The Pennsylvania State University
The Graduate School
Eberly College of Science

NON-PARAMETRIC FINITE MULTIVARIATE MIXTURE MODELS WITH APPLICATIONS

A Dissertation in
Statistics
by
Xiaotian Zhu

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2014

The dissertation of Xiaotian Zhu was reviewed and approved* by the following:

DAVID R. HUNTER
Professor of Statistics, Statistics Department Head
Dissertation Advisor, Chair of Committee

BRUCE G. LINDSAY
Professor of Statistics

RUNZE LI
Professor of Statistics

LE BAO
Assistant Professor of Statistics

JOHN C. LIECHTY
Professor of Marketing and Statistics, Partner at In4mation Insights

STEPHANIE T. LANZA
Scientific Director and Senior Research Associate, The Methodology Center
Research Associate Professor, College of Health and Human Development

ALEKSANDRA B. SLAVKOVIĆ
Associate Professor of Statistics, Chair of Graduate Studies in the Department of Statistics

*Signatures are on file in the Graduate School.

Abstract

This research set out to investigate and build upon the foundation for the non-parametric estimation of finite multivariate mixture models given the conditional independence assumption, set forth in a series of studies over the last decade. We proposed a novel formulation of the objective function in terms of penalized smoothed Kullback-Leibler divergence under a reduced parameter space. A special optimization landscape and scheme was discovered in working out the majorization-minimization method for the estimation problem which leads to a closed form of the nonlinearly smoothed majorization-minimization (NSMM) algorithm. We established a sharpened monotonicity property that precisely measures the distance between successive iterates of the algorithm and proved the existence of a solution to the main optimization problem for the first time in literature. The estimation theory for this basic model together with the special optimization scheme can be adapted to the investigation of an important extension of the model that incorporates component-wise independent component analysis (ICA). The NSMM-ICA algorithm has been developed and a discretized version of it, which interweaves NSMM and weighted FastICA has been implemented in the R package **icamix** as a model-based clustering tool. We demonstrated the use of the newly developed

methods/algorithms by applications in image analysis and unsupervised learning.

Table of Contents

List of Figures

List of Tables

List of Symbols

r	dimension of data, p. 2
m	number of clusters, p. 2
n	number of data points, p. 2
i, j, k	indices, p. 2
f_{mix}	mixture density, p. 2, 46
$f_1, ..., f_r$	component densities, p. 2
$f_{i,k}$	component marginal densities, 2
$\lambda_1, ..., \lambda_r$	mixing weights, p. 2
$e_1, ..., e_r$	weighted component densities, p. 16
$x_1, ..., x_r$	real valued scalars, p. 2
$\mathbf{x}$	vectors consiting of real valued scalars, p. 2
$X_1, ..., X_r$	random variables, p. 2
$\mathbf{X}$	random vector, p. 2
A	unmixing matrix in ICA model, p. 46
Ω	subset in R^r, p. 16
$\Theta^{(p)}$	parameters that determine the estimation from the (p+1) step, p. 44
$\mathcal{N}_h$	smoothing operator with bandwidth h, p. 17

Acknowledgments

Foremost, I would like to express my sincere gratitude and appreciation to my advisor Prof. David R. Hunter for introducing me to my dissertation topic, for his continuous supporting of my Ph.D study and research, for his genorosity, warm heart, encouragement, patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis and also with my professional development. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Bruce G. Lindsay, Prof. Runze Li, Prof. Le Bao, Prof. John C. Liechty and Prof. Stephanie T. Lanza for their instruction, help, encouragement and insightful comments.

Chapter 1

Introduction

1.1 Overview

Over the last decade, a series of studies (Bordes et al., 2007; Benaglia, Chauveau, & Hunter, 2009; Benaglia et al., 2011; Levine et al., 2011; Chauveau, Hunter, & Levine, 2014) which explore the use of expectation-maximization (EM) and its generalization called majorization-minimization (MM), for the nonparametric estimation for conditional independence multivariate finite mixture models, has advanced with algorithmic and theoretical development. This thesis follows up on this line of research, and proposes a simplified theoretical grounding that we believe will further the understanding of the estimation theory and shed light on establishing true rate of convergence and proving asymptotic results. The framework has led us to find an improved descent property and helped prove the existence of a solution to the main optimization problem for the first time. Also theoretical justification has been given to the nonlinear smoothed majorization-minimization algorithm (NSMM) through deriving it from this new perspective. In addition we explore an approach that extends the model and algorithm to utilize independent

component analysis (ICA), which can be regarded as a model-based clustering method. The R package **icamix**, which is available on the comprehensive R archive network (CRAN), has been developed for the fast implementation of the algorithms considered.

1.2 Theoretical Background

1.2.1 Conditional Independence Multivariate Finite Mixture Models

Conditional independence multivariate finite mixture models have fundamental importance in both statistical theory and applications (Laird & Ware, 1982; Hall & Zhou, 2003; Bonhomme et al., 2011). The basic setup assumes that r-dimensional vectors $\mathbf{X}_i = (X_{i,1}, X_{i,2}, ..., X_{i,r})^T$, $1 \leq i \leq n$, is a simple random sample from a finite mixture of $m > 1$ components with positive mixing proportions $\lambda_1, \lambda_2, ..., \lambda_m$ that sum up to 1, and density functions $f_1, f_2, ..., f_m$, respectively. Here, m is known. Furthermore, the conditional independence assumption, which arises naturally in analysis of data with repeated measurements, says each f_j, $1 \leq j \leq m$, is equivalent to the product of its marginal densities $f_{j,1}, f_{j,2}, ..., f_{j,r}$. So the mixture density is

$$f_{\text{mix}}(\mathbf{x}) = \sum_{j=1}^{m} \lambda_j f_j(\mathbf{x}) = \sum_{j=1}^{m} \lambda_j \prod_{k=1}^{r} f_{j,k}(x_k), \tag{1.1}$$

for any $\mathbf{x} = (x_1, ...x_r)^T \in R^r$. This is often regarded as a semi-parametric model with λ_j, $1 \leq j \leq m$ being the Euclidean parameters and $f_{j,k}$, $1 \leq j \leq m$, $1 \leq k \leq r$ being the functional parameters. Let θ denote all of these parameters. Note that this model is different from the type of mixture models studied in Lindsay (1995)

where the mixing distribution is completely unspecified but the component densities are known to come from a parametric family.

The identifiability for the parameters in the model (1.1) was not clear until the breakthrough in Hall & Zhou (2003) which established the identifiability when $m = 2$ and $r \geq 3$. Some follow-up works appeared, for example, Hall et al. (2005) and Kasahara & Shimotsu (2009), until the fundamental result that established generic identifiability of (1.1) for $r \geq 3$ was obtained (Allman et al., 2009) based on an algebraic result of Kruskal (1976, 1977).

Bordes et al. (2007) proposed a stochastic nonparametric EM algorithm (npEM) estimation algorithm for the estimation of semiparametric mixture models. Benaglia, Chauveau, & Hunter (2009) and Benaglia et al. (2011) proposed a deterministic version of the algorithm for the estimation of (1.1) and studied bandwidth slection related to it. However, all these algorithms lack an objective function as well as the descent property which chracterizes any traditional EM algorithm (Dempster et al., 1977). A significant improvement comes from Levine et al. (2011) which proposes a smoothed likelihood as the objective function which led to a smoothed version of the npEM that does possesses the desired descent property. The authors point out the similarity between their approach and the one in Eggermont (1999) for non-mixtures. However, the formulation of the objective function is not satisfactory because the parameter space has a lot of redundancy that makes the optimization difficult. Rigorous justification is needed in the derivation of the algorithm, and the existence of a solution to the main optimization problem had not been shown. Also the descent property can be improved. These issues will be addressed by the current paper.

Other than the above line of works based on EM and MM, different approaches to the estimation of (1.1) can be found in Bonhomme et al. (2011) and Kasahara

& Shimotsu (2014).

There is also a large literature on variations or special cases of the model (1.1). A version where all the marginal densities are the same for the same component is considered in Hettmansperger & Thomas (2000), Elmore & Wang (2003) and Cruz-Medina & Hettmansperger (2004). For univariate cases, Bordes et al. (2006) and Hunter et al. (2007) have considered identifiability and estimation for a univariate location-shift semiparametric mixture model with symmetic component densities. Silverman (1986) has studied the univariate mixture of the two distributions where only one of them is unknown and its application to local false discovery rate (FDR) estimation. Further development along this line of research on semiparametric mixture models which rely on special assumptions can be seen from Bordes & Vandekerkhove (2010), Chauveau, Saby, et al. (2014), and Hohmann & Holzmann (2013).

1.2.2 Independent Component Analysis

In Chapter 3 we will study extension of the conditional independence multivariate finite mixture model to incorporate component ICA (independent component analysis). ICA, first treated in Jutten & Herault (1991), is a statistical and computational method for finding a suitable representation of a set of random variables by linear transformation that gives statistically independent components, each of which is often an interesting but hidden feature underlying the corresponding data set. Theoretically, it is of interest in statistics and information theory. As a technique, it is suited for unsupervised classification. ICA has important applications in fields where it was originated, namely, neural networks and blind source separation in signal processing (Hyvarinen et al., 2002). In addition, applications of ICA can

be found in other fields such as psychometry, finance, biometry, coding and text analysis, and many more (Naik & Kumar, 2011).

ICA involves a generative model involving latent variables. Let $S_1, S_2, ..., S_r$ be unobserved non-Gaussian statistically independent random variables, or components. Suppose some non-degenerate linear combination of these independent components generate the observed data $X_1, X_2, ..., X_r$, so that

$$X_l = \sum_{k=1}^{r} a_{l,k} S_k \qquad \text{for} \quad l = 1, ..., r. \tag{1.2}$$

The mixing coefficients $a_{l,k}$ are also unobserved. The goal of ICA is the estimation of both the mixing coefficients and independent components, given the observed data. Under fairly weak regularity conditions, it can be shown that Model (1.2) is identifiable (Comon, 1994; Tong et al., 1993). Alternatively, we may view ICA as a model-free procedure, which does not assume any statistical model for the data. The purpose of ICA is then to find a linear decomposition which minimizes certain dependence measure, such as mutual information among the recovered components.

ICA relies on higher-order statistics that fully reveal the dependence structure of multivariate data sets in general (Nandi, 1999), while the more traditional PCA (principal component analysis) relies on second-order statistics, the mean and covariance.

Estimation methods for ICA have been developed based on the infomax principle, maximum non-Gaussianity, maximum likelihood or minimum mutual information, tensorial methods, nonlinear decorrelation, and nonlinear PCA (Hyvarinen et al., 2002). In the following, we will review the FastICA algorithm (Hyvarinen, 1999) and a novel weighted version will be developed and integrated into our computation in Chapter 3. The algorithm uses a fixed-point iteration scheme to

maximize measures of non-Gaussianity which leads to maximization of statistical independence. Alternatively, it can be viewed as an approximative Newton iteration.

Let $\mathbf{X} = (X_1, ..., X_r)^T$, $\mathsf{A} = \{a_{l,k}\}_{1 \leq l,k \leq r}$ and $\mathbf{S} = (S_1, ..., S_r)^T$. Three assumptions will be made for ICA to be successful: 1) $S_1, ..., S_r$ are statistically independent; 2) All S_k, $1 \leq k \leq r$, except at most one, have non-Gaussian distributions. This means that the corresponding higher-order cumulants are non-zero; 3) The matrix A is invertible. Under these assumptions, A and the distribution of $\mathbf{S}$ are identifiable up to permutation and/or rescaling of the independent components $S_1, ..., S_r$. To proceed with ICA, first the observed vector needs to be centerd and "whitened", which means centered and standardized to have mean zero and covariance matrix the identity. An eigenvalue decomposition technique can be employed for this whitening step, which transforms $\mathbf{X}$ to $\mathbf{Z}$, the rows of which are same as the independent components up to an orthogonal transformation of which the dimension is approximately $\frac{1}{2}r^2$. Considering the dimension of A which is r^2, we see that the whitening step has reduced the complexity of the problem. After this step is done, we continue with ICA by finding out the orthogonal transformation, W, which results in components, as rows in WZ, with maximized non-Gaussianity. Letting $\mathsf{W} = [\mathbf{w}_1, ..., \mathbf{w}_r]^T$, this can be done by iteratively applying the transformations

$$\mathbf{w} \leftarrow E\{\mathbf{X}g(\mathbf{w}^T\mathbf{X})\} - E\{\mathbf{X}g\prime(\mathbf{w}^T\mathbf{X})\}\mathbf{w}, \tag{1.3}$$

$$\mathbf{w} \leftarrow \mathbf{w}/\|\mathbf{w}\|, \tag{1.4}$$

where $g(y)$ is the derivative of a nonquadratic function such as $\frac{1}{a}\log\cosh(ay)$ or $-\exp(-y^2/2)$ which is chosen for robust estimation. We can compute the vectors $\mathbf{w}_1, ..., \mathbf{w}_r$ one after another, which results in the so-called deflationary

orthogonalization, or we can compute all r of them by updating them all at the same time at each iteration which results in the so-called symmetric orthogonalization. In Equation (1.3), expectation will be replaced by its sample estimate when dealing with real data.

1.3 Multivariate Model-Based Clustering

The models and algorithms that are studied in Chapter 2, and especially their extensions to incorporate ICA methods proposed in Chapter 3, can be considered interesting approaches to semiparametric multivariate model-based clustering under weak assumptions on the components, namely, that each has independent coordinates, or can be linearly transformed to have independent coordinates. Hence, we will look into the related literature on multivariate model-based clustering to see how our work contributes in this context.

Cluster analysis (Hartigan, 1975; Kaufman & Rousseeuw, 2009) is usually considered as one of several general approaches to the subject of unsupervised learning, that is, without any labels given, as is different from "discriminant analysis". It is also termed as "unsupervised classification". The main task is to identify and estimate the underlying cluster structure of the population and to assign each observation to one of these clusters, in either a hard way (i.e., with probability 1), or a soft way (i.e., with probability between 0 and 1), just by knowing the values or attributes of those observations.

In practice, clustering is often based on heuristic ideas and intuitive measures, and without assuming any probability model. For example, hierarchical clustering builds a cluster tree by using a linkage (connectivity) criterion that specifies dissimilarity between groups of observations (Hastie et al., 2009). K-means clustering is

another example that is not based on any statistical model (MacQueen et al., 1967). It is centroid-based, where clusters are represented by a central vector and the algorithm aims at minimizing within-cluster sums of squares of distances. Unsupervised nearest neighbors is another example of a non-model-based clustering technique. Strategies for determining the number of clusters for the above mentioned methods have been surveyed by Bock (1996).

Model-based clustering (Fraley & Raftery, 1998; Banfield & Raftery, 1993; Fraley & Raftery, 2002) veiws data as coming from a mixture of probability distributions, each representing a cluster. Some mixture modeling structure, such as a finite mixture model, is assumed and the group membership is learned by iterative algorithms which are often variations of the expectation-maximization method (Dempster et al., 1977). Since Wolfe (1963), one of the first to have presented model-based cluster analysis, there has been a growing, and by now very large, literature on model-based clustering techniques. Gaussian mixtures and hidden Markov chains are two basic model structures employed in such approaches. Various applications of multivariate Gaussian mixtures can be found in Murtagh & Raftery (1984), Dasgupta & Raftery (1998), Campbell et al. (1997) and Mukherjee et al. (1998). Fraley (1998) proposed efficient algorithms for hierarchical clustering with various parameterizations of Gaussian mixture models. Classification likelihood-based hierarchical agglomeration (Banfield & Raftery, 1992) is known to produce reasonably good partitions even without any information about the underlying mixtures. This can complement EM algorithms for maximum likelihood estimation of multivariate mixture models (Celeux & Govaert, 1995; McLachlan & Basford, 1988), which is known to be sensitive to initialization because of local modes and often produces improved results given a good starting partition. A brief review of literature on using EM variants in developing algorithms in this context and utilizing

the Bayesian Information Criterion (BIC) for model comparison and selection of parameterization as well as number of mixture components can be found in Fraley & Raftery (1998). Variable selection via BIC for model-based clustering is further discussed by Raftery & Dean (2006). Zhong & Ghosh (2003) presents a unified framework for probabilistic model-based clustering relying on a bipartite graph view of data and models that highlights the commonalities and differences among most model-based clustering algorithms.

In recent years, with the rise of computational power, people have seen more studies focusing on semi-parametric or non-parametric model-based clustering. A semiparametric model-based clustering analysis for DNA microarray data can be found in Han & Davis (2006). Azzalini & Torelli (2007) proposed nonparametric density estimation for clustering via identification of subpopulations with regions with high density of the underlying probability distribution. Delaunay triangulation is utilized in this study. Li et al. (2007) developed a new clustering approach based on mode identification by applying new optimization techniques to a nonparametric density estimator. Vichi (2008) fits semiparametric clustering models to dissimilarity data. In Zhang et al. (2009) a semiparametric model is introduced to account for varying impacts of factors over clusters by using cluster-level covariates. Mallapragada et al. (2010) proposes a non-parametric mixture model (NMM) for data clustering. Guglielmi et al. (2014) fitted Bayesian semiparametric logit models to grouped data of in-hospital survival outcomes of patients hospitalised with ST-segment Elevation Myocardial Infarction diagnosis. Mixture of linear regressions also falls under the category of semiparametric model based clustering. Hunter & Young (2012) presents an algorithm for estimating parameters in a mixture-of-regressions model in which the errors are assumed to be independent and identically distributed but no other assumption is made. The method is applied to analyze

Cohen's tone data. Some related studies can be found in De Veaux (1989), Viele & Tong (2002), Mallick et al. (2002), Ng & McLachlan (2003), Lu & Peng (2008) and Bordes et al. (2013). Huang et al. (2013) proposes nonparametric finite mixture of regression models for analysis of U.S. house price index (HPI) data. Vandekerkhove (2013) studies estimation of a semiparametric mixture of regressions model of two components with one being known. Bajari et al. (2011) views a game abstractly as a semiparametric mixture distribution and studies the semiparametric efficiency bound of this model. Butucea & Vandekerkhove (2014) considers a semiparametric mixture of two distributions equal up to a shift parameter.

1.4 Motivating Applications

Piaget & Inhelder (1956) proposed a water level experiment to assess children's understanding of spatial concepts. In the experiment, each child was given a bottle one-quarter filled with water and asked to predict the direction of the water level when the bottle was put in various oblique orientations. They concluded that children in early childhood have little understanding of the concepts of horizontality and verticality, which is believed to be developed through well-defined stages and is only fully achieved in later years. Thomas et al. (1993) raised questions about Piaget's theory and went on to conduct more systematic studies of this water level experiment, which focus on investigating the development of horizontality and evaluating Piaget's stages while allowing for differences due to age, sex, bottle shape and bottle orientation. In conclusion, they pointed out that children's acquisition of horizontality is apparently more complex than has been believed and depends on more than muturation of cognitive structure and repeated experiences. One interesting finding is that there are systematic differences among subgroups,

so it makes sense to consider decomposition of the samples into several clusters. The first mixture model based analysis of Thomas' water level data set is given by Hettmansperger & Thomas (2000), which assumed normality and identically distributed coordinates and used a single cut point (Cruz-Medina et al., 2004) to convert the continuous angle measurements into binomial data. Elmore et al. (2004) presents an extension of this analysis which uses multiple cut points that transform the angle measurements into multinomial data. Benaglia, Chauveau, & Hunter (2009) points out the presence of differences among the coordinate distributions, and proposes grouping the coordinates into four blocks of two i.i.d. coordinates and presents analysis of the water level data by the npEM algorithm based on conditional non-parametric finite multivariate mixture models. Analyses with three or four mixing components are both presented. The results show the data points are sensibly classified into groups that no previous analysis was able to reveal. In Chapter 4 we will present results from an analysis of the data by the NSMM-ICA algorithm we developed which is based on the more general non-parametric ICA mixture model which does not assume conditional independence among the coordinates.

Cruz-Medina et al. (2004) applies mixture model methodology to a reaction time data analysis that is common in a variety of domains in developmental psychology. This same psychometric data set is also analyzed by Leung & Qin (2006), which develops an exponential tilt model-based semi-parametric method for fitting multivariate mixtures under the assumption of conditional independence. Except for the exponential tilt assumption, the marginal distributions of the observations are completely arbitrary. The results show that the reaction time distributions for different tasks are not all the same due to the relaxation of the assumption of identical coordinates. In Chapter 4 we will present results from an

analysis of the data by our NSMM-ICA algorithm that does not assume conditional independence, but rather models it by incorporating ICA.

Lee et al. (1999) and Lee et al. (2000) propose parametric ICA Mixture Models with algorithms based on the infomax principle for various unsupervised classification problems. Detailed examples include automatic context switching in blind signal separation, iris data classification, and learning efficient codes for images containing both natural scenes and text. In the last application, pixel patches are randomly selected from two different type of images, and the task is to recover their class membership while at the same time to learn basis class functions. This problem is of great interest in the area of image processing. The authors claim that, compared to previous approaches for these problems, a higher degree of flexibility is achieved by the proposed model, which shows promise for modelling non-Gaussian structure in high-dimensional data and has many potential applications. Supporting this claim, Shah et al. (2004) applies the ICA mixture model methodology to the problem of unsupervised classification of hyperspectral or multispectal imagery where image data are captured at multiple or a continuous range of frequencies across the electromagnetic spectrum. This is an important application of remote sensing and land cover classification. Palmer et al. (2008) derives an asymptotic Newton algorithm for Quasi-Maximum Likelihood estimation of the parametric ICA mixture model and presents its application to EEG segmentation. We will present an application of our NSMM-ICA algorithm based on non-parametric ICA mixture models for learning efficient code of images, and another application for classification of multispectal images in Chapter 4. The results are quite encouraging.

As a motivating example in marketing research, the estimation of Beauty Contest Auctions (Yoganarasimhan, 2013) further elucidates the potential use of our estimation framework for non-parametric multivariate mixtures with conditional

independence. A beauty contest auction is a reverse type of auction in which the buyer first posts the auction and its aspects, which become accessible to prospective sellers in a market. Then, the sellers submit their bids to the auction before the buyer makes a final decision to select one of the bids or reject them all. The name 'beauty contest' comes from the fact that in beauty pageants, there is no deterministic scoring rule by which a winner is picked. In fact the final decision could be based on subjective judgement and could take into consideration many attributes. So there is no closed-form strategy for bidding. Furthermore, unobserved auction heterogeneity exists for almost all beauty contest auctions. To recover the unobservable auction types and the distribution of bids, i.e., the equilibrium bidding strategy of sellers, a non-parametric npEM algorithm similar in spirit to that developed by Benaglia, Chauveau, & Hunter (2009), can be employed. These first step results can then be used to estimate cost distribution in a second step which helps in understanding the market power that sellers have, how competitive the market is, and the efficiency of the marketplace. It is mentioned in Yoganarasimhan (2013) that we can use the algorithm proposed in Levine et al. (2011), which in practical terms is equivalent to our NSMM algorithm and has the desirable descent property. This framework was applied to data from a leading freelance firm and also data from a counter-factual simulation in Yoganarasimhan (2013) to obtain useful inference and guidelines for managers of online freelance firms. However, one thing we should be cautious about here is that the model in Yoganarasimhan (2013) is not exactly a non-parametric multivariate mixture as defined in this thesis, and the npEM algorithm is similar in spirit to the one developed by Benaglia, Chauveau, & Hunter (2009) but not the same. Nevertheless, this example provides clear evidence of potential applications of the framework of our proposed estimation method and algorithms in marketing science.

1.5 Outline of Thesis

The remaining contents of the thesis are structured as following. In Chapter 2 a novel estimation theory under a reduced parameter space with many fewer constraints for the basic conditional independence model (1.1) will be established. A closed form of the NSMM algorithm will be derived by following the majorization-minimization method, as a result of a newly derived optimization scheme. A sharpend monotonicity property of NSMM will be shown, which plays a crucial part in the subsequent proof of existence of at least one solution to the estimation problem. Some other properties of NSMM and the solution will be discussed as well. In Chapter 3 we will adapt the new theory for the basic model to the investigation of an extension of the model called the non-parametric ICA mixtures. Justification for bringing in ICA techniques will be discussed. The NSMM-ICA algorithm will be derived and a discretized version of it, the NSMM-FastICA algorithm, will be developed and implemented in the R package '**icamix**' which can be downloaded from CRAN. An iterative scheme for choosing bandwidth for each component and coordinate is dicussed and implemented. In Chapter 4 simulation studies on both Gaussian and non-Guassian mixture components are carried out. Also, the use of the proposed methodology as a new approach to model-based clustering will be demonstrated using real datasets for testing unsupervised classifiers as well as applications such as learning efficient code for images in the area of image processing. Finally, in Chapter 5 a summary of the thesis as well as a plan for future research will be presented.

Chapter 2

Estimation Theory

2.1 Nonparametric Estimation

The distinction between the estimation approach we propose here and that of Levine et al. (2011) may appear subtle, but actually it has critical implications for subsequent studies. On one hand, the new approach eliminates the Euclidean parameter $\boldsymbol{\lambda}$ becuase it is not necessary to explicitly require that the e_j, $1 \leq j \leq m$, sum up to a proper probability density function. On the other hand, the main optimization problem formulated here is truly a penalized smoothed Kullback-Leibler divergence. Levine et al. (2011) discussed the optimization as a penalized smoothed Kullback-Leibler divergence minimization problem, but the formulation in our approach seems purer. Comparing the regularity conditions of the two approaches, we have added boundedness of the kernel function of the smoother $\mathcal{N}_h$, which is used in the proof of existence of a solution to the main optimization problem.

In the following, we first consider an ideal setting where the target density is known (i.e., the sample size is infinity). Then we replace the target density by its

empirical version and obtain the discrete algorithm.

2.1.1 Setup and Notation

Let $\mathbf{x} = (x_1, x_2, \cdots, x_r)^T \in R^r$. Let Ω be a compact and convex set in R^r. Without loss of generality, assume Ω is the closed r-dimensional cube $[a, b]^r$. Let g denote a target density on R^r, with support in the interior of Ω. We are interested in the case when g is a finite mixture of products of fully unspecified univariate measures, with unknown mixing parameters.

We make the following assumptions:

(i) Let the number of mixing components in g be fixed and denoted by m. There exist non-negative functions $e_j(x)$, $1 \leq j \leq m$, such that

$$g(x) = \sum_{j=1}^{m} e_j(x). \tag{2.1}$$

(ii) For each $1 \leq j \leq m$,

$$e_j(\mathbf{x}) = \theta_j \prod_{k=1}^{r} e_{j,k}(x_k), \tag{2.2}$$

where $\theta_j > 0$ and for each k, $1 \leq k \leq r$, $e_{j,k} \in L^1(R)$ is positive with support in $[a, b]$. Hence each $e_j(\mathbf{x})$ is in $L^1(R^r)$, positive, and with support in Ω.

Given a bandwidth $h \in R$, let $s_h(\cdot, \cdot) \in L^1(R \times R)$ be nonnegative and with support in $[a, b] \times [a, b]$, such that

(iii) For $v, z \in R$,

$$\int s_h(v, z)\mathrm{d}z = \int s_h(v, z)\mathrm{d}v = 1. \tag{2.3}$$

(iv) There exist positive numbers $M_1(h)$ and M_2 such that for any $v, z \in [a, b]$,

$$M_1(h) \leq s_h(v, z) \leq M_2. \tag{2.4}$$

(v) The function s_h has continuous first-order partial derivatives on $(a, b) \times (a, b)$ and there exists a constant B such that for any $u, x \in (a, b)$,

$$\left| \frac{\partial}{\partial v} s_h(v, z)|_{v=u} \right| \leq B \qquad \text{and} \qquad \left| \frac{\partial}{\partial z} s_h(v, z)|_{z=x} \right| \leq B. \tag{2.5}$$

(vi) If we define $f_j(\mathbf{x}) = e_j(\mathbf{x}) / \int e_j(\mathbf{z}) d\mathbf{z}$, then

$$f_j(\mathbf{x}) \geq (M_1(h))^r, \tag{2.6}$$

for all $\mathbf{x} \in \Omega$ and for each $j \in \{1, 2, ..., m\}$.

Before stating the optimization problem, we define the smoothing operators, S_h, S_h^* and $\mathcal{N}_h$, as follows.

For any $f \in L^1(R^r)$, let

$$(S_h f)(\mathbf{x}) = \int \tilde{s}_h(\mathbf{x}, \mathbf{u}) f(\mathbf{u}) \mathrm{d}\mathbf{u} \quad \text{and} \quad (S_h^* f)(\mathbf{x}) = \int \tilde{s}_h(\mathbf{u}, \mathbf{x}) f(\mathbf{u}) \mathrm{d}\mathbf{u}, \tag{2.7}$$

where

$$\tilde{s}_h(\mathbf{x}, \mathbf{u}) = \prod_{k=1}^{r} s_h(x_k, u_k) \qquad \text{for } \mathbf{x}, \mathbf{u} \in R^r. \tag{2.8}$$

Furthermore, let

$$(\mathcal{N}_h f)(\mathbf{x}) = \begin{cases} \exp[(S_h^* \log f)(\mathbf{x})] & \text{for } \mathbf{x} \in \Omega, \\ 0 & \text{elsewhere.} \end{cases} \tag{2.9}$$

These smoothing operators are well-known and have many desirable properties (Eggermont, 1999). For instance, Lemma 1.1 of Eggermont (1999) states the following:

Proposition 2.1.1. *For any nonnegative functions g_1 and g_2 in $L^1(R^r)$,*

$$KL\left(S_h g_1, S_h g_2\right) \leq KL\left(g_1, g_2\right), \tag{2.10}$$

Where KL is the Kullback-Leibler divergence defined by

$$KL\left(g_1, g_2\right) = \int \left[g_1 \log \frac{g_1}{g_2} + g_2 - g_1 \right]. \tag{2.11}$$

2.1.2 The Projection Operator P

Let us introduce notation to facilitate further discussion. For any nonnegative function f on R^r such that $\int f > 0$, and $x = (x_1, x_2, \cdots, x_r)^T \in R^r$, let the operator P, which factorizes f as a product of marginal functions on R^r, be defined as

$$(Pf)(\mathbf{x}) = \frac{\left[\prod_{k=1}^{r} \int_{R^{r-1}} f(\mathbf{x}) \mathrm{d}x_1 \mathrm{d}x_2 \cdots \mathrm{d}x_{k-1} \mathrm{d}x_{k+1} \cdots \mathrm{d}x_r \right]}{\left[\int f\right]^{(r-1)}}. \tag{2.12}$$

When f is a density on R^r, the right side of (2.12) simplifies because the denominator is 1. The P operator is interesting in its own right, and this section concludes with proofs of two of its properties.

Proposition 2.1.2. *Assume f is an integrable nonnegative function on R^r with support in a compact set Ω. We have*

$$(P \circ S_h) f = (S_h \circ P) f. \tag{2.13}$$

Proof. Since $(P \circ S_h)$ is linear, we only need to prove the case where f is a density function. By Fubini's Theorem and Equation (2.12),

$$
\begin{aligned}
&[(P \circ S_h)f](\mathbf{x}) \\
&= \prod_{k=1}^{r} \int_{R^{r-1}} \left(\int_{R^r} \tilde{s}_h(\mathbf{x}, \mathbf{u}) f(\mathbf{u}) \mathrm{d}\mathbf{u} \right) \mathrm{d}x_1 \mathrm{d}x_2 \cdots \mathrm{d}x_{k-1} \mathrm{d}x_{k+1} \cdots \mathrm{d}x_r \\
&= \prod_{k=1}^{r} \int_{R^r} \left(\int_{R^{r-1}} \tilde{s}_h(\mathbf{x}, \mathbf{u}) f(\mathbf{u}) \mathrm{d}x_1 \mathrm{d}x_2 \cdots \mathrm{d}x_{k-1} \mathrm{d}x_{k+1} \cdots \mathrm{d}x_r \right) \mathrm{d}\mathbf{u} \\
&= \prod_{k=1}^{r} \int_{R} s_h(x_k, u_k) \left(\int_{R^{r-1}} f(\mathbf{u}) \mathrm{d}u_1 \mathrm{d}u_2 \cdots \mathrm{d}u_{k-1} \mathrm{d}u_{k+1} \cdots \mathrm{d}u_r \right) \mathrm{d}u_k \\
&= \int_{R^r} \left(\prod_{k=1}^{r} s_h(x_k, u_k) \right) \cdot \prod_{k=1}^{r} \left(\int_{R^{r-1}} f(\mathbf{u}) \mathrm{d}u_1 \mathrm{d}u_2 \cdots \mathrm{d}u_{k-1} \mathrm{d}u_{k+1} \cdots \mathrm{d}u_r \right) \mathrm{d}\mathbf{u} \\
&= [(S_h \circ P)f](\mathbf{x}).
\end{aligned}
$$

□

Proposition 2.1.3. *For any densities f and g on R^r,*

$$\int f(\mathbf{x}) \log (Pg)(\mathbf{x}) d\mathbf{x} = \int (Pf)(\mathbf{x}) \log (Pg)(\mathbf{x}) d\mathbf{x}. \tag{2.14}$$

Proof.

$$
\begin{aligned}
&\int f(\mathbf{x}) \log (Pg)(\mathbf{x}) \mathrm{d}\mathbf{x} \\
&= \sum_{k=1}^{r} \int f(\mathbf{x}) \log (Pg)_k(x_k) \mathrm{d}\mathbf{x} \\
&= \sum_{k=1}^{r} \int (Pf)_k(x_k) \log (Pg)_k(x_k) \mathrm{d}x_k
\end{aligned}
$$

$$= \sum_{k=1}^{r} \int (Pf)(\mathbf{x}) \log (Pg)_k(x_k) \mathrm{d}\mathbf{x}$$
$$= \int (Pf)(\mathbf{x}) \log (Pg)(\mathbf{x}) \mathrm{d}\mathbf{x}.$$

□

2.1.3 Main Optimization Problem

Now, we assume conditions (i) through (vi) and propose to estimate $\mathbf{e}$ by minimizing the function

$$l(\mathbf{e}) = \int g(\mathbf{x}) \log \left[g(\mathbf{x}) / \sum_{j=1}^{m} (\mathcal{N}_h e_j)(\mathbf{x}) \right] \mathrm{d}\mathbf{x} + \int \left[\sum_{j=1}^{m} e_j(\mathbf{x}) \right] \mathrm{d}\mathbf{x} \tag{2.15}$$

subject to these conditions. In fact the only assumptions that impose any constraints on $\mathbf{e}$ are (ii) and (vi). Minimizing of $l(\mathbf{e})$ can be written equivalently as minimization of the penalized smoothed Kullback-Leibler divergence

$$KL\left(g, \sum_{j=1}^{m} (\mathcal{N}_h e_j) \right) + \int \left[\sum_{j=1}^{m} e_j - \sum_{j=1}^{m} (\mathcal{N}_h e_j) \right] (\mathbf{x}) \mathrm{d}\mathbf{x}, \tag{2.16}$$

where in (2.16) the second term acts like a roughness penalty.

The discrete version of the optimization problem replaces $g(\mathbf{x})\mathrm{d}\mathbf{x}$ by $\mathrm{d}G_n(\mathbf{x})$, where G_n is the empirical distribution function of a random sample of size n, and in this case we minimize

$$l_{\text{discrete}}(\mathbf{e}) = -\frac{1}{n} \sum_{i=1}^{n} \log \sum_{j=1}^{m} (\mathcal{N}_h e_j)(\mathbf{x}_i) + \int \sum_{j=1}^{m} e_j(\mathbf{x}). \tag{2.17}$$

Although we do not constraint $\mathbf{e}$ to require that the sum of all e_i is a density as

required by Equation (2.1), this property is guaranteed by the main optimization:

Theorem 2.1.4. *Any solution* $\mathbf{e}$ *to (2.15) or (2.17) satisfies*

$$\int \sum_{j=1}^{m} e_j(\mathbf{x}) = 1. \tag{2.18}$$

Proof. For any fixed $\mathbf{e}$, differentiation shows that the function $l(\alpha\mathbf{e})$ is minimized at the unique value

$$\hat{\alpha} = 1 \Big/ \int \sum_{j=1}^{m} e_j(\mathbf{x}) \, . \tag{2.19}$$

Thus if $\mathbf{e}$ is a minimizer then we must have $\hat{\alpha} = 1$. □

From (2.18), we see that for each $1 \leq j \leq m$, $\int e_j^S$ can be interpreted as the coresponding mixing weight.

2.1.4 NSMM Algorithm

In this section, we derive an iterative algorithm, using majorization-minimization (Hunter & Lange, 2004), to solve (2.15). The algorithm coincides with that of Levine et al. (2011), despite the slightly different derivation.

Given the current estimate $\mathbf{e}^{(0)}$, which satisfies assumptions (ii) and (vi),

$$\begin{aligned}
&l(\mathbf{e}) - l(\mathbf{e}^{(0)}) \\
&\quad = -\int g(\mathbf{x}) \log \sum_{j=1}^{m} \frac{(\mathcal{N}_h e_j^{(0)})(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(0)})\mathbf{x}} \cdot \frac{(\mathcal{N}_h e_j)(\mathbf{x})}{(\mathcal{N}_h e_j^{(0)})(\mathbf{x})} \mathrm{d}\mathbf{x} + \int \left(\sum_{j=1}^{m} e_j - \sum_{j=1}^{m} e_j^{(0)} \right) \\
&\quad \leq -\int g(\mathbf{x}) \sum_{j=1}^{m} \frac{(\mathcal{N}_h e_j^{(0)})(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(0)})(\mathbf{x})} \cdot \log \frac{(\mathcal{N}_h e_j)(\mathbf{x})}{(\mathcal{N}_h e_j^{(0)})(\mathbf{x})} \mathrm{d}\mathbf{x} + \int \left(\sum_{j=1}^{m} e_j - \sum_{j=1}^{m} e_j^{(0)} \right).
\end{aligned} \tag{2.20}$$

So if we let

$$b^{(0)}(\mathbf{e}) = -\int g(\mathbf{x}) \sum_{j=1}^{m} w_j^{(0)}(\mathbf{x}) \cdot \log\left(\mathcal{N}_h e_j\right)(\mathbf{x}) \mathrm{d}\mathbf{x} + \int \left(\sum_{j=1}^{m} e_j\right), \tag{2.21}$$

where

$$w_j^{(0)}(\mathbf{x}) = \frac{(\mathcal{N}_h e_j^{(0)})(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(0)})(\mathbf{x})}, \tag{2.22}$$

then

$$l(\mathbf{e}) - l(\mathbf{e}^{(0)}) \leq b^{(0)}(\mathbf{e}) - b^{(0)}(\mathbf{e}^{(0)}). \tag{2.23}$$

Using the MM algorithm terminology of Hunter & Lange (2004), Inequality (2.23) means tha $b^{(0)}$ may be said to majorizes l at $\mathbf{e}^{(0)}$, up to an additive constant. Minimizing $b^{(0)}$ therefore yields a function $\mathbf{e}^{(1)}$ satisfying

$$l(\mathbf{e}^{(1)}) \leq l(\mathbf{e}^{(0)}), \tag{2.24}$$

so we now consider how to minimize $b^{(0)}(\mathbf{e})$, subject to the assumptions on $\mathbf{e}$ that were stated at the beginning. This is to be done component-wise. That is, for each j, we wish to minimize

$$\begin{aligned} b_j^{(0)}(\mathbf{e}) &= -\int g(\mathbf{x}) w_j^{(0)}(\mathbf{x}) \cdot \int \tilde{s}_h(\mathbf{u}, \mathbf{x}) \log e_j(\mathbf{u}) \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{x} + \int e_j \\ &= -\iint g(\mathbf{x}) w_j^{(0)}(\mathbf{x}) \cdot \tilde{s}_h(\mathbf{u}, \mathbf{x}) \left[\sum_{k=1}^{r} \log e_{j,k}(u_k) + \log \theta_j\right] \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{x} \\ &\qquad + \int \theta_j \prod_{k=1}^{r} e_{j,k}(u_k) \mathrm{d}\mathbf{u}. \end{aligned} \tag{2.25}$$

Up to an additive term that doesn't involve any $e_{j,k}$, (2.25) is

$$-\sum_{k=1}^{r}\iint g(\mathbf{x})w_j^{(0)}(\mathbf{x})\cdot s_h(u_k,x_k)\log e_{j,k}(u_k)\mathrm{d}u_k\ \mathrm{d}\mathbf{x}+\int\theta_j\prod_{k=1}^{r}e_{j,k}(u_k)\mathrm{d}\mathbf{u}. \quad (2.26)$$

For any k among $1,2,\cdots,r$, using Fubini's Theorem, we can view Expression (2.26) as an integral with respect to $\mathrm{d}u_k$. The integrand, up to an additive term that doesn't involve the chosen k, is

$$-\int g(\mathbf{x})w_j^{(0)}(\mathbf{x})\cdot s_h(u_k,x_k)\ \mathrm{d}\mathbf{x}\log e_{j,k}(u_k)+\theta_j\left[\prod_{l\neq k}\int e_{j,l}(u_l)\mathrm{d}u_l\right]\cdot e_{j,k}(u_k). \quad (2.27)$$

Viewing this as a function of the single value $e_{j,k}(u_k)$, we differentiate it with respect to $e_{j,k}(u_k)$ to obtain

$$-\frac{\int g(\mathbf{x})w_j^{(0)}(\mathbf{x})\cdot s_h(u_k,x_k)\ \mathrm{d}\mathbf{x}}{e_{j,k}(u_k)}+\theta_j\left[\prod_{l\neq k}\int e_{j,l}(u_l)\mathrm{d}u_l\right], \quad (2.28)$$

which is an increasing function of the value $e_{j,k}(u_k)$, so setting Expression (2.28) equal to zero does give a minimum. We conclude that the minimizer satisfies

$$\hat{e}_{j,k}(u_k)\propto\int g(\mathbf{x})w_j^{(0)}(\mathbf{x})\cdot s_h(u_k,x_k)\ \mathrm{d}\mathbf{x}. \quad (2.29)$$

This tells us, according to (2.2), that

$$\hat{e}_j(\mathbf{u})=\alpha_j\prod_{k=1}^{r}\int g(\mathbf{x})w_j^{(0)}(\mathbf{x})\cdot s_h(u_k,x_k)\ \mathrm{d}\mathbf{x} \quad (2.30)$$

for some constant α_j. To find α_j, we plug (2.30) into (2.25) and differentiate with

respect to α_j, which gives

$$\alpha_j = \frac{1}{\left[\int g(\mathbf{x}) w_j^{(0)}(\mathbf{x})\ \mathrm{d}\mathbf{x}\right]^{r-1}}, \tag{2.31}$$

and hence

$$\hat{e}_j(\mathbf{u}) = \frac{\prod_{k=1}^{r} \int g(\mathbf{x}) w_j^{(0)}(\mathbf{x}) \cdot s_h(u_k, x_k)\ \mathrm{d}\mathbf{x}}{\left[\int g(\mathbf{x}) w_j^{(0)}(\mathbf{x})\ \mathrm{d}\mathbf{x}\right]^{r-1}}. \tag{2.32}$$

Equation (2.32) gives each majorization-minimization step.

To summarize, our nonlinearly smoothed majorization-minimization (NSMM) algorithm starts with some initial estimate $e^{(0)}$ which satisfies assumptions (ii) and (vi), and iterates according to

$$e^{(p+1)}(\mathbf{u}) = G(e^{(p)})(\mathbf{u}), \tag{2.33}$$

where $G(\cdot)$ performs the one-step update of Equation (2.32). Note that $G(\cdot)$ can also be expressed concisely using the P and S_h operators:

$$\left(G(e^{(p)})\right)_j(\mathbf{u}) = \left[P \circ S_h(g \cdot w_j^{(p)})\right](\mathbf{u}). \tag{2.34}$$

In practical terms, NSMM is identical to the algorithm proposed in Levine et al. (2011). However, we are optimizing in a simpler sapce and the normalization involved in each step of the algorithm is now a result of optimization. In addition, we have rigorously derived the NSMM algorithm as a special case of the majorization-minimization method. In the discrete case, the algorithm iterates according to the following until convergence, assuming $e^{(p)}$ is the current step estimate.

Majorization Step: For $1 \leq i \leq n$, $1 \leq j \leq m$, compute

$$w_j^{(p)}(\mathbf{x}_i) = \frac{(\mathcal{N}_h e_j^{(p)})(\mathbf{x}_i)}{\sum_{j=1}^{m} (\mathcal{N}_h e_j^{(p)})(\mathbf{x}_i)}. \tag{2.35}$$

Minimization Step:

$$e_j^{(p+1)}(\mathbf{u}) = \frac{\prod_{k=1}^{r} \sum_{i=1}^{n} \frac{1}{n} w_j^{(p)}(\mathbf{x}_i) s_h(u_k, x_{ik})}{\left(\sum_{i=1}^{n} \frac{1}{n} w_j^{(p)}(\mathbf{x}_i) \right)^{r-1}}. \tag{2.36}$$

Remark 2.1.5. *For implementing the $\mathcal{N}_h$ operator in the Majorization Step (2.35), we will need numerical convolution.*

2.1.5 Monotonicity Property

For any MM algorithm, including any EM algorithm, the well-known monotonicity property (2.24) says that the value of the objective function moves, at each iteration, toward the direction of being optimized (Hunter & Lange, 2004). For the NSMM algorithm, this descent property was proved in Levine et al. (2011). In this section, we will present a novel result which strengthens Inequality (2.24). The lower bound on $l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)})$, instead of being zero, will be related to a distance measure between density estimates from successive iterations. This nonzero lower bound, we believe, will help in future investigation of the true convergence rate or asymptotic results. Another novel result of this section is the demonstration that any minimizer of $l(\mathbf{e})$ must be a fixed point of the NSMM algorithm.

Lemma 2.1.6. *In the continuous (infinite-sample) version of the NSMM algorithm,*

at any step p, we have

$$l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)}) = \sum_{j=1}^{m} KL(e_j^{(p+1)}, e_j^{(p)}) + \sum_{j=1}^{m} KL(g \cdot w_j^{(p)}, g \cdot w_j^{(p+1)}). \tag{2.37}$$

Proof.

$$\begin{aligned}
&l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)}) \\
&= \int g(\mathbf{x}) \log \frac{\sum_{c=1}^{m} (\mathcal{N}_h e_c^{(p+1)})(\mathbf{x})}{\sum_{d=1}^{m} (\mathcal{N}_h e_d^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} = \sum_{j=1}^{m} \int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \log \frac{\sum_{c=1}^{m} (\mathcal{N}_h e_c^{(p+1)})(\mathbf{x})}{\sum_{d=1}^{m} (\mathcal{N}_h e_d^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} \int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \log \frac{w_j^{(p)}(\mathbf{x})}{w_j^{(p+1)}(\mathbf{x})} \cdot \frac{(\mathcal{N}_h e_j^{(p+1)})(\mathbf{x})}{(\mathcal{N}_h e_j^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} \int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \log \frac{(\mathcal{N}_h e_j^{(p+1)})(\mathbf{x})}{(\mathcal{N}_h e_j^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} + \sum_{j=1}^{m} \int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \log \frac{w_j^{(p)}(\mathbf{x})}{w_j^{(p+1)}(\mathbf{x})} \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} KL(e_j^{(p+1)}, e_j^{(p)}) \\
&\quad + \sum_{j=1}^{m} \int \left[g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \log \frac{g(\mathbf{x}) w_j^{(p)}(\mathbf{x})}{g(\mathbf{x}) w_j^{(p+1)}(\mathbf{x})} + g(\mathbf{x}) w_j^{(p+1)}(\mathbf{x}) - g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \right] \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} KL(e_j^{(p+1)}, e_j^{(p)}) + \sum_{j=1}^{m} KL(g \cdot w_j^{(p)}, g \cdot w_j^{(p+1)}).
\end{aligned} \tag{2.38}$$

□

Remark 2.1.7. *The discrete version of Lemma 2.1.6 is*

$$l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)}) = \sum_{j=1}^{m} KL(e_j^{(p+1)}, e_j^{(p)}) + \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{1}{n} w_j^{(p)}(\mathbf{x}_i) \log \frac{w_j^{(p)}(\mathbf{x}_i)}{w_j^{(p+1)}(\mathbf{x}_i)} \tag{2.39}$$

Lemma 2.1.6 implies the following corollary:

Corollary 2.1.8. *In the NSMM algorithm, at any step p, we have*

$$l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)}) \geq \sum_{j=1}^{m} KL(e_j^{(p+1)}, e_j^{(p)}). \tag{2.40}$$

Inequality (2.40) may be established directly, using Jensen's Inequality, and we include this proof here because it is interesting in its own right.

Direct Proof of Corollary (2.1.8). If we define $\lambda_j = \int e_j(\mathbf{x})\mathrm{d}\mathbf{x}$ and $f_j(\mathbf{x}) = e_j(\mathbf{x})/\lambda_j$, then

$$\begin{aligned}
&l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)}) \\
&= \int g(\mathbf{x}) \log \frac{\sum_{j=1}^{m} (\mathcal{N}_h e_j^{(p+1)})(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} = \int g(\mathbf{x}) \log \sum_{j=1}^{m} \frac{(\mathcal{N}_h e_j^{(p)})(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(p)})(\mathbf{x})} \cdot \frac{(\mathcal{N}_h e_j^{(p+1)})(\mathbf{x})}{(\mathcal{N}_h e_j^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} \\
&\geq \int g(\mathbf{x}) \sum_{j=1}^{m} \frac{(\mathcal{N}_h e_j^{(p)})(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(p)})(\mathbf{x})} \cdot \log \frac{(\mathcal{N}_h e_j^{(p+1)})(\mathbf{x})}{(\mathcal{N}_h e_j^{(p)})(\mathbf{x})} \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} \int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \log \frac{\lambda_j^{(p+1)} \prod_{k=1}^{r} \mathcal{N}_h f_{j,k}^{(p+1)}(x_k)}{\lambda_j^{(p)} \prod_{k=1}^{r} \mathcal{N}_h f_{j,k}^{(p)}(x_k)} \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} \int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) \left[\log \frac{\lambda_j^{(p+1)}}{\lambda_j^{(p)}} + \sum_{k=1}^{r} \int s_h(u_k, x_k) \log \frac{f_{j,k}^{(p+1)}(u_k)}{f_{j,k}^{(p)}(u_k)} \mathrm{d}u_k \right] \mathrm{d}\mathbf{x} \\
&= \sum_{j=1}^{m} \lambda_j^{(p+1)} \log \frac{\lambda_j^{(p+1)}}{\lambda_j^{(p)}} + \sum_{j=1}^{m} \sum_{k=1}^{r} \int \left(\int g(\mathbf{x}) w_j^{(p)}(\mathbf{x}) s_h(u_k, x_k) \mathrm{d}\mathbf{x} \right) \log \frac{f_{j,k}^{(p+1)}(u_k)}{f_{j,k}^{(p)}(u_k)} \mathrm{d}u_k \\
&= \sum_{j=1}^{m} \lambda_j^{(p+1)} \log \frac{\lambda_j^{(p+1)}}{\lambda_j^{(p)}} + \sum_{j=1}^{m} \sum_{k=1}^{r} \int \lambda_j^{(p+1)} f_{j,k}^{(p+1)}(u_k) \log \frac{f_{j,k}^{(p+1)}(u_k)}{f_{j,k}^{(p)}(u_k)} \mathrm{d}u_k \\
&= \sum_{j=1}^{m} \lambda_j^{(p+1)} \log \frac{\lambda_j^{(p+1)}}{\lambda_j^{(p)}} + \sum_{j=1}^{m} \int \lambda_j^{(p+1)} \left(\prod_{k=1}^{r} f_{j,k}^{(p+1)}(u_k) \right) \log \frac{\prod_{k=1}^{r} f_{j,k}^{(p+1)}(u_k)}{\prod_{k=1}^{r} f_{j,k}^{(p)}(u_k)} \mathrm{d}\mathbf{u}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=1}^{m} \int \lambda_j^{(p+1)} \left(\prod_{k=1}^{r} f_{j,k}^{(p+1)}(u_k) \right) \log \frac{\lambda_j^{(p+1)} \prod_{k=1}^{r} f_{j,k}^{(p+1)}(u_k)}{\lambda_j^{(p)} \prod_{k=1}^{r} f_{j,k}^{(p)}(u_k)} \mathrm{d}\mathbf{u} \\
&= \sum_{j=1}^{m} \int e_j^{(p+1)}(\mathbf{u}) \log \frac{e_j^{(p+1)}(\mathbf{u})}{e_j^{(p)}(\mathbf{u})} \mathrm{d}\mathbf{u} \\
&= \sum_{j=1}^{m} \int \left(e_j^{(p+1)}(\mathbf{u}) \log \frac{e_j^{(p+1)}(\mathbf{u})}{e_j^{(p)}(\mathbf{u})} + e_j^{(p)}(\mathbf{u}) - e_j^{(p+1)}(\mathbf{u}) \right) \mathrm{d}\mathbf{u} \\
&= \sum_{j=1}^{m} KL(e_j^{(p+1)}, e_j^{(p)}).
\end{aligned}
$$

□

Corollary (2.1.8) implies the following two corollaries.

Corollary 2.1.9. *Any minimizer* $\mathbf{e}$ *of* $l(\mathbf{e})$ *or* $l_{discrete}(\mathbf{e})$ *is a fixed point of the corresponding NSMM algorithm.*

Proof. Since the right side of (2.40) is strictly positive when $e_j^{(p+1)} \neq e_j^{(p)}$ for any j, a necessary condition for $\mathbf{e}^{(p)}$ to minimize $l(\mathbf{e})$ is that $\mathbf{e}^{(p+1)} = \mathbf{e}^{(p)}$, i.e., that $\mathbf{e}^{(p)}$ is a fixed point of the algorithm.

□

Corollary 2.1.9 guarantees that we only need to search among fixed point(s) of the NSMM algorithm for any solution. This gives the theoretical basis of using the NSMM algorithm for this estimation problem.

Corollary 2.1.10. *In the NSMM algorithm, at any step* p*, we have*

$$
l(\mathbf{e}^{(p)}) - l(\mathbf{e}^{(p+1)}) \geq \sum_{j=1}^{m} \frac{1}{4} ||e_j^{(p+1)}, e_j^{(p)}||_1^2, \tag{2.41}
$$

where $||\cdot||$ *denotes the* L^1 *norm.*

Proof. The result follows from Inequality (3.21) in Eggermont & LaRiccia (2001), which states that

$$KL(g_1, g_2) \geq \frac{1}{4}||g_1, g_2||_1^2 \tag{2.42}$$

for functions g_1 and g_2. □

Corollary 2.1.10 ensures that the L^1 distance between estimates of adjacent steps from an NSMM sequence will tend to zero. This is helpful in proving a convergence property of the NSMM algorithm, as will be shown in section (2.3).

2.2 Existence of Solution

In this section, we verify the existence of at least one solution to the main optimization problem of Section 2.1.3. This result is novel in the literature on non-parametric estimation of finite multivariate mixture models with the conditional independence assumption.

Lemma 2.2.1. *Given* $\mathbf{e}$ *satisfying assumption (ii), we have* $l(\mathbf{e}) \geq 1$. *In the discrete case, we have* $l_{discrete}(\mathbf{e}) \geq -\log M_2$.

Proof. For each j, $1 \leq j \leq m$, and $\mathbf{x} \in \Omega$, Jensen's Inequality gives

$$\begin{aligned}
\left(\mathcal{N}_h e_j\right)(\mathbf{x}) &= \exp\left\{\int \tilde{s}_h(\mathbf{u}, \mathbf{x}) \log e_j(\mathbf{u}) \mathrm{d}\mathbf{u}\right\} \\
&\leq \int \tilde{s}_h(\mathbf{u}, \mathbf{x}) \exp\left[\log e_j(\mathbf{u})\right] \mathrm{d}\mathbf{u} \\
&= \int \tilde{s}_h(\mathbf{u}, \mathbf{x}) e_j(\mathbf{u}) \mathrm{d}\mathbf{u}.
\end{aligned} \tag{2.43}$$

Integrate both sides with respect to $\mathbf{x}$, then use Fubini's Theorem to obtain

$$\int \left(\mathcal{N}_h e_j\right)(\mathbf{x})\mathrm{d}\mathbf{x} \leq \int e_j(\mathbf{u})\mathrm{d}\mathbf{u}. \tag{2.44}$$

Summing over j, we get

$$\int \sum_{j=1}^{m} (\mathcal{N}_h e_j) \leq \int \sum_{j=1}^{m} e_j. \tag{2.45}$$

Therefore, all three terms on the right hand side of

$$l(\mathbf{e}) = KL\left(g, \sum_{j=1}^{m} (\mathcal{N}_h e_j)\right) + \int g + \left[\int \sum_{j=1}^{m} e_j - \int \sum_{j=1}^{m} (\mathcal{N}_h e_j)\right] \tag{2.46}$$

are nonnegative and the middle term is 1, which implies that $l(\cdot)$ is always bounded below by 1.

For discrete case, Jensen's Inequality gives

$$\begin{aligned} l_{\text{discrete}}(\mathbf{e}) &= -\frac{1}{n}\sum_{i=1}^{n} \log \sum_{j=1}^{m} (\mathcal{N}_h e_j)(\mathbf{x}_i) + \int \sum_{j=1}^{m} e_j(\mathbf{x}_i) \\ &\geq -\frac{1}{n}\sum_{i=1}^{n} \log \sum_{j=1}^{m} (\mathcal{N}_h e_j)(\mathbf{x}_i) \\ &\geq -\frac{1}{n}\sum_{i=1}^{n} \log \sum_{j=1}^{m} (\mathcal{S}_h^* e_j)(\mathbf{x}_i) \geq -\frac{1}{n}\sum_{i=1}^{n} \log M_2 = -\log M_2 \end{aligned} \tag{2.47}$$

□

Together, Lemma 2.1.8 and Lemma 2.2.1 imply the following corollary.

Corollary 2.2.2. *In the NSMM algorithm, $l(\mathbf{e}^{(p)})$ will tend to a finite limit as p goes to infinity. In the discrete case, this still holds.*

We now establish some technical results that lead to the main conclusion of this

section, namely, the existence of a minimizer of both $l(\mathbf{e})$ and $l_{\text{discrete}}(\mathbf{e})$

Lemma 2.2.3. *Assume conditions (i) through (vi). For each j, $1 \leq j \leq m$, any NSMM sequence $\{e_j^{(p)}\}_{1 \leq p < \infty}$ is uniformly bounded and equicontinuous on Ω. This is also true for the discrete case.*

Proof. In the continuous case, for $p \geq 1$ and $\mathbf{u} \in \Omega$,

$$e_j^{(p)}(\mathbf{u}) = \frac{\prod_{k=1}^{r} \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \cdot s_h(u_k, x_k) \, \mathrm{d}\mathbf{x}}{\left[\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \, \mathrm{d}\mathbf{x}\right]^{r-1}} \leq M_2^r \cdot \left[\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \, \mathrm{d}\mathbf{x}\right] \leq M_2^r.$$

Thus $\{e_j^{(p)}\}_{1 \leq p < \infty}$ is uniformly bounded. Also, for any $\mathbf{u}$ in the interior of Ω,

$$\begin{aligned}
\left|\frac{\partial}{\partial u_l} e_j^{(p)}(\mathbf{u})\right| &= \left| \frac{\left[\prod_{k \neq l}^{r} \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) s_h(u_k, x_k) \, \mathrm{d}\mathbf{x}\right] \cdot \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \frac{\partial}{\partial u_l} s_h(u_l, x_l) \, \mathrm{d}\mathbf{x}}{\left[\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \, \mathrm{d}\mathbf{x}\right]^{r-1}} \right| \\
&\leq B \cdot M_2^{r-1} \cdot \left[\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \, \mathrm{d}\mathbf{x}\right] \\
&\leq B \cdot M_2^{r-1}. \qquad (2.48)
\end{aligned}$$

By the Dominated Convergence Theorem, the above differentiation under the integral is allowed because the term $|g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \frac{\partial}{\partial u_l} s_h(u_l, x_l)|$ is uniformly bounded by the integrable function $B \cdot g(\mathbf{x})$.

Now by the Mean Value Theorem for functions of several variables, for any $\mathbf{u}, \mathbf{v} \in \Omega$, there is some $d \in (0, 1)$ such that

$$e_j^{(p)}(\mathbf{u}) - e_j^{(p)}(\mathbf{v}) = \nabla e_j^{(p)}[(1-d)\mathbf{v} + d\mathbf{u}] \cdot (\mathbf{u} - \mathbf{v}). \qquad (2.49)$$

So

$$\left|e_j^{(p)}(\mathbf{u}) - e_j^{(p)}(\mathbf{v})\right| \leq [B \cdot M_2^{r-1}] \cdot ||\mathbf{u} - \mathbf{v}||_1, \tag{2.50}$$

which shows that $\{e_j^{(p)}\}_{1 \leq p < \infty}$ is equicontinuous on Ω in the L^1 norm.

This proof can be readily adapted to the discrete case, after replacing the integrals by summations. □

The above proof implies more generally we have:

Lemma 2.2.4. *For* $\mathbf{e}$ *satisfying assumptions (i) through (vi), in either the discrete or the continuous case, for* $1 \leq j \leq m$ *and* $\mathbf{u}$, $\mathbf{v} \in \Omega$, *we have:*

$$(G(\mathbf{e}))_j(\mathbf{u}) \leq M_2^r, \tag{2.51}$$

$$|(G(\mathbf{e}))_j(\mathbf{u}) - (G(\mathbf{e}))_j(\mathbf{v})| \leq [B \cdot M_2^{r-1}] \cdot ||\mathbf{u} - \mathbf{v}||_1. \tag{2.52}$$

The following lemma establishes a sort of lower semi-continuity of the functional $l(\cdot)$, which will be needed in proving existence of at least one solution to the main optimization problem.

Lemma 2.2.5. *Let* $\gamma_j^{(p)} \in L^1(R^r)$ *be nonnegative and with support in* Ω *for each* p *and* j, *where* $0 \leq p \leq \infty$ *and* $1 \leq j \leq m$. *Assume each* $\gamma_j^{(p)}$ *uniformly converges to* $\gamma_j^{(\infty)}$ *in* $L^1(R^r)$ *and that all* $\gamma_j^{(p)}$ *are bounded from above by a constant* $Q > 1$. *Let* $\boldsymbol{\gamma}^{(p)}$ *and* $\boldsymbol{\gamma}^{(\infty)}$ *represent* $(\gamma_1^{(p)}, \cdots, \gamma_m^{(p)})$ *and* $(\gamma_1^{(\infty)}, \cdots, \gamma_m^{(\infty)})$, *respectively. Then we have*

$$l(\boldsymbol{\gamma}^{(\infty)}) \leq \liminf_{p \to \infty} l(\boldsymbol{\gamma}^{(p)}). \tag{2.53}$$

This is also true for the discrete case.

Proof. We first consider the continuous case. In the following, Fatou's Lemma will be applied twice to get the desired result. First, we show by Jensen's Inequality

that all $\mathcal{N}_h\gamma_j^{(p)}$ are bounded from above by Q:

$$\begin{aligned}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x}) &= \exp\left\{\int \tilde{s}_h(\mathbf{u},\mathbf{x})\log\gamma_j^{(p)}(\mathbf{u})\mathrm{d}\mathbf{u}\right\} \\ &\leq \exp\left\{\int \tilde{s}_h(\mathbf{u},\mathbf{x})\log Q\mathrm{d}\mathbf{u}\right\} = Q. \end{aligned} \tag{2.54}$$

Now, for any fixed value of $\mathbf{x}$, the nonnegative measurable function $\tilde{s}_h(\cdot,\mathbf{x})[Q-\log\gamma_j^{(p)}(\cdot)]$ converges to $\tilde{s}_h(\cdot,\mathbf{x})[Q-\log\gamma_j^{(\infty)}(\cdot)]$ pointwise in $L^1(R^r)$. These functions are allowed to attain the value $+\infty$. By Fatou's Lemma, we have

$$\liminf_{p\to\infty}\int \tilde{s}_h(\mathbf{u},\mathbf{x})[Q-\log\gamma_j^{(p)}(\mathbf{u})]\mathrm{d}\mathbf{u} \geq \int \tilde{s}_h(\mathbf{u},\mathbf{x})[Q-\log\gamma_j^{(\infty)}(\mathbf{u})]\mathrm{d}\mathbf{u}. \tag{2.55}$$

Exponentiating, this implies

$$\limsup_{p\to\infty}\exp\left\{\int \tilde{s}_h(\mathbf{u},\mathbf{x})\log\gamma_j^{(p)}(\mathbf{u})\mathrm{d}\mathbf{u}\right\} \leq \exp\left\{\int \tilde{s}_h(\mathbf{u},\mathbf{x})\log\gamma_j^{(\infty)}(\mathbf{u})\mathrm{d}\mathbf{u}\right\}. \tag{2.56}$$

That is,

$$\limsup_{p\to\infty}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x}) \leq \left(\mathcal{N}_h\gamma_j^{(\infty)}\right)(\mathbf{x}), \tag{2.57}$$

which implies that

$$\liminf_{p\to\infty} g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})} \geq g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(\infty)}\right)(\mathbf{x})}. \tag{2.58}$$

Since $a\log\frac{a}{b}+b-a$ is nonnegative for all $a,b\geq 0$, we have

$$g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})} \geq g(\mathbf{x}) - \sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x}) \geq -m\cdot Q. \tag{2.59}$$

Thus, we can rewrite (2.58) as

$$\liminf_{p\to\infty}\left[g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})}+m\cdot Q\right]\geq g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(\infty)}\right)(\mathbf{x})}+m\cdot Q, \tag{2.60}$$

so that both sides are nonnegative.

Now apply Fatou's Lemma again to obtain

$$\begin{aligned}
&\liminf_{p\to\infty}\int\left[g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})}+m\cdot Q\right]\mathrm{d}\mathbf{x}\\
&\quad\geq\int\left[\liminf_{p\to\infty}g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})}+m\cdot Q\right]\mathrm{d}\mathbf{x}\\
&\quad\geq\int\left[g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(\infty)}\right)(\mathbf{x})}+m\cdot Q\right]\mathrm{d}\mathbf{x}.
\end{aligned} \tag{2.61}$$

We conclude that

$$\liminf_{p\to\infty}\int g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})}\mathrm{d}\mathbf{x}\geq\int g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(\infty)}\right)(\mathbf{x})}\mathrm{d}\mathbf{x}. \tag{2.62}$$

The uniform convergence of $\gamma_j^{(p)}$ to $\gamma_j^{(\infty)}$ for each j together with (2.62), implies

$$\liminf_{p\to\infty}\left[\int g(\mathbf{x})\log\frac{g(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h\gamma_j^{(p)}\right)(\mathbf{x})}\mathrm{d}\mathbf{x}+\int\sum_{j=1}^{m}\gamma_j^{(p)}(\mathbf{x})\mathrm{d}\mathbf{x}\right]$$

$$\geq \int g(\mathbf{x}) \log \frac{g(\mathbf{x})}{\sum_{j=1}^{m} \left(\mathcal{N}_h \gamma_j^{(\infty)}\right)(\mathbf{x})} \mathrm{d}\mathbf{x} + \int \sum_{j=1}^{m} \gamma_j^{(\infty)}(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2.63}$$

That is,

$$\liminf_{p \to \infty} l(\gamma^{(p)}) \geq l(\gamma^{(\infty)}), \tag{2.64}$$

which establishes the desired lower semi-continuity.

The proof can be adpated to the discrete case, after the integrals are replaced by summations. □

Theorem 2.2.6. *Under assumptions (i) through (vi), there exists at least one solution to the main optimization problem (2.15). This is also true in the discrete case.*

Proof. By Lemma 2.2.1, $\tau := \inf\{l(\mathbf{e}) | \mathbf{e}$ satisfies assumptions (ii) and (vi)$\}$ is a finite constant. So there exists a sequence $\{\boldsymbol{\psi}^{(p)}\}_{0 \leq p \leq \infty}$ satisfying assumptions (ii) and (vi) such that

$$\lim_{p \to \infty} l(\boldsymbol{\psi}^{(p)}) = \tau. \tag{2.65}$$

By Lemma 2.2.4, for each j, $1 \leq j \leq m$, the sequence $\{(G(\boldsymbol{\psi}^{(p)}))_j\}_{0 \leq p \leq \infty}$ is bounded and equicontinuous.

By the Arzelà-Ascoli theorem, we know that $\{(G(\boldsymbol{\psi}^{(p)}))_j\}_{0 \leq p \leq \infty}$ has a uniformly convergent subsequence. Applying this theorem m times to $\{(G(\boldsymbol{\psi}^{(p)}))\}_{0 \leq p \leq \infty}$ we can extract a subsequence that converges uniformly in every component. This sequence also satisfies (ii) and (vi).

That is, there exists a sequence $\{(G(\boldsymbol{\psi}^{(p_k)}))\}_{0 \leq k \leq \infty}$, such that, for each j, $1 \leq j \leq m$, $\{(G(\boldsymbol{\psi}^{(p_k)}))_j\}_{0 \leq k \leq \infty}$ converges uniformly to a limit function in $L^1(R^r)$. Denote this limit function by $\tilde{\psi}_j$. As usual, let $\tilde{\boldsymbol{\psi}}$ denote the m-tuples $(\tilde{\psi}_1, \cdots, \tilde{\psi}_m)$. If all components of $\tilde{\boldsymbol{\psi}}$ are nonzero, then $\tilde{\boldsymbol{\psi}}$ satisfies (iii). If not, we can split up

some nonzero components of $\tilde{\boldsymbol{\psi}}$ so that all components become nonzero, which does not change the value of $l(\tilde{\boldsymbol{\psi}})$. In a word, we can assume that $\tilde{\boldsymbol{\psi}}$ satisfies (vi).

Now, by Lemma 2.2.5 and the fact that G does not increase the value of l (see the proof of Lemma 2.1.8), we have

$$\tau \leq l(\tilde{\boldsymbol{\psi}}) \leq \lim_{k\to\infty} l(G(\boldsymbol{\psi}^{(p_k)})) \leq \lim_{k\to\infty} l(\boldsymbol{\psi}^{(p_k)}) = \lim_{p\to\infty} l(\boldsymbol{\psi}^{(p)}) = \tau, \tag{2.66}$$

so that $l(\tilde{\boldsymbol{\psi}}) = \tau$. Apply the operator G to $\tilde{\boldsymbol{\psi}}$. By Lemma 2.1.8 and the fact that $l(\tilde{\boldsymbol{\psi}})$ has already attained the infimum value in this setting, we have

$$0 \geq l(\tilde{\boldsymbol{\psi}}) - l(G(\tilde{\boldsymbol{\psi}})) \geq \sum_{j=1}^{m} KL((G(\tilde{\boldsymbol{\psi}}))_j, \tilde{\psi}_j) \geq 0. \tag{2.67}$$

So for each j, $1 \leq j \leq m$, $G(\tilde{\boldsymbol{\psi}})_j = \tilde{\psi}_j$ in $L^1(R^r)$. Thus in particular, by (2.34), $\tilde{\boldsymbol{\psi}}$ also satisfies assumption (ii). We have proved the existence of a solution, $\tilde{\boldsymbol{\psi}}$, to the main optimization problem (2.15).

The proof can readily be adapted to the discrete case, as mentioned earlier in this section. □

To conclude this section, we discuss the rationale behind assumption (vi) and related issues such as why the $\mathcal{N}_h$ operator is well-defined as we applied it.

Lemma 2.2.7. *In an NSMM sequence* $\{\mathbf{e}^{(p)}\}_{0\leq p\leq\infty}$, $e_j^{(p)}$, $1 \leq j \leq m$, *are all strictly positive. Moreover, if we let* $\lambda_j^{(p)} = \int e_j^{(p)}$, *and let* $f_j^{(p)}(\mathbf{u}) = e_j^{(p)}(\mathbf{u})/\lambda_j^{(p)}$, *then*

$$(M_1(h))^r \leq f_j^{(p)}(\mathbf{u}) \leq M_2^r \qquad \textit{for all } \mathbf{u} \in \Omega, p > 0 \textit{ and } 1 \leq j \leq m. \tag{2.68}$$

Proof. First, by assumption (vi), each $e_j^{(0)}$ is strictly positive on Ω. So given any

$\mathbf{x} \in \Omega$,

$$\left(\mathcal{N}_h e_j^{(0)}\right)(\mathbf{x}) = \exp\left[\left(S_h^* \log e_j^{(0)}\right)(\mathbf{x})\right] > 0. \tag{2.69}$$

Thus,

$$w_j^{(0)}(\mathbf{x}) = \frac{\left(\mathcal{N}_h e_j^{(0)}\right)(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h e_j^{(0)}\right)(\mathbf{x})} > 0, \tag{2.70}$$

which implies

$$\int g(\mathbf{x}) w_j^{(0)}(\mathbf{x})\ \mathrm{d}\mathbf{x} > 0. \tag{2.71}$$

Now, we use induction. Assume

$$\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x})\ \mathrm{d}\mathbf{x} > 0. \tag{2.72}$$

We have

$$f_j^{(p)}(u) = \frac{\prod_{k=1}^{r} \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \cdot s_h(u_k, x_k)\ \mathrm{d}\mathbf{x}}{\left[\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x})\ \mathrm{d}\mathbf{x}\right]^r} \leq M_2^r. \tag{2.73}$$

Similarly,

$$f_j^{(p)}(u) \quad = \frac{\prod_{k=1}^{r} \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x}) \cdot s_h(u_k, x_k)\ \mathrm{d}\mathbf{x}}{\left[\int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x})\ \mathrm{d}\mathbf{x}\right]^r} \geq (M_1(h))^r. \tag{2.74}$$

Therefore,

$$\begin{aligned}\left(\mathcal{N}_h e_j^{(p)}\right)(\mathbf{x}) &= \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x})\ \mathrm{d}\mathbf{x} \cdot \exp\left[\left(S_h^* \log f_j^{(p)}\right)(\mathbf{x})\right] \\ &\geq \int g(\mathbf{x}) w_j^{(p-1)}(\mathbf{x})\ \mathrm{d}\mathbf{x} \cdot (M_1(h))^r \\ &> 0. \end{aligned} \tag{2.75}$$

We conclude that

$$w_j^{(p)}(\mathbf{x}) = \frac{\left(\mathcal{N}_h e_j^{(p)}\right)(\mathbf{x})}{\sum_{j=1}^{m}\left(\mathcal{N}_h e_j^{(p)}\right)(\mathbf{x})} > 0, \tag{2.76}$$

which gives

$$\int g(\mathbf{x}) w_j^{(p)}(\mathbf{x})\, \mathrm{d}\mathbf{x} > 0. \tag{2.77}$$

The next step of the induction follows in the same way, and the result is established.

□

Lemma (2.2.7) shows why in assumption (vi) we require the marginal densities of each mixture component to be bounded below by $(M_1(h))^r$ and guarantees that dividing by zero never occurs in any NSMM sequence.

2.2.1 Properties of Solution

Here we present several properties of any solution of the estimation problem of Section 2.1.3. Such a solution will be denoted by $\mathbf{e^S}$. Thus far these properties have not led to any additional results, but we include them here because they could be useful in establishing theoretical properties of the NSMM algorithm. First, we prove a symmetry result:

Lemma 2.2.8. *Let v be a permutation of the set $\{1, 2, \cdots, m\}$. Then at any step p of the NSMM algorithm, we have*

$$l(\mathbf{e}^{(p)}) - l(\mathbf{e}^S) = \sum_{j=1}^{m} KL(\mathbf{e}^S, \mathbf{e}_{v(j)}^{(p)}) - \sum_{j=1}^{m} KL(g \cdot w_j^S, g \cdot w_{v(j)}^{(p)}). \tag{2.78}$$

Proof.

$$l(\mathbf{e}^{(p)}) - l(\mathbf{e}^S) \tag{2.79}$$

$$
\begin{aligned}
&= \int g(x) \log \frac{\sum_{c=1}^{m} (\mathcal{N}_h e_c^S)(x)}{\sum_{b=1}^{m} (\mathcal{N}_h e_b^{(p)})(x)} \mathrm{d}x \\
&= \int g(x) \frac{\sum_{j=1}^{m} (\mathcal{N}_h e_j^S)(x)}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^S)(x)} \log \frac{\sum_{c=1}^{m} (\mathcal{N}_h e_c^S)(x)}{\sum_{b=1}^{m} (\mathcal{N}_h e_b^{(p)})(x)} \mathrm{d}x \\
&= \sum_{j=1}^{m} \int g(x) \frac{(\mathcal{N}_h e_j^S)(x)}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^S)(x)} \log \frac{\sum_{c=1}^{m} (\mathcal{N}_h e_c^S)(x)}{\sum_{b=1}^{m} (\mathcal{N}_h e_b^{(p)})(x)} \mathrm{d}x \\
&= \sum_{j=1}^{m} \int g(x) \frac{(\mathcal{N}_h e_j^S)(x)}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^S)(x)} \log \left[\frac{(\mathcal{N}_h e_j^S)(x)}{(\mathcal{N}_h e_{v(j)}^{(p)})(x)} \cdot \frac{w_{v(j)}^{(p)}(x)}{w_j^S(x)} \right] \mathrm{d}x \\
&= \sum_{j=1}^{m} \int g(x) w_j^S(x) \log \frac{(\mathcal{N}_h e_j^S)(x)}{(\mathcal{N}_h e_{v(j)}^{(p)})(x)} \mathrm{d}x - \sum_{j=1}^{m} \int g(x) w_j^S(x) \log \frac{w_j^S(x)}{w_{v(j)}^{(p)}(x)} \mathrm{d}x. \quad (2.80)
\end{aligned}
$$

By Corollary 2.1.9, $\mathbf{e}^S$ is a fixed point of the NMMS algorithm. So for any j, $1 \le j \le m$, we have

$$
\int g(x) w_j^S(x) \mathrm{d}x = \int (G(e^S))_j(x) \mathrm{d}x = \int e_j^S(x) \mathrm{d}x = \lambda_j^S. \quad (2.81)
$$

So the first term on the right side of (2.79) simplifies as follows:

$$
\begin{aligned}
&\sum_{j=1}^{m} \int g(x) w_j^S(x) \log \frac{(\mathcal{N}_h e_j^S)(x)}{(\mathcal{N}_h e_{v(j)}^{(p)})(x)} \mathrm{d}x \\
&= \sum_{j=1}^{m} \int g(x) w_j^S(x) \log \frac{\lambda_j^S \prod_{k=1}^{r} (\mathcal{N}_h f_{j,k}^S)(x_k)}{\lambda_{v(j)}^{(p)} \prod_{k=1}^{r} (\mathcal{N}_h f_{v(j),k}^{(p)})(x_k)} \mathrm{d}x \\
&= \sum_{j=1}^{m} \int g(x) w_j^S(x) \left[\log \frac{\lambda_j^S}{\lambda_{v(j)}^{(p)}} + \sum_{k=1}^{r} \log \frac{(\mathcal{N}_h f_{j,k}^S)(x_k)}{(\mathcal{N}_h f_{v(j),k}^{(p)})(x_k)} \right] \mathrm{d}x
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{j=1}^{m} \int g(x) w_j^S(x) \left[\log \frac{\lambda_j^S}{\lambda_{v(j)}^{(p)}} + \sum_{k=1}^{r} \int s_h(u_k, x_k) \log \frac{f_{j,k}^S(u_k)}{f_{v(j),k}^{(p)}(u_k)} \mathrm{d}u_k \right] \mathrm{d}x \\
&= \sum_{j=1}^{m} \lambda_j^S \log \frac{\lambda_j^S}{\lambda_{v(j)}^{(p)}} + \sum_{j=1}^{m} \sum_{k=1}^{r} \int \left(\int g(x) w_j^S(x) s_h(u_k, x_k) \mathrm{d}x \right) \log \frac{f_{j,k}^S(u_k)}{f_{v(j),k}^{(p)}(u_k)} \mathrm{d}u_k \\
&= \sum_{j=1}^{m} \lambda_j^S \log \frac{\lambda_j^S}{\lambda_{v(j)}^{(p)}} + \sum_{j=1}^{m} \sum_{k=1}^{r} \int \lambda_j^S f_{j,k}^S(u_k) \log \frac{f_{j,k}^S(u_k)}{f_{v(j),k}^{(p)}(u_k)} \mathrm{d}u_k \\
&= \sum_{j=1}^{m} \lambda_j^S \log \frac{\lambda_j^S}{\lambda_{v(j)}^{(p)}} + \sum_{j=1}^{m} \int \lambda_j^S \left(\prod_{k=1}^{r} f_{j,k}^S(u_k) \right) \log \frac{\prod_{k=1}^{r} f_{j,k}^S(u_k)}{\prod_{k=1}^{r} f_{v(j),k}^{(p)}(u_k)} \mathrm{d}u \\
&= \sum_{j=1}^{m} \int \lambda_j^S \left(\prod_{k=1}^{r} f_{j,k}^S(u_k) \right) \log \frac{\lambda_j^S \prod_{k=1}^{r} f_{j,k}^S(u_k)}{\lambda_{v(j)}^{(p)} \prod_{k=1}^{r} f_{v(j),k}^{(p)}(u_k)} \mathrm{d}u \\
&= \sum_{j=1}^{m} \int e_j^S(u) \log \frac{e_j^S(u)}{e_{v(j)}^{(p)}(u)} \mathrm{d}u \\
&= \sum_{j=1}^{m} \int \left[e_j^S(u) \log \frac{e_j^S(u)}{e_{v(j)}^{(p)}(u)} + e_{v(j)}^{(p)}(u) - e_j^S(u) \right] \mathrm{d}u \\
&= \sum_{j=1}^{m} KL(e_j^S, e_{v(j)}^{(p)}).
\end{aligned} \tag{2.82}
$$

The second term on the right side of (2.79) is

$$
\begin{aligned}
&- \sum_{j=1}^{m} \int g(x) w_j^S(x) \log \frac{w_j^S(x)}{w_{v(j)}^{(p)}(x)} \mathrm{d}x \\
&= - \sum_{j=1}^{m} \int g(x) w_j^S(x) \log \frac{g(x) w_j^S(x)}{g(x) w_{v(j)}^{(p)}(x)} \mathrm{d}x \\
&= - \sum_{j=1}^{m} \int \left[g(x) w_j^S(x) \log \frac{g(x) w_j^S(x)}{g(x) w_{v(j)}^{(p)}(x)} + g(x) w_{v(j)}^{(p)}(x) - g(x) w_j^S(x) \right] \mathrm{d}x \\
&= - \sum_{j=1}^{m} KL(g \cdot w_j^S, g \cdot w_{v(j)}^{(p)}).
\end{aligned} \tag{2.83}
$$

This concludes the proof. □

Lemma 2.2.9 is inspired by the attempts to prove a crucial inequality in an unpublished manuscript by Hunter, Levine, Chauveau and Bordes, which would be an analogous result to Inequality (1.16) in Eggermont & LaRiccia (1995). It gives lower and upper bounds for $l(\mathbf{e}) - l(\mathbf{e}^S)$, where $\mathbf{e} = (e_1, e_2, ..., e_m)$, $e_j \in L^1(R^r)$ for $1 \leq j \leq m$ and $\int \sum_{j=1}^{m} e_j = 1$.

Let

$$r_S(x) = \frac{g(x)}{\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x)} \tag{2.84}$$

and

$$r(x) = \frac{g(x)}{\sum_{j=1}^{m} \mathcal{N}_h e_j(x)}. \tag{2.85}$$

Lemma 2.2.9.

$$\frac{1}{2}\left(\int \sqrt{r_S(x)}\Big|\sum_{j=1}^{m}\mathcal{N}_h e_j(x) - \sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)\Big|dx\right)^2 - \int g(x)\frac{\sum_{j=1}^{m}\mathcal{N}_h e_j(x)}{\sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)}dx + 1$$

$$\leq l(\mathbf{e}) - l(\mathbf{e}^S) \leq$$

$$-\frac{1}{2}\left(\int \sqrt{r(x)}\Big|\sum_{j=1}^{m}\mathcal{N}_h e_j(x) - \sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)\Big|dx\right)^2 + \int g(x)\frac{\sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)}{\sum_{j=1}^{m}\mathcal{N}_h e_j(x)}dx - 1. \tag{2.86}$$

Proof.

$$\left(\int \sqrt{r_S(x)}\Big|\sum_{j=1}^{m}\mathcal{N}_h e_j(x) - \sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)\Big|\mathrm{d}x\right)^2$$

$$= \left(\int \sqrt{r_S(x)}\sqrt{\left[\sum_{j=1}^{m}\mathcal{N}_h e_j(x) - \sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)\right]^2}\,\mathrm{d}x\right)^2$$

$$
\begin{aligned}
&\leq \left(\int \sqrt{r_S(x)} \sqrt{\frac{4}{3}\sum_{j=1}^{m} \mathcal{N}_h e_j(x) + \frac{2}{3}\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x)} \right. \\
&\qquad \left. \cdot \sqrt{\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x) \log \frac{\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x)}{\sum_{j=1}^{m} \mathcal{N}_h e_j(x)} + \sum_{j=1}^{m} \mathcal{N}_h e_j(x) - \sum_{j=1}^{m} \mathcal{N}_h e_j^S(x) \mathrm{d}x} \right)^2 \\
&\leq \int \left(\frac{4}{3}\sum_{j=1}^{m} \mathcal{N}_h e_j(x) + \frac{2}{3}\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x) \right) \mathrm{d}x \\
&\qquad \cdot \int \left[g(x) \log \frac{\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x)}{\sum_{j=1}^{m} \mathcal{N}_h e_j(x)} + r_S(x) \sum_{j=1}^{m} \mathcal{N}_h e_j(x) - g(x) \right] \mathrm{d}x \\
&\leq \int \left(\frac{4}{3}\sum_{j=1}^{m} e_j(x) + \frac{2}{3}\sum_{j=1}^{m} e_j^S(x) \right) \mathrm{d}x \\
&\qquad \cdot \int \left[g(x) \log \frac{\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x)}{\sum_{j=1}^{m} \mathcal{N}_h e_j(x)} + r_S(x) \sum_{j=1}^{m} \mathcal{N}_h e_j(x) - g(x) \right] \mathrm{d}x \\
&= 2 \int \left[g(x) \log \frac{\sum_{j=1}^{m} \mathcal{N}_h e_j^S(x)}{\sum_{j=1}^{m} \mathcal{N}_h e_j(x)} + r_S(x) \sum_{j=1}^{m} \mathcal{N}_h e_j(x) - g(x) \right] \mathrm{d}x \\
&= 2\left(l(\mathbf{e}) - l(\mathbf{e}^S) \right) + 2 \int r_S(x) \sum_{j=1}^{m} \mathcal{N}_h e_j(x) \mathrm{d}x - 2. \qquad (2.87)
\end{aligned}
$$

That is,

$$
\begin{aligned}
&\left(\int \sqrt{r_S(x)} \Big| \sum_{j=1}^{m} \mathcal{N}_h e_j(x) - \sum_{j=1}^{m} \mathcal{N}_h e_j^S(x) \Big| \mathrm{d}x \right)^2 \\
&\leq 2\left(l(\mathbf{e}) - l(\mathbf{e}^S) \right) + 2 \int r_S(x) \sum_{j=1}^{m} \mathcal{N}_h e_j(x) \mathrm{d}x - 2. \qquad (2.88)
\end{aligned}
$$

Similarly,

$$\left(\int \sqrt{r(x)}\Big|\sum_{j=1}^{m}\mathcal{N}_h e_j(x)-\sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)\Big|\mathrm{d}x\right)^2$$
$$\leq 2\left(l(\mathbf{e}^S)-l(\mathbf{e})\right)+2\int r(x)\sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)\mathrm{d}x-2. \tag{2.89}$$

Combining (2.88) and (2.89) concludes the proof.

□

The upper bound in (2.86) implies the following:

Corollary 2.2.10.

$$\int g(x)\frac{\sum_{j=1}^{m}\mathcal{N}_h e_j^S(x)}{\sum_{j=1}^{m}\mathcal{N}_h e_j(x)}dx\geq 1 \tag{2.90}$$

2.3 Convergence of NSMM in the Discrete Case

The discrete version of NSMM, described in (2.35) and (2.36), is the version that is actually employed in practical applications. So it is of interest to know its convergence properties. In the following, a few primitive results are presented. Hopefully they will be useful for further investigation.

According to (2.35) and (2.36), the estimate $\mathbf{e}^{(p+1)}$ is parameterized by the values of the $w_j^{(p)}(\mathbf{x}_i)$, $1\leq i\leq n$ and $1\leq j\leq m$, which are finite in number. Denote the collection of these finitely many parameters by $\Theta^{(p)}$. So we can write $\mathbf{e}^{(p+1)}=\mathbf{e}(\Theta^{(p)})$.

Using the same idea that was used in the proofs of the results from Section 2.2, we apply the Arzelà-Ascoli theorem to a sequence $\{e_j^{(p)}\}_{1\leq p<\infty}$ obtained from

NSMM to show that $\{e_j^{(p)}\}_{1\leq p<\infty}$ has a uniformly convergent subsequence. Applying the theorem m times to $\{\mathbf{e}^{(p)}\}_{1\leq p<\infty}$, we can extract a subsequence that converge uniformly in every component. Thus we have

Theorem 2.3.1. *For any sequence $\{e_j^{(p)}\}_{1\leq p<\infty}$ obtained from the discrete version of NSMM, there exists $\tilde{\mathbf{e}}$ in $C(\Omega)$ such that $\{\mathbf{e}^{(p)}\}_{1\leq p<\infty}$ has a subsequence that converges uniformly to $\tilde{\mathbf{e}}$. Furthermore, $\tilde{\mathbf{e}}$ is a stationary point.*

Furthermore:

Before stating the next Theorem, we review the definition of limit point, derived set and Theorem 28.1 of Ostrowski (1960).

Definition 2.3.2. *Assume $\mathcal{S}$ is a subset of a topological space $\mathcal{T}$. A limit point of $\mathcal{S}$ in $\mathcal{T}$ is any point $x \in \mathcal{T}$ such that every open neighbourhood of x contains at least one point, $y \in \mathcal{S}$, such that y is different from x.*

Definition 2.3.3. *Assume $\mathcal{S}$ is a subset of a topological space $\mathcal{T}$. The derived set of $\mathcal{S}$, denoted by $\mathcal{S}'$, is the set of all limit points of $\mathcal{S}$ in $\mathcal{T}$.*

Theorem 2.3.4 (Theorm 28.1 of Ostrowski (1960))**.** *Assume $\mathcal{S}$ is a bounded sequence of points $\mathbf{e}_t, 0 \leq t \leq \infty$, in R^n, for which $\|\mathbf{e}_{t+1} - \mathbf{e}_t\|$ tends to zero as t goes to ∞. Then the derived set $\mathcal{S}'$ of $\mathcal{S}$ is a continuum, which is a closed set of points that cannot be decomposed into the union of two disjoint closed sets of points.*

Note that $\{\Theta^{(p)}\}_{0\leq p<\infty}$ is a bounded sequence in R^{nm}. Using Lemma 2.1.8, the proof of Theorem 2.3.1, the properties of $\tilde{\mathbf{e}}$, and Theorem 28.1 of Ostrowski (1960), we can prove the following result.

Theorem 2.3.5. *Assume (i) through (vi). Then $\lim_{p\to\infty} l(\mathbf{e}_d(\Theta^{(p)}))$ exists and $\{\Theta^{(p)}\}_{0\leq p<\infty}$ converges to a compact connected component of $\{\Theta | l(\mathbf{e}_d(\Theta)) = \lim_{p\to\infty} l(\mathbf{e}_d(\Theta^{(p)}))\}$.*

Each element in the derived set of $\{\Theta^{(p)}\}_{0\leq p<\infty}$ *is a fixed point of the NSMM algorithm, and is also a stationary point of* $l(\mathbf{e}_d(\cdot))$.

Chapter 3
Component-wise ICA

3.1 Model Extension and Estimation

The conditional independence assumption of model (1.1) may seem too strong. One way to introduce some dependence structure among covariates of an observation from a particular mixture component is by linearly transforming each component away from independence. To this end, for any nonnegative function f on R^r, $r \times r$ matrix A, and $\mathbf{x} \in R^r$, define f_{A} as

$$f_{\mathsf{A}}(\mathbf{x}) = f(\mathsf{A}^{-1}\mathbf{x})|\det \mathsf{A}|^{-1}, \tag{3.1}$$

which can be used for describing the density function of a linearly transformed random vector.

As in model (1.1), we still assume that

$$g = \sum_{j=1}^{m} \lambda_j f_j. \tag{3.2}$$

Suppose that if $\mathbf{X}$ is a random vector drawn from the component density f_j,

then there exists an $r \times r$ invertible matrix A_j and a random vector $\mathbf{S}$ such that

$$\mathbf{X} = \mathsf{A}_j \mathbf{S}, \tag{3.3}$$

where $\mathbf{S}$ consists of coordinates that are mutually independent random variables. In other words, $\mathbf{S}$ follows a distribution whose density looks like

$$q_j(\mathbf{s}) = \prod_{k=1}^{r} q_{j,k}(s_k) \tag{3.4}$$

and

$$f_j(\mathbf{x}) = (q_j)_{\mathsf{A}_j}(\mathbf{x}). \tag{3.5}$$

For estimation purposes, we write

$$(e_j)_{\mathsf{A}_j} = \lambda_j f_j, \tag{3.6}$$

so (2.2) holds and in addition $e_j(\mathbf{x}) = \lambda_j q_j(\mathbf{x})$. To reduce ambiguities caused by over-parametrization, we further assume

$$E\{\mathbf{S}\mathbf{S}^{\mathbf{T}}\} = I_r, \tag{3.7}$$

where I_r is the $r \times r$ identity matrix. This helps fix the magnitudes of the independent components. Here, assumptions (3.3), (3.4) and (3.7) are commonly used in the literature on independent component analysis (Hyvarinen et al., 2002).

The $q_{j,k}$ do not follow any parametric form; any univariate density function in $L^1(R)$ is a valid candidate. Hence, we call the model described above a nonparametric ICA mixture model. This model is closely related to the ICA mixture model described in Lee et al. (2000), where each mixing component is parameterized by

finitely many parameters. Here we relax this parametric assumption.

Now we consider the estimation for $\mathbf{e} = (e_1, e_2, ..., e_m)$ and $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, ..., \mathsf{A}_m)$. The idea is to use a penalized smoothed Kullback-Leibler divergence as in Chapter 2. However, the incoporation of the A_j requires some delicacy. It is tempting to look at the Kullback-Leibler divergence between the target density and $\sum_{j=1}^{m}(\mathcal{N}_h e_{j\mathsf{A}_j})(\mathbf{x})$. But applying the smoothing after the linear transformation ignores the different scale and direction of each coordinate of an observation from a mixture component. We believe a better approach is to compare the target density and $\sum_{j=1}^{m}(\mathcal{N}_h e_j)_{\mathsf{A}_j}(\mathbf{x})$ instead of $\sum_{j=1}^{m}(\mathcal{N}_h e_{j\mathsf{A}_j})(\mathbf{x})$. Later we will see that, mathematically, this apporach does lead to sensible estimates consistent with the geometric intuition that, in constructing the density estimates, the smoothing is done before the linear transformation. Hence, we propose to minimize

$$l(\mathbf{e}, \mathsf{A}) = \int g(\mathbf{x}) \log \left[g(\mathbf{x}) / \sum_{j=1}^{m} (\mathcal{N}_h e_j)_{\mathsf{A}_j}(\mathbf{x}) \right] \mathrm{d}\mathbf{x} + \int \left[\sum_{j=1}^{m} (e_j)_{\mathsf{A}_j}(\mathbf{x}) \right] \mathrm{d}\mathbf{x}. \quad (3.8)$$

Figure 3.1 helps visualize why we make this choice through a two dimensional example. On the right hand side, the data are plotted according to their (x, y)-coordinates. In order to determine an appropriate choice of the kernel density in estimating the joint distribution of $(\mathbf{x}, \mathbf{y})$, the data are first transformed to (x^*, y^*) (left hand side), a "normalized" scale where the coordinates are approximately independent and standardized. In the "normalized" scale we put a spherical kernel (represented by circles) with a chosen bandwidth at each data point. Then these kernels are transformed back (which are represented by the ellipses) along with the data points to the original scale (right hand side). We believe this approach is better than the crude method of putting a spherical kernel at each data point in the original scale, which does not discern among relative scales along different

directions.

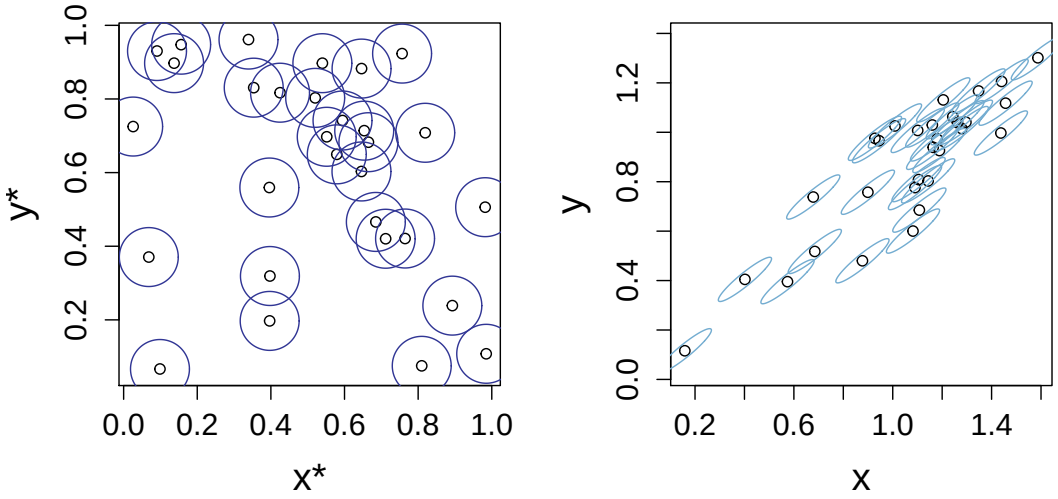


Figure 3.1. Illustration of smoothing followed by linear transformation.On the right, coordinates are on the original scale. After transformation to a normalized scale (on the left), a spherical smoothing kernel is used at each point. Transformating back to the original scale makes the smoothing kernels appears elliptical, as shown on the right.

Now we derive the ICA version of the NSMM algorithm, in a way similar to that presented in section 2.1.4. Given the current estimate $\mathbf{e}^{(0)}$ and $\mathsf{A}^{(0)}$, we obtain

$$
\begin{aligned}
& l(\mathbf{e},\mathsf{A}) - l(\mathbf{e}^{(0)},\mathsf{A}^{(0)}) \\
&= -\int g(\mathbf{x}) \log \sum_{j=1}^{m} \frac{(\mathcal{N}_h e_j^{(0)})_{\mathsf{A}_j^{(0)}}(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(0)})_{\mathsf{A}_{j'}^{(0)}}(\mathbf{x})} \cdot \frac{(\mathcal{N}_h e_j)_{\mathsf{A}_j}(\mathbf{x})}{(\mathcal{N}_h e_j^{(0)})_{\mathsf{A}_j^{(0)}}(\mathbf{x})} \mathrm{d}\mathbf{x} + \int \left(\sum_{j=1}^{m} (e_j)_{\mathsf{A}_j} - \sum_{j=1}^{m} (e_j^{(0)})_{\mathsf{A}_j^{(0)}} \right) \\
&\leq -\int g(\mathbf{x}) \sum_{j=1}^{m} \frac{(\mathcal{N}_h e_j^{(0)})_{\mathsf{A}_j^{(0)}}(\mathbf{x})}{\sum_{j'=1}^{m} (\mathcal{N}_h e_{j'}^{(0)})_{\mathsf{A}_{j'}^{(0)}}(\mathbf{x})} \cdot \log \frac{(\mathcal{N}_h e_j)_{\mathsf{A}_j}(\mathbf{x})}{(\mathcal{N}_h e_j^{(0)})_{\mathsf{A}_j^{(0)}}(\mathbf{x})} \mathrm{d}\mathbf{x} + \int \left(\sum_{j=1}^{m} (e_j)_{\mathsf{A}_j} - \sum_{j=1}^{m} (e_j^{(0)})_{\mathsf{A}_j^{(0)}} \right).
\end{aligned}
\tag{3.9}
$$

Thus, if we let

$$b^{(0)}(\mathbf{e},\mathsf{A}) = -\int g(\mathbf{x})\sum_{j=1}^{m} w_j^{(0)}(\mathbf{x})\cdot\log\left(\mathcal{N}_h e_j\right)_{\mathsf{A}_j}(\mathbf{x})\mathrm{d}\mathbf{x} + \int\left(\sum_{j=1}^{m}(e_j)_{\mathsf{A}_j}\right), \quad (3.10)$$

where

$$w_j^{(0)}(\mathbf{x}) = \frac{\left(\mathcal{N}_h e_j^{(0)}\right)_{\mathsf{A}_j^{(0)}}(\mathbf{x})}{\sum_{j'=1}^{m}\left(\mathcal{N}_h e_{j'}^{(0)}\right)_{\mathsf{A}_{j'}^{(0)}}(\mathbf{x})}, \quad (3.11)$$

then

$$l(\mathbf{e},\mathsf{A}) - l(\mathbf{e}^{(0)},\mathsf{A}^{(0)}) \le b^{(0)}(\mathbf{e},\mathsf{A}) - b^{(0)}(\mathbf{e}^{(0)},\mathsf{A}^{(0)}). \quad (3.12)$$

Therefore $b^{(0)}$ majorizes l at $(\mathbf{e}^{(0)},\mathsf{A}^{(0)})$ up to an additive constant. Now we consider how to minimize $b^{(0)}(\mathbf{e},\mathsf{A})$, subject to the assumptions on $\mathbf{e}$ stated in (3.2) through (3.6). That is, for each j, $1 \le j \le m$, we wish to minimize

$$b_j^{(0)}(e_j,\mathsf{A}_j) = -\int g(\mathbf{x})w_j^{(0)}(\mathbf{x})\cdot\log\left(\mathcal{N}_h e_j\right)_{\mathsf{A}_j}(\mathbf{x})\mathrm{d}\mathbf{x} + \int (e_j)_{\mathsf{A}_j}. \quad (3.13)$$

Since $\int (e_j)_{\mathsf{A}_j} = \int e_j$, the change of variables $\mathbf{x} = \mathsf{A}_j\mathbf{y}$ transforms (3.13) into

$$-\int g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})\cdot\log\left\{(\mathcal{N}_h e_j)(\mathbf{y})|\det\mathsf{A}_j|^{-1}\right\}|\det\mathsf{A}_j|\mathrm{d}\mathbf{y} + \int e_j. \quad (3.14)$$

For now, consider A_j to be fixed. Profiling out other parameters, we find that minimizing $b_j^{(0)}(e_j,\mathsf{A}_j)$ with A_j fixed is equivalent to minimizing

$$-\int g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})\int \bar{s}_h(\mathbf{u},\mathbf{y})\log e_j(\mathbf{u})|\det\mathsf{A}_j|\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{y} + \int e_j, \quad (3.15)$$

which equals

$$-\iint g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})\bar{s}_h(\mathbf{u},\mathbf{y})\left[\sum_{k=1}^{r}\log e_{j,k}(u_k)+\log\theta_j\right]|\det\mathsf{A}_j|\mathrm{d}\mathbf{u}\mathrm{d}\mathbf{y}+\int e_j\mathrm{d}\mathbf{u}. \tag{3.16}$$

Picking a specific k and integrating Expression (3.16) with respect to all $\mathrm{d}u_l$ except $\mathrm{d}u_k$ leads to, up to a term free of $e_{j,k}$,

$$\begin{aligned}-\iint g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})s_h(u_k,y_k)\log e_{j,k}(u_k)|\det\mathsf{A}_j|\mathrm{d}u_k\mathrm{d}\mathbf{y}\\ +\theta_j\left[\prod_{l\neq k}\int e_{j,k}(u_l)\mathrm{d}u_l\right]e_{j,k}\mathrm{d}u_k.\end{aligned} \tag{3.17}$$

Viewing this expression as an integral with respect to $\mathrm{d}u_k$, we minimize it by minimizing the value of its integrand at each point. Differentiating with respect to $e_{j,k}(u_k)$ and setting it equal to zero gives

$$-\frac{\int g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})s_h(u_k,y_k)|\det\mathsf{A}_j|\mathrm{d}\mathbf{y}}{e_{j,k}(u_k)}+\theta_j\left[\prod_{l\neq k}\int e_{j,k}(u_l)\mathrm{d}u_l\right]=0, \tag{3.18}$$

yielding

$$\hat{e}_{j,k}(u_k)\propto\int g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})s_h(u_k,y_k)\mathrm{d}\mathbf{y}, \tag{3.19}$$

which implies

$$\hat{e}_j(\mathbf{u})=\alpha_j\prod_{k=1}^{r}\int g(\mathsf{A}_j\mathbf{y})w_j^{(0)}(\mathsf{A}_j\mathbf{y})s_h(u_k,y_k)\mathrm{d}\mathbf{y} \tag{3.20}$$

for some constant α_j. To find α_j, we plug (3.20) into (3.15), differentiate with respect to α_j, and set the result equal to zero to obtain

$$\alpha_j=\frac{|\det\mathsf{A}_j|}{\left[\int g(\mathsf{A}_j\mathbf{x})w_j^{(0)}(\mathsf{A}_j\mathbf{x})\ \mathrm{d}\mathbf{x}\right]^{r-1}}, \tag{3.21}$$

which implies that

$$\hat{e}_j(\mathbf{u}) = \frac{|\det \mathsf{A}_j|}{\left[\int g(\mathsf{A}_j\mathbf{x}) w_j^{(0)}(\mathsf{A}_j\mathbf{x})\, \mathrm{d}\mathbf{x}\right]^{r-1}} \cdot \prod_{k=1}^{r} \int g(\mathsf{A}_j\mathbf{y}) w_j^{(0)}(\mathsf{A}_j\mathbf{y}) s_h(u_k, y_k) \mathrm{d}\mathbf{y}, \quad (3.22)$$

which can be equivalently written as

$$\hat{e}_j(\mathbf{u}) = \left[P \circ S_h(|\det \mathsf{A}_j| \cdot (g \cdot w_j^{(p)}) \circ \mathsf{A}_j)\right](\mathbf{u}). \quad (3.23)$$

Combining (3.2) and (3.23) yields

$$\begin{aligned}
(\hat{e}_j)_{\mathsf{A}_j}(\mathbf{u}) &= \frac{|\det \mathsf{A}_j|^{-1}}{\left[\int g(\mathbf{x}) w_j^{(0)}(\mathbf{x})\, \mathrm{d}\mathbf{x}\right]^{r-1}} \cdot \prod_{k=1}^{r} \int g(\mathbf{v}) w_j^{(0)}(\mathbf{v}) s_h[(\mathsf{A}_j^{-1}\mathbf{u})_k, (\mathsf{A}_j^{-1}\mathbf{v})_k] \mathrm{d}\mathbf{v} \\
&= P \circ S_h[(g \cdot w_j^{(p)}) \circ \mathsf{A}_j](\mathsf{A}_j^{-1}\mathbf{u}) \\
&= S_h \circ P[(g \cdot w_j^{(p)}) \circ \mathsf{A}_j](\mathsf{A}_j^{-1}\mathbf{u}) \\
&= P\{[(S_h)_{\mathsf{A}_j}(g \cdot w_j^{(p)})] \circ \mathsf{A}_j\}(\mathsf{A}_j^{-1}\mathbf{u}) \\
&= \left[P_{\mathsf{A}_j} \circ (S_h)_{\mathsf{A}_j}(g \cdot w_j^{(p)})\right](\mathbf{u}), \quad (3.24)
\end{aligned}$$

where

$$(S_h)_{\mathsf{A}_j} f(\mathbf{x}) = \int |\det \mathsf{A}_j|^{-1} \tilde{s}_h(\mathsf{A}_j^{-1}\mathbf{x}, \mathsf{A}_j^{-1}\mathbf{u}) f(\mathbf{u}) \mathrm{d}\mathbf{u} \quad (3.25)$$

and

$$P_{\mathsf{A}_j} f(\mathbf{u}) = [P(f_{\mathsf{A}_j^{-1}})]_{\mathsf{A}_j}(\mathbf{u}) = [P(f \circ \mathsf{A}_j)](\mathsf{A}_j^{-1}\mathbf{u}). \quad (3.26)$$

For the geometric interpretation associated with Figure 3.1 and algebraic convenience, we note that, in general, for any nonnegative function f on R^r,

$$S_h(f \circ \mathsf{A}_j) = (S_h)_{\mathsf{A}_j}(f) \circ \mathsf{A}_j. \quad (3.27)$$

Equantion (3.22) also implies that, according to the assumption in (3.4),

$$\hat{q}_{j,k}(u_k) = \frac{|\det \mathsf{A}_j|}{\int (g \cdot w_j^{(0)})} \int g(\mathsf{A}_j \mathbf{y}) w_j^{(0)}(\mathsf{A}_j \mathbf{y}) s_h(u_k, y_k) \mathrm{d}\mathbf{y} \tag{3.28}$$

and

$$\int \hat{e}_j(\mathbf{u}) \mathrm{d}\mathbf{u} = \int g(\mathbf{x}) w_j^{(0)}(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \tag{3.29}$$

If we plug $\hat{e}_j(\mathbf{u})$ into the majorized component (3.13), the only terms involving A_j are

$$\begin{aligned} &- |\det \mathsf{A}_j| \sum_{k=1}^{r} \int \left[\int g(\mathsf{A}_j \mathbf{y}) w_j^{(0)}(\mathsf{A}_j \mathbf{y}) s_h(u_k, y_k) \mathrm{d}\mathbf{y} \right] \log \left[\int g(\mathsf{A}_j \mathbf{y}) w_j^{(0)}(\mathsf{A}_j \mathbf{y}) s_h(u_k, y_k) \mathrm{d}\mathbf{y} \right] \mathrm{d}u_k \\ &- (r-1) \log |\det \mathsf{A}_j| \int g(\mathbf{x}) w_j^{(0)}(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \end{aligned} \tag{3.30}$$

Minimizing (3.30) is equivalent to minimizing

$$\log |\det \mathsf{A}_j| + \sum_{k=1}^{r} \int \hat{q}_{j,k} \log \hat{q}_{j,k} \tag{3.31}$$

with respect to A_j, where $\hat{q}_{j,k}$ depends on A_j through (3.34).

In Expression (3.31), $\hat{q}_{j,k}$ is the kth margin of the kernel smoothed version of $(g \cdot w_j^{(0)})_{\mathsf{A}_j^{-1}} / \int g \cdot w_j^{(0)}$. In the discrete case, $\hat{q}_{j,k}$ is the kth margin of the kernel density estimation based on the linearly transformed (by A_j^{-1}) weighted observed data set, where the weight for the data point $\mathbf{x}_i$ is $w_j^{(0)}(\mathbf{x}_i)$. Let us denote this weighted data set by $\mathsf{D}_j^{(0)}$ and hence its linear transformation by $\mathsf{A}_i^{-1}\mathsf{D}_j^{(0)}$. By (3.34), the optimization mechanism at the current step views $\mathsf{A}_j^{-1}\mathsf{D}_j^{(0)}$ as a weighted sample generated from the unknown density function q_j, where $\mathsf{D}_j^{(0)}$ is a weighted sample from the jth mixing component and A_j^{-1} is the matrix that recovers the associated ICA signals. Let us call A_j^{-1} a recovering matrix. Note that $|\det A_j|$ can be treated

as fixed given the weighted data $\mathsf{A}_j^{-1}\mathsf{D}_j^{(0)}$, due to (3.7). The second term in (3.31) is an estimate of the sum of marginal entropies of q_j, which is equal, up to a term that does not involve A_j, to the mutual information of marginals of q_j. According to Hyvarinen et al. (2002), minimizing mutual information in this setting (that is, minimizing the mutual information of $\mathsf{A}_j^{-1}\mathbf{S}$ given a randomly chosen weighted sample from $\mathbf{S}$) is essentially equivalent to maximizing some negentropy of the marginals, and this leads to the linear decomposition with least dependence measure (Hyvarinen et al., 2002). In other words, estimating A_j, $1 \leq j \leq m$, at the current step leads to independent component analysis based on the weighted data set $\mathsf{D}_j^{(0)}$, and thus can be acheived by existing ICA algorithms.

To summarize, in general, the NSMM-ICA iterative algorithm will iterate as follows, starting from $(\mathbf{e}^{(p)}, \mathsf{A}^{(p)})$:

Majorization-Step: For $1 \leq j \leq m$, compute

$$w_j^{(p)}(\mathbf{x}) = \frac{(\mathcal{N}_h e_j^{(p)})_{\mathsf{A}_j^{(p)}}(\mathbf{x})}{\sum_{j=1}^{m} (\mathcal{N}_h e_j^{(p)})_{\mathsf{A}_j^{(p)}}(\mathbf{x})}. \tag{3.32}$$

ICA-Step: Use ICA techniques to find $\mathsf{A}_1^{(p+1)}, \mathsf{A}_2^{(p+1)}, ..., \mathsf{A}_m^{(p+1)}$ under (3.7), that minimize for each j, $1 \leq j \leq m$,

$$\sum_{k=1}^{r} \int \hat{q}_{j,k}^{(p+1)} \log \hat{q}_{j,k}^{(p+1)}, \tag{3.33}$$

where

$$\hat{q}_{j,k}^{(p+1)}(u_k) = \frac{|\det \mathsf{A}_j^{(p+1)}|}{\int (g \cdot w_j^{(p)})} \int g(\mathsf{A}_j^{(p+1)}\mathbf{y}) w_j^{(p)}(\mathsf{A}_j^{(p+1)}\mathbf{y}) s_h(u_k, y_k) \mathrm{d}\mathbf{y}. \tag{3.34}$$

Minimization-Step: Let

$$e_j^{(p+1)}(\mathbf{u}) = \hat{\lambda}_j^{(p+1)} \hat{q}_j^{(p+1)}(\mathbf{u}) = \hat{\lambda}_j^{(p+1)} \prod_{k=1}^{r} \hat{q}_{j,k}^{(p+1)}(u_k), \tag{3.35}$$

where

$$\hat{\lambda}_j^{(p+1)} = \int (g \cdot w_j^{(p)}). \tag{3.36}$$

3.2 Integrating NSMM with FastICA

Section 3.1 suggests alternating NSMM and ICA methods to form an iterative algorithm for the estimation of the nonparametric ICA mixture model. Among all ICA techniques, FastICA (Hyvarinen et al., 2002) is quite efficient and well tested. This consideration motivates the implementation of the following algorithm which integrates NSMM and FastICA.

To interweave each iteration of NSMM with FastICA, FastICA will be applied to a weighted dataset, where the weights for component j are determined by the estimates, given the information available at the present iteration, of the probability that each observation falls into component j.

Assume we are given raw data as a matrix $\mathsf{X}^T = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}^T$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, x_{i3}, ..., x_{ir})^T$ for $1 \leq i \leq n$. Let us sketch out the algorithm, which iterates through step 1 through 4 until some convergence criterion is met.

1. E-step: Let

$$p_{ij}^{(t)} = \frac{\lambda_j^{(t)} (\mathcal{N}_h q_j^{(t)})_{\mathsf{A}_j^{(t)}}(\mathbf{x}_i)}{\sum_{j\prime=1}^{m} \lambda_{j\prime}^{(t)} (\mathcal{N}_h q_{j\prime}^{(t)})_{\mathsf{A}_{j\prime}^{(t)}}(\mathbf{x}_i)}$$

$$= \frac{\lambda_j^{(t)} \left|\det \mathsf{A}_j^{(t)}\right|^{-1} \prod_{k=1}^{r} \left(\mathcal{N}_h g_{j,k}^{(t)}\right) \left([(\mathsf{A}_j^{(t)})^{-1}\mathbf{x}_i]_k\right)}{\sum_{j'=1}^{m} \lambda_{j'}^{(t)} \left|\det \mathsf{A}_{j'}^{(t)}\right|^{-1} \prod_{k=1}^{r} \left(\mathcal{N}_h g_{j',k}^{(t)}\right) \left([(\mathsf{A}_{j'}^{(t)})^{-1}\mathbf{x}_i]_k\right)}. \tag{3.37}$$

Here the $(\mathsf{A}_j^{(t)})^{-1}$ are the estimates for the component-specific uncorrelating matrices at iteration t.

2. M-step:

$$\lambda_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} p_{ij}^{(t)}. \tag{3.38}$$

3a. Centering FastICA-step for component j:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - \frac{\sum_{i=1}^{n} \mathbf{x}_i p_{ij}^{(t)}}{\sum_{i=1}^{n} p_{ij}^{(t)}}. \tag{3.39}$$

3b. Decorrelating FastICA-step for component j: We first obtain the eigenvalue decomposition as

$$\frac{\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T p_{ij}^{(t)}}{\sum_{i=1}^{n} p_{ij}^{(t)}} = EDE^T, \tag{3.40}$$

then let

$$V = ED^{-1/2}E^T \tag{3.41}$$

and

$$\mathbf{z}_i = V\mathbf{x}_i, \qquad i = 1, ..., n. \tag{3.42}$$

Therefore,

$$\frac{\sum_{i=1}^{n} \mathbf{z}_i \mathbf{z}_i^T p_{ij}^{(t)}}{\sum_{i=1}^{n} p_{ij}^{(t)}} = VEDE^TV^T = I. \tag{3.43}$$

The transformed data $\mathbf{z}_i$ with weights $p_{ij}^{(t)}$, $1 \leq i \leq n$, are thus whitened (i.e., their cordinates are uncorrelated and standardized) according to (3.43). Since that $\mathbf{Z} = V\mathbf{X} = V\mathsf{A}_j\mathbf{S}$, we need to first estimate $(V\mathsf{A}_j)^{-1}$ (of which the ith row is the same as $\mathbf{w}_i$ in (3.44) below) and multiply it by V on the right to get an update of A_j^{-1}.

3c. Symmetric orthogonalization FastICA-step for component j: The goal is to pick

$$\mathbf{w}_i \in R^r \text{ such that } \mathbf{w}_i^T\mathbf{w}_i = 1, i = 1, ..., r.$$

The first time we enter step 3c, we may choose such $\mathbf{w}_i$ arbitrarily, for example setting $\mathbf{w}_i$ equal to the ith standard basis vector. At succeeding iterations, we begin an inner loop to modify the $\mathbf{w}_i$ from their previous values: We begin with

$$\mathbf{w}_i \leftarrow \frac{\sum_{i=1}^{n} \mathbf{z}_i g(\mathbf{w}_i^T\mathbf{z}_i)p_{ij}^{(t)}}{\sum_{i=1}^{n} p_{ij}^{(t)}} - \mathbf{w}_i \frac{\sum_{i=1}^{n} g\prime(\mathbf{w}_i^T\mathbf{z}_i)p_{ij}^{(t)}}{\sum_{i=1}^{n} p_{ij}^{(t)}}, \tag{3.44}$$

where g may be chosen to be either $g(y) = \tanh(\alpha_1 y)$ for some $1 \leq \alpha_1 \leq 2$ or $g(y) = y\exp(-y^2/2)$ (Hyvarinen et al., 2002). Denote $W = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_r]^T$ and then symmetrize and orthogonalize by

$$W \leftarrow (WW^T)^{-1/2}W. \tag{3.45}$$

Iteratively update the $w_i, i = 1, ..., r$ using Equation (3.44) and (3.45) until convergence is achieved. More precisely, we choose a tolerance τ and stop updating when

$$\max_{1 \leq i \leq r} \left\{ \left| |\left(w_i^{(previous)}\right)^T \cdot w_i^{(current)}| - 1 \right| \right\} \leq \tau. \tag{3.46}$$

Finally, set

$$\mathsf{A}_j^{(t+1)} = V^{-1}W^{-1}. \tag{3.47}$$

4. Non-parametric density estimation step: Let

$$q_{jk}^{(t+1)}(\mathbf{u}) = \frac{\sum_{i=1}^{n} p_{ij}^{(t+1)} \frac{1}{h} K\left(\frac{\mathbf{u} - [(\mathsf{A}_j^{(t+1)})^{-1}\mathbf{x}_i]_k}{h}\right)}{\sum_{i=1}^{n} p_{ij}^{(t+1)}}. \tag{3.48}$$

Empirical evidence shows that NSMM and the npEM algorithm of Benaglia, Chauveau, & Hunter (2009) tend to give very similar estimates (Levine et al., 2011). The reason is that usually $\mathcal{N}_h f$ is close to f itself. So the smoothed version of the algorithm can possibly be replaced by the non-smoothed version when taking into account the fact that the former is more computationally burdensome. When implementing the algorithm above, we therefore modified the E-step to be the non-smoothed version:

$$\begin{aligned} p_{ij}^{(t)} &= \frac{\lambda_j^{(t)} f_j(\mathbf{x}_i)}{\sum_{j\prime=1}^{m} \lambda_{j\prime}^{(t)} f_{j\prime}(\mathbf{x}_i)} \\ &= \frac{\lambda_j^{(t)} \left|\det \mathsf{A}_j^{(t)}\right|^{-1} \prod_{k=1}^{r} g_{j,k}^{(t)}\left([(\mathsf{A}_j^{(t)})^{-1}\mathbf{x}_i]_k\right)}{\sum_{j\prime=1}^{m} \lambda_{j\prime}^{(t)} \left|\det \mathsf{A}_{j\prime}^{(t)}\right|^{-1} \prod_{k=1}^{r} g_{j\prime,k}^{(t)}\left([(\mathsf{A}_{j\prime}^{(t)})^{-1}\mathbf{x}_i]_k\right)}. \end{aligned} \tag{3.49}$$

3.3 Bandwidth Selection

Until now, the bandwidth h is assumed to be chosen beforehand and to remain constant at all iterations of the NSMM-ICA algorithm. However, as in almost any statistical procedure involving non-parametric density estimation, the choice of

the bandwidth, or smoothing parameter, is one of the most important aspect in practice. For instance, it is widely agreed that the choice of bandwidth is more critical than the choice of the kernel. General statistical insights about this topic are discussed by Eggermont & LaRiccia (2001). Through minimizing Asymptotic Mean Integrated Squared Error (AMISE) and a plug-in method, Silverman (1986) suggests, for univariate kernel density estimation,

$$h = 0.9 \cdot \min\{\mathrm{SD}, \mathrm{IQR}/1.349\} \cdot (n)^{-0.2}, \tag{3.50}$$

where SD and IQR are the standard deviation and interquartile range of the data, respectively, and n is the sample size.

For the npEM algorithm developed by Benaglia, Chauveau, & Hunter (2009), the bandwidth is chosen at the start of the algorithm, with the same value used for all components and blocks, by using Silverman's rule of thumb (3.50), where all coordinate values of the sample are grouped together treated as one big data set of univariate observations and so n in (3.50) replaced by the product of the true sample size and the dimension r. However, this uniform choice of bandwidth allows no flexibility and is liable to overestimate or underestimate the ideal bandwidth and thus create difficulties for the estimation and classification. For example, for the analysis of the water level data by Benaglia, Chauveau, & Hunter (2009), this method calculates the bandwidth to be 1.47, which leads to "bumpy-looking" density estimates, and so the authors use a larger value of the bandwidth instead. This problem is discussed in Benaglia et al. (2011), which proposes an iterative scheme for bandwidth selection that leads to one bandwidth for each component and each block. The scheme is based on Silverman's rule of thumb but with a weighted sample iteratively updated by the EM part of the algorithm. For iteration

t, component j and block l, the bandwidth is updated through

$$h_{j,l}^{t+1} = 0.9 \cdot \min\left\{\mathrm{SD}_{j,l}^{t+1}, \mathrm{IQR}_{j,l}^{t+1}/1.349\right\} \cdot (nC_l\lambda_j^{t+1})^{-0.2}, \tag{3.51}$$

where $nC_l\lambda_j^{t+1}$ is the size of the weighted sample for the current iteration for each component and block, and $\mathrm{SD}_{j,l}^{t+1}$ and $\mathrm{IQR}_{j,l}^{t+1}$ are respectively the standard deviation and interquartile range of this weighted sample. This approach takes into account the mixture structure and difference between components and blocks in choosing a bandwidth and shows improved density estimation. The npMSL algorithm implemented by Levine et al. (2011) also has the option of iteratively updating the bandwidth in the same way, as explored by Chauveau, Hunter, & Levine (2014).

In the NSMM-ICA algorithm we have developed, a single fixed bandwidth calculated from the data will not be sensible, especially because the scale is now changing according to the ICA framework. Thus we propose an iterative scheme for choosing the bandwidth similar to that of Benaglia et al. (2011), whereby

$$h_{j,k}^{t+1} = 0.5 \cdot \mathrm{SD}_{j,k}^{t+1} \cdot (n\lambda_j^{t+1})^{-0.2} = 0.5 \cdot (n\lambda_j^{t+1})^{-0.2}. \tag{3.52}$$

Due to label switching, block structure among coordinates is not implemented. Also for simplicity we replace $\min\left\{\mathrm{SD}_{j,l}^{t+1}, \mathrm{IQR}_{j,l}^{t+1}/1.349\right\}$ by $\mathrm{SD}_{j,l}^{t+1}$, which turns out to equal 1. We propose the ad hoc choice of 0.5, rather than Silverman's 0.9, as the coefficient in the formula in order to capture fine features of the density for better performance in the classification task. One reason for this choice is our experience that using 0.9 tends to oversmooth the estimated density. Simulation studies and applications we have run suggest that (3.52) works well.

Chapter 4

Computation and Applications

This chapter considers both simulation studies and analysis of real data sets.

4.1 R package: icamix

We have developed a package for the R environment (R Core Team, 2014), called **icamix** which implements the NSMM-ICA algorithm for the discrete case described in Section 3.2. It makes use of the **Rcpp** (Eddelbuettel & François, 2011; Eddelbuettel, 2013) and **RcppArmadillo** (Eddelbuettel & Sanderson, 2014) packages for compiling and calling C++ code to speed up the calculations. The R function `EMFASTICAALG` serves as the front-end user interface. Features of this function are summarized in Table 4.1 and Table 4.2. The **icamix** packages and the packages on which it depends are publicly available on CRAN, the Comprehensive R Archive Network (http://cran.r-project.org/).

Table 4.1. Input Arguments of `EMFASTICAALG`

Input Arguments	
DataMatrix	A matrix of which the rows are data entries. Its dimension is n by r.
numCluster	Predetermined number of mixing components m.
h	Bandwidth. If h is set equal zero (default), iterative bandwidth selection will be used.
maxiter	Maximum number of iterations. Default is 300.
icaiter	Maximum number of ICA iterations in each step. Default is 150.
tol	Threshold that defines convergence (of the outer loop). Default is 10^{-6}.
verb	`TRUE` (default) or `FALSE` indicating whether to print out info at each iteration.
combine	`TRUE` (default) or `FALSE` indicating whether to implement the ICA step.

4.2 Simulation Studies

To assess the proposed method, several simulation studies have been carried out. In the following, simulated data sets have been used for testing the algorithms we have implemented in the R package **icamix**.

4.2.1 Well Separated Non-Normal Components:

In this first simulation study, the proposed method is tested on a data set with simple structure and clearly separated classes and is seen to perform satisfactorily

Table 4.2. Output Values of `EMFASTICAALG`

Values of Output List	
$ InputData	A matrix of which the columns are data entries. Its dimension is r by n.
$ Lambdas	A matrix where rows store estimated mixing weights from each iteration.
$ WMtrs	List of $r \times r$ unmixing matrices for each of the m clusters.
$ WUnmixZ	List of unmixing matrices for whitened data for each of the m clusters.
$ OriginalSignals	List of Recovered ICA components for each of the m clusters.
$ ProductDensity	m by n matrix where each row stores the estimated density value of the observed data points for each of the m clusters.
$ MembershipProbs	n by m matrix where each row stores the component membership probabilities of the corresponding data point.
$ ObjValue	Vector holding values of data loglikelihood.
$ ICABandWidth	Matrix holding choices of bandwidth for original signals.

as expected. We generate random data drawn from three mixture components, each of dimension two, and use the NSMM-ICA algorithm to learn the mixing parameters, component densities and component unmixing matrices and to classify the data.

The first mixture component is a linear transformation, represented by the transformation matrix $\left(\begin{smallmatrix} 5 & 0 \\ 0 & 1.7 \end{smallmatrix}\right)$, of a 2-dimensional random vector with independent

coordinates, each of which follows a $t(7)$ distribution up to an additive constant. The second mixture component is a linear transformation, represented by the transformation matrix $\left(\begin{smallmatrix} 6 & -12 \\ 9 & 15 \end{smallmatrix}\right)$, of a 2-dimensional random vector with independent coordinates, each of which follows a uniform distribution on $[1, 3]$. The third mixture component is a linear transformation, represented by the transformation matrix $\left(\begin{smallmatrix} 1 & 1.2 \\ 0.6 & 3 \end{smallmatrix}\right)$, of a 2-dimensional random vector with independent coordinates each of which follows a Laplacian distribution with location parameter 20 and scale parameter 2.

From each mixture component, 500 random samples are collected. And a total of 1500 samples form the data set. On a side note, technically, this is not a sample from a mixture model with mixing weights $\lambda_1 = \lambda_2 = \lambda_3 = 1/3$: By fixing the number of samples from each component to make estimation of $\boldsymbol{\lambda}$ easier, we are introducing a small amount of dependence. In a true sample from the model, however, the number from each component would be random. In a typical run, our NSMM-ICA algorithm takes around 200 iterations and the computing time is about 100 seconds. The final bandwidth selected by the algorithm are 0.145, 0.144 and 0.144 for each of the three mixture components respectively on the independent signal scale.

Figure 4.1 shows that the true mixture component class membership has been fully recovered. Using the estimated most likely membership and unmixing matrix for each mixture component, we can also plot the recovered independent signals (Figure 4.2).

We can also evaluate the estimated density for each component and compare them with that of the true model. We calculate the integrated squared errors between the estimates and the true densities to be 7.914674e-05 and 1.761606e-04, respectively.

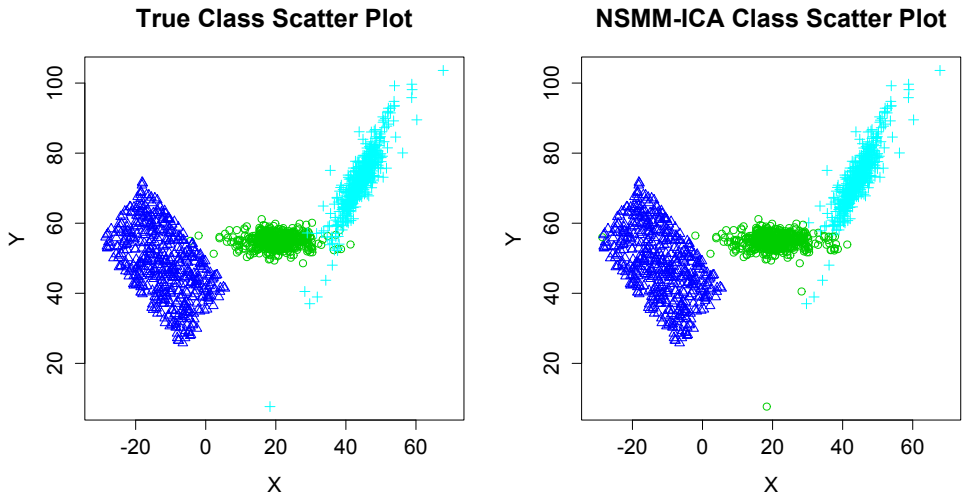


Figure 4.1. Simulation 1: Scatter plot with true (left) and estimated (right) class info.

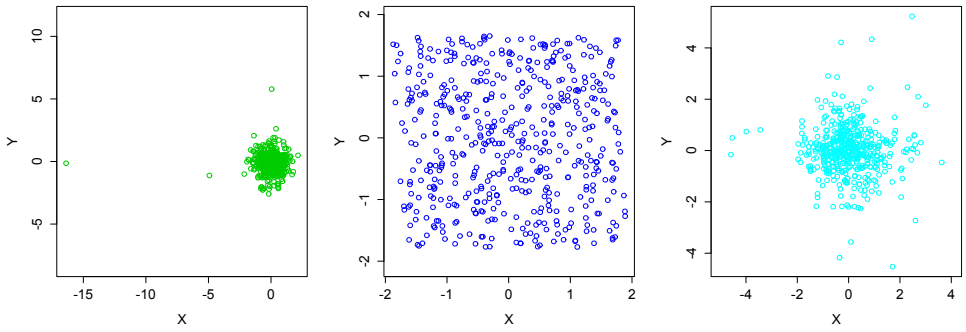


Figure 4.2. Simulation 1: Recovered independent signals for bivariate t distributed (left), uniformly distributed (center) and Laplace distributed (right) data.

Finally, we plot the data together with contours of the estimated component densities in Figure 4.3. For each true component, we plot the column vectors (in red) of the true transformation matrix, pointing out from the center of that component. We also do this for the transformation matrix for each estimated component (in green and blue). The plots show clearly that, because of the non-Gaussian nature of the data, we are able to recover the transformation matrix up to rescaling the permuting of columns.

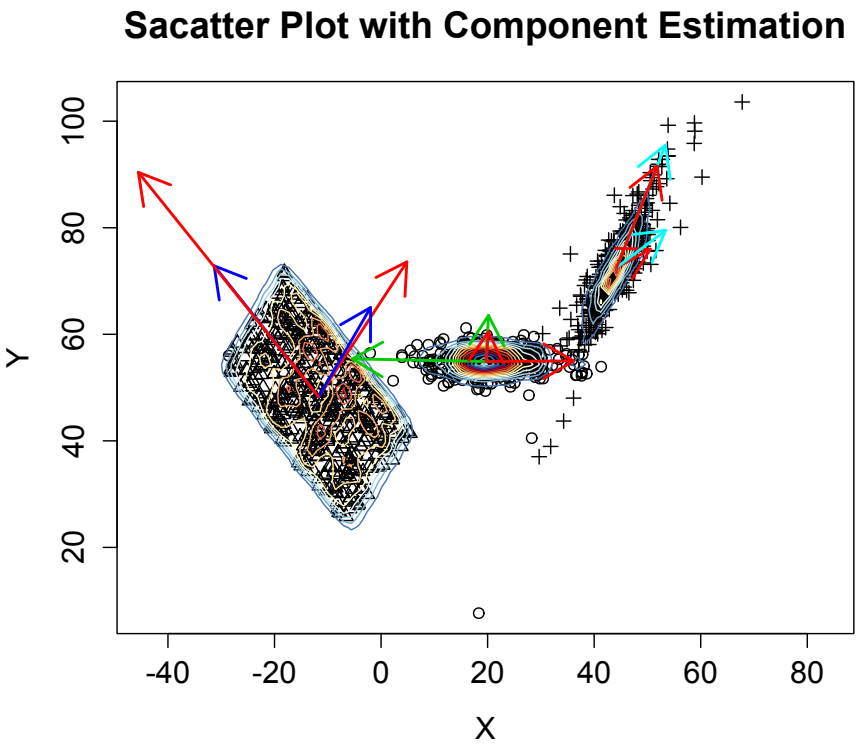


Figure 4.3. Simulation 1: Contours of estimated component densities. The red arrows visualize the columns of the true transformation matrices while the others indicate those of the estimated transformation matrices.

After repeating the simulation 500 times, we calculate the squre root of the mean integrated squared error (MISE) for each component density (Table 4.3). To prevent label switching, we match the components according to their relative position, say, revealed by the weighted avergage of the first coordinate of the samples. For comparision, we also report the results obtained by the npEM algorithm. Table 4.3 shows that NSMM-ICA gives significantly lower MISE values for all components where the conditional independence assumption fails. Figure 4.4 summarizes the classification rate of each of the two methods in the 500 repetitions of the simulation, which confirms that NSMM-ICA performs better.

For reproducing the results and graphs, the R scripts "simulation1.R" and

Table 4.3. Simulation 1: Squareroot of MISE.

	Component 1	Component 2	Component 3
npEM	0.02405972	0.02027120	0.05468689
NSMM-ICA	0.013296438	0.009198624	0.013456935

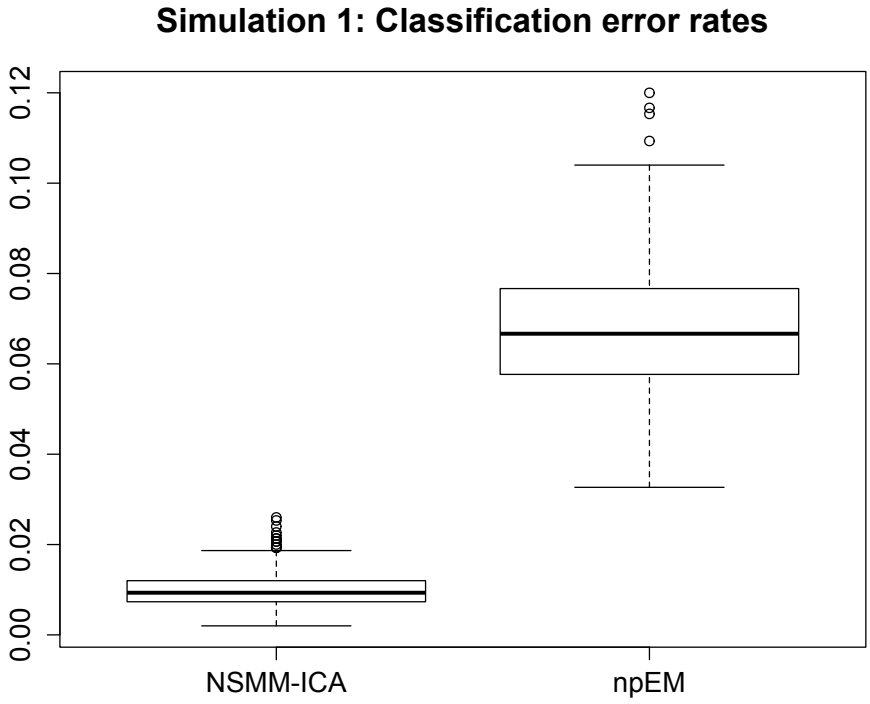


Figure 4.4. Simulation 1: classification error rate of NSMM-ICA (left) and of npEM (right).

"simulation1mise.R" are included in Appendices and , respectively.

4.2.2 Well-Separated Normal Components

In this second simulation study, we consider mixture components that are multivariate normal random vectors. We generate random data drawn from two mixture components, each of dimension two, and use the NSMM-ICA algorithm to learn

the mixing parameters, component densities and component unmixing matrices and to classify the data. Furthermore, we assess the performance of our proposed method by reporting the square root of MISE.

The first mixture component is a 2-dimensional random vector with independent coordinates each of which is normal with mean and variance 1. The second mixture component is a linear transformation, represented by the transformation matrix $\left(\begin{smallmatrix} 2 & -3 \\ 1 & 1.6 \end{smallmatrix}\right)$, of a 2-dimensional random vector with independent coordinates each of which is normal with mean 5 and variance 1.

We draw 360 and 640 random samples from the first and second components, respectively. So the mixing weights are approximately $\lambda_1 = 0.36$ and $\lambda_2 = 0.64$, respectively. On average, our NSMM-ICA algorithm takes less than 5 seconds and 30 iterations to analyze the data. The final bandwidth selected by the algorithm are 0.137 and 0.154 for each of the two mixture components respectively on the independent signal scale.

Figure 4.5 shows the contours of the estimated component densities from a typical run of the algorithm on one simulated data set. For each true compnent, we plot the column vectors (in red) of the true transformation matrix, pointing out from the center of that component. We also do this for the transformation matrix for each estimated component (in green and blue). Figure 4.5 shows clearly that, because of the gaussian nature of the data, we are not able to recover the transformation matrix. Figure 4.6 shows the estimated component densities by the NSMM-ICA and npEM algorithms.

After repeating the simulation 500 times, we calculate the squre root of MISE for each component density (Table 4.4). To prevent label switching, we match the components according to their mixing weights. For comparision, we also report the results obtained by the npEM algorithm. In terms of $\sqrt{\text{MISE}}$, NSMM-ICA and

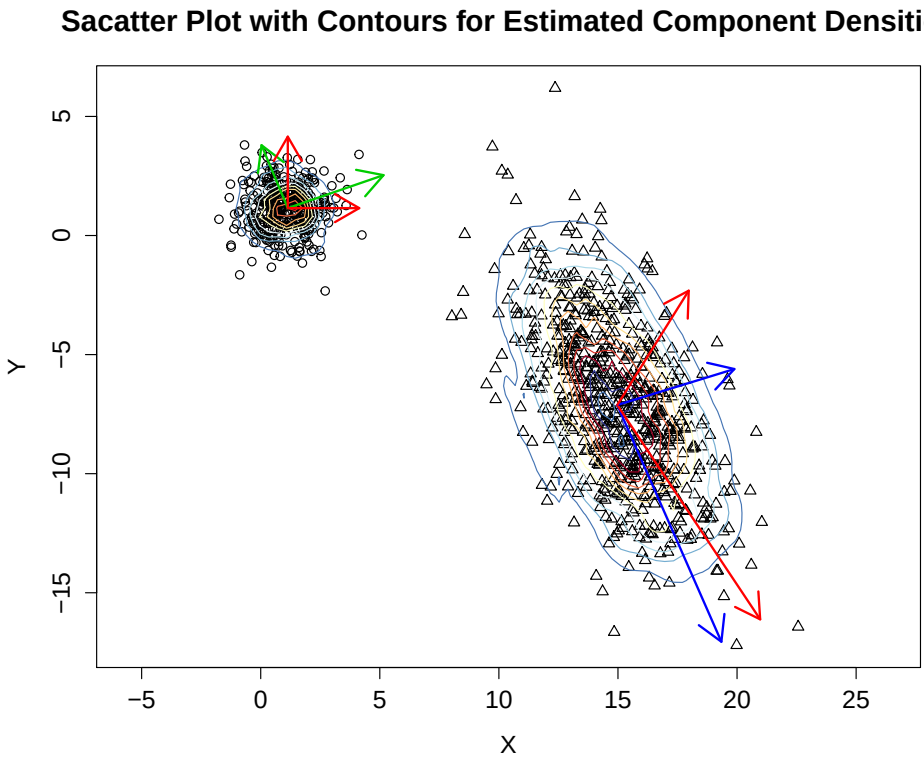


Figure 4.5. Simulation 2: Contours of estimated component densities. The red arrows visualize the columns of the true transformation matrices while the others indicate those of the estimated transformation matrices.

npEM perform more or less the same for the first component for which the conditional independence assumption holds, while NSMM-ICA gives significantly better estimates for the second component where the conditional independence assumption fails. For reproducing the results and graphs, the R scripts "simulation2.R" and

Table 4.4. Simulation 2: Squareroot of MISE

	Component 1	Component 2
`npEM`	0.04341876	0.03950780
`NSMM-ICA`	0.04965566	0.01732556

"simulation2mise.R" are included in Appendices and , respectively.

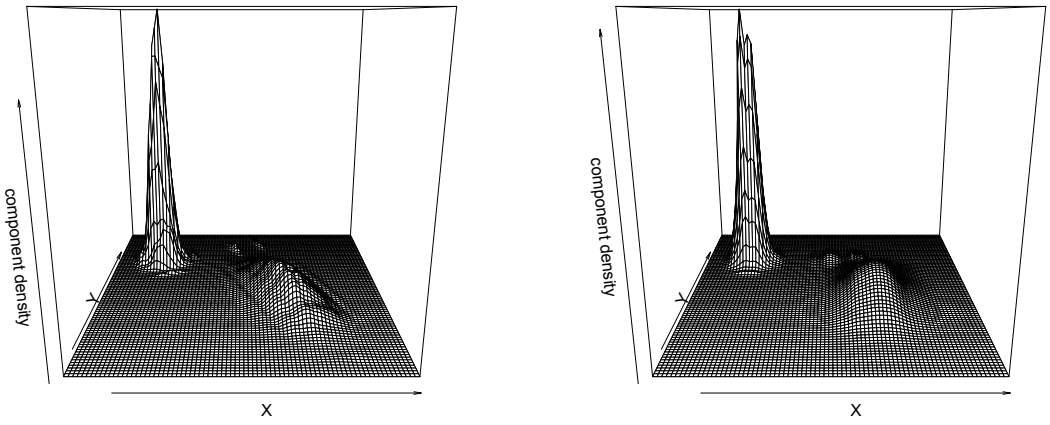


Figure 4.6. Simulation 2: 3D plots of estimated component densities by the NSMM-ICA (left) and npEM (right) algorithms.

.

4.3 Iris Flower Classification

We consider the well-known iris flower data set, which is a multivariate data set introduced by Fisher (1936) as an example of discriminant analysis. Edgar Anderson collected the data to quantify the morphologic variation of iris flowers of three related species: Iris setosa, Iris virginica and Iris versicolor. There are 50 samples from each of the three species. Four features were measured on each sample: the length and the width of the sepals and petals, in centimetres. One class is easily separated from the others, but the other two are relatively close to each other.

This data set has been popular as a benchmark test case for supervised learning, based on the linear discriminant model developed by Fisher (1936). Here, we view the data as unlabeled and apply our NSMM-ICA algorithm to it for unsupervised

learning. Figure 4.7 shows a comparison of true species information and results from NSMM-ICA as well as K-Means algorithms. We find that the NSMM-ICA

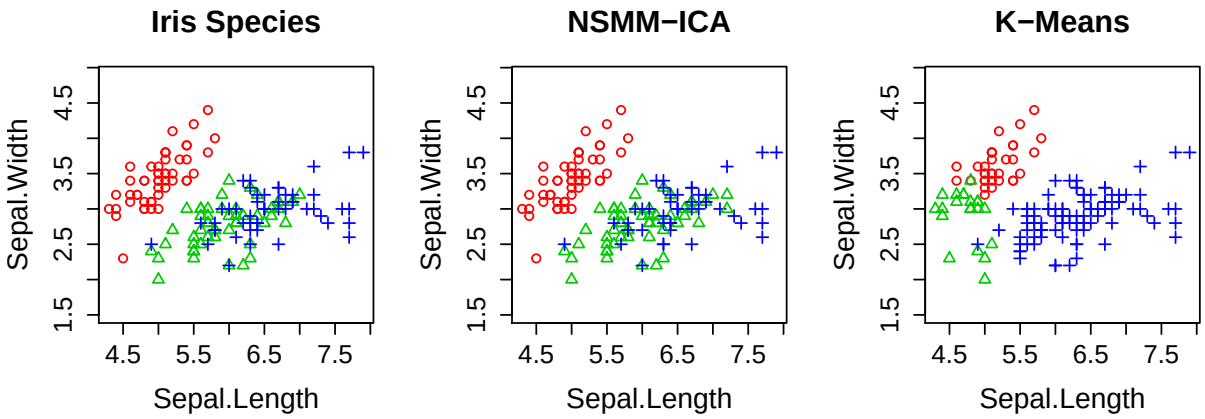


Figure 4.7. Iris Data: Comparison of true species information (far left) and two results from NSMM-ICA as well as K-Means algorithms.

achieves a quite stable classification error of 4.67%, while the results from K-Means clustering are unstable and often inaccurate with an error rate greater than 30%. For reproducing the results and graphs, an R script is included in Appendix .

4.4 Italian Wine Classification

The Italian wine data set is another popular data set used for comparing various classifiers. (Forina et al., 1988; Aeberhard et al., 1992). It contains results of a chemical analysis of wines grown in Italy but derived from three different cultivars. A total of 178 observations are recorded, each with 13 continuous attributes such as color intensity, magnesium and malic acid . There are 59, 71 and 48 instances in the first (Barolo), second (Grignolino) and third (Barbera) wine classes respectively.

If we feed the unlabeled data directly to the NSMM-ICA algorithm, we find a less-than-ideal classification result, with a classification error rate equal to 28.65%,

Table 4.5. Wine Data: PCA+NSMM-ICA Classification

	Class 1	Class 2	Class 3
Barolo	59	0	0
Grignolino	6	61	4
Barbera	0	0	48

prompting us to consider remedies. it seems that given the small number of observations, the relatively large number of attributes may be somewhat challenging as there are too many parameters to estimate and some of the attributes may consist of noise. So instead of using all 13 attributes, we first run principal component analysis (PCA) on the attributes and then select the 5 PCA scores that explain the largest proportion of variance in the attributes. Finally, we run the NSMM-ICA algorithm on the data set with the chosen PCA scores as attributes. In this way, the classification performance improves quite a lot, with a classification error rate equal to 5.62%. Hence, in situations with relatively large numbers of coordinates, it might be worthwhile to utilize a feature extraction technique folllowed by the NSMM-ICA algorithm. Figure 4.8 shows a comparison of true species information and results from our unsupervised learning algorithms. For reproducing the results and graphs, an R script is included in Appendix .

4.5 Water Level Data

The water level data set contains measurements from psychometric experiment for assessing children's understanding of horizontality. It contains measurements on 405 children, both boys and girls, aged 11 to 16 years, each of whom is shown a bottle in eight different oblique directions: 11, 4, 2, 7, 10, 5, 1 and 8 o'clock. They are asked to use a line to indicate the position of the water level (surface) were the

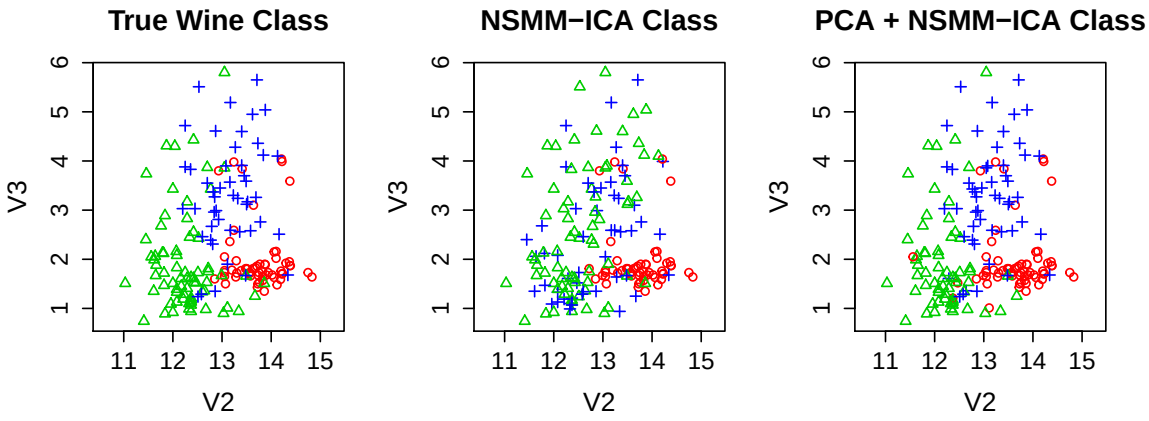


Figure 4.8. Wine Data: Comparison of true species Information (far left) and two results from our unsupervised learning algorithms.

bottle filled with water. The signed acute angle between the horizontal and the line drawn, in degrees, is measured and reported in the data set.

For comparing our NSMM-ICA algorithm and the npEM algorithm from Benaglia, Chauveau, & Hunter (2009), since NSMM-ICA doesn't implement block structure among coordinates, we will group the coordinates manually. We restructure the data set so that the angle values for 11 o'clock are stored with those of 5 o'clock in one column, and same with those of 4 and 10 o'clock, 2 and 8 o'clock, and 7 and 1 o'clock. In this way, some information of the data set is lost. However, more information is available for estimating fewer component marginal densities.

Table 4.6 shows the estimated mixing weights from running the two algorithms. And the classifications by the two approaches differ by 20.62%. Figures 4.9 and

Table 4.6. Water Data: Estimated Mixing Weights

	Group 1	Group 2	Group 3
`npEM`	0.47328230	0.44912883	0.07758887
`NSMM-ICA`	0.49120059	0.42464337	0.08415604

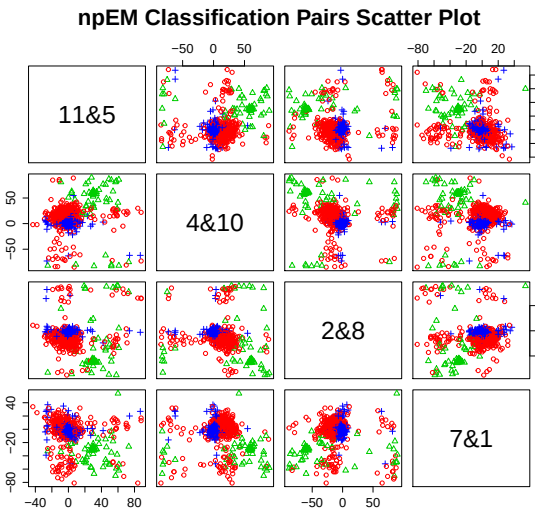


Figure 4.9. Waterlevel npEM Results. Colors indicate estimated class membership.

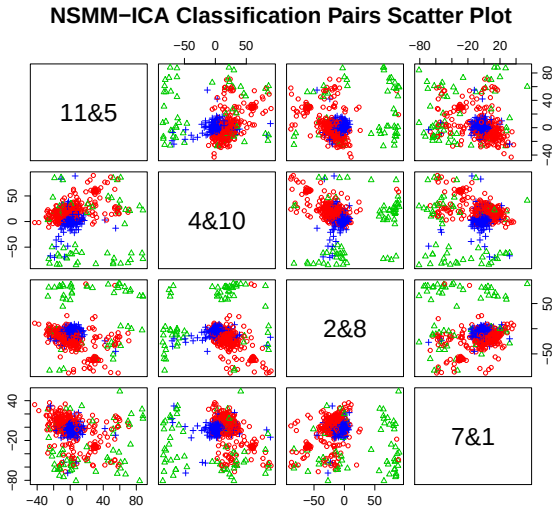


Figure 4.10. Waterlevel NSMM-ICA Results. Colors indicate estimated class membership.

4.10 compare the results by plotting the data with color according to the recovered membership information. It shows that the two results confirm each other on the findings about group 1 and group 2, the two groups with largest probability.

Comparing of the two results also helps study the effect and suitability of the conditional independence assumption. For example, let us take a look at the estimated mixing matrices by the NSMM-ICA algorithm. Here `d` is the result from running the command `d <- EMFASTICAALG(WaterModified, 3)`.

```
> solve(d$WMtrs[[1]])
           [,1]       [,2]        [,3]      [,4]
[1,]  -2.058268 -1.6555927 15.47156897 -11.56747
[2,]  -5.009812  2.2581155 -1.05649965 -16.89123
[3,] -12.919979 -0.2857619 -0.02650691  16.18224
[4,]   2.556894 15.5903755  0.72012604  12.20257

> solve(d$WMtrs[[2]])
           [,1]       [,2]        [,3]       [,4]
[1,]  -3.000654  -6.097546 -34.3078689  -3.244923
```

```
[2,] -44.150579  32.428180  -2.0126229   4.171715
[3,] -46.603486 -29.293682   0.7760072  -1.848921
[4,]   1.422766  -3.664026  -2.6615647 -33.895998

> solve(d$WMtrs[[3]])
           [,1]       [,2]       [,3]      [,4]
[1,] -0.5129936  5.0383414  1.3601686  2.982748
[2,] -3.3701586 -0.4704390 -7.6871990  8.335422
[3,] -2.4443036 -3.4093234  3.5079041  3.138356
[4,] -6.5935724 -0.4596964  0.5432347 -1.837893
```

The three estimated mixing matrices exhibit interesting patterns which could be related to evidence against the conditional independence assumption.

A different view is obtained when looking at Figures 4.11 and 4.12, which compare the component marginal density esitmates obtained from npEM and NSMM-ICA, respectively. As can be seen, they are close, which may be viewed as partial evidence that justifies the conditional independence assumption. Hence this raises the question of whether it's necessary to bring into the ICA structure as well as assessing the goodness of fit of both models, in particular, testing the conditional independence assumption of the npEM algorithm. For reproducing the results and graphs, an R script is included in Appendix .

4.6 Reaction Time Data

The reaction time data set contains measurements on 197 children who give correct responses to 6 task trials in a psychometric experiment. Each child sits in front of a monitor. The target stimulus on the left shows drawings of a simple shape, while the same image or its mirror image is shown on the right stimulus. The child is supposed to press one of two keys to indicate which case they observe, that is, identical image or mirror image. There are in total 6 tasks on which each child's

Figure 4.11. Waterlevel npEM component marginal density estimates

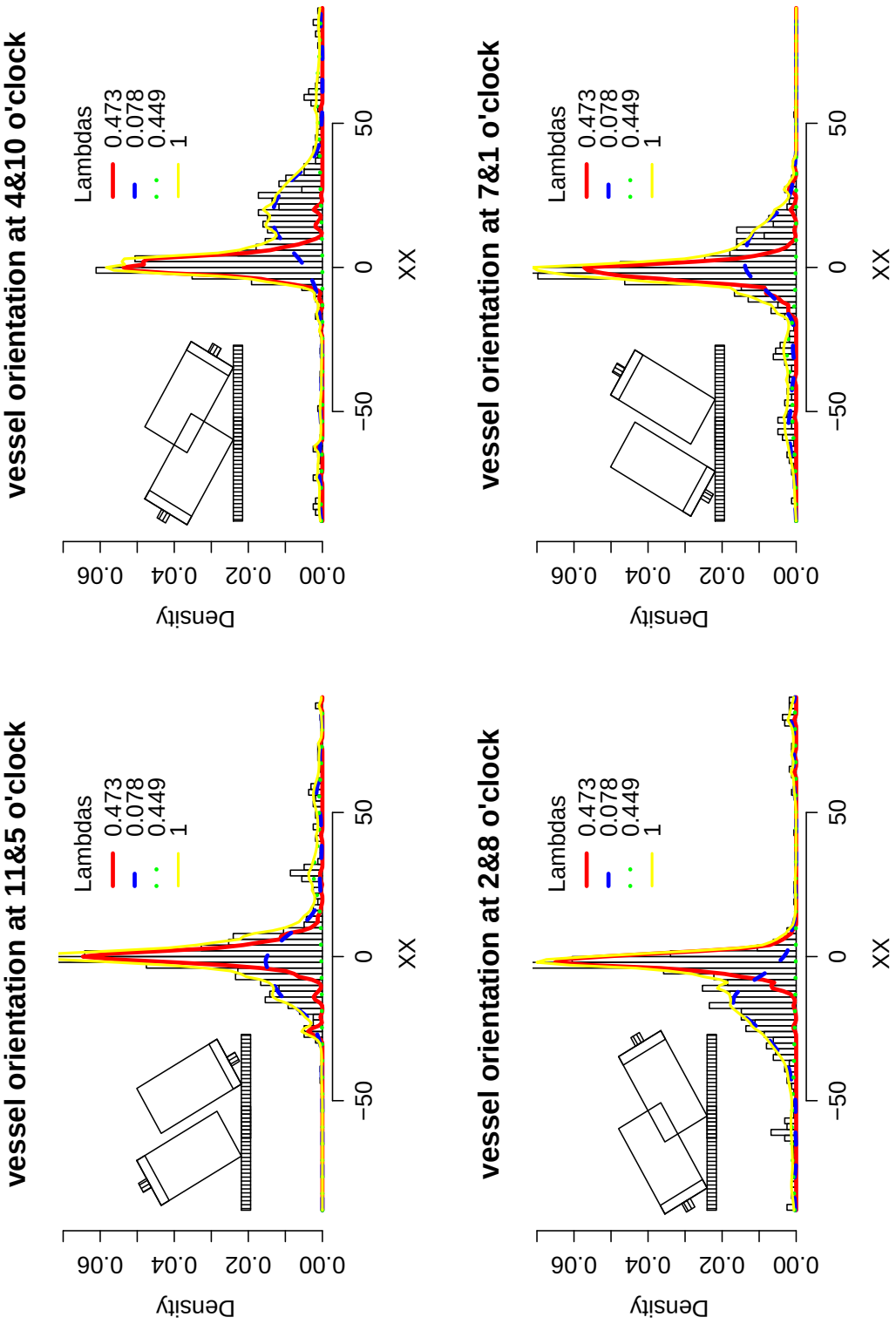
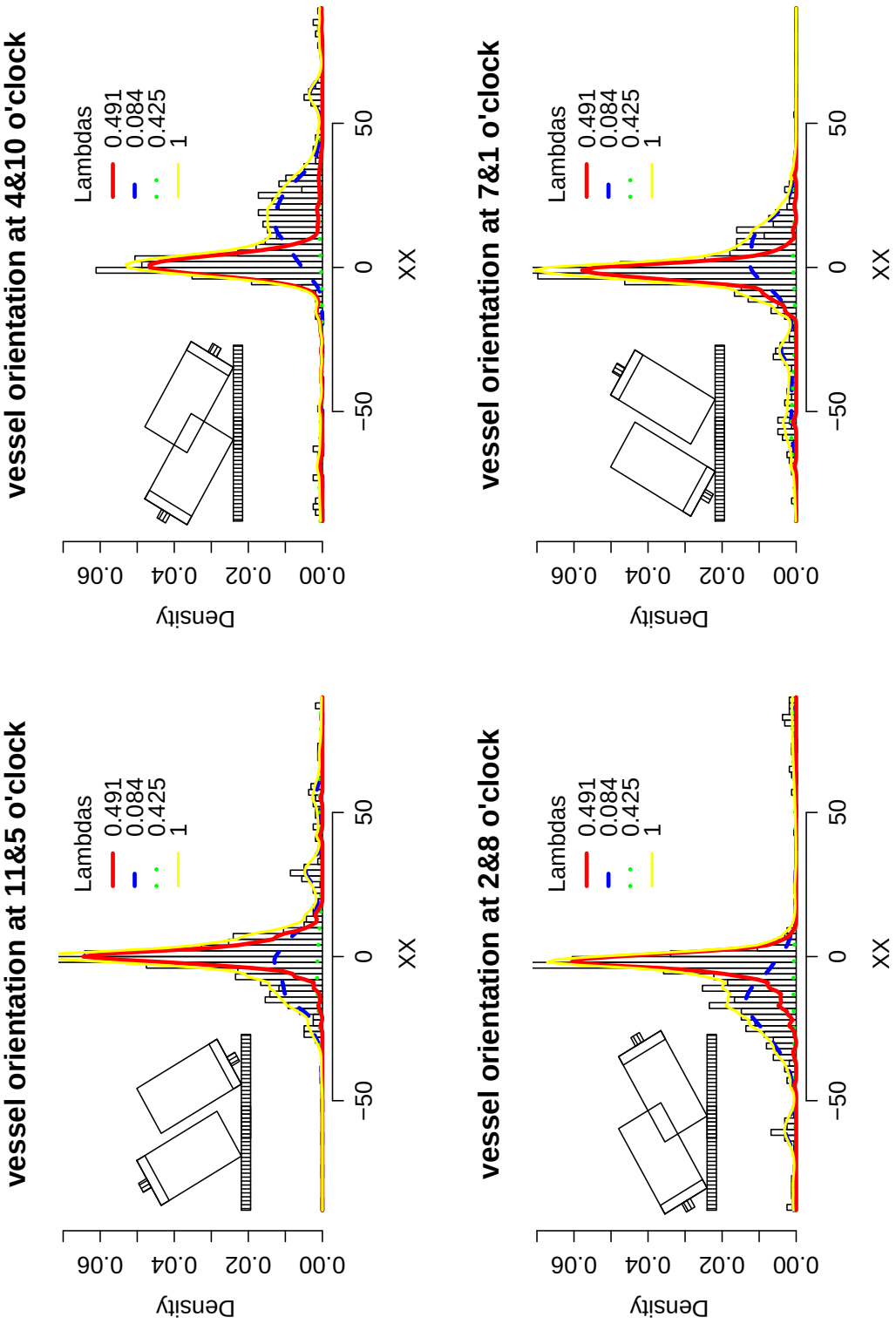


Figure 4.12. Waterlevel NSMM-ICA component marginal density estimates



reaction times, in milliseconds, are recorded. So the data set is 197 by 6.

This type of study is common in the area of developmental psychology. One question of interest is whether the whole population of children can be decomposed into subgroups, each having a different reaction time distribution. For analysis of the data set, Cruz-Medina & Hettmansperger (2004) proposes a semi-parametric mixture model with a multinomial cut point approach which extends the work by Hettmansperger & Thomas (2000). Assuming conditional independence and identical distribution of repeated measures, this study recommends two components and estimates the mixing weights to be 0.59 and 0.41. These findings agree with other conceptually related tasks for which evidence of two primary subgroups has been reported. However, characterization and estimation of the two primary groups remains an open issue, as the authors point out.

Here we run both the npEM algorithm by Benaglia, Chauveau, & Hunter (2009) and our NSMM-ICA algorithm on the reaction time data. The npEM leads to the estimated mixing weights 0.4009 and 0.5991, which roughly confirms with the result in Cruz-Medina & Hettmansperger (2004), since both estimations assume conditional independence. However, the NSMM-ICA, which uses ICA to model the dependence structure rather than assuming conditional independence, leads to the estimated mixing weights 0.1929 and 0.8071. The two results are quite different, though all of these models assume two subgroups.

Figures 4.13 and 4.14 compare the results by plotting the data with color according to the recovered membership information. It shows the estimated mixing weights differ more in the results by NSMM-ICA algorithm. For reproducing the results and graphs, an R script is included in Appendix .

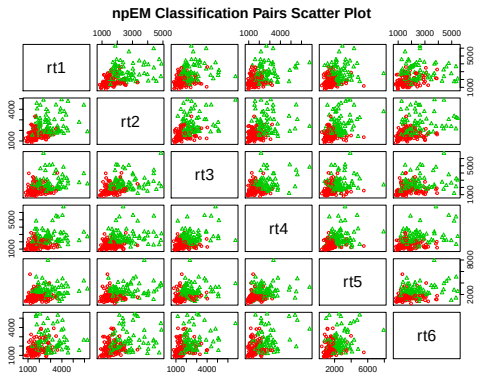


Figure 4.13. RTdata pairs scatter plot with colors indicating class membership estimated by npEM algorithm.

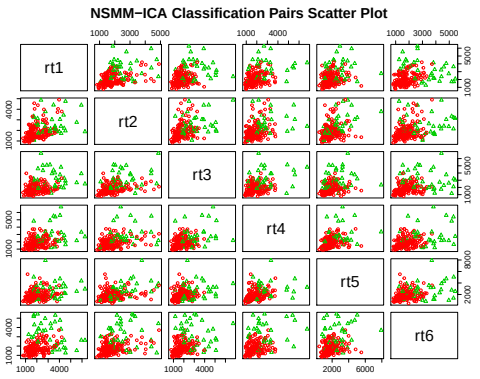


Figure 4.14. RTdata pairs scatter plot with colors indicating class membership estimated by NSMM-ICA algorithm.

4.7 Tone Data

The tone perception experiment and data were first introduced by Cohen (1984). These data have been analyzed by De Veaux (1989), Viele & Tong (2002)and (Hunter & Young, 2012) in the context of mixture of regressions. In each trial of the experiment, a musician is presented with a fundamental tone plus a series of overtones which is determined by a stretching ratio. Then the musician is asked to tune an adjustable tone to one octave above the fundamental tone. Both the stretching ratio and the ratio of the adjusted tone to the fundamental are reported for each trial. There are five musicians involved in the experiment. However, the tone data set only contains 150 trials with the same musician. The problem of interest in conducting this experiment is to investigate the theory that the musician would either tune the tones to the nominal octave at a ratio of 2:1 to the fundamental tone (i.e., the interval memory hypothesis) or use the overtone to tune the tone to the stretching ratio (i.e., the partial matching hypothesis). The findings by Hunter & Young (2012) via modelling through a semi-parametric mixture of

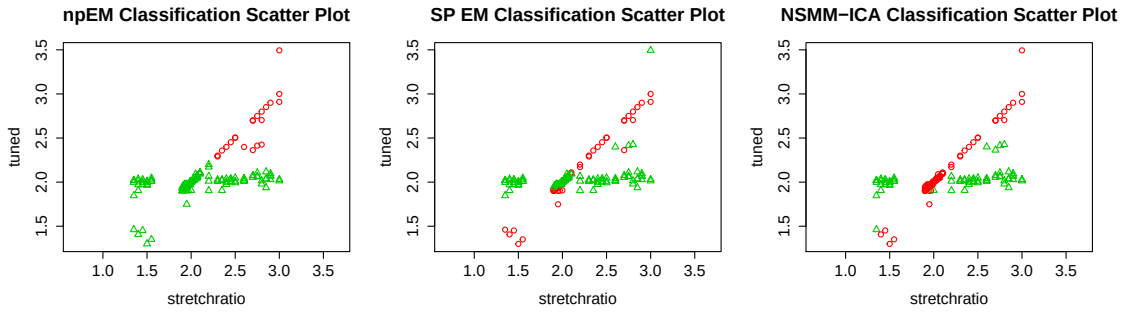


Figure 4.15. Tonedata: Comparison of Unsupervised Learning Results

regressions conforms with this theory.

For this unsupervised learning task, we run both the npEM algorithm by Benaglia, Chauveau, & Hunter (2009) and our NSMM-ICA algorithm on the tone data. Figure 4.15 shows that our NSMM-ICA algorithm does a good job of classification, very close to that obtained by by Hunter & Young (2012), without any regression structure. The reason why results of npEM algorithm shown in Figure 4.15 do not look nearly as good is because with regression lines that have nonzero slopes the mixture is far from being conditionally independent.

For comparing our result with that of Hunter & Young (2012), we can use the estimated mixing weights obtained from NSMM-ICA to fit a weighted ordinary least squares model to obtain the regression coefficients. The results, summarized in Table 4.7, reflects the difference in estimated membership between SP EM and our NSMM-ICA lies in the intersection of the two components, which is responsible for the noticeable difference in the estimated mixing weights. For reproducing the results and graphs, an R script is included in Appendix .

Table 4.7. Comparision of mixtures of least squares fits for the tone dataset of Cohen (1984).

		SP EM	NSMM-ICA/Weighted LS
component 1	$\hat{\beta}_0$	1.77533	1.82215
	$\hat{\beta}_1$	0.11954	0.09076
component 2	$\hat{\beta}_0$	0.02121	-0.12111
	$\hat{\beta}_1$	0.97929	1.05584
$\hat{\lambda}_1$		0.67653	0.46779

4.8 Learning Efficient Codes for Images

Learning efficient codes for images obtained from different sources or contexts is a problem of great interest and importance in the area of image processing. The task involves extracting intrinsic structure in images by clustering and finnding a complete set of efficient linear basis functions for each image sources, which results in coefficient values being as statistically independent as possible. Techniques that utilize a parametric form of ICA mixture models have been proposed in Lee et al. (1999), Sejnowski (1999) and Lee et al. (2000).

Here we propose using the non-parametric ICA mixture model (3.2, 3.4, 3.5) and applying the NSMM-ICA algorithm to the task of unsupervised learning of image codes. The reason is that the non-parametric modelling provides a higher degree of flexibility.

The two images shown in Figure 4.16 will be used as sources for the data set of the application. One is a painting image (2508× 1808 pixels) and the other is a newspaper image (2057×1365 pixels). The images are already transformed to grey scale. And each 1×1 pixel consists of a pixel intensity value ranging from 0 (black) to 1 (white). We select 5000 12×12 pixel patches randomly from each

Figure 4.16. Two images as sources for data: Newspaper and painting.

image. So the complete data set is of dimension 10000×144. The NSMM-ICA algorithm converges after 19 iterations, which lasts a little less than 8 hours. Again each bandwidth is automatically learned by the iterative scheme we implemented. The result shows a very good recovery of the class-membership information, with a classification error rate of 1.2%.

The learned basis functions (i.e., a basis for the linear space of the pixel patches) show interesting patterns. Figure 4.17 shows the basis functions for each image. The ones for the painting image appear smoother but more irregular, while the ones for the newspaper image look spottier and more regular. For reproducing the results and graphs, an R script is included in Appendix .

4.9 Classification of Multi-spectral Imagery

In this section, we test the NSMM-ICA algorithm for a task of classification of multi-spectral imagery. A multi-spectral image consists of information on pixel intensity at specific frequencies across the electromagnetic spectrum. These frequency bands are discrete and somewhat narrow when compared to those of hyper-spectral imagery which are over a continuous spectral range. In remote sensing via orbital platforms, aerial sensor technologies are used to acquire and collect large-scale multi-spectral

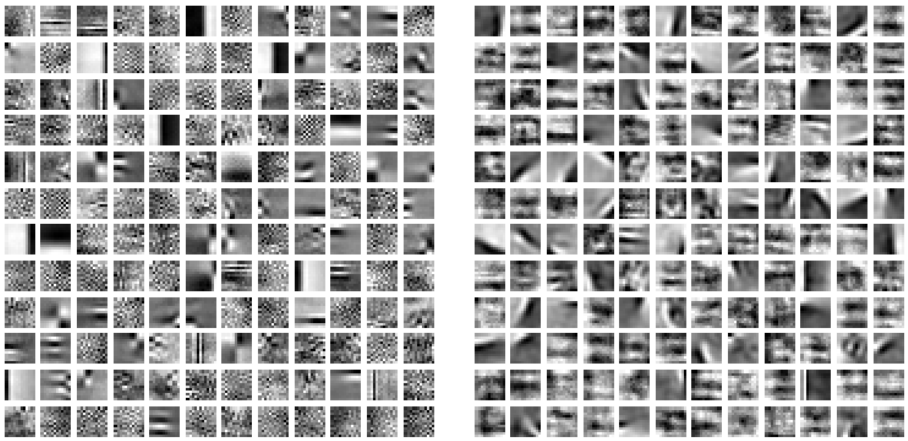


Figure 4.17. Learned basis functions for the newspaper (left) and painting (right). Each basis function, which is a 144 dimentional vector, is standardized to be within 0 and 1, and then printed out as a 12×12 image patch.

images of earth's surface. These data are useful in research on natural phenomena, natural resource management, land usage and conservation, and national security. One particular use of remote sensing multi-spectral imagery is in deriving effective land cover classification information. We have downloaded a data set for such an application (Statlog Landsat Satellite Data Set by Ashwin Srinivasan) from the UCI machine learning repository (http://archive.ics.uci.edu/ml/). The database consists of the multi-spectral values of pixels in 3x3 neighbourhoods (6435 in total) in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values. For each pixel, multi-spectral values in four different spectral bands are provided. Two of these are in the visible region (corresponding approximately to green and red regions of the visible spectrum) and two are in the (near) infra-red. In each line of data, the four spectral values for the top-left pixel are given first, followed by the four spectral values for the top-middle pixel and then those for the top-right pixel, and so on with the pixels read out in sequence from left-to-right

and top-to-bottom. Thus, the four spectral values for the central pixel are given by attributes 17,18,19 and 20. According to the author, we use only these four attributes, while ignoring the others. This avoids the problem which arises when a 3x3 neighbourhood straddles a boundary. So, the essential data we use is of dimension 6435 by 4. There are 6 decision classes: red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, very damp grey soil. For simplicity, we have combined cotton crop and soil with vegetation stubble into one category, and the rest into a second category. The sensibility of this combining step is also confirmed by the result of running the NSMM-ICA algorithm, as shown in Table 4.8. We have acheived an error classification rate of 6.4%. Figures 4.18 and 4.19 compare the true class membership with the estimated one. As can be seen, they are very close. For reproducing the results and graphs, an R script is included in Appendix .

Table 4.8. True and estimated mixing weights for the landsat dataset.

	Group 1	Group 2
true class	0.2191142	0.7808858
`NSMM-ICA`	0.2486402	0.7513598

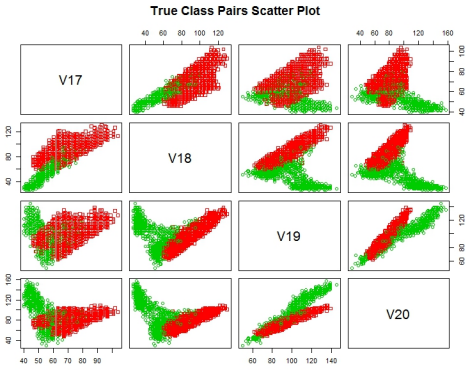


Figure 4.18. Landsat npEM results. Colors indicate true class membership.

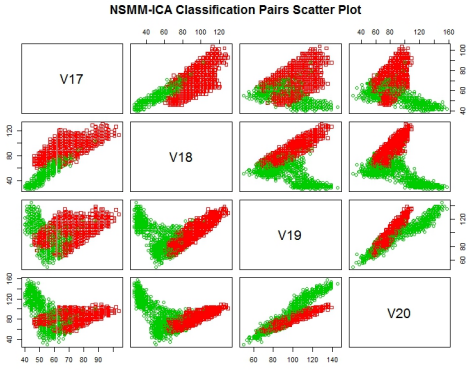


Figure 4.19. Landsat NSMM-ICA results. Colors indicate estimated class membership.

Chapter 5
Conclusions

5.1 Contributions of the Thesis

This thesis set out to investigate and build upon the foundation for the non-parametric estimation of finite multivariate mixture models given the conditional independence assumption, set forth in the work of Benaglia, Chauveau, & Hunter (2009) and Levine et al. (2011). Targeting the shortcomings of existing theories and algorithms, we have proposed an equivalent but crucially simplified view of the model that leads to a novel, and the first mathematically coherent, version of the penalized Kullback-Leibler divergence as the main optimization problem for the estimation. Under this new framework, certain key constraints that were previously put on the parameter space have now followed naturally from a special type of optimization scheme native to the problem setting. These contributions help to rigorously justify the non-parametric maximum smoothed likelihood (npMSL) algorithm that has been established by Levine et al. (2011) for the estimation. As part of the investigation, we have discovered a sharper monotonicity property of the NSMM algorithm, which enables us to prove for the first time the existence of

at least one solution for the estimation problem of this model. In addition, we have done some preliminary investigation of properties of the convergence of NSMM and that of any solution to the main optimization problem as well. Because of the elegant simplicity and mathematical tractability that come along with this framework, we believe it will serve as the basis for future research on the non-parametric estimation of finite multivariate mixture models given the conditional independence assumption.

Furthermore, this framework has shown its merit in our study of the extension of the model that loosens the conditional independence assumption by assuming instead that each component can be linearly transformed to having independent coordinates. One favorable feature of this extended model is that it allows for linear feature extraction techniques, as illustrated by one of the applications in Chapter 4. An optimization scheme similar in spirit to that under the original model helps develop the theoretical justification for the NSMM-ICA algorithm, which comines the smoothing majorization-minimization method with independent component analysis (ICA) techniques for the estimation of the extended model. The R (R Core Team, 2014) package **icamix**, available on CRAN, has been written for the practical implementation of the NSMM-ICA algorithm, which currently interweaves a weighted version of the FastICA algorithms (Hyvarinen et al., 2002) with the npEM algorithm (Benaglia, Chauveau, & Hunter, 2009). The core algorithms have all been written in C++ code. utilizing **Rcpp** (Eddelbuettel & François, 2011; Eddelbuettel, 2013) and RcppArmadillo (Eddelbuettel & Sanderson, 2014) for compiling and linking to the C++ code for better performance. The package also implements an automated and adaptive scheme for bandwidth selection that is based on the work of Benaglia et al. (2011). The proposed methodology has been tested with simulation studies as well as quite a few applications in the area of

unsupervised classification, with examples ranging from a cognitive study assessing children's understanding of spacial concepts to image analysis learning efficient codes. Overall consistent and desirable performances have been observed. Given its flexibility and hence wide applicability, we propose this extended model and algorithm as a new approach to model-based clustering. During this research, we continue to encounter problems, both theoretical and practical, that arise naturally. They often lead to directions in which this work may continue in the future. These will be discussed in the next section.

5.2 Future Work

5.2.1 Large Sample Theory and Convergence Rate

In Section 2.2 we have shown the existence of at least one solution to the main optimization problem (2.15) and the discretized optimization problem (2.17). However, besides a few empirical studies (Benaglia, Chauveau, & Hunter, 2009; Levine et al., 2011), there has not been much statistical theory discovered pertaining to the consistency, accuracy and precision of these solutions in terms of approximating the true target density $g(x)$ with large samples, as well as the convergence properties of the NSMM algorithm, and how these two aspects interact with each other. In this section we will compile a list of unanswered questions about these desired properties. To remind us of the solutions' dependence on the tuning parameter h, we denote them by $\mathbf{e}^S(h)$ and $\mathbf{e}^n(h)$, respectively.

a) How close is $\sum_{j=1}^{m} (\mathbf{e}^S(h))_j(x)$ to $g(x)$? And how does this depend on h? In particular, will the difference go to zero as h goes to zero?

b) Will $\mathbf{e}^n(h)$ converge to $\mathbf{e}^S(h)$ as n goes to infinity? And how fast?

c) Is there a way of combining a) and b), say, to let h and n vary in a simultaneous scheme which leads to convergence of $\sum_{j=1}^{m} (\mathbf{e}^n(h))_j(x)$ to $g(x)$?

d) In computing $\mathbf{e}^n(h)$, will the iterations eventually converge to the solution as the global optimizer? What is the convergence rate? Correspondingly, what is the convergence property of the thoeretical computation of $\mathbf{e}^S(h)$?

e) In case convergence to a global optimizer is not always guaranteed, can we construct counterexamples where the algorithm leads to a local optimizer or a saddle point?

The work of Silverman et al. (1990) and Eggermont & LaRiccia (1995) raises hope in approaching some of the above questions.

5.2.2 Generic Identifiability of Non-parametric ICA

In Chapter 3 we have proposed the penalized Kullback-Leibler divergence-based estimation for the non-parametric finite multivariate ICA mixture models and investigated its optimization landscape. And in Chapter 4 we went on to carry out this idea in the implementation of the NSMM-ICA algorithm and we observe good performance in both simulation studies and real world applications. However, nonthing has been established about the identifiability of the parameters of this model. That is, given $g^*(x)$ as in assumption (i), can we identify the density of each mixture component in $\mathbf{e}$ up to label switching? In case identifiability holds, it is understood that for each mixture component, the columns of the corresponding mixing matrix, or equivalently, the underlying independent signals, are only expected to be identified up to permutation and rescaling. Furthurmore, it is reasonable to only aim for generic identifiability, which is identifiability except on a set with Lebesgue measure zero.

The work of Allman et al. (2009) has proved generic identifiability for the non-parametric finite multivariate mixture models given that the dimension is greater then or equal to three. It is within a proper algebraic variety that the identifiability could fail. The authors make use of the theory of algebraic geometry and have shown the implications of this method in several somehow generalized but similar situations. It is quite possible that these ideas and techniques are worthwhile to consider when trying to prove the generic identifiability for our setting where the ICA structure is incorporated.

5.2.3 Implementing Fast Gaussian Transform

The NSMM algorithm, the NSMM-ICA algorithm in the **icamix** package we have developed, and even the earlier npEM algorithm (Benaglia, Chauveau, & Hunter, 2009) all rely on evaluating sums of univariate Gaussian kernels, which is known as the discrete Gauss transform in numerical computing. It is a key computational component in our implementation of the algorithm, because, through trial and error, we have identified this as the most time-consuming part because of the many loops involved in this mostly repeated step. In the statistical computing community, currently, the algorithms employed for this key task are often very straightforward from the definition of the discrete Gauss transform, and most often it takes $O(MN)$ time to evaluate the transform at M target points due to N source points. This means expensive computation for large-scale problems. In our setting, the scale grows quickly with the number of mixture components and the dimension of the data points. So it is definitely worthwhile to consider using advanced computational methods to reduce the cost of computing for this part. Currently, by exploiting certain symmetries, we have roughly decreased the computing time

by half, compared to the npEM and related algorithms implemented in **mixtools** (Benaglia, Chauveau, Hunter, & Young, 2009). But the order of the complexity is still the same $O(MN)$.

Raykar et al. (2005) propose a Fast Gauss Transform method which has computational complexity of $O(M + N)$. The method originates from the earlier work of Greengard & Strain (1991), though at that time the work did not receive much attention due to hardware. Raykar & Duraiswami (2007) have further proposed the Improved Fast Gauss Transform (IFGT) that still has complexity $O(M + N)$. It is an ϵ-exact approximation algorithm that reduces computating time at the cost of precision, which however can be arbitrary.

Appendix

R Scripts

(http://sites.stat.psu.edu/~xxz131/doc/)

simulation1.R

This R script is for carrying out simulation study 1 of which the results are summarized in Section 4.2.1.

```
############################
## NSMM-ICA Simulation 1
##
## Written by Xiaotian Zhu
############################

library(Rcpp)
library(RcppArmadillo)
library(icamix)

set.seed(123456)

##
## set sample sizes and dimension r
##
n1 <- 500
n2 <- 500
n0 <- 500
r <- 2

##
## generating mixture component 0:
## linear transform of joint distribution with t marginals
```

```
##
tdeg <- 7 # degree of freedom for the t's
TT <- matrix(rt(n0*r, tdeg), n0, r) # Draw a sample of size n0 and
    dimension 2
CC <- matrix(c(5,0,0,1.7), 2, 2, byrow = TRUE)
X0 <- TT %*% CC
X0[,1] <- X0[,1]+20
X0[,2] <- X0[,2]+55
displacement <- c(20, 55)/c(5, 1.7)

f0ind <- function(grid){# mixture component 0 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * dt((grid[j,i]-displacement[j]),
          tdeg)
    }
  }
  answer
}

##
## generate mixture component 1:
## linear transform of joint distribution with uniform marginals
##
S <- matrix(runif(n1*r, 1, 3), n1, r)
A <- matrix(c(6, 9, -12, 15), 2, 2, byrow = TRUE)
X1 <- S %*% A

f1ind <- function(grid){# mixture component 1 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * (grid[j,i] >= 1 & grid[j,i] <= 3) /
          2
    }
  }
  answer
}

##
## generate mixture component 2
## linear transform of joint distribution with Laplacian marginals
##
library(Runuran)
l <- 20
s <- 2
```

```
distr <- udlaplace(l, s) # Create distribution object for standard
    Laplace distribution
gen <- pinvd.new(distr) # Generate generator object; use method
   PINV (inversion)
L <- matrix(ur(gen,n2*r), n2, r) # Draw a sample of size 500 and
   dimension 2
B <- matrix(c(1, 0.6, 1.2, 3), 2, 2, byrow = TRUE)
X2 <- L %*% B

f2ind <- function(grid){# mixture component 2 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:2){
      answer[i] <- answer[i] * exp(-abs(grid[j,i]-l)/s) /2 /s
    }
  }
  answer
}

# make a list of the mixture component original signal densities
# and a list of the mixing matrices
fInd <- list(f0ind, f1ind, f2ind)
MixMtrs <- list(t(CC), t(A), t(B))

##
## combine the components 1&2 to form the mixture dataset
## and run the algorithm
##
X<-rbind(X0, X1, X2)
rst1 <- EMFASTICAALG(X, 3, tol=1e-8)

##
## scatter plot with true and estimated class infomation
##
trueclass <- factor(c(rep(1,n0),rep(2,n1),rep(3,n2))) # true class
    info

estimatedclass <- ESTIMATEDMEMBER(rst1) # estimated class info
levels(estimatedclass) <- c(3,2,1) # numbering the estimated
   species, very important!

par(mfrow = c(1, 2))
plot(X, asp = 1, xlab='X',ylab='Y',col = as.numeric(levels(
   trueclass))[trueclass]+2, pch = as.numeric(levels(trueclass))[
   trueclass], main = "True Class Scatter Plot")
plot(X, asp = 1, xlab='X',ylab='Y',col = as.numeric(levels(
   estimatedclass))[estimatedclass]+2, pch = as.numeric(levels(
   estimatedclass))[estimatedclass], main = "NSMM-ICA Class
   Scatter Plot")
```

```
par(mfrow = c(1, 1))

##
## plot recovered independent signals
##
par(mfrow = c(1,3))
for(clsind in 1:3){
  keyind <- order(levels(estimatedclass))[clsind]
  plot(t(rst1$OriginalSignals[[keyind]])[estimatedclass==clsind,],
      asp = 1, col=clsind+2,xlab='X',ylab='Y',)
}
par(mfrow=c(1,1)) # reset par

##
## compare true and estimated unmixing matrix
##
rst1$WMtrs[[1]]
solve(MixMtrs[[as.numeric(levels(estimatedclass))[1]]])
rst1$WMtrs[[1]]/solve(MixMtrs[[as.numeric(levels(estimatedclass))
   [1]]])[2:1,]

rst1$WMtrs[[2]]
solve(MixMtrs[[as.numeric(levels(estimatedclass))[2]]])
rst1$WMtrs[[2]]/solve(MixMtrs[[as.numeric(levels(estimatedclass))
   [2]]])[2:1,]

rst1$WMtrs[[3]]
solve(MixMtrs[[as.numeric(levels(estimatedclass))[3]]])
rst1$WMtrs[[3]]/solve(MixMtrs[[as.numeric(levels(estimatedclass))
   [3]]])[2:1,]

##
## classification difference rate
##
CLASSDIFFRATE(trueclass,estimatedclass)

##
## plot data loglikelihood
##
plot.ts(rst1$ObjValue[-1,],xlab="iteration",ylab="data
   loglikelihood")

##
## investigate estimated component joint densities
##

# set range and grid points
Xrange <- 1.2*(range(X[,1])-mean(X[,1]))+mean(X[,1])
Yrange <- 1.2*(range(X[,2])-mean(X[,2]))+mean(X[,2])
ngrid <- 80
```

```
gridpoints <- t(as.matrix(expand.grid(seq(Xrange[1],Xrange[2],
   length.out=ngrid <- 80), seq(Yrange[1],Yrange[2],length.out=
   ngrid <- 80))))

# mixture components density evaluation
fvalues <- list()
for(clsind in 1:3){
  fvalues[[clsind]] <- ATRANSDENSITY(gridpoints, MixMtrs[[clsind
     ]], fInd[[clsind]])# we can check this by, say, "plot(t(
     gridpoints), col=(fvalues[[1]]!=0)+3)" and "sum(fvalues[[1]])
     *(Xrange[2]-Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid
     -1)"
}

# estimated mixture components original signal density evaluation
festInd <- list()
for(clsind in 1:3){
  keyind <- order(levels(estimatedclass))[clsind]
  festInd[[clsind]] <- function(grid){
    n<-ncol(grid)
    answer <- rep(1,n)
    for(i in 1:n){
      for(j in 1:r){
        answer[i] <- answer[i] * WtsKde(rst1$OriginalSignals[[
           keyind]][j,],rst1$MembershipProbs[,keyind],grid[j,i],
           rst1$ICABandWidth[keyind])$weightedkde
      }
    }
    answer <- answer/(sum(rst1$MembershipProbs[,keyind]))^r
    answer
  }
}

# estimated mixture components density evaluation
festvalues <- list()
for(clsind in 1:3){
  keyind <- order(levels(estimatedclass))[clsind]
  festvalues[[clsind]] <- ATRANSDENSITY(gridpoints-colSums(X*
     rst1$MembershipProbs[,keyind]/sum(rst1$MembershipProbs[,
     keyind])), solve(rst1$WMtrs[[keyind]]), festInd[[keyind]]) #
     we can check this by "plot(t(gridpoints), col=(festvalues
     [[1]]>0.0001)+3)" and "sum(festvalues[[1]])*(Xrange[2]-Xrange
     [1])/79*(Yrange[2]-Yrange[1])/79"
}

# integrated squared error
ise <- rep(0,3)
for(i in 1:3){
  ise[i] <- sum((fvalues[[i]]-festvalues[[i]])^2)*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
}
```

```
# plot estimated component densities and their contours
sumtrueden <- rep(0, ngrid*ngrid)
sumden <- rep(0, ngrid*ngrid)
sumwtsden <- rep(0, ngrid*ngrid)
for(clsind in 1:3){
  sumtrueden <- sumtrueden + fvalues[[clsind]]
  sumden <- sumden + festvalues[[clsind]]
  keyind <- order(levels(estimatedclass))[clsind]
  sumwtsden <- sumwtsden + festvalues[[clsind]]*rst1$Lambdas[nrow(
     rst1$Lambdas),keyind]
}
persp(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(sumtrueden,ngrid),xlab='X',
   ylab='Y',zlab='true component density')
persp(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(sumden,ngrid),xlab='X',ylab
   ='Y',zlab='component density')
persp(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(sumwtsden,ngrid),xlab='X',
   ylab='Y',zlab='weighted component density')

library(RColorBrewer)
library(MASS)

k <- 11
my.cols <- rev(brewer.pal(k, "RdYlBu")) # generate colors for
   contours
plot(X, asp = 1, xlab='X',ylab='Y', pch = as.numeric(levels(
   trueclass))[trueclass], main = "Sacatter Plot with Contours for
    Estimated Component Densities")
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(festvalues[[1]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(festvalues[[2]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(festvalues[[3]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)


for(clsind in 1:3){
  keyind <- order(levels(estimatedclass))[clsind]
  truecenter <- colSums(X*(trueclass==clsind)/sum(trueclass==
     clsind))
  center <- colSums(X*rst1$MembershipProbs[,keyind]/sum(
     rst1$MembershipProbs[,keyind]))
  estimatedA <- solve(rst1$WMtrs[[keyind]])
  trueA <- MixMtrs[[clsind]]
  for(j in 1:r){
```

```
    arrows(center[1],center[2],center[1]+3*estimatedA[1,j],center
       [2]+3*estimatedA[2,j],col=clsind+2,lwd=2)
    arrows(center[1],center[2],truecenter[1]+clsind^2*3*trueA[1,j
       ],truecenter[2]+clsind^2*3*trueA[2,j],col=2,lwd=2)

  }
}


##
## calculate integrated squared error by npEM
##
library(mixtools)
set.seed(100)
rst1npem <- npEM(X, mu0=3, samebw=FALSE)
#rst1npem_ <- EMFASTICAALG(X, 2, maxiter = 500, tol = 1e-8,
   combine = FALSE, seednum = 100)

postmember_rst1npem <- rst1npem$posteriors == apply (
   rst1npem$posteriors ,1, max) # estimated membership matrix
estimatedmember_rst1npem <- factor(rep((1:3) ,nrow(X))[as.vector(t
   (postmember_rst1npem))]) # estimated membership info
levels(estimatedmember_rst1npem) <- c(1,2,3)  # numbering the
   estimated species according to weights, very important!

margingrid <- cbind(seq(Xrange[1],Xrange[2],length.out=ngrid), seq
   (Yrange[1],Yrange[2],length.out=ngrid))

fnpemvalues <- list()
for(clsind in 1:3){
  keyind <- order(levels(estimatedmember_rst1npem))[clsind]
  fnpemvalues[[clsind]] <- 1
  for(j in 1:r){
    h <- rst1npem$bandwidth[j,keyind]
    fnpemvalues[[clsind]] <- apply(expand.grid(fnpemvalues[[clsind
       ]],WtsKde(X[,j], rst1npem$posteriors[,keyind], margingrid[,
       j], h)$weightedkde),1,prod)
    fnpemvalues[[clsind]] <- fnpemvalues[[clsind]]/sum(
       rst1npem$posteriors[,keyind])
  }
}

isenpem <- rep(0,3)
for(i in 1:3){
  isenpem[i] <- sum((fvalues[[i]]-fnpemvalues[[i]])^2)*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
}

sumdennpem <- rep(0, ngrid*ngrid)
sumwtsdennpem <- rep(0, ngrid*ngrid)
for(clsind in 1:3){
```

```
  sumdennpem <- sumdennpem + fnpemvalues[[clsind]]
  keyind <- order(levels(estimatedmember_rst1npem))[clsind]
  sumwtsdennpem <- sumwtsdennpem + fnpemvalues[[clsind]]*
     rst1npem$lambda[nrow(rst1npem$lambda),keyind]
}
persp(margingrid[,1], margingrid[,2], matrix(sumdennpem,ngrid),
   xlab='X',ylab='Y',zlab='component density')
persp(margingrid[,1], margingrid[,2], matrix(sumwtsdennpem,ngrid),
   xlab='X',ylab='Y',zlab='weighted component density')


plot(X, asp = 1, xlab='X',ylab='Y', pch = as.numeric(levels(
   trueclass))[trueclass], main = "Sacatter Plot with Contours for
    npEM Densities")
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(fnpemvalues[[1]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(fnpemvalues[[2]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(fnpemvalues[[3]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
```

simulation1mise.R

This R script is for calculating the squareroot of MISE in simulation 1 of which the results are summarized in Section 4.2.1.

```
################################################
## NSMM-ICA Simulation 1 sqrt(MISE) Calculation
##
## Written by Xiaotian Zhu
################################################

library(Rcpp)
library(RcppArmadillo)
library(icamix)
library(mixtools)
library(Runuran)

set.seed(196)

##
## set sample sizes and dimension r
##
n1 <- 500
n2 <- 500
n0 <- 500
```

```
r <- 2

displacement <- c(20, 55)/c(5, 1.7)
m <- 500 # number of repetition

##
## component densities
##

f0ind <- function(grid){# mixture component 0 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * dt((grid[j,i]-displacement[j]),
          tdeg)
    }
  }
  answer
}


f1ind <- function(grid){# mixture component 1 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * (grid[j,i] >= 1 & grid[j,i] <= 3) /
          2
    }
  }
  answer
}

f2ind <- function(grid){# mixture component 2 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:2){
      answer[i] <- answer[i] * exp(-abs(grid[j,i]-l)/s) /2 /s
    }
  }
  answer
}

fInd <- list(f0ind,f1ind, f2ind)

##
```

```
## mixting matrices
##

A0 <- matrix(c(5,0,0,1.7), 2, 2, byrow = TRUE)
A1 <- matrix(c(6, 9, -12, 15), 2, 2, byrow = TRUE)
A2 <- matrix(c(1, 0.6, 1.2, 3), 2, 2, byrow = TRUE)

MixMtrs <- list(t(A0), t(A1), t(A2))

##
## True Class Info
##
trueclass <- factor(c(rep(1,n0),rep(2,n1),rep(3,n2))) # true class
    info

##
## set up array for holding ISE and Classification Error Rate
##
ISE <- matrix(0,nrow=m,ncol=3)
CER <- matrix(0,nrow=m,ncol=1)
ISEnpEM <- matrix(0,nrow=m,ncol=3)
CERnpEM <- matrix(0,nrow=m,ncol=1)

##
## entering loop for calculating sqrt(MISE)
##
for(i in 1:m){

##
## generating mixture component 0:
## linear transform of joint distribution with t marginals
##
tdeg <- 7 # degree of freedom for the t's
S0 <- matrix(rt(n0*r, tdeg), n0, r) # Draw a sample of size n0 and
    dimension 2
X0 <- S0 %*% A0
X0[,1] <- X0[,1]+20
X0[,2] <- X0[,2]+55

  ##
  ## generate mixture component 1:
  ## linear transform of joint distribution with uniform marginals
  ##
  S1 <- matrix(runif(n1*r, 1, 3), n1, r)
  X1 <- S1 %*% A1

  ##
  ## generate mixture component 2:
  ## linear transform of joint distribution with Laplacian
     marginals
  ##
```

```
l <- 20
s <- 2
distr <- udlaplace(l, s) # Create distribution object for
   standard Laplace distribution
gen <- pinvd.new(distr) # Generate generator object; use method
   PINV (inversion)
S2 <- matrix(ur(gen,n2*r), n2, r) # Draw a sample of size 500
   and dimension 2
X2 <- S2 %*% A2

##
## combine the components 1&2 to form the mixture dataset
## and run the algorithm
##
X<-rbind(X0, X1, X2)
rst1 <- EMFASTICAALG(X, 3)

##
## estimated class infomation
##
estimatedclass <- ESTIMATEDMEMBER(rst1) # estimated class info
firstcordmean <- rep(0,3)
for(i_ in 1:3){
  firstcordmean[i_] <- sum(X[,1]*rst1$MembershipProbs[,i_]/sum(
     rst1$MembershipProbs[,i_]))
}
levels(estimatedclass) <- c(2,1,3)[rank(firstcordmean)] #
   numbering the estimated species, very important!

##
## classification difference rate
##
CER[i,1] = CLASSDIFFRATE(trueclass,estimatedclass)

##
## ISE calculation
##

# set range and grid points
Xrange <- 1.2*(range(X[,1])-mean(X[,1]))+mean(X[,1])
Yrange <- 1.2*(range(X[,2])-mean(X[,2]))+mean(X[,2])
ngrid <- 80
gridpoints <- t(as.matrix(expand.grid(seq(Xrange[1],Xrange[2],
   length.out=ngrid), seq(Yrange[1],Yrange[2],length.out=ngrid))
   ))

# mixture components density evaluation
fvalues <- list()
for(clsind in 1:3){
  fvalues[[clsind]] <- ATRANSDENSITY(gridpoints, MixMtrs[[clsind
     ]], fInd[[clsind]])# we can check this by, say, "plot(t(
```

```
      gridpoints), col=(fvalues[[1]]!=0)+3)" and "sum(fvalues
      [[1]])*(Xrange[2]-Xrange[1])/79*(Yrange[2]-Yrange[1])/79"
}

# estimated mixture components original signal density
   evaluation
festInd <- list()
for(clsind in 1:3){
  keyind <- order(levels(estimatedclass))[clsind]
  festInd[[clsind]] <- function(grid){
    n<-ncol(grid)
    answer <- rep(1,n)
    for(i in 1:n){
      for(j in 1:r){
        answer[i] <- answer[i] * WtsKde(rst1$OriginalSignals[[
           keyind]][j,],rst1$MembershipProbs[,keyind],grid[j,i],
           rst1$ICABandWidth[keyind])$weightedkde
      }
    }
    answer <- answer/(sum(rst1$MembershipProbs[,keyind]))^r
    answer
  }
}

# estimated mixture components density evaluation
festvalues <- list()
for(clsind in 1:3){
  keyind <- order(levels(estimatedclass))[clsind]
  festvalues[[clsind]] <- ATRANSDENSITY(gridpoints-colSums(X*
     rst1$MembershipProbs[,keyind]/sum(rst1$MembershipProbs[,
     keyind])), solve(rst1$WMtrs[[keyind]]), festInd[[keyind]])
     # we can check this by "plot(t(gridpoints), col=(festvalues
     [[1]]>0.0001)+3)" and "sum(festvalues[[1]])*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)"
}

# integrated squared error
for(j in 1:3){
  ISE[i,j] <- sum((fvalues[[j]]-festvalues[[j]])^2)*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
}

##
## calculate integrated squared error by npEM
##
rst1npem <- npEM(X, mu0=3, samebw=FALSE)
#rst1npem_ <- EMFASTICAALG(X, 2, maxiter = 500, tol = 1e-8,
   combine = FALSE, seednum = 100)

postmember_rst1npem <- rst1npem$posteriors == apply (
   rst1npem$posteriors ,1, max) # estimated membership matrix
```

```
  estimatedmember_rst1npem <- factor (rep((1:3) ,nrow(X))[as.
     vector(t(postmember_rst1npem))]) # estimated membership info
  firstcordmean_ <- rep(0,3)
  for(i_ in 1:3){
    firstcordmean_[i_] <- sum(X[,1]*rst1npem$posteriors[,i_]/sum(
       rst1npem$posteriors[,i_]))
  }
  levels(estimatedmember_rst1npem) <- c(2,1,3)[rank(firstcordmean_
     )] # numbering the estimated species according to weights,
     very important!

  margingrid <- cbind(seq(Xrange[1],Xrange[2],length.out=ngrid),
     seq(Yrange[1],Yrange[2],length.out=ngrid))

  fnpemvalues <- list()
  for(clsind in 1:3){
    keyind <- order(levels(estimatedmember_rst1npem))[clsind]
    fnpemvalues[[clsind]] <- 1
    for(j in 1:r){
      h <- rst1npem$bandwidth[j,keyind]
      fnpemvalues[[clsind]] <- apply(expand.grid(fnpemvalues[[
         clsind]],WtsKde(X[,j], rst1npem$posteriors[,keyind],
         margingrid[,j], h)$weightedkde),1,prod)
      fnpemvalues[[clsind]] <- fnpemvalues[[clsind]]/sum(
         rst1npem$posteriors[,keyind])
    }
  }

  for(j in 1:3){
    ISEnpEM[i,j] <- sum((fvalues[[j]]-fnpemvalues[[j]])^2)*(Xrange
       [2]-Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
  }

  CERnpEM[i,1] = CLASSDIFFRATE(trueclass,estimatedmember_rst1npem)

}

##
## sqrt(MISE)
##
sqrt(colMeans(ISE))

sqrt(colMeans(ISEnpEM))

##
## classification error rates
##
summary(CER[,1])

summary(CERnpEM[,1])
```

```
boxplot(CER[,1],CERnpEM[,1][CERnpEM[,1]<0.4],names=c("NSMM-ICA","
    npEM"),main="Simulation 1: Classification error rates")
```

simulation2.R

This R script is for carrying out simulation 2 of which the results are summarized in Section 4.2.2.

```
############################
## NSMM-ICA Simulation 2
##
## Written by Xiaotian Zhu
############################

library(Rcpp)
library(RcppArmadillo)
library(icamix)

set.seed(16)

##
## set sample sizes and dimension r
##
n1 <- 360
n2 <- 640
r <- 2

##
## generate mixture component 1:
## linear transform of standard multivariate normal with mean
    (1,1)
##
S1 <- matrix(rnorm(n1*r, mean=1), n1, r)
A1 <- matrix(c(1, 0, 0, 1), 2, 2, byrow = TRUE)
X1 <- S1 %*% A1

f1ind <- function(grid){# mixture component 1 original signal
    density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * dnorm(grid[j,i]-1)
    }
  }
  answer
}

##
```

```
## generate mixture component 2:
## linear transform of standard multivariate normal with mean
   (5,5)
##
S2 <- matrix(rnorm(n2*r, mean=5), n2, r)
A2 <- matrix(c(2, -3, 1, 1.6), 2, 2, byrow = TRUE)
X2 <- S2 %*% A2

f2ind <- function(grid){# mixture component 1 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * dnorm(grid[j,i]-5)
    }
  }
  answer
}

# make a list of the mixture component original signal densities
# and a list of the mixing matrices
fInd <- list(f1ind, f2ind)
MixMtrs <- list(t(A1), t(A2))

##
## combine the components 1&2 to form the mixture dataset
## and run the NSMM-ICA algorithm and the npEM algorithm
##
X<-rbind(X1, X2)
rst2 <- EMFASTICAALG(X, 2)

##
## scatter plot with true and estimated class infomation
##
trueclass <- factor(c(rep(1,n1),rep(2,n2))) # true class info

estimatedclass <- ESTIMATEDMEMBER(rst2) # estimated class info
levels(estimatedclass) <- rank(rst2$Lambdas[nrow(rst2$Lambdas),])
   # numbering the estimated species, very important!

par(mfrow = c(1, 2))
plot(X, asp = 1, xlab='X',ylab='Y',col = as.numeric(levels(
   trueclass))[trueclass]+2, pch = as.numeric(levels(trueclass))[
   trueclass], main = "True Class Scatter Plot")
plot(X, asp = 1, xlab='X',ylab='Y',col = as.numeric(levels(
   estimatedclass))[estimatedclass]+2, pch = as.numeric(levels(
   estimatedclass))[estimatedclass], main = "NSMM-ICA Class
   Scatter Plot")

##
```

```
## plot recovered independent signals
##
par(mfrow = c(1,2))
for(clsind in 1:2){
  keyind <- order(levels(estimatedclass))[clsind]
  plot(t(rst2$OriginalSignals[[keyind]])[estimatedclass==clsind,],
      asp = 1, col=clsind+2,xlab='X',ylab='Y',)
}
par(mfrow=c(1,1)) # reset par

##
## compare true and estimated unmixing matrix
##
rst2$WMtrs[[1]]
solve(MixMtrs[[as.numeric(levels(estimatedclass))[1]]])
rst2$WMtrs[[1]]/solve(MixMtrs[[as.numeric(levels(estimatedclass))
   [1]]])[1:2,]

rst2$WMtrs[[2]]
solve(as.numeric(levels(estimatedclass))[2])
rst2$WMtrs[[2]]/solve(MixMtrs[[as.numeric(levels(estimatedclass))
   [2]]])[1:2,]

##
## classification error rate
##
CLASSDIFFRATE(trueclass,estimatedclass)

##
## plot data loglikelihood
##
plot.ts(rst2$ObjValue[-1,],xlab="iteration",ylab="data
   loglikelihood")

##
## investigate estimated component joint densities
##

# set range and grid points
Xrange <- 1.2*(range(X[,1])-mean(X[,1]))+mean(X[,1])
Yrange <- 1.2*(range(X[,2])-mean(X[,2]))+mean(X[,2])
ngrid <- 80
gridpoints <- t(as.matrix(expand.grid(seq(Xrange[1],Xrange[2],
   length.out=ngrid), seq(Yrange[1],Yrange[2],length.out=ngrid))))

# mixture components density evaluation
fvalues <- list()
for(clsind in 1:2){
  fvalues[[clsind]] <- ATRANSDENSITY(gridpoints, MixMtrs[[clsind
     ]], fInd[[clsind]])# we can check this by, say, "plot(t(
     gridpoints), col=(fvalues[[1]]!=0)+3)" and "sum(fvalues[[1]])
```

```
     *(Xrange[2]-Xrange[1])/79*(Yrange[2]-Yrange[1])/79"
}

# estimated mixture components original signal density evaluation
festInd <- list()
for(clsind in 1:2){
  keyind <- order(levels(estimatedclass))[clsind]
  festInd[[clsind]] <- function(grid){
    n<-ncol(grid)
    answer <- rep(1,n)
    for(i in 1:n){
      for(j in 1:r){
        answer[i] <- answer[i] * WtsKde(rst2$OriginalSignals[[
           keyind]][j,],rst2$MembershipProbs[,keyind],grid[j,i],
           rst2$ICABandWidth[keyind])$weightedkde
      }
    }
    answer <- answer/(sum(rst2$MembershipProbs[,keyind]))^r
    answer
  }
}

# estimated mixture components density evaluation
festvalues <- list()
for(clsind in 1:2){
  keyind <- order(levels(estimatedclass))[clsind]
  festvalues[[clsind]] <- ATRANSDENSITY(gridpoints-colSums(X*
     rst2$MembershipProbs[,keyind]/sum(rst2$MembershipProbs[,
     keyind])), solve(rst2$WMtrs[[keyind]]), festInd[[keyind]]) #
     we can check this by "plot(t(gridpoints), col=(festvalues
     [[1]]>0.0001)+3)" and "sum(festvalues[[1]])*(Xrange[2]-Xrange
     [1])/79*(Yrange[2]-Yrange[1])/79"
}

# integrated squared error
ise <- rep(0,2)
for(i in 1:2){
  ise[i] <- sum((fvalues[[i]]-festvalues[[i]])^2)*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
}

# plot estimated component densities and their contours
sumden <- rep(0, ngrid*ngrid)
sumwtsden <- rep(0, ngrid*ngrid)
for(clsind in 1:2){
  sumden <- sumden + festvalues[[clsind]]
  keyind <- order(levels(estimatedclass))[clsind]
  sumwtsden <- sumwtsden + festvalues[[clsind]]*rst2$Lambdas[nrow(
     rst2$Lambdas),keyind]
}
```

```
persp(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(sumden,ngrid),xlab='X',ylab
   ='Y',zlab='component density')
persp(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(sumwtsden,ngrid),xlab='X',
   ylab='Y',zlab='weighted component density')

library(RColorBrewer)
library(MASS)

k <- 11
my.cols <- rev(brewer.pal(k, "RdYlBu")) # generate colors for
   contours
plot(X, asp = 1, xlab='X',ylab='Y', pch = as.numeric(levels(
   trueclass))[trueclass], main = "Sacatter Plot with Contours for
    Estimated Component Densities")
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(festvalues[[1]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(festvalues[[2]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)

for(clsind in 1:2){
  keyind <- order(levels(estimatedclass))[clsind]
  truecenter <- colSums(X*(trueclass==clsind)/sum(trueclass==
     clsind))
  center <- colSums(X*rst2$MembershipProbs[,keyind]/sum(
     rst2$MembershipProbs[,keyind]))
  estimatedA <- solve(rst2$WMtrs[[keyind]])
  trueA <- MixMtrs[[clsind]]
  for(j in 1:r){
    arrows(center[1],center[2],center[1]+3*estimatedA[1,j],center
       [2]+3*estimatedA[2,j],col=clsind+2,lwd=2)
    arrows(center[1],center[2],truecenter[1]+3*trueA[1,j],
       truecenter[2]+3*trueA[2,j],col=2,lwd=2)

  }
}

##
## calculate integrated squared error by npEM
##
library(mixtools)
set.seed(100)
rst2npem <- npEM(X, mu0=2, samebw=FALSE)
#rst2npem_ <- EMFASTICAALG(X, 2, maxiter = 500, tol = 1e-8,
   combine = FALSE, seednum = 100)

postmember_rst2npem <- rst2npem$posteriors == apply (
   rst2npem$posteriors ,1, max) # estimated membership matrix
```

```
estimatedmember_rst2npem <- factor (rep((1:2) ,nrow(X))[as.vector(
   t(postmember_rst2npem))]) # estimated membership info
levels(estimatedmember_rst2npem) <- rank(rst2npem$lambda[nrow(
   rst2npem$lambda),]) # numbering the estimated species according
    to weights, very important!

margingrid <- cbind(seq(Xrange[1],Xrange[2],length.out=ngrid), seq
   (Yrange[1],Yrange[2],length.out=ngrid))

fnpemvalues <- list()
for(clsind in 1:2){
  keyind <- order(levels(estimatedmember_rst2npem))[clsind]
  fnpemvalues[[clsind]] <- 1
  for(j in 1:r){
    h <- rst2npem$bandwidth[j,keyind]
    fnpemvalues[[clsind]] <- apply(expand.grid(fnpemvalues[[clsind
       ]],WtsKde(X[,j], rst2npem$posteriors[,keyind], margingrid[,
       j], h)$weightedkde),1,prod)
    fnpemvalues[[clsind]] <- fnpemvalues[[clsind]]/sum(
       rst2npem$posteriors[,keyind])
  }
}

isenpem <- rep(0,2)
for(i in 1:2){
  isenpem[i] <- sum((fvalues[[i]]-fnpemvalues[[i]])^2)*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
}

sumdennpem <- rep(0, ngrid*ngrid)
sumwtsdennpem <- rep(0, ngrid*ngrid)
for(clsind in 1:2){
  sumdennpem <- sumdennpem + fnpemvalues[[clsind]]
  keyind <- order(levels(estimatedmember_rst2npem))[clsind]
  sumwtsdennpem <- sumwtsdennpem + fnpemvalues[[clsind]]*
     rst2npem$lambda[nrow(rst2npem$lambda),keyind]
}
persp(margingrid[,1], margingrid[,2], matrix(sumdennpem,ngrid),
   xlab='X',ylab='Y',zlab='component density')
persp(margingrid[,1], margingrid[,2], matrix(sumwtsdennpem,ngrid),
   xlab='X',ylab='Y',zlab='weighted component density')

plot(X, asp = 1, xlab='X',ylab='Y', pch = as.numeric(levels(
   trueclass))[trueclass], main = "Sacatter Plot with Contours for
    npEM Densities")
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(fnpemvalues[[1]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
contour(seq(Xrange[1],Xrange[2],length.out=ngrid), seq(Yrange[1],
   Yrange[2],length.out=ngrid), matrix(fnpemvalues[[2]],ngrid),
   drawlabels=FALSE, nlevels=k, col=my.cols, add=TRUE)
```

simulation2mise.R

This R script is for calculating the squareroot of MISE in simulation 2 of which the results are summarized in Section 4.2.2.

```
################################################
## NSMM-ICA Simulation 2 sqrt(MISE) Calculation
##
## Written by Xiaotian Zhu
################################################

library(Rcpp)
library(RcppArmadillo)
library(icamix)
library(mixtools)

set.seed(196)

##
## set sample sizes and dimension r
##
n1 <- 360 # sample size of component 1
n2 <- 640 # sample size of component 2
r <- 2 # dimension of data
m <- 500 # number of repetition

##
## component densities
##
f1ind <- function(grid){# mixture component 1 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * dnorm(grid[j,i]-1)
    }
  }
  answer
}

f2ind <- function(grid){# mixture component 1 original signal
   density function
  n<-ncol(grid)
  answer <- rep(1,n)
  for(i in 1:n){
    for(j in 1:r){
      answer[i] <- answer[i] * dnorm(grid[j,i]-5)
    }
  }
  answer
```

```
}

fInd <- list(f1ind, f2ind)

##
## mixting matrices
##
A1 <- matrix(c(1, 0, 0, 1), 2, 2, byrow = TRUE)
A2 <- matrix(c(2, -3, 1, 1.6), 2, 2, byrow = TRUE)

MixMtrs <- list(t(A1), t(A2))

##
## True Class Info
##
trueclass <- factor(c(rep(1,n1),rep(2,n2))) # true class info

##
## set up array for holding ISE and Classification Error Rate
##
ISE <- matrix(0,nrow=m,ncol=2)
CER <- matrix(0,nrow=m,ncol=1)
ISEnpEM <- matrix(0,nrow=m,ncol=2)
CERnpEM <- matrix(0,nrow=m,ncol=2)

##
## entering loop for calculating sqrt(MISE)
##
for(i in 1:m){
  ##
  ## generate mixture component 1:
  ## linear transform of standard multivariate normal with mean
     (1,1)
  ##
  S1 <- matrix(rnorm(n1*r, mean=1), n1, r)
  X1 <- S1 %*% A1

  ##
  ## generate mixture component 2:
  ## linear transform of standard multivariate normal with mean
     (5,5)
  ##
  S2 <- matrix(rnorm(n2*r, mean=5), n2, r)
  X2 <- S2 %*% A2

  ##
  ## combine the components 1&2 to form the mixture dataset
  ## and run the algorithm
  ##
  X<-rbind(X1, X2)
  rst2 <- EMFASTICAALG(X, 2)
```

```
##
## estimated class infomation
##
estimatedclass <- ESTIMATEDMEMBER(rst2) # estimated class info
levels(estimatedclass) <- rank(rst2$Lambdas[nrow(rst2$Lambdas)
    ,]) # numbering the estimated species, very important!

##
## classification difference rate
##
CER[i,1] = CLASSDIFFRATE(trueclass,estimatedclass)

##
## ISE calculation
##

# set range and grid points
Xrange <- 1.2*(range(X[,1])-mean(X[,1]))+mean(X[,1])
Yrange <- 1.2*(range(X[,2])-mean(X[,2]))+mean(X[,2])
ngrid <- 80
gridpoints <- t(as.matrix(expand.grid(seq(Xrange[1],Xrange[2],
    length.out=ngrid), seq(Yrange[1],Yrange[2],length.out=ngrid))
    ))

# mixture components density evaluation
fvalues <- list()
for(clsind in 1:2){
  fvalues[[clsind]] <- ATRANSDENSITY(gridpoints, MixMtrs[[clsind
      ]], fInd[[clsind]])# we can check this by, say, "plot(t(
      gridpoints), col=(fvalues[[1]]!=0)+3)" and "sum(fvalues
      [[1]])*(Xrange[2]-Xrange[1])/79*(Yrange[2]-Yrange[1])/79"
}

# estimated mixture components original signal density
    evaluation
festInd <- list()
for(clsind in 1:2){
  keyind <- order(levels(estimatedclass))[clsind]
  festInd[[clsind]] <- function(grid){
    n<-ncol(grid)
    answer <- rep(1,n)
    for(i in 1:n){
      for(j in 1:r){
        answer[i] <- answer[i] * WtsKde(rst2$OriginalSignals[[
            keyind]][j,],rst2$MembershipProbs[,keyind],grid[j,i],
            rst2$ICABandWidth[keyind])$weightedkde
      }
    }
    answer <- answer/(sum(rst2$MembershipProbs[,keyind]))^r
    answer
```

```
  }
}

# estimated mixture components density evaluation
festvalues <- list()
for(clsind in 1:2){
  keyind <- order(levels(estimatedclass))[clsind]
  festvalues[[clsind]] <- ATRANSDENSITY(gridpoints-colSums(X*
     rst2$MembershipProbs[,keyind]/sum(rst2$MembershipProbs[,
     keyind])), solve(rst2$WMtrs[[keyind]]), festInd[[keyind]])
     # we can check this by "plot(t(gridpoints), col=(festvalues
     [[1]]>0.0001)+3)" and "sum(festvalues[[1]])*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)"
}

# integrated squared error
for(j in 1:2){
  ISE[i,j] <- sum((fvalues[[j]]-festvalues[[j]])^2)*(Xrange[2]-
     Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
}

##
## calculate integrated squared error by npEM
##
rst2npem <- npEM(X, mu0=2, samebw=FALSE)
#rst2npem_ <- EMFASTICAALG(X, 2, maxiter = 500, tol = 1e-8,
   combine = FALSE, seednum = 100)

postmember_rst2npem <- rst2npem$posteriors == apply (
   rst2npem$posteriors ,1, max) # estimated membership matrix
estimatedmember_rst2npem <- factor (rep((1:2) ,nrow(X))[as.
   vector(t(postmember_rst2npem))]) # estimated membership info
levels(estimatedmember_rst2npem) <- rank(rst2npem$lambda[nrow(
   rst2npem$lambda),]) # numbering the estimated species
   according to weights, very important!

margingrid <- cbind(seq(Xrange[1],Xrange[2],length.out=ngrid),
   seq(Yrange[1],Yrange[2],length.out=ngrid))

fnpemvalues <- list()
for(clsind in 1:2){
  keyind <- order(levels(estimatedmember_rst2npem))[clsind]
  fnpemvalues[[clsind]] <- 1
  for(j in 1:r){
    h <- rst2npem$bandwidth[j,keyind]
    fnpemvalues[[clsind]] <- apply(expand.grid(fnpemvalues[[
       clsind]],WtsKde(X[,j], rst2npem$posteriors[,keyind],
       margingrid[,j], h)$weightedkde),1,prod)
    fnpemvalues[[clsind]] <- fnpemvalues[[clsind]]/sum(
       rst2npem$posteriors[,keyind])
  }
```

```
  }

  for(j in 1:2){
    ISEnpEM[i,j] <- sum((fvalues[[j]]-fnpemvalues[[j]])^2)*(Xrange
       [2]-Xrange[1])/(ngrid-1)*(Yrange[2]-Yrange[1])/(ngrid-1)
  }

  CERnpEM[i,1] = CLASSDIFFRATE(trueclass,estimatedmember_rst2npem)

}

##
## sqrt(MISE)
##
sqrt(colMeans(ISE))

sqrt(colMeans(ISEnpEM))

##
## classification error rates
##
summary(CER[,1])

summary(CERnpEM[,1])
```

iris.R

This R script runs the NSMM-ICA algorithm to analyse the well known iris data set. The results are summarized in Section 4.3.

```
####################################
## Iris Flower Data Classification
##
## Written by Xiaotian Zhu
####################################

library(Rcpp)
library(RcppArmadillo)
library(icamix)

##
## read in data
##
data(iris)
irisdata <- as.matrix(iris[,1:4])

##
## run NSMM-ICA algorithm
##
```

```
iris1 <- EMFASTICAALG(irisdata, 3) # with iterative bandwidth
   selection

##
## run K-Means clustering
##
iriskmeans <- kmeans(irisdata, 3)

##
## visualize and compare results
##
truespecies <- iris[,5] # true species info
levels(truespecies) <- c(1,2,3) # numbering the true species

estimatedspecies <- ESTIMATEDMEMBER(iris1) # estimated species
   info
levels(estimatedspecies) <- c(3, 1, 2) # numbering the estimated
   species

kmeanspecies <- factor(iriskmeans$cluster) # K-Means species info
levels(kmeanspecies) <- c(1, 2, 3) # numbering the K-Means species

# 2-dim plots
par(mfrow = c(1, 3))
plot(irisdata[,1:2], asp = 1, col = as.numeric(levels(truespecies)
   )[truespecies]+1, pch = as.numeric(levels(truespecies))[
   truespecies], main = "Iris Species",cex.lab=1.1)
plot(irisdata[,1:2], asp = 1, col=as.numeric(levels(
   estimatedspecies))[estimatedspecies]+1, pch = as.numeric(levels
   (estimatedspecies))[estimatedspecies], main = "NSMM-ICA",cex.
   lab=1.1)
plot(irisdata[,1:2], asp = 1, col=as.numeric(levels(kmeanspecies))
   [kmeanspecies]+1, pch = as.numeric(levels(kmeanspecies))[
   kmeanspecies], main = "K-Means",cex.lab=1.1)

# pairs scatter plots
pairs(irisdata, col = as.numeric(levels(truespecies))[truespecies
   ]+1, pch = as.numeric(levels(truespecies))[truespecies], main =
    "Iris Species Pairs Scatter Plot")
pairs(irisdata, col=as.numeric(levels(estimatedspecies))[
   estimatedspecies]+1, pch = as.numeric(levels(estimatedspecies))
   [estimatedspecies], main ="NSMM-ICA Classification Pairs
   Scatter Plot")

##
## calculate classification error rate
##
CLASSDIFFRATE(truespecies, estimatedspecies)
```

wine.R

This R script runs the NSMM-ICA algorithm to analyse the italian wine data set. The results are summarized in Section 4.4.

```
############################
## Wine data classification
##
## Written by Xiaotian Zhu
############################

library(devtools)
library(RcppArmadillo)
library(icamix)

##
## read in data
##
winedata <- as.matrix(read.csv(file="wine.csv", header=FALSE))

##
## running "EMFASTICAALG" on winedata
##
wineresult <- EMFASTICAALG(winedata[,-1], 3, icaiter = 500, tol =
   1e-8)

##
## interpret results
##
estimatedwine <- ESTIMATEDMEMBER(wineresult)
levels(estimatedwine) <- c(1,3,2)

truewine <- factor(winedata[,1])

# 2-dim plots
par(mfrow = c(1, 2))
pairs(winedata[,-1], asp = 1, col = as.numeric(levels(truewine))[
   truewine]+1, pch = as.numeric(levels(truewine))[truewine], main
    = "True Wine Class",cex.lab=1.1)
pairs(winedata[,-1], asp = 1, col=as.numeric(levels(estimatedwine)
   )[estimatedwine]+1, pch = as.numeric(levels(estimatedwine))[
   estimatedwine], main = "NSMM-ICA Estimated Wine Class",cex.lab
   =1.1)
par(mfrow = c(1, 1))

##
## calculate classification error rate
##
CLASSDIFFRATE(truewine, estimatedwine)

##
```

```
## Pricipal Components Analysis followed by NSMM-ICA
## entering raw data and extracting PCs
## from the correlation matrix
##
pcafit <- princomp(winedata[,-1], cor=TRUE)
summary(pcafit) # print variance accounted for
loadings(pcafit) # pc loadings
plot(pcafit,type="lines") # scree plot
pcafit$scores # the principal components
biplot(pcafit)

pcawineresult <- EMFASTICAALG(pcafit$scores[,1:5], 3)
pcaestimatedwine <- ESTIMATEDMEMBER(pcawineresult)
levels(pcaestimatedwine) <- c(3,1,2)
CLASSDIFFRATE(truewine, pcaestimatedwine)
library(phyclust)
RRand(as.numeric(levels(truewine)[truewine]),as.numeric(levels(
    pcaestimatedwine)[pcaestimatedwine]))

# 2-dim plots
par(mfrow = c(1, 3))
plot(winedata[,2:3], asp = 1, col = as.numeric(levels(truewine))[
    truewine]+1, pch = as.numeric(levels(truewine))[truewine], main
     = "True Wine Class",cex.lab=1.1)
plot(winedata[,2:3], asp = 1, col=as.numeric(levels(estimatedwine)
    )[estimatedwine]+1, pch = as.numeric(levels(estimatedwine))[
    estimatedwine], main = "NSMM-ICA Class",cex.lab=1.1)
plot(winedata[,2:3], asp = 1, col=as.numeric(levels(
    pcaestimatedwine))[pcaestimatedwine]+1, pch = as.numeric(levels
    (pcaestimatedwine))[pcaestimatedwine], main = "PCA + NSMM-ICA
    Class",cex.lab=1.1)
par(mfrow = c(1, 1))
```

waterlevel.R

This R script runs the NSMM-ICA algorithm to analyse the water level data set. The results are summarized in Section 4.5.

```
############################
## Waterlevel Data Analysis
##
## Written by Xiaotian Zhu
############################

library(Rcpp)
library(RcppArmadillo)
library(icamix)

##
```

```
## get the data
##
library(mixtools)
data(Waterdata)

##
## modify the data
##
WaterModified <- matrix(0,nrow=2*nrow(Waterdata),4)
WaterModified[1:nrow(Waterdata),] <- as.matrix(Waterdata[,c
   (1,2,3,4)+2])
WaterModified[(1+nrow(Waterdata)):(2*nrow(Waterdata)),] <- as.
   matrix(Waterdata[,c(2,1,4,3)+6])
colnames(WaterModified)<-c("11&5","4&10", "2&8", "7&1")

##
## run the algorithms on modified data
##
set.seed(100)
c <- npEM(WaterModified, mu0=3, samebw=FALSE)
d <- EMFASTICAALG(WaterModified, 3, tol=1e-8)

##
## interpret results of the analysis of the modified data
##
postmembership_c <- c$posteriors == apply (c$posteriors ,1, max) #
    estimated membership matrix for c
estimatedmember_c <- factor (rep((1:3) ,nrow(WaterModified))[as.
   vector(t(postmembership_c))]) # estimated membership info for c
levels ( estimatedmember_c ) <- c(3, 1, 2) # numbering the
   estimated membership info for c

estimatedmember_d <- ESTIMATEDMEMBER(d) # estimated membership
   info for d
levels(estimatedmember_d) <- c(1, 2, 3) # numbering the estimated
   membership info for d

pairs(WaterModified, col = as.numeric(levels(estimatedmember_c))[
   estimatedmember_c]+1, pch = as.numeric(levels(estimatedmember_c
   ))[estimatedmember_c], main = "npEM Classification Pairs
   Scatter Plot", cex.lab=1.1)
pairs(WaterModified, col = as.numeric(levels(estimatedmember_d))[
   estimatedmember_d]+1, pch = as.numeric(levels(estimatedmember_d
   ))[estimatedmember_d], main = "NSMM-ICA Classification Pairs
   Scatter Plot", cex.lab=1.1)

##
## classification difference rate
##
CLASSDIFFRATE(estimatedmember_c,estimatedmember_d)
```

```
##
## estimated mixing matrices for each cluster
##
solve(d$WMtrs[[1]])
solve(d$WMtrs[[2]])
solve(d$WMtrs[[3]])

c_3 <- WtsFastICA(t(WaterModified), c$posteriors[,1], matrix(0))
c_1 <- WtsFastICA(t(WaterModified), c$posteriors[,2], matrix(0))
c_2 <- WtsFastICA(t(WaterModified), c$posteriors[,3], matrix(0))

solve(c_1$UnmixX)
solve(c_2$UnmixX)
solve(c_3$UnmixX)

##
## compare mixture component marginal density estimates
##

# a function for drawing vessels (courtesy of Dr. David Hunter)
vessel <- function(centerx, centery, wd, ht, angle, xtoy) {
  theta = angle * pi/180
  Rot = matrix(c(cos(theta), -sin(theta), sin(theta), cos(theta)),
      2,2)
  r1 = rbind(wd/xtoy * c(.5, .5, -.5, -.5), ht * c(.5, -.5, -.5,
     .5))
  r2 = rbind(wd/xtoy * c(.5, .5, -.5, -.5), ht * c(.5, .4, .4, .5)
     )
  r3 = rbind(wd/xtoy * c(.1, .1, -.1, -.1), ht * c(.5, .6, .6, .5)
     )
  Rr1 = Rot %*% r1
  Rr2 = Rot %*% r2
  Rr3 = Rot %*% r3
  polygon(centerx + xtoy*Rr1[1,], centery + Rr1[2,])
  polygon(centerx + xtoy*Rr2[1,], centery + Rr2[2,])
  polygon(centerx + xtoy*Rr3[1,], centery + Rr3[2,], density=25,
     angle=90-angle)
  polygon(centerx + wd*c(-1,-1,1,1), centery + min(Rr1[2,]) - ht*c
     (0,.1,.1,0),
          density = 25, angle=90)
}

#npem result
XLIM <- range(WaterModified)
x <- seq(XLIM[1], XLIM[2], 1)
wtddensity <- array(0,c(length(x),ncol(WaterModified),3))

for(i in 1:3){
  for(j in 1:ncol(WaterModified)){
    tempbdw <- c$bandwidth[j,i]
```

```
    wtddensity[,j,i] <- WtsKde(WaterModified[,j], c$posteriors[,i
       ], x, tempbdw)$weightedkde
  }
}
wtddensity<-wtddensity/nrow(WaterModified)

par(mfrow=c(2,2))

angle1<-c(-30,-60,-120,-150)
angle2<-c(150,120,60,30)
for(pic in 1:4){

    hist(WaterModified[,pic], freq=FALSE, breaks=seq(XLIM[1], XLIM
       [2], 2), xlab="XX", ylab="Density", xlim=XLIM, ylim=c(0,
       max(wtddensity)*1.05), main=paste("vessel orientation at",
       colnames(WaterModified)[pic],"o'clock"),asp=((XLIM[2]-XLIM
       [1])/max(wtddensity)/1.05))
    lines(x, wtddensity[,pic,1],col="red",lwd=3, lty=1)
    lines(x, wtddensity[,pic,2],col="blue",lwd=3, lty=2)
    lines(x, wtddensity[,pic,3],col="green",lwd=3, lty=3)
    lines(x, wtddensity[,pic,1]+wtddensity[,pic,2]+wtddensity[,pic
       ,3], col="yellow", lwd=2)
    legend(0.4*XLIM[1]+0.6*XLIM[2],max(wtddensity)*1.05, c
       (0.473,0.078,0.449, 1), lty=c(1,2,3,1), lwd=c(3,3,3,2),col=
       c("red","blue","green", "yellow"),title="Lambdas",bty = "n
       ")
    vessel(-70,0.036,18,0.025,angle1[pic],((XLIM[2]-XLIM[1])/max(
       wtddensity)/2))
    vessel(-45,0.036,18,0.025,angle2[pic],((XLIM[2]-XLIM[1])/max(
       wtddensity)/2))
}


#NSMM-ica result
XLIM <- range(WaterModified)
x <- seq(XLIM[1], XLIM[2], 1)
wtddensity_ <- array(0,c(length(x),ncol(WaterModified),3))

for(i in 1:3){
  marginalbdwica <- sqrt(diag(solve(d$WMtrs[[i]])%*%diag(
     d$ICABandWidth[i,1]^2,ncol(WaterModified))%*%t(solve(d$WMtrs
     [[i]]))))
  for(j in 1:ncol(WaterModified)){
    tempbdw <- marginalbdwica[j]
    wtddensity_[,j,i] <- WtsKde(WaterModified[,j],
       d$MembershipProbs[,i], x, tempbdw)$weightedkde
  }
}
wtddensity_<-wtddensity_/nrow(WaterModified)

par(mfrow=c(2,2))
```

```
angle1<-c(-30,-60,-120,-150)
angle2<-c(150,120,60,30)
for(pic in 1:4){

  hist(WaterModified[,pic], freq=FALSE, breaks=seq(XLIM[1], XLIM
     [2], 2), xlab="XX", ylab="Density", xlim=XLIM, ylim=c(0, max(
     wtddensity)*1.05), main=paste("vessel orientation at",
     colnames(WaterModified)[pic],"o'clock"),asp=((XLIM[2]-XLIM
     [1])/max(wtddensity)/1.05))
  lines(x, wtddensity_[,pic,1],col="blue",lwd=3, lty=2)
  lines(x, wtddensity_[,pic,2],col="green",lwd=3, lty=3)
  lines(x, wtddensity_[,pic,3],col="red",lwd=3, lty=1)
  lines(x, wtddensity_[,pic,1]+wtddensity_[,pic,2]+wtddensity_[,
     pic,3], col="yellow", lwd=2)
  legend(0.4*XLIM[1]+0.6*XLIM[2],max(wtddensity)*1.05, c
     (0.491,0.084,0.425, 1), lty=c(1,2,3,1), lwd=c(3,3,3,2),col=c
     ("red","blue","green", "yellow"),title="Lambdas",bty = "n")
  vessel(-70,0.036,18,0.025,angle1[pic],((XLIM[2]-XLIM[1])/max(
     wtddensity)/2))
  vessel(-45,0.036,18,0.025,angle2[pic],((XLIM[2]-XLIM[1])/max(
     wtddensity)/2))
}

##
## plot data loglikelihood
##
plot.ts(d$ObjValue[-1,],xlab="iteration",ylab="data loglikelihood
   ")
```

reactiontime.R

This R script runs the NSMM-ICA algorithm to analyse the reaction time data set. The results are summarized in Section 4.6.

```
############################
## Children Reaction Time
## Data Analysis
##
## Written by Xiaotian Zhu
############################

library(Rcpp)
library(RcppArmadillo)
library(icamix)

##
## get the data
##
```

```
library(mixtools)
data(RTdata)

##
## run the algorithm
##
set.seed(100)
a <- npEM(RTdata, mu0=2, samebw=FALSE)
b <- EMFASTICAALG(RTdata, 2, h=0, tol=1e-8)

##
## interpret results
##
postmembership_a <- a$posteriors == apply(a$posteriors,1,max) #
   estimated membership matrix for a
estimatedmember_a <- factor(rep((1:2),nrow(RTdata))[as.vector(t(
   postmembership_a))]) # estimated membership info for a
levels(estimatedmember_a) <- c(2,1) # numbering the estimated
   membership info for a

estimatedmember_b <- ESTIMATEDMEMBER(b) # estimated membership
   info for b
levels(estimatedmember_b) <- c(2,1) # numbering the estimated
   membership info for b

pairs(RTdata, col = as.numeric(levels(estimatedmember_a))[
   estimatedmember_a]+1, pch = as.numeric(levels(estimatedmember_a
   ))[estimatedmember_a], main = "npEM Classification Pairs
   Scatter Plot")
pairs(RTdata, col = as.numeric(levels(estimatedmember_b))[
   estimatedmember_b]+1, pch = as.numeric(levels(estimatedmember_b
   ))[estimatedmember_b], main = "NSMM-ICA Classification Pairs
   Scatter Plot")

##
## classification difference rate
##
CLASSDIFFRATE(estimatedmember_a,estimatedmember_b)

##
## plot data loglikelihood
##
plot.ts(b$ObjValue[-1,],xlab="iteration",ylab="data loglikelihood
   ")
```

tone.R

This R script runs the NSMM-ICA algorithm to analyse Cohen's tone data set. The results are summarized in Section 4.7.

```
##############################
## Cohen's Tone Data Analysis
##
## Written by Xiaotian Zhu
##############################

library(Rcpp)
library(RcppArmadillo)
library(icamix)

##
## get the data
##
library(mixtools)
data(tonedata)

##
## Parameter estimates for finite mixtures of linear regressions
   with unspecified error structure.
## based on Hunter and Young (2012).
##
set.seed(100)
c<-spregmix(tuned~stretchratio, data=tonedata, verb=TRUE)
c$beta

postmembership_c <- c$posterior == apply(c$posterior,1,max) #
   estimated membership matrix
estimatedmember_c <- factor(rep((1:2),nrow(tonedata))[as.vector(t(
   postmembership_c))]) # estimated membership info
levels(estimatedmember_c) <- c(1,2) # numbering the estimated
   membership info

##
## run the algorithm
##
set.seed(100)
a <- npEM(tonedata, mu0=2, samebw=FALSE)
b <- EMFASTICAALG(tonedata, 2, h=0, tol=1e-8)

##
## interpret results
##
postmembership_a <- a$posteriors == apply(a$posteriors,1,max) #
   estimated membership matrix for a
estimatedmember_a <- factor(rep((1:2),nrow(tonedata))[as.vector(t(
   postmembership_a))]) # estimated membership info for a
levels(estimatedmember_a) <- c(1,2) # numbering the estimated
   membership info for a

estimatedmember_b <- ESTIMATEDMEMBER(b) # estimated membership
   info for b
```

```
levels(estimatedmember_b) <- c(1,2) # numbering the estimated
   membership info for b

par(mfrow=c(1,3))
plot(tonedata, col = as.numeric(levels(estimatedmember_a))[
   estimatedmember_a]+1, pch = as.numeric(levels(estimatedmember_a
   ))[estimatedmember_a], main = "npEM Classification Scatter Plot
   ",asp=1)
plot(tonedata, col = as.numeric(levels(estimatedmember_c))[
   estimatedmember_c]+1, pch = as.numeric(levels(estimatedmember_c
   ))[estimatedmember_c], main = "SP EM Classification Scatter
   Plot",asp=1)
plot(tonedata, col = as.numeric(levels(estimatedmember_b))[
   estimatedmember_b]+1, pch = as.numeric(levels(estimatedmember_b
   ))[estimatedmember_b], main = "NSMM-ICA Classification Scatter
   Plot",asp=1)
par(mfrow=c(1,1)) # reset par

##
## plot data loglikelihood
##
plot.ts(b$ObjValue[-1,],xlab="iteration",ylab="data loglikelihood
   ")

##
## fit weighted least square for each estimated component
##
lsfit <- list()
for(i in 1:2){
  lsfit[[i]] <- lm(tuned~stretchratio,data=tonedata, weights=
     b$MembershipProbs[,i])
}

for(i in 1:2){
  cat("--- estimated component ")
  cat(i)
  print(summary(lsfit[[i]]))
}

##
## classification difference rate between SP EM and NSMM-ICA
##
CLASSDIFFRATE(estimatedmember_b,estimatedmember_c)
```

learnimagecode.R

This R script runs the NSMM-ICA algorithm for a task of unsupervised learning of efficient codes of images. The results are summarized in Section 4.8.

```
############################
## Learning Image Code
##
## Written by Xiaotian Zhu
############################

##
## preparing image files
##
library("EBImage")

newspaper <-readImage("NewspaperPage.jpg")
painting <- readImage("WayToCalvary.jpg")
news <- channel(newspaper, 'grey')
paint <- channel(painting, 'grey')
writeImage(news,  'news.jpeg', quality=100)
writeImage(paint, 'paint.jpeg', quality=100)

##
## read in image files
##
news <- as.matrix(read.csv(file="news.csv", header=TRUE))
paint <- as.matrix(read.csv(file="paint.csv", header=TRUE))

##
## generate data set for task of learning efficient image code by
## randomly taking npatch patches of d by d pixels from both
   images
##
npatch <- 5000
d <- 12

X1<-matrix(0, nrow=npatch, ncol=d*d)
xs<-floor(runif(npatch,min=1,max=1+ncol(paint)-d))
ys<-floor(runif(npatch,min=1,max=1+nrow(paint)-d))
for(i in 1:npatch){
  X1[i,] <- as.vector(paint[(ys[i]):(ys[i]+d-1),(xs[i]):(xs[i]+d
     -1)])
}

X2<-matrix(0, nrow=npatch, ncol=d*d)
xs<-floor(runif(npatch,min=1,max=1+ncol(news)-d))
ys<-floor(runif(npatch,min=1,max=1+nrow(news)-d))
for(i in 1:npatch){
  X2[i,] = as.vector(news[(ys[i]):(ys[i]+d-1),(xs[i]):(xs[i]+d-1)
     ])
}

X<-matrix(0, nrow=2*npatch, ncol=d*d)
positionsX1<-sample(1:(2*npatch),npatch,replace=FALSE)
X[positionsX1,]<-X1
```

```
X[-positionsX1,]<-X2

##
## testing "EMFASTICAALG" on the combined dataset X and output
   results to "imagecoding.log"
##
library(devtools)
library(RcppArmadillo)
library(icamix)
sink("imagecoding.log")

Xresult <- EMFASTICAALG(X, 2, icaiter=300)

XresultWMtrs = as.data.frame(do.call(cbind, Xresult$WMtrs))

write.csv(XresultWMtrs, file="XresultWMtrs.csv",row.names=FALSE)

write.csv(Xresult$MembershipProbs, file="XresultMembershipProbs.
   csv",row.names=FALSE)

write.csv(positionsX1, file="positionsX1.csv",row.names=FALSE)

sink()

##
## Calculate classification error rate
##
MembershipProbs <- as.matrix(read.csv("XresultMembershipProbs.csv
   ", header=TRUE))
MembershipMatrix <- MembershipProbs == apply(MembershipProbs,1,max
   )
estimatedMember <- factor(rep((1:2),nrow(MembershipProbs))[as.
   vector(t(MembershipMatrix))])

positionsX1 <- (read.csv("positionsX1.csv", header=TRUE))$x
trueMember <- c(1:nrow(MembershipProbs))
trueMember[positionsX1] <- 1
trueMember[-positionsX1] <- 2
sum(as.numeric(estimatedMember) != trueMember)/nrow(
   MembershipProbs)

##
## plotting basis class for each mixing component
##
WMtrs <- as.matrix(read.csv("XresultWMtrs.csv", header=TRUE))

A1<-solve(WMtrs[,1:(d*d)])
for(i in 1:(d*d)){
tempmin=min(A1[,i])
A1[,i]=A1[,i]-tempmin
tempmax=max(A1[,i])
```

```
A1[,i]=A1[,i]/tempmax
}
#display(matrix(A1[,1],d))
B1<-array(A1,c(d,d,d*d))
#display(B1,method="raster",all=TRUE)

plot(c(0,d*d+d), c(0,d*d+d), type = "n", xlab = "", ylab = "",asp
    =1)
for(i in 1:d){
for(j in 1:d){
rasterImage(B1[,,i+(j-1)*d],1+(i-1)*(d+1),1+(j-1)*(d+1),d+(i-1)*(d
    +1),d+(j-1)*(d+1),interpolate=FALSE)
}
}


A2<-solve(WMtrs[,(1+(d*d)):(2*d*d)])
for(i in 1:(d*d)){
tempmin=min(A2[,i])
A2[,i]=A2[,i]-tempmin
tempmax=max(A2[,i])
A2[,i]=A2[,i]/tempmax
}
#display(matrix(A1[,1],d))
B2<-array(A2,c(d,d,d*d))
#display(B2,method="raster",all=TRUE)

plot(c(0,d*d+d), c(0,d*d+d), type = "n", xlab = "", ylab = "",asp
    =1)
for(i in 1:d){
for(j in 1:d){
rasterImage(B2[,,i+(j-1)*d],1+(i-1)*(d+1),1+(j-1)*(d+1),d+(i-1)*(d
    +1),d+(j-1)*(d+1),interpolate=FALSE)
}
}
```

multispectral.R

This R script runs the NSMM-ICA algorithm for a task of classification of multispectral images. The results are summarized in Section 4.9.

```
##########################################
## multispectral data classification
##
## Written by Xiaotian Zhu
##########################################

library(devtools)
library(RcppArmadillo)
```

```
library(icamix)

##
## read in data
##
SATRAW <- as.matrix(read.csv(file="sat.csv", header=FALSE))
SAT <- SATRAW[,-37]

##
## running "EMFASTICAALG" on only the 4 attributes of the center
   pixel of each 3x3 patch
##
sink("multispectral_log.txt")
SATresult <- EMFASTICAALG(SAT[,17:20], 2, icaiter = 500)
sink()

SATWMtrs = as.data.frame(do.call(cbind, SATresult$WMtrs))

##
## saving results
##
write.csv(SATWMtrs, file="SATWMtrs.csv",row.names=FALSE)

write.csv(SATresult$MembershipProbs, file="SATMembershipProbs.csv
   ",row.names=FALSE)

##
## interpreting results
##
MembershipProbs <- as.matrix(read.csv("SATMembershipProbs.csv",
   header=TRUE))

postmembership <- MembershipProbs == apply(MembershipProbs,1,max)
   # estimated membership matrix
estimatedmember <- factor(rep((1:2),nrow(SAT))[as.vector(t(
   postmembership))]) # estimated membership info
levels(estimatedmember) <- c(0, 1) # numbering the estimated
   membership info

truemember <- ((SATRAW[,37]==2)|(SATRAW[,37]==5))*1

sum(truemember != estimatedmember)/nrow(SAT)

pairs(SAT[,17:20], col = truemember+2, pch = truemember, main = "
   npEM Classification Pairs Scatter Plot")
pairs(SAT[,17:20], col = as.numeric(levels(estimatedmember))[
   estimatedmember]+2, pch = as.numeric(levels(estimatedmember))[
   estimatedmember], main = "NSMM-ICA Classification Pairs Scatter
    Plot")
```

References

Aeberhard, S., Coomans, D., & De Vel, O. (1992). Comparison of classifiers in high dimensional settings. *Dept. Math. Statist., James Cook Univ., North Queensland, Australia, Tech. Rep*(92-02).

Allman, E. S., Matias, C., & Rhodes, J. A. (2009). Identifiability of parameters in latent structure models with many observed variables. *The Annals of Statistics*, 3099–3132.

Azzalini, A., & Torelli, N. (2007). Clustering via nonparametric density estimation. *Statistics and Computing*, *17*(1), 71–80.

Bajari, P., Hahn, J., Hong, H., & Ridder, G. (2011). A note on semiparametric estimation of finite mixtures of discrete choice models with application to game theoretic models. *International Economic Review*, *52*(3), 807–824.

Banfield, J. D., & Raftery, A. E. (1992). Ice floe identification in satellite images using mathematical morphology and clustering about principal curves. *Journal of the American Statistical Association*, *87*(417), 7–16.

Banfield, J. D., & Raftery, A. E. (1993). Model-based gaussian and non-gaussian clustering. *Biometrics*, 803–821.

Benaglia, T., Chauveau, D., & Hunter, D. R. (2009). An EM-like algorithm for semi-and nonparametric estimation in multivariate mixtures. *Journal of Computational and Graphical Statistics*, *18*(2), 505–526.

Benaglia, T., Chauveau, D., Hunter, D. R., et al. (2011). Bandwidth selection in an EM-like algorithm for nonparametric multivariate mixtures. *Nonparametric Statistics and Mixture Models: A Festschrift in Honor of Thomas P. Hettmansperger*, 15–27.

Benaglia, T., Chauveau, D., Hunter, D. R., & Young, D. (2009). mixtools: An R package for analyzing finite mixture models. *Journal of Statistical Software*, *32*(6), 1–29. Retrieved from http://www.jstatsoft.org/v32/i06/

Bock, H. H. (1996). Probabilistic models in cluster analysis. *Computational Statistics & Data Analysis*, *23*(1), 5–28.

Bonhomme, S., Jochmans, K., & Robin, J.-M. (2011). *Nonparametric estimation of finite mixtures* (Tech. Rep.). Technical report, Department of Statistics.

Bordes, L., Chauveau, D., & Vandekerkhove, P. (2007). A stochastic EM algorithm for a semiparametric mixture model. *Computational Statistics & Data Analysis*, *51*(11), 5429–5443.

Bordes, L., Kojadinovic, I., Vandekerkhove, P., et al. (2013). Semiparametric estimation of a two-component mixture of linear regressions in which one component is known. *Electronic Journal of Statistics*, *7*, 2603–2644.

Bordes, L., Mottelet, S., Vandekerkhove, P., et al. (2006). Semiparametric estimation of a two-component mixture model. *The Annals of Statistics*, *34*(3), 1204–1232.

Bordes, L., & Vandekerkhove, P. (2010). Semiparametric two-component mixture model with a known component: An asymptotically normal estimator. *Mathematical Methods of Statistics*, *19*(1), 22–41.

Butucea, C., & Vandekerkhove, P. (2014). Semiparametric mixtures of symmetric distributions. *Scandinavian Journal of Statistics*, *41*(1), 227–239.

Campbell, J. G., Fraley, C., Murtagh, F., & Raftery, A. E. (1997). Linear flaw detection in woven textiles using model-based clustering. *Pattern Recognition Letters*, *18*(14), 1539–1548.

Celeux, G., & Govaert, G. (1995). Gaussian parsimonious clustering models. *Pattern recognition*, *28*(5), 781–793.

Chauveau, D., Hunter, D. R., & Levine, M. (2014). Semi-parametric estimation for conditional independence multivariate finite mixture models.

Chauveau, D., Saby, N., Orton, T. G., Lemercier, B., Walter, C., & Arrouays, D. (2014). Large-scale simultaneous hypothesis testing in monitoring carbon content from french soil databaseała semi-parametric mixture approach. *Geoderma*, *219*, 117–124.

Cohen, E. A. (1984). Some effects of inharmonic partials on interval perception. *Music Perception*, 323–349.

Comon, P. (1994). Independent component analysis, a new concept? *Signal processing*, *36*(3), 287–314.

Cruz-Medina, I., & Hettmansperger, T. (2004). Nonparametric estimation in semi-parametric univariate mixture models. *Journal of Statistical Computation and Simulation*, *74*(7), 513–524.

Cruz-Medina, I., Hettmansperger, T., & Thomas, H. (2004). Semiparametric mixture models and repeated measures: the multinomial cut point model. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *53*(3), 463–474.

Dasgupta, A., & Raftery, A. E. (1998). Detecting features in spatial point processes with clutter via model-based clustering. *Journal of the American Statistical Association*, *93*(441), 294–302.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1–38.

De Veaux, R. D. (1989). Mixtures of linear regressions. *Computational Statistics & Data Analysis*, *8*(3), 227–245.

Eddelbuettel, D. (2013). *Seamless R and C++ integration with Rcpp*. New York: Springer. (ISBN 978-1-4614-6867-7)

Eddelbuettel, D., & François, R. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, *40*(8), 1–18. Retrieved from http://www.jstatsoft.org/v40/i08/

Eddelbuettel, D., & Sanderson, C. (2014, March). Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, *71*, 1054–1063. Retrieved from http://dx.doi.org/10.1016/j.csda.2013.02.005

Eggermont, P. (1999). Nonlinear smoothing and the EM algorithm for positive integral equations of the first kind. *Applied mathematics & optimization*, *39*(1), 75–91.

Eggermont, P., & LaRiccia, V. (1995). Maximum smoothed likelihood density estimation for inverse problems. *The Annals of Statistics*, *23*(1), 199–220.

Eggermont, P., & LaRiccia, V. (2001). *Maximum penalized likelihood estimation: Volume i: Density estimation* (Vol. 1). Springer.

Elmore, R. T., Hettmansperger, T. P., & Thomas, H. (2004). Estimating component cumulative distribution functions in finite mixture models. *Communications in Statistics-Theory and Methods*, *33*(9), 2075–2086.

Elmore, R. T., & Wang, S. (2003). Identifiability and estimation in finite mixture models with multinomial components. *Technical report*.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, *7*(2), 179–188.

Forina, M., Leardi, R., Armanino, C., & Lanteri, S. (1988). Parvus an extendible package for data exploration. *Classification and Correla*.

Fraley, C. (1998). Algorithms for model-based gaussian hierarchical clustering. *SIAM Journal on Scientific Computing*, *20*(1), 270–281.

Fraley, C., & Raftery, A. E. (1998). How many clusters? which clustering method? answers via model-based cluster analysis. *The computer journal*, *41*(8), 578–588.

Fraley, C., & Raftery, A. E. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, *97*(458), 611–631.

Greengard, L., & Strain, J. (1991). The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, *12*(1), 79–94.

Guglielmi, A., Ieva, F., Paganoni, A. M., Ruggeri, F., & Soriano, J. (2014). Semiparametric bayesian models for clustering and classification in the presence of unbalanced in-hospital survival. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *63*(1), 25–46.

Hall, P., Neeman, A., Pakyari, R., & Elmore, R. (2005). Nonparametric inference in multivariate mixtures. *Biometrika*, *92*(3), 667–678.

Hall, P., & Zhou, X.-H. (2003). Nonparametric estimation of component distributions in a multivariate mixture. *Annals of Statistics*, 201–224.

Han, B., & Davis, L. S. (2006). Semi-parametric model-based clustering for dna microarray data. In *Pattern recognition, 2006. icpr 2006. 18th international conference on* (Vol. 3, pp. 324–327).

Hartigan, J. A. (1975). *Clustering algorithms*. Wiley.

Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., & Tibshirani, R. (2009). *The elements of statistical learning* (Vol. 2) (No. 1). Springer.

Hettmansperger, T., & Thomas, H. (2000). Almost nonparametric inference for repeated measures in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *62*(4), 811–825.

Hohmann, D., & Holzmann, H. (2013). Semiparametric location mixtures with distinct components. *Statistics*, *47*(2), 348–362.

Huang, M., Li, R., & Wang, S. (2013). Nonparametric mixture of regression models. *Journal of the American Statistical Association*, *108*(503), 929–941.

Hunter, D. R., & Lange, K. (2004). A tutorial on mm algorithms. *The American Statistician*, *58*(1), 30–37.

Hunter, D. R., Wang, S., & Hettmansperger, T. P. (2007). Inference for mixtures of symmetric distributions. *The Annals of Statistics*, 224–251.

Hunter, D. R., & Young, D. S. (2012). Semiparametric mixtures of regressions. *Journal of Nonparametric Statistics*, *24*(1), 19–38.

Hyvarinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *Neural Networks, IEEE Transactions on*, *10*(3), 626–634.

Hyvarinen, A., Karhunen, J., & Oja, E. (2002). Independent component analysis. *Studies in Informatics and Control*, *11*(2), 205–207.

Jutten, C., & Herault, J. (1991). Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, *24*(1), 1–10.

Kasahara, H., & Shimotsu, K. (2009). Nonparametric identification of finite mixture models of dynamic discrete choices. *Econometrica*, *77*(1), 135–175.

Kasahara, H., & Shimotsu, K. (2014). Non-parametric identification and estimation of the number of components in multivariate mixtures. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *76*(1), 97–111.

Kaufman, L., & Rousseeuw, P. J. (2009). *Finding groups in data: an introduction to cluster analysis* (Vol. 344). John Wiley & Sons.

Kruskal, J. B. (1976). More factors than subjects, tests and treatments: an indeterminacy theorem for canonical decomposition and individual differences scaling. *Psychometrika*, *41*(3), 281–293.

Kruskal, J. B. (1977). Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, *18*(2), 95–138.

Laird, N. M., & Ware, J. H. (1982). Random-effects models for longitudinal data. *Biometrics*, 963–974.

Lee, T.-W., Lewicki, M. S., & Sejnowski, T. J. (1999). Unsupervised classification with non-gaussian mixture models using ICA. *Advances in neural information processing systems*, 508–514.

Lee, T.-W., Lewicki, M. S., & Sejnowski, T. J. (2000). ICA mixture models for unsupervised classification of non-gaussian classes and automatic context switching in blind signal separation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *22*(10), 1078–1089.

Leung, D., & Qin, J. (2006). Semi-parametric inference in a bivariate (multivariate) mixture model. *Statistica Sinica*, *16*(1), 153.

Levine, M., Hunter, D., & Chauveau, D. (2011). Maximum smoothed likelihood for multivariate mixtures. *Biometrika*, *98*(2), 403–416.

Li, J., Ray, S., & Lindsay, B. G. (2007). A nonparametric statistical approach to clustering via mode identification. *Journal of Machine Learning Research*, *8*(8), 1687–1723.

Lindsay, B. G. (1995). Mixture models: theory, geometry and applications. In *Nsf-cbms regional conference series in probability and statistics* (pp. i–163).

Lu, W., & Peng, L. (2008). Semiparametric analysis of mixture regression models with competing risks data. *Lifetime data analysis*, *14*(3), 231–252.

MacQueen, J., et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297).

Mallapragada, P. K., Jin, R., & Jain, A. (2010). Non-parametric mixture models for clustering. In *Structural, syntactic, and statistical pattern recognition* (pp. 334–343). Springer.

Mallick, B., Hoffman, F. O., & Carroll, R. J. (2002). Semiparametric regression modeling with mixtures of berkson and classical error, with application to fallout from the nevada test site. *Biometrics*, *58*(1), 13–20.

McLachlan, G. J., & Basford, K. E. (1988). Mixture models. inference and applications to clustering. *Statistics: Textbooks and Monographs, New York: Dekker, 1988*, *1*.

Mukherjee, S., Feigelson, E. D., Babu, G. J., Murtagh, F., Fraley, C., & Raftery, A. (1998). Three types of gamma-ray bursts. *The Astrophysical Journal*, *508*(1), 314.

Murtagh, F., & Raftery, A. E. (1984). Fitting straight lines to point patterns. *Pattern recognition*, *17*(5), 479–483.

Naik, G. R., & Kumar, D. K. (2011). An overview of independent component analysis and its applications. *Informatica: An International Journal of Computing and Informatics*, *35*(1), 63–81.

Nandi, A. K. (1999). *Blind estimation using higher-order statistics*. Springer.

Ng, S., & McLachlan, G. (2003). An EM-based semi-parametric mixture model approach to the regression analysis of competing-risks data. *Statistics in medicine*, *22*(7), 1097–1111.

Ostrowski, A. (1960). Solution of equations and systems of equations. *New York and London*.

Palmer, J. A., Makeig, S., Kreutz-Delgado, K., & Rao, B. D. (2008). Newton method for the ICA mixture model. In *Icassp* (pp. 1805–1808).

Piaget, J., & Inhelder, B. (1956). The child's concept of space. *New York: Humanities Pr*.

R Core Team. (2014). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from http://www.R-project.org/

Raftery, A. E., & Dean, N. (2006). Variable selection for model-based clustering. *Journal of the American Statistical Association*, *101*(473), 168–178.

Raykar, V. C., & Duraiswami, R. (2007). The improved fast gauss transform with applications to machine learning. *Large Scale Kernel Machines*, 175–201.

Raykar, V. C., Yang, C., Duraiswami, R., & Gumerov, N. (2005). Fast computation of sums of gaussians in high dimensions.

Sejnowski, T. J. (1999). ICA mixture models for image processing.

Shah, C. A., Arora, M. K., & Varshney, P. K. (2004). Unsupervised classification of hyperspectral data: an ICA mixture model based approach. *International Journal of Remote Sensing*, *25*(2), 481–487.

Silverman, B. (1986). *Density estimation for statistics and data analysis* (Vol. 26). CRC press.

Silverman, B., Jones, M., Wilson, J., & Nychka, D. (1990). A smoothed EM approach to indirect estimation problems, with particular, reference to stereology and emission tomography. *Journal of the Royal Statistical Society. Series B (Methodological)*, 271–324.

Thomas, H., Lohaus, A., & Brainerd, C. J. (1993). Modeling growth and individual differences in spatial tasks. *Monographs of the Society for Research in Child Development*, i–190.

Tong, L., Inouye, Y., & Liu, R.-w. (1993). Waveform-preserving blind estimation of multiple independent sources. *Signal Processing, IEEE Transactions on*, *41*(7), 2461–2470.

Vandekerkhove, P. (2013). Estimation of a semiparametric mixture of regressions model. *Journal of Nonparametric Statistics*, *25*(1), 181–208.

Vichi, M. (2008). Fitting semiparametric clustering models to dissimilarity data. *Advances in Data Analysis and Classification*, *2*(2), 121–161.

Viele, K., & Tong, B. (2002). Modeling with mixtures of linear regressions. *Statistics and Computing*, *12*(4), 315–330.

Wolfe, J. H. (1963). *Object cluster analysis of social areas* (Unpublished doctoral dissertation). University of California.

Yoganarasimhan, H. (2013). Estimation of beauty contest auctions. *Available at SSRN 2250472*.

Zhang, W., Fan, J., & Sun, Y. (2009). A semiparametric model for cluster data. *Annals of statistics*, *37*(5A), 2377.

Zhong, S., & Ghosh, J. (2003). A unified framework for model-based clustering. *The Journal of Machine Learning Research*, *4*, 1001–1037.

Vita

Xiaotian Zhu

Personal Data

Phone: 814-883-0182

Email: xxz131@psu.edu

Education

Dec 2014 Ph.D. in Statistics (minor in CS) Penn State Univ., State College

Dec 2008 M.A. in Mathematics Penn State Univ., State College

Jun 2004 B.S. in Mathematics Zhejiang Univ., Hangzhou, China

Experience

Summer & Fall 2014 Graduate Researcher

Spring 2014 TA of Math/Stat 415 Introduction to Mathematical Statistics

May - Dec 2012 Statistical Research & Design Intern at Minitab Inc.

Sep - Dec 2011 OTS Intern at FDA CDER

Feb - June 2011 Statistical Research & Design Intern at Minitab Inc.

Fall 2009 Student Statistics Consultant

Aug 2007 - May 2008 Instructor of Math 220 Linear Algebra

Aug 2006 - May 2007 Instructor of Math 021 College Algebra

Software Package

icamix (http://cran.r-project.org/web/packages/icamix/index.html)