

Cours Machine Learning

Chapitre 05 Apprentissage Non-Supervisé

Objectifs d'apprentissage :

- Maîtriser les algorithmes de clustering (K-Means, DBSCAN, Hierarchical)
- Comprendre la réduction de dimensionnalité (PCA, t-SNE, UMAP)
- Détecter des anomalies dans les données
- Appliquer ces techniques à des problèmes réels

Prérequis : Chapitres 00, 01, 02

Durée estimée : 5-7 heures

Notebooks : 05_*.ipynb

Table des matières

1	Introduction	3
I	Clustering	3
2	K-Means	3
2.1	Principe	3
2.2	Algorithme	4
2.3	Initialisation	4
2.4	Choix de K (Méthode du Coude)	4
2.5	Avantages et Limites	5
3	DBSCAN	5
3.1	Principe	5
3.2	Algorithme	6
3.3	Avantages et Limites	6
4	Clustering Hiérarchique	7
4.1	Principe	7
4.2	Linkage Criteria	7
4.3	Dendrogramme	8
II	Réduction de Dimensionnalité	8
5	Principal Component Analysis (PCA)	8
5.1	Motivation	8
5.2	Principe Mathématique	8
5.3	Variance Expliquée	9
5.4	Avantages et Limites	9
6	t-SNE	10
6.1	Principe	10
6.2	Hyperparamètre : Perplexity	10
6.3	Avantages et Limites	10
7	UMAP	11
III	Détection d'Anomalies	11
8	Introduction	12
9	Méthodes	12
9.1	Isolation Forest	12
9.2	One-Class SVM	12
9.3	Local Outlier Factor (LOF)	12

10 Résumé	13
10.1 Points Clés	13
11 Pour Aller Plus Loin	13

1 Introduction

Définition

Apprentissage Non-Supervisé L'apprentissage non-supervisé consiste à découvrir des structures cachées dans des données **non étiquetées** $\{\mathbf{x}_i\}_{i=1}^n$ sans labels y .

Différence avec apprentissage supervisé :

- **Supervisé** : Apprendre $f : X \rightarrow Y$ à partir de $\{(\mathbf{x}_i, y_i)\}$
- **Non-supervisé** : Découvrir la structure de $\{\mathbf{x}_i\}$ (groupes, dimensions, anomalies)

Tâches principales :

1. **Clustering** : Regrouper instances similaires
2. **Réduction de dimensionnalité** : Projeter données dans espace réduit
3. **Détection d'anomalies** : Identifier points atypiques
4. **Apprentissage de représentation** : Autoencoders, embeddings

Exemple

Applications

- **Marketing** : Segmentation de clients (clustering)
- **Biologie** : Découverte de nouveaux types cellulaires
- **Compression** : PCA pour réduire taille des données
- **Visualisation** : t-SNE pour visualiser données haute dimension
- **Sécurité** : Détection de fraudes, intrusions réseau

Première partie

Clustering

2 K-Means

2.1 Principe

Définition

K-Means K-Means partitionne n instances en K clusters en minimisant la variance intra-cluster.

Objectif : Minimiser l'inertie (somme des distances carrées au centroïde) :

$$J = \sum_{k=1}^K \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \mu_k\|^2 \quad (1)$$

où μ_k est le centroïde du cluster C_k .

2.2 Algorithme

Algorithm 1 K-Means (Lloyd's Algorithm)

Require: Données $\{\mathbf{x}_i\}_{i=1}^n$, nombre de clusters K

Ensure: Clusters $\{C_1, \dots, C_K\}$, centroïdes $\{\mu_1, \dots, \mu_K\}$

- 1: Initialiser K centroïdes aléatoirement
 - 2: **repeat**
 - 3: **(E-step)** Assigner chaque point au centroïde le plus proche :
 - 4: $C_k = \{\mathbf{x}_i : k = \operatorname{argmin}_j \|\mathbf{x}_i - \mu_j\|^2\}$
 - 5: **(M-step)** Recalculer centroïdes :
 - 6: $\mu_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$
 - 7: **until** Convergence (centroïdes ne bougent plus)
-

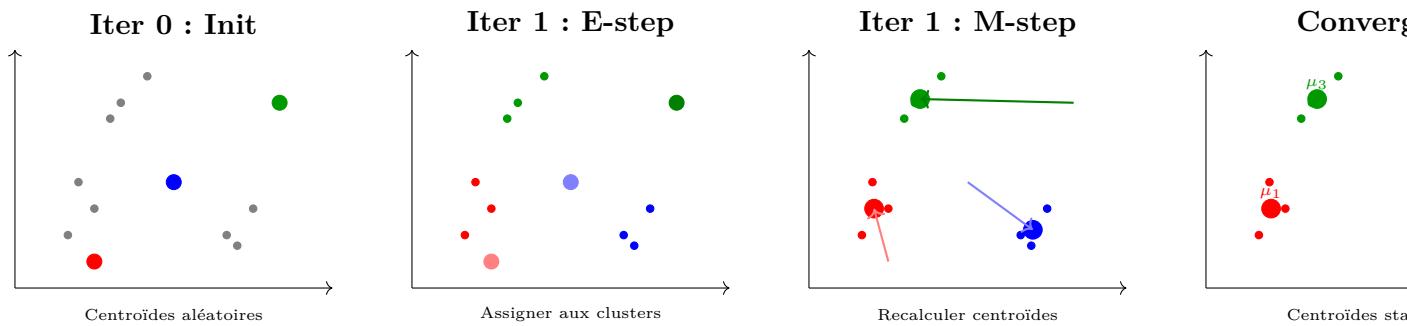


FIGURE 1 – Itérations de K-Means : (1) initialisation aléatoire des centroïdes, (2) E-step assigne chaque point au centroïde le plus proche, (3) M-step recalcule les centroïdes comme moyenne de chaque cluster, (4) convergence quand les centroïdes ne bougent plus.

Complexité : $O(nKdT)$ où T = nombre d'itérations (généralement petit).

2.3 Initialisation

Problème : K-Means sensible à l'initialisation (peut converger vers minimum local).

Solutions :

- **Random** : Choisir K points aléatoires
- **K-Means++** : Initialisation intelligente (éloigner les centroïdes initiaux)
- **Multiple runs** : Exécuter plusieurs fois, garder meilleur résultat

2.4 Choix de K (Méthode du Coude)

Elbow Method :

1. Exécuter K-Means pour différentes valeurs de K
2. Tracer inertie en fonction de K
3. Choisir K au "coude" de la courbe (compromis)

Autres méthodes :

- **Silhouette Score** : Mesure la cohésion et séparation des clusters
- **Gap Statistic** : Compare inertie à une baseline aléatoire

2.5 Avantages et Limites

Avantages :

- Simple et rapide
- Scalable (grands datasets)
- Facile à interpréter

Limites :

- Nécessite spécifier K a priori
- Suppose clusters sphériques et de taille similaire
- Sensible aux outliers
- Sensible à l'initialisation
- Nécessite normalisation des features

```

1 from sklearn.cluster import KMeans
2 from sklearn.preprocessing import StandardScaler
3
4 # Normalisation (important!)
5 scaler = StandardScaler()
6 X_scaled = scaler.fit_transform(X)
7
8 # K-Means
9 kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, random_state
   =42)
10 labels = kmeans.fit_predict(X_scaled)
11 centroids = kmeans.cluster_centers_
12
13 # Inertie
14 print(f"Inertie: {kmeans.inertia_:.2f}")

```

Listing 1 – K-Means avec scikit-learn

3 DBSCAN

3.1 Principe

Définition

DBSCAN (Density-Based Spatial Clustering) DBSCAN regroupe les points densément connectés et identifie les outliers. Basé sur deux paramètres : ϵ (rayon) et `min_samples` (densité minimale).

Types de points :

- **Core point** : Point avec au moins `min_samples` voisins dans rayon ϵ
- **Border point** : Voisin d'un core point mais pas core lui-même
- **Noise point** : Ni core ni border (outlier)

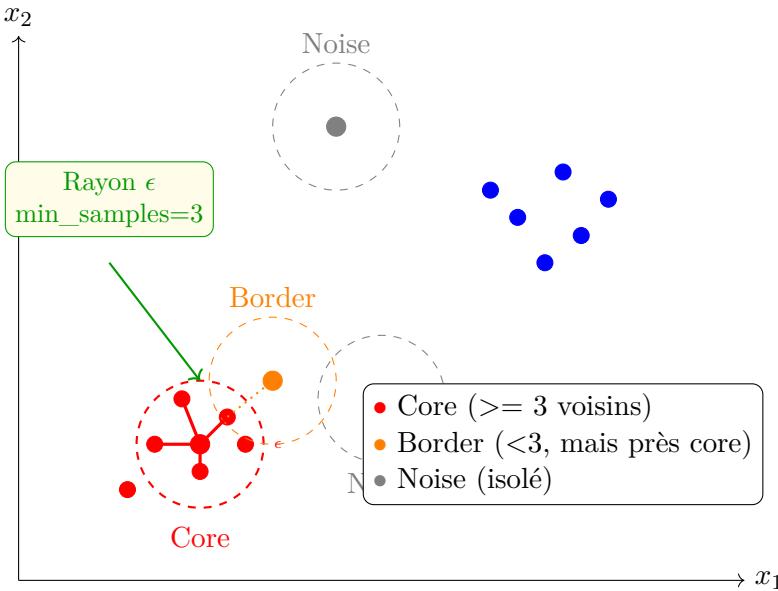


FIGURE 2 – DBSCAN : types de points selon densité. Les core points (rouge) ont $\geq \text{min_samples}$ voisins dans rayon ϵ , les border points (orange) sont proches d'un core mais pas core eux-mêmes, les noise points (gris) sont isolés et considérés comme outliers.

3.2 Algorithme

Algorithm 2 DBSCAN

Require: Données $\{\mathbf{x}_i\}$, ϵ , min_samples

- 1: Marquer tous points comme non-visités
- 2: **for** chaque point \mathbf{x}_i non-visité **do**
- 3: Marquer \mathbf{x}_i comme visité
- 4: Trouver voisins dans rayon ϵ
- 5: **if** moins de min_samples voisins **then**
- 6: Marquer comme noise
- 7: **else**
- 8: Créer nouveau cluster, ajouter \mathbf{x}_i
- 9: Étendre cluster récursivement avec voisins
- 10: **end if**
- 11: **end for**

3.3 Avantages et Limites

Avantages :

- Pas besoin de spécifier K
- Détecte automatiquement les outliers
- Gère clusters de formes arbitraires
- Robuste au bruit

Limites :

- Choix délicat de ϵ et min_samples
- Difficile avec densités variables

- Coût : $O(n^2)$ (ou $O(n \log n)$ avec index spatial)

```

1 from sklearn.cluster import DBSCAN
2
3 dbSCAN = DBSCAN(eps=0.5, min_samples=5)
4 labels = dbSCAN.fit_predict(X_scaled)
5
6 # -1 = noise/outliers
7 n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
8 n_noise = list(labels).count(-1)
9 print(f"Clusters: {n_clusters}, Outliers: {n_noise}")

```

Listing 2 – DBSCAN

4 Clustering Hiérarchique

4.1 Principe

Définition

Clustering Hiérarchique Construit une hiérarchie de clusters (dendrogramme) en fusionnant ou divisant successivement les clusters.

Deux approches :

- **Agglomératif (bottom-up)** : Commence avec n clusters (1 point chacun), fusionne itérativement
- **Divisif (top-down)** : Commence avec 1 cluster (tous points), divise itérativement
L'approche agglomérative est la plus courante.

4.2 Linkage Criteria

Critères pour mesurer distance entre clusters :

- **Single linkage** : Distance minimale entre points de 2 clusters

$$d(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}) \quad (2)$$

- **Complete linkage** : Distance maximale

$$d(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}) \quad (3)$$

- **Average linkage** : Distance moyenne

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y}) \quad (4)$$

- **Ward** : Minimise variance intra-cluster (comme K-Means)

4.3 Dendrogramme

Le dendrogramme visualise la hiérarchie. On peut "couper" à une hauteur pour obtenir K clusters.

```

1 from sklearn.cluster import AgglomerativeClustering
2 from scipy.cluster.hierarchy import dendrogram, linkage
3 import matplotlib.pyplot as plt
4
5 # Clustering
6 hc = AgglomerativeClustering(n_clusters=3, linkage='ward')
7 labels = hc.fit_predict(X_scaled)
8
9 # Dendrogramme
10 linkage_matrix = linkage(X_scaled, method='ward')
11 plt.figure(figsize=(10, 5))
12 dendrogram(linkage_matrix)
13 plt.title('Dendrogramme')
14 plt.show()

```

Listing 3 – Clustering Hiérarchique

Deuxième partie

Réduction de Dimensionnalité

5 Principal Component Analysis (PCA)

5.1 Motivation

Problèmes en haute dimension :

- Visualisation difficile ($d > 3$)
- Curse of dimensionality
- Redondance (features corrélées)
- Coût de calcul élevé

Objectif PCA : Trouver projection linéaire dans espace de dimension $k < d$ qui préserve au maximum la variance.

5.2 Principe Mathématique

Définition

PCA PCA cherche les directions (composantes principales) de variance maximale via décomposition en valeurs propres de la matrice de covariance.

Algorithme :

1. Centrer les données : $\mathbf{X}_c = \mathbf{X} - \bar{\mathbf{x}}$
2. Calculer matrice de covariance : $\mathbf{C} = \frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$

3. Diagonaliser : trouver valeurs propres λ_i et vecteurs propres \mathbf{v}_i de \mathbf{C}
4. Trier par λ_i décroissant
5. Garder les k premiers vecteurs propres (composantes principales)
6. Projeter : $\mathbf{Z} = \mathbf{X}_c \mathbf{V}_k$

Via SVD (plus efficace) :

$$\mathbf{X}_c = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^T \quad (5)$$

Les colonnes de \mathbf{V} sont les composantes principales.

5.3 Variance Expliquée

Variance expliquée par k composantes :

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^d \lambda_i} \quad (6)$$

Choix de k :

- Conserver 90% ou 95% de la variance
- Méthode du coude (scree plot)

5.4 Avantages et Limites

Avantages :

- Réduction efficace de dimension
- Élimine redondance (décorrèle features)
- Accélère algorithmes downstream
- Visualisation (projection 2D/3D)
- Interprétable (composantes = combinaisons linéaires)

Limites :

- Transformation **linéaire** uniquement
- Suppose variance = information (pas toujours vrai)
- Sensible à l'échelle (nécessite standardisation)
- Perte d'interprétabilité des features originales

```

1 from sklearn.decomposition import PCA
2
3 # PCA
4 pca = PCA(n_components=2)
5 X_pca = pca.fit_transform(X_scaled)
6
7 # Variance expliquée
8 print("Variance expliquée par composante:")
9 print(pca.explained_variance_ratio_)
10 print(f"Total: {pca.explained_variance_ratio_.sum():.2%}")
11
12 # Visualisation
13 plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
14 plt.xlabel('PC1')
15 plt.ylabel('PC2')
```

```
16 plt.show()
```

Listing 4 – PCA

6 t-SNE

6.1 Principe

Définition

t-SNE (t-Distributed Stochastic Neighbor Embedding) t-SNE est une technique de réduction de dimensionnalité **non-linéaire** optimisée pour la visualisation. Préserve les structures locales (voisinages).

Idée :

1. Modéliser similarités entre points en haute dimension (gaussienne)
2. Modéliser similarités en basse dimension (t-Student)
3. Minimiser divergence KL entre les deux distributions

6.2 Hyperparamètre : Perplexity

perplexity contrôle le nombre effectif de voisins considérés :

- Petite perplexity (5-10) : Focus sur structure locale
- Grande perplexity (30-50) : Focus sur structure globale
- Typiquement : 30-50 pour datasets moyens

6.3 Avantages et Limites

Avantages :

- Excellente visualisation de structures complexes
- Capture relations non-linéaires
- Révèle clusters naturels

Limites :

- **Coût** : $O(n^2)$ (lent pour gros datasets)
- **Non-déterministe** : Résultats varient selon initialisation
- **Pas d'embedding new data** : Doit réentraîner
- **Distances non interprétables** : Uniquement pour visualisation
- **Sensible à hyperparamètres**

/!\ Attention

t-SNE est conçu pour **visualisation**, pas pour réduction de dimension comme preprocesing. Utiliser PCA pour preprocessing.

```
1 from sklearn.manifold import TSNE
2
3 # t-SNE (lent, r duire dimension avec PCA d'abord si d > 50)
4 if X_scaled.shape[1] > 50:
```

```

5     pca = PCA(n_components=50)
6     X_reduced = pca.fit_transform(X_scaled)
7 else:
8     X_reduced = X_scaled
9
10    tsne = TSNE(n_components=2, perplexity=30, random_state=42)
11    X_tsne = tsne.fit_transform(X_reduced)
12
13    plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')
14    plt.title('t-SNE Visualization')
15    plt.show()

```

Listing 5 – t-SNE

7 UMAP

Définition

UMAP (Uniform Manifold Approximation and Projection) UMAP est une alternative récente à t-SNE, basée sur la théorie des variétés. Plus rapide et préserve mieux la structure globale.

Avantages vs t-SNE :

- Plus rapide ($O(n)$) avec approximations
- Meilleure préservation structure globale
- Peut projeter new data
- Moins sensible aux hyperparamètres

```

1 import umap
2
3 reducer = umap.UMAP(n_components=2, random_state=42)
4 X_umap = reducer.fit_transform(X_scaled)
5
6 plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y, cmap='viridis')
7 plt.title('UMAP Visualization')
8 plt.show()

```

Listing 6 – UMAP

Troisième partie

Détection d'Anomalies

8 Introduction

Définition

Anomalie (Outlier) Point significativement différent de la majorité des données. Peut être une erreur de mesure, fraude, événement rare.

Applications :

- Détection de fraudes (cartes bancaires)
- Intrusions réseau, cyberattaques
- Défauts de fabrication
- Diagnostic médical (cas atypiques)

9 Méthodes

9.1 Isolation Forest

Idée : Anomalies sont rares et différentes \Rightarrow faciles à isoler.

Définition

Isolation Forest Construit arbres aléatoires qui isolent les points. Anomalies nécessitent moins de splits pour être isolées.

Anomaly Score : Profondeur moyenne d'isolation (normalisée). Score proche de 1 = anomalie.

```

1 from sklearn.ensemble import IsolationForest
2
3 iso_forest = IsolationForest(contamination=0.1, random_state=42)
4 predictions = iso_forest.fit_predict(X_scaled)
5 # -1 = anomalie, 1 = normal
6
7 anomaly_scores = iso_forest.score_samples(X_scaled)
8 # Scores n gatifs, plus n gatif = plus anormal

```

Listing 7 – Isolation Forest

9.2 One-Class SVM

SVM entraîné à apprendre la frontière de la distribution "normale". Points hors frontière = anomalies.

9.3 Local Outlier Factor (LOF)

Mesure la densité locale d'un point vs ses voisins. Point moins dense que ses voisins = outlier.

10 Résumé

10.1 Points Clés

Clustering :

- **K-Means** : Rapide, clusters sphériques, nécessite K
- **DBSCAN** : Déetecte outliers, formes arbitraires, sensible ϵ
- **Hiérarchique** : Dendrogramme, flexible

Réduction dimensionnalité :

- **PCA** : Linéaire, variance maximale, preprocessing
- **t-SNE** : Non-linéaire, visualisation, lent
- **UMAP** : Comme t-SNE mais plus rapide

Détection anomalies :

- **Isolation Forest** : Rapide, efficace
- **One-Class SVM, LOF** : Alternatives

11 Pour Aller Plus Loin

Chapitre suivant : **Chapitre 06 - Réseaux de Neurones Fondamentaux**

Références

1. Hastie, T., et al. (2009). *The Elements of Statistical Learning*. Springer.
2. van der Maaten, L., & Hinton, G. (2008). "Visualizing Data using t-SNE". *JMLR*.
3. McInnes, L., et al. (2018). "UMAP : Uniform Manifold Approximation and Projection". *arXiv*.