Université de Lille

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

Clustering de données numériques et catégorielles

Réalisé par :
Audrey Bilon

Encadrants :
Adan JOSE GARCIA
Julie JACQUES
Clarisse DHAENENS

June 2023

Contents

1 Résumé

Les dossiers de santé électroniques sont importants à étudier en milieu hospitalier pour différencier les patients en fonction de leurs attributs, les regrouper en cluster et adapter les traitements de chaque patient en fonction du cluster où ils sont.

Pour cela, nous utilisons des algorithmes de clustering. Le problème que nous rencontrons est que les données médicales sont mixtes et que parmi les algorithmes officiels, il n'en existe qu'un traitant de clustering. Ce dernier ne s'occupe que de données numériques.
Notre objectif ici est de coder un algorithme capable de s'occuper aussi bien des données catégorielles que numériques et de pouvoir l'utiliser pour prédire dans quel cluster sera chaque patient.

Pour cela, nous coderons K-Means qui gère les données numériques, K-Modes les données catégorielles et K-Prototype qui s'occupera des deux, nous verifierons la véracité des résultats à l'aide des métriques que nous coderons. Le lien du dépôt où les fonctions sont codées est : le lien du dépôt .

2 Introduction

Une population peut être mieux gérer lorsqu'elle est divisée en clusters. Ceux-là sont choisis en fonction des attributs que leurs individus possèdent. Plus ces personnes sont similaires et plus ils ont de chance d'appartenir au même groupe.

Les algorithmes de clustering permettent de créer des partitions liant chaque individu ou échantillon à son cluster. L'algorithme peut être réalisé avec des attributs numériques grâce à la fonction K-Means de la librairie python nommée scikit-learn. Cependant, aucune fonction n'existe pour réaliser cela avec des données comportant des attributs catégorielles et mixtes.

Comment implémenter ces algorithmes?

Nous verrons quelles méthodes ont été pensées pour réussir à implémenter l'algorithme de clustering avec divers types de données et pourquoi nous ne les choisissons pas ici. Puis nous présenterons les algorithmes implémentés pour les données numériques, catégorielles et mixtes. Nous observerons ensuite l'exactitude de ces algorithmes.

3 Algorithmes de clustering

Les dossiers de santé électroniques sont importants à étudier en milieu hospitalier pour différencier les patients en fonction de leurs attributs, les regrouper en cluster et adapter les traitements de chaque patient en fonction du cluster où ils sont. Pour cela, nous utilisons des algorithmes de clustering. Le problème que nous rencontrons est que les données médicales sont mixtes et que parmi les algorithmes officiels, il n'en existe qu'un traitant de clustering. Ce dernier ne s'occupe que de données numériques. Notre objectif ici est de coder un algorithme capable de s'occuper aussi bien des données catégorielles que numériques et de pouvoir l'utiliser pour prédire dans quel cluster sera chaque patient.
Pour cela, nous coderons K-Means qui gère les données numériques, K-Modes les données catégorielles et K-Prototype qui s'occupera des deux.

3.1 Diverses méthodes

L'algorithme de clustering est un algorithme permettant de classer des objets dans divers clusters. Celui-ci est efficace car nous n'avons aucune information sur les sorties des données en entrée. Ainsi, nous nous aidons des similitudes entre les objets pour obtenir la sortie, autrement nommé ici label.

Un algorithme de clustering comporte forcément ces trois variables en entrée:
- **data**, correspondant aux données que nous allons étudier.
- **K**, le nombre de clusters que nous voulons obtenir.
- **max_iter** correspondant à l'itération maximale à partir de laquelle nous pourrons nous arrêter si l'algorithme ne le fait pas par lui-même avant.

Data possède *n lignes* qui correspond au **nombre d'échantillons** dans l'ensemble et *d colonnes* correspondant au **nombre d'attributs** que possèdent chaque échantillon.

Les algorithmes de clustering que nous verrons fonctionnent tous de cette façon:

1. Nous posons aléatoirement un centre
2. Nous associons chaque échantillon à un cluster
3. Nous recalculons le barycentre de chaque cluster pour en faire le nouveau centre

Et nous répétons l'étape 2 et 3 jusqu'à ce que les centres soient stables ou que le nombre d'itérations soit suffisant pour l'utilisateur.

Nous verrons donc les algorithmes K-Means, K-Modes et K-Prototype qui utilisent ce fonctionnement.

3.2 Algorithmes implémentés

3.2.1 K-Means

K-Means est une fonction de clustering permettant de répartir les objets dans divers clusters en fonction des valeurs que prennent les attributs. Ces dernières sont numériques.

Nous avons rencontré des difficultés différentes en fonction de la façon de choisir le premier centre.

Lorsque nous choisissons les centres de façon aléatoire grâce à la fonction Random Centers, en fonction de la seed, il peut y avoir des clusters vides. Ce qui n'est pas le cas avec la fonction Mean Centers, qui assure dès le début que le cluster n'est pas vide.

Ici, nous prenons K points aléatoires choisis uniformément tel que pour chaque attribut la valeur obtenue se situe entre les valeur maximale et minimale prises par les points.

Algorithm 1 Random Centers: K, data

```
return [np.random.uniform(data.min(axis=0),data.max(axis=0)) for i in
range(K)]
```

MeanCenter a été programmé pour découper en K parties notre échantillon et y calculer la moyenne de chaque champs de ces K parties pour obtenir nos K centres.

Algorithm 2 Mean Centers: K, data

```
center ← []
for tous les centres de clusters do
    center.append(data[i*data.shape[0]//K:(i+1)*data.shape[0]//K].mean(
    axis=0))
end for
return [np.random.uniform(data.min(axis=0),data.max(axis=0)) for i in
range(K)]
```

Notre algorithme K-Means a donc été codé pour pouvoir choisir les centres initiaux de diverses façons grâce à l'argument function_initial_centers. Celui-ci pourra prendre Random Centers ou Mean Centers comme valeur.

Algorithm 3 K_Means: K, data, max_iter, function_initial_centers

$c \leftarrow 0$
$centers \leftarrow function\_initial\_centers(K, data)$
while les centres ne sont pas stables ou que nous n'avons pas atteint le maximum d'itérations **do**
  $cluster \leftarrow$ dict((i,[]) for i in range(K))
  $label \leftarrow$ []
  for tous les points dans data **do**
    $clust \leftarrow$ (None, float('inf'))
    for tous les clusters **do**
      On calcule la distance du point au cluster
      if La distance au cluster est plus proches que des autres **then**
        $clust \leftarrow (numero\_cluster, distance\_au\_cluster)$
      end if
    end for
    cluster[clust[0]].append(le point étudié)
    label.append(clust[0])
  end for
  On calcul les nouveaux centres : moyennes de chaque cluster
  On compare les nouveaux centres aux anciens pour voir si on continue la boucle
end while
return (centers,cluster,label)

3.2.2 K-Modes

K-Modes est une fonction de clustering permettant de classer les données lorsqu'elle ne comporte que des données catégorielles.

Pour les manipuler, nous avons codé les fonctions auxiliaires suivantes:

1. **Dissimilarity nous permet de calculer la distance entre deux points.**
Elle joue le rôle de la fonction euclidienne dans K-Means. Nous ajoutons 1 à la dissimilarité à chaque fois que les valeurs d'un attribut pour les deux points à comparer sont différentes.

Algorithm 4 Dissimilarity : Point1, Point2

```
return sum([1 for k in point1.index if point1[k] != point2[k]])
```

2. **K_Mode_Centers nous permet de choisir les K centres initiaux de l'algorithme.** [2]
Pour cela nous utilisons une méthode dans laquelle nous réalisons un tableau dans lequel nous classons pour chaque attribut les valeurs classées dans l'ordre décroissant de nombre de fois où elles apparaissent pour tous les objets. Ce tableau correspond à values, qui contient les valeurs par attribut sous forme de sous-tableaux.

Ce tableau nous permet ensuite de choisir les centres initiaux en prenant les valeurs d'attributs les plus fréquentes tout en faisant en sorte que les centres ne soient pas les mêmes et qu'il permette de créer des clusters distincts.

Pour cela nous prenons les valeurs de values, de façon à faire une diagonale, si l'indice est trop élevé par rapport aux nombres de valeurs de l'attribut concerné. Nous prenons la première valeur la plus fréquemment rencontrée. Cette méthode nous permet de créer un centre. Nous incrémentons de 1 chaque indice de la valeur précédemment prise par attribut pour choisir la valeur du centre suivant.

Algorithm 5 K_Modes_Centers : K, data

values ← []
for tous les attributs de data **do**
possibility ← dict((i,len(data[data[k]==i])) for i in list(set(data[k])))
On ajoute à values les valeurs prises par l'attribut sous forme de liste trié par ordre apparitions croissantes
end for
centers ← []
length_table ← [len(liste) for liste in values]
center_indices ← [i if i < length_table[i] else 0 for i in range(len(values))]
for tous les clusters **do**
On ajoute à centers la valeur des attributs que nous choisissons pour le centre actuelle
On ajoute 1 à tous les indices contenus dans center_indices, si l'indice est trop grand on le met à 0
end for
On transforme centers en dataframe
return centers

3. **New_Center_KModes nous permet de recalculer les nouveaux centre de l'étape 3.** Pour cela, nous prenons les valeurs les plus fréquentes par attribut dans un cluster et cela forme le nouveau centre du cluster.

Notre fonction ne renvoit que le centre d'un cluster.

Algorithm 6 New_Center_KModes: cluster

center ← []
for tous les éléments de data **do**
On ajoute à center les attributs qui apparaissent le plus fréquemment dans le cluster
end for
On transforme center en dataframe
return center

Nous pouvons remarquer que nous renvoyons à chaque fois des données sous forme de dataframes. Cela est lié au fait que les données en entrée sont souvent sous forme de dataframes lorsque les attributs sont catégoriques. Nous utilisons des noms de colonnes permettant de savoir à quel attribut est associé quelle valeur.

Ainsi grâce aux fonctions auxiliaires, nous pouvons codé K_Modes en utilisant les 3 étapes vues précédemment.

Ici, minimum_center représente le centre le plus proche du point. Nous associons donc le point étudié au cluster associé à ce centre.

Algorithm 7 K_Modes: K, data, max_iter

centers $\leftarrow$ $K\_Modes\_Centers$(K,data)
while On itère pas max_iter fois ou que les centres ne sont pas stables **do**
label $\leftarrow$ []
clusters $\leftarrow$ dict((i,pd.DataFrame(columns=data.columns)) for i in range(K))
for tous les points dans data **do**
minimum_center $\leftarrow$ min([(dissimilarity(centers.iloc[i],data.iloc[d]),i) for i in range(K)])
On ajoute au bon cluster le point étudié
On ajoute le label du point étudié dans label
end for
new_centers $\leftarrow$ pd.concat([New_Center_KModes(cluster[i]) for i in range(K)],ignore_index=True)
On compare les anciens centres aux nouveaux pour voir si nous pouvons continuer la boucle
end while
return (centers,clusters,label)

3.2.3 K-Prototypes

K-Prototype est une fonction de clustering permettant de classer les données **catégorielles et numérique**s.

Pour pouvoir l'utiliser correctement, nous devons **préparer le dataset** de la façon suivante:
1. Préparation des données pour ne plus avoir de 'NaN' dans les colonnes de données numériques.
2. Mettre le plus à gauche du tableau les données numériques et le plus à droite les données catégorielles
3. Normaliser les données catégorielles.

L'étape 1 est essentielle pour pouvoir calculer la moyenne car NaN n'est pas considérer comme un nombre.
Pour obtenir la valeur numérique de NaN nous allons calculer sa moyenne.
Nous chercherons donc à voir quelles données possèdent un élément NaN parmis ses attributs numériques, puis remplacer cette données par la bonne valeur grâce à la fonction suivante :

Algorithm 8 Preparation_Data : Data

```
numericalKeys ← []
for attribut de Data do
  if 1er élément de Data avec cette attribut est numérique then
    Ajout de cet attribut dans numericalKeys
  end if
end for
for tous les éléments de Df do
  if l'élément possède un NaN parmis ses attributs then
    for tous les attributs catégoriques do
      if l'attribut vaut NaN then
        On remplace cet attribut de Data par la valeur moyenne de cet
        attribut pour les autres objets
      end if
    end for
  end if
end for
return Data
```

L'étape 2 nous permet d'obtenir un dataset plus simple à manipuler tandis que l'étape 3 nous permet de mieux calculer les clusters, comme les valeurs sont moins éparpillées il est plus facile de les regrouper.

Lors de l'implémentation de K**-Prototype**, nous avons codé la fonction auxiliaire *new_center_KProt*. Cette fonction nous a permis à partir d'un cluster de calculer le **nouveau centre** en prenant le barycentres de chaque attribut.

Pour cela nous avons choisis de calculer la moyenne de tous les champs numériques tout d'abord, puis d'ajouter la valeur apparaissant le plus souvent dans chaque champ comme valeur représentative du champ.

L'algorithme sera implémenté de la façon suivante :

Algorithm 9 New_Center_KPrototype: cluster,numeric_keys,categoric_keys

center ← [cluster[k].mean() for k in numeric_keys]
for toutes les clés d'éléments catégoriques **do**
  values ← dict((i,cluster[cluster[k]==i].shape[0]) for i in set(cluster[k]))
  On ajoute à la liste des centres la valeur la plus frequemment prise (celle avec le nombre le plus élevé en value du dictionnaire)
end for
On transforme center en dataframe
return center

Une fois la fonction réalisée, nous pourrons coder K-Prototype.

Pour cela nous avons tout d'abord réaliser l'étape 2, en créant deux tableaux. L'un est composé des clés d'éléments catégorielles et l'autre des clés d'éléments numériques.
Nous pourrons donc concaténer les tableaux obtenus grâce aux clés afin d'obtenir un DataFrame ordonné.
Nous réaliserons ensuite l'étape 3 de la normalisation.

Nous calculerons ensuite les K centres aléatoires en utilisant des indices choisis aléatoirement.

L'étape 2, nous permettra de **calculer plus facilement les distances entre les points du DataFrame et les centres**, qui sont sous forme de Séries.
La distance de KPrototype est calculé comme un mixte entre la distance de KMeans et KModes.

La méthode est la suivante :
1. Nous calculons la distance des attributs numériques en calculant la norme de la différence des deux points.
2. Nous calculons celle des attributs catégoriques en utilisant la fonction dissimilarity de KModes 3.2.2.
3. Nous appliquons à la distance des attributs catégorielles un coefficient γ, puis nous sommons la multiplication par la distance des attributs numériques.

Nous pouvons résumer les étapes avec la formule suivante : [1]
On pose pointA et pointB, deux points sous forme de séries.
$Distance = Distance\_KMeans(pointA[cles\_numeriques] - pointB[cles\_numeriques]) + \gamma * Dissimilarity(pointA[cles\_categoriques], pointB[cles\_categoriques]$

Cette formule sera codé dans la fonction gamma_distance :

Algorithm 10 gamma_distance : pointA,pointB,numericalKeys, categoricalKeys

return $np.linalg.norm(np.array(pointA[numericalKeys]) - np.array(pointB[numericalKeys])) + gamma * dissimilarity(pointB[categoricalKeys], pointA[categoricalKeys])$

Grâce à ces éléments nous pouvons coder K-Prototype :

Algorithm 11 KPrototype : K, data, max_iter,gamma

numberKeys ← []
categoricalKeys ← []
for les attributs de data **do**
  if l'attribut de data est numérique **then**
    Ajout de cet attribut dans *numberKeys*
  end if
  if l'attribut est catégorique **then**
    Ajout de cet attribut dans *categoricalKeys*
  end if
end for
categorical ← data[categoricalKeys]
numerical ← pd.DataFrame(minmax_scale(np.array(list(list(data[k]) for k in numberKeys))).T,columns=numberKeys)
dataset ← pd.concat([numerical,categorical],axis=1)
centers ← dataset.iloc[randint(0,data.shape[0],K)]
while les centres ne sont pas stables ou le nombre d'itération n'est pas dépasser **do**
  label ← []
  cluster ← dict((i,pd.DataFrame(columns=(numberKeys+categoricalKeys))) for i in range(K))
  for tous les éléments de data **do**
    value_clusters ← []
    for tous les clusters **do**
      distance ←On calcule la distance du point au centre du cluster
      On ajoute à value_clusters cette distance avec le numéro du cluster associé
    end for
    On calcule pour pour quel centre la distance est la plus petite en prenant la valeur minimale de *value_clusters*
    On prend la valeur du numéro du cluster associée pour l'ajouter dans le bon cluster du dictionnaire *cluster*
    On rajoute le numéro du cluster dans le *label*
  end for
  new_centers ← On recalcule les centres grâce à *New_Center_KPrototype*
end while
return (*new_centers*,*clusters*,*label*)

3.3 Résultats expérimentales

3.3.1 Présentation des données utilisées

Lors du lancement des algorithmes, nous avons utilisé les données suivantes:

1. **Iris Dataset**

Iris Dataset est un dataset contenu dans la librairie Scikit-learn. Elle possède 4 attributs, qui représentent les longueurs et largeurs des sépales et des pétales observées.
Ce dataset contient 50 échantillons, qui contiennent 3 espèces d'iris différentes : des Setosa, Versicolour et Virginica.
Elle contient ici, 3 classes, visible dans le champs "target" de Iris.
Comme Iris Dataset ne contient que des nombres, il ne peut pas être utilisé par KModes. Nous l'utiliserons ici uniquement pour KMeans.

2. **Bank Dataset**

Bank Dataset est un dataset obtenu sur le site d'UCI.
Elle possède 16 attributs de 45 211 échantillons, il y a beaucoup de données catégorielles, seul un attribut est numérique.
Le dataset représente les caractéristiques des clients en fonction des réponses d'un appel pour la campagne marketing d'une banque portugaise. Ce dataset nous permet de voir si le client a souscrit au contrat. Nous avons donc de quoi comparer les données obtenus par nos algorithmes.
Nous formerons un cluster regroupant les individus n'ayant pas souscrit au contrat et un autre l'ayant fait.

3. **Credit Approval Dataset**

Nous avons obtenu ce dataset sur le site d'UCI.
Elle possède 15 attributs et 690 échantillons, elle possède plus de données catégoriques que de données numériques. Mais ces dernières sont à peu près équilibrées.
Credit Approval Dataset représente les données de demandes de cartes de crédit.
Nous ne pouvons rien déduire des informations car les noms d'attributs et leurs valeurs ont été changé pour garder les informations confidentielles.

Cependant l'attribut "y" nous montre qu'il y a 2 classes, l'une se nommant '+' et l'autre '-'. Nous utiliserons cette fonction avec 2 clusters.

4. **Vote House Dataset**

Nous avons obtenu ce dataset sur le site d'UCI.
Ce dataset possède 16 attributs et 435 échantillons. Les attributs sont tous catégoriques.
Vote House dataset représente les votes sur divers sujets des 435 membres du Congrés des Etats-Unis.
Ici, nous chercherons avec les algorithmes implémentés deux clusters. Pour les comparer à l'attribut "Class Name" qui définit si le votant appartient au partie démocrate ou républicain.

3.3.2 Présentation des métriques utilisées

Nous avons implémenté les métriques suivantes: Intra-variance clustering, Silhouette, Rand Index, Adjusted Rand Index et Accuracy. Ces métriques permettent de comparer 2 partitions d'un même ensemble. L'une des partitions étant correcte et l'autre étant prédite.

Tout d'abord nous présenterons comment nous avons implémenter **Rand Index**.

L'objectif de Rand Index est de voir le taux de paire d'éléments qui sont classés de la même façon aussi bien dans le label prédit que dans le label réel.

La formule de Rand Index est la suivante :

$$\frac{PV + PF}{Touteslespairesd'objetspossibles}$$

Ici :

- PV, correspond au nombre de paires d'objets qui sont dans le même cluster d'après leurs labels réels et leurs labels prédits
- PF, correspond au nombre de paires d'objets qui ne sont pas dans le même cluster d'après leurs labels réels et leurs labels prédits

Autrement synthétisés de cette façon :

	groupés dans π_2	**séparés π_2**
groupés dans π_1	a	b
séparés dans π_1	c	d

Figure 1: Table de consistance

Ici, π_1 représente une partition (ex: label réel) et π_2 la deuxième (ex: label prédit).
Posons 2 objets d'une paire de l'ensemble de départ x,y. Ayant chacun un label donné par la fonction fp pour le label prédit et ft pour le label réel.

Alors:

- a $(= PV)$ correspond au nombre de paires tel que $fp(x) == fp(y)$ and $ft(x) = ft(y)$
- b correspond au nombre de paires tel que $fp(x) \neq fp(y)$ and $ft(x) == ft(y)$
- c correspond au nombre de paires tel que $fp(x) == fp(y)$ and $ft(x) \neq ft(y)$
- d $(= PF)$ correspond au nombre de paires tel que $fp(x) \neq fp(y)$ and $ft(x) \neq ft(y)$

Nous programmerons le tableau pour trouver le résultat de Rand Index. Pour les programmes à venir labelTrue et labelPredicted sont des listes de nombres.

Algorithm 12 Rand_Index: labelTrue, labelPredicted

```
table ← np.zeros((2,2))
for tous les éléments do
  for tous les éléments situés après l'élément en cours do
    if les deux labelTrue des objets sont égaux then
      if les labelPredicted des deux objets sont égaux then
        table[0,0] ← table[0,0] + 1
      else
        table[0,1] ← table[0,1] + 1
      end if
    else
      if les labelPredicted des deux objets sont égaux then
        table[1,0] ← table[1,0] + 1
      else
        table[1,1] ← table[1,1] + 1
      end if
    end if
  end for
end for
return (table[0,0] + table[1,1])/table.sum()
```

Nous avons fait les boucles de cette façon pour éviter de comptabiliser deux fois la même paire.
Nous avons pu retourner la valeur de Rand Index plus facilement grâce au tableau.

Nous avons ensuite codé **Adjusted Rand Index**.

Pour cela, nous utilisons la table de contingence :

$X \diagdown Y$	Y_1	Y_2	$\cdots$	Y_s	sums
X_1	n_{11}	n_{12}	$\cdots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\cdots$	n_{rs}	a_r
sums	b_1	b_2	$\cdots$	b_s	

Figure 2: Table de contingence

Dans la table de contingence, nous avons X et Y, les deux partitions à comparer. Les valeurs de ces partitions sont respectivement en indexes de ligne et de colonne.
Les valeurs mises dans le tableau correspondent à 0, si les éléments en index de la ligne et de la colonne ne possède pas un label identique ou 1 dans le cas contraire.

Ici :

- a_i, correspond au nombre de fois où X_i a une valeur commune aux éléments de Y
- b_i, correspond au nombre de fois où Y_i a une valeur commune aux éléments de X

En prenant compte des éléments du tableau, Adjusted Rand Index suit la formule suivante:

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}}$$

Nous coderons donc Adjusted Rand Index de cette façon :

Algorithm 13 Adjusted_Rand_Index: labelTrue, labelPredicted

$clusterPredicted \leftarrow list(set(labelPredicted))$
$clusterTrue \leftarrow list(set(labelTrue))$
$X \leftarrow np.zeros((len(clusterTrue), len(clusterPredicted)))$
for tous les labels pouvant être pris **do**
  $t \leftarrow clusterTrue.index(labelTrue[i])$
  $p \leftarrow clusterPredicted.index(labelPredicted[i])$
  $X[t,p] \leftarrow X[t,p] + 1$
end for
On transforme tous les éléments de X en entier
$A \leftarrow X.sum(axis = 1)$
$B \leftarrow X.sum(axis = 0)$
$sommeB \leftarrow sum([comb(n,2)forninB])$
$E \leftarrow (sum([comb(n,2) * sommeBforninA])/comb(len(labelTrue), 2))$
$RI \leftarrow sum([comb(n,2)forligneinXforninligne])$
$max\_RI \leftarrow 0.5 * (sum([comb(n,2)forninA]) + sommeB)$
return $(RI - E)/(max\_RI - E)$

Ici, A et B correspondent à la colonne des a_i et à la ligne des b_i. Nous avons regroupé les termes de l'équation dans E, RI et max_RI pour éviter de faire des calculs inutiles et pour les rendre plus simples.

Nous avons ensuite codé **Accuracy**.
Accuracy consiste à regarder le taux de labels mal prédits pour les objets de l'ensemble que nous étudions.

La formule représentant Accuracy est la suivante :

$$\frac{Nombred'objetsbienclasses}{Nombred'objetsdel'ensemble}$$

Le problème rencontré lors de l'implémentation de cette variable a été que 2 clusters similaires dans les partitions, ne sont pas forcément nommés de la même façon.

Nous nous sommes donc aidé des indexes pour voir si les indexes étaient bien classés. Pour cela nous avons regroupé les éléments d'un même cluster en utilisant leurs indexes. Nous l'avons fait pour les mentions de labelPredicted et labelTrue.
Puis nous avons regardé quel ensemble de labelPredicted correspondait le mieux à un ensemble de labelTrue.

Ainsi nous coderons ce code :

Algorithm 14 Accuracy: labelTrue, labelPredicted

$nameClusterT \leftarrow list(set(labelTrue))$
$nameClusterP \leftarrow list(set(labelPredicted))$
$clusterTrue \leftarrow [set()foriinrange(len(nameclusterT))]$
$clusterPredicted \leftarrow [set()foriinrange(len(nameclusterP))]$
for tous les noms de labels possible par partition **do**
  On ajoute à l'ensemble de ClusterTrue qui correspond à la valeur du label réel de notre élément l'indice de l'objet
  On fait de même avec ClusterPredicted
end for
$acc \leftarrow 0$
for tous les clusters de ClusterPredicted **do**
  $max\_similitude \leftarrow 0$
  for tous les éléments de ClusterTrue qui n'ont pas encore été comparé **do**
    $index \leftarrow 0$
    $similitude \leftarrow len(i.intersection(clusterTrue[j]))$
    if $max\_similitude < similitude$ **then**
      $max\_similitude \leftarrow similitude$
      $index \leftarrow$ index de l'ensemble qui a une meilleur similitude avec l'ensemble de ClusterPredicted
    end if
    $acc \leftarrow acc + max\_similitude$
    On supprime de ClusterTrue l'élément qui a été comparé avec l'ensemble de ClusterPredicted
  end for
end for
return $acc/len(labelTrue)$

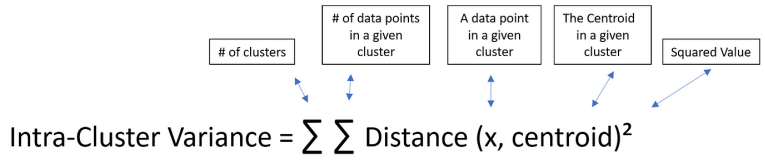
Ici max_similitude correspond aux similitudes entre deux ensembles, l'un étant l'ensemble des indices d'un cluster créé grâce aux labels réels et l'autre grâce aux labels prédits.

Notons ici, que l'algorithme est efficace si les partitions sont bien réalisées et qu'il y ait de nombreux échantillons. Si ce n'est pas le cas, l'accuracy sera proche du résultat voulu mais pas exact.

La métrique codé est la **variance intra cluster**, autrement appelé Intra-Cluster Variance, que nous nommerons ici **ICV**.

Le but de l'ICV est de regarder à quel point les clusters sont compressés sur eux.

La formule correspondant à ce phénomène est la suivante :



Nous sommons les distances de chaque point du cluster à leurs centres, nous faisons ça pour tous les clusters et nous sommons les résultats précédemment obtenus.
Ici, distance dépend du type de données que nous étudions, nous utilisons la fonction dissimilarity si nous avons des données catégoriques et la norme euclidienne si nous avons des données numériques.

Le code de la métrique est le suivant :
ICV prend en paramètre une liste avec les centres des clusters et un dictionnaire représentant les clusters. Ce dictionnaire est composé de clés qui représente le label pris, et d'une liste comportant les objets qui correspondent au label.

Algorithm 15 ICV: centers,cluster

$res \leftarrow 0$
for tous les centres **do**
  $distances \leftarrow np.array(cluster[i] - centroid[i])$
  $res \leftarrow res + np.sum(np.linalg.norm(distance, axis = 1)^2)$
end for
return res

Nous avons ensuite codé **Silhouette**.

Silhouette est une métrique permettant de voir la qualité des clusters. Elle nous permet de voir si les clusters sont bien éloignés les uns des autres tout en voyant la variance intra cluster.

Pour cela, nous devons étudier **pour chaque point, sa silhouette** à l'aide de cette formule :

$$s_{sil}(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Ici :
- i représente le point que nous étudions actuellement dans Data.
- $a(i)$ représente la distance intra-cluster
- $b(i)$ la distance entre le point étudié et tous les points du cluster le plus proche à ce dernier

Ainsi **a(i) est calculé de la façon suivante** :

$$a(i) = \frac{1}{|I_k|-1} \sum_{j \in I_k, j \neq i} d(x^i, x^j)$$

Ici, Ik, correspond aux points du cluster auquel appartient i.
Nous voyons qu'il faut sommer toutes les distances du point aux points du cluster auquel il appartient.

Et **b(i) de celle-ci** :

$$b(i) = \min_{k' \neq k} \frac{1}{|I_{k'}|} \sum_{i' \in I_{k'}} d(x^i, x^{i'})$$

Ici, Ik représente le cluster qui n'est pas celui du point, auquel nous comparons les distances du point entre les siens. Nous prendrons, la somme des distances la plus petite.

Pour avoir **un point de vue plus globale**, du rapport entre la distance au plus proche cluster et celle de l'intra-variance cluster. Nous appliquons cette formule pour tous les points de chaque cluster:

$$S_{sil} = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|I_k|} \sum_{i \in I_k} s_{sil}(i)$$

K ici, représente le nombre de cluster que nous étudions. et Ik, chaque cluster où Silhouette est calculé pour chacun de ses points.

En suivant ces formules, nous obtenons l'algorithme suivant :

Algorithm 16 Silhouette: Cluster

```
silClust ← 0
for tous les clusters do
    sc ← 0
    for tous les éléments du cluster do
        ai ← somme de tous la distance entre tous les éléments du cluster et
        le point étudié
        bi ← []
        for tous les cluster qui ne sont pas le cluster étudié do
            On ajoute à bi la distance entre tous les points du cluster non étudié
            au point étudié
        end for
        sc ← sc + (min(bi) − ai)/max(ai, min(bi))
    end for
    silClust ← silClust + sc/len(cluster_etudie)
end for
Sil ← silClust/len(Cluster)
return Sil
```

En fonction du type de donnée, numpy array ou DataFrame, nous obtenons des tableaux avec divers structures. Mais Cluster reste un dictionnaire avec pour clé, le numéro du cluster et comme valeur associé, ses éléments. Pour le numpy array, les éléments seront aussi des numpys array, de même pour les DataFrames.

3.3.3 Interprétation des résultats

1. Lors de l'exécution du programme, nous voulons **comparer les valeurs obtenus des métriques précédemment implémentées**.

La particularité des datasets que nous avons est que les données en sortie sont déjà donnés pour tous les datasets. Nous avons donc déjà des valeurs à comparer avec les valeurs obtenus par les algorithmes implémentés.

Nous chercherons donc à calculer la partition de notre fonction K-Means, que nous comparerons à la cible de Iris Dataset. Nous rentrons ensuite ces valeurs dans les fonction **rand index** et **adjusted rand index** de Scikit-learn et les nôtres pour obtenir les résultats et les comparer.
Nous obtenons les mêmes pour les fonctions correspondantes. Ce qui nous a prouvé que les algorithmes étaient bien implémenté.

Nous avons implémenté **Accuracy** mais il n'y avait aucune fonction équivalente dans les librairies python. Nous n'avons donc pas pu comparer cette métrique.

Nous avons ensuite comparer **Intra Cluster Variance** en utilisant les clusters et centres obtenues avec la fonction K-Means implémenté par Scikit-Learn et son attribut inertia.
Et nous obtenons les mêmes résultats.

Pour vérifier **Silhouette** nous utilisons le cluster obtenu grâce à K-Means car ce dernier est plus facile à manipuler avec la fonction Silhouette_score de la librairie. Et nous avons comparer la valeur renvoyé par Silhouette et Silhouette_score. Le résultat est semblable au centième près.
La fonction est donc validée.

Nous pouvons donc étudié les données obtenues par nos fonctions avec les métriques implémentés.

Les **particularités de la fonction K-Means testé** sont les suivantes:

K-Means n'est utilisable qu'avec un dataset, Iris Dataset 3.3.1.
Cela est dû au fait qu'Iris dataset est le seul a avoir des données uniquement numériques.
Nous nous sommes donc basé sur ce dataset pour obtenir nos résultats.

Nous avons voulu obtenir des données en lançant plusieurs fois l'algorithme K-Means avec la version du choix des centres initiaux aléatoire.
Cela nous a posé problème car dans certains cas de seed le cluster était vide.

Nous avons donc utilisé la version Mean_Centers 3.2.1 pour obtenir le résultat.
Nous n'avons pu lancer le test qu'une seule fois car Mean_Centers trouve toujours les mêmes centres initiaux pour un dataset donné.

La **particularité de K-Modes** est que nous ne la testerons pas sur Iris Datasets car ces données ne sont à aucun moment catégorique.

Nous nous intéresserons plus à Bank, Votes, Credit Approval car ces derniers sont composés de valeurs catégoriques.
Cependant Bank et Credit Approval ont des valeurs numériques que K-Modes discrétisera. La différence entre ces deux datasets est que Bank ne comporte qu'un attribut numérique alors que Credit Approval est composé à moitié d'attributs numériques.
Nous verrons donc dans ces cas quel algorithme entre K-Prototypes et K-Modes est le plus efficace.

La **particularité de K-Prototype** est que lors de son utilisation, nous pouvons utiliser tous les types de données. Le seul problème donné par cet algorithme est de savoir quel gamma poser pour avoir des résultats plus optimaux.

Voici l**e tableau synthétisé des données** que nous utiliserons :

Datasets	K	N	D	Description
Iris	3	50	4	Numérique
Vote	2	435	16	Catégorique
Bank	2	45211	16	Mixte (un attribut numérique uniquement)
Credit Approval	2	690	15	Mixte (équilibré)

Le tableau possède les champs ***K***, qui représente le*nombre de clusters recherchés* à travers nos algorithmes. ***N*** représente le *nombre d'échantillon*s, ***D*** le *nombre d'attributs* et ***Description*** si il y a des *données catégorielles et/ou numériques*.

K-Mode a la possibilité de discrétiser les valeurs numériques contrairement à K-Prototype qui sait aussi bien gérer les valeurs numériques que catégorielle.

Cependant, lorsque nous lançons l'algorithme K-Prototype avec un dataset uniquement catégorique ou numérique, nous obtenons une erreur.
Malgré cette erreur, K-Prototype devrait fonctionner de la même façon que K-Means pour les données numériques ou K-Modes pour les données catégoriques et nous donner les mêmes résultats.

Par conséquent voici la tableau représentant l'association des données aux algorithmes :

Algorithmes	Iris	Bank	Vote	Credit Approval
K-Means	V	X	X	X
K-Modes	X	V	V	V
K-Prototypes	X	V	X	V

Nous chercherons donc à répondre à cette problématique:

1. Quel est le meilleur algorithme à utiliser pour ce dataset?

Nous aurions pu *calculer pour* ***quel*** *K il fallait implémenter nos algorithmes.*
Pour cela, il aurait fallu obtenir les valeurs des silhouettes pour différents K et voir laquelle de ces valeurs est la plus élevé aussi bien pour K-Prototype que K-Modes. Cependant le K nous étant donnée. Il est inutile de mener cette opération.

Pour réaliser cette expérience, il faut savoir que pour nous pouvons utiliser K-Modes et K-Prototypes uniquement. Cela s'explique par le fait que Credit Approval comporte des données catégorielles et numériques. Or K-Means ne peut traiter que les données numériques et même si K-Modes ne peut traiter que les données catégorielles, elle peut discrétiser les données numériques pour en faire des données catégorielles.

Nous allons donc implémenter pour un K étant égal à 2 car nous savons qu'il y a deux cluster pour K-Modes et K-Prototypes, les valeurs de Silhouette obtenue et plus cette valeur sera élevé et plus elle sera l'indicatrice d'un meilleur cluster.

Nous obtenons le tableau suivant :

Dataset	K-Modes	K-Prototype
Credit Approval	0.08	0.22

Ici, nous avons utilisé pour K-Prototype, γ valant 1.5, nous expliquerons pourquoi nous avons choisi ce résultat plus tard.
Silhouette est une valeur qui est comprise en -1 et 1. Plus sa valeur est élevée et plus le cluster a un bon équilibre entre la variance intra cluster et la variance entre clusters.
Ici, nous voyons donc que les **clusters de Credit Approval sont mieux choisis avec K-Prototype.**
Cela s'explique par le fait qu'en discrétisant les valeurs numériques, nous perdons des informations sur les données.

Nous répondrons maintenant à la question suivante :

2. Comment choisir le bon gamma lorsque nous utilisons K-Prototype?

Pour cela, il suffit de prendre des paramètres identiques à part pour γ. Nous choisirons donc d'étudier les données avec Credit Approval et Bank associés à K valant 2.

Nous obtiendrons ces résultats :

Datasets	$\gamma = 0.5$	$\gamma = 1$	$\gamma = 1.5$
Credit Approval	0.222	0.220	0.226
Bank	0.19	0.17	0.11

Nous observons donc ici, que γ **valant 1.5 possède une meilleure silhouette**, raison pour laquelle nous avons utilisé cette valeur.
Nous choisissons de prendre γ valant 0.5 pour Bank pour la même raison.

γ détermine l'importance qu'ont les valeurs catégorielles aux valeurs numériques. Nous remarquons que Bank a seulement une valeur numérique par rapport aux catégorielles. Le coefficient étant inférieur à 1 nous montre que les valeurs numériques ont plus de poids sur les catégorielles. Cela permet de donner autant d'importance aux variables catégoriques que numériques.

Nous nous sommes ensuite intéressé **aux métriques obtenus en utilisant nos algorithmes**.

Pour obtenir ces métriques, nous avons lancé plusieurs fois les algorithmes de clustering avec les mêmes paramètres mais des seed différentes pour obtenir les résultats. Nous avons ensuite fait les moyennes et calculé l'écart-type des valeurs obtenus.

Nous avons donc obtenu ce tableau représentant les résultats en rapport avec Accuracy :

Accuracy	Iris	Vote	Bank	Credit Approval
K-Means	0.8 ± 0.15	X	X	X
K-Modes	X	0.87	0.74	0.49
K-Prototype	X	X	0.53 ± 0.12	0.70 ± 0.17

Ce tableau représente Rand Index :

Rand Index	Iris	Vote	Bank	Credit Approval
K-Means	0.86 ± 0.003	X	X	X
K-Modes	X	0.77	0.62	0.50
K-Prototype	X	X	0.53 ± 0.02	0.63 ± 0.1

Et ce tableau Adjusted Rand Index :

ARI	Iris	Vote	Bank	Credit Approval
K-Means	0.72 ± 0.006	X	X	X
K-Modes	X	0.54	0	0
K-Prototype	X	X	0.01	0.27 ± 0.19

Nous pouvons voir grâce aux tableaux que K-Modes permet de donner de meilleurs résultats en tous points que K-Prototype lorsqu'il s'agit de regrouper les clusters. Cela s'explique par la présence d'un seul attribut numérique dans Bank dataset.

Ce qui est l'inverse pour Credit Approval qui possède presque autant d'attribut numériques que catégoriques.

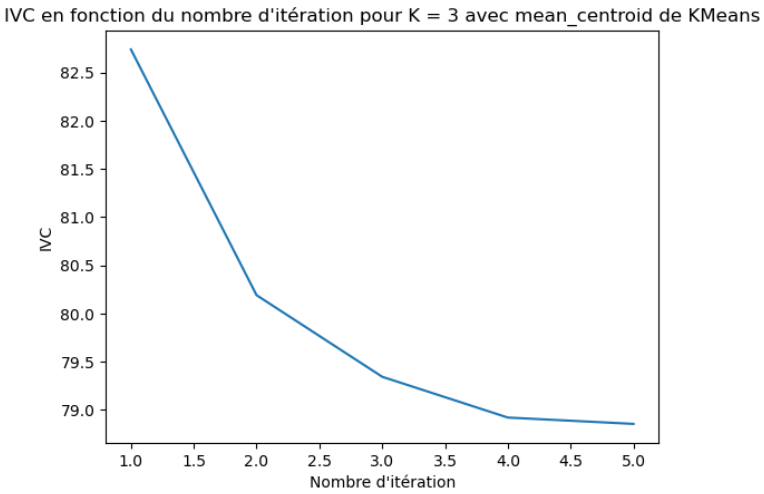
Nous pouvons voir que le Rand Index de chaque expérience est généralement plus petit que l'Accuracy, cela est dû au fait que le rand index, compare toutes

les paires de chaques partitions entre elles.

Nous observons aussi que les valeurs d'ARI sont toujours plus petites que celle de Rand Index, cela s'explique par le fait le fait que Adjusted Rand Index, s'intéresse à la probabilité des chances de former des paires. Par conséquent plus le dataset est grand et plus ARI sera petit, comme nous le voyons sur la colonne de Bank.

Si nous comparons les résultats, il est plus difficile d'obtenir de bon résultat avec K-Prototype, cela pourrait s'expliquer par le choix du coefficient γ. Tandis que K-Means et K-Modes donnent des résultats corrects.

La dernière expérience est de regarder la distance intra-clustering converger à travers ce graphique :



Ici, nous voyons que l'objectif de regrouper des points en fonction de leurs proximités aux autres points dans les clusters est réussi. Plus le nombre d' itérations est élevé et plus l'intra-variance clustering est basse.
Nous constatons aussi, des endroits où il y a des cassures sur la courbe, qui d'après la méthode du coude indiquerait le nombre de cluster optimal. 3 en fait parti.

4 Conclusion

K-Means, K-Modes et K-Prototype sont les algorithmes de clustering que nous avons implémenté chacun ayant pour spécificité divers types de dataset. K-Means s'occupant des données numériques, K-Modes des données catégoriques et K-Prototype des données mixtes.
Nous avons implémenté les métriques : Rand Index, Adjusted Rand Index, Intra-Variance Clustering, Accuracy et Silhouette dans le but d'étudier les algorithmes de clustering.
Nous avons ainsi découvert que K-Prototype est plus efficace que K-Modes lorsque le dataset contenait autant de valeurs numériques que catégoriques. Mais malgré sa spécificité K-Modes pouvait être plus pratique pour des datasets contenant des valeurs numériques, uniquement quand ces derniers sont très peu nombreux.
Pour cela, K-Modes catégorise les valeurs numériques.

References

[1] Amir Ahmad and Lipika Dey. A k-mean clustering algorithm for mixed numeric and categorical data. *Data & Knowledge Engineering*, 63(2):503–527, 2007.

[2] Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, 2(3):283–304, 1998.