Rapport de RODM

Antonio Tavares, Mohamed Salah MIMOUNA, Julien Dallot

30 avril 2022

1 Jeux de données

— `ecoli.txt` : 7 attributs par instance, 327 instances, 5 classes. La classe regroupant le nombre minimal d'instances en possède 20, le choix a été fait de supprimer les instances étant dans des classes de petites tailles ($\leq$ 5 instances par classe).
— `prnn.txt` : 2 attributs par instance, 250 instances, 2 classes. Autant d'instances dans chaque classe (125).

Ces deux jeux de données proviennent du site `www.openml.org`.

2 Résultats : `main()`

La fonction `main()` n'utilise pas de regroupement, les performances sur les 5 jeux de données considérés sont les suivantes :

	Séparation	Temps (s)	gap	Erreurs train/test
D = 2	Univarié	1.3s	0.0%	5/1
	Multivarié	0.9s	0.0%	1/0
D = 3	Univarié	8.0s	0.0%	0/4
	Multivarié	0.8s	0.0%	0/2
D = 4	Univarié	10.5s	0.0%	0/4
	Multivarié	4.2s	0.0%	0/1

TABLE 1 – Résultats jeu de données `iris` (train size 120, test size 30, features count : 4)

	Séparation	Temps (s)	gap	Erreurs train/test
D = 2	Univarié	12.4s	0.0%	10/2
	Multivarié	1.4s	0.0%	0/2
D = 3	Univarié	30.2s	3.7%	6/2
	Multivarié	3.2s	0.0%	0/1
D = 4	Univarié	30.5s	7.7%	11/2
	Multivarié	18.9s	0.0%	0/2

TABLE 2 – Résultats jeu de données `seeds`

	Séparation	Temps (s)	gap	Erreurs train/test
D = 2	Univarié	14.5s	0.0%	5/2
	Multivarié	0.3s	0.0%	0/2
D = 3	Univarié	27.1s	3.7%	0/2
	Multivarié	1.2s	0.0%	0/1
D = 4	Univarié	18.9s	7.7%	0/1
	Multivarié	3.7s	0.0%	0/3

TABLE 3 – Résultats jeu de données `wine`

	Séparation	Temps (s)	gap	Erreurs train/test
D = 2	Univarié	30.1s	6.6%	42/15
	Multivarié	30.1s	5.6%	33/15
D = 3	Univarié	30.4s	28.6%	58/20
	Multivarié	30.3s	7.0%	16/13
D = 4	Univarié	31.1s	54.4%	83/26
	Multivarié	31.1s	137.7%	151/33

TABLE 4 – Résultats jeu de données `ecoli`

	Séparation	Temps (s)	gap	Erreurs train/test
D = 2	Univarié	30.1s	7.4%	24/12
	Multivarié	30.1s	10.5%	19/7
D = 3	Univarié	30.3s	10.5%	19/11
	Multivarié	30.5s	13.0%	22/5
D = 4	Univarié	30.7s	11.1%	20/12
	Multivarié	30.6s	22.7%	35/6

TABLE 5 – Résultats jeu de données `prnn`

On remarque que le merge simple permet d'obtenir des bons résultats (un gap faible) en un temps réduit pour des petites instances notamment dans le cas d'une séparation multivarié et avec des faibles erreurs. Néanmoins ceci n'est plus

le cas pour des données avec des tailles plus importants comme "ecoli" et "prnn" ou on observe un temps d'éxecution plus important un gap plus important en comparant par rapport aux instances "iris", "seeds" et "wine". Aussi pour des données avec une taille plus grande et plus de features (le cas d'"ecoli et "prnn") on observe plus d'erreurs dans les résultats.
Afin d'essayer d'améliorer les résultats obtenus on a essayé d'implémenter et comparer les résultats obtenues pour deux méthodes de clustering : Kmeans et DBSCAN.

3 Question d'ouverture

Nous avons décidé de tester d'autres méthode de clustering pour les comparer aux méthodes déjà implémentées ; nous avons implémenter les méthodes classiques DBscan et Kmean, bien documentées sur internet.

Avec l'algorithme KMean. Nous avons implémenté l'algorithme en initialisant les k premiers centroïdes avec k données prises au hasard. Pour chacun des tests suivants, nous avons fait tourné l'algorithme avec k variant de 3 à 30 ; nous n'affichons que le meilleur résultat ainsi que sa valeur du paramètre k correspondante.

	Séparation	Temps (s)	k meilleur résultat	gap	Erreurs train/test
D = 2	Univarié	0.4	24	0.0%	5/1
	Multivarié	0.3s	26	0.0%	3/1
D = 3	Univarié	1.4s	23	0.0%	4/1
	Multivarié	0.2s	20	0.0%	2/1
D = 4	Univarié	6.3s	19	0.0%	3/1
	Multivarié	0.5s	25	0.0%	3/1

TABLE 6 – Résultats pour KMean sur le jeu de données `iris`

	Séparation	Temps (s)	k meilleur résultat	gap	Erreurs train/test
D = 2	Univarié	1.5	43	0.0%	15/4
	Multivarié	0.1s	39	0.0%	10/3
D = 3	Univarié	9.2s	35	0.0%	13/4
	Multivarié	1.3s	35	0.0%	8/4
D = 4	Univarié	10.1s	39	10.5%	12/5
	Multivarié	1.2s	40	0.0%	7/3

TABLE 7 – Résultats pour KMean sur le jeu de données `seeds` - k varie de 3 à 50 pour de meilleurs résultats

L'algorithme Kmean que nous avons codé ne prend pas en compte les classes d'entraînement stockées dans le vecteur y. Cela se constate clairement en ce

que les résultats sont globalement moins bons que l'approche naïve avec entraînement. Le nombre d'erreurs supplémentaire est néanmoins borné par une constante raisonnable dans nos essais – il n'est jamais plus grand que 50% de la valeur calculée par l'algorithme naïf. On testant l'algorithme dans la bonne plage de valeurs de k, on s'assure donc un résultat de qualité raisonnable en un temps de calcule bien plus court, avec un gap quasi systématiquement égal à 0.