2017

Feature and Decision Level Fusion Using Multiple Kernel Learning and Fuzzy Integrals

Tony Pinar
Michigan Technological University, ajpinar@mtu.edu

FEATURE AND DECISION LEVEL FUSION USING MULTIPLE KERNEL LEARNING AND FUZZY INTEGRALS

By

Anthony J. Pinar

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Electrical Engineering.

Department of Electrical and Computer Engineering

Dissertation Advisor:	*Dr. Timothy C. Havens*
Committee Member:	*Dr. Jeremy Bos*
Committee Member:	*Dr. Timothy Schulz*
Committee Member:	*Dr. Thomas Oommen*
Department Chair:	*Dr. Daniel Fuhrmann*

Dedication

To Noelle and Malcolm

who collectively are my greatest inspiration.

Contents

List of Figures . . . . . **xv**

List of Tables . . . . . **xxiii**

Preface . . . . . **xxvii**

Abstract . . . . . **xxxi**

1 Introduction **1**

1.1 Problem Context 3

1.1.1 SVMs and Kernels 5

1.1.2 The Choquet Fuzzy Integral 7

1.2 Dissertation Outline and Contributions 8

1.3 List of Relevant Publications 10

2 Efficient Multiple Kernel Classification using Feature and Decision Level Fusion **13**

2.1 Introduction 14

2.2 Data Fusion 17

2.2.1 Feature In—Feature Out Fusion 18
2.2.2 Decision In—Decision Out Fusion 19
2.3 Fuzzy Measures and Fuzzy Integrals 20
2.3.1 Fuzzy measure 20
2.3.2 Fuzzy integral 22
2.4 Multiple Kernel Learning 23
2.4.1 The GAMKL$_p$ algorithm 25
2.4.2 The DeFIMKL algorithm 29
2.4.3 The DeLSMKL Algorithm 34
2.4.4 The DeGAMKL$_p$ Algorithm 36
2.5 The Nyström Approximation for Gram Matrices 37
2.5.1 Background 37
2.5.2 Error Bounds and Other Development 38
2.5.3 Efficient MKL using the Nyström Approximation 40
2.6 Preliminary Experiments and Results 42
2.6.1 Datasets and Algorithm Parameters 43
2.6.2 Results 44
2.7 Experiments with the Nyström Approximation 47
2.7.1 Results 48
2.8 Application of the Nyström Approximation to a Large Data Set . . 53
2.8.1 Results 53

2.9 Conclusion 55

3 Visualization and Learning of the Choquet Integral With Limited Training Data 61

3.1 Introduction 62

3.2 Fuzzy Measures and Fuzzy Integrals 64

3.2.1 Fuzzy measures 65

3.2.2 Fuzzy integrals 66

3.2.3 Common Aggregations via the Choquet Integral 67

3.2.4 Visualizing the Fuzzy Integral 67

3.3 The DeFIMKL Algorithm 70

3.3.1 FM Learning Behavior with Insufficient Training Data . . . 76

3.4 FM Learning with a Specified Goal 77

3.4.1 ℓ_2- Goal Regularization 79

3.4.2 ℓ_1- Goal Regularization 80

3.4.3 Specific Aggregation Examples with Goal Regularization . . 82

3.4.3.1 Minimum Aggregation 83

3.4.3.2 Maximum Aggregation 83

3.4.3.3 Mean Aggregation 84

3.5 Learning the Goal 85

3.5.1 Defining a FM from a LOS 85

3.5.2 ℓ_2-LOS Regularization 87

3.5.3 ℓ_1-LOS Regularization 88

3.5.3.1 LOS Aggregation 90

3.6 Experiments 91

3.6.1 Results 91

3.7 Conclusion 93

4 Measures of the Shapley Index for Learning Lower Complexity Fuzzy Integrals 95

4.1 Introduction 96

4.2 Fuzzy Measure and Integral 100

4.2.1 Shapley and Interaction Indices 102

4.2.2 Existing Indices for Capacity Complexity 105

4.2.3 New Indices of Complexity Based on the Shapley Values . . 113

4.3 Sum of Squared Error and Quadratic Programming 118

4.4 Gini-Simpson Index-Based Regularization of the Shapley Values . . 122

4.5 ℓ_0-Norm Based Regularization of the Shapley Values 125

4.6 Experiments 127

4.6.1 Experiment 1: Important, Relevant and Irrelevant Inputs . . 128

4.6.2 Experiment 2: Random AWGN Noise 132

4.6.3 Experiment 3: Iteratively Reweighted ℓ_1-Norm 132

4.6.4 Experiment 4: Multiple Kernel Learning 134

4.7 Conclusion and Future Work 138

5 Applications to Explosive Hazard Detection with Ground Penetrating Radar 141

5.1 Introduction 142

I A Comparison of Robust Principal Component Analysis Techniques for Buried Object Detection in Downward Looking GPR Sensor Data 144

5.2 Classical Principal Component Analysis (PCA) and Robust Principal Component Analysis (RPCA) 145

5.2.1 Robust Principal Component Analysis (RPCA) 146

5.3 Ground Penetrating Radar 152

5.3.1 Data Format and Visualization 153

5.3.2 RPCA Decomposition of GPR Data 154

5.3.3 Returned Energy and Signal to Clutter Ratio 156

5.4 RPCA Experiments 158

5.4.1 Overall Results 159

5.4.2 Results Based on Target Type 160

5.4.3 Decomposition Time 163

5.4.4 Effect of Parameter Selection on Select Algorithms 164

5.4.5 Preprocessing Conclusions 168

II Approach to Explosive Hazard Detection Using Sensor Fusion and Multiple Kernel Learning with Downward-Looking GPR and EMI Sensor Data 170

5.5 Image Formation 171

5.5.1 Clutter Removal 173

5.5.2 Image Ensemble 174

5.6 CFAR Prescreener 175

5.7 Sensor Fusion 177

5.7.1 Run Packing 177

5.7.2 Composite Confidence Maps 178

5.7.2.1 CCM via Summation Method 179

5.7.2.2 CCM via Maximum Method 179

5.7.2.3 Blurring Functions 180

5.8 Features 181

5.8.1 Local Statistics 181

5.8.2 Histogram of Oriented Gradients 182

5.8.3 Local Binary Patterns 183

5.8.4 Fast Finite Shearlet Transform 186

5.9 Results 188

5.9.1 Performance Metric: NAUC 188

5.9.2 Prescreener Results 190

5.9.2.1 Prescreener with RP 191
5.9.2.2 Prescreener with CCM 193
5.9.3 Prescreener with RP and CCM 193
5.9.4 SKSVM 194
5.9.4.1 SKSVM using Prescreener Hits 194
5.9.4.2 SKSVM using CCM 195
5.9.4.3 SKSVM using RP + CCM 196
5.9.5 MKLSVM 197
5.9.5.1 MKLSVM using Prescreener Hits 197
5.9.5.2 MKLSVM using CCM 198
5.9.5.3 MKLSVM using RP + CCM 199
5.9.5.4 Results Summary 199
5.10 Conclusion 201

III Explosive Hazard Detection with Feature and Decision Level Fusion, Multiple Kernel Learning, and Fuzzy Integrals 203

5.11 Explosive Hazard Detection Dataset 204
5.11.1 Prescreener and Feature Extraction 204
5.12 Experiments and Results 205
5.12.1 Experiment 1 206

5.12.2 Experiment 2 208

5.13 Conclusion 209

6 Conclusion 211

6.1 Future Work 213

References 217

A Support Vector Machines and MKLGL 241

A.0.1 Linear Support Vector Machines 241

A.0.2 Single Kernel Support Vector Machines 243

A.0.3 Multiple Kernel Learning Support Vector Machines 244

B Tibshirani's Lasso Algorithm 247

B.0.1 Summary of Algorithm 248

C Letters of Permission 251

List of Figures

1.1 High-level block diagram of feature-level fusion. 4

1.2 High-level block diagram of decision-level fusion. 5

2.1 Lattice of FM elements for $n = 3$. Monotonicity (P5) is illustrated by the size of each circle, i.e., $g(\{x_1\}) \leq g(\{x_1, x_2\})$ as $\{x_1\} \subset \{x_1, x_2\}$. 22

2.2 DeFIMKL performance using regularization on Sonar data. Error bars indicate ± one standard deviation. 46

2.3 DeFIMKL performance using regularization on Dermatology data—classes $\{1, 2, 3\}$ versus $\{4, 5, 6\}$. Error bars indicate ± one standard deviation. 47

2.4 Results of using GAMKL_1 on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%. 50

2.5 Results of using $GAMKL_2$ on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%. 50

2.6 Results of using $DeFIMKL_1$ on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%. 51

2.7 Results of using $DeFIMKL_2$ on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%. 51

2.8 Average speed-up percentages of classifiers using the Nyström approximation. 52

2.9 DeFIMKL performance using regularization on MNIST data. Error bars indicate ± one standard deviation. 54

3.1 Lattice of FM elements for $n = 3$. Monotonicity (P5) is illustrated by the size of each node, i.e., $g(\{x_1\}) \leq g(\{x_1, x_2\})$ as $\{x_1\} \subset \{x_1, x_2\}$. Note that shorthand notation is used where $g(1, 3)$ is equivalent to $g(\{x_1, x_3\})$. 68

3.2 The path taken by the Choquet integral due to a single input inducing the permutation $\pi = \{2, 1, 3\}$. Note that the FM was arbitrarily defined in this example, and their distribution (ordering) follows that of Figure 3.1. 69

3.3 Lattice of learned FM and paths for random training data from the Ionosphere data set using $m = 10$. Note there are numerous untouched nodes and their learned values are driven by the constraints in (3.9). 69

4.1 Experiment 1 results. (a,b) Learned FM values in lexicographical order for $\lambda = 0$ to 50. Bin 1 is $\mathbf{u}(1) = g(x_1)$, bin $N + 1$ is $\mathbf{u}(N + 1) = g(\{x_1, x_2\})$, etc. Height of bar indicates FM value; color indicates λ value. (c,d) Plots showing performance of each regularization method in terms of SSE and Gini-Simpson index of the learned FM at each regularization parameter λ. 130

(a) Learned FM values using Gini-Simpson regularization 130

(b) Learned FM values using ℓ_1 regularization 130

(c) Gini-Simpson regularization performance plots 130

(d) ℓ_1 regularization performance plots 130

4.2 Experiment 2 results. (a,b) Learned FM values in lexicographical order. Bin 1 is $\mathbf{u}(1) = g(x_1)$, bin $N+1$ is $\mathbf{u}(N+1) = g(\{x_1, x_2\})$, etc. Height of the bar indicates FM value; color indicates λ value. (c,d) Plots showing performance of each regularization method in terms of SSE and Gini-Simpson index values of the learned FM at each regularization parameter λ. 133

(a) Learned FM values using Gini-Simpson regularization 133

(b) Learned FM values using ℓ_1 regularization 133

(c) Gini-Simpson regularization performance plots 133

(d) ℓ_1 regularization performance plots 133

4.3 Experiment 3 results. Learned FM values in lexicographical order for Experiment 1. Bin 1 is $\mathbf{u}(1) = g(x_1)$, bin $N+1$ is $\mathbf{u}(N+1) = g(\{x_1, x_2\})$, etc. Height of the bar indicates FM value; color indicates iteration number. Plot of the Shapley values of the learned FM for Experiment 1 at each iteration. 134

(a) Learned FM values using iteratively reweighted ℓ_1 regularization for Experiment 1 134

(b) Shapley values for (a) 134

5.1 An image contaminated with a small amount of noise and its low-rank and sparse decomposition. 147

5.2 An image contaminated with a large amount of noise and its low-rank and sparse decomposition. 148

5.3 The low-rank and sparse decomposition of an image with only one pixel contaminated with a large amount of noise. 149

5.4 NAUC 153

5.5 Examples of typical GPR data visualization 154

5.6 Sparse component of GPR B-scan from Figure 5.5(b), computed using RPCA 155

5.7 Returned energy for a single sweep across a lane, over a target. Vertical lines indicate the region in which the target is buried. 157

5.8 Example calculation of the peak SCR 158

5.9 Unprocessed target B-scan used in Figures 5.10 and 5.12 165

5.10 A single B-scan processed with PCP-AD using multiple values for λ along with the integrated energies. Note that image is labeled either *time* or *frequency*, corresponding to the domain in which the PCP-AD algorithm was implemented. Dashed vertical lines denote the target region. 166

5.11 Effect of the tuning parameter on the SCR when using PCP-AD. Note that the frequency domain trace is clamped to zero for situations where computing the SCR is numerically limited, i.e., in cases where 0/0 is approached. 167

5.12 A single B-scan processed with OP-RPCA in the time domain using multiple values for λ along with the integrated energies. Dashed vertical lines denote the target region. 168

5.13 Effect of the tuning parameter on the SCR when using OP-RPCA. 169

5.14 Scattered plot of integrated A-scan energy for Lane A, GPR Channel 1. 171

5.15 Linear interpolation of scatter plot in Figure 5.14. 172

5.16 Results of applying clutter removal with $m = 0.85$ to Figure 5.14. . 172

5.17 Ascan 175

5.18 Ascan 177

5.19 CCMExample 180

5.20 Sub-image at hit candidate location. 182

5.21 Gradient calculation with 3×3 cell arrangement. 183

5.22 Cell based 9-bin HOG feature. 184

5.23 freqPartition 187

5.24 FFSTfilters 187

5.25 NAUC 190

5.26 prelanes 192

5.27 Integrated energy ground map and an example queue point with 10, 20, 30, and 40 cm disks. 205

5.28 DeFIMKL performance using regularization. Error bars indicate ± one standard deviation. 208

List of Tables

2.1 Acronyms and Select Notation 16

2.2 UCI Benchmark Data Sets 29

2.3 RBF Kernel Parameter Ranges 43

2.4 Classification Accuracy Results on Benchmark Data Sets* 58

2.5 Additional Data Sets for Nyström Verification 59

2.6 Nyström Sampling Percentage Required to Achieve Equivalent Classification Results as Full Sample 59

3.1 Underlying and learned FMs (excluding $g(\{\emptyset\})$ and $g(X)$ whose values are 0 and 1, respectively, due to the boundary conditions). The learned FM terms marked with an asterisk are not addressed by the training data. Regularization labels indicate the type of norm employed and the aggregation goal. For example, "ℓ_1-min" indicates a goal of that for minimum aggregation ($\mathbf{g} = \mathbf{0}$) and ℓ_1-regularization ($\|\mathbf{u} - \mathbf{g}\|_1$). 78

3.2 Classification Accuracy of Various Regularization Functions* 92

4.1 Numeric Examples, for $N = 3$, Illustrating ℓ_0 and Gini-Simpson Differences. 117

4.2 RBF Kernel Parameter Ranges and Data Set Properties 137

4.3 Classifier Performances—Means and Standard Deviations 137

5.1 Average SCR over all targets* 160

5.2 Average SCR over all wire targets* 161

5.3 Average SCR over all landmine targets* 161

5.4 Average SCR over all pressure plate targets* 162

5.5 Average SCR over all main charge targets* 163

5.6 Average RPCA decomposition time per sweep in seconds* 164

5.7 Length of Features and Full Feature Set. 181

5.8 Results of CFAR Prescreener on each Channel and Lane. 191

5.9 NAUCs for the ROCs. 193

5.10 Results of SKSVM Classifier on each Channel and Lane using Different Features—Prescreener Hits. 195

5.11 Results of SKSVM Classifier on each Channel and Lane using Different Features—CCM Hits. 196

5.12 Results of SKSVM Classifier on each Channel and Lane using Different Features—RP + CCM Hits. 197

5.13 Results of MKLSVM Classifier on each Channel and Lane using Different Features—Prescreener Hits. 198

5.14 Results of MKLSVM Classifier on each Channel and Lane using Different Features—CCM Hits. 199

5.15 Results of MKLSVM Classifier on each Channel and Lane using Different Features—RP + CCM Hits. 200

5.16 Summary of Best Results. 201

5.17 NAUCs and percentage improvement compared to the prescreener*. 207

Preface

Some chapters of this dissertation contain published material. The following list indicates which publications were used along with notes on author contributions.

Chapter 2

A. Pinar, T.C. Havens, D.T. Anderson, and L. Hu (2015). "Feature and decision level fusion using multiple kernel learning and fuzzy integrals," *Proc. FUZZ-IEEE*, Aug 2015, pp. 1–7. (See [1].)

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and the ideas proposed in the paper stemmed from conversations with D.T. Anderson and L. Hu.

A.J. Pinar, J. Rice, L. Hu, D.T. Anderson, and T.C. Havens (2016). "Efficient multiple kernel classification using feature and decision level fusion," *IEEE Trans. Fuzzy Systems*, Vol. PP. (See [2].)

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and the ideas proposed

in the paper stemmed from conversations with D.T. Anderson and L. Hu. J. Rice contributed ideas to improve the implementation of the various algorithms discussed in the paper.

Chapter 3

A.J. Pinar, T.C. Havens, M.A. Islam, and D.T. Anderson (2017). "Visualization and learning of the Choquet integral with limited training data," *To appear, Proc. FUZZ-IEEE.*

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and the ideas proposed in the paper stemmed from conversations with D.T. Anderson and M.A. Islam.

A.J. Pinar, T.C. Havens, M.A. Islam, and D.T. Anderson (2017). "Learning the Choquet Integral with a goal," *In preparation, IEEE Trans. Fuzzy Systems.*

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and the ideas proposed in the paper stemmed from conversations with D.T. Anderson and M.A. Islam.

Chapter 4

A.J. Pinar, D.T. Anderson, A. Zare, T.C. Havens, and T. Adeyeba (2017). "Measures of the Shapley Index for Learning Lower Complexity Fuzzy Integrals," *In review, Springer Granular Computing.*

The ideas presented in this paper are the result of discussions between all listed authors. A.J. Pinar is the corresponding author for this paper and generated the experimental results. D.T. Anderson and T.C. Havens contributed the theoretical background.

Chapter 5

A. Pinar, T.C. Havens, J. Rice, M. Masarik, J. Burns, and B. Thelen (2016). "A comparison of robust principal component analysis techniques for buried object detection in downward looking GPR and EMI sensor data," *Proc. SPIE*, pp. 98230T. (See [3].)

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and J. Rice assisted with the code written for the experiments. M. Masarik, J. Burns, and B. Thelen provided

data for use in the experiments.

A.J. Pinar, J. Rice, T.C. Havens, M. Masarik, J. Burns, and D.T. Anderson (2016). "Explosive hazard detection with feature and decision level fusion, multiple kernel learning, and fuzzy integrals," *Proc. IEEE CISDA*, pp. 1–8. (See [4].)

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and J. Rice assisted with the code written for the experiments. M. Masarik and J. Burns provided data for use in the experiments, and D.T. Anderson provided guidance with the theoretical components of the paper.

A. Pinar, M. Masarik, T.C. Havens, J. Burns, B. Thelen, and J. Becker (2015). "Approach to explosive hazard detection using sensor fusion and multiple kernel learning with downward-looking GPR and EMI sensor data," *Proc. SPIE*, pp. 94540B. (See [5].)

A.J. Pinar is the leading researcher in this work and is the corresponding author. The research was performed under the guidance of T.C. Havens, and M. Masarik, J. Burns, and B. Thelen provided data and algorithms for use in the experiments. Some ideas in this work originated in conversations with J. Becker.

Abstract

The work collected in this dissertation addresses the problem of *data fusion*. In other words, this is the problem of making decisions (also known as the problem of *classification* in the machine learning and statistics communities) when data from multiple sources are available, or when decisions/confidence levels from a panel of decision-makers are accessible. This problem has become increasingly important in recent years, especially with the ever-increasing popularity of autonomous systems outfitted with suites of sensors and the dawn of the "age of big data." While data fusion is a very broad topic, the work in this dissertation considers two very specific techniques: *feature-level fusion* and *decision-level fusion*. In general, the fusion methods proposed throughout this dissertation rely on *kernel methods* and *fuzzy integrals*. Both are very powerful tools, however, they also come with challenges, some of which are summarized below. I address these challenges in this dissertation.

Kernel methods for classification is a well-studied area in which data are implicitly mapped from a lower-dimensional space to a higher-dimensional space to improve classification accuracy. However, for most kernel methods, one must still choose a kernel to use for the problem. Since there is, in general, no way of knowing which kernel is the best, *multiple kernel learning* (MKL) is a technique used to learn the

aggregation of a set of valid kernels into a single (ideally) superior kernel. The aggregation can be done using weighted sums of the pre-computed kernels, but determining the summation weights is not a trivial task. Furthermore, MKL does not work well with large datasets because of limited storage space and prediction speed. These challenges are tackled by the introduction of many new algorithms in the following chapters. I also address MKL's storage and speed drawbacks, allowing MKL-based techniques to be applied to big data efficiently.

Some algorithms in this work are based on the *Choquet fuzzy integral*, a powerful nonlinear aggregation operator parameterized by the *fuzzy measure* (FM). These decision-level fusion algorithms learn a fuzzy measure by minimizing a *sum of squared error* (SSE) criterion based on a set of training data. The flexibility of the Choquet integral comes with a cost, however—given a set of N decision makers, the size of the FM the algorithm must learn is 2^N. This means that the training data must be diverse enough to include 2^N independent observations, though this is rarely encountered in practice. I address this in the following chapters via many different *regularization* functions, a popular technique in machine learning and statistics used to prevent overfitting and increase model generalization. Finally, it is worth noting that the aggregation behavior of the Choquet integral is not intuitive. I tackle this by proposing a quantitative visualization strategy allowing the FM and Choquet integral behavior to be shown simultaneously.

Chapter 1

Introduction

Information fusion is a broad multi-disciplinary topic with many applications. Generally, it refers to the process of aggregating multiple sets of data (or other type of information) all explaining a common "thing". For example, in the case of an autonomous robot, the "thing" might be its environment and the data will likely be collected from a suite of heterogeneous sensors outfitted on the vehicle. The robot can gain full autonomy if there are algorithms present that can effectively fuse the various data it collects, and make decisions based on their aggregation. Another example more related to the autonomous agents of our species is the discrimination of edible food versus inedible food. When presented with a nutritional candidate, humans use their inborn sensor suites to extract informative features regarding its edibility. For example, its smell may contain information regarding the candidate's

freshness—if the candidate smells rotten, it is likely inedible. Sight allows humans to categorize the candidate which also helps the decision process—if it looks like other foods known to be edible, it may also be edible. Ultimately, humans combine the information gathered by these senses (and others) to make the decision to eat or not to eat.

The above examples are examples of *data-level* or *feature-level* fusion, where the decision maker essentially combined the data before any decisions were made. Another flavor of information fusion is *decision-level* fusion, where there exists a panel of decision makers, each making his or her own decision based on the data at hand, and an overall decision is then determined through some process, e.g., voting. It is important to note that in this context the decision can be a "confidence" or "rating." A popular example of this type of fusion is the judging process in gymnastic or figure skating competitions—a panel of judges each rate the performance of a competitor by casting grades based on their own observations, and an overall score is assigned to the competitor by aggregating the judges' ratings.

These high-level examples illuminate the two types of information fusion addressed in this dissertation: data-level fusion and feature-level fusion. The remainder of this chapter provides some context on how the low-level tools comprising the bulk of this dissertation fall under the roof of information fusion, and it will conclude with an outline of contributions. Furthermore, salient terminology is introduced and

explained throughout the chapter.

1.1 Problem Context

It is no coincidence that the high-level examples of the previous section ultimately ended with a *decision*; essentially all of the algorithms discussed in this dissertation are binary decision makers, i.e., binary classifiers.[1] Hence, following the typical workflow for many machine learning or statistical prediction methods, the algorithms discussed later will be *trained* on a set of *training data*—data accurately labeled with known classes (labels) and is representative of the problem at hand, then tested on *testing data*—data with unknown labels. The training process allows the classifier to learn the underlying model so that accurate predictions can be made on the testing data.

An example of a high-level feature-level fusion pipeline is shown in Figure 1.1. The input data, which can be from various sources or even a single source, is shown on the left and is fed into the first processing blocks that process the data in some way[2]. Next, the features are fused in some manner before a single classifier gives an overall decision. In the work that follows, I use an "off the shelf" classifier known as a *support*

[1]While the classifier I use in these algorithms is the *support vector machine* (SVM), generally any classifier can be used in its place.
[2]In the jargon introduced in the following section, these blocks are all different *kernels*.

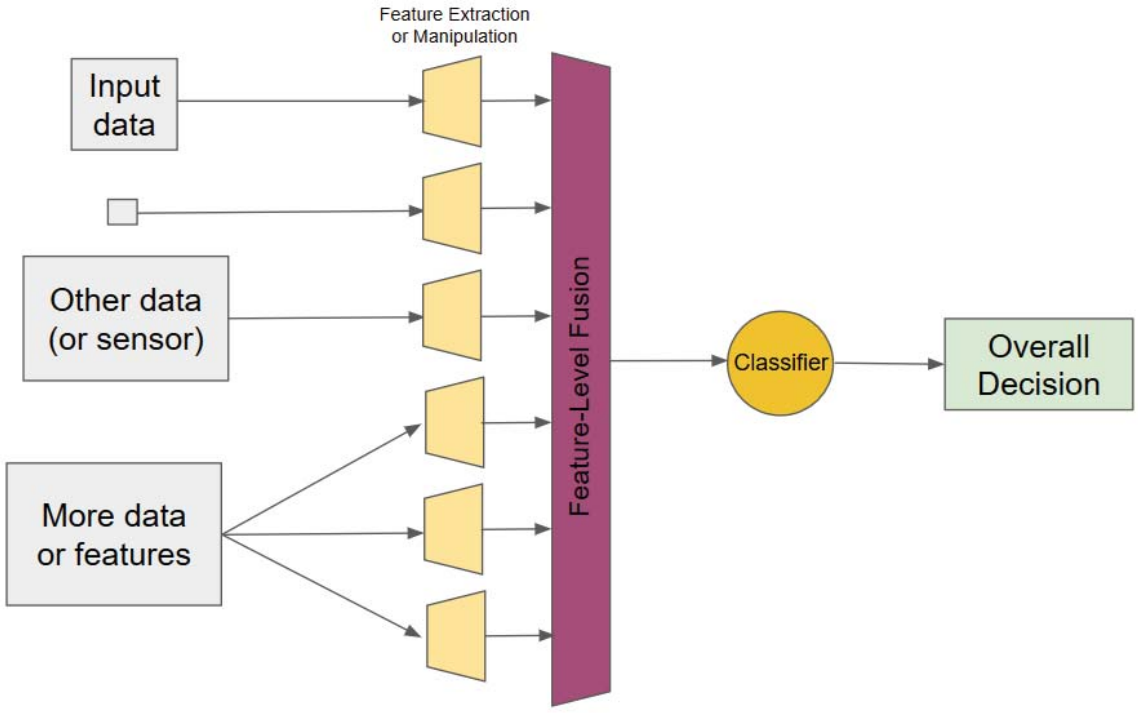


Figure 1.1: High-level block diagram of feature-level fusion.

vector machine (SVM), and the feature-level fusion algorithms I propose focus on the feature fusion block just before classification.

Figure 1.2 shows a similar block diagram for a decision-level fusion pipeline. Just as with feature-level fusion, the input data is on the left and is fed to some processing blocks. The difference here is that classification is performed *before* the fusion block; each processing block gets its very own classifier. The decisions generated by the different classifiers are then aggregated to form an overall decision by the fusion block. Again, the classifiers I use are SVMs and the decision-level fusion methods discussed later are included in the decision fusion block.

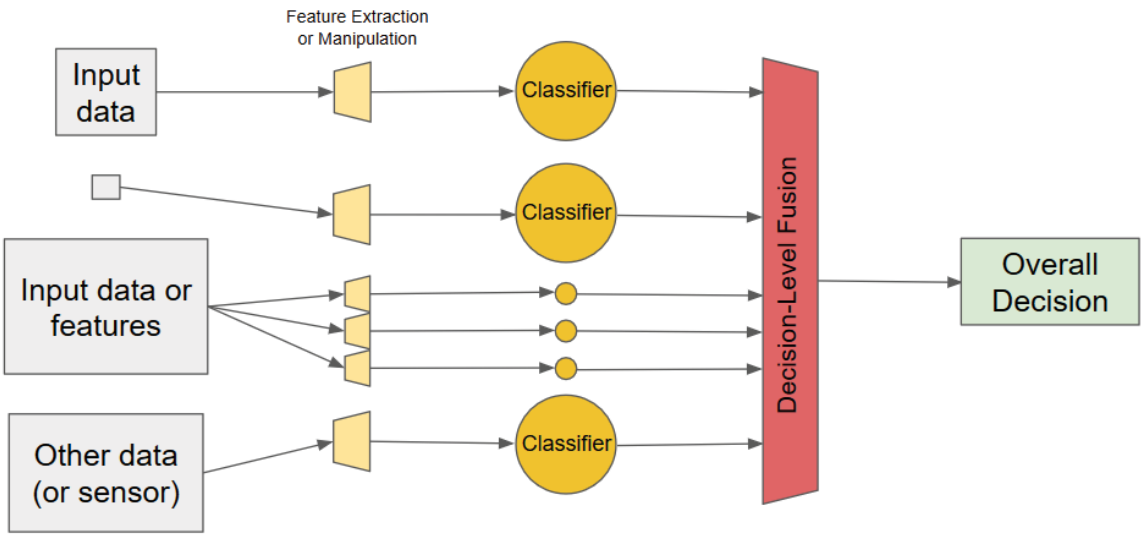


Figure 1.2: High-level block diagram of decision-level fusion.

The following sections briefly explain the tools used for fusion in the following chapters. Specifically, kernel SVMs and multiple kernel learning are discussed as the tools chosen for classification and feature-level fusion, respectively, and the Choquet fuzzy integral is introduced as the tool of choice for decision-level fusion.

1.1.1 SVMs and Kernels

A support vector machine is a type of binary classifier that finds a hyperplane in some space that discriminates between two classes of data; for linearly separable data, the SVM will work perfectly. This is not to say, however, that the SVM cannot be applied to more "complex" data—data that are not linearly separable can be accurately classified with a *kernel* SVM, i.e., a SVM that has been extended using the *kernel trick*.

The kernel trick allows data to be nonlinearly mapped to a new higher-dimensional space termed the *reproducing kernel Hilbert space* (RKHS), where the data are (potentially) linearly separable. A linear classifier implemented in the RKHS can then perfectly discriminate the two classes. The SVM is one of the most popular classifiers to utilize the kernel trick since its formulation turns out to be very efficient—the nonlinear mapping can be performed *implicitly* through the use of kernel matrices, Hermitian matrices whose elements represent all pairwise inner products of the training data. The elements of a kernel matrix are computed using a *kernel function*, which represents the inner product of two vectors in a RKHS defined by the kernel function chosen. There are many kernel functions to choose from, e.g., various *radial basis function* kernels, *polynomial* kernels, *sigmoidal* kernels, etc., and they each have at least one free parameter that must be chosen. This abundance of choice leads to the problem of determining which kernel (and parameter) to employ with the SVM. Recall that the goal of using a kernel is to project the data to a space where the data *are* linearly separable, something not all kernels can achieve. This is the challenge that *multiple kernel learning* (MKL) addresses.

MKL assumes that the kernel used as described in the previous paragraph is actually a linear combination of pre-selected base kernels. One must still choose the various base kernels with this MKL approach, but the process of learning the mixing coefficients generally minimizes the influence of kernels that do not work well and

maximizes the contribution of kernels that do separate the data well. In a mathematical nutshell, given m base kernel matrices, K_k, MKL is the process of learning the mixing coefficients, σ_k, that form an "optimal" kernel as

$$\mathcal{K} = \sum_{k=1}^{m} \sigma_k K_k. \tag{1.1}$$

The MKL algorithms in this dissertation all assume the formulation in (1.1), and many address the problem of learning a suitable set of mixing coefficients. Appendix A provides a more quantitative discussion of SVMs including their kernel extension.

1.1.2 The Choquet Fuzzy Integral

Most of the decision-level fusion work in this dissertation uses the *Choquet fuzzy integral* to combine the outputs of an ensemble of decision makers into a single overall decision. This integral is extremely flexible and is parametrized by the *fuzzy measure* (FM), a function that maps the power set of all decision makers to the unit interval and can be thought of as the "worth" of a set. Therefore, we can say the Choquet fuzzy integral is "uber-parametrized," since aggregating the decisions from a set of m decision-makers using the integral requires 2^m terms in its FM[3]. Similar to MKL's goal of learning the "optimal" mixing coefficients based on training data, techniques

[3]Note that due to some properties of the FM, the number of required terms is actually $2^m - 2$. This will be explained in later chapters.

using the Choquet integral learn the FM that fits the training data.

The number of required terms of the FM explodes as 2^m, so learning the FM quickly becomes an underdetermined problem since sets of training data will rarely have the diversity to include 2^m independent observations. This manifests as a learned FM that is only partially accurate—values of the FM driven by the training data are very accurately learned, but the remaining values are driven only by constraints; their values are essentially erroneous. Thus, when faced with testing data that utilizes the incorrectly learned FM values the classification accuracy will generally suffer.

Much of the work in the following chapters addresses this problem through the use of *regularization*, a technique commonly used in machine learning to prevent overfitting. Doing so reduces the influence of the constraints on the learned FM and rather reassigns the influence to the regularization function; the choice of regularization function defines *how* the values of the FM not driven by training data are learned.

1.2 Dissertation Outline and Contributions

The following chapters summarize my work on the data fusion problem along with some application-specific contributions to *ground penetrating radar* (GPR). The remainder of this section describes each chapter more concretely and explains the novel

contributions of each chapter.

Chapter 2 proposes multiple feature-level and decision-level fusion algorithms, demonstrates their performance when used with support vector machine classifiers, and proposes a general method for extending multiple kernel learning-based algorithms to large datasets through the use of the Nyström approximation. Experimental results demonstrate the algorithms' utility as well as validate their extension to "big data". A decision-level fusion algorithm proposed in this paper, namely *decision-level fuzzy integral multiple kernel learning* (DeFIMKL), is a common thread also appearing in the chapters that follow.

Chapters 3 and 4 further extend the fuzzy integral-based decision-level fusion algorithm introduced in Chapter 2 in many ways. The novelty in **Chapter 3** allows the algorithm's behavior to be more finely "tuned" towards various aggregation operators, and that of **Chapter 4** aims to improve the algorithm's generalization by penalizing its complexity during the learning process.

Chapter 5 applies some of the fusion techniques presented in this dissertation to the problem of explosive hazard detection using ground penetrating radar (GPR). The chapter is broken into three parts—**Part I** presents an exploration of various *robust principal component analysis* (RPCA) techniques employed as a data-preprocessing

step. Next, **Part II** summarizes an approach to the detection process using state-of-the-art fusion methods and providing a picture of the entire detection pipeline including prescreening, feature extraction, and classification. Finally, **Part III** applies the fusion techniques from Chapter 2 to the GPR data.

Finally, **Chapter 6** concludes the dissertation and discusses future work.

1.3 List of Relevant Publications

The research summarized in this dissertation is based on the following publications:

1. A. Pinar, T.C. Havens, D.T. Anderson, and L. Hu (2015). "Feature and decision level fusion using multiple kernel learning and fuzzy integrals," *Proc. FUZZ-IEEE*, Aug 2015, pp. 1–7. (See [1].)

2. A.J. Pinar, J. Rice, L. Hu, D.T. Anderson, and T.C. Havens (2016). "Efficient multiple kernel classification using feature and decision level fusion," *IEEE Trans. Fuzzy Systems*, Vol. PP. (See [2].)

3. A.J. Pinar, T.C. Havens, M.A. Islam, and D.T. Anderson (2017). "Visualization and learning of the Choquet integral with limited training data," *To appear, Proc. FUZZ-IEEE*.

4. A.J. Pinar, T.C. Havens, M.A. Islam, and D.T. Anderson (2017). "Learning the Choquet Integral with a goal," *In preparation, IEEE Trans. Fuzzy Systems.*

5. A.J. Pinar, D.T. Anderson, A. Zare, T.C. Havens, and T. Adeyeba (2017). "Measures of the Shapley Index for Learning Lower Complexity Fuzzy Integrals," *In review, Springer Granular Computing.*

6. A. Pinar, T.C. Havens, J. Rice, M. Masarik, J. Burns, and B. Thelen (2016). "A comparison of robust principal component analysis techniques for buried object detection in downward looking GPR and EMI sensor data," *Proc. SPIE*, pp. 98230T. (See [3].)

7. A.J. Pinar, J. Rice, T.C. Havens, M. Masarik, J. Burns, and D.T. Anderson (2016). "Explosive hazard detection with feature and decision level fusion, multiple kernel learning, and fuzzy integrals," *Proc. IEEE CISDA*, pp. 1–8. (See [4].)

8. A. Pinar, M. Masarik, T.C. Havens, J. Burns, B. Thelen, and J. Becker (2015). "Approach to explosive hazard detection using sensor fusion and multiple kernel learning with downward-looking GPR and EMI sensor data," *Proc. SPIE*, pp. 94540B. (See [5].)

Other collaborations related to this work include:

1. M.A. Islam, D.T. Anderson, **A.J. Pinar**, and T.C. Havens (2016). "Data-driven compression and efficient learning of the Choquet integral," *In review, IEEE Trans. Fuzzy Systems.*

2. T.C. Havens, J. Becker, **A. Pinar**, and T.J. Schulz (2014). "Multi-band sensor-fused explosive hazard detection in forward-looking ground-penetrating radar," *Proc. SPIE*, vol. 9072. (See [6].)

3. T.C. Havens, D.T. Anderson, K. Stone, J. Becker, and **A.J. Pinar** (2016). "Computational Intelligence in Forward Looking Explosive Hazard Detection," Chapter in *Recent Advances in Computational Intelligence in Defense and Security.* Berlin: Springer. (See [7].)

Proof of copyright permission for the necessary publications used in this dissertation are provided in Appendix C.

Chapter 2

Efficient Multiple Kernel Classification using Feature and Decision Level Fusion

The material in this chapter was previously published in IEEE Transactions on Fuzzy Systems, Vol. PP, no. 99, 2016.

2.1 Introduction

Consider a set of numerical *feature-vector* data that has the form $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, where the coordinates of $\mathbf{x}_i$ provide feature values (e.g., bits per second, speed, volts, etc.) describing some object (e.g., a wireless sensor network node, traffic camera, or radar). We are also given a set of training labels for each feature vector, such that we have the pair $(\mathbf{y}, X)$, where $\mathbf{y} = (y_1, \ldots, y_n)^T$ and y_i is the label of ith object. Each y_i is associated with a respective feature vector $\mathbf{x}_i$. The classifier learning task is thus to learn some prediction function f, such that we can predict the label of the feature-vectors, i.e., $y = f(\mathbf{x})$.

Most classifiers delineate the classes by finding some "best" decision boundary in the feature space. Perceptrons and linear *support vector machines* (SVMs) find hyperplanes[1]. These classifiers are easy to train, often can be effective, and are computationally very efficient (the operational decision is just a single dot-product in the feature space). However, they are ineffective for classes that are not linearly separable, i.e., by a hyperplane. Hence, we will use kernel classifiers to non-linearly project the features into a high-dimensional space, where hyperplanes may be more easily found that serve as good decision boundaries.

Specifically, we will focus on *multiple kernel learning* (MKL) in this chapter. As

[1]See Appendix A for more information regarding SVMs.

its name implies, MKL combines multiple kernels together to form a new kernel, and thus a new classification space. Furthermore, since kernels known to exploit the data's various features can be used as building blocks for MKL, it can do very well with heterogeneous data. There are many works that discuss MKL [8, 9, 10, 11, 12, 13, 14], and nearly all of them rely on operations that aggregate kernels in ways that preserve symmetry and positive semi-definiteness, such as element-wise addition and multiplication. Most MKL algorithms learn a "best" kernel space in which to classify by learning respective weights on each component kernel. Details are contained in Section 2.4.

Two MKL formulations explored in this chapter focus on aggregation using the Choquet *fuzzy integral* (FI) with respect to a *fuzzy measure* (FM) [15]. First, we investigate our previously proposed *fuzzy integral: genetic algorithm* (FIGA) approach to MKL [11, 12], proving that it reduces to a special kind of *linear convex sum* (LCS) kernel aggregation. This leads to the proposition of the *p-norm genetic algorithm* MKL (GAMKL$_p$) approach, which learns an MKL classifier using a genetic algorithm and generalized p-norm weight domain. These algorithms perform a feature-level aggregation of the kernel matrices, producing a new feature representation. We also propose a decision-level MKL called DeFIMKL, which learns a FM with respect to the Choquet FI to fuse decisions from individual kernel classifiers. The FM is learned from training data with a regularized *quadratic program* (QP) approach [16]. We

Table 2.1
Acronyms and Select Notation

SVM	support vector machine
MKL	multiple kernel learning
FM	fuzzy measure
FI	fuzzy integral
FIGA	fuzzy integral: genetic algorithm
LCS	linear convex sum
GAMKL$_p$	p-norm genetic algorithm MKL
DeFIMKL	decision-level fuzzy integral MKL
DeGAMKL$_p$	p-norm decision-level genetic algorithm MKL
DeLSMKL	decision-level least squares MKL
QP	quadratic program
MKLGL	MKL group lasso
MKLGL$_p$	MKLGL with p-norm regularization
RBF	radial basis function
X	feature-vector data, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$
$\mathbf{y}$	data labels, $\mathbf{y} = (y_1, \ldots, y_n)^T$
$f(\mathbf{x})$	prediction function
g	fuzzy measure
$\pi(i)$	sorting index in Choquet integral
$\phi(\mathbf{x})$	non-linear mapping of $\mathbf{x}$
$\kappa(\mathbf{x}_i, \mathbf{x}_j)$	kernel function, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$
K	kernel matrix $K = [K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)]$
$f_k(\mathbf{x})$	decision function using kth kernel, K_k
$f^g(\mathbf{x})$	decision function using Choquet integral, wrt FM g

further explore two additional decision-level methods based on a least-squares formulation. We start with *decision-level least-squares MKL* (DeLSMKL) where we compute the weights for decision values from an ensemble of classifiers using a closed form expression. We then extend this method using a nonlinear cost function and use a genetic algorithm to compute the weights in *decision-level genetic algorithm MKL* (DeGAMKL).

A drawback of MKL methods is the fact that multiple kernel matrices must be stored.

Since the size of these kernels is directly related to the number of feature-vectors in the dataset, large datasets lead to large kernels. Thus, approximations to the kernel matrices that reduce the required number of values to store could allow MKL methods to be used for these large datasets. We explore the use of the Nyström approximation for this task, and show the effects of the approximation on classifier accuracy.

The FI-based MKL approaches are first compared with a leading machine learning MKL method, called MKL *group lasso* (MKLGL)[2] [9] on several benchmark data sets. We also investigate the behavior of regularization on the results of DeFIMKL. In Section 2.2 we briefly review data fusion, and Section 2.3 introduces FMs and FIs, specifically the fuzzy Choquet integral. Section 2.4 details the MKL methods. A review of the preliminary experimental results generated in [17] are presented in Section 2.6, Section 2.7 presents the details and results of our Nyström experiments, and Section 2.8 discusses our final experiment with a large data set. Table 2.1 contains acronyms and selected notation used in this chapter.

2.2 Data Fusion

Data fusion is a broad term for methods that use multiple sets of data, perhaps data from different sensors or the output of multiple processes applied to the same data

[2]See Appendix A for more information regarding MKLGL.

set, to improve some performance metric from a baseline established using only *one* set [18]. It is a very broad area of study, and there exists a vast pool literature relating to it; for a review of data fusion methods see [19] and [20]. Because of the breadth of the topic, we restrict this brief overview to the types of fusion techniques most related to the methods we employ.

Data fusion can be classified in many ways [21, 22, 23]. The taxonomy in [22] is most appropriate to apply to our approach, describing five categories of data fusion. The categories that encompass our fusion methods are termed *feature in—feature out* (FEI-FEO) and *decision in—decision out* (DEI-DEO) and are briefly discussed in the following sections.

2.2.1 Feature In—Feature Out Fusion

FEI-FEO fusion is also known as feature fusion, on which many computer vision methods rely [24, 25, 26, 27]. A popular and powerful method of feature fusion combines the features in a multidimensional feature space using kernel methods [28, 29, 30, 31]. This allows the use of multiple kernels with classification, giving the advantage that particular kernels can exploit certain features better than other kernels. The SVM is a popular classifier for MKL classification, however comparable results have been shown using a logistic regression-based classifier [32].

2.2.2 Decision In—Decision Out Fusion

DEI-DIO fusion is commonly referred to as decision fusion. This approach is very closely related to concept of ensemble learning, where the decisions from multiple classifiers are combined to determine the overall decision. Indeed, this is precisely what the DeFIMKL algorithm discussed in Section 2.4.2 does. Due to the use of multiple classifiers, decision fusion is generally slower than feature fusion, which only requires one classifier [33].

Decision fusion can be done in two general ways: hard or soft. Hard decision fusion is done using the class labels from the ensemble of classifiers. A straightforward method of hard decision fusion is the majority vote approach. Soft decision fusion is performed using other outputs from the classifier ensemble such as the posterior probabilities, evidences, hypotheses, etc. A simple example in this case is to linearly combine the posterior probabilities [34]. Alternatively, for ensembles of fuzzy classifiers, the soft decision fusion approach could be used by aggregating the fuzzy class memberships determined by the classifiers [35].

2.3 Fuzzy Measures and Fuzzy Integrals

FIs and FMs have been proposed for many applications and for many types of data, from simple numeric data to intervals and type-2 fuzzy sets [36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46]. While manual specification of the FM works for small sets of sources (there are already 16 possible combinations of sources in the power set of 4 sources), manually specifying the values of the FM for large collections of sources is virtually impossible. Thus, automatic methods have been proposed, such as the Sugeno λ-measure [39] and the S-decomposable measure [47], which build the measure from the densities (the worth of individual sources), and genetic algorithm [11, 12, 38, 48], Gibbs sampling [49] and other learning methods [16, 50, 51], which build the measure by using training data. Other works [52, 53, 54] have proposed learning FMs that reflect trends in the data and have been specifically applied to crowd-sourcing, where the worth of individuals is not known, but extracted from the data.

2.3.1 Fuzzy measure

A measurable space is the tuple (X, Ω), where X is a set and Ω is a Ω-algebra or set of subsets of X such that

P1. $X \in \Omega$;

P2. For $A \subseteq X$, if $A \in \Omega$, then $A^c \in \Omega$;

P3. If $\forall A_i \in \Omega$, then $\bigcup_{i=1}^{\infty} A_i \in \Omega$.

A FM is a function, $g : \Omega \to [0, 1]$, with the following properties:

P4. (Boundary conditions) $g(\emptyset) = 0$ and $g(X) = 1$;

P5. (Monotonicity) If $A, B \in \Omega$ and $A \subseteq B$, then $g(A) \leq g(B)$.

If Ω is an infinite set, then there is also a third property guaranteeing continuity; in practice and in this chapter, Ω is finite and thus this property is unnecessary. The FM values of the singletons, $g(\{x_i\}) = g^i$ are commonly called the *densities*. Figure 2.1 illustrates the lattice of a FM for the case of $n = 3$.

The arguably most popular FM is the Sugeno λ-measure, which has the attractive property of being able to be defined completely by the values of the densities. The λ-measure has the following additional property. For $A, B \in \Omega$ and $A \cap B = \emptyset$,

$$g_\lambda(A \cup B) = g_\lambda(A) + g_\lambda(B) + \lambda g_\lambda(A) g_\lambda(B), \tag{2.1a}$$

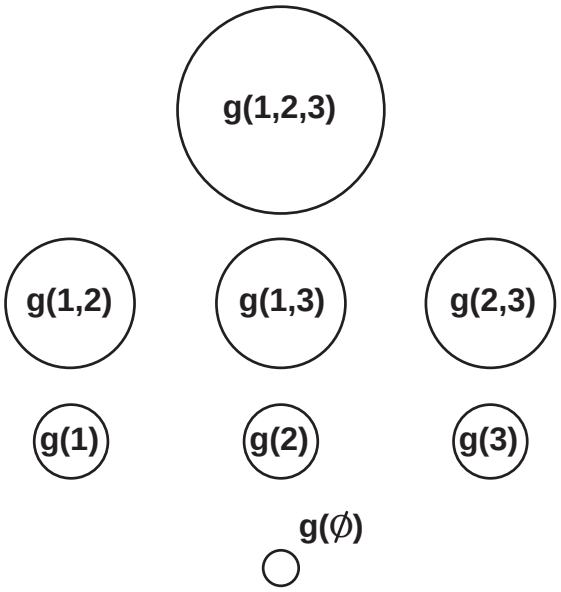


Figure 2.1: Lattice of FM elements for $n = 3$. Monotonicity (P5) is illustrated by the size of each circle, i.e., $g(\{x_1\}) \leq g(\{x_1, x_2\})$ as $\{x_1\} \subset \{x_1, x_2\}$.

where it can be shown that λ can be found by solving [39]

$$\lambda + 1 = \prod_{i=1}^{n} \left(1 + \lambda g^i\right), \quad \lambda > -1. \tag{2.1b}$$

2.3.2 Fuzzy integral

There are many forms of the FI; see [39] for detailed discussion. In general, they are parametric aggregation operators based on the fuzzy measure, hence the selection of measure leads to a specific aggregation operators. In practice, FIs are frequently used for evidence fusion [48, 55, 56, 57, 58]. They combine sources of information

by accounting for both the support of the question (the evidence) and the expected worth of each subset of sources (as supplied by the FM g). Here, we focus on the fuzzy Choquet integral, proposed by Murofushi and Sugeno [59, 60]. Let $h : X \to \mathbb{R}$ be a real-valued function that represents the evidence or support of a particular hypothesis.[3] The discrete (finite Ω) fuzzy Choquet integral is defined as

$$\int_C h \circ g = C_g(h) = \sum_{i=1}^{n} h(x_{\pi(i)}) \left[g(A_i) - g(A_{i-1}) \right], \tag{2.2}$$

where π is a permutation of X, such that $h(x_{\pi(1)}) \geq h(x_{\pi(2)}) \geq \ldots \geq h(x_{\pi(n)})$, $A_i = \{x_{\pi(1)}, \ldots, x_{\pi(i)}\}$, and $g(A_0) = 0$ [15, 42]. Detailed treatments of the properties of FIs can be found in [15, 42, 61]. We now move on to showing how MKL can be achieved using the FM and FI.

2.4 Multiple Kernel Learning

Consider some non-linear mapping function $\phi : \mathbf{x}_i \to \phi(\mathbf{x}_i) \in \mathbb{R}^{D_K}$, where D_K is the dimensionality of the transformed feature vector $\phi(\mathbf{x}_i)$. For brevity, we will denote $\phi(\mathbf{x}_i)$ as ϕ_i. With kernel algorithms, one does not need to explicitly transform $\mathbf{x}_i$, one simply needs to represent the dot product $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. The kernel function κ can take many forms, with the polynomial $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^p$ and

[3] Generally, when dealing with information fusion problems it is convenient to have $h : X \to [0, 1]$, where each source is normalized to the unit-interval.

radial-basis-function (RBF) $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(\sigma||\mathbf{x}_i - \mathbf{x}_j||^2)$ being two of the most well known. Given a set of n feature-vectors X, one can thus construct an $n \times n$ kernel matrix $K = [K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)]_{n \times n}$. This kernel matrix K represents all pairwise dot products of the feature vectors in the transformed high-dimensional space $\mathcal{H}_K$—called the *Reproducing Kernel Hilbert Space* (RKHS).

There are many algorithms that use kernels to transform the input data to an appropriate and useful space; in this chapter, we focus on kernel-based classification, such as the SVM [62, 63]. Multiple kernel algorithms, such as MKLGL [9] and FIGA [11], take single kernel algorithms a step further by representing the feature-vector with multiple kernels and then combining them to produce a single decision output. The kernel combination can be computed in many ways, as long as the combination results in a Mercer kernel [64]. For the feature-level fusion algorithms in this chapter, we will assume that the kernel $\mathcal{K}$ is composed by a weighted combination of pre-computed kernel matrices, i.e.,

$$\mathcal{K} = \sum_{k=1}^{m} \sigma_k K_k, \tag{2.3}$$

where there are m kernels and σ_k is the weight applied to the kth kernel. The domain of σ is very important and many MKL implementations only work for a single domain. For example, $\Delta_2 = \{\sigma \in \mathbb{R}^m : \|\sigma\|_2 = 1, \sigma_k \geq 0\}$ is the ℓ_2-norm MKL [8, 10, 13]. MKLGL [9] uses a generalized MKL instantiation that allows for an ℓ_p-norm domain $\Delta_p = \{\sigma \in \mathbb{R}^m : \|\sigma\|_p = 1, \sigma_k \geq 0\}$, simultaneously learning σ and the parameters of

an SVM on the resultant kernel $\mathcal{K}$. FIGA [11] generalizes (2.3) by representing the computation of $\mathcal{K}$ by the Choquet FI,

$$\mathcal{K} = \sum_{k=1}^{m} \left[g(A_k) - g(A_{k-1}) \right] K_{\pi(k)}, \tag{2.4}$$

where $A_k = \{K_{\pi(1)}, \ldots, K_{\pi(k)}\}$ is a set of kernel matrices sorted by a base-learner quality measure and the FM g is learned by a genetic algorithm (GA); in essence, the entries of $\mathcal{K}$ are each the result of a Choquet FI. In Section 2.4.1 we show that the FIGA algorithm is actually learning an LCS MKL and is equivalent to (2.3) with $\sigma \in \Delta_1$; we will use this new discovery to propose the GAMKL$_p$ algorithm.

2.4.1 The GAMKL$_p$ algorithm

The FIGA algorithm produces an MKL classifier by learning one on the composite kernel $\mathcal{K}$ with the Choquet FI as shown in (2.4). The final classification function is learned on the kernel $\mathcal{K}$, and, in past works [11, 12, 17], we have used the SVM for this final learner. The basic steps of FIGA are as follows:

1. Compute kernel matrices $K_k = [\kappa_k(\mathbf{x}_i, \mathbf{x}_j)]^{n \times n}$, $k = 1, \ldots, m$;

2. Train a base-learner (e.g., SVM) on each kernel K_k and record the classification accuracy η_k, $k = 1, \ldots, m$;

3. Collect sorting indices π, such that $\eta_{\pi(1)} \geq \eta_{\pi(2)} \geq \ldots \geq \eta_{\pi(k)}$;

4. Use a GA to learn the FM g, such that the classification accuracy of a learner (e.g., SVM) on $\mathcal{K}$ at (2.4) is maximized.

The fitness of each chromosome in step 4) of FIGA is the classification accuracy of the learner on $\mathcal{K}$, while the genes are $(m-1)$ distinct values of the FM.[4] Because FIGA only learns the sorting π once, in step 2), the GA only needs to learn $(m-1)$ FM values, $g(\{K_{\pi(1)}\})$, $g(\{K_{\pi(1)}, K_{\pi(2)}\})$, ..., $g(\{K_{\pi(1)}, \ldots, K_{\pi(m-1)}\})$; by property P4, $g(A_0) = 0$ and $g(A_m) = 1$. This leads to Proposition 1

Proposition 1. Since the sorting order π is only found once in step 2) of FIGA, the Choquet integral at (2.4) can be rewritten as

$$\mathcal{K} = \sum_{k=1}^{m} \sigma_{\pi(k)} K_{\pi(k)}, \tag{2.5}$$

where $\sigma_{\pi(k)} = g(A_k) - g(A_{k-1})$.

Proof. Because the sorting is not updated, the sets A_k also remain unchanged; hence, the summation weight on $K_{\pi(k)}$ is the subtraction of the FM values of the same sets (no matter their values). Hence, we can attach a single weight $\sigma_{\pi(k)}$ to each $K_{\pi(k)}$. □

[4] In [12], an additional gene was added to indicate different types of FMs and a slightly better performance was noted.

Remark 1. Proposition 1 shows that the FIGA kernel composition at (2.4) is independent of the initial sorting by π because the summation at (2.5) can be performed in any arbitrary order and give the same result. Hence, step 3) of FIGA is unnecessary.

Proposition 2. In FIGA, the domain of $\sigma_{\pi(k)}$ is Δ_1.

Proof. The ℓ_1 norm of σ is

$$\sum_{k=1}^{m} \sigma_{\pi(k)} = \sum_{k=1}^{m} g(A_k) - g(A_{k-1}) = g(A_m) - g(A_0) = 1. \tag{2.6}$$

Furthermore, due to the monotonicity property (P5) of g, $\sigma_{\pi(k)} = g(A_k) - g(A_{k-1}) \geq 0$. □

Remark 2. Proposition 2 shows the domain of σ upon which FIGA learns. Taking Propositions 1 and 2 together shows that FIGA is equivalent to using a GA to learn the weights $\sigma \in \Delta_1$ in the kernel combination at (2.3).

In light of this discovery, we propose a GAMKL$_p$ algorithm that uses a GA to learn the weights $\sigma \in \Delta_p$ of (2.3). When $p = 1$, we have shown that this is equivalent to FIGA. However, because of our discoveries in Propositions 1 and 2, we can simplify and generalize FIGA to allow for learning σ in the generalized domain Δ_p. The genes of the GAMKL$_p$ algorithm are the values of the m weights of (2.3), i.e., $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$.

To ensure the GAMKL$_p$ genes lie in the ℓ_p-norm domain Δ_p, all candidate genes $\tilde{\sigma}$ are ℓ_p-norm normalized to form σ as

$$\sigma = \frac{\tilde{\sigma}}{\sqrt[p]{\sum_{i=1}^{m} |\tilde{\sigma}_i|^p}}. \tag{2.7}$$

The fitness of each chromosome in GAMKL$_p$ is the 5-fold cross-validation classification accuracy of the learning algorithm—in this chapter, an SVM—trained on each chromosome's aggregated $\mathcal{K}$.

Remark 3. While Propositions 1 and 2 show that FIGA is equivalent to GAMKL$_1$, the GAMKL$_p$ algorithm has the additional benefit that the genes of each chromosome are not constrained to be monotonically increasing (as in FIGA). Hence, GAMKL$_p$ is algorithmically more simple.

In Section 2.6, we will further investigate the performance of GAMKL$_p$ for real-world classification tasks and in comparison with other MKL classification methods. Now we turn to proposing a decision-level MKL fusion method using the fuzzy integral.

Table 2.2
UCI Benchmark Data Sets

	Data Set		
	Sonar	**Dermatology**	**Wine**
No. of Objects	208	366	178
No. of Features	60	33	13
Binary Classes	{1} vs. {2}	{1–3} vs. {4–6}	{1} vs. {2,3}
	Ionosphere	**Ecoli**	**Glass**
No. of Objects	351	336	214
No. of Features	34	7	9
Binary Classes	{0} vs. {1}	{1–4} vs. {5–8}	{1–3} vs. {4–6}

2.4.2 The DeFIMKL algorithm

Let $f_k(\mathbf{x}_i)$ be the decision-value on feature-vector $\mathbf{x}_i$ produced by the kth classifier in an ensemble. The overall decision of the ensemble is computed by the Choquet integral, where the evidence h is the set of decisions by the classifier ensemble and g encodes the relative worth of each classifier in the ensemble. So, mathematically, the ensemble decision $f^g(\mathbf{x}_i)$ on feature-vector $\mathbf{x}_i$ with respect to the FM g is produced by

$$f^g(\mathbf{x}_i) = \sum_{k=1}^{m} f_{\pi(k)}(\mathbf{x}_i) \left[g(A_k) - g(A_{k-1}) \right], \tag{2.8}$$

where $A_k = \{f_{\pi(1)}(\mathbf{x}_i), \ldots, f_{\pi(k)}(\mathbf{x}_i)\}$, such that $f_{\pi(1)}(\mathbf{x}_i) \geq f_{\pi(2)}(\mathbf{x}_i) \geq \ldots \geq f_{\pi(m)}(\mathbf{x}_i)$. This is a generalized classifier fusion method that has been explored in many previous works [45, 57, 58, 65].

In [16], we proposed a method to learn the FM g from training data with a regularized

sum-of-squared error (SSE) optimization, which we now briefly describe. Let the SSE be defined as

$$E^2 = \sum_{i=1}^{n} \left(f^g(\mathbf{x}_i) - y_i\right)^2, \tag{2.9}$$

where y_i is the class label for $\mathbf{x}_i$. It can be shown that (2.8), as a Choquet integral, can be reformulated as

$$f^g(\mathbf{x}_i) = \sum_{k=1}^{m} \left[f_{\pi(k)}(\mathbf{x}_i) - f_{\pi(k+1)}(\mathbf{x}_i)\right] g(A_k), \tag{2.10}$$

where $f_{\pi(m+1)} = 0$ [15]. The SSE can thus be expanded as

$$E^2 = \sum_{i=1}^{n} \left(H_{\mathbf{x}_i}^T \mathbf{u} - y_i\right)^2, \tag{2.11a}$$

where $\mathbf{u}$ is the lexicographically ordered FM g, i.e., $\mathbf{u} = (g(\{x_1\}), g(\{x_2\}), \ldots, g(\{x_1 \cup x_2\}), g(\{x_1 \cup x_3\}), \ldots, g(\{x_1 \cup x_2 \cup \ldots \cup x_m\})$, and

$$H_{\mathbf{x}_i} = \begin{pmatrix} \vdots \\ f_{\pi(1)}(\mathbf{x}_i) - f_{\pi(2)}(\mathbf{x}_i) \\ \vdots \\ 0 \\ \vdots \\ f_{\pi(m)}(\mathbf{x}_i) - 0 \end{pmatrix}, \tag{2.11b}$$

where $H_{\mathbf{x}_i}$ is of size $(2^m - 1) \times 1$ and contains all the difference terms $f_{\pi(k)}(\mathbf{x}_i) - f_{\pi(k+1)}(\mathbf{x}_i)$ at the corresponding locations of A_k in $\mathbf{u}$. We can now fold out the squared term in (2.11a), producing

$$\begin{aligned} E^2 &= \sum_{i=1}^{n} \left(\mathbf{u}^T H_{\mathbf{x}_i} H_{\mathbf{x}_i}^T \mathbf{u} - 2 y_i H_{\mathbf{x}_i}^T \mathbf{u} + y_i^2 \right) \\ &= \mathbf{u}^T D \mathbf{u} + \mathbf{f}^T \mathbf{u} + \sum_{i=1}^{n} y_i^2, \\ D &= \sum_{i=1}^{n} H_{\mathbf{x}_i} H_{\mathbf{x}_i}^T, \quad \mathbf{f} = -\sum_{i=1}^{n} 2 y_i H_{\mathbf{x}_i}. \end{aligned} \tag{2.12}$$

Note that (2.12) is a quadratic function; hence, we can add in the constraints on $\mathbf{u}$, such that it represents a FM, producing a constrained QP. We can write the monotonicity constraint on $\mathbf{u}$, according to properties P4 and P5, as $C\mathbf{u} \leq 0$, where

$$C = \begin{pmatrix} \\ \Psi_1^T \\ \\ \Psi_2^T \\ \\ \vdots \\ \\ \Psi_{n+1}^T \\ \\ \vdots \\ \\ \Psi_{m(2^{m-1}-1)}^T \end{pmatrix} \tag{2.13}$$

and Ψ_1^T is a vector representation of the monotonicity constraint, $g(\{x_1\}) - g(\{x_1 \cup x_2\}) \leq 0$. Hence, C is simply a matrix of $\{0, 1, -1\}$ values of size $(m(2^{m-1} - 1)) \times (2^m - 1)$. See [16] for more details about the form of C. Thus, the full QP to learn

the FM $\mathbf{u}$ is

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T\hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u}, \quad C\mathbf{u} \leq \mathbf{0}, \quad (\mathbf{0}, 1)^T \leq \mathbf{u} \leq \mathbf{1}, \tag{2.14}$$

where $\hat{D} = 2D$. We will also test the performance of ℓ_2 and ℓ_1 regularization on the optimization at (2.14), i.e.,

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T\hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u} + \lambda\|\mathbf{u}\|_p, \tag{2.15}$$

where $p = 1$ for ℓ_1 regularization and $p = 2$ for ℓ_2. Again, see [16] for more discussion on this topic. The QPs at (2.14) and (2.15) provide a method to learn the FM $\mathbf{u}$ (i.e., g) from training data. We now propose a method for using this learning method for ensemble learning with kernel SVMs.

We propose that each learner $f_k(\mathbf{x}_i)$ is a kernel classifier, each trained on a separate kernel K_k; here, we will use the SVM. The SVM classifier decision value is

$$\eta_k(\mathbf{x}) = \sum_{i=1}^{n} \alpha_{ik} y_i \kappa_k(\mathbf{x}_i, \mathbf{x}) - b_k, \tag{2.16}$$

which is the distance of $\mathbf{x}$ from the hyperplane defined by the learned SVM model parameters, α_{ik} and b_k [62, 63]. Typically, the class label is then computed as $\text{sgn}\{\eta_k(\mathbf{x})\}$. One could use $f_k(\mathbf{x}) = \text{sgn}\{\eta_k(\mathbf{x})\}$ as the training input to the FM learning at (2.12), but this eliminates information about which kernel produces the largest class separation—essentially, the difference between $\eta_k(\mathbf{x})$ for classes labeled

$y = +1$ and $y = -1$. Hence, we remap $\eta_k(\mathbf{x})$ onto the interval $[-1, +1]$, creating the inputs for learning by the sigmoid function,

$$f_k(\mathbf{x}) = \frac{\eta_k(\mathbf{x})}{\sqrt{1 + \eta_k^2(\mathbf{x})}}. \tag{2.17}$$

Thus, the training data for DeFIMKL are $(\{K_k = [\kappa_k(\mathbf{x}_i, \mathbf{x}_j)], \mathbf{f}_k(X)\}, \mathbf{y})$, $k = 1, \ldots, m$, where K_k are the kernel matrices for each kernel function κ_k, $\mathbf{f}_k(X) = (f_k(\mathbf{x}_1), \ldots, f_k(\mathbf{x}_n))^T$ are the remapped SVM decision values, and $\mathbf{y} = (y_1, \ldots, y_n)$ are the ground-truth labels of $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, respectively. The output of the QP learner is the FM g. A new feature vector $\mathbf{x}$—from a test data set—can be classified by the trained algorithm with the following procedure:

1. Compute the SVM decision values $f_k(\mathbf{x})$ by using (2.16) and (2.17);

2. Apply the Choquet integral at (2.8) with respect to the learned FM g;

3. Compute the class label by $\text{sgn}\{f^g(\mathbf{x})\}$.

In previous work [17], the MKL algorithms discussed here have been applied to the benchmark data sets shown in Table 2.2. The results are reviewed in the Section 2.6.

2.4.3 The DeLSMKL Algorithm

Similar to the DeFIMKL algorithm, the following algorithms find a weighted combination of decision-values from an ensemble of classifiers to compute an overall decision. Again, each learner is a kernel SVM classifier. Thus, the SVM classifier decision value $\eta_k(\mathbf{x})$ in (2.16) is normalized by a remapping onto the interval $[-1, +1]$ using the sigmoid function as in (2.17) to create the inputs for learning.

Consider the linear aggregation of the decisions from an ensemble of classifiers. The overall decision in this case is

$$f(\mathbf{x}_i) = \operatorname{sgn}\left\{\sum_{k=1}^{m} \sigma'_k f_k(\mathbf{x}_i)\right\}, \tag{2.18}$$

where $f_k(\mathbf{x}_i)$ denotes the normalized decision-value on feature-vector $\mathbf{x}_i$ by the kth classifier in an ensemble, and we wish to compute the weights σ'_k such that $f_k(\mathbf{x}_i)$ minimizes a particular error function as discussed later in this section. Now, given a training set of N objects and an ensemble of M classifiers, we can form a vector of class labels as $\mathbf{y} \in \mathbb{R}^N$. We can also form a matrix $\mathbf{F} \in \mathbb{R}^{M \times N}$ defined as

$$\mathbf{F}_{ij} = f_i(\mathbf{x}_j), \tag{2.19}$$

where $f_i(\mathbf{x}_j)$ is the decision value on the j^{th} feature vector in the training data by the

i^{th} classifier in the ensemble. Using this notation, the overall decision in (2.18) can be rewritten in vector form as

$$\mathbf{f} = \operatorname{sgn}\left\{ \left(\boldsymbol{\sigma}'\right)^T \mathbf{F} \right\}, \tag{2.20}$$

where $\boldsymbol{\sigma}' \in \mathbb{R}^M$ is the vector of weights we wish to learn and $\mathbf{f} \in \mathbb{R}^N$ contains the overall ensemble decisions for each member of the training data.

Let us temporarily ignore the nonlinear signum operator in (2.20) for the remainder of this section. In this case, the overall decision formulation, i.e., the argument of the signum function in (2.18) and (2.20), has precisely the same form as that of linear regression problems. Thus, the same methods used to solve regression problems can also lend themselves to this situation. One standard method computes the weight vector $\boldsymbol{\sigma}'$ in (2.20) that minimizes the squared error between the true class labels and the predicted class labels, or

$$E_{LS} = \left\| \mathbf{y} - \left(\boldsymbol{\sigma}'\right)^T \mathbf{F} \right\|^2. \tag{2.21}$$

This formulation is the well-known method of linear least squares and has the closed form solution

$$\boldsymbol{\sigma}' = \mathbf{y}^T \mathbf{F}^\dagger, \tag{2.22}$$

where $\mathbf{F}^{\dagger}$ denotes the Moore-Penrose pseudoinverse of $\mathbf{F}$. Thus we can readily compute $\boldsymbol{\sigma}'$ using an ensemble of classifiers on training data and applying (2.22). We term this approach *decision-level least squares MKL* (DeLSMKL).

Note that this method forces the values of $\boldsymbol{\sigma}'^{T}\mathbf{F}$ to lie as close as possible to the true class labels $\mathbf{y}$; however, their proximity to the true class labels is not important. The only requirement is that the signs of $\boldsymbol{\sigma}'^{T}\mathbf{F}$ and $\mathbf{y}$ match, thus the cost function in (2.21) is too restrictive on the values of $\boldsymbol{\sigma}'^{T}\mathbf{F}$. We address this in the next section where we include the nonlinear signum operator from (2.20) in the cost function.

2.4.4 The DeGAMKL$_p$ Algorithm

By introducing the signum function, the overconstrained cost function in the DeLSMKL algorithm given in (2.23) can be relaxed to be

$$E = \left\| \mathbf{y} - \operatorname{sgn}\{(\boldsymbol{\sigma}')^{T}\,\mathbf{F}\} \right\|^{2}. \tag{2.23}$$

While this could potentially improve the results, there is no longer a closed form solution as in the case of DeLSMKL. We employ a genetic algorithm to find a weight vector $\boldsymbol{\sigma}'$ that minimizes this nonlinear cost. The DeGAMKL$_p$ algorithm parallels the GAMKL$_p$ algorithm closely, including the normalization in (2.7). However, instead of using the 5-fold cross-validation classification accuracy as the fitness of each

chromosome as in GAMKL$_p$, we simply use the classification accuracy for the entire training set; empirically, DeGAMKL$_p$ works best using this fitness.

While all of the described fusion methods have their merit, they all share the detriment that one must store m $n \times n$ kernel matrices. We address this in the next section, producing an efficient way to perform MKL in its various forms.

2.5 The Nyström Approximation for Gram Matrices

2.5.1 Background

The Nyström method has its roots in numerical solutions of integral equations, and it was first explicitly shown that a Gram matrix $K \in \mathbb{R}^{n \times n}$ can be approximated by

$$\tilde{K} = K_z K_{zz}^{\dagger} K_z^T, \tag{2.24}$$

by means of eigendecomposition approximation [66]. In this notation, z corresponds to the indices of $|z|$ sampled columns of K; hence K_z is the $n \times |z|$ rectangular matrix composed of the sampled columns of K, and K_{zz} is the $|z| \times |z|$ matrix composed

of the sampled row and columns of K. Note that K_{zz} is part of K_z, and $K_{zz}^{\dagger}$ is the Moore-Penrose pseudoinverse of K_{zz}.

Algorithms for approximating Gram matrices via PCA methods are generally $\mathcal{O}(n^3)$; however, the complexity of the Nystrom approximation is $\mathcal{O}(m^2 n)$, where $m < n$. Note that a sparse greedy matrix approximation that has an identical form of the Nyström approximation was introduced prior to [66] in [67], but it is computationally more expensive and is not derived using the Nyström approximation explicitly.

2.5.2 Error Bounds and Other Development

Since the proposal of the Nyström approximation, there has been much work regarding efficient computation, explicit bound derivation, and column sampling methods. The work presented in [68] generalizes the work in [66] and develops an algorithm to compute the matrix approximation in $\mathcal{O}(n)$ time. Furthermore, it is shown that by using the Nyström approximation to approximate the Gram matrix K, the error is bounded with high probability by

$$\|K - \tilde{K}_k\|_\xi \leq \|K - K_k\|_\xi + \epsilon \sum_{i=1}^{n} K_{ii}^2, \tag{2.25}$$

where K_k is the best rank-k approximate to K. Note that this holds in both the ℓ_2-norm sense ($\xi = 2$) and Frobenius norm sense ($\xi = F$). The work presented in

[69] expresses this bound in terms of the spectral norm and further tightens the error bound from $O(N/\sqrt{m})$ to $O(N/m^{1-\rho})$, where $\rho \in (0, 1/2)$ is a constant characterizing the eigengap; ρ is small for large eigengaps.

Zhang and Kwok explicitly showed how the Nyström approximation error depends on the choice of columns [70]. They note that the columns of the kernel matrix are essentially quantized by sampling a subset, and the error bound is directly related to the quantization error due to column sampling. Thus to minimize the error, it is necessary to choose the columns (also called *landmarks*) that essentially contain the most information, i.e., are most able to accurately represent unsampled columns. Their algorithm chooses the landmarks as the cluster centers after using k-means clustering on the columns of the kernel matrix, and while the results show improvements over the sampling methods in [68], the algorithm is computationally inefficient for large-scale problems since the entire kernel matrix must be available to find the cluster centers (landmarks).

Kumar et al. proposed an ensemble Nyström method [71], where multiple Nyström approximations to the kernel matrix are computed and linearly combined to form the ensemble approximation

$$\tilde{K}_{ens} = \sum_{r=1}^{p} \mu_r \tilde{K}_r. \tag{2.26}$$

Here, μ_r are the mixture weights that can be defined in many ways, and $\tilde{K}_r$ is the rth Nyström approximation of the kernel matrix K. Experimental results show that this

approach generally improves the results from a single Nyström approximation, but the improvements come at the cost of p times more computation. Kumar et al. also presented a thorough treatment of sampling methods for the Nyström approximation [72].

A recursive algorithm for computing the Nyström approximation was proposed, which used a greedy approach to column selection [73]. This method showed experimental performance improvements over many other methods and achieved similar performance to the k-means clustering method [70].

2.5.3 Efficient MKL using the Nyström Approximation

MKL can be difficult or impossible to apply to large datasets since multiple kernels must be stored to learn the kernel weights. Applying the Nyström approximation to MKL can significantly reduce this storage requirement, thus allowing MKL to be used on datasets that have been too large to utilize MKL in the past.

Applying the Nyström approximation to MKL starts by first replacing the kernels K_k in (2.3) with their column-sampled versions $(K_k)_z$, resulting in a rectangular matrix representing the column-sampled version of $\mathcal{K}$, or

$$\mathcal{K}_z = \sum_{k=1}^{m} \sigma_k (K_k)_z. \tag{2.27}$$

Note that since $\mathcal{K}_{zz} \in \mathcal{K}_z$, the Nyström approximation of the MKL kernel can be computed via

$$\tilde{\mathcal{K}} = \mathcal{K}_z \mathcal{K}_{zz}^{\dagger} \mathcal{K}_z^T, \tag{2.28}$$

and $\tilde{\mathcal{K}}$ can then be used in the kernel classifier. Also note that the full MKL with m kernel matrices requires mn^2 values to be stored, but by applying the Nyström approximation by sampling c columns, where $c << n$, the required number of values to store drops by a factor of c/n to mnc.

The resulting kernel $\tilde{\mathcal{K}}$ is positive semi-definite, thus it can be linearized as $\tilde{\mathcal{K}} = \tilde{\mathcal{X}}\tilde{\mathcal{X}}^T$, given the appropriate $\tilde{\mathcal{X}}$. The resulting linearized model $\tilde{\mathcal{X}}$ can then be used in a linear classifier to achieve equivalent results to the kernelized verison. We compute $\tilde{\mathcal{X}}$ via eigendecomposition of the $c \times c$ matrix $\mathcal{K}_{zz}$. Let the columns of U_z and the diagonal of Λ_z represent the eigenvectors and eigenvalues of $\mathcal{K}_{zz}$, respectively. $\mathcal{K}_{zz}^{\dagger}$ can then be decomposed as

$$\mathcal{K}_{zz}^{\dagger} = U_z \Lambda_z^{-1} U_z^T. \tag{2.29}$$

It follows that (2.28) can be rewritten as

$$\begin{aligned} \tilde{\mathcal{K}} &= \mathcal{K}_z U_z \Lambda_z^{-1} U_z^T \mathcal{K}_z^T \\ &= \mathcal{K}_z U_z \Lambda^{-1/2} \Lambda^{-1/2} U_z^T \mathcal{K}_z^T \\ &= \left(\mathcal{K}_z U_z \Lambda_z^{-1/2}\right) \left(\mathcal{K}_z U_z \Lambda_z^{-1/2}\right)^T, \end{aligned} \tag{2.30}$$

and the linearized model $\tilde{\mathcal{X}}$ becomes

$$\tilde{\mathcal{X}} = \mathcal{K}_z U_z \Lambda_z^{-1/2}, \tag{2.31}$$

which can then be used in linear classifiers and may run more quickly than their kernelized counterparts. Note that while the eigendecomposition is a computationally expensive procedure, it must only be performed on the relatively small $c \times c$ matrix ($\mathcal{O}(c^3)$). This linearization approach for MKL follows that for the single kernel approach known as low-rand linearized SVM proposed in [74] and packaged in [75].

2.6 Preliminary Experiments and Results

Here we review the results of the GAMKL$_p$ and DeFIMKL algorithms after applying them to benchmark data sets using SVM classifiers; we use LIBSVM to implement the classifiers [76]. Additionally, we present the results of the DeGAMKL$_p$ and DeLSMKL. The performance of these algorithms is compared to that of the MKLGL algorithm discussed in Section 2.4.

Table 2.3
RBF Kernel Parameter Ranges

Data Set					
Sonar	**Dermatology**	**Wine**	**Ionosphere**	**Ecoli**	**Glass**
[-2.2, -0.5]	[-2.3, 0.2]	[-2.5, 2]	[-2.1, 1.2]	[-3, 3]	[-2, 2]

2.6.1 Datasets and Algorithm Parameters

The benchmark UCI data sets [77] shown in Table 2.2 are used to evaluate the algorithms. Note that in some cases multiple classes are joined together such that the classification decision is binary. Each experiment consists of 100 trials so the results can be statistically analyzed using a two-sample t-test. In each trial, a random draw of 80% of the data is used for training and the remaining 20% is sequestered for testing. Ten RBF kernels are used in each algorithm with respective RBF width σ linearly spaced on the interval defined in Table 2.3; the same RBF parameters are used for each algorithm.

The genetic algorithms in GAMKL$_p$ and DeGAMKL$_p$ have a population of 31 chromosomes, where each chromosome is the set of ℓ_p-norm normalized weights vectors. The GA runs for 25 generations using roulette wheel selection and elitism, where the fittest individual is kept from each generation. One-point crossover with a rate of 60% and a mutation rate of 5% are used, where mutation is simply a random perturbation of the chromosome. In GAMKL$_p$, fitness is the result of 5-fold cross validation of the kernel-SVM accuracy using the kernel weights comprising each individual, where

cross validation is used to suppress the effects of over training. DeGAMKL$_p$ uses the kernel-SVM accuracy of the entire training set, as mentioned in Section 2.4.4. Two experiments are performed with each ℓ_p-norm GAMKL$_p$: one with the initial population generated randomly and another where the initial population is also seeded with the result of the MKLGL algorithm.

The only parameter in the DeFIMKL algorithm is the regularization coefficient λ. Once λ is defined, the QPs at (2.14) and (2.15) are solved via an interior-point solver to obtain the FM. The results for regularization using the DeFIMKL algorithm are generated using 10 different nonzero λs as well as the case where $\lambda = 0$, corresponding to no regularization of (2.14).

There are no parameters that vary in the DeLSMKL algorithm. Once an ensemble of classifiers is created and applied to the training data, the weight vector $\boldsymbol{\sigma}'$ is calculated using (2.22).

2.6.2 Results

The classification accuracies of the GAMKL$_p$, DeFIMKL, DeGAMKL$_p$, DeLSMKL, and MKLGL algorithms are shown in Table 2.4 along with the standard deviations over the 100 trials. The best algorithm(s) for each data set are shown in bold font; a two-sample t-test at a 5% significance level is used to determine the statistically

best algorithm(s)—hence, more than one algorithm can be considered as best. The t-tests compare each algorithm's distribution of accuracies against the accuracies of every other algorithm. We see that at least two versions of the GAMKL$_p$ algorithm have superior performance on each data set, even outperforming the well established MKLGL algorithm on the Sonar data set. In the other data sets, the performances of GAMKL$_p$ and MKLGL fall very close to each other and their differences are statistically insignificant.

The DeFIMKL results show that it is not as promising as the GAMKL$_p$ algorithm, and regularization generally dampens performance. However, at least one version of the DeFIMKL algorithm still appears in the group of superior results for half of the data sets. Figures 2.2 and 2.3 show the results of the DeFIMKL algorithm applied to the Sonar and Dermatology data sets, respectively. The error bars indicate plus/minus one standard deviation over the 200 runs. Figure 2.2 and Table 2.4 show that the ℓ_1-norm normalized kernel weights found using DeFIMKL had the best performance when $\lambda = 2$ for the Sonar data set; however, the performance of DeFIMKL is inversely proportional to the value of λ for the Dermatology data set. Thus, the best result obtained from the regularized DeFIMKL algorithm occurs when the kernel weights are ℓ_2-norm normalized with $\lambda = 0.5$. It is worthwhile to mention that even in the cases where DeFIMKL algorithm did not achieve superior results, it was only beaten by approximately 3%.

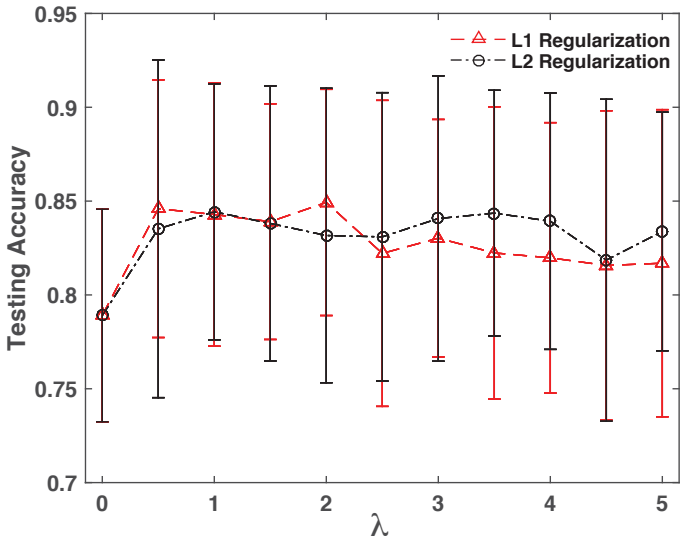


Figure 2.2: DeFIMKL performance using regularization on Sonar data. Error bars indicate ± one standard deviation.

Overall, the DeGAMKL$_p$ and DeLSMKL algorithms were not able to perform as well as MKLGL, GAMKL$_p$, or DeFIMKL. We see that the results of DeLSMKL are strongly dependent on the dataset, due to the over-constrained cost formulation discussed in Section 2.4.3. Relaxing the cost function via the DeGAMKL$_p$ algorithm generally improves classification accuracy compared to DeLSMKL, as well as increases the stability of the classifier across data sets.

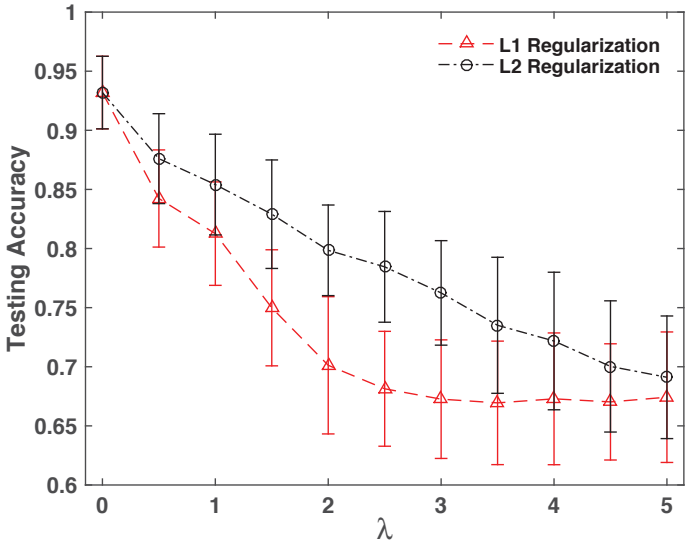


Figure 2.3: DeFIMKL performance using regularization on Dermatology data—classes $\{1, 2, 3\}$ versus $\{4, 5, 6\}$. Error bars indicate $\pm$ one standard deviation.

2.7 Experiments with the Nyström Approximation

The following experiments are used to observe the merit of applying the Nyström approximation to some of the previously discussed algorithms. The DeFIMKL and GAMKL$_p$ algorithms are modified to utilize the Nyström approximation by applying (2.27) and (2.28) as discussed in Section 2.5.3, where the indices in z are randomly selected; the algorithms also use the LIBLINEAR SVM implementation [78]. The algorithms are evaluated using the data sets shown in Table 2.5 from the KEEL [79] and UCI [77] datasets, in addition to the data sets in Table 2.2. The algorithm parameters discussed in Section 2.6.1 are used again in this experiment; however, the Nyström

approximation is applied to the kernel matrices. The experiments are repeated using Nyström sampling quantities ranging from 1% to 100% of the size of the data set, allowing the effect of the Nyström sampling quantity on the classification accuracy and run-time to be clearly visualized. Furthermore, the sampling and classification is performed 100 times; thus, all results are based on the average of the 100 trials.

Each MKL algorithm uses 10 RBF kernels of varying widths (i.e., σ in the RBF kernel definition): the first kernel width is always $\sigma_1 = \frac{1}{n_f}$, where n_f is the number of features in the data set. The remainder of the kernels are chosen such that they are linearly spaced in the interval $[n_f/10, 10n_f]$.

2.7.1 Results

Figures 2.4 through 2.7 show the typical trend of classification accuracy versus the Nyström sampling quantity for the GAMKL and DeFIMKL algorithms. The figures depict the classification accuracy of the full algorithms (algorithms not using the Nyström approximation) as dashed lines, the trend of the classification accuracy as the Nyström sampling percentage is varied as solid lines, and the points at which the Nyström-based algorithms' performances drop to 5% of the performance of the full algorithms as circles.

Of these examples, the Wine dataset clearly achieves the best performance both in

terms of accuracy and robustness to the Nyström sampling percentage. The plots show that by applying the Nyström approximation, we are able to use less than 10% of the kernel matrix to achieve results essentially equivalent to the full algorithm results, which requires the entire kernel matrix. The trend for the algorithms applied to the Ionosphere dataset are very similar.

Figures 2.4 through 2.7 also show an example that highlights a case where the Nyström approximation has a stronger effect on the classification accuracy. The plots given for the Sonar dataset show that the performance decreases much more dramatically with respect to the Nyström sampling quantity. This is typical of a high-dimensional dataset in which a large proportion consists of points that are far apart from each other in the kernel space, increasing the rank of the kernel matrix. In this case, a larger number of data points are required to accurately approximate the others and thus the matrix approximation suffers as the number of sampled points are limited.

Table 2.6 summarizes the results of applying the Nyström approximation to GAMKL$_p$ and DeFIMKL. The values in the table represent the percentage of the full data set required by the Nyström approximation to achieve results within 5% of the classification accuracy acheived using the full data set, i.e., full GAMKL or DeFIMKL; these points correspond to the circled points in Figures 2.4 through 2.7.

Note how similar the performance degradation (with respect to the Nyström sampling percentage) of the different MKL approaches applied to the various datasets

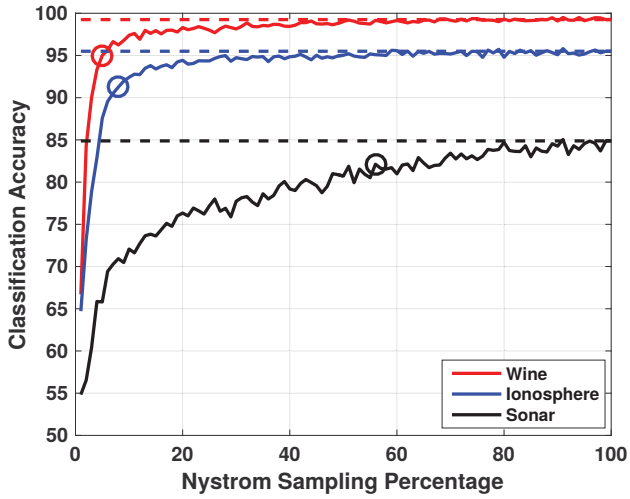


Figure 2.4: Results of using GAMKL_1 on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%.

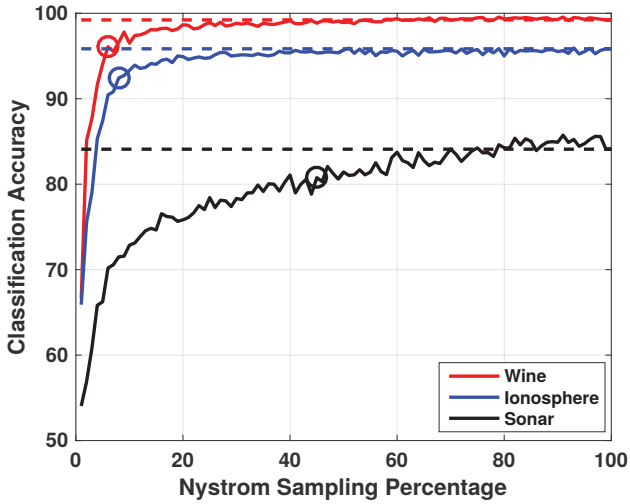


Figure 2.5: Results of using GAMKL_2 on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%.

is. Table 2.6 shows that we can regularly sample less than 10% of the kernel yet incur negligible performance degradation. This general invariance to datasets or MKL

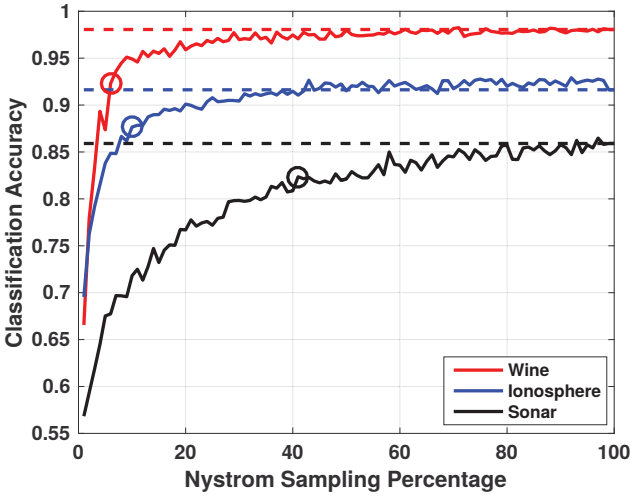


Figure 2.6: Results of using DeFIMKL$_1$ on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%.

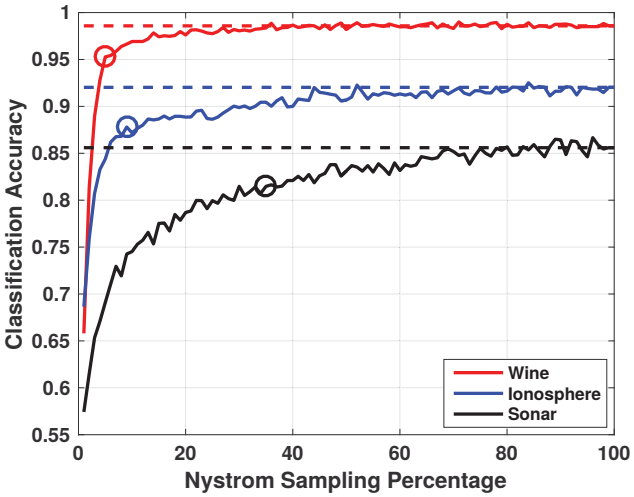


Figure 2.7: Results of using DeFIMKL$_2$ on the Wine, Ionosphere, and Sonar datasets with the Nyström approximation. Dashed line indicates full sample performance; circle indicates sample percentage at which performance drops 5%.

methods suggests that the performance degradation is mostly, if not all, due to the Nyström approximation error, thus this approximation technique can be applied to

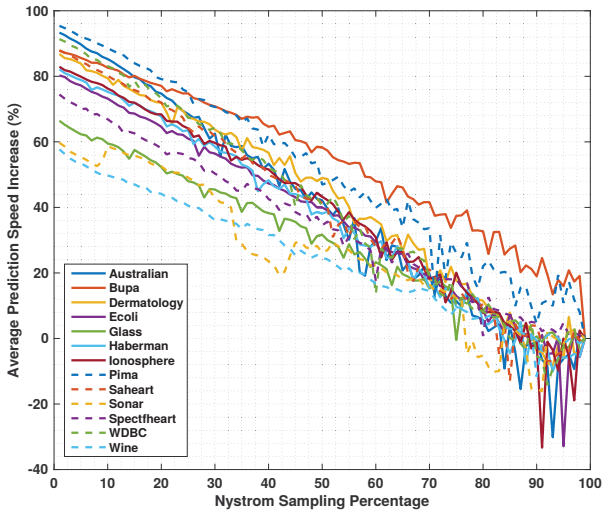


Figure 2.8: Average speed-up percentages of classifiers using the Nyström approximation.

other MKL approaches to yield similar results. Furthermore, the application of the Nyström approximation makes the MKL approaches more memory-efficient, making the application of MKL approaches to large datasets possible.

The prediction time of each experiment was recorded to show how the Nyström approximation can also speed up classifiers. Figure 2.8 shows the average increase in speed of the classifiers on the datasets. Notice that if we only use 20% of the data, which Table 2.6 clearly indicates we can routinely do without sacrificing classifier performance, the prediction speed is increased by 45—80%, depending on the dataset. And if we choose only 5% of the data, again as Table 2.6 shows we can do with most datasets, the prediction speed can be increased up to 92%. Therefore, in addition to making MKL methods more memory-efficient, the Nyström approximation also makes them faster and even more applicable to large datasets.

2.8 Application of the Nyström Approximation to a Large Data Set

The DeFIMKL algorithm, modified to use the Nyström approximation, is applied to the MNIST database [80]. This database contains 60,000 training images and 10,000 testing images, each 28×28 pixels in dimension. There are nine total classes in this data set, corresponding to the handwritten digits 0–9, though we split the data into binary classes as odd vs. even. The pixel values of each image are used as the features, thus each data point has a feature dimension of 784. The DeFIMKL algorithm uses 10 kernels as discussed in Section 2.7, and the Nyström sampling quantity is set to 1% such that the classifier can be implemented on a desktop machine with less than 8GB of available memory without parallel computing or memory-efficient mapping techniques. Experiments were performed 20 times, and results are presented as the average of the 20 trials.

2.8.1 Results

Figure 2.9 shows the results of applying the Nyström-based DeFIMKL algorithm to the MNIST data. The trend is similar to DeFIMKL's performance on the Sonar dataset shown in Figure 2.2 in that both ℓ_1- and ℓ_2-regularized versions perform

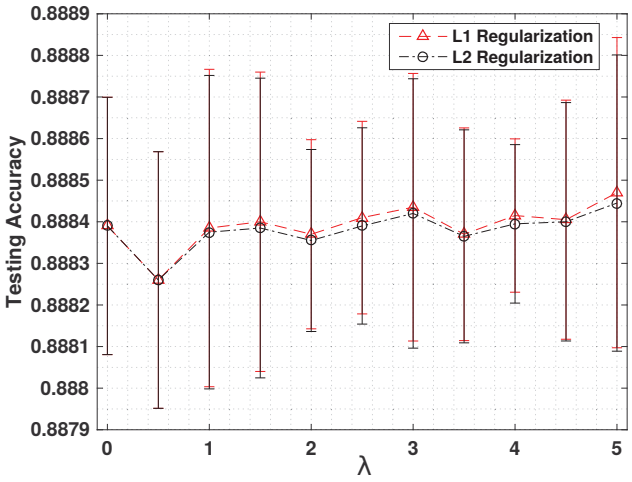


Figure 2.9: DeFIMKL performance using regularization on MNIST data. Error bars indicate ± one standard deviation.

very similarly across the range of λ, however in this case the performance differences across the range of λ are statistically insignificant. Figure 2.9 also shows that for choices of $\lambda \geq 1$, DeFIMKL$_1$ tends to outperform DeFIMKL$_2$, though inspection of the vertical axis resolution proves that the difference is minute.

Since regularization is employed to reduce the effects of overtraining, it is no surprise that the performance of DeFIMKL on the MNIST data is not closely tied with the choice of the regularization parameter λ. Since we are sampling only 1% of the data for use in the Nyström approximation, we are inherently losing information contained in the training data, and thus already "protected" from the perils of overtraining.

2.9 Conclusion

This chapter presents a feature-level fusion algorithm—GAMKL$_p$—that we show is equivalent to a fuzzy integral-based MKL approach known as FIGA. However, unlike FIGA, GAMKL$_p$ is generalized such that σ can lie in the ℓ_p -norm domain Δ_p . We also presented a decision-level fusion algorithm—DeFIMKL—that aggregates kernels through the use of the Choquet fuzzy integral with respect to a fuzzy measure learned by a regularized quadratic programming approach. Additionally, we propose two other decision-level fusion algorithms, DeGAMKL$_p$ and DeLSMKL, which rely on a genetic algorithm and a least-squares formulation, respectively. Our results show that the GAMKL$_p$ algorithm achieves equivalent, and sometimes better, classification accuracy than the state-of-the-art MKLGL algorithm. The DeFIMKL algorithm, while generally not as successful as GAMKL$_p$, is still able to match or beat MKLGL in half of the experiments. The DeGAMKL$_p$ and DeLSMKL methods were outperformed in every experiment and were not able to achieve the performance of the DeFIMKL algorithm. This highlights the merit of using nonlinear aggregation for decision-level fusion, as done by DeFIMKL.

Interpreting the results at a higher level reveals that feature-level fusion algorithms are generally more powerful than decision-level fusion algorithms in terms of classification

accuracy. Regardless of the data set used in each experiment, at least one feature-level fusion algorithm emerged as a leading performer. The same cannot be said of the decision-level fusion algorithms, though it is clear that DeFIMKL is often the best in its class. In general, we recommend the GAMKL algorithm for situations where processing time is not a concern since it suffers from the disadvantages associated with genetic algorithms; the MKLGL algorithm is faster than GAMKL, however GAMKL has been shown to converge to better results. If the situation warrants decision-level fusion, DeFIMKL's results suggest that it is the algorithm of choice.

Further experiments using the GAMKL$_p$ and MKLGL algorithms show that MKL methods can be made much more efficient via the Nyström approximation with negligible impact on classifier performance. This allows MKL methods to be applied to very large datasets where the size of the full kernel matrices is too large to store, which we explored in an experiment with the large MNIST handwritten digit data set.

In future work we will apply the methods discussed in this chapter to larger datasets. Furthermore, we are working on a feature-level method for aggregating kernels with a non-linear fuzzy integral. The main goal is to preserve the ability of the fuzzy integral to produce non-linear aggregations of the individual kernels, while ensuring that the result is a Mercer kernel. In order to achieve this, one must develop a way of sorting the kernel matrix terms in the Choquet integral (and not just once with the

base-learner training data accuracy, as does FIGA) and still aggregate with a Mercer kernel preserving operation.

Table 2.4
Classification Accuracy Results on Benchmark Data Sets*

	Data Set		
Algorithm	Sonar	Derm	Wine
$MKLGL_1$	83.0 (5.81)	**97.3** (1.99)	**99.6** (0.97)
$MKLGL_2$	84.6 (5.11)	**97.2** (1.60)	**99.6** (1.02)
$GAMKL_1$	84.0 (6.00)	**97.1** (1.70)	99.4 (1.16)
$GAMKL_1^+$	84.6 (5.67)	**97.3** (1.75)	**99.6** (1.00)
$GAMKL_2$	**86.0** (5.64)	**97.1** (1.55)	**99.5** (1.10)
$GAMKL_2^+$	**86.4** (5.62)	**96.8** (1.84)	99.4 (1.16)
DeFIMKL	78.9 (5.66)	93.2 (3.07)	99.4 (1.17)
$DeFIMKL_1$	**84.9** (6.03) $\lambda = 2$	84.2 (4.12) $\lambda = 0.5$	**99.5** (1.10) $\lambda = 0.5$
$DeFIMKL_2$	84.4 (6.82) $\lambda = 1$	87.6 (3.80) $\lambda = 0.5$	**99.7** (0.87) $\lambda = 1.5$
DeGAMKL	82.5 (5.59)	95.6 (3.01)	92.4 (2.03)
$DeGAMKL_1$	83.2 (5.56)	94.9 (3.63)	93.0 (7.91)
$DeGAMKL_2$	82.5 (5.79)	95.5 (3.09)	92.6 (8.68)
DeLSMKL	69.9 (8.86)	87.4 (3.81)	96.9 (2.73)
Algorithm	Ionosphere	Ecoli	Glass
$MKLGL_1$	**95.2** (2.36)	**97.1** (1.71)	**94.5** (3.29)
$MKLGL_2$	**95.5** (2.40)	**97.2** (1.80)	**94.0** (3.53)
$GAMKL_1$	94.8 (2.59)	**97.1** (1.93)	**94.0** (3.87)
$GAMKL_1^+$	94.8 (2.53)	96.9 (1.86)	93.3 (3.99)
$GAMKL_2$	**95.1** (2.29)	**97.5** (1.60)	**94.0** (3.24)
$GAMKL_2^+$	**95.7** (2.39)	**97.4** (1.68)	**94.2** (3.49)
DeFIMKL	92.3 (7.13)	**97.3** (1.77)	91.2 (3.78)
$DeFIMKL_1$	88.8 (3.26) $\lambda = 4$	91.8 (3.00) $\lambda = 3$	78.1 (6.20) $\lambda = 0.5$
$DeFIMKL_2$	90.0 (3.35) $\lambda = 0.5$	91.9 (3.26) $\lambda = 2.5$	83.1 (4.83) $\lambda = 0.5$
DeGAMKL	87.4 (10.20)	91.7 (3.20)	85.7 (5.80)
$DeGAMKL_1$	84.9 (11.31)	91.7 (3.15)	84.3 (7.42)
$DeGAMKL_2$	90.3 (7.37)	91.6 (2.85)	84.6 (7.52)
DeLSMKL	65.9 (5.22)	91.2 (2.89)	84.2 (6.36)

*Bold indicates best result according to a two-valued t-test at a 5% significance level.

+The initial seed of the genetic algorithm is the weight vector found with MKLGL.

Table 2.5
Additional Data Sets for Nyström Verification

	Data Set			
	SPECTF Heart	**Haberman**	**Pima**	**WDBC**
No. Objects	267	306	768	569
No. Features	44	3	8	30
Binary Classes	{0} vs. {1}	{+} vs. {-}	{+} vs. {-}	{M} vs. {B}
	Australian	**Bupa**	**SA Heart**	
No. Objects	690	345	462	
No. Features	14	6	9	
Binary Classes	{0} vs. {1}	{1} vs. {2}	{0} vs. {1}	

Table 2.6
Nyström Sampling Percentage Required to Achieve Equivalent Classification Results as Full Sample

	Algorithm			
Data Set	$GAMKL_1$	$GAMKL_2$	$DeFIMKL_1$	$DeFIMKL_2$
Sonar	56%	45%	41%	35%
Dermatology	10%	9%	7%	6%
Wine	5%	6%	7%	5%
Ionosphere	8%	8%	10%	11%
Ecoli	4%	5%	1%	2%
Glass	12%	19%	8%	5%
SPECTF Heart	1%	1%	5%	1%
Haberman	1%	1%	1%	1%
Pima	3%	3%	3%	1%
Australian	2%	2%	3%	2%
Bupa	8%	10%	12%	17%
SA Heart	3%	1%	1%	1%
WDBC	2%	2%	3%	3%

Chapter 3

Visualization and Learning of the Choquet Integral With Limited Training Data

The material in this chapter is submitted for publication in IEEE Transactions on Fuzzy Systems, April 2017.

3.1 Introduction

In many fields, we are often faced with the task of making decisions based on a set of feature-vector data $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \subset \mathbb{R}^d$. This data is typically accompanied by a set of training labels for each feature-vector, giving the pair $(\mathbf{y}, X)$, where $\mathbf{y} = (y_1, y_2, ..., y_n)^T$ is a vector of labels such that y_i is the label of feature-vector $\mathbf{x}_i$. This problem can be considered a classification task, and is typically tackled by training a classifier such that it can accurately predict the class label of a new sample of data where the label is not known. More concretely, the data $(\mathbf{y}, X)$ are used to learn some prediction function f such that we can accurately predict the label of feature vectors as $y = f(\mathbf{x})$.

Linear classifiers are typically nothing more than a hyperplane in the feature-space representing the decision boundary, and training these classifiers involves finding the hyperplane's parameters in some optimal way. A very popular hyperplane classifier is the *support vector machine* (SVM) because it is easy to train and computationally efficient. The drawback to linear SVMs (and other linear classifiers), however, is that they require the data to be linearly separable—a distribution very rarely encountered with real data. One way around this is to instead use their kernel-based variants where the data are non-linearly projected to a high-dimensional space where a suitable hyperplane is more likely to be found. While this appears to solve the problem of

non-separable data, it has its own baggage: what kernel function should be used?

Multiple kernel learning (MKL) typically answers this question by learning a new kernel through the combination of predetermined kernels while maintaining symmetry and positive-semidefiniteness, an approach discussed in many works [2, 8, 9, 10, 11, 12, 17]. These approaches fall under the roof of feature-level fusion in that they combine different "looks" at the data (each represented by an individual kernel) and use a single classifier to determine the predicted class label. Another MKL technique uses multiple kernel-based classifiers, each utilizing a different kernel. The outputs of these classifiers is then combined at the decision-level using some aggregation function. This approach to decision-level fusion is the premise for the *decision-level fuzzy integral multiple kernel learning* (DeFIMKL) classifier discussed in Section 3.3, where aggregation is performed via the Choquet *fuzzy integral* (FI) with respect to a *fuzzy measure* (FM). Once again though we have a roadblock: how do we specify the FM?

The task we investigate in this work is learning a FM. Many previous works [81, 82, 83] have shown that an underlying FM can be learned from training data, though here we show that only a subset of the FM is accurately learned from the training data and the remaining FM terms simply follow the constraints from the learning process. In other words, only a subset of the FM is learned in a data-driven manner. Thus when asked to classify a new sample of data using the Choquet FI, we risk utilizing terms from the FM that were not learned accurately from the training data, leading to an

erroneous prediction. In this work, we propose a method to more accurately learn the FM terms that are not data-driven. The method assumes that some knowledge of the underlying FM structure is known and thus can be encoded in the learning process as discussed in Section 3.4.

The remainder of this chapter is organized as follows. Section 3.2 discusses fuzzy measures and the Choquet fuzzy integral; it also introduces our strategy of simultaneously visualizing the FM and behavior of the Choquet integral. Section 3.3 reviews learning a fuzzy measure through minimizing the *sum-of-squared error* (SSE) via *quadratic programming* (QP)—the backbone of the DeFIMKL algorithm—as well as its behavior with insufficient training data. Section 3.4 proposes an extension to the DeFIMKL algorithm, allowing knowledge of the underlying FM to be encoded into the QP, and Section 3.6 summarizes experiments with real-world and contrived datasets. Finally, Section 3.7 concludes the chapter and discusses our future work.

3.2 Fuzzy Measures and Fuzzy Integrals

FIs and FMs are used for many applications and for many types of data, from simple numeric data to intervals and type-2 fuzzy sets [36, 37, 42, 44, 46]. While manual specification of the FM works for small sets of sources, manually specifying the values of

the FM for large collections of sources is virtually impossible. Thus, automatic methods have been proposed, such as the Sugeno λ-measure [39] and the S-decomposable measure [47], which build the measure from the densities[1], and genetic algorithm [12, 48], Gibbs sampling [49] and other learning methods, which build the measure by using training data. Other works [52, 53, 54] have proposed learning FMs that reflect trends in the data and have been specifically applied to crowd-sourcing, where the worth of individuals is not known, and is thus extracted from the data.

3.2.1 Fuzzy measures

A measurable space is the tuple (X, Ω), where X is a set and Ω is an Ω-algebra or set of subsets of X such that

P1. $X \in \Omega$;

P2. For $A \subseteq X$, if $A \in \Omega$, then $A^c \in \Omega$;

P3. If $\forall A_i \in \Omega$, then $\bigcup_{i=1}^{\infty} A_i \in \Omega$.

A FM is a set-valued function, $g : \Omega \to [0, 1]$, with the following properties:

P4. (Boundary conditions) $g(\emptyset) = 0$ and $g(X) = 1$;

[1]The FM values of the singletons, $g(\{x_i\}) = g^i$ are commonly called the *densities*.

P5. (Monotonicity) If $A, B \in \Omega$ and $A \subseteq B$, $g(A) \leq g(B)$.

If Ω is an infinite set, then there is also a third property to guarantee continuity; however, in practice and in this chapter, Ω is finite and thus this property is unnecessary. While fuzzy measures provide a way for quantifying the worth of combinations of sources, fuzzy integrals can be used to aggregate the information from these sources.

3.2.2 Fuzzy integrals

There are many forms of the FI; see [39] for detailed discussion. In practice, FIs are frequently used for evidence fusion [48, 55, 56, 57]. They combine sources of information by accounting for both the support of the question (the evidence) and the expected worth of each subset of sources (as supplied by the FM g). Here, we focus on the fuzzy Choquet integral, proposed by Murofushi and Sugeno [59, 60]. Let $h : X \to \mathbb{R}$ be a real-valued function that represents the evidence or support of a particular hypothesis.[2] The discrete (finite Ω) fuzzy Choquet integral is defined as

$$\int_C h \circ g = C_g(h) = \sum_{i=1}^{n} h(x_{\pi(i)}) \left[g(A_i) - g(A_{i-1}) \right], \tag{3.1}$$

[2]Generally, when dealing with information fusion problems it is convenient to have $h : X \to [0, 1]$, where each source is normalized to the unit-interval.

where π is a permutation of X, such that $h(x_{\pi(1)}) \geq h(x_{\pi(2)}) \geq \ldots \geq h(x_{\pi(n)})$, $A_i = \{x_{\pi(1)}, \ldots, x_{\pi(i)}\}$, and $g(A_0) = 0$ [15, 42]. Detailed treatments of the properties of FIs can be found in [15, 42, 61].

3.2.3 Common Aggregations via the Choquet Integral

It is well known that the Choquet integral is a powerful aggregation operator parametrized by a FM, and thus can represent many aggregation functions [58]. For example, the Choquet integral acts as the maximum operator when the FM is all 1s (except $g\{\emptyset\} = 0$, due to boundary constraints), the minimum operator when the FM is all 0s (except $g\{X\} = 1$, due to boundary constraints), and the mean operator when $g(A_i) = |A_i|/n$, $\forall A_i \subset X$.

3.2.4 Visualizing the Fuzzy Integral

The FM lattice (Hasse diagram) is a convenient method to visualize a FM; Figure 3.1 illustrates the lattice of a FM for the case of $n = 3$. Note that the size of the individual nodes in the lattice indicates their relative magnitude, and monotonicity is apparent since nodes at higher levels in the lattice are larger—or at least as large—than those below.

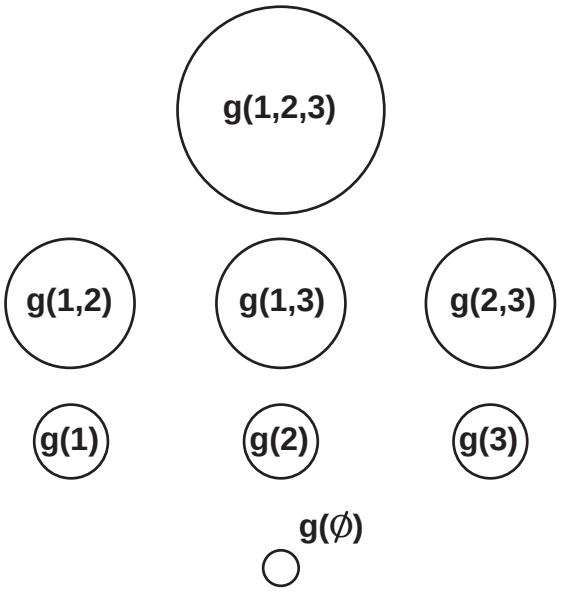


Figure 3.1: Lattice of FM elements for $n = 3$. Monotonicity (P5) is illustrated by the size of each node, i.e., $g(\{x_1\}) \leq g(\{x_1, x_2\})$ as $\{x_1\} \subset \{x_1, x_2\}$. Note that shorthand notation is used where $g(1, 3)$ is equivalent to $g(\{x_1, x_3\})$.

The FM lattice alone, while useful for showing a FM, does not give insight into how the Choquet integral at (3.1) utilizes the lattice due to the π-permutation. Therefore, for a particular input we also show the path through the lattice followed by the Choquet integral. For example, suppose that a particular data sample x and hypothesis h gives rise to the permutation $\pi = \{2, 1, 3\}$. Then, for an arbitrary FM, the lattice visualization includes the path shown in Figure 3.2. This visualization strategy allows us to summarize the FM as well as the Choquet integral's paths.

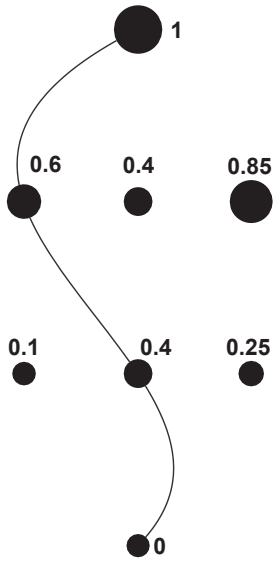


Figure 3.2: The path taken by the Choquet integral due to a single input inducing the permutation $\pi = \{2, 1, 3\}$. Note that the FM was arbitrarily defined in this example, and their distribution (ordering) follows that of Figure 3.1.

Figure 3.3: Lattice of learned FM and paths for random training data from the Ionosphere data set using $m = 10$. Note there are numerous untouched nodes and their learned values are driven by the constraints in (3.9).

3.3 The DeFIMKL Algorithm

The DeFIMKL algorithm was introduced in [17] as a method of decision-level fusion in the context of classification, where a set of decisions from an ensemble of classifiers are non-linearly fused via the Choquet FI. To mathematically describe the algorithm, let the decision-value for feature-vector $\mathbf{x}_i$ from the kth classifier in an ensemble be $f_k(\mathbf{x}_i)$; the set of decisions from the ensemble comprise the evidence h for the Choquet integral. The evidence is then integrated with respect to the FM g, which encodes the relative worth of each classifier in the ensemble. This results in the ensemble decision $f_g(\mathbf{x}_i)$ for feature-vector $\mathbf{x}_i$ with respect to the FM g,

$$f_g(\mathbf{x}_i) = \sum_{k=1}^{m} f_{\pi(k)}(\mathbf{x}_i) \left[g(A_k) - g(A_{k-1}) \right], \tag{3.2}$$

where $A_k = \{f_{\pi(1)}(\mathbf{x}_i), \ldots, f_{\pi(k)}(\mathbf{x}_i)\}$, such that $f_{\pi(1)}(\mathbf{x}_i) \geq f_{\pi(2)}(\mathbf{x}_i) \geq \ldots \geq f_{\pi(m)}(\mathbf{x}_i)$. This method has been explored in many previous works as a generalized classifier fusion method [45, 57, 58, 65].

The FM completely specifies the behavior of the Choquet integral. Thus, the next step in understanding the DeFIMKL algorithm is assigning a FM for the Choquet integral in (3.2), of which there are many methods. For example, the Sugeno λ-measure [39] may be naively used after specifying the FM values of the singletons; however, there

is no guarantee that this choice of FM will yield acceptable results when used with (3.2) since it does not take training data into account. To address this, we suggested a data-driven method to learn the FM g through regularized *sum-of-squared error* (SSE) optimization in [16]. This method is summarized next.

Let the SSE be defined as

$$E^2 = \sum_{i=1}^{n} \left(f_g(\mathbf{x}_i) - y_i \right)^2 . \tag{3.3}$$

It can be shown that (3.2), as a Choquet integral, can be reformulated as

$$f_g(\mathbf{x}_i) = \sum_{k=1}^{m} \left[f_{\pi(k)}(\mathbf{x}_i) - f_{\pi(k+1)}(\mathbf{x}_i) \right] g(A_k), \tag{3.4}$$

where $f_{\pi(m+1)} = 0$ [15]. We can then expand the SSE as

$$E^2 = \sum_{i=1}^{n} \left(H_{\mathbf{x}_i}^T \mathbf{u} - y_i \right)^2 , \tag{3.5a}$$

where $\mathbf{u}$ is the lexicographically ordered FM g, i.e., $\mathbf{u} =$

$(g(\{x_1\}), g(\{x_2\}), \ldots, g(\{x_1, x_2\}), g(\{x_1, x_3\}), \ldots, g(\{x_1, x_2, \ldots, x_m\}))$, and

$$H_{\mathbf{x}_i} = \begin{pmatrix} \vdots \\ f_{\pi(1)}(\mathbf{x}_i) - f_{\pi(2)}(\mathbf{x}_i) \\ \vdots \\ 0 \\ \vdots \\ f_{\pi(m)}(\mathbf{x}_i) - 0 \end{pmatrix}, \tag{3.5b}$$

where $H_{\mathbf{x}_i}$ is of size $(2^m - 1) \times 1$ and contains all the difference terms $f_{\pi(k)}(\mathbf{x}_i) - f_{\pi(k+1)}(\mathbf{x}_i)$ at the corresponding locations of A_k in $\mathbf{u}$. Finally, folding out the squared term in (3.5a) produces

$$\begin{aligned} E^2 &= \sum_{i=1}^{n} \left(\mathbf{u}^T H_{\mathbf{x}_i} H_{\mathbf{x}_i}^T \mathbf{u} - 2 y_i H_{\mathbf{x}_i}^T \mathbf{u} + y_i^2 \right) \\ &= \mathbf{u}^T D \mathbf{u} + \mathbf{f}^T \mathbf{u} + \sum_{i=1}^{n} y_i^2, \\ D &= \sum_{i=1}^{n} H_{\mathbf{x}_i} H_{\mathbf{x}_i}^T, \quad \mathbf{f} = -\sum_{i=1}^{n} 2 y_i H_{\mathbf{x}_i}. \end{aligned} \tag{3.6}$$

Since (3.6) is a quadratic function, we can add constraints on $\mathbf{u}$ such that it represents a FM, leading to a constrained QP. We can write the boundary and monotonicity

constraints on $\mathbf{u}$ (see properties P4 and P5) as $C\mathbf{u} \leq 0$, where

$$C = \begin{pmatrix} \\ \Psi_1^T \\ \\ \Psi_2^T \\ \\ \vdots \\ \\ \Psi_{n+1}^T \\ \\ \vdots \\ \\ \Psi_{m(2^{m-1}-1)}^T \\ \\ \end{pmatrix} \tag{3.7}$$

and Ψ_1^T is a vector representation of the monotonicity constraint, $g\{x_1\} - g\{x_1, x_2\} \leq 0$. Hence, C is simply a matrix of $\{0, 1, -1\}$ values of size $(m(2^{m-1}-1)) \times (2^m - 1)$ with the form

$$C = \begin{bmatrix} 1 & 0 & \cdots & -1 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & 0 & -1 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & -1 \end{bmatrix}. \tag{3.8}$$

Thus, the full QP to learn the FM $\mathbf{u}$ is

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T \hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u}, \quad C\mathbf{u} \leq \mathbf{0}, \quad (\mathbf{0}, 1)^T \leq \mathbf{u} \leq \mathbf{1}, \tag{3.9}$$

where $\hat{D} = 2D$. Note that an additional regularization term can be included in the

QP as

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T\hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u} + \lambda v_*(\mathbf{u}), \tag{3.10}$$

where λ is the regularization weight and $v_*(\cdot)$ is some regularization function. For example, ℓ_p-norm regularization is applied when $v_*(\mathbf{u}) = \|\mathbf{u}\|_p$. ℓ_1 and ℓ_2 regularization of this QP are discussed in [16, 17].

The QPs at (3.9) and (3.10) provide a method to learn the FM $\mathbf{u}$ (i.e., g) from training data, thus completing the requirements for calculating the Choquet integral at (3.2). We now review how to use a kernel classifier to determine the decision-value $f_k(\mathbf{x}_i)$. Specifically, we will show how to use the SVM with this algorithm.

Suppose that each learner $f_k(\mathbf{x}_i)$ is a kernel SVM, each trained on a separate kernel K_k. The SVM classifier decision value is

$$\eta_k(\mathbf{x}) = \sum_{i=1}^{n} \alpha_{ik} y_i \kappa_k(\mathbf{x}_i, \mathbf{x}) - b_k, \tag{3.11}$$

which is interpreted as the distance of $\mathbf{x}$ from the hyperplane defined by the learned SVM model parameters, α_{ik} and b_k [62, 63]. The class label is typically computed as $\text{sgn}\{\eta_k(\mathbf{x})\}$,[3] which could be used as the training input to the FM learning at (3.6), however, we remap $\eta_k(\mathbf{x})$ onto the interval $[-1, +1]$ via the sigmoid function to create

[3]Note that the sgn$(\cdot)$ function discards information about how well the kernel separates the classes of data.

inputs for learning as

$$f_k(\mathbf{x}) = \frac{\eta_k(\mathbf{x})}{\sqrt{1+\eta_k^2(\mathbf{x})}}. \tag{3.12}$$

Thus, the training data for DeFIMKL are $(\{K_k = [\kappa_k(\mathbf{x}_i, \mathbf{x}_j)], \mathbf{f}_k(X)\}, \mathbf{y})$, $k = 1, \ldots, m$, where K_k are the kernel matrices for each kernel function κ_k, $\mathbf{f}_k(X) = (f_k(\mathbf{x}_1), \ldots, f_k(\mathbf{x}_n))^T$ are the remapped SVM decision values, and $\mathbf{y} = (y_1, \ldots, y_n)$ are the ground-truth labels of $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, respectively; the output of the QP learner is the FM g. Algorithm 1 summarizes the training process. After training, a new feature vector $\mathbf{x}$—from a test data set—can be classified by via the procedure summarized in Algorithm 2.

Algorithm 1: DeFIMKL Classifier Training

Data: $(\mathbf{x}_i, y_i)$ - feature vector and label pairs; K_k - kernel matrices
Result: $\mathbf{u}$ - Lexicographically ordered fuzzy measure vector
for *each kernel matrix* **do**
    Compute the kernel SVM classifier decision values, η_k, as in (3.11).
    Remap the decision values onto the interval $[-1, +1]$ as f_k using (3.12).
Solve the minimization problem in (3.9) for the FM $\mathbf{u}$.

Algorithm 2: DeFIMKL Classifier Prediction

Data: $\mathbf{x}$ - feature vector; K_k - kernel matrices; $\mathbf{u}$ - learned fuzzy measure vector
Result: y - Predicted class label
Compute the SVM decision values $f_k(\mathbf{x})$ by using (3.11) and (3.12).
Apply the Choquet integral at (3.2) with respect to the learned FM $\mathbf{u}$.
Compute the class label as $y = \text{sgn}\{f_g(\mathbf{x})\}$.

3.3.1 FM Learning Behavior with Insufficient Training Data

Learning the entire FM for a DeFIMKL classifier utilizing m classifiers requires at least 2^m (or $2^m - 2$, observing the boundary conditions in property P4) *rank-independent* observations. Therefore, since so many rank-independent observations are rarely encountered in training data sets, there will likely be values of the FM that are not data-driven. Figure 3.3 shows an example of this in the wild, where the Ionosphere dataset [77] was used to train DeFIMKL with 10 classifiers. Note that there are many nodes in the lattice that are never "touched" by the training data; the learned values for these nodes is completely driven by the monotonicity constraints in the QP, the choice of regularization used, and the initialization used in the QP solver. It is therefore highly unlikely that the learned values at these nodes accurately represent the underlying FM, and if Algorithm 2 is applied to a new data point that utilizes one or more of the untouched nodes, prediction accuracy will suffer. The following contrived example demonstrates the behavior of the ℓ_2-regularized DeFIMKL algorithm with insufficient training data.

Example 1. *Learning an Underdetermined FM via ℓ_2-regularized DeFIMKL.* A three-SVM ℓ_2-regularized DeFIMKL algorithm (i.e., $m = 3$, however these results are also indicative of the behavior when $m > 3$) is trained with a synthetic dataset that purposefully avoids two nodes in the fuzzy lattice and was generated using the

underlying FM shown in Table 3.1; the underlying FM was arbitrarily assigned. The FM learned by the DeFIMKL algorithm with $\lambda = 1$ is also shown in Table 3.1, labeled as "ℓ_2-min". Note that two nodes in the lattice, corresponding to $g(\{x_2\})$ and $g(\{x_1, x_2\})$ were not driven by the training data, and thus are essentially driven by the monotonicity constraints.

What we see is that all nodes touched by the training data (i.e., nodes traversed by the Choquet integral) are learned successfully with minimal error (well within 5%). However, the two nodes untouched by the training data are assigned values based on monotonicity constraints. The node corresponding to $g(\{x_2\})$ gets a value of essentially 0, satisfying the monotonicity constraint that $g(\{\emptyset\}) \leq g(\{x_2\}) \leq min(g(\{x_1, x_2\}), g(\{x_2, x_3\}))$, and the node corresponding to $g(\{x_1, x_2\})$ gets a value of 0.14 to satisfy the constraint $max(g(\{x_1\}), g(\{x_2\})) \leq g(\{x_1, x_2\}) \leq g(\{X\})$. Note that in both of these cases, the learned FM value is essentially the minimum value permitted by the monotonicity constraints. This, as will be shown in the following section, is due to the ℓ_2-regularization of the DeFIMKL algorithm.

3.4 FM Learning with a Specified Goal

The standard DeFIMKL algorithm discussed in the previous section assumes that the structure of the underlying FM is not known, thus no information regarding the

Table 3.1
Underlying and learned FMs (excluding $g(\{\emptyset\})$ and $g(X)$ whose values are 0 and 1, respectively, due to the boundary conditions). The learned FM terms marked with an asterisk are not addressed by the training data. Regularization labels indicate the type of norm employed and the aggregation goal. For example, "ℓ_1-min" indicates a goal of that for minimum aggregation ($\mathbf{g} = \mathbf{0}$) and ℓ_1-regularization ($\|\mathbf{u} - \mathbf{g}\|_1$).

		Goal Regularization		
FM Term	**Underlying**	ℓ_2**-min**†	ℓ_2**-max**	ℓ_2**-mean**
$g(\{x_1\})$	0.14	0.14	0.19	0.15
$g(\{x_2\})^*$	0.29	0.00017	0.93	0.33
$g(\{x_3\})$	0.43	0.43	0.44	0.44
$g(\{x_1, x_2\})^*$	0.57	0.14	1	0.67
$g(\{x_1, x_3\})$	0.71	0.69	0.71	0.71
$g(\{x_2, x_3\})$	0.86	0.83	0.93	0.87
FM Term	**Underlying**	ℓ_1**-min**‡	ℓ_1**-max**	ℓ_1**-mean**
$g(\{x_1\})$	0.14	0.12	0.14	0.15
$g(\{x_2\})^*$	0.29	8.7e-9	0.86	0.33
$g(\{x_3\})$	0.43	0.43	0.43	0.43
$g(\{x_1, x_2\})^*$	0.57	0.12	1	0.67
$g(\{x_1, x_3\})$	0.71	0.71	0.71	0.7
$g(\{x_2, x_3\})$	0.86	0.85	0.86	0.85
FM Term	**Underlying**	ℓ_2**-LOS**	ℓ_1**-LOS**	
$g(\{x_1\})$	0.14	0.15	0.14	
$g(\{x_2\})^*$	0.29	0.29	0.22	
$g(\{x_3\})$	0.43	0.43	0.43	
$g(\{x_1, x_2\})^*$	0.57	0.78	0.69	
$g(\{x_1, x_3\})$	0.71	0.71	0.71	
$g(\{x_2, x_3\})$	0.86	0.86	0.86	

† Equivalent to ℓ_2-regularization on the FM vector directly, i.e., $v_*(\mathbf{u}) = \lambda\|\mathbf{u}\|_2$.
† Equivalent to ℓ_1-regularization on the FM vector directly, i.e., $v_*(\mathbf{u}) = \lambda\|\mathbf{u}\|_1$.

underlying FM is encoded in the QP. If, however, the FM is partially known, the QP at (3.10) should include that information. To this end, we propose the regularization function

$$v_*(\mathbf{u}) = \lambda\|\mathbf{u} - \mathbf{g}\|_p, \tag{3.13}$$

where $\mathbf{g}$ represents a goal of what we expect the underlying FM to look like and p defines the norm type. The following sections describe the solution to the regularized problem with ℓ_2- and ℓ_1-regularization.

3.4.1 ℓ_2- Goal Regularization

Including the regularization function from (3.13) in the QP with $p = 2$ gives[4]

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T \hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u} + \lambda\|\mathbf{u} - \mathbf{g}\|_2^2, \tag{3.14}$$

and the QP then also simultaneously minimizes the Euclidean distance between the learned FM $\mathbf{u}$ and the goal $\mathbf{g}$. Expanding the regularization term in (3.14) leads to

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T \left(\hat{D} + \lambda I\right)\mathbf{u} + (\mathbf{f} - 2\lambda\mathbf{g})^T\mathbf{u}, \tag{3.15}$$

showing that the inclusion of this regularization function still results in a valid QP, though this comes as no surprise since the regularization function in (3.13) is quadratic in $\mathbf{u}$; the minimization problem at (3.15) can be solved by any QP solver.

[4]Note that we square the regularization term in this case for mathematical convenience; the problem remains convex.

3.4.2 ℓ_1- Goal Regularization

The regularization function in (3.13) with $p = 1$ forces $\mathbf{u}$ to lie close to the goal $\mathbf{g}$ in the ℓ_1-sense. Including this regularization function in the QP gives

$$\min_{\mathbf{u}} 0.5\mathbf{u}^T\hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u} + \lambda\|\mathbf{u} - \mathbf{g}\|_1, \tag{3.16}$$

however, this formulation cannot simply be reduced or combined in the objective function any further as done in the previous case of ℓ_2-regularization; the difference within the norm will not, in general, be non-negative. To address this we move the regularization term to the constraints through the use of Tibshirani's iterative lasso algorithm [84]; see Appendix B for a brief description of the method, Algorithm 3 describes the process in terms of this problem.

This algorithm solves the unregularized problem while iteratively updating its constraints to enforce sparsity in the difference $(\mathbf{u} - \mathbf{g})$, although the problem in (3.16) must be reformulated as follows. Let us first lump $\mathbf{u}$ and $\mathbf{g}$ into a single long vector $\mathbf{w}$ as $\mathbf{w} = [\mathbf{u}^T \quad \mathbf{g}^T]^T \in \mathbb{R}^{2^{m+1}-2}$. The unregularized QP in (3.9) can then be rewritten as

$$\min_{\mathbf{w}} 0.5\mathbf{w}^T\hat{D}_{\mathbf{w}}\mathbf{w} + \mathbf{f}_{\mathbf{w}}^T\mathbf{w}, \quad C_{\mathbf{w}}\mathbf{w} \leq \mathbf{0}, \quad \mathbf{b}_l \leq \mathbf{w} \leq \mathbf{b}_r, \tag{3.17}$$

where

$$\hat{D}_{\mathbf{w}} = \begin{bmatrix} \hat{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{f}_{\mathbf{w}} = [\mathbf{f}^T \quad \mathbf{0}^T]^T, \quad C_{\mathbf{w}} = [C \quad \mathbf{0}], \\ \mathbf{b}_l = [(\mathbf{0}, 1)^T \quad \mathbf{g}^T]^T, \quad \mathbf{b}_r = [\mathbf{1} \quad \mathbf{g}^T]^T. \tag{3.18}$$

Denote the vector of the sign of the differences at the i^{th} iteration as

$$\boldsymbol{\delta_i} = \text{sign}(\mathbf{u}_i - \mathbf{g}), \tag{3.19}$$

and the matrix including all $\boldsymbol{\delta}$s from iteration 0 as

$$G^i = \begin{bmatrix} \boldsymbol{\delta_0}^T & -\boldsymbol{\delta_0}^T \\ \boldsymbol{\delta_1}^T & -\boldsymbol{\delta_1}^T \\ \vdots & \vdots \\ \boldsymbol{\delta_i}^T & -\boldsymbol{\delta_i}^T \end{bmatrix}, \tag{3.20}$$

such that multiplication of G_i^i, the i^{th} row of G^i, with the vector $\mathbf{w}$ represent an ℓ_1-summation, e.g.,

$$G_i^i \mathbf{w}_i = \|\mathbf{u}_i - \mathbf{g}\|_1, \tag{3.21}$$

where $\mathbf{u}_i$ is the learned FM from the i^{th} iteration of the algorithm. Finally, let the regularization parameter be $t \propto 1/\lambda$, and the vectorized version is denoted as $\mathbf{t} = t\mathbf{1}$. The iterative lasso algorithm for this problem is given in Algorithm 3 following the

notation presented in (3.19)—(3.21).

Algorithm 3: ℓ_1-Goal Regularization via Tibshirani's Lasso Algorithm

Data: The QP at (3.17) and regularization parameter t.
Result: $\mathbf{u}$ - Lexicographically ordered fuzzy measure vector (recovered from $\mathbf{w}$).
$i = 0$;
$\mathbf{w}_0 \leftarrow$ solve the unregularized QP in (3.17);
$\boldsymbol{\delta_0} \leftarrow$ find the sign vector of (3.19);
$G^0 \leftarrow$ add $\boldsymbol{\delta_0}$ to G as shown in (3.20);
while $G_i^i \mathbf{w}_i > \mathbf{t}$ **do**
  $i \leftarrow i + 1$
  $\mathbf{w}_i \leftarrow$ solve the QP in (3.17) with the additional constraint $G^{i-1}\mathbf{w}_{i-1} \leq \mathbf{t}$.
  $\boldsymbol{\delta_i} \leftarrow$ find the sign vector of (3.19).
  $G^i \leftarrow$ add $\boldsymbol{\delta_i}$ to G as shown in (3.20).
Recover $\mathbf{u}$ from $\mathbf{w}_i$ as $\mathbf{u} = \begin{bmatrix} \boldsymbol{I} & \mathbf{0} \end{bmatrix} \mathbf{w}_i$, where $\boldsymbol{I}$ is the identity matrix.

3.4.3 Specific Aggregation Examples with Goal Regularization

The following sections describe specific aggregation examples using the ℓ_1- and ℓ_2-goal regularizations presented in the previous section. The value of the regularization parameter for ℓ_1-goal regularization (t) is given in each of the following subsections, and unless explicitly stated otherwise $\lambda = 1$ for each ℓ_2-goal regularization experiment that follows[5].

[5]Experiments have shown that the behavior of ℓ_1-goal regularization is much more sensitive to t than that of ℓ_2-goal regularization to λ.

3.4.3.1 Minimum Aggregation

It is interesting to note that when $\mathbf{g} = 0$, the regularization function in (3.13) reduces to that of ℓ_p-norm regularization of the FM vector. This is precisely why the learned FM's untouched nodes of Example 1 "default" to lie at the lowest end of their allowable range as shown in Table 3.1—we are essentially forcing the untouched FM values to be as close to zero as possible through our choice of ℓ_p-norm regularization of the FM vector. Tying this with the aggregation operators discussed in Section 3.2.3, we recognize that when $\mathbf{g} = 0$ we are forcing the Choquet integral to aggregate like the minimum function. For the ℓ_1- regularized example $t = 3.23$.

3.4.3.2 Maximum Aggregation

Defining the goal as all 1s causes the untouched nodes to default to the maximum end of their allowable range, tuning the Choquet integral's behavior to that of maximum aggregation (see Section 3.2.3). Rerunning the example in Section 3.3.1 with this goal yields the FM summarized in Table 3.1, where it is obvious that the untouched nodes are essentially assigned the maximum possible value permitted by the monotonicity constraints. Note that in this example the learned FM values for $g(\{x_1\})$ and $g(\{x_2, x_3\})$ using ℓ_2-goal regularization have been pushed farther from the underlying FM, though they still lie fairly close. This discrepancy is due to the choice of

λ, which essentially "tunes" where the error is incurred in the QP at (3.14)–a larger value of λ will force the learned FM to look like the goal $\mathbf{g}$ despite perturbing the data-driven nodes away from their underlying values. As previously mentioned, λ was arbitrarily set to 1 in these experiments for ℓ_2-goal regularization; $t = 2$ for ℓ_1-goal regularization.

3.4.3.3 Mean Aggregation

As a final example, we define the goal of the FM to be that of mean aggregation as explained in Section 3.2.3. Doing so leads to the learned FM shown in Table 3.1. Interestingly, the learned FM at the data-driven nodes is more accurate than that of the previous case of maximum aggregation. We attribute this to the fact that the goal of mean aggregation is more similar to the underlying FM than the goal of maximum aggregation. Interestingly, the results of both ℓ_1- and ℓ_2-goal regularized experiments are almost identical, where $t = 0.5$ for the ℓ_1-goal regularized experiment.

3.5 Learning the Goal

The previous section described a strategy for learning a FM when knowledge of the underlying FM is known and can be encoded in the QP, however, information regarding the underlying FM is rarely available in real-world problems. This section presents a method for addressing this issue by simultaneously learning the FM at data-driven nodes and learning an appropriate goal for the remaining nodes. To restrict the number of degrees of freedom we assume the underlying FM can be approximated by a *linear order statistic* (LOS), which is a special case of the Choquet integral.

3.5.1 Defining a FM from a LOS

A normalized[6] LOS for a sample $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is

$$L_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{m} g_i x_{(i)} = \mathbf{g}^T \mathbf{x}, \tag{3.22}$$

where $x_{(1)} \geq x_{(2)} \geq \cdots \geq x_{(m)}$, $w_i \geq 0$, and $\sum_{i=1}^{m} g_i = 1$. Note the ordering of the elements is similar to that of the Choquet integral, thus it should come as no surprise that the normalized LOS can be represented by a Choquet integral with respect to a

[6]The generalized LOS is $\frac{\sum_{i=1}^{m} g_i x_{(i)}}{\sum_{i=1}^{m} g_i}$, but we adapt the constraint of $\sum_{i=1}^{m} g_i = 1$ in our formulation.

symmetric FM[7].

Let the vector $\mathbf{g} \in \mathbb{R}^m$ now represent the LOS weight vector we wish to learn. We define the matrix $A \in \mathbb{R}^{(2^m-1)\times m}$ to map the LOS weight vector $\mathbf{g}$ into the domain of the FM $\mathbf{u}$. Then, for any $\mathbf{g}$ we can form an equivalent FM as $\hat{\mathbf{u}} = A\mathbf{g}$. For example, consider the LOS $\mathbf{g} \in \mathbb{R}^3$ and the FM vector has the ordering $\hat{\mathbf{u}} = (g(\{x_1\}), g(\{x_2\}), g(\{x_3\}), g(\{x_1, x_2\}), g(\{x_1, x_3\}), g(\{x_2, x_3\}), g(\{x_1, x_2, x_3\}),)$. Then A is defined as

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.23}$$

where each row sifts a LOS weight from $\mathbf{g}$ and assigns it to its corresponding FM term in $\hat{\mathbf{u}}$.

[7]A FM is symmetric if $\forall |A| = |B|$, $g(A) = g(B)$.

3.5.2 ℓ_2-LOS Regularization

Define the ℓ_2-LOS regularization function as

$$v_*(\mathbf{u}) = \lambda\|\mathbf{u} - A\mathbf{g}\|_2^2 = \lambda(\mathbf{u}^T\mathbf{u} - 2\mathbf{g}^T A^T\mathbf{u} + \mathbf{g}^T A^T A\mathbf{g}), \tag{3.24}$$

such that the QP at (3.10) becomes

$$\min_{\mathbf{u},\mathbf{g}} 0.5\mathbf{u}^T\hat{D}\mathbf{u} + \mathbf{f}^T\mathbf{u} + \lambda(\mathbf{u}^T\mathbf{u} - 2\mathbf{g}^T A^T\mathbf{u} + \mathbf{g}^T A^T A\mathbf{g}). \tag{3.25}$$

Note that we are now seeking to learn $\mathbf{u}$ *and* $\mathbf{g}$. As was done in Section 3.4.2, we lump $\mathbf{u}$ and $\mathbf{g}$ into $\mathbf{v}$ as $\mathbf{v} = [\mathbf{u}^T \quad \mathbf{g}^T]^T$ such that the QP can be rewritten as

$$\min_{\mathbf{v}} 0.5\mathbf{v}^T\hat{D}_{\mathbf{v}}\mathbf{v} + \mathbf{f}_{\mathbf{v}}^T\mathbf{v}, \quad C_{\mathbf{v}}\mathbf{v} \leq \mathbf{0}, \quad \mathbf{b}_{\mathbf{v}l} \leq \mathbf{v} \leq \mathbf{b}_{\mathbf{v}r}, \tag{3.26}$$

where

$$\hat{D}_{\mathbf{v}} = \begin{bmatrix} \hat{D} + \lambda\mathbf{I} & -\lambda A \\ -\lambda A^T & \lambda A^T A \end{bmatrix}, \quad \mathbf{f}_{\mathbf{v}} = [\mathbf{f}^T \quad \mathbf{0}^T]^T, \quad C_{\mathbf{v}} = \begin{bmatrix} C & \mathbf{0} \end{bmatrix}, \tag{3.27}$$

$$\mathbf{b}_{\mathbf{v}l} = [(\mathbf{0}, 1)^T \quad \mathbf{0}^T]^T, \quad \mathbf{b}_{\mathbf{v}r} = \mathbf{1}.$$

Once again, the QP at (3.26) is in "standard" form and can be solved via any QP solver.

3.5.3 ℓ_1-LOS Regularization

Define the ℓ_1-LOS regularization function as

$$v_*(\mathbf{u}) = \lambda\|\mathbf{u} - A\mathbf{g}\|_1. \tag{3.28}$$

Again, due to the absolute value operator in the ℓ_1 norm we cannot lump the regularization function into the QP as compactly as in the ℓ_2 case, thus the solution to this problem will parallel that of Section 3.4.2 closely; we will rewrite the unregularized QP, then add constraints iteratively. First lump $\mathbf{u}$ and $\mathbf{g}$ into $\mathbf{z} = [\mathbf{u}^T \quad \mathbf{g}^T]^T$, then rewrite the unregularized QP as

$$\min_{\mathbf{z}} 0.5\mathbf{z}^T\hat{D}_{\mathbf{z}}\mathbf{z} + \mathbf{f}_{\mathbf{z}}^T\mathbf{z}, \quad C_{\mathbf{z}}\mathbf{z} \leq \mathbf{0}, \quad \mathbf{b}_{\mathbf{z}l} \leq \mathbf{z} \leq \mathbf{b}_{\mathbf{z}r}, \tag{3.29}$$

where

$$\hat{D}_{\mathbf{z}} = \begin{bmatrix} \hat{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{f}_{\mathbf{z}} = [\mathbf{f}^T \quad \mathbf{0}^T]^T, \quad C_{\mathbf{z}} = \begin{bmatrix} C & \mathbf{0} \end{bmatrix}, \tag{3.30}$$

$$\mathbf{b}_{\mathbf{z}l} = [(\mathbf{0}, 1)^T \quad -\mathbf{1}^T]^T, \quad \mathbf{b}_{\mathbf{z}r} = \mathbf{1}.$$

Denote the vector of the sign of the differences at the i^{th} iteration as

$$\boldsymbol{\delta_i} = \text{sign}(S\mathbf{z}_i), \tag{3.31}$$

where

$$S = [\mathbf{I} \quad -A], \tag{3.32}$$

and define the matrix $\hat{S}^i$ as

$$[\hat{S}^i]_{:,j} = \boldsymbol{\delta_i} \circ [S]_{:,j}, \tag{3.33}$$

where $[S]_{:,j}$ is the j^{th} column of S and $\circ$ denotes the Hadamard product. We can then write the ℓ_1-sum at the i^{th} iteration as

$$\|\mathbf{u}_i - A\mathbf{g}_i\|_1 = \mathbf{1}^T \hat{S}^i \mathbf{z}_i, \tag{3.34}$$

Finally, we form the matrix G as

$$G^i = \begin{bmatrix} \mathbf{1}^T \hat{S}^0 \\ \mathbf{1}^T \hat{S}^1 \\ \vdots \\ \mathbf{1}^T \hat{S}^i \end{bmatrix}, \tag{3.35}$$

and denote its j^{th} row as G^i_j. Using this notation, Algorithm 4 describes the process of solving the ℓ_1-LOS regularized QP.

Algorithm 4: ℓ_1-LOS Regularization via Tibshirani's Lasso Algorithm

Data: The QP at (3.29) and regularization parameter t.
Result: $\mathbf{u}$ - Lexicographically ordered fuzzy measure vector (recovered from $\mathbf{z}$).
$i = 0$;
$\mathbf{z}_0 \leftarrow$ solve the unregularized QP in (3.29);
$G^0 \leftarrow$ form the matrix in (3.35);
while $G_i^i \mathbf{z}_i > t$ **do**
  $i \leftarrow i + 1$
  $\mathbf{z}_i \leftarrow$ solve the QP in (3.29) with the additional constraint $G^{i-1}\mathbf{z}_i \leq \mathbf{t}$.
  $G^i \leftarrow$ add $\mathbf{1}^T (\boldsymbol{\delta_i} \circ S)$ to G as shown in (3.35).
Recover $\mathbf{u}$ from $\mathbf{z}_i$ as $\mathbf{u} = \begin{bmatrix} \boldsymbol{I} & \mathbf{0} \end{bmatrix} \mathbf{z}_i$.

3.5.3.1 LOS Aggregation

Applying ℓ_1- and ℓ_2-LOS regularization ($\lambda = t = 1$) to our example in Section 3.3.1 leads to the learned FMs in Table 3.1. Similar to the previous methods, the FM of the data-driven nodes is learned with high accuracy in both cases. What is noteworthy, however, is how the nodes untouched by the training data are learned—the learned FM assigned to these untouched nodes is essentially the mean of the touched nodes at the same level in the lattice (touched nodes with the same cardinality). In other words, $g(\{x_2\}) = \frac{g(\{x_1\})+g(\{x_3\})}{2}$ and $g(\{x_1, x_2\}) = \frac{g(\{x_1,x_3\})+g(\{x_2,x_3\})}{2}$. Note that for this particular example, it is only a coincidence that the learned FM term $g(\{x_2\})$ exactly matches the underlying FM for ℓ_2-LOS regularization.

3.6 Experiments

Experiments were performed using no regularization, ℓ_p-norm regularization, and the goal-based regularization function in (3.13) with the DeFIMKL algorithm on various datasets from the UCI Machine Learning repository as well as toy datasets generated to purposefully exclude 80% of the training of nodes in an arbitrarily generated fuzzy lattice (three toy datasets were generated using 3, 5, and 8 densities, respectively). Each experiment consists of 100 trials, where in each trial a random partition of 80% of the data is used for training and the remaining data is sequestered for testing; the results we report comprise the mean and standard deviation of classification accuracies. Finally, we vary the regularization parameter, λ, to explore its effect on classification accuracy and the results with the best λs are reported; so, essentially we are comparing the best from each algorithm.

3.6.1 Results

Table 3.2 summarizes the results of these experiments. The best algorithms for each dataset are shown in bold font; a two-sample t-test at a 5% significance level is used to determine the statistically best results—hence, more than one algorithm can be considered as best. In all experiments at least one goal-based regularization function

Table 3.2
Classification Accuracy of Various Regularization Functions*

	Data Set			
Regularization	Sonar	Derm	Ecoli	Glass
None	**80.5 (5.63)**	94.3 (2.61)	**97.3 (1.90)**	91.2 (4.39)
ℓ_1	**78.4 (7.23)** $\lambda = 0.5$	89.6 (4.35) $\lambda = 0.5$	91.8 (2.79) $\lambda = 5$	82.7 (6.77) $\lambda = 0.5$
ℓ_2 (min)	**80.0 (6.72)** $\lambda = 0.5$	91.9 (3.09) $\lambda = 0.5$	91.8 (2.79) $\lambda = 0.5$	85.9 (5.79) $\lambda = 0.5$
max	72.0 (7.58) $\lambda = 0.5$	**97.4 (1.88)** $\lambda = 1.5$	**97.9 (1.91)** $\lambda = 4.5$	**94.2 (3.97)** $\lambda = 4.5$
mean	**76.6 (7.17)** $\lambda = 0.5$	**97.7 (1.49)** $\lambda = 3$	**97.1 (2.14)** $\lambda = 1.5$	**95.2 (3.07)** $\lambda = 1$
Regularization	Toy 3	Toy 5	Toy 8	
None	84.7 (6.48)	**95.0 (3.39)**	**98.4 (1.76)**	
ℓ_1	64.4 (7.23) $\lambda = 2.5$	91.0 (4.87) $\lambda = 0.5$	**96.9 (2.74)** $\lambda = 5$	
ℓ_2 (min)	64.2 (7.05) $\lambda = 2.5$	92.4 (3.88) $\lambda = 0.5$	91.4 (4.36) $\lambda = 5$	
max	88.5 (6.35) $\lambda = 1.5$	94.3 (2.84) $\lambda = 2$	**98.9 (1.61)** $\lambda = 2.5$	
mean	**94.8 (4.50)** $\lambda = 1.5$	**96.7 (2.30)** $\lambda = 5$	**98.4 (1.89)** $\lambda = 1$	

*Bold indicates best result according to a two-valued t-test at a 5% significance level.

emerges as a top performer. We also find that the max and mean goal-based regularization functions achieve superior results on the *Dermatology* and *Glass* datasets, suggesting that the data define an underlying FM that is most similar to mean or max aggregation. There is no clear trend in the results versus the regularization parameter λ, and not surprisingly the best selection of λ varies based on the dataset used.

3.7 Conclusion

This chapter first introduced a visualization technique that shows both the FM as well as the Choquet integral's path through the lattice. We also proposed and applied a new regularization function to our previously developed decision-level aggregation algorithm known as DeFIMKL. Including this new regularization function in the DeFIMKL algorithm allows knowledge of an underlying FM to be encoded into the algorithm's training procedure; thus, the user can define a particular goal for the FM before learning. We discussed the application of the new regularization function and demonstrated its behavior using synthetic and real-world datasets.

Chapter 4

Measures of the Shapley Index for Learning Lower Complexity Fuzzy Integrals

The material in this chapter is submitted for publication in Springer Granular Computing, April 2017.

4.1 Introduction

At the heart of challenges like machine intelligence, robotics, Big Data, geospatial intelligence, and humanitarian demining, to name a few, is the dilemma of data and information fusion. A key element of fusion is the underlying mathematics to convert multiple, potentially heterogeneous inputs into fewer outputs (typically one). The general hope is that the operation either summarizes well or enhances a system's performance by exploiting interactions across the different sources. A famous scenario is where "the whole can be worth more than the sum of its parts." Herein, we consider the *fuzzy integral* (FI), specifically Sugeno's fuzzy *Choquet integral* (CI), for data and information aggregation [85]. The CI is an aggregation operator generator as it is parametrized by the *fuzzy measure* (FM) [85], aka monotone and normal capacity. In [86], it was shown that the CI can produce numerous useful and common aggregation operators based on the properties of the capacity; for example the minimum, maximum, median, mean, soft maximum (minimum), other linear order statistics, etc. However, the CI can also produce a wealth of other more unique and tailored aggregation operators. The point is, the CI is a powerful non-linear function that is used often for fusion.

It is important to note that the CI is not trivial in any respect. The capacity has $2^N - 2$

free parameters, for N inputs, that must be either specified or learned. This exponential number can (and often does) impact an application rather quickly. For example, 10 inputs already gives rise to 1,022 values that must be specified or learned (and 5,110 monotonicity constraints at that). The reader can refer to [85, 87, 88, 89] for reviews of *analytical methods* for specifying a capacity relative to just knowledge about the worth of the individual sources (called densities, $g(\{x_i\})$ for $i \in \{1, 2, ..., N\}$). Examples of FMs include the Sugeno λ-FM [85], S-Decomposable FMs [87, 90], Belief (and Plausibility) and Possibility (and Necessity) FMs [90], and k-additive approaches (which model a limited number of capacity terms up to a k-tuple number) [91].

To date, numerous methods have been proposed to learn the CI. While similar in underlying objective, these methodologies can and often do vary greatly with respect to factors like application domain, mathematics and how the CI is being used (e.g., signal and image processing, regression, decision-level fusion, etc.). In [88, 89], Grabisch proposed *quadratic programming* (QP) to acquire the full capacity based on the idea of minimizing the *sum of squared error* (SSE). In [92], Keller et al. used gradient descent and then penalty and reward [93] to learn the densities in combination with the Sugeno λ-FM. In [83], Beliakov used linear programming and, in [82], a genetic algorithm was used to learn a higher-order (type-1) fuzzy set-valued capacity for the *Sugeno integral* (SI). Alternatively, the works [52, 94, 95] propose different ways to automatically acquire, and subsequently aggregate, full capacities of specificity and agreement based on the idea of crowd sourcing when the worth of the individuals is

not known but is instead extracted from the data. The reader can refer to [81] for a detailed review of other existing work prior to 2008.

An underrepresented and unsolved challenge is learning the CI with respect to more than one criteria. Herein, we focus on minimization of the SSE criteria, but do it in conjunction with model complexity. In [96], Mendez-Vazquez and Gader are the first that we are aware to study the inclusion of an information-theoretic index of capacity-complexity in learning the CI. Specifically, Mendez-Vazquez and Gader explored the task of sparsity promotion in learning the CI and provided examples in decision (namely algorithm) level fusion. Their work has two parts, the Gibbs sampler and the exploration of a lexicographically encoded capacity vector as the ℓ_p-norm complexity term. The goal of their regularization term was to explicitly reduce the number of nonzero parameters in the capacity to eliminate non-informative or "useless" information sources. In [97], the idea of learning the CI based on the use of a QP solver and a lexicographically encoded capacity vector was also explored. The novelty in that work is the study of different properties of the regularization term in an attempt to unearth what it was promoting in different scenarios (with respect to both the capacity but also the resultant aggregation operator induced by the CI). In the theme of information theoretic measures of capacities, but not regularization with respect to such indices, Kojadinovic et al. [98] and Yager [99, 100] both explored the concept of the entropy of an FM. Furthermore, in [101], Labreuche explored the identification of an FM with an ℓ_1 entropy. Labreuche proposed a linearized entropy

of an FM, allowing for the identification of an FM using linear programming.

For the most part, it appears that the vast majority of these works are largely unaware of each other. Therefore, in Section 4.2.2 we bridge this gap by analyzing and comparing different properties of these indices. In general, there does not appear to be a clear "winner." That is, different indices exist and are useful for various applications and goals (contexts). Therefore, it is important to understand each index and ultimately what context it supports. In addition, we put forth a few new indicies based on the Shapley values. Our intent is to promote "simpler models" that have either lower diversity in the Shapley values or fewer numbers of inputs (fewer non-zero Shapley terms). In fields such as statistics and machine learning, such a strategy can help with addressing challenges like preventing overfitting (aka improving the generalizability of a learned model). It is also the case that we are often concerned with problems having too many inputs, as more inputs are typically associated with higher *cost*—e.g., greater financial cost of different sensors, more computational or memory resources, time, or even physical cost in a health setting where an input is something like the result of a bone marrow biopsy.

This chapter is structured as such. Section 4.2 is a review of important concepts in FM and FI theory. Section 4.2.1 discusses the Shapley and interaction indices and Section 4.2.2 reviews and compares existing indices for measuring different notions of complexity of a capacity. In Section 4.2.3, the ℓ_0-norm, ℓ_1-norm and Gini-Simpson

index of the Shapley values are proposed and Section 4.3 discusses optimization of the CI relative to the SSE criteria based on the QP. We propose Gini-Simpson index-based regularization of the Shapley values in Section 4.4, Section 4.5 discusses ℓ_0-norm based regularization, and Section 4.6 contains experiments illustrating different scenarios encountered in practice.

4.2 Fuzzy Measure and Integral

The aggregation of data/information using the FI has a rich history. Much of the theory and several applications can be found in [86, 87, 88, 89, 102, 103, 104, 105]. There are a number of (high-level) ways to describe the FI, e.g., motivated by Calculus, signal processing, pattern recognition, fuzzy set theory, etc. Herein, we set the stage by considering a finite set of N sources of information, $X = \{x_1, ..., x_N\}$, and a function h that maps X into some domain (initially $[0, 1]$) that represents the partial support of a hypothesis from the standpoint of each source of information. Depending on the problem domain, X can be a set of experts, sensors, features, pattern recognition algorithms, etc. The hypothesis is often thought of as an alternative in a decision process or a class label in pattern recognition. Both Choquet and Sugeno integrals take partial support for the hypothesis from the standpoint of each source of information and fuse it with the (perhaps subjective) worth (or reliability) of each subset of X in a non-linear fashion. This worth is encoded in an FM (aka capacity).

Initially, the function h (integrand) and FM ($g : 2^X \rightarrow [0,1]$) took real number values in $[0,1]$. Certainly, the output range for the support function and capacity can be (and have been) defined more generally, e.g., $\Re_0^+$, but it is convenient to think of them on $[0,1]$ for confidence fusion. We now review the capacity and FI.

Definition 1 (Fuzzy Measure [85])**.** The FM is a set function, $g : 2^X \rightarrow \Re_0^+$, such that

P1. (Boundary condition) $g(\phi) = 0$ (often $g(X) = 1$);

P2. (Monotonicity) If $A, B \subseteq X$ and $A \subseteq B$, $g(A) \leq g(B)$.

Note, if X is an infinite set, a third condition guaranteeing continuity is required, but this is a moot point for finite X. As already noted, the FM has 2^N values; actually, $2^N - 2$ "free parameters" as $g(\phi) = 0$ and $g(X) = 1$. Before a definition can be given for the FI, notation must be established for the training data used to learn the capacity.

Definition 2 (Training Data)**.** Let a training data set, T, be

$$T = \{(O_j, \alpha_j) | j = 1, ..., m\}$$

where $\mathbf{O} = \{O_1, ..., O_j, ..., O_m\}$ is a set of "objects" and α_j are their corresponding labels (specifically, $\Re$-valued numbers). For example, O_j could be the strengths in

some hypothesis from N different experts, signal inputs at time j, algorithm outputs for input j, kernel inputs or kernel classifier outputs for feature vector j, etc. Subsequently, α_j could be the corresponding function output, class label, membership degree, etc. Next, we provide a definition for the FI, namely the CI with respect to T. To that end, let h_j be the jth integrand, i.e., $h_j(x_i)$ is the input for the ith source with respect to object j.

Definition 3. The discrete CI, for finite X and object O_j is

$$\begin{aligned} \int h_j \circ g &= C_g(h_j) \\ &= \sum_{i=1}^{N} [h_j(x_{\pi_j(i)}) - h_j(x_{\pi_j(i+1)})] g(A_{\pi_j(i)}), \end{aligned} \tag{4.1}$$

for $A_{\pi_j(i)} = \{x_{\pi_j(1)}, ..., x_{\pi_j(i)}\}$ and permutation π_j such that $h_j(x_{\pi_j(1)}) \geq ... \geq h_j(x_{\pi_j(N)})$, where $h_j(x_{\pi_j(N+1)}) = 0$ [85].

4.2.1 Shapley and Interaction Indices

The CI is parametrized by the capacity. Specifically, the capacity encodes all of the rich tuple-wise interactions between the different sources and the CI utilizes this information to aggregate the inputs (the integrand, h). It is important to note that the CI operates on a weaker (and richer) premise than a great number of other aggregation operators that assume additivity (a stronger property than monotonicity). However,

the capacity has a large number of values. It is not trivial to understand a capacity. For example, a commonly encountered question is what is the so-called *worth* of a single individual source? Information theoretic indices aid us in summarizing information such as this in the capacity. The point is, most of our questions are not about a single capacity value; we are interested in a complex question whose answer is dispersed across the capacity. For example, the Shapley index has been proposed to summarize the so-called worth of an individual source and the interaction index summarizes interactions between different sources.

Definition 4 (Shapley Index)**.** The Shapley values of g are

$$\Phi_g(i) = \sum_{K \subseteq X \backslash \{i\}} \zeta_{X,1}(K) \left(g(K \cup \{i\}) - g(K) \right), \tag{4.2a}$$

$$\zeta_{X,1}(K) = \frac{(|X| - |K| - 1)!|K|!}{|X|!}, \tag{4.2b}$$

where $X \backslash \{i\}$ denotes all subsets from X that do not include source i. The Shapley value of g is the vector $\Phi_g = (\Phi_g(1), ..., \Phi_g(N))^t$ and $\sum_{i=1}^{N} \Phi_g(i) = 1$. The Shapley index can be interpreted as the average amount of *contribution* of source i across all coalitions. Equation (4.2a) makes its decision based on the weighted sum of (positive-valued) numeric differences between consecutive steps (layers) in the capacity.

Remark 4. It is important to note the following property. When $g(A) = 0, \forall A \subset X$, the CI is the minimum operator. The Shapley values are $\Phi_g(1) = \Phi_g(2) = ... =$

$\Phi_g(N)$. This is easily verifiable as the Shapley is a weighted sum of differences between $g(X)$ and $g(X \backslash x_i)$ (one of our inputs). Thus, each Shapley value reduces to the same calculation, $\zeta_{X,1}(K)$, where $K \in 2^X$ and $|K| = |X| - 1$.

Definition 5 (Interaction Index)**.** The interaction index (Murofushi and Soneda [106]) between i and j is

$$\begin{aligned} \mathbf{I}_g(i,j) = & \sum_{K \subseteq X \backslash \{i,j\}} \zeta_{X,2}(K)(g(K \cup \{i,j\}) \\ & - g(K \cup \{i\}) - g(K \cup \{j\}) + g(K)), \quad i = 1, ..., N, \end{aligned} \tag{4.3a}$$

$$\zeta_{X,2}(K) = \frac{(|X| - |K| - 2)!|K|!}{(|X| - 1)!}, \tag{4.3b}$$

where $\mathbf{I}_g(i,j) \in [-1,1], \forall i,j$. A value of 1 (respectively, -1) represents the maximum complementarity (respective redundancy) between i and j. The reader can refer to [107] for further details about the interaction index, its connections to game theory and interpretations. Grabisch later extended the interaction index to the general case of any coalition [107].

Definition 6 (Interaction Index for coalition A)**.** The interaction index for any coalition $A \subseteq X$ is

$$\mathbf{I}_g(A) = \sum_{K \subseteq X \setminus A} \zeta_{X,3}(K, A) \sum_{C \subseteq A} (-1)^{|A \setminus C|} g(C \cup K),$$

$$i = 1, ..., N, \tag{4.4a}$$

$$\zeta_{X,3}(K, A) = \frac{(|X| - |K| - |A|)!|K|!}{(|X| - |A| + 1)!}. \tag{4.4b}$$

Equation (4.4a) is a generalization of both the Shapley value and Murofushi and Soneda's interaction index as $\Phi_g(i)$ corresponds with $\mathbf{I}_g(\{i\})$ and $\mathbf{I}_g(i, j)$ with $\mathbf{I}_g(\{i, j\})$.

While the Shapley and interaction indices are extremely useful, they do not, in their current explicit form, inform us about capacity complexity. In the next subsection, we review additional information theoretic capacity indices and we discuss their interpretations.

4.2.2 Existing Indices for Capacity Complexity

Excluding indices that are subsumed by others, the bottom line is various indices exist for different reasons. First, some indices are simpler computationally while others are mathematically simpler in terms of our ability to manipulate and use them for tasks

like optimization. Second, and arguably the most important, complexity can and often does mean different things to different people/applications. As we discuss in this chapter, there is no clear winner (i.e., index). Different indices are important for different applications and knowledge of their existence and associated benefits is what is ultimately important. In this section we review existing information theoretic indices for complexity.

Definition 7 (ℓ_1-Norm of a Lexicographically Encoded Capacity Vector)**.** Let $\mathbf{u}$ (a vector of size $(2^N - 1) \times 1$) be $\mathbf{u} = (g_1, g_2, ..., g_{12}, g_{13}..., g_{12...N})^t$. A relatively simple index of the complexity of g is

$$v_{\ell 1}(g) = \sum_{j=1}^{2^N-2} |\mathbf{u}_j| = \sum_{j=1}^{2^N-2} \mathbf{u}_j. \tag{4.5}$$

As stated in [96, 97], the intent of $v_{\ell 1}(g)$ was to help reduce the number of nonzero parameters in the capacity to eliminate non-informative or useless information sources. However, this index is not as sophisticated as desired. The index is minimized when all $\mathbf{u}_j$ are equal to zero, i.e., the FM $g(A) = 0, \forall A \subset X$, which is a minimum operator for the CI [86]. We also note that this is an FM of "ignorance," as we assert that the answer resides in X, however we have assigned $g(A) = 0$ to all subsets outside X. The index is maximized for the FM $g(A) = 1, \forall A \subseteq X$, which is a maximum operator for the CI [86]. There are really two problems with this index. First, it does not take advantage of any capacity summary mechanism like the Shapley index, interaction

index or k-additivity. Second, it is well-known that the ℓ_1 is (geometrically) inferior to the ℓ_0 when it comes to promoting sparsity. However, the ℓ_1-norm gives rise to convex problems that we can more easily solve for while the later does not.

Definition 8 (Marichal's Index)**.** Marichal's Shannon-based entropy of g [108, 109],

$$v_M(g) = (-1)\sum_{j=1}^{N}\Big(\sum_{K\subseteq X\setminus\{j\}} \zeta_{X,1}(K)\,(g(K\cup\{j\}) - g(K))$$
$$\ln(g(K\cup\{j\}) - g(K))\Big), \tag{4.6a}$$

is motivated in terms of the following [110]. Consider the set of all maximal chains of the Hasse diagram $(2^N, \subseteq)$. A maximal chain in $(2^N, \subseteq)$ is a sequence

$$\phi, \{x_{\pi(1)}\}, \{x_{\pi(1)}, x_{\pi(2)}\}, ..., \{x_{\pi(1)}, ..., x_{\pi(N)}\},$$

where π is a permutation of N. On each chain, we can define a "probability distribution,"

$$p_\pi^g(i) := g(\{x_{\pi(i)}, ..., x_{\pi(N)}\}) - g(\{x_{\pi(i+1)}, ..., x_{\pi(N)}\}),$$
$$i = 1...N, \pi \in \Pi_N.$$

It is not entirely clear to us why this is called a probability distribution. For example, it is confusing why this is the case for a Belief measure, a Possibility measure, etc. We assume it is interpreted as such due to the properties of positivity and the distribution

sums to 1. Furthermore, [110] states that "...the intuitive notion of *uniformity of a capacity* g on N can then be defined as an average of the uniformity values of the probability distributions" (distributions provided according to $p_\pi^g(i)$) [110]. Regardless, this account of entropy is the average of the uniformity values of the underlying probability distributions. In general, such an index can be of help with respect to the *maximum entropy* principle. Furthermore, maximization of index $v_M(g)$ is non-linear and not quadratic [101]. As stated in [101], we can obtain a quadratic problem under linear constraints, considering a special case of Renyi entropy.

It is trivial to prove that minimum entropy for Equation (4.6a) occurs if and only if $g(K \cup \{j\}) - g(K)$ yields values in $\{0, 1\}$. Note, Equation (4.6a) is defined for

$$\ln\left(g(K \cup \{j\}) - g(K)\right) = 0$$

by choosing 0. While many properties of this definition of entropy are discussed in [110], a few important properties were not discussed. First, there is not a single unique "solution" (minimum). That is, an FM of all 0s (minimum operator) and an FM of all 1s (maximum operator) both satisfy this criteria. There are other FMs that satisfy this criteria as, e.g., the N different order-statistics where a single input becomes the output and all other inputs are discarded (one input has a Shapley value of 1 and all other inputs have a Shapley value of 0). Also, there is the case of the *ordered weighted average* (OWA) [111]. An OWA is a class of FMs (when paired with the CI) in which

sets of equal size cardinality have equal measure value. The OWA weights are simply the differences between the constant tuple values, i.e., $w_i = g(A_i) - g(A_i \setminus \{i\})$. For N inputs, we have N such OWAs that yield the mentioned minimum—capacities with values of 1 for all sets $A \subseteq X$ with $|A| \geq k$ and 0 otherwise, for $k = 1...N$. Note, two of these N cases are the maximum and minimum aggregation operators. On the other hand, maximum entropy occurs in the case of a "uniform distribution" (all $\frac{1}{N}$ values). This only occurs in the case of a capacity in which $g(A) = |A|/|X|$, which is a CI-based average operator. This uniqueness of the maximization case was one of the motivating reasons for the proposal of Marichal's index (maximum entropy principle).

Definition 9 (Shannon's Entropy of the Shapley Values)**.** In [97], a related but different formulation of Shannon's entropy was explored in terms of the Shapley values,

$$v_S(g) = (-1) \sum_{j=1}^{N} \Phi_g(j) \ln(\Phi_g(j)). \tag{4.7}$$

Note, the Shapley index values sum to 1, i.e.,

$$\sum_{j=1}^{N} \Phi_g = 1.$$

Furthermore, Equation (4.7) is not defined for $\Phi_g(j) = 0$; it is by choosing $\ln(\Phi_g(j)) = 0$. When only one source is needed, a single value is 1 and the others are 0, i.e., Equation (4.7) equals 0. There are N such unique cases. There are

also not any such cases in which the Shapley values are all 0s or 1s (by definition). On the other hand, the more uniformly distributed the Shapley values become, the more inputs are required (each are important relative to solving the task at hand). In the extreme case, as when all Shapley values are $\frac{1}{N}$, all sources are needed and we obtain maximum entropy. This occurs when g causes the CI to reduce to an OWA and there is an infinite set of such capacities/OWAs (for a $\Re$-valued FM).

In summary, there are fewer and more easily rationalized solutions for Equation (4.7) versus Equation (4.6a) in the case of minimizing the entropy of a capacity and the latter has fewer solutions for maximizing entropy. However, while there are more solutions in the case of Equation (4.7), they can easily be rationalized (all such capacities treat the inputs as equally important in terms of the CI). These two definitions of entropy are similar but not equivalent.

Definition 10. Kojadinovic's variance of g is [110]

$$v_K(g) = \frac{1}{N}\sum_{j=1}^{N}\Big(\sum_{K\subseteq X\setminus\{j\}} \zeta_{X,1}(K) \big(g(K\cup\{j\}) - g(K) - \frac{1}{N}\big)^2\Big). \tag{4.8a}$$

It is trivial to verify that this index equals 0 if and only if the differences between the capacity terms equal $\frac{1}{N}$. This is unique in the fact that it only occurs in the case of $g(A) = |A|/|X|$ (i.e., a CI that reduces to the average operator). As Kojadinovic

discusses, Equation (4.8a) is a simpler way (versus Marichal's index which involves logarithms) to measure the uniformity of a distribution. Also, Equation (4.8a) equates to 0 if and only if we have the case of a "uniform distribution." Kojadinovic's goal, in the theme of Marichal's notion of entropy, is that of maximum entropy—the "least specific" capacity compatible with the initial preferences of a decision maker. Kojadinovic's variance is maximized in the case that the difference of the two capacity terms equals 0 or 1. This occurs in the case of a minimum operator, maximum operator, or the other $(N-2)$ OWAs discussed in the case of Marichal's entropy. Thus, Kojadinovic's variance and Marichal's entropy are tightly coupled, while Equation (4.7) is once again different in its design and set of relevant solutions.

Definition 11. Labreuche's linearized entropy of g is [101]

$$v_L(g) = (-1)\sum_{j=1}^{N}\Bigg(\sum_{K\subseteq X\setminus\{j\}} \zeta_{X,1}(K) \left|g(K\cup\{j\}) - g(K) - \frac{1}{N}\right|\Bigg). \tag{4.9a}$$

The primary goal of this index is to linearize Kojadinovic's index to assist in optimization (apply linear programming). Labreuche's goal was to also satisfy, with respect to the different probability distributions, symmetry (value regardless of input permutation), maximality and minimality (probability distribution of all $\frac{1}{N}$ values and the distribution of all zeros with a single value of one). Kojadinovic's index does not satisfy the last two properties. Furthermore, index $v_L(g)$ has a single minimum, the

capacity $g(A) = \frac{|A|}{|X|}$, which results in the CI becoming the mean operator. Labreuche's index also has a single maximum, one for each probability distribution. In terms of the capacity, this equates to the minimum operator, the maximum operator, and the other $(N-2)$ OWAs discussed for Kojadinovic's index.

Definition 12. The k-additive based index is [112, 113, 114]

$$v_T(g) = \sum_{A \subseteq X} f(|A|)|\mathcal{M}(A)|, \tag{4.10}$$

where f is a strictly increasing function defined on the cardinality of subsets of X and $\mathcal{M}$ is the Mobius transform of g [89, 107]. The Mobius transform of g is used here to highlight and exploit k-additivity, i.e., $\mathcal{M}(B) = 0, \forall B \subseteq X$ with $|B| > k$. This is a different approach as k-additivity allows for what could be considered a "compact" representation of g (under a set of restrictions) to combat the otherwise combinatorial explosion of g: $\sum_{i=1}^{k} \binom{N}{i}$ terms versus 2^N. In summary, $v_T(g)$ favors the restriction that capacities have a low level of nonadditivity.

It is well-known that the sum of the Mobius terms for the capacity is equal to one [115]. However, $v_T(g)$ considers the sum of the absolute values of the Mobius terms. It is trivial to prove that $v_T(g)$ has a single maximum for the case of a capacity of all ones, $g(A) = 1, \forall A \subseteq X$, i.e., the maximum operator. Although these values sum to one, they can be any value in $[-1, 1]$. This index does not have a unique minimum. For example, a capacity of all zeros (except $g(X)$) has an index value of 1, the mean

operator, $g(A) = \frac{|A|}{|X|}$, has an index of 1, and a capacity where a single input has a Shapley value of 1 has an index of 1. In general, the higher the k-additivity, the greater the $v_T(g)$.

While the above indicies are all useful in their respective contexts, none truly address the desire to favor fewer numbers of inputs. In the next subsection we explore a few new indices to achieve this goal based on utilization of the Shapley values.

4.2.3 New Indices of Complexity Based on the Shapley Values

Definition 13 (ℓ_1-Norm of Shapley Values)**.** Let $\mathbf{\Phi}_g = (\Phi_g(1)\ \Phi_g(2)\ ...\ \Phi_g(N))^t$, a vector of size $N \times 1$, be the vector of Shapley values. The so-called ℓ_1-norm of $\mathbf{\Phi}_g$ is

$$\|\mathbf{\Phi}_g\|_1 = \sum_{i=1}^{N} |\Phi_g(i)| = \sum_{i=1}^{N} \Phi_g(i) = 1. \tag{4.11}$$

It is important to note that the constraint that the Shapley values sum to 1 renders the ℓ_1 index useless for regularization (as it yields a constant). Next, we explore the ℓ_0.

Definition 14 (ℓ_0-Norm of Shapley Values)**.** Let $\mathbf{\Phi}_g = (\Phi_g(1)\ \Phi_g(2)\ ...\ \Phi_g(N))^t$, a

vector of size $N \times 1$, be the vector of Shapley values. The so-called ℓ_0-norm of $\mathbf{\Phi}_g$ is

$$\|\mathbf{\Phi}_g\|_0 = |\{i : \Phi_g(i) \neq 0\}| . \tag{4.12}$$

Technically, the ℓ_0-norm is not really a norm. It's a cardinality function that counts the number of non-zero terms. The ℓ_0-norm has been used extensively in areas like compressive sensing, where the goal is to typically find the sparsest solution for an under-determined linear system. If we define Equation (4.12) on the lexicographically encoded capacity vector, versus the Shapley values vector, then we would be back in the same predicament of striving for a capacity of all 0s (except for $g(X) = 1$), viz., the minimum operator for the CI. It is clear that Equation (4.12) has its minimum for the case of one Shapley value equal to 1 (thus all other Shapley values are equal to 0). Its next smallest value is for the case of two Shapley values greater than zero and all other Shapley values are equal to zero (and so forth). It is clear to see that sparsity, in the sense of the fewest number of non-zero values, is preserved via the ℓ_0-norm. Specifically, Equation (4.12) has N minima, e.g., $\mathbf{\Phi}_g = (1, 0, ..., 0)^t$, $\mathbf{\Phi}_g = (0, 1, 0, ..., 0)^t$, ..., $\mathbf{\Phi}_g = (0, 0, ..., 0, 1)^t$. For two non-zero values, there are $\binom{N}{2}$ such solutions ($\binom{N}{k}$ in general for k non-zero inputs).

As an index, the ℓ_0 with respect to the Shapley values is fantastic at helping promote fewer number of non-zero parameters (inputs). Problem solved, correct? Not entirely.

One (big) challenge is that the ℓ_0 results in a non-convex optimization problem that is NP-hard. Before we consider ℓ_0 approximation, we investigate an alternative, but theoretically inferior (the tradeoff), index that is simpler to solve for based on the Gini-Simpson coefficient.

The Gini coefficient (aka Gini index or Gini ratio) is a summary statistic of the Lorenz curve and it was developed by the Italian statistician Corrado Gini in 1912. It is important to note that numerous mathematical formulations exist, from Corrado's original formula to Brown's Gini-style index measuring inequalities in health [116, 117]. A full review of the Gini index and its various discrete and continuous formulations is beyond the scope of this chapter (for a recent review, see [118]). The Gini index is used extensively in areas like biological systems (for measuring species similarity [119]), knowledge discovery in databases (often referred to as an "impurity function"), social sciences and economics. For example, it is often used as a measure of income inequality within a population. On one extreme, the Gini index equates to perfect "equality" (everyone has the same income) and, at the other extreme, to perfect "inequality" (one person has all the wealth and everyone else has zero income). Herein, we use a mathematically simple, but pleasing nevertheless, instantiation of the Gini index—at Equation (4.13)—often referred to as the Gini-Simpson index (or in ecology as the probability of interspecific encounter) with respect to a probability distribution (the Shapley values satisfy this criterion). However, the Gini-Simpson function belongs to a larger family of functions parametrized by a variable q (the

sensitivity parameter) and Z (a matrix of similarity values between elements in the distribution) [119]. Based on q and Z, we get different diversity measures, e.g., Shannon's entropy, Rao's quadratic entropy, "species richness" index, etc.

Definition 15 (Gini-Simpson Index of Shapley Values)**.** The Gini-Simpson index of the Shapley values is

$$v_G(g) = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} \Phi_g(i)\Phi_g(j) = 1 - \sum_{i=1}^{N} \Phi_g(i)^2. \tag{4.13}$$

Note, $v_G(g) = 0$ if and only if there is a single Shapley value equal to 1 (therefore all other values are equal to 0). There are N such possible unique solutions to this criteria. If $\Phi_g(i) = 1$ and $\Phi_g(j) = 0, \forall j \neq i$, then all g subsets that contain input i are of value 1 and 0 elsewhere. Also, the maximum of $v_G(g)$ occurs only when all Shapley values are equal. This equates to an infinite number of particular OWA solutions in which all inputs are "equally important." It is obvious that Equation (4.13) is nothing more than one minus the squared ℓ_2-norm of the Shapley values. Next, we provide simple numeric examples (Table 4.1) to (empirically) demonstrate some similarities and differences between the ℓ_0 and the Gini-Simpson.

Again, the ℓ_0 wants the fewest number of non-zero parameters and the Gini-Simpson index is a measure of diversity in the Shapley values, or more specifically one minus the squared ℓ_2-norm, that ultimately aims to promote, in the extreme case, a single

Table 4.1
Numeric Examples, for $N = 3$, Illustrating ℓ_0 and Gini-Simpson Differences.

FM	**Shapley Values**	ℓ_0	**Gini-Simpson**
g_a	$\mathbf{\Phi}_{g_a} = (1, 0, 0)$	1	0
g_b	$\mathbf{\Phi}_{g_b} = (.8, .2, 0)$	2	0.320
g_c	$\mathbf{\Phi}_{g_c} = (.5, .5, 0)$	2	0.500
g_d	$\mathbf{\Phi}_{g_d} = (.8, .1, .1)$	3	0.340
g_e	$\mathbf{\Phi}_{g_e} = (.4, .3, .3)$	3	0.660
g_f	$\mathbf{\Phi}_{g_f} = (.999, .0005, .0005)$	3	0.002
g_g	$\mathbf{\Phi}_{g_g} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$	3	0.667

dominant input (one Shapley value of 1 and all other values equal to 0). They both have lowest value for a single input (case g_a) and maximum value for the case of a uniform distribution (case g_g). While their trends are often similar, e.g., both prefer g_a to g_b and g_b to g_d and g_e, they do not always obviously agree. For example, consider g_c and g_d. The ℓ_0-norm prefers g_c to g_d as the prior has one zero term and the latter has no zero terms. However, the Gini-Simpson index prefers g_d to g_c. In g_c, the Shapley values indicate that one input is not important while the other two inputs are equally important. In g_d, the Shapley values indicate that one input is important and the other two inputs are equal and not that important at that. According to the Gini-Simpson index, g_d is closer to a single input vs g_c. This behavior is further emphasized by g_f.

In addition, due to the relationship between the Gini-Simpson and the ℓ_2-norm for the Shapley values, the underlying geometric interpretation and sparseness of solutions for the family of ℓ_p-norms is well-known and heavily published (e.g., see [120]). In the

following sections, we outline a way to perform regularization based on the ℓ_0-norm and the Gini-Simpson index of the Shapley values. However, we first review QP based solutions to capacity learning and ℓ_p-norm regularization.

4.3 Sum of Squared Error and Quadratic Programming

Definition 16 (Sum of Squared Error of CI and T)**.** Let the SSE between T and the CI be [89, 97]

$$E_1 = \sum_{j=1}^{m} (C_g(h_j)) - \alpha_j)^2. \tag{4.14}$$

Equation (4.14) can be expanded as follows,

$$E_1 = \sum_{j=1}^{m} (\mathbf{A}_{O_j}^t \mathbf{u} - \alpha_j)^2,$$

$\mathbf{A}_{O_j}^t = \big(..., h_j(x_{\pi_j(1)}) - h_j(x_{\pi_j(2)}), ..., 0, ..., h_j(x_{\pi_j(N)})\big)^t$ is of size $1 \times (2^N - 1)$. Note, the function differences, $h_j(x_{\pi_j(i)}) - h_j(x_{\pi_j(i+1)})$, correspond to their respective g locations

in $\mathbf{u}$. Folding Equation (4.14) out further, we find

$$\begin{aligned} E_1 &= \sum_{j=1}^{m} (\mathbf{u}^t \mathbf{A}_{O_j} \mathbf{A}_{O_j}^t \mathbf{u} - 2\alpha_j \mathbf{A}_{O_j}^t \mathbf{u} + \alpha_j^2) \\ &= \mathbf{u}^t \mathbf{D} \mathbf{u} + \mathbf{f}^t \mathbf{u} + \sum_{j=1}^{m} \alpha_j^2, \end{aligned} \tag{4.15}$$

where $\mathbf{D} = \sum_{j=1}^{m} \mathbf{A}_{O_j} \mathbf{A}_{O_j}^t$ and $\mathbf{f} = \sum_{j=1}^{m} (-2\alpha_j \mathbf{A}_{O_j})$. In total, the capacity has $(N(2^{N-1} - 1))$ monotonicity constraints. These constraints can be represented in a compact linear algebra (aka matrix) form. The following is the minimum number of constraints needed to represent the FM. Let $\mathbf{C}\mathbf{u} + \mathbf{b} \leq \mathbf{0}$, where $\mathbf{C}^t = \left(\Psi_1^t, \Psi_2^t, ..., \Psi_{N+1}^t, ..., \Psi_{N(2^{N-1}-1)}^t\right)^t$, and Ψ_1 is a vector representation of constraint 1, $g_1 - g_{12} \leq 0$. Specifically, $\Psi_1^t \mathbf{u}$ recovers $\mathbf{u}_1 - \mathbf{u}_{N+1}$. Thus, C is simply a matrix of $\{0, 1, -1\}$ values,

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & ... & -1 & 0 & ... & ... & 0 \\ 1 & 0 & ... & 0 & -1 & ... & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & ... & 0 & 0 & ... & 1 & -1 \end{bmatrix}, \tag{4.16}$$

which is of size $(N(2^{N-1} - 1)) \times (2^N - 1)$. Also, $\mathbf{b} = \mathbf{0}$, a vector of all zeroes. Note, in some works, $\mathbf{u}$ is of size $(2^N - 2)$, as $g(\phi) = 0$ and $g(X) = 1$ are explicitly encoded. In such a case, $\mathbf{b}$ is a vector of 0s and the last N entries are of value -1. Herein, we use the $(2^N - 1)$ format as it simplifies the subsequent Shapley index mathematics.

Given T, the search for FM g reduces to a QP of the form

$$\min_{\mathbf{u}} \frac{1}{2}\mathbf{u}^t\hat{\mathbf{D}}\mathbf{u} + \mathbf{f}^t\mathbf{u}, \tag{4.17}$$

subject to $\mathbf{Cu} + \mathbf{b} \geq \mathbf{0}$ and $(\mathbf{0}, 1)^t \leq \mathbf{u} \leq \mathbf{1}$. The difference between Equation (4.17) and (4.15) is $\hat{\mathbf{D}} = 2\mathbf{D}$ and inequality in Equation (4.16) need only be multiplied by -1.

In [96], it was pointed out that the QP approach for learning the CI is not without flaw due to the exponential size of the input. While scalability is definitely of concern, many techniques have and continue to be proposed for solving QPs with respect to fairly large and sparse matrices [121]. This attention and progress is coming primarily as a response to machine learning, statistics and signal processing. A somewhat large and sparse matrix is not a "game stopper." We do agree that there is mathematically a point where the task at hand does become extremely difficult to solve and may eventually become intractable. However, most FI applications utilize a relatively small number of inputs, i.e., on the order of 3 to 5, versus 50, 100. The notion that the QP has little-to-no value just because it is difficult (and may become intractable) to solve with respect to a sparse matrix for a large number of inputs is no reason to dismiss it. This challenge as akin to the current Big Data revolution, where previously intractable problems are being solved on a daily basis.

In general, the challenge of QP-based learning of the CI relative to a regularization term for tasks like decision-level fusion is the optimization of

$$E_2 = \sum_{j=1}^{m} (\mathbf{u}^t \mathbf{A}_{O_j} \mathbf{A}_{O_j}^t \mathbf{u} - 2\alpha_j \mathbf{A}_{O_j}^t \mathbf{u} + \alpha_j^2) + \lambda v_*(g), \tag{4.18}$$

where $v_*(g)$ is one of our indices. In order for Equation (4.18) to be suitable for the QP, v_* must be linear or quadratic.

Note, in certain problems one can simply fold the ℓ_1 regularization term into the linear term of the quadratic objective. We can rewrite $\|\mathbf{u}\|_1 = \mathbf{1}^t\mathbf{u}$, where $\mathbf{1}$ is the vector of all ones. Adding the regularization term to the QP, we get

$$\min_{\mathbf{u}} \frac{1}{2}\mathbf{u}^t\hat{\mathbf{D}}\mathbf{u} + \mathbf{f}^t\mathbf{u} + \lambda\mathbf{1}^t\mathbf{u} = \min_{\mathbf{u}} \frac{1}{2}\mathbf{u}^t\hat{\mathbf{D}}\mathbf{u} + (\mathbf{f} + \lambda\mathbf{1})^t\mathbf{u}. \tag{4.19}$$

4.4 Gini-Simpson Index-Based Regularization of the Shapley Values

We begin by considering a vectorial encoding of the Shapley index. The Shapley value of input 1 is

$$\Phi_g(1) = \sum_{K \subseteq X \setminus \{i=1\}} \mathbf{\Gamma}_X(K)(g(K \cup \{i=1\}) - g(K)), \tag{4.20a}$$

$$= \eta_1 g(\{x_1\}) + \eta_2[(g(\{x_1, x_2\}) - g(\{x_2\}))$$

$$+ (g(\{x_1, x_3\}) - g(\{x_3\})) + ...] + ..., \tag{4.20b}$$

$$= \eta_1 g(\{x_1\}) - [\eta_2 g(\{x_2\}) + ... + \eta_2 g(\{x_N\})]$$

$$+ [\eta_2 g(\{x_1, x_2\}) + \eta_2 g(\{x_1, x_N\})] + ..., \tag{4.20c}$$

where $\eta_i = \mathbf{\Gamma}_X(K)$, and $K \in 2^X$, s.t. $|K| = i - 1$ (Shapley normalization constants). What Equation (4.20a) tells us is the following. The Shapley index can be represented in linear algebra/vectorial form,

$$\mathbf{\Gamma}_i = (\mathbf{\Gamma}_{i,1}, \mathbf{\Gamma}_{i,2}, ...)^t, \tag{4.21}$$

where $\mathbf{\Gamma}_i$ is the same size as g and the $\mathbf{\Gamma}_i$ terms are the coefficients of Equation (4.20a). For example, for $N = 3$,

$$\Gamma_1 = \left(\frac{1}{3}, -\frac{1}{6}, -\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, -\frac{1}{3}, \frac{1}{3}\right)^t.$$

Thus, we can formulate a compact expression of an individual Shapley value as such,

$$\Phi_g(i) = \mathbf{\Gamma}_i^t \mathbf{u}, \tag{4.22}$$

where $\Phi_g(i) \in [0, 1]$. Therefore, the Gini-Simpson index in linear algebra form becomes

$$v_G(g) = 1 - \sum_{k=1}^{N} \left(\mathbf{\Gamma}_k^t \mathbf{u}\right)^2. \tag{4.23}$$

Expanding Equation (4.23) exposes an attractive property:

$$\begin{aligned} v_G(g) =& 1 - \sum_{k=1}^{N} \left(\mathbf{\Gamma}_k^t \mathbf{u}\right)^2, \\ =& 1 - \sum_{k=1}^{N} \left(\mathbf{u}^t \mathbf{\Gamma}_k \mathbf{\Gamma}_k^t \mathbf{u}\right), \\ =& 1 - \mathbf{u}^t \mathbf{Z} \mathbf{u}, \end{aligned} \tag{4.24}$$

where $\mathbf{Z} = \mathbf{\Gamma}_1 \mathbf{\Gamma}_1^t + ... + \mathbf{\Gamma}_N \mathbf{\Gamma}_N^t$. First, $\mathbf{\Gamma}_k \mathbf{\Gamma}_k^t$ is *positive semi-definite* (PSD). Hence, $\mathbf{Z}$ is also PSD, as it is simply the addition of PSD matrices and addition preserves the PSD property. We propose a Gini-Simpson index-based regularization of E_1 at

(4.14) as follows.

Definition 17 (SSE with Gini-Simpson Index Regularization)**.** The Gini-Simpson index regularization is

$$E_3 = \mathbf{u}^t\mathbf{D}\mathbf{u} + \mathbf{f}^t\mathbf{u} + \sum_{j=1}^{m} \alpha_j^2 + \lambda - \lambda\left(\mathbf{u}^t\mathbf{Z}\mathbf{u}\right), \tag{4.25}$$

where the regularization term can be simply folded into the quadratic term in the SSE yielding

$$\min_{\mathbf{u}} \mathbf{u}^t\left(\mathbf{D} - \lambda\mathbf{Z}\right)\mathbf{u} + \mathbf{f}^t\mathbf{u}, \tag{4.26}$$

subject to $\mathbf{C}\mathbf{u} \geq \mathbf{0}$ and $(\mathbf{0}, 1)^t \leq \mathbf{u} \leq \mathbf{1}$.

This is of the form of Tikhonov regularization, where $-\lambda\mathbf{Z}$ is the Tikhonov matrix [122]. As one can clearly see, the Gini-Simpson index does not result in a linear term and the constant is not part of the QP formulation. All that makes it into the quadratic term is $\mathbf{u}^t\mathbf{Z}\mathbf{u}$, our scaled (squared) ℓ_2-norm.

4.5 ℓ_0-Norm Based Regularization of the Shapley Values

The main difficulty behind the ℓ_0-norm of the Shapley values is how do we carry out its optimization? Our QP task with an ℓ_0-norm is a non convex problem, which makes it difficult to understand theoretically and solve computationally (NP-hard problem). There are numerous articles focused on approximation techniques for the ℓ_0-norm. Herein, we take the approach of enhancing sparsity through reweighted ℓ_1 minimization. In [123], Candes proposed a simple and computationally attractive recursively reweighted formulation of ℓ_1-norm minimization designed to more democratically penalize nonzero coefficients. His approach finds a local minimum of a concave penalty function that approximates the ℓ_0-norm. Specifically, the weighted ℓ_1 minimization task can be viewed as a relaxation of a weighted ℓ_0 minimization task.

Definition 18 (SSE with Weighted ℓ_1-Norm)**.** The SSE and weighted ℓ_1-norm of the Shapley index based regularization is

$$E_4 = \mathbf{u}^t\mathbf{D}\mathbf{u} + \mathbf{f}^t\mathbf{u} + \sum_{j=1}^{m} \alpha_j^2 - \left(\lambda_1\mathbf{\Gamma}_1 + ... + \lambda_N\mathbf{\Gamma}_N\right)^t \mathbf{u}. \tag{4.27}$$

Thus, our goal is

$$\min_{\mathbf{u}} \mathbf{u}^t \mathbf{D}\mathbf{u} + \left(\mathbf{f} - (\lambda_1 \mathbf{\Gamma}_1 + ... + \lambda_N \mathbf{\Gamma}_N)\right)^t \mathbf{u}, \tag{4.28}$$

subject to $\mathbf{C}\mathbf{u} + \mathbf{b} \geq \mathbf{0}$ and $(\mathbf{0}, 1)^t \leq \mathbf{u} \leq \mathbf{1}$.

Algorithm 5 is exactly the method of Candes et al. just with the Shapley values as the parameters. For further mathematical analysis of Candes's approximation, see [123]. Herein, our goal is not to advance this approximation technique. Instead, we simply apply it for learning the CI. As better approximations become available, the reader can employ those strategies. In Algorithm 5, $\epsilon > 0$ is used to provide stability and to ensure that a zero-valued component in $1 - \mathbf{\Gamma}_k^t(t-1)\mathbf{u}(t-1)$ does not strictly prohibit a nonzero estimate at the next step (as done in [123]). Intuitively, the update step takes the previous $\lambda_k(t-1)$ terms and divides them by one minus their respective Shapley values. Thus, the "more important" (the larger) the Shapley value the smaller the divisor (number in $[0, 1]$) and therefore the larger the $\lambda_k(t)$. Different stopping criteria exist for Algorithm 5. For example, the user can provide a maximum allowable SSE. The user can also compare the difference between the weights from iteration to iteration relative to a user specified threshold. Furthermore, the user can provide a maximum number of allowable iterations.

Algorithm 5: Weighted Iterative l_1-Norm Regularization

Initialize the weights, e.g., $\lambda_k(t) = 1$, for $1 \leq k \leq N$
Initialize the counter, $t = 1$
while *NOT DONE* **do**
    Solve for $\mathbf{u}(t)$ by minimizing E_4 given $\lambda_k(t)$
    $t = t + 1$
    Update, $\lambda_k(t) = \frac{\lambda_k(t-1)}{\left(1-\mathbf{\Gamma}_k^t(t-1)\mathbf{u}(t-1)\right)+\epsilon}$

4.6 Experiments

In this section, we explore both synthetic and real-world data sets. The goal of the synthetic experiments is to investigate the general behavior of the proposed theories under controlled settings in which we know the "answer" (the generating capacity). The goal of the real-world experiment is to investigate classification performance on benchmark community data sets. In Section 4.2.2 we reviewed and compared, mathematically, different indices. However, we do not include all indices in our experiments as they do not operate on the same basis. Each index more-or-less interprets complexity differently and, thus, each has its own place (application) and rationale for existence, both in terms of capacity theory and also in terms of how the CI is applied. In this section, we restrict analysis to the study of the Gini-Simpson and the ℓ_0-norm of the Shapley values and we compare it to the most related indices for decision-level fusion—specifically, the ℓ_1 and ℓ_2-norm of a lexiographically encoded capacity vector and the Mobius-based index.

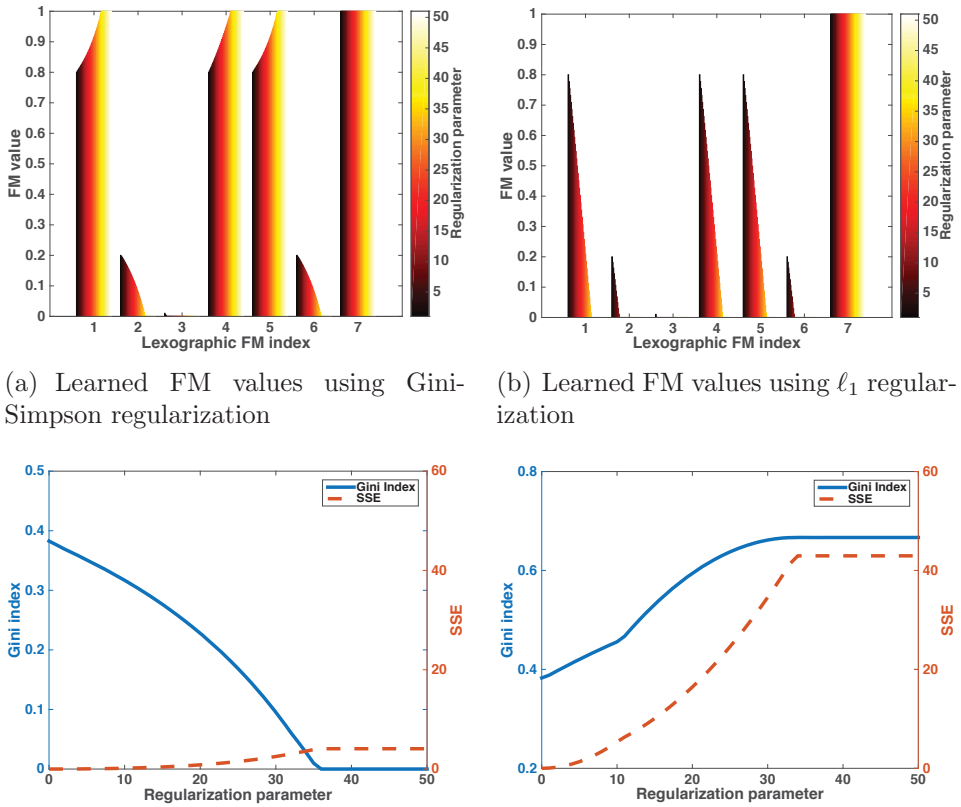
In our synthetic experiments we elected to not report a single summarized number or statistic, e.g., classification accuracy. Instead, we show the behavior of our technique across different possible choices of the regularization parameter λ. While somewhat overwhelming at first, we believe it is important to give the reader a better (more detailed) feel for how the methods behave in general. However, it is worth briefly noting some λ selection strategies used in practice. For example, we can pick a "winner" by trying a range of values of λ in the context of cross validation (i.e., a grid search). Such an experiment emphasizes learning less complex models with respect to the idea of avoiding over fitting (one use of an information theoretic index). We employ the same strategy in our real-world benchmark data set experiment for kernel classification. If the reader desires, they can refer to one of many works in statistics or machine learning for further assistance in automatically determining or experimentally selecting λ [123].

4.6.1 Experiment 1: Important, Relevant and Irrelevant Inputs

For this experiment, we consider the case of three inputs ($N = 3$). While this experiment is easily generalized to more than three inputs, the advantage of $N = 3$ is that we can clearly visualize the results. It becomes difficult to view the results for more inputs as the number of elements in the capacity grows exponentially with

respect to N. We let the worth of the first input to be far greater than the other inputs, $g(x_1) = 0.8$; input two is considered relevant but has a (relatively) low worth, $g(x_2) = 0.2$; and the last input is, for all intents and purposes, *irrelevant*, $g(x_3) = 0.01$. Granted, only densities have been specified. However, we use an S-Decomposable FM, specifically a possibility measure, to determine the value of the capacity terms beyond the densities; $g(A) = \max_{x_i \in A} g(x_i)$, for $A \in 2^X \backslash \{x_1, x_2, ..., x_N, X\}$. Since we use a possibility measure, the above story with respect to the different inputs holds. Additionally, 500 uniform (pseudo)randomly generated samples were used. That is, 500 random N-tuples were generated, each value between $[0, 1]$, and the label was produced using the discussed capacity.

We expect to see the following behavior. We would like for the third input to be ignored and the second input should be driven down to zero *worth* (in the Shapley sense) before the first input. Figure 4.1 shows the results of this experiment. Views (a,b) show the FM values learned for values of λ between 0 and 50—the left side of each bar (the black) corresponds to the learned FM values at $\lambda = 0$ and the right side of each bar (the bright yellow) corresponds to the FM values at $\lambda = 50$. Views (c,d) show the value of the Gini-Simpson index for the learned FM and the resulting SSE versus each value of λ. The scale for the solid blue line—the Gini-Simpson index—is shown on the left of each plot and the scale for the dashed red line—the SSE—is shown on the right of each plot.



(a) Learned FM values using Gini-Simpson regularization

(b) Learned FM values using ℓ_1 regularization

(c) Gini-Simpson regularization performance plots

(d) ℓ_1 regularization performance plots

Figure 4.1: Experiment 1 results. (a,b) Learned FM values in lexicographical order for $\lambda = 0$ to 50. Bin 1 is $\mathbf{u}(1) = g(x_1)$, bin $N+1$ is $\mathbf{u}(N+1) = g(\{x_1, x_2\})$, etc. Height of bar indicates FM value; color indicates λ value. (c,d) Plots showing performance of each regularization method in terms of SSE and Gini-Simpson index of the learned FM at each regularization parameter λ.

Figure 4.1 tells the following story. The black color bars in views (a,b) show that both methods recover the desired possibility measure (the one that minimizes just the SSE criteria) when no regularization $\lambda = 0$ is used—after all, the methods are equivalent, i.e., no regularization, when $\lambda = 0$. View (a) shows that the Gini-Simpson index regularization pushes the contribution of input 3 to zero very quickly—at $\lambda \approx 5$—and the contribution of input 2 is reduced to zero at $\lambda \approx 35$. The contribution

of $\mathbf{u}_6 = g(\{x_2, x_3\})$ is also pushed to zero as λ increases. On the contrary, the contribution of input 1 and the FM values in the lattice that include input 1—i.e., $\mathbf{u}_4$ and $\mathbf{u}_5$—are gradually increased with increasing λ. Figure 4.1(c) shows that as λ is increased the Gini-Simpson index decreases, which is echoed in the FM values shown in view (a). As the model becomes more simple, by increasing λ, the SSE increases (albeit, slightly). At $\lambda \approx 35$, the Gini-Simpson index goes to zero, indicating the model is as simple as it can get. Increasing $\lambda > 35$ has no effect on the model because it is already as simple as possible, with only one input (#1) being considered in the solution. The SSE of this minimum Gini-Simpson index model is about 4.

Figures 4.1(b,d) show visualizations of the same experiment for ℓ_1 regularization. As view (b) shows, this regularization starts decreasing all of the FM values as λ is increased. The contribution of input 3, $\mathbf{u}_3 = g(x_3)$, is quickly pushed to zero, at $\lambda \approx 2$, while the values $\mathbf{u}_2 = g(x_2)$ and $\mathbf{u}_6 = g(\{x_2, x_3\})$ go to zero at $\lambda \approx 10$. Last, $\mathbf{u}_1 = g(x_1)$, $\mathbf{u}_4 = g(\{x_1, x_2\})$, and $\mathbf{u}_5 = g(\{x_1, x_3\})$ go to zero at $\lambda \approx 32$. At $\lambda \gtrsim 32$, the ℓ_1 regularization learns, as expected, the FM of ignorance. Figure 4.1(d) shows that despite a lower complexity model, in terms of ℓ_1-norm, the Gini-Simpson index increases as λ is increased; SSE also increases, as expected. The FM of ignorance learned at $\lambda \gtrsim 32$ has an SSE of about 45. Compare this to the SSE of 4 achieved with the lowest complexity model with Gini-Simpson regularization.

In summary, this initial experiment shows that the Gini-Simpson index regularization

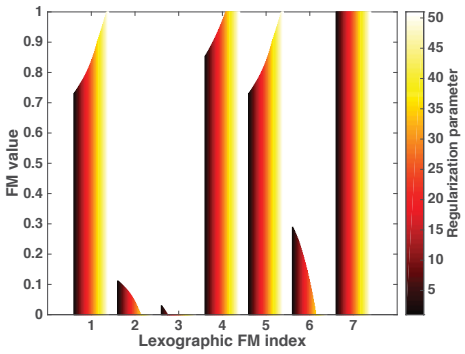
and ℓ_1-regularization of a lexicographically encoded capacity vector do as advertised.

4.6.2 Experiment 2: Random AWGN Noise

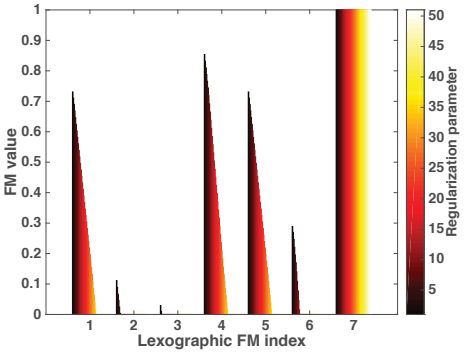
In Experiment 2, we use our setup from Experiment 1; however, (pseudo)random AWGN noise ($\sigma = 0.2$) is added to the labels. Figure 4.2 shows the results of Experiment 2. As views (c,d) show, neither procedure perfectly fits the data now due to the noise in the training labels. The Gini-Simpson procedure, shown in views (a,c), can find a solution close to our noise-free goal at small values of λ. If regularization is increased, $\lambda \gtrsim 45$, we eventually identify a single input, which interestingly still fits the data well (only a small increase in SSE). Again, the ℓ_1 procedure is only able to achieve low SSE at low λ values. As λ is increased the SSE is significantly increased (beyond that achieved by the Gini-Simpson).

4.6.3 Experiment 3: Iteratively Reweighted ℓ_1-Norm

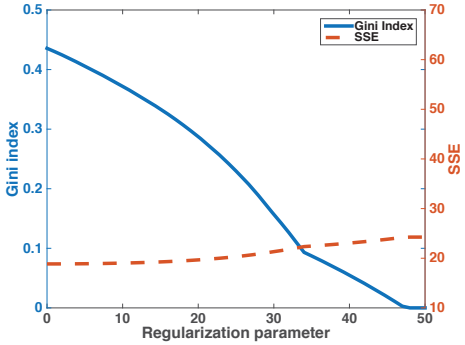
In Experiment 3, we use our setup from Experiment 1 to demonstrate the recursively reweighted ℓ_1 minimization procedure. The result (Figure 4.3(a,b)) for the possibility FM with densities $g(x_1) = 0.8$, $g(x_2) = 0.2$, $g(x_3) = 0.01$, is as expected. After a few iterations we see the Shapley value increasing for input x_1 and decreasing for inputs



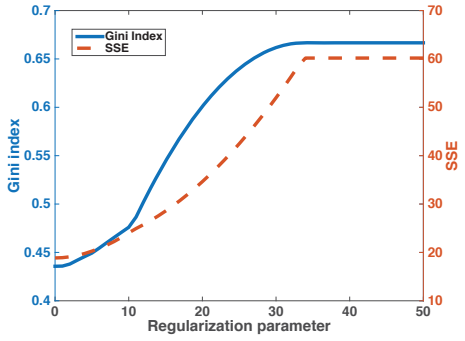
(a) Learned FM values using Gini-Simpson regularization



(b) Learned FM values using ℓ_1 regularization



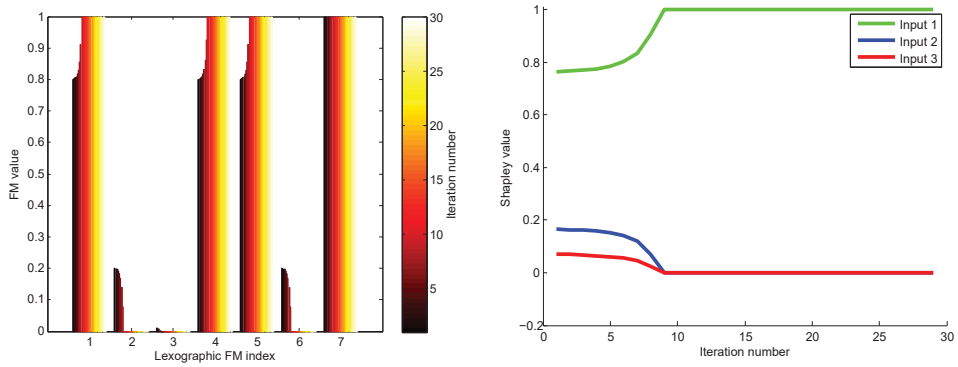
(c) Gini-Simpson regularization performance plots



(d) ℓ_1 regularization performance plots

Figure 4.2: Experiment 2 results. (a,b) Learned FM values in lexicographical order. Bin 1 is $\mathbf{u}(1) = g(x_1)$, bin $N+1$ is $\mathbf{u}(N+1) = g(\{x_1, x_2\})$, etc. Height of the bar indicates FM value; color indicates λ value. (c,d) Plots showing performance of each regularization method in terms of SSE and Gini-Simpson index values of the learned FM at each regularization parameter λ.

x_2 and x_3. This is the same trend and final answer that we saw in Experiment 1 with respect to the Gini-Simpson index and we obviously get a different final solution than the ℓ_1 with respect to the lexographically encoded capacity vector (eventual solution of 0s and corresponding CI minimum operator).



(a) Learned FM values using iteratively reweighted ℓ_1 regularization for Experiment 1

(b) Shapley values for (a)

Figure 4.3: Experiment 3 results. Learned FM values in lexicographical order for Experiment 1. Bin 1 is $\mathbf{u}(1) = g(x_1)$, bin $N+1$ is $\mathbf{u}(N+1) = g(\{x_1, x_2\})$, etc. Height of the bar indicates FM value; color indicates iteration number. Plot of the Shapley values of the learned FM for Experiment 1 at each iteration.

4.6.4 Experiment 4: Multiple Kernel Learning

In this final experiment we consider a problem from pattern recognition. Kernel methods for classification is a well-studied field in which data are implicitly mapped from a lower-dimensional space to a higher-dimensional space, called the *reproducing kernel Hilbert space* (RKHS), to improve classification accuracy. The ultimate challenge is that we are not privileged to know what transform (kernel) solves a particular task at hand—we only have an existence theorem. *Multiple kernel learning* (MKL) is a way to learn the fusion of multiple known Mercer kernels (the building blocks) to identify a superior kernel. In [1, 2, 124, 125], a *genetic algorithm* (GA) based ℓ_p-norm linear convex sum of kernels called GAMKLp for feature-level fusion was proposed. In [1],

the nonlinear fusion of kernels was also explored. Specifically, kernel classifiers were fused at the decision-level based on the fuzzy integral, a procedure called *decision-level fuzzy integral MKL* (DeFIMKL). In this experiment, we explore the use of QP learning and regularization for CI-based MKL in the context of *support vector machine* (SVM) classification with respect to well-known community benchmark data sets. In [1], the benefit of DeFIMKL and GAMKL was demonstrated versus other state-of-the-art MKL algorithms from machine learning, e.g., *MKL group lasso* (MKLGL). Herein, the goal is not to reestablish DeFIMKL but to explore the proposed indices and their relative performances. Note, in the other experiments we knew the answer, i.e., the "generating capacity". However, while SVMs are supervised learners and our data has labels, we do not know the true capacity in the case of MKL. Herein, like often in machine learning, success is instead evaluated in terms of ones ability to improve classification performance. The fusion of classifiers via DeFIMKL results in a classifier and this experiment demonstrates the ability of regularization to help learn an improved classifier that does not succumb to overfitting and generalizes better.

Each learner, i.e., input to fusion, is a kernel classifier trained on a separate kernel. The kth ($1 \leq k \leq N$) SVM classifier decision value is

$$\eta_k(\mathbf{x}) = \sum_{i=1}^{D} \alpha_{ik} y_i \kappa_k(\mathbf{x}_i, \mathbf{x}) - b_k,$$

which is the distance of $\mathbf{x}$ (an object from T) from the hyperplane defined by the

data labels, y, the kth kernel, $\kappa_k(\mathbf{x}_i, \mathbf{x})$, and the learned SVM model parameters, α_{ik} and b_k. For the two-class SVM binary decision problem, the class label is typically computed as $\text{sgn}\{\eta_k(\mathbf{x})\}$. One could use $\text{sgn}\{\eta_k(\mathbf{x})\}$ as the training input to the capacity learning, however this eliminates information about which kernel produces the largest class separation—essentially, the difference between $\eta_k(\mathbf{x})$ for classes labeled $y = +1$ and $y = -1$. In [1] $\eta_k(\mathbf{x})$ is remapped onto the interval $[-1, +1]$, creating the inputs for learning by the sigmoid function $\frac{\eta_k(\mathbf{x})}{\sqrt{1+\eta_k^2(\mathbf{x})}}$. For training, we use our labeled data and cast the learning process as a SSE problem and the CI is learned using QP and regularization (see [1] for a full mathematical description).

The well-known LIBSVM library was used to implement the classifiers [76]. The machine learning UCI benchmark data sets used are sonar, dermatology, wine, ecoli and glass. Each experiment consists of 100 randomly sampled trials in which 80% of the data is used for training and the remaining 20% is sequestered for testing. Each index was applied to the same random sample to guarantee a level playing field. Note that in some cases multiple classes are joined together such that the classification decision is binary. Five *radial basis kernels* (RBF) kernels are used in each algorithm with respective RBF width σ linearly spaced on the interval defined in Table 4.2; the same RBF parameters are used for each algorithm. For the ℓ_1, ℓ_2, Gini-Simpson and k-additive indices, a dense grid search (of λ) was used and the "winner" was picked according to the highest classification accuracy on the test data. For the iteratively reweighted ℓ_1 approximation, we used Algorithm 5. Table 4.3 is the result

Table 4.2
RBF Kernel Parameter Ranges and Data Set Properties

	Data Set		
	Sonar	**Dermatology**	**Wine**
Parameter Ranges	[-2.2, -0.2]	[-2.3, 0]	[-6,-3]
No. of Objects	208	366	178
No. of Features	60	33	13
Binary Classes	$\{1\}$ vs. $\{2\}$	$\{1,2,3\}$ vs. $\{4,5,6\}$	$\{1\}$ vs. $\{2,3\}$
	Ecoli	**Glass**	
Parameter Ranges	[-3, 3]	[-2, 2]	
No. of Objects	336	214	
No. of Features	7	9	
Binary Classes	$\{1,2,3,4\}$ vs. $\{5,6,7,8\}$	$\{1,2,3\}$ vs. $\{4,5,6\}$	

Table 4.3
Classifier Performances—Means and Standard Deviations

	Sonar	**Dermatology**	**Wine**
No Regularization	80.43 (9.25)	94.51 (3.89)	93.00 (9.02)
Lexicographic ℓ_1	86.52 (7.55)	94.57 (3.91)	93.44 (8.52)
Lexicographic ℓ_2	86.43 (7.42)	94.57 (3.91)	94.00 (8.27)
Gini-Simpson	87.14 (6.98)	**98.22 (2.15)**	94.22 (7.97)
Shapley ℓ_0 Approximation	**87.38 (6.98)**	97.76 (2.40)	**94.56 (7.65)**
$k = 2$ additive	84.90 (7.63)	94.57 (3.91)	93.78 (8.25)
$k = 3$ additive	85.67 (7.49)	94.57 (3.91)	93.89 (8.26)
$k = 4$ additive	86.48 (7.37)	94.92 (3.81)	94.00 (8.27)
$k = 5$ additive	86.48 (7.37)	94.92 (3.81)	94.00 (8.27)
	Ecoli	**Glass**	
No Regularization	96.71 (2.90)	94.33 (5.23)	
Lexicographic ℓ_1	96.71 (2.90)	**96.33 (4.73)**	
Lexicographic ℓ_2	96.71 (2.90)	96.00 (4.82)	
Gini-Simpson	**97.15 (2.63)**	96.14 (4.67)	
Shapley ℓ_0 Approximation	97.12 (2.71)	96.05 (4.59)	
$k = 2$ additive	96.71 (2.90)	96.19 (4.69)	
$k = 3$ additive	96.71 (2.90)	95.52 (4.78)	
$k = 4$ additive	96.71 (2.90)	95.38 (4.90)	
$k = 5$ additive	96.71 (2.90)	95.38 (4.90)	

of regularization on DeFIMKL.

Table 4.3 tells the following story. First, in each instance regularization helps. In

many instances, e.g., ecoli, glass and wine, the regularization results are extremely close. However, in other cases, e.g., sonar and dermatology, the regularization results vary more (both in terms of means and standard deviations). Note, we ran the k-additive index with different levels of *forced* k-additivity. This was done to explore the impact of assuming and working with subsets of the capacity. In our other experiments we were able to analyze specific conditions and properties relating to the fusion process. While this experiment is encouraging, i.e., better classification performance, we are sadly unable to connect a story to the results. The regularization results are what they are. We cannot go the next step and inform the reader why a Gini-Simpson or k-additive index is more well-suited given our limited knowledge about the machine learning classification task.

4.7 Conclusion and Future Work

In this chapter, we explored a new data-driven way to learn the CI in the context of decision-level fusion relative to the joint minimization of the SSE criteria and desire to obtain minimum model complexity. We brought together and analyzed a number of existing indices, put forth new indices based on the Shapley values, and explored their role in regularization-based learning of the CI. Our first proposed index promotes sparsity (specifically, fewer number of non-zero parameters), however it is complicated to optimize (NP-hard). Our second index is a tradeoff with respect to

modeling accuracy relative to solution simplicity. The proposed indices and regularization approach are compared theoretically. We showed that there is no "winning index", as these indices strive for different goals and are therefore valid in different contexts. Synthetic and real-world data set experiments are shown that demonstrate the benefits of the proposed indices and CI learning technique.

In future work, we will seek more efficient and scalable ways to solve the problem investigated here as the number of inputs (N) grows—since the number of capacity terms and subsequently associated monotonicity constraints increases at an exponential rate. We will also explore if there are other information theoretic measures that have additional benefit towards learning lower complexity and useful CIs.

Chapter 5

Applications to Explosive Hazard Detection with Ground Penetrating Radar

The material in this chapter was previously published in: Proc. SPIE, pp. 98230T, 2016; Proc. SPIE, pp.94540B, 2015; and Proc. IEEE CISDA, pp. 1-8, 2016.

5.1 Introduction

Buried explosive hazards are one of the greatest threats to life in modern combat, and also pose dangers to civilian populations in former war zones. Every month, there are over 300 casualties due to these explosive devices, on average, and they wound an additional 800+ [126]. *Ground penetrating radar* (GPR) is a common tool used to detect these hazards and comes in two flavors: forward-looking and downward-looking. Both systems have their benefits and drawbacks. For example, forward-looking systems have the advantage that they offer greater standoff distances between the radar system and the targets compared to downward-looking systems. Downward-looking GPR, on the other hand, is able to receive much more reflected radar energy due to the engagement geometry and thus typically have higher *signal-to-noise ratios* (SNR)[1]; the data used in this chapter are from a downward-looking system.

This chapter begins with an exploration of various *robust principal component analysis* (RPCA) techniques for preprocessing the GPR data in Part I. After showing that RPCA can increase the *signal-to-clutter ratio* (SCR), Part II applies state-of-the-art feature-level fusion algorithms to GPR data along with details of the full data-processing pipeline. Finally, Part III applies some of the fusion algorithms introduced

[1]Most of the energy emitted from forward-looking systems reflects off the scene *away* from the radar receivers.

in previous chapters to another dataset derived from the same GPR system.

Part I

A Comparison of Robust Principal Component Analysis Techniques for Buried Object Detection in Downward Looking GPR Sensor Data

5.2 Classical Principal Component Analysis (PCA) and Robust Principal Component Analysis (RPCA)

Suppose we have a data matrix M that is corrupted by noise. Since data generally lie on a low-dimensional subspace, we can model the noisy data matrix as $M = L + N$, where L is a low-rank matrix and N is a (sparse) perturbation matrix. In this example, we seek to find the low-rank matrix L as our estimate of the underlying data. Mathematically, the problem is

$$\begin{aligned} \underset{L}{\text{minimize}} \quad & \|M - L\|_2 \\ \text{subject to} \quad & \text{rank}(L) \leq k, \end{aligned} \tag{5.1}$$

which can be solved via classical principal component analysis[127]. While this is a very popular technique since it works well when the noise component is small, it simply breaks down if the magnitude of the noise is too large. Figures 5.1 and 5.2 show the behavior of PCA in low and high noise applications. Figure 5.1(a) presents a clean image, to which we add a small amount of noise as shown in Figure 5.1(b). The low-rank and sparse components are then computed using PCA in Figures 5.1(c)—5.1(d). In this case, the low-rank component is reasonably recovered from the noisy

image, though there is obviously some loss in quality. Figure 5.2(b) shows the image corrupted by ten times more noise. Note that the low-rank component computed with classical PCA shown in Figure 5.2(c) no longer accurately estimates the original image—it is completely lost in the noise.

The fragility of classical PCA can also be highlighted by gross contamination of a *single* pixel. Consider the same original image in Figure 5.1(a). We select a random pixel in this image and assign it an arbitrary value of 10 (the pixels in the original image are normalized in the range of $[0, 1]$). In this case, the low-rank image we recover using PCA loses essentially all of its contrast as shown in Figure 5.3(a).

The examples shown in this section demonstrate the need for a PCA method that is robust to large noise and outliers. The following section will introduce a method that aims to achieve this robustness by modifying the problem in (5.1).

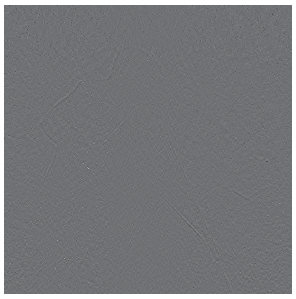
5.2.1 Robust Principal Component Analysis (RPCA)

The groundwork for RPCA was developed by Candès et al. in their seminal paper *Robust principal component analysis?*[128], where they proposed a new problem—essentially a modification to the problem in (5.1). In their new framework, the data matrix is the superposition of a low-rank matrix L and a sparse matrix S, or $M = L + S$. Note that this assumption is equivalent to that made in classical PCA, though

(a) Original image.

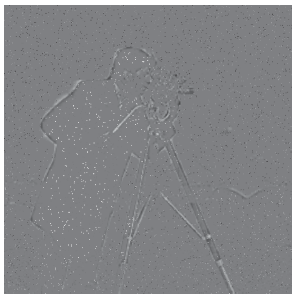
(b) Noisy image, small noise.

(c) Low-rank image via PCA.



(b) Sparse image via PCA.

(e) Low-rank image via RPCA.



(b) Sparse image via RPCA.

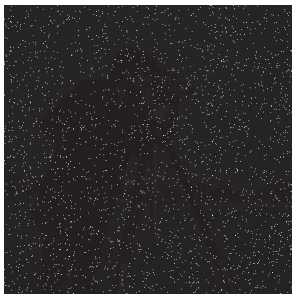
Figure 5.1: An image contaminated with a small amount of noise and its low-rank and sparse decomposition.

the sparse component is no longer referred to as noise[2]. To decompose the data matrix into L and S, Candès et al. propose the new *convex* minimization problem,

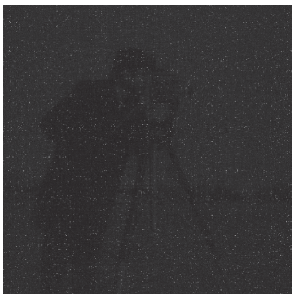
$$\begin{aligned} \underset{L,S}{\text{minimize}} \quad & \|L\|_* + \lambda \|S\|_1 \\ \text{subject to} \quad & L + S = M, \end{aligned} \tag{5.2}$$

[2]In many cases, we are actually concerned with the sparse component.

(a) Original image.



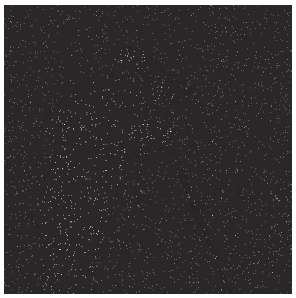
(b) Noisy image, large noise.



(c) Low-rank image via PCA.

(b) Sparse image via PCA.

(e) Low-rank image via RPCA.



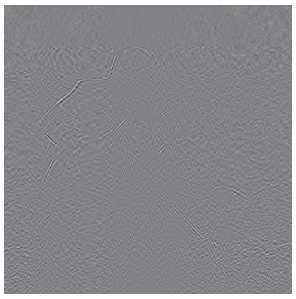
(b) Sparse image via RPCA.

Figure 5.2: An image contaminated with a large amount of noise and its low-rank and sparse decomposition.

where $\|L\|_* = \sum_i \sigma_i(L)$ is the nuclear norm[3] of the matrix L, i.e., the sum of the singular values of L, and $\|S\|_1 = \sum_{ij} |S_{ij}|$ is the ℓ_1-norm of L when seen as a long vector. Candès et al. propose an algorithm called *principal component pursuit by alternating directions* (PCP-AD) that *exactly* recovers the low-rank and sparse components of M, given that weak conditions are met. Empirically, we have shown that PCP still

[3]The nuclear norm is the convex envelope of the rank operator [129].

(a) Low-rank image via PCA.



(b) Sparse image via PCA.

(c) Low-rank image via RPCA.

(b) Sparse image via RPCA.

Figure 5.3: The low-rank and sparse decomposition of an image with only one pixel contaminated with a large amount of noise.

tends to work well even if the conditions are not satisfied [5, 130].

The main difference between the classical PCA problem in (5.1) and the RPCA problem in (5.2) is the promotion of sparsity in the S matrix from the ℓ_1-regularization term in (5.2). The RPCA formulation still encourages L to be low-rank via the nuclear norm (in the PCA problem this is instantiated as a constraint), and it forces L to lie "close" to M (the PCA problem uses this as the cost).

The behavior of RPCA is shown in Figures 5.1–5.3. In all three cases, we see that the RPCA decompositions of the noisy images are better than those of the classical PCA, though in the high-noise case of Figure 5.2 the low-rank component loses some

contrast.

The remainder of this section briefly introduces the RPCA algorithms used in this work. Their derivations and the underlying mathematics are much too involved to reproduce here, though we highlight the salient differences in each approach and cite the original works where the methods were proposed.

Fast PCP [131] modifies the problem in (5.2) by swapping the nuclear norm penalty with the constraints to obtain

$$\begin{aligned} &\underset{L,S}{\text{minimize}} \quad \|L+S-M\|_F + \lambda\|S\|_1 \\ &\text{subject to} \quad \text{rank}(L) = t. \end{aligned} \tag{5.3}$$

The minimization problem in (5.3) is then solved via the alternating minimization

$$\begin{aligned} L_{k+1} = &\underset{L}{\text{minimize}} \quad \|L+S_k-M\|_F + \lambda\|S\|_1 \\ &\text{subject to} \quad \text{rank}(L) = t; \end{aligned} \tag{5.4a}$$

$$S_{k+1} = \underset{S}{\text{minimize}} \quad \|L_{k+1}+S-M\|_F + \lambda\|S\|_1, \tag{5.4b}$$

which is shown to monotonically converge to the desired solution and requires low memory allowing for real-time implementation.

Robust PCA via Outlier Pursuit (OP-RPCA)[132] assumes the data matrix

M can be written as

$$M = L + C_0, \tag{5.5}$$

where L is a low-rank matrix and C_0 is non-zero in a fraction of the columns. This method relies on finding a basis that spans the low-dimensional subspace in which L lies, and the algorithm has been shown that it converges to the correct basis under weak assumptions. Additionally, OP-RPCA has been shown to work in the noisy case where the matrix in (5.5) is corrupted by an additional noise term.

Augmented Lagrange Multiplier (ALM)[133], like other RPCA methods, assumes the data matrix has the form $M = L + S$. It recovers the low rank and sparse components using

$$\begin{aligned} \underset{L,S}{\text{minimize}} \quad & \|L\|_* + \kappa\,(1-\lambda)\,\|L\|_{2,1} + \kappa\lambda\|S\|_{2,1} \\ \text{subject to} \quad & L + S = M, \end{aligned} \tag{5.6a}$$

where

$$\|X\|_{2,1} = \sum_j \|X_j\|_2. \tag{5.6b}$$

ℓ_1-Filtering (L1F) is able to exactly solve the nuclear norm RPCA problem in (5.2) in linear time [134]. It assumes the underlying matrix is low-rank enough to be accurately approximated using a much smaller submatrix. Once this "seed" matrix is defined, all other block matrices comprising the low-rank and sparse components

are computed using straightforward linear algebra approximations and much smaller minimization problems.

Active Subspace RPCA (ASRPCA) solves (5.2) by factoring the large matrix M into two smaller matrices, one of which is orthonormal, known as the active subspace [135]. This allows the problem to be solved more efficiently, and thus opens the door for RPCA to be applied to large matrices.

Variational Bayesian RPCA (VBRPCA) was proposed by Babacan et al.; it parameterizes the low-rank matrix and assigns Gaussian priors to all latent variables [136]. From this fully defined Bayesian model, they use variational Bayesian inference to approximate the low-rank factors, sparse component, and the hyperparameters repeatedly until convergence.

5.3 Ground Penetrating Radar

The data used in this chapter were collected by GPR sensors on an experimental hand held demonstrator. This demonstration unit allows the GPR sensors to sweep over the ground, covering approximately one meter per sweep, while incrementing forward at approximately 1 meter per second down experimental test lanes. Each sensor collects 256 samples from the radar return with a sampling rate of approximately 32

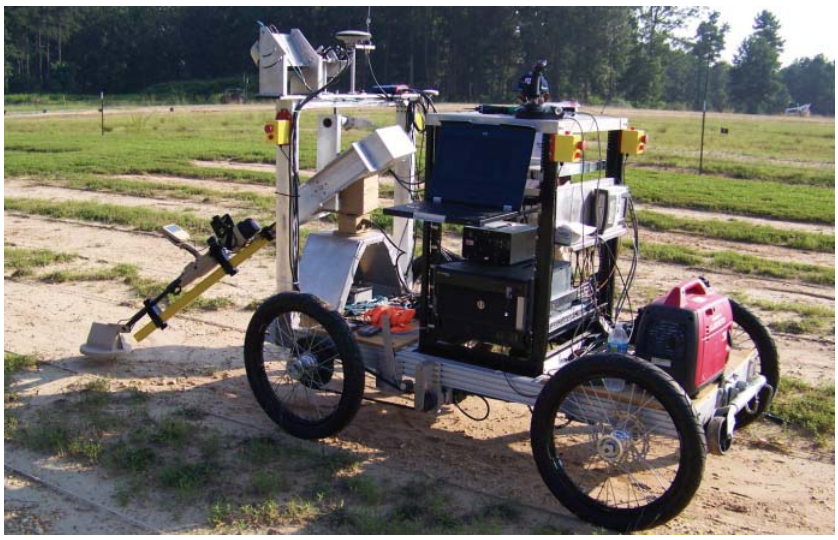
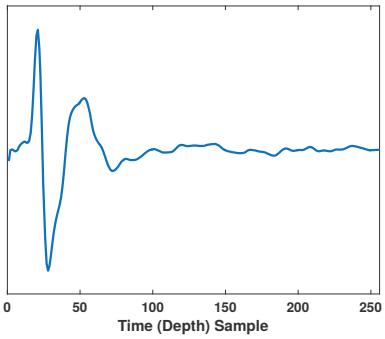


Figure 5.4: Experimental Hand Held Demonstrator

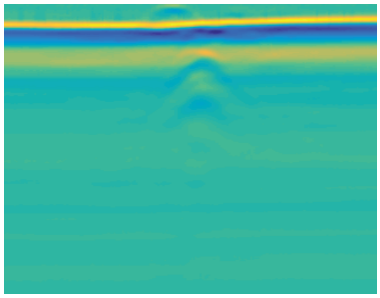
GHz, and each radar return is labeled with a location stamp using differential GPS. Figure 5.4 shows the hand held demonstrator during a test run.

5.3.1 Data Format and Visualization

The individual radar returns are known as an *A-scan*. Each A-scan is a time series of 256 samples of the returned radar intensity. A group of A-scans representing adjacent returns are often rotated vertically, stacked next to one another, and plotted in grey-scale or a false-color image to form what is known as a *B-scan*. The B-scans tend to be particularly useful in the detection of subsurface objects because of their ability to highlight objects with a hyperbolic signature. The plots in Figure 5.5 show an example of each type of scan. The A-scan plotted in Figure 5.5(a) shows a large



(a) GPR A-scan



(b) GPR B-scan

Figure 5.5: Examples of typical GPR data visualization

return within the first 50 time samples corresponding to the air/ground interface and relatively small returns thereafter. If a large target is present below the ground, we would expect to see another large return in a group of time bins sometime after the ground reflection, though due to large amounts of clutter and signal attenuation the reflections from buried targets are often lost in noise. Figure 5.5(b) shows an example B-scan representing a slice of earth in which a target is buried. Note the faint hyperbolic signature of the target near the center of the image. Though the hyperbolic signature in this example is apparent, often times they are hidden, again due to large amounts of clutter, signal attenuation, and large ground reflections.

5.3.2 RPCA Decomposition of GPR Data

Section 5.2.1 showed the advantages of using RPCA as a means to decompose an image into a low-rank component and a sparse component. Under this decomposition,

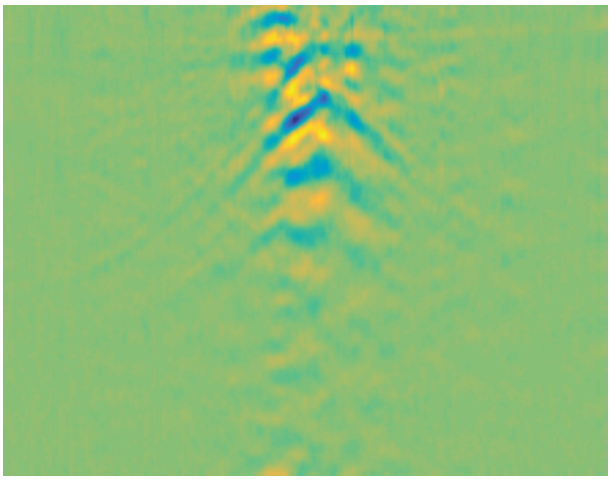


Figure 5.6: Sparse component of GPR B-scan from Figure 5.5(b), computed using RPCA

buried targets should be easily seen in the sparse component of the B-scans since the background of the B-scan, i.e., the ground reflection and some constant clutter, will be removed as the low-rank component. Indeed this is often the case as shown in Figure 5.6, which shows the sparse component of the B-scan in Figure 5.5(b). The target appears to be separated from the rest of the image by much more of a contrast after applying RPCA, suggesting that RPCA can boost the performance of detection algorithms.

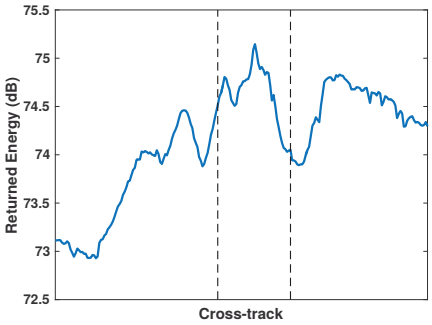
5.3.3 Returned Energy and Signal to Clutter Ratio

The total energy return of a particular A-scan, $x[n]$, is computed as

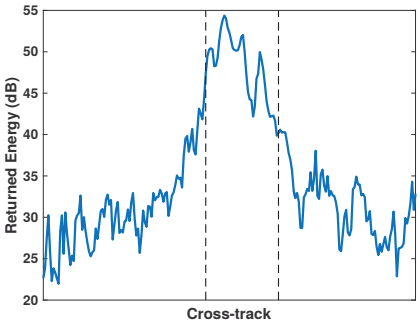
$$E = \frac{1}{N} \sum_{n=1}^{N} x^2[n], \tag{5.7}$$

where N is the number of samples in the A-scan. When a target is below the radar sensor, the reflected energy increases, thus detection can be performed by monitoring this returned energy. Additional value lies in computing the returned energy for each A-scan comprising a B-scan; the B-scan's energy can be plotted to represent the amount of returned energy at each discrete location while sweeping across a lane, making the presence of a target more apparent. Examples of these plots are shown in Figure 5.7, where the two vertical lines indicate the target region—the region around a target's ground truth location. Note that the plots correspond to the same physical B-scan location, though Figure 5.7(a) corresponds to the energy derived from the original B-scan, and Figure 5.7(b) corresponds to the energy derived from the original B-scan's sparse component shown in Figure 5.6.

We have proposed CFAR detection algorithms based on returned energy in GPR data in previous work [5, 130]. These types of detectors generally make their decisions by comparing the energy in some region with the energy in the surrounding region, thus



(a) Returned energy of B-scan in Figure 5.5(b)



(b) Returned energy of B-scan in Figure 5.6

Figure 5.7: Returned energy for a single sweep across a lane, over a target. Vertical lines indicate the region in which the target is buried.

the *signal-to-clutter ratio* (SCR) can be used as an indication of the performance of the detector. For the experiments in this chapter we use the peak SCR as the performance metric, defined as

$$SCR = 10 \log_{10} \left(\frac{S_{pk}}{\mathrm{E}\left[C\right]} \right), \tag{5.8}$$

where S_{pk} is the maximum returned energy in the target region, and $\mathrm{E}\left[C\right]$ is the mean of the clutter energy received outside of a 25-sample guard band on either side of the target region. Figure 5.8 shows the quantities used to compute the SCR; the

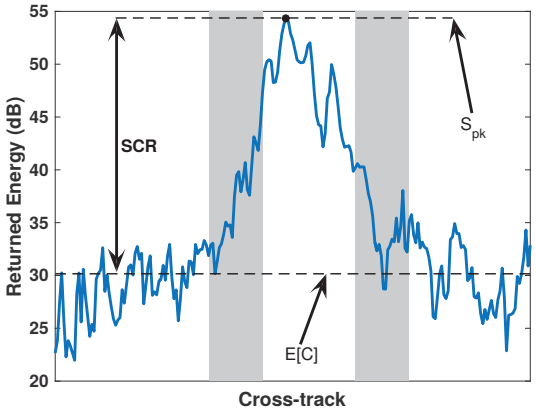


Figure 5.8: Example calculation of the peak SCR

guard bands are shaded, and the maximum signal return, S_{pk}, and mean of the clutter energy (outside of the guard bands), $\mathrm{E}[C]$, are indicated. In this example, the SCR is 24.2 dB.

5.4 RPCA Experiments

The experiments in this work were designed to explore the impact of the various RPCA algorithms on SCR, as well as the effects of any tuning parameters required by the algorithms. B-scans of a group of 83 unique targets were formed using the raw GPR data and the peak SCR for each B-scan was computed, giving a benchmark performance metric. RPCA algorithms[137] were then applied to the B-scans and the new SCRs were computed to show how RPCA affects the SCR. The algorithms were applied many times while varying the tuning parameters, and they were

also implemented in both the spatial domain and 1-D frequency domain, i.e., the Fourier transform of each individual A-scan, to further investigate the differences in performance.

5.4.1 Overall Results

The average SCR over all targets in the dataset is shown in Table 5.1; the standard deviations of the SCR values over the 83 targets are shown in parentheses. Bold values in the table indicate the best performance according to a two-sided t-test at the 95% confidence interval. It is clear that essentially all RPCA algorithms are able to increase the SCR, though the improvement using VBRPCA is marginal. The overall winner is the L1F algorithm, which achieves an SCR of 13.82 in the time domain; its performance breaks down when implemented in the frequency domain. PCP-AD and OP-RPCA both do well in either the time or frequency domain, though PCP-AD is the superior algorithm in either case. Another interesting note, which is consistent in the subsequent target type-specific results, is that OP-RPCA performs equally well in either domain. The Fast PCP and ALM algorithms are both able to increase SCR when implemented in either domain, though their improvement is far inferior to the other more successful method. Finally, the ASRPCA algorithm is able to increase the SCR considerably, though only when implemented in the time domain—frequency domain implementation does not seem to affect the SCR.

Table 5.1
Average SCR over all targets*

RPCA Algorithm	Domain		Lambda
	Time	Frequency	
None	0.88 (1.01)	—	—
PCP-AD	**12.48** (8.01)	**13.47** (5.45)	0.2
Fast PCP	3.94 (3.96)	2.56 (3.94)	1
OP-RPCA	11.55 (5.28)	11.55 (5.28)	0.3
ALM	4.04 (3.59)	4.04 (3.59)	—
L1F	**13.82** (5.09)	0.57 (0.90)	—
ASRPCA	9.45 (5.60)	0.88 (1.00)	0.05, 0
VBRPCA	0.89 (1.01)	1.05 (1.67)	—

*Standard deviation of SCR shown in parentheses. Bold indicates best performance according to t-test at 95% confidence interval.

5.4.2 Results Based on Target Type

Tables 5.2–5.5 show the performance of the RPCA-processed B-scans for the four target types: wires, landmines, pressure plates, and main charges. The overall trends in these tables are generally all similar to those in Table 5.1, though there are some exceptions that will be discussed.

PCP-AD implemented in the time domain achieves the highest SCR for wire targets. It is followed in close second by the L1F algorithm, with similar results shown for PCP-AD's implementation in the frequency domain. Once again, OP-RPCA and ASRPCA both achieve a decent SCR increase, and VBRPCA's effect on the SCR is marginal in either domain.

Table 5.2
Average SCR over all wire targets*

RPCA Algorithm	Domain		Lambda
	Time	Frequency	
None	0.69 (0.87)	—	—
PCP-AD	**11.50** (10.18)	**10.55** (5.89)	0.2, 0.1
Fast PCP	1.38 (3.87)	1.38 (3.85)	1
OP-RPCA	**8.34** (4.49)	**8.34** (4.49)	0.3
ALM	3.91 (3.24)	3.91 (3.24)	—
L1F	**10.95** (4.00)	0.54 (0.83)	—
ASRPCA	**6.44** (5.19)	0.69 (0.87)	0.05, 0
VBRPCA	0.69 (0.88)	0.73 (0.97)	—

*Standard deviation of SCR shown in parentheses. Bold indicates best performance according to t-test at 95% confidence interval.

Table 5.3
Average SCR over all landmine targets*

RPCA Algorithm	Domain		Lambda
	Time	Frequency	
None	0.78 (0.89)	—	—
PCP-AD	**11.50** (7.92)	**16.38** (6.10)	0.2, 0.15
Fast PCP	2.19 (3.77)	2.17 (3.75)	1
OP-RPCA	11.74 (5.24)	11.74 (5.24)	0.3
ALM	3.81 (3.60)	3.81 (3.60)	—
L1F	**13.91** (4.92)	0.47 (0.74)	—
ASRPCA	9.43 (5.51)	0.78 (0.89)	0.05,0
VBRPCA	0.78 (0.89)	0.95 (1.43)	—

*Standard deviation of SCR shown in parentheses. Bold indicates best performance according to t-test at 95% confidence interval.

Over all landmine targets, PCP-AD in the frequency domain reigns as the best algorithm. OP-RPCA and L1F both do significantly better with landmine targets when compared to wire targets, though this trend is generally exhibited by the other algorithms likely due to the increased energy return from the larger landmine targets.

Pressure plate targets generally achieve some of the highest SCRs over all targets

Table 5.4
Average SCR over all pressure plate targets*

RPCA Algorithm	Domain		Lambda
	Time	Frequency	
None	1.19 (1.68)	—	—
PCP-AD	**17.64** (11.58)	**15.02** (5.59)	0.2, 0.1
Fast PCP	5.34 (4.16)	5.31 (4.15)	1
OP-RPCA	**14.09** (6.18)	**14.09** (6.18)	0.3
ALM	5.31 (3.92)	5.31 (3.92)	—
L1F	**16.17** (6.94)	0.84 (1.57)	—
ASRPCA	**11.78** (6.08)	1.19 (1.68)	0.05, 0
VBRPCA	1.19 (1.68)	1.52 (3.35)	—

*Standard deviation of SCR shown in parentheses. Bold indicates best performance according to t-test at 95% confidence interval.

when using RPCA. PCP-AD achieves the highest SCR of the experiments when implemented in the time domain for these targets. L1F implemented in the time domain achieves the second best performance, though PCP-AD in the frequency domain and the OP-RPCA algorithm achieve approximately similar results. Once more to no surprise the VBRPCA algorithm achieves negligible performance increase.

Main charge targets, like pressure plates, achieve high SCR increases because they are relatively large targets, thus the returned energy is greater. The PCP-AD algorithm outperforms all others in this category, with a considerable lead on the second place algorithm, L1F. Interestingly, the PCP-AD algorithm achieves the experiment's highest SCR in this category, though the same cannot be said for any of the other algorithms.

Table 5.5
Average SCR over all main charge targets*

RPCA Algorithm	Domain		Lambda
	Time	Frequency	
None	1.29 (0.85)	—	—
PCP-AD	**15.28** (12.84)	**17.72** (7.03)	0.25, 0.15
Fast PCP	3.28 (3.85)	3.27 (3.83)	1
OP-RPCA	**11.26** (4.19)	11.26 (4.19)	0.35
ALM	4.38 (3.33)	4.38 (3.33)	—
L1F	**13.74** (4.06)	0.86 (0.92)	—
ASRPCA	**9.99** (5.09)	1.29 (0.85)	0.05, 0
VBRPCA	1.30 (0.86)	1.43 (1.26)	—

*Standard deviation of SCR shown in parentheses. Bold indicates best performance according to t-test at 95% confidence interval.

5.4.3 Decomposition Time

Table 5.6 shows the results of our experiment to compare the time to compute the sparse component of a single B-scan. Note that the times given for the algorithms in the frequency domain include the time for the FFT and inverse FFT. Fast PCP lives up to its name and is the fastest algorithm in this experiment. The next fastest algorithm is ASRPCA implemented in the frequency domain because the decomposition converged almost instantly, though its results are far inferior to the other algorithms. The fastest implementations of ALM and VBRPCA are able to compute the sparse component in about one second. L1F clocks in at just over a second for the time domain implementation, and PCP-AD in the same domain takes about a half-second longer. Finally, OP-RPCA exhibits the longest decomposition time when implemented in the frequency domain at almost 11 seconds.

Table 5.6
Average RPCA decomposition time per sweep in seconds*

	Domain	
RPCA Algorithm	Time	Frequency
PCP-AD	1.876 (0.658)	3.899 (1.348)
Fast PCP	0.027 (0.010)	0.048 (0.025)
OP-RPCA	4.914 (2.457)	10.809 (5.470)
ALM	0.919 (0.447)	1.993 (0.115)
L1F	1.242 (0.073)	0.109 (0.012)
ASRPCA	5.299 (2.913)	0.054 (0.289)
VBRPCA	0.887 (1.009)	1.049 (1.675)

*Standard deviation of time shown in parentheses.

5.4.4 Effect of Parameter Selection on Select Algorithms

Not all algorithms use a tuning parameter, though of the ones that do PCP-AD and OP-RPCA both achieve significant increases in SCR. This section presents the effects that the tuning parameter has on the sparse component and the SCR for these algorithms.

Figure 5.9 shows an unprocessed target B-scan to which PCP-AD and OP-RPCA are applied. The target is visible in the unprocessed image, though it is quite subtle. The results for the PCP-AD algorithm are summarized in Figure 5.10, which shows the processed B-scans and their integrated energies for four different choices of λ. Note that the PCP-AD method was applied in both the time and frequency domains as specified in the captions of the figure.

Figure 5.10 highlights how increasing λ promotes additional sparsity. In fact, the

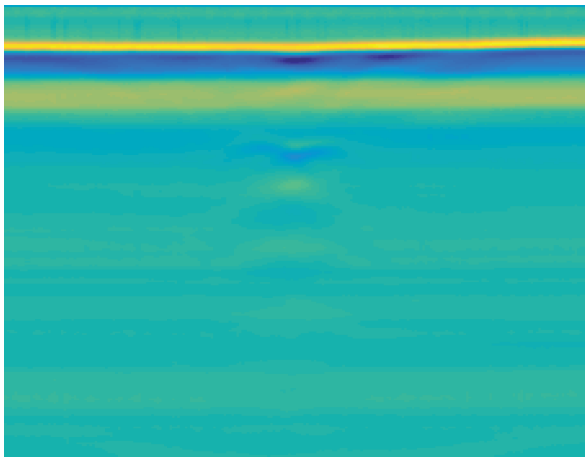
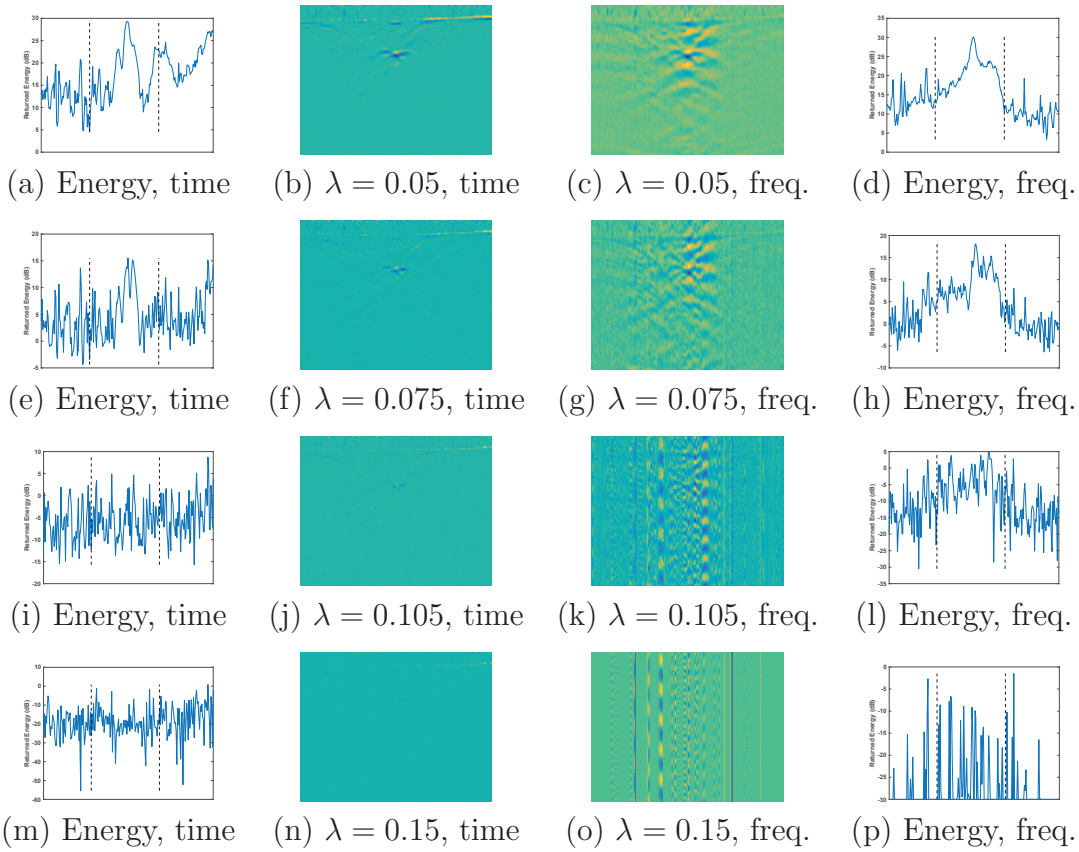


Figure 5.9: Unprocessed target B-scan used in Figures 5.10 and 5.12

results from PCP-AD in the time domain are essentially worthless when $\lambda > 0.075$. The same applies to the results from the frequency domain application, though for slightly higher values of λ.

Figure 5.11 shows λ's effect on the SCR more clearly. In both the time and frequency domain implementations, we see two peaks in the SCR. The first peak occurs in roughly the same location for both implementations—$\lambda = 0.06$. At this point, the images are closely related to the second row in Figure 5.10 and the target is clearly seen in either the B-scan or the energy plots. The second peaks in the SCR plot do not align as closely, but both correspond to the same phenomena. For example, see the B-scan in Figure 5.10(o) and its integrated energy in in Figure 5.10(p). Here we see that many of the columns of the B-scan have shrunk to zero because of the sparsity induced by the relatively high value of λ. The SCR is heavily influenced by the density of the impulses in the energy plot, of which there are few that correspond to clutter, thus inflating the SCR. It should be noted however that this second peak



(a) Energy, time (b) $\lambda = 0.05$, time (c) $\lambda = 0.05$, freq. (d) Energy, freq.

(e) Energy, time (f) $\lambda = 0.075$, time (g) $\lambda = 0.075$, freq. (h) Energy, freq.

(i) Energy, time (j) $\lambda = 0.105$, time (k) $\lambda = 0.105$, freq. (l) Energy, freq.

(m) Energy, time (n) $\lambda = 0.15$, time (o) $\lambda = 0.15$, freq. (p) Energy, freq.

Figure 5.10: A single B-scan processed with PCP-AD using multiple values for λ along with the integrated energies. Note that image is labeled either *time* or *frequency*, corresponding to the domain in which the PCP-AD algorithm was implemented. Dashed vertical lines denote the target region.

is not reliable across multiple targets since it is heavily influenced by the magnitude of the returns and distribution of clutter; the first peak in the SCR plot is empirically more robust and is not affected by these target variations as much as the second peak.

The images in Figure 5.12 show the results of applying OP-RPCA to the B-scan in Figure 5.9. Note that it is only implemented in the time domain since experiments have shown that the results are so similar when applied in the frequency domain.

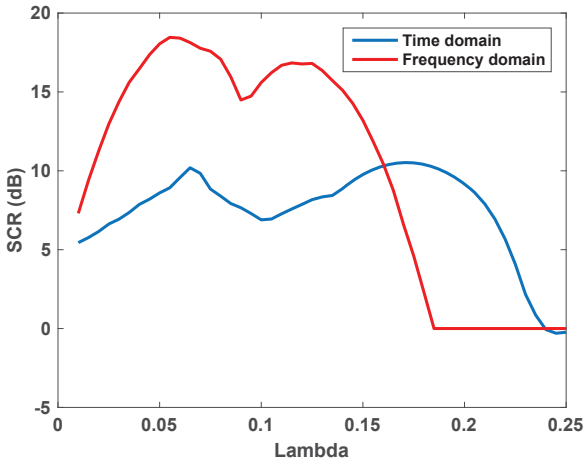
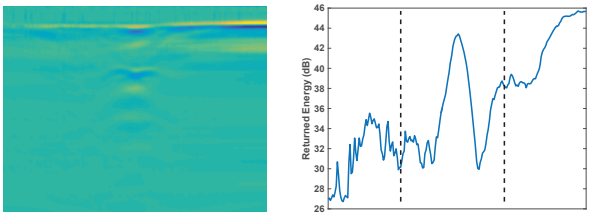


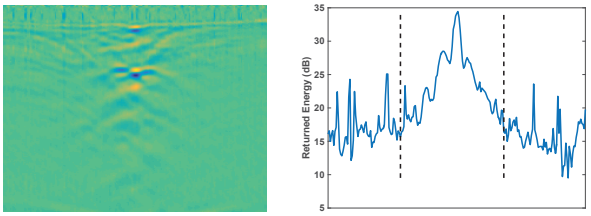
Figure 5.11: Effect of the tuning parameter on the SCR when using PCP-AD. Note that the frequency domain trace is clamped to zero for situations where computing the SCR is numerically limited, i.e., in cases where 0/0 is approached.

Again we see that as λ is increased it has a major affect on the resulting images and their energy content. For $\lambda \geq 0.5$, the images highlight the sparse objects perhaps too much—hyperbolic signatures of various point scatterers can be seen throughout the images. Visually optimal results are achieved when $\lambda = 0.3$ as shown in Figure 5.12(c).

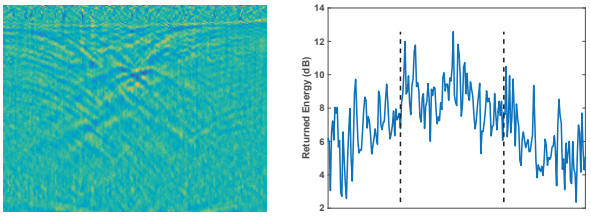
Figure 5.13 shows the effect of λ on the SCR. Like the previous experiment with PCP-AD, we see two peaks, though the second can be discarded for similar reasons. The first peak shows that OP-RPCA performs almost identically to PCP-AD when λ is tuned to its optimal setting—approximately 0.3 in this case. The SCR is also less sensitive to changes in λ in this region when compared to that of PCP-AD.



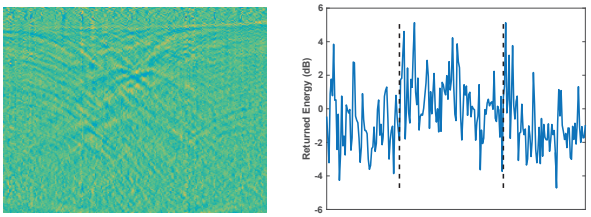
(a) $\lambda = 0.1$ (b) Energy



(c) $\lambda = 0.3$ (b) Energy



(e) $\lambda = 0.5$ (b) Energy



(g) $\lambda = 0.7$ (b) Energy

Figure 5.12: A single B-scan processed with OP-RPCA in the time domain using multiple values for λ along with the integrated energies. Dashed vertical lines denote the target region.

5.4.5 Preprocessing Conclusions

Thus far, this chapter focused on preprocessing the GPR data with RPCA. The results show that the ℓ_1-filtering algorithm achieves the best results overall, when

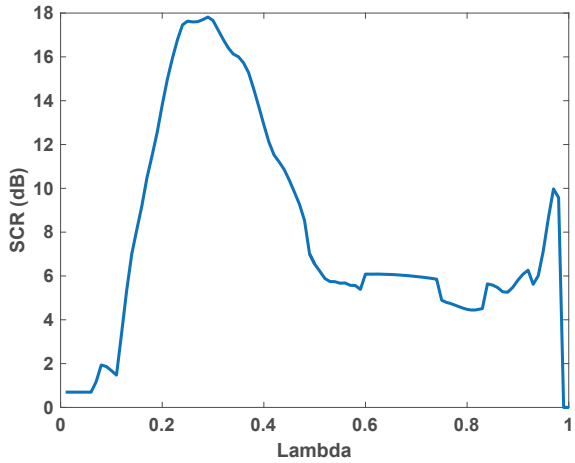


Figure 5.13: Effect of the tuning parameter on the SCR when using OP-RPCA.

implemented in the time domain. Further target-specific experiments also show that the PCP-AD and OP-RPCA algorithms also tend to perform very well. The timing results suggest that, of these approaches, the ℓ_1-filtering algorithm performs best in terms of both SCR increase and decomposition time, though Fast PCP is clearly superior only in terms of the time to convergence. Finally, experiments inspecting the effect of tuning parameter λ show that while SCR is generally increased for a wide range of λ, there is typically a clear choice for λ that maximizes the SCR.

As one of the best performing algorithms, PCP-AD is used to preprocess the GPR data for the remainder of this chapter. Next, an approach to feature-level fusion using the GPR system described in Section 5.3 is presented.

Part II

Approach to Explosive Hazard Detection Using Sensor Fusion and Multiple Kernel Learning with Downward-Looking GPR and EMI Sensor Data

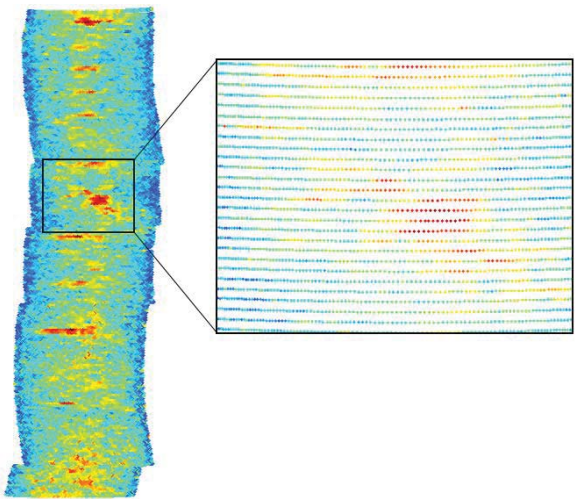


Figure 5.14: Scattered plot of integrated A-scan energy for Lane A, GPR Channel 1.

5.5 Image Formation

The GPR data comprise a discrete waveform representing the radar return (A-scan) at each spatial sample of the lane; an example of a GPR A-scan waveform is shown in Figure 5.5(a). The energy of each waveform at location (i, j) is calculated from each return $x_{i,j}$ as

$$E_{i,j} = \frac{1}{N} \sum_{n=1}^{N} x_{i,j}[n] x_{i,j}^{*}[n], \tag{5.9}$$

where N represents the number of samples in each A-scan and x^* is the complex conjugate of x. These data points are then scattered onto their respective spatial location as shown in Figure 5.14.

The discrete energy points are linearly interpolated over a common grid and the lane

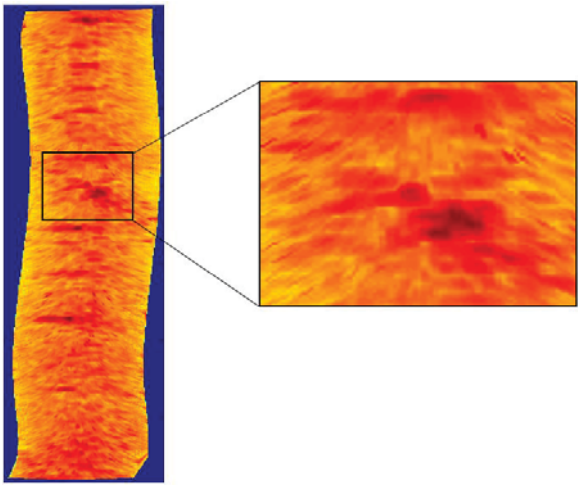


Figure 5.15: Linear interpolation of scatter plot in Figure 5.14.

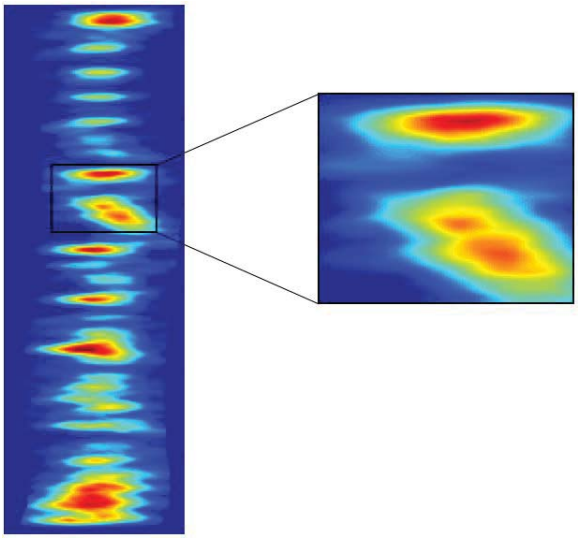


Figure 5.16: Results of applying clutter removal with $m = 0.85$ to Figure 5.14.

edges are masked to eliminate the interpolation artifacts that lie outside the sweep area. An example image is shown in Figure 5.15. Note that the horizontal-vertical axes are not to scale; the vertical axis depicts approximately 30 m and the horizontal axis is approximately 1.5 m.

5.5.1 Clutter Removal

The target locations in Figure 5.15 are all located in the center of the lane[4], so it is clear that a good deal of clutter exist in areas relatively far away from targets. To remove most clutter, we compare the linearly interpolated energy image to a threshold. All pixels in the image that are less than the threshold are set to zero and all other pixels remain unchanged, or

$$\begin{aligned} I'_{i,j} &= 0, \forall (i,j) \in \{arg_{(i,j)}\{I_{i,j} \leq T\}\} \\ I'_{i,j} &= I_{i,j}, otherwise, \end{aligned} \tag{5.10}$$

where $I_{i,j}$ is the original image, T is the threshold, and $I'_{i,j}$ is the clutter suppressed image.

Because images will generally have different pixel value distributions, we use a method of generating a threshold based on the histogram of pixel values for a particular image. The pixel threshold T is chosen to satisfy

$$m = \frac{\int_0^T h_I(n)dn}{\int_0^N h_I(n)dn}, \tag{5.11}$$

where m is the selection of "mass" proportion, $h_I(n)$ is the histogram of the image,

[4]See Figure 5.17 for target locations.

and N is the maximum pixel value in the image; m is labeled as a mass because of the similarity of this equation with the calculation of mass given a density. Dividing by the total mass of the image allows the mass proportion m we choose to be in the range of 0 to 1. This allows the selection of *one* mass proportion to generate the pixel threshold for *each* particular image. Figure 5.16 shows the results of applying a mass proportion of 0.85 to the image in Figure 5.15.

5.5.2 Image Ensemble

Images formed with data from different sensors generally differ, especially in the case of the EMI sensor. However, there are obviously some differences that can be beneficial during data fusion. Figure 5.17 shows an ensemble of images of Lane A. The first three images are from the three GPR channels and the bottom image is from the EMI sensor data. The EMI sensor's image exhibits very crisp definitions of the metallic objects and virtually no extraneous hot spots; the other images from the GPR channels are all very similar. Ground truth locations of targets are shown as red asterisks.

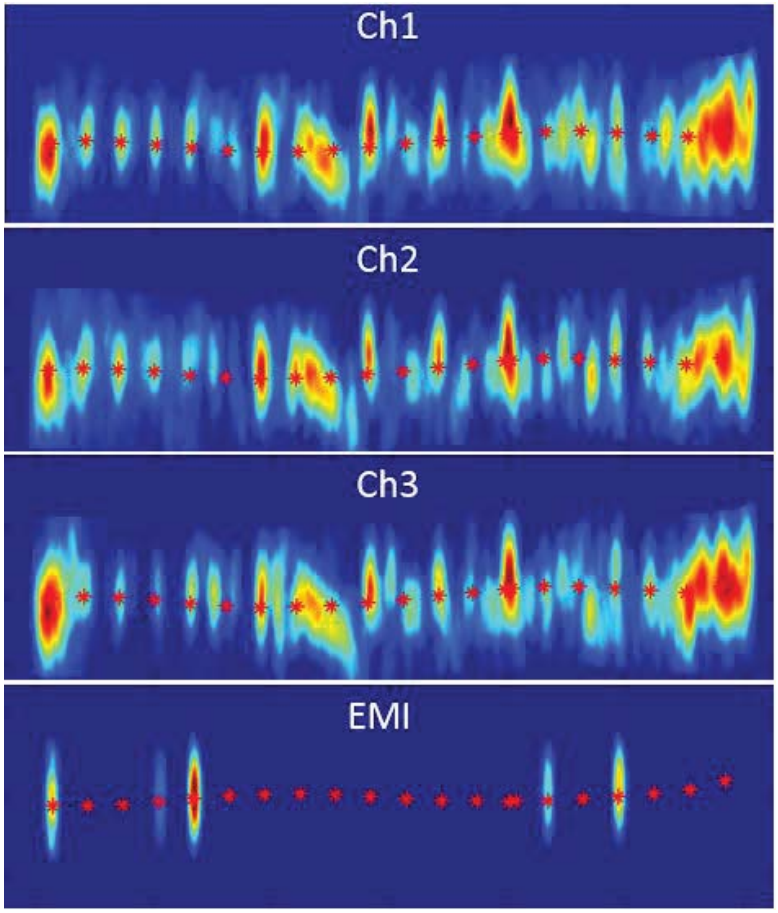


Figure 5.17: Image Ensemble for Lane A with ground truths labeled with asterisks.

5.6 CFAR Prescreener

The prescreening detector is the first algorithm applied to the lane images to generate a hit list of candidate detection locations. The prescreening approach used in this chapter is very similar to that used in previous work[6, 138]; however, we give a brief overview of the approach here.

Images are first filtered to detect local maxima using a size-contrast filter. This is implemented by first generating two new images from the clutter-removed image $I'(i,j)$ as

$$I_{\mu_c}(i,j) = \frac{\{I' * H_c\}}{\sum H_c}(i,j); \tag{5.12a}$$

$$I_{\mu_h}(i,j) = \frac{\{I' * H_h\}}{\sum H_h}(i,j); \tag{5.12b}$$

where $*$ indicates convolution and H_c and H_h are the circular convolution kernels shown in Figure 5.18. I_{μ_c} and I_{μ_h} are essentially the mean of the center cluster of pixels and the mean of the surrounding halo of pixels, respectively. The difference of these two values is the final size-contrast filtered image,[5]

$$I_{sc}(i,j) = I_{\mu_c}(i,j) - I_{\mu_h}(i,j). \tag{5.13}$$

Since thresholding I_{sc} can result in a dense cluster of detections in a small local region, we employ a maximum order-filter to eliminate redundant hits. The maximum order-filter image, I_o, is generated using a circular kernel with radius of 0.3 meters, and detection locations are indicated by

$$A = arg_{(i,j)}\{I_{sc}(i,j) = I_o(i,j)\}, \tag{5.14}$$

[5]Note that this prescreener is based on the *difference of means*. Other prescreeners can be formulated using other difference measures, such as the Bhattacharyya distance.[6]

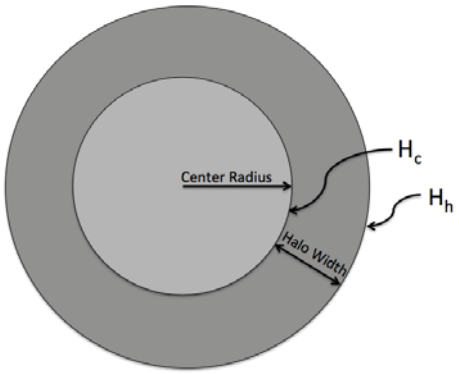


Figure 5.18: Circular convolution kernel used in prescreener.

where A is the set of cross-range and down-range detection locations. Features are extracted at each of these detection locations as discussed in Section 5.8. For a more complete discussion of the methods employed in this prescreener, see our previous papers.[6, 138]

5.7 Sensor Fusion

5.7.1 Run Packing

Run packing (RP) is an algorithm proposed by Glenn et al.[139] for *alarm set fusion* (ASF), also called decision- or confidence-level fusion. During training, the algorithm compiles a joint sequence of confidence values from multiple alarm sources using a greedy packing strategy. This sequence is then used to compute a monotonically

non-decreasing mapping function for each individual alarm source. When using RP blindly on new data from the same alarm source set, the confidence values from the alarm sources get mapped according to the mapping function found during training. These new confidence values are then used for detection. For a more detailed account of RP, see the paper by Glenn et al.[139].

5.7.2 Composite Confidence Maps

The method of generating a *composite confidence map* (CCM) has been utilized in previous work[140], however our implementation is somewhat different. The idea is that the detections from the CFAR prescreener discussed in Section 5.6 are blurred by some circularly symmetric blurring function, and then all the detections are aggregated on a common map. In this chapter, we explore the use of a Gaussian blurring function and the aggregation is performed using a summation method and a maximum-retaining method.

5.7.2.1 CCM via Summation Method

Mathematically, the generation of the CCM for A sensors using the summation method is

$$M_s(i,j) = \sum_{a=1}^{A} I_a(i,j) * B_W(i-m, j-n), \tag{5.15}$$

where M_s is the CCM, $I_a(i,j)$ is the confidence of the hit at location (i,j) (if there is not a hit at (i,j) then $I_a(i,j) = 0$), and B_W is a blurring influence function masked by a 2-dimensional window defining the width of the blurring function.

5.7.2.2 CCM via Maximum Method

An alternative method for generating the CCM is retaining the value of the individual confidence map (after blurring) whose value is largest. More concretely,

$$M_s(i,j) = \max_{A}\{I_a(i,j) * B_W(i-m, j-n)\}. \tag{5.16}$$

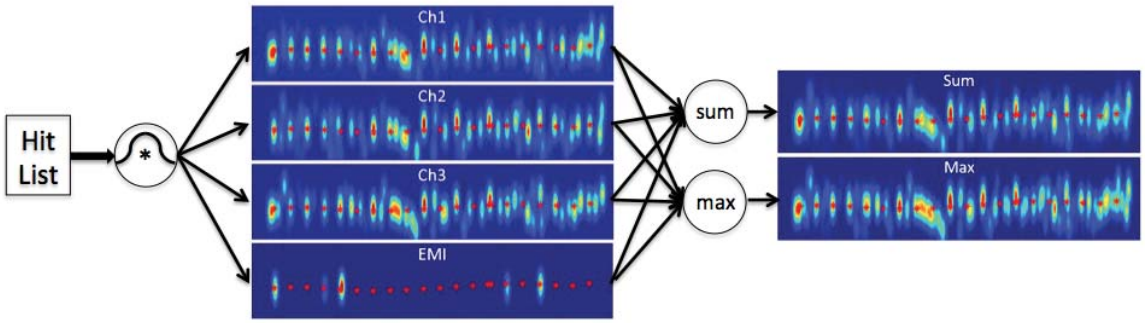


Figure 5.19: CCM examples for Lane A using a Gaussian influence function. Asterisks indicate ground truth locations.

5.7.2.3 Blurring Functions

The definition of B_W for the Gaussian case is

$$B_W(m,n) = e^{\frac{m^2+n^2}{\sigma^2}}, \tag{5.17}$$

where the standard deviation, σ, is chosen to make the Guassian influence function an appropriate size. In this work, we use an influence function size of 0.5 meters and $\sigma = 0.25$ meters. Figure 5.19 shows examples of a CCM for Lane A using this Gaussian blurring influence function. Note that while we explored both CCM via the summation method and maximum method, the summing method is consistently superior to the maximum method. Hence, all results in this work use the summation method.

Table 5.7
Length of Features and Full Feature Set.

Feature	Dimension
LSTAT	27
HOG	81
LBP	90
FFST	261
Full Feature Set	**459**

5.8 Features

At each prescreener detection, we extract a 16 pixel square sub-image centered on the hit location. The sub-image is divided into a 3×3 cell arrangement where each 8 pixel square cell overlaps its neighbor by 50%. The features discussed in the following sections are then extracted from each cell for use in classification via *support vector machines* (SVMs). Table 5.7 summarizes the dimension of each feature as well as the feature set dimension.

5.8.1 Local Statistics

The *local statistics* (LSTAT) feature vector is calculated for each of the 9 cells in the sub-image surrounding a candidate alarm. This vector consists of the mean, median, and standard deviation of the pixels. Hence, each target location is described by a

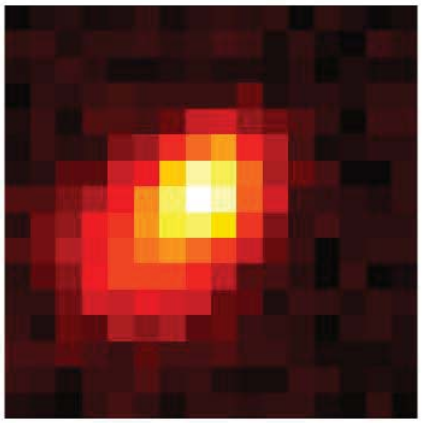


Figure 5.20: Sub-image at hit candidate location.

LSTAT feature vector of length $9 \times 3 = 27$. These values are normalized over all hits by subtracting the mean and dividing by the standard deviation of the feature vectors.

5.8.2 Histogram of Oriented Gradients

The histogram of oriented gradients (HOG) feature vector represents texture by calculating the gradients in a local area and compiling the gradients in a histogram descriptor[141, 142]. The use of this feature for explosive hazard detection is not novel, but it has been shown to be useful in similar studies[6, 138, 143].

For each cell in the sub-image, a HOG feature vector is created. A HOG feature vector with 9 histogram bin centers is computed for each of the 9 cells in the sub-image,

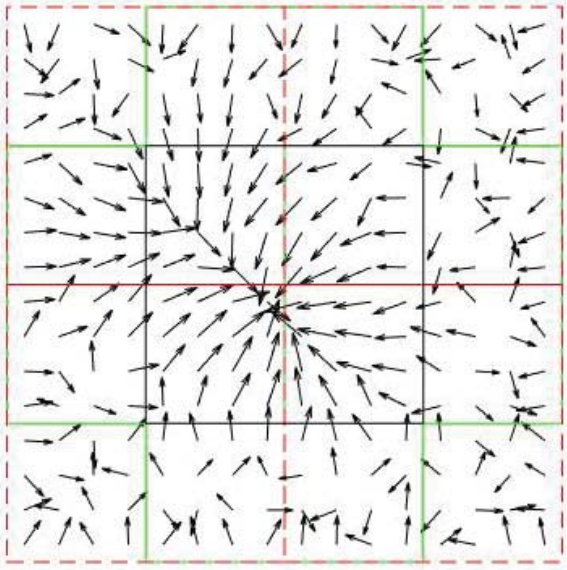


Figure 5.21: Gradient calculation with 3×3 cell arrangement.

resulting in a total of $9 \times 9 = 81$ feature values describing each candidate alarm. An example of the application of HOG on a candidate alarm location is shown in Figures 5.20—5.22. Figure 5.20 shows the 16×16 pixel image extracted at a candidate location. Figure 5.21 shows the cell partitioning and gradient calculations at each pixel, and Figure 5.22 shows a compilation of the gradients within each cell.

5.8.3 Local Binary Patterns

Local binary patterns (LBPs) are a gray-scale and rotation invariant texture classification. They are similar to HOG in that they are based on the gradients of the pixels within a cell, however with LBPs only the existence of gradients is recorded in the feature vector rather than the gradient magnitude and direction.

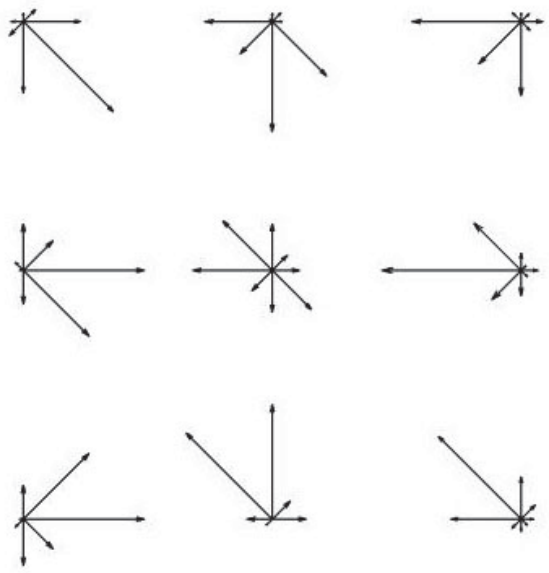


Figure 5.22: Cell based 9-bin HOG feature.

In this work we use a uniform, rotational-invariant LBP developed by Ojala et al.[144] While the formulation of the LBP neighborhood can be generalized, we compute the LBP features for each pixel in a cell by looking at its 8 neighboring pixels in a 1 pixel radius. Using the standard notation [144], each neighborhood is used to compute

$$LBP_{8,1} = \sum_{n=1}^{8} s(t_p - t_c)2^p, \tag{5.18}$$

where

$$s(x) = \begin{cases} 1 & x \geq 0, \\ \\ 0 & x < 0. \end{cases} \tag{5.19}$$

Note that the summation in (5.18) constructs an 8-bit binary string which characterizes the image texture in the 1-radius around the center pixel. This string achieves rotational invariance by rolling the binary string to a standard form, then uniforming it using a look-up table. Finally, the pixel is labeled with a unique label in the set $\{0, 9\}$. For a more in-depth discussion of the LBP methods used in this work, see the paper by Ojala et al. [144]

The LBP feature is extracted from each cell as a 10-bin histogram. The histogram is computed by counting the occurrences of the labels in each cell, i.e.,

$$h_{LBP}(m) = \sum_{u,v \in \text{cell}} S\{LBP_{i,1}(u, v) = m\}, m = 0, 1, ..., 9, \tag{5.20}$$

where $S\{B\}$ is a Boolean function taking the value 1 when the argument B is true, and 0 when B is false. Finally, the histogram is normalized as

$$\tilde{h}_{LBP}(m) = \frac{h_{LBP}(m)}{\sum_{i=1}^{10} h_{LBP}(i)}, \tag{5.21}$$

and $\tilde{h}_{LBP}(m)$ is the feature extracted from the cell. Since this LBP feature is extracted for each of the 9 cells surrounding a candidate location, the overall LBP feature length of a candidate hit is $9 \times 10 = 90$.

5.8.4 Fast Finite Shearlet Transform

Shearlets were developed by Guo et al.[145] as dilations, shear transformations, and translations to a single *mother function*, which are used to extract anisotropic features from an image and perform multiresolution analysis. In this respect, they are an extension of wavelets. The mother function, most easily represented in the frequency domain as $\hat{\psi}(\omega_1, \omega_2) = \hat{\psi}_1(\omega_1)\hat{\psi}_2(\frac{\omega_2}{\omega_1})$, where $\psi_1(\cdot)$ and $\psi_2(\cdot)$ in the time (or spatial) domain satisfy the properties described by Guo et al.[145] and Hauser et al.[146] Shearlet features have been utilized in explosive hazard detection previously [147]; however, the application was in the context of forward looking infrared imaging.

The implementation of the shearlet transform we use is termed the fast finite shearlet transform (FFST)[146], based on *discrete shearlets on the cone*[145]. This discretizes the support of the shearlets in the Fourier domain to bandpass regions parameterized by scale, direction, and frequencies. For three scales, the frequency tiling is shown in Figure 5.23. The indices in Figure 5.23 correspond to the indices shown in Figure 5.24 depicting the frequency domain magnitude of the FFST filters. Note the less interesting low-pass filter corresponding to frequency tile 1 in Figure 5.23 is not shown in Figure 5.24.

Recall that the FFST features are generated using the FFST implementation given by

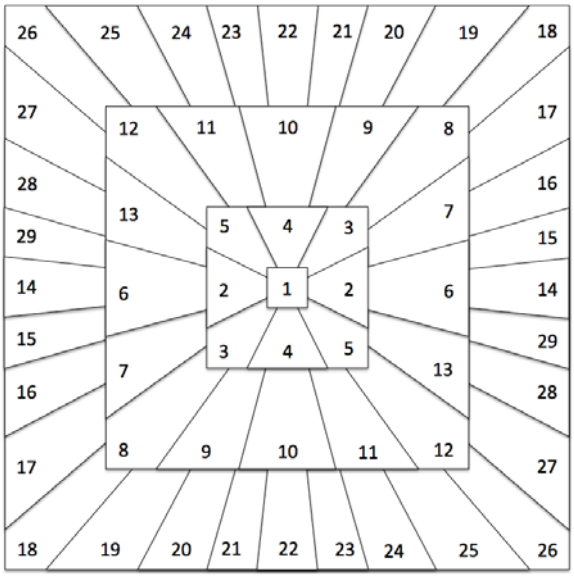


Figure 5.23: Frequency tiling structure for three scales.

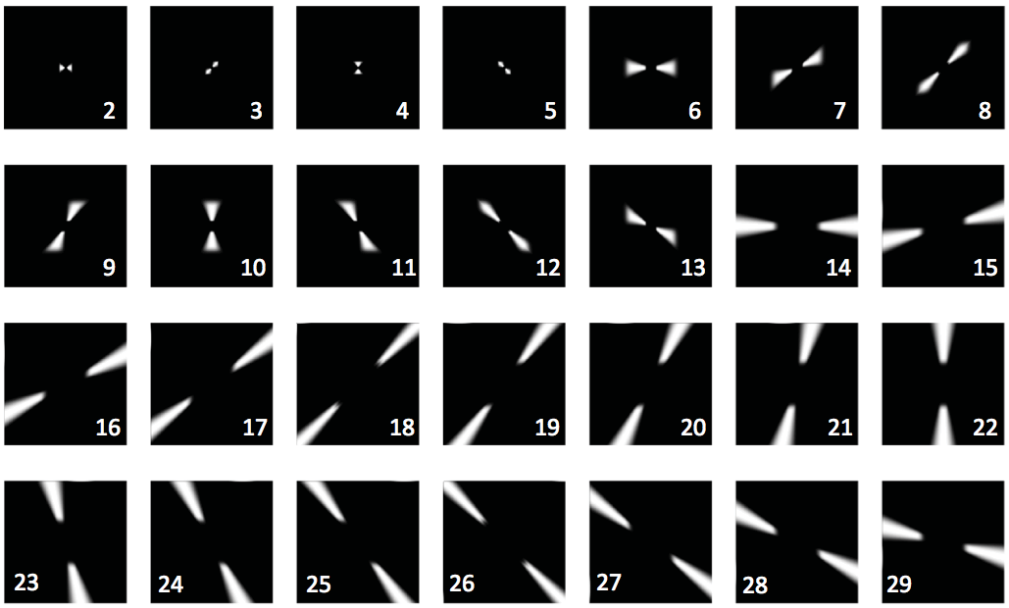


Figure 5.24: Frequency domain magnitude of FFST filters.

Hauser et al.[146] For an $m \times n$ image, the FFST computes $m \times n$ coefficients for each scale index. As demonstrated in Figure 5.23, the FFST over j scales results in $2^{j+2} - 3$

scale indices. Thus, if the FFST was generated for each of the 9 cells surrounding a candidate location over 3 frequency scales, the overall FFST feature length of a candidate hit would be $9 \times 8 \times 8 \times 29 = 16,704$. Since this is obviously a ludicrous feature length, we use a histogram approach similar to that proposed by Schwartz et al.[148] known as *histograms of shearlet coefficients* (HSC). This method works as follows. For each scale index j, a histogram is generated with $2^{j+2} - 3$ bins (one bin for each orientation). Each bin is then computed as the sum of the absolute values of the FFST coefficients of that bin's respective orientation. After the j histograms are computed, they are concatenated and ℓ_2-norm normalized resulting in the HSC feature. Extracting the HSC feature over 3 frequency scales from each of the 9 cells surrounding a candidate alarm results in a feature length of $9 \times 29 = 261$.

5.9 Results

5.9.1 Performance Metric: NAUC

To compare the results of different detectors and different lanes, we generate the detector's *receiver-operating-characteristic* (ROC). The ROC plot is based on the confidences of the hit list as well as the labels of the hits. The horizontal axis, though labeled as false alarms per unit area, is also directly proportional to a threshold

against which the confidence of the hits is compared. As the threshold is increased, the relation of the probability of detection and *false alarm rate* (FAR) is shown.

To quantify the results of a particular ROC, we find the *normalized area under the ROC* (NAUC) up to a FAR of 0.1 FA/m^2, then normalize the result so that a perfect detector will have an NAUC of 1. The NAUC equation is

$$NAUC = \frac{1}{0.1} \int_0^{0.1} P_D(FAR) dFAR, \tag{5.22}$$

where $P_D(FAR)$ is the probability of detection at false alarm rate, FAR. Equation (5.22) shows that the minimum value of the NAUC is zero if $P_D(FAR) = 0$ for $FAR \in [0, 0.1]$. It is also clear from Eq. (5.22) that an NAUC of 1 corresponds to perfect detection at zero FAR. An example ROC plot is shown in Figure 5.25. This ROC plot is shaded up to 0.1 FA/m^2 to explicitly show how we calculate the NAUC. As shown in the figure, the NAUC for this particular detector, lane, and channel is 0.35.

The miss-distance halo size used for these results is a 0.25 meter radius circle. In other words, if an indicated hit is within 0.25 meters of a ground-truth target location, then it is considered a detection. This is justified by the fact that if one were to dig and explore a point on the ground, it is very probable that the surrounding 0.25 meters is also explored.

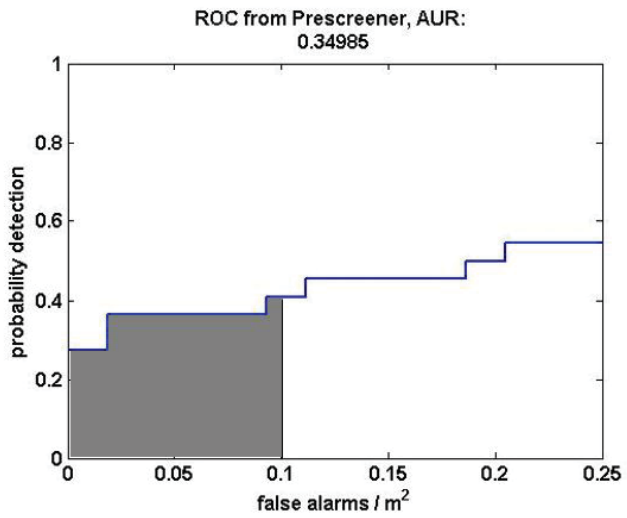


Figure 5.25: Normalized area-under-ROC (NAUC) calculation.

5.9.2 Prescreener Results

The prescreener was run on each individual channel from each lane. The images were generated using an exclusion threshold of 0.85, and the spatial parameters of the size-contrast filter were set such that the center cluster of pixels had a radius of 15 cm and the halo had a width of 12 cm. Table 5.8 summarizes the NAUCs for each sensor on each lane, and the solid line in Figure 5.26 shows the overall ROC for each lane where the hits from all sensors are lumped together to form one ROC. The NAUCs for these conglomerate ROCs are given in the second column of Table 5.9.

Table 5.8 shows that Channel 1 outperforms all other sensors on half the lanes, while no other sensor outperforms another on more than one lane. Note that the EMI

Table 5.8
Results of CFAR Prescreener on each Channel and Lane.

Lane	Channel 1	Channel 2	Channel 3	EMI
A	**0.375**	0.142	0.175	0.207
B	**0.352**	0.281	0.066	0
C	0.117	0.149	**0.208**	0.176
D	0.059	**0.077**	0	0
E	**0.311**	0.214	0	0
F	0.067	0.021	0.021	**0.071**

results cannot compare fairly to the other sensors because of the nature of the ROCs. The EMI sensor does extremely well finding metallic targets, however all others are missed. This explains why the EMI ROC is typically a flat line; if the EMI sensor does detect something, it detects it very well and we see a nonzero value of the ROC at zero FAR. Any false alarms found by the EMI sensor will have such low confidences that they do not appear in the ROC in the range of 0 to 0.25 FAR.

5.9.2.1 Prescreener with RP

The dashed lines in Figure 5.26 show the ROCs for each lane after RP is applied to the prescreener hit list. Since training is required with RP, five lanes are used to train and the remaining lane is used as the test lane. Thus, each RP ROC in Figure 5.26 shows the results after testing on its respective lane and training on all others. The NAUCs calculated from these ROCs are shown in column 3 of Table 5.9.

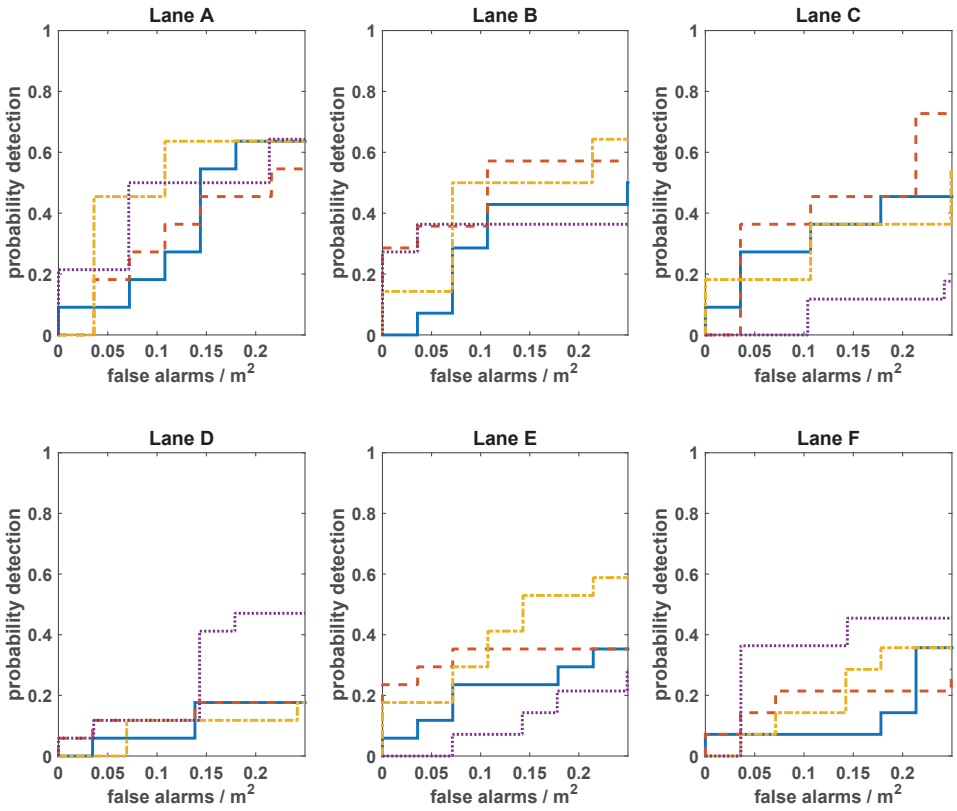


Figure 5.26: Lumped ROCs for each lane; solid: prescreener, dashed: RP, dotted: CCM, dashed/dotted: RP + CCM.

From Figure 5.26 it is apparent that the RP algorithm tends to pull the ROC up and left. This is also highlighted by the fact that NAUC after RP is applied is greater than that of the raw prescreener hits for every lane (see columns 2 and 3 of Table 5.9). Indeed, the application of RP doubles the NAUC when compared to results from the raw prescreener hits.

Table 5.9
NAUCs for the ROCs.

Lane	No RP	RP	CCM	RP + CCM
A	0.116	0.142	0.233	0.291
B	0.108	0.332	0.296	0.245
C	0.208	**0.234**	**0.331**	0.182
D	0.038	**0.097**	0	0.036
E	0.130	0.290	0.097	0.210
F	0.071	**0.138**	0.021	0.067

5.9.2.2 Prescreener with CCM

Applying CCM discussed in Section 5.7.2 to the prescreener hits gives rise to the ROCs shown as dotted lines in Figure 5.26. The NAUCs for these ROCs are also given in the fourth column of Table 5.9. Overall, CCM improves performance by approximately 25% from the prescreener alone, however its results vary significantly. For example, on Lane B the NAUC is almost raised by a factor of 4, however on Lane D the performance is decreased to zero.

5.9.3 Prescreener with RP and CCM

Finally, the dashed-dotted lines in Figure 5.26 show the ROCs after applying both the RP algorithm and CCM to the prescreener hits, and their respective NAUCs are

shown in column five of Table 5.9. Similar to the results for that of CCM only, these have a significant amount of variance and their performance depends heaviliy on the lane. While the average performance increase is about 50%, the results are actually inferior to the raw prescreener results on half the lanes.

5.9.4 SKSVM

The results in this section are the NAUCs using a SKSVM classifier using a RBF kernel with $C = 1$ and $\gamma = 1/D$, where D is the length of the feature vector. Refer to Table 5.7 for a summary of the feature vector lengths. The tables show the results for each lane after training on the five remaining lanes, and the results for each individual feature type are also given.

5.9.4.1 SKSVM using Prescreener Hits

Table 5.10 shows the results of the SKSVM classifier using the raw hits from the prescreener. We see that the HOG feature is superior to all other features, and most of its results are better than the best prescreener results in Table 5.8. Another feature worth of mentioning is the FFST feature which achieves better results than the prescreener. The local statistic feature vector is by far the worst, achieving a NAUC of zero for four lanes. This classifier achieves the best results for Lane A and

Table 5.10
Results of SKSVM Classifier on each Channel and Lane using Different Features—Prescreener Hits.

	Lane							
Feature Type	A	B	C	D	E	F	Avg.	Std. Dev.
HOG	**0.389**	0.306	**0.345**	**0.097**	0.050	**0.138**	0.221	0.131
LBP	0.349	0.066	0.091	0	0.088	**0.071**	0.111	0.111
LSTAT	0.204	0	0	0	0	0.062	0.044	0.075
FFST	0.233	0.046	0.117	**0.118**	0.038	**0.133**	0.114	0.064
All Features	0.309	0.163	0.111	0.038	0.113	0.067	0.134	0.088

Lane C.

5.9.4.2 SKSVM using CCM

The application of CCM before using the SKSVM gives the NAUCs given in Table 5.11. Once again, the HOG feature outperforms the others, however a comparison with Table 5.10 reveals the application of CCM obviously hinders performance, in general. Interestingly, the local statistics feature is successful on more lanes, however its average performance increase is marginal. Additionally, the results for the use of all the features on Lane C more than doubled, while all results for Lane D dropped to 0.

Table 5.11
Results of SKSVM Classifier on each Channel and Lane using Different Features—CCM Hits.

	Lane							
Feature Type	A	B	C	D	E	F	Avg.	Std. Dev.
HOG	0.175	0.071	**0.228**	0	0.210	**0.092**	0.129	0.081
LBP	0.058	0.153	0.182	0	0.092	0	0.081	0.070
LSTAT	0	0.046	0.105	0	0.113	0.021	0.048	0.046
FFST	0	0.143	0.143	0	0.055	0.046	0.065	0.059
All Features	0.058	0.041	**0.273**	0	0.04	0.02	0.072	0.092

5.9.4.3 SKSVM using RP + CCM

The final experiment with SKSVM used the RP-scaled confidence values of the prescreener hits to form a CCM from which new hits were generated and used for the results in Table 5.12. While half of the average performances are an improvement over those of Table 5.11, only the local statistics feature average result is an improvement over the prescreener results in Table 5.9. The FFST feature results for two lanes dropped to zero from the addition of RP, however its performance on Lane E doubled. It is also worthwhile to note that using all features yields the best results thus far on Lane D.

Table 5.12
Results of SKSVM Classifier on each Channel and Lane using Different Features—RP + CCM Hits.

	Lane							
Feature Type	A	B	C	D	E	F	Avg.	Std. Dev.
HOG	0.331	0.082	0	0	0.176	0.020	0.102	0.120
LBP	0.059	0.046	0.119	0.059	0.185	0.046	0.086	0.051
LSTAT	0	0.046	0.119	**0.113**	0.093	0	0.062	0.050
FFST	0	0	0	0	0.110	0.046	0.026	0.041
All Features	0.078	0.138	**0.273**	**0.134**	0.033	0	0.109	0.088

5.9.5 MKLSVM

The results in this section are the NAUCs using a MKLSVM classifier using a collection of five RBF kernels. In each experiment, the five RBF parameters are set as $\gamma_i = 10^i/D, i \in \{-2, -1, 0, 1, 2\}$, where D is the length of the feature vector. See Table 5.7 for a breakdown of the values of D for each feature type. The tables show the results for each lane after training on the five remaining lanes, and the results for each individual feature type are also given.

5.9.5.1 MKLSVM using Prescreener Hits

Table 5.13 shows the results of applying MKLSVM to the raw prescreener hits. The local statistics feature's average results are increased by a factor of three from the

Table 5.13
Results of MKLSVM Classifier on each Channel and Lane using Different Features—Prescreener Hits.

	Lane							
Feature Type	A	B	C	D	E	F	Avg.	Std. Dev.
HOG	0.204	0.306	0.202	0	0.017	**0.189**	0.153	0.081
LBP	**0.381**	0.071	0	0	0.055	0.041	0.091	0.132
LSTAT	0.342	0.214	0.111	0	0.092	0.067	0.138	0.111
FFST	0.233	0.046	**0.228**	**0.136**	0.038	**0.133**	0.136	0.077
All Features	0.084	0.260	0.026	**0.176**	0.076	0	0.104	0.089

SKSVM case, the average results using all the features in MKLSVM improve that of SKSVM by approximately 30%, and the HOG feature is generally still the best feature. This classifier is the best classifier explored in this work with respect to Lane D and Lane E.

5.9.5.2 MKLSVM using CCM

Forming a CCM from the prescreener hits then applying a MKLSVM gives the results in Table 5.14. This modification to the MKLSVM approach seems to severly hinder its performance, and the HOG feature is no longer the best feature. Note that the only results for this classifier that beat the prescreened results occur on Lane C.

Table 5.14
Results of MKLSVM Classifier on each Channel and Lane using Different Features—CCM Hits.

	Lane							
Feature Type	A	B	C	D	E	F	Avg.	Std. Dev.
HOG	0	0.179	0.080	0	0.13	0	0.065	0.071
LBP	0.025	0.087	**0.299**	0	0	0	0.069	0.108
LSTAT	0	0.066	**0.299**	0.036	0.113	0.021	0.089	0.100
FFST	0	0.092	0.085	0	0	0	0.030	0.042
All Features	0.084	0.041	**0.273**	0	0.097	0.021	0.086	0.090

5.9.5.3 MKLSVM using RP + CCM

Finally, Table 5.15 shows the results of the MKLSVM after forming a CCM using the RP-normalized confidences from the prescreener. The average results of the LSTAT feature were almost doubled from the CCM only case, and the average results of using all features was raised by approximately 50%. The other feature results all decreased from the CCM case, however two only dropped marginally.

5.9.5.4 Results Summary

The summary of the best test results from the experiments described above are shown in Table 5.16. It shows that the HOG feature appears in the best method for half of the lanes, and interestingly, the CFAR prescreener achieved superior results on

Table 5.15
Results of MKLSVM Classifier on each Channel and Lane using Different Features—RP + CCM Hits.

	Lane							
Feature Type	A	B	C	D	E	F	Avg.	Std. Dev.
HOG	0.026	0.133	0.028	0	0.185	0	0.062	0.071
LBP	0.059	0.021	0.119	0.017	0.156	0.020	0.065	0.054
LSTAT	0.208	0.103	**0.273**	**0.118**	0.110	0.066	0.146	0.071
FFST	0	0	0.091	0	0	0	0.015	0.034
All Features	0.176	0.138	**0.273**	**0.134**	0.017	0	0.123	0.093

two lanes using the data from Channel 1 only. SKSVM and MKLSVM served as the best classifiers on two lanes each, with one instance of MKLSVM performing best using all the extracted features. We also note that while the RP and CCM sensor fusion algorithms do not appear in the list of the best classifiers, they are able to boost performance with respect to the CFAR prescreener as demonstrated by the bold numbers in Tables 5.11, 5.12, 5.14, and 5.15.

The major challenge we encountered with RP algorithm was that it results in multiple hits on false alarm targets; hence, while the confidence on each target might be relatively increased, the number of false alarms is also increased. In practice, this may not be a problem as one would only dig one hole for a group of closely-arranged hits. The CCM algorithm was designed to attack this specific deficiency, but in some cases it took hits that were originally within the target halo and dragged them away from targets through its fusion process.

Table 5.16
Summary of Best Results.

Lane	Best Performing Classifier	NAUC
A	SKSVM using HOG feature on prescreener hits	0.389
B	CFAR Prescreener on Channel 1	0.352
C	SKSVM using HOG feature on prescreener hits	0.345
D	MKLSVM using all features on prescreener hits	0.176
E	CFAR Prescreener on Channel 1	0.311
F	MKLSVM using HOG feature on prescreener hits	0.189

5.10 Conclusion

In this part, we applied sensor fusion and multiple kernel learning to SVM-based classifiers on images generated from integrated energy from a downward-looking GPR. The sensor fusion approaches we explored include run packing and composite confidence mapping. We also inspected the utility of combining the two fusion techniques. Features are extracted from both the candidate hit locations and the regions surrounding them, making the features context-based. Features we used include the well established HOG and LBP features, as well as a feature describing local statistics and another based on the coefficients of the fast finite shearlet transform. RBF kernels were applied to these features for classification with SKSVM and MKLSVM classifiers. We presented the results for these features and classifiers using the sensor

fusion approaches previously mentioned, and our results showed that while no best algorithm exists, some approaches we explored deserve additional analysis.

The final part of this chapter applies the fusion algorithms discussed in Chapter 2 are to energy-based features from the RPCA-preprocessed GPR data.

Part III

Explosive Hazard Detection with Feature and Decision Level Fusion, Multiple Kernel Learning, and Fuzzy Integrals

5.11 Explosive Hazard Detection Dataset

The explosive hazard dataset used in the experiments that follow is different than that derived in Part II and is composed of a collection of 1,955 11-dimensional feature vectors with class labels $\{-1, +1\}$, corresponding to true negatives and true positives, respectively. These feature vectors are computed as discussed in the following section.

5.11.1 Prescreener and Feature Extraction

A simple energy-based prescreener using the RPCA sparse component was utilized to identify queue points to be investigated using the classifiers. The energy of each discrete radar return was found and a ground map was formed as shown in Figure 5.27. The prescreener then flags local maxima in the integrated energy ground map as queue points, and features are then extracted from those points. The features that were collected from the queue points for this experiment were based on energy and localized contrast. Specifically, for each queue point location, the features include

† the energy at the detection location;

† the energy in a disk of radius 20 cm in the integrated energy ground map;

† the ratios of energy in disks of radius 10, 20, 30, and 40 cm to the energy in a

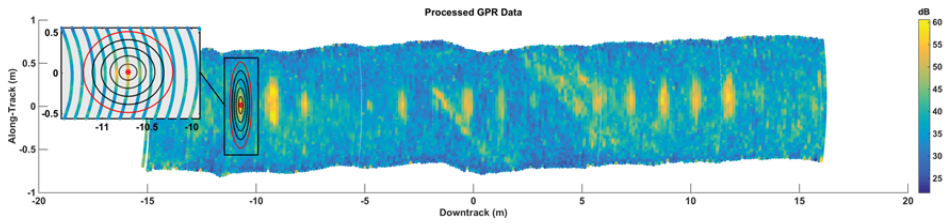


Figure 5.27: Integrated energy ground map and an example queue point with 10, 20, 30, and 40 cm disks.

disk of radius 50 cm in the integrated energy ground map;

† the ratios of energy in circles of radius 10, 20, and 30% of total image size to total energy in the B-scan image[6].

5.12 Experiments and Results

Here we present the results of the GAMKL$_p$ and DeFIMKL algorithms after applying them to the GPR data set described in Section 5.11 using SVM classifiers; we use LIBSVM to implement the classifiers [76]. Their performance is presented alongside the results of the state-of-the-art MKLGL algorithm. Additionally, the results are compared with those of the prescreener such that the overall improvement can be evaluated.

Each experiment consists of 100 trials, where the results of these trials are statistically

[6]The B-scan image is a collection of individual radar returns surrounding the queue point location. This image essentially represents a vertical slice of earth at the queue point location. More details on these radar returns and B-scans can be found in [5, 130].

compared via a two-sample t-test at a 5% significance level. Including the standard deviation in the results highlights the sensitivity (variance) of each classifier to the selection of training data. In each trial the data set is partitioned into five partitions, each holding 20% of the data. The training/testing cycle is performed five times, where four partitions are used as training data and the remaining partition is used as the testing data; the testing results from each partition are combined to form the overall ROC for each trial and the NAUC is extracted as the performance metric.

Fifty RBF kernels are used in each algorithm with respective RBF widths σ logarithmically spaced on the interval $[10^{-2}, 10^{1.6}]$; the same RBF parameters are used for each algorithm.

5.12.1 Experiment 1

The first experiment was designed to compare the results of the MKL methods discussed in this chapter with the prescreener and MKLGL algorithms. This experiment applies the different MKL classifiers to the same data partitions such that the results can be compared equally. Table 5.17 summarizes the average NAUCs from this experiment along with their improvement over the prescreener; the standard deviations are given in parentheses.

Table 5.17
NAUCs and percentage improvement compared to the prescreener*.

	Results	
Algorithm	NAUC	% Improvement
Prescreener	0.204	-
$MKLGL_1$	0.438 (0.013)	115%
$MKLGL_2$	0.466 (0.013)	129%
$GAMKL_1$	**0.504 (0.012)**	**147%**
$GAMKL_2$	0.494 (0.013)	142%
DeFIMKL	0.350 (0.074)	72%
$DeFIMKL_1$ $\lambda = 1$	0.461 (0.041)	126%
$DeFIMKL_2$ $\lambda = 2$	0.486 (0.017)	138%

*Bold indicates the best performer according to a two-valued t-test at a 5% significance level..

The results show that the GAMKL algorithm has superior performance when compared to the MKLGL algorithm and the regularized DeFIMKL algorithms' performance is comparable to MKLGL's performance, however, the ℓ_2-regularized DeFIMKL algorithm does beat MKLGL. The standard deviation of $DeFIMKL_2$ is marginally higher than that of MKLGL, suggesting that the DeFIMKL training is more closely dependent on the selection of training data and thus more susceptible to overtraining. This conclusion is further supported by the relatively large standard deviation exhibited by the unregularized DeFIMKL algorithm, which is to be expected since regularization was not employed to suppress the possibility of overtraining (i.e., classifier variance). GAMKL, on the other hand, has essentially equivalent classifier variance, i.e., it is just as susceptible to overtraining as MKLGL.

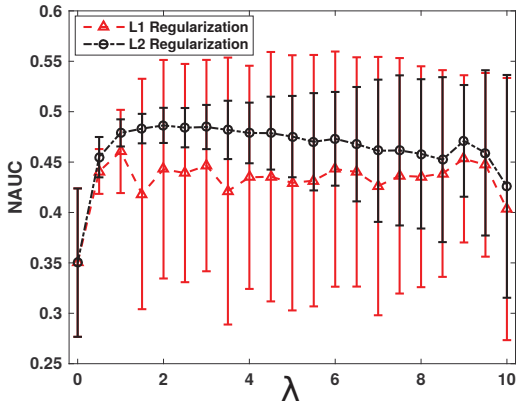


Figure 5.28: DeFIMKL performance using regularization. Error bars indicate ± one standard deviation.

5.12.2 Experiment 2

A second experiment was performed with the DeFIMKL algorithm to observe the effects of the regularization parameter λ. This experiment applies the regularized DeFIMKL algorithms to the data while varying λ over the range $[0, 10]$. Figure 5.28 summarizes the results of this experiment.

The trend of the plot shows the importance of including regularization with the DeFIMKL algorithm, since both DeFIMKL$_1$ and DeFIMKL$_2$ benefit by using a nonzero λ. However, the average NAUC generally decreases as λ is increased. Furthermore, the standard deviation of the DeFIMKL$_2$ results increases nearly consistently with increasing λ, though the trend is opposite with the DeFIMKL$_1$ standard deviation.

5.13 Conclusion

The latter part of this chapter applied a feature-level fusion algorithm, GAMKL$_p$, and a decision-level fusion algorithm, DeFIMKL, to a dataset derived from ground penetrating radar for explosive hazard detection. GAMKL$_p$ uses a genetic algorithm to find the multiple kernel mixing coefficients, $\boldsymbol{\sigma}$, and is generalized to allow $\boldsymbol{\sigma}$ to lie in the ℓ_p-norm domain, Δ_p. The DeFIMKL algorithm aggregates kernels through the use of the Choquet fuzzy integral with respect to a fuzzy measure learned by a regularized quadratic programming approach. We use MKLGL as the benchmark MKL algorithm, and show that both GAMKL$_p$ and DeFIMKL can outperform MKLGL.

Chapter 6

Conclusion

This dissertation presents some novel feature-level fusion and decision-level fusion techniques, discusses many extensions of the feature-level fusion algorithm we call DeFIMKL, and shows the various experiments on real-world and synthetic datasets used to validate their performance. Furthermore, the preceding chapter outlines the application of some of these algorithms to the problem of explosive hazard detection with ground penetrating radar. Summarized below are the challenges undertaken in this dissertation along with a brief explanation of how they were addressed.

Multiple kernel learning challenges include the determination of what kernel combination is best, and the scalability problem—MKL generally requires a large amount of memory when used with large datasets. The former is addressed with

the proposal of new fusion techniques, i.e., GAMKL, DeFIMKL, DeGAMKL, and DeLSMKL, and I apply the Nyström approximation to the multiple kernel matrices to tame MKL's scalability. Experiments show that some of the new fusion algorithms can compete with state-of-the-art methods, and the Nyström approximation tends to allow most of the kernel matrices to be discarded, therefore alleviating the demanding memory requirements.

Learning underdetermined fuzzy measures was tackled by using various regularization functions that either generalize the DeFIMKL model, reduce model complexity, or encode information about the underlying FM in the learning process. This problem arises due to the fact that training data are essentially never diverse enough to determine the underlying FM. Experimental results using the various regularization functions show that they do indeed alter the behavior of the algorithm, however, the best one is always data-dependent. This work also spawned a visualization strategy to easily observe the behavior of the FM and fuzzy integral.

Applications to explosive hazard detection was discussed, as well as my robust principal component analysis preprocessing investigation. I presented an example of the full detection pipeline including data processing and feature extraction, and applied some of the novel fusion algorithms presented in earlier chapters to the problem where I showed that the DeFIMKL algorithm outperforms a current state-of-the-art

MKL algorithm.

6.1 Future Work

The following ideas are candidates for future research on the topics addressed in the previous chapters.

Nonlinear aggregations of kernels. The Choquet fuzzy integral was used extensively for decision aggregation in this dissertation, but can it (or a similar nonlinear aggregation operator) be used for combining kernels for feature-level fusion? All MKL methods used in this work assume the kernels are combined via linear combination which is well known to result in a new *valid* kernel (i.e., a Mercer kernel), but would the fusion further benefit from the power generally exhibited by a nonlinear aggregation? This is a nontrivial problem since one must first prove that the nonlinear aggregation operator used results in a Mercer kernel—itself a difficult task.

Improving the algorithms' scalability. The previous chapters have shown that these algorithms suffer from scalability problems. Specifically, MKL methods are generally demanding in terms of memory and computation time, and fuzzy integral-based algorithms requires the specification or learning of a fuzzy measure that explodes very

quickly (recall that for a panel of m decision-makers, the FM includes 2^m unique values). The Nyström approximation was utilized to address the former issue, and while its results are very promising, the speedup it provides with large datasets is only approximately linear in the number of training data. Perhaps there is a further extension or completely new approximation method that would allow better scalability in this sense. The fuzzy integral algorithms also suffer from a similar issue—the number of free parameters to learn and the number of constraints can quickly swamp memory and processing resources. There are current collaborations that have started exploring this issue, but more work is needed to develop novel methods of increasing the utility and practicality of fuzzy integrals with many inputs.

Algorithm stability across datasets. As is always the case with classification problems, the success of a particular algorithm is always dependent on the data at hand. The methods proposed in this dissertation are no exception—in many cases they tend to do very well (either on par with or beating other state-of-the-art methods), however, there are the other cases when they simply do not work so well. Thus, more work is needed to explore methods of increasing the algorithms' stability across datasets. The choice of regularization function is a large part of this (many of which were discussed in the dissertation), but perhaps there are other methods that can be employed or developed to address this issue.

Interactive visualizations to improve fuzzy integral intuition and education. The fuzzy integral is not an intuitive operator and there is much to be learned by the general practitioner about its behavior from helpful visualizations. The simple visualizations presented in this dissertation convey a lot of information and illuminate the integral's behavior, however, there is much more that can be done. Specifically, an *interactive* visualization would allow the user to explore the lattice deeper. I imagine a visualization that allows the user to inspect individual data points (paths) and nodes in the lattice. Not only will this provide deeper insight into the fuzzy integral's behavior, it can also serve as an educational platform for students or researchers getting started in this field.

References

[1] A. Pinar, T. C. Havens, D. T. Anderson, and L. Hu, "Feature and decision level fusion using multiple kernel learning and fuzzy integrals," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Aug 2015, pp. 1–7.

[2] A. J. Pinar, J. Rice, L. Hu, D. T. Anderson, and T. C. Havens, "Efficient multiple kernel classification using feature and decision level fusion," *IEEE Transactions on Fuzzy Systems*, vol. PP, no. 99, pp. 1–1, 2016.

[3] A. Pinar, T. C. Havens, J. Rice, M. Masarik, J. Burns, and B. Thelen, "A comparison of robust principal component analysis techniques for buried object detection in downward looking gpr sensor data," in *SPIE Defense+ Security*. International Society for Optics and Photonics, 2016, pp. 98 230T–98 230T.

[4] A. J. Pinar, J. Rice, T. C. Havens, M. Masarik, J. Burns, and D. T. Anderson, "Explosive hazard detection with feature and decision level fusion, multiple

kernel learning, and fuzzy integrals," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec 2016, pp. 1–8.

[5] A. Pinar, M. Masarik, T. C. Havens, J. Burns, B. Thelen, and J. Becker, "Approach to explosive hazard detection using sensor fusion and multiple kernel learning with downward-looking GPR and emi sensor data," in *Proc. SPIE*, vol. 9454, 2015, pp. 94 540B–94 540B–20.

[6] T. C. Havens, J. Becker, A. Pinar, and T. J. Schulz, "Multi-band sensor-fused explosive hazards detection in forward-looking ground penetrating radar," in *Proc. SPIE*, vol. 9072, 2014.

[7] T. C. Havens, D. T. Anderson, K. Stone, J. Becker, and A. J. Pinar, "Computational intelligence methods in forward-looking explosive hazard detection," in *Recent Advances in Computational Intelligence in Defense and Security*. Springer, 2016, pp. 13–44.

[8] M. Kloft, U. Brefeld, P. Laskov, and S. Sonnenburg, "Non-sparse multiple kernel learning," 2008.

[9] Z. Xu, R. Jin, H. Yang, I. King, and M. Lyu, "Simple and efficient multiple kernel learning by group lasso," in *Proc. Int. Conf. Machine Learning*, 2010, pp. 1175–1182.

[10] C. Cortes, M. Mohri, and A. Rostamizadeh, "ℓ_2 regularization for learning kernels," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2009, pp. 109–116.

[11] L. Hu, D. T. Anderson, and T. C. Havens, "Multiple kernel aggregation using fuzzy integrals," in *IEEE International Conference on Fuzzy Systems*. IEEE, 2013, pp. 1–7.

[12] L. Hu, D. T. Anderson, T. C. Havens, and J. M. Keller, "Efficient and scalable nonlinear multiple kernel aggregation using the choquet integral," in *Proc. Int. Conf. Info. Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer, 2014, pp. 206–215.

[13] M. Kloft, U. Brefeld, S. Sonnenburg, and A. Zien, "Lp-norm multiple kernel learning," *The Journal of Machine Learning Research*, vol. 12, pp. 953–997, 2011.

[14] M. Gönen and E. Alpaydın, "Multiple kernel learning algorithms," *The Journal of Machine Learning Research*, vol. 12, pp. 2211–2268, 2011.

[15] M. Sugeno, "Theory of fuzzy integrals and its applications," Ph.D. dissertation, Tokyo Institute of Technology, 1974.

[16] D. Anderson, S. Price, and T. Havens, "Regularization-based learning of the choquet integral," in *IEEE International Conference on Fuzzy Systems*, July 2014, pp. 2519–2526.

[17] A. Pinar, T. C. Havens, D. T. Anderson, and L. Hu, "Feature and decision level fusion using multiple kernel learning and fuzzy integrals," in *Proc. IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Aug 2015, pp. 1–7.

[18] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[19] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28–44, 2013.

[20] F. Castanedo, "A review of data fusion techniques," *The Scientific World Journal*, vol. 2013, 2013.

[21] H. F. Durrant-Whyte, "Sensor models and multisensor integration," *The International Journal of Robotics Research*, vol. 7, no. 6, pp. 97–113, 1988.

[22] B. V. Dasarathy, "Sensor fusion potential exploitation-innovative architectures and illustrative applications," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 24–38, 1997.

[23] R. C. Luo, C.-C. Yih, and K. L. Su, "Multisensor fusion and integration: approaches, applications, and future research directions," *Sensors Journal, IEEE*, vol. 2, no. 2, pp. 107–119, 2002.

[24] J. K. Aggarwal, *Multisensor fusion for computer vision.* Springer Science & Business Media, 2013, vol. 99.

[25] F. S. Khan, J. Van de Weijer, and M. Vanrell, "Top-down color attention for object recognition," in *IEEE 12th International Conference on Computer Vision.* IEEE, 2009, pp. 979–986.

[26] P. Natarajan, S. Wu, S. Vitaladevuni, X. Zhuang, S. Tsakalidis, U. Park, R. Prasad, and P. Natarajan, "Multimodal feature fusion for robust event detection in web videos," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2012, pp. 1298–1305.

[27] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of intelligent & robotic systems*, vol. 61, no. 1-4, pp. 287–299, 2011.

[28] Y.-R. Yeh, T.-C. Lin, Y.-Y. Chung, and Y.-C. F. Wang, "A novel multiple kernel learning framework for heterogeneous feature fusion and variable selection," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 563–574, 2012.

[29] P. Gehler and S. Nowozin, "On feature combination for multiclass object classification," in *IEEE 12th International Conference on Computer Vision.* IEEE, 2009, pp. 221–228.

[30] L. Cao, J. Luo, F. Liang, and T. S. Huang, "Heterogeneous feature machines

for visual recognition," in *IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 1095–1102.

[31] J. Yang, Y. Li, Y. Tian, L. Duan, and W. Gao, "Group-sensitive multiple kernel learning for object categorization," in *IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 436–443.

[32] B. Fernando, E. Fromont, D. Muselet, and M. Sebban, "Discriminative feature fusion for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3434–3441.

[33] C. H. Chan, M. A. Tahir, J. Kittler, and M. Pietikainen, "Multiscale local phase quantization for robust component-based face recognition using kernel fusion of multiple descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 5, pp. 1164–1177, 2013.

[34] Y. Zhang, H. L. Yang, S. Prasad, E. Pasolli, J. Jung, and M. Crawford, "Ensemble multiple kernel active learning for classification of multisource remote sensing data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 2, pp. 845–858, 2015.

[35] B. Bigdeli, F. Samadzadegan, and P. Reinartz, "Fusion of hyperspectral and lidar data using decision template-based fuzzy multiple classifier system," *International Journal of Applied Earth Observation and Geoinformation*, vol. 38, pp. 309–320, 2015.

[36] D. Anderson, T. Havens, C. Wagner, J. Keller, M. Anderson, and D. Wescott, "Extension of the fuzzy integral for general fuzzy set-valued information," *IEEE Transactions on Fuzzy Systems*, vol. 22, no. 6, pp. 1625–1639, Dec 2014.

[37] C. Wagner, D. Anderson, and T. Havens, "Generalization of the fuzzy integral for discontinuous interval- and non-convex interval fuzzy set-valued inputs," in *IEEE International Conference on Fuzzy Systems*, July 2013, pp. 1–8.

[38] D. T. Anderson, J. M. Keller, and T. C. Havens, "Learning fuzzy-valued fuzzy measures for the fuzzy-valued sugeno fuzzy integral," in *Computational Intelligence for Knowledge-Based Systems Design*. Springer, 2010, pp. 502–511.

[39] M. Grabisch, *Fuzzy Measures and Integrals: Theory and Applications*, M. Sugeno and T. Murofushi, Eds. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000.

[40] D. Zhang and Z. Wang, "Fuzzy integrals of fuzzy-valued functions," *Fuzzy Sets and Systems*, vol. 54, no. 1, pp. 63–67, 1993.

[41] R. Yang, Z. Wang, P.-A. Heng, and K.-S. Leung, "Fuzzified choquet integral with a fuzzy-valued integrand and its application on temperature prediction," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 2, pp. 367–380, 2008.

[42] M. Grabisch, H. T. Nguyen, and E. A. Walker, *Fundamentals of uncertainty*

calculi with applications to fuzzy inference. Springer Science & Business Media, 2013, vol. 30.

[43] T. Havens, D. Anderson, and J. Keller, "A fuzzy choquet integral with an interval type-2 fuzzy number-valued integrand," in *IEEE International Conference on Fuzzy Systems*, July 2010, pp. 1–8.

[44] D. Anderson, T. Havens, C. Wagner, J. Keller, M. Anderson, and D. Wescott, "Sugeno fuzzy integral generalizations for sub-normal fuzzy set-valued inputs," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, June 2012, pp. 1–8.

[45] X. Wang, A. Chen, and H. Feng, "Upper integral network with extreme learning mechanism," *Neurocomputing*, vol. 74, no. 16, pp. 2520–2525, 2011.

[46] X. Liang, C. Wei, and Z. Chen, "An intuitionistic fuzzy weighted owa operator and its application," *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 6, pp. 713–719, 2013.

[47] D. J. Dubois, *Fuzzy sets and systems: theory and applications*. Academic press, 1980, vol. 144.

[48] M. F. Anderson, D. T. Anderson, and D. J. Wescott, "Estimation of adult skeletal age-at-death using the sugeno fuzzy integral," *American journal of physical anthropology*, vol. 142, no. 1, pp. 30–41, 2010.

[49] A. Mendez-Vazquez and P. Gader, "Sparsity promotion models for the choquet integral," in *IEEE Symposium on Foundations of Computational Intelligence*. IEEE, 2007, pp. 454–459.

[50] J. Keller and J. Osborn, "A reward/punishment scheme to learn fuzzy densities for the fuzzy integral," in *International Fuzzy Systems Association World Congress*, 1995, pp. 97–100.

[51] ——, "Training the fuzzy integral," *International Journal of Approximate Reasoning*, vol. 15, no. 1, pp. 1–24, 1996.

[52] C. Wagner and D. T. Anderson, "Extracting meta-measures from data for fuzzy aggregation of crowd sourced information," in *IEEE Int. Conf. Fuzzy Systems*, June 2012, pp. 1 –8.

[53] T. C. Havens, D. T. Anderson, C. Wagner, H. Deilamsalehy, and D. Wonnacott, "Fuzzy integrals of crowd-sourced intervals using a measure of generalized accord," in *IEEE International Conference on Fuzzy Systems*. IEEE, 2013, pp. 1–8.

[54] T. Havens, D. Anderson, and C. Wagner, "Data-informed fuzzy measures for fuzzy integration of intervals and fuzzy numbers," *IEEE Transactions on Fuzzy Systems*, vol. PP, no. 99, pp. 1–1, 2014.

[55] M. Grabisch, "Fuzzy integral for classification and feature extraction," in *Fuzzy*

Measures and Integrals: Theory and Applications. Springer-Verlag New York, Inc., 2000, pp. 415–434.

[56] J. Keller, P. Gader, and A. Hocaoglu, "Fuzzy integral in image processing and recognition," in *Fuzzy Measures and Integrals: Theory and Applications*. Springer-Verlag New York, Inc., 2000, pp. 435–466.

[57] S. Auephanwiriyakul, J. M. Keller, and P. D. Gader, "Generalized choquet fuzzy integral fusion," *Information Fusion*, vol. 3, no. 1, pp. 69–85, 2002.

[58] H. Tahani and J. M. Keller, "Information fusion in computer vision using the fuzzy integral," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 3, pp. 733–741, 1990.

[59] G. Choquet, "Theory of capacities," in *Annales de l'institut Fourier*, vol. 5. Institut Fourier, 1954, pp. 131–295.

[60] T. Murofushi and M. Sugeno, "An interpretation of fuzzy measures and the choquet integral as an integral with respect to a fuzzy measure," *Fuzzy sets and Systems*, vol. 29, no. 2, pp. 201–227, 1989.

[61] M. Grabisch, "Fuzzy integral in multicriteria decision making," *Fuzzy sets and Systems*, vol. 69, no. 3, pp. 279–298, 1995.

[62] C. Cortes and V. N. Vapnik, "Support-vector networks," vol. 20, no. 3, 1995, pp. 273–297.

[63] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[64] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," vol. 209, 1909, pp. 441–458.

[65] J. Zhai, H. Xu, and Y. Li, "Fusion of extreme learning machine with fuzzy integral," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 21, no. 2, pp. 23–34, 2013.

[66] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," *Annual Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*, pp. 682–688, 2001.

[67] A. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 911–918, 2000.

[68] P. Drineas and M. Mahoney, "On the Nyström method for approximating a gram matrix for improved kernel-based learning," *The Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[69] R. Jin, T. Yang, M. Mahdavi, Y.-F. Li, and Z.-H. Zhou, "Improved bounds for the Nyström method with application to kernel classification," *IEEE Transactions on Information Theory*, vol. 59, no. 10, pp. 6939–6949, 2013.

[70] K. Zhang and J. Kwok, "Clustered Nyström method for large scale manifold learning and dimension reduction," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1576–1587, Oct 2010.

[71] S. Kumar, M. Mohri, and A. Talwalkar, "Ensemble Nyström method," *Advances in Neural Information Systems 22*, pp. 1060–1068, 2009.

[72] ——, "Sampling techniques for the Nyström method," *Journal of Machine Learning Research*, vol. 13, pp. 981–1006, Apr 2012.

[73] A. K. Farahat, A. Ghodsi, and M. S. Kamel, "A novel greedy algorithm for Nyström approximation," *International Conference on Artificial Intelligence and Statistics*, pp. 269–277, 2011.

[74] K. Zhang, L. Lan, Z. Wang, and F. Moerchen, "Scaling up kernel svm on limited resources: A low-rank linearization approach," in *International Conference on Artificial Intelligence and Statistics*, 2012, pp. 1425–1434.

[75] N. Djuric, L. Lan, S. Vucetic, and Z. Wang, "Budgetedsvm: A toolbox for scalable svm approximations," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 3813–3817, 2013.

[76] C. C. Chang and C. J. Lin, "LIBSVM: a library for support vector machines," *ACM Trans. Intell. Sys. Tech.*, vol. 2, no. 27, pp. 1–27, 2011.

[77] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[78] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *The Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[79] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2010.

[80] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[81] M. Grabisch, I. Kojadinovic, and P. Meyer, "A review of methods for capacity identification in choquet integral based multi-attribute utility theory: Applications of the kappalab r package," *European Journal of Operational Research*, vol. 186, no. 2, pp. 766 – 785, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221707002330

[82] D. T. Anderson, J. M. Keller, and T. C. Havens, "Learning fuzzy-valued fuzzy

measures for the fuzzy-valued sugeno fuzzy integral," in *International conference on information processing and management of uncertainty*, 2010, pp. 502–511.

[83] G. Beliakov, "Construction of aggregation functions from data using linear programming," *Fuzzy Sets and Systems*, vol. 160, pp. 65–75, 2009.

[84] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[85] M. Sugeno, "Theory of fuzzy integrals and its applications," *Ph.D. thesis*, vol. Tokyo Institute of Technology, 1974.

[86] H. Tahani and J. Keller, "Information fusion in computer vision using the fuzzy integral," *IEEE Transactions System Man Cybernetics*, vol. 20, pp. 733–741, 1990.

[87] D. T. Anderson, T. C. Havens, C. Wagner, J. M. Keller, M. F. Anderson, and D. Wescott, "Extension of the fuzzy integral for general fuzzy set-valued information," *IEEE Transactions on Fuzzy Systems*, vol. 22, no. 6, pp. 1625–1639, Dec 2014.

[88] M. Grabisch, H. Nguyen, and E. Walker, *Fundamentals of uncertainty calculi, with applications to fuzzy inference.* Kluwer Academic, Dordrecht, 1995.

[89] M. Grabisch, T. Murofushi, and M. Sugeno, *Fuzzy Measures and Integrals: Theory and Applications*. Physica-Verlag, Heidelberg, 2000.

[90] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[91] M. Grabisch, "k-order additive discrete fuzzy measures and their representation," *Fuzzy Sets and Systems*, vol. 92, no. 2, pp. 167 – 189, 1997, fuzzy Measures and Integrals. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165011497001681

[92] J. M. Keller and J. Osborn, "Training the fuzzy integral," *International Journal of Approximate Reasoning*, vol. 15, no. 1, pp. 1 – 24, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0888613X9500132Z

[93] ——, "A reward/punishment scheme to learn fuzzy densities for the fuzzy integral," *International Fuzzy Systems Association World Congress*, pp. 97–100, 1995.

[94] T. C. Havens, D. T. Anderson, C. Wagner, H. Deilamsalehy, and D. Wonnacott, "Fuzzy integrals of crowd-sourced intervals using a measure of generalized accord," *IEEE International Conference on Fuzzy Systems*, 2013.

[95] T. C. Havens, D. T. Anderson, and C. Wagner, "Constructing meta-measures from data-informed fuzzy measures for fuzzy integration of interval inputs and fuzzy number inputs," *IEEE Transactions on Fuzzy Systems*, 2015.

[96] A. Mendez-Vazquez and P. Gader, "Sparsity promotion models for the choquet integral," *IEEE Symposium on Foundations of Computational Intelligence*, pp. 454–459, 2007.

[97] D. T. Anderson, S. Price, and T. C. Havens, "Regularization-based learning of the choquet integral," in *2014 IEEE International Conference on Fuzzy Systems*, July 2014, pp. 2519–2526.

[98] I. Kojadinovic, J.-L. Marichal, and M. Roubens, "An axiomatic approach to the definition of the entropy of a discrete choquet capacity," *Information Sciences*, vol. 172, no. 1–2, pp. 131 – 153, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025504001811

[99] R. Yager, "On the entropy of fuzzy measures," *Fuzzy Systems, IEEE Transactions on*, vol. 8, no. 4, pp. 453–461, Aug 2000.

[100] R. R. Yager, "Uncertainty representation using fuzzy measures," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 1, pp. 13–20, Feb 2002.

[101] C. Labreuche, "Identification of a fuzzy measure with an l1 entropy," in *in Proc. of IPMU*, 2008, pp. 1476–1483.

[102] R. Yang, Z. Wang, P. A. Heng, and K. S. Leung, "Fuzzified choquet integral with a fuzzy-valued integrand and its application on temperature prediction,"

IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 38, no. 2, pp. 367–380, April 2008.

[103] S.-B. Cho and J. H. Kim, "Combining multiple neural networks by fuzzy integral for robust classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 2, pp. 380–384, Feb 1995.

[104] P. Melin, O. Mendoza, and O. Castillo, "Face recognition with an improved interval type-2 fuzzy logic sugeno integral and modular neural networks," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 5, pp. 1001–1012, Sept 2011.

[105] Q. Wu, Z. Wang, F. Deng, Z. Chi, and D. D. Feng, "Realistic human action recognition with multimodal feature selection and fusion," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 4, pp. 875–885, July 2013.

[106] T. Murofushi and S. Soneda, "Techniques for reading fuzzy measures (iii): Interaction index," in *Proceedings of the 9th Fuzzy Systems Symposium, Sapporo, Japan*, 1993, pp. 693–696.

[107] M. Grabisch and M. Roubens, "Application of the choquet integral in multicriteria decision making," in *Fuzzy measures and integrals*. Physica Verlag, 2000, pp. 348–374.

[108] J. L. Marichal, "Aggregation operators for multicriteria decision aid," *Ph.D. thesis, University of Liege, Liege, Belgium*, 1998.

[109] ——, "Entropy of discrete choquet capacities," *European Journal of Operational Research*, vol. 3 (137), pp. 612–624, 2000.

[110] I. Kojadinovic, "Minimum variance capacity identification," *Quaterly J. of Operations Research (4OR)*, vol. 12, pp. 23–36, 2006.

[111] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *IEEE Trans. Syst. Man Cybern.*, vol. 18, no. 1, pp. 183–190, Jan. 1988. [Online]. Available: http://dx.doi.org/10.1109/21.87068

[112] A. F. Tehrani, W. Cheng, K. Dembczyński, and E. Hüllermeier, "Learning monotone nonlinear models using the choquet integral," *Machine Learning*, vol. 89, no. 1-2, pp. 183–211, 2012.

[113] A. F. Tehrani and E. Hüllermeier, "Ordinal choquistic regression," *in EUSFLAT conference*, 2013.

[114] A. F. Tehrani, "Learning nonlinear monotone classifiers using the choquet integral," *PhD dissertation*, 2013.

[115] G. Beliakov, A. Pradera, and T. Calvo, *Aggregation Functions: A Guide for Practitioners*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[116] C. Gini, "On the measure of concentration with special reference to income and statistics," *Colorado College Publication*, no. 208, pp. 73–79, 1936.

[117] M. Brown, "Using Gini-style indices to evaluate the spatial patterns of health practitioners: theoretical considerations and an application based on Alberta data," *Social Science Medicine*, vol. 38, no. 9, pp. 1243–1256, 1994.

[118] F. A. Farris, "The gini index and measures of inequality," *The American Mathematical Monthly*, vol. 117, 2010.

[119] T. Leinster and C. A. Cobbold, "Measuring diversity: the importance of species similarity," vol. 93, no. 3, Mar. 2012, pp. 477–489. [Online]. Available: http://dx.doi.org/10.1890/10-2402.1

[120] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight, "Sparsity and smoothness via the fused lasso," *Journal of the Royal Statistical Society Series B*, pp. 91–108, 2005.

[121] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for Big Data: scalable, randomized, and parallel algorithms for big data analytics," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 32–43, 2014.

[122] A. N. Tikhonov, "[on the stability of inverse problems]," *Doklady Akademii Nauk SSSR*, vol. 39, no. 5, pp. 195–198, 1943.

[123] E. Candes, M. Wakin, and S. Boyd, "Enhancing sparsity by reweighted l1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, pp. 877–905, 2008.

[124] L. Hu, D. T. Anderson, T. C. Havens, and J. M. Keller, "Efficient and scalable nonlinear multiple kernel aggregation using the choquet integral," in *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, vol. 442, 2014, pp. 206–215.

[125] L. Hu, D. T. Anderson, and T. C. Havens, "Multiple kernel aggregation using fuzzy integrals," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2013, pp. 1–7.

[126] JIEDDO COIC MID. (2012) Global IED monthly summary report. [Online]. Available: https://publicintelligence.net/jieddo-global-ieds-aug-2012/

[127] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[128] E. J. Candes, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM*, vol. 58, no. 3, May 2011.

[129] M. Fazel, H. Hindi, and S. P. Boyd, "A rank minimization heuristic with application to minimum order system approximation," in *Proc. IEEE American Control Conference*, vol. 6. IEEE, 2001, pp. 4734–4739.

[130] M. P. Masarik, J. Burns, B. T. Thelen, J. Kelly, and T. C. Havens, "GPR anomaly detection with robust principal component analysis," in *Proc. SPIE*, vol. 9454, 2015, pp. 945 414–945 414–11.

[131] P. Rodriguez and B. Wohlberg, "Fast principal component pursuit via alternating minimization," in *20th IEEE International Conference on Image Processing (ICIP)*, Sept 2013, pp. 69–73.

[132] H. Xu, C. Caramanis, and S. Sanghavi, "Robust pca via outlier pursuit," *IEEE Transactions on Information Theory*, vol. 58, no. 5, pp. 3047–3064, May 2012.

[133] G. Tang and A. Nehorai, "Robust principal component analysis based on low-rank and block-sparse matrix decomposition," in *45th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2011, pp. 1–5.

[134] R. Liu, Z. Lin, S. Wei, and Z. Su, "Solving principal component pursuit in linear time via l_1 filtering," *arXiv preprint arXiv:1108.5359*, 2011.

[135] G. Liu and S. Yan, "Active subspace: Toward scalable low-rank learning," *Neural computation*, vol. 24, no. 12, pp. 3371–3394, 2012.

[136] S. D. Babacan, M. Luessi, R. Molina, and A. K. Katsaggelos, "Sparse bayesian methods for low-rank matrix estimation," *IEEE Transactions on Signal Processing*, vol. 60, no. 8, pp. 3964–3977, 2012.

[137] A. Sobral, T. Bouwmans, and E. hadi Zahzah, "Lrslibrary: Low-rank and sparse tools for background modeling and subtraction in videos," in *Robust Low-Rank and Sparse Matrix Decomposition: Applications in Image and Video Processing*. CRC Press, Taylor and Francis Group., 2015.

[138] T. C. Havens, K. Stone, D. T. Anderson, J. M. Keller, K. C. Ho, T. T. Ton, D. C. Wong, and M. Soumekh, "Multiple kernel learning for explosive hazard detection in forward-looking ground-penetrating radar," vol. 8357, 2012.

[139] T. Glenn, B. Smock, J. Wilson, and P. Gader, "A run packing technique for multiple sensor fusion," vol. 8709, 2013.

[140] T. C. Havens, J. M. Keller, K. C. Ho, T. T. Ton, D. C. Wong, and M. Soumekh, "Narrow-band processing and fusion approach for explosive hazard detection in flgpr," vol. 8017, 2011.

[141] D. Lowe, "Distinctive image features from scale invariant keypoints," *Int. Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[142] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," vol. 1, June 2005, pp. 886–893 vol. 1.

[143] D. Anderson, K. Stone, J. Keller, and C. Spain, "Combination of anomaly algorithms and image features for explosive hazard detection in forward looking infrared imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 1, pp. 313–323, Feb 2012.

[144] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, Jul 2002.

[145] K. Guo, G. Kutyniok, and D. Labate, "Sparse multidimensional representations using anisotropic dilation and shear operators," in *Int. Conf. Interaction between Wavelets and Splines*, 2005, pp. 1–6.

[146] S. Häuser and G. Steidl, "Fast finite shearlet transform: a tutorial," *ArXiv*, vol. 1202, no. 1773, 2014.

[147] B. Tuomanen, K. Stone, T. Madison, M. Popescu, and J. Keller, "Buried target detection in flir images using shearlet features," in *Proc. SPIE*, vol. 8709, 2013, p. 870919.

[148] W. Schwartz, R. da Silva, L. Davis, and H. Pedrini, "A novel feature descriptor based on the shearlet transform," in *18th IEEE International Conference on Image Processing*, Sept 2011, pp. 1033–1036.

[149] C. Cortes and V. N. Vapnik, "Support-vector networks," vol. 20, no. 3, 1995, pp. 273–297.

[150] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," vol. 209, 1909, pp. 441–458.

Appendix A

Support Vector Machines and MKLGL

A.0.1 Linear Support Vector Machines

The linear SVM is a two-class classifier based on a class-separating hyperplane, $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$. The hyperplane parameters $(\mathbf{w}, w_0)$ are found via the optimization problem,

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2, \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x_i} + w_0) \geq 1,\ i = 1, \ldots, n, \end{aligned} \tag{A.1}$$

where $y_i \in \{-1, 1\}$ are the class labels. The optimal hyperplane is centered between the nearest data point from each class, and it is oriented such that the margin is maximized. This only works for linearly-separable classes, however.

This idea is extended to the soft-margin SVM for non-separable classes, which is a similar optimization problem posed as

$$\begin{aligned} \underset{\mathbf{w},\xi}{\text{minimize}} \quad & J(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i, \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x_i} + w_0) \geq 1 - \xi_i,\ \xi_i \geq 0,\ i = 1, \ldots, n, \end{aligned} \tag{A.2}$$

where ξ_i are *slack variables*. For data that complies with the constraints in (A.1), the slack variables $\xi = 0$. For all other data, i.e., data inside the SVM margin and/or on the opposite side of the hyperplane, $\xi > 0$. The cost function in (A.2) shows that the optimal hyperplane parameters of the soft-margin SVM still attempt to maximize the margin, however the additional term forces the hyperplane to minimize the number of data points with $\xi > 0$, where $C > 0$ is a constant weight defining the influence of the slack variable term. In essence, C determines how many errors are allowed during training[149], and a suitable value is generally found during cross-validation.

A.0.2 Single Kernel Support Vector Machines

The single kernel soft-margin SVM (SKSVM) is an extension of the linear soft-margin SVM discussed in the previous section. The optimization problem is equivalent to that in (A.2), however instead of formulating the SVM from the primal optimization problem, the dual form is used. The dual form of the optimization problem is

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \text{ for } i = 1, \ldots, n; \ \sum_{i=1}^{n} \alpha_i y_i = 0, \end{aligned} \tag{A.3}$$

where $\kappa(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^{n \times n}$ is a kernel matrix. Note that if we define the kernel matrix to be the Euclidean dot product, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$, the SKSVM reduces to the linear SVM discussed in the previous section. Other popular kernels include the polynomial kernal, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^p$, and the *radial basis function* (RBF) kernel, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$.

We use LIBSVM to solve the SKSVM problem.[76] The classifier model generated by LIBSVM includes the Lagrange multiplier vector α and the bias b. These are used to classify a measured feature vector $\mathbf{x}$ by computing

$$y = \text{sgn} \left[\sum_{i=1}^{n} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) - b \right], \tag{A.4}$$

where sgn is the signum function.

A.0.3 Multiple Kernel Learning Support Vector Machines

Multiple kernel learning (MKL) can be applied to SVMs as a straightforward generalization of the SKSVM discussed in the previous section. MKLSVM allows the use of weighted combinations of multiple kernels, with the requirement that the resulting kernel is positive definite, i.e., a Mercer kernel.[150] Here we assume the kernel matrix K is a linear combination of precomputed kernel matrices, or

$$K = \sum_{k=1}^{m} \sigma_k K_k = \sum_{k=1}^{m} \sigma_k \kappa_k(\mathbf{x}_i, \mathbf{x}), \tag{A.5}$$

where m represents the total number of kernels and σ_k is the weight applied to the kth kernel, K_k. Using the composite kernel K in the SKSVM optimization given in (A.3) and also optimizing over the weights σ_k gives the formulation for a MKLSVM as the minimax problem,

$$\begin{aligned} \underset{\sigma}{\text{minimize}}\ \underset{\alpha}{\text{maximize}} \quad & L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\sum_{k=1}^{m} \sigma_k \kappa_k(\mathbf{x}_i, \mathbf{x}_j) \right), \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \text{ for } i = 1, \ldots, n; \ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned} \tag{A.6}$$

Note that if we define $\boldsymbol{\alpha}$ as the vector of α_i, and we also vectorize the y_is as $\mathbf{y}$, we can rewrite (A.6) in a more compact form as

$$\begin{gathered}\underset{\sigma\in\boldsymbol{\Delta}}{\text{minimize}}\ \underset{\alpha}{\text{maximize}}\quad \left\{\mathbf{1}^T\boldsymbol{\alpha}-\frac{1}{2}\left(\boldsymbol{\alpha}\circ\mathbf{y}\right)^T\left(\sum_{k=1}^{m}\sigma_k K_k\right)\left(\boldsymbol{\alpha}\circ\mathbf{y}\right)\right\},\\ \text{subject to}\quad 0\leq\alpha_i\leq C \text{ for } i=1,\ldots,n;\ \boldsymbol{\alpha}^T\mathbf{y}=0,\end{gathered} \tag{A.7}$$

where Δ is the domain of σ and $\circ$ denotes the Hadamard product. As mentioned before, the MKLSVM problem is a generalization of the SKSVM problem, and it reduces to the SKSVM problem when the weights are all assumed constant.

To solve the minimax optimization problem at (A.7), we use an alternating optimization procedure proposed by Xu et al.[9] The procedure, termed MKL *group lasso* (MKLGL), iteratively solves the inner maximization, then computes the weights to solve the outer minimization; MKLGL repeats this process until convergence. Conveniently, this procedure also has a closed form solution for solving the outer minimization in (A.7) given as

$$\sigma_k=\frac{f_k^{2/(1+p)}}{\left(\sum_{k=1}^{m}f_k^{2p/(1+p)}\right)^{1/p}},\ k=1,...,m; \tag{A.8a}$$

$$f_k=\sigma_k^2(\boldsymbol{\alpha}\circ\mathbf{y})^T K_k(\boldsymbol{\alpha}\circ\mathbf{y}). \tag{A.8b}$$

Algorithm 6 outlines the method with which the MKLGL classifier is trained.

Algorithm 6: MKLGL Classifier Training[9]

Data: $(\mathbf{x}_i, y_i)$ - feature vector and label pairs; K_k - kernel matrices
Result: $\boldsymbol{\alpha}$ - MKLGL classifier solution
Initialize $\sigma_k = 1/m,\ k = 1, ..., m$ - set kernel weights equal
while *not converged* **do**
    Solve unbalanced SKSVM for kernel matrix $K = \sum_{k=1}^{m} \sigma_k K_k$ for the optimal solution $\boldsymbol{\alpha}$
    Update the kernel weights, σ_k using (A.8)

Appendix B

Tibshirani's Lasso Algorithm

Tibshirani proposed an iterative method of solving the ℓ_1-regularization problem in his seminal lasso regression work [84]. That method is summarized here for the case of a general objective function with ℓ_1-regularization, or

$$\min_{\mathbf{x}} J(\mathbf{x}) + \lambda \|\mathbf{x}\|_1, \tag{B.1}$$

where $\mathbf{x} \in \mathbb{R}^N$. We start by first noting that the minimization in (B.1) can be rewritten as

$$\min_{\mathbf{x}} J(\mathbf{x}), \quad \text{s.t.} \quad \|\mathbf{x}\|_1 \leq t, \tag{B.2}$$

where $t \propto 1/\lambda$. Tibshirani noted that this problem can be equivalently stated as

$$\min_{\mathbf{x}} J(\mathbf{x}), \quad \text{s.t.} \quad G\mathbf{x} \leq \mathbf{t}, \tag{B.3}$$

where $G \in \mathbb{R}^{2^N \times N}$ and the term $G\mathbf{x}$ represents all possible linear combinations of $\mathbf{x}$ with unit coefficients. For example, if $N = 2$ then

$$G = \begin{bmatrix} +1 & +1 \\ +1 & -1 \\ -1 & +1 \\ -1 & -1 \end{bmatrix}, \tag{B.4}$$

and since the number of rows of G grows as 2^N, the number of constraints quickly becomes intractable. To address this, an iterative algorithm is applied where constraints are added sequentially. The following section describes the use of this algorithm to solve (B.1).

B.0.1 Summary of Algorithm

First, solve the unconstrained/unregularized problem

$$\hat{\mathbf{x}}_0 = \min_{\mathbf{x}} J(\mathbf{x}), \tag{B.5}$$

let $\boldsymbol{\delta}_i = \text{sign}(\hat{\mathbf{x}}_i)$, and

$$G_i = \begin{bmatrix} \boldsymbol{\delta}_0^T \\ \boldsymbol{\delta}_1^T \\ \vdots \\ \boldsymbol{\delta}_i^T \end{bmatrix}.$$

Then solve the constrained/regularized problem

$$\hat{\mathbf{x}}_i = \min_{\mathbf{x}} J(\mathbf{x}), \quad \text{s.t.} \quad G_{i-1}\hat{\mathbf{x}}_{i-1} \leq t, \tag{B.6}$$

until $\boldsymbol{\delta}_i^T \hat{\mathbf{x}}_i \leq t$.

Appendix C

Letters of Permission

Permission letter for content in Chapter 5.

Permission letter for content in Chapter 2.

Permission letter for content in Chapter 2.