COURS 5

APPRENTISSAGE NON SUPERVISÉ

1. RÉDUCTION DE DIMENSION
   - 1.1. ANALYSE EN COMPOSANTES PRINCIPALES
   - 1.2. t-SNE

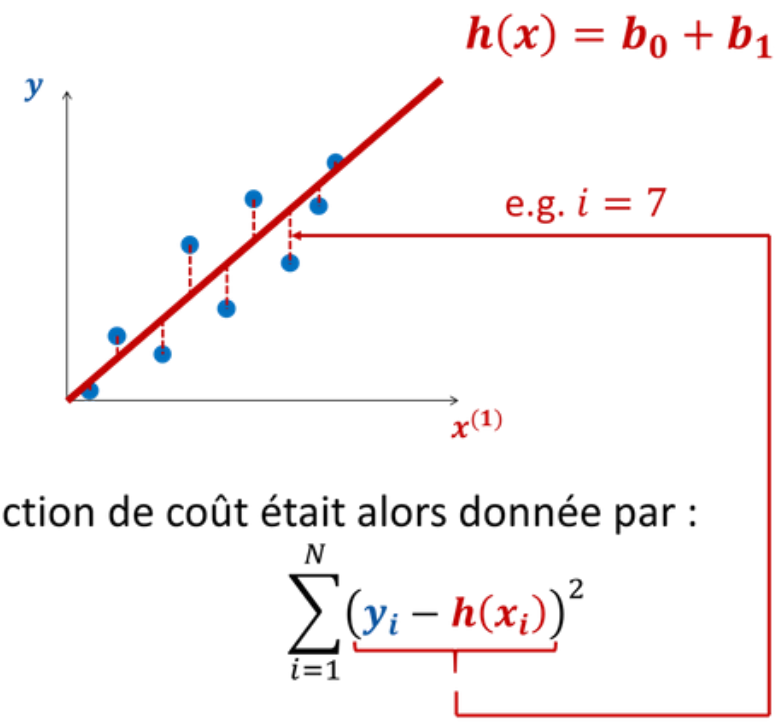
2. REGROUPEMENT
   - 2.1. K-MOYENNES
   - 2.2. DÉCALAGE DE MOYENNE
   - 2.3. GROUPEMENT SPATIAL D'APPLICATIONS BASÉ SUR LA DENSITÉ AVEC BRUIT
   - 2.4. MODÈLES DE MÉLANGE GAUSSIEN
   - 2.5. MODÈLES HIÉRARCHIQUES
   - 2.6. VALIDATION
   - 2.7. EXEMPLES JOUETS
   - 2.8. EXEMPLES SIGNATURE (SIMULÉE)
   - 2.9. MODÈLES D'ENSEMBLES

1. RÉDUCTION DE DIMENSION

1.1. ANALYSE EN COMPOSANTES PRINCIPALES

1.1. ANALYSE EN COMPOSANTES PRINCIPALE

Rappelons le modèle de la régression linéaire.



La fonction de coût était alors donnée par :

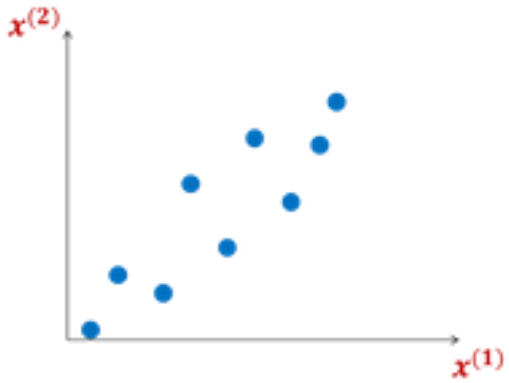
$$\sum_{i=1}^{N} \left(y_i - h(x_i)\right)^2$$

L'erreur était alors calculée selon l'axe y.

1.1. ANALYSE EN COMPOSANTES PRINCIPALE

En apprentissage non supervisé, on n'a pas de variable « cible » (i.e. y).

- On a seulement des caractéristiques (i.e. variables indépendantes).



De plus, en réduction de dimension spécifiquement, l'objectif est de **conserver un maximum d'information** quant à la structure statistique des données…

- …tout en **minimisant le nombre de dimensions** sur lesquelles ces données sont distribuées.

Encore plus spécifiquement, dans le cas de l'analyse en composantes principales, on cherche à trouver, en ordre décroissant, de nouvelles dimensions qui permettraient d'expliquer un maximum de variabilité.

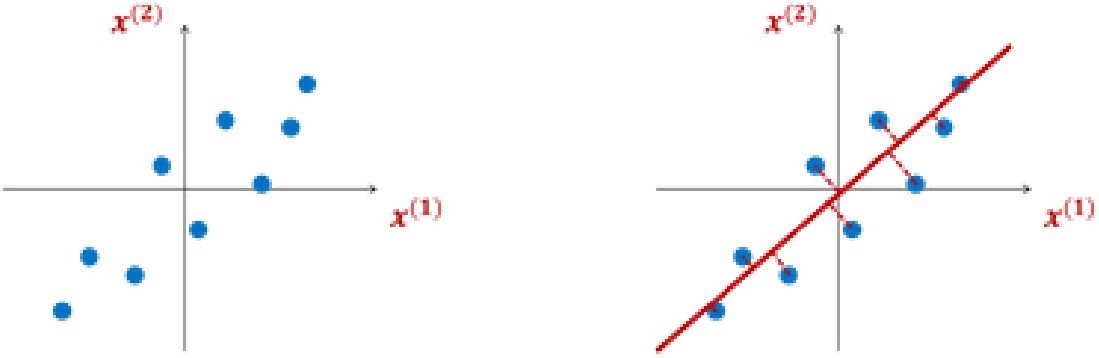
- En d'autres termes, on cherche à minimiser la distance entre chacun de ces points et ces nouvelles dimensions.

1.1. ANALYSE EN COMPOSANTES PRINCIPALE

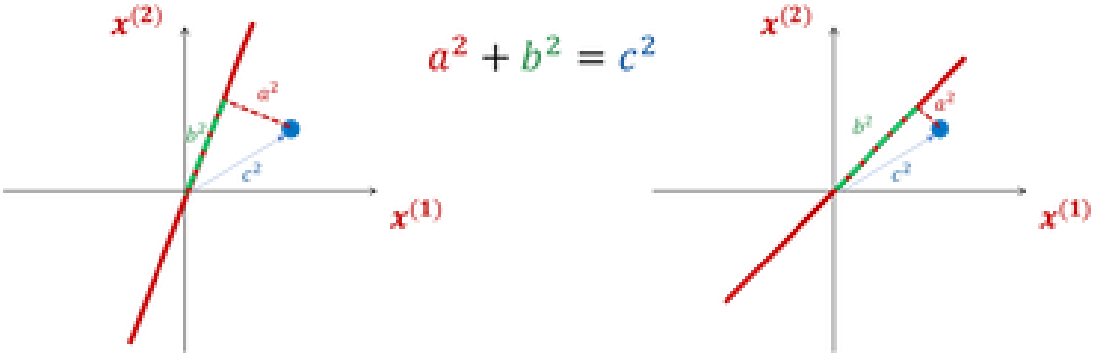
Pour ce faire, on procède aux étapes suivantes :

1. On met chaque dimension sur une échelle semblable.
   - On standardise les données.
   - Les données sont alors centrées en 0.
   - Les valeurs des données sur chaque axe sont alors exprimées en termes d'écarts types.

2. On trouve le nouvel axe qui permettrait de minimiser la distance entre chacun des points et cet axe.



   - Notons que ceci est équivalent à chercher l'axe qui maximise l'écart entre la projection des données sur le nouvel axe et l'origine.

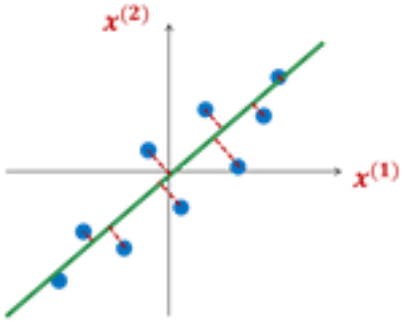


3. On répète l'étape 2 pour chaque nouvel axe.
   - Tout nouvel axe subséquent doit être perpendiculaire par rapport aux axes déjà construits.

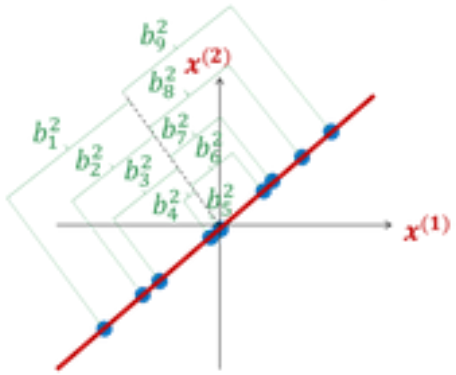
1.1. ANALYSE EN COMPOSANTES PRINCIPALE

Considérons le « 1er nouvel axe » suivant :

On trouve le meilleur nouvel axe

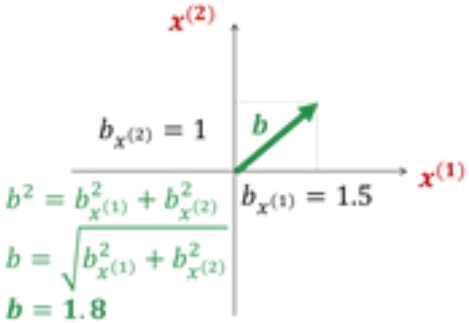


Le « meilleur axe » est celui sur lequel les projections des exemples a la plus grande somme des carrés à l'origine.



Cette somme des carrés est alors appelée $"Valeur\ propre_1" = \sum_{i=1}^{N} b_i^2$

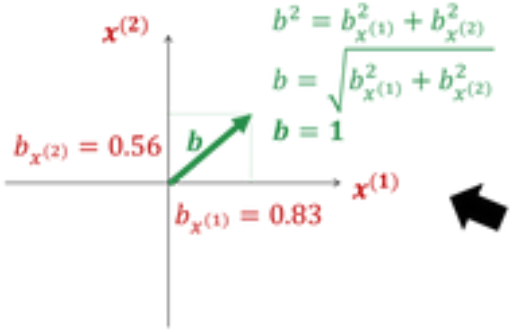
On calcule la contribution de chacun des axes originaux au nouvel axe



Contribution de $x^{(1)}$: $\frac{1.5}{1.8} = 0.56$

Contribution de $x^{(2)}$: $\frac{1}{1.8} = 0.83$

Le vecteur résultant de ces contributions est nommé « vecteur propre ».



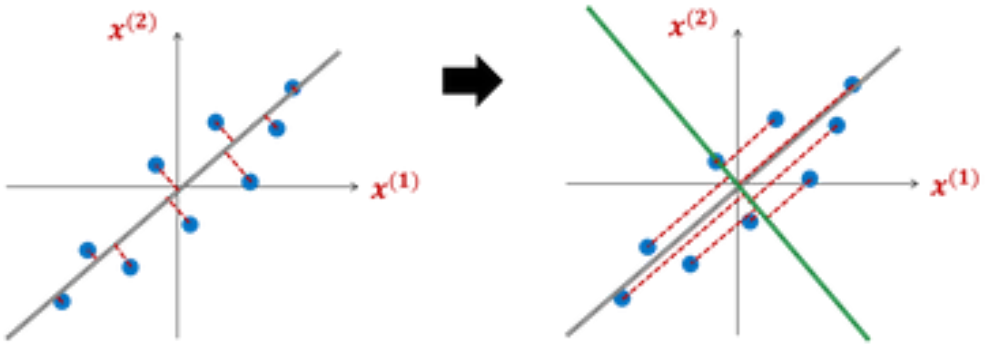
À retenir:

- **L'algorithme trouve le vecteur propre ayant la plus grande valeur propre.**
- **La contribution d'une caractéristique originale dans le nouvel axe correspond à la projection du vecteur propre sur l'axe correspondant à cette caractéristique originale.**

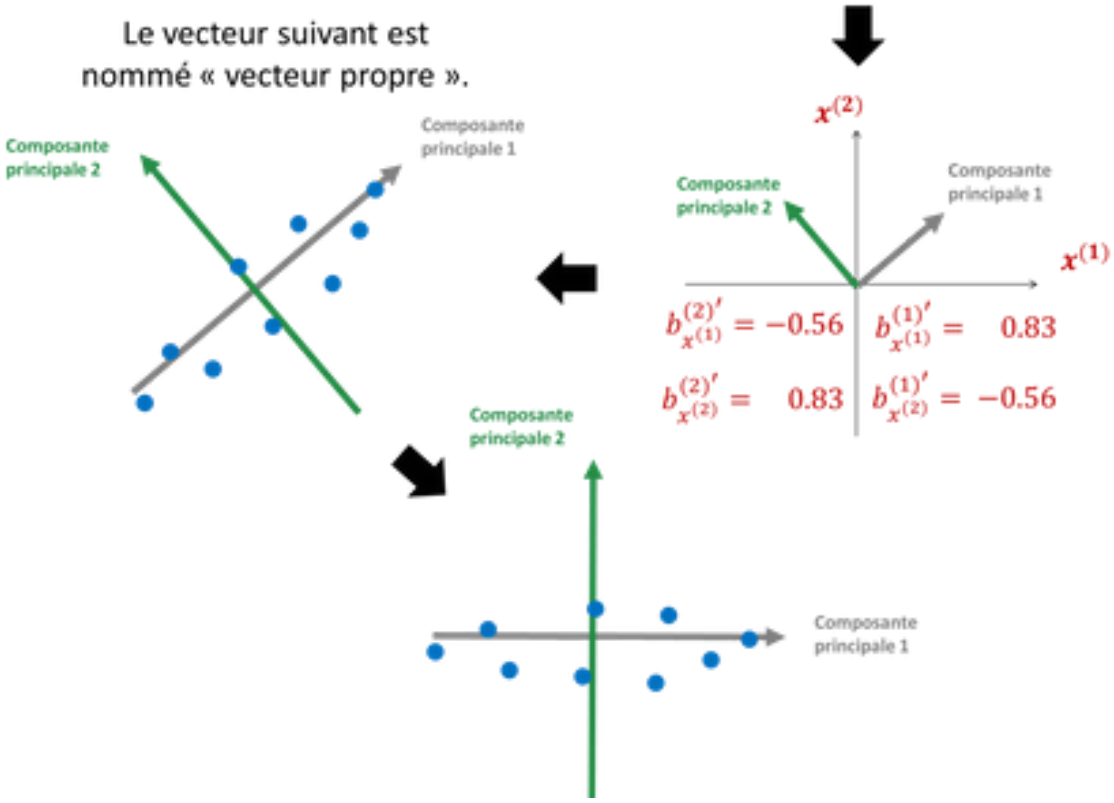
1.1. ANALYSE EN COMPOSANTES PRINCIPALE

On trouve ensuite le meilleur « 2e nouvel axe », qui doit être perpendiculaire à tout autre « nouvel axe construit précédemment.

Ici, comme on a seulement deux dimensions original, on a seulement une possibilité :



Le vecteur suivant est nommé « vecteur propre ».



1.1. ANALYSE EN COMPOSANTES PRINCIPALE

Advenant qu'on ait plutôt débuté avec 5 caractéristiques.

- (i.e. 5 variables indépendantes).
- (i.e. 5 axes originaux).

..alors, on aurait pu trouver jusqu'à 5 composantes principales.

Toutefois, on aurait aussi pu s'arrêter avant (e.g. après deux composantes principales).

Plus la portion de la variance des données qui est expliquée par ces deux composantes principales est grande, plus ces composantes seront utiles…:

- …pour visualiser la distribution des données.

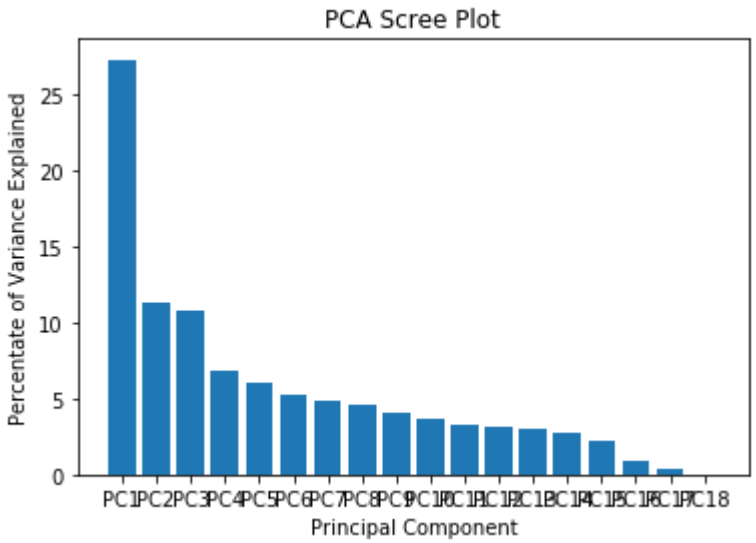
- …pour utiliser comme caractéristiques d'entrée d'un autre algorithme d'apprentissage automatique (e.g. régression, classification ou regroupement).

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()


features_cols = ['UPPSPNU', 'UPPSPPU', 'UPPSPPR', 'UPPSPPE', 'UPPSP
SS', 'PHQ9TT', 'CEVQOTT', 'CEVQOTP', 'CEVQOTS', \
                 'DAST10TT', 'AUDITTT', 'STAIYTT', 'BGHTTA', 'BGHTT
B', 'BGHTTC', 'BGHTTD', 'AGE', 'SES']
X = data.loc[:, features_cols]

scaler = StandardScaler()
X = scaler.fit_transform(X)

n_components = 18
pca = PCA(n_components=n_components)
principalComponents = pca.fit_transform(X)
principalDataframe = pd.DataFrame(data = principalComponents, colum
ns = \
                                  ['PC1', 'PC2', 'PC3', 'PC4', 'PC
5', 'PC6', 'PC7', 'PC8', 'PC9', \
                                   'PC10', 'PC11', 'PC12', 'PC13',
'PC14', 'PC15', 'PC16', 'PC17', 'PC18'])


#print('\nComposantes de la PCA : \n\n', pca.components_)
print('\nVariance expliquée par les composantes : \n\n', pca.explai
```

```
ned_variance_)

percent_variance = np.round(pca.explained_variance_ratio_* 100, dec
imals =2)
pc_columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC
8', 'PC9', 'PC10', \
              'PC11', 'PC12', 'PC13', 'PC14', 'PC15', 'PC16', 'PC1
7', 'PC18']
plt.bar(x= range(1,n_components+1), height=percent_variance, tick_l
abel=pc_columns)
plt.ylabel('Percentate of Variance Explained')
plt.xlabel('Principal Component')
plt.title('PCA Scree Plot')
plt.show()
```

```
Variance expliquée par les composantes :

 [4.92695188e+00 2.05205792e+00 1.94831782e+00 1.244405
26e+00
 1.09002437e+00 9.37242169e-01 8.64148015e-01 8.2004664
2e-01
 7.25602648e-01 6.61884701e-01 5.82023059e-01 5.5776779
6e-01
 5.35518371e-01 4.88496079e-01 3.93459185e-01 1.5601477
8e-01
 6.22299546e-02 2.32687393e-03]
```



In [2]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de sélection de données
from sklearn.feature_selection import VarianceThreshold

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA

# Importer une fonction qui nous permette de construire aléatoireme
nt les ensembles "Entraînement" et "Test"
from sklearn.model_selection import train_test_split

# Importer le modèle de régression logistique de sklearn
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

# Importer la fonction de validation croisée
from sklearn.model_selection import StratifiedKFold, GridSearchCV

# Importer la fonction permettant d'afficher le rapport de classifi
cation
from sklearn.metrics import roc_auc_score

np.random.seed(42)

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
```

```
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()

# Noms des colonnes correspondant aux caractéristiques et à la cibl
e
features_cols = ['UPPSPNU', 'UPPSPPU', 'UPPSPPR', 'UPPSPPE', 'UPPSP
SS', 'PHQ9TT', 'CEVQOTT', 'CEVQOTP', 'CEVQOTS', \
                 'DAST10TT', 'AUDITTT', 'STAIYTT', 'BGHTTA', 'BGHTT
B', 'BGHTTC', 'BGHTTD', 'AGE']
target_col = 'WHODASTTB'

# Données importées (X: caractéristiques, y: cible)
X = data.loc[:, features_cols]
y = data['WHODASTTB']

# Séparation en données d'entraînement et en données de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2)

# Standatdisation des entrées
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# On effectue la PCA
n_components = 17
pca = PCA(n_components=n_components)
pca.fit(X_train)

X_train_PCA = pca.transform(X_train)
X_test_PCA = pca.transform(X_test)

# ----------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ----------------------------------------------------------------
------------

# Définir le modèle
model = DecisionTreeClassifier()

# Définir les hyperparamètres
hyperparams = {'max_depth':[1, 2, 4, 8, 12, 16], 'min_samples_spli
t':[2, 4, 6, 8, 10, 12, 14, 16]}

# Définir les plus de la validation croisée
cv_folds = StratifiedKFold(n_splits=5, random_state=42)

# Définir le type de score utilisé pour sélectionner les hyperparam
ètres dans la validation croisée
scoring='roc_auc'
```

```
X_train_final = X_train_PCA
X_test_final = X_test_PCA
#X_train_final = X_train
#X_test_final = X_test

# Réaliser la validation croisée avec grille de recherche pour les
 hyperparamètres.
cv_valid = GridSearchCV(estimator=model, param_grid=hyperparams, cv
=cv_folds, scoring=scoring, iid=False)
cv_valid.fit(X_train_final, y_train)
best_params = cv_valid.best_params_
best_score = cv_valid.best_score_
model = cv_valid.best_estimator_
print('\nMeilleurs hyperparamètres: \n', best_params)
print('\nScore = \n', best_score)

# Entraîner le modèle final avec toutes les données d'entraînement
model.fit(X_train_final, y_train)
```

```
Meilleurs hyperparamètres:
 {'max_depth': 12, 'min_samples_split': 12}

Score =
 0.6279874463689813
```

Out[2]:

```
DecisionTreeClassifier(class_weight=None, criterion='gi
ni', max_depth=12,
                       max_features=None, max_leaf_node
s=None,
                       min_impurity_decrease=0.0, min_i
mpurity_split=None,
                       min_samples_leaf=1, min_samples_
split=12,
                       min_weight_fraction_leaf=0.0, pr
esort=False,
                       random_state=None, splitter='bes
t')
```

In [3]:

```
# On teste l'algorithme final en prédisant de nouvelles données.
y_pred = model.predict(X_test_final)

# On évalue les prédictions de l'algorithme final.
auc = roc_auc_score(y_test, y_pred)

# On affiche le résultat
print('\nTest AUC = ', auc)
```

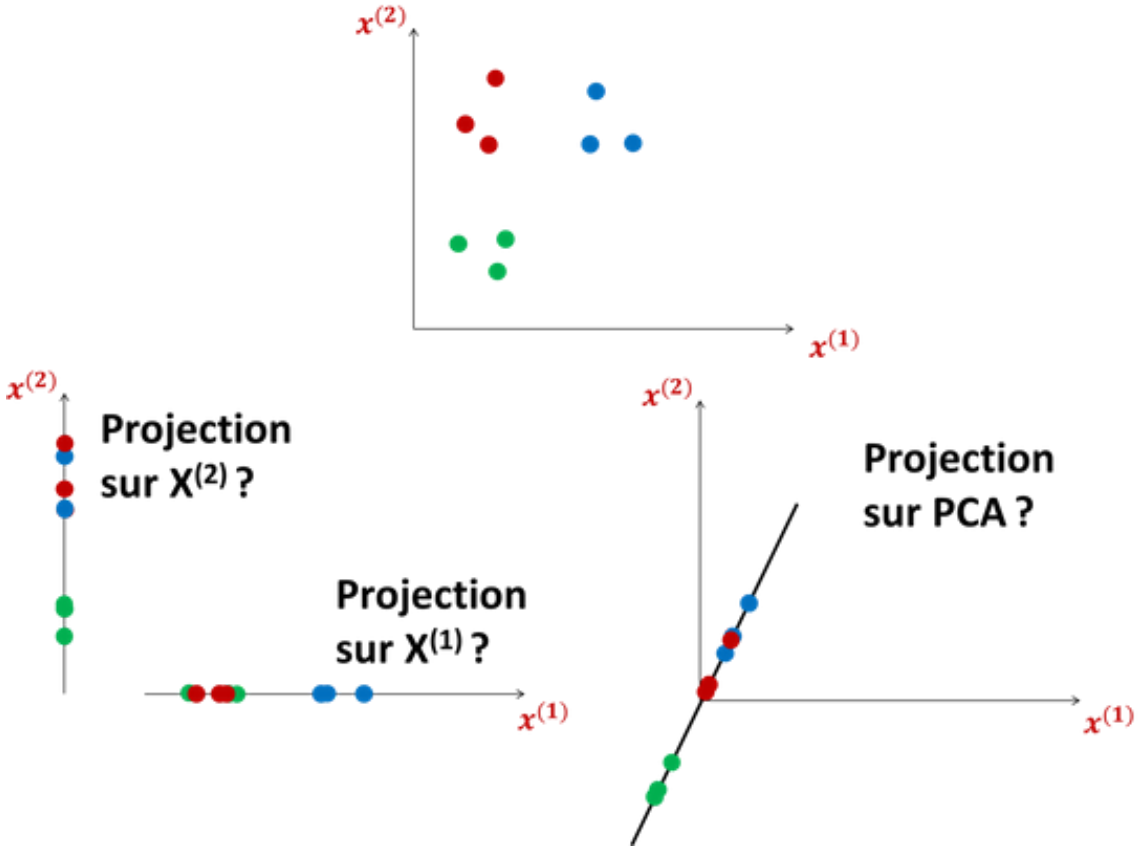
```
Test AUC =  0.6994261119081779
```

1.2. T-SNE

1.2. T-SNE

Comment préserver les regroupements dans un affichage comportant un nombre inférieur de dimensions ?

Plus précisément, ici, comment projeter les regroupements en deux dimensions dans un affichage en une dimension.

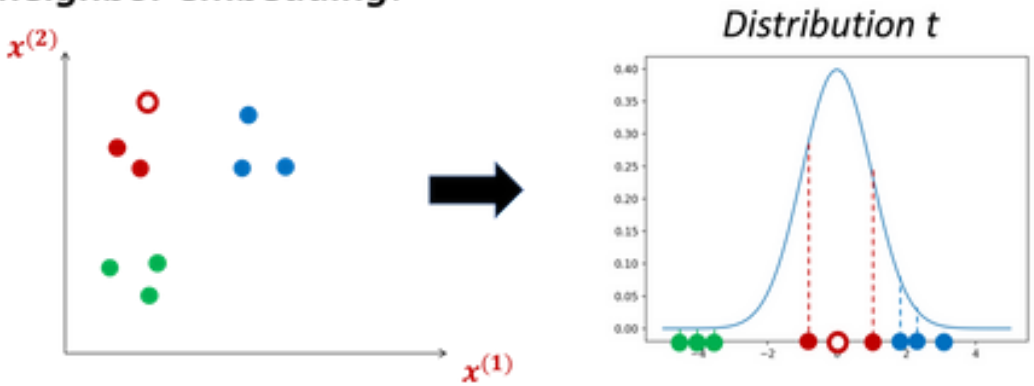


1.2. T-SNE

Une approche plus efficace est d'extraire, pour chaque exemple, la distribution des probabilités qu'a cet exemple d'être près de chacun des autres exemples **dans l'espace en hautes dimensions…**

…puis, de chercher à reconstruire, pour chaque exemple, la distribution des probabilités qu'a cet exemple d'être près de chacun des autres exemples **dans l'espace en basses dimensions.**

Comme la distribution utilisée correspond à la distribution ***t***, la méthode est nommée ***t-distributed stochastic neighbor embedding***.



On répète cette étape pour tous les exemples (on fait la moyenne des deux probabilités de proximité obtenues pour chaque paire d'exemples).

On obtient ainsi un tableau avec une probabilité de proximité pour chaque paire d'exemples.

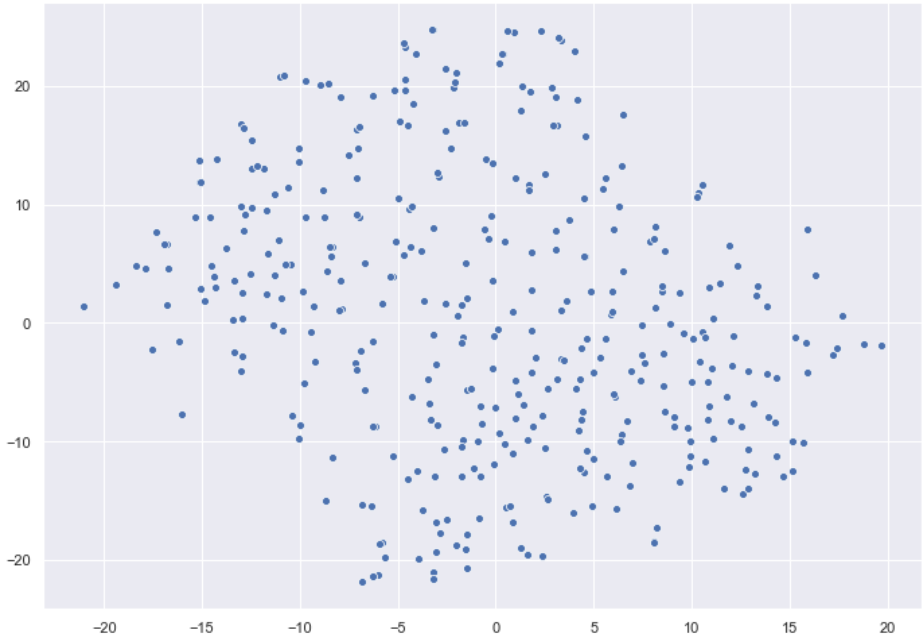
1.2. T-SNE

In [4]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data.dropna()


features_cols = ['UPPSPNU', 'UPPSPPU', 'UPPSPPR', 'UPPSPPE', 'UPPSP
SS', 'PHQ9TT', 'CEVQOTT', 'CEVQOTP', 'CEVQOTS', \
                 'DAST10TT', 'AUDITTT', 'STAIYTT', 'BGHTTA', 'BGHTT
B', 'BGHTTC', 'BGHTTD', 'AGE', 'SES']

X = data.loc[:, features_cols]

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_embedded = TSNE(n_components=2).fit_transform(X)

sns.set(rc={'figure.figsize':(11.7,8.27)})
palette = sns.color_palette("bright", 10)
sns.scatterplot(X_embedded[:,0], X_embedded[:,1])
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe733790>
```



2. REGROUPEMENT

2.1. K-MOYENNES

2.1. K-MOYENNES

Étapes :

1. Sélectionner ***k***, le nombre de groupes.
2. Choisir au hasard ***k*** exemples pour devenir les premiers centroïdes.
3. Assigner chaque point au groupe ayant le centroïde le plus proche.
4. Recalculer les positions des centroïdes.
   - Position moyenne de tous les points ayant été assignés à un même groupe.
5. Répéter les étapes 3 et 4 jusqu'à ce les positions des centroïdes soient les mêmes durant deux itérations consécutives.

Hyperparamètre principal :

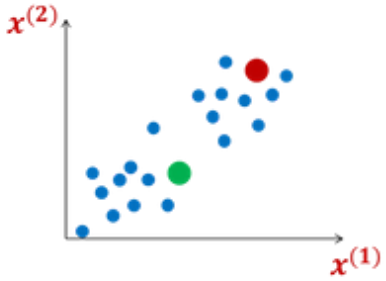
- ***k***, le nombre de groupes.
  - Sklearn *:* ***n_clusters*** (valeur par défaut : 8)

2.1. K-MOYENNES

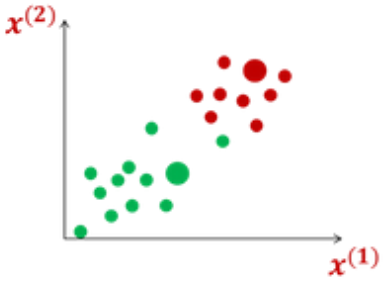
Exemple avec deux groupes :

1. Sélectionner k, le nombre de groupes : ici, $k = 2$.

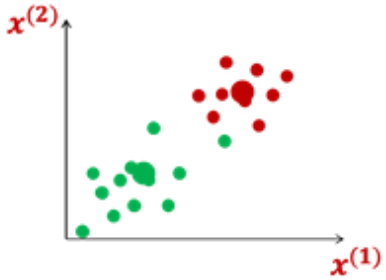
2. Choisir au hasard k exemples pour devenir les premiers centroïdes.



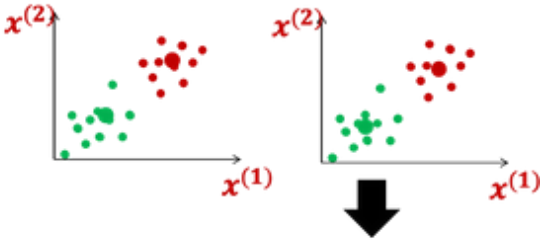
3. Assigner chaque point au groupe ayant le centroïde le plus proche.



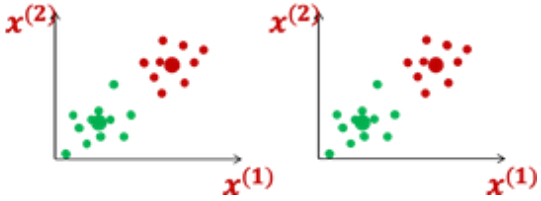
4. Recalculer les positions des centroïdes.



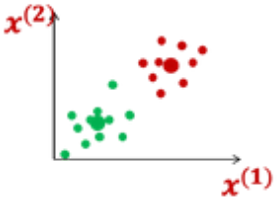
5a. Répéter les étapes 3 et 4 jusqu'à ce que la solution converge.



5b. Répéter les étapes 3 et 4 jusqu'à ce que la solution converge.



Groupements finaux



2.2. DÉCALAGE DE MOYENNE

2.2. DÉCALAGE DE MOYENNE

Étapes :

1. Sélectionner la **bande passante** (*bandwidth*) d'une fonction noyau (généralement, une distribution gaussienne).
2. Sélectionner au hasard un **centroïde** correspondant à la position d'un exemple.
3. Calculer le centroïde de la fonction noyau selon les exemples se trouvant à l'intérieur de la bande passante.
4. Déplacer la fonction noyau pour que son centroïde corresponde au nouveau centroïde calculé.
5. Répéter les étapes 3 et 4 jusqu'à ce que la position du centroïde soit la même durant deux itérations consécutives.
6. Répéter les étapes 2 à 5 à partir de la position de chaque exemple.
   - Lorsqu'un exemple converge vers un point, ce point est nommé « mode ».
   - Ce point correspond à l'une des classes finales de l'algorithme de regroupement.
7. Tous les exemples qui convergent vers un même mode appartiennent au même groupement.

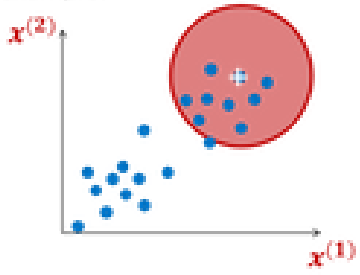
Hyperparamètre principal :

- Le rayon du cercle.
  - Sklearn *:* ***bandwith*** (valeur par défaut estimée par sklearn automatiquement).

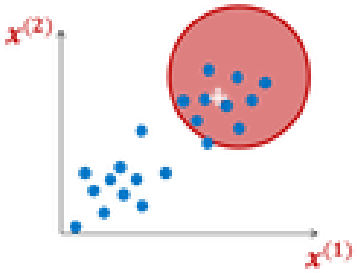
2.2. DÉCALAGE DE MOYENNE

1. Sélectionner la **bande passante** (*bandwidth*) d'une fonction noyau (généralement, une distribution gaussienne). .

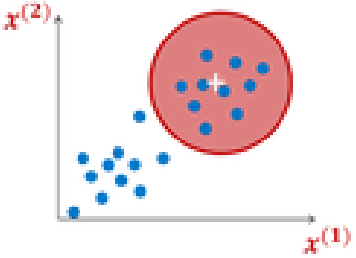
3. Sélectionner au hasard un **centroïde** correspondant à la position d'un exemple.



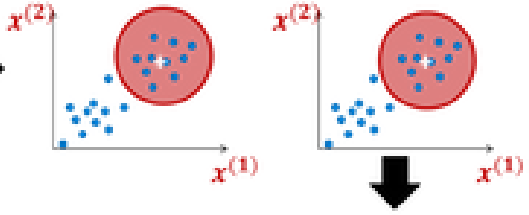
4. alculer le centroïde de la fonction noyau selon les exemples se trouvant à l'intérieur de la bande passante.



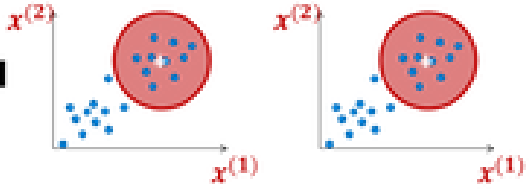
4. Déplacer la fonction noyau pour que son centroïde corresponde au nouveau centroïde calculé.



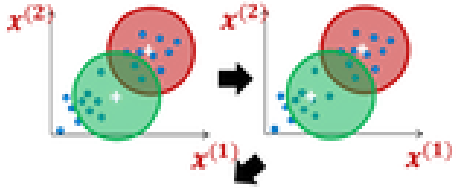
5a. Répéter les étapes 3 et 4 jusqu'à ce que la position du centroïde soit la même durant deux itérations consécutives.



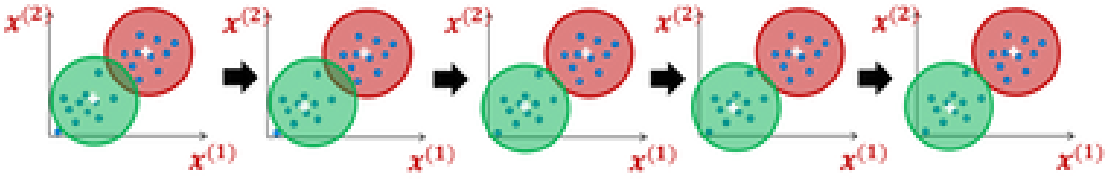
5b. Répéter les étapes 3 et 4 jusqu'à ce que la position du centroïde soit la même durant deux itérations consécutives.



6a. Répéter les étapes 2 à 5 à partir de la position de chaque exemple.



Groupements finaux



- Lorsqu'un exemple converge vers un point, ce point est nommé « mode ».
- Ce point correspond à l'une des classes finales de l'algorithme de regroupement.

7. Tous les exemples qui convergent vers un même mode appartiennent au même groupement.

2.3. DBSCAN

2.3. DBSCAN

« Groupement spatial d'applications basé sur la densité avec bruit ».

Étapes :

1. Choisir une distance ***ε* (*epsilon*)** et une taille de groupe minimum (***min_samples***).
2. Choisir un exemple aléatoire.
3. Trouver tous les exemples se trouvant à l'intérieur d'une distance ***ε*** de l'exemple initial.
   - Ces exemples sont ajoutés temporairement au même groupe.
4. Répéter l'étape 3 pour tous les exemples ajoutés à ce groupe.
5. Si le nombre final d'exemples ajoutés à un groupe est inférieur à **min*_samples***, ces exemples sont finalement étiquetés comme étant du bruit.
   - Sinon, ces exemples forment officiellement un groupe.
6. Les étapes 2 à 5 sont répétées pour un exemple aléatoire n'ayant pas encore été considéré.
7. L'algorithme se termine lorsque chaque exemple appartient à un groupe ou a été étiqueté comme étant du bruit.

Hyperparamètres principaux :

- Distance de recherche ***ε* (*epsilon*)**.
  - Sklearn : ***eps*** (valeur par défaut : 0.5).
- Nombre d'exemples minimum pour former un groupement.
  - Sklearn : ***min_samples*** (valeur par défaut : 5).

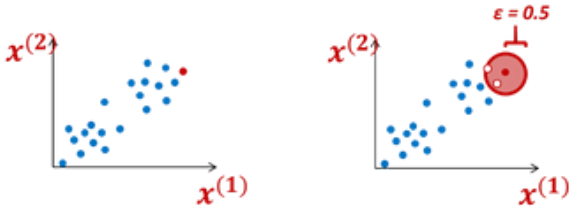
2.3. DBSCAN

Exemple avec deux groupes :

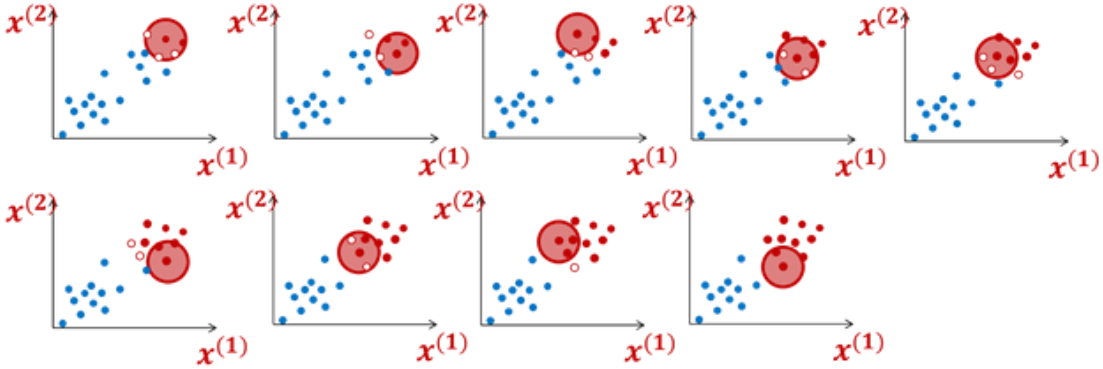
1. Choisir une distance ***ε* (*epsilon*)** et une taille de groupe minimum (***min_samples***).
   - Ex. ***ε = 0.5*** et ***min_samples = 5***.

2. Choisir un exemple aléatoire.

3. Trouver tous les exemples se trouvant à l'intérieur d'une distance ***ε*** de l'exemple initial.



4. Répéter l'étape 3 pour tous les exemples ajoutés à ce groupe.

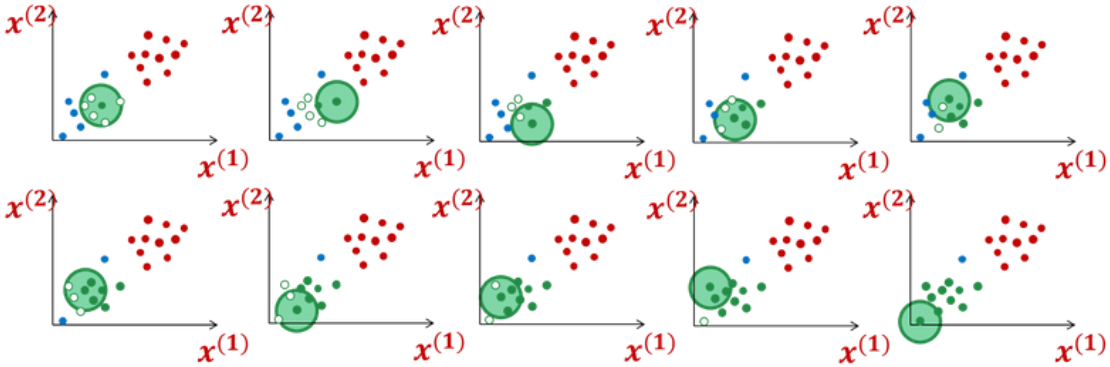


5. Comme le nombre final d'exemples ajoutés à un groupe est supérieur à **min_*samples* = 5**, ces exemples forment bel et bien un premier groupe.

2.3. DBSCAN

Exemple (suite) :

6a. Les étapes 2 à 5 sont répétées pour un exemple aléatoire n'ayant pas encore été considéré.

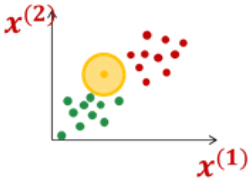


Comme le nombre final d'exemples ajoutés à un groupe est supérieur à **min_*samples* = 5**, ces exemples forment bel et bien un deuxième groupe.

2.3. DBSCAN

Exemple (suite) :

6b. Les étapes 2 à 5 sont répétées pour un exemple aléatoire n'ayant pas encore été considéré.



Comme on n'a qu'un seul exemple dans ce groupe, le nombre d'exemples est inférieur à **min_*samples* = 5** et cet exemple est étiqueté comme étant du bruit.

2.4. MODÈLES DE MÉLANGE GAUSSIEN

2.4. MODÈLES DE MÉLANGE GAUSSIEN

Étapes :

1. Sélectionner ***k***, le nombre de groupes.
2. Initialiser les paramètres du modèle pour chaque groupe :
   - $\hat{\phi}_k = \frac{1}{k} : probabilité\ a\ priori\ d'appartenir\ au\ k^{ième}\ groupe$
   - $\hat{\mu}_k = exemple\ tiré\ au\ hasard.$
   - $\hat{\sigma}_k = \frac{1}{N}\sum_{i=1}^{k}(x_i - \bar{x})^2\ , variance\ des\ exemples$
3. Calculer la probabilité que chaque exemple appartienne à chacun des ***k*** groupes (**Espérance**).
   - Ces calculs considèrent que chaque ensemble de paramètre $(\hat{\mu}_k, \hat{\sigma}_k)$ définit une distribution normale.
   - $\hat{\gamma}_{ik} = \frac{\hat{\phi}_k \mathcal{N}(x_i|\hat{\mu}_k,\hat{\sigma}_k)}{\sum_{j=1}^{k} \mathcal{N}(x_i|\hat{\mu}_k,\hat{\sigma}_k)}$
4. Étant donné ces probabilités, calculer les nouvelles valeurs des paramètres (**Maximisation**).
   - $\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$
     - Moyenne des probabilités des exemples d'appartenir au k^{ème} groupe.
   - $\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
     - Moyenne pondérée des positions des exemples.
   - $\hat{\sigma}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik}(x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
     - Moyenne pondérée des variances des exemples.
5. Répéter les étapes 3 et 4 jusqu'à ce que les valeurs des coefficients convergent vers une solution.
6. Chaque exemplaire appartient alors au groupe auquel il a la plus grande probabilité d'appartenir.

2.4. MODÈLES DE MÉLANGE GAUSSIEN

À retenir:

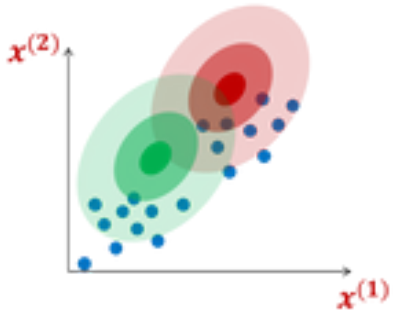
- À chaque itération :
  i. On calcule les probabilités que chaque exemple appartienne à chaque groupe.
     - **Espérance (*Expectation*)**
  ii. On utilise ces probabilités pour ajuster les valeurs des paramètres de la distribution normale correspondant à chaque groupe
     - **Maximisation (*Maximization*)**
- On effectue cette boucle jusqu'à ce que les valeurs des paramètres soient stables.

- Nombre de composantes (i.e. nombre de groupes ***k***).
  - Sklearn : ***n_components*** (valeur par défaut : 1).

2.4. MODÈLES DE MÉLANGE GAUSSIEN

Exemple :

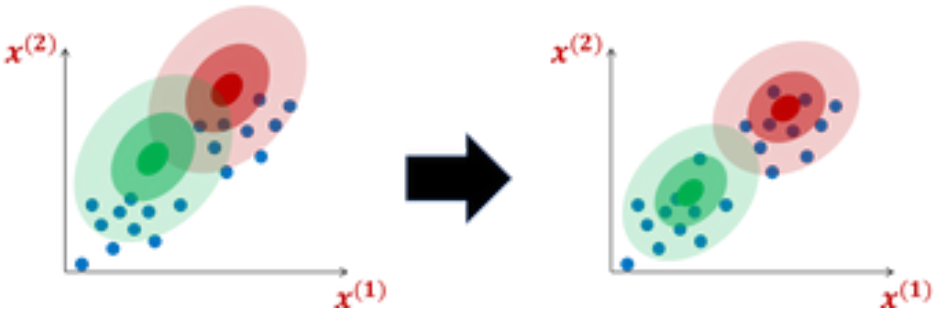
1. Sélectionner ***k***, le nombre de groupes (e.g. ici, ***k = 2***)
2. Initialiser les paramètres du modèle pour chaque groupe :
   - $\hat{\phi}_1 = 0.5, \quad \hat{\phi}_2 = 0.5$
   - $\hat{\mu}_1 = x_{11}, \quad \hat{\mu}_2 = x_4$
   - $\hat{\sigma}_k = calculs \ldots$



3. Calculer la probabilité que chaque exemple appartienne à chacun des ***k*** groupes (**Espérance ; *Expectation***).

$$\hat{\gamma}_{ik} = \frac{\hat{\phi}_k \mathcal{N}(x_i|\hat{\mu}_k, \hat{\sigma}_k)}{\sum_{j=1}^{k} \mathcal{N}(x_i|\hat{\mu}_k, \hat{\sigma}_k)}$$

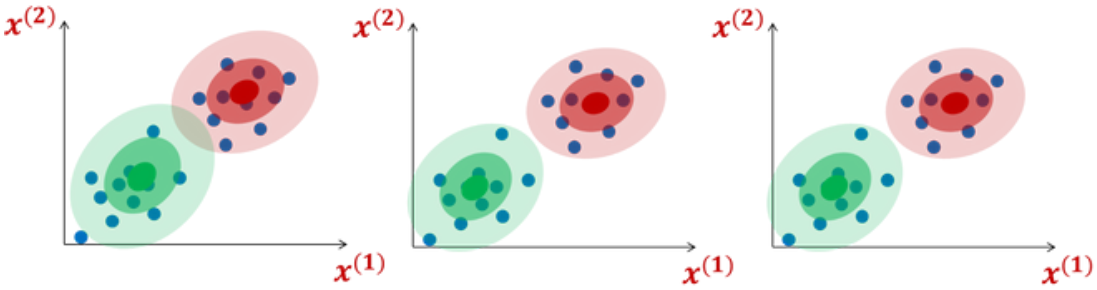
1. Étant donné ces probabilités, calculer les nouvelles valeurs des paramètres (**Maximisation ; *Maximization***).
   - $\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$
     - Moyenne des probabilités des exemples d'appartenir au k^{ème} groupe.
   - $\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
     - Moyenne pondérée des positions des exemples.
   - $\hat{\sigma}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} (x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
     - Moyenne pondérée des variances des exemples.



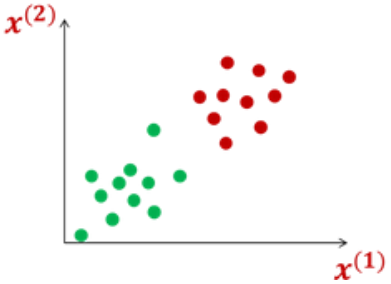
2.4. MODÈLES DE MÉLANGE GAUSSIEN

Exemple (suite) :

5. Répéter les étapes 3 et 4 jusqu'à ce que les valeurs des coefficients convergent vers une solution.



6. Chaque exemplaire appartient alors au groupe auquel il a la plus grande probabilité d'appartenir.



2.5. REGROUPEMENT HIÉRARCHIQUE

2.5. REGROUPEMENT HIÉRARCHIQUE

Deux types :

- Ascendant (*bottom-up*) ; ou « agglomératif » (*agglomerative)*
- Descendant (*top-down*) ; ou « divisif » (*divisive*)

Ascendant

- Étapes :
  1. Considérer chaque exemple comme un **groupe**.
  2. Calculer la distance entre chaque **paire de groupes**.
  3. **Regrouper** les deux groupes les plus proches.
  4. Répéter les étapes 2 et 3 jusqu'à ce qu'il n'y ait qu'**un seul regroupement**.

Hyperparamètres principaux :

- Nombre de groupes à trouver.
  - Sklearn : ***n_clusters*** (valeur par défaut : 2).
- Type de métrique utilisée pour calculer les distances.
  - Sklearn : ***affinity*** (valeur par défaut : 'euclidean').
- Points utilisés pour calculer les distances.
  - Sklearn : ***linkage*** (valeur par défaut : 'ward').
  - Note 1: avec « ward », on doit nécessairement utiliser « *affinity='euclidean'* ».

2.5. REGROUPEMENT HIÉRARCHIQUE

Deux types :

- Ascendant (*bottom-up*) ; ou « agglomératif » (*agglomerative)*
- Descendant (*top-down*) ; ou « divisif » (*divisive*)

Descendant

- Étapes :
  1. Considérer tous les exemples comme faisant partie d'**un seul et même groupe**.
  2. Utiliser un algorithme non-hiérarchique pour diviser le groupe en deux regroupements (e.g. **k-moyennes**).
  3. Répéter l'étape 2 pour chaque nouveau groupement, jusqu'à ce qu'il n'y ait qu'**un exemple par groupe**.

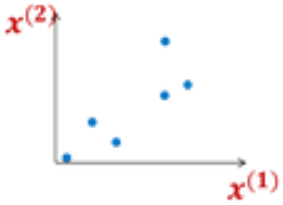
Note:

- **Sklearn N'implémente PAS la version descendante du regroupement hiérarchique.**

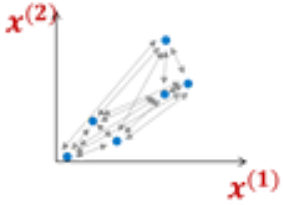
2.5. REGROUPEMENT HIÉRARCHIQUE

Exemple **ascendant** :

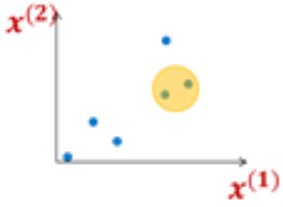
1. Considérer chaque exemple comme un **groupe**.



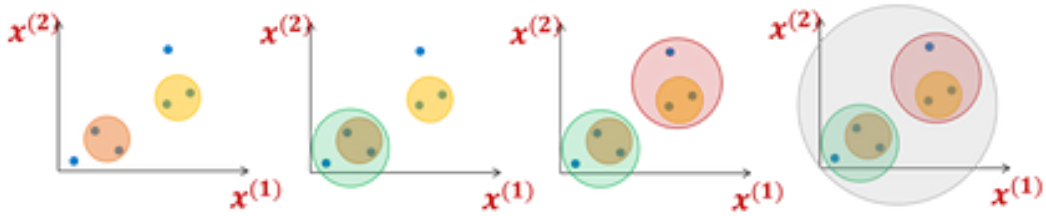
2. Calculer la distance entre chaque **paire de groupes**.



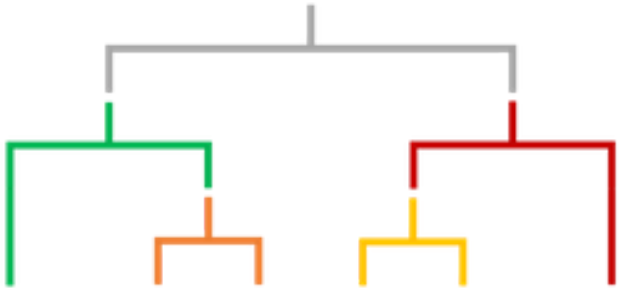
3. **Regrouper** les deux groupes les plus proches



4. Répéter les étapes 2 et 3 pour chaque nouveau groupement, jusqu'à ce qu'il n'y ait qu'**un exemple par groupe**.



❖ Généralement présenté sous forme de dendrogramme :



❖ On pourrait alors décider de « couper » le dendrogramme à la hauteur qui nous convient.

- ➢ i.e. avec le nombre de groupements qui nous convient.

2.6. VALIDATION

2.6. VALIDATION

Validation d'un algorithme de regroupement:

Deux types :

- **Validation interne :**
  - Grande similarité intragroupe.
  - Faible similarité intergroupe.
  - Exemples:
    - ***Coefficient Silhouette (e.g. dans scikit learn)***
    - *Index Calinski-Harabasz*
    - *Index Davies Bouldin*
    - *Matrice de contingence*

- **Validation externe :**
  - Comparaison à une classification réelle.
  - Exemples:
    - ***Rand index ajusté (e.g. dans scikit learn)***
    - *Scores basés sur l'information mutuelle*
    - *Mesure V*
    - *Score Fowlkes-Mallows*

2.6. VALIDATION

Validation interne :

On recherche:

- Grande similarité intragroupe.
- Petite similarité intergroupe.

Exemple : **Silhouette**

$$\frac{1}{N}\sum_{i=1}^{N}\frac{b_i - a_i}{max(a_i, b_i)}$$

où... :

- $\boldsymbol{N}$: nombre d'exemples
- $\boldsymbol{b_i}$: distance moyenne entre le i^{ème} exemple et tous les autres exemples du regroupement **le plus près** (auquel il n'appartient pas).
- $\boldsymbol{a_i}$: distance moyenne entre le i^{ème} exemple et tous les autres exemples du regroupement **auquel il appartient**.

Ainsi… :

- Pour un même ***b***, plus les classes sont compactes, plus ***a*** est petit et donc plus le score Silhouette est grand.
  - Un petit ***a*** implique une grande similarité intragroupe.
- Pour un même ***a***, plus les classes sont éloignées les unes des autres, plus ***b*** est grand et donc plus le score Silhouette est grand.
  - Un grand ***b*** implique une faible similarité intergroupe.

2.6. VALIDATION

Validation externe :

On recherche:

- Les mêmes classes suite à l'algorithme de regroupement que dans la variable cible (***y***).

Exemple : **Rand index** (***RI***) :

$$RI = \frac{a + b}{C_2^N}$$

où…:

- ***N*** : nombre d'exemples
- ***a*** : nombre de paires d'exemples qui sont **dans la même classe**, tant suite à l'algorithme de regroupement que selon la variable cible (***y***).
- ***b*** : nombre de paires d'exemples qui sont **dans des classes différentes**, tant suite à l'algorithme de regroupement que selon la variable cible (***y***).
- C_2^N **:** nombre de paires d'exemples.

Ainsi… :

- Pour un même ***b***, plus il y a de paires d'exemples regroupés ensemble dans les deux cas, plus ***a*** est grand et donc plus le score ***RI*** est grand.
- Pour un même ***a***, plus il y a de paires d'exemples regroupés dans des classes distinctes dans les deux cas, plus ***b*** est grand et donc plus le score ***RI*** est grand.

2.6. VALIDATION

Validation externe :

On recherche:

- Les mêmes classes suite à l'algorithme de regroupement que dans la variable cible (***y***).

Exemple : **Rand index** (***RI***) :

$$RI = \frac{a+b}{C_2^N}$$

On peut observer ***a*** et ***b*** dans un tableau de contingence:

		VALEURS CIBLE (*y*)	
		Co-groupées	Non co-groupées
ALGORITHME DE REGROUPEMENT	Co-groupées	a	c
	Non co-groupées	d	b

$$RI = \frac{a+b}{a+b+c+d}$$

2.6. VALIDATION

Validation externe :

On recherche:

- Les mêmes classes suite à l'algorithme de regroupement que dans la variable cible (***y***).

Exemple : **Rand index** (***RI***) :

$$RI = \frac{a+b}{C_2^N}$$

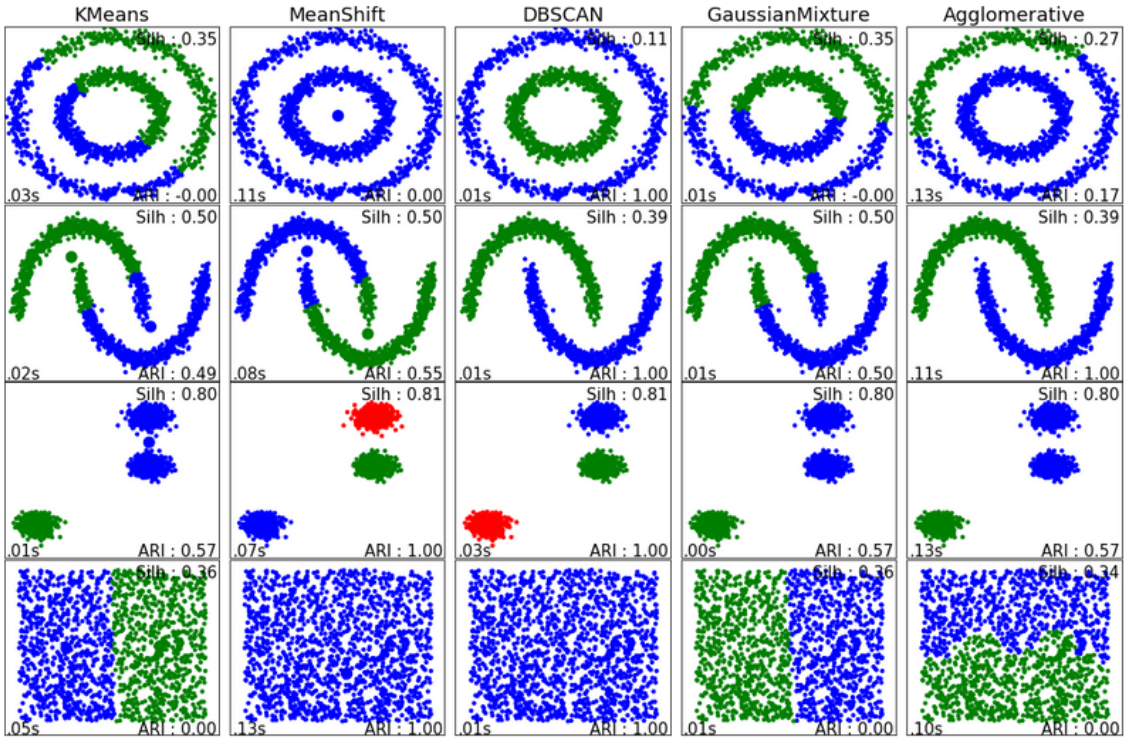
Problème :

- Ne tient pas compte du nombre de cas que l'on obtiendrait à partir de regroupements au hasard.

Solution :

- Corriger en fonction de la valeur attendue (***Expectation***) pour un score ***RI*** si les groupements sont effectués au hasard.
- Nom : ***Adjusted rand index*** (***ARI***).
  - Un score de « 0 » correspond alors à ce à quoi on s'attendrait avec des regroupements aléatoires.
  - Un score de « 1 » correspond à un groupement identique dans les deux ensembles.
  - Un score négatif veut dire qu'on fait pire qu'un regroupement aléatoire.

2.7. ENSEMBLE JOUET (SKLEARN)



2.8. BANQUE DE DONNÉES SIGNATURE (SIMULÉE)

2.8.1. KMEANS

In [5]:

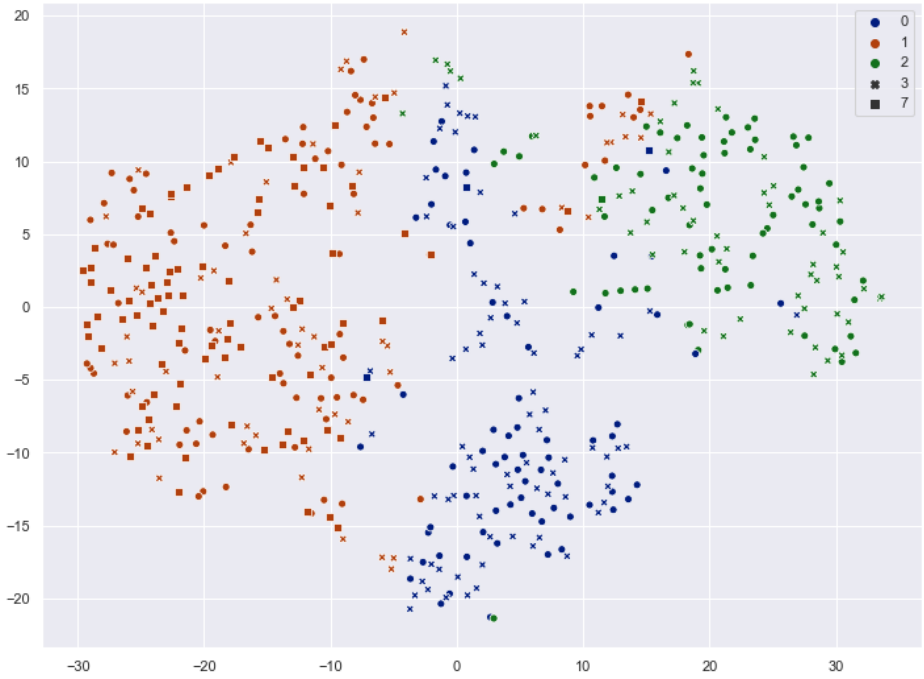
```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Importer une fonction de regroupement
from sklearn.cluster import KMeans

# Importer des métriques permettant d'évaluer le algorithmes
from sklearn import metrics

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.lines import Line2D
from matplotlib.patches import Patch

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------


# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
```

```
data = data[data.CIMDX != 1]
data = data[data.CIMDX != 4]
data = data[data.CIMDX != 5]
data = data[data.CIMDX != 6]
data = data[data.CIMDX != 0]

features_cols = ['PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITTT', 'STAIY
TT']

X = data.loc[:, features_cols]
y = data['CIMDX'].astype(int)

scaler = StandardScaler()
X = scaler.fit_transform(X)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ------------------------------------------------------------------
------------

k = 3

X_PCA = PCA(n_components=2).fit_transform(X)

#X_final = X_PCA
X_final = X

model = KMeans(n_clusters=k, random_state=0)
model.fit(X_final)

labels_pred = model.labels_
labels_true = np.array(y.astype(int))

if np.max(labels_pred) > 0:
    silhouette = metrics.silhouette_score(X, labels_pred)
ARI = metrics.adjusted_rand_score(labels_true, labels_pred)

print('\nSilhouette : ', silhouette)
print('\nARI : ', ARI)
print('\nMatrice de contingence : \n', metrics.cluster.contingency_
matrix(labels_true, labels_pred))

# tSNE
X_embedded = TSNE(n_components=2).fit_transform(X_final)
sns.set(rc={'figure.figsize':(12,9)})
palette = sns.color_palette("dark", k)
sns.scatterplot(X_embedded[:,0], X_embedded[:,1], hue=labels_pred,
style=labels_true, legend='full', palette=palette)
```

```
Silhouette :  0.24020523985744308

ARI :  0.04446898144586278

Matrice de contingence :
 [[68 88 69]
 [76 61 54]
 [ 3 89  1]]
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe87ab90>
```



2.8. BANQUE DE DONNÉES SIGNATURE (SIMULÉE)

2.8.2. MEAN SHIFT

In [6]:

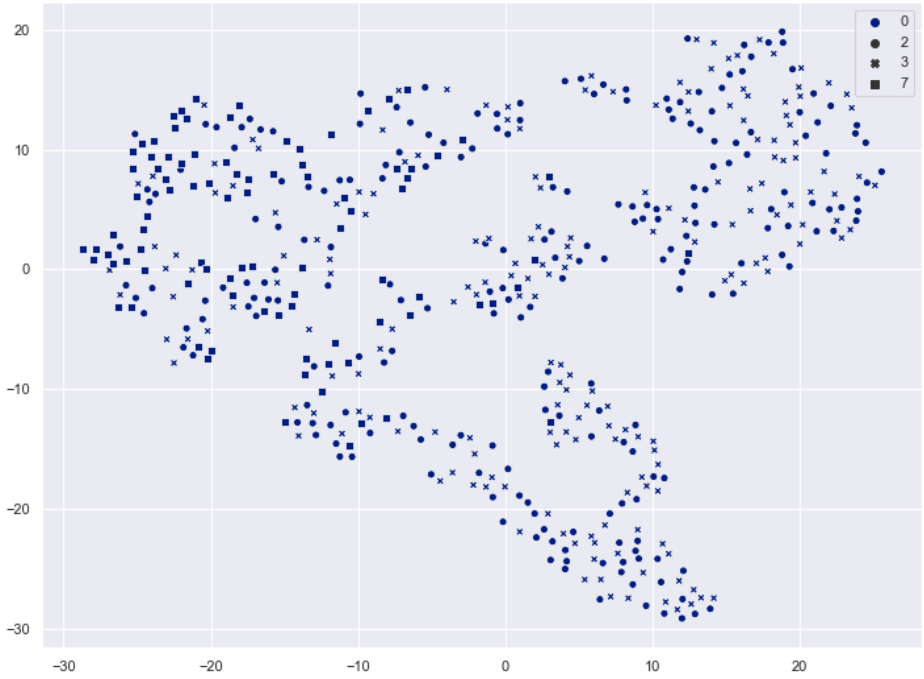
```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Importer une fonction de regroupement
from sklearn.cluster import MeanShift, estimate_bandwidth

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.lines import Line2D
from matplotlib.patches import Patch

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data[data.CIMDX != 1]
data = data[data.CIMDX != 4]
data = data[data.CIMDX != 5]
data = data[data.CIMDX != 6]
```

```
data = data[data.CIMDX != 0]

features_cols = ['PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITTT', 'STAIY
TT']

X = data.loc[:, features_cols]
y = data['CIMDX']

scaler = StandardScaler()
X = scaler.fit_transform(X)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ------------------------------------------------------------------
------------

X_PCA = PCA(n_components=2).fit_transform(X)

X_final = X_PCA
#X_final = X

bandwidth = estimate_bandwidth(X_final, quantile=0.3)

model = MeanShift(bandwidth=bandwidth, cluster_all=True)
model.fit(X_final)

labels_pred = (model.labels_).astype(int)
labels_true = np.array(y.astype(int))

if np.max(labels_pred) > 0:
    silhouette = metrics.silhouette_score(X, labels_pred)
ARI = metrics.adjusted_rand_score(labels_true, labels_pred)

print('\nSilhouette : ', silhouette)
print('\nARI : ', ARI)
print('\nMatrice de contingence : \n', metrics.cluster.contingency_
matrix(labels_true, labels_pred))

# tSNE
X_embedded = TSNE(n_components=2).fit_transform(X_final)
sns.set(rc={'figure.figsize':(12,9)})
palette = sns.color_palette("dark", np.size(np.unique(labels_pred
)))
sns.scatterplot(X_embedded[:,0], X_embedded[:,1], hue=labels_pred,
style=labels_true, legend='full', palette=palette)
```

```
Silhouette :  0.24020523985744308

ARI :  0.0

Matrice de contingence :
 [[225]
 [191]
 [ 93]]
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe944790>
```



2.8. BANQUE DE DONNÉES SIGNATURE (SIMULÉE)

2.8.3. DBSCAN

In [7]:

```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Importer une fonction de regroupement
from sklearn.mixture import GaussianMixture

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data[data.CIMDX != 1]
data = data[data.CIMDX != 4]
data = data[data.CIMDX != 5]
data = data[data.CIMDX != 6]
data = data[data.CIMDX != 0]

features_cols = ['PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITTT', 'STAIY
TT']
```

```
X = data.loc[:, features_cols]
y = data['CIMDX'].astype(int)

scaler = StandardScaler()
X = scaler.fit_transform(X)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ------------------------------------------------------------------
------------

k = 3

X_PCA = PCA(n_components=2).fit_transform(X)

X_final = X_PCA
#X_final = X

model = GaussianMixture(n_components=k, max_iter=1000, init_params=
'kmeans')
model.fit(X_final)

labels_pred = model.predict(X_final)
labels_true = np.array(y.astype(int))

if np.max(labels_pred) > 0:
    silhouette = metrics.silhouette_score(X, labels_pred)
ARI = metrics.adjusted_rand_score(labels_true, labels_pred)

print('\nSilhouette : ', silhouette)
print('\nARI : ', ARI)
print('\nMatrice de contingence : \n', metrics.cluster.contingency_
matrix(labels_true, labels_pred))

# tSNE
X_embedded = TSNE(n_components=2).fit_transform(X_final)
sns.set(rc={'figure.figsize':(12,9)})
palette = sns.color_palette("dark", np.size(np.unique(labels_pred
)))
sns.scatterplot(X_embedded[:,0], X_embedded[:,1], hue=labels_pred,
style=labels_true, legend='full', palette=palette)
```

```
Silhouette :  0.18397793868493506

ARI :  0.06638520714430979

Matrice de contingence :
 [[67 93 65]
 [43 93 55]
 [80 13  0]]
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd495cd0>
```

2.8. BANQUE DE DONNÉES SIGNATURE (SIMULÉE)

2.8.4. GAUSSIAN MIXTURE

In [8]:

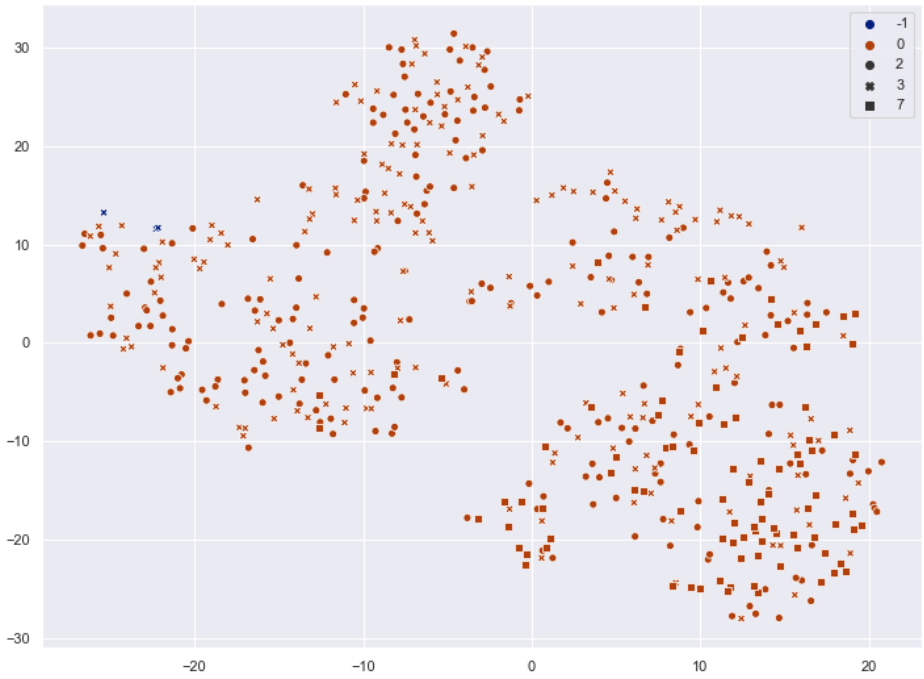
```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Importer une fonction de regroupement
from sklearn.cluster import DBSCAN

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.lines import Line2D
from matplotlib.patches import Patch

from sklearn import metrics

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data[data.CIMDX != 1]
data = data[data.CIMDX != 4]
```

```
data = data[data.CIMDX != 5]
data = data[data.CIMDX != 6]
data = data[data.CIMDX != 0]

features_cols = ['PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITTT', 'STAIY
TT']

X = data.loc[:, features_cols]
y = data['CIMDX'].astype(int)

scaler = StandardScaler()
X = scaler.fit_transform(X)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ------------------------------------------------------------------
------------

X_PCA = PCA(n_components=2).fit_transform(X)

#X_final = X_PCA
X_final = X

model = DBSCAN(eps=2, min_samples=5)
model.fit(X_final)

labels_pred = model.labels_
labels_true = np.array(y.astype(int))

if np.max(labels_pred) > 0:
    silhouette = metrics.silhouette_score(X_final, labels_pred)
ARI = metrics.adjusted_rand_score(labels_true, labels_pred)

print('\nSilhouette : ', silhouette)
print('\nARI : ', ARI)
print('\nMatrice de contingence : \n', metrics.cluster.contingency_
matrix(labels_true, labels_pred))

# tSNE
X_embedded = TSNE(n_components=2).fit_transform(X_final)
sns.set(rc={'figure.figsize':(12,9)})
palette = sns.color_palette("dark", np.size(np.unique(labels_pred
)))
sns.scatterplot(X_embedded[:,0], X_embedded[:,1], hue=labels_pred,
style=labels_true, legend='full', palette=palette)
```

```
Silhouette :  0.18397793868493506

ARI :  -0.00011908956818362425

Matrice de contingence :
 [[  0 225]
 [  3 188]
 [  0  93]]
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe961290>
```



2.8. BANQUE DE DONNÉES SIGNATURE (SIMULÉE)

2.8.5. AGGLOMERATIVE

In [9]:

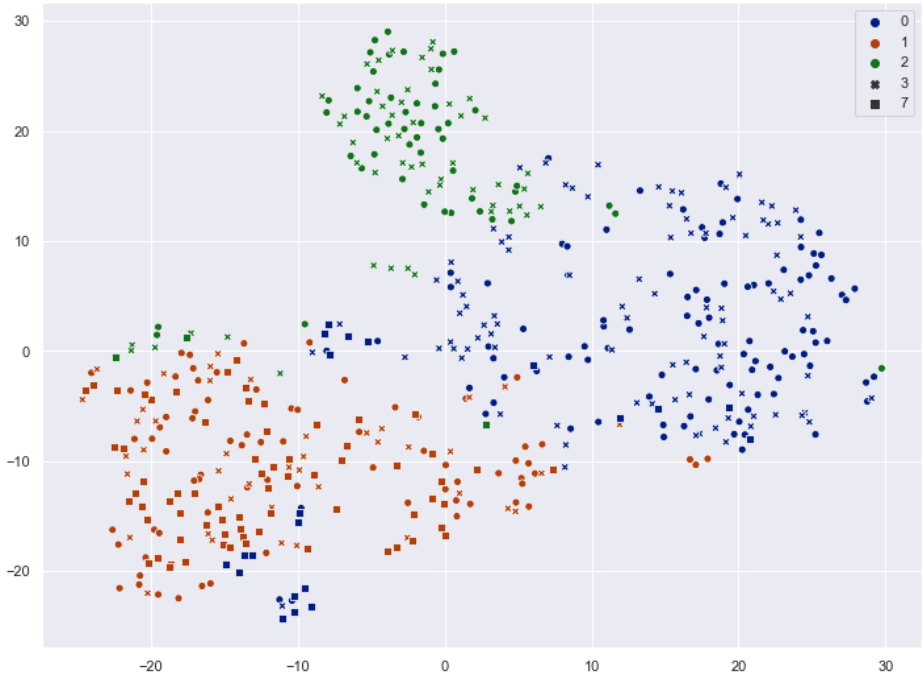
```
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore',category=DeprecationWarning
)
warnings.filterwarnings(action='ignore',category=FutureWarning)

# ----------------------------------------------------------------
------------
# ÉTAPE 1 : importer les librairies utiles
# ----------------------------------------------------------------
------------

%matplotlib inline

# Importer les librairies utiles pour l'analyse
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# ----------------------------------------------------------------
------------
# ÉTAPE 2 : importer les fonctions utiles
# ----------------------------------------------------------------
------------

# Importer les fonctions de prétraitement
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Importer une fonction de réduction de dimension
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Importer une fonction de regroupement
from sklearn.cluster import AgglomerativeClustering
from sklearn.neighbors import kneighbors_graph

# Importer les métriques
from sklearn import metrics

# ----------------------------------------------------------------
------------
# ÉTAPE 3 : importer et préparer le jeu de données
# ----------------------------------------------------------------
------------

# Importons un ensemble de données
data = pd.read_csv('../data/sim_data_signature_small.csv')
data = data[data.CIMDX != 1]
data = data[data.CIMDX != 4]
data = data[data.CIMDX != 5]
```

```
data = data[data.CIMDX != 6]
data = data[data.CIMDX != 0]

features_cols = ['PHQ9TT', 'CEVQOTT', 'DAST10TT', 'AUDITTT', 'STAIY
TT']

X = data.loc[:, features_cols]
y = data['CIMDX'].astype(int)

scaler = StandardScaler()
X = scaler.fit_transform(X)

# ------------------------------------------------------------------
------------
# ÉTAPE 4 : définir et entraîner le modèle
# ------------------------------------------------------------------
------------

k = 3

X_PCA = PCA(n_components=2).fit_transform(X)

#X_final = X_PCA
X_final = X

model = AgglomerativeClustering(linkage='complete', affinity='cosin
e', n_clusters=k)
model.fit(X_final)

labels_pred = model.labels_
labels_true = np.array(y.astype(int))

if np.max(labels_pred) > 0:
    silhouette = metrics.silhouette_score(X_final, labels_pred)
ARI = metrics.adjusted_rand_score(labels_true, labels_pred)

print('\nSilhouette : ', silhouette)
print('\nARI : ', ARI)
print('\nMatrice de contingence : \n', metrics.cluster.contingency_
matrix(labels_true, labels_pred))

# tSNE
X_embedded = TSNE(n_components=2).fit_transform(X_final)
sns.set(rc={'figure.figsize':(12,9)})
palette = sns.color_palette("dark", np.size(np.unique(labels_pred
)))
sns.scatterplot(X_embedded[:,0], X_embedded[:,1], hue=labels_pred,
style=labels_true, legend='full', palette=palette)
```

```
Silhouette :  0.1577373153859903

ARI :  0.040091574072661l2

Matrice de contingence :
 [[99 75 51]
 [95 44 52]
 [21 69  3]]
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xf37ffd0>
```



2.9. MÉTHODES D'ENSEMBLES

2.9. MÉTHODES D'ENSEMBLES

Afin de diminuer la variance de notre solution, on peut utiliser une **méthode d'ensemble**.

L'idée est la même que lorsque nous avions vu le *bagging* et le *boosting*.

Il n'existe toutefois actuellement pas de fonction dans *Sklearn* permettant d'utiliser une méthode d'ensemble dans le cadre d'un algorithme de regroupement.

- Néanmoins, il est possible d'utiliser des librairies proposées par différents auteurs (ex. *Ronan, 2018*).
- https://github.com/NaegleLab/OpenEnsembles

RÉCAPITULATIF DES MÉTHODES DE REGROUPEMENT

Voici certains avantages/inconvénients de ces méthodes de regroupement :

K-moyennes (basée sur le partitionnement)

- Avantages
  - Rapide
  - Produit des groupements compacts
- Inconvénients
  - Difficile de trouver le bon nombre de regroupements (*k*).
  - Sensible à l'état initial.
    (utiliser l'option d'initialisation *kmeans++*)
  - Limité à des regroupements relativement circulaires.

Décalage de moyenne (basée sur la densité)

- Avantages
  - Flexible quant à la forme des groupes
    (mais moins que DBSCAN)
  - N'a pas besoin de connaître le nombre de regroupements.
- Inconvénients
  - La « bande passante » (*bandwidth*) est difficile à ajuster.
  - Moins efficace si la densité n'est pas homogène.
  - Demande beaucoup de mémoire si beaucoup de données.

DBSCAN (basée sur la densité)

- Avantages
  - Efficace peu importe la forme des données.