Mixture Models Project

Mahdiou Diallo, Helmy El Rais, and Dah Diarra

April 2020

Abstract

As a project for our Mixture Models class in the context of a Master's degree at the University of Paris, we were asked to run several algorithms of clustering and coclustering on known datasets. We compare them and explore the benefits of finding the consensus between algorithms that are known to be well suited to a task, but optimize different kinds of criteria. This report describes our experiments and results. The code of this project can be found in our github repository[1].

Keywords: clustering, coclustering, consensus, gaussian models, multinomial models

1 Introduction

The mixture models approach supposes a dataset is a sample obtained from a population containing several subgroups with similar characteristics. For example a dataset containing the pixel values of several face images can be viewed as a gaussian mixture of it's classes that can then be described with their mean vectors and covariance matrices.
Algorithms used are based on the Expectation-Maximization algorithm. They usually differ on the following points: the underlying model, the similarity/dissimilarity metric used, the simplifying assumptions, etc. They have their strengths and weaknesses. In the same way faces can be similar in terms of their expression, orientation, or the subject of the picture, different models group data points differently. Knowing that, we can use models that are known empirically to work well in a domain and combine the different partitions that they give. The consensus obtained might better represent the groups of individuals in the population. In this report, we start by describing the datasets used, then the different algorithms applied for clustering and for finding the consensus, and finally we present our results and discuss them.

2 Datasets

We worked on two types of datasets: datasets with continuous variables and contingency tables. We describe the datasets according to the number of samples, the number of features, the number of classes, the sparsity (ratio of zeros), and the balance (ratio of the minority class over the majority class).

2.1 Continuous

We used datasets with continuous values in the first exercise. We were asked to use the following datasets JAFFE, MNIST5, MFEA, USPS. and OPTIDIGITS which are described in table 1.
We can see that the given datasets are well balanced. Algorithms that assume equal proportions should not face difficulties.

As MNIST5 and USPS were too large to get results in time, they were excluded from our study.

[1] https://github.com/mahdiou/Projet_MixMod

Table 1: Description of continuous datasets

datasets	# samples	# features	# classes	sparsity	balance
JAFFE	213	676	213	0.96	0.87
OPTDIGITS	5620	64	10	0.51	0.97
MFEA	2000	240	10	0.61	1.00
MNIST5	3495	784	10	0.19	0.80
USPS	9298	256	10	0.99	0.46

2.2 Contingency

In the remaining exercises, we were asked to use 4 datasets from the package CLUTO [Kar02]. We chose the datasets shown in table 2 according to the following criteria: small enough size, fewer features than samples, large number of features, different levels of balance.

Table 2: Description of contingency tables.

datasets	# samples	# features	# classes	sparsity	balance
re0	1504	2886	13	0.98	0.02
classic	7094	41681	4	0.99	0.32
fbis	2463	2000	17	0.92	0.08
wap	1560	8460	20	0.98	0.01

3 Methods

3.1 Clustering

3.1.1 Gaussian Mixture Models

For clustering the continuous dataset, we used the package `mclust` [Scr+16]. It uses Gaussian finite mixture models (GMM). Given a dataset $X = \{x_1, ..., x_i, ..., x_n\}$ of n independent identically distributed points, the distribution of every observation is specified by a probability density function f through a finite mixture model of G components:

$$f(x_i, \Psi) = \sum_{k=1}^{G} \pi_k f_k(x_i; \mu_k, \Sigma_k)$$

where Ψ is the set of parameters, π_k, μ_k, and Σ_k are the proportion, the mean and covariance matrix of the k^{th} mixture component, respectively.
With different constraints on the values of π_k and Σ_k, `mclust` is able to generate 28 different models.

When we executed the algorithm, we saw that some of the parameter were not performing well, because the data matrix were not full rank, so to make sure that the algorithm will be able to inverse the variance co-variance matrix, we projected the data along all eigen vector whom eigen value are different than 0, this way the data matrix will be full rank and the algorithms will be able to performs on more parameter, as we can see on table 3.1.1.

Table 3: Description of mixture model parameter.

datasets	# models	# parttions
jaffe	EII, VII, EEI, VEI, EVI, VVI	6
optdigits	EII, VII, EEI, VEI, EVI, VVI, EEE	7
mfea	EII, EEI, EEE, EEV	4

3.1.2 von Mises-Fisher (vMF) Models

These models were used on the contingency matrices. It is known to work well on very sparse datasets and directional data. The distribution of points following a mixture of vMF distributions can be written as follows:

$$f(x_i|\Theta) = \sum_k \pi_k f_k(x_i|\mu_k, \kappa_k)$$

where pi_k, μ_k, and κ_k are the proportion of class k and mean direction and concentration parameter of class k in the formula of the vMF distribution. In the optimization of the complete log-likelihood, the most difficult operation is the estimation of the *confluent hypergeometric limit function* $C_d(\kappa)$. Several algorithms exist to estimate its value. We use the R package `movMF` [HG14] which offers 7 algorithms for this estimatiom and 3 variants for the E step of the EM algorithm.
We also use [Hor+12] in R which is derived from the vMF model with some simplifying assumptions. In python, we use the implementation from the package `CoClust`.

3.2 Coclustering

Coclustering algorithms attempt to cluster the rows and columns (samples and variables) of a data table. Although they can be applied to both continuous data and contingency tables, in this project we were asked to use them only on the contingency tables. We used the coclustering functions provided by the `CoClust` Python library [RMN18]. This library provides various clustering and coclustering algorithms that optimize different criteria. The ones used in this project are the following:

- **`CoClustInfo`**: Information-Theoretic Co-clustering
- **`CoClustMod`**: Co-clustering by direct maximization of graph modularity.
- **`CoClustSpecMod`**: Co-clustering by spectral approximation of the modularity matrix.

3.3 Consensus

After generating several partitions both from several runs of the same model and from multiple models, we look for a way to find a combination of those partitions that will better represent our subpopulations. Here we describe the methods used.

3.3.1 Multinomial model

In this method, we make a new matrix by concatenating the partitions obtained by each algorithm. For a dataset with n samples and after obtaining r partitions, we have a table $P_{n\times r}$. The values in the tables are taken as categorical values that can also be clustered using a multinomial model. We use `Rmixmod` [Leb+15]. The obtained partition is a consensus over those labels that minimizes the BIC criterion.

3.3.2 Hypergraphs

Here finding the consensus is formalized as a combinatorial optimization problem in terms of shared mutual information [SG03]. We use the Python library `Cluster_Ensembles` which provides 3 hypergraph partitioning algorithms:

- **`CSPA`** (Cluster-based Similarity Partitioning Algorithm): it defines an induced similarity measure based on the fact that samples belong to the same cluster.
- **`HGPA`** (HyperGraph Partitioning Algorithm): the best partition is found by approximating the maximum mutual information objective with a constrained minimum cut objective.
- **`MCLA`** (Meta-CLustering Algorithm): it tries to find and consolidate groups of clusters.

Due to problems encountered in the execution of the code, we did not get results for the HGPA algorithm.

3.3.3 Co-association matrices

Given a partition p of our dataset $X_{n\times d}$, a co-association matrix $C_{n\times n}$ is a matrix where $C_{ij} == 1$ if X_i and X_j belong to the same partition. With multiple partition, we can build a total co-association matrix C_{total} that is the sum of all the association matrices. We then execute a coclustering algorithm on this matrix to get our consensus. The algorithm used in this project was `CoClustInfo`.

4 Results

4.1 Continuous dataset (Q2)

Table 4: NMI results on continous datasets

datasets	best	consensus	consensus$_{BIC}$	consensus$_{loglik}$
JAFFE	**0.95**	0.47	0.86	0.85
OPTDIGITS	0.68	**0.74**	0.71	0.71
MFEA	0.01	0.62	0.68	**0.75**

As we can see on table 4.1, the Mixture Model get an overall good NMI score except for the *mfea* dataset but the consensus over all possible Mixture model allow us to get better NMI score in comparison, as we can see the *mfea* dataset get a big increase in term of NMI score, by passing from 0.01 (with the best Mixture Model over all Mixture Model) to 0.62 (with the consensus over all possible Mixture Model).

We can also see a decrease in term of NMI score for the *jaffe* dataset, this can be explained by the fact that most of the *jaffe* dataset partition came from weak model with big likelihood score, big BIC score and of course low NMI score, indeed 4 out of 6 model have a NMI score near 0.4 while the 2 remaining model have a NMI score near 0.8, so we have a majority of weak model that lower the global NMI score that we get with the consensus.
we can see that we have similar result between the consensus on the partition from all possible Mixture Model and the consensus on the partition from the best Mixture Model (in term of log likelihood or BIC score), the goal of this approach is to delete all the partition that came from weak model (in term of log likelihood score and BIC score), and indeed with the *jaffe* dataset we double the NMI score by lowering the influence of weak model.

4.2 Contingency dataset

4.2.1 Mutual Information (Q3)

The ColustInfo algorithm being an algorithm depending on its initialization, we have performed 15 iteration on each data set and kept the 5 best partitions. these better partitions are then used to make a consensus and a co-association in order to obtain much more profitable partitions. knowing that the spherical kmeans algorithm gives good results on the textual data or sparse matrix we also introduce its best partitions too.To compare the results we have used two metric namely:

- normalized mutual info (NMI)
- adjusted rand Index (ARI)

Table 5: Comparison of NMI and ARI of the co-association partition and consensus partitions for CoClustInfo

	Assoc Co-Info		Conssen Co-Info		Assoc Co-Info-sk		Conssen Co-Info-sk	
dataset	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
re0	0.28	0.10	0.38	0.15	0.35	0.16	**0.39**	0.17
fbis	0.44	0.28	0.52	0.28	0.52	0.34	**0.55**	0.35
wap	0.43	0.30	0.50	0.25	**0.52**	0.34	**0.52**	0.30

Table 6: Comparison of NMI and ARI of the co-association partition and consensus partitionsfor CoClustInfo x2

	Assoc Co-Info x2		Conssen Co-Info x2		Assoc Co-Info-sk x2		Conssen Co-Info-sk x2	
dataset	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
re0	0.24	0.07	0.29	0.10	0.35	0.16	**0.38**	0.16
fbis	0.40	0.22	0.51	0.32	0.52	0.34	**0.57**	0.34
wap	0.40	0.18	**0.56**	0.41	0.52	0.34	0.54	0.31

Table 7: Comparison of NMI and ARI of the co-association partition and consensus partitions for CoClustInfo x3

	Assoc Co-Info x3		Conssen Co-Info x3		Assoc Co-Info-sk x3		Conssen Co-Info-sk x3	
dataset	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
re0	0.26	0.12	0.36	0.14	0.38	0.19	**0.39**	0.18
fbis	0.44	0.26	0.55	0.30	0.55	0.39	**0.58**	0.40
wap	0.39	0.28	0.52	0.31	0.55	0.41	**0.54**	0.33

As shown in table 5, table 6 and table 7 we can see that the consensus has a much better result in terms of NMI and ARI than the co-association and also the addition of the spherical kmeans partitions improves the performance of the clustering. We can also notice that increasing the number of classes of terms also improves the results.

4.2.2 Modularity (Q4)

As said in Q3 we did the same process with CoclustMod and CoclustSpecMod. We also made a consensus on the best partitions of all algorithms. It should be noted that we did not increase the size of the class of terms because it was not requested in this study and the two previous methods do not allow to do so.

Table 8: Comparison of NMI and ARI of the co-association partition and consensus partitions for CoClustMod

	Assoc Co-Mod		Conssen Co-Mod		Assoc Co-Mod-sk		Conssen Co-Mod-sk	
dataset	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
re0	0.29	0.12	0.34	0.18	0.35	0.18	**0.41**	0.21
fbis	0.46	0.31	0.40	0.23	0.55	0.36	**0.57**	0.39
wap	0.46	0.39	0.46	0.33	**0.58**	0.46	0.46	0.33

As shown in table 8, table 9 and 10 it can be seen that the consensus and co-association methods do not seem to have a great difference in general and the partitions of the spherical kmeans largely contribute to the improvement of the results. We can also see that the consensus obtained on the set of partitions of each method is comparable to a means of the results obtained previously.

4.2.3 von Mises-Fisher models (Q5)

As said above, we had the choice with the method used at the E step and the algorithm for the estimation of $C_d(\kappa)$. We used the default algorithm for the estimation and used all three available options for the E step. We kept 5 copies of each model and we combined the 15 models obtained in the following ways:

- Keep the best partition the Bayesian Information Criterion (BIC).
- Combine all the partitions using `Rmixmod`
- Combine the 10 best partitions according to the BIC.

The NMI and ARI values are summarized in table 11. We can see that in general, the consensus is either as good as the best model or even better. That is because although the "best" model is the one that minimizes the BIC, that does not tell us that the partition obtained from it will have the best NMI or ARI. Using several models can help reduce that problem. Here are alternative ways to combine the partitions:

Table 9: Comparison of NMI and ARI of the co-association partition and consensus partitions for CoClustSpec-Mod

	Assoc Co-spec-Mod		Conssen Co-spec-Mod		Assoc Co-spec-Mod-sk		Conssen spec-Mod-sk	
dataset	NMI	ARI	NMI	ARI	NMI	ARI	NMI	ARI
re0	0.36	0.14	0.29	0.07	0.37	0.19	**0.38**	0.13
fbis	0.49	0.30	0.45	0.30	0.49	0.35	**0.52**	0.37
wap	**0.53**	0.35	0.44	0.22	0.50	0.39	0.51	0.30

Table 10: Comparison of NMI and ARI of the co-association partition and consensus partitions for CoClustSpec-Mod

	Conssen all algo	
dataset	NMI	ARI
re0	0.38	0.17
fbis	0.50	0.31
wap	0.54	0.35

- Get the 10 best partitions according to a criterion (e.g. BIC) and add 5 partitions randomly chosen from the all the partitions
- use the 5 best partitions according to the BIC, 5 according to the log-likelihood, according to the AIC, etc. Using different criteria, even if they are related could give different kinds of models. Also a model that is picked by multiple criteria will have more weight.

We did not explore those possibilities due to time constraints.

Table 11: Comparison of NMI and ARI of the best partition and consensus partitions

	Best		Consensus All		Consensus Top 10	
dataset	NMI	ARI	NMI	ARI	NMI	ARI
re0	**0.35**	0.20	**0.35**	0.24	0.33	0.18
classic	0.30	0.09	0.33	0.12	**0.35**	0.16
fbis	**0.53**	0.34	0.51	0.34	0.5	0.32
wap	0.52	0.34	**0.56**	0.42	0.55	0.31

In the next part, we ran the spherical K-means algorithm on the datasets with different values of k (from 2 to 30). To find the best consensus partition, we ran Rmixmod's clustering on the labels using all the values of k. We kept the best partition according to the BIC. The results are shown in table 12.
We can see that except in the case of the *classic* dataset, the estimated number of partitions was close to the real number. The NMI values are almost as good as the ones obtained using the more general vMF model. The ARI values are not as good. We think that those results could be further improved by doing a consensus only using the best models as done above.

5 Conclusion

In this project, we observed how consensus methods can help improve the partitions formed on a dataset. We saw different ways to combine the labels obtained from different methods. The results showed dramatic improvements in some cases, but as always in machine learning we know that there is no single best solution. It all depends on the specific applications. We saw that consensus methods are more computationally expensive due to the fact that we need to generate several partitions. We also saw that in general we should not just combine the results of all the models without thinking. Sometimes we need to filter the results before combining them. The MFEA datasets reminded us that we should never overlook data exploration and the necessary preprocessing so that the algorithms will work correctly.
The results of the CoClustInfo consensus showed us that even for a symmetric matrix, choosing different numbers

Table 12: Consensus on Spherical K-means results

dataset	# classes	# partitions	NMI	ARI
re0	13	10	0.37	0.24
classic	4	25	0.31	0.17
fbis	17	15	0.56	0.39
wap	20	12	0.54	0.36

of clusters for rows and columns could lead to better results. The main takeaway is that combining good algorithms that optimize different criteria can improve the results of a clustering.

6 Future Works

As noted in the text, the definition of "best" always depends on the criterion used. Therefore, we can look for ways to combine generated partitions by considering different criteria, by adding randomness, or by find some kind of weights that will be received from the algorithms that generated the partitions.

References

[Kar02] George Karypis. *CLUTO A Clustering Toolkit*. Tech. rep. 02-017. Available at http://www.cs.umn.edu˜cluto. Dept. of Computer Science, University of Minnesota, 2002.

[SG03] Alexander Strehl and Joydeep Ghosh. "Cluster Ensembles — a Knowledge Reuse Framework for Combining Multiple Partitions". In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 583–617. ISSN: 1532-4435. DOI: 10.1162/153244303321897735. URL: https://doi.org/10.1162/153244303321897735.

[Hor+12] Kurt Hornik et al. "Spherical k-Means Clustering". In: *Journal of Statistical Software* 50 (Sept. 2012), pp. 1–22. DOI: 10.18637/jss.v050.i10.

[HG14] Kurt Hornik and Bettina Grün. "movMF: An R Package for Fitting Mixtures of von Mises-Fisher Distributions". In: *Journal of Statistical Software* 58.10 (2014), pp. 1–31. DOI: 10.18637/jss.v058.i10.

[Leb+15] Rémi Lebret et al. "Rmixmod: The R Package of the Model-Based Unsupervised, Supervised, and Semi-Supervised Classification Mixmod Library". In: *Journal of Statistical Software, Articles* 67.6 (2015), pp. 1–29. ISSN: 1548-7660. DOI: 10.18637/jss.v067.i06. URL: https://www.jstatsoft.org/v067/i06.

[Scr+16] Luca Scrucca et al. "mclust 5: Clustering, Classification and Density Estimation Using Gaussian Finite Mixture Models." In: *R J.* 8.1 (2016), p. 289. URL: http://dblp.uni-trier.de/db/journals/rjour/rjour8.html#ScruccaFMR16.

[RMN18] François Role, Stanislas Morbieu, and Mohamed Nadif. "CoClust: A Python Package for Co-clustering". working paper or preprint. May 2018. URL: https://hal.archives-ouvertes.fr/hal-01804528.