COMPTE RENDU

1. Resumé

Dans le contexte de l'apprentissage supervisé l'objectif est de trouver un modèle de classification capable d'assigner une classe à un nouveau modèle en se basant sur les données. L'un des problèmes dans ce contexte est de classer des motifs dans des classes séparables non linéaires. Pour résoudre un tel problème, les arbres de décision binaires sont une très bonne approche.
Mais dans le cas du big data où on traite de très gros volume de données on rencontre de nombreuses difficultés qui sont liées au temps d'exécution, c'est la raison pour laquelle diverses technologies comme map reduce ont été développées pour être utilisées dans le cas particulier du big data. Ainsi l'article traite sur la mise en place d'un algorithme parallèle appelé MR-Tree qui utilise le modèle de programmation Map Reduce et montre la garantie du passage à l'échelle.

2. Montrer la garantie du passage à l'échelle.

Pour montrer la garantie du passage à l'échelle j'ai décidé de travailler avec 2 data sets en me basant sur l'hypothèse suivante :

Si avec un jeu de donnée de 32561 lignes j'arrive à avoir des résultats similaires à ceux de l'article cela veut dire qu'avec un jeu de donnée de 4 millions de lignes je devrais avoir les mêmes résultats.

Les contraintes utilisées pour l'expérimentation sont les suivantes :

- Utilisation d'une crossvalidation avec 5 blocs
- Utilisation du paramètre maxdepth avec plusieurs valeurs possibles

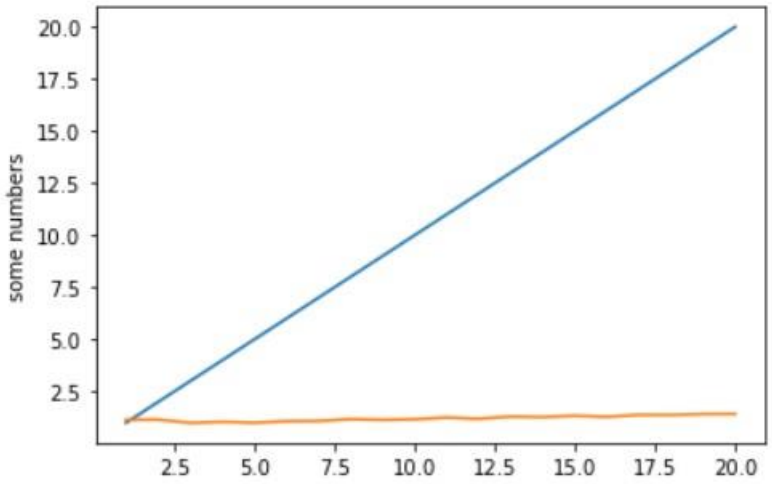
J'ai choisi ces contraintes car ce sont des opérations couteuses ainsi elles permettent de mieux évaluer la pertinence de l'article.

Expérimentation sur un dataset de 32561 lignes :

J'ai choisi de prendre comme scale factor SF=5% qui représente 5% du jeu de donnée soit 1628 lignes Résultats obtenus:

Dataset size (%)	SF - dataset size	Execution time(secondes)	SF - execution time
5%	1	41.687	1.14909537
10%	2	41.821	1.15278183
15%	3	36.278	1
20%	4	37.973	1.04670874
25%	5	36.464	1.00512937
30%	6	39.038	1.07607943
35%	7	39.537	1.08982384
40%	8	42.721	1.17759086
45%	9	41.376	1.14053732
50%	10	42.521	1.17209206
55%	11	45.718	1.26021935
60%	12	43.453	1.19778655
65%	13	47.325	1.30450824
70%	14	46.424	1.27967977
75%	15	48.719	1.34293799
80%	16	46.846	1.291303
85%	17	50.147	1.38230701

90%	18	49.975	1.37755354
95%	19	51.574	1.42161913
100%	20	51.798	1.42780727



Expérimentation sur un dataset de 2millions de lignes :

j'ai choisi de prendre comme scale factor SF=5% qui représente 5% du jeu de donnée

Dataset size (%)	SF - dataset size	Execution time(secondes)	SF - execution time
5%	1	284.878	1.07220457
10%	2	273.202	1.028261
15%	3	281.397	1.05910519
20%	4	265.693	1
25%	5	275.886	1.03836367
30%	6	275.695	1.03764239
35%	7	279.54	1.05211547
40%	8	283.064	1.06537963
45%	9	368.933	1.38856716
50%	10	331.279	1.24684587
55%	11	342.574	1.28935941
60%	12	354.937	1.33588832
65%	13	343.872	1.29424306
70%	14	348.28	1.31083283
75%	15	357.328	1.34488921
80%	16	367.078	1.38158475
85%	17	376.934	1.41868166
90%	18	383.415	1.44307154
95%	19	440.342	1.65733245
100%	20	446.344	1.67992307

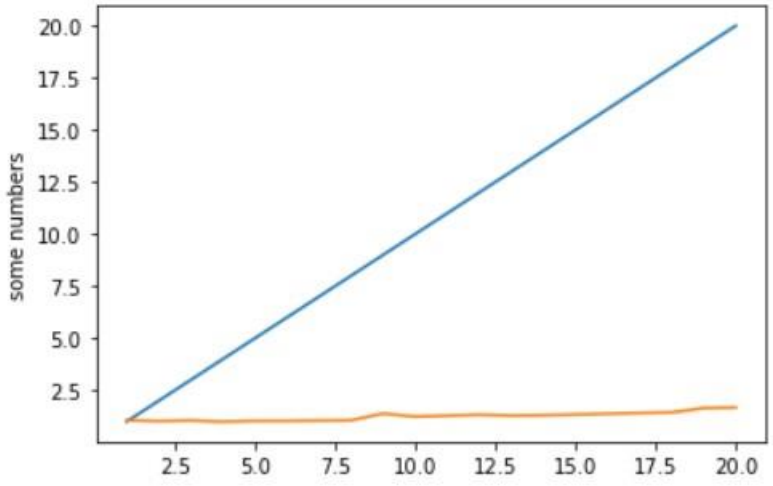
Stages for All Jobs

Completed Stages: 984
Skipped Stages: 1

Completed Stages (984)

Page: 1 2 3 4 5 6 7 8 9 10 > 10 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
984	collectAsMap at MulticlassMetrics.scala:61 +details	2022/10/17 15:49:48	3 ms	1/1			301.0 B	
983	map at MulticlassMetrics.scala:52 +details	2022/10/17 15:49:48	0,3 s	1/1	3.8 MiB			301.0 B
982	collect at C:\Users\arthu\AppData\Local\Temp\ipykernel_19412\1821786944.py:30 +details	2022/10/17 15:49:47	66 ms	1/1	3.8 MiB			
981	collect at C:\Users\arthu\AppData\Local\Temp\ipykernel_19412\1821786944.py:29 +details	2022/10/17 15:49:47	0,3 s	1/1	3.8 MiB			
980	collectAsMap at MulticlassMetrics.scala:61 +details	2022/10/17 15:49:47	3 ms	1/1			301.0 B	
979	map at MulticlassMetrics.scala:52 +details	2022/10/17 15:49:46	0,4 s	1/1	3.8 MiB			301.0 B
978	collectAsMap at RandomForest.scala:663 +details	2022/10/17 15:49:46	45 ms	1/1			93.6 KiB	
977	mapPartitions at RandomForest.scala:644 +details	2022/10/17 15:49:46	41 ms	1/1	2.7 MiB			93.6 KiB
976	collectAsMap at RandomForest.scala:663 +details	2022/10/17 15:49:46	36 ms	1/1			76.2 KiB	
975	mapPartitions at RandomForest.scala:644 +details	2022/10/17 15:49:46	51 ms	1/1	2.7 MiB			76.2 KiB
974	collectAsMap at RandomForest.scala:663 +details	2022/10/17 15:49:46	25 ms	1/1			56.7 KiB	
973	mapPartitions at RandomForest.scala:644 +details	2022/10/17 15:49:46	36 ms	1/1	2.7 MiB			56.7 KiB



Conclusion

Comme peut le constater grâce aux deux courbes on obtient effectivement des résultats très similaire à ceux de l'article tout en aillant rajouté des opérations couteuses comme la crossvalidation ainsi on peut affirmer que le passage à l'échelle est garanti.

3. Montrer la nécessité de la distribution de l'apprentissage dans le cas du BigData

Pour montrer la nécessité de la distribution de l'apprentissage dans le cas du BigData, j'ai fait mes expérimentations de deux manières différentes afin d'évaluer le temps d'exécution et la précision de la machine.

a. Premier mode d'expérimentation

Dans ce cas de figure pour être sûr de bien évaluer la précision de la machine par rapport au nombre de cœur, nous avons choisi de redémarrer le noyau de jupyter notebook à chaque fois que l'on passe d'un nombre de cœur x à un nombre de cœur y.

Résultat de l'expérimentation

Avec 2 cœurs : le temps d'exécution est de 67.5s et la precision est à 82.21%

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6835  609]
 [1072 1304]]
Accuracy: 0.8221606703311388
Classifier trained in 67.5 seconds
```

Avec 4 cœurs : le temps d'exécution est de 58.632s et la precision est à 82.07%

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6658  663]
 [1024 1344]]
Accuracy: 0.8207963024149566
Classifier trained in 58.632 seconds
```

Avec 6 cœurs : le temps d'exécution est de 54.836 s et la precision est à 81.16%

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6801  640]
 [1084 1216]]
Accuracy: 0.8161258590991362
Classifier trained in 54.836 seconds
```

Avec 8 cœurs : le temps d'exécution est de 53.111s et la précision est à 82.64%

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6778  651]
 [1036 1345]]
Accuracy: 0.8226412451104003
Classifier trained in 53.111 seconds
```

Interprétation des résultats :

Comme on peut le constater le nombre de cœur influence considérablement le temps d'exécution cependant la précision varie beaucoup mais tourne d'une certaine valeur et cela peut être due au fait que le choix du jeu d'entrainement et de test est fait de manière aléatoire à cause de la fonction randomSplit([0.7,0.3]).

b. Deuxième mode d'expérimentation

Dans ce cas de figure pour toujours évaluer la précision et le temps d'exécution de la machine par rapport au nombre de cœur, nous avons choisi de faire un sc.stop() à chaque fois que l'on passe d'un nombre de cœur x à un nombre de cœur y.

Résultat de l'expérimentation

Avec 2 cœurs on a 85.26% de précision et 41.718s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 41.718 seconds
```

Avec 4 cœurs on a 85.26% de précision et 40.095s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 40.095 seconds
```

Avec 6 cœurs on a 85.26% de précision et 39.699s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.699 seconds
```

Avec 8 cœurs on a 85.26% de précision et 39.579s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.579 seconds
```

Avec 10 cœurs on a 85.26% de précision et 39.465s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.465 seconds
```

Avec 12 cœurs on a 85.26% de précision et 39.424s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.424 seconds
```

Avec 14 cœurs on a 85.26% de précision et 39.434s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.434 seconds
```

Avec 16 cœurs on a 85.26% de précision et 39.56s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predict
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.56 seconds
```

Avec 18 cœurs on a 85.26% de précision et 39.519s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.519 seconds
```

Avec 20 cœurs on a 85.26% de précision et 39.462s de temps d'exécution

```
Prediction Accuracy:  DataFrame[features: vector, classe_indexed: double, rawPrediction: vector, probability: vector, predictio
n: double]
Confusion Matrix:
[[6850  626]
 [ 809 1591]]
Accuracy: 0.8526970939481168
Classifier trained in 39.462 seconds
```

Interpretation des résultats :

Comme on peut le constater le nombre de cœur influence considérablement le temps d'exécution cependant la précision n'a pas varié peut importe le nombre de cœurs peut importe le nombre de coeur **Conclusion**

Je peux affirmer à partir de mes résultats qu'il est nécessaire de distribuer l'apprentissage dans le cas du big data car cela permet d'améliorer les temps d'exécution sur de très grosses quantité de donnée pour ce qui est de la précision je ne peux pas me prononcer car dans ce cas de figure nous n'avons pas essayer de bâtir un modèle capable de faire de bonne prédiction cependant j'ai pu constater que la précision tourne toujours autour d'une certaine valeur

4. Montrer qu'une validation croisée distribuée pour trouver le meilleur modèle, est indispensable

Pour faire la comparaison j'ai fourni plusieurs valeurs pour le maxdepth dans l'implémentation avec scikit-learn en python et MLlib en pyspark, comme le montre les images suivantes : • **scikit-learn en python**

```
for depth in [2,5,10,15,20,25]:
    model= DecisionTreeClassifier(max_depth=depth)
    score=cross_val_score(model,X_train,y_train,cv=5, scoring='accuracy').mean()
```

- **MLlib en pyspark**

```
classifier = DecisionTreeClassifier(labelCol="classe_indexed", featuresCol="features")
# add empty parameter grid
paramGrid = (ParamGridBuilder()
             .addGrid(classifier.maxDepth, [2,5,10,15,20,25]) # max depth parameter
             .build())

# create evaluator
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="classe_indexed")

# create cross validation object
crossval = CrossValidator(estimator=classifier,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)
```

Après entrainement j'ai pu obtenir les temps d'entrainement suivant pour un jeu de données de 2 millioins de lignes :

- **scikit-learn en python**

```
Classifier trained in 136.432 seconds
```

- **MLlib en pyspark**

```
Classifier trained in 41.315 seconds
```

Ces résultats nous montrent effectivement que le recours à une validation croisée distribuée est une nécessité car le temps d'exécution est beaucoup plus rapide.