Big Data & Machine Learning

SÉANCE 1 – INTRO, MODÉLISATION SPARK & XGBOOST

Organisation

Séance 1
- Introduction sur l'évolution du Big Data et du Machine Learning
- Modélisation avec le Framework Spark (technique de Random Forest)
- Modélisation avec XGBoost (Gradient Boosting)

Séance 2
- Le pourquoi du Framework DASK?
- Utilisation de DASK via une régression logistique et un Gradient Boosting
- Introduction à l'explicabilité / interprétabilité des modèles (techniques Shap et Lime)
- Introduction à TPOT (algo évolutionniste génétique de sélection de modèles)

Séance 3
- Implémentation des techniques de clustering (KMeans, KNN, Spectral)
- Comparaison CPU vs GPU (Framework RAPIDS de Nvidia)

Séance 4
- Distinction des différentes méthodes de réseaux de neurones (Perceptron, CNN, RNN)
- Classification de texte, analyse de sentiment, NLP

Informatique + statistique = ML ?

Les techniques statistiques couplées à la puissance informatique ont permis de décupler les possibilités

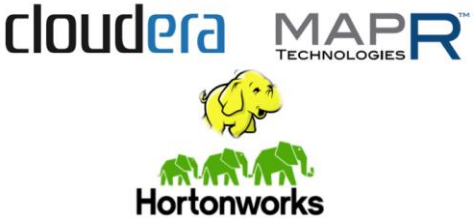
- Naissance du Framework Spark, et des techniques de parallélisation/distribution YARN/Hadoop

- Apparition des technologies cloud (Docker, Kubernetes…)

- Développement des langages Python et R autour des librairies de data science
  - Croissance des communautés autour des Frameworks de Machine Learning : scikit-learn, matplotlib, XGBoost, Dask…
  - Forte adhésion autour de librairies/Frameworks communs

Spark / YARN

Rôles



Fournisseur de distribution, permettant d'installer toute une stack d'outils : Hbase, Kafka, HDFS, Spark/YARN…
Fusion de Cloudera et Hortonworks

Spark

- Framework écrit en scala
- Plusieurs interfaces (Scala, SQL, Python, R)
- Traitement de la données (filtres, agrégations, transformations)
- Réalisation de traitements de Machine Learning (MLlib)
- Traitements graph (GraphX)
- Framework Lazy, les transformations ne s'exécutent réellement que lorsqu'une action est soumise (écriture, affichage…)
- Construction d'un graph d'opérations (DAG), afin d'optimiser l'enchainement des tâches (shuffle de données…)
- Stockage des données en mémoire (sérialisation sur disques possible)

YARN

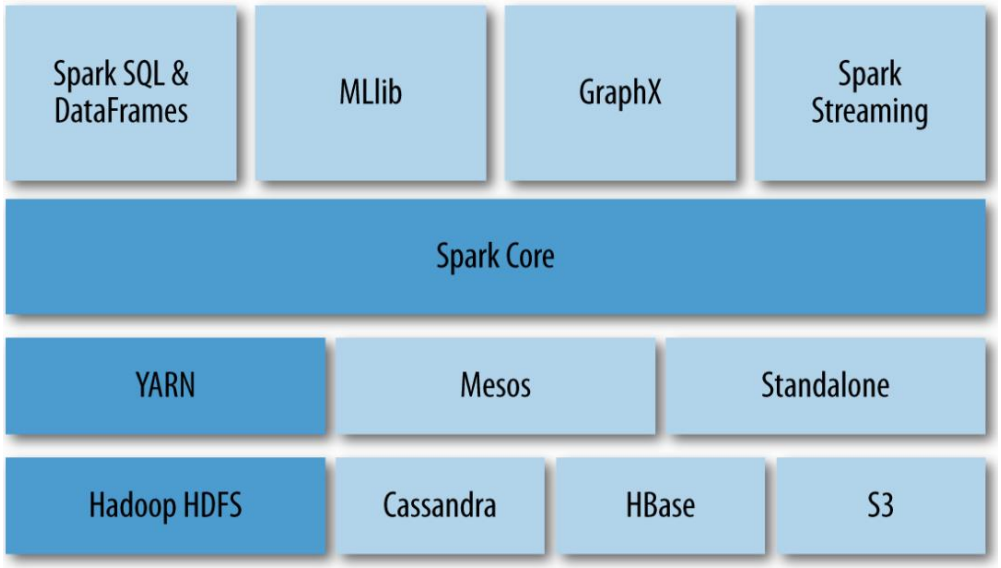
- Orchestrateur de traitements
- Négociateur de ressources (allocation de ressources, gestion de traitements concurrents)
- *Exécution des programmes au plus près de la donnée (paradigme remis en cause)*
- Ordonnancement de l'ensemble des tâches
- Tolérance à la panne, l'ensemble est supervisé (si une tâche tombe en erreur, elle est relancée automatiquement)

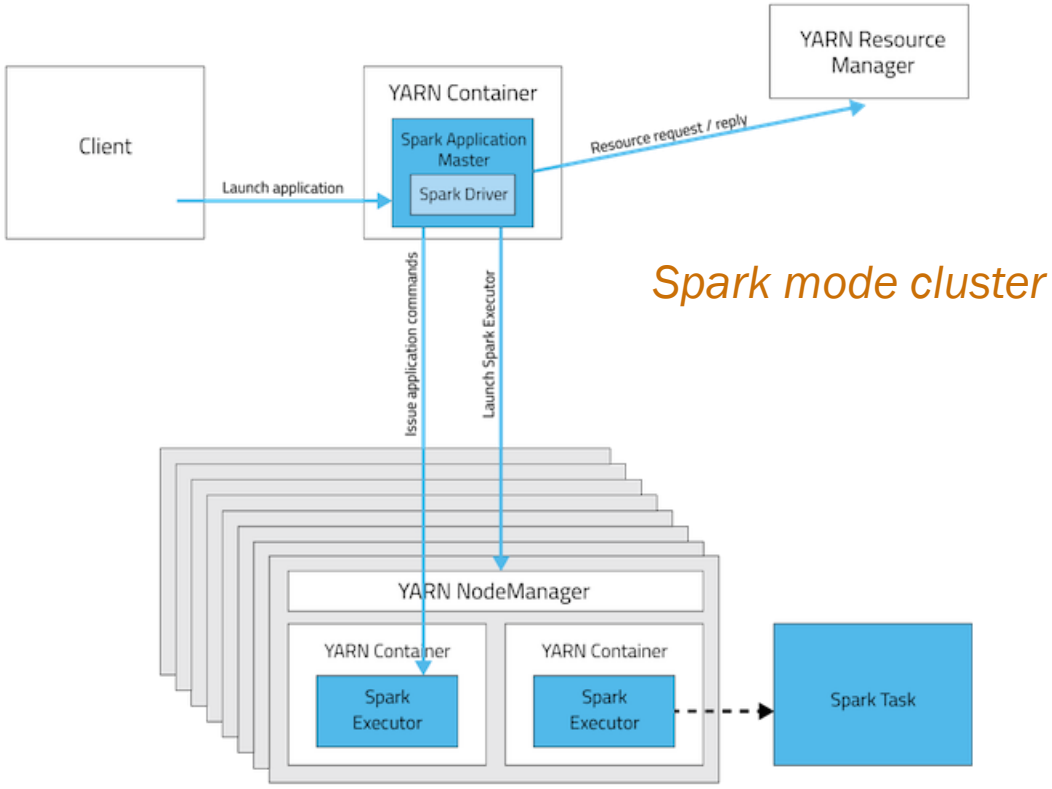
HDFS

- Surcouche sur le système de fichier Linux /ext3, ext4, afin d'établir un système de fichiers distribués, avec réplication (résilient à la perte de données)

Spark / YARN

Fonctionnement





Spark mode cluster

MR
Map Reduce

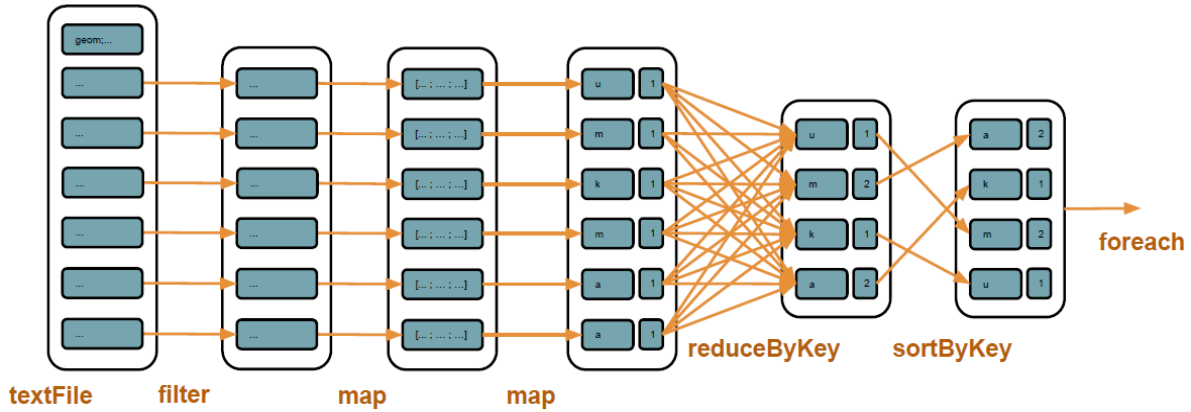
- Map : fonction obligatoire, réalisée sur toutes les valeurs
- Reduce : fonction optionnelle, agrégation, filtre…

Parallélisation des traitements sous forme (K,V), clé / valeur

Ex : Comptage du nombre d'espèces de plante

```
geom_x_y;circonfere;adresse;hauteurenm;espece;varieteouc;dateplanta
48.8648454814, 2.3094155344;140.0;COURS ALBERT 1ER;10.0;Aesculus hippocastanum;;
48.8782668139, 2.29806967519;100.0;PLACE DES TERNES;15.0;Tilia platyphyllos;;
48.889306184, 2.30400164126;38.0;BOULEVARD MALESHERBES;0.0;Platanus x hispanica;;
48.8599934405, 2.29504883623;65.0;QUAI BRANLY;10.0;Paulownia tomentosa;;1996-02-29
```

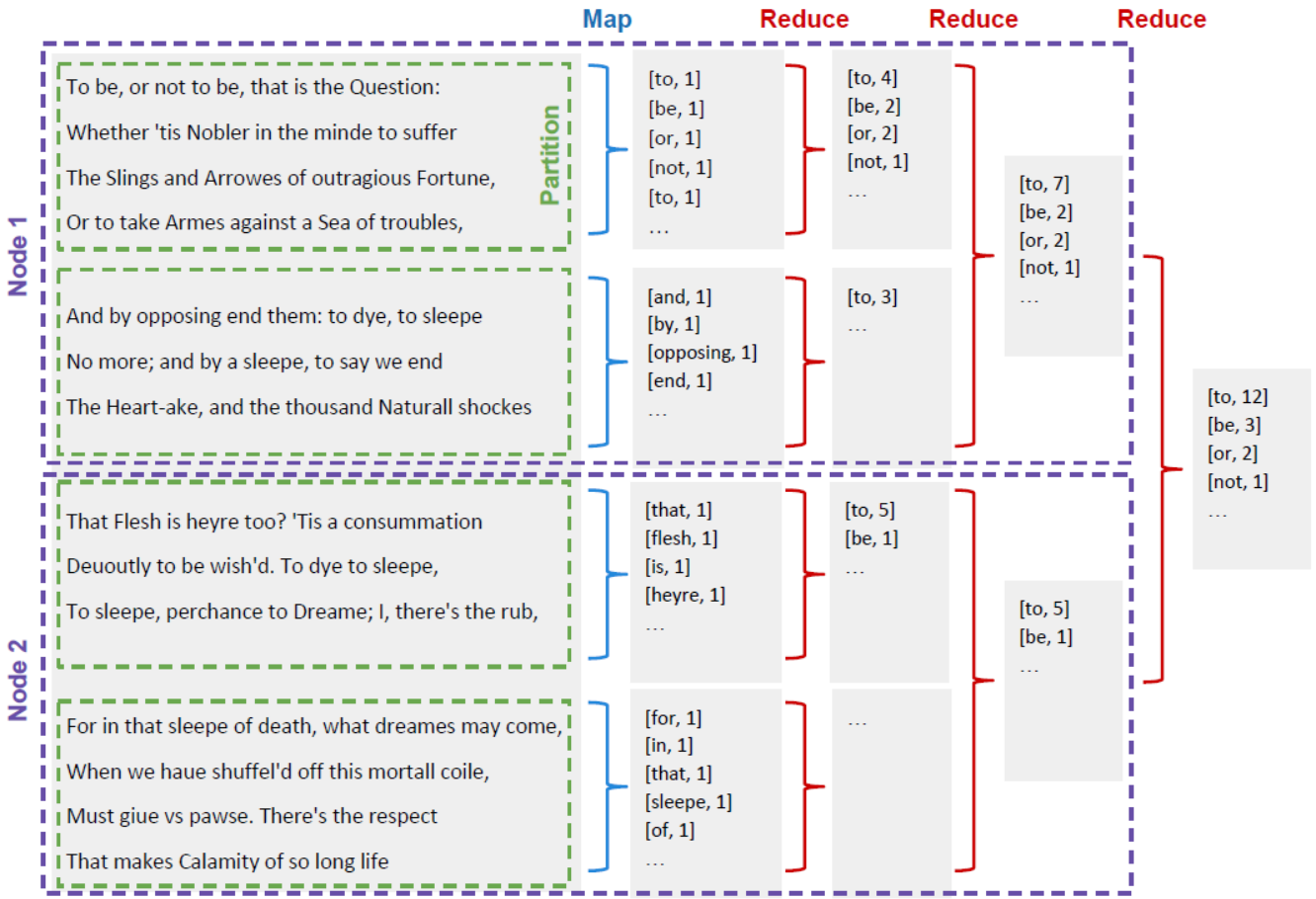
```
val sc = new SparkContext("local", "arbres")

sc.textFile("data/arbresalignementparis2010.csv")
        .filter(line => !line.startsWith("geom"))
        .map(line => line.split(";", -1))
        .map(fields => (fields(4), 1)) // creates a Tuple2
        .reduceByKey(_+_)
        .sortByKey()
        .foreach(t => println(t._1 + " : " + t._2))
```



MR

Map Reduce

Autre exemple : words count



YARN vs Kubernetes

YARN - Localité de la donnée

Sollicitation disque

- Déplacement des binaires au plus près des blocs de données sur les nœuds du cluster
- Sollicitation des vitesses de lecture / écriture des disques
- Il est conseillé d'avoir des baies de disques, et du matériel réseau (switch…), dédiés,
- Privilégier les machines physiques (vs VM), au moins pour les Data Nodes

Kubernetes - Shuffling de la donnée

Sollicitation bande passante réseau

- Les binaires sont exécutés et parallélisés sur les noeuds, indépendamment de l'emplacement des données
- Sollicitation du débit réseau
- Il est également conseillé d'avoir du matériel et des disques dédiés à ces usages (éviter de rentrer en concurrences avec d'autres services)

NB : Spark permet désormais de se baser également sur un système Kubernetes

Dask vs Spark

Dask

- Interface sur les Frameworks existants : scikit-learn, XGBoost, pandas, numpy
- Même syntaxe, pas d'apprentissage supplémentaire
- Capitalisation sur les communautés existantes, plus de fonctionnalités, meilleure réactivité
- Dask s'intègre à Jupyter pour les consoles de suivi de traitements
- Les traitements de ML sont parallélisés et/ou distribués

Spark

- Framework développé en scala, ne s'appuyant pas sur les librairies existantes
- Nécessite d'apprendre une nouvelle syntaxe et une façon de fonctionner
- Moins de réactivité, redéveloppement des librairies, pas d'appuis sur les communautés
- Les traitements de ML sont principalement parallélisés

GPU

Nvidia

Librairie RAPIDS de Nvidia

- cuPY : numpy
- cuDF : Pandas
- cuML : scikit-learn
- cuDNN : interface pour le Deep Learning

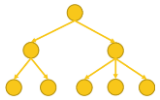
PyTorch et Tensorflow sont compatibles CPU et GPU

- A l'instar de Dask, RAPIDS conserve la même syntaxe que les librairies existantes
- Le passage de CPU à GPU est ainsi transparent pour les utilisateurs
- Dask fournit une interopérabilité avec les librairies RAPIDS, qui permet de parraléliser certains traitements sur plusieurs GPU

Nouvelles techniques de modélisation
Machine/Deep Learning

Traitements statistiques

- Modèles de régression généralisés (GLM) : linéaire, logistique, de poisson, pénalisées
- Approche bayésienne (probabilités) : naïves bayes, réseaux bayésien
- Analyses factorielles : ACP, ACM, AFC

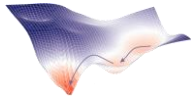


Machine Learning

- Arbres de décision : decision tree, random forest, gradient boosting…
- Clustering : Kmeans, KNN, Spectral, CAH…

Deep Learning

- Multi-layer perceptron (MLP)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)



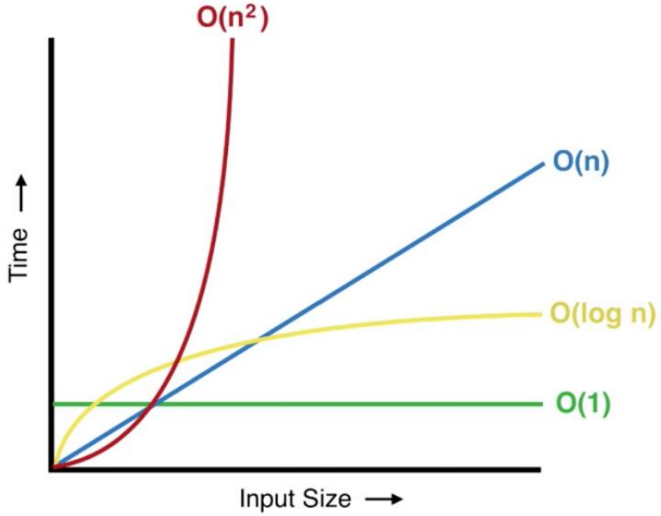
Descente de gradient

- Ligne (stochastique)
- Lot / batch
- Totalité

2 types de problèmes

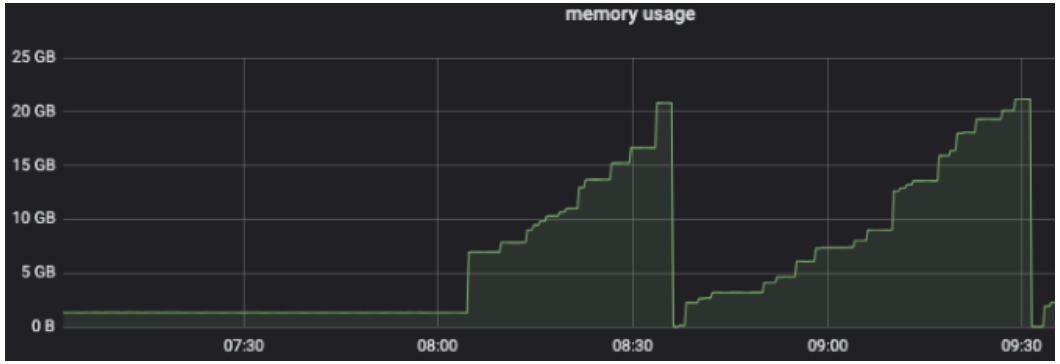
Complexité des algorithmes

Algorithmes longs en temps d'exécution et gourmand en calcul pour converger.
Besoin de parallélisation



La taille des jeux de données

La taille des jeux de données peut ne pas tenir en mémoire sur un seul nœud/serveur, il faut alors distribuer le traitement



Random Forest

Principes

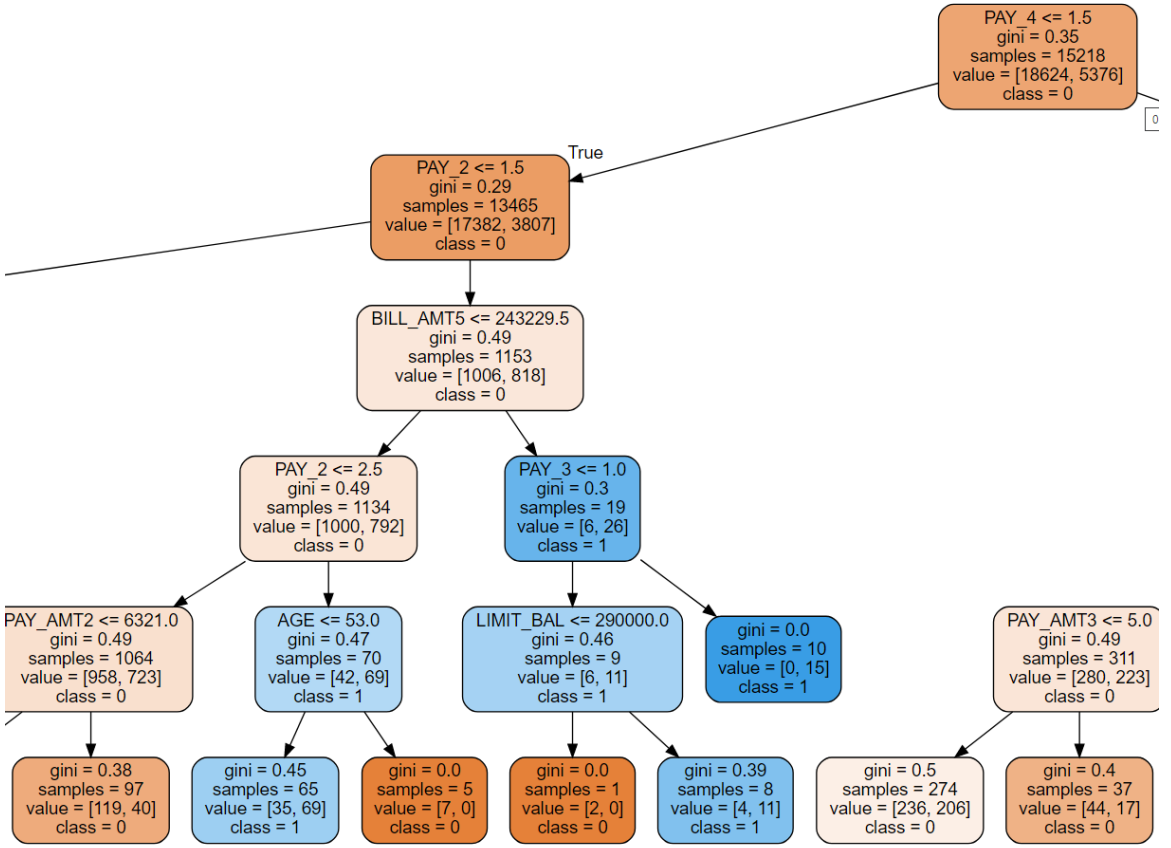
Indice de Gini :

$$G = \sum(pk * (1 - pk))$$

$$G = 1 - \sum(pk)^2$$

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default
0	1	20000.0	2	2	1	24	2	2	0.0	0.0	0.0	689.0	0.0	0.0	0.0	0.0	1
1	2	120000.0	2	2	2	26	-1	2	3455.0	3261.0	0.0	1000.0	1000.0	1000.0	0.0	2000.0	1
2	3	90000.0	2	2	2	34	0	0	14948.0	15549.0	1518.0	1500.0	1000.0	1000.0	1000.0	5000.0	0
3	4	50000.0	2	2	1	37	0	0	28959.0	29547.0	2000.0	2019.0	1200.0	1100.0	1069.0	1000.0	0
4	5	50000.0	1	2	1	57	-1	0	19146.0	19131.0	2000.0	36681.0	10000.0	9000.0	689.0	679.0	0

- Création d'un grand nombre d'arbres (200, 500…)
- Chaque arbres possède une profondeur, avec un nombre de nœuds de scission
- Classification : pour chaque nœud on sélectionne une variable avec une valeur qui minimise le plus l'indice de Gini (meilleur split)
- Régression : diminution du MSE/RMSE
- On répète l'opération jusqu'à obtenir les nœuds les plus pures, où jusqu'à atteindre le nombre minimal d'individus par nœud
- In fine, on réalise un vote sur l'ensemble des arbres, où une moyenne dans le cas d'une régression



Random Forest

Hyperparamètres

Scikit-learn

- n_estimators : nombre d'arbres de la forêt
- max_depth : profondeur maximale des arbres
- max_features : nombre maximal de variables considéré à chaque nœud
- boostrap (True/False) : si True, un sous-ensemble de données est sélectionné à chaque nœud (~63,2%), si False, l'entrainement est réalisé sur la totalité du jeu de données

Chaque arbre a donc 2 sources de variation : les variables, et le sous-ensemble de données sélectionné

Exercice

Modélisation Spark

- Lien du dataset
- Lien du Notebook Google Colab

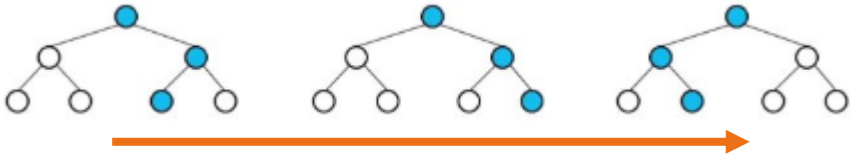
Particularité de Spark

Pour utiliser les fonctions statistiques ou de ML de Spark, il faut au préalable créer un vecteur des variables souhaitées

age	income	features_vector
25	35000	[25.0, 35000.0]

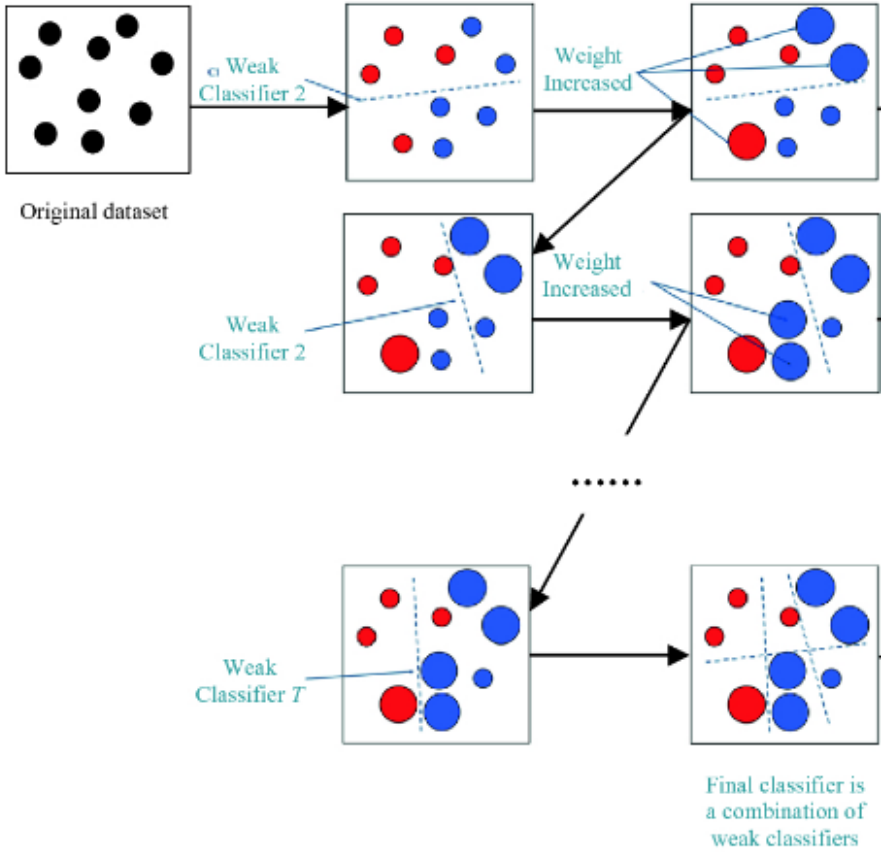
Gradient boosting

Principes



Même principe, mais ici les arbres sont construits de manière séquentielle, les erreurs de l'arbre a-1 sont pondérées afin que l'arbre suivant affine ses résultats

Le système de vote final est ensuite pondéré selon la performance en classement, plus l'arbre à un bon score, plus son vote est important.



Gradient boosting

Hyperparamètres

XGBoost

A l'instar de la régression pénalisée Elastic net, il permet de mixer les techniques du Random Forest et du Gradient Boosting, le choix est ainsi libre de tendre vers l'une ou l'autre.

- **n_estimators** : nombre d'itérations, jusqu'à ce que ce nombre ou **early_stopping_rounds** soit atteint
- **learning_rate** : le taux d'apprentissage choisi est en general entre 0.1 et 0.01, pour une convergence lente
- **subsample** : sous ensemble de données considéré à chaque noeud, ici 63% peut être sélectionné, pour approcher les techniques du random forest
- **colsample_bynode** : nombre maximal de colonnes testées à chaque noeud (equivalent au **max_features** du random forest), ici un ratio est attendu
- **num_parallel_tree** : nombre d'arbres entrainés en parallèle, si 10, 10 arbres sont entrainés simultanément à chaque iteration
  - Si 1 seul arbre est entrainé à chaque itération : gradient boosting
  - Si 500 arbres sont entrainés en une seule itération : random forest
- **early_stopping_rounds** : arrête l'entrainement si le modèle ne s'est pas amélioré sur X iterations, en se basant sur la dernière métrique passée dans : **eval_metric list**
  - Valable pour tous les modèles itératifs : regressions, reseaux de neurons…
- **eval_set** : contient une liste de tuples de datasets (X, y), après chaque iterations le modèle est évalué sur ces datasets
- **eval_metric** : liste de métriques pour évaluer le modèle après chaque iteration : rmse, mae, logloss, aucpr, auc…

Exercice

Modélisation XGBoost

- Même dataset
- Lien du Notebook Google Colab