Université de Paris

BIG DATA

Parcours : Machine Learning for Data science

Projet de big data avec Apache Spark

Mohamed GHARBI
Nassim ELKEFIF

Professeur : M. Morbieu STANISLAS

Année universitaire : 2020/2021

Table des matières

Table des figures

Introduction

De nos jours, avec l'augmentation du flux de l'information et leur traitement differt en fonction des besoins exprimés et de leurs types " numériques, textuelles et multimédia". En effet ce volume de données non structurés est un problème majeur pour les entreprises afin de les exploiter au temps réel.
Dans ce projet nous allons voir un autre aspect de traitement et exploitation de données qui est une solution alternative pour le traitement de doonnées volumineuses et non structurées ce que nous appellons "le big data".
Ce présent rapport sera divisé comme suit :

- Chapitre 1 " état de l'art" : Dans ce chapitre nous allons aborder les notions théoriques du projet.
- chapitre 2 " Réalisation" : Ce chapitre fera sujet d'une études et d'interprétation des résultats obtenues.

En finalité, une conclusion générale englobant ce que nous avons fait et une comparaison des résultats obtenus.

Chapitre 1

Etat de l'art

1.1 Introduction

L'implication de l'informatique et plus précisément de la science des données et de l'intelligence artificielle dans le traitement des données est devenue essentielle et incontournable pour avoir de meilleures prédictions dans certains domaines et aussi des décisions précises, mais nous trouvons d'autres alternatives afin de traiter les données volumineuses et non structurées en un temps records. Dans ce chapitre nous allons parler des différentes méthodes qui sont utilisées pour traiter ce type de données en soulignant :

— Apach spark et ses différents pôles.

— Kmeans.

1.2 Apach Spark

[1] Apache Spark est une infrastructure de traitement parallèle open source qui permet d'exécuter des applications analytiques d'envergure, en utilisant des machines en clusters.
Apache Spark peut traiter des données issues de différents référentiels de données, dont Hadoop Distributed File System (HDFS), des bases de données NoSQL et des data stores de données relationnels, tels qu'Apache Hive. En outre, Spark prend en charge le traitement In-memory, dopant ainsi les performances des applications d'analytique Big Data. Toutefois, il permet aussi un traitement conventionnel sur disque lorsque les ensembles de données sont trop volumineux pour la mémoire système disponible.

1.3 PySpark

[2] PySpark est une interface pour Apache Spark en Python. Elle vous permet non seulement d'écrire des applications Spark à l'aide d'API Python, mais fournit également le shell PySpark pour analyser interactivement vos données dans un environnement distribué. PySpark supporte la plupart des fonctionnalités de Spark telles que Spark SQL, DataFrame, Streaming, MLlib (Machine Learning) et Spark Core.

1.3.1 Spark SQL

[3] Spark SQL est un module Spark pour le traitement des données structurées. Contrairement à l'API Spark RDD, les interfaces fournies par Spark SQL donnent à Spark plus d'informations sur la structure à la fois des données et du calcul effectué. En interne, Spark SQL utilise ces informations supplémentaires pour effectuer des optimisations supplémentaires. Il existe plusieurs façons d'interagir avec Spark SQL, y compris SQL et l'API Dataset. Lors du calcul d'un résultat, le même moteur d'exécution est utilisé, indépendamment de l'API/du langage que vous utilisez pour exprimer le calcul. Cette unification signifie que les développeurs peuvent facilement passer d'une API à l'autre en fonction de celle qui offre le moyen le plus naturel d'exprimer une transformation donnée.

1.3.2 MLlib

[3] Construite au-dessus de Spark, MLlib est une bibliothèque évolutive d'apprentissage automatique qui fournit un ensemble uniforme d'API de haut niveau permettant aux utilisateurs de créer et de régler des pipelines d'apprentissage automatique pratiques.

1.4 Spark Core

Spark Core est le moteur d'exécution général sous-jacent de la plateforme Spark, sur lequel toutes les autres fonctionnalités sont construites. Il fournit un RDD (Resilient Distributed Dataset) et des capacités de calcul en mémoire.

1.4.1 RDD

[3] Un RDD est la représentation Spark d'un tableau de données. C'est une collection d'éléments que l'on peut utiliser pour contenir des tuples, des dictionnaires, des listes…
La force d'un RDD réside dans sa capacité à évaluer le code de façon paresseuse : le lancement des calculs est reporté jusqu'à ce que ce soit absolument nécessaire.

1.5 KMeans

[5] K-means est un algorithme non supervisé de clustering non hiérarchique. Il permet de regrouper en K clusters distincts les observations du dataset. Ainsi les données similaires se retrouvent dans un même cluster. Par ailleurs, une observation ne peut se retrouver que dans un cluster à la fois (exclusivité d'appartenance). Une même observation, ne pourra donc, appartenir à deux clusters différents.

Chapitre 2

Réalisation et interprétation

2.1 Introduction

Après avoir terminé l'état de l'art, nous entamons la partie réalisation et mise en œuvre de la solution. Ce chapitre présente les étapes de réalisation de notre solution, qui sera divisé en trois grandes parties à savoir :

- mise en place de l'environnement et la génération des données.
- analyse descriptive et clustering
- Implémentation de la méthode k-Means

2.2 Mise en place de l'environnement et la génération des données

Dans cette partie nous avons conçu un environnent spark où nous avons créé un objet sparkContext et spark session. Comme deuxième étape, nous avons généré deux jeux de données tel que :

- Le premier : Contenant 1000 points avec deux dimensions "var1 et var2" et trois classes.
- Le deuxième : Contenant 10000 points avec quatre dimensions "var1, var2, var3, var4" et trois classes.

Ces deux jeux de données sont de 3 types à savoir :

1. Dataframe
2. Spark Dataframe
3. RDD

Var1	Var2	label
0.46661789679484156	3.8657130258372034	0
2.8438280686964625	3.326509447571979	0
0.6112148620002373	2.512459779407206	0
3.816533648852741	1.65175932359417 97	1
1.2809724409460734	0.6282738814866966	1
0.9885020621853124	5.8730469419842946	0
1.0509448092182645	-0.10052806829392724	1
0.9146436764508477	4.1964820511572 15	0
0.8778175541210628	3.6403090410862826	0
0.9133661022290314	-0.4133067105032755	1
0.8903393115748823	4.498080265494162	0
3.0943639618112706	1.3767588840361653	1
1.2824956384546757	1.7797202593009476	1
2.6145665028237 62	-0.16603193108761105	1
-0.9694330046174637	3.391003833313098	0
-0.07228288652059756	2.883769390269415	0
4.506497240507954	0.6864576763064169	1
0.5070131985838355	1.4296382989451282	1
3.1725728369695423	0.37876345551308477	1
1.7287065437536702	0.8894537776683228	1

only showing top 20 rows

var1	var2	var3	var4	y
0.6581214348347171	3.330950208806035	-3.1915145399441256	8.996709086804 36	1
2.201357680941256	2.638305471651551	2.8011225663376274	1.0743820468193288	0
2.1141737865997605	3.8081873285890175	2.728857401172996	1.013392596608 99	0
0.8502711116492576	3.9081446043865014	2.6283803113546913	0.15629458427565868	0
-2.341107973378 13	4.4072873906336385	-0.5634404916797624	7.167195851675335	1
1.4927888618190013	4.465759181979671	3.9695437384544654	1.5828726378719	0
9.208569818907 92	-2.224851678200011	5.055311128007941	1.8971986484032808	2
0.5292787812679975	6.373262719119204	1.1158432028169358	-0.15446050710021608	0
-0.11940115042601729	3.210890721122 36	1.7242569818031132	1.347952490585281	0
1.3305155097988 81	4.802400627468 65	1.8120560760650128	0.7604026811248425	0
1.2064981592308 39	4.452608084930084	5.3479618434 8843	0.3952889478120635	0
11.7742886050538 78	-4.470591435176094	3.9266299474919117	1.9191333559237291	2
-0.3560253883 0832575	6.194191030530046	1.7449566718536613	1.002418704200167	0
10.644347902420199	-2.021768871507 2826	6.915501750201172	1.3192504946577297	2
-1.211583082281884	2.785090916638 8282	-3.178903825181668	8.124302956375024	1
9.330496590620855	-2.698465188097312	5.908792490915004	0.025482817525757095	2
-0.7928879812659034	1.945889484330 7906	-0.7603787480972444	8.678115716991327	1
-2.368324869840016 3	2.030401671801 7373	-0.4397158471769717 6	7.062445011625135	1
-0.2698951270579253	4.651105467271 0745	1.4585837772802637	8.368985480256272	1
-2.144138084166 05	3.062718461202 7367	-1.7316616879804039	8.976900327285703	1

only showing top 20 rows

FIGURE 2.1 – les deux jeux de données

2.3 Analyse descriptive

Dans cette partie, nous allons faire l'étude en se basant sur trois points :

- Vérification du nombre de classes.
- statistique de chaque variable.
- Variance de chaque variable.

2.3.1 Vérification du nombre de classes

D'après la figure ci-dessous, nous trouvons que le nombre de classes est trois et leurs distribution est uniforme.

y	count
0	3334
1	3333
2	3333

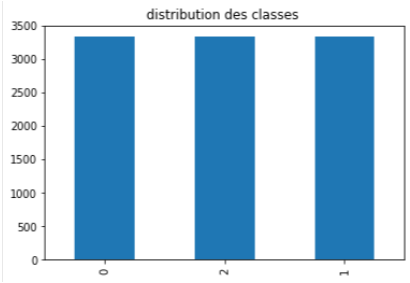

FIGURE 2.2 – nombre de classes

2.3.2 Statistique de chaque variable

L'utilisation de la fonction describe sous python nous a donné les résultats de chaque variable en sepécifiant la moyenne, l'écart-type, le min, le max et le mod. La figure suivante résume ceci.

summary	var1	var2	var3	var4	y
count	10000	10000	10000	10000	10000
mean	2.90304289682991 93	1.6262767143171584	2.2020675130790908	3.1073537965856106	0.9999
stddev	4.7125802344158	3.0316801434818212	3.073757112403275	3.5028826880181483	0.8165578158750719
min	-5.9735362545812	-5.9744639393971841	-4.519159445017838	-3.003147473741441	0
max	13.077099249044569	7.683327455400309	9.34372381743226	11.471561715895099	2

FIGURE 2.3 – Discriptif des variables

En voulant pousser l'analyse plus loins, nous avons étudié la corrélations des variables, tel que la figure *"FIGURE 2.4"* montre que les variables ne sont pas corrélées entre elles. De ce fait, nous constatons que chaque variable apporte de l'information pour notre étude " toute les variables sont crutiales" et que nous n'avons pas de redondance de l'information.

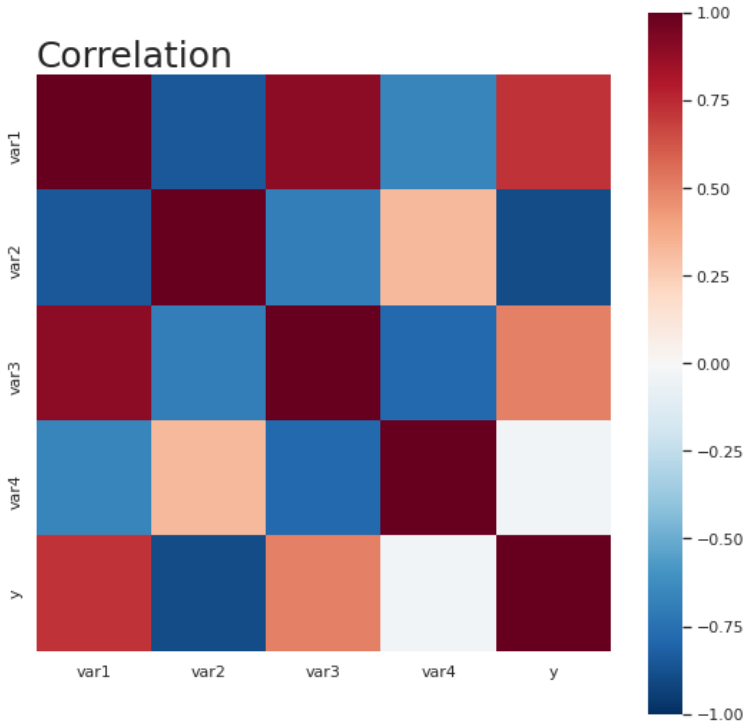

FIGURE 2.4 – Corrélation entre les variables

2.3.3 Variance de chaque variable

y	var1	var2	var3	var4
0	0.989301	0.972445	0.993508	0.980659
1	0.967669	0.973861	1.012933	0.957460
2	1.006423	1.008627	1.026715	1.030712

FIGURE 2.5 – Variance de chaque variable

D'après la figure ci-dessus, nous constatons que les différentes variances sont tous proches de 1, ce qui signifie que les données sont dispersées.

2.4 Clustering avec des implémentations disponibles

Dans Cette partie, nous allons utilisé la méthode k-means existante dans les deux packages de sklearn et pyspark " MLlib" en faisant une comparaison basé sur le temps de traitement et les résultats obtenus de l'application de la metrique ***"NMI"***.

2.4.1 Comparaison du temps d'execution

D'après la figure ci-dessous, nous constatons que la comparaison du temps d'éxecution se base sur deux critères :

- Cpu times.
- Wall time.

D'après le wall time des différentes méthodes nous constatons que scikit-learn a fournis de bons résultats, néanmoins d'après la définition du wall time "le temps écoulé, y compris le temps passé à attendre son tour sur le CPU", de ce fait nous ne pouvons pas en conclure qu'avec ce temps.
En allant plus loins et en comparant le CPU times, nous trouvons que la méthode pyspark avec la RDD donne de meilleurs résultats que les des autres et que le scikit-learn est le plus faible.

```
CPU times: user 78.1 ms, sys: 61.3 ms, total: 139 ms
Wall time: 85.5 ms
```
Scikit-learn

```
CPU times: user 57.2 ms, sys: 16.9 ms, total: 74 ms
Wall time: 5.62 s
```
Spark "dataframe"

```
CPU times: user 26.6 ms, sys: 2 ms, total: 28.6 ms
Wall time: 1.73 s
```
Spark " RDD"

FIGURE 2.6 – Temps d'execution

2.4.2 Comparaison du NMI

Après avoir executé Le k-means, nous avons utilisé la métrique NMI afin de comparer les performance de chaque méthode, de ce fait nous constatons que les deux méthodes ont données le même résultat.
Nous pouvons conclure notre comparaison, tel que la méthode "pyspark" minimise le temps d'execution.

2.5 Implémentation du K-Means

Dans cette partie nous allons implémenter le k-means et analyser les résultats obtenus sur les deux cas à savoir :

- Le cas unidimensionnel
- Le cas multidimensionnel

2.5.1 Le cas unidimensionnel

Dans le cas unidimensionnel, nous remarquons que le plus on a d'observations le mieux est le NMI score et ceci peu importe le nombre de clusters. Par contre si nous gardons le même nombre d'observations et nous cherchons à trouver un nombre de cluster optimale, nous devons travailler avec un plus grand nombre de clusters possible.
Dans un premier cas Nous avons : NMI score pour 10000 d'observation et 5 cluster : 42%, par contre le NMI score pour 10000 d'observation et 3 cluster : 39%.
dans un deuxieme cas nous avons : NMI score pour 500 d'observation et 3 cluster : 38%, par contre le NMI score pour 500 d'observation et 2 cluster : 67%. Nous concluons, pour les données unidimensionnel et pour la cas de nombre d'observations relativement grand : Un grand nombre de cluster est plus BON.
Pour le cas de nombre d'observation relativement non important, un nombre de cluster moyen est plus BON.

2.5.2 Le cas Multidimensionnel

Dans cette partie, on s'est basé sur la première implimentation du Kmeans pour pouvoir implimenter le Kmeans multidimensionnel, le même principe a été utilisé, avec quelque modifications sur les dimensions.

- La fonction calculate_sum calcule la somme des dimensions afin de pouvoir calculer la variable sum_by_cluster.
- La fonction squared_distances_multiple est responsable de retourner les distance entre le point et chaque cluster.

Pour ce cas, Nous remarquons que nous avons exactement le même résultat que le Kmeans de Sickit-Learn.

Conclusion

Le travail présenté dans ce rapport concerne l'étude et l'interprétation des résultats de l'application des méthodes de machine Learning spécialement les méthodes non supervisée sur des données aléatoires en utilisant deux méthodes d'études à savoir :

- Scikit-learn.
- Pyspark.

References

[1] https://spark.apache.org/docs/latest/api/python/

[2] https://spark.apache.org/sql/

[3] https://datascientest.com/pyspark

[4] ACP ou Analyse en Composantes Principales - ANTEL

[5] Tout ce que vous voulez savoir sur l'algorithme K-Means (mrmint.fr)