SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

PhD Thesis

Data clustering with mixtures of multidimensional distributions

mgr Mateusz Kania

supervisor: prof. zw dr hab inż Andrzej Polański

The following thesis was financially supported by European Union funds project AIDA (Applied Integrative Data Analysis) POWR.03.02.00-00-I029.

This page intentionally left blank.

Contents

Symbols

Symbol	Description
α	mixing proportion
α_k	is a mixing proportion of k component
x	observed value
μ	mean
σ^2 VAR	Variance
σ	Standard deviation
π	Pi
D	Count of observations
exp	exponentiation
Σ	Variance-covariance matrix
$\lvert\Sigma\rvert$	Matrix determinant
Σ_U	Diagonal covariance matrix
Σ^{-1}	Matrix inverse
x^T	Transposition
$\sum$	Symbol of sumation
f	function
Π	Symbol of multiplication
Θ_K	Parameters
l	likelihood function
l^O	likelihood function of observed data
l^C	likelihood function of complete data
L	Log-likelihood function
L^O	Log-likelihood function of observed data
L^C	Log-likelihoodlihood function of complete data
log	Natural logarithm
$\begin{pmatrix} \text{n} \\ \text{k} \end{pmatrix}$	binomial coefficient
N	number of observations
k	component number in the mixture
K	number of components in the mixture
p	probability
Q	Q-function
z	hidden variable

Abbreviations

Type	Description
HC	Hierarchical Clustering
EM	Expectation-Maximization
MMM	Multinomial Mixture Models EM
MMMK	Multinomial Mixture Models EM k-means
MANHC	Hierarchical Clustering with Manhattan distance
KMED	K-medoids
KMN	K-means
KMN++	K-means++
CMN	C-means
MANCMN	C-means with Manhattan distance
WSMC	Weighted Simple Matching Coefficient
SMC	Simple Matching Coefficient
BBM	Beta-Binomial Mean
WJACC	Weighted Jaccard
ARI	Adjusted Rand Index
BACCU	Balanced Accuracy
SVD	Singular Value Decompostion
tSVD	truncated Singular Value Decompostion
PCA	Pricipal Component Analysis
RPRO	Random Projection
tSNE	t-Distributed Stochastic Neighbor Embedding
BIC	Bayesian Information Criterion
BWA	Burrows-Wheeler Aligner
BAM	Binary Alignment Map
DNA	Deoxyribonucleic Acid
VEP	Variant Effect Predictor

1 Introduction

Unsupervised clustering is a group of algorithms that belong to scientific areas of data analysis, machine learning and artificial intelligence. They aim to solve problems of assigning a certain number of objects/items into groups based on some similarity/distance criterion/metrics between objects.

Unsupervised clustering is a vital and fast-developing area with numerous applications in current data science algorithms. In scientific data applications, unsupervised clustering can be defined as an independent problem, with suitably specified quality criteria or as a part of some data analysis pipelines with many possible functions, e.g., data filtering, estimation of data structure, computing of some quality indices of algorithms or their parts [1]. There are many approaches to constructing unsupervised clustering algorithms [2, 3], and many surveys devoted to comparisons between different unsupervised algorithms [4].

Despite very intensive research already done in the area, problems still require attention and more profound studies. One of the problems data scientists often face in their research work very often encounter is the choice of the unsupervised clustering algorithm. The choice becomes difficult with many available methods often accompanied by software implementations. The expensive and tedious solution is implementing and comparing many unsupervised clustering algorithms for a studied problem. The possibility which can support a decision on the choice of the algorithm is using results of studies comparing classes of algorithms. Two classes defined in [5] are model-based clustering algorithms versus heuristic clustering algorithms. In this thesis, we distinguish two classes of unsupervised clustering algorithms, which more rigorously are defined as follows:

- Model-based Algorithms: unsupervised clustering algorithms based on mixtures of multivariate distributions of feature vectors/observations vectors,

- Distance-based Algorithms: unsupervised clustering algorithms based on distance functions defined for pairs of feature vectors/observations vectors.

The above-defined classes correspond to the algorithms defined in [5]. The aim of the study in this thesis, as specified below, is a comparison of algorithms for these two classes.

1.1 Aim

The PhD project's aim, realised and described in this document, was to derive, implement and compare models and related algorithms of unsupervised clustering. The work emphasises the usage of model-based algorithms using multivariate mixture distributions. We compare them

with distance-based algorithms, k-means, k-medoids, agglomerative hierarchical clustering and fuzzy c-means.

We implemented two model-based unsupervised clustering algorithms and four distance-based unsupervised clustering algorithms to achieve this aim. The first model-based algorithm, Gaussian Mixture EM, is based on a multivariable mixture of normal distributions. The second one, Multinomial Mixture EM, is based on a mixture of multinomial distributions. The naming convention with EM stresses that these clustering algorithms rely numerically on using expectation maximisation (EM) algorithms for mixtures [6]. Distance-based algorithms are agglomerative hierarchical clustering, k-means, k-medoids, and fuzzy c-means.

Along with implementing model-based and distance algorithms, we applied those algorithms to several data sets. Part of the data was simulated mixtures of multivariable distribution, both gaussian and multinomial. The other part, and most data, consists of actual data downloaded from various, mostly publicly available sources. Having the clustering results, to quantify them, we have used a few different metrics. Those metrics included the Adjusted Rand Index, Simple Matching Coefficient with weighted variant, Weighted Jaccard index, Balanced Accuracy and metrics based on Beta-Binomial conjugate distribution. Then, we presented our findings graphically, along with a brief description of the results.

1.2 Theses

1. Unsupervised clustering methods based on mixtures of distributions achieve optimal performance when data statistics are consistent with actual distributions.

2. Unsupervised clustering based on distributions' mixtures is competitive compared to distance-based methods.

3. Applicability of clustering based on mixtures of distributions to practical problems relies on elaborating algorithmic implementation specialized for large sizes of datasets.

1.3 Original elements of the thesis and publications related to the thesis

The original elements and contributions of the submitted theses are as follows:

- Formulating algorithms for decomposing mixtures of multivariable Gaussian and multinomial distributions

- Elaborating software tools in an R language environment implementing unsupervised clustering algorithms based on mixtures of Gaussian and multinomial distributions. Optimizing the elaborated implementation such that it enables clustering of large datasets or order of hundreds of thousands of features/observations.

- Based on code sources available in the literature, implementing several distance-based clustering algorithms.
- Elaborating software tools implementing a collection of quality indices of clustering in the R language environment
- Elaborating software tools for simulating multidimensional data of Gaussian or multinomial distributions
- Creating a collection of the real dataset for comparison study with possible variable structure and sizes of practical importance
- Performing comparison study for all analyzed clustering algorithms for the real and simulated dataset

Publications/conference presentation related with this thesis are:

Kania, M., Polański, A., Unsupervised clustering for detection of gene expression patterns in human cancers. 2022 , Recent Advances in Computational Oncology and Personalized Medicine, Volume 2, Silesian University of Technology Publishing House

The publication consist of the comparison of distance and model based algorithms in the gene expression data of different human cancers. We compared how various unsupervised algorithms can distinguish different cancer patterns.

Unsupervised clustering of gene data of TCGA patients by using mixtures of multidimensional Gaussian distributions,5th Advanced Online & Onsite Course on Data Science & Machine Learning | August 22-26, 2022, Castelnuovo Berardenga (Siena) Tuscany, Italy

This conference presentation describes the use of Gaussian Mixture Models, along with distance based algorithms, to compare and find patterns in various TCGA expressions.

In the papers below, unsupervised clustering techniques were used / implemented as parts of data analysis scenarios.

Mika, J., Tobiasz, J., Zyla, J., Papiez, A., Bach, M., Werner, A., Kozielski, M., Kania, M., Gruca, A., Piotrowski, D. and Sobala-Szczygieł, B., 2021. Symptom-based early-stage differentiation between SARS-CoV-2 versus other respiratory tract infections—Upper Silesia pilot study. Scientific reports, 11(1), pp.1-13.

Henzel, J., Tobiasz, J., Kozielski, M., Bach, M., Foszner, P., Gruca, A., Kania, M., Mika, J., Papiez, A., Werner, A. and Zyla, J., 2021. Screening Support System Based on Patient Survey Data—Case Study on Classification of Initial, Locally Collected COVID-19 Data. Applied Sciences, 11(22), p.10790.

Kania, M., Szymiczek, K., Labaj, W., Foszner, P., Gruca A., Szczęsna A., Polanski A., Computational methods for modelling cancer clonal evolution, 2022,

(in press), POB2, Artificial Intelligence and Data Processing, Silesian University of Technology Publishing House.

1.4 Code availability

The code is available on the github platform, by the link or code QR below.

To install the package, R should be installed. It is free, open-source programming environment, available on the website: **https://www.r-project.org/**

Commands to execute after opening R:

```
install.packages("devtools")
install_github("callimae/multivarEM")
```

2 Model-based algorithms

In this chapter, we describe model-based, unsupervised clustering algorithms implemented in the thesis using mixtures of multivariable distributions. In the beginning, we list the probability distribution used in modelling. Then, we introduce models of mixtures and related concepts. Finally, we present algorithms constructed with the help of the expectation maximization (EM) method.

2.1 Foundation

2.1.1 Probability distribution models

Probability distributions more or less accurately reflect natural phenomena around the world.

The basis for unsupervised clustering is parametric, multivariate probability distribution models, which we described in this subsection. Two models are suitable and often applied in multivariate distributions, multivariate Gaussian and multinomial distributions.

2.1.1.1 Univariate and Multivariate Gaussian distribution

Univariate Gaussian (normal) distribution The normal distribution has two parameters: μ the mean value and σ, the standard deviation. The mathematical notation of normal distribution is $X \sim N(\mu, \sigma^2)$. Independently of mean and standard deviation values, all normal distributions have symmetric, bell-curved shapes.

The standard normal distribution is the normal distribution which has a mean equal to 0 and a standard deviation equal to 1. The following formula shows the probability density function[7]:

$$f(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp^{\frac{-(x-\mu)^2}{2\sigma^2}}, \tag{2.1}$$

where: x is observation, μ is a mean, and σ is standard deviation

Multivariate Gaussian (normal) distribution The multivariate normal distribution is a generalization of the univariate normal distribution. It may have n dimensions where $n \in \{0, \infty\}$. The multivariate normal distribution plays a fundamental role in a multivariate analysis, thanks to its various properties. While it is true that real data is never exactly multivariate normal, it is often helpful to use normal density because of its close approximation to the "true" population distribution.

Due to a central limit theorem, the sampling distributions of many multivariate statistics are approximately normal, despite of the form of the parent population. An n-dimensional random variable X with mean vector and covariance matrix Σ is said to have a non-singular multivariate normal distribution when its density function is of the form[7]:

$$f(x, \mu, \Sigma) = \frac{1}{(2\pi)^{M/2}|\Sigma|^{1/2}} \exp^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{2.2}$$

where:

$x = [x_1, x_2...x_M]$ - a vector of observations

$\mu = [\mu_1, \mu_2, \ldots, \mu_M]$ - a vector of means

$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1M} \\ \sigma_{12} & \sigma_{22} & \cdots & \sigma_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{M1} & \sigma_{M2} & \cdots & \sigma_{MM} \end{bmatrix}$ is a covariance matrix

$|\Sigma|$ - denotes matrix determinant

x^T - stands for vector x transposition.

Multivariable diagonal normal distribution In the case of normal distributions, an important aspect is a high requirement for computational power and memory requirement for multidimensional cases. If we have an observation given by a vector with $1000 = 10^3$ entries, the size of the covariance matrix will be $\dim(\Sigma) = 10^3 \times 10^3$, which requires one million records of memory space. This calls for more efficient approaches to handling this kind of data.

We define *multivariable diagonal normal distribution* as a multivariable normal distribution with a diagonal covariance matrix. Elements of observation vector x are uncorrelated, so we use the index "U" to distinguish it. Its probability density function is therefore defined as follows

$$f_U(x, \mu, \Sigma_U) = \frac{1}{(2\pi)^{M/2}|\Sigma_U|^{1/2}} \exp^{-\frac{1}{2}(x-\mu)^T\Sigma_U^{-1}(x-\mu)} \tag{2.3}$$

where:

$x = [x_1, x_2...x_M]$ - a vector of (uncorrelated) observations,

$\mu = [\mu_1, \mu_2, \ldots, \mu_M]$ - vector of means,

$\Sigma_U = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_M^2 \end{bmatrix}$ - a diagonal covariance matrix,

$|\Sigma_U|$ - denotes matrix determinant,

Σ_U^{-1} - superscript -1 denotes inverse of the matrix,

x^T - stands for vector x transposition.

2.1.1.2 Bernoulli, binomial and multinomial distribution

In order to present multinomial distributions, we begin with Bernoulli and binomial distributions. The reason is that the binomial distribution is a generalization of Bernoulli and a multinomial generalization of the binomial distribution. We can express the values of those distributions as non-negative integers.

Bernoulii distribution If we consider a probabilistic experiment with two outcomes, it is called a Bernoulli trial. The result might be a success with a probability of p or failure with a probability of $1 - p$. An intuitive example of a Bernoulli trial might be a quality check of products in the factory. It is either a success or a failure.

Binomial distribution Binomial distribution describes results of repeating Bernoulli trials with probability of success p. The most common example is tossing a coin a finite number of times and more than one. We can assume that the tail is a success and the head is a failure. The following formula gives a binomial distribution probability function[7]:

$$Pr(k, n, p) = \begin{pmatrix} \mathrm{n} \\ \mathrm{k} \end{pmatrix} p^{\mathrm{k}}(1-p)^{\mathrm{n-k}}, \quad \mathrm{n} = 0, 1, \ldots, n. \tag{2.4}$$

in the above:
$\begin{pmatrix} \mathrm{n} \\ \mathrm{k} \end{pmatrix}$ - binomial coefficient,
n - number of trials,
k - number of successes,
p^{k} - probability of success,
$(1-p)^{\mathrm{n-k}}$ - probability of failure.

Multinomial distribution We can consider multinomial distribution as a multidimensional generalization of the binomial distribution. It inherits binomial properties and introduces new ones. The name "multi" suggests that we have more than two categories. A typical example of multinomial distribution is rolling a die a fixed number of times. Whether the die is fair or not, each side, called category, has some probability p. As a different case, consider testing the durability of an intricate car component under crash conditions. The part may be damaged in different ways, each with distinct probabilities. We could apply the multinomial distribution to estimate the probability of a particular combination of failures.

The following equation describes the multinomial distribution probability function [8]:

$$Pr(N, x, p) = \frac{N!}{n_1! \cdots n_M!} p_1^{n_1} \cdots p_M^{n_M} = N! \prod_{i=1}^{M} \left(\frac{p_i^{n_i}}{n_i!} \right), \tag{2.5}$$

$$n_1 + n_2 + ... + n_M = N \tag{2.6}$$

$$p_1 + p_2 + ... + p_M = 1 \tag{2.7}$$

in the above:

N - the number of trials,

$n = [n_1, n_2...n_M]$ - one observation vector of recorded counts of categories,

$p = [p_1, p_2, ...p_M]$ - a vector of probabilities of categories

2.1.1.3 Mixture distributions

It is a worth reading story about mixtures related to the biologist Raphael Weldon and mathematician Karl Pearson, which presents probably the first mathematical approach to a mixture distribution [9].

A straightforward but less common example of the mixture distribution might be shown upon different races of dogs. There is a significant difference between, e.g. labrador and chihuahua. Those differences account for the weight, height, but also size of organs as well. If we combine such measurements from many dogs, we will receive a mixture of dog breeds. Then, if we would like to describe such data with single multivariate normal distribution, we will lose much information.

In general, mixture models provide a broader spectrum of information than single distributions.

In medicine, they might be used to analyze gene expression data or in early drug development [10]. They are also successfully used to approximate specificity and sensitivity in the case of a lack of the golden standard. Albeit, mixture models are broader than just biology and medicine. Fields like astronomy, psychology and engineering, to name a few, also benefit from them. Their flexibility and usefulness are described in several books [11][10]

A mixture distribution is a mixture of at least two distributions of the same or different types. As an example, the discrete case of distribution of weight in the population of adults might be expressed as follows:

$$w(weight) = p(man) * w_{man}(weight) + p(woman)w_{woman}(weight|woman)$$

Where the probabilities p(man), p(woman) are also called mixing probabilities or proportions w_{man} and w_{woman} are probability density functions of weights of man and woman.

More formally, if a random variable or random vector x, takes values in sample space, Ω, a K component mixture distribution $f(x)$ is represented as follows:

$$f(x) = \alpha_1 f_1(x) + \cdots + \alpha_K f_K(x) \qquad (x \in \Omega) \tag{2.8}$$

where:

$\alpha_j > 0, \quad j = 1, \ldots, K; \quad \alpha_1 + \cdots \alpha_K = 1$, - mixing proportions or component weights,

$f_j(x), j = 1, \ldots, K$ - probability density functions of K component distributions

Most often component densities have forms dependent on parameters,

$$f_k(x) = f_k(x \mid \Theta_k)$$

thus we write

$$f(x|\Theta) = \alpha_1 f_1(x \mid \Theta_1) + \cdots + \alpha_K f_K(x \mid \Theta_K) = \sum_{k=1}^{K} \alpha_k f_k(x \mid \Theta_k) \tag{2.9}$$

where:
α_k- mixing proportion of the k-th component,
Θ_k- parameters of of the k-th component,
$f_k(x \mid \Theta_k)$ - probability density function of data x given parameters Θ_k

Mixtures of normal distributions Mixtures that consist of only normal distributions are either univariate or multivariate. Simply put, a mixture of univariate distribution consists of one only feature, while multivariate has many. Sometimes we also distinguish bivariate normal distributions.

Mixture of univariate normal distribution It is a mixture where each component that belongs to K follows its respective normal distribution.

A typical example of the mixture of the univariate normal distribution is the height of some populations. However, how we will create and interpret clusters depends on many circumstances. Let us consider the situation when we want to distinguish between a man's height and a woman's. When our data is labelled, we can use statistical tests to check if there are statistically significant differences. In the case of unlabelled data, we can no longer rely on statistical tests. We could no longer ask about differences between the two groups, as we are presented with one feature. At this point, we want to answer whether there are two subgroups in the given data. The potential trap here is that we will find as many clusters as we want. We need further analysis to verify how significant those results are for our research.

$$f(x|\mu, \sigma, \alpha) = \sum_{k=1}^{K} \alpha_k f_k(x|\mu_k, \sigma_k) \tag{2.10}$$

where:
$\mu = [\mu_1, \mu_2, ...\mu_K]$,
$\sigma = [\sigma_1, \sigma_2, ...\sigma_K]$,
$\alpha = [\alpha_1, \alpha_2, ...\alpha_K]$,
parameters of components are:
μ_k - a mean of the k-th component
σ_k - a standard deviation of the k-th component,
α_k - a mixing proportion of the k-th component.

Mixture of multivariate normal distributions Each component is a multivariable normal density function. The probability density function of the mixture is as follows

$$f(x|\mu, \Sigma, \alpha) = \sum_{k=1}^{K} \alpha_k f_k(x|\mu_k, \Sigma_k) \tag{2.11}$$

where:

arguments of the density functions are:

$\mu = [\mu_1, \mu_2, ...\mu_K]$ - a matrix composed of vectors of means,

$\Sigma = [\Sigma_1, \Sigma_2, ...\Sigma_K]$ - a list of covariance matrices,

$\alpha = [\alpha_1, \alpha_2, ...\alpha_K]$ - a vector of mixing proportions.

parameters of components are:

$\mu_k = [\mu_{k1}, \mu_{k2}, ...\mu_{KM}]$ - a vector of means of k-th component,

$\Sigma_k = \begin{bmatrix} \sigma_{k,11} & \sigma_{k,12} & \cdots & \sigma_{k,1M} \\ \sigma_{k,12} & \sigma_{k,22} & \cdots & \sigma_{k,2M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{k,M1} & \sigma_{k,M2} & \cdots & \sigma_{k,MM} \end{bmatrix}$ - a covariance matrix of k-th component,

α_k - a mixing proportion of k-th component

Mixture of diagonal multivariate normal distributions For a reason previously stated, we focus on the particular case of multivariable normal distributions - diagonal multivariable normal distribution. The probability density function of the mixture is as follows:

$$f(x|\mu, \Sigma^U, \alpha) = \sum_{k=1}^{K} \alpha_k f_k(x|\mu_k, \Sigma_k^U) \tag{2.12}$$

where:

$f(x|\mu, \Sigma^U, \alpha)$ - is probability density function of normally distributed random variable X given parameters,

arguments of the density function on the left hand side are:

$\mu = [\mu_1, \mu_2, ...\mu_K]$ - a matrix composed of vectors of means,

$\Sigma^U = [\Sigma_1^U, \Sigma_2^U, ...\Sigma_K^U]$ - a list of covariance matrices,

$\alpha = [\alpha_1, \alpha_2, ...\alpha_K]$ - a vector of mixing proportions

parameters of components are:

μ_k - a vector of means of k-th component,

$\Sigma_k^U = \begin{bmatrix} \sigma_{k,1}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{k,2}^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \sigma_{k,M}^2 \end{bmatrix}$ - a diagonal covariance matrix of k-th component,

α_k - a mixing proportion of k-th component.

Mixtures of multinomial distributions

Mixtures of binomial distributions Binomial mixtures consist of more than one binomial distribution and can be described by the following probability function

$$Pr(l, N, p) = \sum_{k=1}^{K} \alpha_k \begin{pmatrix} \text{n} \\ \text{k} \end{pmatrix} p_k^{\text{k}} (1 - p_k)^{\text{n-k}}, \quad l = 0, 1, \ldots, n, \tag{2.13}$$

where:

$\begin{pmatrix} \text{n} \\ \text{k} \end{pmatrix}$ - a binomial coefficient,

n - number of trials,

k - number of successes,

α_k - a mixing proportion of k-th component,

p_k^l - a probability of success for k-th component,

$(1 - p_k)^{N-l}$ - a probability of failure for k-th component

The above formula described one observation (i.e., l successes) from a mixture of k binomial distributions. Assume that we record D observations l_1, l_2,..., l_D.

Mixtures of multinomial distribution Below we present the formula for the probability function of K component mixtures of multinomial distributions for one observation:

$$p_{n_1,n_2,\ldots,n_M} = \frac{N!}{n_1!n_2!\ldots.n_M!} \sum_{k=1}^{K} \alpha_k p_{k1}^{n_1} p_{k2}^{n_2} \cdots p_{kM}^{n_M} \tag{2.14}$$

$$n_1 + n_2 + ... + n_M = N \tag{2.15}$$

$$p_1 + p_2 + ... + p_M = 1 \tag{2.16}$$

in the above:

N - the number of observations

K - the number of components in the mixture

α_k - a mixing proportion of k-th component

$n = [n_1, n_2 ... n_M]$ - a vector of observed counts of observed categories,

$p_k = [p_{k1}, p_{k2}, ... p_{kM}]$ - a vector of probabilities of categories for k-th component

2.2 EM algorithm

Clustering methods are based on a matrix of similarity or dissimilarity measures between objects. The purpose was to segregate data so that objects in one group were similar to themselves but different from the other groups. However, this approach still lefts many

questions unanswered. Which clustering method should we use for particular data? What to do with outliers or objects that did not fall into groups? How do we assess uncertainty about clustering results?

A statistical approach that uses probability distributions might address some of those questions. We can use it for clustering problems using the finite mixture model (FMM). Inside this model, each mixture component is described by a probability distribution. One of the first successful methods that used FMM and answered some of the questions was presented in the 1950s by Paul Felix Lazarsfeld. The model was called the latent class model, and it used discrete multivariate data as input. The model was based on the assumption that within each group, its characteristics were statistically independent [(Lazarsfeld, 1950a,c)]. In 1963, Wolfe introduced a model for clustering continuous data, along with the software NORMIX that he was developing. It allowed us to analyze mixtures of multivariate normal distributions. In his proposal, the estimation of model parameters was done by maximum likelihood using the Expectation-Maximization algorithm. It was followed with relevant theory in the next years (Wolfe, 1965, 1967, 1970) [5].

EM gained worldwide popularity after publishing it in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. The article "Maximum Likelihood from Incomplete Data via EM algorithm" presented a structured and general way to parameter estimation method based on the maximum-likelihood method. The algorithm was called **EM** since each step consisted of an **expectation** step, followed by a **maximization** step.[12].

The Expectation-Maximization (EM) algorithm is an iterative method for finding maximum likelihood (ML) estimates in problems with latent or hidden variables. The main idea behind EM algorithms is existing of a hidden or latent variable which is never directly observed. It is often referred to as a factor that combines several other variables and indicators. An example of such a hidden variable is the existence of groups among observations. The algorithm itself alternates between two steps: the E-step (Expectation), where the expected value of the latent variables given the current estimates of the parameters is computed. Then is the M-step (Maximization), where the parameters are re-estimated based on the expected values of the latent variables calculated in the E-step. These two steps are repeated until convergence occurs, i.e., the parameters' estimates stop changing.

The EM algorithm has become popular in machine learning and statistics because it can be applied to many models. The list includes Gaussian mixture models, hidden Markov models, missing data, truncated distributions, and censored or grouped data. As McLachlan points out, also in statistical models such as random effects, convolutions, log-linear models, latent class and latent variable structures. However, one of the limitations of the EM algorithm is that it can be sensitive to the initial parameter estimates and may converge to a local maximum of the likelihood function instead of the global maximum [6].

2.3 Multivariate Gaussian Mixture EM

2.3.1 Derivation of Univariate Gaussian Mixture EM

The steps below present the derivation of the Univariate Gaussian Mixture EM.

2.3.1.1 Likelihood and log-likelihood of the observed data

After observing some $x = [x_1, x_2, ...x_N]$ that is a vector of observations containing N observations, the likelihood will be written as follows:

$$l^O(x|\mu, \sigma, \alpha) = \prod_{i=1}^{N} \sum_{k=1}^{K} \alpha_k f_k(x_i, \mu_k, \sigma_k) \tag{2.17}$$

The notation of l^O indicates that the function describes the likelihood of the **observed data**. Then the **log-likelihood** function of the observed data will take the following form,

$$L^O(x|\mu, \Sigma, \alpha) = \sum_{i=1}^{N} \ln \left[\sum_{k=1}^{K} \alpha_k f_k(x_i, \mu_k, \sigma_k) \right] \tag{2.18}$$

However, the above log-likelihood function cannot be maximised analytically. One solution to that is the EM algorithm. It involves introducing hidden variables $z_1, z_2, ...z_N$ and the idea of complete data, as described in the following subsections.

2.3.1.2 Hidden variables

Hidden variables $z_1, z_2, ...z_N$ are defined as follows:

$z_i = k$ if observation x_i was generated by the component of the mixture of number k

2.3.1.3 Complete data

By combining observed and hidden variables, we define complete data vectors:

$$[x_i, z_i].$$

One can observe that maximizing likelihood becomes tractable under the assumption of the knowledge of complete data, as described below.

2.3.1.4 Likelihood and log-likelihood of the complete data

The **complete data likelihood,** thanks to introducing the hidden variables z_i, has the following simple form:

$$l^C(x, \mu, \Sigma, \alpha) = \prod_{i=1}^{N} \alpha_{z_i} f_{z_i}(x_i, \mu_{z_i}, \sigma_{z_i}) \tag{2.19}$$

Complete data have the C as superscript to distinguish it from the observed data. Now, we can compute the complete data log-likelihood function

$$L^C(x, \mu, \Sigma, \alpha) = \sum_{i=1}^{N} \ln \alpha_{z_i} + \sum_{i=1}^{N} \ln \left[f(x_i, \mu_{z_i}, \sigma_{z_i}) \right]. \tag{2.20}$$

Maximizing complete data log-likelihood function L^C can be done analytically, but we do not know the hidden variable z. Therefore we use the iterative approach.

2.3.1.5 Conditional distribution of hidden variables

The first thing needed is a parameter guess, denoted by the superscript letter "g", to initialize the iterations and calculate the conditional distribution of the hidden variable through the application of Bayes Theorem.

$$p(z_i = k | \mu_k^{\mathrm{g}}, \sigma_k^{\mathrm{g}}, \alpha_k^{\mathrm{g}}) = p(k \mid i) = \frac{\alpha_k^{\mathrm{g}} f\left(x_i, \mu_k^{\mathrm{g}}, \sigma_k^{\mathrm{g}}\right)}{\sum_{\chi=1}^{K} \alpha_\chi^{\mathrm{g}} f\left(x_i, \mu_\chi^{\mathrm{g}}, \sigma_\chi^{\mathrm{g}}\right)} \tag{2.21}$$

Likelihood is first multiplied by its corresponding prior probability (mixing proportion). Then, the resulting value is standardized by summing it with the product of the likelihood and prior of all the other mixture components. This computation yields the posterior probability of the variable z_i, generated by the k-th mixture component.

2.3.1.6 Conditional expectation of the log likelihood function (Q-function)

Using 2.21 we can derive conditional expectation of the log-likelihood function (2.20). Following nomenclature often used in the literature we also call this conditional expectation a Q-function.

$$E\left(L^C \mid \text{ data, parameter guess }\right) = Q =$$
$$= \sum_{i=1}^{N} \sum_{k=1}^{K} (\ln \alpha_k)\, p(k \mid i) + \sum_{i=1}^{N} \sum_{k=1}^{K} \left[\ln(\frac{\pi}{2}) - \ln \sigma_k - \frac{1}{2} \frac{(x_i - \mu_k)^2}{\sigma^2} \right] p(k \mid i) \tag{2.22}$$

In the above equation, terms were already explained before.

Maximizing Q-function with respect to parameters α_k, μ_k, σ_k we obtain

$$\hat{\alpha}_k = \frac{\sum_{i=1}^{N} p(k \mid i)}{N} \tag{2.23}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} x_i p(k \mid i)}{\sum_{i=1}^{N} p(k \mid i)}, \quad k = 1, 2, \ldots, K \tag{2.24}$$

$$(\hat{\sigma}_k)^2 = \frac{\sum_{i=1}^{N} (x_i - \hat{\mu}_k)^2 \, p(k \mid i)}{\sum_{i=1}^{N} p(k \mid i)} k = 1, 2, \ldots, K \tag{2.25}$$

2.3.2 Derivation of Multivariate Gaussian Mixture EM

2.3.2.1 Likelihood and log-likelihood of the observed data

To describe the likelihood of the observed data $x = [x_1, x_2, ...x_N]$, where x is a matrix of observations consisting of M-dimensional vectors $x_1, x_2, ...x_N$, we can use the following expression:

$$l^O(x|\mu, \Sigma, \alpha) = \prod_{i=1}^{N} \sum_{k=1}^{K} \alpha_k f_k(x_i, \mu_k, \Sigma_k) \tag{2.26}$$

Once again l^O indicates that the function describes the likelihood of the **observed data**. Its logarithmic equivalent will take following form,

$$L^O(x|\mu, \Sigma, \alpha) = \sum_{i=1}^{N} \ln \left[\sum_{k=1}^{K} \alpha_k f_k(x_i, \mu_k, \Sigma_k) \right] \tag{2.27}$$

In the form above, multiplication was changed to summation. However, in this state, the function is not possible to maximize analytically. To overcome this, we use the EM algorithm, which involves introducing hidden variables $z_1, z_2, ...z_N$ and the idea of complete data as described in the following subsections.

2.3.2.2 Hidden variables

Hidden variables $z_1, z_2, ...z_N$ are defined as follows:

$z_i = k$ if observation x_i was generated by the component of the mixture of number k

2.3.2.3 Complete data

Complete data vectors are defined by merging the observed and hidden variables:

$$[x_i, z_i]$$

One can observe that under the assumption of the knowledge of complete data the task of maximizing likelihood becomes tractable, as described below.

2.3.2.4 Likelihood and log-likelihood of the complete data

Incorporating the hidden variables z_i into the likelihood expression results in a following complete data likelihood equation:

$$l^C(x, \mu, \Sigma, \alpha) = \prod_{i=1}^{N} \alpha_{z_i} f_{z_i}(x_i, \mu_{z_i}, \Sigma_{z_i}) \tag{2.28}$$

Complete data have the C as superscript to distinguish it from the observed data. Now, we can compute complete data log-likelihood function

$$L^C(x, \mu, \Sigma, \alpha) = \sum_{i=1}^{N} \ln \alpha_{z_i} + \sum_{i=1}^{N} \ln \left[f(x_i, \mu_{z_i}, \Sigma_{z_i}) \right] \tag{2.29}$$

Analytical maximization of the log-likelihood function L^C, which requires complete data, is feasible. However, an iterative method is employed since the hidden variable z_i needs to be estimated.

2.3.2.5 Conditional distribution of hidden variables

Some values of parameters are accepted as initial values for iterations and referred to as parameter guess, indicated by the letter "g" in superscript. The conditional distribution of the hidden variable using parameter guess can be calculated using Bayes Theorem.

$$p(z_i = k | \mu_k^{\mathrm{g}}, \Sigma_k^{\mathrm{g}}, \alpha_k^{\mathrm{g}}) = p(k \mid i) = \frac{\alpha_k^{\mathrm{g}} f\left(x_i, \mu_k^{\mathrm{g}}, \Sigma_k^{\mathrm{g}}\right)}{\sum_{\chi=1}^{K} \alpha_\chi^{\mathrm{g}} f\left(x_i, \mu_\chi^{\mathrm{g}}, \Sigma_\chi^{\mathrm{g}}\right)} \tag{2.30}$$

The likelihood is multiplied by its prior probability (mixing proportion). Then the standardization is done by summing the likelihood multiplied by the prior of all other mixture components. The result is the posterior probability of z_i generated by k mixture component.

2.3.2.6 Conditional expectation of the log likelihood function (Q-function)

Using 2.30 we can derive conditional expectation of the log-likelihood function (2.29). Following nomenclatrure often used in the literature we also call this conditional expectation a Q-function.

$$E\left(L^C \mid \text{ data, parameter guess }\right) = Q =$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} (\ln \alpha_k)\, p(k \mid i) + \sum_{i=1}^{N} \sum_{k=1}^{K} \left[-\frac{M}{2} \ln(2\pi) - \frac{1}{2} \ln |\Sigma_k| - \frac{1}{2} (x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k) \right] p(k \mid i) \tag{2.31}$$

In the equation above terms were already explained before. M is a number of dimensions. Maximizing Q-function with respect to parameters α_k,μ_k,Σ_k we obtain

$$\hat{\alpha}_k = \frac{\sum_{i=1}^{N} p(k \mid i)}{N} \tag{2.32}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} x_i p(k \mid i)}{\sum_{i=1}^{N} p(k \mid i)}, \quad k = 1, 2, \ldots, K \tag{2.33}$$

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^\top p(k \mid i)}{\sum_{i=1}^{N} p(k \mid i)} \tag{2.34}$$

2.3.3 Derivation of Diagonal Multivariate Gaussian Mixture EM

2.3.3.1 Likelihood and log-likelihood of the observed data

If we observed some $x = [x_1, x_2, ...x_N]$ which is a matrix of observations containing M - dimensional vectors $x_1, x_2, ...x_N$, to describe its likelihood of the data we have:

$$l^O(x|\mu, \Sigma^U, \alpha) = \prod_{i=1}^{N} \sum_{k=1}^{K} \alpha_k f_k(x_i, \mu_k, \Sigma_k^U) \tag{2.35}$$

Once again l^O indicates that the function describes the likelihood of the **observed data**. Its logarithmic equivalent will take following form,

$$L^O(x|\mu, \Sigma^U, \alpha) = \sum_{i=1}^{N} \ln \left[\sum_{k=1}^{K} \alpha_k f_k(x_i, \mu_k, \Sigma_k^U) \right] \tag{2.36}$$

In the form above, multiplication was changed to summation. However, in this state, the function is not possible to maximize analytically. To overcome this, we use the EM algorithm, which introduces hidden variables $z_1, z_2, ...z_N$ and the idea of complete data as described in the following subsections.

2.3.3.2 Hidden variables

Hidden variables $z_1, z_2, ...z_N$ are defined as follows:

$z_i = k$ if observation x_i was generated by the component of the mixture of number k

2.3.3.3 Complete data

Complete data vectors are formed by the combination of observed and hidden variables.

$$[x_i, z_i]$$

If we assume that we know all the data completely, then we can optimize the likelihood is feasible, as explained below.

2.3.3.4 Likelihood and log-likelihood of the complete data

The **complete data likelihood,** thanks to introducing the hidden variables z_i, has the following simple form

$$l^C(x, \mu, \Sigma^U, \alpha) = \prod_{i=1}^{N} \alpha_{z_i} f_{z_i}(x_i, \mu_{z_i}, \Sigma^U_{z_i}) \tag{2.37}$$

Complete data have the C as superscript to distinguish it from the observed data. Now, we can compute complete data log-likelihood function

$$L^C(x, \mu, \Sigma^U, \alpha) = \sum_{i=1}^{N} \ln \alpha_{z_i} + \sum_{i=1}^{N} \ln \left[f(x_i, \mu_{z_i}, \Sigma^U_{z_i}) \right] \tag{2.38}$$

Analytical maximization of the log-likelihood function L^C, which requires complete data, is feasible. However, an iterative method is employed since the hidden variable z_i needs to be estimated.

2.3.3.5 Conditional distribution of hidden variables

We accept some values of parameters as initial values for iterations. We call this parameter guess. We indicate those guessed parameters by the letter g in superscript. Using Bayes Theorem, we can calculate the conditional distribution of the hidden variable using parameter guess.

$$p(z_i = k | \mu^{\mathrm{g}}_k, (\Sigma^U)^{\mathrm{g}}_k, \alpha^{\mathrm{g}}_k) = p(k \mid i) = \frac{\alpha^{\mathrm{g}}_k f\left(x_i, \mu^{\mathrm{g}}_k, (\Sigma^U)^{\mathrm{g}}_k\right)}{\sum_{\chi=1}^{K} \alpha^{\mathrm{g}}_\chi f\left(x_i, \mu^{\mathrm{g}}_\chi, (\Sigma^U)^{\mathrm{g}}_\chi\right)} \tag{2.39}$$

The likelihood is multiplied by its prior probability (mixing proportion). Then the standardization is done by summing the likelihood multiplied by the prior of all other mixture components. The result is the posterior probability of z_i generated by k mixture component.

2.3.3.6 Conditional expectation of the log likelihood function (Q-function)

Using 2.39 we can derive conditional expectation of the log-likelihood function (2.38). Following nomenclatrure often used in the literature we also call this conditional expectation a Q-function.

$$E\left(L^C \mid \text{ data, parameter guess }\right) = Q =$$
$$= \sum_{i=1}^{N}\sum_{k=1}^{K} (\ln \alpha_k)\, p(k \mid i) + \sum_{i=1}^{N}\sum_{k=1}^{K} \left[-\frac{M}{2}\ln(2\pi) - \frac{1}{2}\ln\left|\Sigma^U_k\right| - \frac{1}{2}(x_i - \mu_k)^T (\Sigma^U)^{-1}_k (x_i - \mu_k) \right] p(k \mid i) \tag{2.40}$$

In the equation above terms were already explained before. M is a number of dimensions.

Maximizing Q-function with respect to parameters α_k,μ_k,Σ_k we obtain

$$\hat{\alpha}_k = \frac{\sum_{i=1}^{N} p(k \mid i)}{N} \tag{2.41}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} x_i p(k \mid i)}{\sum_{i=1}^{N} p(k \mid i)}, \quad k = 1, 2, \ldots, K \tag{2.42}$$

$$\hat{\Sigma}_k^U = \frac{\sum_{i=1}^{N} [\mathrm{diag}\,(x_i - \hat{\mu}_k)]^2 p(k \mid i)}{\sum_{i=1}^{N} p(k \mid i)} \tag{2.43}$$

where:

$\mathrm{diag}(y) = \begin{bmatrix} y_1 & 0 & \cdots & 0 \\ 0 & y_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & y_M \end{bmatrix}$ - is a diagonal matrix composed with elements of a vector y

$y = [y_1, y_2, ..., y_M]^T$

2.3.4 Implementation

The implementation part consist of a way of implementing algorithm in the R programming language.

Algorithm 2.1 Pseudocode of EM algorithm for Multivariate Gaussian Mixture Model

Input : **X**: data matrix, k: number of clusters, em.itr: maximum number of iterations
Output: **pki**: log-likelihood of the prior, means: cluster means, vars: cluster variances, alpha: cluster mixing proportions

change ← 100; *itr* ← 0
pki ← matrix with nrow(**X**) rows and k columns
for *i* ← *1* **to** *100* **do**
  clusters ← sample(1:nrow(**X**), size = nrow(**X**))
  means ← applyByCluster(X = **X**, fun = colMeans, by = clusters)
  vars ← abs(means) + 1e-4
  varmin ← *vars*
  alpha ← runif(k, min = 0.05); *alpha* ← *alpha*/sum(*alpha*)
  for *j* ← *1* **to** *k* **do**
    pki[,j] ← LLmvnorm(**X**, *means*[j,], *vars*[j,], *alpha*[j], ncol(X))
  end
  iniLL ← log(sum(exp(**pki**)))
end
while *change* > 1*e* − 8 ***and*** *itr* ≤ *em.itr* **do**
  $means_o$ ← *means*; $vars_o$ ← *vars*; $alpha_o$ ← *alpha*
  for *j* ← *1* **to** *k* **do**
    pki[,j] ← LLmvnorm(t(**X**), means[j,], vars[j,], alph[j], ncol(X))
  end
  denum ← colMaxs(**pki**)
  post ← **pki** - denum
  post_plus ← exp(post - rowLogSumExps(post))
  alphas ← colsums(*post_plus*)/nrow(*post*)
  means ← crossprod(post_plus, **X**)/colsums(post_plus)
  for *j* ← *1* **to** *k* **do**
    xvar ← tcrossprod(*post_plus*[,j], ((t(**X**) - means[j,])^2))
    vars[j,] ← c(*xvar*/sum(*post_plus*[,j]))
    vars[j,][vars[j,] ≤ 0] ← varmin[j,][vars[j,] ≤ 0]
  end
  varmin ← *vars*
  change ← abs(sum($means_o$ - *means*)) + abs(sum($vars_o$ - *vars*)) + abs(sum($alpha_o$ - *alpha*))
  itr ← itr + 1
end
return **pki**, means, vars, alphas

2.3.4.1 Initialization

The initialization step requires first estimates of the parameters: μ_k, $\hat{\Sigma}_k^U$, and α_k, of the k-th component. It was already shown in the literature that a good choice of starting point for EM might be deciding factor for its quick and correct convergence [13][14]. We have tested two types of initialization - random and based on k-means.

Random initialization - means

The first choice was **random sampling** from the dataset. Given K mixture components, I created a vector whose length was equal to the number of observations N in the dataset. Each k should appear at least once to provide a starting point for each component. Then I calculated their means within each k.

k-means initialization - means

I based k-means on the armadillo library as it was already parallelized and offered random initialization and enhanced **kmeans**++ variant. Here, centroids were used instead of cluster assignments.

Variance

The variance was calculated in the same way despite the initialization method. To calculate initial variances, the absolute value of the means was multiplied by $1e-4$. In this way, no initial variance contained negative or zero values, as they would result in a numerical error in the later calculations.

A copy matrix of variances was created to replace any variances that will become zero or less.

Mixing proportions

Mixing propotrions were sampled from **uniform** distribution, were minium value was 0.1 and maximum 0.9.

On top of that, intialization was repeated many times, in order to found the highest initial likelihood.

2.3.4.2 E-step

In the numerator of 2.39 we obtain the likelihoods of the values x_n (observations) over the guessed parameters μ_k (mean), $\hat{\Sigma}_k^U$ and α_k. Next, they are multiplied by the probability of occurrence (mixing proportion of k-th element). The results are interpreted as the probability that observations$x_n$belong to the normal distribution with parameters μ_k, Σ_k^U, and α_k. We repeat this step for all k components. The denominator is used to normalize the results so that they can sum up to 1. However, calculating probability density should be done on the

logarithmic scale. It is crucial, especially in multidimensional data. Any value below 1e-323 will become zero and might result in a numerical error. The other important thing in the E-step is that only diagonal part of covariance matrix was used. The result of the expectation step is a $p(k|i)$ matrix that contains likelihood values for each k group. This matrix will be used in the maximization step.

2.3.4.3 Maximization step

Since we know the likelihood of the data under k different parameters, they can be updated with obtained equations. The first equation will allow us to calculate new mixing proportions where we sum the likelihood of k columns and divide by the number of observations N. To calculate new vectors of $\hat{\mu}_k$we had to multiplicate the data by likelihood and divide by the sum of likelihood. Finally, we subtract the mean value from the observations and multiply it by the likelihood of updating the variances. We sum all the terms in the numerator across the columns to standardize them by the sum of likelihood.

Since we are dealing with a log-likelihood matrix, we can standardize the log-likelihood matrix, staying on the logarithmic scale. We can take the largest value from each row and then subtract created vector from each matrix column. This result will be standardized again, but now using the LSE trick.

Small numbers and LSE trick

One of the challanges in machine learning are extremaly small numbers. Following statements were produced in R software:

$$1e{-}323 > 0; \quad 1e{-}324 == 0$$

The first statement returns **TRUE**, and the second statement also returns **TRUE**. In the example above, we checked if one after 323 decimal places is bigger than zero. The result was, of course, accurate. However, the result was false when we tested whether one after 324 decimal places was bigger. Another test returned true for equality between this value and zero. Value 1e-324 was approximated to zero. R language uses the IEEE 754-2008/IEC60559:2011 standard, called the 'binary64' format or double-precision floating point. Computers use floating points to represent fractions of values. The problem with tiny numbers is common in computer sciences, especially machine learning. Although it is not a rule, it might occur if the data has many dimensions, like in multivariate datasets.

The exemplary algorithm that should address this kind of issue is multidimensional Expectation-Maximization. Because of its nature, it often has to deal with exceptionally small or large numbers. When calculating variances or the probability of categories, a similar will be valid for overly big numbers. Moreover, we will face a similar problem operating on high numbers, like $1.124124e{+}308$. Shown example is the limit in R, after which we will receive the inf value. Staying on a logarithmic scale allows us to avoid such problems in both cases. However, to

standardize logarithmic values, we cannot simply add them because such an operation equals multiplication.

Beneficial is Log-Sum-Exp (LSE) function. It allows for logarithms standardization, avoiding zero error at the same time. The general formula for the LSE is as below [15]:

$$\mathrm{LSE}(x_1, x_2...., x_M) = \ln(\exp(x_1) + \ln(\exp(x_2) + ... + \ln(\exp(x_M)) \tag{2.44}$$

where:
x_M- observation
ln - natural logarithm
exp - exponentiation

2.3.4.4 Stop criteria

The algorithm finishes when convergence occurs. It might be done in several ways. One is to use a change in the parameters or stop after some number of iterations. The algorithm stopped when the absolute change between old and new parameters was less than 1e-8. Otherwise, it stopped after 500 iterations.

2.3.5 Key points

Despite the R language in which the EM algorithm was implemented, it is considerably fast. Mainly because only diagonal variances are calculated instead of a full covariance matrix. However, it comes at the cost of accuracy.

2.3.6 Methods of initialization

To compare initial conditions of convergence, a few initialization methods were compared:

- random
- k-means with a random subset

In addition to different types of initialization, the capabilities of some existing GMM algorithms have also been examined. One of the most popular is Mclust, an excellent implementation of the EM algorithm. However, a considerable drawback regarding multidimensional data has been identified in it. The full covariance matrix is computed, making it exceptionally computationally demanding.

Here, another version of GaussEM has been presented, which should perform well even with limited resources.

2.4 Multinomial Mixture EM

2.4.1 Derivation of Multinomial Mixture EM

2.4.1.1 Likelihood and log-likelihood of the observed data

The likelihood function (l^O) of the observed data for mixture of the multinomial distribution is as follows:

$$l^O = \prod_{d=1}^{D} \frac{N!}{n_{d1}!n_{d2}!\ldots.n_{dM}!} \sum_{k=1}^{K} \alpha_k p_{k1}^{n_{d1}} p_{k2}^{n_{d2}} \ldots p_{kM}^{n_{dM}} \tag{2.45}$$

$$n_1 + n_2 + ... + n_M = N \tag{2.46}$$

$$p_1 + p_2 + ... + p_M = 1 \tag{2.47}$$

in the above:

N - the number of observations

k - the number of components in the mixture

α_k - a mixing proportion of component k

$n = [n_1, n_2...n_M]$ - vector of observed counts of observed categories,

$p = [p_1, p_2, ...p_M]$ - vector of probabilities of categories

It is often helpful to change the likelihood function to log-likelihood. It will make partial derivatives easier to obtain. Thus, changing to logarithm we will have

$$L^O = C + \sum_{d=1}^{D} \ln \left[\sum_{k=1}^{K} \alpha_k p_{k1}^{n_{d1}} p_{k2}^{n_{d2}} \ldots p_{kM}^{n_{dM}} \right] \tag{2.48}$$

In the above:

C is $\frac{N!}{n_{d1}!n_{d2}!....n_{dM}!}$ which is constant term

However, as before, this function is difficult to maximize. The solution for that is to introduce hidden variable z.

2.4.1.2 Hidden variables

Similarly as in the previous cases, hidden variables $z_1, z_2, ...z_N$ are defined as follows:

$z_i = k$ if observation x_i was generated by the component of the mixture of number k

2.4.1.3 Complete data

By combining observed and hidden variables, we define complete data vectors:

$$[x_i, z_i].$$

One can observe that maximising likelihood becomes tractable under the assumption of the knowledge of complete data, as described below.

2.4.1.4 Likelihood and log-likelihood of the complete data

The function for the **complete data likelihood,** together with hidden variablez_i, takes the following form:

$$l^C = \frac{N!}{n_{d1}!n_{d2}!\ldots.n_{dM}!}\prod_{d=1}^{D}\alpha_{zd}p_{zd,1}^{n_{d1}}p_{zd,2}^{n_{d2}}\cdots p_{zd,M}^{n_{dM}} \tag{2.49}$$

And its log-likelihood equivalent:

$$L^c = C + \sum_{n=1}^{N}\left[\ln\left(\alpha_{z_n}\right) + \sum_{m=1}^{M} n_{dm}\ln\left(p_{zd,m}\right)\right] \tag{2.50}$$

Although the complete data log-likelihood function L^C can be maximized analytically, the hidden variable z is unnkown. Therefore we use the iterative approach.

2.4.1.5 Conditional distribution of hidden variables

As before, guessed parameters are denoted with "g" in superscript. They are needed initialize the iterations and calculate the conditional distribution of the hidden variable through the Bayes Theorem.

$$p(k|d) = \frac{\alpha_k^{\mathbf{g}} \times p_{k_1}^{\mathbf{g},nd_1} \times p_{k_2}^{\mathbf{g},nd_2} \times \ldots \times p_{k_M}^{\mathbf{g},nd_M}}{\sum_{\kappa=1}^{K}\ \alpha_\kappa^{\mathbf{g}} p_{\kappa_1}^{\mathbf{g},nd_1} \times p_{\kappa_2}^{\mathbf{g},nd_2} \times \ldots \times p_{\kappa_M}^{\mathbf{g},nd_M}} = \frac{\alpha_k^{\mathbf{g}}\prod_{m=1}^{M} p_{k_m}^{\mathbf{g},nd_m}}{\sum_{\kappa=1}^{K}\ \alpha_\kappa^{\mathbf{g}}\prod_{m=1}^{M} p_{\kappa_M}^{\mathbf{g},nd_M}} \tag{2.51}$$

To obtain the posterior probability of the variable z_i generated by the k-th mixture component, the corresponding prior probability (mixing proportion) is first multiplied by the likelihood. The resulting value is then standardized by adding it to the product of the likelihood and prior probabilities of all the other mixture components.

2.4.1.6 Conditional expectation of the log likelihood function (Q-function)

Using 2.51 we can derive conditional expectation of the log-likelihood function (2.50). Following the jargon often used in the literature, we call this conditional expectation a Q-function.

$$\begin{gathered} E\left(L^C \mid \text{ data, parameter guess }\right) = Q = \\ = C + \sum_{d=1}^{D}\left[\sum_{k=1}^{K}\ln\left(\alpha_k\right)p(k \mid d) + \sum_{m=1}^{M}\sum_{k=1}^{K} n_{dm}\ln\left(p_{km}\right)p(k \mid d)\right] \end{gathered} \tag{2.52}$$

Maximizing Q-function with respect to parameters α_k, p_k, we obtain:\

$$\hat{\alpha} = \frac{\sum_{d=1}^{D} p(k|d)}{D} \tag{2.53}$$

where:

$\hat{\alpha}$ - is vector of new mixing proportions

D - is the count of observations

$$\hat{p}_{km} = \frac{\sum_{d=1}^{D} n_{dm} p(k|d)}{\sum_{n=1}^{M} \sum_{d=1}^{D} n_{dm} p(k|d)} \tag{2.54}$$

where:

n_{dm} - is an observation vector

$\hat{p}_{km}$ - is a vector of new probabilities proportions

2.4.2 Implementation

In the case of multinomial mixture EM, process of implementation is similar with a few important differences.

2.4.2.1 Input data

For presented cases, we treat each observation as a row, and each variable as a column.

Data points belong to $\mathbb{N}$. The data shows counts of how many times observation was noted within each variable.

$$\begin{gathered} n_{11} + n_{12} + n_{1m} \ldots + n_{dM} \\ n_{21} + n_{22} + n_{2m} \ldots + n_{dM} \\ n_{d1} + n_{d1} + n_{dm} \ldots + n_{dM} \\ \vdots \\ n_{Dm} + n_{Dm} + n_{Dm} \ldots n_{DM} \end{gathered} \tag{2.55}$$

The usual requirement to start EM for multinomial mixture is to provide k number of clusters. Knowing the number of subgroups in the data, we are able to initialize parameters in the first step of an algorithm.

2.4.2.2 Initialization

During initialization, we need to create first guess of the parameters. In case of multinomial mixture we need to initialize **mixing proportions** (α) and **probabilities** (p) for each category/variable. This should be done for each mixture component $k \in \{1..K\}$.

Mixing proportions indicates how much of the mixture space belong to K. Depending on the number of components K we need to provide equal number of α_k 2.56.

$$\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \cdots \quad \alpha_k \tag{2.56}$$

Algorithm 2.2 Pseudocode of EM algorithm for Multinomial Mixture Model

Input : **X**, k, itrs = 1000
Output: List containing **pki** and parameters

Function *maximization(***pki**, **X***)*:
  alphas ← colSums(**pki**)/nrow(**X**)
  prob.mat ← abs(crossprod(**pki**, **X**))
  rS ← rowSums(prob.mat)
  rS[rS==0] ← min(rS[rS!=0])
  probs ← prob.mat/rS
  return *probs, alphas*
end

Let probs ← matrix of uniform probabilities from interval [0.1, 1]
Normalize each row of probs to have unit sum.
$\alpha \leftarrow \text{uniform}(k, \min = 0.1, \max = 1); \alpha \leftarrow \frac{\alpha}{\sum_{i=1}^{k} \alpha_i}$
$\delta \leftarrow 100$; itr ← 0
while $\delta > 1e-6$ ***and*** $itr \neq itrs$ **do**
  itr ← itr + 1
  probs.old ← probs
  alphas.old ← alphas
  prob.not.stand ← tcrossprod(**X**, log(probs)) + log(alphas)
  pki ← exp(prob.not.stand - logSumExp(prob.not.stand))
  alpha, probs ← maximization(**pki** = **pki**, **X** = **X**)
  $\delta \leftarrow$ sum(abs(alphas.old - alphas))+sum(probs.old - probs)
end
return pki = **pki**, *parameters* = *parameters*

Assume that we have a mixture where $k = 3$. In that case, we need to create 3 α parameters. We can use uniform distribution $\alpha_K \sim U(0.1, 1)$ to obtain initial alphas. It is better to keep the interval within the range of [0.1, 1]. Shallow values might cause over-dominance of larger α during the estimation step. After choosing values from the uniform distribution, they should be standardized.

$\hat{\alpha}_K = \frac{\alpha_1+\alpha_2+...+\alpha_k}{\sum_{k=1}^{K} \alpha_k}$ and $\sum_{k=1}^{K} \hat{\alpha}_k = 1$

The number of probabilities equals the number of dimensions/categories in the data. If the dataset consists of 10 categories, we should provide a probability for each category, for each element in K.

Probabilities depict how likely given variables will occur in the given group.

$$\begin{aligned} &p_{11} + p_{12} + \ldots + p_{kM} \\ &p_{21} + p_{22} + \ldots + p_{kM} \\ &p_{k1} + p_{k2} + \ldots + p_{kM} \\ &\vdots \\ &p_{Km} + p_{Km} + \ldots p_{KM} \end{aligned} \tag{2.57}$$

The first guess of the parameters, α_k(mixing proportion) and p_k of the k-th component, was done in two ways - randomly and based on the k-means results.

Random initialization - means

In the case of **random sampling**, for each k I sampled M observation from Poisson distribution. Lambda was equal to the number of M. Then, I standardized them, so they will sum up to 1 for each k.

k-means initialization - means

k-means initialization was based on the armadillo library. I used centroids and standardized them, to sum up to 1 for each k component.

Mixing proportions

The mixing proportions were already described above.

2.4.2.3 E-step

During the E-step we calculate likelihood for each observation. It is done by multiplying each mixing proportion parameter α_k by all probabiliby successes rised to their number of occurences in the data (respective observation). Then we standardize this value, by divinding numerator by the sum of calculated numerators for all of the components.

$$p(k|d) = \frac{\alpha_k \times p_{k_1}^{nd_1} \times p_{k_2}^{nd_2} \times ... \times p_{k_M}^{nd_M}}{\sum_{k=1}^{K} \alpha_k p_{k_1}^{nd_1} \times p_{k_2}^{nd_2} \times ... \times p_{k_M}^{nd_M}} = \frac{\alpha_k \prod_{m=1}^{M} p_{k_m}^{nd_m}}{\sum_{k=1}^{K} \alpha_k \prod_{m=1}^{M} p_{k_m}^{nd_m}} \tag{2.58}$$

The critical part here is that we can change to a logarithmic scale. Then, because of the law of logarithms, powers are changed to multiplication, and we can use matrix multiplication. It is usually much faster than creating any loop. After that, instead of multiplying by alpha, we subtract the logarithm of alpha. Standardization might be done using the already mentioned LSE trick. The values can then be safely exponentiated.

2.4.2.4 M-step

In the Maximization step, the model parameters are updated based on the posterior probabilities computed in the E-step. The goal is to maximize the expected complete data log-

likelihood function with respect to the model parameters derived from Q-function 2.53 and 2.54.

2.4.2.5 Stop criteria

Generally, the algorithm was stopped when the absolute change between old and new parameters was lower than 1e-8. In some cases, especially when the data was not suited, the algorithm finished after 1000 iterations.

3 Distance-based algorithms

We often perceive and describe the distance between objects as we constantly use various metric measures. For example, we must estimate the distance between cars to know if we can safely change lanes. When we are in a hurry, we might want to choose the shortest path to our destination.

In this chapter, we will present four algorithms based on distance functions. Those algorithms are based on various statistical distance metrics, like euclidean or manhattan.

3.1 Foundation

3.1.1 Input

Assume there are n objects, e.g. trees, people, stocks, words. To gather them in groups, we need some way to compare how similar or not they are. Clustering applications typically work on either two type of structure.

In the first structure, we can describe objects with their attributes. Those might be, e.g., height, weight, count of words or value. We can also provide the real value for each: 165 cm, 55 kg, 15 words or 17 euros. The first structure can be arranged into an $n \times m$ matrix, where rows correspond to objects and columns to the attributes. Such an objects-by-variables matrix is two-mode because rows and columns differ.

The second structure has the same set of objects in rows and columns. It contains two types of proximity measures between all pairs of objects. They are called similarity and dissimilarity. Similarity measures how much objects resemble each other, while dissimilarity estimates how far away two objects are from each other. This second structure, object-by-object matrix, is called one-mode [3].

3.1.2 Variables

3.1.2.1 Scale

Units of measures have a significant impact on clustering results in the case of vector data points. When expressed in different scales, some coordinates of data vectors can dominate clustering results. Therefore, aiming to reduce/compensate for that, we apply scaling and standardization procedures presented below.

3.1.2.2 Different variables

Different variables, i.e., different data vector entries, can exhibit ranges variability based on two factors. One factor will be that they are different types and measured in different units. Another factor will be that even when measured in the same system of units, they can exhibit physical differences leading to differences that need to be compensated.

3.1.2.3 Standardization

Standardization makes cluster structure more shallow by reducing the effect of large samples since it lowers their contribution. Standardization is an attempt to achieve objectivity across different units. Standardization makes data independent of unit measures. We can put all units on the same scale.

In the procedure of standardization, first, we have to calculate the mean value of some variable given by:

$$\mu = \frac{1}{n}\left(x_1 + x_2 + \cdots + x_i\right) \tag{3.1}$$

where: n - number of observations, μ - a mean, x_i - observation

Next, we have to calculate the measure of dispersion in the data. One of the most commonly used is **standard deviation**:

$$\sigma = \sqrt{\frac{1}{n-1}\left\{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_n - \mu)^2\right\}} \tag{3.2}$$

The equation for standardization is as follows:

$$z_n = \frac{x_i - \mu}{\sigma} \tag{3.3}$$

Sometimes those standardized measurements are called z-scores. They lack unit measures because the numerator and denominator are expressed in the same units.

3.1.3 Statistical distance

Statistical distance contains a broad spectrum of probability or information theory methods. Different measures can be used to quantify the distance between two statistical objects. It can be a distance between two probability distributions, random variables or samples. We present definitions and properties of distance functions and will show the most crucial function measures applied in this thesis.

Statistical distances are essential notions for developing applications of grouping algorithms. Four properties will characterize them.

No.	**Property**
1	$d(x, y) \geq 0$ (non-negativity)
2	$d(x, y) = 0$ if and only if $x = y$
3	$d(x, y) = d(y, x)$ (symmetry)
4	$d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

Table 3.1: Conditions to determine different types of metrics

1. Distances are always nonnegative, meaning that they can never be negative values.
2. The distance from an object to itself is always equal to 0, as an object is always the same distance away from itself.
3. The distance between object x and object y is equal to the distance between object j and object i, suggesting that the distance between two objects is independent of the order in which they are listed.
4. If there are three objects x, y, and z, and the distance from x to y is shorter than the combined distance of going through object z, then the shortest distance between x and y is the direct distance between them.

Generally, it is a distance or metric if all four conditions are met. It is essential to distinguish between them because when only the first and second property is satisfied, the statistical distance is called a divergence. In the algorithms that we are presenting, all four criteria are satisfied.

3.1.3.1 Distance between two points

A necessary step to quantify the degree of dissimilarity between objects is the choice of distance metric. We must compute the distance between each i and j pair of objects. One commonly used metric to do that is a Euclidean or Manhattan distance.

Euclidean distance

Pythagorean famous theorem allows us to calculate the hypotenuse of a perpendicular triangle. Since $AB^2 + BC^2 = AC^2$ then

$AC = \sqrt{(AB^2 + BC^2)}$. This straight line is the shortest distance between points A and C. Now, let us impose the triangle on the Cartesian plane and assume that two points are single objects. Each is described by x and y coordinates, so they are within two dimensions. Analogically as in the Pythagorean theorem, we can calculate the distance between objects A and C as $(A_{x1} - C_{x2})^2 + (A_{y1} - C_{y1})^2$.

The Euclidean distance is the straight-line distance between any two points. It is a non-negative real number such that $\mathbb{R}_{\geq 0} = \{x \epsilon R | x \geq 0\}$.

If we consider two points from distribution $X = (x_1, x_2 \dots x_n)$ and from distribution $Y = (y_1, y_2 \dots y_n)$ in Cartesian coordinates, then Euclidean distance, between those points are given by the general equation:

$$d_{Euc}(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots (x_n - y_n)^2} = \tag{3.4}$$

$$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{3.5}$$

where:

$\mathbf{X}, \mathbf{Y}$ - variables

$\mathbf{x_i}, \mathbf{y_i}$ - realizations of X and Y

$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$- root of sum of squares between all x_i and y_i

Euclidean distance is a practical and straightforward tool for finding the nearest hospital, considering an emergency helicopter flight (disregarding weather and terrain it is the shorthest path). It is increasingly used in molecular conformation in bioinformatics, localization of sensor networks, or dimensionality reduction methods [16][17].

Minkowski distance

Minkowski distance is a generalization of both Euclidean and Manhattan distance. It is also called Lp metric and is given by the following equation where p is any non-negative, real number.

$$d_{Mink}(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{3.6}$$

where:

$\mathbf{X}, \mathbf{Y}$ - variables

$\mathbf{x_i}, \mathbf{y_i}$ - realizations of X and Y

$\sum_{i=1}^{n} |x_i - y_i|^p$ - sum of absolute difference between all x_i and y_i raised to the power of p.

Minkowski distance is a generalization of Euclidean and Manhattan distance. If $p = 2$, it is the same as Euclidean distance; with $p = 1$, it is equivalent to Manhattan distance.

Minkowski distance is applied in fuzzy clustering, and all applications are from Euclidean distance and Manhattan distance.

Manhattan distance

Manhattan distance, also called city block distance, owns its peculiar name thanks to the following reasoning. Think of a city where streets are part of a grid, so we have only vertical and horizontal line segments. Suppose one wants to get from block **A** to block **B**. Then, the

shortest possible distance might be described by the following equation: $|x_A - x_B| + |y_A - y_B|$. It assumes a helicopter is out of the question and flying is generally impossible. Otherwise, the shortest path will be a euclidean distance. The following formula gives the Manhattan distance:

$$d_{Man}(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.7}$$

where:

$\mathbf{X}, \mathbf{Y}$ - variables

$\mathbf{x_i}, \mathbf{y_i}$ - realizations of X and Y

$\sum_{i=1}^{n} |x_i - y_i|^p$ - sum of absolute difference between all x_i and y_i

The Manhattan distance can be used in various applications such as image processing, pattern recognition, and data analysis. It is particularly useful when calculating the distance between discrete objects or points with known coordinates. [18]

3.2 Hierarchical clustering

Hierarchical clustering owns its name thanks to its structure and how it creates the clusters. It produces a series of partitions. Specifically, data points are gathered into clusters resembling an upside-down tree shape (pic!). The choice of a presented shape is arbitrary, as it might be circular or graph-like (pic!). What is shared by all three pictures is a visible hierarchy. Each data point is part of a systematically growing cluster that incorporates all the data at the end. To achieve this result, we can agglomerate or divide the data. In the first case, the agglomerative or top-down method, individual data points are grouped into larger clusters until they cover the data set. In the divisive or bottom-down approach, data is divided into smaller groups up to single data points[19].

3.2.1 Agglomerative clustering

Agglomerative clustering is the most common approach in Hierarchical Clustering.

Before performing clustering, two decisions has to be made. The first one is how one will measure the distance or similarity between points. It might be Euclidean, Manhattan distance or many others. The second decision is the linkage method. It dramatically influences how data points will be clustered in consecutive iterations. Some common examples are single, complete or average linkage. Following clustering, we can decide on any number of clusters without repeating the calculations. It is a characteristic feature of Hierarchical Clustering not found in other unsupervised algorithms.

In the first step, we need to measure how far points lie from each other. To do that, we can use similarity (e.g. Jaccard index) or distance (e.g. Euclidean) metric. The choice depends on scientific questions and the data itself. Next is the choice of linkage method. It will determine the way how the data will be clustered together. It has a tremendous impact on the results.

The last but important thing is deciding the number of clusters we would like to see. We are provided with two different approaches to that problem. The first one is to choose the number of groups exactly. Another way is to cut the branches at a specific tree height. The tree's height depends on the largest distance between the two clusters in the data. Thus choosing this way will give us groups of data with alike similarity or distances between each other [20] .

3.2.2 Divisive clustering

Divisive hierarchical clustering reverses the question presented in agglomerative hierarchical clustering. It asks how to divide trivial solutions with one cluster to n singleton clusters. This approach is much less common than agglomerative methods, primarily due to the extensive use of computational power. In divisive methods, one cluster is consecutively split into two smaller clusters. The challenge is how to find an optimal splitting space. Exploring all the possibilities is computationally expensive, as in the first step, $2^{(n-1)-1}$ ways exist to split the first cluster into two separate ones. The DIvisive ANAlysis Clustering (DIANA) presented one solution to this problem.

Initially, DIANA has to split the data into two separate groups. It is done without considering all possible divisions but rather iteratively. First, the average distance between the object and the group of objects is calculated for each of them. Then, the object with the most significant average dissimilarity is chosen to initiate the "splinter group". Next, average dissimilarity is calculated between the object and the remaining objects in the larger group. The results are compared with the splinter group. Object with the highest average dissimilarity difference is then moved to the splinter group, creating one cluster. [3].

3.2.3 Linkage methods

When the distance metric was calculated, it showed how similar or how close the objects were to each other. Linkage methods join such objects or observations using distance measures as a base. In other words, it is a function that creates clusters from the objects based on similarity. There are several linkage methods, each providing slightly different results. In the default, the R package provides eight different methods: single, complete, average, Ward's, Ward's 2, McQuitty, median and centroid. Some are presented below.

3.2.3.1 Single linkage

The two clusters with the smallest minimum pairwise distance are merged in each step.

$$l_s(X,Y) = \min_{x \in X, y \in Y} d(x,y) \tag{3.8}$$

where:

l_s - single linkage

X, Y - two sets of elements (clusters)

$\min_{x \in X, y \in Y} d(x, y)$ - minimal distance between the two elements $x \in X$ and $y \in Y$.

3.2.3.2 Complete linkage

In each step, the two clusters with the smallest maximum pairwise distance are merged.

$$l_c(X, Y) = \max_{x \in X, y \in Y} d(x, y) \tag{3.9}$$

where:

l_c - complete linkage

X, Y - two sets of elements (clusters)

$\max_{x \in X, y \in Y} d(x, y)$ - maximum distance between the two elements $x \in X$ and $y \in Y$.

3.2.3.3 Average linkage

It might be found under name Unweighted Pair Group Method with Arithmetic Mean or **UPGMA**. In this method, distance between points from both clusters is measured. Then, the average distance is calculated and clusters with smallest average distance are merged.

$$\frac{1}{|X| \cdot |\mathcal{Y}|} \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(x, y) \tag{3.10}$$

$$l_{(\mathcal{X} \cup Y), Z} = \frac{|\mathcal{X}| \cdot d_{\mathcal{X}, Z} + |\mathcal{Y}| \cdot d_{\mathcal{Y}, Z}}{|\mathcal{X}| + |\mathcal{Y}|} \tag{3.11}$$

where:

$l_{(\mathcal{X} \cup Y), Z}$ - average linkage

X, Y - two sets of elements (clusters)

$d(x, y)$ - distance between the two elements $x \in X$ and $y \in Y$.

3.2.3.4 Ward's

Ward's method shares some similarities with analysis of variance (ANOVA). It aims to minimize error sum of squares (ESS), when linking clusters together.

$$\mathrm{ESS}(X) = \sum_{i=1}^{N_X} \left| x_i - \frac{1}{N_X} \sum_{j=1}^{N_X} x_j \right|^2 \tag{3.12}$$

$$l(X, Y) = ESS(XY) - [ESS(X) + ESS(Y)] \tag{3.13}$$

where:

$\mathrm{ESS}(X)$ - error sum of squares of variable X

ESS(Y) - error sum of squares of variable Y
$d(x, y)$ - distance between the two elements $x \in X$ and $y \in Y$,
N_X - number of observations in set X
Ward's linkage method was used in the Hierarchical Clustering for all datasets.

3.2.4 Distance metric - euclidean or manhattan

Manhattan and Euclidean distance are commonly used distance metrics in machine learning and data analysis. While both metrics have their strengths and weaknesses, there are cases where one metric may be more appropriate than the other.

In general, Manhattan distance (also known as city block distance or taxicab distance) is better than Euclidean distance for high-dimensional data. This is because, Manhattan distance measures the distance between two points by adding up the absolute differences between their coordinates. This makes it less sensitive to changes in any single dimension and more effective at capturing the distance between points in high-dimensional space.

On the other hand, Euclidean distance is more prone to being affected by changes in individual dimensions, which makes it less effective in accurately measuring the distance between points in high-dimensional space [21].

3.3 K-means

The k-means algorithm is an iterative, distance-based algorithm. It is relatively fast, simple and computationally effective. In addition, low memory usage makes it suitable for cluster detection in large data sets. It is a significant advantage over hierarchical clustering methods based on dissimilarity matrix. Although they allow exploration of any number of clusters, they might become computationally demanding with growing data. Lastly, k-means might be successfully used as a starting step in other algorithms. A concrete example of that is the Expectation-Maximization algorithm's initialisation part.

From a mathematical point of view, k-means is an approximation of the normal mixture model. Parameters are estimated using the maximum likelihood method. The main idea behind the k-means is that observations are gathered around artificially introduced centres called centroids. Centroid can be perceived as a generalisation of the mean, a geometric centre of a convex object. In general, the distance between centres and observations should be minimal. Data points closest to the particular centre are part of its cluster.

The initial number of centroids is equivalent to the number of clusters k in the data. k is required to start the algorithm. If we have a strong assumption regarding the number of clusters, we can use it. Otherwise, there are few ways to determine the number of clusters experimentally. The algorithm stops when it reaches stability, which depends on the convergence criteria. An example of that will be creating groups with the highest similarity of points within a given cluster and the lowest between different clusters. In the commonly used

Hartigan-Wang implementation, the stop criterion can be satisfied in two ways. One is minimising the total sum of variance within clusters (WCSS) as given in the following equation [22].

$$WCSS = \sum_{i=1}^{k} \sum_{j=1}^{n_i} \|x_{ij} - c_i\|^2 \tag{3.14}$$

where:

$WCSS$ - total sum of variance within clusters

$\mathbf{c_i}$ - centroid

x_{ij} - observation ij

There are few k-means algorithms: Lloyd, Forgy, MacQueen and the one already mentioned Hartigan-Wong. The last one is the default k-means algorithm in the R software, that we have used in our comparison.

3.3.1 Hartigan-Wong

The Hartigan-Wong algorithm aims to find the best way to group data into clusters based on minimizing the sum of squared errors (SSE) within each cluster. To achieve this aim, it may reassign data points to a different cluster, even if they are currently assigned to the closest centroid if it would result in a lower total SSE for the entire set of clusters. In other words, the algorithm prioritizes finding the best overall fit for the data rather than strictly sticking to the closest centroid for each data point[23].

$$SSE2 = \frac{N_i \sum_j \|x_{ij} - c_i\|^2}{N_i - 1} < SSE1 = \frac{N_1 \sum_j \|x_{1j} - c_1\|^2}{N_1 - 1} \tag{3.15}$$

where:

$SSE1$ - the sum of the squared Euclidean distances of each point to the first centroid

$SSE1$ - the sum of the squared Euclidean distances of each point to the second centroid

$\mathbf{c_i}$ - centroid

x_{ij} - observation ij

3.3.2 Initialization

We compared different two different types of initializations for kmeans:

- Hartigan wong, as implemented in R
- k-means++ from package ClusterR

From the results we can see that

3.4 Fuzzy clustering

Fuzzy k-means clustering, often called fuzzy c-means (**FCM**), can be considered a soft version of k-means. Each observation has some set of degree of belonging to a cluster. This is different approach than in k-means, where membership is binary and given observation belong (1) or not (0) to particular group.

FCM centroids are the means of all observations weighted by their degree of membership to the cluster. They can be calculated using the formula:

$$c_k = \frac{\sum_i w_k \left(x_i\right)^m \times x_i}{\sum_i w_k \left(x_i\right)^m} \tag{3.16}$$

where:

c_k - centroid of a cluster k,

w_k - degree of membership to the cluster,

m - fuzzifier coeficient

The FCM tries to minimize objective function:

$$\arg\min_C \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^m \left\|\mathbf{x}_i - \mathbf{c}_j\right\|^2, \tag{3.17}$$

where: w_{ij} indicates the degree of the belonging to the cluster. It allows to calculate the distance between the observation and the centroid k. Formula is given by :

$$w_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\left\|\mathbf{x}_i - \mathbf{c}_j\right\|}{\left\|\mathbf{x}_i - \mathbf{c}_k\right\|}\right)^{\frac{2}{m-1}}} \tag{3.18}$$

where:

m - fuzzifier coeficient

The fuzzifier m playes important role in fuzziness of clusters. It can take values from [1, ∞]. A large value of m results in lower degree of belonging w_{ij}, resulting in fuzzier groups. For the data without any apriori knowledge, parameter m is commonly set to 2. When the m = 1, the cluster membership should converge to 0 or 1, as in the k-means algorithms.

Due to vast similarities to k-means, FMC suffers from the same problem. It does not guarantee convergence to the global optimum. It might get stuck in local minima. In addition, the results heavily depend on the initial choice of weights.

Different distance metrics, namely the Manhattan and Euclidean distances, were also examined for fuzzy c-means.

3.5 k-medoids

The most significant advantage of the k-means algorithm is computational efficiency. Calculating averages and assigning them to the closest mean is algebraically fast. It is relatively

easy to parallelize. Those merits make this algorithm applicable to large databases.

Unfortunately, it is based on the square of the Euclidean distance. However we are not always interested in this kind measure of similarity. Such a measure is also susceptible to large distances, and a single outlier can significantly affect the sum of squared distances.

PAM works similarly to k-means, except that the centres of the clusters are the cases in the data set (called centroids or clusters). Therefore, the set of possible cluster centres in the PAM method is much smaller than in the k-means method. Usually, the results of the PAM method are more stable. Moreover, the k-medoid algorithm will allow us to use other distance measures at the price of higher computational complexity. In other words k-medoids method is an extension of the k-means method.

The k-medoids method finds such objects representing clusters (k-medoids) among the observations, to minimize the sum of the distances of all non-medoid elements from their closest medoids.

Another difference between the k-means algorithm and the k-medoid method is how the distance between observations is defined. In the case of the k-means Euclidean distance is used, while the PAM uses mainly Manhattan distance. In addition, k-medoids are supposed to deal with the problem of outliers as well as noise in the data. It is considered more robust than k-means. However, its computational usage can be considered high even by today's standards. It performs slowly in large data sets.

We can distinguish two phases of PAM namely construction phase and swap phase.

Construction phase:

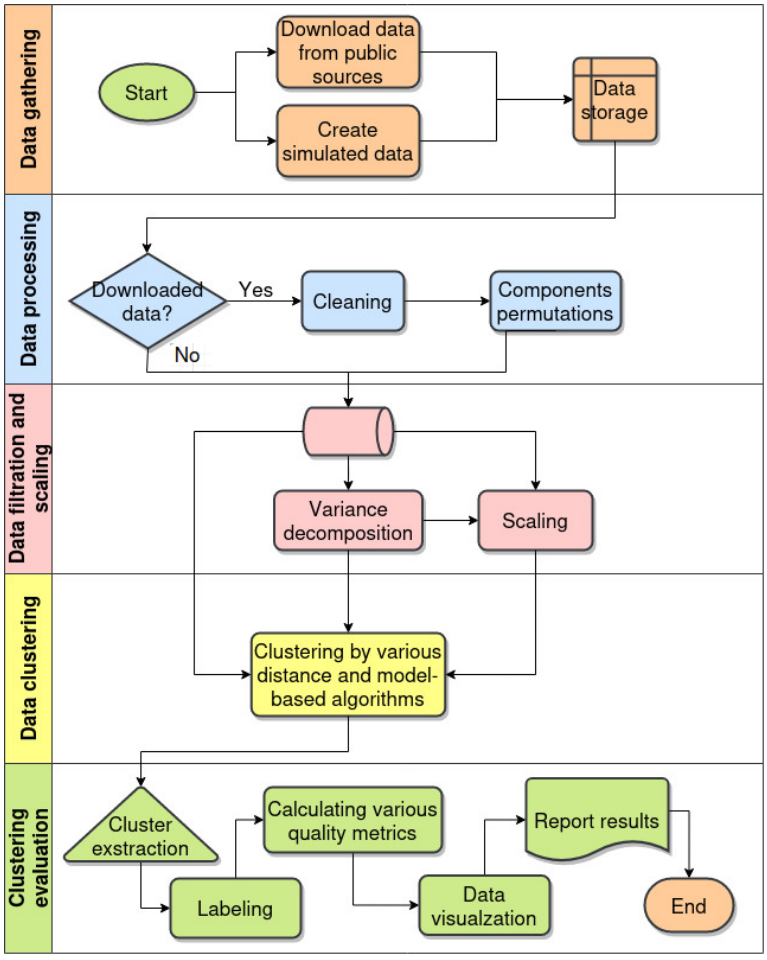
1. Split the dataset into k clusters with k medoids assigned
2. Calculate the distance matrix between the medoids and the other observations
3. Assign each observation (non-medoid) to the closest cluster

Swap phase:

1. Using iteration, replace one of the medoids with one of the non-medoids and check that the distances of all non-medoids from their nearest medoids are smaller.
2. If at least one medoid swap has occurred, repeat step 3. Otherwise, end the algorithm.

4 Study pipeline

This chapter presents the entire pipeline, starting with the method for choosing and initially preparing datasets. The filtration and scaling methods that were used are then presented. Following that, cluster evaluation methods are introduced, which consist of visualization techniques for data and ways to present the algorithms' performance or efficiency.



The chart will be explained in the next sections.

4.1 Data gathering

All datasets can be divided into two parts. The first one comprises artificially created data that can present a more controlled challenge to algorithm performance. The second subset includes real datasets that were selected with the criterion of multidimensionality in mind.

4.1.1 Simulated data

Data simulation is a vast research field with plenty of practical applications. We can use it in data prediction and during testing new programs or algorithms, to name a few. Simulation of data is a way to generate random numbers from the stochastic process. A stochastic process is a time-ordered sequence of random variables or something described by probability distributions. For example, we can model height population data using a mixed normal-uniform distribution model. [24].

$h_i \sim \text{Normal}(\mu, \sigma)$

$\mu \sim \text{Normal}(160, 15)$,

$\sigma \sim \text{Uniform}(0, 10)$

where h_i is i-th observation,

Both μ and σ parameters are generated from separate distributions. First parameter, μ is generated from the normal distribution with parameters $\mu = 160$, and $\sigma = 15$. Second parameter, σ is generated from the uniform distribution with minimal value of 0 and maximum value of 10. Here, our exemplary model doesn't describe the height of any particular population. It is a very rough example that is within reasonable limits. Extreme values of this model will barely go lower than 95 cm and exceed 240 cm. The reason is that the shortest person recorded was Chandra Bahadur Dangi, who measured 54.6 cm [25]. On the opposite, the record for the tallest man recorded belongs to Robert Wadlow, with 272 cm of height [26].

Data simulation can become pretty handy. The ground truth is fully known, contrary to the actual data, when we are sometimes only partially aware of the groups in the data. We know the exact model parameters and can compare them with the estimated values.

Some models might get exceptionally complicated by having many different parameters. It may become untraceable to predict their effect on the data. However, since we have complete control over model parameters, we can tune them accordingly and observe the result. In this manner, we can perceive data simulation as a reflection of some natural system and a controlled experiment [27].

Using data simulation techniques, we are no longer limited by actual measurements. We can generate an almost infinite number of data with the structure and parameters we want. It allows us to test different kinds of data. However, it might not reflect the real data accurately, giving us false estimations of model efficiency. It is based on a few limitations. The very special that should be taken into mind is Random Number Generation.

Random Number Generator (RNG)

As of the time of writing the thesis, there is no genuinely random number generator. Instead, we have PRNG. The letter "P" stands for the "pseudo". Although some efforts exist to create such a system using quantum computers, we will focus on PRNG. It is a program, or part of it, that generates numbers based on some initial information. This base information is called a seed. It can use many different sources of initial data. Depending on the software, it can be

actual time, active processor time or key inputs transformed into numerical values. It is why pseudo-random number generators don't produce random numbers. When a seed initializes PRNG, it can take an infinite number of different values. However, most of the time, it is restricted by software design or by computer memory.

4.1.1.1 Simulated multivariate normal data

Mixtures were generated according to multivariate normal distribution properties. Each mixture component consisted of random vector μ, matrix Σ, mixing proportion α and a fixed number of dimensions d, across all mixture components. Each random vector μ was generated from uniform distribution such that $\mu : U(0, 10)$ from the interval [0,10]. Σ was created assuming that the correlation between variables (elements of data vectors) equals 0, so they are independent. Variances were then generated from a uniform distribution from the interval [0.1, 1] to avoid 0 variances. Mixing proportions were obtained from a uniform distribution from the interval [0.1, 1] to prevent one component from dominating the mixture. The vector of mixing ratios was standardized, to sum up to 1.

Counting different filtration and scaling, over 15 000 files were created. They contained from 2 to 10 clusters and between 5 and 1600 dimensions. Step between dimension was taken arbitrally. The higher number of files than in multinomial data comes from the fact that two different PRGN packages were used, namely MASS and MultiRNG.

4.1.1.2 Simulated multinomial data

Mixtures were generated according to multinomial distribution properties. Each mixture component consisted of: random vector p of probability successes, number of observations n, mixing proportion α and d fixed across all groups. p was generated from a uniform distribution from the interval $[0, 1]$. Then all values were standardized so that all elements sum up to 1. n was generated from the interval $[3, n]$. Mixing proportions of all components were calculated using uniform distribution from the interval $[0.1, 1]$ and standardized, to sum up to 1.

Over 8,000 files were created with different types of filtration. Similarly to multivariate data, these files contained 2 to 10 clusters between 5 and 1600 dimensions with arbitrarily selected steps between dimensions.

4.1.2 Real data

The most important part of our comparative analysis of unsupervised clustering algorithms is computational experiments concerning some publicly available data sets. Extensive real-world datasets collections can be used for comparative experiments of unsupervised clustering algorithms, e.g., Kaggle, UCI, NCBI or NCI. When choosing data sets for clustering, the following criteria were used:

- high dimensionality

- numerical features (real or integer).
- variety of data types
- ground-truth availability

According to mentioned criteria, the data should have many dimensions. The mixture should not be univariate or bivariate, and features cannot be categorical.

Various data types assume that data comes from different expertise fields or is measured differently. For example, somatic mutation count and gene expression came from the same TCGA project but were measured differently.

Real data consist of various datasets from a few different fields, although majority have some genetic and medical background. As such following data was chosen:

Data set	Source
Somatic Mutations Counts	TCGA
Gene Expressions	TCGA/cBioportal
Arthythmia	UCI
Codons Frequency	UCI
Sport Activities	UCI
The Free Music Archive	GitHub
NASA Kepplers	Kaggle

Table 4.1: Real data-set with their sources

The TCGA project started in 2005, with the aim of identification of complex genomic interactions in majors cancers. Five year later, the TCGA was made available to public. Then, in December 2013, TCGA concluded sample collection with over 20 000 biospecimens across 32 types of cancer. The data become publicitly available for academic purposes worldwide.

The UCI repository is a vast collection of databases and data generators. It is an open access repository of data, which people worldwide donate. The machine learning community generally uses it for empirically studying various ML algorithms. UCI repository was firstly created in 1987 as an ftp archive by David Aha and students of UC Irvine. As UCI site reports, since that time, it was cited over 1000 times, placing it among the top 100 most quoted "papers" in all computer science.

cBioPortal is a web-based platform that allows researchers to explore and analyze large-scale cancer genomics datasets. It provides access to a wide range of cancer genomics data, including DNA copy number, gene expression, DNA methylation, protein expression, and phosphoprotein expression data. The platform is free and open to the public. It has been widely used in cancer research to identify potential biomarkers, therapeutic targets, and other cancer-associated genomic alterations.

Kaggle is a platform for data science competitions, where companies and researchers can post their data and problem statements and invite the data science community to develop solutions. It was founded in 2010 and acquired by Google in 2017. Participants can compete

individually or as teams to build the best models and algorithms to solve the problem. Kaggle also offers courses and tutorials on data science and machine learning, but it is also a vast data source that the community shares.

4.2 Data processing

4.2.1 Preparation

After the data was downloaded and cleaned, it was parsed into a fixed format. A matrix of size $n \times d$ was created for each dataset, with observations n placed in rows and variables d in columns to ensure consistency across all analyzed datasets.

Except for the NASA Kepler dataset, the remaining datasets typically contained ten or more classes. The permutation method was employed to generate 10 datasets for 2, 3, 4, 5, and 6 component mixtures to create multiple datasets for analysis. In this way, each set of six component mixtures always contained at least one or more different classes. Consequently, 50 datasets with classes in various configurations were created from a single real dataset.

This methodology can be applied to all datasets for analysis.

4.3 Data filtration and scaling

In the filtration part, either the data was left unchanged, or the variance decomposition method was employed to reduce the "noise" in the data.

4.3.1 No filtration

In this case, the datasets containing all the original variables were left unchanged. They were used in the scaling step and in the clustering stage of the analysis.

4.3.2 Variance decomposition.

The respective variance for each variable was calculated, resulting in a vector of n variances with varying values of n, dependent on the dataset used. The resulting vector was assumed to represent a one-dimensional mixture comprising essential variables and additional noise. A mixture decomposition technique was employed using the mclust package to differentiate between these elements. The mixtures were aligned to contain two to twenty-five components, and the final number of groups was determined using the Bayesian Information Criterion.

4.3.2.1 Bayesian Information Criterion (BIC)

Bayesian Information Criterion is a usefull tool in model selection. Given by following equation:

$$\mathrm{BIC} = k \ln(n) - 2 \ln(\widehat{L}). \tag{4.1}$$

where:

$\hat{L}$- the maximized value of the likelihood function of the model M i.e.$\hat{L} = p(x \mid \widehat{\theta}, M)$, where $\widehat{\theta}$ are the parameter values that maximize the likelihood function;

x - the observed data;

n - the number of data points in x, the number of observations, or equivalently, the sample size;

k - the number of parameters estimated by the model.

It penalizes model for using more parameters and is scalable because it takes into equation number of observations. I have used it during variance mixture decomposition, to group variances into k groups.

4.3.3 Scaling

Scaling was performed separately on the complete and reduced datasets to preserve the variance. The variables were scaled by subtracting the mean value from the vector of observations and dividing the resulting values by the standard deviation. After scaling, two additional files were created for each dataset.

4.4 Data clustering

No	Abbreviation	Full name
1	CMN	C-means
2	MANCMN	C-means with Manhattan distance
3	GMM	Gaussian Mixture EM
4	GMMK	Gaussian Mixture Models with k-means
5	HC	Hierarchical Clustering
6	MANHC	Hierarchical Clustering with Manhattan distance
7	KMN	K-means
8	KMN++	K-means++
9	KMD	K-medoids
10	MMM	Multinomial Mixture EM
11	MMMK	Multinomial Mixture EM with k-means

Table 4.2: Algorithms used in the clustering

All the clustering algorithms mentioned above were employed to cluster simulated data on multivariate normal mixtures, simulated multinomial mixtures, and real data. Each clustering algorithm generated a separate file, which was saved to disk for future processing.

4.5 Clusters evaluation

In this study, clusters were evaluated based on their correct assignment. For this purpose, the Hungarian algorithm was employed to assign results to the labels. Then, a few metrics

were implemented to compare algorithms' performance from various perspectives. Finally, the results were visualized using different plot types.

4.5.1 Cluster assigment - Hungarian algorithm

The increased number of clusters poses a few challenges. One of them is assigning clustering results to appropriate labels. Because, in principle, unsupervised clustering is done without prior group knowledge, results are unlabeled.

One potential solution to this problem is The Hungarian matching algorithm, also called the Kuhn-Munkres algorithm.

It is a combinatorial optimisation algorithm used to solve the assignment problem in mathematics. The goal is to assign a set of workers to a group of tasks while minimising the total cost or maximising the total profit. The algorithm uses a matrix to represent the cost of assigning each worker to each job and iteratively finds the optimal assignment utilising a series of augmenting paths. The result is a one-to-one mapping between workers and tasks that minimises the total cost or maximises the total profit. In the case of this study, clustered observations were assigned to corresponding labels based on their maximisation.

4.5.2 Clusters validation

There is a plethora of various metrics that allow us to compare the similarity or dissimilarities between two clustering methods. We applied a few different to describe the algorithms' performance by comparing created clusters with the ground truth.

4.5.2.1 Rand Index and Adjusted Rand Index

Rand index is a popular metric to measure the results of clustering.

$$Rand\ Index = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.2}$$

where:**TP** - number of true positives**; FP**- the number of false positives**; FN**- the number of false negatives**; TN**- the number of true negatives.

The RI index takes ranges between 0 and 1, including themselves. In the perfect aligment, the term $n_{21} + n_{12} = 0$ and $Rand\ Index = 1$.

This scenario is sufficient only for two-class problems. If we have to estimate clustering quality or similarity of more than two classes, the table remains squared but its dimensionalty equals to k. Matrix below is similar to the matrix from the chapter 2. However, it shows groups of clusters. Bold characters indicates correctly assigned observations, that are located on the diagonal line. We will use this assumption for the rest of the thesis. Note that directly after the clustering, groups often will be scattered on the matrix. However, we ensured that this statement is true by assigning labels to clusters, using the Hungarian algorithm. Then, sorting is easily done internally in the R, when creating confusion matrix.

X	Y_1	Y_2	Y_m	$\cdots$	Y_M	sums
X_1	$\mathbf{n_{11}}$	n_{12}	n_{1m}	$\cdots$	n_{1M}	a_1
X_2	n_{21}	$\mathbf{n_{22}}$	n_{2m}	$\cdots$	n_{2M}	a_2
X_d	n_{d1}	n_{d2}	$\mathbf{n_{dm}}$	$\cdots$	n_{dM}	a_d
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_D	n_{D1}	n_{D2}	n_{Dm}	$\cdots$	$\mathbf{n_{DM}}$	a_D
sums	b_1	b_2	b_m	$\cdots$	b_M	n

(4.3)

In the above matrix:

X - ground truth labels,

Y - clustering labels,

a_D - a sum of all observations that belongs to X_D,

b_M - a sum of all observations that belongs to Y_M

$n = \sum_i \sum_j n_{ij}$ - all observations

In this case, Adjusted Rand Index (ARI) was proposed[28]:

$$ARI = \frac{\sum_{dm} \binom{n_{dm}}{2} - \left[\sum_d \binom{a_d}{2} \sum_m \binom{b_m}{2}\right] / \binom{\mathrm{n}}{2}}{\frac{1}{2}\left[\sum_d \binom{a_d}{2} + \sum_m \binom{b_m}{2}\right] - \left[\sum_d \binom{a_d}{2} \sum_m \binom{b_m}{2}\right] / \binom{\mathrm{n}}{2}} \tag{4.4}$$

4.5.2.2 Jaccard and Weigted Jaccard Index

In simplified form for two-class case, Jaccard index takes a following formula:

$$Jaccard = \frac{TP}{TP + FP + FN} \tag{4.5}$$

The basic difference from the Rand Index, is that it does not contain TN counts [29].

The **Weighted Jaccard** index, based on the two class equation. First, we substituded denominator as in following

We are substracting term n_{11}, since both a_1 and b_1 have and we need only one. Then, we can generalize the equation for all of the groups

$$Weighted\ Jaccard = WJACC = \frac{\sum_{d,m} \frac{n_{dd}}{a_d + b_m - n_{dd}}}{K} \tag{4.6}$$

where: a_d - is the sum of FP values in any group, b_m- is the sum of FN values in any group, n_{dm} - is the value of TP in any group, k is the number of components

4.5.2.3 Accuracy and Balanced Accuracy Index

Accuracy is a common evaluation metric used in machine learning to measure a classification model's performance. It is the ratio of correctly classified samples to the total number of samples in the dataset. Accuracy can be expressed as:

$$Accuracy = ACC = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.7}$$

In the equasion above, Accuracy index is the same as Rand index, thus sometimes it is called Rand accuracy. In our comparison we used its variation, called Balanced Accuracy.

However, Accuracy may not always be the most suitable metric to use, particularly in cases where the dataset is imbalanced. For example, if the dataset contains 90% of one class and only 10% of another, a model that always predicts the majority class will have an accuracy of 90% but may not be useful for practical purposes. Balanced accuracy is a modified version of accuracy that considers the dataset's imbalance by calculating each class's average accuracy. It is defined as the average of each class's sensitivity (true positive rate) and specificity (true negative rate). Balanced accuracy is as follows:

$$BalancedAccuracy = BACC = \frac{Sensitivity + Specificity}{2K}$$

where:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Sensitivity = \frac{TN}{TN + FP}$$

Balanced accuracy provides a more meaningful evaluation metric when dealing with imbalanced datasets, as it gives equal weight to both positive and negative classes rather than favouring the majority class.

This metric was used to create the Median Accuracy assignment plot. It shows how, on median given algorithm assigned observations to the correct group. The higher the bar, the better. For the bigger picture, the mean with standard deviation was superimposed on the plot. From the perfect assignment, we expect a mean of size 1 and 0 variances at the same point as the median value.

4.5.2.4 Simple Matching Coefficient

The SMC is easy statistic, which calculates efficiency of the clustering. To obtain this metric, we have to divide sum of all true postivie values by all observations. As one may already knew, this metric might not be suitable choice, when we have significalntly different number of observations.

$$SMC = \frac{TP}{TP + FP + FN + TN} = \frac{\sum_d n_{dd}}{\sum_{dm} n_{dm}}$$

In the SMC metric we calculate numerator in a different way, than it was in the case accuracy index.

When we calculate the value of TN it contains

4.5.2.5 Weighted Simple Matching Coefficient

The presented WSMC is balanced version of SMC. It accounts for different number observation in the groups, thus might be more accurate, where the classes are not balanced.

$$WSMC = \frac{\sum_{d,m} \frac{n_{dm}}{a_d}}{K}$$

The weighting factor, $\frac{n_{DM}}{a_D}$, represents the proportion of correct predictions for each row, which is a way to incorporate the varying number of predictions made for each row. The metric calculates the weighted average of these row-wise proportions, with each row weighted by the total number of predictions made for that row.

4.5.2.6 Beta-binomial conjugate distribution

Beta-binomial conjugation is a statistical concept that involves using a beta distribution as a prior distribution for a binomial distribution. In Bayesian inference, a prior distribution is a probability distribution that represents the degree of belief or uncertainty about the model's parameters before the data is observed. A prior conjugate is a prior distribution that, when combined with the likelihood function, results in a posterior distribution of the same family as the prior distribution. The binomial distribution and the beta distribution are conjugated. This means that if the prior distribution for the parameter of a binomial distribution is a beta distribution, the resulting posterior distribution will also be a beta distribution.

The beta-binomial model is often used to model count data. The outcome is the number of successes in a fixed number of trials, and the probability of success in each trial is unknown. The beta distribution is used as the prior distribution for the probability of success, and the binomial distribution is used as the likelihood function for the observed data. The resulting posterior distribution is also a beta distribution, which provides a way to update the prior belief about the probability of success based on the observed data. It provides a valuable framework for estimating the probability of success while allowing for data uncertainty and variability.

Beta distribution consist of two parameters: α and β. Thanks to conjugation they can be easily updated:

$$p(\theta|\mathrm{k}, \alpha, \beta) \sim \mathrm{Beta}(\alpha, \beta) \tag{4.8}$$

$$\alpha' = \alpha + \mathrm{k} \tag{4.9}$$

$$\beta' = \beta + \mathrm{n} - \mathrm{k} \tag{4.10}$$

where: n - a number of trials, k - number of successes

Using parameters α and β, we can calculate its mean and standard deviation.

$$\mu = \frac{\alpha}{\alpha' + \beta'} \tag{4.11}$$

$$\sigma = \sqrt{\frac{\alpha'\beta'}{(\alpha' + \beta')^2(\alpha' + \beta' + 1)}} \tag{4.12}$$

The beta distribution is a conjugate prior to the binomial distribution. When we use a beta distribution as a prior distribution for the binomial likelihood function, the resulting posterior distribution will also be a beta distribution. This makes it easier to update the prior distribution with new data, as we do not have to recalculate the model efficiency repeatedly. Additionally, the beta-binomial distribution has a simple and robust structure, making it a popular choice for Bayesian analysis.

If a model is given a higher probability of better performance, the beta-binomial distribution will adjust its parameters accordingly once evidence is seen. However, if an initial prior is provided without evidence, beta-binomial distribution will assign a higher probability to specific outcomes. This will be immediately noticeable to the reader.

4.5.3 Visualization

In the tesis, all of the graphic charts were created using ggplot2 package as the base[30].

4.5.3.1 Dimensionality reduction

Dimensionality reduction methods allow the representation of observations in space with fewer dimensions than in the original data. Dimensionality reduction is often an intermediate step in classification, cluster analysis, or regression. In certain situations, it improves the effectiveness of these methods, increases stability and sometimes allows the inclusion of many variables in the analysis. The data is reduced to a two-dimensional space, making it easy to present them on a graph. Methods from this group are also called feature extraction methods. It is also a popular method for visualising multidimensional variables, which was used in this study.

PCA

Its purpose is to extract the essential data from the matrix. It is then represented as a set of new orthogonal variables called principal components to display the pattern of similarity of the observations and the variables as points in two or three-dimensional space.

The principal component analysis defines new variables by combining their smallest possible subset. They are linear, weighted combinations of the original data set. The aim of creating new variables is that they will explain the total variability in the data set as much as possible. The new variables will form an orthogonal basis in the feature space. We should select variables so that the first variable reflects as much variability in the data as possible. After projecting the observations onto this vector, we want the variance of the projections to be the highest. After determining the first variable, the second is orthogonal and explains as much of the remaining variability as possible. Another variable should be orthogonal to the first two, and the procedure continues until no variable remains.

SVD and tSVD

Singular Value Decomposition (SVD) is a factorization of a real or complex matrix into three matrices: U, Σ, and $V*$. The matrix U and $V*$ are unitary matrices, and Σ is a diagonal matrix containing the singular values of the original matrix. Truncated Singular Value Decomposition (tSVD) is a variation of SVD that only retains a subset of the singular values and corresponding singular vectors of a matrix. The idea behind tSNE is to reduce the computational cost of SVD by only considering the largest singular values that capture essential information of the original matrix. tSNE is commonly used in dimension reduction, denoising, and compression tasks.

tSVD was used to visualize datasets.

tSNE

tSNE stands for t-Distributed Stochastic Neighbor Embedding. It is a non-linear, unsupervised technique of dimensionality reduction. Its primary purpose is to explore and visualise multidimensional data. The tSNE algorithm computes a measure of similarity between pairs in large- and small-dimensional spaces. For the next step, it tries to optimise both measures of similarity. In simple terms, tSNE tries to simplify how multidimensional data is distributed in euclidean space. tSNE is not designed to cluster data but primarily to explore and visualise data. However, we can assess how many groups may be hidden in the data by visualising data in two or three-dimensional space. [31]

Several R packages can create a tSNE map: *tsne*, *Rtsne* or *cuda.ml*.

In the thesis, package cuda.ml was used, as it utilizes graphic card capabilities to produce results, aided by Rtsne. Using hundreds of cores is a huge advantage when dealing with large datasets. The major drawback is the VRAM storage. The VRAM capacity in the average graphics card oscillates between 4GB-8GB. The 6GB I used was insufficient for some datasets, so we had to resort to RAM/CPU calculations.

Random projection

Random projection is another computational technique used for dimensionality reduction. It is a method of representing high-dimensional data in a lower-dimensional space by using random linear projections.

In random projection, a random matrix is generated and multiplied with the original high-dimensional data points to produce the lower-dimensional projections. The idea behind random projection is that, for many applications, the high-dimensional data lies on or near a lower-dimensional subspace. The random projection provides a way to estimate this lower-dimensional subspace. One of the critical advantages of random projection is that it is fast and computationally efficient. Unlike dimensionality reduction techniques like PCA or SVD, random projection does not require eigendecomposition or singular value decomposition, which can be time-consuming.

4.5.3.2 Metrics report

Colours that are occuring across report are consistent and assigned to each algorithm.

The ARI Index

The ARI Index was shown by using boxplots.

Beta binomial distribution

Beta-binomial distrubtion was drawn using beta probability density function

A median of Balanced Accuracy

A median of Balanced Accuracy was drawn using bars on a polar coordinate plane. The height of a bar was related to the value of the median BACCU. In addition, mean and standard deviation were added.

WSMC, SMC and WJACC

The three metrics were drawn similarly altogether as violin plots. Those plots also consist of means and points of scores. The points were randomly scattered across the x-axis, which completes the density.

Correlation plot

The correlation coefficient was calculated between every two metrics to measure the linear relationship's strength and direction. A strong correlation indicates strong agreement regarding clustering quality.

4.6 General Computation optimization

Optimizing performance is an essential aspect of many applications, including solving arithmetic equations quickly. For instance, matrix or vector multiplication may involve several steps to obtain the desired result. To streamline this process, high-level programming languages like Python and R and mid-level languages like C++ come with pre-built libraries that offer these functionalities. By utilizing these libraries, developers can save time and effort manually coding these computations.

BLAS

The shortcut refers to Basic Linear Algebra Subprograms. In general, BLAS provides classic interfaces for linear algebra calculations. Inside the BLAS library, we can distinguish three different BLAS levels. BLAS1 (level 1) allows for vector-vector operations. BLAS2 is responsible for matrix-vector operations. Finally, BLAS3 makes matrix-matrix operations possible. In summary, BLAS is the computational kernel for linear algebra and other scientific applications. If we can make BLAS optimise its libraries, the whole application written with it will benefit.

LAPACK

LAPACK stands for Linear Algebra PACKage. It is used along with BLAS. The actual language that LAPACK is written in is Fortran 90. It consists of routines for more comhttps://spankbang.com/2 algebra, like least-squares, eigenvalues or singular value decomposition. LAPACK also handles routines for solving QR factorisation, lower–upper (LU) decomposition, or Cholesky decomposition, to name a few. It handles dense and some sparse matrices, precisely banded ones. However, it lacks routines for sparse matrices in general. Regarding types of numbers, similar functionality is provided for real and complex matrices, and both double and single precision is supported.

The relation between BLAS and LAPACK is as follows: LAPACK heavily depends on BLAS as it is built upon it.

https://csantill.github.io/RPerformanceWBLAS/ provides some results that compare mentioned alternatives to BLAS. In this comparison, each library has its fortes. However, Intel MKL has the highest number of the best results in all comparisons. However, it does not mean it is the best choice. Mostly because the test comparison is precise. It is more difficult to say how those libraries will perform in more complex applications.

In our computations, we have used OpenBLAS libraries. The decision was based upon the fact that OpenBLAS was open source from the beginning, and its performance was second best.

OpenMP

OpenMP is a multi-threading API for C, C++, and Fortran programming languages. It provides a parallel programming model for shared memory systems, making it easier for developers to write parallel applications. OpenMP simplifies the process of parallelizing loops and sections of code, allowing workloads to be distributed among multiple threads for improved performance and faster execution times.

Installing all of the metioned libraries required recompiling R kernel and its libraries. However it resulted in increased performance, without overuse of memory.

R specific optimisation

There is a common idea that R is slow, which is somewhat valid. R is a high-level programming language. It was designed with statistical analysis in mind. Many functions are not created with speed and efficiency but with stability and reliability.

That said, here is the mean() function code found in R. To see what the mean contains, we should execute the mean.default() instead of mean(), as the former will only give us information about methods.

```
mean.default <- function (x, trim = 0, na.rm = FALSE, ...) {
    if (!is.numeric(x) && !is.complex(x) && !is.logical(x)) {
        warning("argument is not numeric or logical: returning NA")
        return(NA_real_)}
    if (isTRUE(na.rm)) x <- x[!is.na(x)]
    if (!is.numeric(trim) || length(trim) != 1L)
        stop("'trim' must be numeric of length one")
    n <- length(x)
    if (trim > 0 && n) {
        if (is.complex(x))
            stop("trimmed means are not defined for complex data")
        if (anyNA(x)) return(NA_real_)
        if (trim >= 0.5)
            return(stats::median(x, na.rm = FALSE))
        lo <- floor(n * trim) + 1
        hi <- n + 1 - lo
        x <- sort.int(x, partial = unique(c(lo, hi)))[lo:hi]}
    .Internal(mean(x))}
```

First, the function includes several sanity checks to ensure correct input, such as checking for NA values. It also handles various data types, including time, date, and date-time classes. While these checks require additional time to execute, they ensure the accuracy and reliability of the results. Using a clean mean function will be faster in the long run, provided the data is appropriately formatted.

While R allows for efficient function writing, it is still a high-level language and optimised C code may offer better performance. Generally, lower-level languages may be preferable when dealing with specific data types or frequently used functions. Nonetheless, there is always room for improvement.

5 Results

The following chapter presents the results of clustering simulated data and real (actual measurements) data. Each dataset clustering is preceded by its brief characteristic and description.

5.1 Simulated data analysis

The first part consists of simulated data analysis, starting with Multivariate Normal Mixtures. The second part presents Multinomial Mixtures.

5.1.1 Multivariate Normal Mixtures

Results

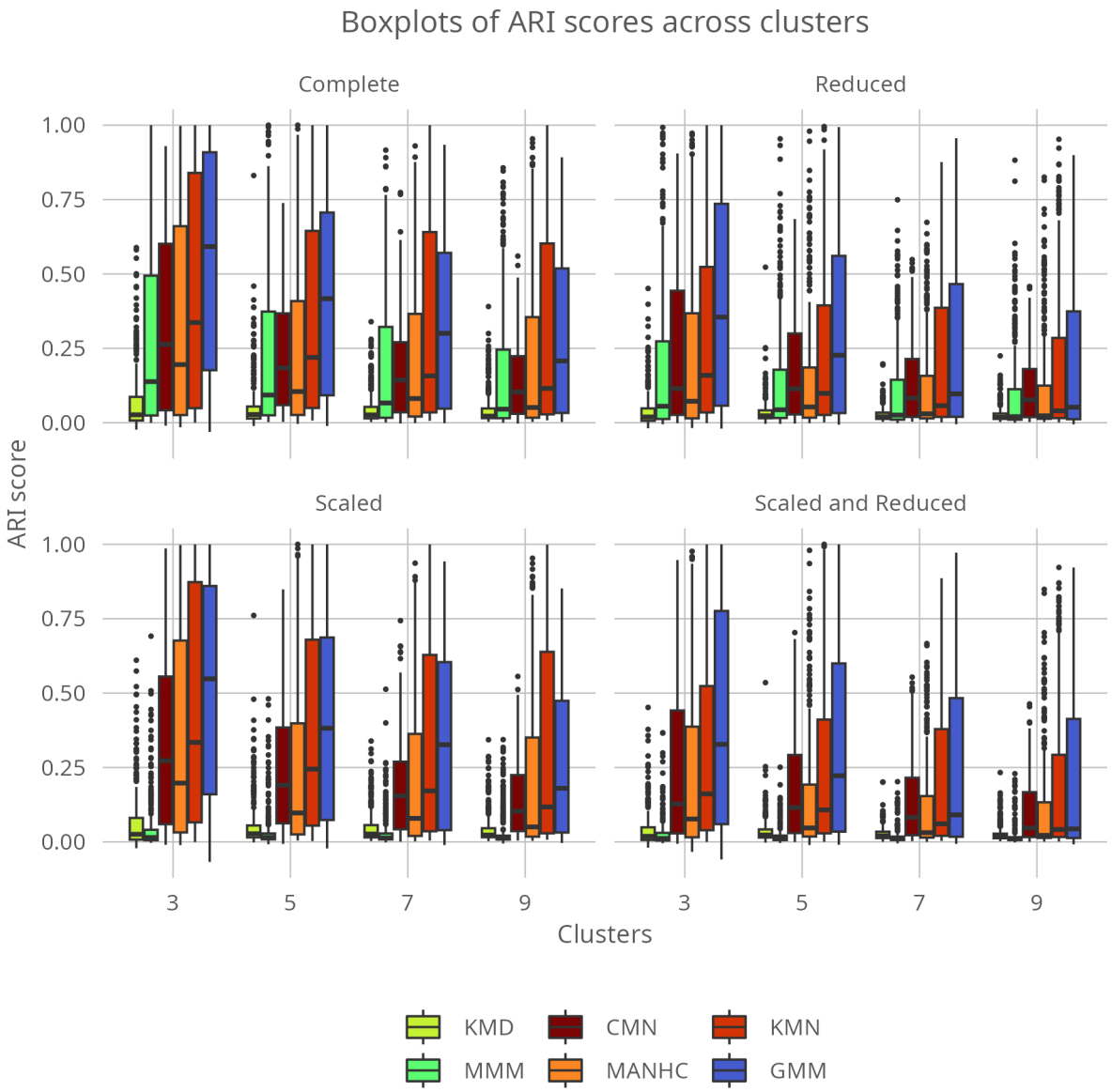


Figure 5.1: The ARI index in Simulated Multivariate Normal Mixtures

The ARI index in Figure 5.1 indicates that GMM was the best-performing algorithm in all datatypes, with consistently high median scores. Although GMM did not necessarily wholly overwhelm the other algorithms, its spectrum of values often reached higher values than the other algorithms. KMN had a similarly broad spectrum of values, but its median score was consistently lower than GMM. It's worth noting that as the number of components increased, the scores of all algorithms tended to decrease, likely due to the limitations of the data simulation scripts favouring overlapping data. In contrast, MMMK performed similarly to HC but with slightly lower median scores in the complete and reduced data. However, when

the data was scaled and reduced, its performance was worse than KMD, which generally had low scores in each data type.

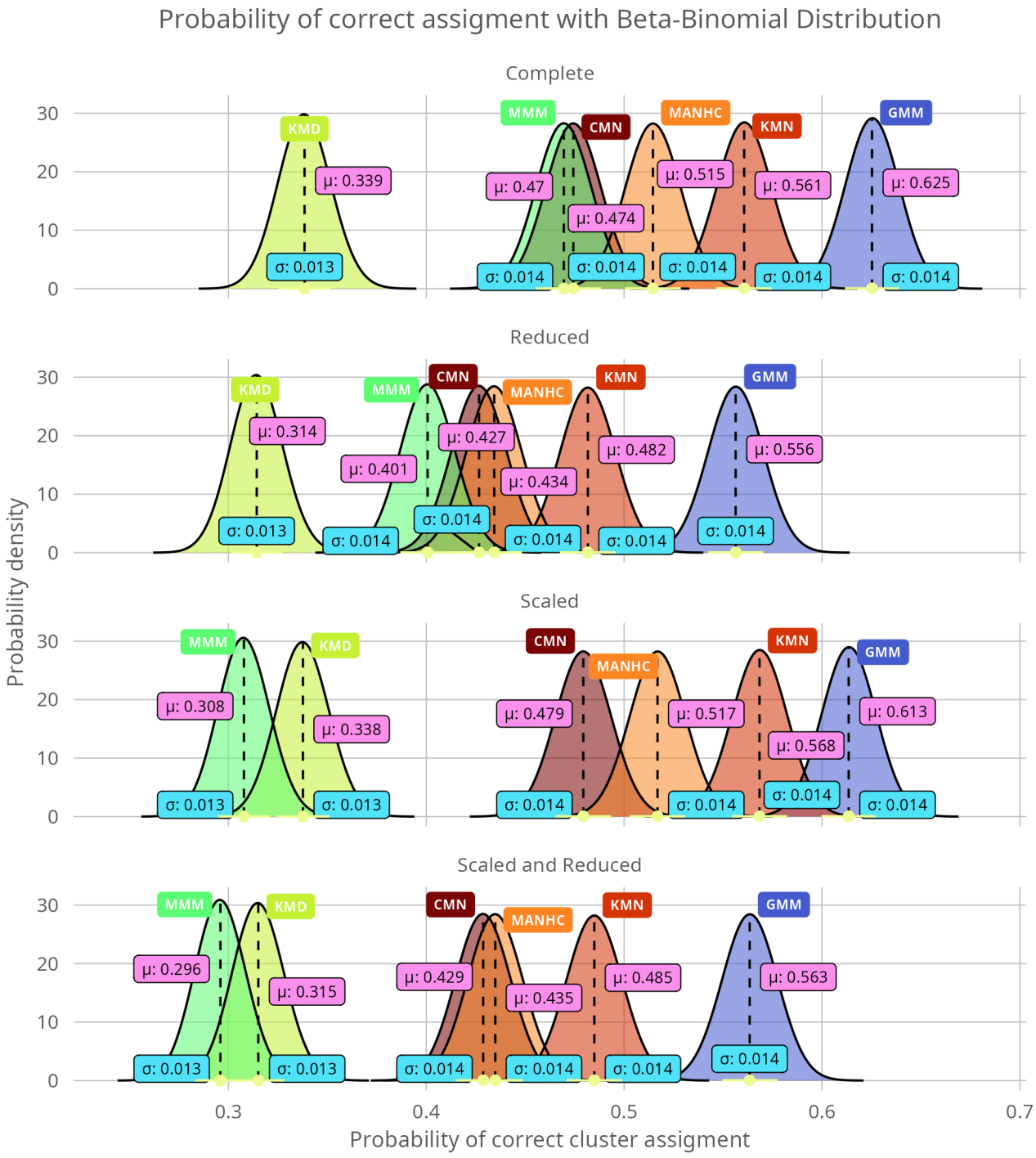


Figure 5.2: Beta-binomial distribution as a quality metric in Simulated Multivariate Normal Distributions

In Figure 5.2 the Beta-Binomial distribution plot indicates that the highest scores of correct clustering were obtained by GMM, located on the furthest right of the figure. The mean difference between GMM and KMN, the second-best performing algorithm, was around 0.05 for

all data types. The first three algorithms (GMM, KMN, and MANHC) scored approximately 0.07-0.1 points, with MANHC performing the worst. The degree of overlap between the top three algorithms varied depending on the data type, with MANHC being more overlapped with KMN in complete and reduced data. However, when data was scaled, KMN and GMM had increased overlap.

CMN performed similarly across different data types, overlapping almost entirely with MANHC when any scaling was involved and partially overlapping when it was not. KMD had better scores in complete and scaled data but performed poorly in reduced data, where it had the lowest performance. MMMK had the best performance when the data was complete, and it overlapped almost entirely with CMN, with a score of 0.47 points. However, its performance decreased when data reduction and scaling were involved, placing it as the worst-performing algorithm.

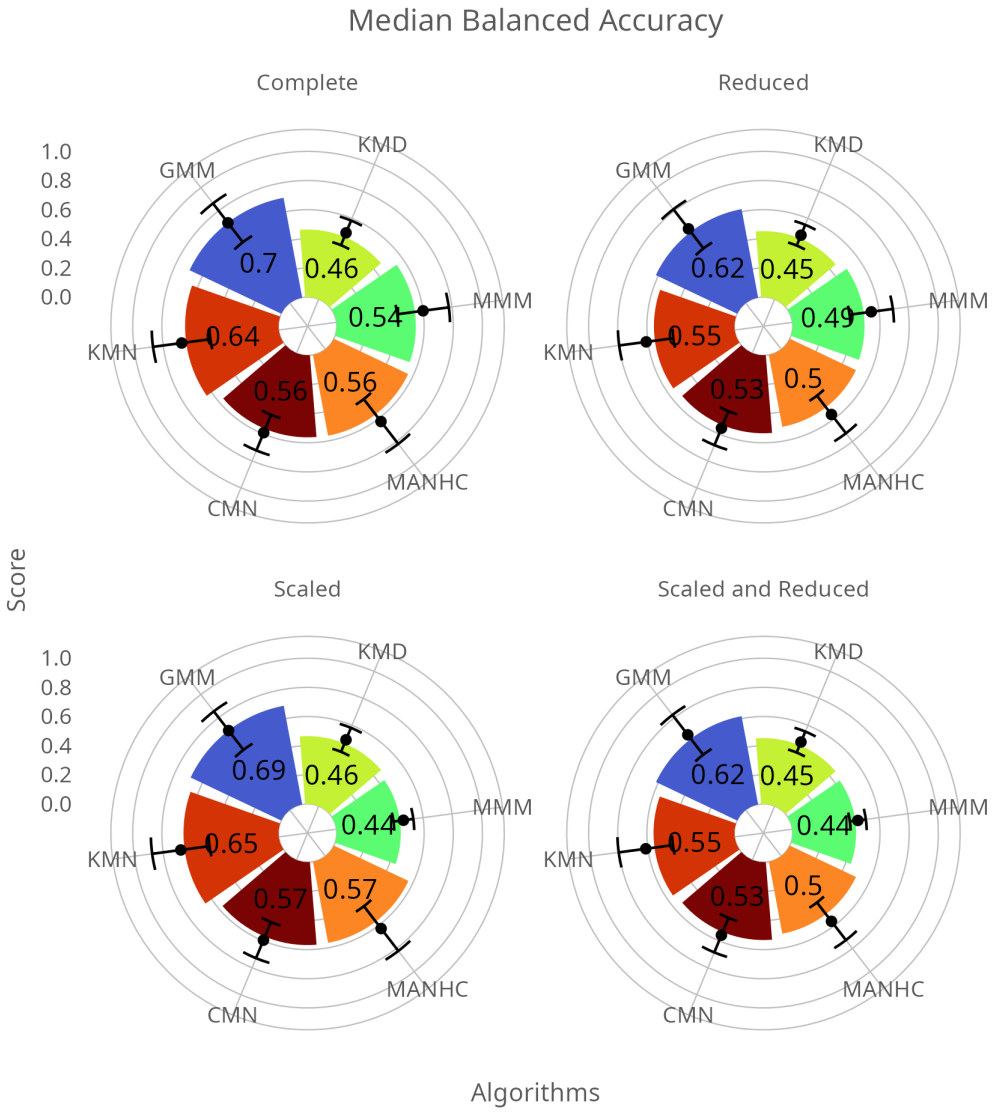


Figure 5.3: A median of balanced accuracy with mean and standard deviation in Simulated Multivariate Normal Distributions

In Figure 5.3, GMM achieved the highest score in the balanced accuracy metric in each data type. Despite a slight difference of approximately 0.04-0.05 points between GMM and unfiltered k-means, this result was based on many datasets, increasing its reliability. The trends followed by CMN and MANHC were similar, with CMN performing slightly better. Scaling the data improved by 0.01 points for both algorithms, but reducing the data decreased their scores by 0.04 and 0.07 points, respectively. MMMK and KMD were the second and lowest-scoring algorithms, respectively.

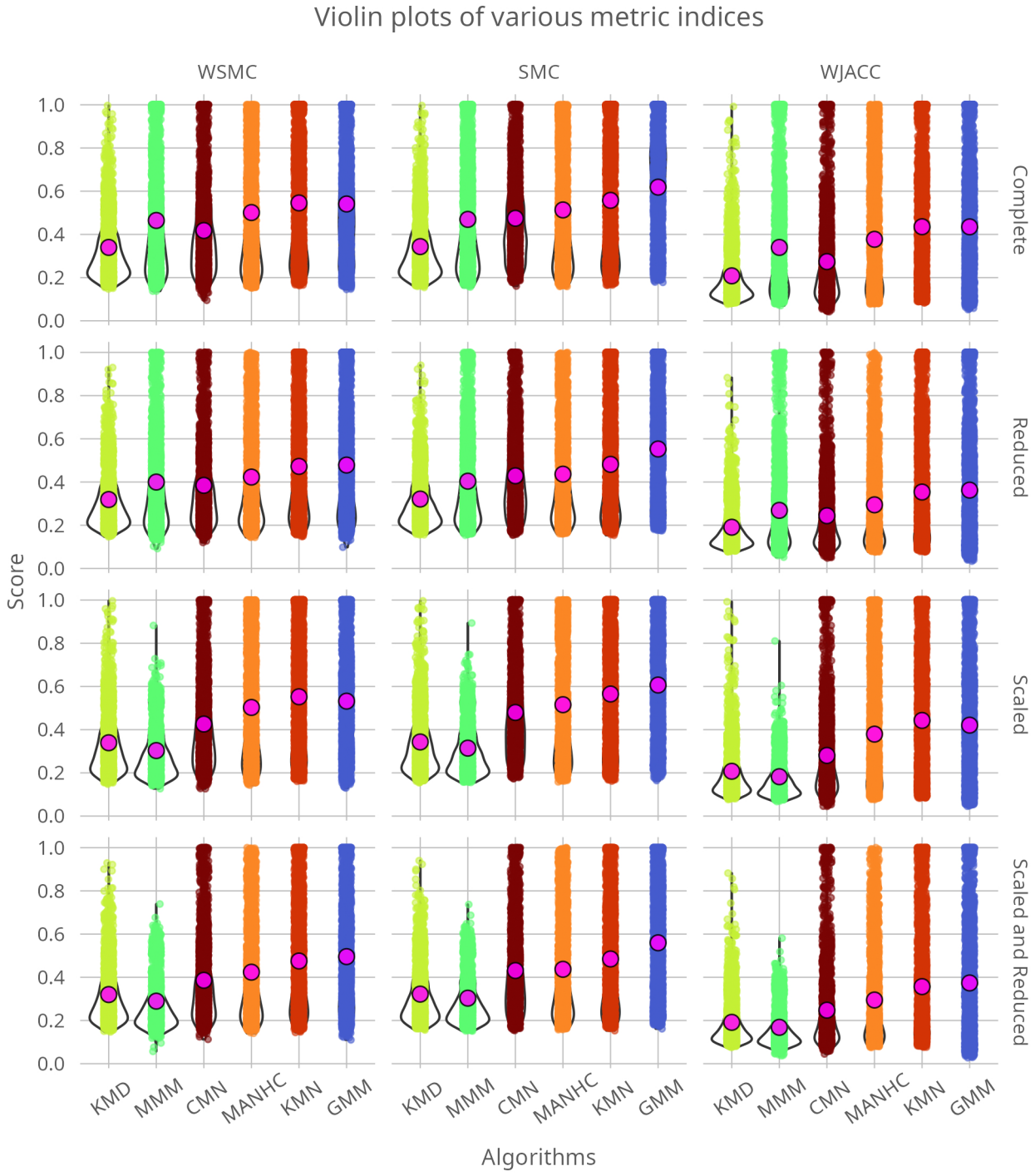


Figure 5.4: Other quality indices in Simulated Multivariate Normal Distributions

The mean values were generally located near the median value, except for some discrepancies in MANHC, MMMK, and KMN, especially for the first two. Nevertheless, even considering the standard deviation and the highest score, GMM still achieved the best result.

By design of simulated data, groups are generally balanced. In Figure 5.4, comparing SMC, WSMC and WJACC, GMM and KMN are generally on pair, when in a few cases, GMM performed slightly better, considering the mean value. However, when the data was not filtered or scaled, the Jaccard index showed a higher mean of k-means than in other algorithms.

GMM also has the broadest spectrum of values, which might be related to suboptimal initialization. It is easiest to observe in the WSMC and WJACC, where some points are closer to 0 than other algorithms. We can also observe that in distance-based algorithms altogether with MMMK, more density is gathered at the lower values. The most condensation is around 0.3 in the case of WSMC and SMC metrics and around 0.1 in the WJACC.

The metrics are highly correlated, which is a close agreement between metrics. GMM performed the best across all presented algorithms, with k-means in second place. MMMK, on the other hand, was the second worst, which is not surprising, given that it uses different probability distributions to estimate the parameters.

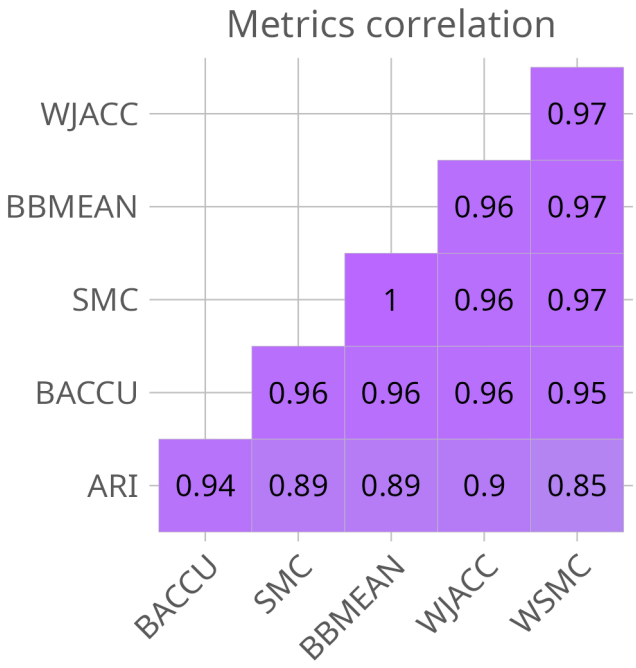


Figure 5.5: Metrics correlation in Simulated Multivariate Normal Distributions

5.1.2 Multinomial Mixtures

Results

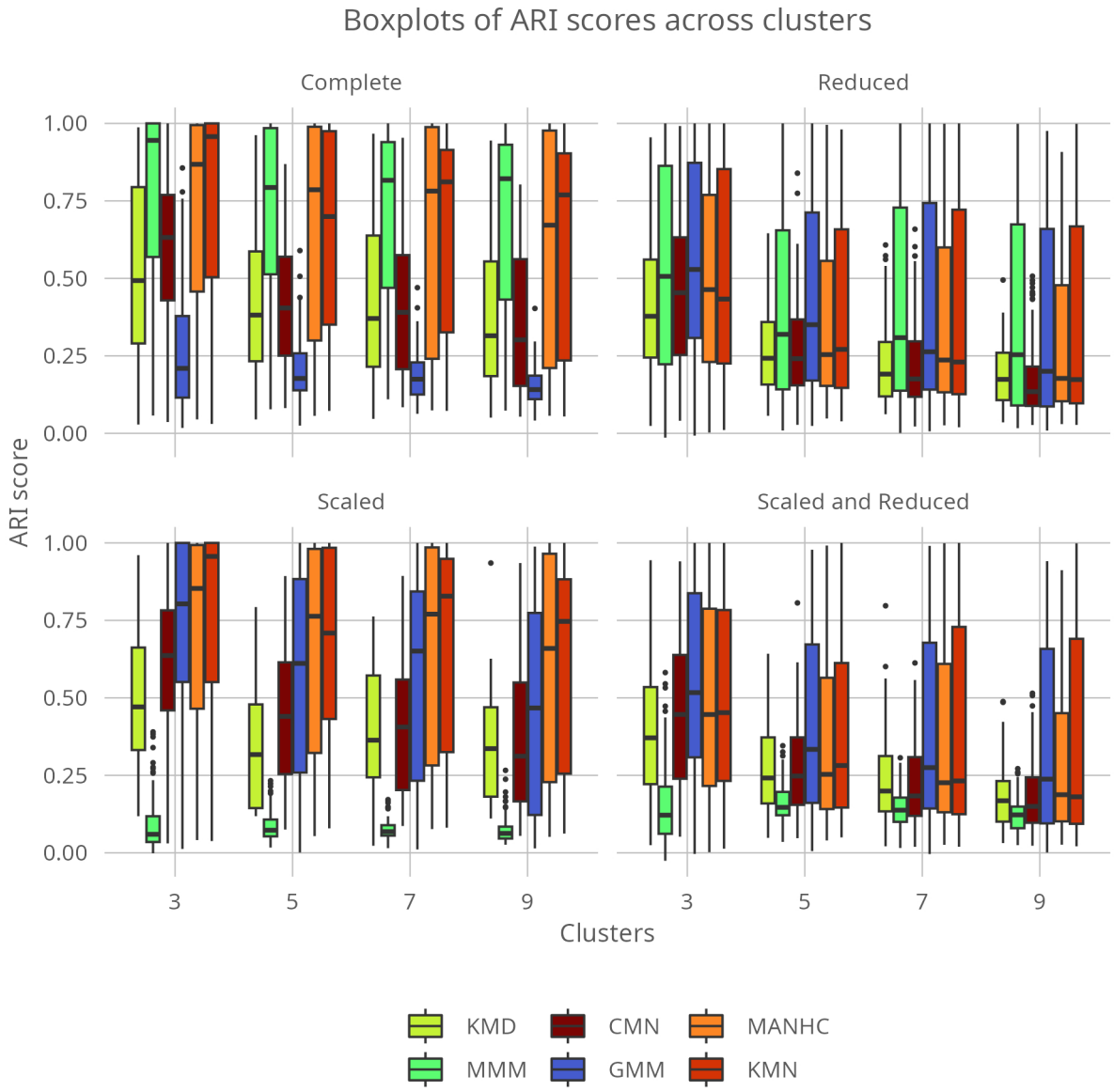


Figure 5.6: The ARI index in Simulated Multinomial Distributions

The ARI chart in Figure 5.6 shows that MMM had the highest score when the data was complete. With the increased number of components, MMM, MANHC and KMN performed slightly worse. On the contrary, KMD CMN had a steady decrease. GMM was only slightly affected but had the lowest performance in this data. When the data was reduced, the algorithm's medians were closer to each other. Initially, GMM was better than other algorithms, but with the increased number of clusters, MMM median score was the highest. Scaling of the data increased scores for all algorithms but MMM. Data reduction with scaling worsened the scores in general but slightly improved those of MMM. However, they were still the lowest.

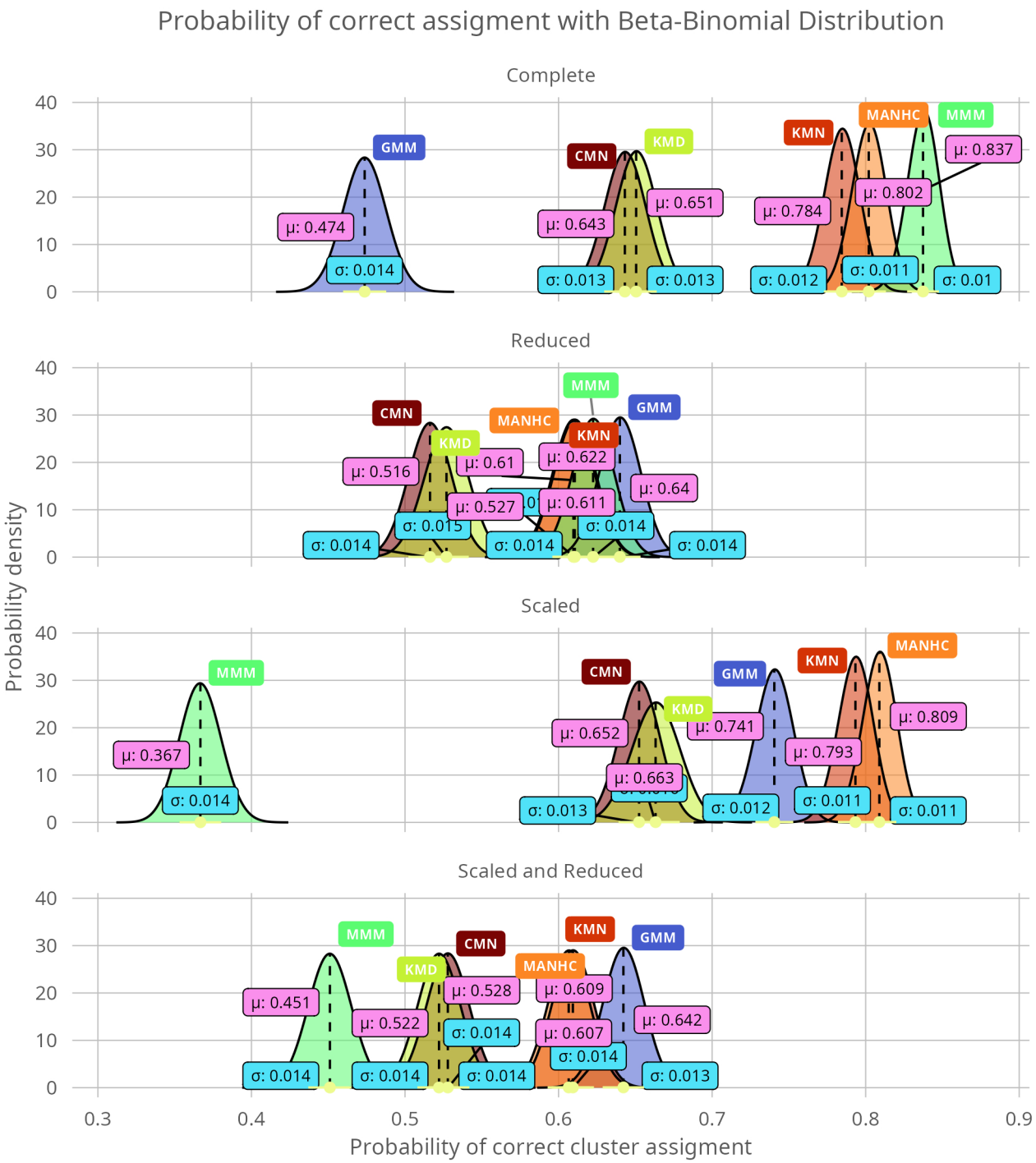


Figure 5.7: Beta-binomial distribution as a quality metric in Simulated Multinomial Distributions

In the figure 5.7, we can see that the highest result belongs to MMM. With approximately 0.84 points, it is the highest score across all types of filtration. Moreover, the probability density of the mean value is also the highest, which gives us more confidence compared to the other algorithms. However, it is valid only when the data is complete. Scaling of the data sent the MMM to the most left position, which was to be expected since the data no longer contained counts. Its probability density also became lower by almost 10 points. Here,

MANHC had the highest mean value of about 0.807. In addition, it overlapped with c-means and k-means, as the difference in the results was about 0.02. MMM scored as the second-best algorithm in the reduced data, overlapping with the GMM, which scored 0.69 points. Also, MMM and GMM have the lowest SD, but only 0.001, compared to the other algorithms. Finally, when the data was scaled and reduced, GMM was again first, with a mean score of 0673. MANHC, c-means, and k-means were highly overlapped, with scores of 0.06 points worse than GMM. MMM was again the lowest-performing algorithm, but its results were better by 0.119 points than when the data was only scaled.

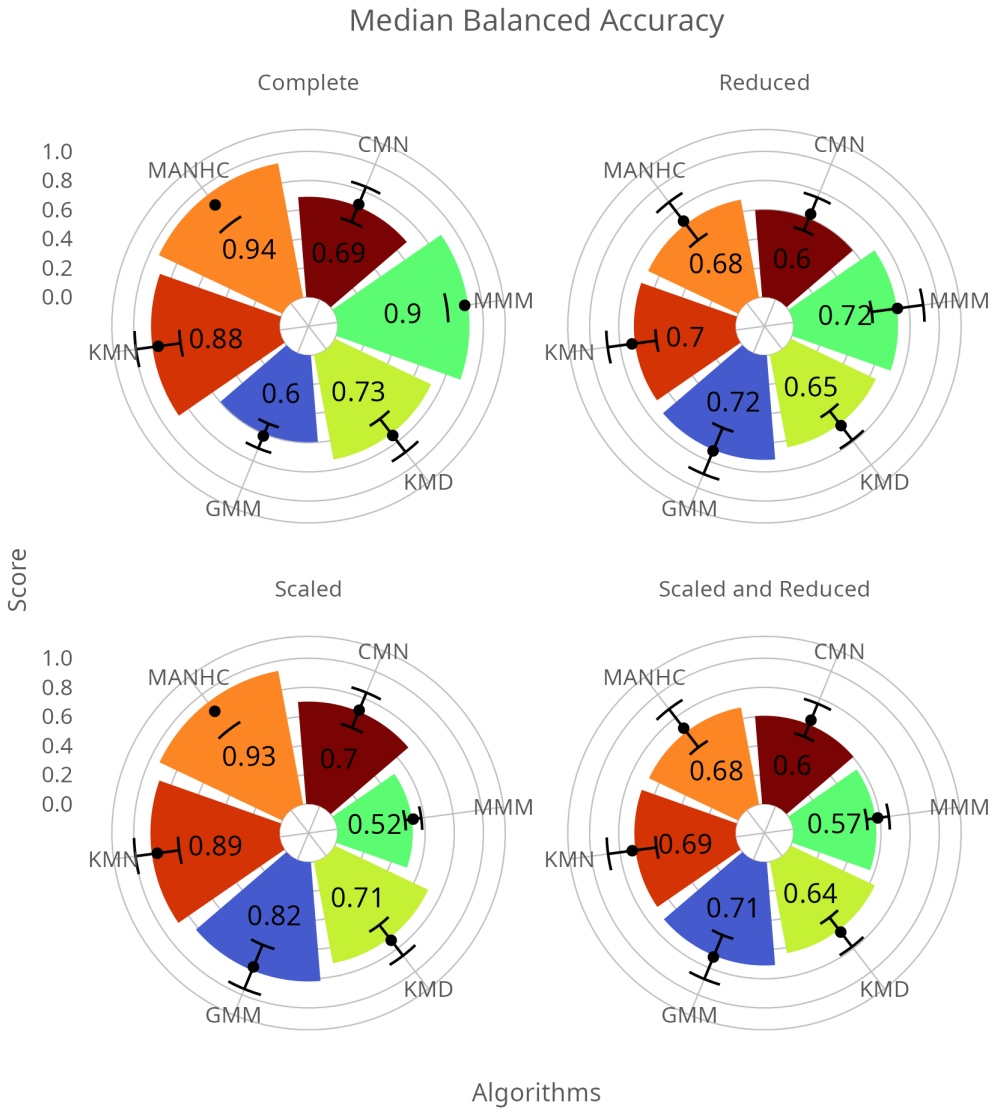


Figure 5.8: A median of balanced accuracy with mean and standard deviation in Simulated Multinomial Distributions

The index of balanced accuracy shows that distance-based algorithms scored the highest results. We can see that MANHC scored 0.94 in the complete data when MMM was about 0.03 points lower. However, in MMM, the black dot of the mean value is in the same spot as the median value, whereas in the case of MANHC, it is below 0.9, with more spread.

GMM scored the lowest, but its mean was slightly higher than the median. Also, its standard deviation is relatively small compared to other algorithms. After the data scaling, we can see that MANHC was barely affected. The same is true about KMN, but not for CMN, which scored higher by 0.03 points. KMD, on the other hand, scored 0.03 points lower. Data scaling enabled GMM to retrieve more information, improving its initial result by almost 0.21 points.

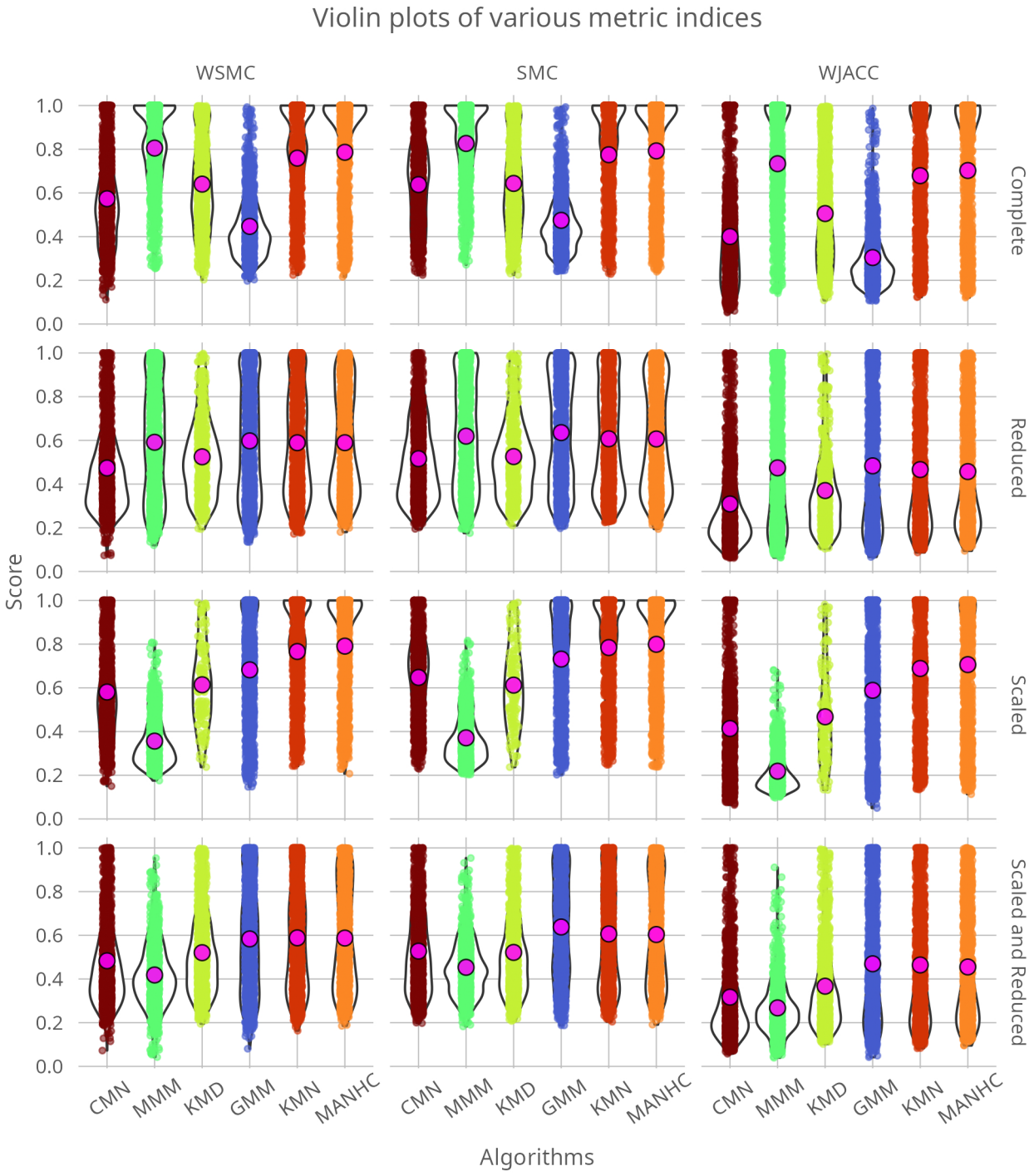


Figure 5.9: Other quality indices in Simulated Multinomial Distributions

Contrarily, it affected MMM negatively, as it received the lowest score across all algorithms.

We also saw it in the beta-binomial distribution, where MMM was on the very left side of the figure.

Data reduction impacted scores negatively for all algorithms but only partially for MMM. Notably, variance decomposition had more impact on distance-based algorithms than model-based ones. GMM had a better score when the data was reduced than in the complete data. Additionally, it outperformed MMM by only 0.01 points. When the data was scaled and reduced, MMM performed better than in the case when data was only scaled. GMM had the highest score of 0.74 points, losing only 0.06, while the other distance-based algorithms lost between 0.16 to 0.25 points, apart from k-medoids, which lost only 0.06 to 0.08 points.

The Figure 5.9 presenting the three indices shows that MMM had the highest scores when the data was complete. It is visible comparing the mean value, as well as density. In MMM it is more condesed close to 1, than in the other algorithms. Contrary, GMM had the lowest scores, which is also seen by the wider distribution near the lower values. KMN and MANHC performed only slightly worse than MMM.

Reduction of the data decreased the score of all algorithms but increases the mean value for GMM. Scaling of the data makes MMM scoring the lowest across all algorithms, elevating HC and KMN. Finally, when the data is scaled and reduced, the mean value of GMM is higher than in the other algorithms. MMM keep the lowest results, which are now slightly higher than when the data was only scaled. From the perspective of MMM, important information is lost during the scaling. It is expected since the algorithm operates on models of Multinomial Mixtures.

In the case of simulated mixtures of multinomial distributions, the correlation of various metrics is even higher than in Normal distributions. However, it is more challenging to point out the best algorithm. Distance-based algorithms had very high scores overall. They did exceptionally well when the data was scaled, which was expected. However, MMM has the best performance if we deal with data that is not transformed or reduced. Notably, if the data is reduced, it gives similar results as GMM. In addition, MMM was the best if we measured the raw number of successes.

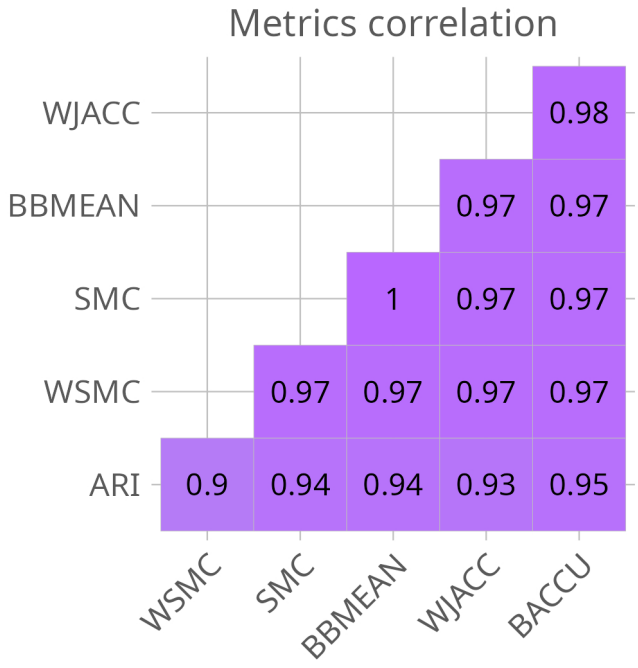


Figure 5.10: Metrics correlation in Simulated Multinomial Distributions

5.2 Real data analysis

5.2.1 Somatic Mutation Counts

The first dataset concerned somatic mutations in DNA in cancer patients. Somatic mutations in cancers are an essential and extensively researched topic. Somatic mutations can be caused by exposure to exogenous or endogenous mutagens or during DNA replication errors. Two major types of somatic mutations are distinguished: driver and passenger. Driver mutations are considered to confer cell growth and are related to cancer development, while passenger mutations have less or no impact on cancer growth in the organism. The difference between driver and passenger mutations can be caused by their positions in DNA or their impact on transcripts (mRNAs, proteins) produced. If a mutation hits an exon, it is more likely to become a driver. Contrary mutations in introns would rather be passengers. In protein-coding, the nucleotide substitution in the DNA can be synonymous or non-synonymous. Synonymous somatic mutations leave the amino acid unchanged, while non-synonymous results in modifying the amino acid or producing no amino acid. However, the problem of distinguishing between driver and passenger mutations is much more complex. Numerous studies are trying to estimate the impact of mutations on the risk of cancer development [32].

We wanted to cluster patients diagnosed with different cancer types based on counts of somatic mutations in genes. Due to the complicated problem of distinguishing between driver and passenger mutations, in our computational experiments, we take numbers (counts) of mutation occuring in genes as our observational vector data. The hypothesis behind this computational experiment is that the recorded information (mutation counts in genes) can be used to distinguish between various types of cancer. The second goal is to determine which clustering algorithms will perform best in the clustering task when the quality criterion agrees between unsupervised clustering results and the ground truth.

Original, raw data was taken from The Cancer Genomic Atlas (TCGA)[33]. BAM files submitted to TCGA were converted to FASTQ format for the initial analysis. DNA sequences were aligned using BWA-MEM or BWA-aln, depending on the read length. The human genome used as a reference was in version GRCh38.d1.vd1. In parallel to the BWA algorithm, GATK was also used to improve the alignment quality. Five Somatic Variant Callers were used in the next step: MuSE, MuTect2, VarScan2, SomaticSniper, and Pindel.

The data we used in the experiment was annotated using somatic variant caller Mutect2. It is one of the few popular tools to detect SNVs (Single Nucleotide Variations) and indels (insertions and deletions in the DNA) [34]. According to Mutect2 probabilistic model, we filtered predicted somatic mutations. The data we extracted included the following information: sample number, which reflects a single, anonymized patient, name of the gene in which mutation occurred and frequency of mutations. For the experiment, we have chosen 10 different types of cancer, semi-arbitrally.

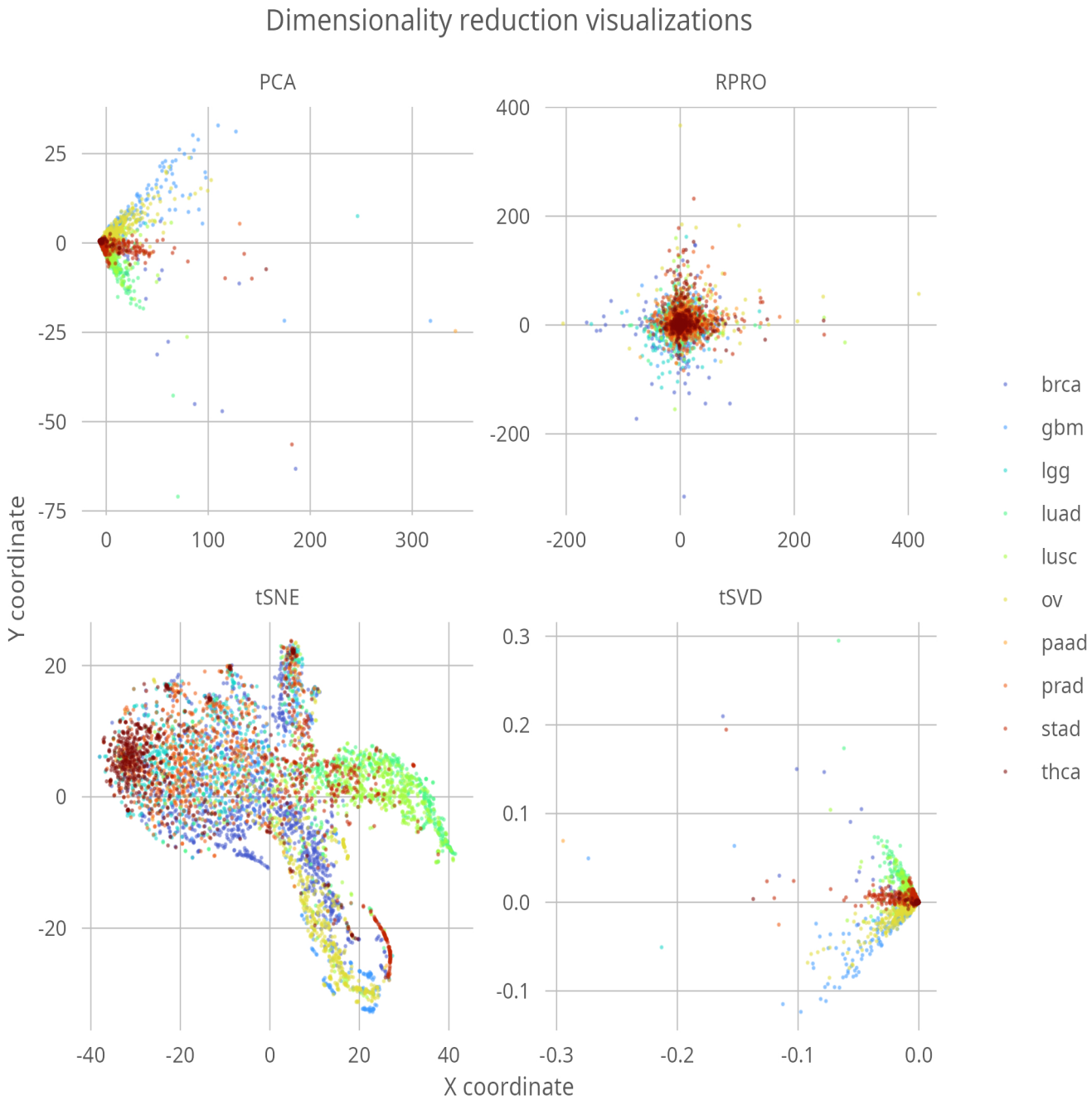


Figure 5.11: Visualization of dimensionality reduction techniques of TCGA Somatic Mutations Counts

Variant Effect Predictor was used for each file. Ensembl database is a comprehensive source of genomic data, and the Variant Effect Predictor (VEP) is a part of it. It is a multipurpose tool to annotate possible mutation effects. It can predict whether the mutation was synonymous, missense, or stop gained. VEP also estimates the potential impact on the organism, which ranges from low to moderate up to high. The vital characteristic of VEP is that it provides many predictions regarding possible gene variants or transcripts. It means that numerous different variants might describe one mutation. In rare cases, we can obtain contracting suggestions, like missense and synonymous variants of the exact mutation location. It also provides information on possible gene mutations that occurred in the desired format. We choose Ensembl genes over common HGNC names. The structure presented by Ensembl seems to be more consistent and not prone to drastic changes, as in the case of HGNC - when one gene might have had a few different aliases.

To filter data variants, if a patient in a particular location had more than one variant, they were added up. The variant that occurred the most frequently, along with its corresponding gene, was selected for analysis. If there were two genes, the first one was taken.

The complete dataset consisted of more than 10 000 observations and 35 000 variables.

As shown in Figure 5.11, the separation of groups is relatively poor regardless of the dimensionality reduction technique used, with only three or four groups appearing more isolated than the rest. The random projection method resulted in the lowest separation, while the t-SNE plots showed the most distinct groups. Although separated groups can also be seen in PCA and t-SVD, they are more overlapped than in the case of t-SNE. It is worth noting that t-SVD is mirrored PCA, but on a different scale.

Results

Figure 5.12 displays the ARI index scores for various algorithms. HC and GMM achieved the highest scores, with MMMK ranking third. The median value increased for all algorithms up to around four components before decreasing, as observed in complete, reduced, and scaled-and-reduced data. This trend was particularly evident in HC and MMMK. Notably, after scaling, most algorithms remained close to zero, including GMM. The MMMK variant performed better due to its k-means initialization, but the median score for HC was still twice as high. Both model-based algorithms consistently scored in the top three. After reduction and reduction with scaling, more algorithms achieved positive scores. Nonetheless, the highest single scores were obtained by GMM in the two-component mixture with complete data and MMMK in the two-component mixture with reduced data. These results suggest opportunities for future improvements.

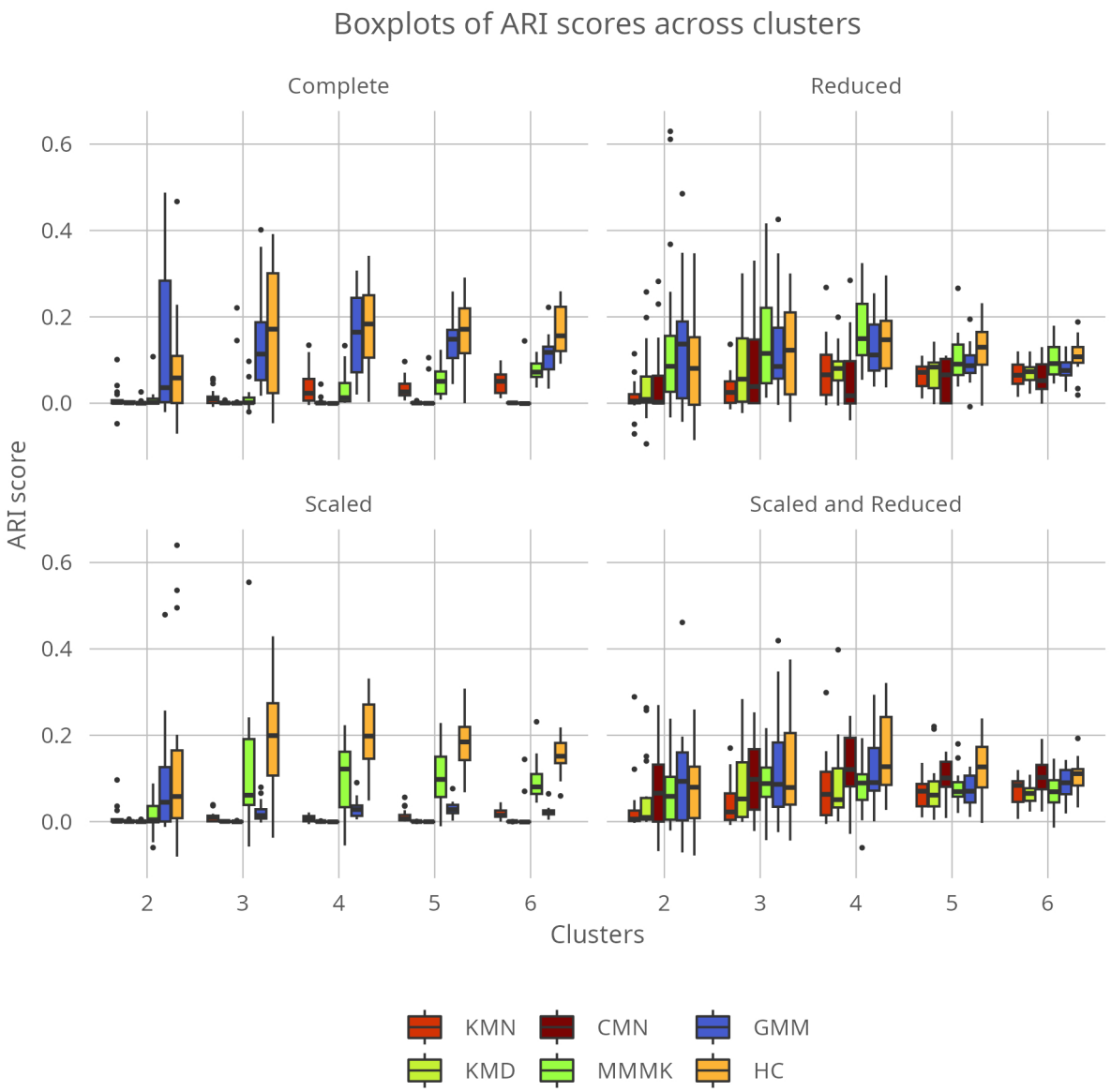


Figure 5.12: The ARI index in TCGA Somatic Mutations Counts

Looking at Beta-binomial distribution in Figure 5.13, we can see that the highest score belongs to HC with 0.462, when the data was complete. It also performed best when the data was scaled and scaled and reduced. However, each time it was also overlapped with other algorithms. Right after HC, we can see that MMMK scored very high when the data was reduced. Its score is comparable with the highest score of HC, performing only by 0.003 points less. Scaling visibly stretched score range of the algorithms,. Suprisingly GMM was the last and MMMK was again second. When the data was scaled and reduced, all algorithms performed within range of point 0.423 to 0.45. Here MMMK was the last and it was almost entriely overlapped with GMM that scored 0.425.

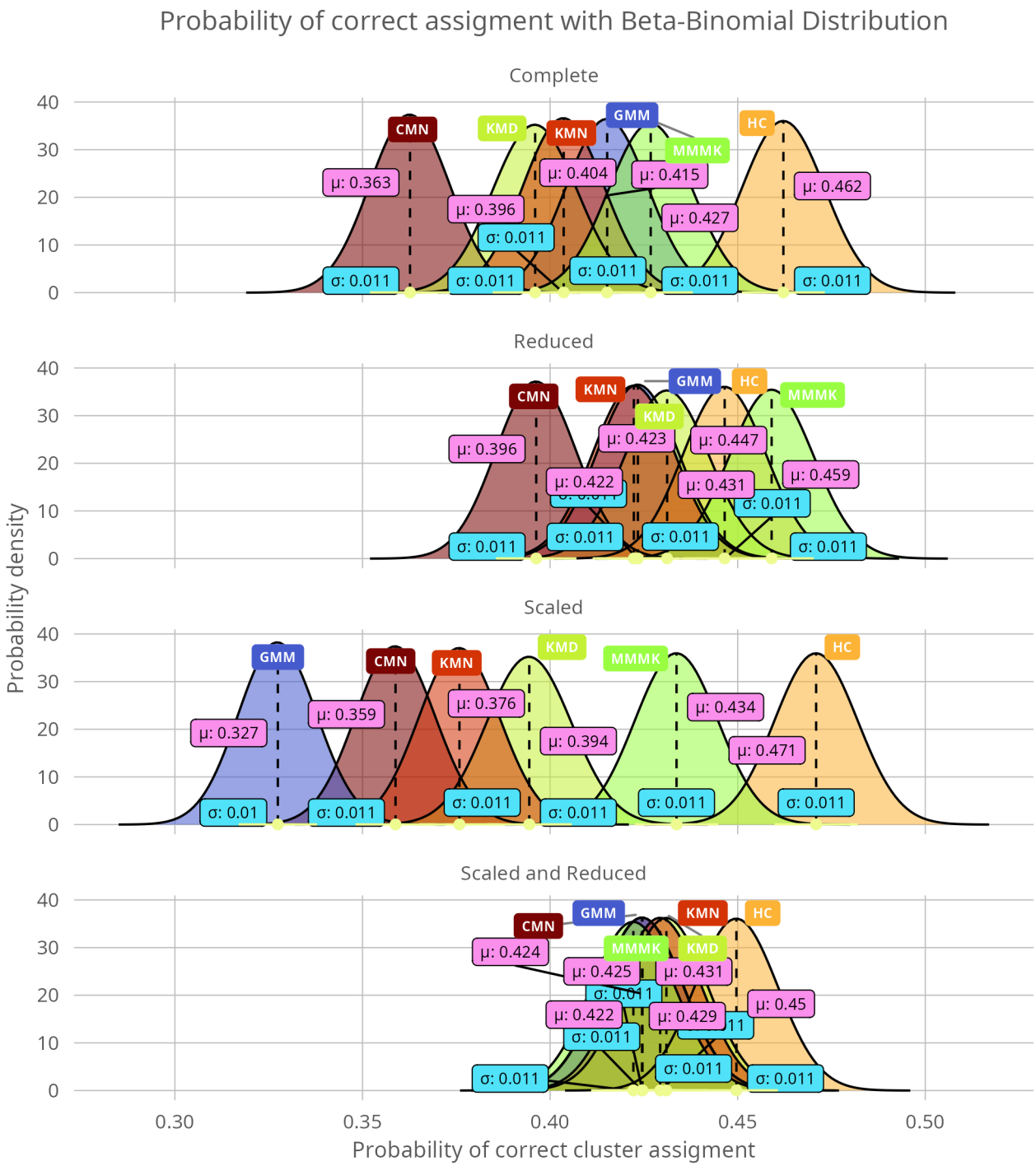


Figure 5.13: Beta-binomial distribution as a quality metric in TCGA Somatic Mutations Counts

The highest score in the balanced accuracy index, as shown in Figure 5.14, was achieved by HC across different data types. MMMK was the second-best performing algorithm, with a score of 0.59 points, which scored highest when the data was reduced. However, MMMK results varied significantly depending on the data type. In general, scaling improved the results of HC and MMM, but it worsened the scores of the other algorithms. On the other hand, reducing the data improved the performance of different algorithms.

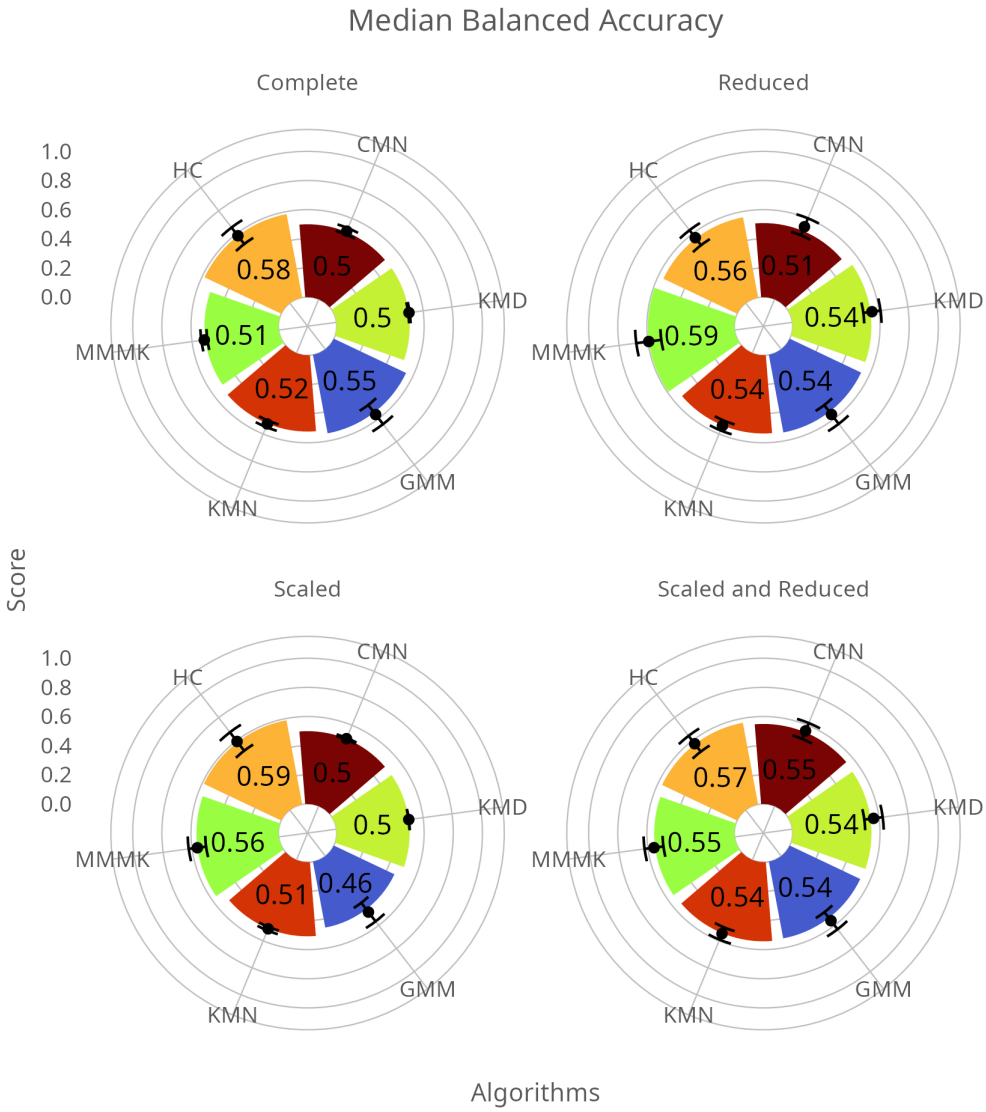


Figure 5.14: A median of balanced accuracy with mean and standard deviation in TCGA Somatic Mutations Counts

The data reduction positively impacted GMM, but only when the data was scaled. In complete data, GMM scored almost 0.55, and scaling worsened the result by 0.09. In scaled and reduced data, it scored 0.54, the same as in the case of only reduced data, which was only 0.01, worse than its best score.

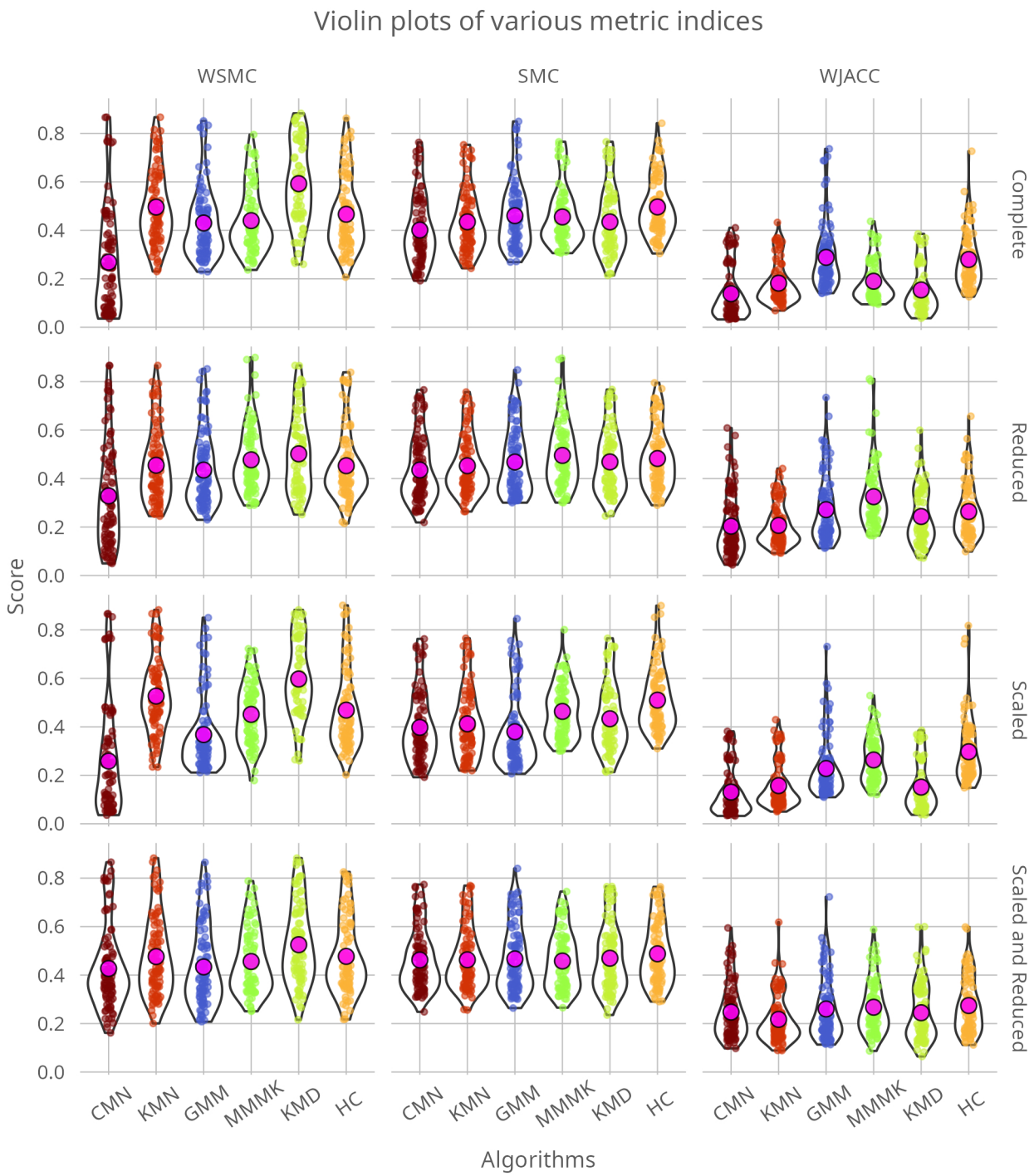


Figure 5.15: Other quality indices in TCGA Somatic Mutations Counts

In Figure 5.15, there is not much agreement between the three presented metrics. Each of the metrics presents a different report. According to the WSMC metric, KMD performed best in all datatypes. It has the highest median, slightly about 0.6 points, compared to the other algorithms. Also, it had the highest density, about 0.8 points, compared to the others. The lowest-performing algorithm was CMN. Then, model-based algorithms were placed in the middle. We can see that the tendency remains through different data types, but the differences are more emphasized or shallow across the algorithms. When the data was complete or scaled,

the difference was much more visible than when the data was reduced and scaled and reduced.

In contrast, looking at the SMC metric, the situation is slightly different. GMM and MMMK are the second and third best-performing algorithms, respectively. In complete data, we can see that GMM had some scores even higher than HC, but there were only a few of them, and more density was placed below 0.6 points. When the data was reduced, MMMK slightly outperformed HC, but the difference was negligible. With fewer variables, the differences in scores across algorithms were lower.

Finally, the WJACC metrics show the most significant differences across the various algorithms. In the complete case, GMM was on par with HC. After data reduction, MMMK performed best, with the highest mean and the lowest values higher than the other algorithms. After scaling the data, MMMK lost some of the highest-scoring values, making it the second-best-performing algorithm behind HC. When the data was scaled and reduced, GMM, CMN, KMN, and KMD performed slightly better, while MMMK and HC performed worse than in the previous data type. Therefore, the mean across algorithms is similar, ranging between 0.2-0.3 points.

While all correlations are positive, the results vary. Especially the ARI index and BACCU metric exhibit more disagreement with the other metrics but at the same time, they are more consistent with each other.

When taking a large number of observations in different cancers, the differences in somatic mutation counts gradually become less observable. This may be due to the homogeneity of somatic mutations across different cancers. Additionally, due to the lack of an explicit probability model for selecting an appropriate variant, we receive a mixture of similar mutations.[35].

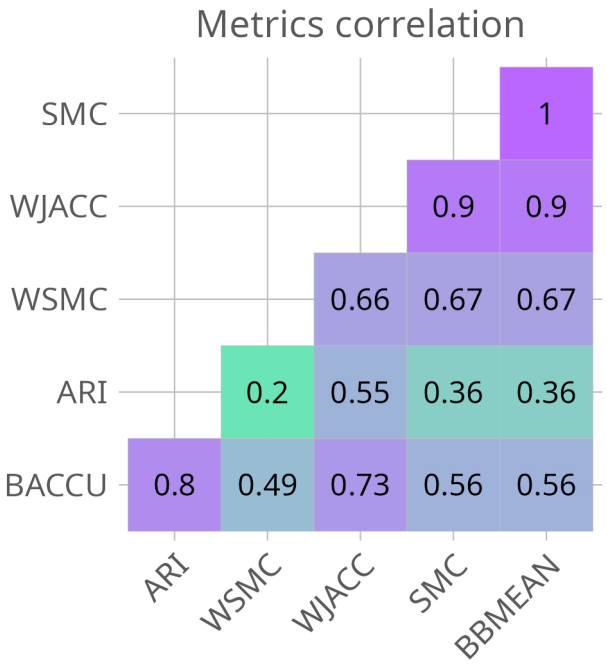


Figure 5.16: Metrics correlation in TCGA Somatic Mutations Counts

5.2.2 Gene Expressions

Gene expression is a series of events leading to information displayed in the cell. Roughly speaking, gene expression lets phenotype arise, something we can observe from genotype.

We can consider the genome as information storage. However, it cannot pass the information to cells on its own. To use biological information encoded in the genome, enzymes and various proteins must participate in complex biochemical reactions that lead to Genom expression. The first product of Genom expression is transcriptome, a group of RNA particles. They come from those protein-coding genes that the cells need the most. The transcriptome is created during transcription when genes are rewritten as RNA particles.

Some of those particles are called mRNAs or messenger RNA. The primary role of mRNA is to function as a template for translation. Their sequences are first translated to amino acids during this process, which then builds functional proteins. During various events, like gene mutation, the expression level of mRNA might be increased, decreased or even halted. It can be related to multiple diseases, including cancer [36].

cBioPortal is an interactive interface to the resources such as TCGA. It provides open access to molecular profiles and clinical attributes of different cancer genomic studies.

The data is primarily multidimensional and contains, but is not limited to, data on DNA methylation, mRNA and microRNA expression or phosphoprotein level data (RPPA). We used mRNA (messenger RNA) expression data for our analysis. First, we wanted to confirm if such kind of data contains enough information to distinguish between different types of cancer. In other words, various cancers show different expression patterns.

The gene expression format of cancers was already suitable for our input purposes, requiring only transposition to convert sample numbers into observations and genes into variables. The complete dataset consisted of more than 10 000 observations and approximately 35 000 variables.

In the Figure 5.17, PCA shows limited distinction within groups, with only a small section of the blue tail and a few orange points pointing to the lower left corner of the figure. Random Projection reveals some groups, but they are centered and overlap. tSVD is similar to PCA but provides better distinction within the cancer group. However, a clearer visualization is achieved with the tSNE plot, which shows well-separated groups with minimal overlap.

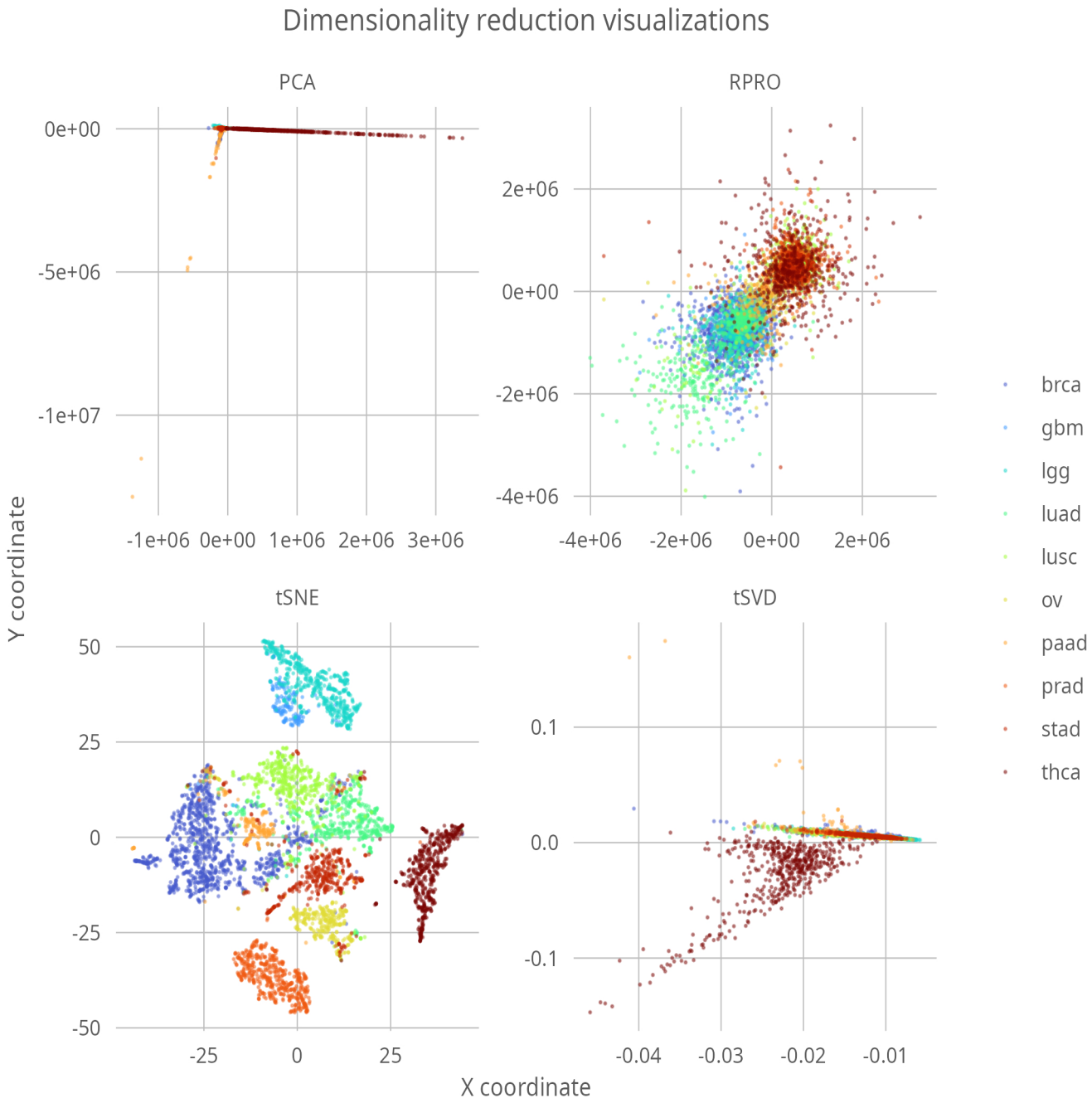


Figure 5.17: Visualization of dimensionality reduction techniques of TCGA Gene Expressions

Results

The analysis of the clustering algorithms using the ARI index is presented in Figure 5.18. Model-based MMM and HC algorithms achieved the highest scores when the data was complete, with MMM scoring higher than HC. Although GMM and MANHC were almost on par in the case of reduced data, MMM outperformed both algorithms in both scenarios.

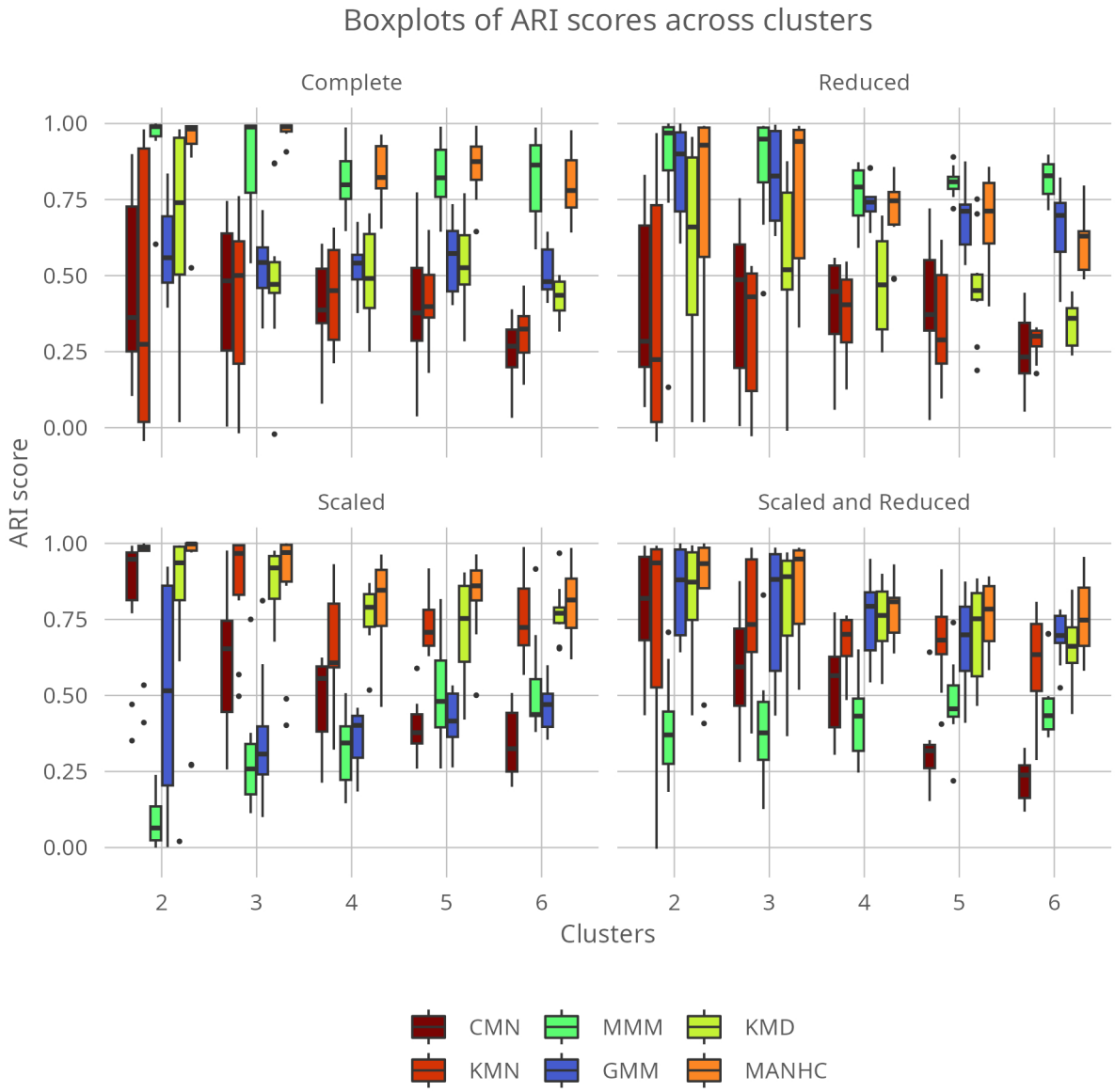


Figure 5.18: The ARI index in TCGA Gene Expressions

On the other hand, when the data is scaled, MMM generally performs worse than other algorithms. However, the performance of MMM increases with the increased number of clusters, which is not easily observed in the case of other algorithms.

Furthermore, we observed that most algorithms have the broadest spectrum of results in two clusters, which gradually decreases with increased clusters, focusing more on the median value. However, this is not necessarily the case for the MMM algorithm, where the scores are

tightly focused around the median, even in two cluster cases.

Finally, it's worth noting that most distance-based algorithms (excluding HC) have lower performance with the increased number of clusters.

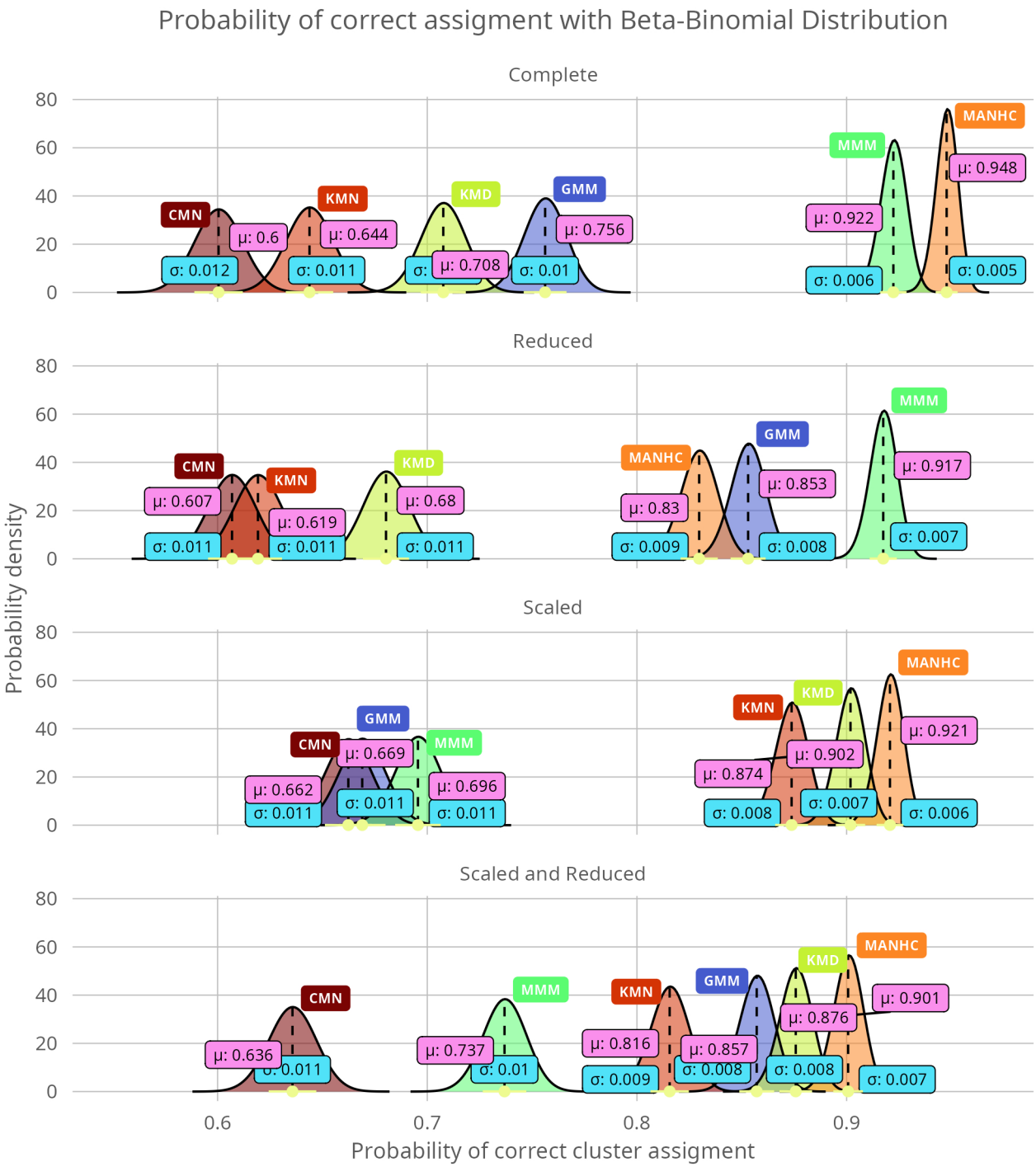


Figure 5.19: Beta-binomial distribution as a quality metric in TCGA Gene Expressions

Figure 5.19 shows the beta-binomial distributions of correct assignments. MANHC scored the highest on the right side of the pitch, followed by MMM. These two algorithms have slightly overlapping scores, with a difference of approximately 0.027 points. Furthermore,

they have the highest probability density, which provides more confidence in their probability of correct assignment than the other algorithms. Notably, both algorithms have standard deviations almost twice as low as the other algorithms, indicating that their scores are more focused on the mean.

When the data was scaled, the model-based algorithms performed visibly worse than the others, with performance levels similar to CMN, which had the lowest score in this data type. However, with the additional reduction of features, MMM showed better performance, with GMM scoring as the third-best algorithm with a result of 0.841

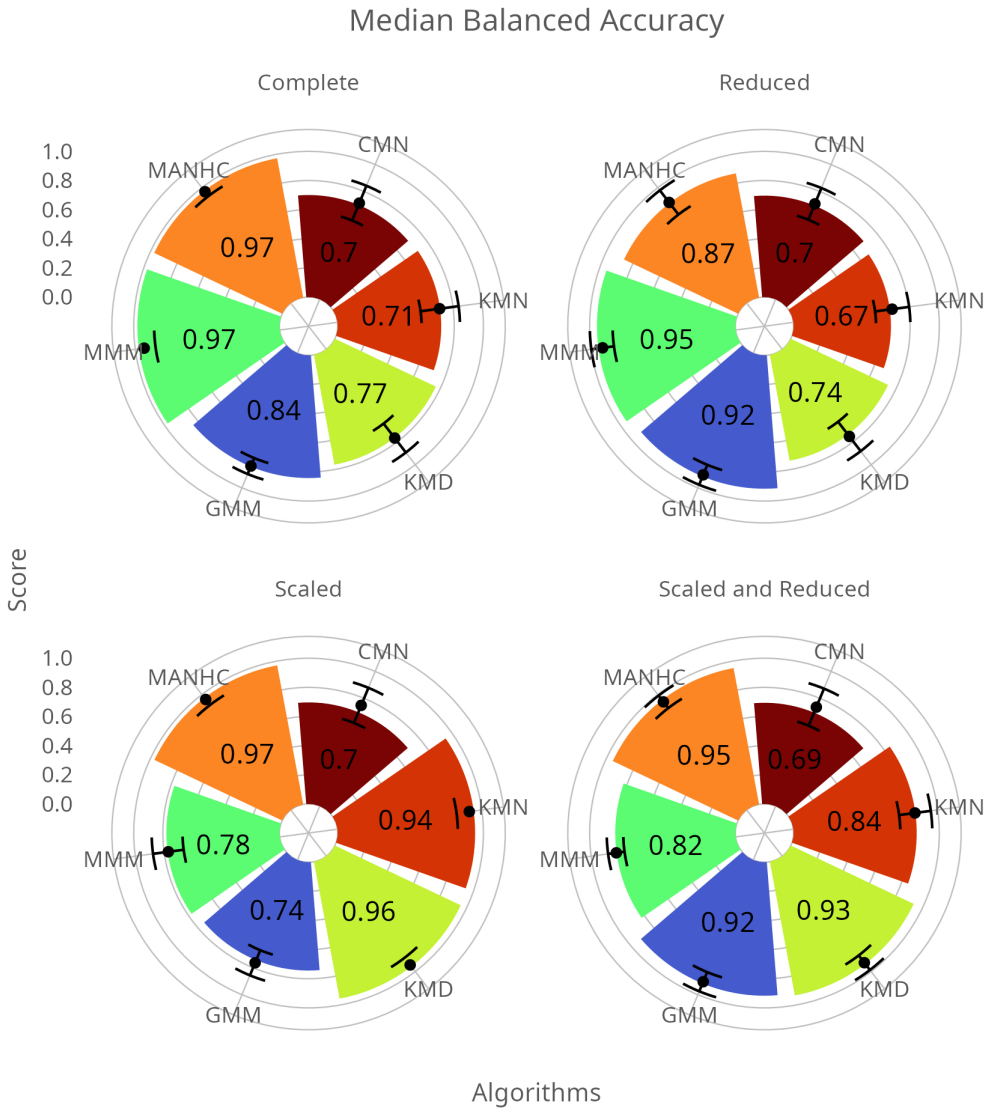


Figure 5.20: A median of balanced accuracy with mean and standard deviation in TCGA Gene Expressionss

Figure 5.20 shows the median balanced accuracy metric, where the means are almost identical to the medians in all cases, except for a slight difference in the scaled CMN. In the unfiltered data, while MMM and MANHC had similar scores, the results of the model-based algorithm were more spread out. However, MMM could correctly assign most data sets in

reduced data, whereas distance-based algorithms performed worse. GMM performed second best in this scenario. Data scaling increased the correct assignment of distance-based algorithms, especially in k-medoids. Finally, when the data was scaled and reduced, MANHC was still the first, but KMD was worse only by 0.02, followed by GMM.

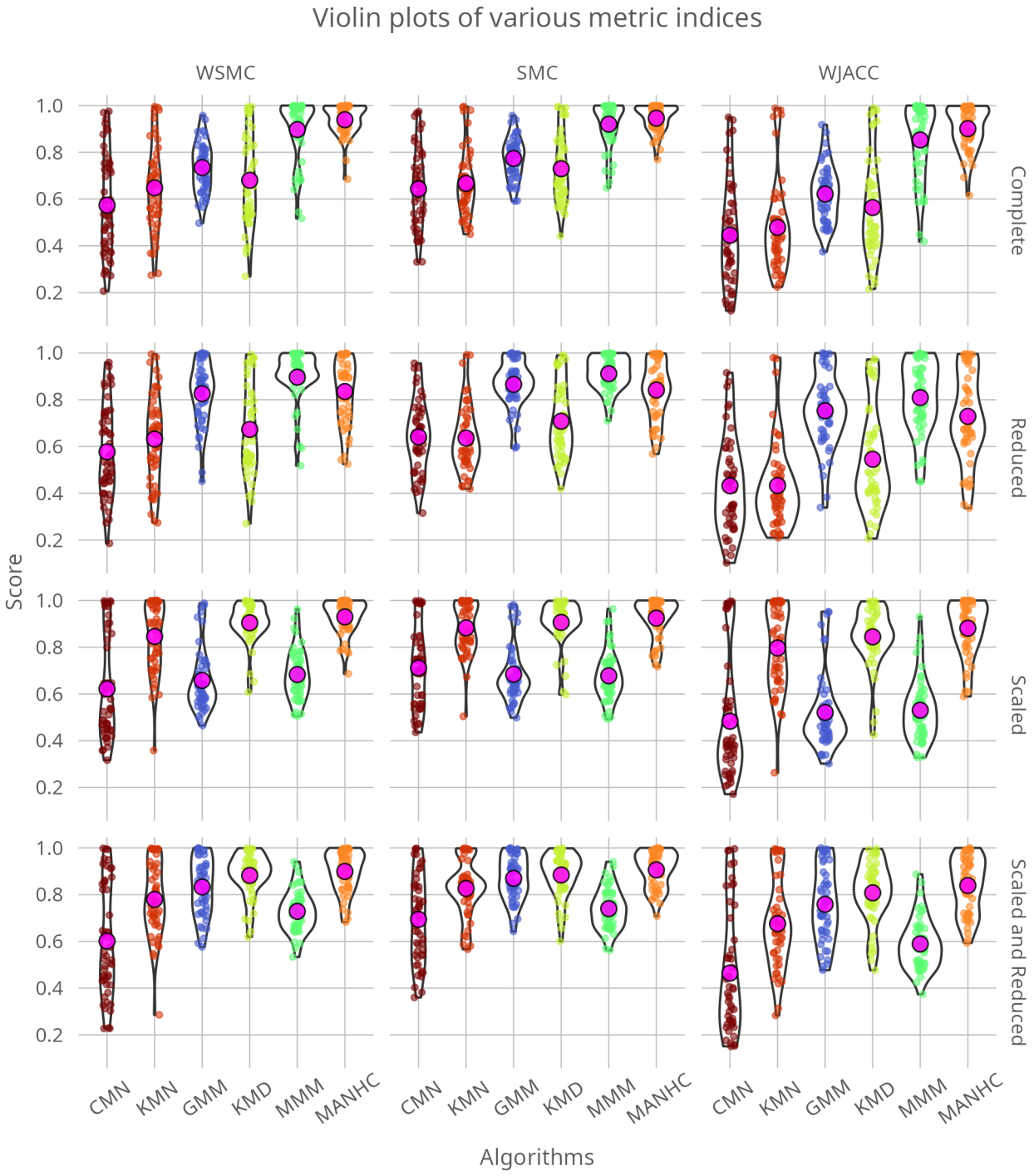


Figure 5.21: Other quality indices in TCGA Gene Expressions

Figure 5.21 presents the results of various metrics, where CMN had the highest spectrum of scores, indicating that it was the worst-performing algorithm. This can be observed in the

WSMC and WJACC metrics, where the violin plot is wider below the value of 0.4. KMN performed better than CMN, with fewer small scores and a mean primarily between bands 0.85 and 0.6, mainly in scaled data.

Regarding GMM, the data type played a crucial role in its performance. It scored worse than KMN only when the data was scaled, but after data reduction, it had better results, with a mean close to 0.9 points.

In Figure 5.22, all metrics show a high positive correlation, indicating significant agreement regarding algorithm performance. When the data was complete, MANHC achieved the highest score, but MMM was only slightly worse by approximately 0.025 points. Overall, the algorithms performed exceptionally well, consistent with the findings from the tSNE map.

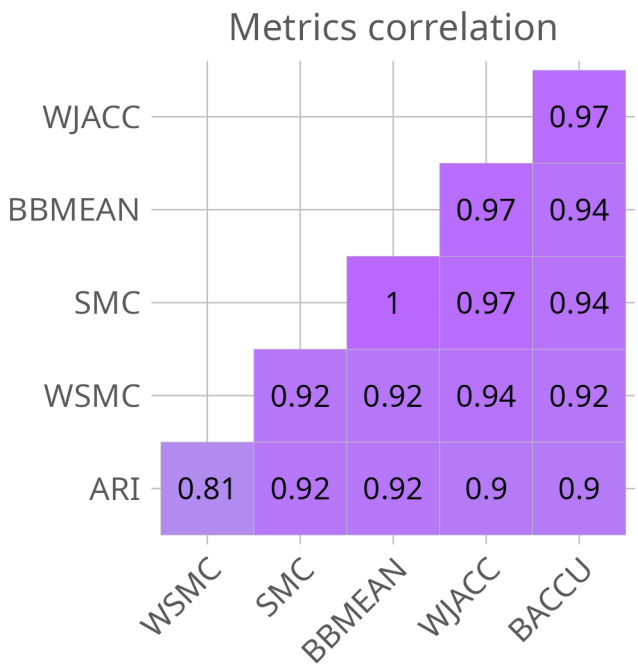


Figure 5.22: Metrics correlation in TCGA expressions

5.2.3 Codons frequency

Proteins are built from codons, with 20
amino acids coded by these codons. Each codon comprises three DNA bases. In the past, the DNA code was believed to be universal among all species, with most using the same codon of three nucleotides to code for a specific amino acid. However, it is now known that this statement is only partially valid, with many exceptions found in various species. Non-standard codons are frequently used in mitochondria genomes, for instance. The frequency of codon usage may also differ across organisms.

Bohdan Khomtchouk from the Section of Computational Biomedicine and Biomedical Data Science, University of Chicago, shared the dataset we used. The codon frequency set was built using CUTG (Codon Usage Tabulated from GenBank, which is available on the site (https://www.kazusa.or.jp/codon/). The data contains frequencies of codon usage by several diverse organisms. Each organism was assigned to its respective kingdom. The "Kingdom" is an abbreviation code consisting of a 3-letter corresponding to the names from the CUTG database. Data from UCI slightly differs from the original data, as the author describes that they manually changed the class of bacteria into archaea, plasmids, and bacteria proper.

The DNA type is coded by an integer representing the genomic composition in the given species. "SpeciesID" is a unique integer number that differentiates various species, along with the "SpeciesName". The codons' number in the column "Ncodons" was obtained by summing the codons for different species found in the CUTG database. Then, the number of codons was normalized by dividing each codon (like UUU, UUA) by the codons species sum, as listed in the "Ncodons" column. That is how frequencies of codons were obtained. All codons columns are floats with five decimal digits.

The goal of the original paper "Codon Usage Bias Levels Predict Taxonomic Identity and Genetic Composition", which used the data, was to build a machine learning classifier to distinguish between species. Meanwhile, we want to determine if we can restore the species' structure by dividing them into their respective kingdoms based on codon frequency with different unsupervised learning algorithms [37].

In this study, plasmids were omitted because they constituted the smallest group, accounting for only 18 observations. The final dataset consisted of about 13 000 and 69 variables.

In figure 5.23 all four methods reveal visible groups; however, their substantial overlap could pose a challenge for clustering.

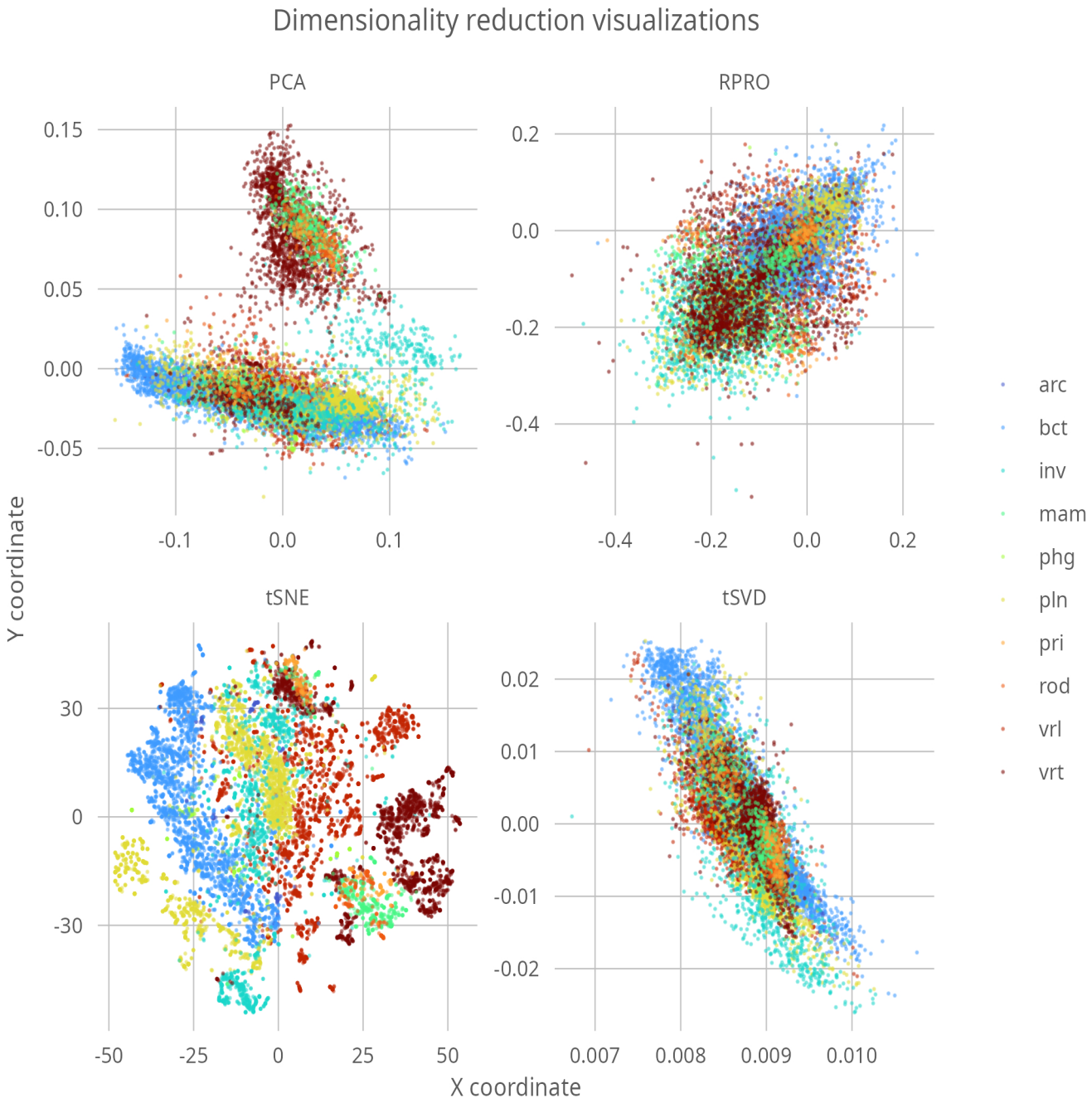


Figure 5.23: Visualization of various dimensionality reduction techniques of standardized Codons by animal Kingdoms

Results

In Figure 5.24 of the ARI index, MMM and GMM algorithms had the highest scores overall. In the complete data case, values of ARI for MMM are much higher than in any of the other algorithms. Also, its median score increases with the increased number of clusters. This tendency can also be seen in the other algorithms, but only for up to four mixture components. After that, median scores decrease for all algorithms but slightly for GMM. The median of MMM increased again in a mixture of five components, to fall a little when six groups were present.

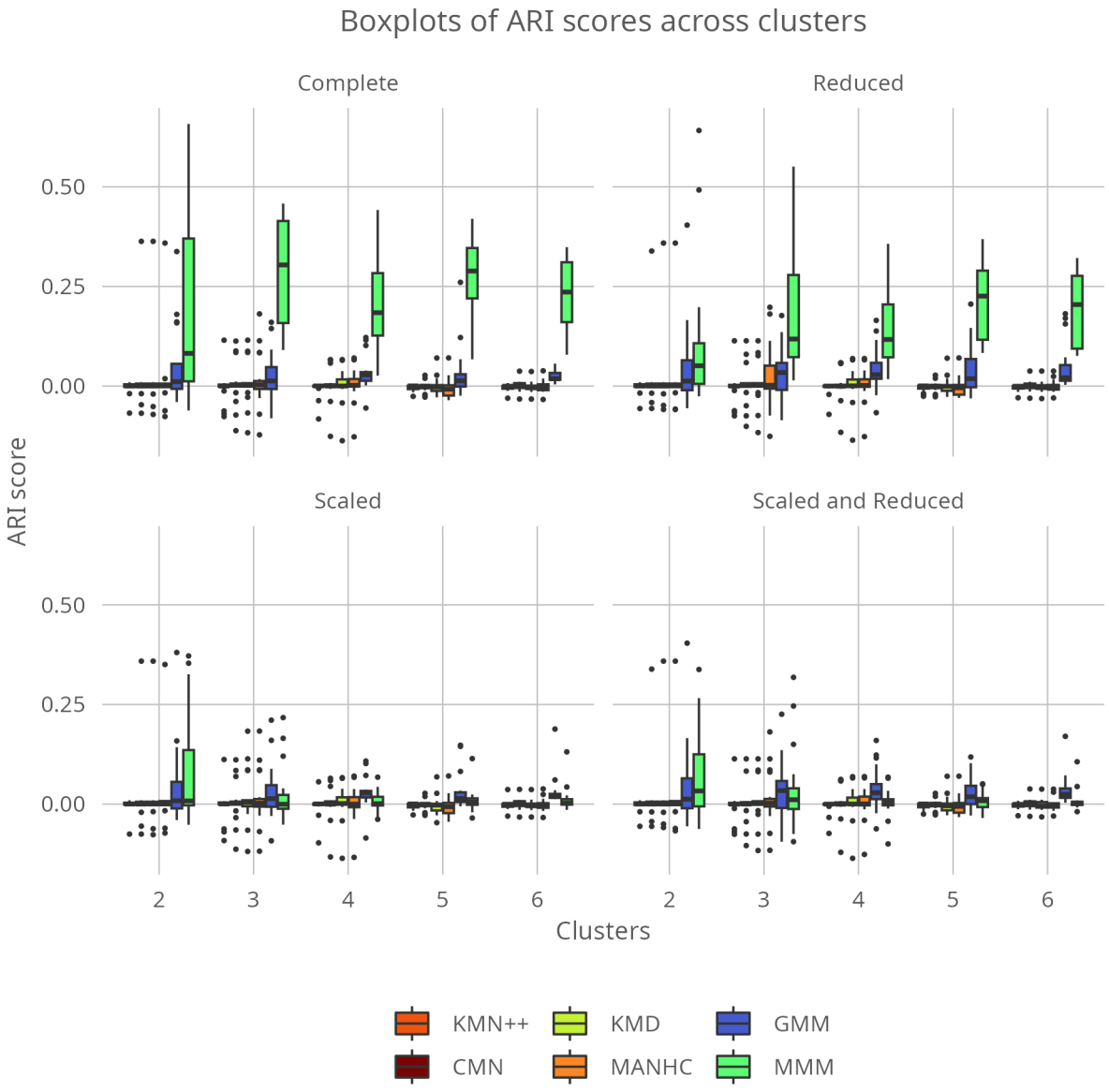


Figure 5.24: The ARI index in Codons

When the data was reduced, we noticed that GMM and MMM started from higher median scores than distance-based algorithms. Moreover, the tendency to decrease or increase the ARI index is similar to those in the complete data. However, here, overall scores were lower, as

well as the difference between the median of MMM and GMM. Additionally, GMM performed better when the data was reduced.

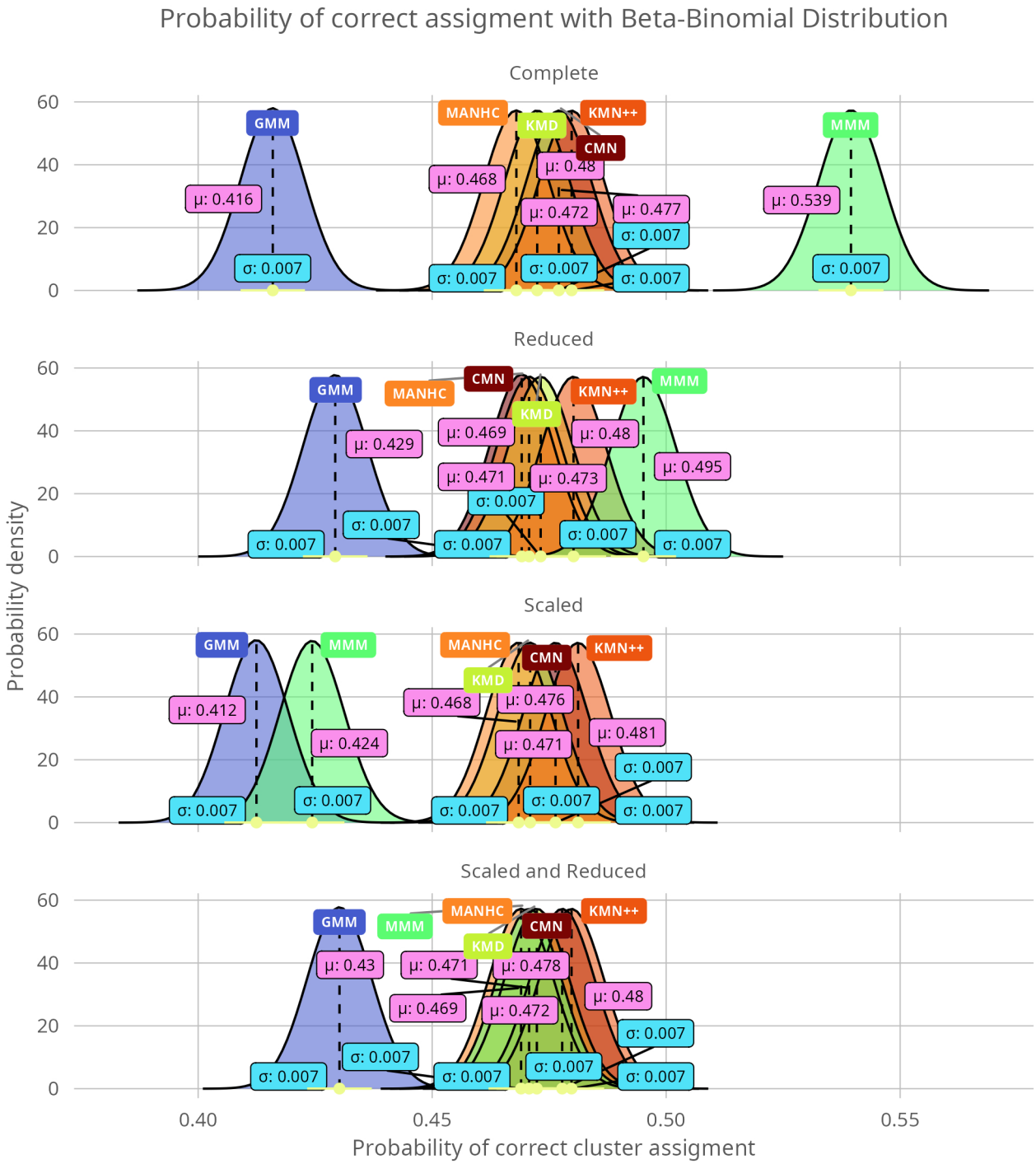


Figure 5.25: Beta-binomial distribution as a quality metric in Codons

Scaling improved the ARI index of all algorithms, but MMM. A considerable difference is visible starting from 2 groups. MANHC, GMM and MMM are almost at a similar level. Data reduction made the GMM score even higher, but its best median score was still relatively low, about 0.1 points.

In the case of codons, we can see a few negative scores. It means that compared partitions have little in common and are not helpful in clustering or pattern recognition. There are fewer such values in model-based than distance-based algorithms.

In Figure 5.25 of beta-binomial distribution, the most right peak belongs to MMM in the case of complete and reduced data. In the former case, it overlaps with the distributions of values from KMN++ and MANHC. After scaling, MMM was the last, which seems to be a repeating pattern. GMM was usually positioned in the last place, with the lowest mean, but when the data was scaled, it performed minimally better than in the original data. It also had a better score when the data was reduced. Finally, when the data was scaled and reduced, it performed better by 0.018 points. The difference between GMM, the last, and KMN++, which scored the first, was 0.024 points. All the algorithms' mean values had the least spread in this type of data. The probability density was similar for all algorithms and was relatively high, over 60.

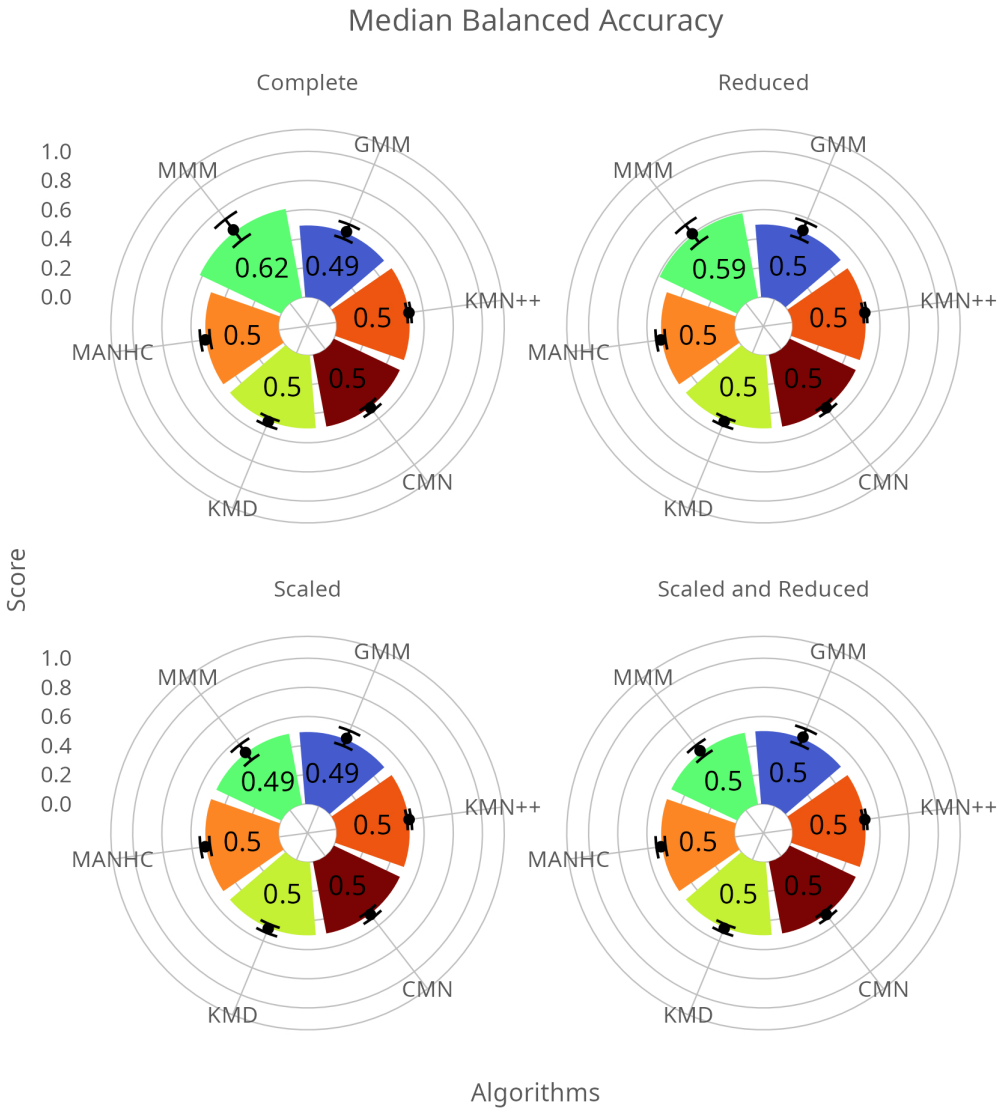


Figure 5.26: A median of balanced accuracy with mean and standard deviation in Codons

In Figure 5.26, the range of values of median balanced accuracy is relatively low because of about 0.11 points. The highest score belonged to the MMM when the data was complete. It was also highest in the metrics shown previously. The rest of the algorithms have very similar scores. It also applies to means and standard deviations. After reducing, the MMM still had the highest score but 0.03 points lower. On the other hand, GMM gained 0.01 points when the scores of the rest of the algorithms remained the same.

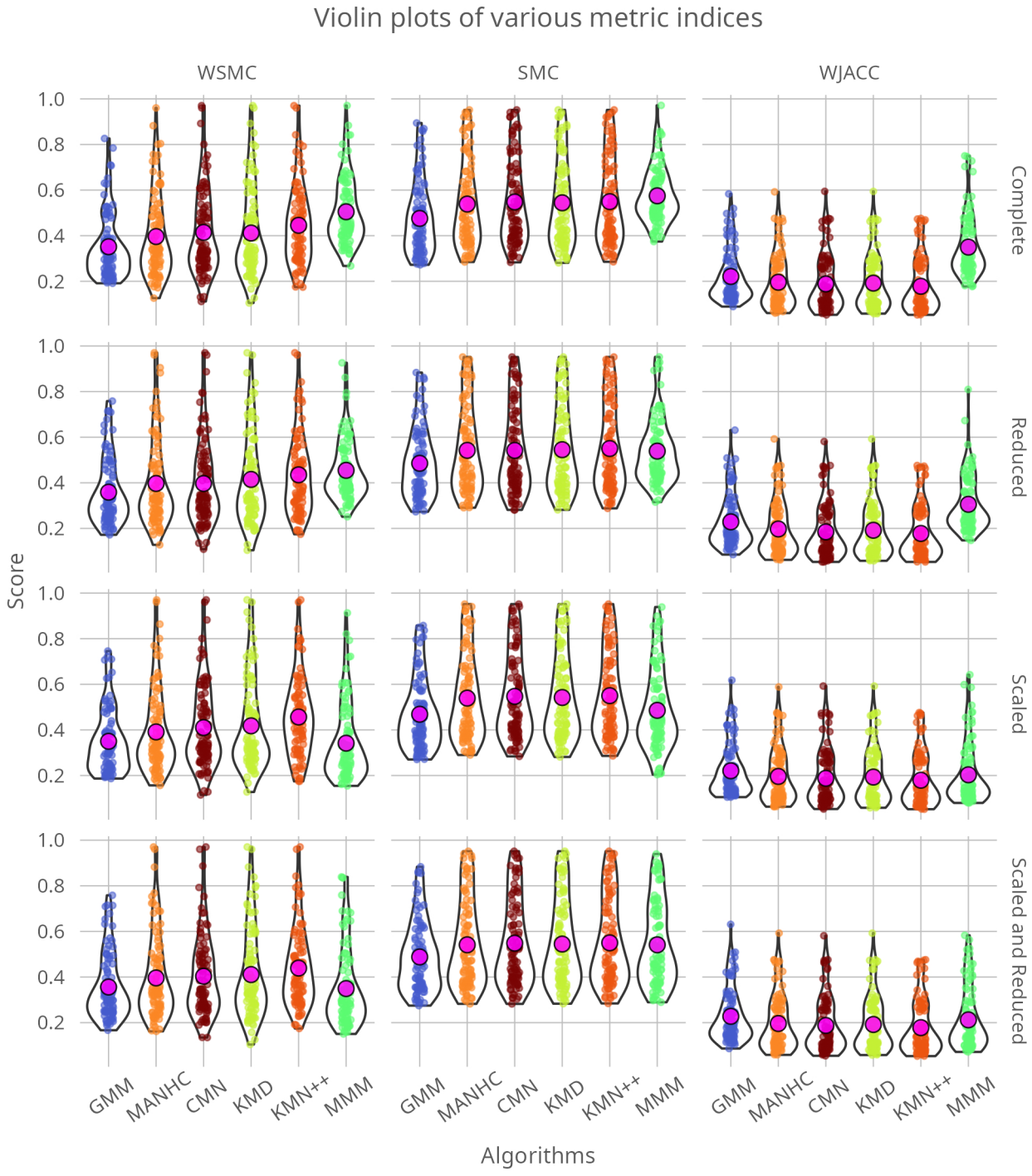


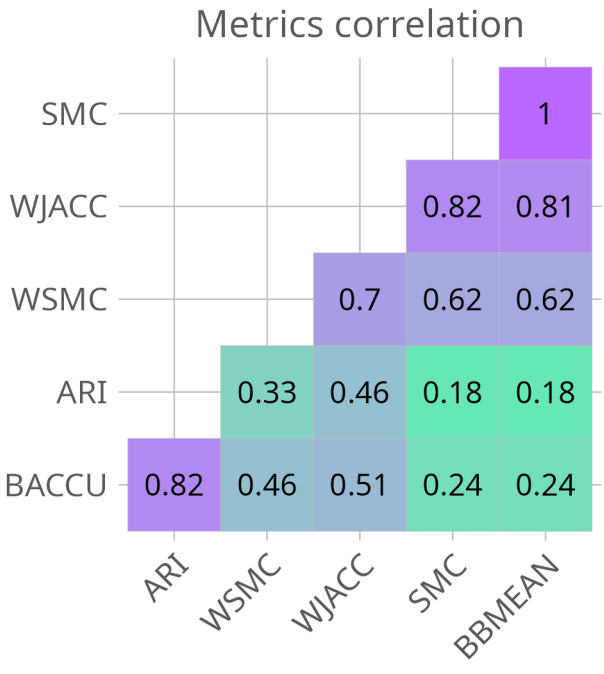
Figure 5.27: Other quality indices in Codons

Data scaling lowered the scores of MMM significantly but only slightly improved the results of the other algorithms. Apart from GMM, which gained 0.01 points more, the means and standard deviations of distance-based algorithms are also higher than in the case of previous data types. Finally, when data was scaled and reduced, MMM scored similarly to GMM because of 0.53. However, the lowest-performing algorithm in this data type, KMN++, scored 0.51, so there are no vast differences.

Figure 5.27 presents various quality metrics. In the complete data, MMM achieved the highest scores in all three metrics, with KMN++ in second place. The lowest scores belonged to GMM and CMN. When the data was reduced, all algorithm scores were slightly lower. However, the mean value of MMM was still the highest, at least in the WJACC and SMC metrics. For WSMC, the mean of KMN++ was slightly higher, but the density around the mean was greater for this algorithm than for MMM.

Scaling the data visibly reduced the MMM scores while elevating the distance-based algorithms' values. Further data reduction almost flattened the means of scores in all metrics except for WSMC. Here, the means of the distance-based algorithms were slightly higher than those of the model-based algorithms. A closer look shows that the model-based algorithms failed to achieve any score higher than 0.85, while the distance-based algorithms did.

The algorithms performed similarly, with no significant differences between their scores. However, MMM achieved the highest scores when the data was complete.



The correlation between metrics exhibits significant variation, with the ARI index displaying the weakest correlation to other metrics. The following weakest correlation is observed in the BACCU metric. However, a high level of correlation (0.76) is observed between the ARI index and BACCU.

From the perspective of ARI index, BACCU or SMC, or BBD, GMM and MMM had the highest scores. However, according to WSMC and WJACC metrics, MANHC and KMN++ performed better.

Figure 5.28: Metrics correlation in Codons

5.2.4 Sport activities

In recent years, there has been a rise in popularity for the terms 'smart' and particularly 'smartwatch', as evidenced by a Google trend. Smartwatches offer more than standard functions, such as phone calls, messaging, and playing music, with the ability to measure heart rate, calorie burn, and daily step count.

Many smartwatches can also gather statistics on sports activities, including distance and speed. Sophisticated smartwatches can automatically detect and record a person's actions and corresponding metrics.

Data shared by Yale professor Billur Barshan on the UCI platform was used for this comparison. Barshan specializes in wearables, machine/deep learning, and robotics. The gathered data included information about various sports activities performed by eight individuals between the ages of 20 and 30, including four males and four females.

Without prior instruction, participants were asked to perform 19 activities, such as jumping, cycling, running, or walking, which may have introduced individual variation among subjects. Various sensors were used to record the data, with a single unit consisting of 9 accelerometers, gyroscopes, and magnetometers in x, y, and z coordinates. In total, five units were placed on the torso, both arms, and legs. The sensors were calibrated to gather data with a sampling frequency of 25 Hz. Each activity lasted 5 minutes, resulting in 45 attributes and 1,140,000 observations.

Looking at the dimensionality reduction plots if Figure 5.29, we see that although sports activities are incredibly mixed, their distribution has some visible pattern. The immense cloud of points in the middle might be the initial position that has been recorded. A few activities like running, walking, and climbing stairs share the same initial position - standing still. However, our study does not filter any potential noise and uses all the available data.

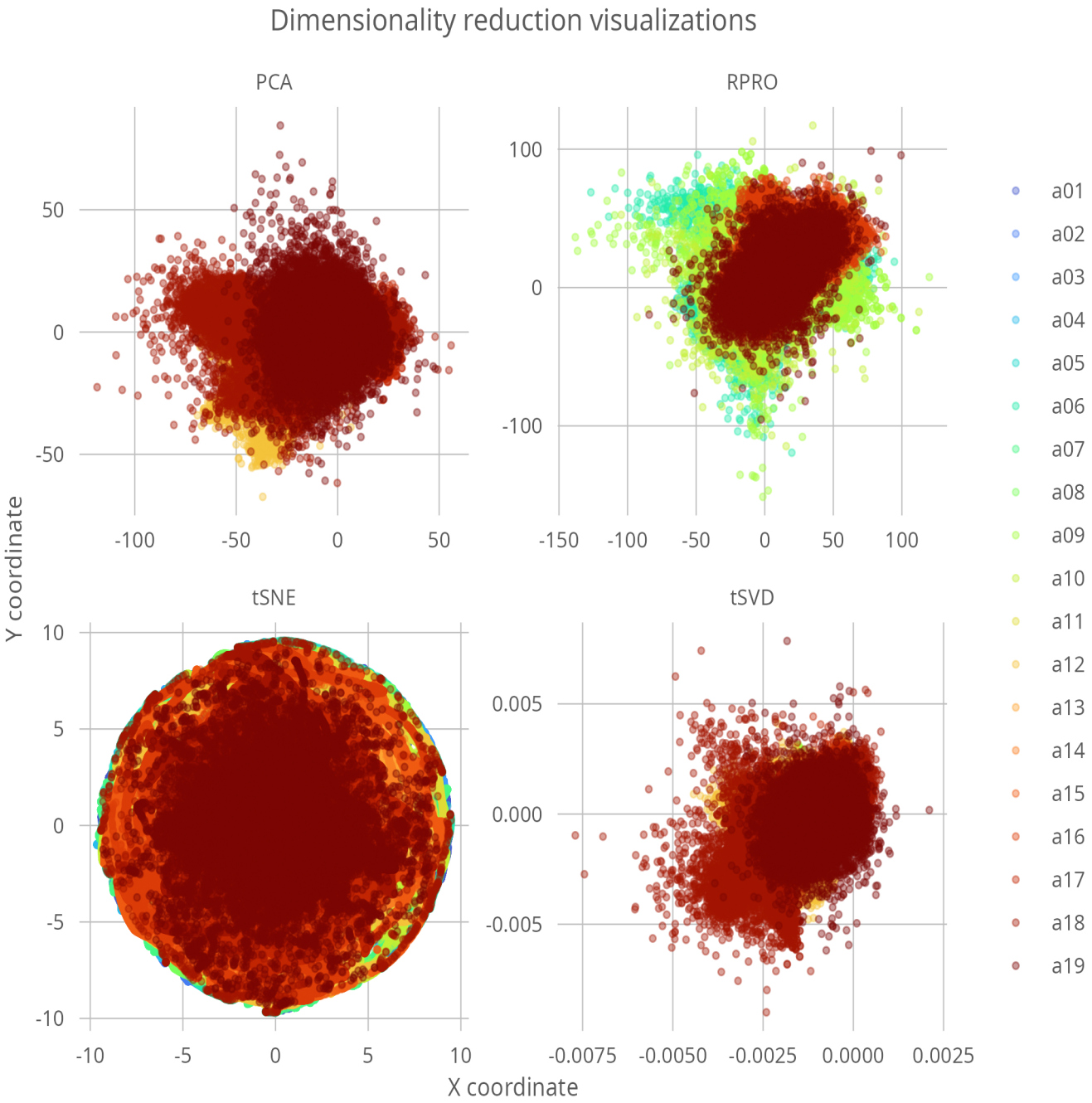


Figure 5.29: Visualization of various dimensionality reduction techniques in Sport Activities

Results

In the case of daily sports activities, including Hierarchical Clustering and k-medoids methods was not possible. This was because even two mixture components comprised over 100 thousand observations, resulting in a minimal RAM allocation of around 50 GB. This increased to as much as 1000 GB with up to 6 components. Although some algorithms can overcome this limitation of RAM, their completion time was significantly higher than any other presented algorithms.

From the perspective of the ARI index in Figure 5.30, two-component mixtures have the broadest spectrum of outcomes across all datasets. It applies to all of the algorithms, but in the case of MMMK, those spectrums are the lowest, similarly to its results.

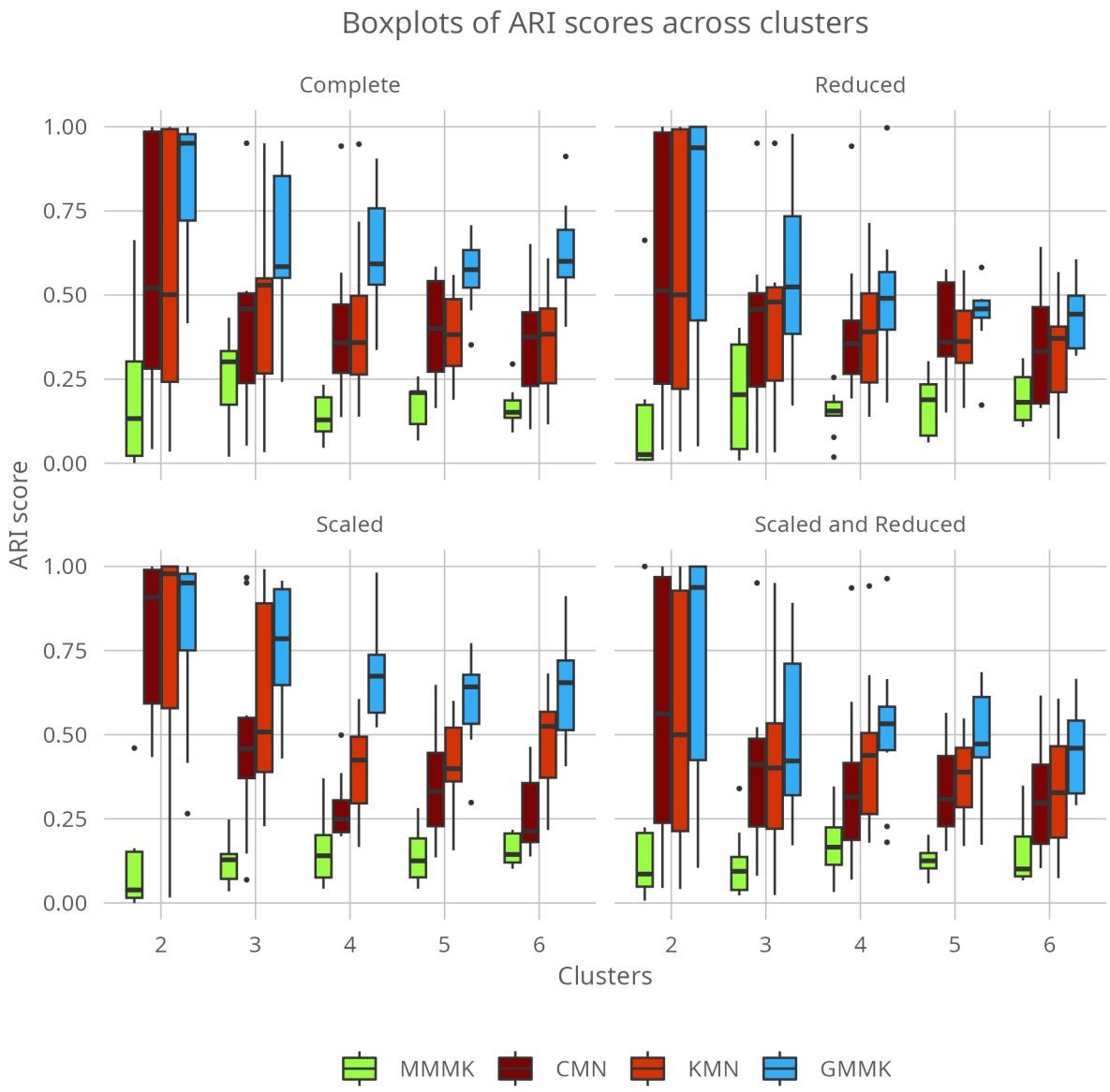


Figure 5.30: The ARI index in Sport Activities

We can see that the GMMK has the best scores of all the algorithms. Notably, even with

the increased number of clusters, it maintained a higher ratio of correct assignments than the others. Scaling benefits all of the algorithms. With the increased number of clusters, results became denser around the median. Contrary to MMM, the range of results became more scattered and chaotic. In the case of variable reduction, algorithms performed worse, as if some information was lost in the process. Once again, scaling positively impacted the reduced data, but results remained relatively low.

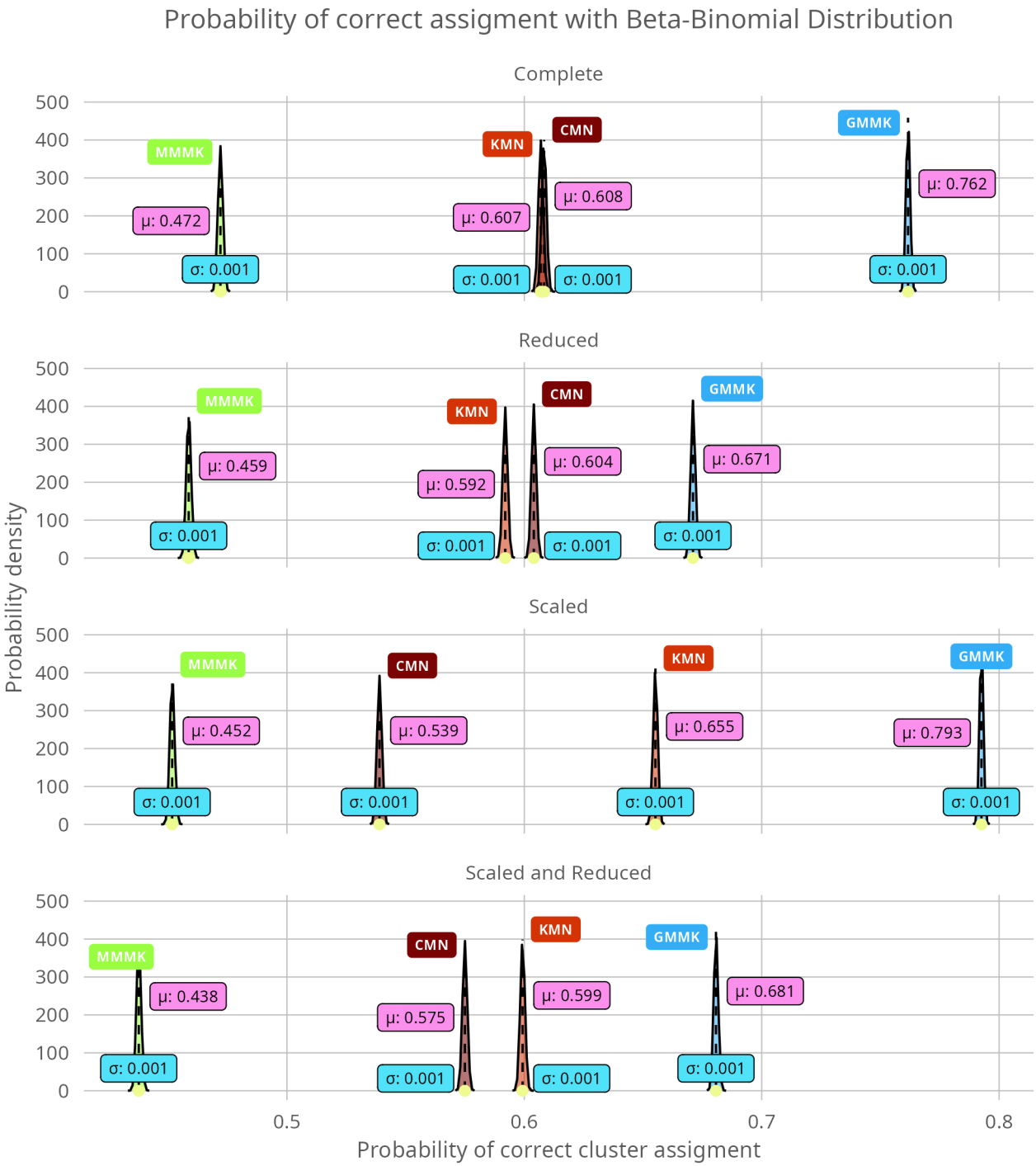


Figure 5.31: Beta-binomial conjugate distribution as a quality metric in Sport Activities

In the Figure 5.31, the highest and most right-side peak belongs to GMMK, with a mean value of 0.76. After data reduction, it was impacted by lowering its performance by almost 0.1, while the other algorithms were affected only slightly. When the complete data was scaled, the result was even higher than when the data was unchanged, probably taking more advantage of kmeans initialization.

Finally, the scaled and reduced data scores of GMM and KMN++ were shifted to the left side, but at the same time, those of MMM and CMN were brought more to the right side. When it was still last, it was the best performance of MMM across different data types.

Regarding probability density, we can see that this of GMMK was higher than in the case of their algorithms. It gives us slightly more credibility of the result. On the opposite, the MMMK algorithm scored the lowest probability density.

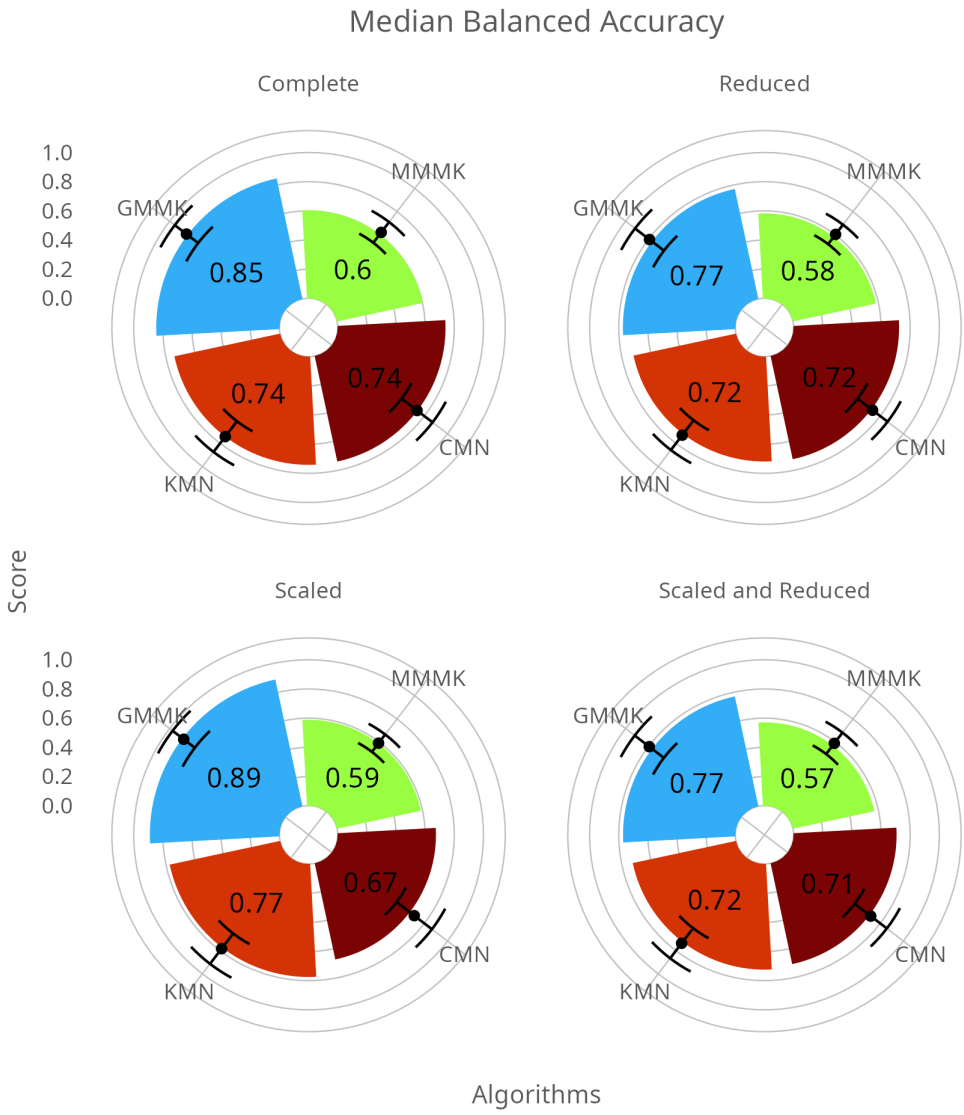


Figure 5.32: A median of balanced accuracy with mean and standard deviation in Sport Activities

Median balanced accuracy (Figure 5.32) also shows that GMMK performed the best across

all the compared algorithms. Scaling improved its scores by 0.04 points. CMN performed minimally better than KMN in complete, reduced, scaled, and reduced data. When the data was scaled, KMN++ had a better score of almost 0.08 than CMN, with a score of 0.75. It was the highest result, excluding GMMK. On the contrary, MMM had the lowest scores across all data sets. However, scaling and reducing the data improved its results slightly.

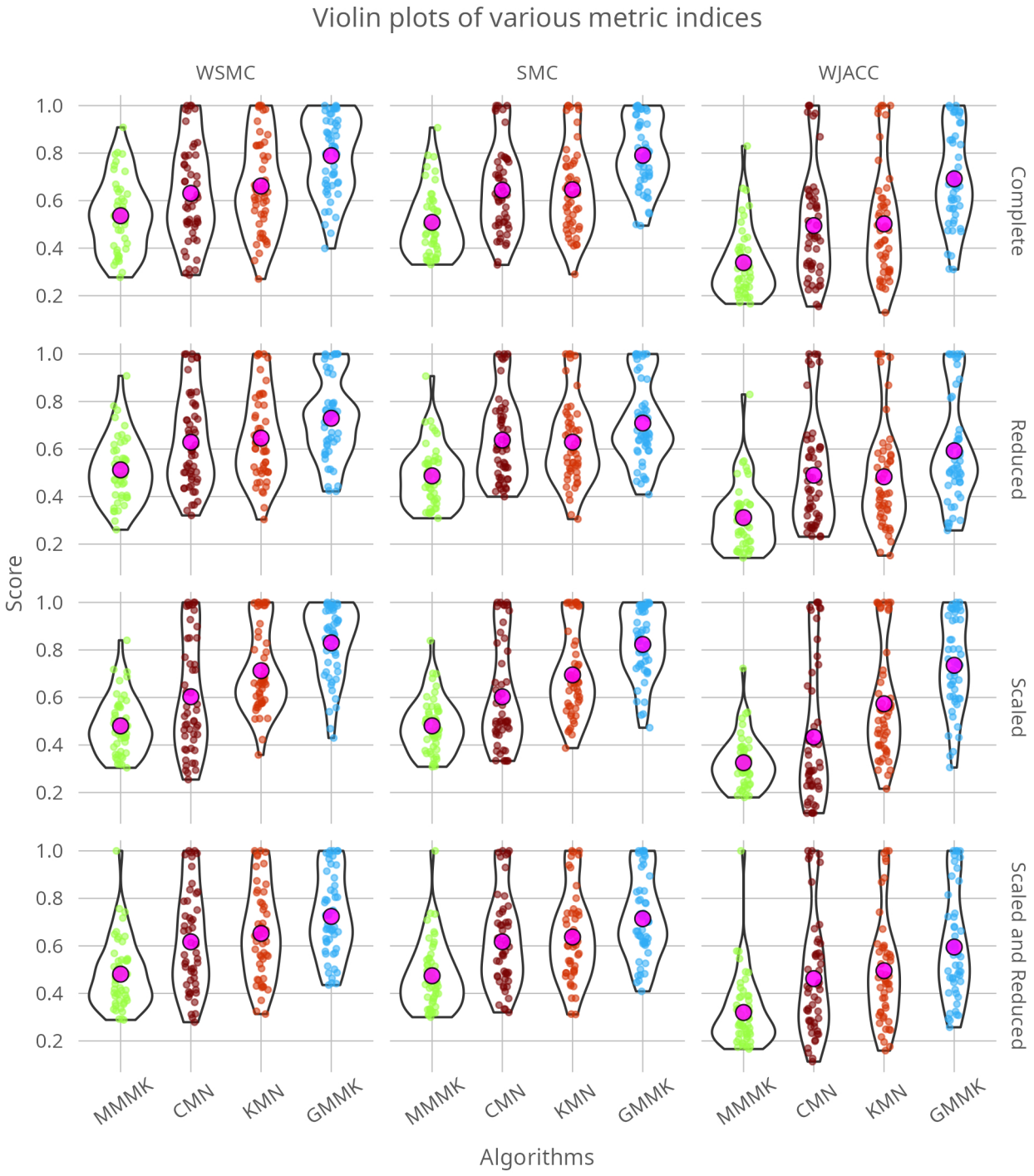


Figure 5.33: Other quality indices in Sport Activities

In Figure 5.33, all metrics agree that GMMK based on k-mean initialization had the highest

scores, despite various data types. Contrary, MMM scored the lowest. When the data was reduced, we can see that its density is located near the higher values than in the case of complete data. Scaling the data helped GMMK obtain even better results, which might be seen by comparing the densities of values. CMN and KMN++ were similarly distributed, but it can be noted that when data was reduced, CMN had small values than KMN++. The opposite was true when the data was scaled.

Figure 5.34 shows that the metrics are highly correlated, which shows vast agreement between them. So far, these Sports Activities' results were the most straightforward ones. We can see that GMMK outperformed other algorithms, even in different data types. It is also notable that it is almost impossible to calculate the distances as a distance matrix when dealing with such a massive number of observations. Thus we cannot use the HC algorithm and KMD in the standard form. We must resort to methods that calculate the distance on the fly, which vastly increases execution time, sometimes to the point of impossibility.

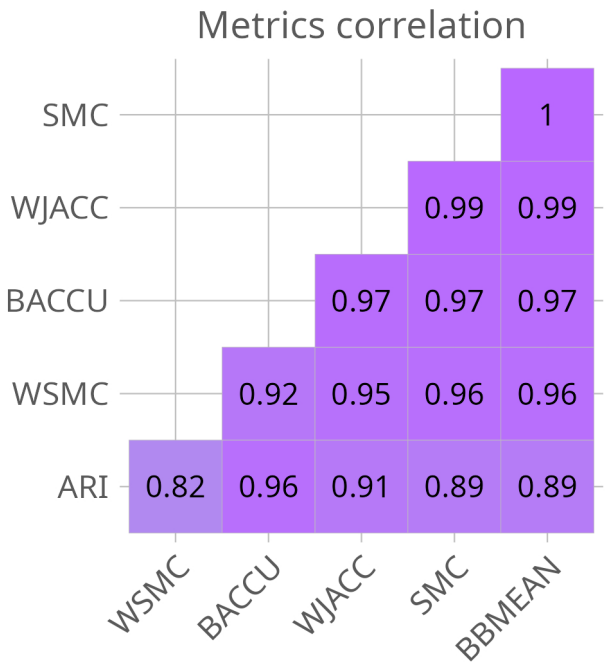


Figure 5.34: Metrics correlation in Sport Activities

5.2.5 The Free Music Archive

The Free Music Archive is a large music analysis project, that was started in 2009 by the East Orange, New Jersey community radio station WFMU, cooperating with KBOO and KEXP. It aimed to publish music under Creative Commons licenses, permitting unrestricted downloading and usage of the music in various other works.

The data was made publicly available around the end of 2016. Its vast library consists of 106 577 songs covered by 16,341 artists. All of that is inside 14,854 albums. The FMA delivers pre-computed audio features jointly with user-level and track-level metadata. It is not only a valuable resource for music researchers and enthusiasts but also for musicians themselves. By providing a platform for free and legal music downloads, the FMA supports independent artists and promotes creative expression.

Moreover, it is also possible to use full-length, high-quality audio—a complete archive weight of around 879 GiB. However, it comes with different packages, limited to selected genres. Over 500 features describe each song. They were pre-computed with a package librosa, a rich python audio and music analysis library. The package allows for low-level feature extraction, such as chromagrams, Mel spectrogram, Mel Frequency Cepstral Coefficient (MFCC), and other spectral and rhythmic qualities. The library on the site https://librosa.org offers more information to create such an analysis. It includes tutorials and documentation [38].

The tracks in the archive are organised into a hierarchical taxonomy of 161 genres, such as rock, jazz, pop, or classical music. However, the data was significantly reduced for the analysis for several reasons. First, roughly half of the songs were assigned multiple genres, and there was no information about the leading or most-voted genre. As a result, some songs had numerous labels, and these were excluded from further analysis to simplify actual labelling values during clustering. Unfortunately, as of late 2018 the project ended.

Moreover, the research data was reduced to the twelve most frequent genres mentioned in the table, and the cutoff value was arbitrary. The last genre in the data, "Spoken", had 423 observations, while the rest that was not included were below 194. This way, the groups were more balanced.

The final dataset consisted of 49598 observations with 518 features. From this point, the procedure followed the general pipeline.

Figure 5.35 presents various dimensionality reduction visualizations. However, it is challenging to point out any specific groups.

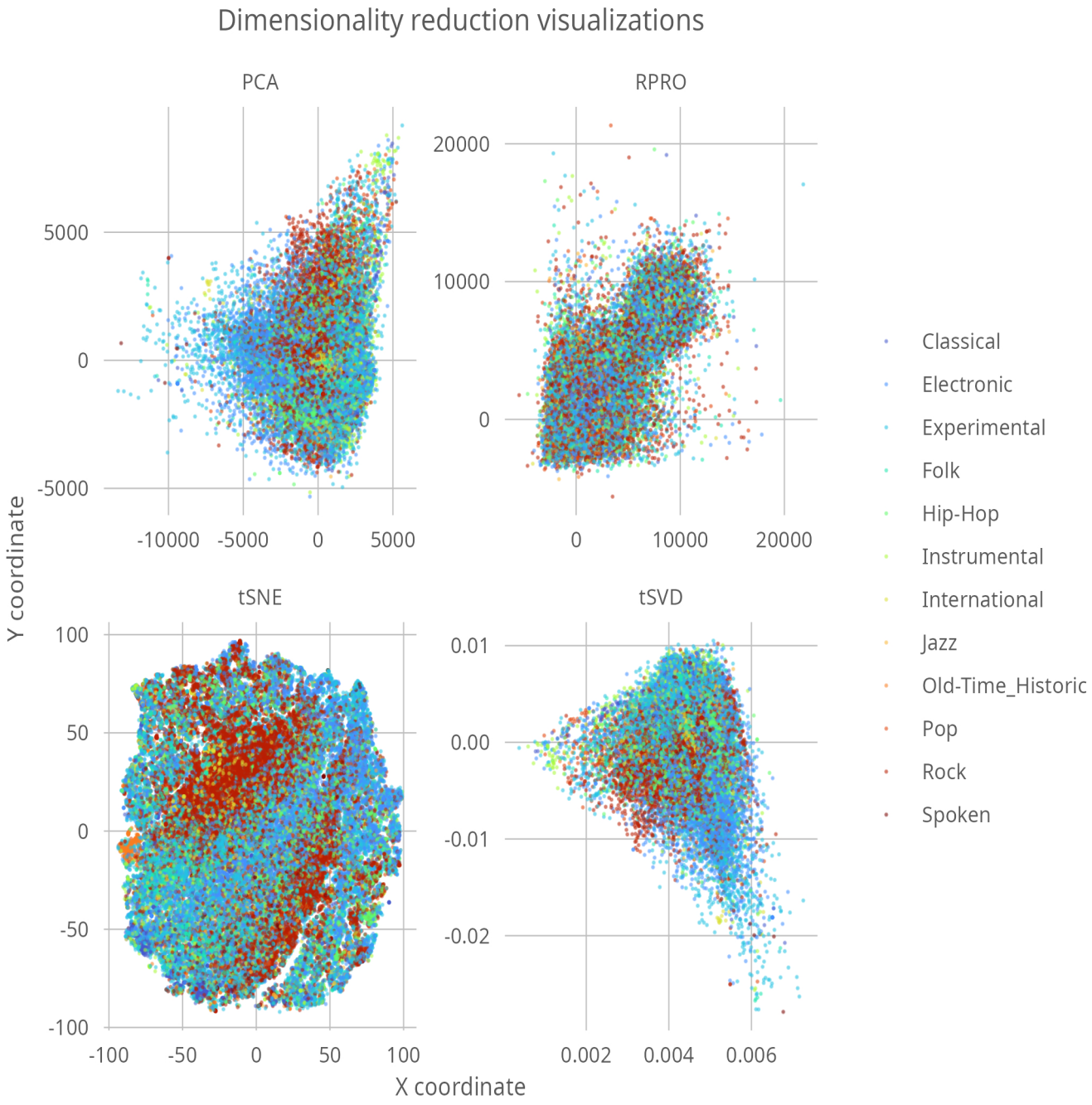


Figure 5.35: Visualization of various dimensionality reduction techniques in The Free Music Archive

Results

In Figure 5.36, the ARI index was low for all of the algorithms, as they barely excceed 0.2 score. The only case when the median value was above that threshold was in the case of HC and KMN++, when the data was scaled and the mixture had three components. MMM performed better when the data was complete, whether GMM when the data was reduced.

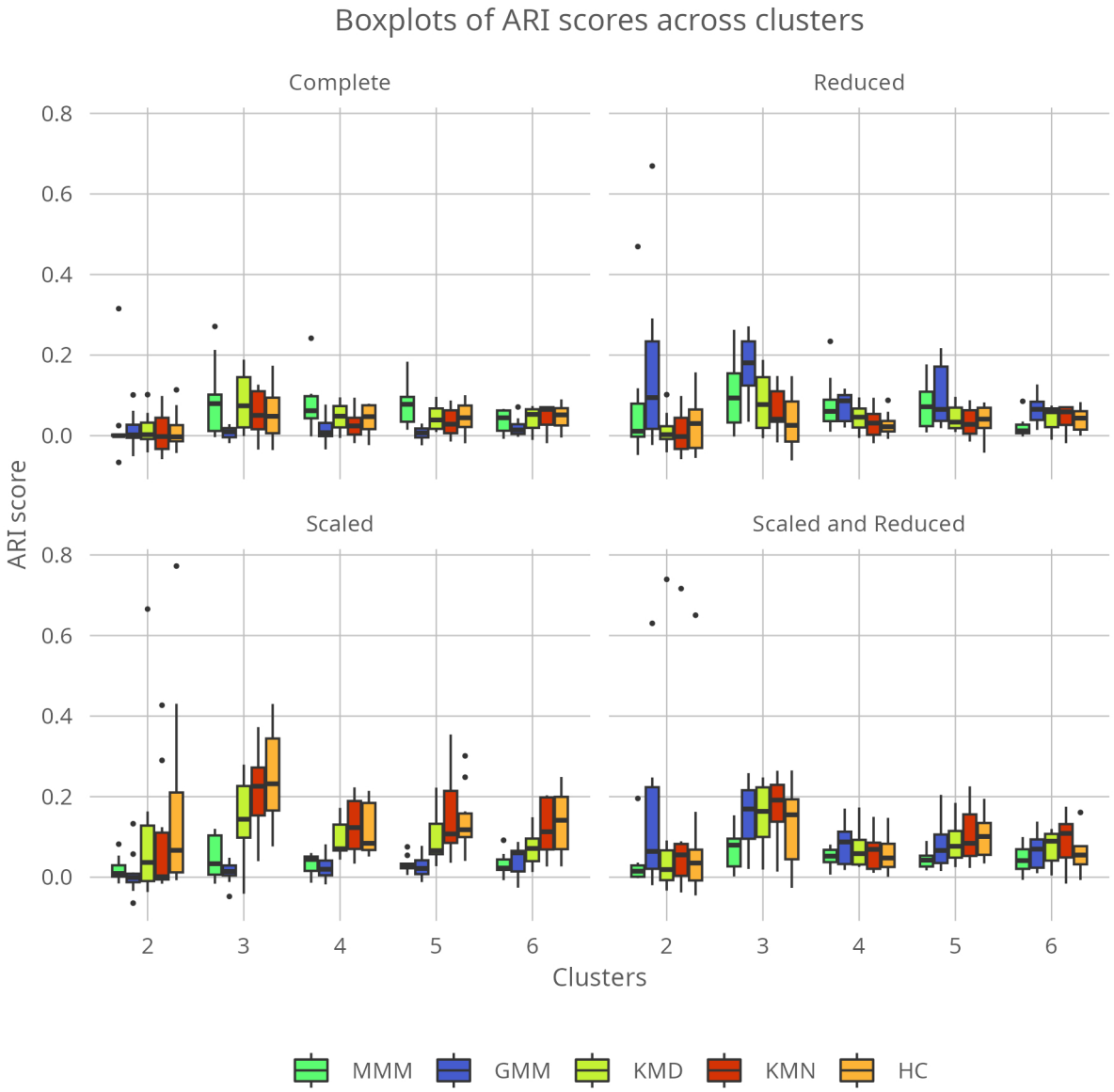


Figure 5.36: The ARI index in The Free Music Archive

Beta-binomial distribution in Figure 5.37 shows that GMM scored about 0.57 with the highest mean across all algorithms when the data was complete. The second one was MMM, lower by about 0.03 points. The distance-based algorithms were placed closely together in the 0.4 to 0.42 points. After data reduction, the differences between scores went down. Although both GMM and MMM preserved their highest position, they switched places, leaving MMM as a better-performing algorithm by 0.04. points.

Scaling improved the performance of distance-based algorithms, but also of GMM, compared to the previous dataset. It impacted MMM negatively, as it took last place, having 0.35 points, the lowest score in those datasets. When the data were reduced, it brought up MANCMN at the front with a score of 0.474. Moreover, it shifted all other algorithms to the left, but MMM improved the score by 0.02. points.

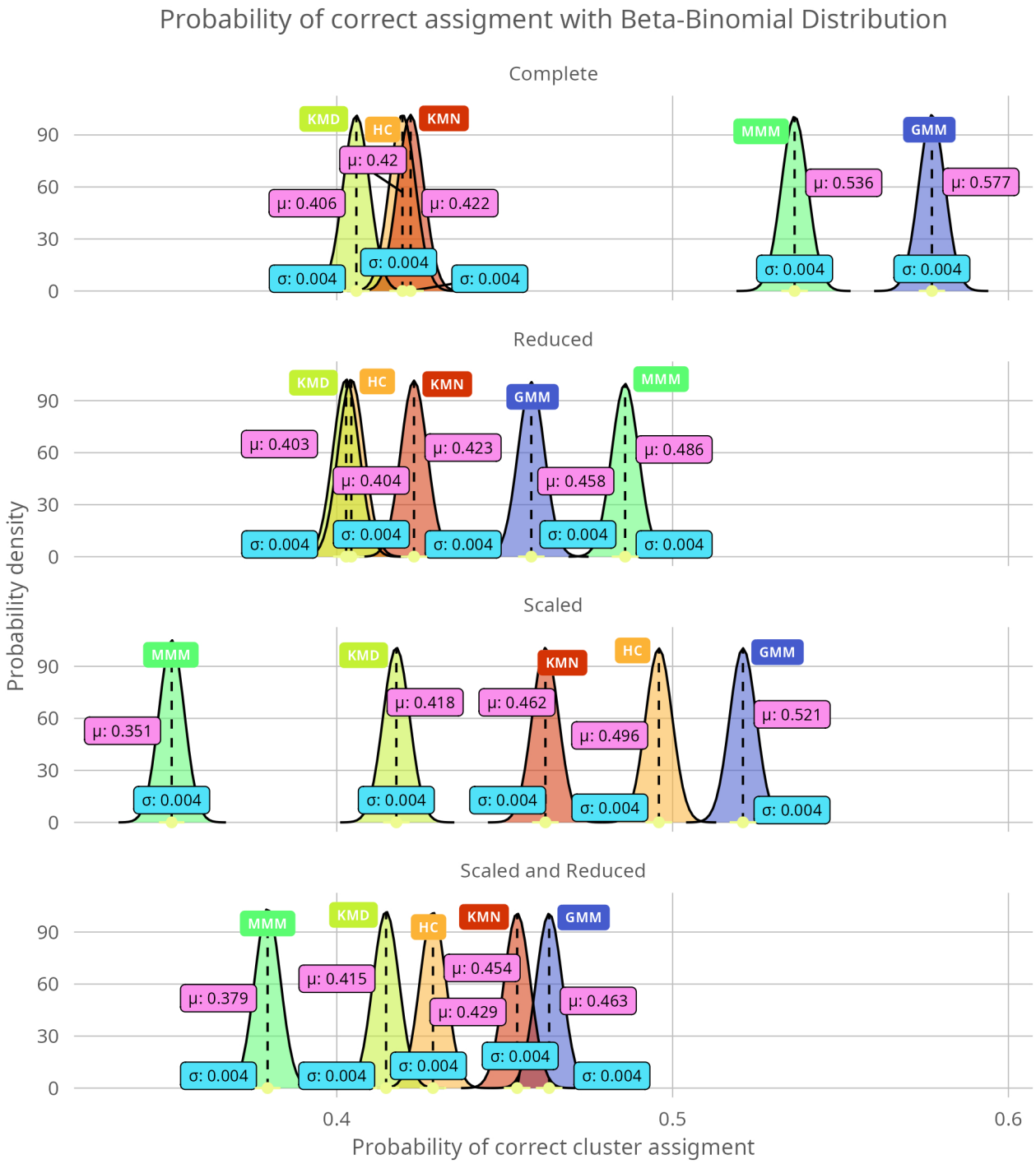


Figure 5.37: Beta-binomial distribution as a quality metric in The Free Music Archive

Median scores of balanced accuracy scores in Figure 5.38 are in the same place as the mean

value. Results from all algorithms are between 0.49 to 0.6. However, the highest score belongs to the MMMK variant regarding not filtered data. Data reduction again positively impacted the model-based algorithms but did not significantly worsen the distance-based algorithms' results. Scaling of the data was advantageous for distance-based algorithms in both cases where data was not filtered and reduced. However, the best scores were obtained by k-means and MANHC when the data was complete and scaled.

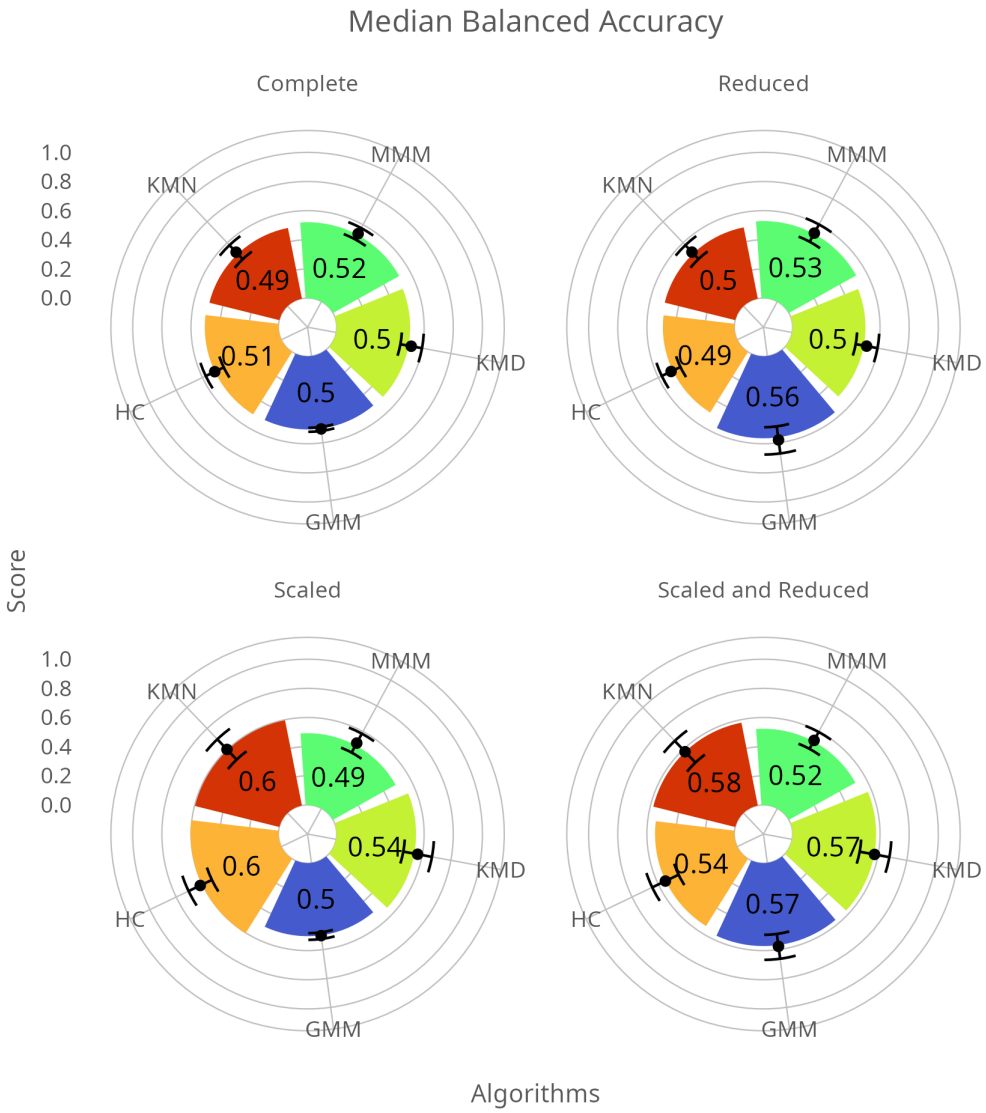


Figure 5.38: A median of balanced accuracy with mean and standard deviation in The Free Music Archive

In the figure that shows other quality metrics, we can see that algorithms show a similar pattern of scores. SMC scores were the highest, then WSMC and the case of WJACC scores were the lowest. In addition, WSMC and WJACC are more consistent together than with SMC. Comparing the complete data, MMMK scored the highest, apart from SMC, which showed that GMMK performed slightly better. When the data was reduced, all metrics had much more agreement. Model-based algorithms scored higher results, with MMMK being

first than distance-based algorithms. Those differences are much more visible in the SMC metric.

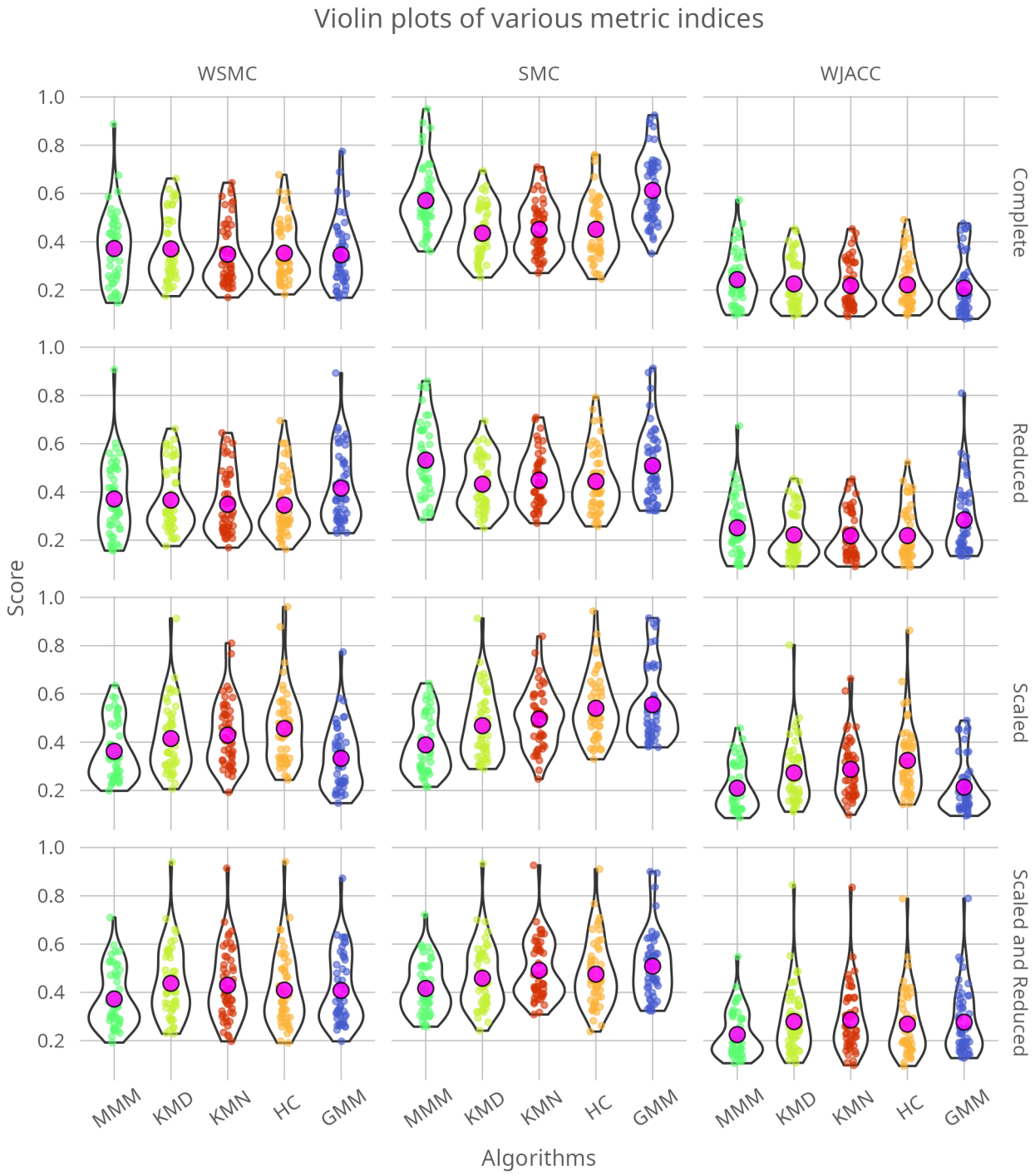


Figure 5.39: Other quality indices in The Free Music Archive

Distance-based metrics performed better when the data was scaled than model-based ones, at least according to WSMC and WJACC. In the case of SMC, GMMK had the highest mean score across all algorithms.

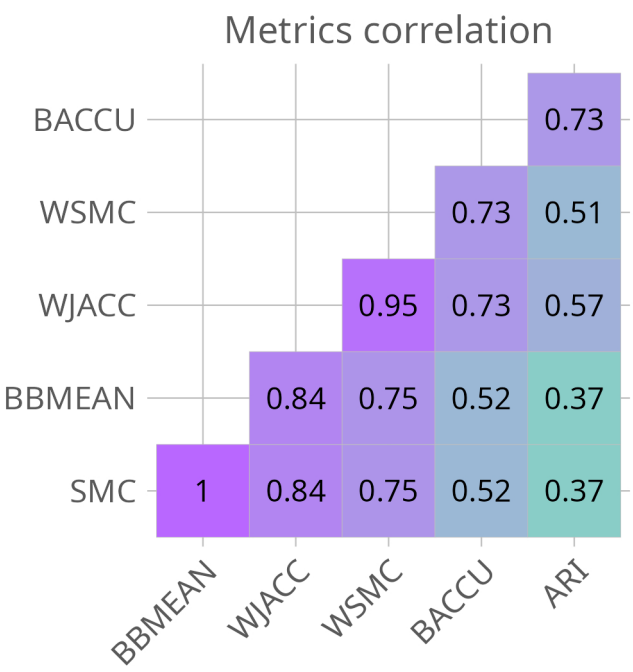


Figure 5.40: Metrics correlation in The Free Music Archive

The presented metrics are slightly correlated. The lowest correlation is observed between the ARI index and SMC. Apart from SMC and Beta mean, the highest correlation between WSMC and Jaccard index is observed. It indicates that there is no firm agreement between different metrics.

The results obtained from the Free Music Archive are intriguing. From the point of correct assignment to groups, they are challenging to interpret. In this perspective, we try to answer the question: "Which algorithm performed the best clustering according to subjected opinion of people?"

5.2.6 Arrhythmia

Atrial fibrillation is a complex and multifaceted condition that poses significant challenges for diagnosis and treatment. Although several risk factors for atrial fibrillation have been identified, including age, hypertension, obesity, and diabetes, the underlying mechanisms of the disease are not fully understood. Recent research suggests that atrial fibrillation may involve multiple factors, such as inflammation, oxidative stress, and autonomic dysfunction, which interact in complex ways. Furthermore, managing atrial fibrillation requires a comprehensive and individualized approach that considers the patient's medical history, symptoms, and comorbidities.

Patients with atrial fibrillation have five times more increased risk of stroke. At the same time, atrial fibrillation causes almost 20% to 30% of strokes. In addition, strokes caused by atrial fibrillation are much more severe and fatal. They led to death much more often than strokes due to other causes.

According to a studies, in 2016, almost 7.6 million people in the European Union had atrial fibrillation. Studies estimate that this number will increase by 89% to 14.4 million by 2060. The current prevalence will rise by 22%, from 7.8% to 9.5%. Last but not least, yearly treatment consumes from 0.28% to 2.6% of European funds spent on healthcare. [39]

The data we will explore in our thesis comes from the study whose original purpose was to differentiate between the presence and absence of cardiac arrhythmia. After that, observations were organized into one of the sixteen groups. The first class, 01, refers to "normal" ECG classes. Then, the number from 02 to 15 refers to various arrhythmia categories. Finally, category 16 refers to the rest of the unclassified ones. For the time being, a computer program exists that classifies the data. However, there are differences between the cardiologist and the grouping done by the program.

Visualizations created by dimensionality reduction techniques in Figure 5.41 share some similarities. The points are scattered. However, after closer examination, they are making many small groups. It might be caused by the low number of observations in the data or the high similarity within groups.

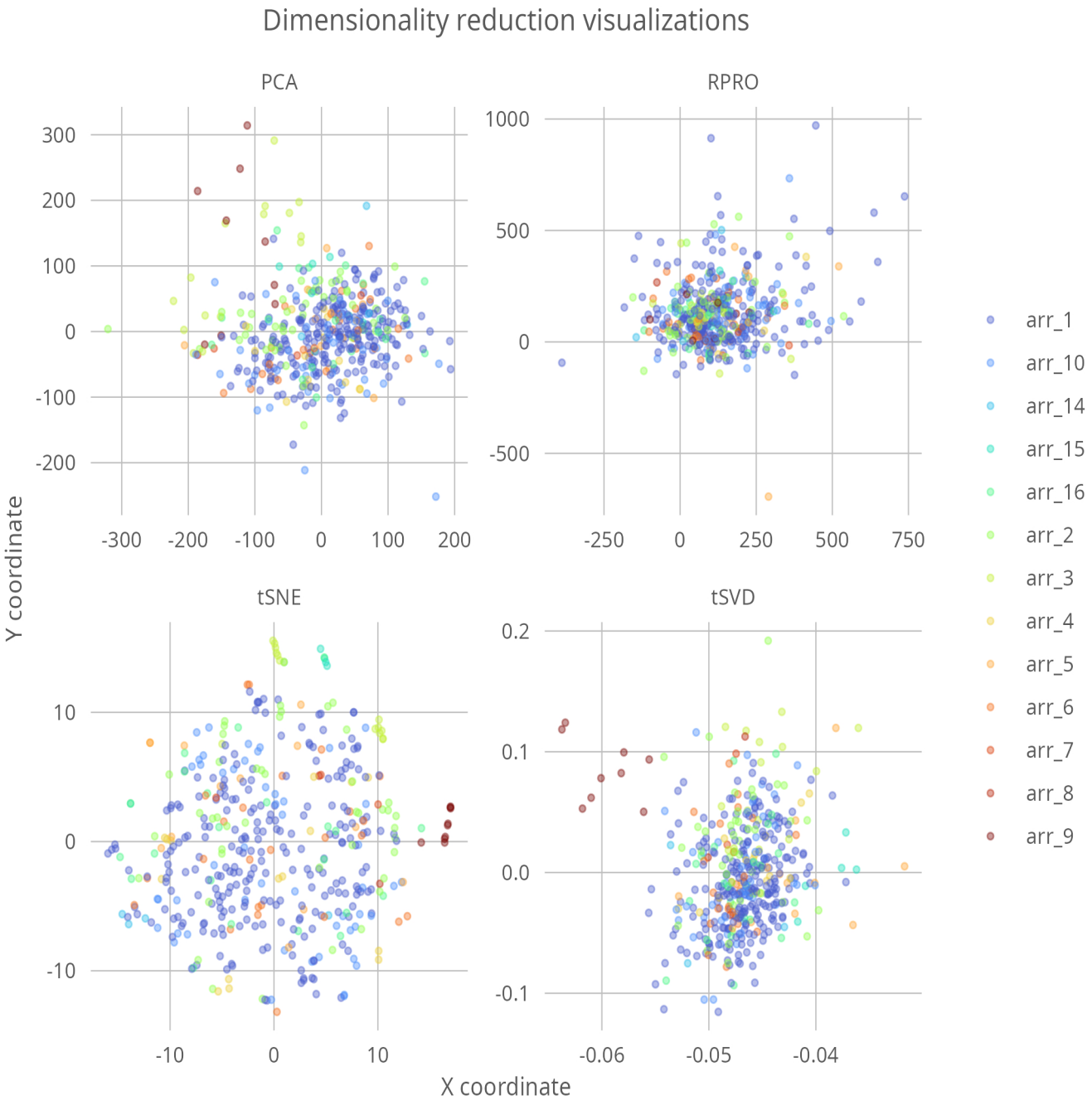


Figure 5.41: Visualization of various dimensionality reduction techniques in Arrhythmia

Results

Figure 5.42 of the ARI index in Arrhythmia shows MMM in the last place. In the first part of the chart, we can see that its spectrum of values is narrower than in the case of other algorithms, with a median value of about 0.3. At the same time, GMM and MANHC medians were lower, close to zero. However, their spectrum of scores was high enough to reach even the highest scores. Other distance-based algorithms also obtained almost 1 in a few cases.

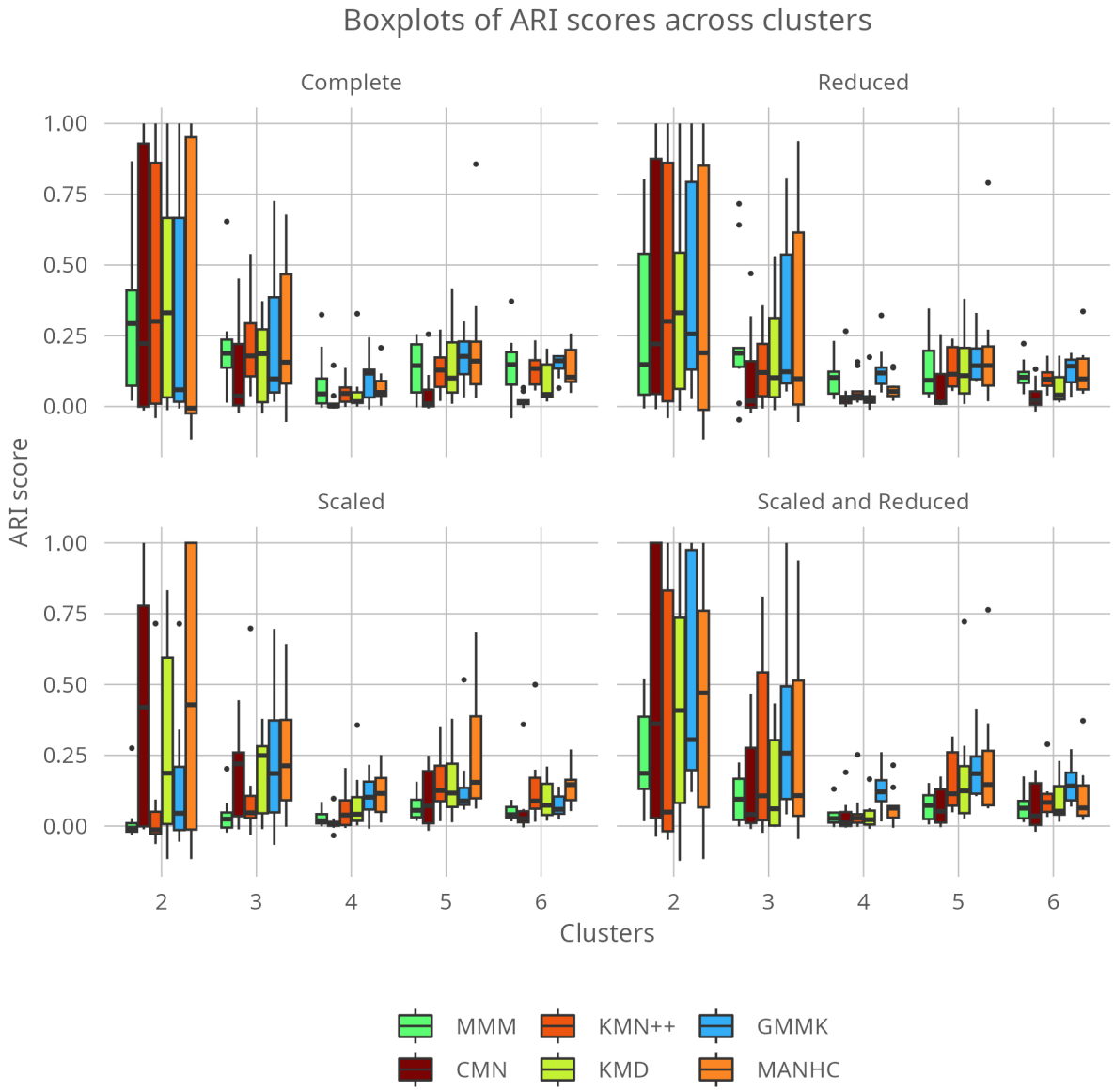


Figure 5.42: The ARI index in Arrhythmia

Data reduction generally had a negative impact, but there are few exemptions. The third quantile of the boxplot of MMM was higher than when the data was complete. Also, the median value of GMM in the two components mixture increased. When more groups were present, the differences were not that visible.

Scaling of the data improved the results for MANHC and CMN but worsened the scores

of the others. GMM obtained the highest results from all other algorithms when the data was reduced and scaled. As for the negative ARI index, most values came from the MANHC, KMD, KMN++ and MMM.

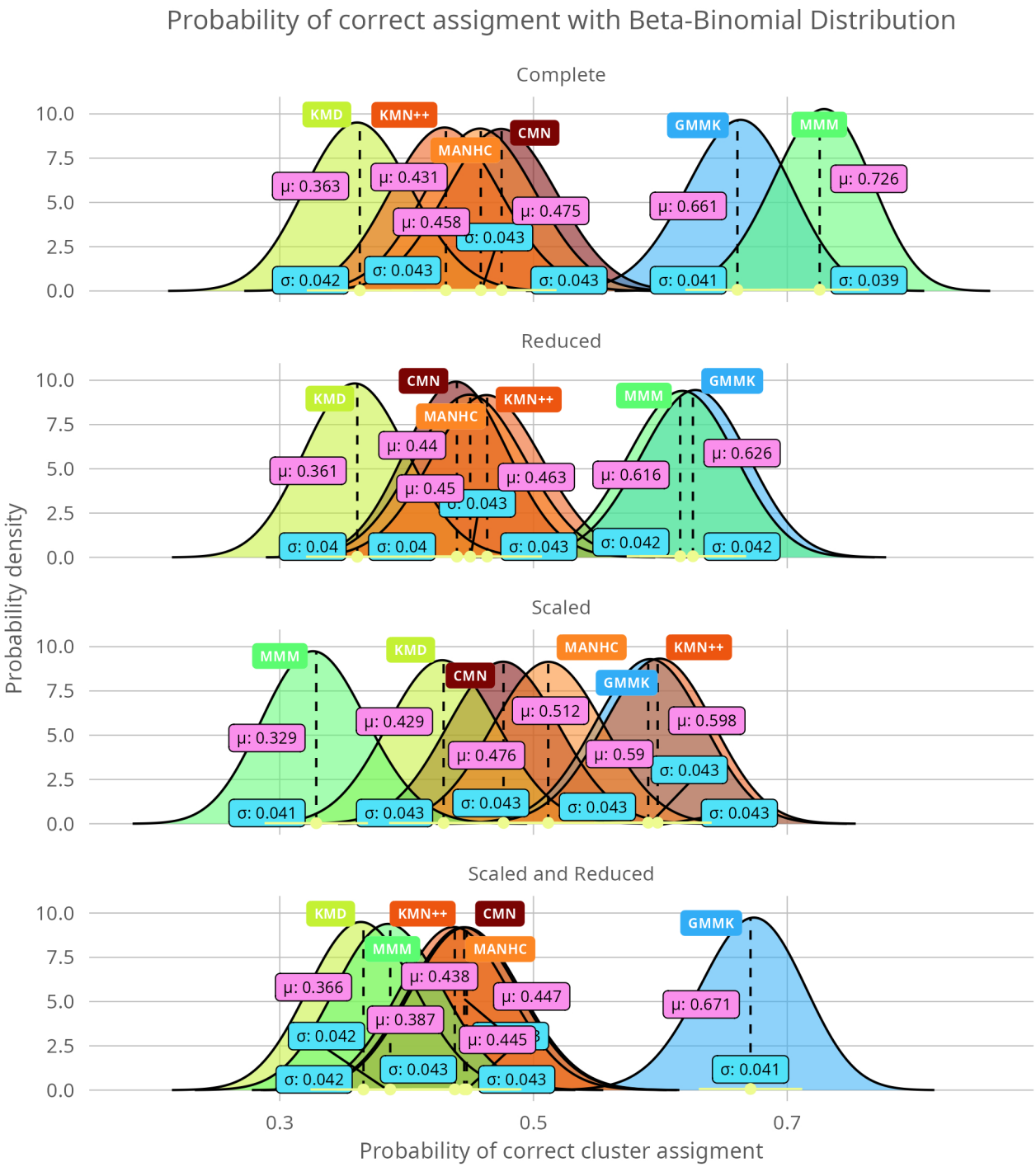


Figure 5.43: Beta-binomial distribution as a quality metric in Arrhythmia

In the case of beta-binomial distribution in Figure 5.43, MMM had the highest score across all algorithms. It means the correct assignment had 0.73 and the lowest standard deviation, nevertheless, only in the case of complete data. When the data was reduced, MMM and

GMMK were at a similar level with almost 0.01 mean difference, having the highest scores. After the complete data was scaled, GMMK had lower scores, had scored lower by nearly 0.04, and MMM was the last. Those two algorithms are highly overlapped as the difference in correct assignment between them is small. When the data were additionally scaled, GMMK performed better than other algorithms, for which data reduction worsened the results. Still, data reduction seems to benefit also MMM, which now was second last. Overall the worst and the best scores, 0.329 and 0.726, respectively, belong to MMM.

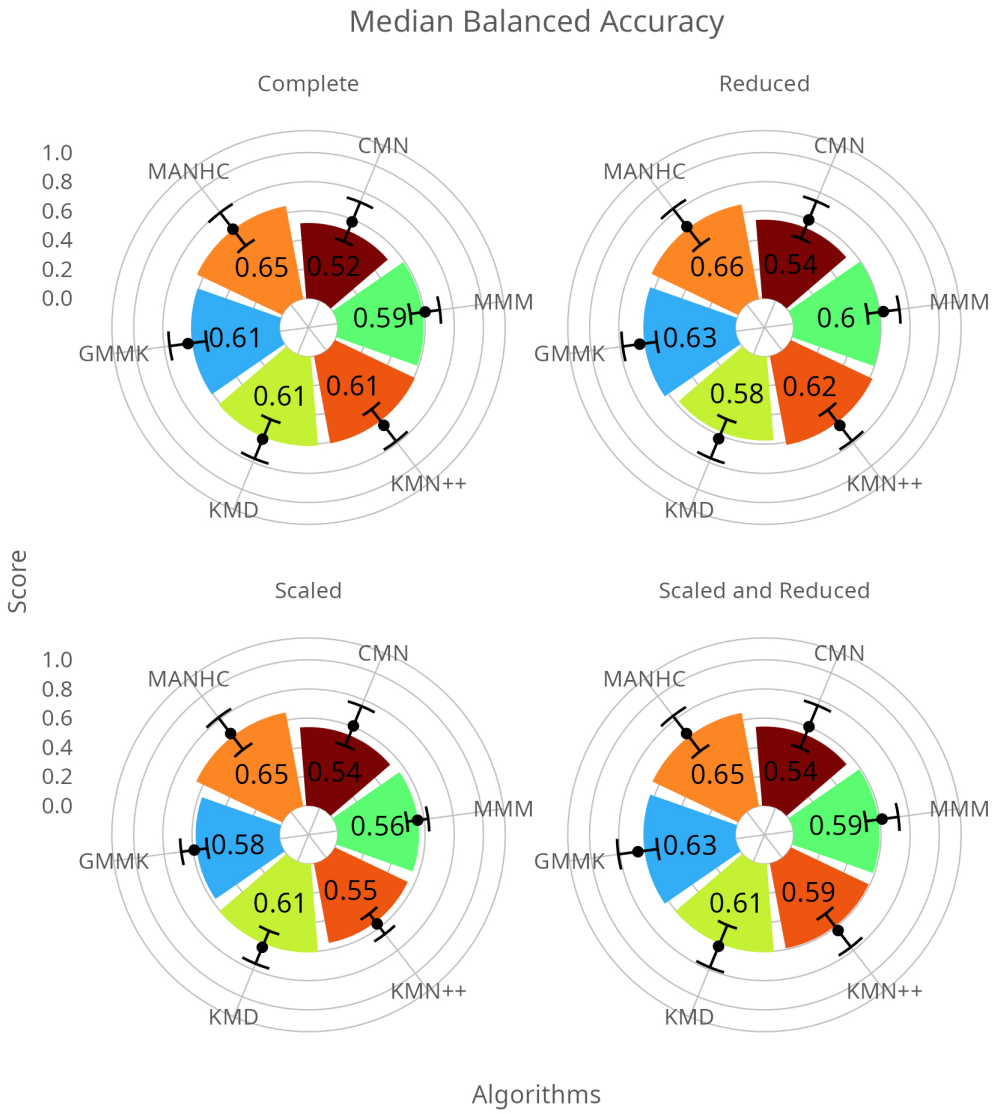


Figure 5.44: A median of balanced accuracy with mean and standard deviation in Arrythmia

According to the median balanced accuracy index in Figure 5.46, the best-performing algorithm was MANHC. The highest result was 0.66, and the lowest was 0.65. The second was GMMK, with the highest result of 0.63 and the lowest of 0.58, so the difference is much more extensive for the advantage of MANHC. Scaling affected GMMK, KMN++ and MMM negatively. However, it did not impact KMD. As for CMN, it even improved its results by 0.02 points. However, when the data was reduced, KMD performed worse than in any other

data type—scaling of reduced data brought back its highest result of 0.61. The reduction of the data also helped GMMK and MMM. We can see that both performed worse after scaling but better after data reduction.

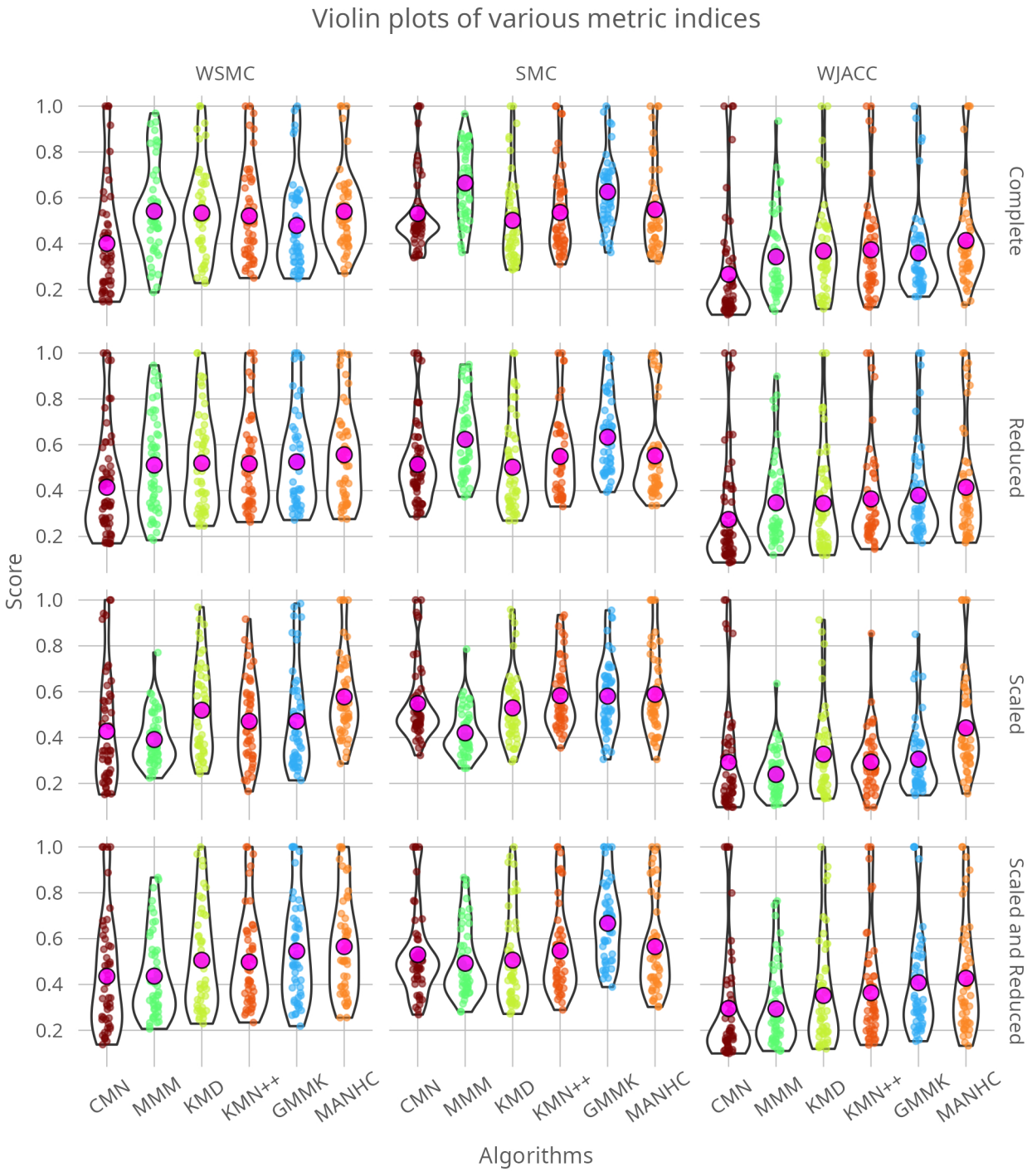


Figure 5.45: Other quality indices in Arrhythmia

Figure 5.45 shows a slight disagreement among the WSMC, SMC, and WJACC results. For instance, in the complete data, the mean value of MMM was comparable to MANHC and slightly above KMD, according to WSMC. However, SMC shows that MMM had the highest

score, followed by GMM and MANHC. On the other hand, MANHC obtained the best mean score in WJACC. After data reduction, the results were slightly worse for all algorithms, but GMM scored better than before.

Scaling of the data slightly improved the results of the distance-based algorithms, even in CMN. The violin plot became more uniformly distributed, and the lower values were not as wide as in the case of complete or reduced data. However, scaling lowered the scores of MMM, with more density observed below 0.4 values. Moreover, it decreased the mean value of GMM, which was previously above 0.6 and slightly below after scaling.

Regarding the raw number of successes, according to the SMC index, MMM scored highest when the data was complete.

Figure 5.46 shows that metrics are positively correlated and only slightly disagree in showing the best-performing algorithm. Considering different metrics, HC was the best-performing algorithm, and GMMK was the second-best. However, if we measure the rought number of successes, then the MMM algorithm had the highest results across all of the algorithms.

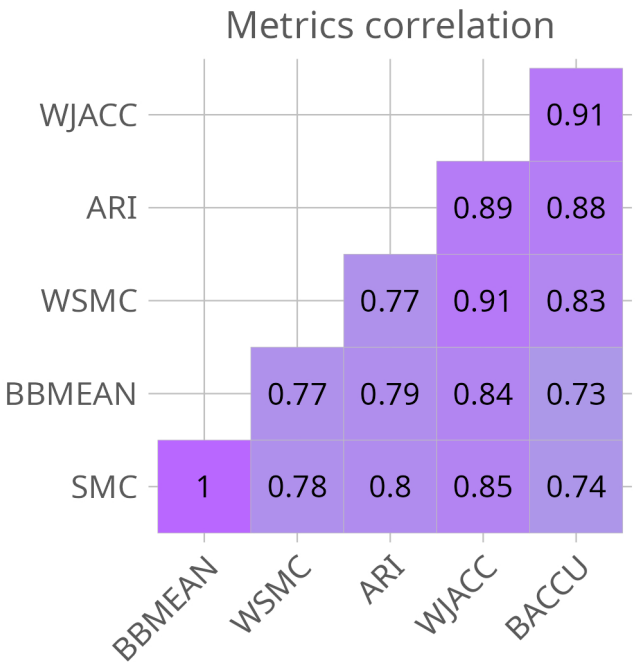


Figure 5.46: Metrics correlation in Arrhythmia

5.2.7 NASA Kepplers

On 6 March 2009, The Delta II rocket took NASA's Kepler Space Telescope and carried it into space. The telescope was focused on an area with about 150,000 stars, like the sun within our solar system. Its ultimate purpose was to identify other habitable planets, excluding our own. Kepler's discoveries contain planets that orbit in so-called habitable areas. The habitable area means it orbits sufficiently far from a star. Sufficient is when the surface temperature may be fit for life-giving liquid water.

The first discovery important discovery was Kepler-22b. It is an example of a habitable zone planet found during the mission. However, because it is almost 2.4 times the size of Earth, it is considered too large to be solid and life-supporting. However, scientists are convinced that different habitable zone planets found by the Kepler mission might be rocky, such as Kepler-62f, which is 40% larger than Earth. A twin to Earth that has the same temperature and size as Earth was yet to be discovered. Still, the analysis is far from over as scientists continue to search the Kepler data for the tiny signature of such a planet. Other Kepler discoveries include hundreds of star systems hosting multiple planets and have established a new class of planetary systems where planets orbit more than one sun.

The mission ended its science observations after a faulty reaction wheel affected the telescope's ability to point precisely. However, the telescope remained in service due to its next-generation mission proposal, K2.

By analysing Kepler's information, the scientific community has recognised over 3,600 candidates considered planets. They confirmed that 961 are indeed planets, many as small as Earth. Findings using the Kepler Space Telescope account for over half of all the known exoplanets. The dataset used was a cumulative Activity Table of Kepler Objects of Interest (KOI). The KOI table contains information about the single KOI activity tables. It represents the actual results of different findings of the Kepler light curves. The goal of the cumulative table is to gather suitable qualities and stellar and planetary data for all KOIs, in one place. All of the data presented originates in other KOI activity tables. The last status update was on 27 September 2018 and is considered complete.

In the Figure 5.47, the observations consited of only two classes - planets confirmed as Keppler type and the false positive ones. PCA one again reflects tSVD results but on a different scale. It is not easy to read the groups from the picture. Randon Projection shows more spread between clusters, but the points are densely packed. In the case of tSNE, we can see some patterns, but the groups are highly overlapping, and it will be challenging to guess them from the picture.

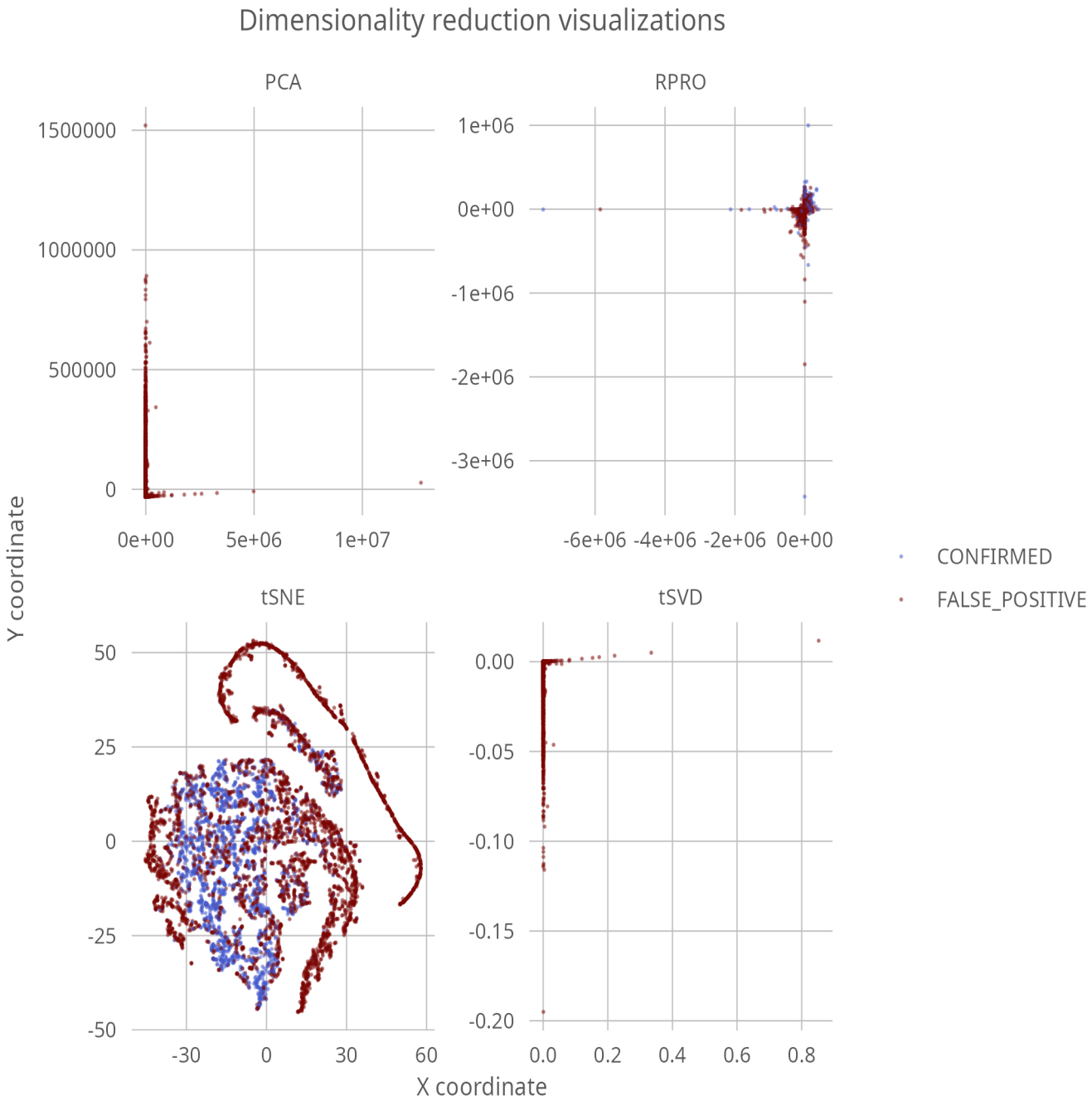


Figure 5.47: Visualization of various dimensionality reduction techniques in Kepplers

Results

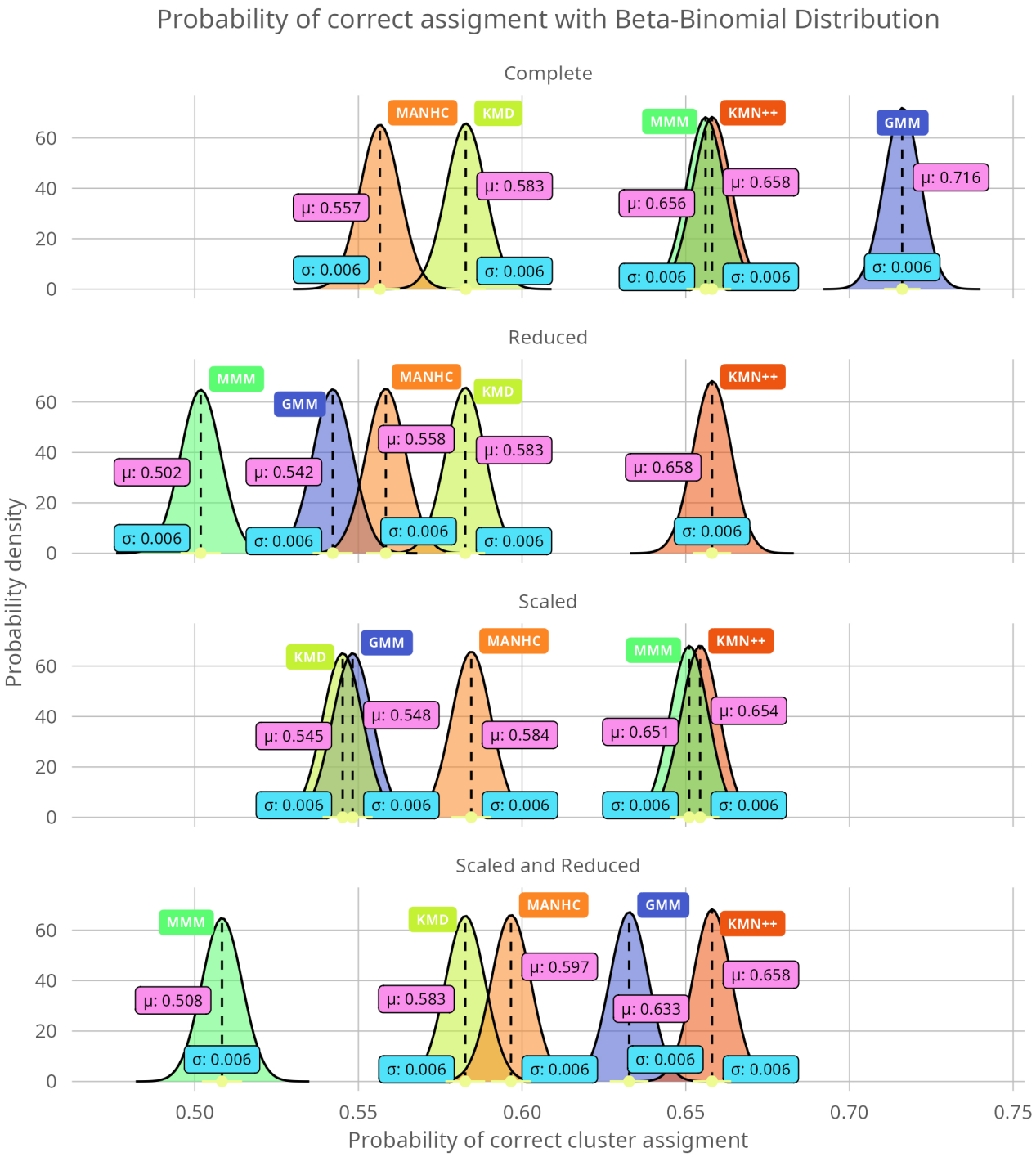


Figure 5.48: Beta-binomial distribution as a quality metric in Kepplers

Figure 5.48 showing beta-binomial distribution has a significant advantage. We can prepare complete visual figures having even one data set. When the data was complete, the highest score overall belonged to the GMM. After scaling the data, KMN++ and MMM are highly overlapped with a difference of 0.003 points, rendering them the best in this data type. KMED, MANCMN, and GMM scored similarly, with a maximum of 0.01 difference between

their means. HC scored better than them, by 0.04 points, but worse than KMN++ and MMM by almost 0.07 points.

Reduction of the data does not impact KMN++ scores. We can see that MMM was very close to it, with a mean of 0.651. Scaling of the reduced data improved the GMM result, making it the best in this file category. In both cases of data reduction, MMM was the last in the comparison.

6 Summary

6.1 Aggregated results

There are plethora numbers of metrics. Depending on the case, some might be better than others, especially in unbalanced data. However, here, the simplest one, a rough number of successes was used in the aggregation. The scores have been normalized using the min-max method, where the lowest score is set to 0, the highest score is set to 1, and all other scores are scaled proportionally. The scores in the table represent the proportion of correctly assigned clusters, with a higher score indicating better performance. Lastly, the data were sorted based on the highest sum across different data types.

6.1.1 Simulated data

No	Algorithm	Complete	Reduced	Scaled	Scaled And Reduced
1	GMM	1.00	0.79	0.96	0.81
2	KMN	0.81	0.57	0.84	0.58
3	KMN++	0.73	0.51	0.75	0.52
4	GMMK	0.74	0.48	0.71	0.41
5	MANHC	0.67	0.42	0.67	0.42
6	HC	0.63	0.40	0.63	0.40
7	CMN	0.54	0.39	0.56	0.40
8	MMMK	0.59	0.37	0.04	0.00
9	MMM	0.53	0.32	0.04	0.00
10	MANCMN	0.16	0.10	0.15	0.10
11	KMD	0.13	0.05	0.13	0.06

Table 6.1: Entries represent numbers of successes (correctly assigned clusters) normalized to all assigments in Simulated Multivariate Normal Mixtures

Looking at the results, the GMM algorithm performed the best overall, with a score of 1 for complete data, 0.79 for reduced data, 0.96 for scaled data, and 0.81 for scaled and reduced data. This suggests that GMM is a good choice for clustering this dataset, mainly when using complete data. The KMN algorithm also performed well, with scores ranging from 0.57 to 0.84 depending on the type of data used. However, it did not perform as well as GMM overall.

The KMN++ and GMMK algorithms had similar performance, with scores ranging from 0.41 to 0.73 depending on the type of data used. They performed better than most other algorithms, except for GMM and KMN. The MANHC and HC algorithms had scores ranging

from 0.4 to 0.67 depending on the type of data used, indicating moderate performance.

The CMN, MMMK, and MMM algorithms had lower scores, ranging from 0 to 0.56, indicating poor performance. Finally, the MANCMN and KMD algorithms had the lowest scores, ranging from 0 to 0.16, showing inferior performance.

The results suggest that GMM and KMN are good choices for clustering in this dataset, mainly when using complete and scaled data.

No	Algorithm	Complete	Reduced	Scaled	Scaled and Reduced
1	MANHC	0.92	0.52	0.94	0.51
2	HC	0.92	0.5	0.92	0.5
3	KMN	0.89	0.52	0.91	0.52
4	KMN++	0.86	0.5	0.89	0.5
5	GMM	0.23	0.58	0.79	0.59
6	GMMK	0.84	0.35	0.58	0.33
7	CMN	0.59	0.32	0.61	0.34
8	KMD	0.6	0.32	0.53	0.33
9	MMM	1	0.54	0	0.18
10	MMMK	0.97	0.54	0.01	0.08
11	MANCMN	0.05	0.13	0.02	0.12

Table 6.2: Entries represent numbers of successes (correctly assigned clusters) normalized to all assigments in Simulated Multinomial Mixtures

Based on the results, the MMM algorithm outperformed all the other algorithms with a score of 1 on the complete dataset, indicating the highest score across algorithms. The MMMK algorithm also performed well, with a score of 0.97 on the complete dataset, showing a high score in cluster assignment. However, MMM performance was vastly in the reduced, scaled, and scaled and reduced datasets.

Among the other algorithms, HC, KMN, and KMN++ also performed well, with scores ranging from 0.86 to 0.92 on the complete dataset. These algorithms showed consistent performance across all the datasets. GMMK performed much better than GMM in the complete dataset but had worse performance on the reduced, scaled, and scaled and reduced datasets. CMN and KMD had lower scores ranging from 0.32 to 0.61 on the complete dataset, while GMMK scored 0.58.

Overall, these results suggest that MMM and MMMK are effective algorithms for clustering, particularly on complete datasets.

6.1.2 Real data

No	Algorithm	Complete	Reduced	Scaled	Scaled and Reduced
1	GMM	0.78	0.49	0.51	0.53
2	GMMK	0.76	0.36	0.64	0.5
3	HC	0.42	0.38	0.77	0.51
4	MMMK	0.91	0.82	0.12	0.2
5	KMN++	0.4	0.44	0.66	0.56
6	KMN	0.41	0.43	0.6	0.55
7	MANHC	0.45	0.41	0.64	0.46
8	MMM	1	0.67	0	0.27
9	KMD	0.38	0.39	0.49	0.44
10	CMN	0.12	0.1	0.72	0.32
11	MANCMN	0.15	0.08	0.39	0.31

Table 6.3: Entries represent aggregated successes (correctly assigned clusters), normalized to all assignments in Real Data

GMM and GMMK had the highest sum scores, with GMM achieving the highest scores in the Scaled dataset and GMMK achieving the highest scores in the Reduced dataset. However, it's worth noting that the highest score achieved belongs to MMM when the data was complete.

HC, KMN, and KMN++ performed similarly across all four datasets, with scores ranging from 0.40 to 0.77. MANHC performed slightly better than HC, with scores ranging from 0.41 to 0.64. KMD and CMN had the lowest scores among all algorithms, ranging from 0.12 to 0.39. MANCMN performed slightly better than KMD and CMN, with scores ranging from 0.08 to 0.39.

In summary, GMM and GMMK had the highest sum scores, but MMM achieved the highest in the complete dataset.

6.2 Conclusions

This research formulated appropriate versions of the EM algorithm to decompose Gaussian Multivariable Mixtures and Multinomial Mixtures. Numerical stability was ensured by switching to the logarithmic scale when necessary and adding small normalizing constants to avoid division by zero.

The implementation of these algorithms, named MultinomEM and GaussEM for Multinomial Mixture Models and Gaussian Mixture Models, respectively, was created in R and is openly available on the GitHub platform. The package is still undergoing rigorous testing, but the algorithm should be stable for most computational instances. Regular updates will be made, and future implementations may involve rewriting some parts of the code in Rcpp or Armadillo for efficiency.

To compare the performance and efficiency of the algorithms, several distance-based algorithms were implemented using the available source code, and several metrics were formulated.

While some of these metrics, such as those derived with the help of the Hungarian algorithm, are highly correlated, each metric has advantages and shortcomings. The ARI index, for example, is easy to implement without additional steps but may only be suitable in some cases. The proposed metric provides additional information about probability density based on beta distribution but requires more mathematical work to balance appropriately.

An extensive simulation study was conducted using thousands of Multivariate Gaussian Mixtures and Multinomial Mixtures to test the algorithms' performance. An R script was prepared for this purpose, allowing the creation of selected mixtures with any desired number of observations, dimensions, and components. This enabled a controlled comparison of the algorithms with differing numbers of parameters, dimensions, and clusters.

A curated set of real datasets from various publicly available sources, including genomics/medical data, was also prepared. Based on these datasets, hundreds of different components were prepared, with each group combination occurring only once in the same set. This allowed testing the algorithm's performance with a controlled and differing number of components.

The model-based algorithms presented in this thesis are potent tools in unsupervised clustering methods and are highly competitive compared to distance-based algorithms.

Acknowledgments

I am deeply grateful to my supervisor and mentor, Professor Andrzej Polański, for his invaluable guidance and support throughout my doctoral studies. His ideas and insights have inspired me and helped shape my work.

I would also like to sincerely thank my reviewers, Professor Zbigniew Świder, Professor Grzegorz Dudek, and Professor Marek Kowal, for their time, effort, and thoughtful feedback on this work. Their comments and suggestions have been invaluable.

I also express my heartfelt thanks to my grandparents and aunt for their unwavering encouragement and support throughout my academic journey. While my mother, unfortunately, passed away before I completed my PhD, her steadfast belief in me and her encouragement to pursue my dreams continue to inspire me.

Finally, I want to express my gratitude to my best friend for always asking about my progress and providing me with the encouragement and support I needed to keep going.

To all of you, thank you for your support, guidance, and encouragement. I could not have completed this journey without you.

Bibliography

[1] Chandan K Reddy, *Data Clustering: Algorithms and Applications*, Chapman and Hall/CRC, 2018.

[2] Brian S Everitt, et al., "Cluster analysis 5th ed", , 2011.

[3] Leonard Kaufman and Peter J Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344, John Wiley & Sons, 2009.

[4] Christian Hennig, et al., *Handbook of cluster analysis*, CRC Press, 2015.

[5] Charles Bouveyron, et al., *Model-based clustering and classification for data science: with applications in R*, vol. 50, Cambridge University Press, 2019.

[6] Geoffrey J McLachlan and Thriyambakam Krishnan, *The EM algorithm and extensions*, vol. 382, John Wiley & Sons, 2007.

[7] Nick T Thomopoulos, "Statistical distributions", *Applications and Parameter Estimates Cham, Switzerland: Springer International Publishing*, 2017.

[8] Merran Evans, et al., *Statistical distributions*, John Wiley & Sons, 2011.

[9] Walter Frank Raphael Weldon, "I. Certain correlated variations in crangon vulgaris", *Proceedings of the Royal Society of London*, 51(308-314), 1892, pp. 1–21.

[10] Peter Schlattmann, *Medical applications of finite mixture models.*, Springer, 2009.

[11] Geoffrey J McLachlan, et al., "Finite mixture models", *Annual review of statistics and its application*, 6, 2019, pp. 355–378.

[12] Arthur P Dempster, et al., "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1), 1977, pp. 1–22.

[13] Usama M Fayyad, et al., "Initialization of Iterative Refinement Clustering Algorithms.", in *KDD*, 1998, pp. 194–198.

[14] Jean-Patrick Baudry and Gilles Celeux, "EM for mixtures: Initialization requires special care", *Statistics and computing*, 25, 2015, pp. 713–726.

[15] Pierre Blanchard, et al., "Accurately computing the log-sum-exp and softmax functions", *IMA Journal of Numerical Analysis*, 41(4), 2021, pp. 2311–2330.

[16] Fengzhen Zhai and Qingna Li, "A Euclidean distance matrix model for protein molecular conformation", *Journal of Global Optimization*, 76, 2020, pp. 709–728.

[17] Kimberly L Elmore and Michael B Richman, "Euclidean distance as a similarity metric for principal component analysis", *Monthly weather review*, 129(3), 2001, pp. 540–549.

[18] MD Malkauthekar, "Analysis of Euclidean distance and Manhattan distance measure in Face recognition", in *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, IET, 2013, pp. 503–507.

[19] Sylvia Fruhwirth-Schnatter, et al., *Handbook of mixture analysis*, CRC press, 2019.

[20] Fionn Murtagh and Pedro Contreras, "Algorithms for hierarchical clustering: an overview", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1), 2012, pp. 86–97.

[21] Jiawei Han, et al., *Data mining: concepts and techniques*, Morgan kaufmann, 2022.

[22] John A Hartigan and Manchek A Wong, "Algorithm AS 136: A k-means clustering algorithm", *Journal of the royal statistical society series c (applied statistics)*, 28(1), 1979, pp. 100–108.

[23] Laurence Morissette and Sylvain Chartier, "The k-means clustering technique: General considerations and implementation in Mathematica", *Tutorials in Quantitative Methods for Psychology*, 9(1), 2013, pp. 15–24.

[24] Richard McElreath, *Statistical rethinking: A Bayesian course with examples in R and Stan*, Chapman and Hall/CRC, 2020.

[25] Jason Offutt, *Ella Ewing: The Missouri Giantess*, Truman State University Press, 2017.

[26] AltonWeb, "The Life and Times of Robert Wadlow", , n.d., URL `http://altonweb.com/history/wadlow/`.

[27] Marc Kéry and J Andrew Royle, *Applied hierarchical modeling in ecology: Analysis of distribution, abundance and species richness in R and BUGS: Volume 2: Dynamic and advanced models*, Academic Press, 2020.

[28] Lawrence Hubert and Phipps Arabie, "Comparing partitions", *Journal of classification*, 2(1), 1985, pp. 193–218.

[29] Raimundo Real and Juan M Vargas, "The probabilistic basis of Jaccard's index of similarity", *Systematic biology*, 45(3), 1996, pp. 380–385.

[30] Hadley Wickham, *ggplot2: Elegant Graphics for Data Analysis*, Springer-Verlag New York, 2016, URL `https://ggplot2.tidyverse.org`.

[31] Laurens Van der Maaten and Geoffrey Hinton, "Visualizing data using t-SNE.", *Journal of machine learning research*, 9(11), 2008.

[32] Eleonore Lebeuf-Taylor, et al., "The distribution of fitness effects among synonymous mutations in a gene under directional selection", *eLife*, 8, 2019, p. e45952, URL `https://doi.org/10.7554/eLife.45952`.

[33] National Cancer Institute, "Genomic Data Commons", `https://portal.gdc.cancer.gov/`, accessed 2023, accessed on March 20, 2023.

[34] David Benjamin, et al., "Calling somatic SNVs and indels with Mutect2", *Biorxiv*, 2019, p. 861054.

[35] Cyriac Kandoth, et al., "Mutational landscape and significance across 12 major cancer types", *Nature*, 502(7471), 2013, pp. 333–339.

[36] Terry A Brown, "Genomes 5", , 2023.

[37] Bohdan B Khomtchouk, "Codon usage bias levels predict taxonomic identity and genetic composition", *bioRxiv*, 2020.

[38] Brian McFee et al., "Librosa: Audio and music signal analysis in Python", `https://librosa.org`, 2015–, accessed on March 19, 2023.

[39] Andreea D Ceornodolea, et al., "Epidemiology and management of atrial fibrillation and stroke: review of data from four European countries", *Stroke research and treatment*, 2017, 2017.

List of Figures

List of Tables