Co-clustering evaluation

Mathieu Pont and Lucas Rodrigues Pereira

February 2020

Abstract

1 Introduction

In the context of the Master's Degree on Machine Learning for Data Science, at the University of Paris (Descartes), we have been given the task to discuss the evaluation of co-clustering in terms of rows clustering but also, and especially, in terms of columns clustering. Indeed, in the literature co-clustering is most of the time evaluated only according the row clusters but rarely the columns clusters are also evaluated.

In this work, in addition to the evaluation of the rows clustering with popular metrics such as NMI, ARI or clustering accuracy, we also evaluate columns clustering with an already proposed method. We evaluate several co-clustering algorithms such as CoclustInfo [Dhillon et al., 2003], CoclustMod [Ailem et al., 2015] and CoclustSpecMod [Labiod and Nadif, 2011]. All algorithms are available in the *CoClust* Python package [Role et al., 2019].

A very popular application for co-clustering is to use textual datasets, we were asked to work on 8 famous datasets of this kind that are commonly used in the literature.

2 Methodology

2.1 Pre-processing

Since we were asked to work on original textual datasets the first step is to pre-process each text to keep only the relevant information. To achieve more accurate results, a robust pre-processing is required. The goal is to keep only the relevant words and remove all the noise.

One of the problem is that, if we don't remove some words of the vocabulary during the pre-processing, the total number of words will be very very high resulting in a dataset with many more variables than individuals. Moreover, some of them will bring noise to the data and can fool the algorithms to get the good results. Some words bring not too much information and therefore we can remove them safely.

Here after a list of some pre-processing steps that we had selected:

- **Remove punctuation**.
- **To lowercase**.
- **Remove line breaks, tabulation etc.**.
- **Remove isolated numbers**.
- **Remove stop words**.
- **Lemmatization**.

- **Make bigrams**.
- **Keep the d most frequent words**.
- **Keep only nouns, adjectives, adverbs and foreign words**.

The before last step can be combined with a step where we remove the top most frequent words because if they are frequent across all documents they will not be discriminative for the co-clustering, however this bias could be manage by TF-IDF that will be explained in 2.2.

Some hyper-parameters are needed to be tuned such as the minimum count of a pair of words to consider them as a bigram or the number of the most frequent words to keep. In order to choose the best parameters we have done many different pre-processing for each dataset, then we encode the diverse pre-processed texts in vector format as described below, and finally run a co-clustering.

Since we have the ground truth label we have computed the NMI, ARI and clustering accuracy for each pre-processing and choose the best for each dataset. This step is crucial in order to validate the pre-processing, to ensure that we have not removed too much information for example.

The last three pre-processing steps were those chosen to be tuned. For the minimum count of a pair of words to treat them as a bigram we choose two values: 5 and 50. The latter is the default parameter but we found that it tends sometimes to not make a lot of bigrams therefore we have decided to also use a smaller value to get more bigrams.

For the number of the most frequent words to keep we have chosen 2000 and 5000 words. These values tend to be in the same order than the average number of documents for each dataset making the resulting matrix quite balanced in terms of number of rows and columns.

Finally the last pre-processing step is kind of a boolean, either keep only the words tag listed above or keep all kinds of words.

We define a notation in this document and in our github repository (link available at the end of this document) in order to make reference to a particular kind of pre-processing:

$$\{2000,\ 5000\}\{\_50,\ \emptyset\}\{\_\text{AllTag},\ \emptyset\}$$

This notation is explained by the table below:

	Notation	**Respective Meaning**
Keep the d most frequent words	$\{2000,\ 5000\}$	$d = \{2000, 5000\}$
Bigram min. count mc	$\{\_50,\ \emptyset\}$	$mc = \{50, 5\}$
Keep only some words tags	$\{\_\text{AllTag},\ \emptyset\}$	$\{Keep\ all\ tag, Keep\ only\ some\ tags\}$

For example "2000_50_AllTag" will be a reference to the pre-processing corresponding to the keeping of the 2000 most frequent words, with a minimum count of 50 to make bigrams and by keeping all kinds of words. On the contrary "2000" will reference to the pre-processing where we still keep the 2000 most frequent words, but with a minimum count of 5 to make bigrams and by keeping only the words tags listed above.

2.2 Encoding documents in vector format

Then, we need to encode all documents in a vector format that will be used as input for the algorithms. We have used two vector representations, bag of words (bow) and TF-IDF with L2 regularization (tf-idf-l2).

Bag of words. One of the simplest encoding is the *bag of words*, it's simply an histogram of the words present in each document. The output is a matrix $\mathbf{X} \in \mathbb{R}_+^{n \times t}$ where n is the number of documents and t the total number of words within all documents. x_{ij} is the number of times the word corresponding to the column j appears in the document i. We call this matrix the document-word matrix.

TF-IDF score. Another method, based on the *bag of words*, is the *TF-IDF score* that stands for *Term Frequency and Inverse Document Frequency*. This score is computed for each word in each document. It gives us in output a matrix $\mathbf{X}$ with the same properties than with the *bag of words* except that we multiply the frequency of the word in the document by a factor depending on the frequency of the word across all the documents like in Eq. 1:

$$x_{ij} = tf_{ij} \times \log \frac{N}{df_j} \tag{1}$$

Where tf_{ij} is the frequency of word j in document i. N the total number of documents. df_j is the number of documents containing the word j. Multiplying the second factor to the frequency of the word allows us to give more importance to word that are rare across all documents. Indeed, words that appear frequently across documents do not add much information to the model and therefore do not contribute to clustering them. Finally we can apply L2 normalization to make each vector unit length, i.e. its norm equals 1.

2.3 Word embedding: encoding words in vector format

Briefly, this set of methods consists in finding a vector representation for each word such that words that have related meaning are similar according to a measure, most of the time, the cosine similarity.

There are many various methods to get such an embedding such as SVD, Word2Vec [Mikolov et al., 2013], GloVe [Pennington et al., 2014] or more recently BERT [Devlin et al., 2018].

For example, Word2Vec is the representation of words as vectors in space, encapsulating their relations to each other, such as synonyms, antonyms or analogies. This representation is created using either Continuous Bag-of-Words (CBOW) or Skip-Gram. Both methods have different approach to the same issue: they create search to predict the missing word given a context (usually called window). As the model is trained using a neural network, the weights of the hidden layer encapsulate the relation between the context and the output word.

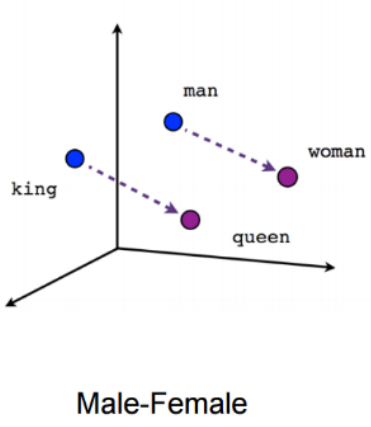


Figure 1: Example of relationship encapsulated in Word2Vec embedding.

Most of the word embedding methods, if not all, are based on the idea of context of a word such as described above.

In this work we have used the very popular and recent language model: BERT (Bidirectional Encoder Representations from Transformers). Briefly, this is a deep bidirectional neural network using the transformer block [Vaswani et al., 2017]. It involves principally two steps:

- **Mask Language Model**: Like Word2Vec, BERT tries to predict masked words of the sequence but in a more general way than Word2Vec. It not just predict one word (CBOW) or all except one (Skip-Gram) but a randomly set of words of the sequence (most of the time 15%).
- **Next Sentence Prediction**: In this step BERT learns the relationships between sequences by trying to predict the next one in the document.

Many pre-trained BERT models are available online and allow people to use this algorithm without the necessity of a somehow huge computation power (because of the training step). Moreover, we can fine-tune these pre-trained models to make them more accurate for our task, however, due to our computation and time limitations we were not able to fine-tune BERT with the datasets used in this work. Nevertheless, we were able to still have good results since the pre-trained model chosen was trained on a large corpus of various kind of texts (wikipedia).

2.4 Evaluate columns clustering

In order to evaluate the columns clustering we will use the notion of word embedding. Once we have such a representation we can measure similarity between words by using their vector representation and the cosine similarity for example.

The idea is then to compute TP, FP, TN et FN for a particular co-clustering. TP will corresponds to the number of times that two highly similar words (having a similarity higher than a chosen threshold α) are in the same column cluster. FP will be the number of times that two very different words are in the same cluster. TN the number of times that two very different words are not in the same cluster. Finally, FN is the number of times that two highly similar words are not in the same cluster.

We can then compute a score for the columns clustering:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

3 Experiments and Results

Dataset	#samples	#features	#classes	Balance	Vector format	Pre-processing
Classic3	3891	4982	3	0.71	tf-idf-l2	5000_50
Classic4	7092	4981	4	0.32	bow	5000_50_AllTag
NG5	4702	1980	5	0.95	tf-idf-l2	2000
NG20	18846	4964	20	0.63	tf-idf-l2	5000_50_AllTag
R8	7673	4983	8	0.01	bow	5000_50_AllTag
R40	11227	4983	46	4e-3	bow	5000_50_AllTag
R52	9099	4983	52	8e-4	bow	5000_50
WEBKB	4150	2000	4	0.31	bow	2000

Table 1: Description of the different datasets.

The vector format and pre-processing of each dataset is selected like explained in 2.1. We see that R8, R40 et R52 are highly imbalanced, indeed for example, for R8 the smallest class ("grain") has 51 samples whereas the biggest ("earn") has 3922.

For each dataset and algorithm we ran 10 experiments and display the mean and standard deviation in the Table below. For each experiment we have used 10 initializations, the best result according the optimized criterion is selected automatically by the *Coclust* package.

Dataset	**Metrics**	**CoclustInfo**	**CoclustMod**	**CoclustSpecMod**
Classic3	NMI	**0.94**	0.93	0.89
	ARI	**0.96**	**0.96**	0.92
	ACC	**0.99**	**0.99**	0.97
Classic4	NMI	0.59 ± 0.01	**0.75 ± 0.02**	0.63
	ARI	0.44 ± 0.01	**0.76 ± 0.02**	0.47
	ACC	0.66 ± 0.01	**0.91 ± 0.01**	0.70
NG5	NMI	**0.49 ± 0.01**	0.48 ± 0.01	0.46
	ARI	0.49 ± 0.03	**0.51 ± 0.01**	0.31
	ACC	0.72 ± 0.06	**0.77 ± 0.01**	0.67
NG20	NMI	**0.55 ± 0.01**	0.47 ± 0.02	0.47 ± 0.01
	ARI	**0.37 ± 0.02**	0.22 ± 0.01	0.18 ± 0.02
	ACC	**0.48 ± 0.02**	0.33 ± 0.03	0.33 ± 0.02
R8	NMI	0.52 ± 0.01	0.52 ± 0.03	**0.53**
	ARI	0.39 ± 0.04	**0.52 ± 0.03**	0.45
	ACC	0.50 ± 0.04	0.66 ± 0.02	**0.69**
R40	NMI	**0.43 ± 0.01**	0.40 ± 0.01	0.37 ± 0.01
	ARI	0.21 ± 0.04	**0.50 ± 0.02**	0.15 ± 0.02
	ACC	0.27 ± 0.03	**0.58 ± 0.01**	0.36 ± 0.02
R52	NMI	0.50 ± 0.01	**0.51 ± 0.02**	0.48 ± 0.02
	ARI	0.14 ± 0.02	**0.53 ± 0.02**	0.34 ± 0.06
	ACC	0.24 ± 0.02	**0.61 ± 0.01**	0.49 ± 0.07
WEBKB	NMI	**0.38 ± 0.01**	0.37 ± 0.01	0.37
	ARI	0.38 ± 0.03	**0.40 ± 0.02**	0.32
	ACC	0.65 ± 0.02	**0.67 ± 0.02**	0.64

Table 2: Metrics of the algorithms on the different datasets for the **rows/individuals** clustering.

Overall we can see in Table 2 that CoclustMod seems to be the best algorithm most of the time, whereas CoclustSpecMod struggles to outperform the other algorithms and finally that CoclustInfo seems to stand between the two latter algorithms.

The results are more or less the same than those that we can find in the original papers of the algorithms in [Ailem et al., 2015] and [Labiod and Nadif, 2011] for the datasets used in common here and in these papers. The fact that we can retrieve these results validate, once again, our pre-processing step.

We can see here that, compared to the other algorithms, CoclustInfo seems to not perform well on the highly imbalanced datasets such as R8, R40 et R52 when we compare in terms of ARI and ACC. However it has good values for NMI, it can make sense since CoclustInfo optimizes the mutual information

criterion related to the NMI (even if the NMI of the table above used the real labels whereas CoclustInfo does not). That is CoclustMod that performs the best for this kind of dataset.

Here after we show the results for the columns clustering with the method explained in 2.4:

Dataset	**Metrics**	**CoclustInfo**	**CoclustMod**	**CoclustSpecMod**
Classic3	$\alpha = 0.9$	**0.66**	**0.66**	**0.66**
	$\alpha = 0.75$	**0.60**	0.59	**0.60**
	$\alpha = 0.5$	**0.40**	0.38	0.39
Classic4	$\alpha = 0.9$	**0.74**	**0.74**	0.64
	$\alpha = 0.75$	**0.63**	0.62	0.59
	$\alpha = 0.5$	0.32	0.30	**0.41**
NG5	$\alpha = 0.9$	0.76 ± 0.01	**0.79**	0.75
	$\alpha = 0.75$	**0.62**	0.61	**0.62**
	$\alpha = 0.5$	0.25 ± 0.01	0.23	**0.26**
NG20	$\alpha = 0.9$	**0.94**	0.83 ± 0.01	0.79 ± 0.01
	$\alpha = 0.75$	**0.71**	0.66 ± 0.01	0.66
	$\alpha = 0.5$	0.10	0.19 ± 0.01	**0.23 ± 0.01**
R8	$\alpha = 0.9$	**0.84 ± 0.01**	0.76 ± 0.02	0.70
	$\alpha = 0.75$	**0.68 ± 0.01**	0.65 ± 0.01	0.62
	$\alpha = 0.5$	0.20 ± 0.01	0.27 ± 0.02	**0.33**
R40	$\alpha = 0.9$	**0.97**	0.79 ± 0.01	0.84 ± 0.03
	$\alpha = 0.75$	**0.72**	0.64 ± 0.01	0.67 ± 0.01
	$\alpha = 0.5$	0.08	**0.25 ± 0.01**	0.20 ± 0.03
R52	$\alpha = 0.9$	**0.97**	0.84 ± 0.01	0.80 ± 0.03
	$\alpha = 0.75$	**0.74**	0.68	0.67 ± 0.02
	$\alpha = 0.5$	0.08	0.20 ± 0.01	**0.24 ± 0.03**
WEBKB	$\alpha = 0.9$	0.72 ± 0.01	**0.74 ± 0.01**	0.66
	$\alpha = 0.75$	0.66 ± 0.01	**0.67 ± 0.01**	0.62
	$\alpha = 0.5$	0.30 ± 0.01	0.29 ± 0.01	**0.36**

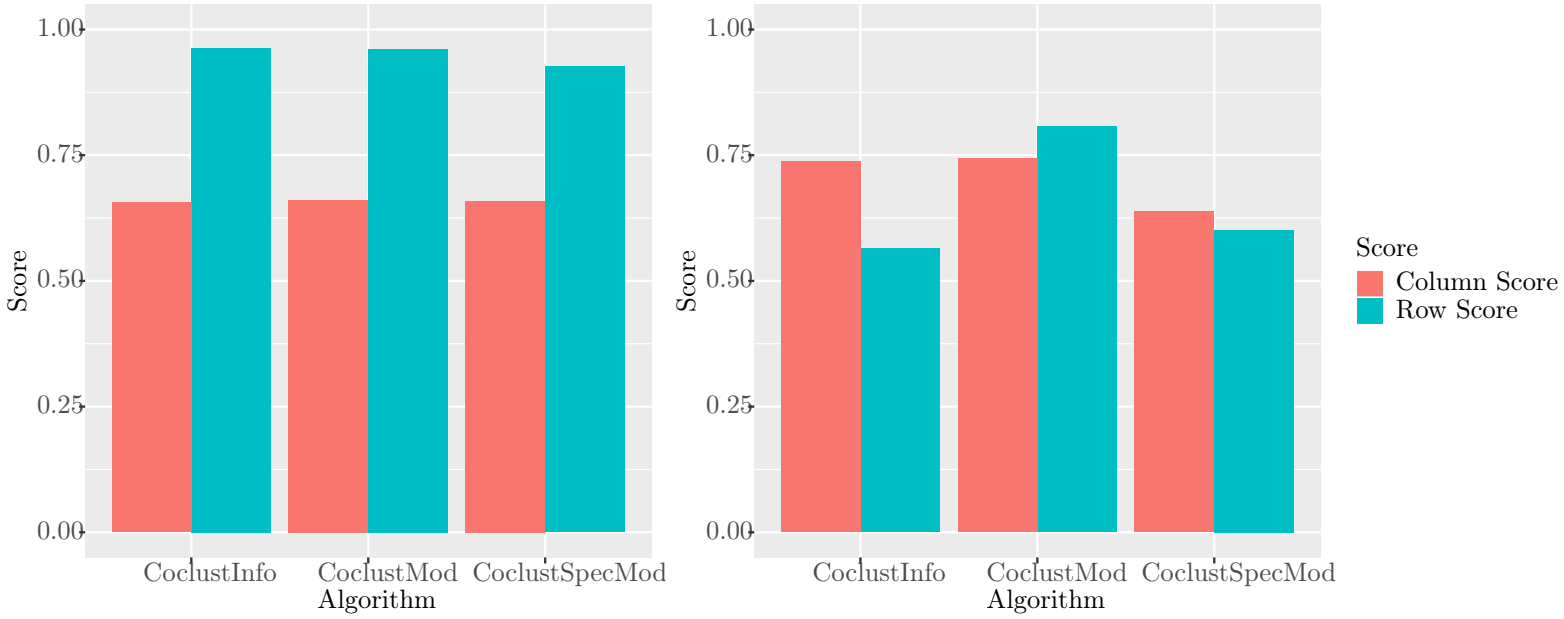
Table 3: Metrics of the algorithms on the different datasets for the **columns/variables** clustering.

Interestingly, the results tend to decrease systematically as α decreases, this behaviour will be discussed in 4.

Here, when we evaluate the columns clustering, it is CoclustInfo that is the best algorithms most of the time, with CoclustMod being less the best algorithm and CoclustSpecMod between the last two algorithms.

It is interesting to note that CoclustSpecMod did not performs very well on the rows clustering compared to the two others whereas it performs relatively well on the columns clustering.

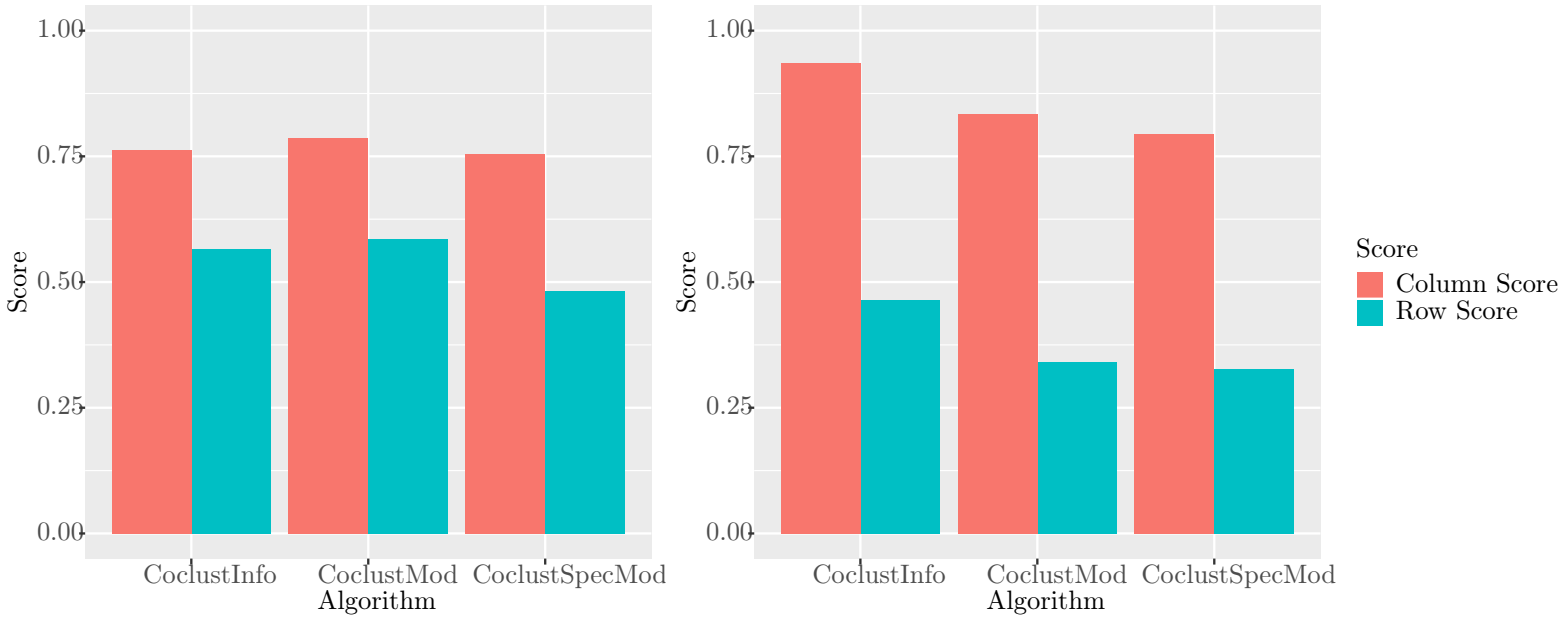
The following figures compare the performance of algorithms for rows and columns clustering in a more compact and visible way. We establish a score for the rows clustering, being the average of the three metrics used: NMI, ARI and clustering accuracy, indeed, each of these metrics tend to capture different information of the performance and we want an algorithm maximizing all of them. For the score of the columns clustering we simply get the higher value for each dataset and algorithm, here is when we choose a threshold $\alpha = 0.9$.



(a) Classic3.

(b) Classic4.

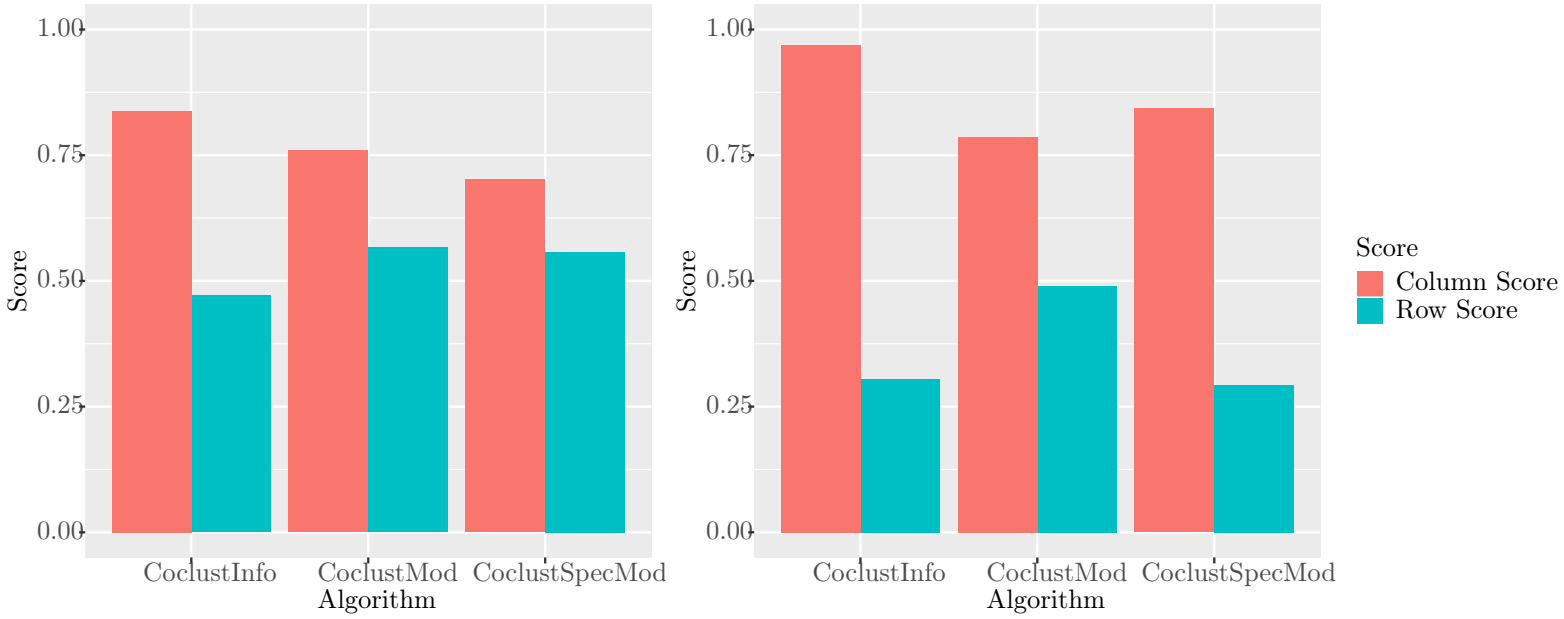
Figure 2: Comparison of row and columns results for Classic3 and Classic4 data set.



(a) NG5.

(b) NG20.

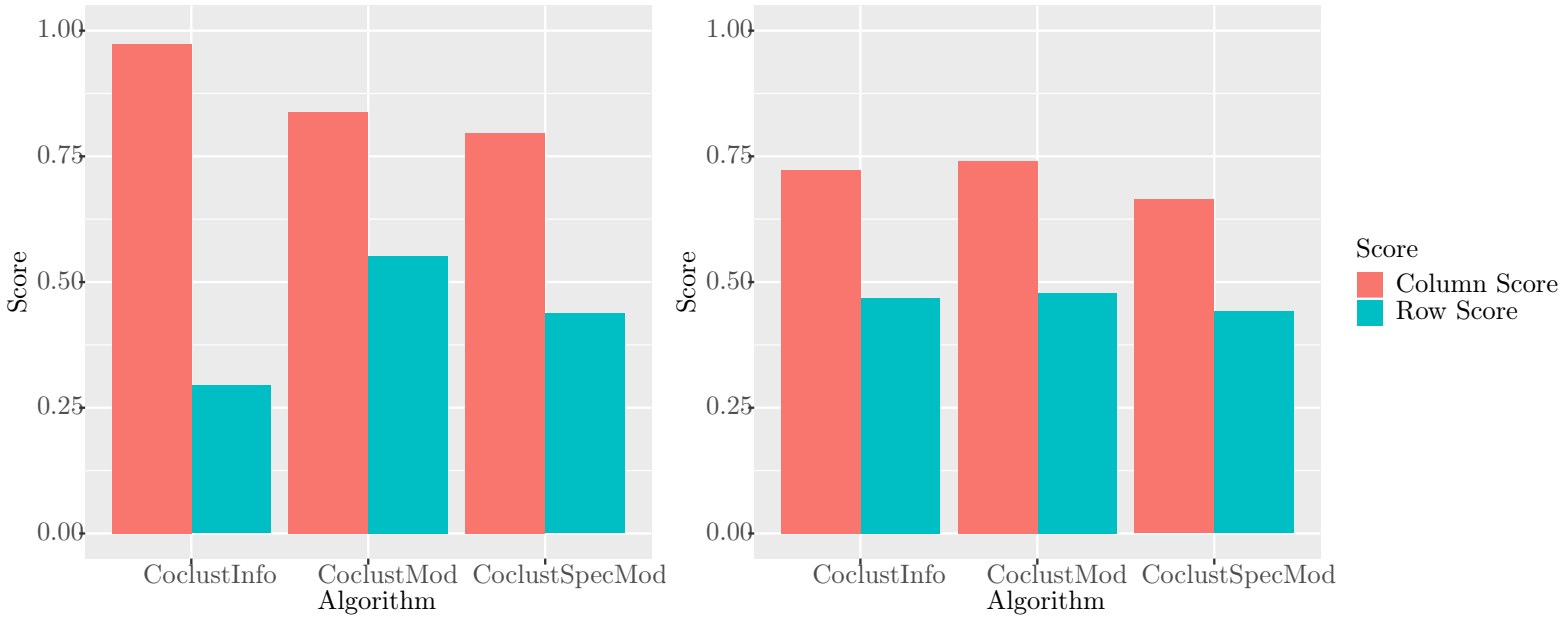
Figure 3: Comparison of row and columns results for NG5 and NG20 data set.



(a) R8.

(b) R40.

Figure 4: Comparison of row and columns results for R8 and R40 data set.



(a) R52.

(b) WEBKB.

Figure 5: Comparison of row and columns results for R52 and WEBKB data set.

We can see in all of these figures that the column score is not necessarily correlated with the row score. For example, for the R40 dataset, CoclustInfo performs very good for the columns clustering but very bad for the rows clustering ; at the contrary CoclustMod has more centered score values for this dataset. Moreover, we can't define a general behaviour of the relation between the row score and the column score for a particular algorithm, it always depends of the dataset used.

Here after we show the worst FP words pairs for each cluster that is the words that are in the same cluster but are very different according the chosen similarity. We also show the FN words pairs, i.e. the similar words being not in the same cluster.

Column Cluster 1			Column Cluster 2			Column Cluster 3		
autosomal	thiazide	0.17	accessibility	twentyfive	0.22	farnborough	rm	0.23
gen	methotrexate	0.18	assist	wiswesser	0.22	ax	farnborough	0.23
lymphoma	seventeen	0.19	assist	substructure	0.23	farnborough	injection	0.23
autosomal	ohp	0.19	accessibility	twentytwo	0.23	axially	farnborough	0.24
blindness	hitherto	0.20	assist	twentyfive	0.24	behave	millisecond	0.24
articulation	thiazide	0.20	assist	uncontrolled	0.25	farnborough	longitudinal	0.24
articulation	ohp	0.20	aimtwx	assist	0.25	farnborough	sec	0.24
articulation	hitherto	0.20	draw	stepbystep	0.25	farnborough	slab	0.24
fourteen	occipital	0.20	assist	twentytwo	0.25	farnborough	rotary	0.24
angiocardiography	hitherto	0.20	assist	stepbystep	0.25	behave	farnborough	0.24
occipital	thirteen	0.20	assist	elaboration	0.26	farnborough	von	0.25
chlorothiazide	gen	0.20	assist	kuhn	0.26	duct	farnborough	0.25
autosomal	water	0.20	assist	syntactic	0.26	farnborough	rev	0.25
autosomal	tloop	0.20	assist	pragmatic	0.26	farnborough	mix	0.25
azathioprine	gen	0.20	draw	twentyfive	0.26	farnborough	transverse	0.25
autosomal	minute	0.20	assist	ifla	0.26	farnborough	fig	0.25
occipital	seventeen	0.20	accessibility	later	0.26	downward	farnborough	0.25
fourteen	lymphoma	0.21	accessibility	arist	0.27	farnborough	millisecond	0.25
autosomal	twelve	0.21	draw	wiswesser	0.27	farnborough	presence	0.25
lymphoma	twelve	0.21	assist	heretofore	0.27	determine	farnborough	0.25
lymphoma	thirteen	0.21	assist	terminological	0.27	cease	millisecond	0.25

Table 4: Worst FP predictions pairs of words and similarity for each cluster for Classic3 with CoclustInfo.

We see in the words pairs results for each dataset and in Table 4 that there is often a word having low similarity with a lot of words in the cluster. For example, for the column cluster 2 for Classic3 (see the table above), the word *assist* has low similarity values with many words of the cluster, same with cluster 1 and *autosomal* and the cluster 3 with the word *farnborough* that is actually a city so it makes sense to have low similarity with other words.

behavioral traumatic 0.95	apparent markedly 0.95	occasionally sometimes 0.95	significantly strictly 0.95	significantly substantially 0.95	inherent intrinsic 0.95	allergic carrier 0.95	illustrate illustrates 0.95	defence defense 0.95

greatly significantly 0.95	eighteen thirteen 0.96	prevent prevents 0.96	seventeen sixteen 0.96	significant significantly 0.96	focal reactive 0.96	fifteen fourteen 0.96	considerable considerably 0.96	considerably significantly 0.96

nineteen seventeen 0.96	eighteen fourteen 0.96	fourteen sixteen 0.96	allows enables 0.96	apparently obviously 0.96	eighteen twelve 0.96	eighteen seventeen 0.97	almost nearly 0.97	considerably greatly 0.97	practically virtually 0.97

Table 5: Worst FN predictions pairs of words and cosine similarity for Classic3 with CoclustInfo.

We can see that the BERT embedding was pertinent for our case, when we look at Table 5, it manages to have high similarity for related words such as number written in letters (these words could have been removed during the pre-processing by the way), or words such as *occasionally* and *sometimes*, *considerably* and *greatly* etc. Unfortunately CoclustInfo did not manages to put these similar words together in a same cluster.

When we look at Table 4 we see that the BERT embeddings are once again pertinent, with words that are different having a low similarity values, however, as we can see there is no negative similarity between words.

This analysis was not done with the datasets having too much clusters, for example R52 has 52 classes making it very difficult or impossible to do that kind of analysis.

4 Discussion

4.1 Choice of α

As we saw in Table 3 the metric values for the columns clustering evaluation tend to increases as α increases. To explain this we need to analyze how TP, FP, TN and FN change when α changes.

At first we will define these four metrics formally:

$$TP = \sum_{i}^{s} \sum_{(w1,w2)\in P_i} 1_{\{sim(w1,w2)\geq\alpha\}} \tag{3}$$

i.e. putting in the same cluster two similar words.

$$FP = \sum_{i}^{s} \sum_{(w1,w2)\in P_i} 1_{\{sim(w1,w2)<\alpha\}} \tag{4}$$

i.e. putting in the same cluster two different words.

$$TN = \sum_{(w1,w2)\in P_{not}} 1_{\{sim(w1,w2)<\alpha\}} \tag{5}$$

i.e. putting in different clusters two different words.

$$FN = \sum_{(w1,w2) \in P_{not}} 1_{\{sim(w1,w2) \geq \alpha\}} \tag{6}$$

i.e. putting in different clusters two similar words.

With s the number of columns clusters ; P_i all the pair of words of the column cluster i ; P_{not} all the pair of words being not in the same column cluster ; $sim(w1, w2)$ the similarity measure of words $w1$ and $w2$ and $1_{condition}$ being 1 if the condition is true else 0. We have used the cosine similarity as a similarity measure, therefore $\alpha \in [-1, 1]$ since it is the range of the cosine similarity.

Naturally, when we increase α, TP and FN decrease while FP and TN increase. This behaviour can be found formally with the four equations above, or empirically, for example, we display the evolution of these four metrics according α for the Classic3 dataset below:

	$\alpha = 0.5$	$\alpha = 0.75$	$\alpha = 0.9$
TP	3,905,450	1,155,568	10,025
FN	7,121,892	1,904,452	15,019
TN	1,025,225	6,245,665	8,132,098
FP	355,104	3,104,986	4,250,526

Table 6: Evolution of TP, TN, FP and FN according α for Classic3.

When α changes, TP increases as much as FP decreases (and vice versa) because the total number of positives does not change, same statement for TN and FN, making the denominator of Eq. 2 constant as α changes, this is the total number of pairs of words. To analyze the behaviour of this equation regarding α we can therefore only relate on the numerator (that is $TP + TN$).

When α is maximal, TN is at its maximum whereas TP is at its minimum and vice versa when α is minimal. Thus, if there are more negatives than positives the score of Eq. 2 will be maximized when α is maximal. If there are more positives than negatives the score will be maximized when α is minimal.

The total number of pairs of words of the corpus is:

$$\frac{d(d-1)}{2} \tag{7}$$

With d the total number of words.

$$P = TP + FP = \sum_{i}^{s} \frac{d_i(d_i - 1)}{2} \tag{8}$$

i.e. the number of pair of words for each cluster.

With d_i the number of words of the column cluster i and s the total number of column cluster. Moreover we have $d = \sum_i^s d_i$.

$$N = TN + FN = \frac{d(d-1)}{2} - P \tag{9}$$

i.e. the total number of pairs of words minus the pairs of words belonging to a same cluster.

As α increases, the score will always increases if $N > P$ or always decreases if $N < P$.

Empirically we saw that the score increases as α increases meaning that it must be more negatives than positives in our case. This can be verified for Classic3 with Table 6 and it is right for the other datasets.

However, we found some cases where this was not true, for example, with $s = 2$, $d_1 = 5000$ and $d_2 = 1000$ then $N < P$. We found, empirically, that when the clusters are not well balanced there will be less negatives than positives and that the more the clusters are balanced the more the number of negatives is higher than the number of positives. Here after we show mathematically that when the clusters are perfectly balanced there will be more negatives than positives:

Proposition 1. When the columns clusters are perfectly balanced then $N > P$ (meaning that Eq. 2 is maximized when α is maximum).

Proof. If clusters are perfectly balanced we have $\forall i \; d_i = d/s$.

With the property above and from the definition of P of Eq. 8:

$$\begin{aligned} P &= \sum_{i}^{s} \frac{d_i(d_i - 1)}{2} \\ &= \sum_{i}^{s} \frac{d/s(d/s - 1)}{2} \\ &= s\frac{d/s(d/s - 1)}{2} \\ &= \frac{d(d/s - 1)}{2} \end{aligned} \tag{10}$$

Then, with this new formulation of P and from the definition of N of Eq. 9:

$$\begin{aligned} N &> P \\ \frac{d(d-1)}{2} - P &> P \\ \frac{d(d-1)}{2} &> 2P \\ \frac{d(d-1)}{2} &> d(\frac{d}{s} - 1) \\ \frac{(d-1)}{2} &> \frac{d}{s} - 1 \\ \frac{d}{2} - \frac{1}{2} &> \frac{d}{s} - 1 \\ \frac{d}{2} + \frac{1}{2} &> \frac{d}{s} \end{aligned} \tag{11}$$

The last inequality is true if $s \geq 2$ i.e. we have at least 2 columns clusters.

End of proof.

To conclude this part, the parameter α seems to be biased and will gives the best results for Eq. 2 at its maximum or its minimum depending on which between the positives and the negatives have the highest cardinality. Perhaps that adding a second parameter β, lower than α, to compute FP and TN can be a solution to this problem.

5 Conclusion

We see through this work an interesting method to evaluate the columns clustering of a co-clustering. Even if we also saw that the parameter α will gives the best results depending on the cardinality of the positives and the negatives. Some experiments with a second parameter β to compute FP and TN could be interesting to do.

As future works, it could have been interesting to use other word embedding methods and compare them with each other, moreover fine-tune the pre-trained BERT model could improve the results, this improvement could be measured with the metrics used in this work and see how much fine-tuning is important in our case. Moreover CoclustInfo is a non-diagonal co-clustering algorithm allowing to have different number of rows clusters and columns clusters, this property could be used in order to obtain more meaningful columns clusters.

Finally, our github repository is open source and free to use[1].

References

[Ailem et al., 2015] Ailem, M., Role, F., and Nadif, M. (2015). Co-clustering document-term matrices by direct maximization of graph modularity.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Dhillon et al., 2003] Dhillon, I. S., Mallela, S., and Modha, D. S. (2003). Information-theoretic co-clustering. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98, New York, NY, USA. ACM.

[Labiod and Nadif, 2011] Labiod, L. and Nadif, M. (2011). Co-clustering for binary data with maximum modularity. In Lu, B.-L., Zhang, L., and Kwok, J., editors, *Neural Information Processing*, pages 700–708, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[Role et al., 2019] Role, F., Morbieu, S., and Nadif, M. (2019). Coclust: A python package for co-clustering. *Journal of statistical software*, 88:1–29.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[1] https://github.com/MatPont/Text_Mining_Project