IFT3700

Devoir 1

Rapport d’analyse la performance des algorithmes

Xuanchen Liu – matricule : 20173286

Van Nam Vu – matricule : 20170148

21 novembre 2021

DATASET : MNIST

Prétraitement : En inspirant du code des démos (Démo 1) de Mathieu, on a converti les données en 0 et 1 pour faciliter le calcul et cela nous aide à voir plus clairement si 2 images sont différentes

Notion de similarité

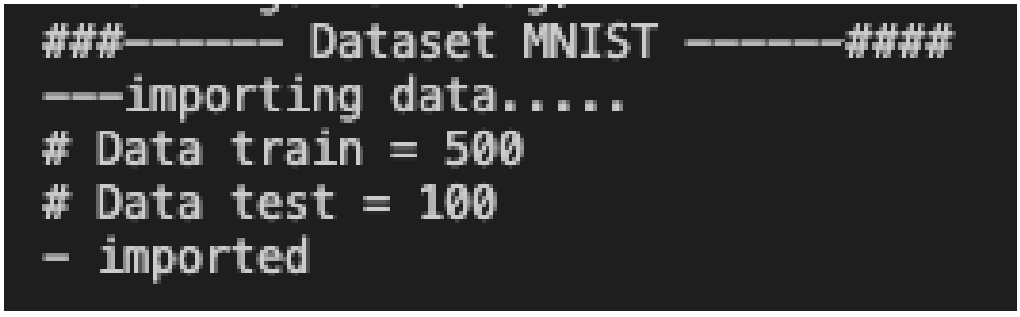
1) Calculer la dis-similarité moyenne (différence moyenne) entre 2 images après avoir appliquer une légère translation. Les connaissances à priori du jeu de données utilisé sont : format pixel 28x28 (avec la densité de 0 à 255), ce sont des images, presque tous les images sont au centre du carré 28x28.
2) Pour chaque paire d'image, mettons X1 et X2, on procède la translation (**1 pixel gauche-droite** et **1 pixel haut-bas**) et calcule la dis-similarité:
   i) Reformer le format des images à 28x28 pour faciliter la translation
   ii) On fait le tour : Pour X1, on fait **la translation à gauche** (enlève la 1ère colonne de la matrice 28x28) et X2, on fait **la translation à droite** (enlève la dernière colonne de la matrice 28x28). Et puis on calcule la **différence entre 2 images en faisant la soustraction des matrices après la translation**. Et on prend la moyenne des différences (appelons **moyenne_1**). On applique le même principe avec X1 à droite et X2 à gauche, alors on aura **moyenne_2.** Et puis on fait la translation en haut et en bas. Pour X1, on fait **la translation en haut** (enlève la 1ère ligne de la matrice 28x28) et X2, on fait **la translation en bas** (enlève la dernière ligne de la matrice 28x28). Et puis on calcule la différence entre 2 images en faisant la soustraction des matrices après la translation. Et on prend la moyenne des différences (appelons **moyenne_3**). On applique le même principe avec X1 à droite et X2 à gauche, alors on aura **moyenne_4.**

iii) Maintenant on prend **la somme des moyenne_1, moyenne_2, moyenne_3 et moyenne 4** pour avoir une valeur de dis-similarité globale pour utiliser dans les algorithmes.

Motivation de la similarité : Pour le jeu de donné MNIST, comme ce sont des images en format pixel 28x28, et représentent sous forme une matrice de 1x784. Notre motivation est de faire une légère translation et regarder la dis-similarité entre 2 images après la translation. Elle se fait à gauche et à droite, puis en haut et en bas, pour avoir une vue globale comment 2 images se ressemblent. Comme selon l'indice du l'énoncé, une légère translation d'n ne devra pas affecter sa similarité. Donc on a inspiré pour faire la translation, et comme ce sont des données numériques, ça facilite les calculs de différence et moyenne ☺

Analyse

Comme notre jeu de donnée contient 60000 images pour s'entraîner et 10000 images, notre premier but est de maximiser l'exactitude. Mais vu que la capacité de notre ordinateur ne permet pas de prendre tout le jeu de données, on a décidé de prendre **500 données d'entrainement** et **100 données de test** (avec le temps d'exécution de 3-4 min). On est bien conscient que ce fait peut réduire effectivement la qualité du projet. Le temps d'exécution est un peu plus long car on doit aussi calculer la matrice de distance euclidienne pour comparer les performances !



1) Algorithme KNN

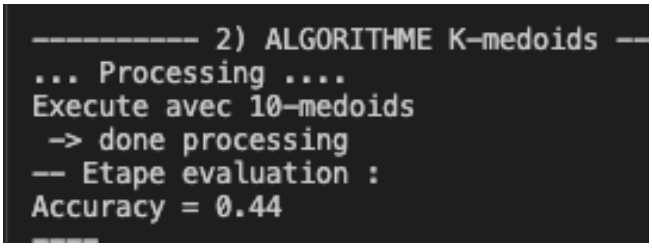
```
---------- 1) ALGORITHME KNN ----------
... Processing ....
Accuracy = 0.77 pour k = 1
____
Accuracy = 0.74 pour k = 2
____
Accuracy = 0.74 pour k = 3
____
Accuracy = 0.74 pour k = 4
____
Accuracy = 0.74 pour k = 5
____
Accuracy = 0.75 pour k = 6
____
Accuracy = 0.75 pour k = 7
____
Accuracy = 0.74 pour k = 8
____
Accuracy = 0.74 pour k = 9
____
Accuracy = 0.74 pour k = 10
____
Accuracy = 0.74 pour k = 11
____
Accuracy = 0.74 pour k = 12
____
Accuracy = 0.75 pour k = 13
____
Accuracy = 0.76 pour k = 14
____
Accuracy = 0.74 pour k = 15
____
Accuracy = 0.75 pour k = 16
____
Accuracy = 0.72 pour k = 17
____
Accuracy = 0.73 pour k = 18
____
Accuracy = 0.73 pour k = 19
____
 -> done processing
Accuracy max = 0.77
avec k = 1
```

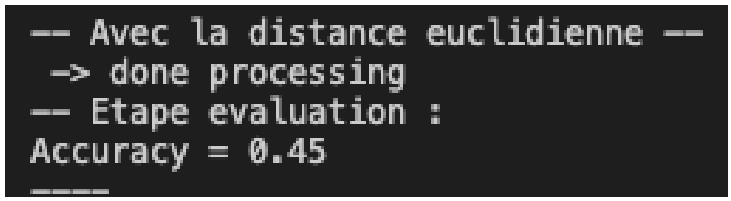
```
-- Avec la distance euclidienne --
Accuracy = 0.76 pour k = 1
____
Accuracy = 0.72 pour k = 2
____
Accuracy = 0.75 pour k = 3
____
Accuracy = 0.78 pour k = 4
____
Accuracy = 0.78 pour k = 5
____
Accuracy = 0.78 pour k = 6
____
Accuracy = 0.76 pour k = 7
____
Accuracy = 0.74 pour k = 8
____
Accuracy = 0.74 pour k = 9
____
Accuracy = 0.74 pour k = 10
____
Accuracy = 0.73 pour k = 11
____
Accuracy = 0.75 pour k = 12
____
Accuracy = 0.77 pour k = 13
____
Accuracy = 0.74 pour k = 14
____
Accuracy = 0.75 pour k = 15
____
Accuracy = 0.75 pour k = 16
____
Accuracy = 0.75 pour k = 17
____
Accuracy = 0.74 pour k = 18
____
Accuracy = 0.74 pour k = 19
____
 -> done processing
Accuracy max = 0.78
avec k = 4
```

En regardant les résultats, on constate qu'avec notre notion de similarité, on a un ratio d'exactitude de 77%. Et avec la distance euclidienne, ce ratio est un peu plus haut (78%). Il me semble que la distance euclidienne est plus juste, car avec le nombre de k = 4, ce ne sera

ni trop large ni trop petit pour trouver le chiffre le plus ressemble. Pendant qu'avec notre notion de similarité, on a k = 1 pour avoir l'exactitude maximale. Mais ce fait peut causer la sous-estimer si on travaille avec un jeu de données plus large. La caractéristique de KNN est de regarder k plus proche et déterminer la classe de cet élément. Alors si on regarde juste 1 voisin à côté, peut-être ce ne sera pas suffisant et comme les chiffres sont écrites assez semblable, on peut tromper facilement. Si on prend k = 6, avec un ratio d'exactitude un peu plus bas de 75%, mais pour long terme ce sera plus stable ! Si 2 mêmes chiffres sont écrits de façon assez semblable, alors notre notion est plus avantageuse car on doit vérifier juste un voisin plus proche.

2) Algorithme K-médoids





En observant les résultats, c'est clair que notre notion de similarité n'est pas efficace pour la partition. Comme notre notion de similarité est calculé par la différence des légères translations, alors cette valeur n'est pas assez significative pour qu'on distingue entre les chiffres pour faire la partition. En plus, selon l'algorithme de k-médoids, les centroids seront recalculé et les éléments vont prendre le centroid le plus proche. Alors par exemple les chiffre 5 et 2 sont ressemblés, ou 1 et 7, ou 8 et 0 et 9, et avec juste 500 chiffres, on ne sait pas la proportion des chiffres donc la classe finale de centroid peut être affecter. Et ce fait peut changer l'exactitude de la prédiction.

3) Algorithme Isomap et KNN

```
... Processing isomap ....
Reduit dimentionnel à 8
 -> done processing
-- Classification avec KNN --
Accuracy = 0.67 pour k = 1
-*-
Accuracy = 0.61 pour k = 2
-*-
Accuracy = 0.63 pour k = 3
-*-
Accuracy = 0.63 pour k = 4
-*-
Accuracy = 0.64 pour k = 5
-*-
Accuracy = 0.62 pour k = 6
-*-
Accuracy = 0.64 pour k = 7
-*-
Accuracy = 0.63 pour k = 8
-*-
Accuracy = 0.65 pour k = 9
-*-
Accuracy = 0.64 pour k = 10
-*-
Accuracy = 0.65 pour k = 11
-*-
Accuracy = 0.64 pour k = 12
-*-
Accuracy = 0.64 pour k = 13
-*-
Accuracy = 0.62 pour k = 14
-*-
Accuracy = 0.62 pour k = 15
-*-
Accuracy = 0.63 pour k = 16
-*-
Accuracy = 0.63 pour k = 17
-*-
Accuracy = 0.6 pour k = 18
-*-
Accuracy = 0.63 pour k = 19
-*-
Accuracy max = 0.67
avec k = 1
```

```
-- Avec la distance euclidienne --
 -> done processing
-- Classification avec KNN --
Accuracy = 0.66 pour k = 1
-*-
Accuracy = 0.61 pour k = 2
-*-
Accuracy = 0.65 pour k = 3
-*-
Accuracy = 0.63 pour k = 4
-*-
Accuracy = 0.64 pour k = 5
-*-
Accuracy = 0.66 pour k = 6
-*-
Accuracy = 0.66 pour k = 7
-*-
Accuracy = 0.66 pour k = 8
-*-
Accuracy = 0.63 pour k = 9
-*-
Accuracy = 0.65 pour k = 10
-*-
Accuracy = 0.63 pour k = 11
-*-
Accuracy = 0.64 pour k = 12
-*-
Accuracy = 0.65 pour k = 13
-*-
Accuracy = 0.64 pour k = 14
-*-
Accuracy = 0.65 pour k = 15
-*-
Accuracy = 0.64 pour k = 16
-*-
Accuracy = 0.63 pour k = 17
-*-
Accuracy = 0.63 pour k = 18
-*-
Accuracy = 0.62 pour k = 19
-*-
Accuracy max = 0.66
avec k = 1
```

Ici on décide de réduit la dimension à 8, au lieu de 784 dimensions au début. Ce fait peut nous aider à réduit le coût en espace mémoire, le temps de calcul et des fois pour améliorer la qualité des modèles d'apprentissage. Mais ce n'est pas dans ce cas que la qualité soit augmentée, mais plutôt inverse ! Avec la caractéristique des données et avec notre fonction de similarité, ça demande beaucoup de « détails » pour mieux comparer et classer les images. On peut voir avec KNN sans réduite dimensionnelle, on a un taux d'exactitude de 76%, mais avec la réduite dimensionnelle et puis KNN, ce taux a baissé à. 67%. Donc on a perdu la précise en réduisant la dimension. C'est à peu près la même chose si on utilise la distance euclidienne. Déjà on a choisi de réduit à 8, mais ça perd quand même beaucoup de précision. En comparant rapidement avec la distance euclidienne, notre notion de similarité devient un peu plus avantageuse.

4) Algorithme PCoA et KNN

```
Reduit dimentionnel à 8
 -> done processing
-- Classification avec KNN --
Accuracy = 0.75 pour k = 1
-*-
Accuracy = 0.76 pour k = 2
-*-
Accuracy = 0.77 pour k = 3
-*-
Accuracy = 0.76 pour k = 4
-*-
Accuracy = 0.73 pour k = 5
-*-
Accuracy = 0.77 pour k = 6
-*-
Accuracy = 0.79 pour k = 7
-*-
Accuracy = 0.75 pour k = 8
-*-
Accuracy = 0.76 pour k = 9
-*-
Accuracy = 0.73 pour k = 10
-*-
Accuracy = 0.74 pour k = 11
-*-
Accuracy = 0.72 pour k = 12
-*-
Accuracy = 0.72 pour k = 13
-*-
Accuracy = 0.69 pour k = 14
-*-
Accuracy = 0.7 pour k = 15
-*-
Accuracy = 0.69 pour k = 16
-*-
Accuracy = 0.68 pour k = 17
-*-
Accuracy = 0.69 pour k = 18
-*-
Accuracy = 0.7 pour k = 19
-*-
Accuracy max = 0.79
avec k = 7
----
```

```
-- Avec la distance euclidienne --
 -> done processing
-- Classification avec KNN --
Accuracy = 0.72 pour k = 1
-*-
Accuracy = 0.71 pour k = 2
-*-
Accuracy = 0.74 pour k = 3
-*-
Accuracy = 0.75 pour k = 4
-*-
Accuracy = 0.78 pour k = 5
-*-
Accuracy = 0.76 pour k = 6
-*-
Accuracy = 0.75 pour k = 7
-*-
Accuracy = 0.76 pour k = 8
-*-
Accuracy = 0.76 pour k = 9
-*-
Accuracy = 0.73 pour k = 10
-*-
Accuracy = 0.75 pour k = 11
-*-
Accuracy = 0.72 pour k = 12
-*-
Accuracy = 0.71 pour k = 13
-*-
Accuracy = 0.74 pour k = 14
-*-
Accuracy = 0.73 pour k = 15
-*-
Accuracy = 0.73 pour k = 16
-*-
Accuracy = 0.72 pour k = 17
-*-
Accuracy = 0.73 pour k = 18
-*-
Accuracy = 0.68 pour k = 19
-*-
Accuracy max = 0.78
avec k = 5
```

Comme Isomap, on a aussi décidé de réduit la dimension à 8 pour pouvoir comparer entre les 2 algorithmes. Contrairement à Isomap, avec PCoa, notre notion de similarité ne perd pas de précision et on peut dire que le modèle est amélioré (le taux de précision est augmenté à 79%) ! Donc effectivement l'algorithme nous aide à donne des meilleurs résultats. Un point fort de notre similarité est que si on ne perd pas de détails, et pour le classement, ça fonctionne assez bien! En comparant rapidement avec la distance euclidienne, notre notion de similarité devient un peu plus avantageuse.

5) Algorithme Partition Binaire

```
---------- 5) ALGORITHME Partition Binaire ----------
... Processing ....
Execute avec 30 partitions
Class partition : [3, 1, 5, 0, 3, 2, 8, 2, 2, 7, 0, 5, 0, 2, 3
, 3, 5, 0, 0, 2, 2, 4, 2, 4, 6, 7, 0, 3, 0, 6]
Accuracy = 0.62
```

```
-- Avec la distance euclidienne --
Class partition : [6, 1, 3, 5, 0, 0, 2, 2, 5, 7, 0, 4, 0, 3, 2
, 0, 6, 4, 0, 3, 9, 2, 0, 2, 2, 2, 3, 3, 8, 3]
Accuracy = 0.6
```

Comme l'algorithme de k-medoids, le résultat de cet algorithme n'est pas haut. Même on a choisi de faire 30 partitions, c'est-à-dire un chiffre a plusieurs façons d'écrire. Peut-être c'est pour ça que le taux d'exactitude est meilleur que k-médoids. Mais comme notre notion de similarité n'est pas assez représentative pour bien répartir les chiffres entre les groupes ! Mais celui de la distance euclidienne n'est pas assez haut non plus.

Les forces :

- Le calcul est facile à faire et à comprendre
- Avec un niveau de précision haut, ça fonctionne assez bien avec la classification (KNN).
- Si avec 2 même chiffre écrite de façon très ressemble, la notion de similarité donne une meilleure prédiction.

Les faibles :

- Le temps de calcul est quand même long car on doit faire les translations
- Ne fonctionne pas très bien avec les algorithmes de partition car ce n'est pas assez significatif pour regrouper dans les classes.
- La performance descend rapidement si on perd de précision (réduite dimensionnelle)

Hypothèses :

1) L'algorithme Isomap est perdu plus de précision que PCoa dans ce cas ?
2) La classification (KNN) sera plus performance si on applique le bon algorithme de réduite dimensionnelle.
3) Le jeu de données MNIST n'est pas favorable pour les algorithmes de partitions.

DATASET : Adult

Notion de similarité

La similarité pour les données ADULT est une somme de similarité associée à chaque colonne. Le processus est le suivant :
Prenons xi et xj comme deux candidats

1. Séparer les données en données discrètes et continues, soit {age, fnlwgt, educational-num, capital-gain, capital-loss, hours-per-week} les données continues et {workclass, education, marrital-status, occupation, relationship, race, gender, native-country} les données discrètes.

2. Calculer une similarité pour chaque colonne et la façon se diffère entre les données continues et discrètes.
   2.1. Façon pour calculer les données continues :

$$1 - \frac{(Value\ of\ xi - Value\ of\ xj)}{Max\ value\ of\ column - \ Min\ \ value\ of\ column}$$

Cela retourne toujours un nombre entre 0 (Le moins similaire) et 1 (Le plus similaire)

   2.2. Façon pour calculer les données discrètes :

-Pour chaque groupe dans la colonne (ex. dans la colonne 'race', 'Black' et 'White' sont des groupes), calculer le ratio des personnes ayant moins que 50K salaire dans ce groupe parmi toutes les personnes dans ce groupe :

$$Ratio(group\ i) = \frac{\#People\ with\ income \leq 50K\ in\ group\ i}{\#People\ in\ group\ i}$$

-Calculer la différence entre deux groupes en soustrayant leur ratio correspondant, i.e.

$$Difference(group\ i, group\ j) = \ Ratio(group\ i) - Ratio(group\ j)$$

-La similarité de ces deux groupes est donc :

$$Similarity(group\ i, group\ j) = 1 - Difference(group\ i, group\ j)$$

   2.2. Pour toutes les colonnes discrètes, on a maintenant une similarité entre chaque deux groupes de cette colonne.

3. Additionner les similarités de toutes les colonnes en associant un poids à chaque similarité, i.e.

$$Similarity(xi, xj) = \sum_{p \in D} cpsp + \sum_{q \in C} cqsq$$

Where
D = Groupe des colonnes discrètes
C = Groupe des colonnes continues
sp = Similarity(group of xi in p, group of xj in p)
sq = Similarity(value of xi in q, value of xj in q)
cp = Poids associe à la colonne p
cq = Poids associe à la colonne q

- $\sum_{p \in D} cp + \sum_{q \in C} cq = 1$,
- $0(Least\ similar) \leq Similarity(xi, xj) \leq 1(Most\ similar)$

Note :
-Le Ratio est toujours <= 1, donc pour facilite de calcul on met la similarité dans 2.1 aussi <=1.
-On considère seulement le cas avec salaire <=50K car avec le cas salaire >50K, le résultat revient au même.
(Ratio<=50K = 1 – Ratio>50K → Diff(Ratio<=50K) = Diff(Ratio>50K))
-Facon d'obtenir les poids :

-Cas discrète : Calculer la moyenne de distinction entre les groupes de la colonne, $\mu(column\ k) \frac{\sum_{i,j \in k} Similarity(group\ i, group\ j)}{\#Pairs\ of\ groups}$ et comparer la valeur entre les différentes colonnes.

-Associer subjectivement des valeurs entre 0 et 1 a chaque colonne en considérant les $\mu(column\ k)$ et l'importance de la colonne sur le salaire.

Motivation de la similarité

1. Deux groupes d'une même colonne se ressemblent quand ils ont une distribution similaire de #Personne avec salaire <= ou > 50K, l'inverse de la différence entre ces deux groupes reflète bien cette similarité.
2. L'inverse des écarts entre les données continues reflète bien la similarité pour la colonne a quelle ils associent.
3. Le poids associe à chaque colonne a pour but de distinguer les influences diverses de chaque colonne sur le salaire.

Analyse

```
###------ Dataset ADULT ------####
---importing data.....
# Data train = 500
# Data test = 125
(500, 500)
(125, 500)
- imported
```

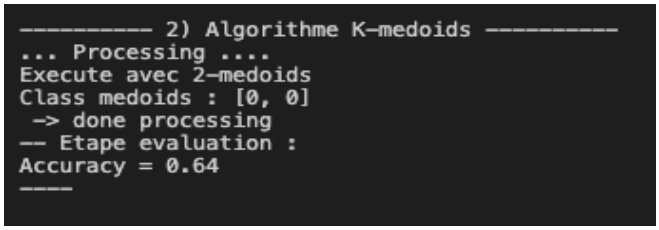
Nombre de données utilisées : 500 pour train, 125 pour test

1) Algorithme KNN

```
---------- 1) Algorithme KNN ----------
... Processing ....
Accuracy = 0.72 pour k = 1
-----
Accuracy = 0.704 pour k = 2
-----
Accuracy = 0.728 pour k = 3
-----
Accuracy = 0.704 pour k = 4
-----
Accuracy = 0.752 pour k = 5
-----
Accuracy = 0.752 pour k = 6
-----
Accuracy = 0.76 pour k = 7
-----
Accuracy = 0.76 pour k = 8
-----
Accuracy = 0.752 pour k = 9
-----
Accuracy = 0.736 pour k = 10
-----
Accuracy = 0.752 pour k = 11
-----
Accuracy = 0.736 pour k = 12
-----
Accuracy = 0.744 pour k = 13
-----
Accuracy = 0.736 pour k = 14
-----
Accuracy = 0.76 pour k = 15
-----
Accuracy = 0.736 pour k = 16
-----
Accuracy = 0.768 pour k = 17
-----
Accuracy = 0.768 pour k = 18
-----
Accuracy = 0.792 pour k = 19
-----
 -> done processing
Accuracy max = 0.792
avec k = 19
-----
```

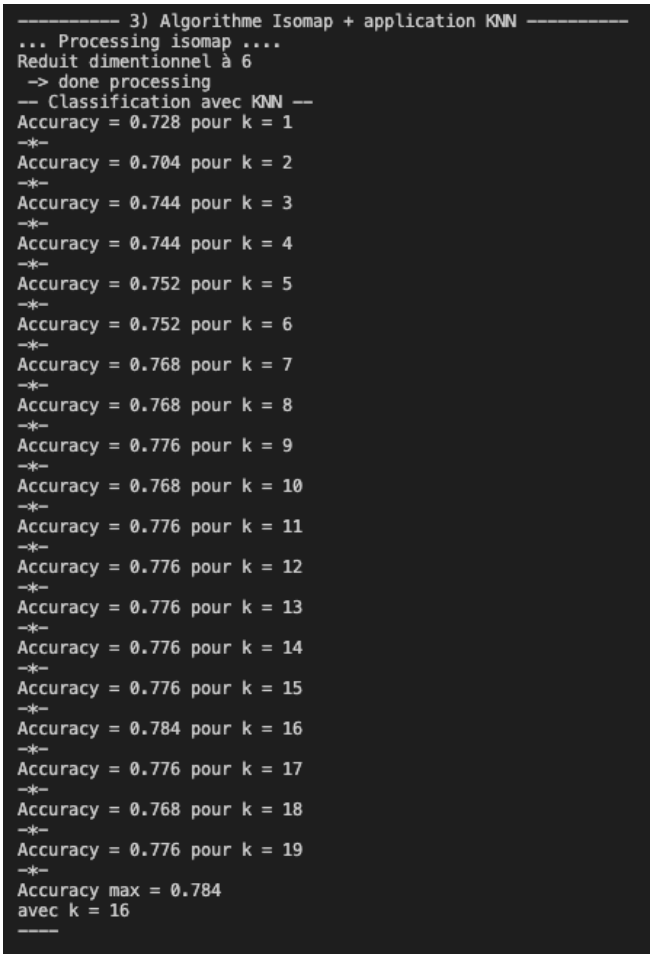
L'exactitude atteint le maximal (0.792) avec k=19. La performance semble bonne.

2) Algorithme K-médoïde



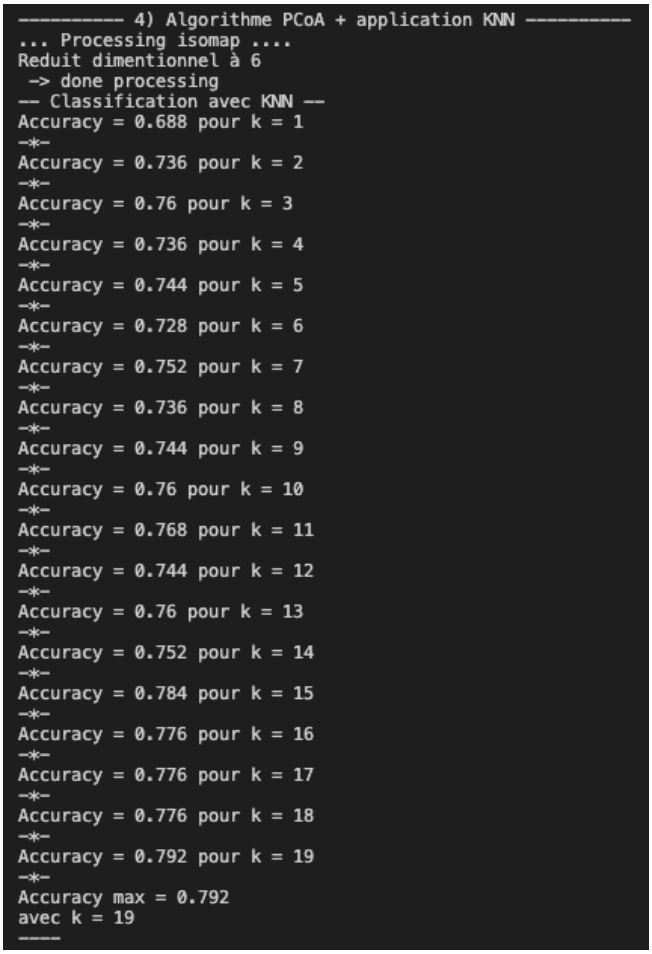
La méthode de K-médoïde donne un résultat moins précis que le modèle naïf. Le 'Class medoids' dans l'image représente la classe qui domine dans le centroïde. On peut voir, pour les données ADULT, il est difficile de distinguer les données intuitivement, on pourrait constater qu'avec la similarité que j'ai proposée, la distribution des données sont presque uniforme et n'ont pas une forte concentration a un centre.

3) Algorithme Isomap et KNN



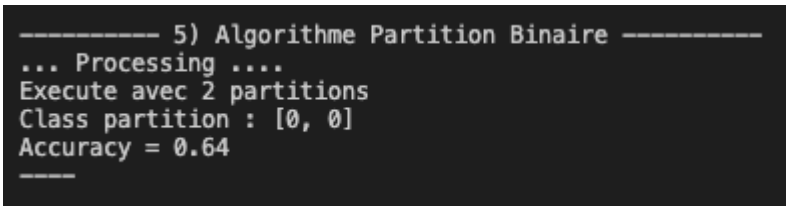
L'algorithme donne une exactitude pas pire quand la dimension est réduite a 6 et avec k=16. L'exactitude a une tendance de croitre avec la croissance de k, et ne change plus considérablement à partir du k=9, la moyenne des exactitudes pour les k entre 9 et 19 est environs 0.776, on peut considérer le 0.784 comme une légère variation accidentelle, la valeur 0.776 décrit plus précisément l'exactitude avec cet algorithme. Comparativement avec l'algorithme KNN, l'exactitude baisse légèrement. Donc il est préférable d'utiliser l'algorithme KNN sans réduction de dimensionnalité avec Isomap.

4) Algorithme PCoA et KNN



L'algorithme atteint son max exactitude =0.792 avec k=19 a dimension 6. L'exactitude varie plus par hasard compare aux exactitudes de Isomap. L'application de l'algorithme KNN est meilleur après la réduction de dimensionnalité avec la méthode PCoA.

5) Algorithme Partition Binaire



Tout comme l'algorithme K-médoïde, la partition binaire ne donne non plus une classification raisonnable avec une exactitude de 0.64.

Force :

-Facile à calculer et interpréter.

-Peut se grouper de façon différente quelles colonnes entrent dans le calcul de la similarité, ce qui est facile de distinguer quelle attribut (colonne) affecte plus le salaire d'une personne.

Faiblesse :

-L'écart entre les données n'est pas significatif, ce qui peut être la cause d'avoir une exactitude basse dans l'algorithme K-médoïde et Partition Binaire

-Le temps de calcul s'élevé considérablement avec l'ajout d'un groupe de plus dans une colonne

-Les poids sur les colonnes sont déterminé de façon avec peu de support de théories et de données.