SCHEMITH JEREMIE

ML FOR BIG DATA

2.1 Faire la synthèse de l'article :

L'article "MR-Tree - Un algorithme extensible MapReduce pour la construction d'arbres de décision" propose un algorithme parallèle d'apprentissage d'arbres de décision, appelé MR-Tree, exprimé dans le modèle de programmation MapReduce. L'objectif de l'algorithme est de construire efficacement des arbres de décision pour de très grands ensembles de données en utilisant la plateforme Apache Hadoop. L'algorithme exploite le modèle MapReduce pour atteindre une extensibilité en fonction de la taille de l'ensemble de données.

Introduction et Contexte :

L'apprentissage des arbres de décision avec de grands ensembles de données est intensif en termes de calcul et peu pratique sur des ordinateurs à nœud unique.

Apache Hadoop, une plateforme de calcul distribuée, est présentée comme solution pour traiter les big data en utilisant des clusters de matériel standard.

L'article propose MR-Tree, un algorithme parallèle d'apprentissage d'arbres de décision, utilisant le modèle de programmation MapReduce sur la plateforme Hadoop pour une extensibilité en fonction de la taille de l'ensemble de données.

Algorithme MR-Tree :

MR-Tree est une version améliorée de l'algorithme ID3 capable de gérer à la fois des attributs discrets et continus.

L'algorithme comporte trois principales sections : le contrôleur, la fonction d'induction d'arbre (ID3) et la fonction d'élagage.

Le paradigme MapReduce est utilisé pour traiter les données et construire efficacement des arbres de décision en parallèle.

Résultats expérimentaux :

L'extensibilité de l'algorithme est testée en utilisant des ensembles de données de tailles différentes, et les résultats expérimentaux démontrent une extensibilité linéaire en fonction de la taille de l'ensemble de données.

Les temps d'exécution sont enregistrés pour différentes tailles d'ensemble de données, montrant une augmentation constante du temps d'exécution en fonction de la taille de l'ensemble de données, ce qui indique l'extensibilité de l'algorithme.

Conclusion :

MR-Tree est présenté comme un algorithme extensible d'apprentissage d'arbres de décision adapté aux grands ensembles de données.

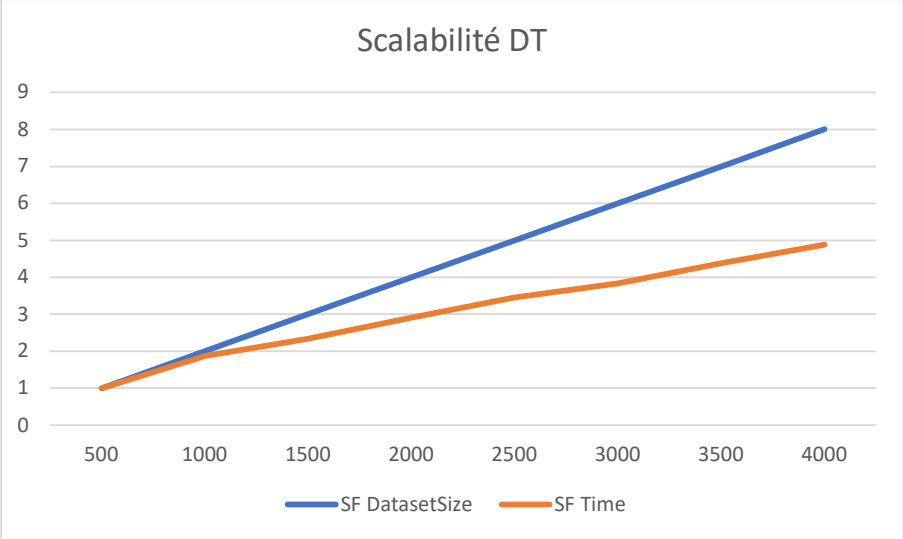
L'article met en avant l'extensibilité de l'algorithme, le rendant adapté au traitement efficace des big data en utilisant la plateforme Apache Hadoop.

2.2.1 Élaborer différents scénarios de surcharge selon le protocole d'expérimentation décrit dans l'article, en s'appuyant sur les deux configurations suivantes :

On execute Spark on mode local avec 8 coeurs. Du fait des limites de notre machine local on réduit Scale Factor.

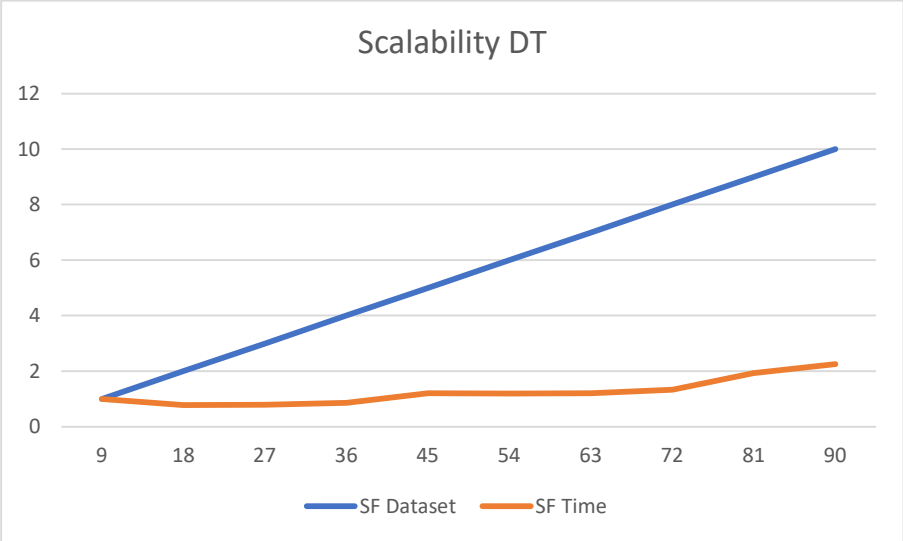
Mode LOCAL :

Dataset Size	SF = Dataset Size	Exécution time	SF = Exécution Timer
(ko)	1 = 500 ko	(secondes)	1 = 6.01
500	1	6.01	1
1000	2	11.26	1.87
1500	3	14.11	2.34
2000	4	17.53	2.91
2500	5	20.75	3.45
3000	6	23.02	3.83
3500	7	26.34	4.38
4000	8	29.36	4.88



Mode CLUSTER :

Dataset Size	SF = Dataset Size	Exécution time	SF = Exécution Timer
(Mo)	1 = 9 Mo	(secondes)	1 = ?
9	1	6.18	1
18	2	4.85	0.78
27	3	4.99	0.80
36	4	5.35	0.86
45	5	7.52	1.21
54	6	7.45	1.20
63	7	7.53	1.21
72	8	8.32	1.34
81	9	12	1.94
90	10	14	2.26



2.2.2 Est-ce-que vous avez obtenu les mêmes tendances que l'article ? Justifiez votre réponse

Pour le **Mode LOCAL** nous n'avons pas exactement les mêmes tendances car si on continuait l'expérience en augmentant le Dataset, nous retrouverons alors la tendance de l'article. Cependant notre machine n'étant pas assez puissante les opérations prennent beaucoup de temps.

Pour le **Mode CLUSTER** la tendance est plus similaire à celle de l'article car on a pris un échantillon de données plus grand pour la taille du Dataset.

2.3.1 Utiliser le mode cluster pour mesurez le temps d'entrainement de l'algorithme MR-ID3, selon différentes configurations en nombre de nœuds (2, 4, 6, 8 et 10) avec un même volume de données pour toutes les configurations. (Fait en mode **LOCAL**)

Pour cette question nous allons utiliser notre machine LOCAL qui est dotée de huit cœurs au maximum. Nous avons utilisé la Dataset kddcup99 pour cette question

Nombre de cœurs	Temps d'entrainement (secondes)	Accuracy
2	579	0.997
4	578	0.998
6	561	0.997
8	536	0.998

2.3.2 Est-ce que la performance du modèle appris augmente du fait de l'entraînement sur un volume important de données ? Justifier votre réponse

Pour cette question nous avons réduit la taille du dataset kddcup à 15 Mo.

```
Classifier trained in 243.698 seconds
Test Accuracy = 0.99436
```

Nous allons encore réduire le dataset à 1.5 Mo.

```
Classifier trained in 115.075 seconds
Test Accuracy = 0.998655
```

La performance du modèle appris peut augmenter en s'entraînant sur un volume important de données, mais il y a des limites à cette amélioration. Au début, l'entraînement sur un volume plus important de données est bénéfique, car cela permet au modèle d'apprendre des motifs plus riches et généraux. Un modèle exposé à une grande variété de données est plus susceptible de généraliser correctement. Un modèle entraîné sur un petit ensemble de données a plus de chances d'overfit ces données, c'est-à-dire de s'adapter trop précisément aux données d'entraînement. L'entraînement sur un ensemble de données plus important réduit le risque de surajustement. Les modèles d'apprentissage automatique ont des hyper-paramètres qui sont estimés à partir des données. Plus il y a de données, plus ces estimations sont précises, ce qui peut améliorer la performance du modèle.

2.4 L'opération de réglage (tuning) des hyper-paramètres d'un modèle appris, comme par exemple la profondeur de l'arbre (maxDepth), le nombre minimum d'instances par nœud (minInstancesPerNode) nécessaire pour une division, est une opération très couteuse en temps et en particulier dans le contexte du bigdata. Montrer, en faisant la comparaison avec scikit-learn en python et MLlib en pyspark, pour un jeu de données important, que le recours à une validation croisée distribuée pour trouver le meilleur modèle, est indispensable. Justifier votre réponse à travers l'expérimentation avec scikit-leran et MLlib et le calcul du temps écoulé

La version distribuée avec Spark est plus rapide pour le réglage des hyper-paramètres sur de gros ensembles de données, car elle répartit la charge de calcul sur plusieurs nœuds. Cependant, la qualité du modèle est similaire.

La justification réside dans le fait que le traitement distribué permet d'explorer un espace d'hyper-paramètres plus rapidement, car il parallélise les calculs sur un cluster de machines.