Université d'Évry Paris-Saclay
Master Ingénierie des Systèmes Complexes — Parcours TNI

Projet Big Data

Analyse Comparative du Clustering K-means
Scikit-learn vs Apache Spark

Travail réalisé par :

Hocine Yacine **BEY**
Amel **FERRAHI**

Tutoré par :
Dr. Kenneth Ezukwoke

03 décembre 2025

Table des matières

Introduction

Ce projet s'inscrit dans le cadre du module **Big Data Clustering** du Master Ingénierie des Systèmes Complexes. L'objectif est d'étudier et de comparer le comportement réel de l'algorithme **K-means** dans deux environnements computationnels présentant des philosophies radicalement différentes :

— **Scikit-learn**, bibliothèque Python optimisée pour le calcul local en mémoire ;
— **Apache Spark**, moteur distribué conçu pour les traitements massifs sur de multiples nœuds.

La comparaison porte exclusivement sur des données réellement exécutées dans les environnements Python et PySpark. Aucune valeur théorique ou dataset non utilisé n'est introduit. Les résultats présentés sont issus :

— du dataset **Wine Quality** (1599 lignes) ;
— du dataset **Adult** (32 561 lignes après nettoyage) ;
— de jeux artificiels générés (50k–200k) pour scikit-learn ;
— de jeux artificiels mal distribués (100k–300k) pour Spark.

Les mesures réelles incluent :

— temps d'exécution,
— CPU/RAM,
— inertie,
— silhouette score (lorsqu'il est calculable).

Chapitre 1

Méthodologie expérimentale

1.1 Choix des environnements et justification

Les deux environnements étudiés incarnent deux logiques computationnelles distinctes.

Scikit-learn : calcul local optimisé

Scikit-learn repose sur :
— un stockage dense en mémoire,
— des opérations vectorisées compilées (BLAS/LAPACK),
— une absence totale de communication inter-nœuds.
Cette approche est extrêmement rapide tant que les données tiennent dans la RAM. Elle est idéale pour l'analyse exploratoire et les volumes moyens.

Apache Spark : calcul distribué

Spark utilise :
— des partitions distribuées,
— des synchronisations à chaque itération de K-means,
— un DAG scheduler introduisant un coût fixe important,
— de la sérialisation/désérialisation permanente.
Cette architecture entraîne un surcoût sur petits volumes mais devient indispensable à grande échelle.

1.2 Jeux de données utilisés

— Wine Quality (1599 lignes, 11 features)
— Adult (32 561 lignes après nettoyage, 104 features après one-hot)
— Blobs scikit-learn : 50k, 100k, 200k (10D)
— Blobs Spark : 100k et 300k (10D)

1.3 Prétraitement identique

Pour assurer une comparaison équitable :
— Normalisation via StandardScaler

— Encodage One-hot sur Adult pour sklearn et Spark
— k constant : 4 (datasets réels), 10 (datasets artificiels)
— n_init = 10

1.4 Rappel de l'algorithme K-means

1.4.1 Formulation mathématique complète de K-means

Soit un ensemble de points $X = \{x_1, \ldots, x_n\}$ avec $x_i \in \mathbb{R}^d$. L'algorithme K-means cherche à partitionner cet ensemble en k clusters $C_1, \ldots, C_k$ en minimisant la fonction objectif suivante :

$$J = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

où μ_j désigne le centroïde du cluster C_j, défini comme la moyenne des points du cluster :

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

L'algorithme procède de manière itérative selon deux étapes successives :
— **Étape d'affectation :** chaque point est affecté au cluster dont le centroïde minimise la distance euclidienne.

$$x_i \in C_j \iff j = \arg\min_{\ell} \|x_i - \mu_\ell\|$$

— **Étape de mise à jour :** les centroïdes sont recalculés comme moyennes des clusters.

K-means converge vers un minimum local de la fonction J. Le résultat dépend donc fortement :
— de l'initialisation des centroïdes,
— de la distribution des données,
— du nombre de clusters k.

1.4.2 Complexité algorithmique de K-means

La complexité théorique d'une exécution de K-means est donnée par :

$$O(n \cdot k \cdot d \cdot i)$$

où :
— n est le nombre d'échantillons,
— k le nombre de clusters,
— d la dimension des données,
— i le nombre d'itérations jusqu'à convergence.

Cette complexité explique :
— l'augmentation quasi linéaire du temps lorsque n double,
— l'impact très fort du One-Hot Encoding qui augmente d,
— la difficulté du passage à l'échelle pour les datasets très larges.

1.5 Erreur d'encodage catégoriel initiale

1.5.1 Analyse mathématique de l'erreur d'encodage

Dans l'encodage initial par entiers, une variable catégorielle X prenant les valeurs :

$$\{\texttt{Private}, \texttt{Self-emp}, \texttt{State-gov}\}$$

était transformée en une variable numérique :

$$X \in \{0, 1, 2\}$$

Dans ce cas, la distance entre deux catégories devient :

$$d(0, 2) = 2$$

ce qui implique l'existence d'un ordre et d'une métrique entre symboles, ce qui est mathématiquement faux car ces catégories n'appartiennent pas à un espace métrique naturel.

Ainsi, la distance euclidienne utilisée par K-means n'est plus une distance entre individus mais une distance arbitraire entre étiquettes. Cela viole l'hypothèse fondamentale de K-means : les données doivent appartenir à un espace vectoriel métrique réel.

1.5.2 Justification géométrique du One-Hot Encoding

Le One-Hot Encoding transforme une variable catégorielle à m modalités en un vecteur binaire de dimension m appartenant à $\mathbb{R}^m$.

Par exemple :

$$\texttt{Male} \to (1, 0), \quad \texttt{Female} \to (0, 1)$$

La distance entre deux catégories devient alors :

$$d((1, 0), (0, 1)) = \sqrt{(1 - 0)^2 + (0 - 1)^2} = \sqrt{2}$$

Toutes les catégories sont donc équidistantes, ce qui respecte enfin la géométrie euclidienne exigée par K-means. Cette transformation restaure la validité mathématique du clustering.

Chapitre 2

Analyse des performances

2.1 Wine Quality (1599 lignes)

Scikit-learn (k = 4)

— Temps : **0.0386 s**
— CPU : 40.7%
— RAM : +0.37 MB
— Inertie : **11 734.23**
— Silhouette : **0.18518**

Spark (k = 4)

— Temps : **5.78 s**
— Inertie : **11 287.81**
— Silhouette : non calculée

Analyse

Spark est **150 fois plus lent** que scikit-learn sur ce petit dataset. Cette différence provient uniquement des coûts structurels du moteur distribué.

2.2 Adult (32 561 lignes)

Scikit-learn — One-hot (104 features)

— Temps : **2.512 s**
— Inertie : **2 925 674.36**
— Silhouette : **0.06296**

Spark — One-hot

— Temps : **66.76 s**
— Inertie : **3 110 291.68**
— Silhouette : **0.10510**

Analyse

Spark est **26 fois plus lent**, mais obtient un silhouette légèrement meilleur que sk-learn.

2.3 Blobs générés : Scikit-learn

TABLE 2.1 – Performances scikit-learn pour k=10 et 10D

n	Temps (s)	Inertie	Silhouette
50 000	0.9508	5.00e5	0.7631
100 000	1.3246	1.00e6	–
200 000	2.8236	2.00e6	–

2.4 Spark — 100k / 300k

— **100k : 57.8789 s**, inertie = 6.0266e4
— **300k : 175.2703 s**, inertie = 1.8131e5

Chapitre 3

Analyse qualitative

3.1 Définition mathématique du silhouette score

Pour tout point x, le score de silhouette est défini par :

$$s(x) = \frac{b(x) - a(x)}{\max(a(x), b(x))}$$

où :

— $a(x)$ est la distance moyenne entre x et les autres points de son cluster,
— $b(x)$ est la plus petite distance moyenne entre x et les clusters voisins.

Le score global est la moyenne des $s(x)$. Il appartient à l'intervalle $[-1, 1]$.

3.2 Silhouette

— Wine sklearn : 0.18518
— Adult sklearn : 0.06296
— Adult Spark : 0.10510

3.3 Interprétation

— sklearn produit des clusters légèrement plus compacts sur Wine.
— Spark s'en sort mieux que sklearn sur Adult, probablement grâce à la représentation sparse vectorisée.

3.4 Interprétation géométrique des faibles scores observés

Les faibles valeurs du silhouette obtenues pour le dataset Adult indiquent que :

$$a(x) \approx b(x)$$

ce qui signifie que les points sont aussi proches de leur propre cluster que des clusters voisins.

Cela traduit :

— un fort chevauchement des clusters,
— une absence de séparation nette dans l'espace des variables,
— une structure intrinsèquement non convexe.

Ce comportement est cohérent avec la nature socio-économique du dataset Adult, caractérisé par des frontières floues entre groupes d'individus.

Chapitre 4

Discussion critique

4.1 Avantages de sklearn

— Temps d'exécution extrêmement rapides
— Idéal pour les petites et moyennes données
— Clustering plus stable numériquement

4.2 Limites de sklearn

— Dépend entièrement de la RAM locale
— Ne peut pas passer à des milliards de lignes

4.3 Avantages de Spark

— Peut traiter d'énormes volumes
— Scalable horizontalement
— Résilience en cas de panne

4.4 Limites de Spark

— Très fort surcoût sur petits et moyens datasets
— Synchronisations coûteuses

4.5 Synthèse

sklearn = rapidité locale ; Spark = capacité et scalabilité.

Recommandations d'utilisation des deux environnements

L'étude menée permet de dégager des recommandations claires quant au choix de l'environnement le plus adapté en fonction du contexte et de la taille des données.

Quand privilégier Scikit-learn ?

Scikit-learn est particulièrement adapté lorsque :

— la taille du dataset reste modérée (jusqu'à quelques centaines de milliers d'échantillons) ;
— les données tiennent intégralement en mémoire RAM ;
— l'objectif est l'exploration rapide, le prototypage ou l'analyse scientifique ;
— la priorité est à la simplicité, à la flexibilité et au faible temps d'exécution.

Sa rapidité d'exécution et la stabilité numérique des résultats en font un outil idéal pour l'expérimentation locale et les tâches d'analyse courantes.

Quand privilégier Apache Spark ?

Apache Spark devient pertinent dans les situations suivantes :

— traitement de données volumineuses dépassant les capacités mémoire locales ;
— besoin de scalabilité horizontale et de distribution du calcul ;
— contexte industriel nécessitant robustesse, tolérance aux pannes et répétabilité ;
— pipelines Big Data intégrant plusieurs étapes distribuées (ETL, apprentissage, stockage).

Malgré un surcoût important pour les petits datasets, Spark est l'environnement à privilégier lorsque la montée en charge et la gestion de données massives constituent la priorité.

Limites et perspectives

Bien que l'étude fournisse une analyse complète des performances de K-means sur deux environnements distincts, certaines limites méthodologiques méritent d'être soulignées.

Limites du travail

— **Absence de calcul du silhouette pour Spark sur certains datasets :** MLlib ne fournit pas nativement cette mesure, ce qui limite la comparaison directe de la qualité du clustering.
— **Sensibilité à la dimensionnalité :** l'encodage One-Hot du dataset Adult augmente fortement la dimension, ce qui influence directement la complexité et les temps de calcul.
— **K-means non optimal pour des clusters non convexes :** certains datasets, notamment Adult, présentent des frontières floues qui limitent la pertinence du silhouette et de l'inertie.
— **Absence d'algorithmes alternatifs :** seule la version classique de K-means a été testée, alors que des variantes comme MiniBatch K-means ou K-means|| auraient pu améliorer certains résultats.

Perspectives

— **Tester des algorithmes parallèles alternatifs :** MiniBatch K-means, Streaming K-means ou des méthodes hiérarchiques distribuées.
— **Intégrer une réduction de dimension (PCA) :** afin d'atténuer l'impact du One-Hot Encoding et d'améliorer la compacité des clusters.
— **Mesurer précisément l'utilisation mémoire dans Spark :** via un monitoring avancé des workers et de l'exécuteur.
— **Comparer d'autres métriques de qualité :** Davies–Bouldin, Calinski–Harabasz, ou la stabilité entre exécutions.
— **Étendre l'étude à des pipelines complets Big Data :** ingestion, transformation, normalisation, clustering et visualisation distribuée.

Ces pistes d'amélioration permettraient d'affiner l'analyse, d'élargir la comparaison à d'autres algorithmes et d'explorer des scénarios industriels plus complexes.

Conclusion

Cette étude comparative, basée exclusivement sur des exécutions réelles, montre que :

— **Scikit-learn** est extrêmement performant pour les jeux de données contenant jusqu'à plusieurs centaines de milliers de points.

— **Apache Spark** devient pertinent uniquement lorsque le volume dépasse les capacités mémoire d'une machine locale.

— La qualité du clustering reste comparable, avec quelques variations dépendant des datasets.

Le choix de l'environnement dépend donc principalement de la **taille des données** et du **contexte d'exécution**, et non de la qualité de l'algorithme lui-même, identique dans les deux frameworks.

Lien GitHub du projet : cliquez ici
`https://github.com/HocineYacineBEY/TP-Big-Data/tree/main`