Université de Montréal

Rapport du travail 1

Par
Geneviève Paul-Hus (20037331)
Abderrahim Tabta (20133680)

Faculté art et sciences

Travail présenté à Alain Tapp
Dans le cadre du cours IFT3700

Novembre 2021

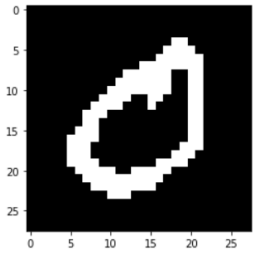
Introduction

À la suite de la lecture de l'énoncé, nous avons compris que la tâche principale était non seulement de trouver deux métriques pour MNIST et ADULT , mais aussi de comprendre la fonction de similarité , la matrice de dissimilarité et de comprendre comment fonctionnent les algorithmes que nous devions implémenter. Une grosse partie de notre travail était alors nos familiariser avec ces concepts, de préparer l'environnement python et d'organiser nos idées sur ce que nous voulions faire.

Notions de similarité

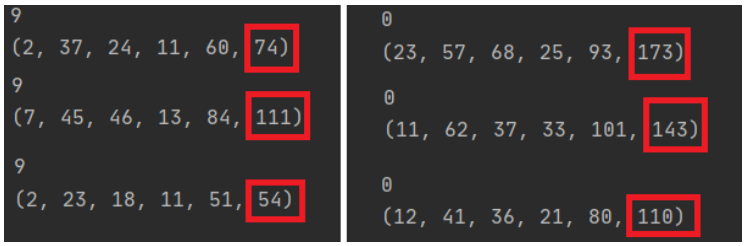
MNIST

Pour ce qui est de la base de données MNIST, nous avions eu, dès le départ, plusieurs idées. Mais nous nous sommes entendus sur le fait de standardiser nos données avant d'entamer nos calculs. En effet, les chiffres sont représentés dans le data set MNIST à travers des vecteurs de taille 784 (28*28 pixels) avec des valeurs variantes entre 0 et 254 soit les niveaux de gris. Donc, nous avions , selon l'intensité, converti toutes ces valeurs en 0 et 1 (blanc et noir) afin de faciliter nos calculs.



Notre première piste lors de la recherche de notre métrique était de chercher quelles informations nous pouvions tirer de nos vecteurs ayant comme valeurs 0 et 1. Le plus intuitif lors de notre première réflexion était simplement de compter le nombre pixels dans nos vecteurs et de comparer.

En effet, il était clair pour nous qu'un chiffre comme « 8 » requiert plus de pixels blancs (de 1) pour être représenté qu'un chiffre comme « 1 ». Il suffit alors de faire la différence entre le total de pixels blancs (1) des deux chiffres (vecteurs). Bien sûr, avant même de faire nos calculs, nous savions que c'était un indicateur loin d'être suffisant. Par exemple, dans la figure suivante, à gauche on voit dans les carrés rouges le nombre de pixels blanc pour plusieurs chiffres 9 et à droite le nombre de pixels blancs pour les chiffres 0. Parfois, dans nos données, nous pouvions tomber sur des heureuses coïncidences, mais le reste du temps c'est assez hasardeux comme on peut le remarquer :



Note : les 4 premières valeurs dans les tuples sont le nombre des pixels dans chacune des 4 sous-matrices (expliqué plus tard) et la 5ème valeur est le périmètre du chiffre (qu'on va expliquer aussi).

C'était un peu dommage, car nous aimions cette idée de compter les pixels et nous avions voulu l'inclure dans notre métrique. Le problème vient probablement du fait que les mêmes chiffres peuvent être représentés avec plus ou moins de pixels dépendamment de leurs « tailles ». En effet, rien nous n'empêche d'écrire un chiffre « 1 » plus gros qu'un autre « 1 » . Nous nous sommes alors demandé si c'était possible d'avoir un ratio qui prend compte la taille. La partie était loin d'être perdue, puisque nous avions eu comme autre idée de calculer le périmètre des chiffres. Autrement dit, calculer uniquement le nombre de pixels sur le frontière des chiffres. La stratégie la plus simple pour procéder était simplement de regarder autour de chaque pixel blanc s'il y'a au moins un pixel noir. Si oui, alors ce pixel est sur la frontière, donc dans le périmètre. L'avantage de calculer le périmètre en plus était de pouvoir faire un ratio entre le périmètre et le nombre de pixels total, ce qui va prend en compte le fait que certains mêmes chiffres sont représentés avec plus ou moins de pixels. Par exemple, si le chiffre "4" a une frontière de 20pixels et un total 40pixels en règle générale. Si on tombe sur un chiffre "4" avec une frontière de 40 pixels et un total de 80 pixels, le rapport reste le même.

C'est un bon début, mais nous voulions savoir quelles autres informations nous pourrions tirer de nos « vecteurs ». Rappelons que notre première idée était de comparer la somme de tous les pixels blancs d'un « chiffre » à un autre. Rappelons aussi que chaque chiffre est représenté par un vecteur de 784 pixels, soit une matrice de 28x28 pixels. L'idée que nous avons eue était de «diviser pour régner» . En termes plus clairs, nous avions séparé notre matrice en 4 sous matrices égales, soit des matrices de dimension 14x14 pixels. Nous avions ensuite calculé le nombre de pixels blancs dans chacune des sous-matrices. Ainsi au lieu de comparer le total de pixels blancs d'une matrice à une autre, on compare les sous-matrices une à une. L'avantage de procéder ainsi vient du fait que , selon nous, Il est fort probable qu'un chiffre comme « 0 » , par exemple, aura tendance à avoir ses pixels bien répartis dans les 4 sous matrices. Tandis qu'un chiffre comme « 6» aura plus de pixels dans les deux sous-matrices en bas. Ainsi de suite. Donc, en comparant les sous-matrices entre elles, nous pouvons remarquer ces similarités.

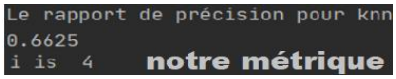
Ensuite, nous avions consulté le PDF « distance et similarité ». Nous avions alors essayé de voir quelle distance autre que la distance euclidienne pourrait donner des résultats raisonnables. Notre attention s'est arrêtée sur la distance d'édition et Hamming. En effet, on voyait les choses comme suit : Nos vecteurs dans notre data set ressemblent à ça : [0, 1, 0, …, 0], ce qui ressemble fortement à des chaines de caractère de 0 et 1. Il était alors clair qu'on pouvait facilement calculer la distance entre ces chaines de caractère. Reprenons l'exemple Hamming du PDF, la distance entre la chaine « 1101 » et « 1111 » est 1, puisqu'il suffit de changer un 1 en 0 pour que les deux chaines soient égales. Il suffit alors de généraliser ce raisonnement à nos vecteurs et ainsi nous pouvons savoir combien de pixels il faut changer pour que le chiffre représenté dans un vecteur soit égal à un autre. Nous avions jugé qu'il était inutile d'implémenter la distance d'édition , puisque Hamming était suffisant pour nos vecteurs standardisés.

En conclusion , **notre métrique** était une combinaison de tout ce qu'on a discuté précédemment. En effet, on prend en compte le ratio (nombre de pixels dans le périmètre / nombre pixels total), on prend en compte la distance euclidienne du total de pixels dans chacune de nos quatre sous-matrices et on prend en compte aussi la distance de Hamming.

De plus, nous avions mis en place une notion de poids. En effet, on a fait des comparaisons pour voir ce qui influence le plus notre métrique (sous-matrices, périmètre ou Hamming) et on a accordé plus d'importance à ce qui varie le plus dans notre métrique.

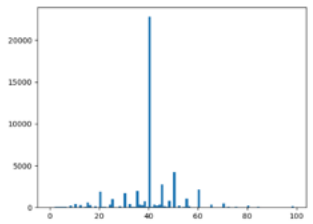
Enfin, avec cette fonction de similarité et après avoir séparé notre jeu de données comme montré dans la démo, nous avions pu calculer la matrice de dissimilarité manuellement pour comprendre comment elle fonctionne. Les résultats respectent la positivité, la symétrie, la séparation et l'inégalité triangulaire puisque notre matrice de dissimilarité est symétrique avec des 0 sur la diagonale.

En conclusion, notre métrique apporte peu d'avantages puisqu'elle est moins efficace que la distance euclidienne, voici les résultats de la comparaison pour l'algorithme KNN utilisant notre data set réduit: 0.775 pour distance euclidienne et 0,6625 pour la nôtre.

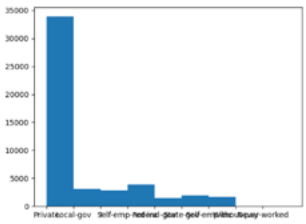


ADULT

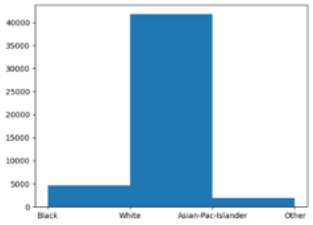
Pour ce qui est de la base de données ADULT, nous avons décidé d'essayer de prédire le genre (homme ou femme) des personnes représentées dans la base de données. Pour ce faire, nous avons effectué plusieurs analyses sur les colonnes pour déterminer quelles étaient les moins pertinentes afin de les retirer de notre modèle. Voici les graphes des colonnes qui nous semblait les moins utiles :



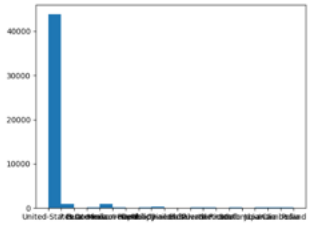
Graphe 1. Colonne hours-per-week



Graphe 2. Colonne workclass



Graphe 3. Colonne race



Graphe 4. Colonne native-country

Les colonnes « hours-per-week », « workclass », « race » et « native-country » n'avaient pas assez de variétés pour être considérées, ces colonnes n'auraient pas particulièrement aidé à faire une prédiction. Dans le cas de la colonne « hours-per-week », plus de la moitié des personnes travaillent 40 heures par semaine. Dans le graphe de la colonne « workclass », la majorité est dans le privé. Et en analysant les colonnes « race » et « native-country », on remarque rapidement que la majorité des personnes dans la base de données sont des caucasiens provenant des États-Unis. Donc ces quatre colonnes n'avaient tout simplement pas assez de variation pour les considérer dans notre modèle. Nous avons aussi retiré les colonnes « capital-gain » et « capital-loss » puisqu'on a jugé qu'il n'y avait pas assez d'information pour les utiliser.

Suite au retrait des colonnes qu'on a jugé non pertinentes, nous avons analysé l'importance des autres colonnes par rapport à notre cible, c'est-à-dire le « genre », avec un arbre de décision et avons reçu ces résultats :

```
[('age', 0.11788247612203237),
 ('educational-num', 0.06510568791986847),
 ('occupation', 0.160964476645311),
 ('relationship', 0.6087062686895065),
 ('income', 0.008985138249881857),
 ('marital-status', 0.038355952373399706)]
```

Nous avons alors décidé de retirer la colonne « income » qui avait le plus bas score de représentation. Nous avions donc nos colonnes : « age », « educational-num », « occupation », « relationship » et « marital-status ».

Notre fonction de similarité constitue à donner des points entre 0 et 1 pour chaque colonne, entre une personne *X* et une personne *Y*. Pour les colonnes « age » et « educational-num », il suffisait de calculer 1 – leur distance euclédienne entre *X* et *Y*. Mais pour les colonnes « occupation », « relationship » et « marital-status » nous devions trouver une meilleure façon de les catégoriser puisque seulement les changer en valeur numérique ne serait pas optimal. Nous devions trouver une façon de trouver leurs similarités. Nous avons alors créé une autre colonne pour chacune de ces colonnes (« occupation », « relationship » et « marital-status ») que nous avons transformées en valeur numérique. Si la valeur de cette colonne pour une personne *X* et *Y* est identique alors on accorde un point de valeur 1, sinon c'est 0. Nous avons par la suite essayé de calculer leur similarité en divisant les colonnes par catégories. Pour la colonne « occupation », nous avons accordé des catégories numériques allant des emplois les plus physiques aux plus intellectuels, puisqu'il y a une certaine division entre les types d'emplois et le genre des personnes. Ceci nous donnait 3 catégories, 0 étant très physique et 2 étant plus intellectuel :

```python
# Occupation classification. From physical (0) to intellectual (2)
occupation_classification = {
    "Adm-clerical": 2,
    "Armed-Forces": 0,
    "Craft-repair": 1,
    "Exec-managerial": 2,
    "Farming-fishing": 0,
    "Handlers-cleaners": 0,
    "Machine-op-inspct": 1,
    "Other-service": 1,
    "Priv-house-serv": 1,
    "Prof-specialty": 1,
    "Protective-serv": 0,
    "Sales": 2,
    "Tech-support": 2,
    "Transport-moving": 1
}
```

Pour la colonne « marital-status », nous avons fait la même chose, nous les avons classifiés selon leur état civil présent, étant dans une union ou non. Donc ceci nous a donné deux catégories :

```
# Marital status classification
marital_status_classification = {
    "Divorced": 1,
    "Married-AF-spouse": 0,
    "Married-civ-spouse": 0,
    "Married-spouse-absent": 1,
    "Never-married": 1,
    "Separated": 1,
    "Widowed": 1
}
```

Et nous avons répété la même formule pour la colonne « relationship ». On a mis un 0 pour les personnes n'ayant personne dans leur vie à 2 pour les gens qui sont mariés. 1 représente les autres types de relations ainsi que d'avoir un enfant puisqu'il ne faut pas nécessairement être marié pour avoir un enfant.

```
# Relationship classification
relationship_classification = {
    'Unmarried': 0,
    'Wife': 2,
    'Husband': 2,
    'Not-in-family': 0,
    'Own-child': 1,
    'Other-relative': 1
}
```

Une fois ces colonnes catégorisées, on les divise chacune par leur plus grande valeur pour nous donner une valeur entre 0 et 1, à l'exception des valeurs « ? » de la base de données sous la catégorie « occupation » où on lui donne directement le score de 0.5 puisqu'on ne connaît pas la valeur et qu'il pourrait être similaire à tout ou rien. Ensuite on effectue 1 - la distance euclidienne pour ces colonnes entre une personne *X* et une personne *Y*.

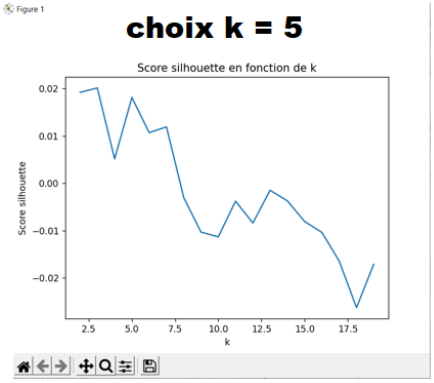
Une fois que les calculs sont faits, on les additionne ce qui nous donnera un score entre 8, étant un score parfait, et 0, représentant une dissimilarité complète.

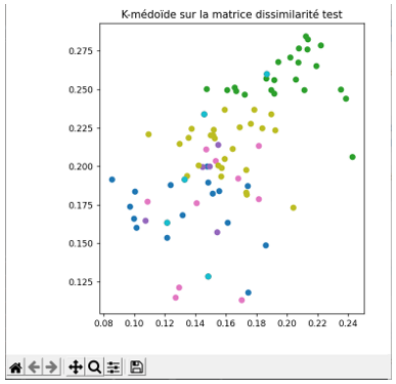
Une des faiblesses de notre fonction est qu'on n'ait pas utilisé toutes les colonnes donc il pourrait y avoir des scores manquant au niveau de la similarité, aussi tel que mentionné plus haut, on accorde 0.5 de point aux occupations « ? » alors qu'il pourrait être très dissimilaire ou similaire. La façon dont on a catégorisé les colonnes « occupation », « marital-status » et « relationship » sont plutôt subjectives puisque quelqu'un d'autre qui écrirait cette fonction pourrait les classer autrement. La colonne « relationship » n'est pas la meilleure colonne non plus n'ayant aucune description claire et précise.

Une des forces de notre algorithme est qu'on a aussi à déterminer des points communs afin de catégoriser les occupations, si ce n'était que de la transformation de données en valeur continue alors on n'aurait pu alors vraiment vérifier les similarités. Donc en comparant deux colonnes par leur type de classification et en ajoutant un point si les valeurs sont identiques, on spécifie beaucoup plus le résultat à savoir si elles sont similaires ou non. En utilisant la distance euclidienne pour les valeurs continues comme « educational-num » et « age », on trouve une valeur directe pour la similarité, ce qui nous permet de venir préciser encore plus notre résultat.

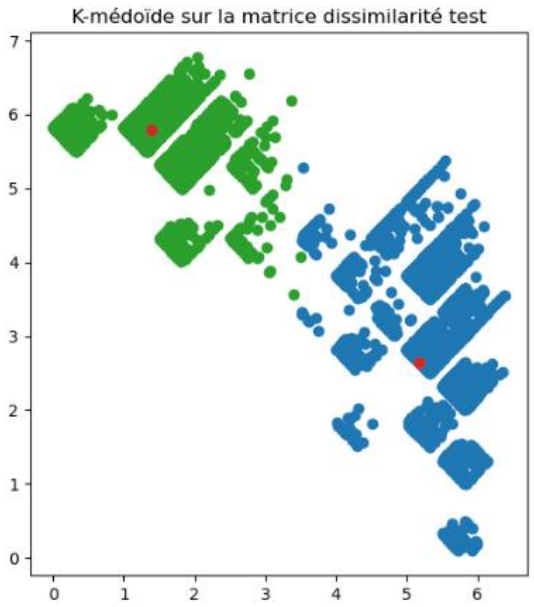
K-médoïde

MNIST





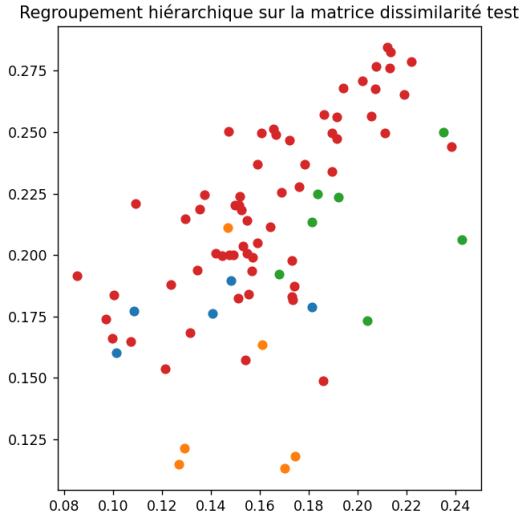
ADULT



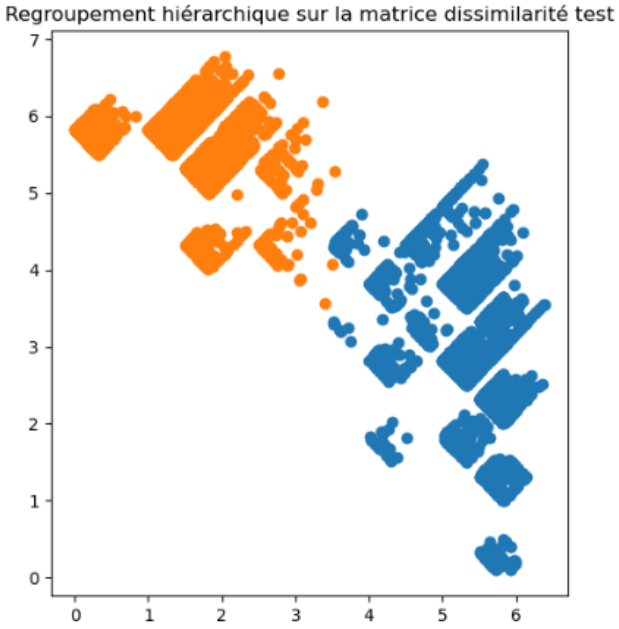
Les points rouges est la distance moyenne entre les deux sections. Puisqu'il n'y a pas de point entre

Partition binaire

MNIST

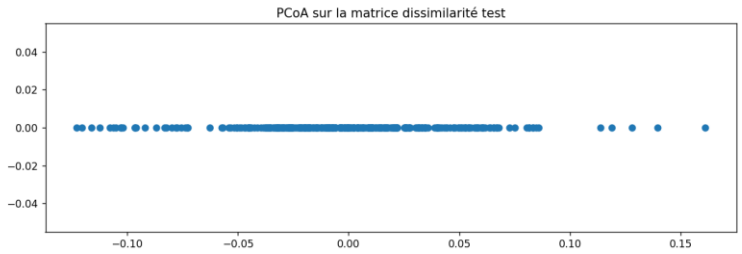


ADULT

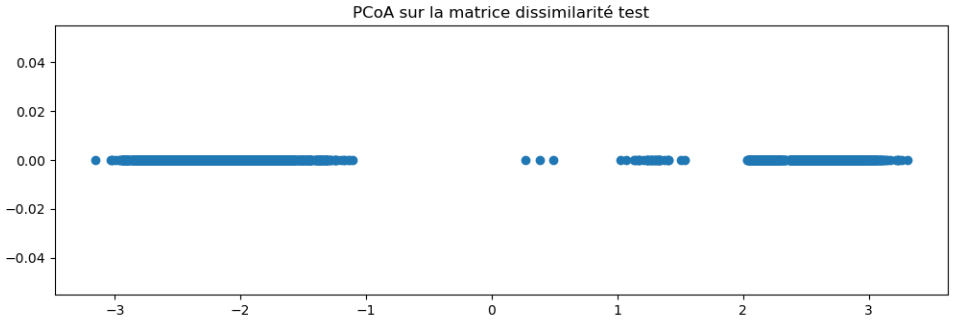


PCoA

MNIST

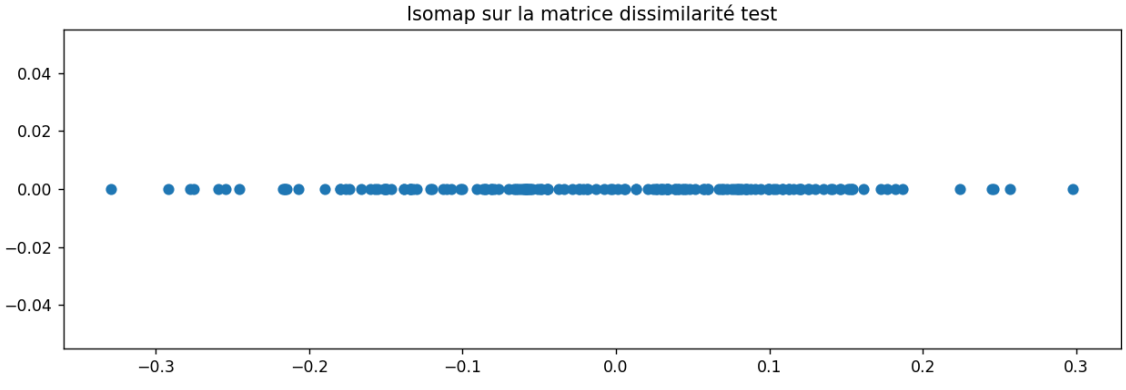


ADULT

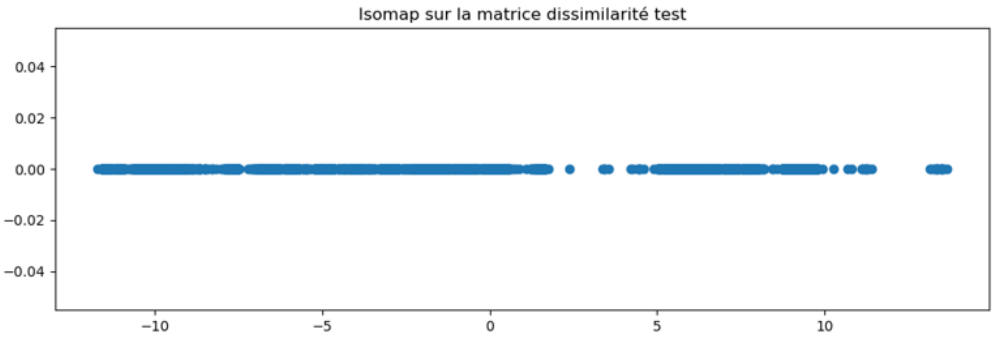


Isomap

MNIST

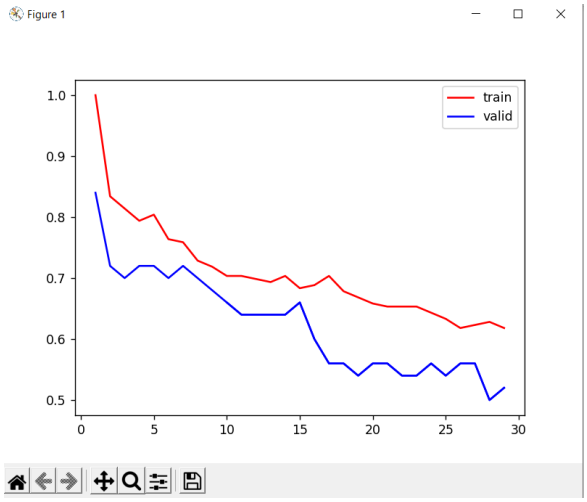


ADULT

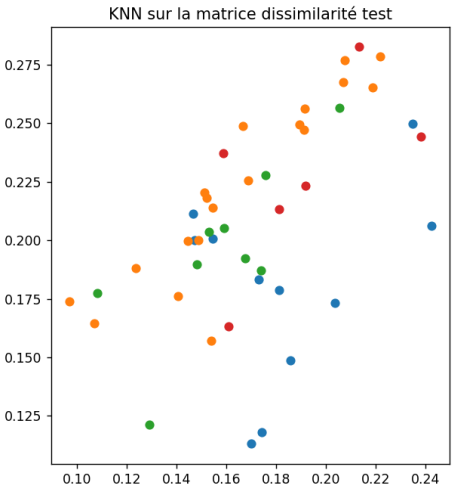


KNN

MNIST



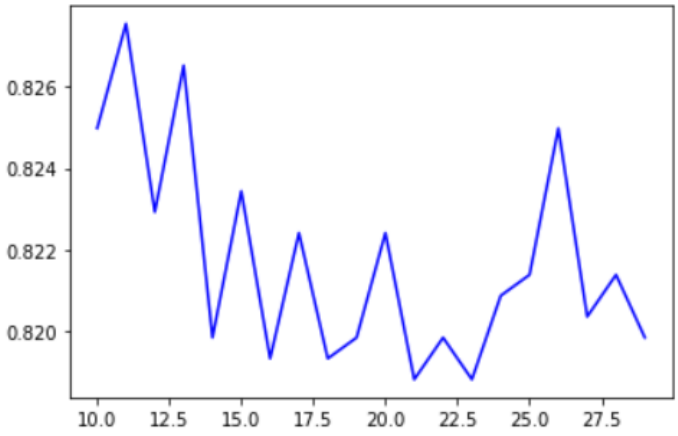

```
i is  4
Le rapport de précision pour knn
0.6625
i is  5
Le rapport de précision pour knn
0.6
```



KNN nous donne un score de 0.6625

ADULT

Avec KNN, nous avons pris un nombre de voisins de valeur 11. Puisque c'était le nombre optimal lorsque nous avons lancé l'algorithme qui nous a donné ce graphique :



KNN nous donne un score de 0.8345957011258957.