \\
&\frac{\partial J}{\partial b_j^L} = \delta_j^\ell && \text{for } 2 \leq \ell \leq L \\
&\frac{\partial J}{\partial w_{jk}^\ell} = \delta_j^\ell \mathbf{a}_k^{\ell-1} && \text{for } 2 \leq \ell \leq L.
\end{aligned}$$

One iteration forward through the network results in an initial estimate of a_L. Using that a_L output from layer L, all the derivatives with respect to weights and biases can be calculated, moving backwards through the network. This is the *back propagation* process. The weights are then updated according to an optimization rule–see Chapter 3 for methods–using these derivatives, and the process repeats.

Chapter 3

Optimization

3.1 Optimization Methods

In machine learning, training the algorithm involves optimizing some cost functional with respect to a set of parameters. In the case of kNN, cross-validation metrics are optimized with respect to the metric and number of neighbors. With NCA and its variants, the number of correct classifications is optimized with respect to the mapping $\mathbf{A}$ to a new space. With neural networks, the chosen cost functional–cross-entropy, quadratic cost, etc.–is optimized with respect to the network weights and biases.

The choice of optimization method has a significant impact on algorithm training, since the choice of optimization algorithm not only impacts the model fit to the data, but also memory and computational time required to complete the training process. In the neural network optimization of Chapter 5, we use multiple algorithms, whereas the neighborhood component analysis-based work in Chapter 6 was originally proposed with gradient descent optimization. In the following sections, we discuss some common optimization routines, advantages and disadvantages of each, and application to machine learning problems.

For consistency, we denote the cost functional as $J(\mathbf{x})$, where $\mathbf{x}$ is vector valued, although the methods are equally applicable to scalars.

All methods presented in this section are iterative methods; i.e., they iterate through points $\mathbf{x}_0, \mathbf{x}_1, \ldots$ until they arrive at an optimal solution $\mathbf{x}^*$. In the ideal case, every iterate is nearer to the optimal solution $\mathbf{x}^*$ than the previous step. Let $\{\mathbf{x}_i\}_{i=0}^{\infty}$ be a sequence of iterative points. The rate of convergence as $\mathbf{x}_i \to \mathbf{x}^*$ is of order r if

$$||\mathbf{x}_{i+1} - \mathbf{x}^*|| \leq C||\mathbf{x}_i - \mathbf{x}^*||^r,$$

where C is a constant [35]. If $r = 1$ and $C \in (0, 1)$, the sequence $\{x_i\}_i^{\infty}$ converges linearly with factor [35]. If $r = 2$, the iterative method converges quadratically. If

$$\lim_{i \to \infty} \frac{||x_{i+1} - x^*||}{||x_i - x^*||} = 0,$$

the convergence is superlinear. These are the convergence definitions used throughout this optimization overview; see [35] for details.

3.1.1 Gradient or Steepest Descent

Steepest, or gradient descent, quantifies the direction of steepest descent for a function, stepping in that direction until a minimum is reached.

Let $J(\mathbf{x})$ be the cost functional to be minimized. For small $\delta\mathbf{x}$, the first order expansion for $J(\mathbf{x}+\delta\mathbf{x})$ is

$$J(\mathbf{x}+\Delta\mathbf{x}) = J(\mathbf{x}) + \nabla J(\mathbf{x})^T \Delta\mathbf{x},$$

where ∇J is the gradient. Cost functionals are often formulated as distance between model output and target output in machine learning. Therefore, the goal is to decrease the cost functional J, as that will drive the model output towards the target output. Algorithmically, we choose the $\Delta\mathbf{x}$ to make $J(\mathbf{x}+\Delta\mathbf{x})$ smaller than $J(\mathbf{x})$, stepping in the direction of smaller cost [28]. Equivalently, we want $\Delta\mathbf{x}$ such that $\nabla J(\mathbf{x})^T\Delta\mathbf{x}$ is minimized. The Cauchy-Schwarz inequality states that for $\mathbf{f},\mathbf{g}\in\mathbb{R}^n$, $|\mathbf{f}^T\mathbf{g}| \leq ||\mathbf{f}||_2||\mathbf{g}||_2$. In this context, this implies that the minimum of $\nabla J(\mathbf{x})^T\Delta\mathbf{x}$ is $-||\nabla J(\mathbf{x})||_2||\Delta\mathbf{x}||_2$, which is minimized when $\Delta\mathbf{x}$ is $-\nabla J(\mathbf{x})$. Therefore, to go in the direction that decreases J, $\Delta\mathbf{x}$ is taken to be $-\nabla J(\mathbf{x})$ [28].

From this, we arrive at the step update. Steepest descent iterative solvers are updated as

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda\nabla J(\mathbf{x}_i), \tag{3.1}$$

where $\mathbf{x}_{i+1}$ is the new step, $\mathbf{x}_i$ is the current step, and λ is the step size or learning rate [35]. The learning rate λ is chosen large enough to avoid stagnation, but not so large that it may step over the optimal point. Variable learning rates are frequently employed; far from an optimal point, the step size is large so as to approach the optimum quickly. Near the optimal point, the step size is small in order not to miss the optimum. For step size choices and variable step size details, see [35]. For machine learning methods, the gradient calculation is the mean of the gradient calculated across all training points [28]. Once an initial point is chosen, a new point is chosen according to this update, and the algorithm is repeated until some stopping criterion is satisfied.

One major difference between traditional optimization and machine learning optimization is the fact that we have sets of vector-valued training points to take into account as we calculate gradients [23]. Often, the cost functional actually decomposes into a sum across training points [23]. In the case of machine learning, the $\mathbf{x}$ that is being updated at each step includes all the parameters in the machine learning architecture. For example, in neural networks, $\mathbf{x}$ would be a vector of all the network weights, with the gradient evaluated across all the training data and averaged for the step update [5]. This is a *batch method*, as the steepest descent algorithm takes into account the entire data set at each step to calculate the gradient and corresponding step update [5, 28].

Recalculating the gradient at each step update for all data points is computationally expensive [28, 35]. To reduce this expense somewhat, gradient descent can also be setup as a *mini-batch method*, where multiple data points are updated at each iteration, but far fewer at a time than the whole data set, or even an *on-line method*, where the iterations update only one data point at a time. This last version is known as *stochastic, or sequential, gradient descent* [5, 23, 28]. We detail stochastic gradient descent in Section 3.1.2. There are advantages and disadvantages to each of these three variations. The batch method approaches exact gradient calculation; there is more error added in the gradient approximations of mini-batch and stochastic methods, as fewer training points are taken into account [23]. However, stochastic gradient descent accounts for data redundancy better, as the average gradient calculated in the batch descent update becomes biased with any repeated points [5, 23]. Generalization error tends to shrink with batch-size, and is often best with stochastic gradient descent. Stochastic gradient descent is also less likely to get stuck in local minima than batch gradient descent [5, 23]. However, due to the variability in gradient estimation with small batch sizes, the learning rate may need to shrink greatly to maintain stability, driving up the number of iterations required for convergence [23].

Choosing the step-size is non-trivial. If the step sizes are too large, the algorithm may overstep the minimum whereas if they are too small, the algorithm makes little to no progress [35]. One way to define the step-size is by enforcing a condition of 'sufficient decrease' in $J(\mathbf{x})$ as the algorithm progresses; i.e.,

$$J(\mathbf{x}_i - \lambda \nabla J(\mathbf{x}_i)) - J(\mathbf{x}_c) < -\alpha\lambda ||\nabla J(\mathbf{x}_c)||^2,$$

where λ is the learning rate, $\lambda = \beta^m$, $\beta \in (0,1)$, $m \geq 0$ and $\alpha \in (0,1)$ [35].

Additionally, the starting point has a large impact on convergence. Two different starting points may lead to algorithmic convergence, but may not converge to the same result [35].

Issues of slow convergence can also make steepest descent a problematic choice for optimization. In batch, mini-batch, and on-line formulation alike, convergence is slow. For a simple quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$, Kelley shows in [35] that gradient descent iterates will satisfy

$$||\mathbf{x}_{i+1}||_{\mathbf{A}} = (1 - \mathcal{O}(\kappa(\mathbf{A}))^{-2})||\mathbf{x}_k||_{\mathbf{A}}, \tag{3.2}$$

where $\kappa(\mathbf{A})$ indicates the condition number of $\mathbf{A}$ and the norm $||\mathbf{x}||_{\mathbf{A}} = \sqrt{\mathbf{x}^\mathbf{T}\mathbf{A}\mathbf{x}}$. The $\mathcal{O}()$ notation defines the growth rate of a function; i.e., for two functions $f(x)$ and $g(x)$,

$$f(x) = \mathcal{O}(g(x)) \iff f(x) < c(g(x))$$

for some constant c over all x. This definition implies gradient descent achieves only linear convergence, even for a simple model, and depends highly on the conditioning of the problem. This is similar for any function that can be locally approximated by a quadratic; convergence remains linear at best. Another concern of gradient descent methods is chatter; the gradient

update guarantees that the direction at the next step is orthogonal to the prior, which can result in an inefficient zig-zag pattern through a cost surface [23].

3.1.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is very similar to steepest descent. It is based on the same principle and first order expansion of $J(\mathbf{x} + \Delta\mathbf{x})$, with the same update,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \nabla J(\mathbf{x}_i).$$

However, steepest descent recalculates the gradient $\nabla J(\mathbf{x}_i)$ at every step and for each training point, and is thus very computationally expensive. To mitigate this, stochastic gradient descent algorithms instead calculate the gradient once for a randomly chosen training point [28]. Some integer, k is randomly sampled with replacement, to select the point $\mathbf{x}^k$ at which the gradient is calculated. The update at step i then becomes

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda \nabla J(\mathbf{x}_i^k). \tag{3.3}$$

Either at every step or some defined interval, the gradient is recalculated at a new randomly selected $\mathbf{x}^k$ [28]. This greatly decreases the computational expense associated with gradient calculations, but does not guarantee a decrease in the cost functional $J(\mathbf{x})$ at every step. Given the slow convergence of the original gradient descent method, however, this trade-off may be worth it, especially in cases where the initial iterate is far from the optimized result. Previously mentioned concerns of slow convergence in stochastic gradient descent caused by variance in gradient estimates can be partially addressed by multiple sampling passes through the data set as training progresses [23].

Again, the choice of step size has a large impact on algorithmic performance. The step size for stochastic gradient descent needs to vary and shrink because the error introduced by approximating the gradient at only a single training point never completely vanishes [23]. To guarantee SGD convergence, a sufficient condition for the learning rate λ is

$$\sum_{k=1}^{\infty} \lambda_k = \infty, \quad \sum_{k=1}^{\infty} \lambda_k^2 < \infty.$$

The learning rate is commonly implemented as a linear decay, $\lambda_k = (1 - \alpha)\lambda_0 + \alpha\lambda_\tau$, until iteration τ, with $\alpha = \frac{k}{\tau}$.

In machine learning, one of the major advantages of stochastic gradient descent is that the computational cost does not grow with the training data size. A batch size of one can offer some regularization via the introduction of noise in the learning process and reduce generalization error [23]. One of the major disadvantages of stochastic gradient descent is directly related to our chosen application in spent nuclear fuel; because the summed gradient over all training points

in standard gradient descent is replaced by the gradient approximation at a single point, SGD requires a large number of training points to iterate through, so that this gradient approximation is close to the true gradient [23]. In our application, we only have 289 training points, which is a small training set as compared to most applications that use SGD. Additionally, when a training data set is small enough that the gradient calculation required for standard GD or other methods is not prohibitive, the per iteration cost decrease of SGD can be outweighed by higher required iterations to convergence, due to its stochasticity [47].

3.1.3 Newton's method

In addition to using information from the first derivative, Newton's method also employs information from the second derivative [35].

Again, we indicate our objective function by $J(x)$, or for vector-valued $\mathbf{x}$, $J(\mathbf{x})$. If J is second-order continuously differentiable, we can approximate the objective function at a point using a second-order expansion

$$J(\mathbf{x} + \Delta\mathbf{x}) = J(\mathbf{x}) + \nabla J(\mathbf{x})^T \Delta\mathbf{x} + \frac{1}{2}(\Delta\mathbf{x})^T \nabla^2 J(\mathbf{x}) \Delta\mathbf{x}$$

about that point. Here ∇ and ∇^2 denote the gradient and Hessian, respectively [35]. To find a critical point, we take the gradient and set it equal to zero. Equivalently, we can take the gradient of the second-order expansion, rather than the original function, and truncate again at the second order. Note that

$$\nabla J(\mathbf{x} + \Delta\mathbf{x}) = \nabla J(\mathbf{x}) + \Delta\mathbf{x}^T \nabla^2 J(\mathbf{x}) = 0.$$

To observe the iterative method that develops from this step, let $\mathbf{x_i}$ be the current step and $\mathbf{x_{i+1}} = \mathbf{x_i} + \Delta\mathbf{x_i}$. Then $\Delta\mathbf{x_i} = \mathbf{x_{i+1}} - \mathbf{x_i}$, and we can rewrite our derivative equation as

$$\nabla J(\mathbf{x_{i+1}}) = \nabla J(\mathbf{x_i}) + (\mathbf{x_{i+1}} - \mathbf{x_i})^T \nabla^2 J(\mathbf{x_i}) = 0.$$

Solving this equation for $\mathbf{x_{i+1}}$, we obtain the Newton's method step update

$$\mathbf{x_{i+1}} = \mathbf{x_i} - (\nabla^2 J(\mathbf{x_i}))^{-1} \nabla J(\mathbf{x_i}).$$

In this formulation, the Newton's method update requires calculating the gradient, the Hessian and the Hessian's inverse at every point [35]. To reduce the required computational effort, it can instead be broken into two steps that are each easier to solve [35]. The first step is calculating the update, $\mathbf{d_i} = -(\nabla^2 J(\mathbf{x_i}))^{-1} \nabla J(\mathbf{x_i})$. Rather than inverting the Hessian, we can rewrite this as a linear equation,

$$\nabla^2 J(\mathbf{x_i}) \mathbf{d_i} = \nabla J(\mathbf{x_i}),$$

and solve for $\mathbf{d_i}$, which is much more computationally efficient [35]. The second step is then to calculate the update,

$$\mathbf{x_{i+1}} = \mathbf{x_i} + \mathbf{d_i}.$$

Even with this modification, Newton's method is computationally expensive. The size of the Hessian is the square of the number of parameters, and it needs to be computed for every training point at every iteration as currently formulated [23]. In neural networks, for example, even small networks have a large number of parameters, making Newton's method quickly unfeasible.

There are several ways that Newton's method can fail. The first two are in the solution for d_i in the first step. If the Hessian is large, or the gradient close to zero, this problem is ill-posed and can drastically slow the convergence of the method. The optimization routine also requires that the Hessian remain positive definite, which may not always be true in deep learning architectures [23]. Additionally, Newton's method only guarantees quadratic convergence *locally* [35]. If the initial value is very far from the true point, this is no longer guaranteed. Still, with a good initial value, this is a large increase in convergence rate over the linear gradient descent method.

Because Newton's Method tries to find a point where the gradient is zero, it can be drawn to saddle points as well as maxima in the cost functional surface [23]. Gradient descent methods focus on following the cost functional as it decreases rather than forcing the gradient to zero, making them less likely to converge to saddle points or maxima [23]. In neural networks and other deep learning architectures, there are often more saddle points than minima, making Newton's Method problematic [23].

Another issue with using Newton's method for our application involves the size of our data set. Methods based strictly on gradient updates perform well on batch sizes as small as 100 [23]. However, methods that involve the Hessian require batch sizes significantly larger; i.e., 10,000, for good performance [23]. Because our spent fuel application has limited amounts of data, with a full training set size of only 289, gradient-based methods, such as gradient descent and its variants, are more appropriate than methods that require the Hessian. However, for completeness, we include these methods for comparison.

3.1.4 Levenberg-Marquardt

The Levenberg-Marquardt (LM) method attempts to retain the convergence advantages of Newton's method and the convergence properties of the standard gradient descent method [35, 45]. The major assumption of Newton's method is that the function being optimized is locally quadratic, or can be locally approximated by a quadratic function [35]. This implies that the Hessian matrix is positive definite and the iterative solver is indeed going towards a minimum [35]. However, this breaks down if the quadratic approximation is invalid. In practice, the LM step updates using a mix of the Newton's method update and the gradient descent update

in order to rely on gradient descent methods wherever the quadratic approximation fails, and Newton's Method wherever the quadratic approximation succeeds [45]. In this way, the LM algorithm is able to take advantage of the best of both methods.

The step update proposed by Levenberg is

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (\mathbf{H} + \lambda \mathbf{I})^{-1} \nabla J(\mathbf{x}_i),$$

where $\mathbf{H}$ is the Hessian matrix of our objective function $J(\mathbf{x})$ [40, 45]. If λ approaches 0, this becomes the Newton's method update, but if λ is significantly larger than H, this becomes the gradient descent update. However, if λ is too large, the information gain from using H is minimal. Marquardt added a second piece to this, where the identity is replaced with the diagonal of the Hessian, defining the update

$$\mathbf{x}_{i+1} = \mathbf{x}_i - (\mathbf{H} + \lambda \text{diag}\mathbf{H})^{-1} \nabla J(\mathbf{x}_i), \tag{3.4}$$

which is now known as the Levenberg-Marquardt update [35, 40, 45]. The Hessian is not calculated directly, but is instead approximated by the Jacobian $\mathcal{J}$, or matrix of directional derivatives, as $\mathbf{H} = \mathcal{J}^{\mathcal{T}} \mathcal{J}$ [35]. Note that $\mathcal{J}$ is *not* the objective function, but denotes the Jacobian matrix.

Unfortunately, while LM retains the convergence improvements of Newton's method, it also retains the problems associated with a large matrix inverse included in the step update [35, 45]. Pseudo-inverse methods assist, but do not eliminate from the computational expense the inverse calculation, making them unfeasible for models with a large number of parameters over which to optimize.

All the advantages and disadvantages of batch versus mini-batch versus stochastic implementations mentioned in the gradient descent section–see Section 3.1.1–apply for LM as well. Now, however, any error incurred in the calculation of the gradient in any batch size is compounded in the Hessian approximation as well. Batch updates reduce the error in the Hessian approximation, but require more computational expense per iteration. Stochastic LM methods are still an area of active research, though some have been in use in convolutional neural network updates for many years [36].

3.1.5 Conjugate Gradient

Conjugate gradient (CG) methods have the objective of avoiding the Hessian calculation and inversion of Newton's method. Rather than enforcing the orthogonality of search direction in the gradient descent updates, CG instead enforces conjugacy; i.e., that $\mathbf{p}_i^T \mathbf{H} \mathbf{p}_{i-1} = 0$ where $\mathbf{H}$ is the Hessian and $\mathbf{p}$ indicates the search direction at a given step [23]. The search direction, and the update as a whole, takes its motivation from a simple linear problem, $\mathbf{Ax} = \mathbf{b}$, and

transitions well into the nonlinear regime as well [35]. The step update takes the form

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \boldsymbol{\alpha}_i \mathbf{p}_i,$$

where $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ is the residual at $\mathbf{x}_i$, $\mathbf{p}_i$ is the associated search direction, and $\boldsymbol{\alpha}_i$ is the learning rate [35]. The initial search direction is the negative gradient, as in the steepest descent methods. In linear problems, the cost functional being optimized is

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} - \mathbf{x}^T\mathbf{b},$$

where $\mathbf{H}$ indicates the Hessian matrix. The residual is $\mathbf{b} - \mathbf{H}\mathbf{x}$, which is just the negative gradient of $J(\mathbf{x})$, so that $-\nabla J(\mathbf{x}) = \mathbf{b} - \mathbf{H}\mathbf{x}$ [35]. For nonlinear problems, the framework is as follows. Allow $\mathbf{r}_0$ and $\mathbf{p}_0$ to both equal $-\nabla J(\mathbf{x}_0)$ and update them via

$$\mathbf{r}_k = -\nabla J(\mathbf{x_k})$$
$$\mathbf{p}_k = \mathbf{r}_k + \boldsymbol{\beta}_k \mathbf{p}_{k-1}.$$

Here the value $\boldsymbol{\beta}_k$ can take several forms. The most common definitions of $\boldsymbol{\beta}_k$ are the Fletcher-Reeves update, where

$$\boldsymbol{\beta}_k = \frac{\nabla J(\mathbf{x}_k)^T \nabla_x J(\mathbf{x}_k)}{\nabla J(\mathbf{x}_{k-1})^T \nabla J(\mathbf{x}_{k-1})}$$

and the Polak-Ribière update, where

$$\boldsymbol{\beta}_k = \frac{(\nabla J(\mathbf{x}_k) - \nabla J\mathbf{x}_{k-1}))^T \nabla J(\mathbf{x}_k)}{\nabla J(\mathbf{x}_{k-1})^T \nabla_x J(\mathbf{x}_{k-1})}.$$

Here $\mathbf{x}$ is the parameter over which J is optimized [23]. This avoids construction of the Hessian matrix while still enforcing that search directions are conjugate. Following the conjugate directions allows the algorithm to retain the work done in previous iterations of finding the minimums in the previous directions.

Applied to network training, it is the weights and biases in the network that are updated with conjugate gradient back propagation. This requires that the weight, network input and transfer functions all have derivatives [26]. One major advantage of this method is that only the gradient and the previous search direction need to be stored at any given time. The CG method is most often implemented as a batch method, although there have been reports of success when used as mini-batch formulations as well [23]. The major advantage of CG is it's short convergence time and fast computation time. Additionally, CG is less prone to getting stuck on local minima or bouncing around in valleys in the cost functional surface than both GD and SGD. The gradient calculation and line search in CG are not inexpensive, but for smaller data sets, like our spent nuclear fuel data, they are not prohibitive.

Chapter 4

Data Fusion Methods

Data fusion refers to the combination of data from multiple sources in such a way to achieve results not possible from a single data source [37]. In the following sections, we outline first the levels at which fusion can be implemented in a classification regime and then discuss several methods.

4.1 Fusion Levels

Traditionally, there are three categories of fusion: data-level, feature-level and decision-level [38]. Data-level fusion involves the direct combination of original raw data, with normalization as needed. The combined data is then used as classifier input [55]. Feature-level fusion starts with a feature extraction on the data from each source separately. It is these feature vectors that are then combined and used as classifier input [55]. Decision-level fusion, in contrast to the other two, handles each data source separately. In practice, this means there are separate classifiers trained on each data source. It is the preliminary outputs from all these classifiers that is then combined to generate a final classification [55].

Bayesian data fusion translates traditional fusion into probability space, taking advantage of Bayes relation,

$$p(y|x) = \frac{p(x|y)p(y)}{\int p(x)dx}$$

in their combinations [51]. Instead of combining the data or the decisions directly, Bayesian data fusion methods combine priors, likelihoods or posteriors within the classification process. We discuss several methods in Sections 4.2.3, 4.2.4 and 4.2.5. For consistent terminology, we will consider the combination of priors to be data-level fusion, the combination of likelihoods to be feature-level, and the combination of posteriors to be decision-level fusion.

4.2 Methods

We discuss the different fusion methodologies used throughout this work in the following sections. Each method has advantages and disadvantages that inform our choice of fusion in the development of our spent fuel fusion frameworks.

4.2.1 Combine-then-Classify

Combine-then-classify is a simple data-level fusion method. The data from each source are directly concatenated to a single input vector. Normalization and/or non-dimensionalization beforehand alleviates scaling issues between sources. The concatenated data is fed directly to a classifier for training, testing and validation. This is simple and straightforward, with any compression implemented in the classification algorithm occurring directly on the concatenated data vectors.

Because of its simplicity, the combine-then-classify method does not have a well-defined algorithm in the literature. However, centralized processing workflows represent the process well [46]. For our application, we set up the consistent workflow defined in Figure 4.1. First, the data is normalized and/or non-dimensionalized. In our spent fuel case, the date consists of greyscale pixel intensity values directly extracted from fuel rod images. Whereas 'intensity' is defined similarly for each data source, they are not defined on the same scale, and thus only require normalization. The presented workflow, however, allows for the incorporation of other preprocessing if needed. The second step is the concatenation of each data source. The final step is to feed the concatenated data vectors into our model for training.

The combine-then-classify method has the advantage that it is simple to implement but can still yield excellent results. In addition, classification algorithms can be applied directly to the (normalized) data, allowing for the training of a single algorithm instead of several, as in decision-level fusion scenarios. However, because the size of an individual data vector after concatenation is now significantly larger, it can exacerbate issues related to the 'curse of dimensionality,' especially in cases where there are few observations.

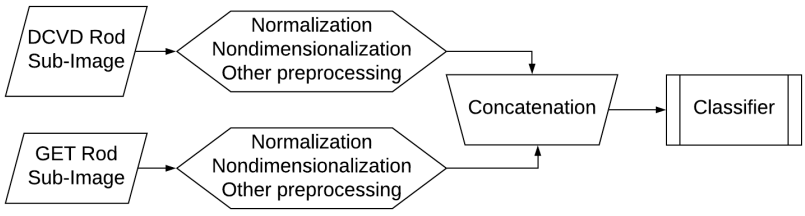


Figure 4.1: Combine-then-classify workflow.

4.2.2 Classify-then-Combine

The classify-then-combine scheme is a naive decision-level fusion method. In this scheme, separate classifiers are trained for each data source. The resulting classification for each source is then combined and thresholded for a final classification.

In this dissertation, the classify-then-combine algorithm includes a weighting of the preliminary classifications based on fuel rod location within an assembly. Let C_{DCVD} and C_{GET} denote the output for a fuel rod from a classifier trained only only DCVD or GET data, respectively. In the binary classification case, where a rod is either defect or non-defect, then the final classification, C_{final}, for a particular rod is

$$C_{final} = \begin{cases} 0 & \text{if } w_{GET}C_{GET} + w_{DCVD}C_{DCVD} < \tau \\ 1 & \text{if } w_{GET}C_{GET} + w_{DCVD}C_{DCVD} \geq \tau \end{cases}$$

for some threshold value τ. Here the weights are

$$w_{GET} = (\text{distance to assembly center})/(\text{max distance to assembly center})$$

$$w_{DCVD} = 1 - w_{GET}.$$

This definition of the weights is based on expert knowledge of where the GET and $DCVD$ measurements are most accurate. The particular distance function used for calculation of the weights can either be a standard Euclidean distance, or some other distance function designed to quantify a specific capability of DCVD or GET. For example, GET rotates around a fuel assembly, yielding near-equal accuracy for the outermost fuel rods. This could be reflected in a Chebychev distance function; see Chapter 6 for a Chebychev distance-informed weighting algorithm. The threshold value can be selected via inspection of a receiver-operator characteristic curve, as outlined in Chapter 5.

4.2.3 Bayesian Prior Fusion

One option for data-level fusion in probability space is Bayesian prior fusion. In our spent fuel monitoring application, there are two options for prior distributions. The first is based on facility declarations; e.g., the probability of a defect being present is low if the facility under inspection declares there are no defects in the cooling pond. Under this scenario, the prior distributions for each detector is the same. The second option is simply a flat prior distribution. Beyond these two options, defining a prior in our application remains difficult, and thus prior fusion methods are considered here primarily for completeness but are not implemented in our research.

The simplest ways to combine priors are the pooling methods outlined in previous sections for combining posteriors, just applied to priors [18]. An issue consistently faced by prior fusion methods, however, is that priors are often informed by multiple subject matter experts who may or may not be statisticians, and thus have varying abilities to define their beliefs about

a parameter as a distribution [18, 58]. This makes the pooling options less practicable, and forms the basis for significant research into the proper elicitation of expert opinion for use in statistical models [18].

An alternative is the 'supra Bayesian,' where each expert opinion feeds in as data to another Bayesian framework, which provides a final fused prior for use in subsequent statistical models [18, 42]. In the spent fuel monitoring application, we do not have the informed priors necessary that would make the supra Bayesian a worthwhile effort.

Another approach to prior fusion is behavioral, where some interaction between experts is created; e.g., a jury or other panel discussion [18, 42, 58]. Behavioral approaches are useful when applicable, and can approximate academic consensus. However, in our application, the 'experts' are two different detector methods rather than people, making inducing interaction between experts impossible.

Overall, prior fusion is a field if continuing growth, but with limited applicability to our problem. Priors in our problem space are not necessarily well-described, and combining priors make little sense if all priors are non-informative or flat. Thus, we focus on other fusion methods.

4.2.4 Independent Likelihood Pool

The independent likelihood pool is a feature-level Bayesian fusion method. It makes the assumption that each data source has the same prior information [44, 46]. The combined posterior distribution over M data sources $p(y|x_1, x_2, \ldots, x_M)$, where the x_i data can be either scalar or vector-valued, is expressed as a product of likelihoods from each data source with a common prior distribution $p(y)$ such that

$$p(y|x_1, \ldots, x_M) \propto \prod_{m=1}^{M} p(x_m|y)p(y).$$

A proportionality constant is used to ensure that the posterior distribution integrates to 1. Note that for this equation to be valid, the likelihoods must be *independent*, though the prior information about y is identical across all data sources. Specific to our spent fuel application, the prior information, as currently defined, is indeed similar across both GET and DCVD. However, the independence of the likelihoods is not necessarily true. It is possible that related physics impacts the signal received by both DCVD and GET, violating the assumption of independence. This indicates that independent likelihood pooling is not applicable here.

In addition, in our application, we pair Bayesian fusion methods with with BNCA, as defined in Chapter 6. Because BNCA is a posterior estimation algorithm, what is propagated through our developed pipeline algorithm is in fact the posterior for each data source rather than the likelihood. While the likelihood could also be calculated, it is not explicitly done so as currently implemented, thus limiting the applicability of the independent likelihood pool for this application.

4.2.5 Linear Opinion Pool

The linear opinion pool (LOP) is the first of two decision-level fusion methods presented here, and arguably the most popular simple Bayesian data fusion method at any fusion level [42, 44, 46].

For LOP, the posterior, $p(y|x_m)$ is estimated/learned separately for each of the $m = 1, \ldots, M$ data sources. Then, the final fused estimate, or *consensus distribution*, for the posterior is

$$p(y|x_1, \ldots, x_M) = \sum_{m=1}^{M} w_m p(y|x_m), \tag{4.1}$$

where w_m are the weights on each data source with $0 \leq w_m \leq 1$ and $\sum_{m=1}^{M} w_m = 1$ [42, 44, 46]. This is applicable whether each x_i is scalar or vector-valued, as long as the weights are scalars. In our application, each x_i is a vector containing the pixel intensities for an image of a single fuel rod. During implementation, the main concern for linear opinion pools arises in the estimation of the distributions for each data source separately. Once they are estimated, the combination is straightforward weighted summation. The weights can be defined similar to those in the classify-then-combine scheme. Expert knowledge or confidence in the data source and posterior estimation method can be reflected in the weights. Elicitation and incorporation of expert knowledge is discussed at length by Genest et al. in [19, 20] but, in our application, we aim to reflect the regions of detector sensitivity to signal produced by rods based on their location within an assembly. Weighting can also be used to eliminate any faulty data or mitigate the impact of missing data. In our nuclear fuel application, however, we assume that neither of our two data sources are faulty to the point of requiring complete removal and that we have data from both sources at every fuel rod.

Because the final posterior estimate is a sum and not a product of individual posterior distributions, linear opinion pooling remains applicable in cases where there are dependencies between information sources [44]. In our application, our two information sources are two different detectors, but this does not necessarily ensure that the signals they measure are completely independent. Thus, the linear opinion pool is a good fusion choice, given that we cannot guarantee independence in the signals and their underlying physics.

Additionally, LOPs address zero distributions in a manner effective for our particular application. In LOPs, because the consensus distribution is a sum of the individual data source distributions, there is no 'automatic veto' should any one source have a zero distribution. In multiplication-based pooling methods, as described in the following logarithmic opinion pooling section, any source returning a zero distribution automatically zeros out the final consensus distribution as well [42, 46]. In our application, there are cases where, for example, assembly hardware blocks all signal to the DCVD. At that same location, GET may pick up a very strong signal. This appears then as a zero distribution in our opinion pooling from the DCVD side, and should not override the much more informative GET result in determining the com-

bined posterior. In LOPs, any zero distributions are averaged out instead, which is much more appropriate in our spent fuel monitoring regime [46].

The biggest disadvantage of LOPs, as compared to their multiplication-based counterparts, is their propensity for multi-modal consensus distributions. Because LOPs are a summation, this often leads to a final fused distribution with several modes, as compared to, for example, the logarithmic opinion pool that generally results in a more unimodal consensus distribution [42, 46].

Many variations on linear opinion pools exist and have been tested extensively, but standard LOPs remain one of the most popular fusion methods. This is the starting point and main focus of the BNCA+Fusion Pipeline Algorithm detailed in Chapter 6 [19, 20].

4.2.6 Logarithmic Opinion Pool

The logarithmic opinion pool (LgOP) is similar to the linear opinion pool, but provides solutions to some of the LOP's issues.

The LgOP defines the final consensus distribution as a product of the distributions estimated by each data source, such that

$$p(y|x_1, \ldots, x_M) \propto \prod_{m=1}^{M} p(y|x_m)^{w_m},$$

with a proportionality constant that normalizes the right hand side so that the consensus distribution integrates to 1 [42]. As in the case of the LOP, the implementation is straightforward; once the distribution is estimated for each data source, the LgOP consensus distribution is then directly calculated.

While LgOP is *externally Bayesian*; i.e., pooling computes with Bayesian updates, LOP is not. Alternately, LgOP does not have the *marginalization property*; i.e., pooling commutes with marginalization, of LOP. Marginalization is the process of summing over all possible values of one variable to assess the contribution of another variable, or, more mathematically, that $\sum_y P(x, y|z) = P(x|z)$ for variables x and y.

The choice of LgOP or LOP is often based on preference between these two properties. In our case, we train all at once under the assumption that our developed framework will be trained with a fixed set of data and deployed rather than updated frequently with new data. Thus we do not require our algorithm to be externally Bayesian. The marginalization property, though not necessary in current implementations, could be advantageous should we choose to incorporate physics-informed variables such as burn-up in addition to class assignments in future work.

An additional consideration arises in the difference in formulation of LOP and LgOP. Because it is based on a summation of distributions, the consensus distribution calculated with LOP is frequently multi-modal. Alternatively, the multiplication in LgOP means that the final consensus distribution is more frequently unimodal. This permits smoothing of the individual

estimated distributions for each data source without information loss. A unimodal result is also much easier to interpret, making LgOP results potentially more practical than LOP.

Another consideration is that of the single-source veto of LgOP; if a single data source has a zero distribution, the consensus distribution will also be zero. However, by detector design and implementation, as discussed in the previous LOP section and again in Chapter 6, this single source veto is highly undesirable in our spent fuel application, and undermines the primary goal of combining DCVD and GET. Thus, while LgOP does provide a unimodal consensus distribution and is externally Bayesian, we focus primarily on LOP.

The difference between the logarithmic opinion pool and linear opinion pool is illustrated in Figure 4.2 from [42]. The dotted lines are two different expert opinion distributions. The solid line is the LOP with equal weights for each expert distribution and the dashed lines are the LgOP with equal weights for both distributions [42]. The LOP encompasses the entire space covered by each expert distribution, whereas the LgOP emphasizes the overlap region of the two expert distributions. Both have advantages and disadvantages; the broader peak, or potentially multiple peaks, of the LOP may make a final classification more difficult, but the strong emphasis on overlap between the expert distributions may cause issues if experts disagree. Our experts, DCVD and GET, do occasionally disagree based on their differing regions of signal sensitivity, favoring LOP for adequate fusion in this application.

The independent opinion pool is a variation on the logarithmic opinion pool with all weights set equal to 1. It again combines posterior distributions under the assumption that prior distributions are independently obtained for each data source [44, 46]. Thus the final posterior estimation can be expressed as

$$p(y|x_1, \ldots, x_M) \propto \prod_{m=1}^{M} p(y|x_m).$$

The independent opinion pool assumes that $p(y)$ is the same for each data type so is contained in the proportionality notation. This method requires that the priors are independent, which is

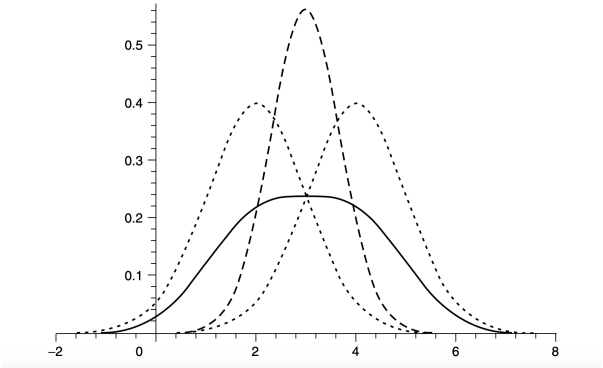


Figure 4.2: LOP (–) versus LgOP(- -) for two expert distributions (··) [42].

often violated in practice, making it less common in real applications. In our case, our 'prior' is the probability of a defect, which should be the same for both of our detectors, and in practice, may be derived from operator/state declarations.

Chapter 5

Non-Bayesian Fusion Methods and Results

In Chapters 1-4, we discussed the theoretical bases for our proof-of-concept research into the novel combination of GET and DCVD data. The work in this chapter encompasses more traditional classification methods; i.e., PCA+kNN and neural networks. Work including PCA+kNN results was published in [11]. Each of the methods presented in this chapter are frequently implemented to solve classification problems, but we uniquely consider fusion regimes within each method as well as address issues specific to defect detection. Additionally, spent fuel safeguards is a novel application of these methods, made more-so by taking into account operational constraints by only using already collected data.

In Chapter 6, we will consider probabilistic metric learning methods, whereas the methods in this chapter are limited to non-probabilistic methods with defined metrics. In both Chapters 5 and 6, the implementation is novel in its inclusion of a fusion scheme, as well is an the application.

5.1 Data Generation

Each image in our data set is of a 17×17 PWR assembly with 264 present rod locations and 25 guide tube/water channel locations. In Figure 5.1, the light regions are the guide tube/water channels while the darker regions are fuel rods.

Data was provided by two sources, with DCVD images coming from the Applied Physics group at Uppsala University in Sweden and the GET images from Pacific Northwest National Lab (PNNL). DCVD images are real images taken in facilities with either one or zero missing rods. The first image in our data set has no missing fuel rods, whereas the second two images have a single defect each in different locations. The tomography images are simulated by PNNL as part of a GET detector validation project [49]. Each image has 11 missing fuel rods in distributed across an assembly in such a way to have representative local geometries.

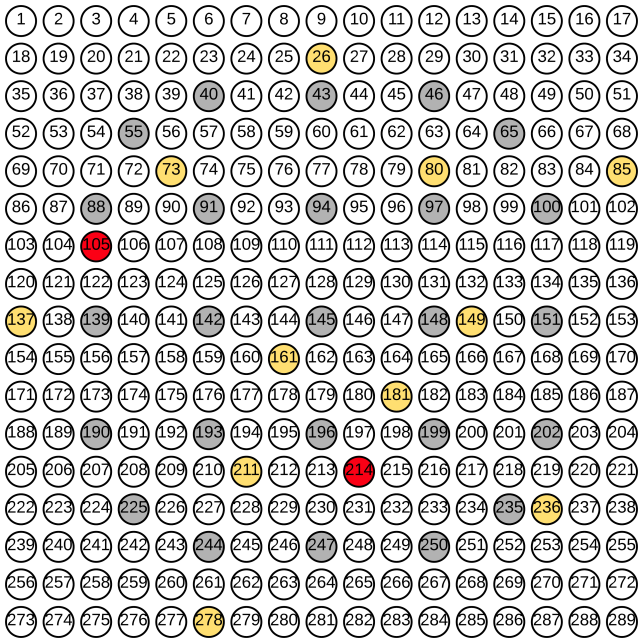


Figure 5.1: 17×17 PWR assembly schematic. White indicates non-defect, gray indicates water channel/guide tube, red indicates DCVD defect locations, and yellow indicates GET defect locations.

Because we are investigating defects on a single rod defect level, the data set for classification needs to be individual fuel rods. Given prior work on matching assembly geometries to a template, we assume that the geometry of the assembly is known [14]. To section out individual fuel rods from the original assembly image, we start with a Hough circular transform to identify the approximate locations of fuel rods and their pixel widths in the original image [61]. The Hough transform step informs a binary mask that we apply to the original image, which enables us to extract individual fuel rod sub-images. We do this simultaneously across all fuel rods in an assembly image at once, with exact rod centers averaged across both rows and columns. The averaging eliminates algorithmic variations between identified fuel rod centers and pixel radii. The diameters are increased by 3 pixels in order to avoid data loss. The Hough transform mask is shown in Figure 5.2.

Each fuel rod sub-image–or equivalently, a pixel intensity matrix–can now be generated by extracting the grayscale intensities of every pixel within a fuel rod, where the fuel rod boundaries are defined by the rod center and radius identified with the Hough transform mask. These pixel intensities directly populate square 23×23 matrices, where 23 pixels is the identified diameter of the fuel rod. Because the fuel rods are round and the populated matrices are square, each fuel rod matrix is initialized to be a matrix of all -1 values before population with the strictly positive pixel intensities. This avoids any issues in the event that all pixel intensities in a fuel rod sub-image are zero. The original assembly image is directly represented by a large grayscale

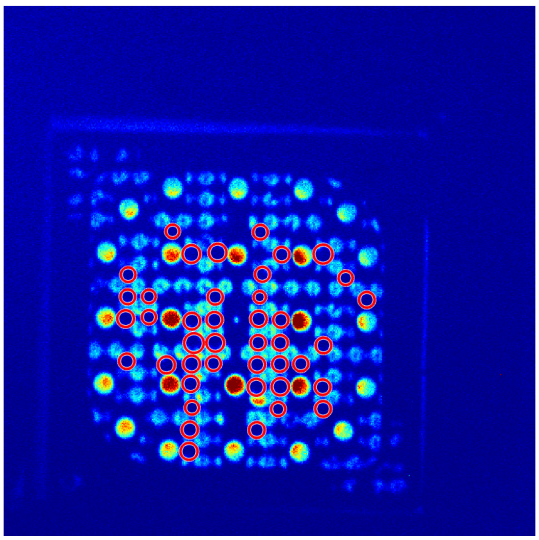


Figure 5.2: Hough transform applied to a DCVD image.

pixel intensity matrix, which is multiplied by a binary [-1,1] mask containing a single circle with radius 11.5 pixels. This mask is the Hough transform mask. The circle within the binary mask is centered at a particular fuel rod's pixel center, and the individual rod matrix is then populated by extracting all the positive intensities in the product of the mask with the original image matrix. Within the mask, the center of the circle is then translated to the next fuel rod center, moving across the rows of the assembly, and the process is repeated to extract the subsequent fuel rod image matrices.

For a given fuel rod, the populated image intensity matrix is 'flattened' into a vector along the rows of the matrix. In other words, the first row–length 23–of the matrix becomes the first 23 components of the vector, the second row becomes the 24-46$^{\text{th}}$ components of the matrix, and so forth. Each of the 289 rods in a spent fuel assembly are now represented by these 'flattened' fuel rod vectors. Finally, these vectors are concatenated to create a 529×289 matrix, where each column is a 'flattened' fuel rod data vector.

To create a consistent data set, defects across both data sets need to be located in the same positions. To create a DCVD with the 11 missing rods present in the tomography data, the two original DCVD defects are replicated with 5% Gaussian noise added, and moved to the tomography defect locations. When the tomography defect location is nearer to the edge than the center, the DCVD defect closest to the edge (position 105) is replicated at the new location, and the more centrally located defect (position 214) replicates defects closer to the center of the assembly. To replicate tomography defects in the DCVD defect locations, the defect within the set of 11 with the most similar local geometry is used, and 5% noise is again added. To replicate non-defects in the defect location, the nearest present rod in the assembly geometry to the defect is replicated with 5% noise. The level of noise added was chosen based on the simulated parameters within the tomography data set provided by PNNL. While Gaussian

Table 5.1: Burnup and cooling time details.

DCVD Data	Burnup	Cooling Time
Case 1	44.3 GWd/tU	4.92 years
Case 2	39.8 GWd/tU	4.05 years
Case 3	36.1 GWd/tU	5.96 years
GET Data	Burnup	Cooling Time
Case 1	220 GWd/tU	5 years
Case 2	10 GWd/tU	40 years

noise was used here for simplicity and ease of future mathematical derivations, Poisson noise is the standard distribution used for nuclear data since it represents counts. Note, however, that the Poisson distribution limits to the Gaussian distribution. All the methods in Section 5.2 are applied directly to the matrices of 'flattened' fuel rod data vectors with consistent defect locations due to this process.

Due to the maximum number of defects being 11/289, we make the necessary assumption that a water channel/guide tube approximates a missing rod in order to expand our 'defect' set in the binary classification problem. In DCVD, this is an accurate assumption, as a missing rod leaves a water channel, which propagates visible light like the present water channels in the assembly. In tomography, this assumption is adequate, as the mean-free path in water is the same in the case of a water channel and missing rod. Guide tubes, however, have a different mean free path than water channels, so the gamma emission signature is different. Because all our data includes defects, replication of the defects and training on a set with an artificially high number of defects may relieve this issue. We provide additional discussion of differences in training sets in Chapter 6.

Another data set issue is the case of burnup (BU) and cooling time (CT), which vary both within and across our data sets. We present burnup and cooling time details in Table 5.1. The DCVD data set was chosen for this work due to the fact that its light intensity is similar across the provided three images, with similar burnup and cooling times. Due to the similar intensities, we are able to ignore the burnup and cooling time information from this data source altogether, instead taking on BU/CT information from the tomography data set. However, in the GET set, there are two different burnup and cooling times that represent the extremum of what would be seen in the field: high burnup with short cooling time, and low burnup with long cooling time. These two scenarios represent the edges of reliability of GET. Should our methodology of combination succeed in these two scenarios, we expect success on intermediate scenarios between these edge cases.

5.2 Methods

This section outlines the tested classification methods. For more theoretical information, see Chapter 2. For the final results using each method, see Section 5.5.

The following classification schemes are all trained and validated on 289 rod sub-images extracted from a single assembly image via the process described in Section 5.1. The trained classification scheme is then deployed and verified on the rod sub-images from each assembly. Locations of defects in each case are known, and so deployed classifier accuracy can be tested against results. Because there are limited real data sets with defects present, this limited classifier training scheme is similar to what would be encountered in industry.

For all the methods in the following sections, single fuel rod sub-images are extracted from an assembly image, 'flattened' across the pixel rows of the sub-image, and then concatenated to be a matrix of single rod data vectors. Details of this process are in Section 5.1. The final data matrix contains 289 fuel rod vectors, each with 529 components.

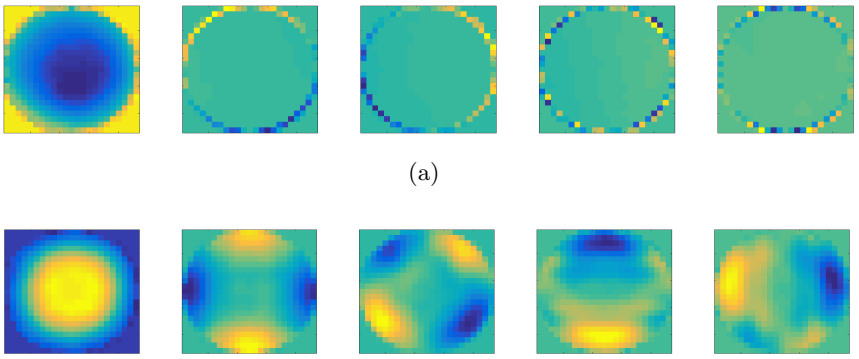
5.2.1 PCA+kNN Implementation

A 529×289 data matrix $\mathbf{X}$ for a single assembly is generated by extracting single rod sub-images from an assembly image using a Hough transform-informed mask. The sub-images are square matrices that are 23×23 pixels with -1 in all pixel locations where the sub-image, trimmed to the circular fuel pin, does not have data. This is followed by collapsing the sub-images across the rows into vectors and concatenating the fuel rod data vectors to create a matrix.

The first step in the principal component analysis (PCA) with k-Nearest Neighbors (kNN) framework–see Sections 2.1.3 and 2.1.2–is compression by mapping into a principal component space smaller than that of the original image. PCA identifies the principal components in a centered data set using singular value decomposition via Equation (2.2). The data is mapped to principal component space by Equation (2.1) wherein kNN is subsequently carried out on the newly mapped data set via Algorithm 1. By mapping into a principal component space, the variance in the data set is maximized and neighborhoods are, theoretically, more identifiable with kNN for classification.

We initially implemented the PCA+kNN process with only DCVD images. Individual rod sub-images are sectioned out of each of the three DCVD images using our binary Hough transform mask. In the DCVD data set, one image has no defects, and the rods from this image were used to develop the compressed principal component space. The data set is comprised of one set of 'defect approximating rods,' with 25 water channels and the center guide tube channel, and 264 'no-defect' fuel rods. Each rod sub-image is 23×23 pixels, or when 'flattened,' vectors of length 529. The PCA approximations of each rod sub-image are reconstructed with some smaller number; e.g., less than 529 principal components.

We plot the first five principal components $u_1, \ldots, u_5$ from Section 2.1.3, or 'eigenfaces' in traditional facial recognition literature, for both DCVD and GET in Figure 5.3.



(a)

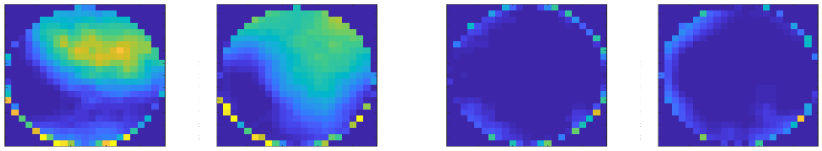
(b)

Figure 5.3: The first five principal components of (a) DCVD and (b) GET fuel rod sub-images.

After calculating the principal components via the singular value decomposition in the manner discussed in Section 2.1.3, we verify our reconstruction by computing a sum-of-squares error difference between the original 529-component data vectors and the vectors after reconstruction with the principal components. For all 529 principal components, the error between original and reconstructed is on the order of 10^{-16}. Hence we have identified the correct principal components. To identify the required number of principal components for a good approximation, we compute sum-of-squares error between original and approximate images for a variety of component-space dimensions. Between 50 and 400 principal components, there is no significant change in the difference, so we chose the lower bound of 50 components. The sub-images for both a DCVD defect and non-defect are shown in Figure 5.4, with the original defect shown in (a), the original non-defect in (c) and the reconstructed defect and non-defect using 50 principal components shown in (b) and (d), respectively.

For the subsequent kNN classification Algorithm 1, we use 10-fold cross-validation measures to identify the parameters best-suited to our problem. We label the known water channels and the known rods in a single DCVD assembly image and use the selected parameters to train the classifier. MATLAB has a built-in cross-validation regime to test a variety of kNN algorithmic parameters, primarily metric and number of neighbors. The built-in cross-validation segments the data into 10 folds, and then trains and tests the kNN classifier across the segments such that one fold is chosen as the test set and the other nine as training sets. MATLAB fits a kNN classifier to 9 folds for each of a variety of pairings of metric and number of neighbors and tests on the final fold, iterating through this process so that each fold is selected as the test set at least once. The MATLAB tool then evaluates the objective function, defined as the misclassification rate $\frac{1}{n}\sum_i (y_i \neq y_i^*)$ for classification problems with observed y_i and true y_i^*. From this, we considered the 20 best performers as ranked by misclassification rate.

We then implemented each of these and performed two blind tests, focusing on performance



(a) (b) (c) (d)

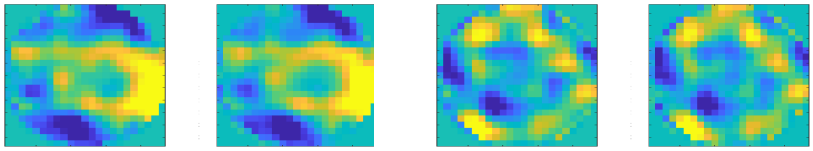
Figure 5.4: DCVD fuel rod sub-image examples: (a) original defect, (b) reconstructed defect, (c) original non-defect, (d) reconstructed non-defect.

on correct classification of the defects rather than the overall test set. Minkowski metrics,

$$d(x, y) = \left(\sum_i |x_i - y_i|^p \right)^{1/p,}$$

have an additional exponential parameter p to optimize, unlike several other considered metrics; e.g., Euclidean, and city block. For Minkowski metrics, we used the results from the built-in optimization as comparison, evaluating the misclassification rate for exponents of 1 to 10. With $p = 5$, our implemented kNN classifier had a misclassification rate of the same order of magnitude as the top five performers, but with improved performance specifically on our defect class. This improvement was consistent for one nearest neighbor, consistent with the recommendation from the original MATLAB routine. We thus select our final kNN parameters to be one nearest neighbor and a Minkowski metric with $p = 5$.

With the tomographic reconstructions, we again focus on individual rod sub-images from each of our four images. We recompute our principal component basis, using 50 components for consistency with DCVD data set approximation. However, to obtain a norm difference similar to the one for DCVD with 50 components, we need 300 tomography principal components; the sum-of-squares error does not decrease between 50 and 300 for the tomography data. This is due, in part, to two separate factors. The first is the difference in scale between the DCVD images and the tomographic images in our separate data sets. The second is the fact that the GET data is a reconstructed cross-section of an assembly. The data is collected as a sinogram, but cross-sections are reconstructed using a method called filtered back projection [49]. Fuel rods and water channels are much more similar in GET reconstructions, given assembly self-attenuation and the low contrast due to filtered back projection's inability to incorporate differences between water and rod regions. Note that the results obtained with combination are consistent when all 529 principal components are used, so this restriction to 50 does not significantly impact the conclusions. Reconstruction of a GET defect and non-defect using 50 principal components are shown in Figure 5.5 (b) and (d), alongside their original sub-images, (a) and (c).



(a) (b) (c) (d)

Figure 5.5: GET fuel rod sub-image examples: (a) original defect, (b) reconstructed defect, (c) original non-defect, and (d) reconstructed non-defect.

The individual rod sub-images from the first assembly image in both DCVD and GET data sets are used for training the kNN classifier. For consistency between data types, we continue to use the $k = 1$ and Minkowski $p = 5$ parameters for the classifier, as norm choice does not significantly impact the presented results; improvements from combination hold regardless of kNN parameters–see Section 5.5. The impact of the norm is more evident when restricting to only a single data type to classify fuel rods. Even with optimal kNN parameters, the automated single data type classification of defects improved only marginally.

We present results in Section 5.5.

5.2.2 NN Implementation

As detailed in Section 5.1, the input data for the neural network is a matrix of 'flattened' grayscale pixel intensities from fuel rod sub-images. Each fuel rod is contained within a sub-region of the image of the entire assembly, and this region is extracted with a mask created with a circular Hough transform edge detection algorithm. The grayscale pixel intensities from the individual fuel rod sub-images populate a 23×23 matrix with -1 values filling the entries outside of the 23 pixel diameter circular rod image. The individual input vectors for the neural networks are individual 23×23 sub-image matrices 'flattened' into a vector by concatenating the rows. An entire assembly is a matrix composed of 289 of these 'flattened' vectors.

In the initial neural net implementation, there is no compression and the network input is an entire fuel rod sub-image. For methods with compression, see Sections 5.2.3 and 5.2.4. Our implemented network has three layers of 5 nodes, with a symmetric sigmoidal activation function. The cross-entropy cost functional from Equation (2.15) drives the optimization problem in training, using conjugate gradient methods. Cross-entropy is a standard functional in binary classification methods, as it heavily penalizes incorrect classifications. The sigmoid function in Equation (2.12) is the activation function, as the small network size adequately reduces concerns of vanishing gradients addressed by other activation functions.

Network size was chosen primarily for computational constraints with the goal of having a quickly trainable, small network to compare with the other methods as a proof-of-concept for combination rather than a truly optimal network. We tested four network sizes: 5 nodes each

in 3 hidden layers (5×3), 10 nodes each in 3 hidden layers (10×3), 50 nodes in 1 hidden layer (50×1) and 529 nodes in 1 hidden layer (529×1). We present optimization results using each network size are presented in Section 5.4, and overall results in Section 5.5.

5.2.3 PCA+NN Implementation

For consistency with the PCA+kNN analysis, we used 50 principal components in the PCA+NN classification scheme, making our input vector length 50 rather than the full length of each image vector at 529. The network is trained the same way as in the NN case, and the network architecture is also the same as before for consistency. PCA is applied separately to the uncombined data in the classify-then-combine scheme, but is applied directly to the combined data vector in the combine-then-classify scheme. We summarize results in Section 5.5.

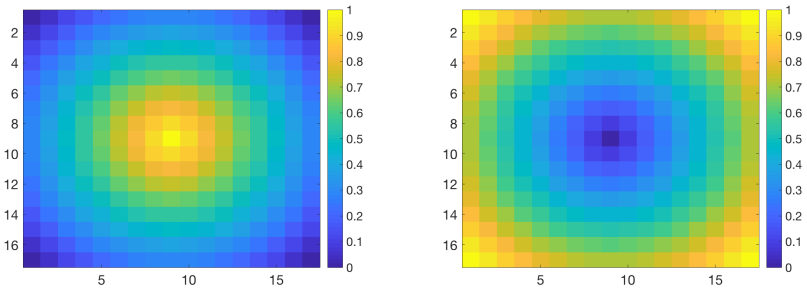
5.2.4 CNN Implementation

Here, cross-entropy–Equation (2.15)–is again the chosen cost functional, which we optimize using conjugate gradient for consistency between CNNs and (PCA+)NN. The activation function in this case, however, is the Rectified Linear Unit (ReLU) from Equation (2.13). Vanishing gradient issues tend to be more pronounced in CNNs as the error is backpropagated through a neural network and the additional convolutional/pooling architecture, and the choice or ReLU helps mitigate that. The neural network implemented in the final step of the CNN is a 5×3 network.

There are 15 5×5 filters in the convolution layer characterized by Equations (2.16) and (2.17), followed by 3×3 mean pooling. In our application, both zero intensity pixels and high intensity pixels are informative, and both need to be taken into consideration in the pooling. Mean pooling provides the best way to screen out physics-related noise; e.g., stray gamma flashes in DCVD, while retaining relevant pixel information. We tested both max and min pooling for completeness, but these had minimal impact on the final results presented in Section 5.5, and are therefore omitted.

5.3 Data Fusion

In this non-Bayesian analysis, we consider two methods of data combination: combine first, then classify or classify first and combine the classifications. Combining first is a data-level fusion method, while classifying first is a decision-level method. For an overview of all fusion methods, see Chapter 4. In the classify-then-combine scenario, discussed in Section 4.2.2, the two data types are classified using their individually trained classifiers, and the resulting classifications are combined in a weighted sum and thresholded for a final classification. The weights in the sum result from expert knowledge of our measurement data types; DCVD is most accurate near the image center, and GET most accurate near the image edge. Thus, if a rod is located near the edge of an assembly, the sum is weighted to rely more on the individual classification



(a) (b)

Figure 5.6: Classify-then-combine weights: (a) DCVD weights, (b) GET weights.

generated with GET. The weights are proportional to the distance from assembly center for GET and inversely proportional to distance to center for DCVD. The distance to the assembly center is taken to be standard Euclidean distance from fuel rod centroid to the centermost fuel rod's centroid. The weighting schemes are presented graphically for both DCVD and GET in Figure 5.6. Because the assemblies are 17×17 arrays, the weights are displayed on a 17×17 grid. In the case of our DCVD data set, there is actually assembly hardware that covers the four corners and the outermost edge of fuel rods. This weighting scheme correspondingly ignores the lack of data from the DCVD in those locations and relies much more on the GET.

The threshold for the final classification in the classify-then-combine scenario is determined experimentally by investigation of a Receiver Operating Characteristic (ROC) curve. Receiver operating characteristic curves are a graphical means of representing the balance between true positive and false positive rates with binary classifiers as a function of some parameter–most commonly as a function of a classification threshold. True and false positive rates at several threshold values for each method are presented in Figure 5.7. While the ROC curve for the convolutional neural networks appears to be a straight line along the y-axis at $x = 0$, this is because its false positive rate stays at 0 while its false negative rate changes much more dramatically for the different threshold values. While not optimal for all methods, a threshold of 0.5 was deemed adequate for comparison and used in all methods for consistency.

In the combine-then-classify scenario, detailed in Section 4.2.1, the data from both sources is directly concatenated, and our algorithms are then trained, tested and deployed on this concatenated data. Because the data from DCVD and GET are originally on vastly different scales, all pixel intensity values were normalized to the range $[0, 1]$ before concatenation. Compression, when applied, is carried out directly on the concatenated vectors. There is no final threshold applied here, so this is not included in our ROC curve.

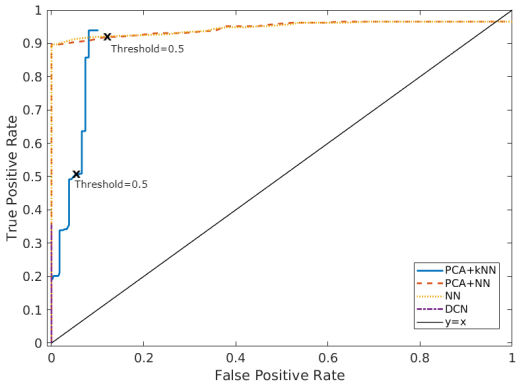


Figure 5.7: ROC Curve for each classification method under the combine-then-classify scenario.

5.4 Comparison of Optimization Methods

In addition to the threshold value tested with our ROC curve in Section 5.3, for neural networks, we also need to take the optimization algorithm into consideration. A standard binary classification cost functional is cross-entropy, where the cost J is defined as

$$J = \sum_{i=1}^{M} \left[-d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i) \right]. \tag{5.1}$$

Here, y_i is the network output and d_i is the target output for input x_i. The output of the network is a function of weights within the network, the activation functions and the inputs. We provide a small example equation in Equation (2.11). During the training phase, the weights of the network are updated according to an optimization algorithm. We provide details of several optimization algorithms in Chapter 3, and test them with our application in this section.

We principally tested the considered optimization methods on the traditional neural network without prior compression in order to isolate the impact of the optimization method itself. We tested optimization methods for four different network sizes: 529 nodes in the first (and only) hidden layer, 50 nodes in 1 hidden layer, 10 nodes in each of three hidden layers, and 5 nodes in each of three hidden layers. The input layer for all four network sizes has 529 nodes, as each image is a vector of length 529 without any compression. The reason for inclusion of the 50 node, 1 hidden layer network structure is that when we include compression, our input is in fact a compressed image vector with 50 components.

We present classification results for each network size and optimization algorithm in Table 5.2. The tested optimization algorithms are gradient descent, stochastic gradient descent, Levenberg-Marquardt, and conjugate gradient with Fletcher-Reeves updates. To present optimization results independent of other algorithmic choices, like fusion weights and threshold values, all results presented in this section are for the combine-then-classify data-level fusion

Table 5.2: NN results under combine-then-classify scheme for all network structures and optimization methods. Abbreviations are as follows: GD–Gradient descent, SGD–Stochastic gradient descent, LM–Levenberg-Marquardt, CG–Conjugate gradient.

	GD		SGD		LM		CG	
	TP(%)	FP(%)	TP(%)	FP(%)	TP(%)	FP(%)	TP(%)	FP(%)
529×1	82.4	1.2	46.5	0.2	-	-	90.9	0.2
50×1	88.8	0.0	36.0	0.5	-	-	90.5	0.1
10×3	89.9	0.0	43.2	26.6	92.8	0.1	92.6	0.1
5×3	87.1	0.0	45.5	36.7	92.8	0.3	91.3	0.2

scheme. For full results including results on DCVD individually, GET individually, and the classify-then-combine fusion scheme, see Tables B-B in Appendix B. Note that for both the neural network with 529 and with 50 nodes in a single hidden layer, the matrix inversion of Levenberg-Marquardt optimization is prohibitively expensive and corresponding results are not available.

Generally, where there's an increase in true positive rate, there's a corresponding increase in false positive rate. In the network structures where Levenberg-Marquardt was applicable, this trend was less pronounced, as the classification accuracy was similar to those generated with the other optimization algorithms, but with slightly reduced false positive rates.

The differences in classification accuracies between optimization algorithms were generally minimal within a fixed network structure/size, with the exception of stochastic gradient descent. Stochastic gradient descent, given its online learning method, learns the lower occurrence defects much more poorly than the other optimization techniques. For optimal stochastic gradient descent performance, the training data size needs to be much larger than the data set we have in order for the stochastic gradient approximation to converge to the true gradient.

Overall, the major issue was computational expense. We compute estimated timings for training a single neural network for the various network structures and optimization methods in Table 5.3. The combine-then-classify scheme requires only the training of a single network, versus the classify-then-combine scheme which trains two networks and can be parallelized, but is not in our implementation. Thus, these timings are for the single network training of the combine-then-classify scheme. Timings are averaged over 10 runs. Moreover, to include all optimization algorithms, the timings are for the 5×3 network structure.

Gradient Descent

We note that gradient descent methods, detailed in Section 3.1.1, take an order of magnitude more iterations to converge to a trained network than the other optimization methods, averaging 500 as opposed to 20 or fewer. This is due to the slow convergence rate evident in Equation (3.2). Even with more iterations, gradient descent still significantly outperforms Levenberg-Marquardt

with regards to timing. However, our goal is to have a high true positive rate while keeping the false positive rate low. Gradient descent performs well, but is not the best-performing option for this goal.

Stochastic Gradient Descent

Similar to gradient descent, stochastic gradient descent is slow to converge in the timings in Table 5.3. It requires at least as many iterations as gradient descent to converge, and often requires more. This is due to the fact that, because it is still a gradient descent method, the overall convergence rate is still governed by the slow convergence demonstrated in Equation (3.2). Consistent with the results presented in [21], the stochastic gradient descent is slower to run and slower to converge than gradient descent, requiring double the time in our tests. There were several outlier cases not included in the average timing where it required an order of magnitude more time and iterations than gradient descent.

The primary advantage that stochastic gradient descent provides is in the context of very large data sets. Using the gradient approximation in Equation (3.3), the costly gradient calculation in standard gradient descent in Equation (3.1) is avoided by approximation at a single point. However, for small training data sets, the gradient calculation is not prohibitively expensive, and thus savings offered by stochastic gradient descent do not outweigh the potentially long convergence time required to train the network. Additionally, stochastic gradient descent updates as it iterates through the sample data points rather than updating once after iterating through every point. In very large data sets, this is a substantive savings in memory but can be more costly than standard gradient descent for small data sets. The stochastic aspect also means that low occurrence events, like the defects in our data set, are rarely taken into account, as they are less likely to be sampled for the gradient approximation. In standard gradient descent and other optimization algorithms, defects are involved in the averaged gradient update more frequently, if not at every step. With at most 36 out of 289, we do not have enough examples of defects for SGD to properly train to identify them. While stochastic gradient descent is highly effective in problems with much larger training sets, it does not provide the accuracy or computational savings in our application that it can provide in big data contexts. For all these reasons, we do not employ stochastic gradient descent for our analysis.

Table 5.3: Timings for 5×3 NN structure under combine-then-classify scheme for all optimization methods.

Optimization Method	Timing (seconds)
Gradient Descent	4.63
Stochastic Gradient Descent	9.19
Levenberg-Marquardt	42.5
Conjugate Gradient	0.90

Levenberg-Marquardt

The Levenberg-Marquardt method detailed in Section 3.1.4 requires the storage of either a Hessian or Jacobian and a subsequent matrix multiply in order to compute the step update in Equation (3.4). For neural networks, storing a Hessian for every variable is prohibitively large for both the 50×1 and 529×1 architectures. However, in the 10×3 and 5×3 networks, we noted a better ratio of true positive rate to false positive rate than with the other optimization methods. The computational expense and time required, while an order of magnitude higher than other methods, is not prohibitive for smaller network structures. However, the nature of Levenberg-Marquardt requires a change in performance metric from cross-entropy to squared error, which impacts how the network is trained and fitted to our data. For binary classification, cross-entropy is the preferred performance metric for it's penalization of misclassification, thus making this requirement for Levenberg-Marquardt undesirable [5].

Conjugate Gradient

The conjugate gradient method, detailed in Section 3.1.5, is fast, requires few iterations to reach a trained network structure, and has the best true positive to false positive ratio of the methods that could be implemented for all tested network structures. This method uses the preferred cross-entropy performance metric, and retains the improved accuracy of the second derivative method, Levenberg-Marquardt, without the larger computational time requirement. Additionally, conjugate gradient remains fast for all network sizes and structures tested here, making it ideal for comparison. Thus, for all results presented in this work, we employ a conjugate gradient optimization algorithm.

For additional consistency between methods, the convolutional neural network uses the optimization method found to perform best for NN.

5.5 Results

Our application in nuclear safeguards requires a high accuracy for defect detection but also a low false positive rate. This combination is required not only to satisfy international nonproliferation agreements, but also to minimize the impact of safeguards operations on facility operations and on IAEA budgetary/inspector constraints. All results in this section are based off 10 realizations of each method and averaged across those realizations. Each realization is randomly initialized, trained on a single assembly of 289 rod sub-images, and then deployed in several blind tests to generate the results shown here. The training set includes an expanded defect set of 25 water channels/guide tubes and 1 true defect for all methods. The blind tests have either 1 or 11 true defects across an assembly in addition to the 25 water channels/guide tubes considered approximate defects. The problem is formulated as a binary classification, so only true non-defects are members of the non-defect class, while defects incorporates this expanded set. There

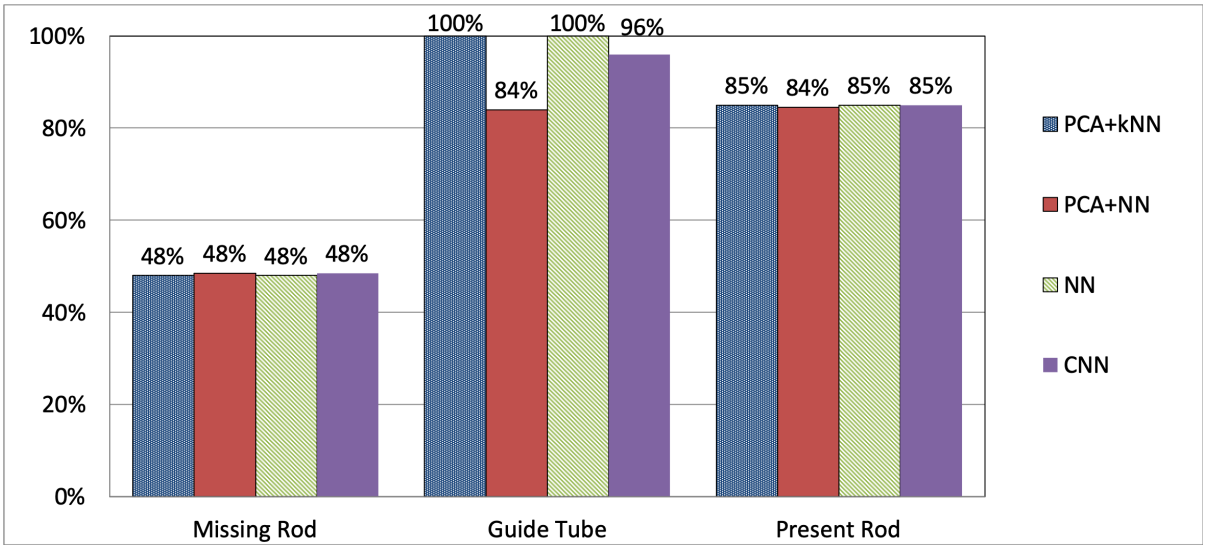


Figure 5.8: Classification results based only on DCVD data.

is no third classification for water channels/guide tubes, but they have been separated in post-processing for better investigation of algorithmic performance on the true defect set.

We present results for each method for both DCVD and GET individually in Figures 5.8 and 5.9. Using a single data source, none of the considered methods combine the desired high defect detection and low false positive rate very well. In general, either defects are classified correctly to a high degree and present rods often misclassified, or present rods are correctly classified and the defects misclassified.

We present results for the classify-then-combine decision-level fusion, as defined in Sections 4.2.1 and 5.3, in Figure 5.10. Note that this figure relies on the threshold value of 0.5. The method that most accurately identifies missing rods with the lowest misclassification of present rods is the PCA+kNN method. Even so, with the percent of present rods classified correctly at only 70%, the false positive rate with this method is too high to be applicable in the industry.

In Figure 5.11, we present the data-level, combine-then-classify fusion results, as defined in Sections 4.2.2 and 5.3. These results do not rely on a threshold, but do include normalization of each data type before data vectors are concatenated for a joint classification. Whereas the percentage of defects classified correctly is lower than in the decision-level fusion, the false positive rates are also much lower. Operationally, these ratios are an improvement over the previous results. Note that the fewer false positives occur, the more likely an inspector is to trust when an algorithm identifies a fuel rod as missing.

5.6 Conclusions

All PCA-based algorithms were tested with as few as 10 principal components. Whereas the combination still yields an increase in accuracy over each data type individually, reducing the

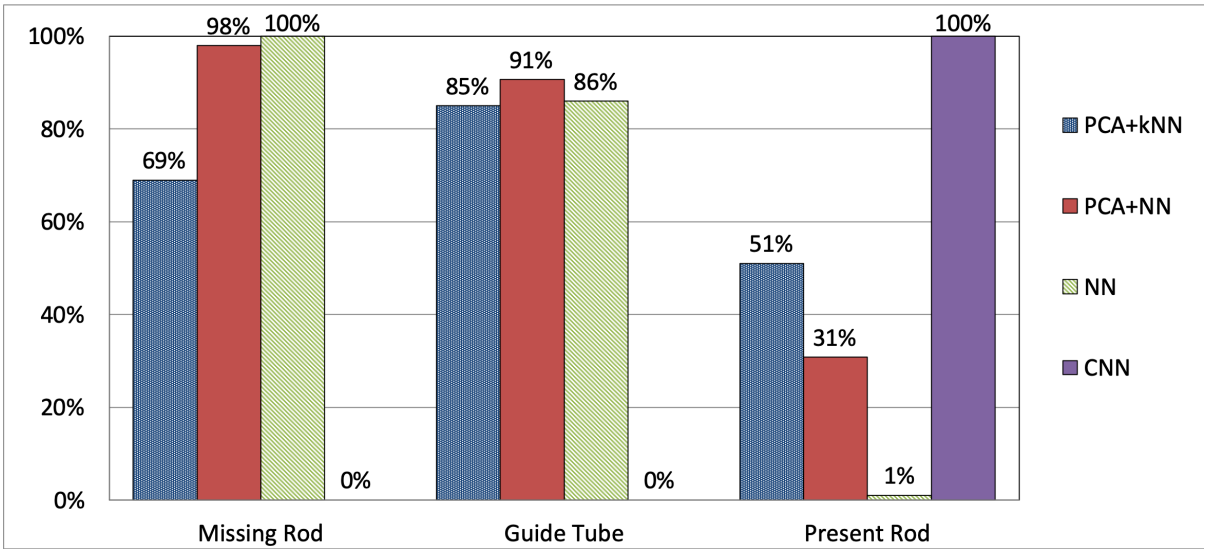


Figure 5.9: Classification results based only on GET data.

number of principal components to 10 or lower increases the number of false positives and decreases the combined defect detection accuracy by about 40% with the same classifier parameters. Choosing 50 principal components, however, combines accuracy and low false positive rates, while still decreasing computational requirements compared to the original fuel rod subimages.

All algorithms are trained on the high burnup, short cooling time scenario because of the high amount of signal in this case. Different training sets are explored in the work of Chapter 6 for Bayesian results. In the blind tests, the trained algorithm is deployed on data from the both the high burnup/short cooling time case as well as on data from the low burnup/long cooling time scenario. The difference in defect detection accuracy is minimal in the combine-then-classify scheme, and is more pronounced in other mechanisms. In the NN and CNN methods, there was no significant change in false positive, false negative, or true positive rates across burnup and cooling time scenarios. The decision-level classify-then-combine scheme was much more sensitive to burnup/cooling time differences between training and deployment.

In general, however, combining the two data sources under any scheme reduced the variation significantly between accuracy on high burnup/short cooling time and low burnup/long cooling time. In all combined methods, the maximum true/false positive rate variation between burnup/cooling time cases is nearly half what is with GET alone. With GET only, NN defect detection rates varied as much as 50% between burnup/cooling time cases, and false positive rates varied as much as 30%. For combined NN with no compression, the differences were closer to 30% and 5%. For both combined PCA methods, PCA+kNN and PCA+NN, the high burnup/short cooling time scenario had false positive rates below 10%, but approaching 50% in the low burnup/long cooling time scenario. For PCA+kNN with GET only, the false positive rate swings from close to 5% in the high burnup/short cooling time scenario to nearly 80% in the

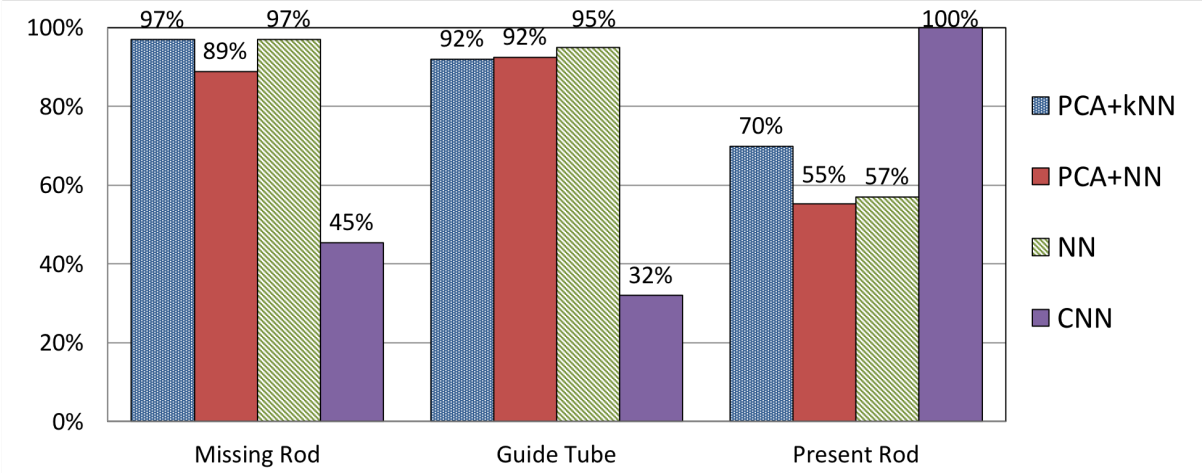


Figure 5.10: Classify-then-combine results for each classification methodology.

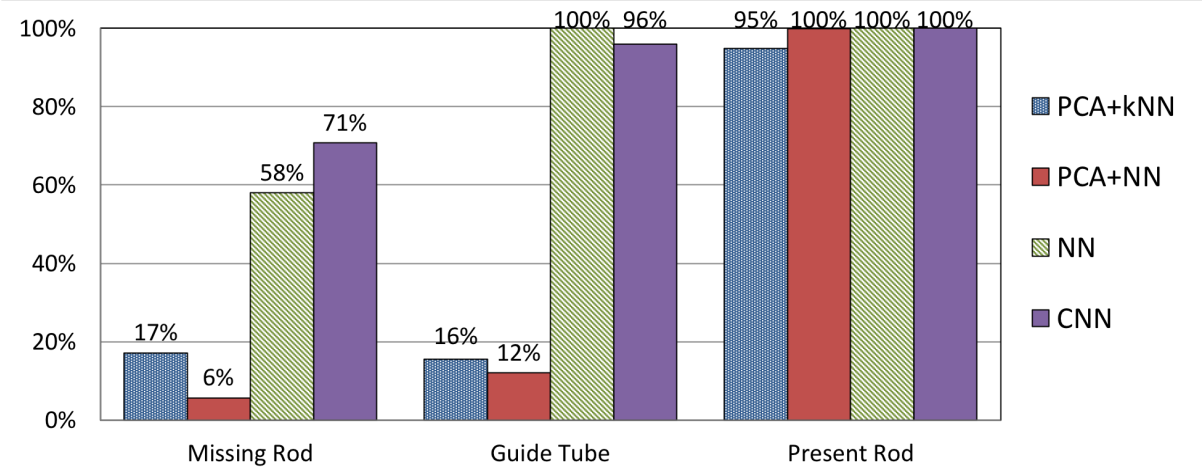


Figure 5.11: Combine-then-classify results for each classification methodology.

low burnup/long cooling time case. While this appears to be an issue of overfitting, good 10-fold cross-validation results across the combined methods indicate that the issue lies elsewhere.

Overall, the best schemes under either data fusion method show an increase in accuracy paired with a lower false positive rate than current industry standard using either GET or DCVD individually in the 17×17 PWR fuel case. GET, the more accurate of the two measurement techniques, has a defect detection rate as low as 20% in the more difficult low burnup/long cooling time case. In the fusion methods tested here, we note defect detection rates as high as 86%.

Chapter 6

Bayesian Fusion Methods and Results

In Section 2.1.6, we defined Bayesian Neighborhood Component Analysis (BNCA). Chapter 4 contains an overview of fusion methods. Chapter 5 covers several approaches to solving the problem of defect detection in spent fuel via fusion of DCVD and GET data. Chapter 5 also details the generation of individual fuel rod sub-images and consistent data sets and this data set is used throughout this chapter as well.

In Chapter 5, we presented non-Bayesian methods and results that could reliably classify defects and non-defects. However, returning to the requirements of our safeguards application, the goal of spent fuel defect detection is not only to reliably identify when defects are present on the single rod level, but to also provide some measure of how accurately we classify defects as well. The non-Bayesian methods from Chapter 5 do not have the ability to quantify confidence in a given classification. We develop here a solution that not only identifies fuel rods as defects or non-defects, but as a Bayesian methodology, is able to estimate the probability that a fuel rod belongs to a particular class. To do this, we develop a novel pipeline algorithm using BNCA with linear opinion pooling (LOP). We first detail the new method and establish the 'pipeline' of data flow through the classifier. We then illustrate testing and implementation in the spent fuel regime.

6.1 BNCA+LOP Pipeline Framework and Algorithm

In BNCA, we learn a Mahalanobis metric–a metric that is represented by a symmetric positive semi-definite matrix $\mathbf{A}$–that maps data points into a new space where every point is most likely to choose as neighbors other points in its same class; i.e., maximize the number of points classified correctly. A full description of this method is provided in Section 2.1.6 and we summarize here only a high-level overview of the relevant equations and definitions necessary for the algorithmic development.

Define $\mathbf{x_i}$ as a data point in a set $\mathbf{X}$, and let y_i be the corresponding categorical label indicating $\mathbf{x_i}$'s true class membership. We define the distance between $\mathbf{x_i}$ and a second point $\mathbf{x_j}$ as

$$d^2_{A^{ij}} = (\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A}(\mathbf{x_i} - \mathbf{x_j}). \tag{6.1}$$

The matrix $\mathbf{A}$ is a Mahalanobis metric; i.e., a symmetric positive semi-definite matrix. In BNCA, we make the assumption that $\mathbf{A}$ can be expressed as linear sum of the eigenvectors of the original data matrix such that

$$\mathbf{A} = \sum_i \gamma_i \mathbf{v}_i \mathbf{v}_i^T. \tag{6.2}$$

The probability that point $\mathbf{x}_i$ selects point $\mathbf{x}_j$ as its neighbor is then expressed as

$$p_{ij} = \frac{\exp(-d^2_{A^{ij}})}{\sum\limits_{t \neq i} \exp(-d^2_{A^{it}})}.$$

Alternately, this can be considered the probability that the true class of point $\mathbf{x}_i$ is the class of point $\mathbf{x}_j$; i.e., $y_i = y_j$. Then define $\{y_i = k\}$ to be the equivalence function such that

$$\{y_i = k\} = \begin{cases} 1, & \text{if } y_i = k \\ 0, & \text{otherwise.} \end{cases}$$

BNCA then defines the probability that the label y_i for a point x_i belongs to class k as

$$p(y_i = k | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{A}) = \frac{\sum\limits_{j \in N_i} 1\{y_j = k\} \exp(-d^2_{A^{ij}})}{\sum\limits_{t \in N_i} \exp(-d^2_{A^{it}})}.$$

Here y_i is the class of point $\mathbf{x}_i$, N_i indicates the neighborhood of points around $\mathbf{x}_i$, with $\mathbf{X}_{N_i}$ and $\mathbf{Y}_{N_i}$ being the points and their classes within that neighborhood. This neighborhood restriction aids in reducing overall computation time of the algorithm and is related to the improvements of FNCA [60] over the original NCA formulation [22].

The definition of $\mathbf{A}$ in Equation (6.2) involves a second assumption. Namely, the assumption that the $\boldsymbol{\gamma}$ vector of γ_i coefficients is sampled from a multi-variate normal distribution. The training phase is actually a variational learning method to approximate this distribution.

Finally, for new data points, $\mathbf{A}$ is sampled by sampling the γ_i values and reconstructing $\mathbf{A}$, using Markov Chain Monte Carlo (MCMC) methods as summarized in Appendix A. The final probability distribution is then estimated for

$$p(y_i = k | \mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{A}).$$

We develop here a BNCA+Fusion Pipeline algorithm for the combination of DCVD and GET data for defect detection and associated confidence. Whereas the classify-then-combine

results of Chapter 5 were promising for combination of DCVD and GET, an advantage offered by BNCA is that of defining the probability that a fuel rod belongs to the defect or non-defect class. BNCA provides not only a classification, but also some estimate of the uncertainty in that classification. This is the main motivation for the separate implementation and development of BNCA. However, its Bayesian nature means that we cannot use the simple fusion techniques of Chapter 5 and must instead develop and implement corresponding Bayesian techniques.

Unique among Bayesian fusion techniques, decision-level fusion permits expert-informed weighting within its data fusion architecture. The choice, then, is primarily between linear or logarithmic opinion pooling (LOP or LgOP, respectively). Discussed in detail in Chapter 4, we reiterate here the primary motivations for each method and our final decision to use linear opinion pooling. The linear and logarithmic opinion pools are some of the most popular decision-level fusion methods and are thus the two mainly considered in our work [46]. One advantage of the logarithmic approach is that it yields a unimodal fusion result, as opposed to potentially multi-modal results with linear opinion pooling [46]. However, in logarithmic opinion pooling, if the density function derived from any data source is zero, the final distribution is also zero. Linear opinion pooling instead incorporates any zero probability densities by averaging them with the densities from other data sources in the final fused posterior distribution [46]. This avoids a single data source 'veto' of information from any other source. In our spent fuel application, each detector is most sensitive to signal from different regions within the assembly. In fact, there are regions of the DCVD images where the signal is blocked by hardware, and the DCVD data source provides little to no information there. Because of this, the averaging of any zero densities in the linear opinion pool framework is more appropriate for our application than the 'veto' power of any given zero distribution in the logarithmic opinion pool framework. We don't want to ignore the zero probability density functions, as that may contain an informative result, but don't want to eliminate the information we receive from any more informative data source.

To implement the BNCA+LOP Pipeline, BNCA is first applied separately to both DCVD and GET for training to learn their respective $\mathbf{A}$ matrices as defined by Equation (6.2) and the original BNCA framework in Section 2.1.6. Probability densities are separately estimated for new data points using the learned $\mathbf{A}$ matrices, and then these densities are fused according to the linear opinion pool method discussed in Section 4.2.5. For a point $\mathbf{x}_i$ with data from M

sources, this takes the form

$$p(y_i = c|\mathbf{x}_i^1, \mathbf{x}_i^2, \ldots, \mathbf{x}_i^M) = \sum_{m=1}^{M} w_m p(y_i|\mathbf{x}_i^m) \tag{6.3}$$

$$= \sum_{m=1}^{M} w_m p(y_i = k|\mathbf{x}_i^m, \mathbf{Y}_{N_i}^m, \mathbf{X}_{N_i}^m, \mathbf{A}^m) \tag{6.4}$$

$$= \sum_{m=1}^{M} w_m \left(\frac{1}{S} \sum_{s=1}^{S} p(y_i|\mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{A}_s^m) \right)_m \tag{6.5}$$

$$= \sum_{m=1}^{M} w_m \left(\frac{1}{S} \sum_{s=1}^{S} \frac{\sum\limits_{j \in N_i} 1\{y_j = c\} \exp(-d^2_{A_s^{ij}})}{\sum\limits_{k \in N_i} \exp(-d^2_{A_s^{ik}})} \right)_m . \tag{6.6}$$

Here $\mathbf{A}_s$ is the MCMC-sampled $\mathbf{A}$ matrix trained on the m^{th} data set, and S is the number of MCMC samples that compose the density approximation.

The Bayesian neighborhood component analysis with linear opinion pool (BNCA+LOP) algorithmic framework is defined in Algorithm 2. Due to the flow of data through a pipeline with BNCA steps preceding the LOP steps, we title the algorithm the BNCA+LOP Pipeline. Note that for space constraints, we omit the variables $\mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{A}$ from our probabilistic definitions in the algorithm overview. The variational update of Equation (2.10) makes it difficult to implement an optimization routine, so we instead iterate to self-consistency; i.e., iterate until the norm of the difference between two iterates is small, here taken to be $||\boldsymbol{\gamma}_k^{DCVD} - \boldsymbol{\gamma}_{k+1}^{DCVD}||_2^2 < 1e-6$. We set the $1e-6$ threshold because we do not observe any increase in performance for smaller thresholds and it increases computational time by an order of magnitude.

6.2 Compression

The BNCA framework is slightly different from original NCA. The $\mathbf{A}$ matrix learned is symmetric positive semi-definite whereas in traditional NCA, the learned $\mathbf{A}$ matrix defines a corresponding symmetric positive semi-definite matrix $\mathbf{Q}$ such that $\mathbf{Q} = \mathbf{A}^T\mathbf{A}$. This small difference in definition directly impacts how compression is implemented. Because there are no constraints on the $\mathbf{A}$ matrix in traditional NCA, compression can be implemented simply by constraining $\mathbf{A}$ to be non-square. That is no longer the case with BNCA, but there are alternatives.

The first assumption of BNCA is that $\mathbf{A}$ is expressed as $\mathbf{A} = \sum\limits_{\ell} \gamma_\ell v_\ell v_\ell^T$, where v_ℓ are the eigenvectors of the data matrix and $\boldsymbol{\gamma}$ the vector of coefficients learned via variational updates as outlined in Section 2.1.6. The outer product guarantees that $\mathbf{A}$ is square and non-negative as long as the eigenvectors are real. One compression option in this framework is to reduce the number of eigenvalues used in the definition of $\mathbf{A}$. This does not reduce the size of $\mathbf{A}$,

Algorithm 2: BNCA+LOP Pipeline

Data: Labeled DCVD and GET fuel rods
Result: Fused distribution $P(y_i = c|x_i^{DCVD}, x_i^{GET})$ for each fuel rod in an assembly
Initialization: $\boldsymbol{\gamma}_0^{DCVD}, \boldsymbol{\gamma}_0^{GET} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$;
begin
  while $||\boldsymbol{\gamma}_k^{DCVD} - \boldsymbol{\gamma}_{k+1}^{DCVD}||_2^2 > 1e-6$ **do**
    Update mean, variance of $\boldsymbol{\gamma}_k^{DCVD}$ distribution according to the variational update outlined in Section 2.1.6;
  end
  while $||\boldsymbol{\gamma}_k^{GET} - \boldsymbol{\gamma}_{k+1}^{GET}||_2^2 > 1e-6$ **do**
    Update mean, variance $\boldsymbol{\gamma}_k^{GET}$ distribution according to the variational update outlined in Section 2.1.6;
  end
  for *iterCount < 5000* **do**
    Randomly sample $\boldsymbol{\gamma}^{DCVD}$ and $\boldsymbol{\gamma}^{GET}$ distributions;
    Calculate $\mathbf{A}_{DCVD}$ and $\mathbf{A}_{GET}$ with their respective $\boldsymbol{\gamma}$;
    Calculate $p(y_i = k|\mathbf{x}_i) = \frac{\sum_{j \in N_i} 1\{y_j = k\} \exp(-d^2_{A^{ij}})}{\sum_{t \in N_i} \exp(-d^2_{A^{it}})}$ for $\mathbf{A}_{DCVD}$, $\mathbf{A}_{GET}$;
    Use pre-defined weights in LOP scheme to combine data sources for each fuel rod $i = 1 : 289$ in an assembly:

$$p(y_i|\mathbf{x}_i^{DCVD}, \mathbf{x}_i^{GET}) = w^i_{DCVD} p(y_i = k|\mathbf{x}_i^{DCVD}, \mathbf{A}_{DCVD}) + w^i_{GET} p(y_i = k|\mathbf{x}_i^{GET}, \mathbf{A}_{GET})$$

  end
  Use kernel density estimator to generate overall distribution from chain.;
  Optional: threshold combined distributions by mean for ultimate binary class membership.
end

but reduces the number of eigenvectors stored throughout the algorithm. Additionally, because the size of $\mathbf{A}$ is unchanged, there is also no compression of new data. Because image data is computationally expensive to work with and store, we would like to be able to compress them as we analyze them. We could also compress the data before training the $\mathbf{A}$ matrix so that it acts directly on compressed data. However, this may cause issues as new data points are added to our space since compression artifacts may obscure relationships that could be learned with an uncompressed $\mathbf{A}$.

A second option for compression, and the method briefly discussed in the original BNCA work by Wang and Tan in [54], is based on the expected value of the BNCA approximation for a new point. In estimating the mappings for new data, Wang and Tan show that the final map $\mathbf{A}$ can be expressed as a weighted multiple of the eigenvector matrix of the original data where the weights are the $\boldsymbol{\gamma}$ coefficients learned in the training stage. Thus, compression in this

algorithm entails learning a size-restricted set of $\boldsymbol{\gamma}$ values and multiplying a compressed matrix of the 'top' eigenvectors–the eigenvectors corresponding to the largest eigenvectors–similar to a singular value decomposition compression. The goal of NCA-based methods, however, is to learn a Mahalanobis metric that maximizes the number of correct classifications in the data. A Mahalanobis metric, by definition, is a symmetric positive semi-definite matrix. This compression algorithm does not necessarily preserve the Mahalanobis properties of $\mathbf{A}$ and is thus undesirable except in cases where computational limitations prohibit training a full-size $\mathbf{A}$. In our application, training an uncompressed $\mathbf{A}$ is not prohibitive, but image data manipulation without compression is still expensive. Additionally, in-field deployment of a trained BNCA+LOP algorithm makes compression desirable for new incoming data.

The final compression option involves a Cholesky decomposition of the BNCA-trained $\mathbf{A}$ matrix followed by compression of the subsequent Cholesky factors. If the diagonals of the Cholesky factors are allowed to be greater than or equal to zero, then the decomposition of $\mathbf{A}$ is guaranteed to exist even though it is only symmetric positive *semi*-definite. Thus, the triangular Cholesky factor, $\mathbf{L}$, decomposes $\mathbf{A}$ such that

$$\mathbf{A} = \mathbf{L}^T\mathbf{L}, \tag{6.7}$$

with the diagonals of $\mathbf{L}$ greater than or equal to zero. The $\mathbf{L}$ matrix is equivalent to the matrix learned in the original NCA formulation with the added Cholesky triangularity requirement. The dimension reduction is carried out on the $\mathbf{L}$ matrix. Due to numerical errors, $\mathbf{A}$ may not always be exactly symmetric positive semi-definite. When this occurs, we replace $\mathbf{A}$ with $(\mathbf{A} + \mathbf{A}^T)/2$. Note that non-uniqueness, though not an issue in our tests, can occur. There are, however, existing methods to address this that are summarized well in [41].

Compressing $\mathbf{L}$ rather than $\mathbf{A}$ retains the necessary properties that define $\mathbf{A}$ as a Mahalanobis metric, while still allowing the mapping of data into a compressed $\mathbf{L}$ space where classification results are optimized. Even if dimension reduction of $\mathbf{L}$ is performed using principal component analysis (PCA), the ultimate (B)NCA learned metric still differs significantly from a PCA carried out directly on the original data.

The original BNCA $\mathbf{A}$-matrix learning process is not affected by the compression, but using the $\mathbf{L}$ matrix enables expressing the original BNCA distance formula from Equation (6.1) as a Euclidean distance; i.e.,

$$d^2_{\mathbf{A}ij} = d^2_{\mathbf{L}ij} = ||\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_j||_2^2 = (\mathbf{x_i} - \mathbf{x_j})^T\mathbf{L}^T\mathbf{L}(\mathbf{x_i} - \mathbf{x_j}).$$

If there is no compression and $\mathbf{A}$ is perfectly described by $\mathbf{L}^T\mathbf{L}$, then this distance metric is the same as previously used. The error induced with compression is expressed as error in the reconstruction of $\mathbf{L}$, thus leading to the distance formula

$$d^2_{\mathbf{A}ij} \approx (\mathbf{x_i} - \mathbf{x_j})^T(\tilde{\mathbf{L}} + \boldsymbol{\mathcal{E}})^T(\tilde{\mathbf{L}} + \boldsymbol{\mathcal{E}})(\mathbf{x_i} - \mathbf{x_j}).$$

Here $\tilde{\mathbf{L}}$ is $\mathbf{L}$ compressed with any method and $\boldsymbol{\mathcal{E}}$ is a compression error matrix where $\boldsymbol{\mathcal{E}}_{ij} = \tilde{\mathbf{L}}_{ij} - \mathbf{L}_{ij}$.

For simplicity, we instead express the error at the $\mathbf{A}$ reconstruction level rather than at the $\mathbf{L}$ compression level; i.e.,

$$d^2_{\mathbf{A}^{ij}} \approx (\mathbf{x_i} - \mathbf{x_j})^T(\mathbf{L}^T\mathbf{L} + \boldsymbol{\varepsilon})(\mathbf{x_i} - \mathbf{x_j}).$$

At the $\mathbf{A}$ reconstruction level, this notation could incorporate any error, including the $\mathbf{L}$ compression error. Thus, we omit the tilde notation and assume perfect compression–and, for example, that multiplication with $\mathbf{L}$ is identical to multiplication with $\tilde{\mathbf{L}}$–with all error captured only in ε. In later steps, this will allow direct comparison between original and reconstructed p_{ij} values. Note also that throughout the following discussion, we omit the set restriction notation of standard BNCA, equivalent to making the neighborhood of a point every other point in the data set.

Because we perform the Cholesky factorization after training, the error need only be propagated through the model in the probability estimations for new data points. In the following derivations, we omit the MCMC step that generates probability estimations and instead focus on calculating the p_{ij} values directly as defined in traditional NCA; see Section 2.1.4.

For a given i and j, we define

$$\tilde{p}_{ij} = \frac{\exp(-(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{L}^T\mathbf{L} + \boldsymbol{\varepsilon})(-\mathbf{x}_i - \mathbf{x}_j))}{\sum\limits_{k \neq i} \exp(-(\mathbf{x}_i - \mathbf{x}_k)^T(\mathbf{L}^T\mathbf{L} + \boldsymbol{\varepsilon})(\mathbf{x}_i - \mathbf{x}_k))}.$$

The numerator, and similarly, the individual terms within the denominator's summation, can be expanded as

$$\exp(-(\mathbf{x}_i - \mathbf{x}_j)^T\mathbf{L}^T\mathbf{L}(\mathbf{x}_i - \mathbf{x_j})\exp(-(\mathbf{x}_i - \mathbf{x}_j)^T\boldsymbol{\varepsilon}(\mathbf{x}_i - \mathbf{x}_j)).$$

For ease of manipulation, we introduce the notation

$$\begin{aligned}\beta_L^{ij} &= \exp(-(\mathbf{x_i} - \mathbf{x}_j)^T\mathbf{L}^T\mathbf{L}(\mathbf{x}_i - \mathbf{x}_j)) \\ \beta_\varepsilon^{ij} &= \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T\boldsymbol{\varepsilon}(\mathbf{x}_i - \mathbf{x}_j)).\end{aligned}$$

Note that each β is a scalar rather than a matrix or vector. Transitioning to this notation, we have

$$\tilde{p}_{ij} = \frac{\beta_L^{ij}\beta_\varepsilon^{ij}}{\sum_{k \neq i}\beta_L^{ik}\beta_\varepsilon^{ik}}. \tag{6.8}$$

In the BNCA framework, this implies that at every step of the MCMC estimation, by sampling the $\boldsymbol{\gamma}$ that defines $\mathbf{A}$, we also sample the corresponding $\mathbf{L}$ and $\boldsymbol{\varepsilon}$. The estimated

distribution for a particular data source then becomes

$$p(y_i = c|\mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{A}) = p(y_i = c|\mathbf{x}_i, \mathbf{Y}_{N_i}, \mathbf{X}_{N_i}, \mathbf{L}) = \frac{1}{T}\sum_{t=1}^{T} \frac{\sum_{j\in N_i} 1\{y_j = c\}\beta_L^{ij}\beta_\varepsilon^{ij}}{\sum_{k\in N_i}\beta_L^{ik}\beta_\varepsilon^{ik}}.$$

Here the outermost summation over T samples estimates the true distribution with MCMC methods.

Finally, we implement this in the LOP step, with the notation that the β terms are derived from GET data and that α_L^{ij} and α_ε^{ij} are the same terms as their β counterparts, defined over DCVD data. It follows that

$$\begin{aligned} &p(y_i = c|x_i^{DCVD}, x_i^{GET}) \\ &\quad = w_{DCVD}\left[\frac{1}{T_1}\sum_{t_1=1}^{T_1} \frac{\sum_{j\in N_i} 1\{y_j = c\}\beta_L^{ij}\beta_\varepsilon^{ij}}{\sum_{k\in N_i}\beta_L^{ik}\beta_\varepsilon^{ik}}\right] + w_{GET}\left[\frac{1}{T_2}\sum_{t_2=1}^{T_2} \frac{\sum_{j\in N_i} 1\{y_j = c\}\alpha_L^{ij}\alpha_\varepsilon^{ij}}{\sum_{k\in N_i}\alpha_L^{ik}\alpha_\varepsilon^{ik}}\right] \end{aligned}$$

where T_1 and T_2 indicate the MCMC chains for DCVD and GET, respectively. While this doesn't explicitly define the difference between the original and compressed distributions, it does provide a compression and error framework not considered in [54].

For practical application of compression limits, we assume that the number of components required to describe $\mathbf{A}$, and thus $\mathbf{L}$, is at most the number of components required to describe the original data. Eigenvalue investigation of the original data to set compression limits is explored in more depth for DCVD and GET separately in Sections 6.4 and 6.5. Once $\mathbf{A}$ is learned, eigenvalues can be re-evaluated in order to assess the accuracy of the compression limits.

6.3 Example Problems and Testing

To implement and test our Pipeline Algorithm 2, we start with a simple problem, mapping three dimensional data to two dimensions [17]. The data is plotted in Figure 6.1. In the original formulation of this problem, there are 300 data points belonging to two classes mapped in three dimensional space, with some overlap between classes. The goal is to separate the two classes by projecting it into a smaller dimension where the separation is most pronounced. With new point, we're also able to calculate the probability that the point belongs to either class. With a single data set, this problem is the same as the example shown in Section 2.1.4. The X, Y and Z components are chosen from uniform random distributions such that for the first class, $X, Y, Z \sim \mathcal{U}(0, 1)$ and for the second class, $X \sim \mathcal{U}(0.25, 1)$, $Y \sim \mathcal{U}(0, 1)$ and $Z \sim \mathcal{U}(0.75, 1)$. The separation between the two classes can be adjusted by modifying the shifts, but this data set has adequate separation to demonstrate both advantages and disadvantages of our method.

To implement our fusion Pipeine Algorithm 2, we need at least two data sets, corresponding

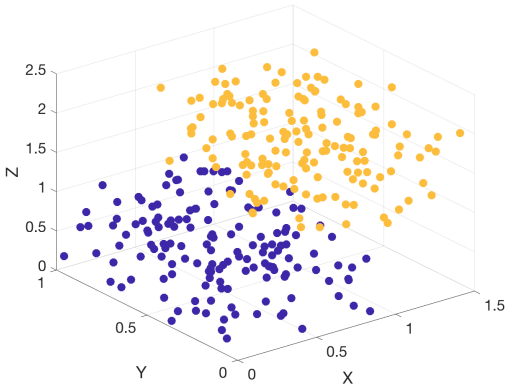


Figure 6.1: Example problem data with classes denoted by color.

to at least two information sources for our 300 observations. We explore two options for a secondary data set. The first scenario comprises two separate data sets each giving noisy XYZ components for the same 300 data points. The X, Y and Z components are selected as in the first set from Figure 6.1, with $X, Y, Z \sim \mathcal{U}(0,1)$ for the first class, and $X \sim \mathcal{U}(0.25,1)$, $Y \sim \mathcal{U}(0,1)$ and $Z \sim \mathcal{U}(0.75,1)$ for the second class. In the second data set, we add 5% Gaussian noise to each point in both classes. With this example, we investigate how noise between data sets impacts our final classification abilities to test the case of aggregating multiple measurements of a single item.

In the second scenario, one data set comprises the XYZ information for the points, while the second data set comprises RGB color map data. With this data set, we investigate how the incorporation of a new data type impacts classification abilities. This second scenario is plotted in Figure 6.2. The separation between classes is the same as Figure 6.1, but the color map in Figure 6.2 is now derived from the second information source. The X, Y, and Z data set is the same as in the original problem, with $X, Y, Z \sim \mathcal{U}(0,1)$ for the first and $X \sim \mathcal{U}(0.25,1)$, $Y \sim \mathcal{U}(0,1)$ and $Z \sim \mathcal{U}(0.75,1)$ for the second class. The RGB values are selected uniformly randomly between 0 and 1 for all three color definitions. However, to separate the classes in our RGB data, the blue component of the first class is randomly selected in the interval $[0, 0.1]$, while the red component of the second class is sampled on the interval $[0, 0.1]$ In this way, colors in the 'red' region indicate the first class of points while colors in the 'blue' region indicate the point belongs to the second class.

In all iterations of this example problem, our BNCA $\mathbf{A}$ matrix is learned using 250 of the 300 data points, and 50 points are used as blind test sets on which accuracy rates are calculated. Both training and testing sets are evenly divided between the two classes. The $\boldsymbol{\gamma}$ values that define the $\mathbf{A}$ matrix are sampled 5000 times for each data set to generate the approximate distributions to be combined with the linear opinion pool. Because the data itself is also randomly computed, the data and algorithm are initialized and run 100 times, with final

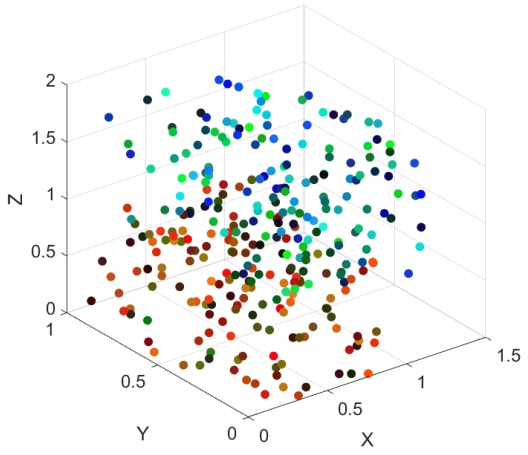


Figure 6.2: Example problem, XYZ and RGB data sets.

average results shown in Table 6.1. The large number of initializations and runs is not feasible for our larger DCVD and GET data sets but provides extensive testing. For comparison, we calculate these classification accuracies both with BNCA on the individual data sources and our BNCA+LOP Pipeline algorithm. If a point has a mean probability greater than 0.5 for a class assignment, we assign it to that class.

For the XYZ data, the initial iterate is always the PCA mapping into 2D plus machine error noise (1×10^{-16}). RGB data cannot sensibly be mapped into 2D, as it changes the color definitions. Instead, for the RGB data, the initial iterate is a PCA mapping into another 3D space with machine error noise again added. This will additionally allow for testing in the case where compression between data differs.

In Table 6.1, we present incorrect classification rates for both training and testing data for both the noisy XYZ and the XYZ with RGB test problem. In the noisy XYZ problem, we do not note significant improvement in the combination of methods when 10% Gaussian noise is added to the original XYZ data to generate the secondary data set. On average, we improve our classification accuracy on the test set by 0.04%, and only 0.1 % on the training set by implementing the pipeline algorithm. However, this method provides a way to aggregate two noisy data sources without any loss of separability between the two classes present in each data set.

In both the training data and testing data, we note and approximately 3% decrease in the error rate when we use the BNCA+LOP Pipeline Algorithm 2 on the XYZ+RGB problem. Unfortunately, the base rate of error classification for each data set without combination is high. This may be the result of implementation, as the BNCA algorithm iterates to self consistency with a fixed maximal iteration count to learn an adequate $\mathbf{A}$ mapping, rather than running until some optimization criteria is met. The relative consistency in error rates between training and testing data indicates that our current implementation slightly underfits our data rather

Table 6.1: Incorrect classification rates for separate and combined data sources.

	XYZ Data	XYZ (noisy) Data	BNCA+LOP Pipeline
Training Data	14.1%	14.1 %	14.0 %
Testing Data	10.3 %	10.2 %	10.2 %
	XYZ Data	RGB Data	BNCA+LOP Pipeline
Training Data	26.2%	21.8 %	17.8 %
Testing Data	21.8 %	15.7 %	12.7%

than overfitting it. Additionally, these results take into account the 25 nearest neighbors of a given point to assess p_{ij} values and subsequent classification. As in kNN, the number of neighbors have a significant impact on final classification accuracy, explored more thoroughly in Section 6.7.

If the impact of the fusion pipeline was minimal, the lower bound on classification inaccuracy would simply be the lowest of the two data sources considered. However, the resulting classification inaccuracies are lower than with either individual data source. This indicates that our BNCA+LOP Pipeline can overcome limitations of individual data sources for improved overall classification results. This is especially true in the case of XYZ+RGB complementary data and is somewhat more limited in the case of two noisy data sets taken on the same points.

6.4 DCVD with BNCA

Before moving directly to BNCA+LOP Pipeline results when we fuse DCVD and GET, we first need to investigate the applicability of neighborhood component-based methods for each data source separately. In this section, we focus on compression limitations with DCVD data as well as some initial results from BNCA applied to DCVD data. In Section 6.5, we perform the same analysis of GET data.

From earlier work with DCVD with PCA in Section 5.2.1, we know that a DCVD fuel rod sub-image can be accurately reproduced with as few as 50 principal components. This is the first assumption we need to test with the NCA and thus BNCA. It may be the case that even with fewer components and a worse reconstruction of the original image, we may still retain good classification ability. In the images shown in Figure 6.3, we retain 16 components as compared with the original 529. We also show in Figure 6.3 the mapping into 16 dimensions for the identity mapping to 16 components by multiplication with the identity matrix $\mathbf{I}$, the PCA mapping into 16 components, and the NCA mapping into 16 components. The colormap scales for each mapping are different, but extend from the rounded integer minimum and maximum of the intensities. These figures then display the maximum variance between defect and non-defect and are comparable for interpretation, even though the colormaps and scales differ. The

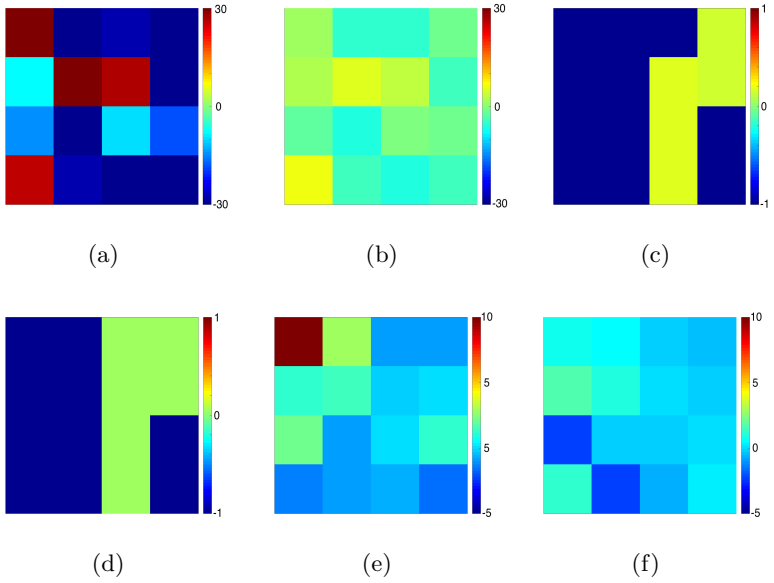


Figure 6.3: Compression of DCVD data to 16 components for (a) NCA-mapped defect and (b) NCA-mapped non-defect; (c) Identity-mapped defect and (d) Identity-mapped non-defect; (e) PCA-mapped defect and (f) PCA-mapped non-defect.

defect and non-defect are visually very different for the NCA implementation, even with only 16 components, whereas the PCA mappings are more similar between defect and non-defect. The key difference is in the number of components that are a different value between the defect and non-defect for each mapping in Figure 6.3. Note, though, that this is only for a single defect and non-defect comparison. Overall, NCA still struggles with this small number of components in the compression stage of our algorithm. Nevertheless, in comparing the NCA mappings with the identity and PCA mappings, we see that NCA methods visually improve the differences between defect and non-defects in the following sense. In the identity map, only one grid location; i.e., one component, is significantly different between defect and non-defect. In the PCA mapping, the average component value is similar in both images with only one component differing significantly between defect and non-defect. In the NCA mapping, a majority of the components are significantly different between the defect and non-defect mapping. This provides confidence that NCA-based methods may improve upon the results determined with PCA-based methods in Chapter 5.

To more adequately assess the compression capacity before significant data loss occurs, we investigate the eigenvalues of all three original DCVD images. Because we have more components than observations, our data matrix is non-square and we plot in Figure 6.4 the singular values for each original DCVD image.

For all three original DCVD images, the singular values are at or below 1 by the 80^{th} singular value. This also specifies the location where the singular values transition from a steep decline

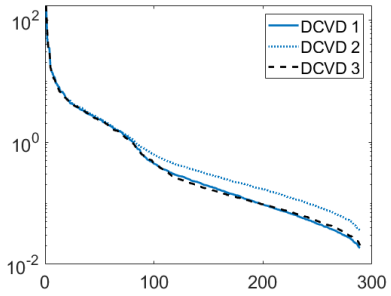


Figure 6.4: Log scale eigenvalue plot for DCVD data.

to a more gradual one. For BNCA and BNCA+LOP Pipeline classification, we use this dropoff in singular values to identify a possible compression size from the original data vectors of length 529 to length 80, making the corresponding compressed BNCA $\mathbf{L}$ matrix from Equations (6.7) and (6.8) size 529×80. This is a significant decrease in our data size. Our training set is only comprised of 289 fuel rods, which means that we have more parameters than observations with the original 529-component vectors for each rod. Compressing down to 80 components yields an over-determined system to replace the original under-determined data set, and could facilitate solving our classification problem.

Compression within the BNCA framework, as previously discussed, requires the compression of a matrix $\mathbf{A}$ defined by Equation (6.2) and learned via variational learning of the mean and variance of the coefficient vector $\boldsymbol{\gamma}$ from Equation (6.2) by the process outlined in Section 2.1.6. We provide a complete discussion of training sets, testing sets and results in Section 6.7, but we plot the eigenvalues of a learned $\mathbf{A}$ in Figure 6.5. The training set here contains 36 defects–25 guide tubes and 11 true defects–and 253 non-defects, all from a single assembly with Case 1 high burnup and short cooling time as defined in Table 5.1. This eigenvalue plot shows a slower decay than the eigenvalues of the original data. In the original eigenvalue investigation, the compression threshold was at the number of components by which the singular values had decreased by two orders of magnitude. There is a very small initial decay, followed by a long slow decay, and the decrease by two orders of magnitude in eigenvalues does not occur until after 500 eigenvalues. However, we maintain the compression limit of 80 components from our initial investigation of the DCVD data, as eigenvalue investigation of $\mathbf{A}_{DCVD}$ from Equation (6.2) learned with BNCA on only DCVD data shows all eigenvalues on the same scale as where we define the singular value threshold in the original data. Additionally, the lack of decrease between eigenvalues of $\mathbf{A}_{DCVD}$ indicates limited improvements in compression for a higher number of components.

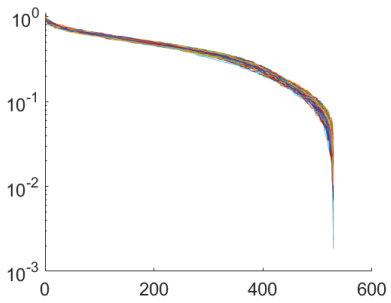


Figure 6.5: Eigenvalues for $\mathbf{A}_{DCVD}$ trained under with high burnup/short cooling time data with 36 overall defect (11 true defects) and 253 non-defects for 500 randomly sampled $\boldsymbol{\gamma}$.

6.5 GET with BNCA

Similar to DCVD, before presenting results with the BNCA+LOP Pipeline, we first assess the applicability of neighborhood component-based analysis to GET data as well as explore the limits of 'compressability' of GET data. We start by investigating differences between PCA and NCA mappings to 16 components. We plot the resulting maps for a representative defect and non-defect in Figure 6.6. The mapping allow for a simple visual test to assess potential for improvement with NCA- versus PCA-based approaches. Unlike in Figure 6.3 (a) and (b), there is limited contrast between the NCA mappings of defects and non-defects with the 16 component maps in Figure 6.6 (a) and (b). The average component value is consistent between the two, and this is after optimizing the NCA functional Equation (2.3). Most components in the identity mapping to 16 components are zero in subfigures (c) and (d), with only 3 components non-zero in both defects and non-defects. However, the non-zero components are different in the defect and non-defect in this mapping. Finally, the PCA mappings into 16 dimensions for a GET defect and non-defect are in Figure 6.6 (e) and (f). Similar to NCA, the average component value is similar between defect and non-defect with few significant differences between individual component values. While these visual tests are partly a result of extreme compression, they also indicate that PCA and NCA-based methods may struggle with GET data classification. This is reinforced by the PCA results in Section 5.5.

We next assess the compression limits possible with GET data using eigenvalue information. We plot the singular values of each fuel assembly in Figure 6.7. There's an initial drop within the first 20 singular values, but then a more gradual decrease. There is also a noticeable difference between the two burnup/cooling time scenarios; the high information content of the high burnup/short cooling time scenario data yields much larger eigenvalues than the lower signal low burnup/long cooling time. Investigation of the eigenvalues of $\mathbf{X}^T\mathbf{X}$, where $\mathbf{X}$ is the data matrix for any of the GET images, show an eigenvalue decrease to 1×10^{-7} after the 285^{th} component. In the DCVD case, the compression cutoff was set to be 80 components because the singular values decrease below 10^0 at that point. In the GET case, for the low burnup/long cooling time

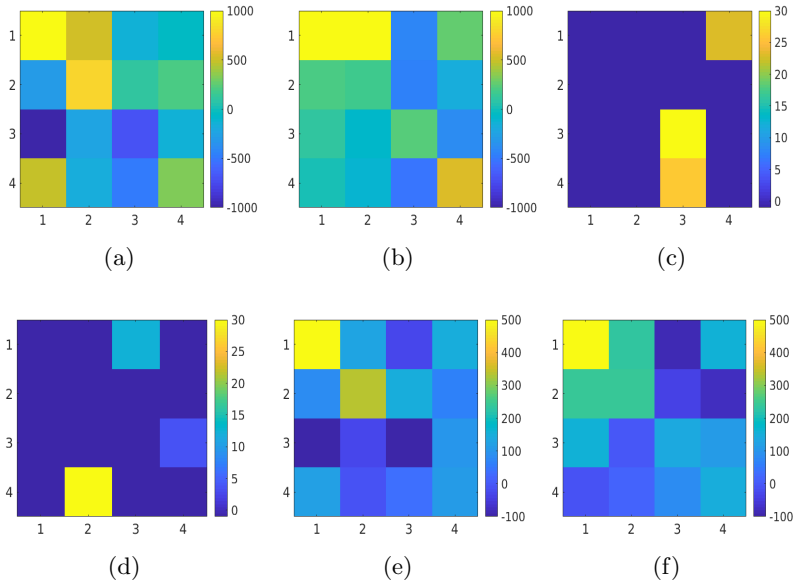


Figure 6.6: Compression of GET data to 16 components for (a) NCA-mapped defect and (b) NCA-mapped non-defect; (c) Identity-mapped defect and (d) Identity-mapped non-defect; (e) PCA-mapped defect and (f) PCA-mapped non-defect.

scenario, the singular values reach 10^0 after the 285^{th} component. In the high burnup/short cooling time scenario, the singular values don't reach 10^0 until the 289^{th} component. Because there is no significant difference in singular values after approximately the 250^{th} component, we retain 245 components for compression in the BNCA+LOP Pipeline Algorithm 2. This enables for compression of the data to approximately half its original dimension.

Figure 6.8 shows the eigenvalues of $\mathbf{A}_{GET}$, the Equation (6.2) $\mathbf{A}$ learned with GET data, as trained by high burnup/short cooling time Case 1 data with 36 defects and 253 non-defects. We note in this case that the eigenvalue decomposition has a steep initial drop off followed by a long stagnation. The stagnation period starts near the 80 component compression threshold of the original DCVD data. This indicates that we may be able to shrink the compression dimension

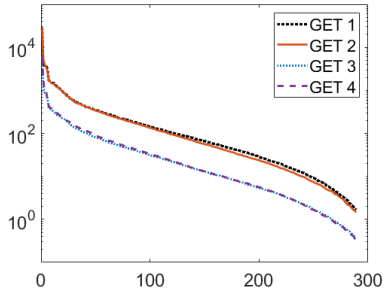


Figure 6.7: Log scale eigenvalue plot for GET data.

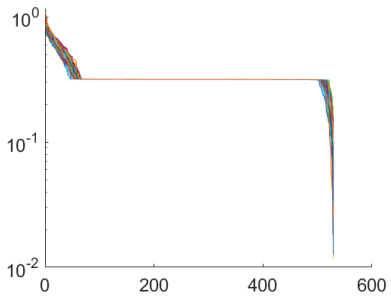


Figure 6.8: Eigenvalues for $\mathbf{A}_{GET}$ trained under high burnup/short cooling time GET data with 36 defects and 253 non-defects for 500 randomly sampled $\boldsymbol{\gamma}$.

well below the 245 required components indicated by investigation of the original data. We maintain our original 245 component compression for consistent comparison to the PCA-based work in Chapter 5.

6.6 Spent Fuel Fusion with BNCA+LOP Pipeline Framework

Now that compression limits from each data source have been identified, we can implement our developed fusion method, the BNCA+LOP Pipeline algorithm 2. However, the weights from Equation (6.3) must be determined for the linear opinion pool step. In our previous work, weighting was informed by 'expert opinion' about our two detectors. Expert opinion indicates that when a fuel rod is near the outside edges of the assembly, GET is the more accurate/sensitive detector, whereas near the center, DCVD is the more accurate/sensitive detector. We discuss techniques to incorporate these detector sensitivities in the following sections.

There are three weighting schemes applicable to our problem. The first is naive weighting, where each fuel rod is equally informed by the two data sources; i.e., the weights in the linear opinion pool are all equal to $\frac{1}{2}$. The other two weighting schemes are both informed by expert opinion, using distance from the center of the assembly to define the weights. However, the distance must be explicitly defined. In Chapter 5, we defined the distance as a standard Euclidean norm. Here, we additionally consider Chebychev distances. Each of these metrics quantify an aspect of a detector method. DCVD's circular field of view is better quantified with a Euclidean metric whereas GET's implementation has roughly equal sensitivity for each rod in any given 'ring' of the assembly and is better quantified by a Chebychev metric.

6.6.1 Naive Weighting

In naive weighting, the probability distributions estimated using BNCA for each data source are equally weighted in the linear opinion pool combination step in our Pipeline Algorithm 2. This is called 'naive' weighting because it does not rely on any expert opinion or pre-existing information about the sensitivity of the data sources. For all the data in Section 6.7, the naive

weights are equal for each data source; i.e., the weights are 0.5 for each data source. The mean of the combined distribution, μ_c from the BNCA+LOP Pipeline is then

$$\mu_c = 0.5\mu_{GET} + 0.5\mu_{DCVD}.$$

This satisfies requirements of the original linear opinion pool framework defined in Equation (4.1) that $\sum_{i=1}^{M} w_i = 1$, where M is the number of data sources. Additionally, the probability resulting from each data source will always be between 0 and 1.

6.6.2 Euclidean Distance Expert-Informed Weighting

In Euclidean weighting, expert opinion does inform the weights. The nearer a fuel rod is to the center of the assembly, the more influential the DCVD-driven distribution estimation. In this method, the distance from center of assembly to edge is calculated using a standard Euclidean distance,

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_k |\mathbf{x}_k - \mathbf{y}_k|^2 \right)^{1/2}, \tag{6.9}$$

where x_k, y_k indicate the k^{th} component of the data vectors $\mathbf{x}$ and $\mathbf{y}$. Distances between fuel rods are calculated as the distance between the pixel centroid of a given rod sub-image and the pixel centroid of the center-most fuel rod in the assembly. One underlying assumption is that the assembly is correctly straightened so that the weights are not also impacted by image orientation. The distance between fuel rod and assembly center are normalized by the maximum possible distance between fuel rod centroid and assembly center. For the Euclidean distance metric, this maximum occurs at the four outer corners of the assembly.

The normalized distance between fuel rod center and assembly center is the GET weight, w_{GET}, in the linear opinion pool. The DCVD weights are subsequently defined to be $w_{DCVD} = 1 - w_{GET}$. The linear opinion pool condition $\sum_{i=1}^{M} w_i = 1$ is thus satisfied for every fuel rod in this weighting scheme. This yields the weights in Figure 6.9 where the colormap is the direct mapping to values of the weights for each fuel rod in the 17×17 assembly grid. The outer corners are most heavily weighted towards GET, and the center most heavily weighted towards DCVD. This quantifies the roughly circular field of view of the DCVD detector.

6.6.3 Chebychev Distance Expert-Informed Weighting

In this expert-informed weighting scheme, the distance is taken to be the maximum distance between components; i.e., the Chebychev distance defined as

$$d(x, y) = \max_i |x_i - y_i|, \tag{6.10}$$

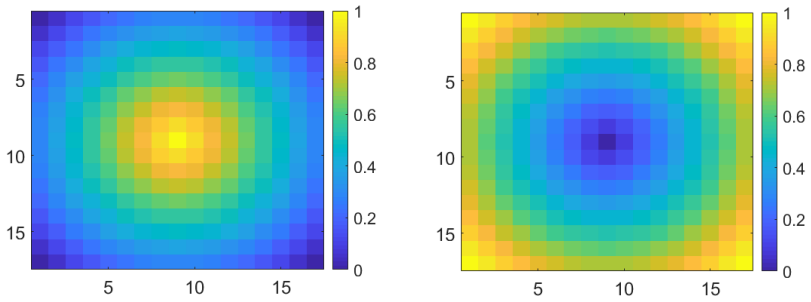


Figure 6.9: Linear opinion pool weights for a 17×17 assembly using Euclidean norm distance metric for (a) DCVD and (b) GET. The weights for each fuel rod are directly mapped to $[0, 1]$ with the colormap axis for every rod location within the 17×17 grid.

or the maximum difference between components. Chebychev distance is a special case of a Minkowski distance,

$$d(x, y) = \left(\sum_k |x_k - x_k|^p \right)^{1/p},$$

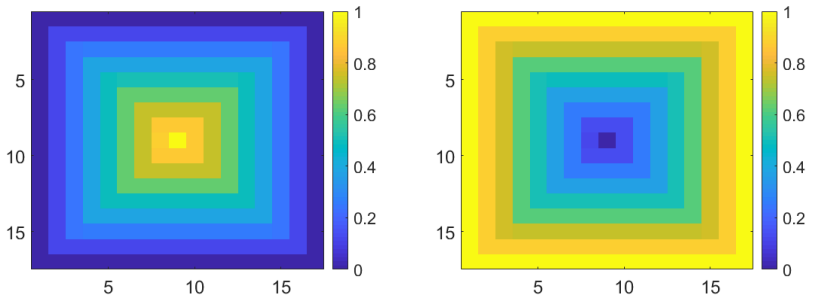
with $p = \infty$. This is equivalent to being the maximum distance between components of two vectors. In our assembly diagrams, this corresponds to each fuel rod along a particular 'ring' of the assembly having the same weight. This is illustrated in Figure 6.10. The GET weight for a given rod is defined directly as the distance between the rod's sub-image centroid and the central pixel of the center-most rod in the assembly. The weights are normalized by the maximal distance within an assembly between a fuel rod and the center-most pixel of the assembly. The DCVD weights, w_{DCVD} are then $1 - w_{GET}$. Because the GET weights are normalized, they are always less than or equal to 1. Defining w_{DCVD} this way ensures that the linear opinion pool requirement that $W_{GET} + w_{DCVD} = 1$ for all fuel rods is satisfied.

The maximal distance between rod and assembly center occurs equivalently at any fuel rod in the outermost ring of the assembly. Given that GET is generally equally sensitive to signal from all fuel rods in a given ring of the assembly, the Chebychev distance accurately reflects GET's capabilities.

6.7 Results

Throughout this section, the legend in Table 6.2 defines consistent notation for all graphical results. The proposed and tested training sets are compiled in Table 6.3

We define compression limits for each data source based on the eigenvalue investigation in Sections 6.4 and 6.5, but aligns well with eigenvalue investigations of learned **A** matrices and corresponding **L** matrices as defined in Section 6.2. For DCVD we retain 80 components/parameters when the data is compressed and for GET we retain 245 components.



(a) (b)

Figure 6.10: Linear opinion pool weights for a 17×17 assembly using Chebychev norm distance metric for (a) DCVD and (b) GET. The weights for each fuel rod are directly mapped with the colormap axis for every rod location within the 17×17 grid.

The major considerations when implementing the BNCA+LOP Pipeline in Algorithm 2 are the prior distributions for the $\boldsymbol{\gamma}$ vectors, the training set under consideration and the number of neighbors to be considered when each p_i value is calculated within the original BNCA framework.

Established in Section 2.1.6, the current standard for $\boldsymbol{\gamma}$ prior distributions are normal distributions $\mathcal{N}(0,1)$. We apply $\mathcal{N}(0,1)$ in our investigation. The kernel density estimates for the distribution of $\boldsymbol{\gamma}$ values for each of the uncompressed 529 components are plotted in Figure 6.11 for both DCVD and GET training data. The y-axis is reduced for visibility, as we plot the $\boldsymbol{\gamma}$ weight for each of 529 components. However, a magnified view of the higher peaks are shown in the cutout in the upper left corner of each image. The average mean of all $\boldsymbol{\gamma}$ distributions is 0.03 for DCVD with an average variance 0.07, while for GET $\boldsymbol{\gamma}$, these are 0.1 and 0.01 respectively. This is also indicative of appropriate compression limits–much fewer components have non-trivial weights in GET than in DCVD, indicating that significantly more components are necessary for data manipulation.

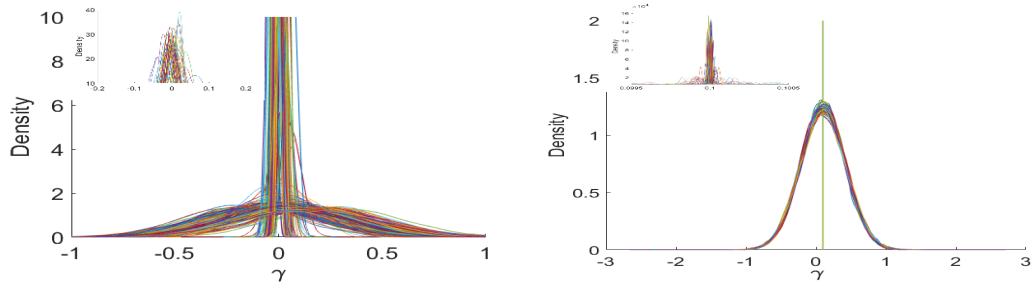
Training Set Variations

In the results of Chapter 5, each classification method was trained on a data set of 289 fuel rods with between 25 and 36 'defects' and this defect set contained both water channels/guide tubes

Table 6.2: BNCA+LOP Pipeline Results Legend.

Class	Graph legend
Non-defect	●
Guide Tube	▲
Defect	×

as well as true defects. There are several possible variations that we test with our algorithm to investigate the impact of the training data set. The DCVD data set contains two images, each with one true defect and 25 water channels/guide tubes. We also have one DCVD image with no true defects and only 25 water channels/guide tubes. In the GET data set, we have four images containing 11 true defects and 25 water channels/guide tubes. The first training set (TR1) is a modified DCVD image of 289 fuel rods, where 11 defects are simulated in the location of the 11 GET defects using existing true DCVD defects with 5% Gaussian noise added. TR1 uses GET data with high burnup/low cooling time; i.e., Case 1 in Table 5.1. TR2 is the same setup for DCVD but uses GET data with Case 2 burnup/cooling time, as defined in Table 5.1, for training. TR3 uses DCVD image with a single true defect in location 214 and Case 1 GET data with a simulated defect in the same location with all other original GET defects replaced by present rods. TR4 again uses the DCVD image with a single true defect in location 214 but uses Case 2 GET data for training. Note that for each TR set, the DCVD and GET defect locations are consistent. As currently implemented, this is a requirement for this algorithm, as the LOP step combines two data sources for the same item. We summarize the TR sets in Table 6.3, where the total defect count is the number of true defects in each data set combined with the standard 25 water channels/guide tubes of a 17×17 fuel assembly that we consider to approximate defects. The data sets denoted 'Modified' are the data sets that required adjusting the location and number of defects in order to match the second data set. We provide a more thorough discussion of this practice in Section 5.1.



(a) (b)

Figure 6.11: γ densities after training for all 529 components of (a) DCVD and (b) GET.

Table 6.3: Training set variations for implementation. Modification here denotes that defect locations are adjusted within the original data set to match the second data source.

	DCVD Data	GET Data	Total defect count
TR1	Modified DCVD	Case 1 GET	36
TR2	Modified DCVD	Case 2 GET	36
TR3	True defect at 214	Case 1 Modified GET	26
TR4	True defect at 214	Case 2 Modified GET	26

Blind Tests

We perform testing with a set of blind tests where we have known class labels for a data set, but the algorithm only sees the data itself. There are 8 blind tests overall, each with one true defect, 25 guide tubes, and 263 non-defects. For TR1 and TR2, the blind tests include the training data for TR3 and TR4. For TR3 and TR4, the test sets TR1 and TR2 are included as blind tests with 11 defects, 25 guide tubes, and 253 non-defects. We summarize the blind tests in Table 6.4. As in Table 6.3, the data sets denoted 'Modified' are the data sets that require adjusting the location and number of defects in order to match the second data set. We provide a more thorough discussion of this practice in Section 5.1. For brevity, in the following sections we summarize results for only a representative subset of employed blind tests. For results using other blind tests, see Appendix C.

Table 6.4: Blind test parameters. Modification here denotes that defect locations are adjusted within the original data set to match the second data source.

	DCVD Data	GET Data	Total defect count
Blind 1	True defect at 214	Case 1 Modified GET	26
Blind 2	True defect at 105	Case 1 Modified GET	26
Blind 3	True defect at 214	Case 1 Modified GET	26
Blind 4	True defect at 105	Case 1 Modified GET	26
Blind 5	True defect at 214	Case 2 Modified GET	26
Blind 6	True defect at 105	Case 2 Modified GET	26
Blind 7	True defect at 214	Case 2 Modified GET	26
Blind 8	True defect at 105	Case 2 Modified GET	26
Blind 1 (TR3, 4)	Modified DCVD	Case 1 GET	36
Blind 5 (TR3, 4)	Modified DCVD	Case 2 GET	36

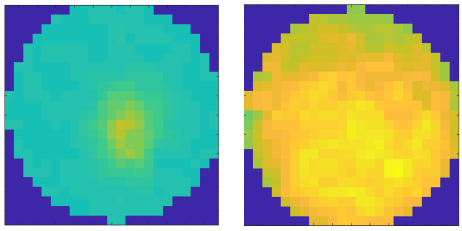
Neighborhoods

Another major consideration in the BNCA+LOP Pipeline implementation in Algorithm 2 is the number of neighbors used in calculation of each p_i distribution defined by Equation (6.1). This number is not necessarily the same for each data source, but we add this constraint for consistency and comparability across results. We implemented tests for 1, 2, 5 and 25 nearest neighbors (NN). Tests on larger numbers of neighbors are prohibited by computational constraints. This is primarily due to the matrix multiplication in the variational parameter update step from Equation (2.10). Because the weight matrices to update the mean, variance and variational parameter are defined with the constructed neighborhoods of every point in the data set, increasing the number of neighbors causes storage errors, as well as large non-sparse matrix multiplication, after approximately 25 neighbors when applied to the non-compressed original image data. Training for 25 nearest neighbors is on the order of minutes, but repeated sampling for large numbers of neighbors in the distribution estimation and storage of the resulting matrix multiplications quickly become prohibitive after 25 nearest neighbors. For each weighting scheme, results in the following sections are presented for 25 NN. We summarize results for 2 NN, 5 NN, and 10 NN in Appendix C. Additionally, above 25 neighbors, the contribution of individual neighbors quickly approaches zero.

6.7.1 Individual DCVD and GET Results

For 25 nearest neighbors, we show the BNCA results on the training data for each data type individually in Figures 6.13(a) and 6.15(b) for TR1. Blind test 1, with the same burnup/cooling time conditions as TR1, is plotted alongside in 6.13(b) and 6.15(b), respectively. We plot in these figures the means of the estimated probability distributions $p(y_i = defect|\mathbf{x}_i)$ from Equation (6.1) for each fuel rod; i.e., the probability that a rod is a defect. The center-most guide tube is consistently marked as a non-defect. This primarily because of consequences due to being at the assembly center. The weighting scheme relies exclusively on DCVD to classify the rod, and this fuel rod has a visibly different signal in DCVD than other guide tubes based on assembly geometry that makes it appear more similar to non-defects. This is demonstrated in Figure 6.12 with the center fuel rod in subfigure (a) and a more representative guide tube in subfigure (b). This signal difference is potentially due to hardware blocking the broader signal and leading to the small bright region in the center of Figure 6.12(a).

In Figure 6.13, we plot the distribution means for membership in the defect class as defined by Equation (6.1). Whereas DCVD is able to separate all the guide tubes from the non-defects, the defects are less consistently separated from the non-defects in Figure 6.13. In the binary classification scheme, guide tubes are classified as defects. While consistently identified separately from non-defects, their distribution means are frequently below 0.5, indicating that they do not belong completely in the defect class or the non-defect class. This mixed result yields artificially low true positive rates for this method–the guide tubes are in fact separate from true defects, but under the binary classification scheme, this is not adequately quantified and

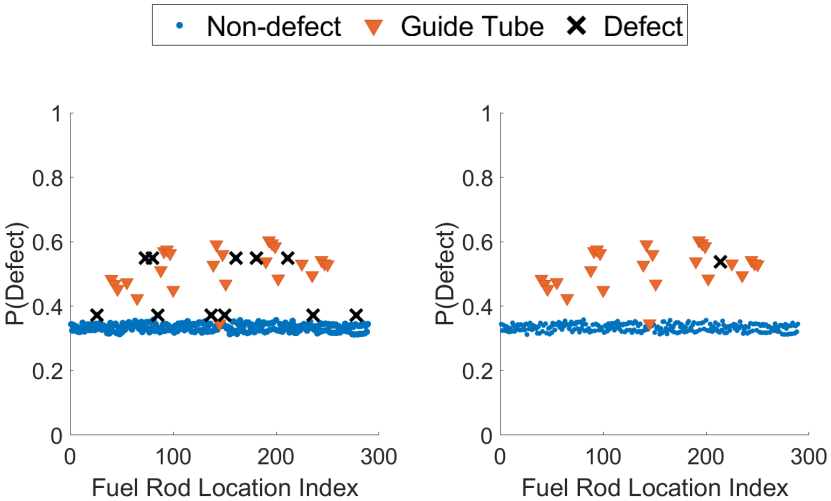


(a) (b)

Figure 6.12: Fuel rod sub-image for (a) non-standard guide tube 145 and (b) representative guide tube 91.

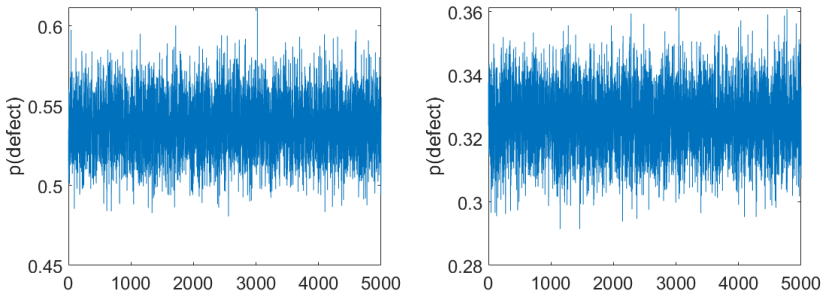
detracts from the defect detection rate.

In Figure 6.13(a), two distinct classes of true defects appear to cluster around 0.4 and 0.6. These correspond to defects nearer the inner ring and defects nearer the outer ring of the assembly. Those true defects nearer the assembly center are consistently identified with distribution means above 0.5 whereas those farther from the assembly center have means closer to 0.4. Because this is DCVD only, there is no implemented weighting forcing this structure. This is a strong indication that our weighting schemes are well-informed as far as regions of DCVD sensitivity, as this aligns with expert opinion that DCVD is better suited to defect detection near the assembly center.



(a) (b)

Figure 6.13: DCVD BNCA $p(y_i = defect|x_i)$ distribution means from Equation (6.1) for (a) TR1 and (b) Blind Test 1 with true defect in rod location 214.



(a) (b)

Figure 6.14: DCVD BNCA MCMC random sampling chains for (a) Blind 1 defect and (b) representative Blind 1 non-defect.

In Figure 6.14, we plot the MCMC random sampling chains for a defect in Blind 1 and an example non-defect. We provide more details about MCMC sampling in Appendix A. Both chains demonstrate good mixing and exploration of the parameter space within the region we expect for a defect/non-defect. That is, for the defect, the chain explores the space of $p(y_i = defect) > 0.5$ on the x-axis and the non-defect chain explores only $p(y_i = defect) < 0.5$. To deal with burn-in, when we perform kernel density estimation to derive the distributions in Figure 6.19, we omit the first 10% of each chain.

We plot the GET results for TR1 and Blind Test 1 in Figure 6.15(a) and (b), respectively. In Figure 6.15(a), the distribution means are forced to zero for every point in the training set. To test for overfitting, we reduced the number of iterations in the BNCA framework to as low as 10, with no change in the resulting distributions. We also tested three other initial points at zero, the identity, and a matrix of all ones as well as the original starting point at PCA of the original data. In all cases, GET forces $p(y_i = defect)$, defined by Equation (6.1), to zero for both defects and non-defects. On the blind test data, however, the true defect is clearly separated with a mean probability of being a defect at 1 exactly. This dichotomy permits very high confidence in defect identifications when they occur, but their rare occurrence means the overall rate of identification remains low. We do not present random sampling chains for GET since GET forced all values to be exactly zero or exactly one in training and testing.

6.7.2 Fused BNCA+LOP Pipeline Results

In this section, we present the results generated by combining DCVD and GET in the BNCA+LOP Pipeline from Algorithm 2. The presented graphical results focus primarily on TR1, as defined in Table 6.3, though we include discussion on other training schemes. In addition, we present results for the naive, Euclidean and Chebychev weighting schemes and compare and contrast them throughout. For extended results, see Appendix C.

We present a summary of results in Tables 6.5 and 6.6. These tables each assess the fused

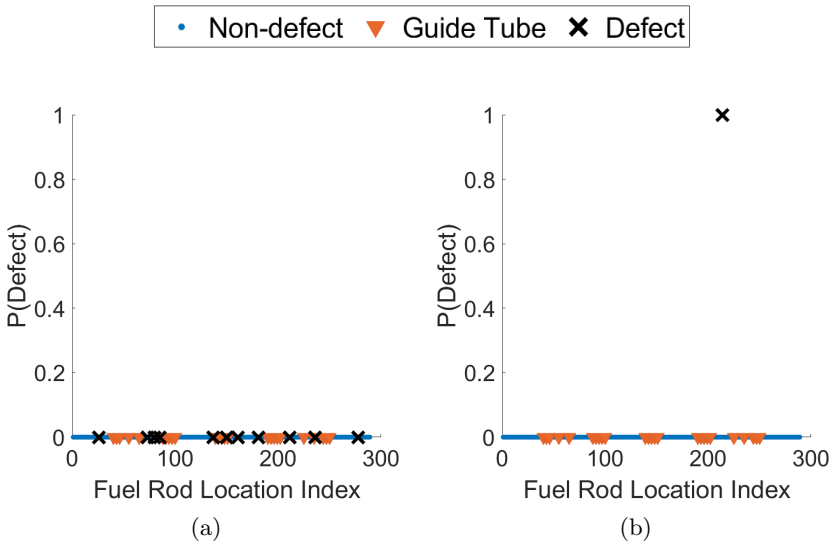


Figure 6.15: GET BNCA $p(y_i = defect|x_i)$ distribution means from Equation (6.1) for (a) TR1 and (b) Blind Test 1 with true defect in rod location 214.

algorithm on a separate metric. The first metric reflects the trend observed in our DCVD results in Section 6.4 and more noticeably in the naive weighting graph in Figure 6.17(a) that guide tubes may be a separate class than true defects. Thus, the first quantity is the average percentage of true defects whose distance from the defect class average distribution mean is greater than the largest distance between a guide tube mean and the average mean; i.e., the average number of defects for which

$$||m_{TD} - \bar{m}_{defect}||_2 \geq \max_i ||m_{\text{Guide Tube}_i} - \bar{m}_{defect}||_2$$

is true. Here m_{TD} indicates the distribution mean for the true defect (TD), the $m_{\text{Guide Tube}_i}$ the mean for the i^{th} guide tube, and $\bar{m}_{defect}$ is the average distribution mean of the defect class. This acts as a metric for whether the true defects and guide tubes belong to the same 'defect' class. In Table 6.6, the metric is a true positive rate; i.e., the percentage of members of the defect class with the mean of $p(y_i = defect|x_i) > 0.5$. While the true positive rate remains low throughout our tests, the frequency with which the true defect is separated from the guide tubes within the defect class is high. This indicates potential future work in expanding the BNCA+LOP Pipeline to a multi-class problem where guide tubes and defects belong to different classes. Note that for 25 nearest neighbors, the true positive rate of 74.55% for DCVD is remarkably high compared to the other DCVD true positive rates. This is because the algorithm correctly identified a majority of the guide tubes as members of the defect class while misidentifying the true defect in a majority of the blind tests. In contrast, the BNCA+LOP Pipeline results have a low false positive rate because while they correctly identify the true defects consistently, they tend to group the guide tubes nearer the non-defect class in the distribution mean space.

Table 6.5: Average frequency that the true defect mean is far from the guide tube subset average mean for DCVD, GET and BNCA+LOP Pipeline for TR1 training.

	True Defect Separates from Guide Tubes(%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	49.49	60.61	44.44	0.0
GET	22.22	33.33	22.22	22.22
BNCA+LOP: Naive	60.61	60.61	55.56	22.22
BNCA+LOP: Euclidean	61.62	60.61	47.47	24.24
BNCA+LOP: Chebychev	63.64	64.65	27.27	27.27

Table 6.6: Average true positive defect classification rate by thresholding distribution means at 0.5 for DCVD, GET and BNCA+LOP Pipeline for TR1 training.

	True Positive Rate 0.5 (%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	4.97	6.82	7.10	74.55
GET	0.43	0.86	0.43	0.43
BNCA+LOP: Naive	0.86	0.86	0.86	0.86
BNCA+LOP: Euclidean	0.86	1.16	0.86	0.86
BNCA+LOP: Chebychev	0.86	1.16	0.86	0.86

We plot the means of the estimated distributions for each fuel rod, after training with TR1, and using the naive, Euclidean and Chebychev weighting schemes in Figure 6.16. Note that although the 11 true defects are plotted as a black cross to differentiate from guide tubes, in the binary problem presented here, they are both members of the same 'defect' class. We clearly see the impact of the separate BNCA results for both DCVD and GET, as the near-zero distribution means of GET results from Figure 6.15 drive the distribution mean much lower as the assembly rod location index moves across regions where GET data is weighted more heavily in Equation (6.3). Whereas the GET mapping for the TR1 training regime maps the mean of each fuel rod to zero, the DCVD mapping learns an appropriate classification better, as reflected in both Table 6.5 and the BNCA-estimated distribution means in Figure 6.13. This is evident in Figure 6.16 as well, with higher means for both defects and non-defects as the rod index moves into regions where DCVD is weighted more heavily in Equation (6.3). This transition between regions of DCVD and GET weights yields a sinusoidal character to Figure 6.16(b) and (c) that is repeated across all our blind tests as well.

The first blind test, Blind 1, is a Case 1 high burnup/short cooling time data set, with only a single defect present in rod location 214. This is the same burnup/cooling time scenario as TR1, with DCVD and GET results separately plotted in Figures 6.13 and 6.15. The BNCA+LOP results for training regime TR1 are presented in Figure 6.17. Rod location 214 is nearer to the

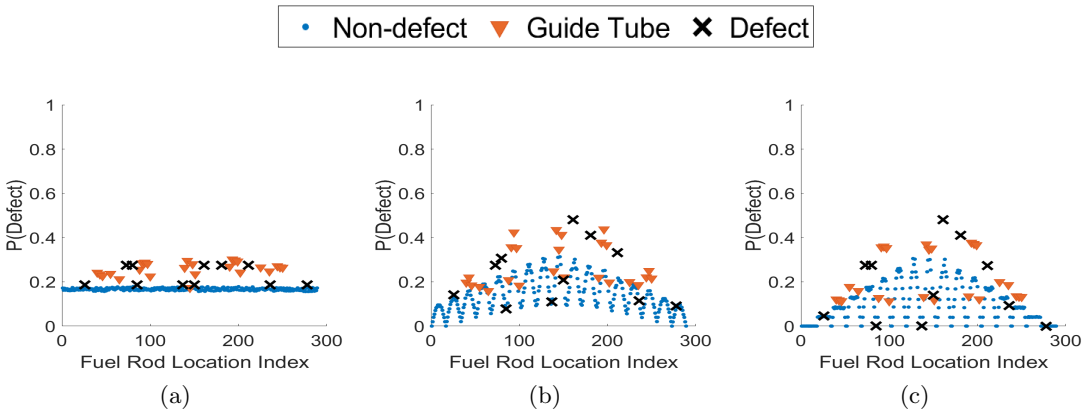


Figure 6.16: Means of estimated distributions for each TR1 fuel rod after training with (a) naive weighting, (b) Euclidean weighting, and (c) Chebychev weighting.

edge of the assembly, so we would expect the GET-trained contribution to be informative and accurate. We see from Figure 6.15(b) that the GET-trained contribution does identify with a very high probability that rod 214 is a defect, which is accurately reflected in the combined results of Figure 6.17.

These combined results classify guide tubes as non-defects, or nearly non-defects, though ground truth classifies them as defects in this binary classifier. This yields an artificially low true positive rate since the true defect is consistently identified as a defect but the guide tubes used to expand our defect set in training are not. Originally, we expanded our very small (1-11) defect class with the guide tubes and considered them all as members of a defect class. However, a key feature especially apparent in the naive weighting scheme, is the separation of most guide tubes from the non-defects as well, indicating the presence of a potential third class. This result indicates that the assumption of equivalence between true defects and guide tubes may not be as valid as initially hypothesized.

Across blind tests with a single defect with the same burnup/cooling time case as our training data, we observe similar results, even in the scenarios where the defect location changes. There is variation between blind test results for similar training regimes, but results are generally consistent within a weighting scheme. One or more water channels/guide tubes occasionally appear alongside the identified true defect, and in our tests, we've had as many as 5 false positives within the blind test set. Even then, the false positive rate for the same burnup/cooling time case stays below 5 %. True positive rates remain low as guide tubes, though members of the defect class, are consistently identified as non-defects. The true defect is not always as visibly separated from the non-defects, with an average of one blind test mapping the true defect to the non-defect distribution mean neighborhood across the weighting schemes and number of neighbors after training. Future work will involve better quantifying the separations between classes. As an interim measure, we simply round distribution means to the nearest integer for ultimate binary classification for true positive rates. This results in low defect detection rates,

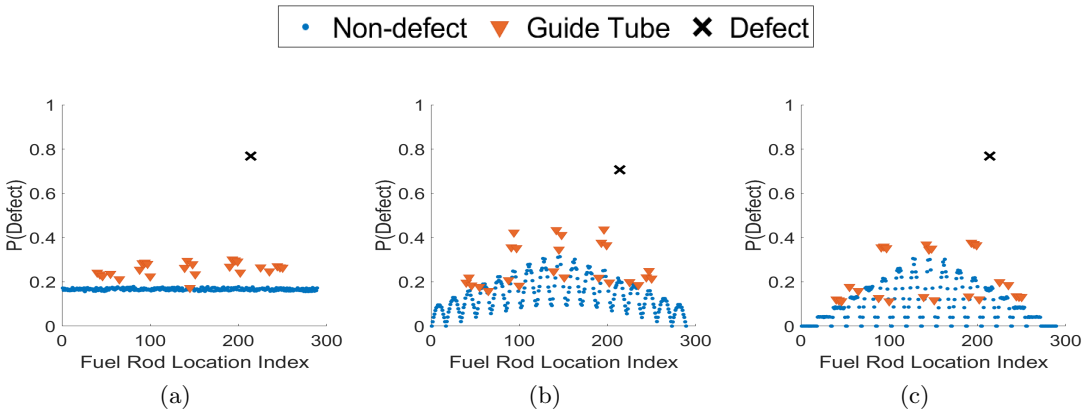


Figure 6.17: Estimated distribution means for Blind 1 with defect in rod index 214 for (a) naive weighting, (b) Euclidean weighting, and (c) Chebychev weighting

though relatively high confidence in defect identifications when they occur.

It is instructive to note the impact of weighting structure reflected in the distribution means plots. Because GET forces all non-defects–and often defects as well–to zero on the $p(defect)$ axis, the weighting scheme is very visible in the non-defect distribution means. The sinusoidal structure is caused by the fuel rod index location moving across rows of the assembly, and thus in and out of the regimes where DCVD and GET dominate. This is observed in Figures 6.9 and 6.10. We would expect to see less of a sinusoidal shape if the classifications resulting from separate DCVD and GET analysis were similar. Instead, we observe the sinusoidal shape as the BNCA+LOP Pipeline weights cause alternation between the noisier non-defect baseline of DCVD and the zero non-defect baseline of GET.

We plot MCMC chains for Blind Test 1 in Figure 6.18 for the defect in location 214. We have good mixing, though only the high probability region is well-explored. This is not necessarily unexpected, as we would expect a particular fuel rod mean estimate to explore the relevant probability space for its class assignment. However, that would encompass the entire space above or below 0.5, and we do not see exploration of that region. This is likely a reflection of the Dirac GET data distributions reflected in Figure 6.19, which greatly narrows the Pipeline results. As discussed in Section 6.7.1, GET trains almost all rods to zero or near-zero, resulting in nearly-Dirac distributions in the Figure 6.19.

The corresponding distributions for the chains in Figure 6.18 under all three weight schemes are plotted in Figure 6.19 alongside an example guide tube. To address burn-in, when we perform kernel density estimation to derive the distributions in Figure 6.19, we omit the first 10% of each chain. We focus on the impact of location of a fuel rod and its resulting distribution mean. The naive weighting distribution is, as defined by its weights, always directly between the contributing DCVD and GET distributions. The example guide tube in Figure 6.19(b), in rod index 91, is on the interface between the two expert-informed weighting schemes transition between detectors. The example non-defect in Figure 6.19(c) is located near the center of the

assembly, so relies primarily on DCVD. The Chebychev weighting scheme varies more with the location of the fuel rod than Euclidean weighting, as we would expect from the step between rings of the assembly in its weighting scheme. We see this reflected in Figure 6.17(c) by the higher amplitude of the sinusoidal behavior than in Figure 6.17(b). This same behavior is harder to visualize in the small example set in Figure 6.19, but is still somewhat visible in that it is closer to the GET mean with the more external defect and non-defect location than the Euclidean distribution at the same locations.

When the burnup/cooling time case is different between the training set and the blind test set, our algorithm still identifies defects, but cannot differentiate between true defects and water channels/guide tubes. The second blind test again has a defect in index location 214 but now is from Case 2 with low burnup and long cooling time. The results of this blind test with a TR1-trained algorithm are shown in Figure 6.20. The low burnup and long cooling time case means that the GET receives a weaker signal than in Case 1. As we would expect, given the differences between Case 1 and Case 2 burnup/cooling time scenarios, the BNCA+LOP Pipeline has difficulty identifying non-defects as truly non-defects, with the average mean P(defect) above 0.6 for all but one non-defect. The true defect in location 214 is no longer clearly separated from the water channels/guide tubes and non-defects. When the defect is located in rod index 105, as in other blind tests, it is not distinguishable as separate from the non-defects for the majority of the blind tests.

Overall, however, the structure that we see underlying the means remains consistent between blind test with both Case 1 and Case 2 burnup/cooling time scenarios. Even when trained using different data sets, this consistency between test sets indicates that although our defect detection capabilities differ between burnup/cooling time scenarios, the underlying data structures identified with the BNCA+LOP Pipeline are consistent.

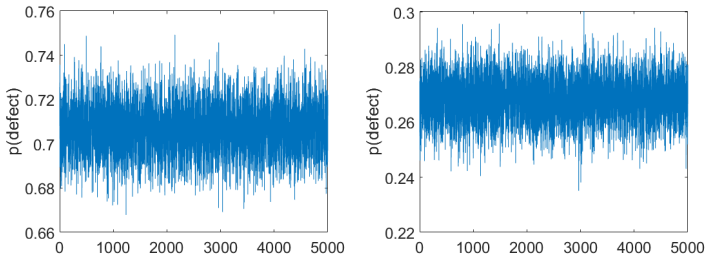
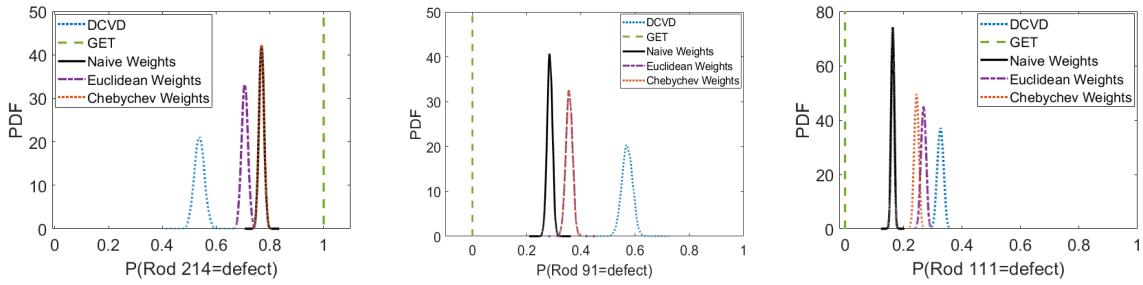


Figure 6.18: Naive weighting MCMC chains for Blind Test 1 for (a) defect rod 214 and (b) non-defect rod 111.



(a) (b) (c)

Figure 6.19: Estimated $p(y_i = defect|x_i)$ for Blind 1: (a) defect rod 214, (b) guide tube rod 91 and (c) non-defect rod 111.

6.8 Conclusions

One of the powerful aspects of the BNCA+LOP Pipeline from Algorithm 2 is the fact that it incorporates data sources with differing dimensions; e.g., data vectors of length 80 and 245 in DCVD and GET data, respectively. The distribution estimates from each data source are combined, rather than combining the BNCA-learned **A** matrices or combining the data directly. This permits implementation of this algorithm in cases where the data sizes between sources vary greatly. In our data, for example, we can compress the original DCVD data without significant data loss, but cannot do so to the same degree with GET, and the pipeline algorithm remains functional. This ability to combine data of different sizes and information content makes the BNCA+LOP Pipeline applicable in a wide variety of scenarios where traditional fusion may not readily apply. The explored weighting schemes allow incorporation of expert opinion in this application but may also be applicable in other scenarios, like the combination of non-expert and expert information [43]. The example problems of noisy XYZ data sets and XYZ+RGB data illustrate two frequently encountered fusion scenarios, and how the BNCA+LOP Pipeline overcomes both.

Although the probability estimation BNCA framework makes it difficult to implement an optimization algorithm directly, we did compare iteration count limits to illustrate optimization. Iterating to a high degree of self-consistency–$||\boldsymbol{\gamma}_k^{DCVD} - \boldsymbol{\gamma}_{k+1}^{DCVD}||_2^2 < 1e-6$ in the Section 2.1.6 with variational update Equation (2.10)–frequently caused overfitting when the iteration count increased above approximately 100 steps. Thus, we set 100 as the maximum iteration count throughout. Overfitting was more of an issue in the GET data set than in the DCVD data set. In DCVD, there is significant difference in pixel intensities between regions within an image. In GET, the pixel intensities between classes are much more similar, causing the GET classifier to fit to noise that differentiates the classes in the training data rather than the true signals of the fuel rods. Even with iteration count restrictions and taking the best results from three different starting points–PCA of the original data, the identity, and a matrix of all ones–the GET model consistently drove the probability of being a defect to near-zero for all fuel rods

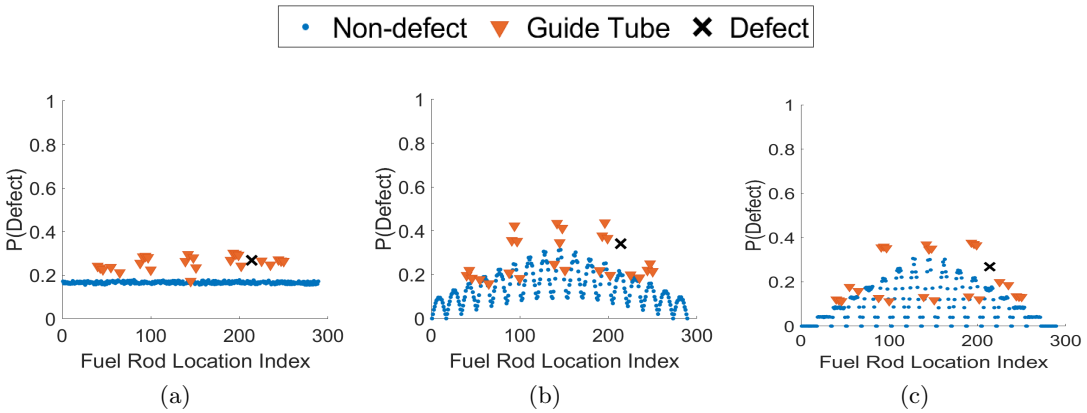


Figure 6.20: Estimated distribution means for Blind 2 with defect in rod index 214 for (a) naive weighting, (b) Euclidean weighting, and (c) Chebychev weighting

in the training data sets. In the cases where the GET-trained model was able to correctly identify a defect, it forced $p(defect)$ to 1. Thus, when GET did identify a defect, there was very little uncertainty associated with the classification. Even though the GET model frequently misidentified defects, the combined BNCA+LOP still benefits from the combination of GET and DCVD, as the small uncertainty in GET defect classifications overall uncertainty in defect identifications. In addition, GET had a 0% false defect detection rate. This drives the false positive rate in the BNCA+LOP Pipeline results to well below IAEA-defined goal rates of 10%. Overall, our BNCA+LOP Pipeline from Algorithm 2 was rarely able to idetify the entire 'defect' class as a defect, due to the high proportion of guide tubes within the class. This led to a true positive rate near 1% when considering this expanded defect set. However, restricted to only true defects, our true positive rate of identification is 56% for TR1 trained-models.

The BNCA+LOP Pipeline algorithm identified several weaknesses in the initial assumptions behind this work. The first is that the water channels/guide tubes approximate defects. The results of this chapter show that this assumption is not valid; not only do the water channels frequently appear as entirely separate grouping within our distribution mean plots, they are generally more aligned with the non-defect class on the whole.

A second assumption of this work is that the data structure of the differing burnup/cooling time cases would be consistent although brightness would vary. This assumption actually seems to hold here; the defects/non-defects shift as a group, indicating a similar underlying structure. We expect the algorithm to struggle with discerning between defect/non-defect in the low burnup/long cooling time Case 2 scenario, as the signal from the original assembly is not as strong as in the high burnup/short cooling time Case 1. We do observe a shift upward in distribution means for both defects and non-defects, indicating that the algorithm has difficulty identifying the rods individually. However, the same clusters occur, indicating the structure similarities between burnup/cooling time cases hold.

One surprising result is the additional capabilities for separation between guide tubes, true

defects, and non-defects that we see primarily in the blind tests for naive weighting. One explanation for this is that the expert-informed scheme we have implemented lacks some underlying structure quantified by naive weighting. Alternatively, this result may indicate a model weakness in defining the guide tubes as defects rather than non-defects that the expert-informed scheme better displays by better capturing location-based variance in the non-defect classifications.

Overall, our contributions are three-fold. The first is establishing a secondary compression setup for BNCA outside of what the original BNCA authors proposed [54]. The second main contribution is the development of a fusion pipeline, the BNCA+LOP Pipeline algorithm, that aggregates data from multiple sources of different sizes and quality, providing probabilistic classifications as output. The final contribution is the application to spent fuel defect detection–the methods contained in both Chapter 5 and 6 improve upon existing defect detection capabilities in the field without requiring extensive modification to existing data collection and safeguards regimes.

Chapter 7

General Conclusions and Future Work

Our research posed the question of whether the combination of two different data sources can reliably identify spent nuclear fuel defects. Additionally, we aimed to address the automation of such a process in a robust way. Using digital Cerenkov viewing device (DCVD) and gamma emission tomography (GET) data, we have used and developed several algorithms that provide proof-of-concept and provide motivation for future research into this idea. Previous research in both DCVD and GET has focused directly on pattern matching that relies on the accuracy of image prediction capabilities. The goal of our research was to avoid the computational difficulty of modeling expected detector results for creating a pattern-matching library and instead focused on simple machine learning algorithms trained with a small example sets for defect detection.

For that objective, this research has been very successful. The non-Bayesian methods of Chapter 5 showed defect detection rates as high as 71% with false positive rates under 5%. The Bayesian methods of Chapter 6 were not nearly as accurate, with an overall true positive rate of only 1% in most tests and a maximum of 6.5% when considering the expanded defect set that contains both true defects and guide tubes. When the blind test has the same burnup/cooling time as the training set, our ability to identify the *true defect* raises to 56%, but guide tube misidentification as non-defects still yields an overall low true positive rate. However, the Bayesian methods provide a way for fusing our data types in Bayesian machine learning framework that allows for both automation of the defect detection process and quantification of uncertainties in algorithmic classifications. Neither the non-Bayesian nor Bayesian methods developed in this dissertation require modification to existing data collection methods in safeguards applications. This provides additional flexibility in potential real-world implementation. However, there are many algorithmic improvements yet to be explored, and these provide significant potential for future work.

Within the non-Bayesian fusion methods of Chapter 5, we primarily demonstrated the

feasibility of combining DCVD and GET data and the applicability of machine learning methods towards automating the subsequent defect detection process in the fused data. We developed two simple fusion regimes that can be implemented not only for the machine learning methods we presented in Chapter 5 but also more generally in applications where expert opinion plays a role in confidence in data sources. There are many options for future work connected to non-Bayesian machine learning fusion with expert knowledge. The first is further optimization of the already-implemented methods using more data for training and testing where available. The second is the implementation of other, more complex machine learning algorithms that may be better suited to image classification. This avenue is somewhat limited, as many machine learning methods are designed exclusively for use in 'big data' situations, and we have few examples of defects on which to train. As research in defect simulation improves, we may be able to incorporate the results for improved and expanded training in new algorithms.

Our main contributions within the Bayesian methods of Chapter 6 are as follows. The first is establishing a secondary compression for Bayesian neighborhood component analysis (BNCA) that differs from that proposed in [54] or other neighborhood components-based methods where compression is frequently just restricting the size of the learned mapping to be a non-square matrix. The second main contribution is the development of a fusion pipeline, the BNCA+LOP Pipeline in Algorithm 2, that provides probabilistic classifications. This is especially advantageous because it can aggregate data from multiple sources of different sizes and quality while still providing a single classification. The BNCA+LOP Pipeline results in Section 6.7 identified model weaknesses that will be addressed in future research. In particular, the assumption that guide tubes approximate defects was directly challenged, as true defects and guide tubes consistently group themselves separately in probability space.

Another area of future research lies in the metrics used to evaluate the Pipeline Algorithm 2 results. We focused on true positive rates based on thresholded probability means; e.g., when the mean of the probability that a rod is a defect is above 0.5, it is identified as a defect. This does not yet take into account the variance of the distributions, and does not adequately assess the behavior of guide tubes as a potentially separate class that consistently maps near this 0.5 threshold. As an alternate metric, future research may return to a nearest neighbor classifier in probability space instead of the 0.5 thresholding. To maintain the probabilistic results of the BNCA+LOP Pipeline in Algorithm 2, this will require the application of probabilistic nearest neighbor algorithms such as those detailed in [39].

Finally, in the Bayesian work in Chapter 6, we use only simple Metropolis MCMC sampling as outlined in Appendix A. Implementing other MCMC sampling techniques with desirable properties such as provable convergence would be straightforward and potentially improve probability space exploration in the distribution construction that yields Figure 6.19.

The main focus of future research, however, will focus on variations in defects. The tested data sets for the entirety of this dissertation contained exclusively single rod defects where the entire rod is missing. The next step is to extend to scenarios where a fuel rod is neither

entirely present or entirely missing, but rather substituted with some other material or only partially missing. These are the defects of significant interest to the International Atomic Energy Agency (IAEA) in the transition to automated defect detection processes. This extension will rely heavily on research in defect modeling, such as that in [7], as there are even fewer real examples of this kind of data compared to completely missing single fuel rods.

A second potential area of future research specific to safeguards applications is extension to other assembly geometries. Underlying inaccuracies with DCVD and GET detectors are exacerbated by the large size of the 17×17 PWR assemblies tested within this thesis. An extension to smaller assemblies and boiling water reactor (BWR) fuel would provide further training and validation of the methods developed and deployed in Chapters 5 and 6 as well as extended applicability in the existing inspections regimes.

BIBLIOGRAPHY

[1] "A used fuel storage pool located at La Hague, France". In: *Management of Radioactive Waste in Australia*. Ansto, 2011. URL: http://large.stanford.edu/courses/2017/ph241/werner2/docs/ansto-jan11.pdf.

[2] C. Andrieu, N. de Freitas, A. Doucet, and M.I. Jordan. "An Introduction to MCMC for Machine Learning". In: *Machine Learning* 50 (1-2 2003), pp. 5–43.

[3] N.F.H Berlis and S. Eklund. *INFCIRC/164–Agreement Between the Government of Canada and the International Atomic Energy Agency for the Application of Safeguards in Connection with the Treaty on the Non-Proliferation of Nuclear Weapons*. 1972.

[4] J.C. Bezdek, S.K. Chuah, and D. Leep. "Generalized k-Nearest Neighbor Rules". In: *Fuzzy Sets and Systems* 18 (1986).

[5] C. M. Bishop. "Continuous Latent Variables". In: *Pattern Recognition and Machine Learning*. Springer, 2006. Chap. 12, pp. 561–590.

[6] D. Bohning. "Multinomial Logistic Regression Algorithm". In: *Annals of the Institute of Statistical Mathematics* 44 (1992), pp. 197–200.

[7] E. Branger. "Enhancing the performance of the Digital Cherenkov Viewing Device: Detecting partial defects in irradiated nuclear fuel assemblies using Cherenkov light". PhD thesis. Uppsala University, 2018.

[8] E. Branger. "Studies of Cherenkov light production in irradiated nuclear fuel assemblies". PhD thesis. Uppsala University, 2016.

[9] E. Branger, S. Jacobsson Svärd, S. Grape, and E. Wernersson. "Towards unattended partial-defect verification of irradiated nuclear fuel assemblies using the DCVD". In: *IAEA Safeguards Symposium*. IAEA, 2014. URL: https://www.iaea.org/safeguards/symposium/2014/home/eproceedings/sg2014-papers/000079.pdf.

[10] E. Branger, E. Wernersson, S. Grape, and S. Jacobsson Svärd. *Image Analysis as a Tool for Improved use of the Digital Cherenkov Viewing Device for inspection of irradiated PWR fuel assemblies*. Tech. rep. Uppsala University, 2014. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-237159.

[11] E. Brayfindley, J. Mattingly, R. Smith, and R. Brigantic. "Automated Defect Detection in Spent Nuclear Fuel Using Combined Cerenkov Radiation and Gamma Emission Tomography Data". In: *Nuclear Technology* (2018).

[12] *Channel Systems: DCVD*. 2019. URL: https://www.channelsystems.ca/products/digital-cerenkov-viewing-device-dcvd/dcvd.

[13] J. D. Chen, D. A. Parcey, R. Kosierb, M. Larsson, K. Axell, J. Dahlberg, B. Lindberg, and E. Sundkvist. "Detection of Partial Defects using a Digital Cerenkov Viewing Device". In: *IAEA Safeguards Symposium*. IAEA, 2010. URL: https://www.iaea.org/safeguards/symposium/2010/Documents/PapersRepository/338.pdf.

[14] A. Davour, S. Jacobsson Svärd, P. Andersson, S. Grape, S. Holcombe, P. Jansson, and M. Troeng. "Applying image analysis techniques to tomographic images of irradiated nuclear fuel assemblies". In: *Annals of Nuclear Energy* 96 (Oct. 2016), pp. 223–229. ISSN: 03064549. DOI: 10.1016/j.anucene.2016.05.024. URL: http://linkinghub.elsevier.com/retrieve/pii/S0306454916303358.

[15] US Department of Energy. *Photo Gallery–Yucca Mountain*. 2010. URL: https://www.energy.gov/sites/prod/files/styles/borealis_photo_gallery_large_respond%20medium/public/fuel_assembly_for_production_of_nuclear_power.jpg?itok=8XJJASbF.

[16] M. Fisher. "New Safeguards Tool Bolsters IAEA's Verification of Spent Nuclear Fuel". In: (2019).

[17] C. Fowlkes. "NCA Example Code". Code package and example problem based on [22]. 2005.

[18] P.H. Garthwaite, J. B. Kadane, and A. O'Hagan. "Statistical Methods for Eliciting Probability Distributions". In: *Journal of the American Statistical Association* 100.470 (2005).

[19] C. Genest and K.J. McConway. "Allocating the weights in the linear opinion pool". In: *Journal of Forecasting* 9 (1990).

[20] C. Genest and J.V. Zidek. "Combining Probability Distrubtions: A Critique and an Annotated Bibliography". In: *Statistical Science* 1 (1986).

[21] F. Giannini, V. Laveglia, A. Rossi, D. Zanca, and A. Zugarini. "Neural Networks for Beginners: A fast implementation in MATLAB, Torch and TensorFlow". In: (2017).

[22] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. "Neighborhood Components Analysis". In: *Advances in Neural Information Processing Systems*. Vol. 17. Neural Information Processing Systems, 2004.

[23] I. Goodfellow, Y. Bengio, and A. Courville. "Convolutional Networks". In: *Deep Learning*. The MIT Press, 2016. Chap. 8,9, pp. 267–361.

[24] Heikki Haario, Marko Laine, Antonietta Mira, and Eero Saksman. "DRAM: Efficient adaptive MCMC". In: *Statistics and Computing* 16.4 (Dec. 2006), pp. 339–354. ISSN: 1573-1375. DOI: 10.1007/s11222-006-9438-0. URL: https://doi.org/10.1007/s11222-006-9438-0.

[25] H. Habibi Aghdam and E. Jahani Heravi. "Convolutional Neural Networks". In: *Guide to Convolutional Neural Networks*. Springer International, 2017. Chap. 3, pp. 85–128.

[26] M. Hagan, H. Demuth, M. Hudson Beale, and O. De Jesus. *Neural Network Design*. PWS Publishing, 1996.

[27] Y. Ham, P. Kerr, S. Sitaraman, R. Swan, R. Rossa, and H. Liljenfeldt. "Partial Defect Verification of Spent Fuel Assemblies by PDET: Principle and Field Testing in Interim Spent Fuel Storage Facility (CLAB) in Sweden". In: *4th International Conference on Advancements in Nuclear Instrumentation Measurement Methods and their Applications*. IEEE, 2015.

[28] C. Higham and D. Higham. "Deep Learning: An Introduction for Applied Mathematicians". In: *arXiv* (2018).

[29] *History*. International Atomic Energy Agency. 2019. URL: https://www.iaea.org/about/overview/history.

[30] *Safeguards Techniques and Equipment: 2011 Edition*. International Atomic Energy Agency, 2011, pp. 29, 34–36. ISBN: 9789201189103. DOI: ISBN978-92-0-118910-3.

[31] S. Jacobsson Svärd, A. Hakansson, A. Backlin, P. Jansson, O. Osifo, and C. Willman. "Tomography for partial-defect verification: experiences from measurements using different devices". In: *ESARDA Bulletin*. Ed. by Vol. 33. 2006, pp. 15–25.

[32] S. Jacobsson Svärd, S. Holcombe, and S. Grape. "Applicability of a set of tomographic reconstruction algorithms for quantitative SPECT on irradiated nuclear fuel assemblies". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 783 (2015), pp. 128–141. DOI: https://doi.org/10.1016/j.nima.2015.02.035.

[33] S. Jacobsson, A. Hakansson, C. Andersson, P. Jansson, and A. Backlin. *Tomographic Method for Verification of the Integrity of Spent Nuclear Fuel*. Tech. rep. Uppsala University, 1998. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-200297.

[34] S. Jacobsson, A. Hakansson, P. Jansson, and A. Backlin. "A Tomographic Method for Verification of the Integrity of Spent Nuclear Fuel Assmeblies-II: Experimental Investigation". In: *Nuclear Technology* 135.2 (2001), pp. 146–153. DOI: http://dx.doi.org/10.13182/NT01-A3212.

[35] C.T. Kelley. "Global Convergence". In: *Iterative Methods for Optimization*. SIAM, 1999. Chap. 2-3.

[36] Y. LeCun, L. Bottou, G. B. Orr, and K.R. Müller. ""Efficient BackProp"". In: *Neural Networks: tricks of the trade*. Springer, 1998.

[37] R. C. Luo, C. C. Chang, and C. C. Lai. "Multisensor Fusion and Integration: Theories, Applications and its Perspectives". In: *IEEE Sensors* 11.12 (2011).

[38] R.C. Luo and K.L. Su. "A review of high-level multisensor fusion: approaches and applications". In: *IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*. IEEE, 1999.

[39] S. Manocha and M. A. Girolami. "An Empirical Analysis of the Probabilistic K-nearest Neighbour Classifier". In: *Pattern Recogn. Lett.* 28.13 (Oct. 2007), pp. 1818–1824. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2007.05.018. URL: https://doi.org/10.1016/j.patrec.2007.05.018.

[40] D. Marquardt. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". In: *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963).

[41] T. McSweeney. "Modified Cholesky Decomposition and Applications". 2017.

[42] A. O'Hagan, C.E. Buck, A. Daneshkhah, J.R. Eiser, P.H. Garthwaite, D.J Jenkinson, J.E. Oakley, and T. Rakow. "Uncertain Judgements: Eliciting Experts' Probabilities". In: John Wiley & Sons, 2006. Chap. 9, pp. 179–192.

[43] K. Pacifici, B. J. Reich, D. A.W. Miller, B. Gardner, G. Stauffer, S. Singh, and J. A. Collazo. "Integrating multiple data sources in species distribution modeling: a framework for data fusion". In: *Ecology* 98.3 (2016).

[44] O. Punska. "Bayesian Approaches to Multi-Sensor Data Fusion". PhD thesis. University of Cambridge, 1999. URL: https://www-sigproc.eng.cam.ac.uk/foswiki/pub/Main/OP205/mphil.pdf.

[45] A. Ranganathan. "The Levenberg-Marquardt Algorithm". In: *Academia.edu* (2004).

[46] J. R. Raol. *Data Fusion Mathematics: Theory and Practice*. CRC Press, 2016.

[47] S. Ruder. "An Overview of Gradient Descent Optimization Methods". In: *arXiv* (2017). URL: https://arxiv.org/pdf/1609.04747.pdf.

[48] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. "Online and Batch Learning of Pseudo-metrics". In: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: ACM, 2004, pp. 94–102. ISBN: 1-58113-838-5. DOI: 10.1145/1015330.1015376. URL: http://doi.acm.org.prox.lib.ncsu.edu/10.1145/1015330.1015376.

[49] E. L. Smith, V. Mozin, E. Miller, N. Deshmukh, R. Wittman, S. Grape, S. Vaccaro, S. Jacobsson Svärd, P. Jansson, T. Honkamaa, T. White, H. Trellue, and A. Davour. *A Viability Study of Gamma Emission Tomography for Spent Fuel Verification: JNT 1955 Phase 1 Technical Report*. Tech. rep. Pacific Northwest National Laboratory, 2016.

[50] L. E. Smith, C. J. Gesh, R. T. Pagh, E. A. Miller, M. W. Shaver, E. D. Ashbaker, M. T. Batdorf, J. E. Ellis, W. R. Kaye, R. J. McConn, G. H. Meriwether, J. J. Ressler, A. B. Valsan, and T. A. Wareing. "Coupling Deterministic and Monte Carlo Transport Methods for the Simulation of Gamma-Ray Spectroscopy Scenarios". In: *IEEE Transactions on Nuclear Science* 55.5 (Oct. 2008), pp. 2598–2606. ISSN: 0018-9499. DOI: 10.1109/TNS.2008.2002819. URL: http://ieeexplore.ieee.org/document/4696606/.

[51] R.C. Smith. *Uncertainty Quantification: Theory, Implementation and Applications*. SIAM Computational Science and Engineering, 2014.

[52] D. Sweeney. "Additional Detection Techniques under Development". In: *Spent Nuclear Fuel Safeguards*. Texas A&M University, Nuclear Security and Safeguards Education Portal, 2017. Chap. Safeguards, p. 35. URL: http://nsspi.tamu.edu/nssep/courses/spent-nuclear-fuel-safeguards.

[53] M. E. Wall, A. Rechtsteiner, and L. M. Rocha. "Singular value decomposition and principal component analysis". In: *A Practical Approach to Microarray Data Analysis*. Kluwer: Norwell, LANL LA-UR-02-4001, 2003, pp. 91–109.

[54] D. Wang and X. Tan. "Bayesian Neighborhood Component Analysis". In: *IEEE Transactions on Neural Networks and Learning Systems* 29 (7 2018).

[55] B. Waske and J.A. Benediktsson. "Decision Fusion, Classification of Multisource Data". In: *Encyclopedia of Remote Sensing*. New York, NY: Springer New York, 2014, pp. 140–144. ISBN: 978-0-387-36699-9. DOI: 10.1007/978-0-387-36699-9_34. URL: https://doi.org/10.1007/978-0-387-36699-9_34.

[56] E.W. Weisstein. *Gram-Schmidt Orthonormalization*. 2019.

[57] T. White, S. Jacobsson Svärd, L. E. Smith, V. Mozin, P. Jansson, A. Davour, S. Grape, H. Trellue, N. Deshmukh, R. S. Wittman, T. Honkamaa, S. Vaccaro, and J. Ely. "Passive Tomography for Spent Fuel Verification: Analysis Framework and Instrument Design Study". In: *ESARDA Symposium*. 2015.

[58] A.G. Wilson and K.M. Fronczyk. "Bayesian reliability: Combining information". In: *Quality Engineering* 29.1 (2017).

[59] Z. Xu, K.Q. Weinberger, and O. Chapelle. "Distance Metric Learning for Kernel Machines". In: *arXiv* (2013).

[60] W. Yang, K. Wang, and W. Zuo. "Fast neighborhood component analysis". In: *Neurocomputing* 83 (2012).

[61] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. "Comparative study of Hough transform methods for circle finding". In: *Image and Vision Computing* 8.1 (1990), pp. 71–77.

APPENDICES

Appendix A

Markov Chain Monte Carlo Algorithms

Markov Chain Monte Carlo (MCMC) methods select statistical samples to approximate the solution to difficult-to-compute combinatorial problems [2]. In a Bayesian framework, one frequently encountered problem is that of posterior density estimation; i.e., estimating the probability that a variable of interest takes a certain value based on observed data [51]. MCMC methods combine Monte Carlo posterior distribution sampling for Equation A.1 with the Markov chain property that every sample depends only on the sample directly before it to estimate analytically intractable posterior distribution estimations [51]. The Markov chains are constructed so that their stationary distribution is the posterior distribution. To approach the true solution, however, many samples are required, and this process was limited by computational shortcomings for many years after it was originally developed. However, with the computational improvements of the last two decades, MCMC methods have gained in feasibility and, correspondingly, in popularity [2].

To define Bayesian MCMC, we start with the functional form of Bayes' theorem. Given a set Q of p random variables with a known prior distribution $\pi_0(q)$, Bayes' theorem states that

$$\pi(q|v_{obs}) = \frac{\pi(v_{obs}|q)\pi_0(q)}{\pi(v_{obs})} = \frac{\pi(v_{obs}|q)\pi_0(q)}{\int_{\mathbb{R}^p} \pi(v_{obs}|q)\pi_0(q)dq}, \tag{A.1}$$

where v_{obs} indicates observed data [51]. Here, the likelihood function $\pi(v|q)$ depends on model and data observation, the prior distribution $\pi_0(q)$ incorporates any prior knowledge about the value of the parameters q before observations and the marginal distribution is an integral over the entire parameter space. The integral over $\mathbb{R}^p$ quickly becomes intractable for p as low as 3. Rather than attempting to directly compute this integral analytically or with quadrature methods, MCMC methods instead sample a proposal function to generate a Markov chain whose stationary distribution converges to the posterior distribution.

Here, we outline the basic metropolis algorithm–one of the simpler MCMC methods. Note,

however, that there are many modifications to the Metropolis algorithm for improved robustness and convergence; e.g. Delayed Rejection, Adaptive Metropolis, etc. See [51] for more details of these variations.

A.1 Metropolis Algorithm

Define a statistical model

$$\Upsilon_i = f_i(Q) + \varepsilon_i \quad i = 1, \ldots, n$$

where Υ_i indicates measurements, $Q = [Q_1, \ldots, Q_p]$ are parameters, and ε_i indicates measurement error [51]. The model response is $f_i(Q)$, and n is the number of measurements. The likelihood, $\pi(v|q)$ quantifies the probability of observing v given a parameter value q.

If we make the assumption that the errors ε are independent and identically distributed (iid) and sample from a normal distribution such that $\varepsilon_i \sim N(0, \sigma^2)$, we can express the likelihood analytically using the definition of a normal distribution [51]. This yields

$$\pi(v|q) = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-SS_q/2\sigma^2} \tag{A.2}$$

where

$$SS_q = \sum_{i=1}^{n} [v_i - f_i(q)]^2$$

is the sum of squares error between the observation and model output. With this equation, we can now calculate the likelihood function for every observation and parameter value q. We can now implement a simple metropolis algorithm, as outlined in Algorithm 3. For simplicity, we assume that the prior is uniform.

A new candidate q^* at the k^{th} iterate depends only on the immediately preceding $k-1$ iterate following Markov Chain definitions. It follows that q^* candidate is constructed with the $k-1$ iterate such that $q^* = q^{k-1} + Rz$. Here $z \sim N(0, 1)$ and R is the Cholesky decomposition of the covariance matrix V of the parameter set Q. This definition of R and q^* ensures that $q^* \sim N(q^{k-1}, V)$–see [51] for details. The covariance matrix can be approximated using frequentist methods if it is not directly calculable.

The Metropolis algorithm then computes the posterior using the q^* candidate via Bayes' theorem in Equation A.1. However, rather than directly calculating a posterior function at each iteration, the algorithm instead calculates a *ratio* of posteriors. In practice, this is a ratio of the likelihood function from Equation (A.2) for uniform prior distributions. This ratio eliminates the normalization factor in the denominator of Bayes' Theorem, and thus eliminates the potential issues with p-dimensional integration. Additionally, given our original assumptions that yield a likelihood as in Equation (A.2), this ratio becomes

$$r(q^*|q^{k-1}) = \frac{\pi(v|q^*)\pi_0(q^*)}{\pi(v|q^{k-1})\pi_0(q^{k-1})} = \frac{1}{(2\pi\sigma^2)^{n/2}} e^{-[SS_q - SS_{q^{k-1}}]/2\sigma^2}.$$

Algorithm 3: Metropolis Algorithm from [51].

Result: Posterior distribution $\pi(q|v)$
Initialization: choose q^0 such that $\pi(q^0|v) > 0$;
while $k< M$ **do**
  Sample $z_k \sim N(0,1)$
  Construct candidate $q^* = q^{k-1} + Rz_k$
  Compute $r(q^*|q^{k-1}) = \frac{\pi(q^*|v)}{\pi(q^{k-1}|v)} = \frac{\pi(v|q^*)\pi_0(q^*)}{\pi(v|q^{k-1})\pi_0(q^{k-1})}$
  if $r>1$ **then**
    Set $q^k = q^*$ with probability α
  else
    Set $q^k = q^{k-1}$
  end
end

When the new candidate q^k increases the likelihood, or equivalently, decreases the sum-of-squares error, it is accepted with some probability α.

Two main adaptations of the Metropolis algorithm are Delayed Rejection and Adaptive Metropolis [51]. Delayed rejection, rather than setting $q^k = q^{k=1}$ whenever q^* is rejected, instead constructs a secondary proposal candidate [24, 51]. Adaptive metropolis modifies how q^* is proposed by incorporating the history of accepted q values in the covariance V [24, 51]. While this means it is no longer a Markov process in the strict sense, it enables inclusion of information learned about the posterior throughout the sampling process and is provably convergent. Both of these improvements on the original metropolis algorithm aid in parameter-space exploration and mixing in the sampling process. However, standard metropolis sampling is sufficient for the work in Chapter 6, as our sampling chains demonstrate sufficient mixing and parameter space exploration. Additionally, increasing our number of samples in Chapter 6 does not significantly impact the final distributions, indicating sufficient convergence for the subsequent analysis.

Appendix B

Neural Network Extended Optimization Results

In this Appendix, we present all neural network results for optimization methods. In the following tables, the true positive (TP) rate and false positive (FP) rate are presented for all data sources, including separating out the two burn-up/cooling time (BU/CT) scenarios of GET data. Due to the relative similarity between DCVD BU/CT scenarios, those are not separated. BU/CT 1 indicates Case 1 in Table 5.1, and BU/CT 1 indicates Case 2. The threshold indicated by the table is the final thresholding applied in the Combine-Then-Classify framework (See Section 5.3).

Table B.1: All NN results for gradient descent optimization.

	529×1		50×1		10×3		5×3	
	TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)
DCVD Only								
	81.1	0.8	85.9	0.0	86.3	0.0	77.5	0.0
GET Only								
BU/CT 1	29.9	15.2	36.7	2.7	23.4	1.9	10.0	0.9
BU/CT 2	63.9	56.9	94.8	89.1	68.0	60.2	33.2	28.1
All	52.3	43.0	75.4	60.3	53.1	40.8	25.4	19.0
Combine-Then-Classify								
BU/CT 1	89.7	1.2	94.1	0.0	94.2	0.0	91.9	0.0
BU/CT 2	70.2	1.2	79.9	0.0	82.7	0.0	79.0	0.0
All	82.4	1.2	88.8	0.0	89.9	0.0	87.1	0.0
Classify-Then-Combine								
Threshold = 0.4								
BU/CT 1	56.9	26.3	67.0	40.0	57.7	24.2	42.6	8.8
BU/CT 2	77.1	25.8	90.6	39.9	85.7	23.5	65.5	8.0
All	63.7	26.3	74.9	40.0	67.0	24.0	50.3	8.6
Threshold = 0.5								
BU/CT 1	50.0	24.5	59.6	25.0	44.4	11.5	33.0	5.4
BU/CT 2	66.4	24.3	86.6	24.6	70.2	10.5	53.2	4.6
All	55.5	24.4	68.6	24.8	53.0	11.2	39.7	5.1
Threshold = 0.6								
BU/CT 1	35.5	12.6	43.7	11.8	35.2	4.3	17.4	1.9
BU/CT 2	56.6	12.1	77.6	10.5	56.6	3.1	27.8	1.5
All	42.6	12.4	55.0	11.4	42.3	3.9	20.9	1.8

Table B.2: All NN results for stochastic gradient descent optimization.

	529×1		50×1		10×3		5×3	
	TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)
DCVD Only								
	81.5	0.5	85.1	0.0	85.8	0.0	76.4	0.0
GET Only								
BU/CT 1	27.4	15.2	38.2	2.9	22.7	1.7	10.5	1.1
BU/CT 2	62.0	54.3	93.0	87.7	67.0	58.9	32.3	27.6
All	50.4	41.3	74.8	58.6	52.2	39.9	25.0	18.8
Combine-Then-Classify								
BU/CT 1	57.1	0.2	43.1	0.4	45.4	26.4	46.6	36.6
BU/CT 2	28.8	0.3	24.1	0.6	39.5	27.0	43.5	37.0
All	46.5	0.2	36.0	0.5	43.2	26.6	45.5	36.7
Classify-Then-Combine								
Threshold = 0.4								
BU/CT 1	57.3	26.4	67.0	40.3	55.4	20.9	41.4	9.3
BU/CT 2	76.7	25.9	89.6	40.3	82.8	20.1	61.0	8.7
All	63.8	26.2	74.5	40.3	64.5	20.6	47.9	9.1
Threshold = 0.5								
BU/CT 1	45.1	19.1	59.2	26.7	44.0	10.8	31.5	3.7
BU/CT 2	60.3	18.8	82.5	26.5	71.5	9.6	48.5	2.8
All	50.1	19.0	67.0	26.6	53.2	10.4	37.2	3.4
Threshold = 0.6								
BU/CT 1	35.5	16.2	42.9	11.8	35.7	5.7	19.4	1.3
BU/CT 2	57.7	15.8	76.9	10.5	60.4	4.7	30.7	0.8
All	42.9	16.1	54.2	11.3	43.9	5.4	23.2	1.2

Table B.3: All NN results for Levenberg-Marquardt optimization.

	10×3		5×3	
	TP (%)	FP (%)	TP (%)	FP (%)
DCVD Only				
	90.9	0.0	90.2	0.0
GET Only				
BU/CT 1	30.8	3.4	28.4	3.1
BU/CT 2	55.2	42.2	48.8	38.9
All	47.1	29.1	42.0	31.0
Combine-Then-Classify				
BU/CT 1	96.4	0.0	96.3	0.1
BU/CT 2	86.7	0.1	86.9	0.5
All	92.8	0.0	92.8	0.3
Classify-Then-Combine				
Threshold = 0.4				
BU/CT 1	57.3	15.8	57.3	16.7
BU/CT 2	89.1	14.6	86.7	15.7
All	67.9	15.4	67.1	16.4
Threshold = 0.5				
BU/CT 1	46.6	8.4	46.5	8.2
BU/CT 2	79.8	7.5	76.9	7.3
All	57.7	8.1	56.6	7.9
Threshold = 0.6				
BU/CT 1	35.7	4.0	32.6	4.1
BU/CT 2	63.2	3.2	59.0	3.5
All	44.8	3.7	41.4	3.9

Table B.4: All NN results for conjugate gradient optimization with Fletcher-Reeves updates.

	529×1		50×1		10×3		5×3	
	TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)
DCVD Only								
	89.4	0.1	89.1	0.0	90.3	0.0	89.8	0.0
GET Only								
BU/CT 1	53.9	7.7	41.6	3.6	27.4	4.0	15.8	2.8
BU/CT 2	99.0	96.1	97.0	90.9	61.2	53.9	35.2	31.4
All	52.3	43.0	78.5	41.2	49.8	37.3	28.7	21.8
Combine-Then-Classify								
BU/CT 1	95.4	0.1	95.2	0.0	95.8	0.0	94.3	0.0
BU/CT 2	83.4	0.3	82.6	0.1	87.3	0.1	86.4	0.4
All	90.9	0.2	90.5	0.1	92.6	0.1	91.3	0.2
Classify-Then-Combine								
Threshold = 0.4								
BU/CT 1	73.4	49.4	69.2	42.0	56.5	20.3	51.5	12.6
BU/CT 2	95.2	49.5	93.5	41.2	87.6	19.1	83.2	11.5
All	80.7	49.5	77.3	41.9	66.9	19.9	62.1	12.2
Threshold = 0.5								
BU/CT 1	69.3	40.6	62.1	27.8	47.6	9.8	39.0	3.7
BU/CT 2	90.7	40.9	90.2	27.5	79.6	8.6	67.4	2.8
All	76.4	40.7	71.5	27.7	58.3	9.4	48.4	3.4
Threshold = 0.6								
BU/CT 1	49.3	31.0	45.4	13.6	35.2	4.3	32.7	2.2
BU/CT 2	86.3	31.0	82.3	12.4	56.6	3.1	58.0	1.4
All	61.6	31.0	57.7	13.2	42.3	3.9	41.2	1.9

Appendix C

BNCA+LOP Pipeline Algorithm Extended Results and Figures

This appendix contains images and data corresponding to every tested pairing of number of neighbors, training set and weighting scheme. Sections are separated by training sets, defined in Table 6.3 and reproduced here in Table C.1. For TR1, we present both tabular and graphical results. For all other training sets, we present only summarized tabular results.

Table C.1: Training set variations for implementation.

	DCVD Data	GET Data	Total defect count
TR1	Modified DCVD	Case 1 GET	36
TR2	Modified DCVD	Case 2 GET	36
TR3	True defect at 214	Case 1 Modified GET	26
TR4	True defect at 214	Case 2 Modified GET	26

Within each training set we present each blind test performed once the original BNCA+LOP Pipeline Algorithm was trained. Each blind test, as defined by Table 6.4 and reproduced here in Table C.2, is presented with three images, one for each weighting scheme, in rows corresponding to each choice of nearest neighbors.

Table C.2: Blind test parameters.

	DCVD Data	GET Data	Total defect count
Blind 1	True defect at 214	Case 1 Modified GET	26
Blind 2	True defect at 105	Case 1 Modified GET	26
Blind 3	True defect at 214	Case 1 Modified GET	26
Blind 4	True defect at 105	Case 1 Modified GET	26
Blind 5	True defect at 214	Case 2 Modified Get	26
Blind 6	True defect at 105	Case 2 Modified Get	26
Blind 7	True defect at 214	Case 2 Modified Get	26
Blind 8	True defect at 105	Case 2 Modified GET	26
Blind 1 (TR3, 4)	Modified DCVD	Case 1 GET	36
Blind 5 (TR3, 4)	Modified DCVD	Case 2 GET	36

C.1 TR1 Results

In Table C.3, we present the results for DCVD only, GET only, and the three BNCA+LOP weighting schemes using two different metrics. The first metric is how often the true defect is presented separately from the guide tubes within the defect class, thus identifying whether a third class is needed in future work. The second metric is a standard true positive rate when the distribution means for each fuel rod are thresholded at 0.5 into the 0/1 non-defect/defect classes.

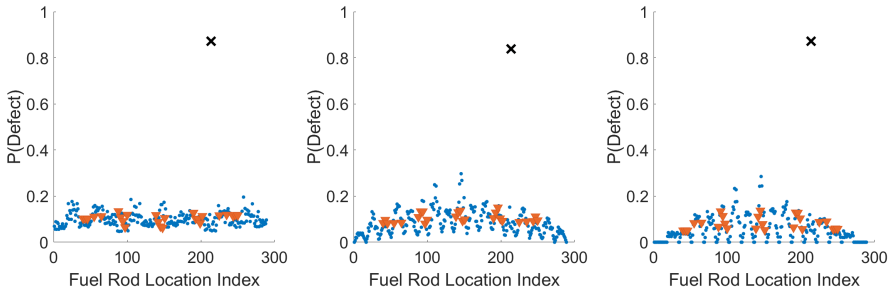
Table C.3: Average frequency that the true defect mean is far from the guide tube subset average mean for DCVD, GET and BNCA+LOP Pipeline for TR1 training.

	True Defect Separates from Guide Tubes(%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	49.49	60.61	44.44	0.0
GET	22.22	33.33	22.22	22.22
BNCA+LOP: Naive	60.61	60.61	55.56	22.22
BNCA+LOP: Euclidean	61.62	60.61	47.47	24.24
BNCA+LOP: Chebychev	63.64	64.65	27.27	27.27

Table C.4: Average true positive defect classification rate by thresholding distribution means at 0.5 for DCVD, GET and BNCA+LOP Pipeline for TR1 training.

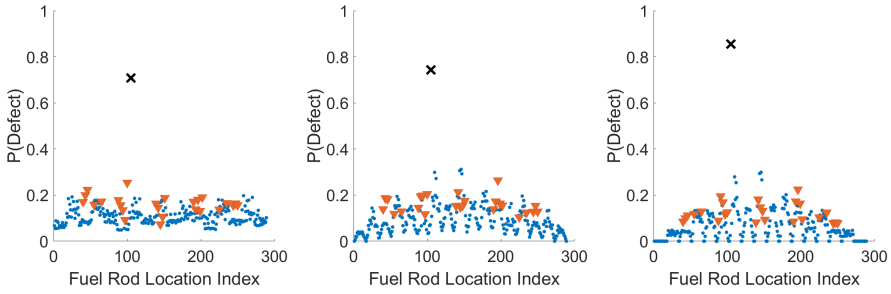
	True Positive Rate 0.5 (%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	4.97	6.82	7.10	74.55
GET	0.43	0.86	0.43	0.43
BNCA+LOP: Naive	0.86	0.86	0.86	0.86
BNCA+LOP: Euclidean	0.86	1.16	0.86	0.86
BNCA+LOP: Chebychev	0.86	1.16	0.86	0.86

C.1.1 2 NN



(a) (b) (c)

Figure C.1: Blind test 1 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.



(a) (b) (c)

Figure C.2: Blind test 2 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

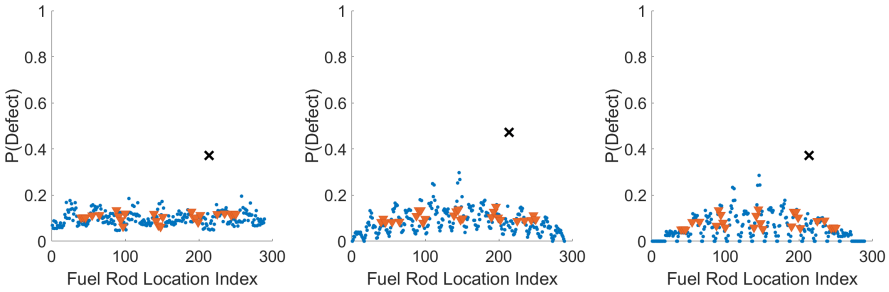


Figure C.3: Blind test 3 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

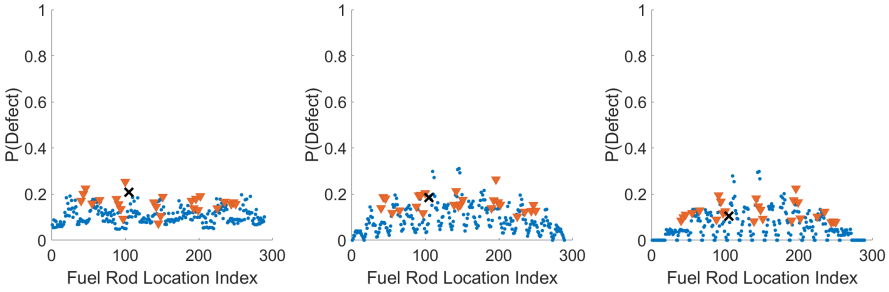


Figure C.4: Blind test 4 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

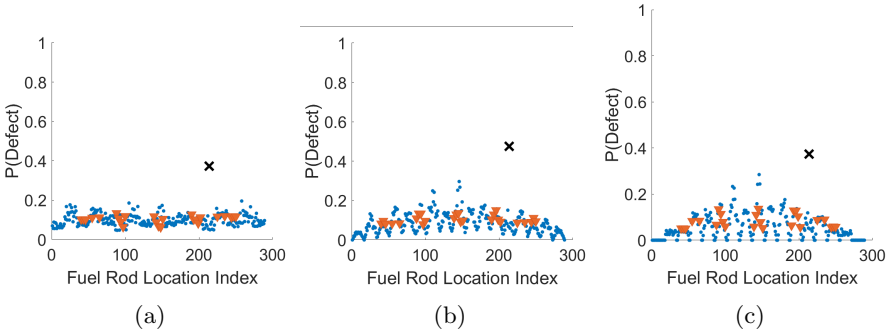


Figure C.5: Blind test 5 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

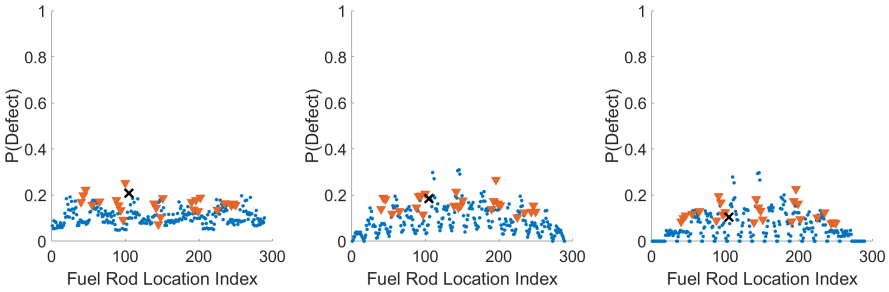


Figure C.6: Blind test 6 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

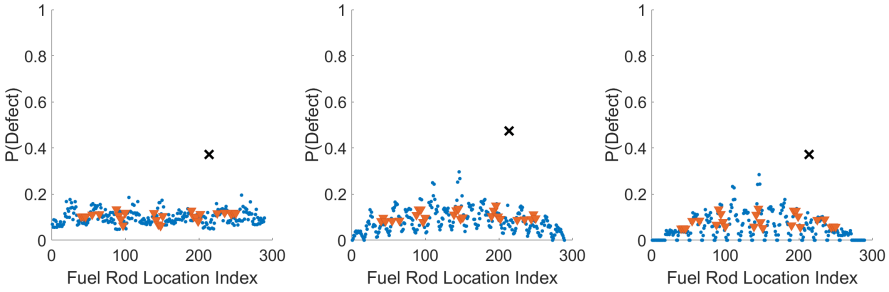


Figure C.7: Blind test 7 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

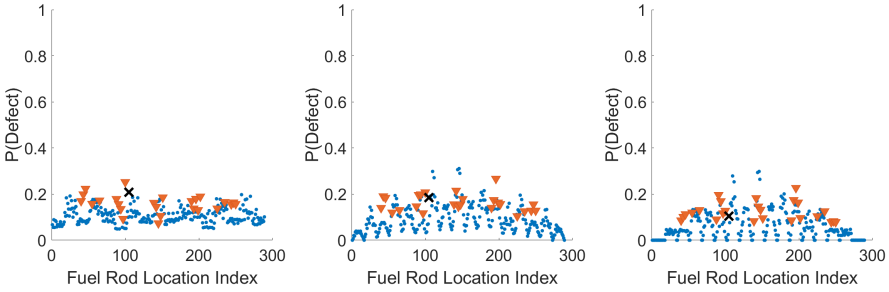


Figure C.8: Blind test 8 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

C.1.2 5 Nearest Neighbors

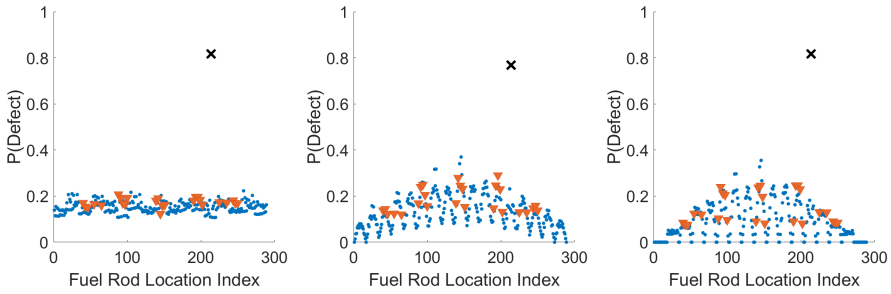


Figure C.9: Blind test 1 results for TR1 with (a) naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

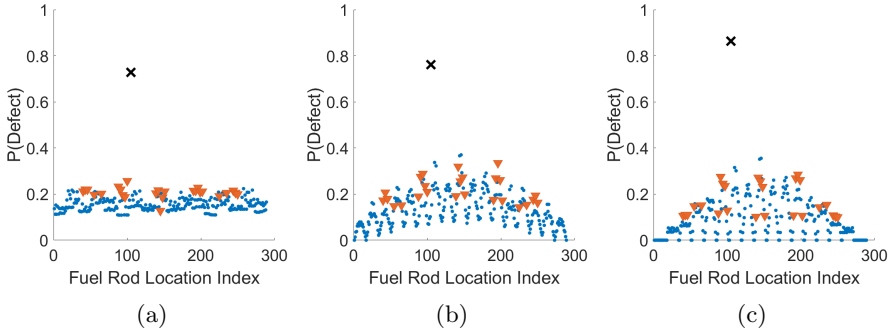


Figure C.10: Blind test 2 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

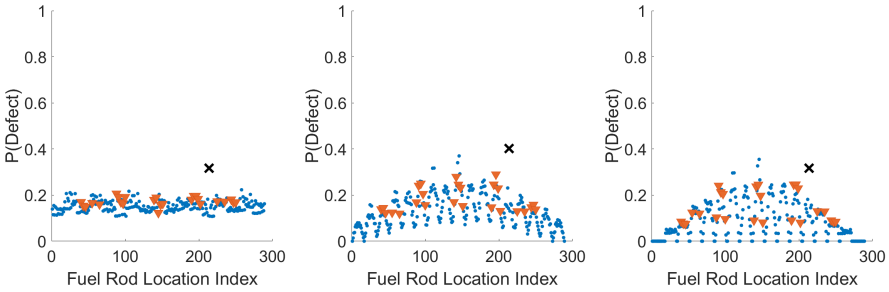


Figure C.11: Blind test 3 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

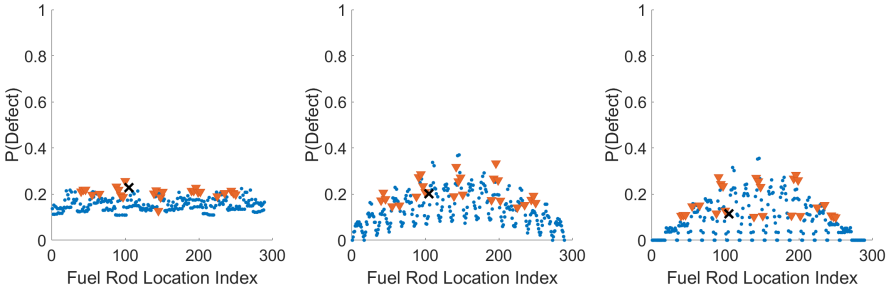


Figure C.12: Blind test 4 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

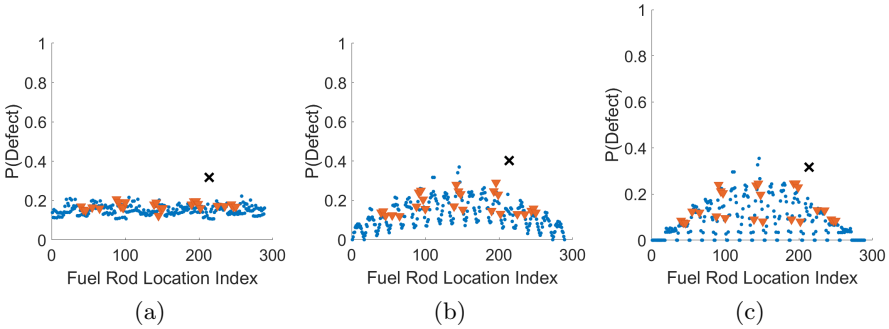


Figure C.13: Blind test 5 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

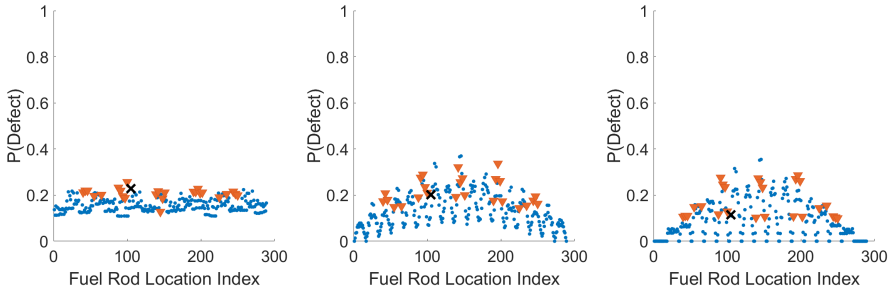


Figure C.14: Blind test 6 results for TR1 with (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

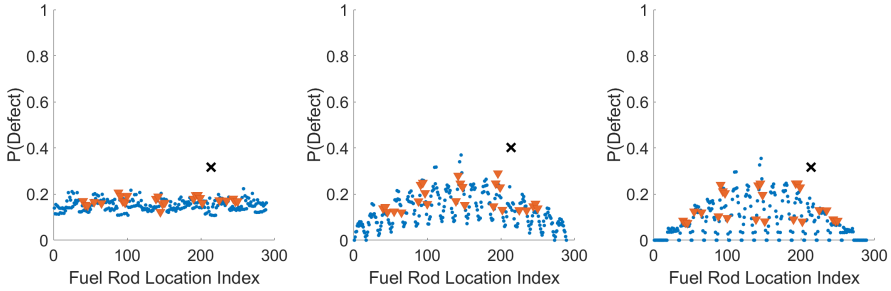


Figure C.15: Blind test 7 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

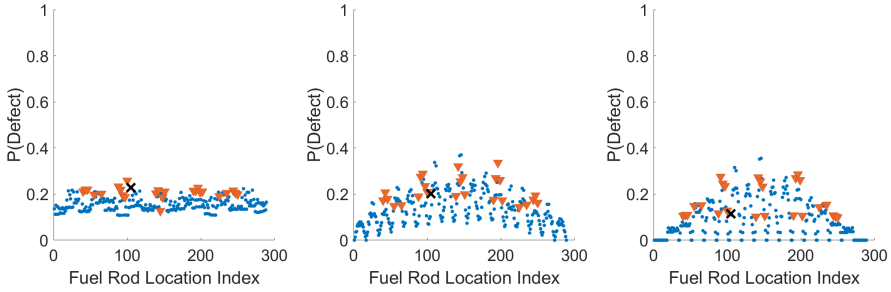


Figure C.16: Blind test 8 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

C.1.3 10 Nearest Neighbors

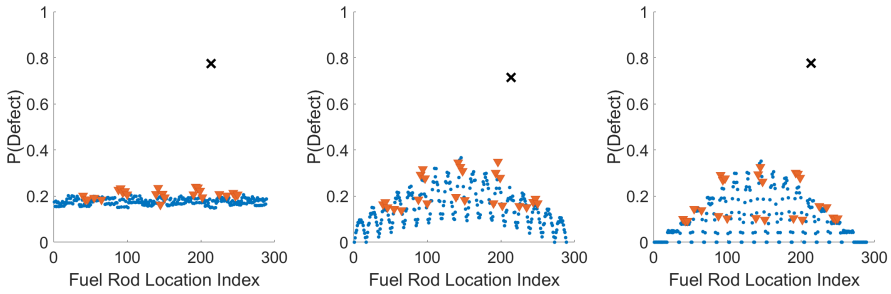


Figure C.17: Blind test 1 results for TR1 for (a) naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

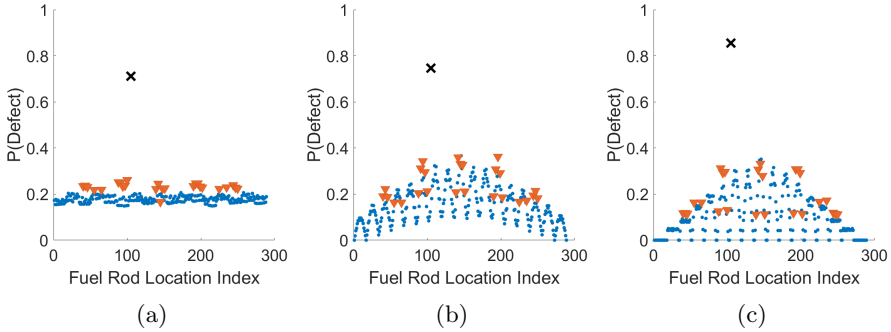


Figure C.18: Blind test 2 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

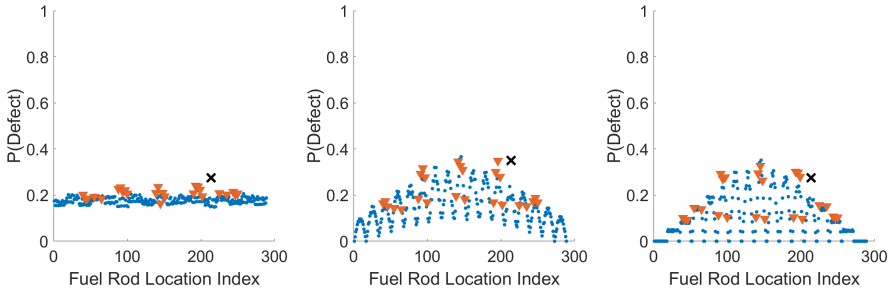


Figure C.19: Blind test 3 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

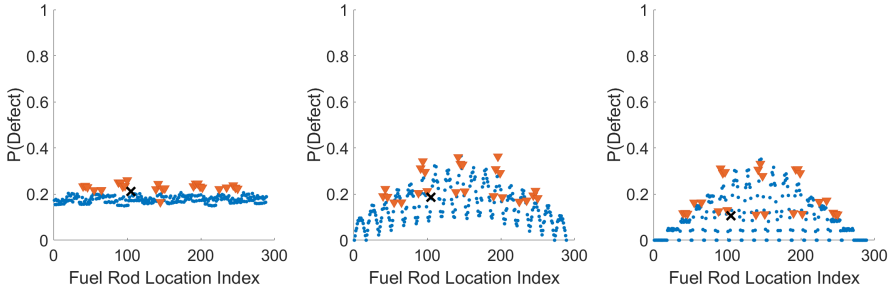


Figure C.20: Blind test 4 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

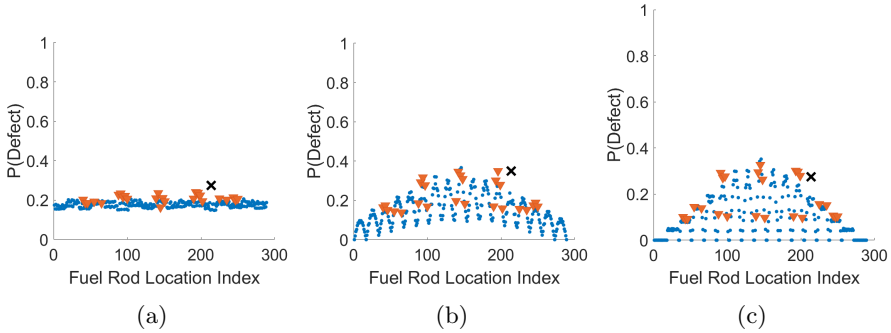


Figure C.21: Blind test 5 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

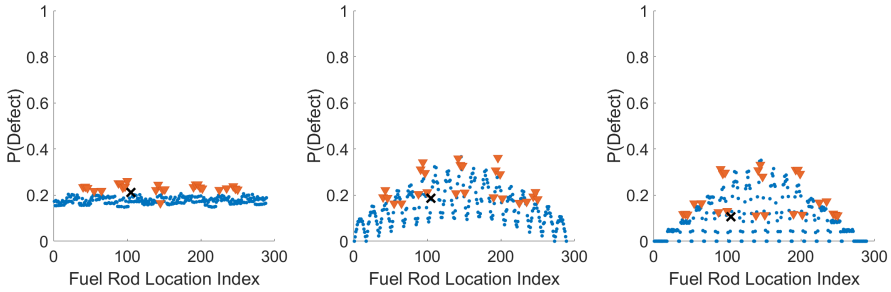


Figure C.22: Blind test 6 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

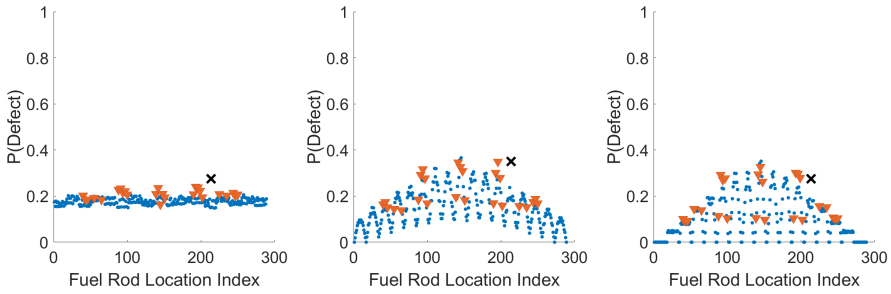


Figure C.23: Blind test 7 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

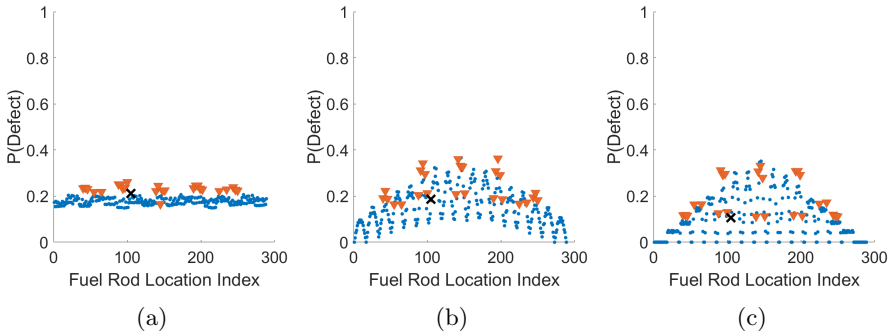


Figure C.24: Blind test 8 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

C.1.4 25 Nearest Neighbors

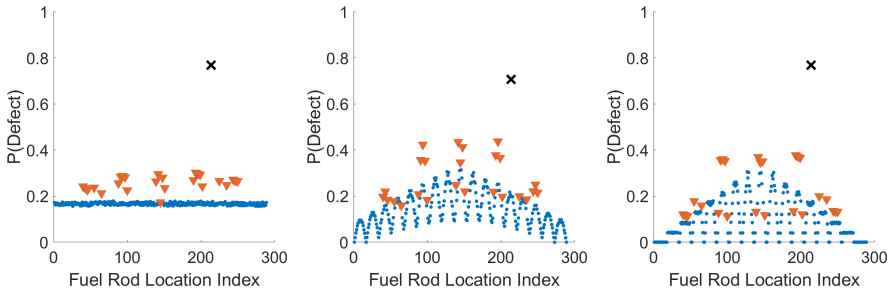


Figure C.25: Blind test 1 results for TR1 for (a) naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

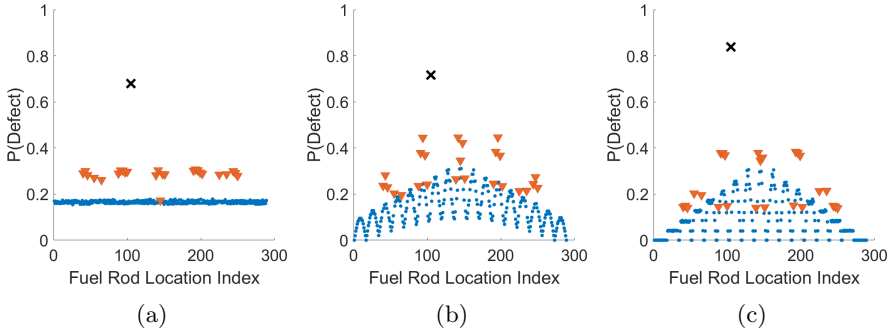


Figure C.26: Blind test 2 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

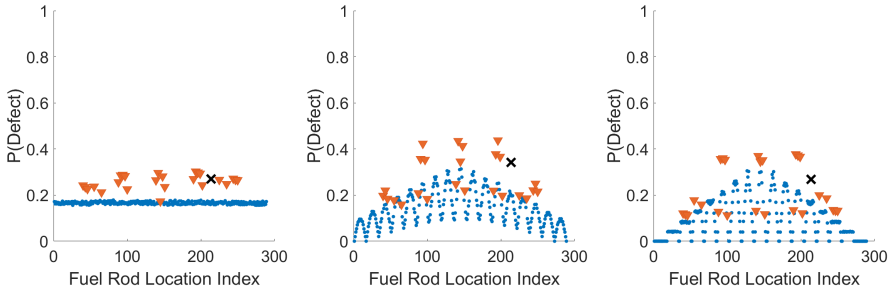


Figure C.27: Blind test 3 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

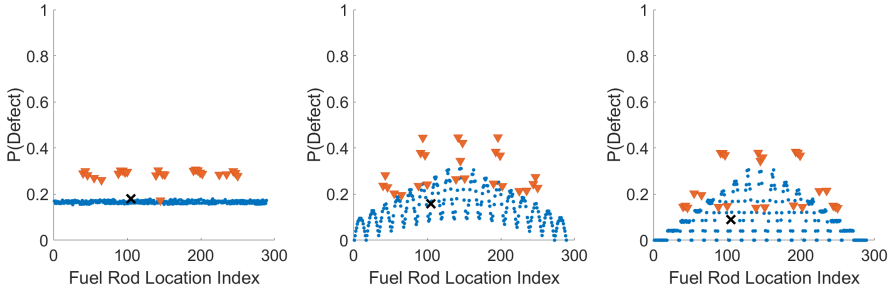


Figure C.28: Blind test 4 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

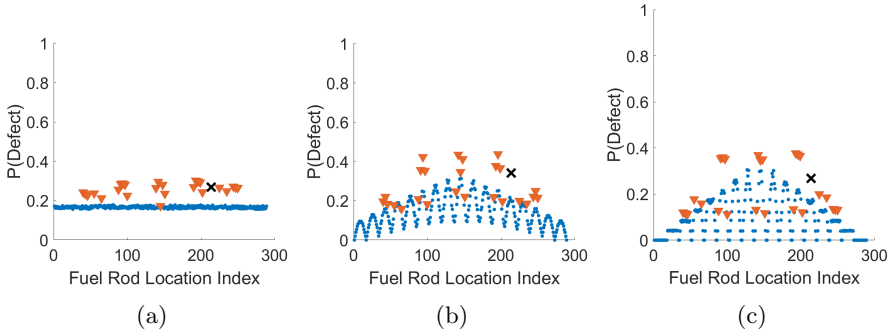


Figure C.29: Blind test 5 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

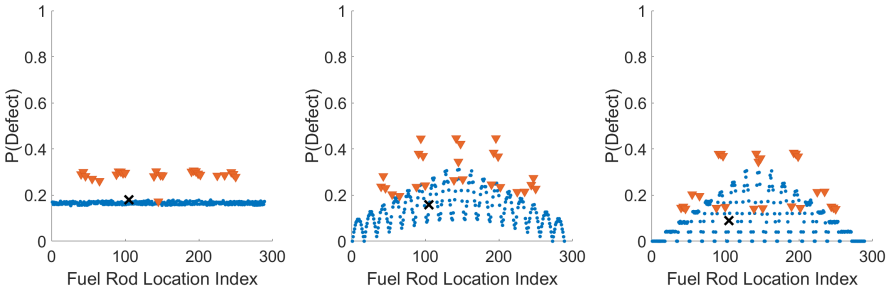


Figure C.30: Blind test 6 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

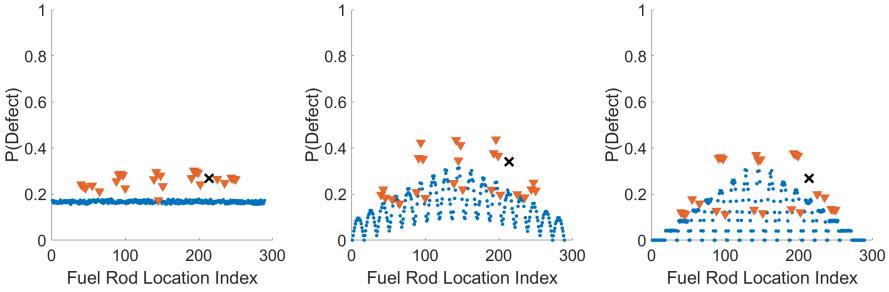


Figure C.31: Blind test 7 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

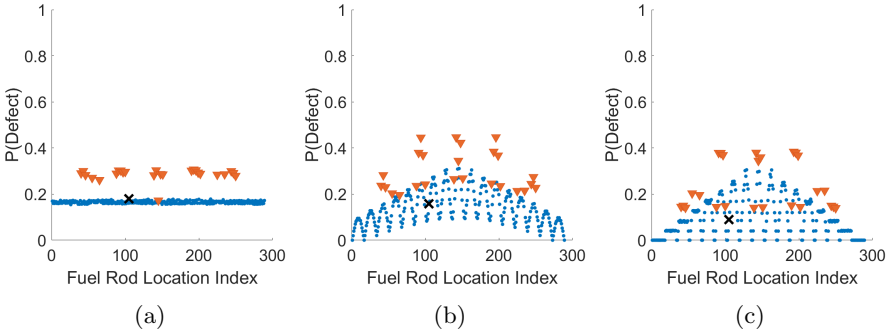


Figure C.32: Blind test 8 results for TR1 for (a) Naive weighting, (b) Euclidean weighting and (c) Chebychev weighting.

C.2 TR2 Results

In Tables C.5 and C.6, we present the results for DCVD only, GET only, and the three BNCA+LOP weighting schemes using two different metrics. The first metric is how often the true defect is presented separately from the guide tubes within the defect class, thus identifying whether a third class is needed in future work. The second metric is a standard true positive rate when the distribution means for each fuel rod are thresholded at 0.5 into the 0/1 non-defect/defect classes.

Table C.5: Average frequency that the true defect mean is far from the guide tube subset average mean for DCVD, GET and BNCA+LOP Pipeline under TR2 training.

	True Defect Separates from Guide Tubes(%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	54.55	54.55	50.00	0.00
GET	20.00	30.00	20.00	30.00
BNCA+LOP: Naive	49.49	60.61	55.56	33.33
BNCA+LOP: Euclidean	50.51	60.61	36.36	24.24
BNCA+LOP: Chebychev	52.53	53.54	27.27	27.27

Table C.6: Average true positive defect classification rate by thresholding distribution means at 0.5 for DCVD, GET and BNCA+LOP Pipeline under TR2 training.

	True Positive Rate 0.5 (%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	4.85	6.52	6.39	73.25
GET	3.46	5.39	3.85	3.85
BNCA+LOP: Naive	2.56	6.84	4.27	5.56
BNCA+LOP: Euclidean	2.14	1.76	4.70	6.41
BNCA+LOP: Chebychev	2.57	7.15	4.70	6.41

C.3 TR3 Results

In Tables C.7 and C.8, we present the results for DCVD only, GET only, and the three BNCA+LOP weighting schemes using two different metrics. The first metric is how often the true defect is presented separately from the guide tubes within the defect class, thus identifying whether a third class is needed in future work. The second metric is a standard true positive rate when the distribution means for each fuel rod are thresholded at 0.5 into the 0/1 non-defect/defect classes.

Table C.7: Average frequency that the true defect mean is far from the guide tube subset average mean for DCVD, GET and BNCA+LOP Pipeline under TR3 training.

	True Defect Separates from Guide Tubes(%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	4.13	5.05	11.11	5.46
GET	6.06	4.04	2.02	2.73
BNCA+LOP: Naive	5.05	5.05	1.01	51.52
BNCA+LOP: Euclidean	54.55	9.09	6.06	48.48
BNCA+LOP: Chebychev	10.10	10.10	7.07	51.52

Table C.8: Average true positive defect classification rate by thresholding distribution means at 0.5 for DCVD, GET and BNCA+LOP Pipeline under TR3 training.

	True Positive Rate 0.5 (%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	1.54	1.54	0.00	56.05
GET	0.31	0.31	0.31	0.56
BNCA+LOP: Naive	0.31	0.31	0.31	0.31
BNCA+LOP: Euclidean	0.93	0.31	0.31	0.31
BNCA+LOP: Chebychev	0.93	0.31	0.31	0.31

C.4 TR4 Results

In Tables C.9 and C.10, we present the results for DCVD only, GET only, and the three BNCA+LOP weighting schemes using two different metrics. The first metric is how often the true defect is presented separately from the guide tubes within the defect class, thus identifying whether a third class is needed in future work. The second metric is a standard true positive rate when the distribution means for each fuel rod are thresholded at 0.5 into the 0/1 non-defect/defect classes.

Table C.9: Average frequency that the true defect mean is far from the guide tube subset average mean for DCVD, GET and BNCA+LOP Pipeline under TR4 training.

	True Defect Separates from Guide Tubes(%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	16.16	16.16	0.00	6.06
GET	5.05	10.74	11.11	11.11
BNCA+LOP: Naive	5.05	16.16	11.11	39.39
BNCA+LOP: Euclidean	43.43	19.19	14.14	34.34
BNCA+LOP: Chebychev	10.10	20.20	16.16	38.38

Table C.10: Average true positive defect classification rate by thresholding distribution means at 0.5 for DCVD, GET and BNCA+LOP Pipeline under TR4 training.

	True Positive Rate 0.5 (%)			
	2 NN	5 NN	10 NN	25 NN
DCVD	1.54	1.54	0.00	56.10
GET	0.62	3.09	2.47	2.47
BNCA+LOP: Naive	0.86	4.70	3.85	5.56
BNCA+LOP: Euclidean	1.47	4.27	4.27	5.98
BNCA+LOP: Chebychev	1.47	4.27	4.27	5.56