Emulie Chhor (20186277)
Tasha Bhojwani (20213638)

IFT 3700 – Science des données

Devoir 1
14 novembre 2021

Content

1. Introduction

Nous allons proposer dans ce rapport deux notions de similarité originales et spécifiquement construites pour les jeux de données MNIST et ADULT afin d'augmenter la performance de divers algorithmes de partitions et classifications.

Pour cela nous établirons pour chaque jeu de données une notion de similarité à partir de nos connaissances sur les données, puis nous testerons les algorithmes un à un en utilisant notre notion.

Pour plus de clarté, voici le schéma des fichiers utilisés :

- **Main.py :** programme principal qui divise les données en plusieurs ensembles, puis forme les matrices de dissimilarité de notre notion de similarité, applique les algorithmes de clustering, classification et réduction de dimension à nos données avec cette notion de dissimilarité et les évalue.

- **Algorithmes de base :** fichiers implémentant les algorithmes que l'on utilise dans le main pour chaque jeu de données
    - Base.py : implémente des fonctions propres aux différents types d'algorithmes (clustering, classification, réduction de dimension)
    - KNN.py
    - KMedoids.py
    - Isomap.py
    - PartitionBinaire.py
    - PCoA.py

- **Fichiers ADULT** :
    - **ADULT.csv**
    - **ADULT_clean.csv** : jeu de données nettoyé créé par preprocessing.py
    - **constants.py** : pose des matrices de dissimilarités et des dictionnaires utilisés dans preprocessing.py
    - **preprocessing.py** : nettoie le jeu de données ADULT avec les constatations observées
    - **dissimilarity.py** : fonctions créant la notion de similarité du jeu de données ADULT

- **Fichiers MNIST** :
    - **Mnist_test.csv, mnist_train.csv**
    - **Mnist_test_clean.csv, mnist_train_clean.csv** : jeux de données nettoyés, créés par preprocessing.py
    - **preprocessing.py** : permet de visualiser les données de MNIST
    - **dissimilarity.py** : fonctions créant la notion de similarité du jeu de données MNIST

2. Jeu de données ADULT

2.1. Exploration des données : preprocessing

Le jeu de données ADULT est composé de caractéristiques et d'une donnée cible « *income* » qui peut prendre deux valeurs : soit >50K, soit <=50K. On cherche à prédire cette valeur cible *the income* à partir des autres caractéristiques qui sont : *age, workclass, fnlwgt, education, educational-num, marital status, occupation, relationship, race, sexe, capital-gain, capital-loss, hours-per-week, native country.*

Il faut garder en tête que selon nos données brutes, environ 76% des individus ont un *income* égal à <=50K. Ainsi, un algorithme naïf permettrait d'obtenir 76% d'exactitude. Donc un bon algorithme de prédiction doit dépasser ce seuil.

2.2. Nettoyage des données

Afin d'avoir des résultats corrects, il faut corriger certaines données aberrantes ou manquantes. Nous avons choisi de faire plusieurs opérations de nettoyage de données :

- **Retirer les doublons**, c'est-à-dire les lignes identiques, pour ne pas fausser nos statistiques (moyenne, médiane, proportions…).

- **Traiter les données absurdes**, c'est-à-dire les données qui ne correspondent pas aux valeurs discrètes de base. Nous devons donc définir ce que sont les données absurdes pour chaque catégorie :
    - Age : tout ce qui n'est pas un entier plus grand que 16, on remplace ces valeurs par la moyenne
    - Hours-per-week : tout ce qui n'est pas un entier entre 0 et 80, on remplace ces valeurs par la moyenne
    - Capital-gain, Capital-loss : tout ce qui n'est pas un entier positif ou nul, on traite ces données comme manquantes
    - Workclass, Education, Marital-status, Occupation, Race, Native-country : tout ce qui ne correspond pas aux valeurs discrètes de base, on traite ces données comme manquantes

Nous n'avons pas traité les données absurdes dans notre code, mais nous avons cherché ces pistes pour les traiter.

- **Traiter les données manquantes.** On remarque en effet que 7,41% des lignes ont des valeurs manquantes. Nous allons travailler avec des ensembles réduits de données et donc nous n'avons pas besoin de toutes les données, nous décidons donc de supprimer ces lignes. Notons qu'il y a alors un biais dans nos données puisque si certaines données sont manquantes, il y a sûrement une raison derrière ce manque.

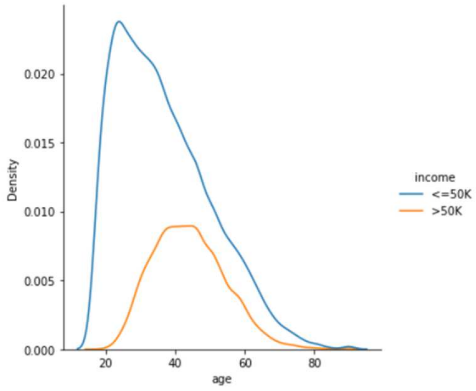
2.3. Tri des colonnes

Nous allons ici regrouper les classes similaires ensemble (*preprocessing*). On veut déterminer si la classe est un bon prédicteur en visualisant nos données (EDA : correlation plots, clustering). Selon nos résultats (les EDA), nous allons faire des hypothèses qui justifieront nos choix de créer de nouvelles catégories ou de ne pas considérer certaines colonnes.

- **Age** :

Nous allons classer les données par tranche d'âge pour pouvoir mieux noter la différence d'âge. En effet, une différence de 10 ans de 16 à 26 ans importe plus qu'entre 70 et 80 ans. L'impact de la différence d'âge n'est pas linéaire.

Nous évaluons l'impact de l'âge sur la colonne income avec un tableau nous donnant les proportions d'être dans la catégorie <50K selon l'âge et le graphe suivant :



- 16-27 ans : ***étudiant*** - ne travaille pas ou très peu à temps partiel - plus de 90% à <50K
- 28-35 ans : ***premier emploi*** - l'individu travaille à temps plein et obtient un salaire moyen - 74 à 87%
- 36-59 ans : ***emploi stable*** - travaille à temps plein avec un bon salaire (apogée du salaire) - 60 à 70%
- 60-67 ans : ***presque-retraité*** - travaille moins d'heures et son salaire diminue - 70 à 77%
- 68+ ans : ***retraité*** - ne travaille plus, obtient des pensions gouvernementales ou des investissements selon son pays d'origine - plus de 78%

Ainsi voici notre matrice de différence :

	Étudiant	**Premier-emploi**	**Emploi stable**	**Presque-retraité**	**Retraité**
Étudiant	0	2	4	3	1
Premier-emploi	2	0	2	1	1
Emploi stable	4	2	0	2	3
Presque retraité	3	1	2	0	2
Retraité	1	1	3	2	0

(Nous avons mis une différence de 1 lorsque les proportions se recoupent)

- **Workclass** :

Il y a plusieurs catégories à considérer ici : *Private, Self--emp--not--inc, Self--emp--inc, Federal--gov, Local--gov, State--gov, Without--pay, Never--worked*. Nous allons, pour cette caractéristique, regrouper certaines catégories ensemble pour ensuite attribuer un "score de différence" entre nos nouvelles catégories.

- ***Government*** : regroupe *Federal-gov, Local-gov, State-gov*. Ces trois catégories correspondent à des emplois publics que l'on va considérer comme équivalents.
- ***Self-employed*** : regroupe *self-emp-not-inc* et *self-emp-inc*. Ces deux catégories sont en effet rapprochables en ce qu'elles traitent toutes les deux de travailleurs indépendants.

- ***Private*** : regroupe les catégories du secteur privé, c'est-à-dire *Private.*
- ***Jobless*** : regroupe *without-pay* et *never-worked* car dans ces deux catégories, la personne n'est pas payée.

Notre matrice de score de différence est la suivante :

	Government	Self-employed	Private	Jobless
Government	0	2	2	3
Self-employed	2	0	1	3
Private	2	1	0	3
Jobless	3	3	3	0

En effet, il est très différent de travailler tout en gagnant de l'argent que de ne pas travailler, d'où une différence de 3. La différence n'est pas très grande entre les indépendants et les travailleurs du secteur privé, donc leur score de différence est de 1. Il y a une différence un peu plus marquée entre les travailleurs du secteur public et du secteur privé ou des indépendants d'où le score de 2.

On obtient les proportions suivantes :

```
workclass         income
?                 <=50K     0.905323
                  >50K      0.094677
Federal-gov       <=50K     0.608240
                  >50K      0.391760
Local-gov         <=50K     0.704401
                  >50K      0.295599
Never-worked      <=50K     1.000000
Private           <=50K     0.782133
                  >50K      0.217867
Self-emp-inc      >50K      0.553392
                  <=50K     0.446608
Self-emp-not-inc  <=50K     0.721129
                  >50K      0.278871
State-gov         <=50K     0.732458
                  >50K      0.267542
Without-pay       <=50K     0.904762
                  >50K      0.095238
Name: income, dtype: float64
```

```
workclass      income
government     <=50K     0.692498
               >50K      0.307502
jobless        <=50K     0.904762
               >50K      0.095238
private        <=50K     0.782184
               >50K      0.217816
self-employed  <=50K     0.637684
               >50K      0.362316
Name: income, dtype: float64
```

- **Fnlwgt** :

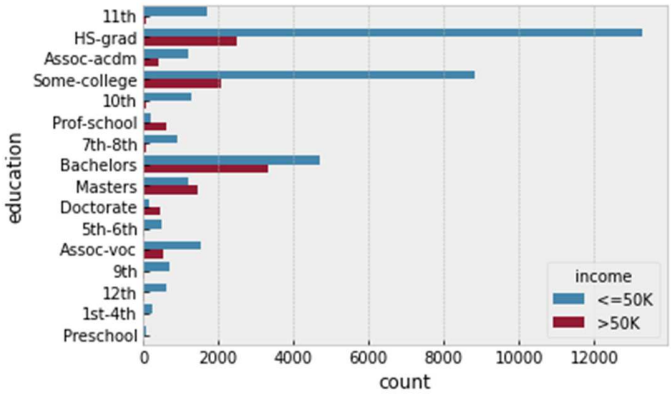
Cette colonne représente la proportion réelle de personnes correspondant à ces caractéristiques selon les auteurs du recensement des données. Nous n'allons pas utiliser cette colonne pour notre notion de similarité puisqu'elle prédit la probabilité qu'une personne corresponde à ces critères, et cela ne nous apporterait pas grand-chose.

- **Education** :

Nous allons utiliser l'hypothèse suivante : plus les gens sont éduqués, plus ils gagnent de l'argent. Mais cela n'est pas linéaire : il existe un seuil à partir duquel être plus éduqué ne permet pas de gagner plus. Nous considèrerons l'hypothèse ainsi que cette limite afin de classifier nos données en catégories distinctes.

Les catégories à classifier sont : *Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc--acdm, Assoc--voc, 9th, 7th--8th, 12th, Masters, 1st--4th, 10th, Doctorate, 5th-6th, Preschool*. Nous décidons de créer trois regroupements seulement, et de garder les autres catégories telles qu'elles :

- ***Below-HS*** : regroupe les individus qui ont au plus un niveau d'études égal à 11, c'est-à-dire avant un *highschool education* (*11th, 10th, 9th, 7th-8th, 12th, 1st-4th, 5th-6th, Preschool*)
- ***Highschool*** : *HS-grad*
- ***Some-College***
- ***Associate*** : regroupe *Assoc-acdm, Assoc-voc*
- ***Bachelors***
- ***Masters***
- ***Doctorate*** : regroupe *Doctorate*, *Prof-school*



Selon nos données, sont dans la classe des *income* de <=50 K :

- 27% des doctorate (26% des Prof-school, et 27% des doctorate)
- 45% des masters
- 41% des bachelors
- 74% des associates (74% des assoc-acdm et 74% des assoc-voc)
- 81% des SomeCollege
- 83% des highschool
- 94% des below-HS (entre 92% et 97% pour toutes les sous-catégories)

On remarque certaines similarités dans les proportions obtenues (par exemple entre bachelors et master). Selon ces résultats, nous posons la matrice de dissimilarité suivante :

	Below-HS	highschool	Some-college	associate	bachelor	master	doctorate

Below-HS	0	2	2	2	4	4	5
highschool	2	0	1	1	3	3	5
Some college	2	1	0	1	3	3	4
associate	2	1	1	0	2	2	4
bachelor	4	3	3	2	0	1	2
master	4	3	3	2	1	0	2
doctorate	5	5	4	4	2	2	0

- **Education num** :

Cette colonne se déduit de la colonne précédente. Puisque des catégories ont été attribuées précédemment aux personnes, et qu'on a déjà établit une notion de similarité pour ces catégories, nous pouvons ignorer cette colonne.

On estime en effet que le diplôme est plus important que le nombre d'années : une différence de 3 ans entre 3 et 6 ans d'éducation n'a pas la même importance qu'une différence de 3 ans entre 12 et 15 ans d'éducation puisque la présence d'un diplôme universitaire ou non joue plus que la présence d'un diplôme d'éducation primaire. Donc l'impact de la différence du nombre d'années d'éducation n'est pas linéaire. Autrement dit, on estime que l'éducation est un bon prédicteur à partir de et jusqu'à un certain point.

C'est pour ces raisons qu'il est préférable d'utiliser des données nominales ici et d'établir notre notion de similarité à partir de ces catégories plutôt que d'entiers. Nous n'utiliserons donc pas cette colonne pour notre notion de similarité.

- **Marital-status** :

Selon nos données, il faudrait regrouper les catégories en deux groupes : *single* et *has-partner*. En effet, si l'individu est célibataire ou n'a pas d'époux présent, on remarque que cela est déterminant. Dans l'autre cas, cela ne nous apporte rien.

Donc, les catégories déterminantes sont : *divorced, married-spouse-absent, never-married, separated, widowed*. Si l'individu appartient à une de ces catégories, il y a environ 90% de chance que son *income* soit de <=50K.

```
marital-status         income
Divorced               <=50K     0.898839
                       >50K      0.101161
Married-AF-spouse      <=50K     0.621622
                       >50K      0.378378
Married-civ-spouse     <=50K     0.553867
                       >50K      0.446133
Married-spouse-absent  <=50K     0.907643
                       >50K      0.092357
Never-married          <=50K     0.954520
                       >50K      0.045480
Separated              <=50K     0.935294
                       >50K      0.064706
Widowed                <=50K     0.915679
                       >50K      0.084321
Name: income, dtype: float64
```

Finalement on décide de différencier selon nos catégories ***single*** et ***has-partner*** avec une valeur booléenne. La différence de catégorie vaudra 1.

- **Occupation** :

Cette colonne prend les valeurs : *Tech--support, Craft--repair, Other--service, Sales, Exec-managerial, Prof--specialty, Handlers--cleaners, Machine--op--inspct, Adm--clerical, Farming--fishing, Transport--moving, Priv--house--serv, Protective--serv, Armed--Forces*.

On observe selon nos valeurs que les personnes ayant un income de <=50K sont :

- *98% Priv--house--serv*
- *95% Other--service*
- *93% Handlers--cleaners*
- *88% Farming--fishing*
- *87% Machine--op--inspct*
- *86% Adm--clerical*
- *79% Transport--moving*
- *77% Craft--repair*
- *73% Sales*
- *70% Tech--support*
- *68% Protective--serv*
- *66% Armed--Forces*
- *54% Prof--specialty*
- *52% Exec-managerial*

Nous allons donc regrouper ces valeurs en catégories en trois groupes distincts : **secteur-primaire (93% de <50K)**, **secteur-mixte (82%)**, **secteur-tertiaire (64%)**. La matrice de différence est alors la suivante :

	Primaire	Mixte	Tertiaire
Primaire	0	1	2
Mixte	1	0	1
Tertiaire	2	1	0

- **Relationship** :

Cette colonne peut prendre les valeurs : *Wife, Own--child, Husband, Not--in--family, Other--relative, Unmarried*. La valeur peut être déterminante ici pour *not-in-family, other-relative, own-child, unmarried* : lorsque l'individu appartient à ces catégories, il y a environ 90% de chances qu'il soit dans la catégorie <=50K.

```
relationship    income
Husband         <=50K     0.551329
                >50K      0.448671
Not-in-family   <=50K     0.898593
                >50K      0.101407
Other-relative  <=50K     0.965471
                >50K      0.034529
Own-child       <=50K     0.985358
                >50K      0.014642
Unmarried       <=50K     0.939707
                >50K      0.060293
Wife            <=50K     0.531103
                >50K      0.468897
Name: income, dtype: float64
```

Nous n'allons pas prendre en compte cette colonne puisqu'elle reprend les informations de la colonne *marital-status*, cela serait donc redondant dans notre calcul.

- **Race** :

Cette colonne peut prendre les valeurs : *White, Asian--Pac--Islander, Amer--Indian--Eskimo, Other, Black*. Observons les données :

```
race                income
Amer-Indian-Eskimo  <=50K     0.882979
                    >50K      0.117021
Asian-Pac-Islander  <=50K     0.730744
                    >50K      0.269256
Black               <=50K     0.879189
                    >50K      0.120811
Other               <=50K     0.876847
                    >50K      0.123153
White               <=50K     0.746013
                    >50K      0.253987
Name: income, dtype: float64
```

On remarque que certaines catégories sont équivalentes, et on peut donc les regrouper de la manière suivante :

- ***Bipoc*** (Black, Indigenous, People Of Colour), environ 87% de <=50K : Amer-indian-eskimo, black, other (bipoc)
- ***Non-bipoc***, environ 74% de <=50K : White, asian-pac-islander (non-bipoc)

Nous allons donc attribuer une différence de 0 lorsque les individus sont dans une même catégorie, et de 1 sinon.

- **Capital-gain** et **capital-loss** :

Ces deux colonnes correspondent au montant de l'augmentation du capital d'investissement d'une personne. Nous les regroupons en une seule colonne "*capital*" et on considèrera que les individus qui investissent sont plus enclin à avoir un *income* de >50K puisqu'ils ont une meilleure éducation financière. Cela se vérifie par nos données :

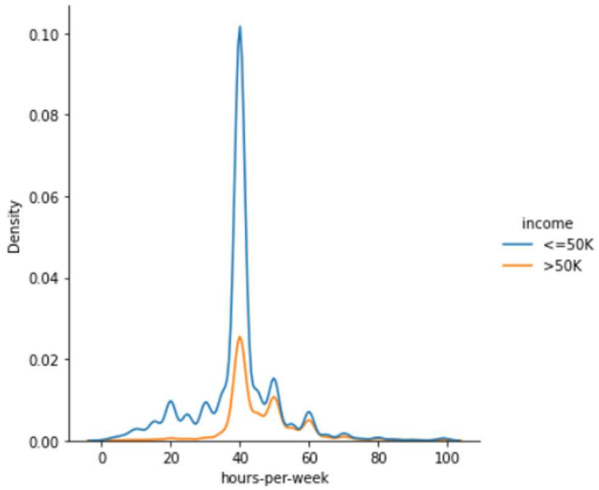
```
is_investing  income
False         <=50K     0.765214
              >50K      0.234786
True          >50K      0.513084
              <=50K     0.486916
Name: income, dtype: float64
```

On voit bien qu'un individu qui investit a beaucoup plus de chance d'être dans la catégorie des >50K : on passe de 23% à 51%.

Ainsi, notre colonne *capital* comprendra une valeur booléenne égale à 1 si l'individu investit ou à 0 dans le cas contraire.

- **Hours-per-week** :

Cette colonne nous indique le nombre d'heures de travail de l'individu par semaine. Avec nos données, nous avons le graphique suivant :



On remarque que la courbe orange se rapproche de plus en plus de la courbe bleue. Cela signifie que plus l'individu travaille, plus il a de chances de gagner de l'argent. La différence entre les proportions de personnes qui ont un income de <=50K et de >50K entre les deux catégories diminue.

Notre notion de similarité se basera donc sur une catégorisation, selon le tableau de proportions généré :

- Entre 0 et 30 heures par semaine : **temps-partiel** *(environ 90% de <50K)*
- Entre 31 et 39 heures par semaine : ***temps-normal*** *(environ 80-90% de <50K)*
- Entre 40 et 43 heures par semaine : ***temps-plein*** *(environ 70-80% de <50K)*
- Plus de 44 heures par semaine : ***overtime*** *(environ 50-65% de <50K)*

Ainsi notre matrice de dissimilarité est la suivante :

	Temps-partiel	**Temps-normal**	**Temps-plein**	**overtime**
Temps-partiel	0	1	2	3
Temps-normal	1	0	1	2
Temps-plein	2	1	0	1
Overtime	3	2	1	0

- **Native-Country** :

Nous décidons de diviser nos données en 3 catégories : les pays riches, les pays en voie de développement et les pays pauvres. Pour autant, puisque les données de ADULT ont été sondées en 1994, on ne doit pas établir le même classement que l'on ferait aujourd'hui. On choisit d'utiliser la classification de : https://www.journaldunet.com/business/salaire/classement/pays/revenus

Ainsi notre classification est la suivante :

- ***Riches (74% de <=50K)*** : US, England, Canada, Germany, Japan, Taiwan, Ireland, France, Scotland, Italy, Portugal, Iran

- ***Moyens (70%)*** : India, Greece, South, China, Cuba, Holand-Netherlands, Outlying-US, Hungary
- ***Pauvres (88%)*** : Cambodia, Puerto-Rico, Guatemala, Nicaragua, Thailand, Yugoslavia, Honduras, Trinidad&Tobago, Peru, Hong, Vietnam, Dominican-Republic

Remarque : on voit bien avec les proportions obtenues que la catégorisation entre pays riches et pays moyens importe peu puisque l'on penserait obtenir une proportion de riches plus grandes dans les pays riches. Notre classification des pays ne se basant pas sur nos données mais sur un classement déjà fait, il est logique que cela soit peu influant dans notre calcul de similarité.

Notre matrice de différence se définit donc comme suit :

	Riche	**Moyen**	**Pauvre**
Riche	0	1	2
Moyen	1	0	1
Pauvre	2	1	0

2.4. Notion de similarité

En observant nos données, nous avons déduit que nous n'allons pas utiliser les colonnes *relationship*, *education-num* et *fnlwgt.* Nous allons aussi regrouper les colonnes *capital-gain* et *capital-loss* en une seule colonne '*capital*'. Ainsi il nous reste à considérer les neuf caractéristiques suivantes : *age, workclass, education, marital-status, occupation, race, capital, hours-per-week et native-country*.

Nous devons déterminer l'importance de chaque catégorie pour leur attribuer des coefficients dans notre calcul de similarité. Pour cela, nous allons regarder si la catégorie est déterminante et la comparer au seuil des 76% de l'algorithme naïf. Parfois, le coefficient dépendra de la valeur de la catégorie.

Dans le tableau suivant, pour chaque catégorie, nous avons recensé le type de valeur de différence que l'on attribue à la catégorie, la valeur la plus déterminante de la catégorie et son importance, la valeur la moins déterminante de la catégorie et son importance, ainsi que le coefficient final que l'on va attribuer à cette catégorie.

La valeur la plus déterminante de la catégorie correspond à la valeur pour laquelle la proportion d'individus qui ont cette caractéristique ont un income de <=50K, et la valeur la moins déterminante est celle qui a la proportion la plus petite.

Nous déterminons le coefficient final à partir de la différence d'importance des valeurs plus et moins déterminantes. Plus la différence est grande, plus la catégorie peut être déterminante.

Colonne	Valeur de différence	Valeur la plus déterminante	Importance de cette valeur (%)	Valeur la moins déterminante	Importance de cette valeur (%)	Coefficient final
Age	Dissimilarité 0 à 4	Étudiant	+ de 90	Emploi-stable	60-70	MOYEN
Workclass	Dissimilarité 0 à 3	Jobless	90	Self-employed	63	MOYEN
Education	Dissimilarité 0 à 5	Below-HS	+ de 92	Doctorate	27	IMP
Marital status	Booléen 0 ou 1	Single	+ de 90	Has-partner	55	MOYEN
Occupation	Dissimilarité 0 à 2	Primaire	93	Tertiaire	64	MOYEN
Race	Booléen 0 ou 1	Bipoc	87	Non-bipoc	74	PAS IMP
Capital	Booléen 0 ou 1	False	76	True	48	PAS IMP – MOYEN
Hours per week	Dissimilarité 0 à 3	Peu travailleur	+ de 90	Trop travailleur	50	MOYEN-IMP
Native country	Dissimilarité 0 à 2	Pauvres	88	Moyens	70	PAS IMP

On attribue les coefficients comme suit :

- Imp : 4
- Moyen-imp : 3
- Moyen : 2
- Pas imp-moyen : 1,5
- Pas imp : 1

Afin de mettre exactement le poids des coefficients que l'on a déterminé à nos colonnes, il faut lisser nos valeurs de différence. On obtiendra alors pour chaque colonne une valeur entre 0 et 1. Ainsi, nous pouvons enfin établir notre calcul de dissimilarité :

*Diff (ind1, ind2) = 2*diff_age + 2/3*diff_workclass + 4/5*diff_education + 2*diff_marital-status + 2/2*diff_occupation + diff_race + 1,5*diff_capital + 3/3 diff_hours-per-week + 1/2*diff_native-country*

Ce qui nous donne finalement :

Diff (ind1, ind2) = 2*diff_age + 2/3*diff_workclass + 0,8*diff_education + 2*diff_marital-status + diff_occupation + diff_race + 1,5*diff_capital + diff_hours-per-week + 0,5*diff_native-country

Nous aurions pu très bien prendre en compte le fait que certaines caractéristiques sont déterminantes lorsqu'elles prennent une certaine valeur, et donc que les coefficients de chaque catégorie dans le calcul de similarité dépendent de la donnée. Par exemple, on remarque qu'il y a plus de 92% d'être

dans la classe <=50K lorsque education = below-HS, et donc cela aurait pu influencer notre calcul en pondérant plus cette valeur.

Le problème de cette méthode est qu'elle risque le surapprentissage. Mais dans le cas où on limiterait la pondération lorsqu'on obtient des valeurs extrêmes (dès 95% par exemple), cela pourrait donner de meilleurs résultats que notre notion de similarité qui ne prend pas en compte la diversité des proportions dans chaque catégorie.

Nous avons décidé de ne pas utiliser cette méthode puisque nous avons remarqué qu'il y a des chances qu'il n'y ait qu'un individu dans la classe lorsque la probabilité d'obtenir la classe <= 50K est supérieure à 95%. Par exemple, pour hours-per-week = 0, il n'y a un seul individu, et la probabilité d'obtenir la classe <=50K est de 100%.

2.5. Division des données

Afin de **réduire le temps de calcul** à un temps raisonnable, nous n'allons pas utiliser tout le jeu de données pour notre étude. Nous allons donc choisir un certain nombre de lignes aléatoires du jeu de données. En effet, il est préférable que nos données soient choisies aléatoirement pour obtenir de meilleurs résultats. Le meilleur choix serait de générer un sous-ensemble aléatoire du jeu de données complet à chaque lancement du programme.

Cependant, puisqu'il nous est demandé que chaque lancement du programme reproduise les mêmes résultats à la virgule près, nous avons généré un sous-ensemble du jeu de données aléatoire fixe que l'on utilisera pour tous les calculs.

Nous avons donc choisi de diviser nos données en sous-ensembles : un premier **sous-ensemble pour l'exploration de données**, un **sous-ensemble d'entraînement**, un **sous-ensemble test** et un **sous-ensemble de validation**. Nous allons pour cela tester les algorithmes un à un avec des roulements de ces sous-ensembles selon le procédé de *cross-validation.*

3. Jeu de données MNIST

3.1. Exploration des données

Le jeu de données MNIST est composé d'images, et l'on souhaite connaître, à partir des images, le chiffre qu'elle représente. Il y a 10 valeurs possibles : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Chaque valeur a environ 10% de chance d'être la valeur d'une image selon nos données.

Ce sont des images en échelle de gris, codées en pixels de 0 à 255. Nous pouvons réduire ces valeurs en binaire et ne considérer que le noir et le blanc, pour simplifier le traitement de l'image. Nous avons décidé d'ensuite combiner les blocs de quatre pixels (en carré) en un seul pixel étant égal à la moyenne des quatre pixels de départ. En effet, cela permet de réduire grandement la dimension de nos images (de 28x28 à 14x14) et gagner en temps de calcul.

3.2. Notion de similarité

On cherche une notion de similarité entre deux images. Pour cela, nous voulons calculer la différence entre deux images. Nous avons pensé à prendre la **distance de Hamming** entre les deux images (c'est-à-dire faire un XOR des deux chaînes de bits puis compter le nombre de 1 dans le résultat). Il faudrait alors calculer plusieurs fois cette distance de Hamming puisque deux images qui correspondent au même chiffre peuvent avoir un angle différent ou être placées différemment.

Mais il faut aussi prendre en compte les **translations**. Cela correspondrait alors à la **distance de Levenshtein**. Nous avons alors décidé de prendre le minimum entre la somme des distances de Levenshtein des lignes de pixels et la somme des distances de Levenshtein des colonnes de pixels. Ainsi, notre distance ne varie pas lors d'une translation horizontale ou verticale d'une des images.

Cette méthode n'est malheureusement pas parfaite pour plusieurs raisons :

Ce faisant, nous avons posé l'hypothèse de **l'indépendance des colonnes**, et de **l'indépendance des lignes** de pixels. Or cela n'est pas vrai, et cela rend notre notion de distance biaisée. Notre translation se fera alors sur quelques lignes/colonnes et pas sur les autres, elle sera hétérogène.

Il faudrait également prendre en compte les rotations d'images. Nous aurions pu rouler l'algorithme de distance de Levenshtein sur les mêmes images plusieurs fois en les faisant changer d'angle. Ainsi, peu importe l'angle de l'image, notre notion de similarité rendrait le même résultat. Malheureusement, nous n'avons pas pu trouver comment coder cela sur des matrices de dimension 14.

Aussi, on aurait pu considérer la grandeur de l'image, c'est-à-dire zoomer l'image lorsqu'il est nécessaire d'agrandir une image par rapport à l'autre pour que les chiffres soient de la même proportion et plus faciles à différencier. Nous avons décidé de ne pas considérer cela puisque la plupart de nos images sont à peu près de la même taille.

L'utilisation de la distance de Levenshtein entraîne une **grande complexité** de notre calcul de la matrice de dissimilarité puisqu'elle roule en O(n*m).

3.3. Division des données

De la même façon que pour ADULT, nous diviserons nos données en ensembles d'entraînement, de validation et de test, afin de procéder par *cross-validation,* pour les mêmes raisons que précédemment (réduire le temps de calcul, et conserver les mêmes ensembles aléatoires).

4. Performance des algorithmes

Afin de tester nos notions de similarité, nous allons évaluer certains algorithmes sur nos données en utilisant nos notions.

Pour **ADULT**, nous allons comparer ici ce que nous obtenons avec l'approche naïve à 76% et avec les résultats des algorithmes vus dans les démonstrations afin de tester la performance des algorithmes demandés avec notre notion de similarité.

Pour **MNIST**, nous avons déjà observé que notre notion de similarité est imparfaite en théorie. Nous allons vérifier cela par la pratique en comparant, dans cette partie, la performance d'algorithmes utilisant notre notion de similarité avec la performance de l'algorithme naïf qui tourne à 10% d'accuracy (selon le code de la démonstration 1).

4.1. Classification : KNN (k-plus proches voisins)

Pour tester l'algorithme KNN qui est un algorithme de classification, nous procédons à plusieurs étapes :

- Nous cherchons d'abord les meilleurs hyperparamètres en entraînant l'algorithme avec l'ensemble d'entraînement, puis en utilisant la méthode grid search.
- Ensuite, on entraîne l'algorithme avec cet hyperparamètre. Et finalement on teste le score de l'algorithme avec l'ensemble de test.

On obtient les résultats suivants pour **ADULT** :

```
The accuracy with the training dataset is: 75.8
The K minimizing the error is: 12
The accuracy with the test dataset is: 75.2
```

KNN a une performance de 75.8% sur notre ensemble d'entraînement et 75.2% sur notre ensemble test. Cette performance est inférieure à celle de l'algorithme naïf qui est à 76%. Ainsi notre performance est plutôt mauvaise. Et cela pour plusieurs raisons :

L'algorithme KNN n'est pas très efficace sur les données massives en général. C'est pourquoi nous nous attendions à obtenir un résultat autour de 76%, la performance de l'algorithme naïf.

De plus, notre notion de distance se base sur les données obtenues lors de l'exploration des données. Or, KNN fonctionne mieux lorsque les données sont normalisées et standardisées. L'utilisation des matrices de similarité n'est donc pas compatible avec l'algorithme KNN.

Pour **MNIST**, on obtient :

```
The accuracy with the training dataset is: 62.0
The K minimizing the error is: 1
The accuracy with the test dataset is: 9.0
```

Le meilleur hyperparamètre k trouvé est 1. Lorsqu'on roule l'algorithme sur l'ensemble d'entraînement, cela correspond alors à comparer chaque élément à lui-même. Il est donc très étrange de trouver 62% et pas plus pour l'ensemble d'entraînement... C'est du surapprentissage.

Notre ensemble de test a un score de validation de 9%. Cela est plus bas que le score naïf, et donc ce n'est pas vraiment ce à quoi l'on s'attendait. Cela peut s'expliquer par le fait que nous utilisons encore une fois des données massives.

Si le meilleur hyperparamètre est 1, cela signifie que notre notion de distance n'est pas assez performante pour comparer les éléments entre eux.

4.2. Clustering : k-medoïde et partitionnement binaire

Kmedoids et le partitionnement binaire (regroupement hiérarchique) correspondent tous les deux à des algorithmes de clustering. Pour évaluer leur performance, nous procédons de la même manière pour ces deux algorithmes :

- Il faut d'abord **trouver les meilleurs hyperparamètres** pour l'algorithme :
  - On entraîne l'algorithme avec notre ensemble d'entraînement pour des certains hyperparamètres.
  - On compare ensuite le score silhouette obtenu sur notre ensemble de validation pour choisir le meilleur hyperparamètre.

  Grâce à cet hyperparamètre, nous pouvons alors former nos clusters.

- Il nous faut ensuite **tester la classification** obtenue. Pour cela, on utilise le fichier main.py où l'on associe chacun des clusters à une classe de la cible (*target*) selon la règle suivante : un cluster correspond à une certaine classe lorsque cette classe est majoritaire dans ce cluster.

  Cette méthode de test n'est **pas la plus optimale**.

  En effet, si le nombre de clusters optimal est plus petit que le nombre de classes, certaines classes ne seront pas représentées. Par exemple, si les classes non représentées représentent 40% de nos données, le *accuracy score* sera au plus de 60%. C'est pourquoi il faudrait avoir au moins 2 clusters pour ADULT, et au moins 10 pour MNIST.

  De la même façon, si un cluster n'est pas homogène, cette technique de classification n'associe pas une grande partie des résultats au bon label. Dans le pire des cas, si un cluster contient 51% éléments de classe A et 49% éléments de classe B, la classification oubliera les 49% d'éléments de classe B et les associera à la classe A.

Nous avons finalement choisi d'utiliser le **score silhouette** pour établir nos hyperparamètres et non la technique de classification. En effet, la technique de classification nous ferait tendre vers un hyperparamètre k trop grand pour maximiser le score obtenu. Elle créerait des sous-classes redondantes et cela risque de devenir de l'*overfitting*.

Notons que si, dans un cluster, il y a autant d'éléments de classe A que de classe B, cela pose problème : l'algorithme de classification choisira la classe au hasard (50%).

Nous avons également pensé à une **autre solution** possible (mais que nous n'avons pas implémentée). Nous aurions pu considérer qu'un cluster est homogène à partir d'un certain seuil (disons de 90%). Grâce à ce seuil, nous aurions pu distinguer les clusters homogènes des clusters non homogènes. Pour les clusters homogènes, nous aurions utilisé la stratégie de classification en comptant la classe la plus populaire, puisque cela aurait un score d'accuracy de 90% minimum. Pour les autres classes dont les clusters ne sont pas homogènes, nous aurions créé plus de sous-clusters, en faisant un peu d'*overfitting* sur notre algorithme, afin d'améliorer la performance de notre algorithme.

On obtient pour **ADULT** :

KMedoids

```
Le meilleur nombre de cluster pour KMedoids est: 2
Accuracy pour KMedoids: 0.82
```

Il y a deux clusters. Cela est logique puisque nous avons deux valeurs pour la donnée cible. S'il y en avait plus, peut être que l'algorithme aurait un meilleur score d'accuracy, mais on risquerait le surapprentissage.

Le score d'accuracy est de 82%. Cela est supérieur au score de 76% obtenu avec l'algorithme naïf. KMedoids est donc meilleur pour prédire le *income* que l'algorithme naïf.

Nous pouvons donc conclure que notre notion de similarité marche bien pour utiliser KMedoids sur le jeu de données ADULT.

Partitionnement binaire

```
Best Silouhette Score: 0.2771559448580024
Le meilleur nombre de cluster pour partition binaire est: 3
Accuracy pour Partitionnement Binaire: 0.78
```

Il y a trois clusters ce qui semble logique puisque nous avons plus de clusters que de valeurs pour la donnée cible (2).

Le score silhouette est à 0,27, ce qui signifie qu'un point est plus similaire à son groupe qu'au groupe voisin. Cela montre que l'algorithme est bon, puisque les éléments sont bien classés dans leurs groupes. (Il est appelé sur l'affichage "best silhouette score" parce qu'on le compare aux scores silhouettes utilisant un nombre de clusters différent.)

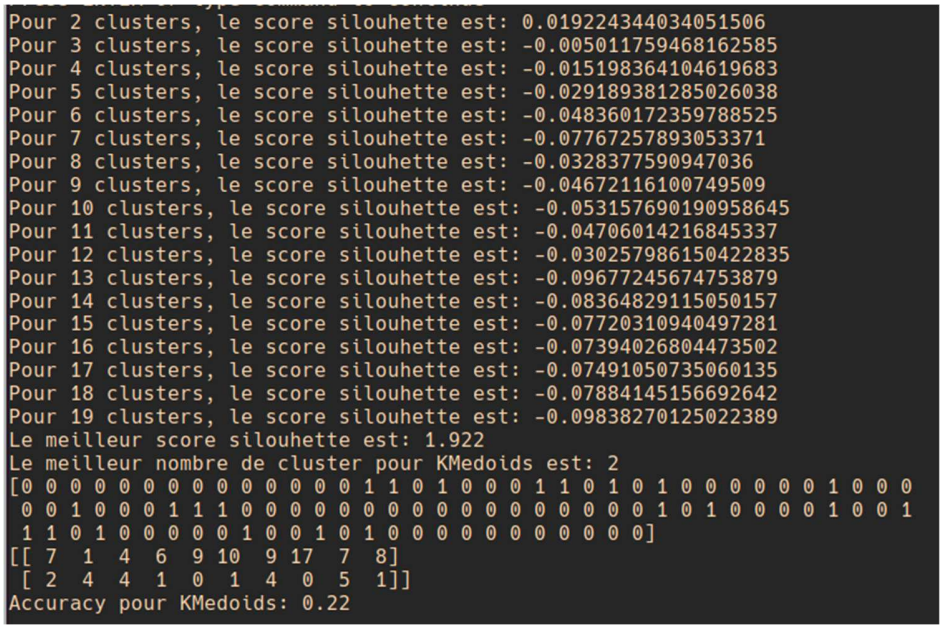
Le score d'accuracy est à 78%. Ce résultat est plutôt bon comparé à l'algorithme naïf, mais il est inférieur à celui qu'on a observé pour KMedoids.

Ainsi, le meilleur algorithme de clustering pour notre notion de similarité est donc KMedoids.

On obtient pour **MNIST** :

KMedoids

NB : Les ensembles training, test et validation sont de taille 100.

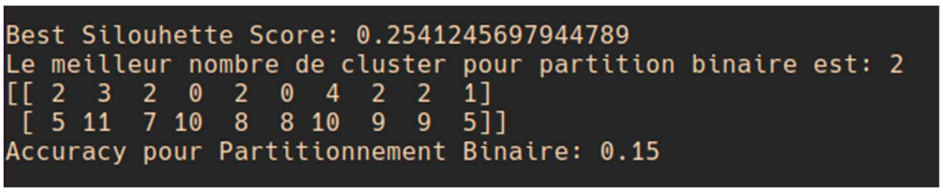


Le meilleur score silhouette est de 1,9 ce qui est très bon, pour les mêmes raisons qu'indiqué précédemment : les points sont bien classés dans leur groupe.

Cependant, le meilleur nombre de clusters est de deux. Cela est très étonnant. Notre donnée cible peut prendre 10 valeurs différentes. On s'attendrait à avoir un nombre de clusters plus grand ou égal à 10. Pourquoi a-t-on trouvé un nombre de clusters si étonnant ? C'est probablement parce que notre notion de similarité n'est pas bien adaptée. Puisqu'il y a deux clusters et que le score silhouette est très bon, cela signifie que notre similarité est à deux extrêmes : soit elle est très grande, soit elle est très petite. D'où l'agglomération des points dans deux clusters bien séparés.

Le score d'accuracy est de 22%. Ce résultat semble mauvais comparé au score d'ADULT et cela est sûrement dû au nombre de clusters trop petit. Mais même si ce résultat semble très mauvais, il est deux fois meilleur que l'algorithme naïf. C'est pourquoi nous pouvons dire qu'il est même plutôt bon !

Partitionnement binaire



Le score silhouette tourne ici autour de 0,25. Cela est moins bon que pour KMedoids, mais toujours correct. Il est positif donc cela signifie que les éléments appartiennent bien au bon groupe.

Le nombre de clusters est aussi étonnant ici puisqu'il est de deux. Cela s'explique encore une fois par les deux extrêmes de notre notion de similarité.

Le score d'accuracy est égal à 15%. Cela est meilleur que l'algorithme naïf, mais un peu moins bon que KMedoids.

Ainsi, les algorithmes de clustering utilisant notre notion de similarité sont meilleurs que l'algorithme naïf à 10%. Le meilleur algorithme de clustering est pour MNIST l'algorithme KMedoids. Pour les deux algorithmes, nous trouvons un nombre de clusters étonnant, mais cela peut s'expliquer par le fait que notre notion de dissimilarité tend vers les extrêmes.

4.3. Réduction de dimension : Isomap et PCoA

Pour les algorithmes de réduction de dimension, Isomap et PCoA, nous avons procédé de la manière suivante :

- On utilise les ensembles d'entrainement et de validation pour **trouver les meilleurs hyperparamètres**. Pour ce faire, on teste plusieurs combinaisons de paramètres avec lequel on entraine l'algorithme et on compare les scores obtenus sur un *classifier*. Nous avons choisi d'utiliser pour *classifier* un KNN classique avec les hyperparamètres trouvés précédemment pour maximiser le score d'*accuracy* de KNN.

- On entraine ensuite l'algorithme final avec les meilleurs hyperparamètres trouvés. On utilise alors nos données réduites dans main.py afin d'**évaluer la performance** de l'algorithme de réduction de dimensionalité. Pour cela, nous utilisons encore un classifieur sur nos données réduites et on calcule le score d'*accuracy*.

Pour ADULT.

Isomap

```
Meilleur training accuracy pour Isomap: 0.7666666666666667
Meilleur n_components pour Isomap: 2
Meilleur n_neighbors pour Isomap: 2
Classification accuracy pour Isomap sur ADULT: 0.8125
```

Le meilleur score d'accuracy pour l'ensemble d'entraînement est de 76%. Cela correspond à n_neighbors = 2, et à n_components = 2.

Ce score d'accuracy est plutôt moyen : il est égal à l'algorithme naïf. Cela est probablement dû au fait que nous enlevons des dimensions sur nos données, donc nous perdons en précision sur nos données d'entraînement.

Le meilleur hyperparamètre n_neighbors est deux. Isomap relie alors les 2 plus proches voisins entre eux.

Le meilleur hyperparamètre n_components est deux. Cela signifie que Isomap va prendre en considération deux caractéristiques pour déterminer la réduction de dimensionalité. Autrement dit, Isomap permettra de réduire nos données en deux dimensions.

Le score d'accuracy de la classification est de 81%. Même si le score sur l'ensemble d'entraînement est moyen, on observe un bon score de classification par rapport à l'algorithme naïf. Ainsi, Isomap est un bon algorithme de réduction de dimensionnalité avec notre notion de similarité.

PCoA

```
The best components is: 2
Classification Accuracy pour PCoA sur ADULT: 0.8125
```

Le meilleur hyperparamètre pour PCoA est 2. Cela signifie que PCoA réduit nos données de 9 dimensions en 2 dimensions. Après avoir réduit nos données et entraîné sur un classifieur KNN, nous obtenons un score d'accuracy autour de 81%, ce qui est nettement mieux que le score de 76% si on utilisait l'algorithme naïf.

Les deux algorithmes de réduction de dimensionalité sont aussi bons l'un que l'autre, et tous les deux meilleurs que l'algorithme naïf.

Pour MNIST.

Isomap

```
Meilleur training accuracy pour Isomap: 0.7666666666666667
Meilleur n_components pour Isomap: 8
Meilleur n_neighbors pour Isomap: 4
Isomap Accuracy: 0.4375
```

Le meilleur score d'accuracy pour l'ensemble d'entraînement est de 76%. Cela correspond à n_neighbors = 4, et à n_components = 8.

Ce score d'accuracy est très bon par rapport à l'algorithme naïf. Cela est probablement dû au fait que ce sont les données d'entraînement, donc on évalue la performance sur les valeurs avec lesquelles on a entraîné l'algorithme. Il n'est pas parfait puisque l'on réduit les dimensions de nos données et donc on perd en information et en précision, comme évoqué précédemment.

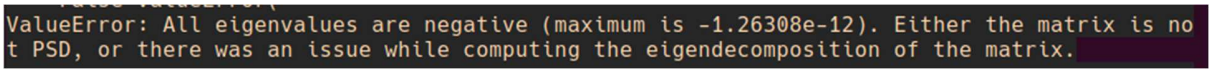
Le meilleur hyperparamètre n_neighbors est 4. Isomap relie alors les 4 plus proches voisins entre eux.

Le meilleur hyperparamètre n_components est 8. Isomap permettra de réduire nos données en huit dimensions. Ainsi on passe de 14x14=196 dimensions à 8. Ce changement est très grand et donc très efficace en gain d'espace, surtout pour des données massives comme MNIST.

Le score d'accuracy de la classification est de 43%. C'est un excellent score pour MNIST par rapport à l'algorithme naïf à 10%. Ainsi, Isomap est très bon avec notre notion de similarité.

PCoA

Nous n'avons pas réussi à faire fonctionner l'algorithme PCoA sur notre notion de similarité pour MNIST à cause de l'erreur suivante :



5. Conclusion

Nous avons pu établir deux notions de similarité pour deux jeux de données distincts et évaluer leur performance sur certains types d'algorithmes.

Concernant la notion de similarité que nous avons établie pour le jeu de données ADULT, nous avons pu observer qu'elle fonctionne bien pour les algorithmes de réduction de dimensionalité et les algorithmes de clustering, mais elle fonctionne moins bien pour l'algorithme KNN de classification.
Notre notion de similarité a l'avantage de prendre en compte beaucoup de caractéristiques du jeu de données (neuf caractéristiques). Cependant, elle n'est pas normalisée ni standardisée, d'où le fait qu'elle ne fonctionne pas très bien pour l'algorithme KNN.

Nous avons établi certains coefficients arbitrairement, et il aurait été mieux de se baser sur nos données pour les trouver. Par exemple, concernant le pays d'origine, nous avons utilisé une classification existante pour ranger nos données dans des catégories alors qu'il aurait été plus judicieux pour l'apprentissage supervisé de nous baser sur la classification faite par les données. Nous avons fait ce choix pour certaines des catégories, et non pas toutes, ce qui réajuste notre notion de similarité.

Pour le jeu de données MNIST, notre notion de similarité est très performante pour les algorithmes de réduction de dimensionalité (Isomap). Elle est aussi plutôt performante pour les algorithmes de clustering, mais elle ne l'est pas vraiment pour l'algorithme de classification KNN.

Notre notion de similarité possède les avantages d'être simple et de prendre en compte les translations selon les axes horizontal et vertical. Son plus grand désavantage est qu'elle semble tendre vers les extrêmes, d'où le nombre de clusters réduit à 2 pour les algorithmes de clustering.

Finalement, nous pouvons conclure que nos notions de similarité ne sont pas parfaites, mais elles sont toutes les deux meilleurs que les algorithmes naïfs. Cela montre qu'elles sont assez efficaces et précises.