An Efficient Implementation of a Robust Clustering Algorithm

AN EFFICIENT IMPLEMENTATION OF A ROBUST CLUSTERING ALGORITHM

BY

MARTIN BLOSTEIN, B.Sc.

A THESIS

SUBMITTED TO THE DEPARTMENT OF MATHEMATICS & STATISTICS

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2016) McMaster University

(Statistics) Hamilton, Ontario, Canada

TITLE:	An Efficient Implementation of a Robust Clustering Algorithm
AUTHOR:	Martin Blostein B.Sc., (Mathematics and Statistics) McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Paul D. McNicholas
NUMBER OF PAGES:	xii, 59

To my parents, Steven and Dorothea Blostein, for their constant support and enthusiasm for all of my endeavours.

Abstract

Clustering and classification are fundamental problems in statistical and machine learning, with a broad range of applications. A common approach is the Gaussian mixture model, which assumes that each cluster or class arises from a distinct Gaussian distribution. This thesis studies a robust, high-dimensional extension of the Gaussian mixture model that automatically detects outliers and noise, and a computationally efficient implementation thereof.

The contaminated Gaussian distribution is a robust elliptic distribution that allows for automatic detection of "bad points", and is used to make robust the usual factor analysis model. In turn, the mixtures of contaminated Gaussian factor analyzers (MCGFA) algorithm allows high-dimesional, robust clustering, classification and detection of bad points. A family of MCGFA models is created through the introduction of different constraints on the covariance structure. A new, efficient implementation of the algorithm is presented, along with an account of its development. The fast implementation permits thorough testing of the MCGFA algorithm, and its performance is compared to two natural competitors: parsimonious Gaussian mixture models (PGMM) and mixtures of modified t factor analyzers (MMtFA). The algorithms are tested systematically on simulated and real data.

Acknowledgements

First, I thank my supervisor Dr. Paul McNicholas for his guidance in the completion of this thesis and the opportunities he has provided for me as a graduate student at McMaster. It was Dr. McNicholas who opened my eyes to the field of statistical learning, and model-based clustering and classification in particular. He is also an invaluable source of career advice, both academic and otherwise.

In addition, I thank Dr. Antonio Punzo of the University of Catania. Dr. Punzo provided feedback and testing of my computational implementations, as well as a plan for how to proceed with my research.

Dr. Narayanaswamy Balakrishnan, Associate Chair of Statistics at McMaster, helped me immensely to prepare for the M.Sc. Statistics program coming from a background in mathematics. Dr. Balakrishnan is a phenomenal resource for all things statistical and his course in Biostatistics helped peak my interest in that field.

I thank Dr. Petar Jevtic and Dr. Bartok Protas for serving on my thesis committee along with Dr. McNicholas. My defence was a very enjoyable experience due the insightful comments, corrections and discussion provided by each committee member.

My colleagues and classmates made my time as a graduate student at McMaster greatly enjoyable, and I thank them all for their support and company.

Contents

List of Tables

List of Figures

Chapter 1

Background

1.1 Finite Mixture Models

A parametric finite mixture model assumes that data arise from one of a finite set of probability distributions with density functions $\{f_1, \ldots, f_G\}$, and parameter vectors $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_G\}$. The density of such a model is written

$$p(\mathbf{x}|\boldsymbol{\Theta}, \boldsymbol{\pi}) = \sum_{g=1}^{G} \pi_g \, f_g(\mathbf{x}|\theta_g), \tag{1.1}$$

where π_g is the prior probability a given observation arises from the g-th distribution. It is often assumed that each mixture component is of the same distributional family, with only parameters varying between components, so $f_1 = \cdots = f_G = f$.

1.2 Clustering and Classification

This thesis is centred on two fundamental problems in statistical learning: *clustering* and *classification*. Each problem has a great breadth of applications across scientific

and industrial fields.

1.2.1 Clustering

Very broadly, the clustering problem is to partition a set of objects or observations into self-similar groups (called clusters), which may have some useful interpretation, or help uncover the structure of the data. Clustering is called an *unsupervised learning* problem, because no examples of correctly partitioned data are used. For a simple example of such a problem, consider a company that possesses data on many of their current or potential costumers and wishes to improve their marketing strategy. If the market could be clustered into segments of customers with similar requirements and priorities, each segment may be targeted specifically with different promotions or campaigns.

Another application in a highly active research area is the clustering of gene expression data. There are many ways clustering is applied in this context, but many approaches involve clustering genes with similar expression patterns. These related genes may contribute similarly to processes in the body and could help further understanding of related diseases or mechanisms. A survey of the application of clustering to this area is given in Jiang *et al.* (2004).

To begin to make the clustering problem statement precise, one must define exactly what a cluster is. There is no universal consensus on the best definition for a cluster, however, this thesis uses the following definition from McNicholas (2016):

> A cluster is a unimodal component within an appropriate finite mixture model.

In this definition, a model is appropriate for a given data set if it can accurately

capture the structure of the data. McNicholas (2016) provides a thorough discussion and justification of this definition and the history that led to it.

Armed with this definition, the clustering problem is to (1) determine the number and nature of clusters in the data and (2) accurately assign data into the correct clusters. Because the true number of clusters is not known, and there are no examples of correctly clustered data, it is not obvious how to test the performance of clustering algorithms. Approaches used herein are described in Chapter 4.

1.2.2 Classification

The classification problem similar to clustering, but less open ended. It also involves assigning observations to groups (or classes), but for classification the number and nature of groups is known, and examples of correctly assigned observations are available. The act of assigning an observation to some class often termed *labelling*. The set of data with known labels is referred to as the *training data*. The existence of this training data makes classification a *supervised learning* problem. The task may be stated as follows:

> Given a set of labelled training data, determine a rule to correctly assign an unlabelled data to the correct

The assignment is performed through the construction of a classification rule, based on the training data. New unlabelled data is then classified in accordance to this rule.

A well-known application is the classification of hand-written letters and digits. Given correctly-labelled examples of each glyph to learn from, the goal is to convert hand-written text into a computer-readable format as accurately as possible.

The performance of a classification algorithm can be easily measured by setting aside some labelled data as *test data*. The labels for the test data are known but hidden from the algorithm. The classification produces by the algorithm may then be compared with the true labels.

1.2.3 Full and Semi-Supervision

This thesis considers both fully supervised and semi-supervised classification. The difference is that in a fully supervised approach, only labelled data is used to construct the classification rule. Meanwhile a semi-supervised method uses both labelled and unlabelled data to train the classifier. The unlabelled training data may help uncover the structure of the data, even if it does not explicitly provide class information.

There are advantages and disadvantages to either approach. Zhu and Goldberg (2009) provide an overview and state "Semi-supervised learning ... can use readily available unlabelled data to improve supervised learning tasks." Meanwhile Cozman *et al.* (2003) caution that the use of unlabelled data can actually worsen classification performance in the presence of model error.

1.3 Model-Based Clustering and Classification

The definition of clustering given in the previous section naturally to a mixture-model based clustering paradigm. With only slight modifications, the same approach can be applied to classification, even though the concept of a mixture model is not needed to define the classification problem. In both cases, each mixture component is viewed as a different group, or subpopulation, or class, or cluster, in the data.

For clustering, the method must allow one to determine the appropriate number of components using only the data. Several mixture models are fit to the data via maximum likelihood estimation, and some model selection procedure is used to determine the best model for the data. This selection procedure thus determines the number of clusters. After parameter estimate and model selection, each observation $\mathbf{x}_i$ is assigned to group g, where g maximizes

$$\hat{z}_{ig} = \frac{f(\mathbf{x}_i|\hat{\boldsymbol{\theta}}_g)}{\sum_{j=1}^{G} f(\mathbf{x}_i|\hat{\boldsymbol{\theta}}_j)}. \tag{1.2}$$

This is known as maximum *a posteriori* (MAP) classification. An advantage of the model-based clustering approach is that each cluster assignment has a natural measure of its degree of certainty. The closer $\hat{z}_{ig}$ is to 1, the more likely it is that $\mathbf{x}_i$ belongs to group g.

The procedure for classification is similar. In a fully supervised approach, the model is fit to the labelled data only, conditioned on the training set membership information. Then, once the model parameters are estimated, the unlabelled data can be classified by MAP classification just as in the clustering case.

In a semi-supervised approach, the model is fit to all of the available data, labelled or otherwise. Thus the unlabelled data contribute information to parameter estimation. Again, once the model is fit, the unlabelled data is classified by MAP.

1.3.1 Gaussian Mixtures Modelling and Alternatives

Gaussian mixture modelling is a well-known approach to model-based clustering. It is assumed that each mixture component is multivariate Gaussian, with parameter

vectors $\{\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g\}_{g=1}^G$. This model is popular due to its familiarity, mathematical convenience, and computational efficiency.

However, as it is built upon the Gaussian distribution, the Gaussian mixture model is not robust to outliers or asymmetry. So, it is common to substitute heavier-tailed or skewed distributions for the Gaussian. Examples include mixtures of Student's t distributions (Peel and McLachlan, 2000), mixtures of skew normal distributions (Lin *et al.*, 2007), and mixtures of generalized hyperbolic distributions (Browne and McNicholas, 2015). A recent and comprehensive treatment of finite mixture modelling is provided by McNicholas (2016).

Additionally, the general Gaussian mixture model is highly parameterized. If the dimensionality of the data is p, the number of parameters grows as p^2. This may lead to overfitting, and heavy computation. In the following section, a model is introduced that is used to control the number of parameters through dimensionality reduction.

1.4 Mixture of Factor Analyzers Model

The factor analysis model originated with Spearman (1904) as a two-factor model of intelligence. In its modern incarnation, this model views p-variate data as arising as a linear combination of q independent standard Gaussian latent (unobserved) factors, where $q < p$.

Given p-dimensional data vectors $\mathbf{X}_1, \ldots, \mathbf{X}_N$, the model assumes that

$$\begin{aligned} \mathbf{X}_i &= \boldsymbol{\mu} + \boldsymbol{\Lambda}\mathbf{U}_i + \boldsymbol{\epsilon}_i \\ \boldsymbol{\epsilon}_i &\sim N_p(\mathbf{0}, \boldsymbol{\Psi}) \\ \boldsymbol{\Psi} &= \text{diag}(\psi_1, \ldots, \psi_p), \end{aligned} \tag{1.3}$$

where $\boldsymbol{\mu}$ is the mean, and $\mathbf{U}_i \sim N(0, \mathbf{I}_q)$ is a q-dimensional vector of latent factors. The $\mathbf{U}_i$ are assumed to be mutually independent and independent from the $\boldsymbol{\epsilon}_i$, which are also independent from one another. Here $\boldsymbol{\Lambda}$ is a $p \times q$ matrix of *factor loadings*, which determine how the q factors are combined to produce the p observed variables. The random variable $\boldsymbol{\epsilon}_i$ represents the error related to the combination, and the diagonal matrix $\boldsymbol{\Psi}$ determines the error variance related to each of the p variables. It follows that

$$\mathbf{X}_i \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \boldsymbol{\Psi}) = N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{1.4}$$

It should be noted that the model is invariant to rotations in the factor loading matrix $\boldsymbol{\Lambda}$; that is, if $\mathbf{Q}$ is any orthogonal matrix and $\boldsymbol{\Lambda}$ is replaced by $\boldsymbol{\Lambda}\mathbf{Q}$, then $\boldsymbol{\Lambda}\mathbf{Q}(\boldsymbol{\Lambda}\mathbf{Q})' = \boldsymbol{\Lambda}\boldsymbol{\Lambda}'$. However, this is only an issue when interpreting the entries of the loading matrix. In the context of clustering and classification, the factor analysis model is used as a tool for dimensionality reduction and so this is not a problem.

Introduced by Ghahramani and Hinton (1997), the mixture of factor analyzers (MFA) model is a Gaussian mixture model in which each mixture component is itself a factor analysis model. In its most general form, each of the parameters $\boldsymbol{\mu}$, $\boldsymbol{\Lambda}$, and $\boldsymbol{\Psi}$ may vary between the mixture components. The model assumption in this case is that given an observation $\mathbf{X}_i$,

$$\mathbf{X}_i = \mu_g + \Lambda_g \mathbf{U}_{ig} + \boldsymbol{\epsilon}_{ig} \tag{1.5}$$

with probability π_g, for $g = 1, \ldots, G$. As usual, π_g is the prior probability of membership to the g-th component. Thus, the latent factors may be combined differently in different regions of the sample space, allowing local dimensionality reduction. Additionally, components may have different error variances. In other words, different

subpopulations can have unique covariance structures. This is why the MFA model is favoured over a two-step approach of global dimensionality reduction, followed by clustering.

In the original formulation of Ghahramani and Hinton (1997), the error variance matrix is fixed across components, i.e. $\mathbf{\Psi}_1 = \cdots = \mathbf{\Psi}_G$. This form of the model is consistent with the interpretation of $\mathbf{\Psi}$ as sensor noise that affects all observations equally. Ghahramani and Hinton (1997) note that this assumption is not necessary and indeed McLachlan *et al.* (2003) consider the general MFA model where $\mathbf{\Psi}$ is allowed to vary between groups. Meanwhile the probabilistic principal components analysis model of Tipping and Bishop (1999) is a special case of MFA that allows the noise variance to vary between groups, but assumes that the noise is isotropic within groups, i.e. $\mathbf{\Psi}_g = \mathbf{I}\psi_g$.

1.5 Parsimonious Gaussian Mixture Models

McNicholas and Murphy (2008) go on to extend and unify the different approaches to MFA with a family of eight parsimonious Gaussian mixture models (PGMM). First the authors suggest that for greater parsimony, the factor loading matrices may also be constrained across groups. This constraint prevents local dimensionality reduction, but if the mixture components indeed share similar covariance structures, provides a simpler model and greater stability for parameter estimation. Together with the two possible constraints on $\mathbf{\Psi}$ discussed above, this gives three possible constraints for the MFA model:

1. Loading matrices constrained across groups, $\mathbf{\Lambda}_1 = \cdots = \mathbf{\Lambda}_G = \mathbf{\Lambda}$

2. Error variance matrices constrained across groups, $\boldsymbol{\Psi}_1 = \cdots = \boldsymbol{\Psi}_G = \boldsymbol{\Psi}$

3. Isotropic errors within groups $\boldsymbol{\Psi}_g = \mathbf{I}\psi_g$

Each constraint may be applied or not, independently of the other two, yielding the eight models. The models are named in three letter codes where U indicates unconstrained and C indicates constrained. The full PGMM family of models is presented in Table 1.1, along with their number of covariance parameters.

Table 1.1: Eight parsimonious covariance structures of the PGMM family.

$\boldsymbol{\Lambda}_g = \boldsymbol{\Lambda}$	$\boldsymbol{\Psi}_g = \boldsymbol{\Psi}$	$\boldsymbol{\Psi}_g = \mathbf{I}\psi_g$	# Cov. Parameters
C	C	C	$pq - q(q-1)/2 + 1$
C	C	U	$pq - q(q-1)/2 + p$
C	U	C	$pq - q(q-1)/2 + G$
C	U	U	$pq - q(q-1)/2 + Gp$
U	C	C	$G[pq - q(q-1)/2] + 1$
U	C	U	$G[pq - q(q-1)/2] + p$
U	U	C	$G[pq - q(q-1)/2] + G$
U	U	U	$G[pq - q(q-1)/2] + Gp$

The MFA model of Ghahramani and Hinton (1997) corresponds to UCU, the more general MFA model of McLachlan *et al.* (2003) is the fully unconstrained model UUU, and probabilistic principal components analysis model of Tipping and Bishop (1999) is UUC.

McNicholas and Murphy (2010) introduce the expanded parsimonious Gaussian mixture models (EPGMM) family. EPGMM extends PGMM to 12 models by reparameterizing the error variance matrix as $\boldsymbol{\Psi}_g = \omega_g \boldsymbol{\Delta}_g$, where ω_g is a positive real number and $\boldsymbol{\Delta}_g$ is a diagonal matrix with determinant 1. The resulting covariance

structure

$$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \omega_g \boldsymbol{\Delta}_g \,, \tag{1.6}$$

is termed the modified factor analysis covariance structure. The matrix $\boldsymbol{\Delta}_g$ determines the relative size of error variance across each variable in group g, while the values $(\omega_1, \ldots, \omega_G)$ represent the overall amount of error in each group. So, some subset of the following four constraints may be introduced on to a mixture of modified factor analyzers:

1. Loading matrices constrained across groups: $\boldsymbol{\Lambda}_1 = \cdots = \boldsymbol{\Lambda}_G$
2. Relative error variances constrained across groups: $\boldsymbol{\Delta}_1 = \cdots = \boldsymbol{\Delta}_G$
3. Overall error variance constrained across groups: $\omega_1 = \cdots = \omega_G$.
4. Isotropic errors within groups $\boldsymbol{\Delta}_g = \mathbf{I}_p$

Ignoring equivalent cases, this leads to the 12 EPGMM models presented in Table 1.2. Where applicable, the equivalent PGMM model is indicated. The EPGMM model is implemented in the `pgmm` package (McNicholas *et al.*, 2015) for the R statistical programming language (R Core Team, 2016). Additionally, a parallel implementation of PGMM is presented by McNicholas *et al.* (2010).

Table 1.2: The twelve parsimonious covariance structures of the EPGMM family, with PGMM equivalencies.

$\boldsymbol{\Lambda}_g = \boldsymbol{\Lambda}$	$\boldsymbol{\Delta}_g = \boldsymbol{\Delta}$	$\omega_g = \omega$	$\boldsymbol{\Delta}_g = \mathbf{I}_p$	PGMM Equivalent	Covariance Structure
C	C	C	C	CCC	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \omega\mathbf{I}_p$
C	C	U	C	CUC	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \omega_g\mathbf{I}_p$
U	C	C	C	UCC	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \omega\mathbf{I}_p$
U	C	U	C	UUC	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \omega_g\mathbf{I}_p$
C	C	C	U	CCU	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \omega\boldsymbol{\Delta}$
C	C	U	U	-	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \omega_g\boldsymbol{\Delta}$
U	C	C	U	UCU	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \omega\boldsymbol{\Delta}$
U	C	U	U	-	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \omega_g\boldsymbol{\Delta}$
C	U	C	U	-	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \omega\boldsymbol{\Delta}_g$
C	U	U	U	CUU	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \omega_g\boldsymbol{\Delta}_g$
U	U	C	U	-	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \omega\boldsymbol{\Delta}_g$
U	U	U	U	UUU	$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \omega_g\boldsymbol{\Delta}_g$

1.6 The Mixtures of Modified t Factor Analyzers Model

Despite its advantages, the EPGMM family of models is still built upon the Gaussian distribution and thus may not be sufficiently robust for some applications. To address this concern, Andrews and McNicholas (2011b) introduce the mixtures of modified t-factor analyzers (MMTFA) model, which generalizes EPGMM by utilizing the multivariate t distribution in place of the Gaussian. The heavier-tailed t distribution is better able to account for outlying points, but it is still an elliptical distribution.

The model of Andrews and McNicholas (2011b) builds on the work of several authors. McLachlan and Peel (1998) originally introduce the mixtures of multivariate

t-distributions model. McLachlan *et al.* (2007) who extend the mixture of factor analyzers model to incorporate the t-mixture models. Andrews and McNicholas (2011a) then develop the mixture of modified t factor analyzers model of McLachlan *et al.* (2007) into a family of six models. The six models are generated by combining constraints on the factor loading matrix $\mathbf{\Lambda}_g$, the error variance matrix $\mathbf{\Psi}_g$, and the degree of freedom parameter of the multivariate t-distribution, ν_g. To be precise, each of these parameters may be constrained to be equal across groups or free to vary.

The MMTFA family of Andrews and McNicholas (2011b) uses the same covariance constraints as the EPGMM, along with the option of an additional constraint on ν_g. This yields a family of 24 models. The MMTFA model is implemented in the `mmtfa` package (Andrews *et al.*, 2015) for R.

Chapter 2

The MCGFA Model

2.1 The Contaminated Gaussian Distribution

The contaminated Gaussian distribution (CGD) is an elliptical, heavy-tailed generalization of the Gaussian distribution. In this way it is similar to the multivariate t distribution; in fact both are Gaussian scale mixture models (Punzo and McNicholas, 2014). However, unlike the t distribution, the CGD explicitly breaks observations into two classes, automatically identifying outlying points.

The CGD is a mixture of two Gaussian components, with the same mean and proportional covariance matrices. With a high probability α, an observation will be "good" and drawn from a Gaussian distribution with covariance matrix $\boldsymbol{\Sigma}$. Otherwise it will be a "bad" observation drawn from a distribution with covariance $\eta\boldsymbol{\Sigma}$, where $\eta > 1$. (The factor η must be strictly greater than one for the identifiability of the GCD model.) Throughout the rest of this thesis, the word "bad" will be used as a blanket term for outliers, noise or spurious observations—any type of atypical observation that may affect the performance of a clustering algorithm.

Because at least half of the data must be "typical", α is constrained to lie in the range $(0.5, 1)$. In practice, this α may be constrained to be even greater. Succinctly, if $\mathbf{X}$ is a contaminated Gaussian random variable with parameters $\{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \alpha, \eta\}$, then

$$\begin{aligned} \mathbf{X} &\sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \text{w.p.} \quad \alpha \\ \mathbf{X} &\sim N(\boldsymbol{\mu}, \eta\boldsymbol{\Sigma}) \quad \text{w.p.} \quad 1-\alpha\,. \end{aligned} \tag{2.1}$$

The density of a contaminated Gaussian random variable with parameters $(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \alpha, \eta)$ is thus

$$p_{CG}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \alpha, \eta) = \alpha\phi(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) + (1-\alpha)\phi(\mathbf{x}; \boldsymbol{\mu}, \eta\boldsymbol{\Sigma}), \tag{2.2}$$

where $\phi(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the usual multivariate Gaussian density. The contaminated Gaussian is indeed a direct generalization of the Gaussian distribution, because if at least one of α or η tend to 1, then $\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

2.2 Probabilistic Model

The factor analysis model may be generalized by introducing a contaminated Gaussian distribution in place of the usual Gaussian density. Following Punzo and McNicholas (2014), the modelling assumption is that given a data vector $\mathbf{X}_i$ and latent factor $\mathbf{U}_i$

$$\begin{pmatrix} \mathbf{X}_i \\ \mathbf{U}_i \end{pmatrix} \sim \text{CN}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*, \alpha, \eta), \tag{2.3}$$

where

$$\boldsymbol{\mu}^* = \begin{pmatrix} \boldsymbol{\mu} \\ \mathbf{0} \end{pmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}^* = \begin{pmatrix} \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \boldsymbol{\Psi} & \boldsymbol{\Lambda} \\ \boldsymbol{\Lambda}' & \mathbf{I} \end{pmatrix}. \tag{2.4}$$

Punzo and McNicholas (2014) note that unlike in that Gaussian factor analyzer case, "the factors $[\mathbf{U}_i]$ and error terms $[\boldsymbol{\epsilon}_i]$ are no longer independently distributed ... however, they are uncorrelated."

Assuming this distribution in each mixture component then generalizes the MFA model to yield the mixtures of contaminated Gaussian factor analyzers (MCGFA) model. The most general MCGFA model has density

$$f(\mathbf{x}) = \sum_{g=1}^{G} \pi_g \, p_{CG}(\mathbf{x} \mid \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g, \alpha_g, \eta_g), \tag{2.5}$$

where

$$\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \boldsymbol{\Psi}_g \,, \tag{2.6}$$

and f_{CG} is the contaminated Gaussian density as defined in Equation 2.2.

As with the MFA model, different constraints can be imposed on the covariance structure of the MCGFA model. In this thesis, MCGFA is extended to a family of eight models completely analogous to the eight PGMM models of McNicholas and Murphy (2008). This family combines the constraints $\boldsymbol{\Lambda}_g = \boldsymbol{\Lambda}$, $\boldsymbol{\Psi}_g = \cdots = \boldsymbol{\Psi}$ and $\boldsymbol{\Psi}_g = \psi_g \mathbf{I}$.

2.2.1 Model Selection

When fitting several MCGFA models to the same data, the Bayesian information criterion (BIC; Schwarz, 1978) is used to select the number of groups G, the number of latent factors q, and the covariance structure of the model. The BIC is defined as follows:

$$\text{BIC} = 2l(\mathbf{x}, \hat{\boldsymbol{\vartheta}}) - k \log n \,, \tag{2.7}$$

where $l(\mathbf{x}, \hat{\boldsymbol{\vartheta}})$ is the log-likehood of the data given the current parameter estimates $\hat{\boldsymbol{\vartheta}}$, k is the number of free parameters in the model, and n is the number of observations.

2.3 Parameter Estimation Algorithms

The expectation-maximization (EM) algorithm, formulated by Dempster *et al.* (1977), is a widely used iterative method for maximum likelihood parameter estimation in the presence of latent variables or missing data. The EM algorithm alternates between the *expectation step* (E-step), in which the expected log-likelihood of the data is calculated based on the current model parameter estimates, and the *maximization step* (M-step), in which that likelihood is maximized with respect to the model parameters.

The EM algorithm is commonly applied in the clustering and classification context by viewing component membership as missing data. In the mixture modelling context, the E-step generally involves updating the expected group membership of each observation, based on the current distributional parameter estimates, and the M-step involves updating the parameter estimates based on the expected group memberships.

The expectation-conditional maximization (ECM) algorithm (Meng and Rubin, 1993) is a variation on the EM algorithm in which the M-step is broken down into computationally simpler steps. It was developed as an alternative to the EM algorithm when maximum likelihood estimation of the complete-data log-likelihood is complicated in its own right. The M-step is replaced by several simpler conditional-maximization steps on different subsets of the parameters. Meng and Rubin (1993) show that the ECM algorithm shares all of the important convergence properties of the EM algorithm.

The alternating expectation-conditional maximization (AECM) algorithm (Meng

and Van Dyk, 1997) is yet another EM variant that incorporates two alternating cycles of the ECM algorithm. Each cycle defines the complete data and missing data differently.

2.4 AECM Algorithm for the MCGFA Model

The MCGFA model parameters are estimated using maximum likelihood estimation. Given data $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the underlying optimization problem is to find the set of parameters $\{\pi_g, \mu_g, \boldsymbol{\Lambda}_g, \boldsymbol{\Psi}_g, \alpha_g, \eta_g\}$ that maximize the likelihood function:

$$\mathcal{L}(\{\pi_g, \boldsymbol{\mu}_g, \boldsymbol{\Lambda}_g, \boldsymbol{\Psi}_g, \alpha_g, \eta_g\}_{g=1}^{G} \mid \mathbf{X}) = \prod_{i=1}^{N} \sum_{g=1}^{G} \pi_g \, p_{CG}(\mathbf{x}_i \mid \boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g, \alpha_g, \eta_g) \,, \tag{2.8}$$

where $\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \boldsymbol{\Psi}_g$. For computational convenience, the *log-likelihood* (logarithm of the likelihood function), is maximized instead.

The AECM algorithm introduced in Section 2.3 is used to iteratively find the maximum likelihood parameters for the MCGFA model. Three notations are introduced to encode missing data. The variables $\{z_{ig}\}$ indicate group membership: $z_{ig} = 1$ if observation i is in group g and $z_{ig} = 0$ otherwise. The status of each observation as a typical or "good" point versus an outlier or "bad" point is represented by the random variables v_{ig}: if observation i in group g is good and $v_{ig} = 0$ if observation i in group g is bad. These random variables are collected into the matrices $\mathbf{Z} = \{z_{ig}\}$ and $\mathbf{V} = \{v_{ig}\}$. Finally, an observation $\mathbf{x}_i$ has a corresponding q-dimensional realization of latent factors, $\boldsymbol{u}_{ig}$ for each of the G factor analysis models in the mixture. Denote the collection of all realizations of the latent factors $\mathcal{U} = \{\boldsymbol{u}_{ig}\}$.

In the first AECM cycle, the complete data is $\{\mathbf{X}, \mathbf{Z}, \mathbf{V}\}$ with missing data

$\{\mathbf{Z}, \mathbf{V}\}$, and in the second cycle, the complete data is $\{\mathbf{X}, \mathbf{Z}, \mathbf{V}, \mathcal{U}\}$ with missing data $\{\mathbf{Z}, \mathbf{V}, \mathbf{U}\}$. The model parameters $\boldsymbol{\vartheta}$ are partitioned as $\boldsymbol{\vartheta} = \{\vartheta_1, \vartheta_2\}$, where

$$\begin{aligned} \boldsymbol{\vartheta}_1 &= \{\pi_g, \boldsymbol{\mu}_g, \alpha_g, \eta_g\}_{g=1}^G \\ \boldsymbol{\vartheta}_2 &= \{\boldsymbol{\Lambda}_g, \boldsymbol{\Psi}_g\}_{g=1}^G \ . \end{aligned} \tag{2.9}$$

The first cycle updates $\boldsymbol{\vartheta}_1$ given $\boldsymbol{\vartheta}_2$, and the second vice-versa.

2.4.1 First cycle

In the first cycle, the missing data are $\{\mathbf{Z}, \mathbf{V}\}$, and the complete data are $\{\mathbf{X}, \mathbf{Z}, \mathbf{V}\}$. In this cycle, the proportion, location and contamination parameters $\{\pi_g, \boldsymbol{\mu}_g, \alpha_g, \eta_g\}_{g=1}^G$ are updated, given the covariance parameters. Because the covariance parameters are fixed, the entire first cycle is identical for each of the eight parsimonious models.

Following Punzo and McNicholas (2014), the complete-data log-likelihood is

$$l_{1c}\left(\boldsymbol{\vartheta}_1\right) = l_{1c_1}\left(\{\pi_g\}_{g=1}^G\right) + l_{1c_2}\left(\{\alpha_g\}_{g=1}^G\right) + l_{1c_3}\left(\{\boldsymbol{\mu}_g, \eta_g\}_{g=1}^G\right) , \tag{2.10}$$

where

$$\begin{aligned} l_{1c_1}\left(\{\pi_g\}_{g=1}^G \mid \mathbf{X}, \mathbf{Z}, \mathbf{V}\right) &= \sum_{i=1}^n \sum_{g=1}^G z_{ig} \log\left(\pi_g\right) \\ l_{1c_2}\left(\{\alpha_g\}_{g=1}^G \mid \mathbf{X}, \mathbf{Z}, \mathbf{V}\right) &= \sum_{i=1}^n \sum_{g=1}^G z_{ig} \left[v_{ig} \log\left(\alpha_g\right) + \left(1 - v_{ig}\right) \log(1 - \alpha_g)\right] \\ l_{1c_3}\left(\{\boldsymbol{\mu}_g, \eta_g\}_{g=1}^G \mid \mathbf{X}, \mathbf{Z}, \mathbf{V}\right) &= -\frac{1}{2} \sum_{i=1}^n \sum_{g=1}^G \Bigg[z_{ig} \log\left|\boldsymbol{\Sigma}_g\right| + z_{ig}(1 - v_{ig}) \log\left(\eta_g\right) \\ &\qquad + z_{ig} \left(v_{ig} - \frac{1 - v_{ig}}{\eta_g}\right) \left(\mathbf{x}_i - \boldsymbol{\mu}_g\right)' \left(\boldsymbol{\Sigma}_g\right)^{-1} \left(\mathbf{x}_i - \boldsymbol{\mu}_g\right) \Bigg], \end{aligned} \tag{2.11}$$

with $\boldsymbol{\Sigma}_g = \boldsymbol{\Lambda}_g \boldsymbol{\Lambda}_g' + \boldsymbol{\Psi}_g$.

E-step

In the E-step, it suffices to replace $\mathbf{Z}$ and $\mathbf{V}$ by their expected values to obtain the expected complete-data log-likelihood $\mathrm{E}_{\boldsymbol{\vartheta}^{(k)}}(l_{1c} \mid \mathbf{X},)$:

$$z_{ig}^{(k)} = \mathrm{E}_{\vartheta^{(k)}}(Z_{ig} \mid \mathbf{x}_i) = \frac{p_{CN}\left(\mathbf{x}_i; \boldsymbol{\mu}_g^{(k)}, \boldsymbol{\Sigma}_g^{(k)}, \alpha_g^{(k)}, \eta_g^{(k)}\right)}{\sum_{h=1}^{G} p_{CN}\left(\mathbf{x}_i; \boldsymbol{\mu}_h^{(k)}, \boldsymbol{\Sigma}_h^{(k)}, \alpha_h^{(k)}, \eta_h^{(k)}\right)} \tag{2.12}$$

$$v_{ig}^{(k)} = \mathrm{E}_{\vartheta^{(k)}}(V_{ig} \mid \mathbf{x}_i) = \frac{\alpha_g^{(k)} \phi\left(\mathbf{x}_i; \boldsymbol{\mu}_g^{(k)}, \boldsymbol{\Sigma}_g^{(k)}\right)}{p_{CN}\left(\mathbf{x}_i; \boldsymbol{\mu}_g^{(k)}, \boldsymbol{\Sigma}_g^{(k)}, \alpha_g^{(k)}, \eta_g^{(k)}\right)}, \tag{2.13}$$

where V_{ig} is the random variable related to v_{ig}.

CM-step 1

The parameters $\{\pi_g, \boldsymbol{\mu}_g, \alpha_g\}_{g=1}^{G}$ are updated, to maximize $\mathbf{E}\left(l_{1c} \mid \mathbf{X}, \boldsymbol{\vartheta}^{(k)}\right)$:

$$\begin{aligned}
\pi_g^{(k+1)} &= n_g^{(k)}/n, \\
\alpha_g^{(k+1)} &= \max\left\{\frac{1}{n_g^{(k)}} \sum_{i=1}^{n} z_{ig}^{(k)} v_{ig}^{(k)}, \alpha_{\min}\right\}, \\
\boldsymbol{\mu}_g^{(k+1)} &= \frac{\sum_{i=1}^{n} z_{ig}^{(k)} \left(v_{ig}^{(k)} + \dfrac{1 - v_{ig}^{(k)}}{\eta_g^{(k)}}\right) \mathbf{x}_i}{\sum_{i=1}^{n} z_{ig}^{(k)} \left(v_{ig}^{(k)} + \dfrac{1 - v_{ig}^{(k)}}{\eta_g^{(k)}}\right)},
\end{aligned}$$

where $n_g^{(k)} = \sum_{i=1}^{n} z_{ig}^{(k)}$.

The update for $\boldsymbol{\mu}_g$ gives an example of how the MCGFA model avoids fitting to noise and outliers and provides greater robustness as compared to the PGMM model. Rather updating the group mean to the usual average value, it is updated to

a weighted average with weights

$$w_{ig}^{(k)} = v_{ig}^{(k)} + \frac{1 - v_{ig}^{(k)}}{\eta_g^{(k)}}. \tag{2.14}$$

Intuitively, if the i-th observation is considered to be a likely bad point in the g-th group, v_{ig} is small, and w_{ig} tends to $1/\eta_g$. So, the larger the estimate of the covariance inflation factor for the g-th group, η_g, the smaller the contribution of the i-th observation to the estimate of $\boldsymbol{\mu}_g$. Thus outlying points have less chance of greatly affecting the estimates of groups means.

CM-step 2

Because no analytical maximum exists, $\{\eta_g\}_{g=1}^G$ are updated numerically to maximize $\mathbf{E}\left(l_{1c} \mid \mathbf{X}, \boldsymbol{\vartheta}_1^{(k+1)}, \boldsymbol{\vartheta}_2^{(k)}\right)$ given the previous updates. This amounts to maximizing

$$l_{1c_3}\left(\left\{\boldsymbol{\mu}_g^{(k+1)}, \eta_g\right\} \mid \mathbf{X}, \mathbf{Z}^{(k+1)}, \mathbf{V}^{(k+1)}\right) \tag{2.15}$$

from (2.11), over the interval $[1, \eta_{\max}]$.

2.4.2 Second cycle

The second cycle of the AECM algorithm treats $\{\mathbf{Z}, \mathbf{V}, \mathcal{U}\}$ as the missing data and $\{\mathbf{X}, \mathbf{Z}, \mathbf{V}, \mathcal{U}\}$ as the complete data. It updates the covariance parameters $\{\boldsymbol{\Sigma}_g, \boldsymbol{\Psi}_g\}_{g=1}^G$ given all others. The details of the updates for this cycle differ between the eight parsimonious models, but are almost identical to the updates for the AECM algorithm for the PGMM model, found in full in Section 3 and Appendix A of McNicholas and

Murphy (2008). The only difference is that the matrix $\boldsymbol{S}$ from PGMM:

$$\boldsymbol{S}_g^{(k+1)} = \frac{1}{n_g}\sum_{i=1}^{n} z_{ig}\left(\mathbf{x}_i - \boldsymbol{\mu}_g^{(k+1)}\right)\left(\mathbf{x}_i - \boldsymbol{\mu}_g^{(k+1)}\right)', \tag{2.16}$$

is adjusted to down-weight observations that are likely to be bad:

$$\boldsymbol{S}_g^{(k+1)} = \frac{1}{n_g}\sum_{i=1}^{n} z_{ig}\left(v_{ig} + \frac{1-v_{ig}}{\eta_g^{(k+1)}}\right)\left(\mathbf{x}_i - \boldsymbol{\mu}_g^{(k+1)}\right)\left(\mathbf{x}_i - \boldsymbol{\mu}_g^{(k+1)}\right)'. \tag{2.17}$$

The complete-data log-likelihood in this cycle is

$$\begin{aligned} l_{2c}\left(\vartheta_2\right) = C + \sum_{g=1}^{G}\Bigg\{ & -\frac{n_g}{2}\log\left|\boldsymbol{\Psi}_g\right| - \frac{n_g}{2}\mathrm{tr}\left(\boldsymbol{\Psi}_g^{-1}\boldsymbol{S}_g^{(k+1)}\right) \\ & + \sum_{i=1}^{n} z_{ig}\left(v_{ig} + \frac{1-v_{ig}}{\eta_g^{(k+1)}}\right)\left(\mathbf{x}_i - \boldsymbol{\mu}_g^{(k+1)}\right)'\boldsymbol{\Psi}_g^{-1}\boldsymbol{\Lambda}_g\boldsymbol{u}_{ig} \\ & - \frac{1}{2}\mathrm{tr}\left[\boldsymbol{\Lambda}_g'\boldsymbol{\Psi}_g^{-1}\boldsymbol{\Lambda}_g\sum_{i=1}^{n} z_{ig}\left(v_{ig} + \frac{1-v_{ig}}{\eta_g^{(k+1)}}\right)\boldsymbol{u}_{ig}\boldsymbol{u}_{ig}'\right]\Bigg\}, \end{aligned}$$

where $n_g = \sum_{i=1}^{n} z_{ig}$, and C is constant respect to $\boldsymbol{\vartheta}_2$.

E-step

The E-step on the second cycle involves the calculation of the expectation of l_{2c} given $\mathbf{X}$ and the parameter updates so far: $\vartheta^{(k+1/2)} = \left\{\vartheta_1^{(k+1)}, \vartheta_2^{(k)}\right\}$. The group membership indicators z_{ig} and v_{ig} are updated exactly as in (2.12) and (2.13) of the E-step in the first cycle and denoted by $z_{ig}^{(k+1/2)}$ and $v_{ig}^{(k+1/2)}$. Following Punzo and McNicholas (2014), the expected complete-data log-likelihood for the second cycle is

$$\begin{aligned} \mathrm{E}_{\boldsymbol{\vartheta}^{(k)}}\left(l_{2c} \mid \mathbf{X}\right) = C + \sum_{g=1}^{G} n_g^{(k+1/2)}\Bigg\{ & \frac{1}{2}\log\left|\boldsymbol{\Psi}_g^{-1}\right| - \frac{1}{2}\mathrm{tr}\left(\boldsymbol{\Psi}_g^{-1}\boldsymbol{S}_g^{(k+1)}\right) \\ & + \mathrm{tr}\left(\boldsymbol{\Psi}_g^{-1}\boldsymbol{\Lambda}_g\boldsymbol{\beta}_g^{(k)}\boldsymbol{S}_g^{(k+1)}\right) - \frac{1}{2}\mathrm{tr}\left[\boldsymbol{\Lambda}_g'\boldsymbol{\Psi}_g^{-1}\boldsymbol{\Lambda}_g\boldsymbol{\Theta}_g^{(k+1/2)}\right]\Bigg\}, \end{aligned}$$

where C is a constant with respect to $\boldsymbol{\vartheta}_2$ and $\boldsymbol{\Theta}_g^{(k+1/2)} = \mathbf{I}_q - \boldsymbol{\beta}_g^{(k)}\boldsymbol{\Lambda}_g^{(k)} + \boldsymbol{\beta}_g^{(k)}\boldsymbol{S}_g^{(k+1)}\boldsymbol{\beta}_g^{(k)'}$.

CM-step

The CM-step of the second cycle is identical to that of PGMM, with $\boldsymbol{S}$ modified as in Equation 2.17. The second term from Equation 2.11, $l_{2c}\left(\vartheta_2\right)$, is maximized with respect to ϑ_2. The resulting updates for ϑ_2 are

$$\begin{aligned} \boldsymbol{\Lambda}_g^{(k+1)} &= \boldsymbol{S}_g^{(k+1)}\boldsymbol{\beta}_g^{(k)'}\left(\boldsymbol{\Theta}_g^{(k+1/2)}\right)^{-1} \\ \boldsymbol{\Psi}_g^{(k+1)} &= \operatorname{diag}\left(\boldsymbol{S}_g^{(k+1)} - \boldsymbol{\Lambda}_g^{(k+1)}\boldsymbol{\beta}_g^{(k)}\boldsymbol{S}_g^{(k+1)}\right). \end{aligned} \tag{2.18}$$

2.4.3 The Woodbury Identity

The updates of $\boldsymbol{z}$, $\boldsymbol{v}$ and η require the inversion of the $p \times p$ covariance matrix $\boldsymbol{\Sigma}_g^{-1}$ for each group—a costly operation. Fortunately, the special form of the matrix $\boldsymbol{\Sigma} = (\boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \boldsymbol{\Psi}_g)^{-1}$ allows the application of the Woodbury identity (Woodbury, 1950):

$$(\boldsymbol{\Lambda}_g\boldsymbol{\Lambda}_g' + \boldsymbol{\Psi}_g)^{-1} = \boldsymbol{\Psi}_g^{-1} - \boldsymbol{\Psi}_g^{-1}\boldsymbol{\Lambda}_g\left[\mathbf{I_p} + \boldsymbol{\Lambda}_g'\boldsymbol{\Psi}_g^{-1}\boldsymbol{\Lambda}_g\right]^{-1}\boldsymbol{\Lambda}_g'\boldsymbol{\Psi}_g^{-1}. \tag{2.19}$$

Here only the diagonal matrix $\boldsymbol{\Psi}$ and the $q \times q$ matrix $\mathbf{I} + \boldsymbol{\Lambda}'\boldsymbol{\Psi}^{-1}\boldsymbol{\Lambda}$ requires inversion. This trick is commonly applied in fitting algorithm for mixtures of factor analyzer models. It was noted in the first presentation of the AECM for Gaussian factor analyzers in Ghahramani and Hinton (1997), and applied in McNicholas and Murphy (2008), and Andrews and McNicholas (2011b).

2.4.4 Convergence Criterion

The stopping time for the AECM algorithm is determined using the Aitken Acceleration (Aitken, 1926), one of several methods to determine termination of iterative

likelihood-based model-fitting algorithms. The Aitken acceleration at the k-th iteration is defined as follows:

$$a^{(k)} = \frac{l^{(k+1)} - k^{(k)}}{l^{(k)} - l^{(k-1)}}, \tag{2.20}$$

where $l^{(k)}$ is the observed log-likelihood at iteration k. As per Böhning *et al.* (1994), the asymptotic estimate of the log-likelihood at iteration $k+1$ is

$$l_{\infty}^{(k+1)} = l^{(k+1)} + \frac{l^{(k+1)} - l^{(k)}}{1 - a^{(k+1)}}. \tag{2.21}$$

As proposed in McNicholas *et al.* (2010), the AECM algorithm terminates when

$$l_{\infty}^{(k+1)} - l^{(k)} < \epsilon \tag{2.22}$$

for a given convergence tolerance ϵ. For the implementation herein, ϵ is a user-specified constant, takes on default value 0.001.

Chapter 3

Implementation

The model fitting algorithm for MCGFA is implemented using a combination of the R and C programming languages. The user interface and model selection code is written in R, with the computationally intensive parameter estimation in C. This combination provides an interface familiar to statisticians, with maximum computational efficiency.

A functioning R version of the MCGFA fitting algorithm existed as a reference for testing, but significant effort is made to improve upon the efficiency. A significant amount of code is adapted from existing C code from the `pgmm` package for R, due to the similarities in the models and parameter estimation procedures.

3.1 Parallelization

Parallel processing is a method of computation where a large task is split up and executed simultaneously on separate processors or "cores". Because most computers today have several processors, this allows a speed up even on consumer level electronics. Dedicated high-performance computing "clusters", tightly connected computers

that work together, may have hundreds of cores. A broad introduction to parallel computing is provided by Barney (2016).

A common model of parallel computation is the master/worker model, in which one master core distributes tasks to the other worker cores, and combines their results. Often master core is also able to perform computation while it is not busy distributing work to the workers, allowing the potential speedup to be close to the number of cores.

Fitting numerous models independently and then selecting the model according to some criterion is a "trivially parallelizable" task. Each worker is able to fit a model with no communication with others, and after fitting is complete the master can simply select the model with the highest BIC.

There are different schemes for distributing tasks to the workers. Dynamic balancing is contrasted with static load balancing as follows. Consider the computation of 100 tasks using 10 parallel processors. A static load balancing scheme assigns 10 of these tasks to each core. This scheme results in low overhead but performs poorly if tasks are not all of the same size, because then at the end of computation some cores will be sitting idle.

A basic dynamic scheme assigns one task to each core, and then assigns the remaining tasks in sequence as cores finish their current task. (For smaller tasks, several may be assigned at once in blocks.) This model can have greater overhead due to more master-worker communication, but promises to keep as many workers busy as possible at any given time.

Because the number of models is not that high, but the different models vary greatly in their required computational time, it is important that the parallelization of model fitting algorithms employ dynamic load balancing.

Another consideration for the use of parallel computing is that of memory architecture. *Memory* stores information for immediate use by a computer, and herein is considered to store both the data and instructions that a processor needs to carry out its task. Two types of architectures are considered, *shared memory* systems and *distributed memory* systems.

In a shared memory system, all processors share access to the same memory. This type of system is usually employed on a single computer that features several processors. The shared memory architecture has the advantage of simplicity for the programmer Barney (2016). However, using shared memory limits the scalability of the parallel implementation. As more processors are added, processes compete for access to the memory, slowing down computation. Additionally, there is a physical limit as to how many processors can be built into a single computer.

In a distributed memory system, each processor has its own private memory. Thus a processor is free to access memory without competing with others. Distributed memory systems easily allows the computation to be spread to large network of independent computers. One downside is that any necessary communication between processors needs to be explicitly implemented by the programmer, and sometimes different algorithms and data structures must be used to accommodate for the complete independence of memory. Luckily, for model fitting the necessary communication is minimal.

Parallelization for MCGFA is implemented using the `parallel` package in R (R Core Team, 2016), which implements shared-memory parallelism with a very convenient interface. The function `mclapply` is used and dynamic load balancing is specified by setting the parameter `mc.preschedule=FALSE`. In the dynamic case, the

`mcapply` function creates a process for each model-fitting task using the UNIX `fork` command. Each of these processes has a separate copy of R that executes one of the model-fitting tasks.

3.2 Profiling

One advantage of writing in C is that, as a multi-purpose language often used in high-performance contexts, there are many existing high-quality tools to assist with improving efficiency. Allinea MAP, made available through the Sharcnet computing consortium, is used to profile the C portion of the `mcgfa` implementation. The main tool applied is a *profiling sampler*, which estimates the proportion of computation time spent in different functions during program execution. The sampler pauses execution in short intervals and records which subroutines of the main program are active at that time.

Profiling is performed once the algorithm is implemented but greater efficiency is desired. A large data set is generated to provide a test case for profiling. Drawn from a two-component Gaussian mixture with noise, it is made up of 3020 observations in 15 dimensions. The unconstrained UUU model is fit with $q = 4$ and $G = 3$ and 100 AECM iterations. To increase profiling sample accuracy, each test is repeated 25 times and the results are averaged.

The initial, pre-profiling version of the code completes the task in 339.1 seconds. The profiling results in Table 3.1 show that, in this incarnation, the algorithm spends approximately 75% of its time updating just $\boldsymbol{z}$, $\boldsymbol{v}$ and η. The BIC value for the resulting model is -138338.3.

Table 3.1: Profiling results for original code, before any improvements.

update	z, v	η	α	S	other
time (%)	47.7	26.9	13.9	10.4	1.1

Each of these three updates requires the calculation of the squared Mahalanobis distance, δ_{ig}, of each observation, as a member of each model component:

$$\delta_{ig} = (\mathbf{x}_i - \boldsymbol{\mu}_g)' \boldsymbol{\Sigma}_g^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_g) \,. \tag{3.1}$$

Here $\boldsymbol{\mu}_g$ and $\boldsymbol{\Sigma}_g$ are the current estimates of the mean and covariance of the g-th group. Investigation of the profile data reveals that the bulk of computation time is spent on the computation of δ_{ig}, due to the matrix inversion and matrix operations associated with the application of the Woodbury identity.

A major inefficiency is found through this profiling—for each cycle of the AECM algorithm, δ_{ig} is calculated three times. By computing the Mahalanobis distance in a separate function and storing it, the same calculation could be used for the η update and the z, v update in the first cycle of the AECM algorithm.

The running time after this modification is 272.5 seconds, for a 19.6% speed-up. The BIC value remained unchanged, as expected. The profiling results are shown in Table 3.2.

Table 3.2: Profiling results for the first code revision, avoiding redundant Mahalanobis-distance updates.

update	δ	α	S	η	z, v	other
time (%)	50.5	16.7	14.1	10.5	7.2	1.0

At this point, the δ-update is still taking up the majority of computational time,

but the next largest task is the α-update. As in the R version of the code, the α-update is performed numerically, even though an analytical update is easily derivable using calculus. It is stated in Punzo and McNicholas (2013) that this is done to allow a constraint on the range of α, i.e. $\alpha \in [\alpha_{\min}, 1)$. However, this constraint can be imposed by simply using the update $\alpha^{(k+1)} = \max\{\alpha_{\min}, \hat{\alpha}\}$, where $\hat{\alpha}$ is the update given in Chapter 3. This works because the log-likelihood is strictly concave down as a function of α, for $z_{ig} \in (0, 1]$, $v_{ig} \in (0, 1)$, and $\alpha_g \in (0, 1)$.

The analytical α update is implemented in Revision 2, and the profiling results of this version are shown in Table 3.3. The analytical α results in a further 13.4% reduction in computation time (total running time: 236.0 seconds). The analytical update yields a slightly different model fit, due to the elimination of numerical approximations. This change is for the better, as the BIC improved to -138338.1.

Table 3.3: The profiling results for the second code revision, with analytical updates of α.

update	Mahal.	S	η	z, v	μ	other
time (%)	58.7	19.3	11.3	8.1	1.3	1.3

With the α update now taking insignificant computational time, the update of the sample covariance matrix S is more dominant. The flop count of this update was cut down by simply taking advantage of the symmetry of S: Rather than compute each of the p^2 elements, $p(p-1)/2$ were computed and then the upper-triangular elements were copied to the lower triangle. Isolating the S-update function and timing 10000 iterations, it is determined that this modification reduced computation time for S by approximately 40%. Revision 3 of the code implements this change, and gives a total running time of 214.7 seconds, a further 9.3% speed up. The BIC remains unchanged.

This speed up is more modest but required only a tiny modification to the code.

Table 3.4: The profiling results the third code revision, with more efficient updates to the sample covariance matrix S.

update	Mahal.	z, v	η	S	μ	other
time (%)	65.4	11.6	10.4	10.2	1.3	1.1

At this point, the δ-updates still dominate the computation, and there is no obvious way to speed up the other updates. Faster algorithms for matrix inversion, like those in the well-known LAPACK linear algebra library were investigated. However, because of the Woodbury trick, the matrices being inverted were only of order q. Thus the overhead involved in calling the LAPACK routines actually slows computation. Testing revealed that it is not until order 15 that the LAPACK algorithms began to outperform the Gaussian Elimination C implementation from `pgmm`. Because fitting models with more than $q = 15$ latent factors is unlikely, this approach is abandoned.

Further investigation of the code profile data reveals that approximately 10% of the operational time is spent just on dynamic memory allocation for matrix calculations. The code is altered so that memory is allocated for these operations once, and then reused, leading to expected 10% speed up.

However, finally, the root cause of the slowdown is determined. The same function both performed the inversion of $\mathbf{\Sigma}_g$, and computed the Mahalanobis distance for each observation. Thus the inverse is computed for each observation and group. By splitting this function apart so that each inverse is only computed once per group, additional time savings of 58% were obtained, for a running time of only 105.7 seconds.

This final iteration of the code, Revision 4, is profiled and the results are shown in Table 3.5. The BIC and model fit remained unchanged from the second revision.

Table 3.5: The profiling results for the fourth and final code revision, with more efficient Mahalanobis distance updates.

update	Mahal.	η	S	z, v	μ	other
time (%)	31.0	22.5	20.9	20.7	2.1	2.8

This actually left the code faster than the PGMM code from which it is adapted, at least on this data. This inefficiency shows the downside to working in a language like C: because matrix operations involve more lines of code, the task can become obscured, and a counter-intuitive implementation may be written.

Overall, the changes resulting from the profiling accounted for a 3.2 times speed up in the code. This demonstrates that profiling is an important tool for the statistical programmer. The evolution of the computational time and breakdown of computation time is shown in Figure 3.1.

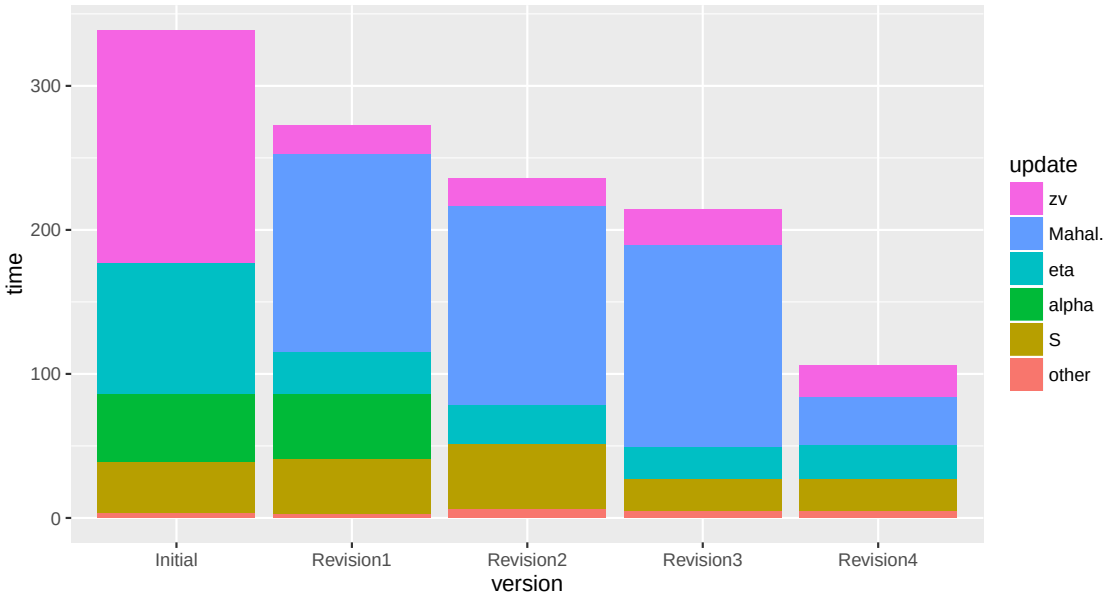


Figure 3.1: Breakdown of computation time by version.

Chapter 4

Performance & Evaluation

In this chapter, the performance of the novel `mcgfa` implementation is evaluated on both simulated and real data. First, the computational efficiency of the new implementation is considered. The speed of serial execution is contrasted with the existing R code, and the efficiency of the parallel implementation is measured. Then, the clustering and classification performance of MCGFA is compared to two natural competitors: EPGMM, as implemented in the R package `pgmm`, and MMTFA, as implemented in `mmtfa`.

It is difficult to directly evaluate the performance of unsupervised learning algorithms like those of model-based clustering. A straightforward approach is to take fully labelled data sets, apply the clustering algorithm as if the labels were not provided, and then evaluate performance using the labels. There are two issues with this approach. First, there is the "label switching problem", which occurs when all of the labels for several classes are permuted. Stephens (2000) discusses this issue in depth, in a way relevant to mixture model-based clustering. The second, more substantial problem is that clustering algorithms may select the wrong number of components.

It is then impossible to use the classification rate as a performance measure.

Both of these issues are resolved by the use of the Rand index (Rand, 1971), which measures agreement between two data partitions by considering all pairs of observations. Given a partition, every pair of two observations are either members of the same component (*together*), or members of different components (*separated*). Given two partitions, the Rand index is then defined as the ratio:

$$R = \frac{\#\text{ pairs together in both partitions} + \#\text{ pairs separated in both partitions}}{\text{total }\#\text{ pairs}}. \tag{4.1}$$

The total number of pairs is simply $\binom{N}{2}$. Perfect agreement between partitions thus gives $R = 1$. Hubert and Arabie (1985) introduce the Adjusted Rand Index (ARI), a modified version of the index that takes into account the expected agreement of two random clusterings. The ARI is defined as:

$$\text{ARI} = \frac{\text{Rand Index} - \text{Expected Rand Index}}{\text{Max. Rand Index} - \text{Expected Rand Index}}. \tag{4.2}$$

The ARI takes the value zero under the expected Rand Index, and is bounded above by 1. If one of the partitions is known to be correct, then the ARI can be viewed as a measure of performance.

4.1 Efficiency of Implementation

In this section, the computational efficiency of the novel MCGFA implementation is measured by timing the total elapsed time of large model-fitting tasks. Execution times are produced using the `system.time` command in R. All timing is performed

on the Orca system of the SHARCNET computing consortium. SHARCNET is a consortium of 18 Canadian academic institutions who share a network of high performance computers. The Orca computing cluster is a network of 394 computers, each of which have either 16 or 24 computing cores. One of the development nodes, `orc-dev4` is used for all tests. The `orc-dev4` has 24 cores, an AMD Opteron 2.2GHz CPU, 32 GB of memory, and runs the CentOS 6.7 operating system.

4.1.1 Serial Implementation

The reduction in computation time due to the novel implementation is not subtle. As a demonstration, a 20-dimensional data set with two components of 100 observations each, and 20 noise points, is produced. Three implementations of `mcgfa` are run on this data: the original R code, the R code modified to make use of the Woodbury identity, and the novel C implementation. For each version, all 8 `mcgfa` models are fitted with $G = 2, 3$, $q = 1, 2, 3$, a stopping tolerance of 0.0001 and a maximum of 200 iterations per model fit. All versions utilized `pgmm` as an initialization. The timing is repeated ten times to account for any irregularities or variations in the CPU activity during testing. The timing results are shown in Table 4.1.

Table 4.1: Running time of MCGFA implementations in seconds.

Repetition	Original R	R with Woodbury	C
1	195.1	190.3	12.0
2	186.9	178.3	11.3
3	192.7	177.4	11.8
4	182.4	174.6	11.2
5	184.8	166.1	11.3
6	188.1	164.5	11.2
7	184.2	171.4	11.2
8	186.9	164.8	11.2
9	184.0	171.4	11.2
10	185.2	171.2	11.2
Mean	187.0	173.0	11.4

These results show that the Woodbury trick does not greatly reduce the computation time of the R implementation, providing only an 8% speedup. This indicates that the inversion of $\mathbf{\Sigma} = \mathbf{\Lambda}'\mathbf{\Lambda} + \mathbf{\Psi}$ does not make up the majority of the computational load, as is the case with the C implementation. Meanwhile, the C implementation allows a speed up of 16.4 times as compared to the original R code, or 15.1 times compared to the R code that makes use of the Woodbury trick.

This test compares the computational efficiency over the exact same set of 48 models, using a single processor. It take advantage of the fact that the C version avoids fitting redundant one-component models. The goal is to demonstrate the speed-up provided by the new implementation on the same models using the same processing power.

The C code is shown to be substantially faster than the original R version, and produces the same results. Thus in the tests below only the C version is indicated when referring to `mcgfa` or `mcgfa_km`.

4.2 Parallel Implementation

Once the serial code is verified to show improved performance, the parallel version of the same code is tested to see how efficiently it utilizes more computational resources. Ideally, a linear speedup would be achieved, in which the use of p processors would decrease computation time by a factor of p. This ideal situation is rarely achieved, except in special circumstances relating to the physical layout of computer hardware.

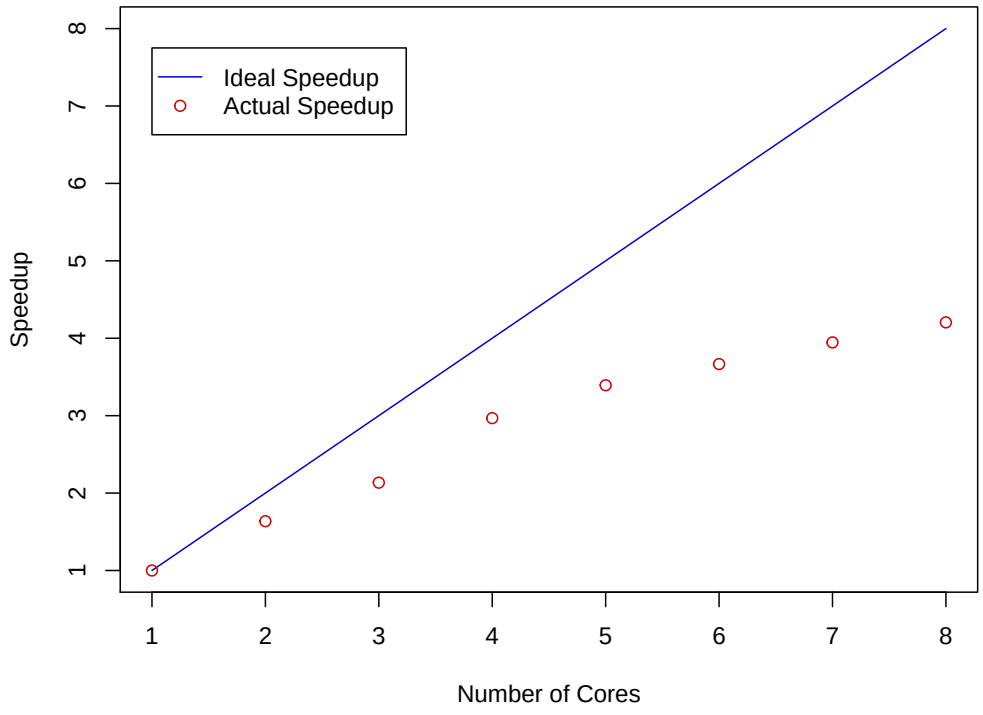


Figure 4.1: Computational speed-up afforded by parallelization, for a typical set of models of differing numbers of components and latent factors.

A simulated data set of 800 observations in 12 dimensions is produced as an example of a moderately large data set. The MCGFA model fitting procedure takes 874 seconds to execute on one core. The procedure is repeated using 2 to 8 processors of the same cluster. The results of the this application of parallelism is not impressive—performance is reasonable only up to 4 cores. Of course, the use of more processors decreased computation time, but as shown in Figure 4.1, the actual speedup fell well short of the ideal.

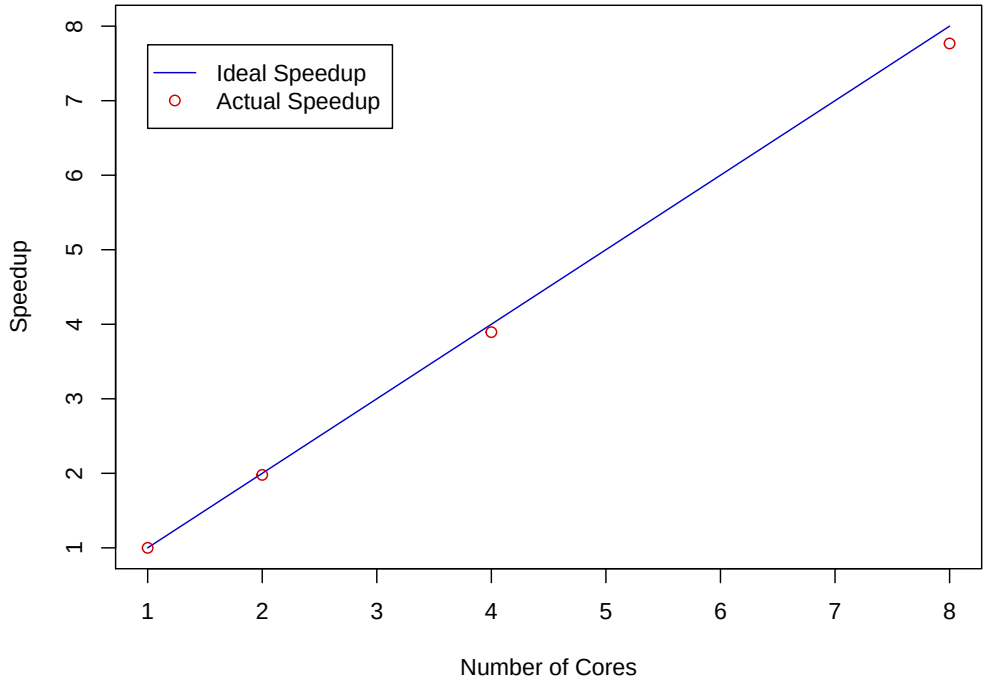
There are several possible reasons for this deviation from the ideal. It is possible that overhead results from communication traffic due to the shared-memory architecture. Creation and deletion of several copies of the current R session could also contribute to overhead. There could also be a loss of efficiency due to the communication needed between cores to implement dynamic load allocation.

Additionally, because the model-fitting tasks vary greatly in size, it is unlikely for all available computational resources to be used for the full duration of computation. It is difficult to account for this variation, because the running time of each model fit cannot be estimated reliably ahead of time. While a more highly-parameterized model (for, instance an unconstrained UUU model with many latent factors) will generally take longer to fit than a simpler one, differences in the number of steps needed for convergence make this difficult to account for. Still, grouping model by their likely computational size and intelligently distributing them among processors could reduce overhead. This additional fine control also necessitates the use of a more flexible tool than the `parallel` package provides.

To determine which of these possibilities explain most of the departure from ideal speedup, the same parallelization method is applied to eight identical model-fitting

tasks, with 1, 2, 4, and 8 cores. When the tasks were all identical, the scaling is near-linear, as shown in Figure 4.2. This indicates that most of the lost speedup is due to the great differences in computational time per task, not overheads inherent in the parallel implementation.

Figure 4.2: Computational speed-up when all tasks are of the same size.



As an alternative to the `parallel` package, the message passing interface (MPI) could be used to provide parallel computing capability. MPI is a specification for communication between processors in a computing cluster, and has a widely available C version. MPI requires a distributed-memory architecture and would thus improve the scalability of the parallel implementation. Furthermore, implementing parallelism within the C code would eliminate any overhead caused by initializing many copies of

R. McNicholas *et al.* (2010) found success applying MPI to the model-based clustering problem.

While message passing can increase overhead in many situations, for the model-fitting application this is unlikely due to the minimal communication required between processes. The use of MPI would not solve the load-balancing issue, however the greater control over the process distribution method might allow more sophisticated load-balancing strategies.

4.3 Simulated Data

In this section, five types of data sets are considered:

1. Gaussian clusters
2. Contaminated Gaussian clusters
3. t-distributed clusters
4. Gaussian clusters with noise
5. Gaussian clusters with one severe outlier

A total of 100 repetitions of each type of data set are produced before any testing happens, using seeds from 1 to 100. This is to prevent any chance of bias in the selection of test data. The repetitions are six-dimensional with two components, to keep computation time reasonable. The first four types have no latent structure, while the last two types are simulated as two-factor latent factor models.

4.3.1 Gaussian Clusters

For each of 100 simulations, two equally-sized six-dimensional Gaussian clusters are generated. The first has mean at the origin and the other has a mean vector drawn from a Gaussian distribution centred at the origin. Random covariance matrices are created for each component using the `genPositiveDefMat` function of the `clusterGeneration` package (Qiu and Joe, 2006). The clustering and model selection are shown in Tables 4.2 and 4.3.

Table 4.2: Clustering performance on Gaussian clusters.

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
Mean ARI	0.780 (0.34)	0.780 (0.34)	**0.785** (0.32)	0.782 (0.32)
Mean BIC	-3167.269	-3167.369	-3151.937	**-3148.111**

Table 4.3: Model selection performance of mixtures of factor analyzer models on Gaussian clusters

G	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
1	14	14	12	12
2	85	86	84	82
3	1	0	4	6
errors	15	**14**	16	18

As expected, `pgmm` achieves the best BIC values—it does not include the extra unnecessary parameters that allow fatter tails. The `mcgfa` models have the worst BIC, likely because they require the estimation of two extra parameters per group (α and η), while `mmtfa` needs only to estimate the degrees of freedom. Still, it is

confirmed that extending `pgmm` to `mcgfa` does not significantly affect the ability to capture Gaussian clusters, and in fact `mcgfa` is most likely to select the correct number of components. Interestingly, `mmtfa` actually outperforms `pgmm` in terms of ARI. This could be due to differences in the implementation of the respective EM algorithms.

4.3.2 Contaminated Gaussian Clusters

Two six-dimensional clusters are generated similarly to the previous section. However, a covariance inflation factor η for each component is drawn from an exponential distribution with mean $\beta = 10$. Ten percent of observations in the first group and twenty percent of those in the second group are designated as "bad." Combining these parameters yields a pair of contaminated Gaussian clusters. The four methods are applied to each of 100 repetitions and the results are shown in Tables 4.4 and 4.5.

Table 4.4: Clustering performance of mixtures of factor analyzer models on contaminated Gaussian clusters.

	`mcgfa`	`mcgfa_km` c	`mmtfa`	`pgmm`
Mean ARI	0.822 (0.27)	**0.832** (0.24)	0.800 (0.24)	0.700 (0.25)
Mean BIC	-2793.4	**-2791.5**	-2808.3	-2835.5

Table 4.5: Model selection performance of mixtures of factor analyzer models on contaminated Gaussian clusters.

G	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
1	7	5	0	0
2	**87**	86	82	19
3	6	9	18	81

As expected, `mcgfa` performs best on its own model, in terms of ARI, BIC and model selection. It is clear that the `pgmm` algorithm is greatly affected by this departure from normality. The application of `mmtfa` is successful, but does tend to over-select for the number cof components.

4.3.3 t-distributed Clusters

Two six-dimensional t-distributed clusters are generated, one with mean at the origin and the other with a mean drawn from a Gaussian distribution centred at the origin. Two positive definite covariance matrices are created using the `genPositiveDefMat` function of the `clusterGeneration` package. The number of degrees of freedom for each component is uniform drawn on the range $1, \ldots, 50$. The results for each of the four algorithms are shown in Tables 4.6 and 4.7.

Table 4.6: Clustering performance of factor analyzer models on t-distributed clusters.

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
Mean ARI	0.841 (0.28)	0.833 (0.30)	**0.857** (0.25)	0.798 (0.32)
Mean BIC	-2786.6	-2788.5	**-2739.5**	-2868.9

Table 4.7: Model selection performance of mixtures of factor analyzer models on t-distributed clusters.

G	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
1	9	9	2	6
2	89	**90**	**90**	84
3	2	1	8	10

While `mcgfa_km` and `mmtfa` both correctly select 2 component models in 90 of the simulations, they make opposite errors: `mmtfa` tends to select too few groups, while `mcgfa` and `mcgfa_km` select too many. Predictably, `mmtfa` performs best on this data in terms of both BIC and ARI. The `pgmm` model is better able to capture the t-distributed clusters than the contaminated Gaussian ones in the previous section.

4.3.4 Gaussian Clusters with Uniform Noise

Two six-dimensional Gaussian clusters with 100 observations each are simulated from a factor analysis model with 2 underlying factors. For each simulation, the mean vectors are generated as above, along with two random loading matrices $(\mathbf{\Lambda}_1, \mathbf{\Lambda}_2)$ are generated, and with two random error variance vectors $(\mathbf{\Psi}_1, \mathbf{\Psi}_2)$. Finally 20 uniform noise points are added to the data. The noise observations are not considered in the evaluation of clustering performance. Results are shown in Tables 4.8 and 4.9.

Table 4.8: Clustering performance of mixtures of factor analyzer models on Gaussian clusters with uniform noise.

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
Mean ARI	**0.901** (0.15)	0.897 (0.15)	0.898 (0.16)	0.840 (0.25)
Mean BIC	**-3254.8**	-3259.8	-3265.6	-3338.0

The `mcgfa` algorithm yielded the best performance on this data. While the mean ARI is very similar for `mcgfa` and `mmtfa`, the `mcgfa` procedure is more likely to take on the correct number of components, and the `mcgfa` models have a higher mean BIC.

Table 4.9: Model selection performance of mixtures of factor analyzer models on Gaussian clusters with uniform noise.

G	mcgfa	mcgfa_km	mmtfa	pgmm	q	mcgfa	mcgfa_km	mmtfa	pgmm
1	0	0	0	0	1	6	9	6	30
2	**97**	95	93	16	2	**92**	89	**92**	70
3	3	5	7	84	3	2	2	2	0

Noise Detection

The mcgfa algorithm proved successful at detecting noisy observations with either initialization scheme. To evaluate this detection, both *sensitivity* and *specificity* are considered. The sensitivity is the proportion of bad points successfully detected, and the specificity is the proportion of good points successfully labelled as such. The detection results are shown in Table 4.10. The specificity figures are impressive considering noise points may easily lie within clusters.

Table 4.10: Outlier detection results for MCGFA algorithms on Gaussian clusters with uniform noise.

	mcgfa	mcgfa_km
Mean # Correctly Detected	17.72	17.77
Mean # Falsely Detected	4.68	5.29
Mean Sensitivity	88.6%	88.9%
Mean Specificity	97.7%	97.4%

4.3.5 Gaussian Clusters with One Severe Outlier

Two six-dimensional Gaussian clusters are simulated, as in the uniform noise case. One fixed outlying point is added to every simulation, at $(15, 0, 0, 0, 0, 0)$. Results are shown in Tables 4.11 and 4.12.

Table 4.11: Clustering Performance on Gaussian Clusters with Single Outlier

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
Mean ARI	0.920 (0.13)	0.915 (0.14)	**0.924** (0.13) (	0.896 (0.18)
Mean BIC	**-1272.9**	-1273.7	-1289.7	-1312.6

Table 4.12: Model Performance on Gaussian Clusters with Single Outlier

G	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`	q	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
1	0	0	0	0	1	17	21	17	37
2	69	71	**82**	21	2	75	74	**80**	63
3	31	29	18	79	3	8	5	3	0

Here the `mmtfa` proved to be the best tool for both model selection and clustering performance, while the `mcgfa` algorithms yielded the best average BIC value. Overall, however, the three robust methods had very similar ARI scores. Predictably the application of `pgmm` is most likely to lead to the selection of a three-component model, as this is the only way the light-tailed model can capture outlying points. On the other hand, `mmtfa` is most likely to lead to the correct 2-component model. This reflects the ability of its fitting algorithm and model to capture the outlying point as an anomalous member of one of the two main components, rather than a separate component.

Noise Detection

With either initialization approach, the outlying point is detected on 86% of the runs, as shown in Table 4.13. Considering the severity of the outlier, this is not impressive. This occurs because a three component model is often selected, with the outlier as the centre of the third component. When considering only runs that lead to the selection of one- or two-component models, the outlier is correctly detected 100% of the time, for both initialization schemes.

Table 4.13: Outlier detection by MCGFA on uniform noise data.

	`mcgfa`	`mcgfa_km`
Mean # Correctly Detected	0.86	0.86
Mean # Falsely Detected	4.08	5.36
Mean Sensitivity	86%	86%
Mean Specificity	95.9%	94.6%

4.4 Real Data Analysis

4.4.1 Breast Cancer Wisconsin Data Set

All of the above tests evaluate the four methods in an unsupervised learning context. The Breast Cancer Wisconsin data set, first studied in Street *et al.* (1993), provides an opportunity to investigate semi-supervised performance. The data consists of 569 observations of 32 features. The numerical features are derived from digital images of breast mass, and are classified as either malignant or benign. One sixth of the data is randomly selected to act as a labelled portion, and semi-supervised classification

is carried out using every version of each method and $q = 1, \ldots, 5$. The classification results are shown in Tables 4.14 and 4.15.

Table 4.14: Contingency tables of each method for breast cancer data.

	`mcgfa`		`mcgfa_km`		`mmtfa`		`pgmm`	
	B	M	B	M	B	M	B	M
benign	300	9	284	25	299	10	275	34
malignant	12	176	20	168	17	171	17	171

Table 4.15: Classification results for each model for breast cancer data.

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
Model	UUU	CUU	UUUC	UUU
Misclass. Rate	**3.4%**	9.1%	5.4%	10.3%
BIC	-10702.7	-11201.91	**-9372.886**	-13080.72

This study highlights the importance of initialization. The `mcgfa` method gives by far the best classification, but only when using the `pgmm` initialization. When initialized by k-means clustering, its performance is nearly as bad as the `pgmm` model itself. This is somewhat up to chance, however, and more random starts may allow the `mcgfa_km` method to find a better local maximum for the model likelihood. `mmtfa` gave the best BIC model, although in a supervised or semi-supervised context, a test set would likely be used as a more direct measure of performance.

4.4.2 Wine Data

The wine data set (Forina *et al.*, 1986) consists of 27 chemical properties of 178 bottles of wine, of three different types: Barolo, Grigolino and Barbera. It is included with

the pgmm package for R. There also exists a thirteen-dimensional version of the wine data, but the full version is considered in this study. Each method is fit to the data with every set of constrains, $G = 1, \ldots, 4$ and $q = 1, 2, 3$. The data was scaled before fitting in every case and model selection was performed with the BIC as usual. The results are shown in Tables 4.16 and 4.17.

Table 4.16: Contingency tables for each model on wine data. Each model is fit with $G \in \{1, \ldots, 4\}$, $q \in \{1, 2, 3\}$ and every covariance structure.

	mcgfa			mcgfa_km			mmtfa					pgmm			
	1	2	3	1	2	3	1	2	3	4	5	1	2	3	4
Barolo	59	0	0	59	0	0	59	0	0	0	0	57	0	2	0
Grignolino	1	68	2	1	68	2	0	49	21	1	0	0	49	22	0
Barbera	0	0	48	0	0	48	0	0	0	24	24	0	0	1	47

Application of both versions of mcgfa recovered the three-component structure, and in fact the exact same partitioning. However, as detailed in McNicholas and Murphy (2008), the extra partitioning in the pgmm and mmtfa may not be erroneous, but correspond to different years of production.

Table 4.17: Performance Measures for each model on wine data.

	mcgfa	mcgfa_km	mmtfa	pgmm
model	CUU	CUU	CCCC	CCUU
ARI	**0.901**	0.897	0.898	0.840
BIC	**-3254.8**	-3259.8	-3265.6	-3338.0

4.4.3 AIS Data

The AIS data Cook and Weisberg (1994) contain 11 numerical measurements of 202 athletes, along with their classification by gender and sport. The methods are evaluated on their ability to separate the athletes by gender, ignoring their sport. All versions of each method were fit to the data with $G \in \{1, \ldots, 4\}$ and $q \in \{1, 2, 3\}$, and models were selected via BIC as usual. The contingency tables and measures of performance are shown in Tables 4.18 and 4.19.

Table 4.18: Contingency tables for each model on AIS Data. Each model is fit with $G \in \{1, \ldots, 4\}$, $q \in \{1, 2, 3\}$ and every covariance structure.

	`mcgfa`			`mcgfa_km`			`mmtfa`			`pgmm`		
	1	2	3	1	2	3	1	2	3	1	2	3
female	1	24	75	58	2	40	59	40	1	0	3	97
male	57	44	1	3	98	1	3	1	98	36	65	1

Every method selected a three-component model, possibly due to the skewness in present in the data. Applying `pgmm` gives the best partition, yielding the least confusion between the genders. The `mcgfa_km` and `mmtfa` clusterings both subdivided women in two components, while that of `pgmm` broke up the men. The `mcgfa` model that used `pgmm` as an initialization yielded the worst partition, confusing a large proportion of men and women.

Table 4.19: Performance Measures for AIS Data, $G = 1, 2, 3$

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
Model	CUU	CUU	CUUC	UUCU
ARI	0.438	0.672	0.687	**0.702**
BIC	**-2908.3**	-2981.8	-2952.6	-2938.7

Because every partition chose the incorrect number of components, the models were fit a second time and forced to choose two components. The contingency tables resulting from forcing $G = 2$ are shown in Table 4.20, and the performance measures in Table 4.21.

Table 4.20: Contingency tables of each method for AIS Data, with the constraint $G = 2$.

	`mcgfa`		`mcgfa_km`		`mmtfa`		`pgmm`	
	F	M	F	M	F	M	F	M
female	97	3	99	1	96	4	97	3
male	3	99	4	98	3	98	2	100

Table 4.21: Performance measures for each method on AIS Data, with the constraint $G = 2$.

	`mcgfa`	`mcgfa_km`	`mmtfa`	`pgmm`
ARI	0.884	**0.903**	0.866	**0.903**
BIC	-3023.0	-3024.0	**-3008.5**	-3105.6

Now the algorithms produce very similar partitions. This means that the issues here were primary with model selection, not with capturing the distribution of the

data. It is notable the model with the best BIC value actually yields the worst clustering performance. This highlights that the model with the best BIC does not necessary provide the most accurate clustering.

Chapter 5

Conclusions

This thesis introduced a family of models extending the mixtures of factors analyzers model, with a new, efficient implementation. The use of the contaminated Gaussian robustifies the factor analysis model and allows automatic outlier detection. The family of eight parsimonious models for the permit different degrees of local or global dimensionality reduction.

Significant computational work led to the discovery concrete efficiency improvements be made to existing implementations of both MCGFA and PGMM. Future computational work will involve a more specialized parallel implementation, increasing the speed up per processor. For example, parallelization at the C level directly will eliminate the overhead of initializing more R sessions and provide better portability.

The increased computational efficiency allowed thorough trial and evaluation of the MCGFA model. Tests on both simulated and real data verified that the contaminated Gaussian distribution provides both robustness and automatic outlier detection. MCGFA was shown to outperform the Gaussian model on a variety of simulated and

real data sets. Overall, the model based on the contaminated norm performed similarly to the MMtFA model in terms of clustering performance and model selection. The key difference is that MCGFA permits automatic detection of outliers, without a subjective choice of threshold.

The two competitors to MCGFA studied above give inspiration for further extension to MCGFA. As the EPGMM model extends PGMM, MCGFA could be expanded to a family of 12 models. Furthermore, the MMTFA allows an additional constraint on the degrees of freedom parameter. The analogous parameters in MCGFA are η and α. Thus, further extensions to MCGFA could allow the constraints $\alpha_g = \alpha$ and/or $\eta_g = \eta$. In fact, Andrews and McNicholas (2011a) state that "models with constrained degrees of freedom can give better clustering performance than the unconstrained models."

Finally, it noted that of the models evaluated herein are based on symmetrical distributions. Employing a contaminated skewed distribution in the mixtures of factor analyzers model could provide the same advantages of the MCGFA model, with even more distributional flexibility. A logical candidate is the contaminated skew normal, discussed in a linear modelling context in Lachos *et al.* (2010).

Bibliography

Aitken, A. C. (1926). A series formula for the roots of algebraic and transcendental equations. *Proceedings of the Royal Society of Edinburgh*, **45**, 14–22.

Andrews, J. L. and McNicholas, P. D. (2011a). Extending mixtures of multivariate t-factor analyzers. *Statistics and Computing*, **21**(3), 361–373.

Andrews, J. L. and McNicholas, P. D. (2011b). Mixtures of modified t-factor analyzers for model-based clustering, classification, and discriminant analysis. *Journal of Statistical Planning and Inference*, **141**(4), 1479–1486.

Andrews, J. L., McNicholas, P. D., and Chalifour, M. (2015). *mmtfa: Model-Based Clustering and Classification with Mixtures of Modified t Factor Analyzers*. R package version 0.1.

Barney, B. (2016). Introduction to parallel computing. `https://computing.llnl.gov/tutorials/parallel_comp`. Accessed: 2016-09-01.

Böhning, D., Dietz, E., Schaub, R., Schlattmann, P., and Lindsay, B. (1994). The distribution of the likelihood ratio for mixtures of densities from the one-parameter exponential family. *Annals of the Institute of Statistical Mathematics*, **46**(2), 373–388.

Browne, R. P. and McNicholas, P. D. (2015). A mixture of generalized hyperbolic distributions. *Canadian Journal of Statistics*, **43**(2), 176–198.

Cook and Weisberg (1994). *An Introduction to Regression Graphics*. Wiley, New York.

Cozman, F. G., Cohen, I., Cirelo, M. C., *et al.* (2003). Semi-supervised learning of mixture models. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 99–106.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, **39**(1), 1–38.

Forina, M., Armanino, C., Castino, M., and Ubigli, M. (1986). Multivariate data analysis as a discriminating method of the origin of wines. *Vitis - Journal of Grapevine Research*, **25**, 189–201.

Ghahramani, Z. and Hinton, G. E. (1997). The EM algorithm for mixtures of factor analyzers. Technical Report G-TR-96-1, University of Toronto.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, **2**(1), 193–218.

Jiang, D., Tang, C., and Zhang, A. (2004). Cluster analysis for gene expression data: a survey. *IEEE Transactions on knowledge and data engineering*, **16**(11), 1370–1386.

Lachos, V. H., Ghosh, P., and Arellano-Valle, R. B. (2010). Likelihood based inference for skew-normal independent linear mixed models. *Statistica Sinica*, **20**, 303–322.

Lin, T. I., Lee, J. C., and Yen, S. Y. (2007). Finite mixture modelling using the skew normal distribution. *Statistica Sinica*, **17**, 909–927.

McLachlan, G., Peel, D., and Bean, R. (2003). Modelling high-dimensional data by mixtures of factor analyzers. *Computational Statistics & Data Analysis*, **41**(34), 379–388.

McLachlan, G. J. and Peel, D. (1998). Robust cluster analysis via mixtures of multivariate t-distributions. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 658–666. Springer.

McLachlan, G. J., Bean, R., and Jones, L. B.-T. (2007). Extension of the mixture of factor analyzers model to incorporate the multivariate t-distribution. *Computational Statistics and Data Analysis*, **51**(11), 5327–5338.

McNicholas, P. D. (2016). *Mixture Model-Based Classification*. Chapman and Hall/CRC, Boca Raton.

McNicholas, P. D. and Murphy, T. B. (2008). Parsimonious Gaussian mixture models. *Statistics and Computing*, **18**(3), 285–296.

McNicholas, P. D. and Murphy, T. B. (2010). Model-based clustering of microarray expression data via latent Gaussian mixture models. *Bioinformatics*, **26**(21), 2705–2712.

McNicholas, P. D., Murphy, T. B., McDaid, A. F., and Frost, D. (2010). Serial and parallel implementations of model-based clustering via parsimonious Gaussian mixture models. *Computational Statistics & Data Analysis*, **54**(3), 711–723.

McNicholas, P. D., ElSherbiny, A., McDaid, A. F., and Murphy, T. B. (2015). *pgmm: Parsimonious Gaussian Mixture Models*. R package version 1.2.

Meng, X.-L. and Rubin, D. B. (1993). Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika*, **80**(2), 267–278.

Meng, X.-L. and Van Dyk, D. (1997). The EM algorithm—an old folk-song sung to a fast new tune. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **59**(3), 511–567.

Peel, D. and McLachlan, G. J. (2000). Robust mixture modelling using the t distribution. *Statistics and Computing*, **10**(4), 339–348.

Punzo, A. and McNicholas, P. D. (2013). Parsimonious mixtures of contaminated Gaussian distributions with application to allometric studies. *arXiv preprint arXiv:1305.4669*.

Punzo, A. and McNicholas, P. D. (2014). Robust high-dimensional modeling with the contaminated Gaussian distribution. *arXiv preprint arXiv:1408.2128v1*.

Qiu, W. and Joe, H. (2006). Generation of random clusters with specified degree of separation. *Journal of Classification*, **23**(2), 315–334.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rand, W. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, **66**(336), 846–850.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, **6**(2), 461–464.

Spearman, C. (1904). "General Intelligence," objectively determined and measured. *The American Journal of Psychology*, **15**(2), 201–292.

Stephens, M. (2000). Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **62**(4), 795–809.

Street, W. N., Wolberg, W. H., and Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. In *Society for Imaging Science and Technology/International Society for Optics and Photonics: Symposium on Electronic Imaging: Science and Technology*, pages 861–870. International Society for Optics and Photonics.

Tipping, M. E. and Bishop, C. M. (1999). Mixtures of probabilistic principal component analyzers. *Neural Computation*, **11**(2), 443–482.

Woodbury, M. A. (1950). Inverting modified matrices. Technical Report 42 of the Statistical Research Group, Princeton University, Princeton, New Jersey.

Zhu, X. and Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, **3**(1), 1–130.