Rapport du projet de Big Data

Exploration des techniques de clustering

Réalisé par : Asmae Latifi

Sous la tutelle de : M. Sergey Kirgizov

Table des matières

Chapitre 1

Introduction

Ce rapport accompagne le code disponible sur le dépôt GitHub, dans le cadre du projet du module Big Data. L'objectif principal de ce projet est d'explorer et d'analyser différentes méthodes de clustering, ainsi que leur implémentation en Python. Ce rapport présente le travail réalisé pour répondre aux deux exercices proposés par M. Sergey Kirgizov.

Le premier exercice consiste à appliquer des méthodes de clustering sur des données générées aléatoirement, en utilisant des distributions **gaussiennes** et **exponentielles**. Le second exercice se concentre sur l'analyse d'un jeu de données réel portant sur la qualité des vins rouges et blancs, avec pour objectif d'identifier des groupes significatifs dans ces données à l'aide des techniques de clustering.

Chapitre 2

L'implémentation K-means

L'objectif de ce projet est de comprendre le fonctionnement des algorithmes de clustering en implémentant l'algorithme classique k-Means ainsi que ses variantes : k-Means++ et mini-batch k-Means. Les différentes méthodes sont testées sur des jeux de données artificielles, avec des dimensions et des distributions variées. Les tâches réalisées dans ce projet incluent :

- Génération et visualisation de données artificielles.
- Implémentation de l'algorithme k-Means classique et de k-Means++.
- Implémentation de mini-batch k-Means pour traiter de grands ensembles de données.
- Analyse comparative des algorithmes.
- Sélection optimale du paramètre k (nombre de clusters) à l'aide des méthodes du coude et de la silhouette.

Le programme offre la possibilité de tester toutes les fonctionnalités en exécutant des commandes simples, telles que :

```
python Kmeans.py visualize --dist_type gaussian --dim 2 --num_points
    10000
python Kmeans.py compare --dist_type gaussian --dim 2 --k 3 --num_points
     10000 --n_runs 10

python Kmeans.py visualize_algos --dist_type gaussian --dim 2 --k 3 --
    num_points 10000

python Kmeans.py minibatch --dist_type gaussian --dim 2 --k 3 --
    num_points 10000 --batch_size 100

python Kmeans.py find_best_k --dist_type gaussian --dim 2 --num_points
    10000 --max_k 10
```

2.1 Génération et Visualisation des Données

```
python Kmeans.py visualize --dist_type gaussian (or exponential) --dim n
     --num_points N
```

Cette commande permet de générer et visualiser des ensembles de données synthétiques. Les paramètres `-dim n` et `-num_points N` déterminent respectivement le nombre de dimensions des données et le nombre total de points à générer.

Génération des Données

Deux types de distributions ont été utilisées pour générer les données :

- **Distribution Gaussienne** : Des points sont générés autour de centres de clusters selon une distribution normale.
- **Distribution Exponentielle** : Les points sont générés à l'aide d'une distribution exponentielle, offrant une répartition différente.

Les données sont créées avec des dimensions variées (1D, 2D, 3D, 4D) et visualisées pour observer la distribution initiale des points.

Visualisation

La visualisation a été effectuée avec des graphiques adaptés à chaque dimension :
— **1D** : Points dispersés le long d'une ligne.
— **2D** : Nuages de points dans un plan.
— **3D** : Visualisation en trois dimensions.
— **4D** : Couleurs ajoutées pour représenter la quatrième dimension.

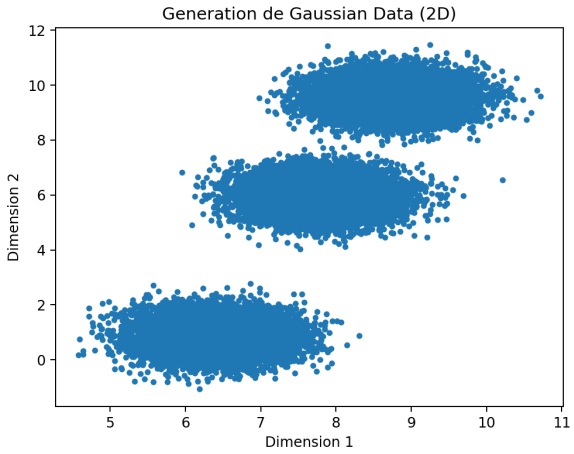

FIGURE 2.1 – Données gaussiennes générées pour une dimensionalité en 2D pour 1000 points.

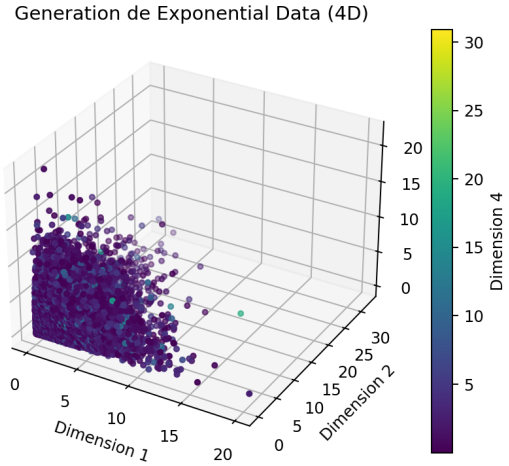

FIGURE 2.2 – Données exponentielles générées pour une dimensionalité en 4D pour 1000 points.

k-Means Classique

L'algorithme classique a été implémenté en itérant entre deux étapes principales :

1. **Assignation des Points** : Chaque point est assigné au cluster le plus proche.
2. **Mise à Jour des Centroides** : Les centroides des clusters sont recalculés en tant que moyenne des points assignés.

Implémentation de k-Means++ et comparaison

L'utilisateur peut utiliser le paramètre `visualize_algos` pour visualiser les résultats de k-Means et k-Means++. Des comparaisons entre k-Means classique et k-Means++ ont été réalisées en mesurant :

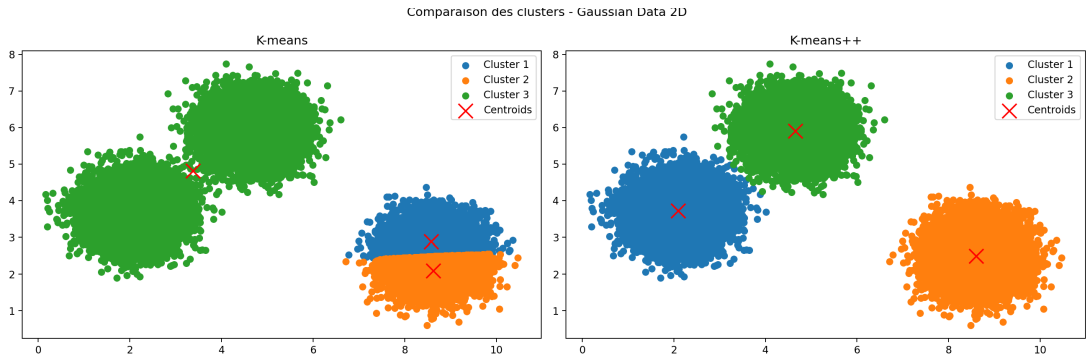

FIGURE 2.3 – Comparaison des résultats de k-Means et k-Means++ pour 100 points générés selon une distribution gaussienne.

— Le temps d'exécution.
— Le nombre d'itérations nécessaires pour converger.
— La qualité de la solution finale (valeur du SSE).

Pour comparer le paramètre `compare` permet de visualiser l'évolution du SSE (Sum of Squared Errors) sur un graphique montrant le SSE en fonction des itérations.

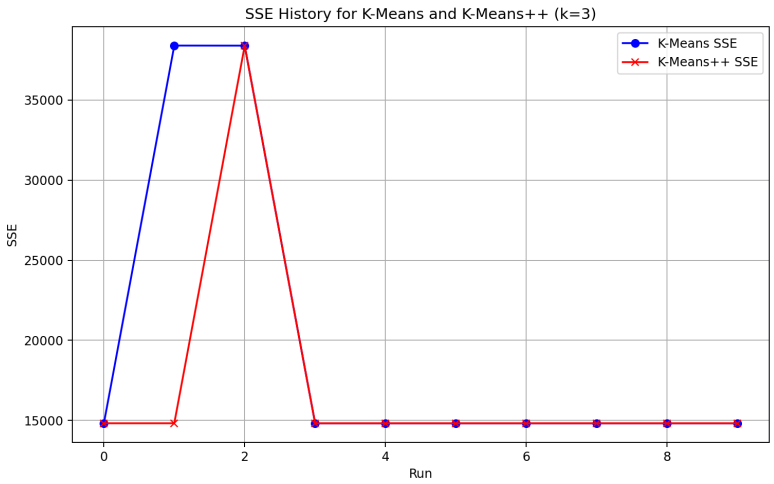

FIGURE 2.4 – Comparaison de l'évolution du SSE pour k-Means et k-Means++.

En plus du graphique, le programme affiche une synthèse de la comparaison directement dans le terminal, mettant en évidence les différences de performances.

```
PS C:\Users\asmae\OneDrive\Desktop\big_data\Projet> python Kmeans.py compare --dist_type gaussian --dim 2 --k 3 --num_points 10000 --n_runs 10

Average Statistics:

kmeans:
Average Time: 0.0822 seconds
Average Iterations: 10.40
Average SSE: 19540.07

kmeans++:
Average Time: 0.7592 seconds
Average Iterations: 12.50
Average SSE: 17181.26
```

FIGURE 2.5 – Synthèse des résultats comparatifs des deux algorithmes affichée dans le terminal.

k-Means++ améliore l'initialisation des centroides en veillant à ce qu'ils soient initialement bien

dispersés. Cette méthode réduit les risques de solutions sous-optimales et accélère la convergence vers un résultat de meilleure qualité.

Mini-Batch k-Means

L'implémentation de cette méthode ajoute une étape où les points sont traités par lot (batch), avant de revenir à la logique classique de k-Means pour mettre à jour les centroïdes.

Pour évaluer les performances du Mini-Batch k-Means, nous avons choisi de visualiser l'évolution du SSE au fil des itérations, ce qui permet de tirer des conclusions sur le comportement de cet algorithme.

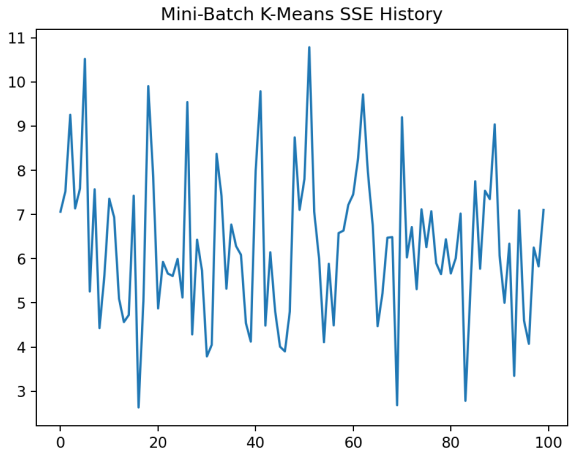

FIGURE 2.6 – Évolution du SSE pour Mini-Batch k-Means avec 1 000 points générés selon une distribution gaussienne.

Les fluctuations observées dans l'évolution du SSE sont dues à la nature stochastique de la méthode, où les centroïdes sont mis à jour à partir d'un sous-ensemble des données à chaque itération. Cela peut entraîner une convergence moins lisse comparée au k-Means classique, mais offre un compromis avantageux entre précision et temps de calcul pour de très grands ensembles de données.

Sélection du Meilleur Nombre de Clusters

Méthode du Coude

L'évolution du SSE (Sum of Squared Errors) a été tracée pour différentes valeurs de k. Le point d'inflexion (ou *coude*) indique le nombre optimal de clusters. Il suffit alors de réutiliser la méthode K-Means avec k variable.

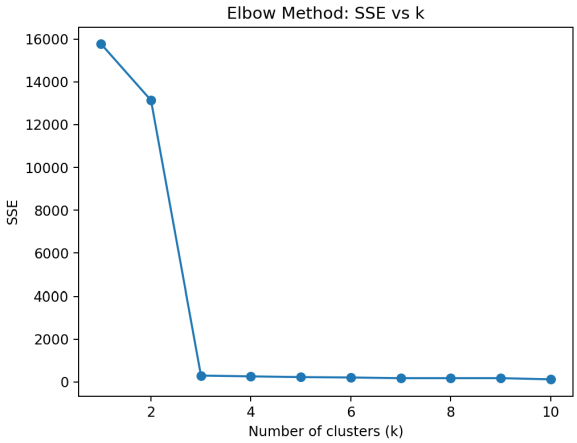

FIGURE 2.7 – Méthode du Coude pour des données à 2 dimensions, 200 points générés selon une distribution gaussienne.

Dans cet exemple, on observe que le meilleur k selon la méthode du coude est 3.

Méthode de la Silhouette

Le score de silhouette a été calculé pour évaluer à la fois la cohésion interne et la séparation entre les clusters. Le k correspondant au score le plus élevé est considéré comme optimal. Nous avons utilisé la méthode implémentée `silhouette_score` de Python, ce qui nous a permis d'obtenir les résultats suivants.

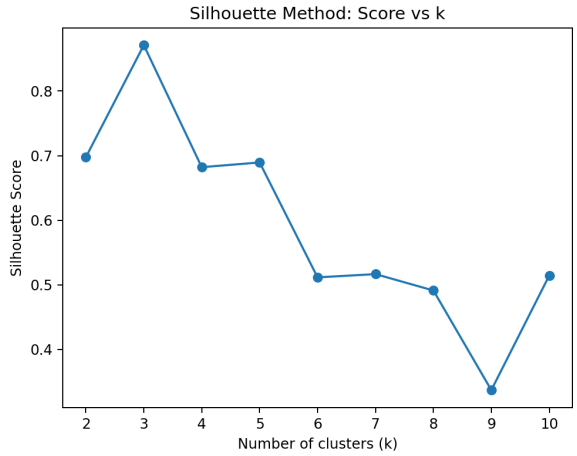

FIGURE 2.8 – Méthode de la Silhouette pour des données à 2 dimensions, 200 points générés selon une distribution gaussienne.

On constate ici de manière plus claire que le meilleur k est également 2. Ces résultats ont été obtenus à l'aide du paramètre `find_best_k`, qui retourne les résultats des deux méthodes.

Commandes Utilisées

Voici un récapitulatif des commandes utilisées dans le programme pour visualiser et exécuter les différentes méthodes de clustering.

1. **Génération et Visualisation des Données** :

```
python kmeans_project.py visualize --dim 2 --dist_type gaussian --
    num_points 200
```

2. **Comparaison des Algorithmes** :

```
python kmeans_project.py compare --dim 3 --dist_type exponential --
    k 4 --n_runs 5
```

3. **Mini-Batch k-Means** :

```
python kmeans_project.py minibatch --dim 4 --dist_type gaussian --k
     5 --batch_size 200
```

4. **Sélection du Meilleur** k :

```
python kmeans_project.py find_best_k --dim 2 --dist_type gaussian
    --max_k 10
```

Clustering des Données de Qualité de Vin

Le but de cette analyse est de segmenter les vins rouges et blancs selon plusieurs caractéristiques physico-chimiques en utilisant des méthodes de clustering. Ces techniques permettent d'identifier des groupes naturels au sein des données, ce qui peut être utile pour comprendre la distribution des caractéristiques des vins et potentiellement prédire la qualité des vins en fonction de ces groupes.

Prétraitement des Données

Les premières étapes du prétraitement ont consisté à lire les fichiers CSV contenant les informations sur les vins rouges et blancs. Après cela, les valeurs manquantes ont été vérifiées, et aucune valeur manquante n'a été trouvée dans les deux jeux de données.

Ensuite, les colonnes de caractéristiques des vins (toutes sauf la colonne 'quality') ont été extraites et normalisées à l'aide de la méthode StandardScaler de Scikit-learn. Cette normalisation a été effectuée afin de mettre toutes les variables sur la même échelle et éviter que certaines caractéristiques n'influencent trop fortement le résultat des analyses de clustering.

Les données normalisées ont été sauvegardées sous forme de nouveaux fichiers CSV.

Méthodes de Clustering

Deux méthodes principales ont été utilisées pour effectuer le clustering : la méthode du coude et le score de silhouette.

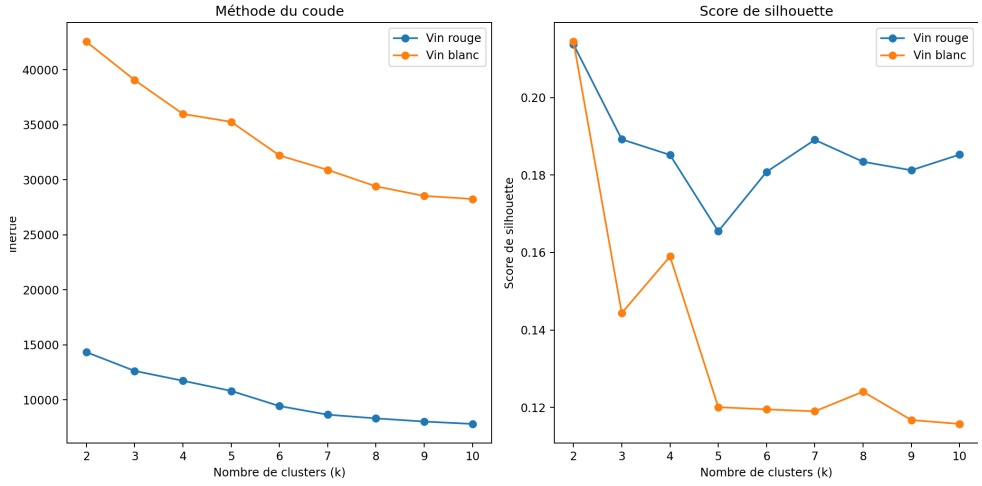

FIGURE 2.9 – Méthode du coude et score de silhouette pour déterminer le meilleur k pour les vins rouges et blancs.

Méthode du Coude

Pour les vins rouges, la méthode du coude a montré que $k = 3$ à $k = 4$ était le nombre optimal de clusters. Pour les vins blancs, le coude se trouvait entre $k = 4$ et $k = 5$.

Score de Silhouette

Pour les vins rouges, la méthode de la silhouette a suggéré que les meilleurs choix pour k étaient 2 ou 4. Pour les vins blancs, les meilleurs choix étaient 2 ou 3.

Résultats du Clustering

Après avoir appliqué les méthodes de clustering, les résultats suivants ont été obtenus :

Vin Rouge :

Le meilleur k était déterminé comme étant $k = 4$, avec une certaine incertitude entre $k = 3$ et $k = 4$ selon la méthode du coude.

Description des clusters pour les vins rouges :

— Cluster 0 : Caractéristiques avec une faible acidité fixe et volatile, et une faible teneur en alcool.
— Cluster 1 : Forte teneur en acide citrique, faibles niveaux de dioxyde de soufre libre et total.
— Cluster 2 : Bon équilibre des caractéristiques avec des niveaux modérés d'acidité et d'alcool.

Vin Blanc :

Le meilleur k était $k = 3$, avec des résultats similaires entre $k = 4$ et $k = 5$ selon la méthode du coude.

Description des clusters pour les vins blancs :

— Cluster 0 : Faible teneur en acide citrique, sucre résiduel élevé, faible densité.
— Cluster 1 : Forte teneur en acide sulfurique libre et total, teneur modérée en alcool.
— Cluster 2 : Forte concentration en chlorures, faible teneur en alcool.
— Cluster 3 : Faibles niveaux de dioxyde de soufre, mais une forte teneur en alcool.

Conclusion

En conclusion, cette étude a permis d'explorer en profondeur l'application des algorithmes de clustering k-Means, k-Means++ et mini-batch k-Means sur différents jeux de données. Les résultats obtenus montrent que k-Means++ améliore la qualité des clusters en minimisant le risque de convergence vers un minimum local, tandis que mini-batch k-Means offre une alternative efficace pour les ensembles de données volumineux en réduisant le temps de calcul. L'application des algorithmes sur des données réelles, telles que le dataset des vins, a permis de mettre en évidence l'importance du choix de la méthode de clustering en fonction des spécificités du jeu de données. En somme, cette étude souligne l'importance de bien comprendre les caractéristiques des algorithmes de clustering pour choisir la méthode la plus adaptée à chaque contexte d'analyse.