Large Scale Machine Learning: Projet

Ce projet consiste dans l'implémentation en Spark d'un algorithme de clustering de type *K-means*, c'est-à-dire segmenter en k groupe distinct un échantillon à l'aide de méthodes quantitatives. La méthode sera implémentée à l'aide de Spark. On effectuera une comparaison entre les deux langages Python et Scala à travers cet outil.

Les deux jeux de données choisi pour cette expérimentation sont :

- *IRIS* : 150 fleurs, différentiables quantitativement selon 4 attributs, répartis selon 3 familles. Jeu de données largement utilisés dans le milieu universitaire pour la classification.
- *Dry Beans* : 13611 haricots secs, différentiables quantitativement selon 16 attributs, répartis selon 7 familles. Jeu de données provenant de la plateforme Kaggle[1], on choisira ce jeu de par son large volume de données ainsi que ses nombreux attributs à caractère uniquement numérique. On pourra ainsi tester la mise à échelle (scaling) de notre solution dans une moindre mesure.

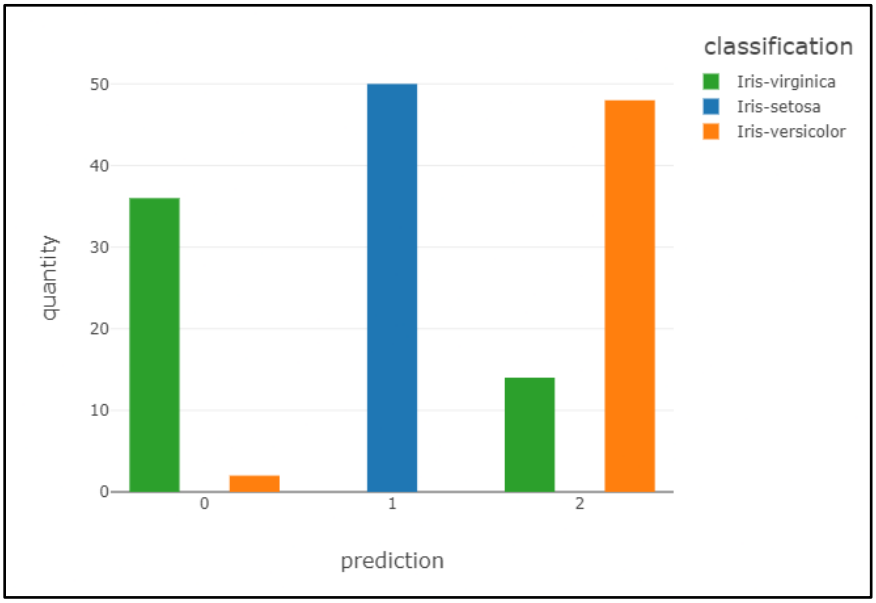


Fig.1 : Première classification IRIS

Après une première implémentation basique sans optimisation, on obtient les résultats ci-dessus pour IRIS. Dans l'ensemble la classification est correcte, seulement les iris Viriginica ont tendance à se confondre avec les iris Versicolor, donnant un cluster 2 avec un peu plus de 75% de Versicolor et presque un quart de de Virginica. Les iris Setosa sont à l'opposé parfaitement identifiés par le modèle.

[1] https://www.kaggle.com/datasets/muratkokludataset/dry-bean-dataset

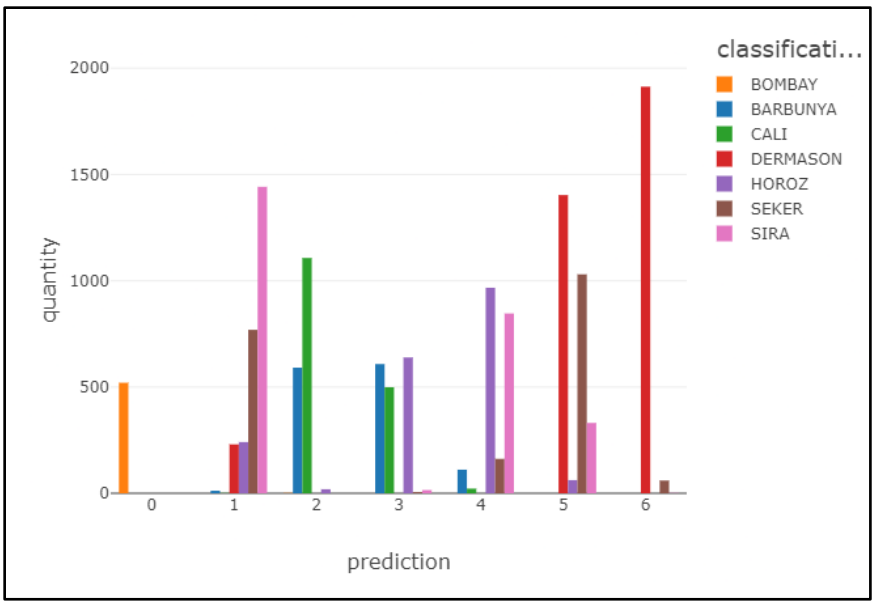
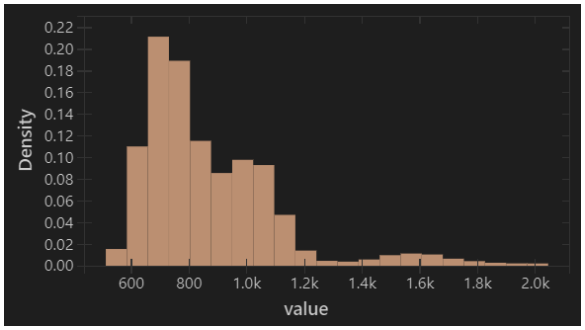


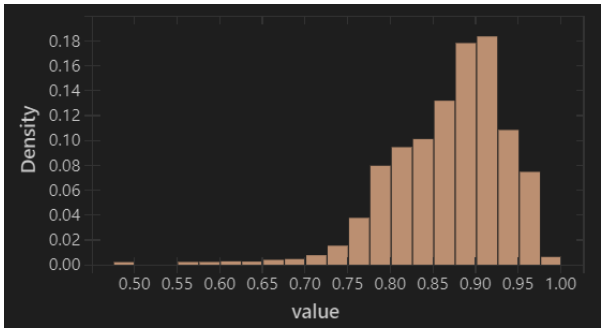
Fig. 2 : Première classification BEANS

Pour ce qui est de BEANS, le modèle est drastiquement plus défaillant. Plusieurs classes s'accaparent des clusters (BOMBAY, DERMASON) et malgré la domination de presque tous les types d'haricots sur au moins un cluster (à l'exception de BARUNYA et SEKER), on identifie des cluster fortement mélangé (1, 3, 5).

Il pourrait s'avérer utile de normaliser les données. En effet, les valeurs prises par les différents attributs sont très différentes, donnant plus de poids à certaines caractéristiques des haricots et déséquilibrant ainsi la classification.



Paramètre « perimeter »



Paramêtre « solidity »

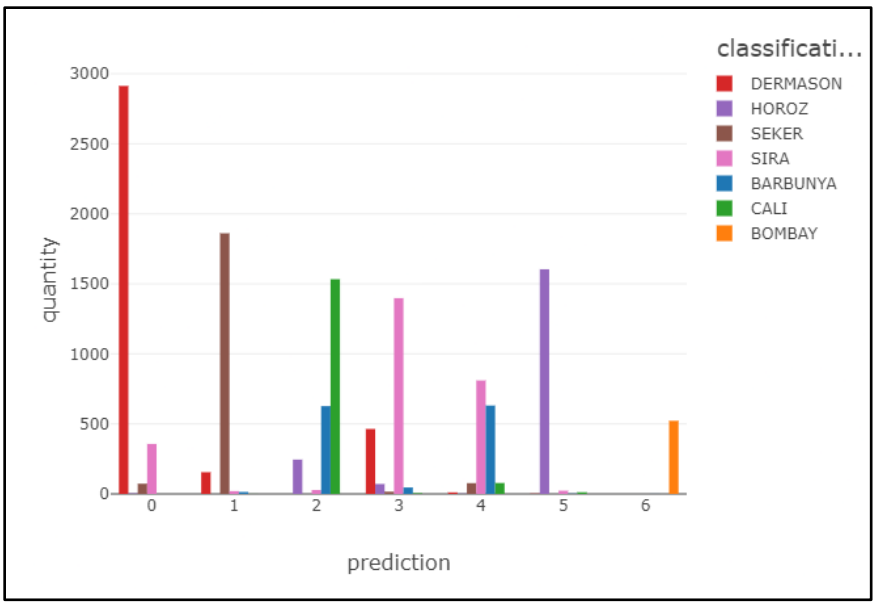


Fig. 3 : Deuxième classification BEANS

On remarque à travers le graphique ci-dessus une amélioration globale des résultats. En effet, les haricots de types DERMASON, surreprésentés dans l'échantillon, ont cette fois ci un cluster bien déterminé et n'envahissent pas les clusters restants. Seul petit défaut de la classification est le manque d'identification d'un cluster BARBUNYA, peut-être dû aux haricots SIRA qui s'imposent comme majoritaire dans 2 clusters différents. Cela peut aussi être dû à un mauvais tirage des centroïdes de départ, provoquant un déséquilibre. La figure 3 a en effet été obtenu après plusieurs tirages, celui-ci affichant les meilleurs résultats (à quelques détails près).

En appliquant la normalisation à IRIS, les résultats sont un peu meilleurs mais le problème de la première classification persiste.

Les mauvaises performances de notre modèle étant potentiellement dû aux mauvais choix initiaux des centroïdes, on implémentera une version permettant le choix de centroïdes écartés, permettant d'obtenir des clusters finaux plus démarqués. Cette méthode est connue dans le monde universitaire sous le nom de *K-means++*[2].

[2] http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

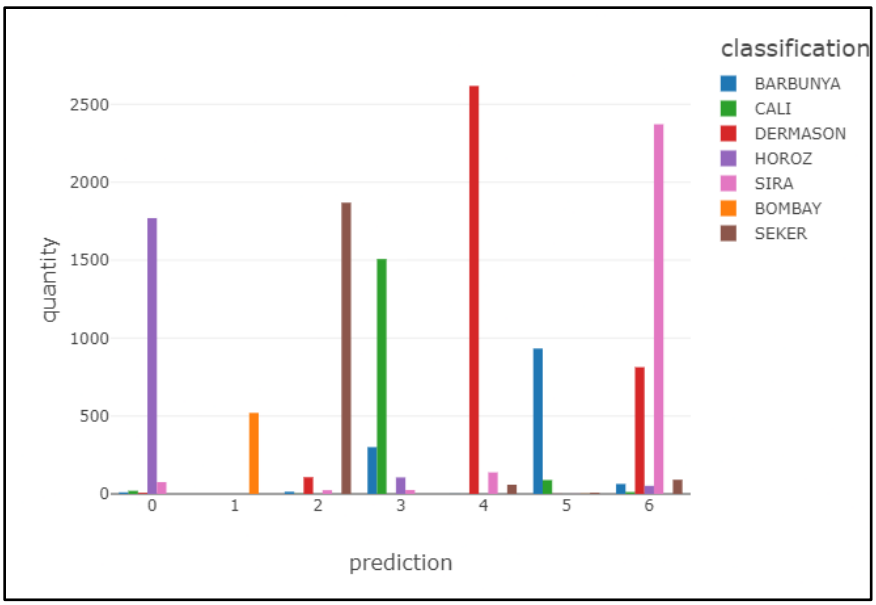


Fig. 4 : Troisième classification classification BEANS, *k-means++*

Cette solution porte ses fruits pour BEANS, et permet à chaque type d'haricots de dominés un cluster. Elle permet de plus de limiter l'utilisation de l'aléatoire dans notre modèle malgré être toujours présent.

Cette solution sera de même appliquer à IRIS, malheureusement sans grande amélioration.

Voici un tableau récapitulatif des résultats des modèles exposés précédemment :

Implémentation		**Scala**		**Python**	
BEANS	Naïve	7.328''	0.504	20.545''	0.13
	Naïve, normalisée	7.653''	1.140	4.173''	0.332
	K-means ++	11.288''	1.117	4.933''	0.222
IRIS	Naïve	1.773''	0.695	4.769''	0.3
	Naïve, normalisée	1.825''	0.782	3.556''	0.385
	K-means ++	1.865''	0.774	4.549''	0.321

Les temps d'exécution sont donnés en secondes, et les scores affichés représentes l'indice de Davies-Bouldin[3]. Il représente la moyenne du rapport maximal entre la distance d'un point au centre de son groupe et la distance entre deux centres de groupes. L'indice varie de 0 (classification resserrée, meilleur classification) à $+\infty$ (classification éparse, pire classification). A noter que les modèles itèrent jusqu'à un maximum de 40 itérations. Les modèles s'arrêtent prématurément lorsque $J(\mu, z)$ n'évoluent pas de plus de 1% par itération.

On remarque que les solutions « optimisés » n'affichent pas forcément de meilleurs résultats, particulièrement en Scala. Les temps d'exécutions d'allongent et les classifications ne gagne pas forcément en cohérence. Cela peut être dû à un chargement des données sur le driver afin d'effectuer le

[3] https://fr.wikipedia.org/wiki/Indice_de_Davies-Bouldin

tirage lors de l'utilisation du modèle k-means++ (Spark ne propose pas de tirage aléatoire pondéré malheureusement nécessaire pour cette solution). La normalisation qui paraissait être une solution à coût en temps quasi-nul, n'apporte pas non plus d'améliorations selon l'indice de Davies-Bouldin malgré que les visualisations par diagrammes en bar semblent plus cohérente avec. Pour les jeux de données volumineux, Python semble être mieux optimisés (malgré les résultats étranges du modèle naïf sur *BEANS*).

Le Coefficient de Silhouette[4] fut implémenté et aurait été préférable à Davies-Bouldin, malheureusement cet indice ne convient pas du tout pour de grands jeux de données (complexité $o(n^2)$, très mauvais « scaling ») rendant les temps d'exécutions très longs…

Notebooks Databricks:

- ***Scala*** **:** https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1911696510408997/874084256956327/3730284634398388/latest.html

- ***Python*** **:** https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1911696510408997/1557586048860159/3730284634398388/latest.html

[4] https://en.wikipedia.org/wiki/Silhouette_(clustering)