Projet Big DATA

Johanne DARIUS
Samuel LAMOUROUX
Abdoul NDIAYE
Célian RINGWALD

4 avril 2016

Table des matières

1 Introduction **2**

2 Présentation de Spark **2**
2.1 SPARK, qu'est-ce que c'est ? . . . 2
2.2 Fonctionnalités de l'outil . . . 2
2.3 Spark vs Hadoop . . . 3

3 Installation **4**

4 Première prise en main **5**
4.1 Les fichiers fournis . . . 5
4.2 Création d'une application Spark . . . 5

5 Ma première application Spark : un clustering de morceaux de Bach **6**
5.1 Création d'un application pyspark . . . 6
5.2 Importation et transformation des données . . . 7
5.3 Clustering . . . 8
5.4 Représentation graphique via Matplot . . . 9
5.5 Lancez votre application et modifiez vos paramètres . . . 10

6 Induction des règles du Poker **11**
6.1 Présentation des données . . . 11
6.2 Processus d'analyse . . . 12
6.3 Première prise en main et processus d'import des données . . . 13
6.4 Decision Tree Classification . . . 15
6.5 Gradient Boosting Classification . . . 15
6.6 SVM et Regression Logistique . . . 15
6.7 Notes . . . 15

7 Conclusion **16**

8 Pour aller plus loin... **16**

1 Introduction

Dans le cadre du Master Statistique & Informatique Socio-Economique (SISE), nous avons plusieurs projets à réaliser. Notre dernier projet de l'année est le projet Big Data, dans lequel nous devons travailler sur l'association de deux outils : SPARK et PYTHON et ainsi réaliser une étude à l'aide d'un jeu de données que nous aurons choisi et faire une présentation de ceci ainsi qu'un rapport rédigé en LateX. Ce projet va nous permettre de mettre en avant un travail d'équipe et de méthodologie mais également de découvrir des outils phares du Big Data.

Dans un premier temps, nous présenterons l'outil SPARK et son cadre d'utilisation, ses limites et ses fonctionnalités. Nous réaliserons ensuite notre première application, qui nous permettra de réaliser un clustering et de le visualiser. Enfin nous lancerons plusieurs algorithmes de machine learning afin de les comparer dans le cadre de l'induction des règles du jeu de Poker.

2 Présentation de Spark

2.1 SPARK, qu'est-ce que c'est ?

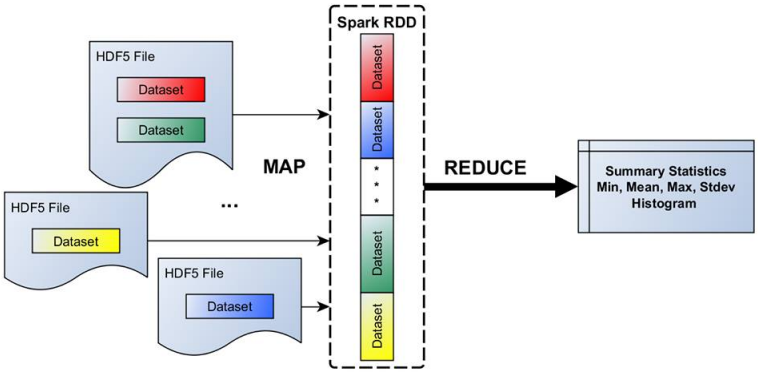
Spark (ou Apache Spark) est un framework open source développé par Matei Zaharia durant sa fin de son doctorat en 2006 afin de répondre à une palette de demande plus large que Map Reduce et en augmenter les capacités de calcul. Ecrit en scala et s'exécutant sur la machine virtuelle Java (JVM), c'est un logiciel d'analyse complet, conçu pour sa rapidité et sa facilité d'utilisation. [1] Il permet de nombreux traitements Big Data pour divers types de jeux de données. Cet outil repose sur deux principaux concepts : le RDD (Resilient Distributed Datasets), une collection calculée à partir d'une source de données et stockée en mémoire vive, qui améliore considérablement certains traitements, dont ceux basés sur des itérations ; ainsi que l'architecture distribué, permettant d'augmenter significativement la vitesse de calcul, et permet de répliquer les données en cas d'accident. Spark dans sa version intiale possède nombreuses librairies :la fouille de données, le traitement des graphes, traitement de flux, management des données via un système SQL...

Il permet à des applications sur clusters Hadoop d'être exécutées jusqu'à 100 fois plus vite en mémoire, 10 fois plus vite sur un disque, et même 100 fois plus d'après son créateur [2]. Spark fournit également plus de stabilité que d'autres outils de traitement temps réel, utilisant Hadoop. Son objectif est donc de permettre aux développeurs de créer des applications Big Data en facilitant l'écriture et en améliorant la vitesse d'exécution de ces applications.

2.2 Fonctionnalités de l'outil

La base de Spark se trouve dans les RDDs (Resilient Distributed Datasets) qui représente une partition des données que l'utilisateur va utiliser pour ses calculs. Ces partitions sont mises en mémoire (RAM), ainsi cela évite au système de faire des lectures disque à chaque fois qu'une donnée est nécessaire à un calcul. C'est pour cela que Spark peut être considéré comme étant beaucoup plus rapide que d'autres solutions. Celles-ci sont stoquées dans le Hdfs de Hadoop (qui est intégré dans Spark), reppliquées sous forme de données Parquet (base de données orientée colonnes. Mais vous pouvez biensur faire appel à d'autres types de Base de données NOSQL, tel que HBase ou Cassandra.

1. Interview de Matei Zaharia par Gregory Piatetsky : http ://www.kdnuggets.com/2015/05/interview-matei-zaharia-creator-apache-spark.html
2. Spark : Cluster Computing withWorking Sets Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica



Les RDDs sont créés par des opérations déterministes appelées transformations avec notamment map, filter et join. La fonction map() retourne un nouvel ensemble de données distribué formé en faisant passer chaque élément de la source à travers une fonction. La fonction filter() retourne un nouvel ensemble de données formée en sélectionnant les éléments de la source. Enfin il est aussi possible de réaliser des jointures via la fonction join. La nature des RDDs leur permette de mieux gérer les cas où des noeuds peuvent être lents en effectuant des copies des tâches lentes comme dans MapReduce. En scala, chaque closure est représentée comme un objet Java qui peut être sérialisés et chargés sur un autre noeud.
Spark propose différentes librairies officielles manipulables en Java, Scala et Python :

- **Spark SQL (Java, Scala etPython) :** API permettant de requêter les RDDs en SQL.
- **Spark Streaming (Java et Scala) :** API permettant d'écrire des applications streaming
- **MLlib (Java, Scala et Python) :** API reprenant certains des algorithmes bien connus du machine learning (kmeans, régression linéaire, analyse en composantes principales, ...)
- **GraphX (Scala) :** API permettant de manipuler et de faire des opérations sur les RDDs à la manière d'un graph

2.3 Spark vs Hadoop

Hadoop est tout comme Spark, un framework libre et open source, projet de la fondation Apache depuis 2009. Ecrit en java, il permet l'écriture d'application de stockage et le traitement des données. Il a su prouver depuis 10 ans être la solution de choix pour le traitement de gros volumes de donnée, notamment avec la solution MapReduce qui permet de développer des tâches de traitement du cluster.

Mais contrairement à Spark, pour les cas d'utilisation nécessitant d'algorithmes et traitements à plusieurs passes, Hadoop n'est plus approprié. En effet, celui-ci est très efficace pour les traitements à passe unique seulement. De plus, lors des étapes de traitement, Hadoop se voit être peu rapide, du fait de la réplication et du stockage sur disque.

Cependant, ces deux solutions n'ont pas les mêmes car objectifs, car par exemple, Hadoop peut distribuer les grandes quantités de données collectées à travers plusieurs nœuds. Alors que Spark ne sait pas faire du stockage distribué. Spark est comme nous l'avons vu précédemment plus rapide que MapReduce qui fonctionne en étape alors que Spark peut travailler sur la totalité des données en une seule fois. Toutefois, si les besoins fonctionnels sont statiques, MapReduce devient suffisant.
Enfin il est utile finalement de préciser que Spark est construit autour de MapReduce, il faut donc voir l'application comme une extension et une généralisation de MapReduce.

3 Installation

Afin d'utiliser Spark, il existe deux stratégies d'installation :

- **sur une machine virtuelle :** ceci permet en outre de profiter des fonctionnalités que propose les commandes unix. Cloudera propose par exemple ce genre de solution. Cependant, pour les non-initiés, cette voie sera très difficile à utiliser, notamment à cause de la prise en main de l'environnement. De plus de nombreuses manipulations de configurations sont nécéssaires même sur la VM quickstart de l'éditeur. Enfin, sur un ordinateur peu récent cette machine virtuelle risque de consommer beaucoup de mémoire, ce qui n'est pas agréable car cela rend pénible la programmation de routines sous Spark.
- **ou en natif sur le système d'exploitation de votre ordinateur :** c'est cette voie que nous avons utilisé. Nous allons donc évoquer ensemble les différentes étapes d'installation de Spark sous Windows.

Dans un premier temps avant d'installer Spark depuis le site d'Apache, il vous faudra cependant vérifier quelques prérequis. Lancer donc votre ligne de commande afin de les vérifier avec nous si vous :

- possédez une version de Java égale ou supérieure à la version 6 : pour vérifier tapez "java -version".
- possédez une version de Scala égale ou supérieure à la version 2.10 : pour vérifier tapez "java -version.
- possédez une version de Python égale ou supérieure à la version 2.6 : pour vérifier lancez python : "python".

Si vous n'êtes pas à jour sur un de ces points, procédez aux installations nécessaires, en n'oubliant pas à la fin de celles-ci d'ajouter aux variables d'environnement (JAVA_HOME, SCALA_HOME...) de windows (PC > Propriétés > Propriétés avancées > Variables d'environnement).

Attention : selon la version de python et scala que vous possédez, vous devrez vous munir de la version de Spark appropriée. Notez que les versions précédentes de Spark ne contiennent pas toutes les fonctions des nouvelles versions.

Vous pourrez ensuite vous rendre sur le site officiel de Spark : http ://spark.apache.org/downloads.html et télécharger la version qui correspond à vos prérequis, le mieux c'est de posséder les dernières versions de Java, Scala et Python, bien entendu. Nous préférerons prendre dans notre cas la version 1.6 de Spark préconstruite avec Hadoop 2.6 (pre-build).Téléchargez donc une version de Spark, décompressez là et placez là où vous le souhaitez (en général à la racine) par exemple : C ://pyspark/spark-1.6.1-bin-hadoop2.6. Accédez à votre ligne de commande à cet emplacement "cd C ://pyspark/spark-1.6.1-bin-hadoop2.6/bin". Il vous suffira ensuite de lancer la commande "pyspark" pour accéder à Spark en python. Vous noterez qu'il est possible de paramétrer le contexte de Spark au lancement de celui-ci : lors de gros calculs par exemple.

Cependant... verifier ces pré-requis n'est pas suffisant : lancez pyspark, vous verrez se cacher une erreur du type : "java.io.IOException". Cette erreur devrait se cacher dans les quelques lignes que vous renverra Spark au démarrage. La connection avec Hadoop n'étant pas réalisée sur Windows, vous ne pourrez pour le moment pas stocker vos données sur le RDD. Spark est très bavard, vous ne vous ennuyerez pas en sa compagnie, vous pouvez vous créer un système d'enregistrement des logs et désactiver l'option verbose de celui-ci, ceci vous permettra de ne pas friser la crise d'épilepsie lorsque vous rechercherez l'origine d'un bug.

Installez donc l'utilitaire **winutils**. Ce petit exécutable se trouve ici : http ://public-repo-1.hortonworks.com/hdp-win-alpha/winutils.exe, téléchargez-le et placez-le à la racine dans
"C ://winutils/bin/winutils.exe" par exemple. N'oubliez pas de configurer la variable d'environnement HadoopHome : HADOOP_HOME = C ://winutils. Puis relancer Pyspark avec "pyspark" depuis "C ://pyspark/spark-1.6.1-bin-hadoop2.6/bin". Vous pouvez désormais accéder à l'interface de pyspark à l'aide du lien : http ://localhost :4040/ depuis votre navigateur web fétiche. A titre de vérification , lancez "run-exemple SparkPi 10".

->Fécilitations vous venez ainsi d'installer PySpark.

4 Première prise en main

4.1 Les fichiers fournis

Dans un premier temps, nous allons lister les fichiers joints à ce tutoriel :

```
|
|-bach –|app        - BachApp_K-means.py
       –|input      - header.csv
                    - jsbach_chorals_harmony.txt
       –|output
|-poker–|app        - PokerApp_DecisionsTrees.py
                    - PokerApp_GradientBoosting.py
                    - PokerApp_SVM_RegLog.py
       –|input      - header.csv
                    poker-test.txt
                    poker-train.txt
       –|output
```

Nous pouvons observer deux dossier principaux : bach et poker, qui contienent eux même trois répertoires : "app" (contenant les applications spark liées à ce tutoriel), "input" les données utilisées par l'application ainsi que "output" le répertoire dans lequel seront exportés nos résultats.

4.2 Création d'une application Spark

Vous l'avez compris, nous allons donc réaliser deux applications, une fois terminées celles-ci pourront être lancées à l'aide de la commande suivante, nous utilisons dans ce cas des paramètres "tuning" qui nous permette d'augmenter la memoire maximale gérée par l'executeur et le driver de Spark à 2G :
pyspark –driver-memory 2g –executor-memory 2g <App.py> <arg>

Concrètement, construire une application est très simple, voici l'exemple d'un comptage de mots dans un fichier txt :

```
#IMPORT DES LIBRAIRIE UTILES
from __future__ import print_function
import sys
from operator import add
from pyspark import SparkContext
#MAIN
if __name__ == "__main__":
    #Eventuellement : utilisation d'arguments lors du lancement de l'appli
    if len(sys.argv) != 2:
        print("Usage: wordcount <file>", file=sys.stderr)
        exit(-1)
    #Définition du nom de l'application
    sc = SparkContext(appName="PythonWordCount")
    #Import du fichier
    lines = sc.textFile(sys.argv[1], 1)
    #comptage des mots
    counts = lines.flatMap(lambda x: x.split(' ')) \
                  .map(lambda x: (x, 1)) \
                  .reduceByKey(add)
    output = counts.collect()
    for (word, count) in output:
        print("%s: %i" % (word, count))
    sc.stop()
```

5 Ma première application Spark : un clustering de morceaux de Bach

Afin de démarrer notre prise en main de Spark, nous vous proposons de classifier les données fournies par l'UCI Machine Learning Repository qui regroupent 5665 représentations de 60 chorales chantant des morceaux de Johann Sebastian Bach.

5.1 Création d'une application pyspark

Les données sont extraites d'extraits musicaux format midi via l'aide d'un extracteur d'harmonies. Ces données sont décrites par 17 variables : 1.l'ID de la chorale interprétant le morceau [String] 2.Le numéro de l'événement de chaque représentation [Integer] 3-14. les notes présentes dans le morceau [Yes-No] 15. La basse dominante du morceau [String] 16. L'accentuation de l'événement (entre 1 et 5) [Integer] 17. L'accord principal résonant durant la musique [String].

Dans un premier temps, nous allons d'abord créer une application Spark, que vous placerez dans ../data/bach/app et nommerez "Mon K-means à moi.py". A l'intérieur, allez donc copier/coller le squellette suivant :

```
from __future__ import print_function
import sys

from pyspark import SparkContext

from pyspark.sql import SQLContext
from pyspark.sql.functions import col

import numpy as np
//Nous importerons les librairies dont nous avons besoin ici//

// Nous écrirons nos fonctions ici //

if __name__ == "__main__":
    #ERREUR SI K NON DEFINI
    if len(sys.argv) != 2:
        print("Usage: k means <k> : number of cluster")
        exit(-1)

    #Define Spark Context
    sc = SparkContext(appName="Mon K-means à moi")
    #Define SQL Context
    sqlContext = SQLContext(sc)

    // Nous écrirons notre programme ici //

    sc.stop()
```

Pour le moment l'application "Mon K-means à moi" ne réalise pas grand chose : elle vérifie qu'un argument qui correponsdra au nombre de cluster que vous souhaiter réaliser, elle définit un contexte Spark et SQL puis les ferme.

5.2 Importation et transformation des données

Importons desormais nos données ! L'importation sous spark peut se faire sous différentes formes, la plus simple est d'utiliser la fonction sc.textFile(), tels que chaque ligne est reconnue comme un String. Nous allons donc devoir découper cette chaine de caractère afin de la transformer en vecteur, nous parsons donc chaque ligne en se basant sur le séparateur ",".

```
    #Import data
    donnee = sc.textFile("../data/bach/input/jsbach_chorals_harmony.txt")
    #Select only numericals data
    data = donnee.map(lambda line: np.array([x for x in line.split(',')]))
```

Cependant à ce stade les variables textuelles [Yes/No] correspondant à la présence d'une note ainsi que d'autres variables sous forme de chaine de caractères ne nous permettent pas de lancer notre k-means. Nous allons donc devoir laisser de coté les variables ID chorale, basse dominante et accord dominant et nous devons traiter les variables attenantes à la présence d'une note de manière à ce que celles-ci soient binaires. Dans cette optique nous créons la fonction suivante, que nous "mapperons" ensuite sur les données que nous avons importées (à placer donc avant le main) :

```
def filterOut(row):
    #Selection des données
    t=row[2:14]
    #Binarisation
    for i in range(0,len(t)):
        if t[i]=="YES" :
            t[i]=1
        else:
            t[i]=0
    #Retourne données traitées et filtrées
    return np.append(t,row[15])
```

Pour finir, nous transformons ces données en integer puis nous récuperons le k (nombre de cluster passé en paramètre (à la suite dans le main) :

```
    #to integer
    data3=data2.map(lambda line: np.array([int(x) for x in line]))

    #define number k of cluster
    k = int(sys.argv[1])
```

Nous allons désormais appliquer notre algorithme du k-means sur nos données !

5.3 Clustering

Nous allons importer la librairie "KMeans" de MLlib, et la librairie time afin de calculer le temps que met Spark pour effectuer le clustering. (en début de fichier)

```
from pyspark.mllib.clustering import KMeans
from time import time
```

Nous appliquons ensuite notre fonction à nos données :

```
    #Lunch k-means
    t0 = time()
    model = KMeans.train(data2, k, maxIterations=10, runs=10,
    initializationMode="random")
    tt = time() - t0

Nous récupérons nos clusters via la commande :
\begin{DDbox}{\linewidth}
\begin{Verbatim}
    #Find cluster
    clust=model.predict(data2)
```

Une fois cela effectué, nous calculons quelques métriques, vous pouvez en implenter d'autres si vous le souhaitez : Nous choissons de calculer l'inertie pour cela nous créons la fonction error, que nous plaçons en dehors de notre main :

```
def error(point):
    center = model.clusterCenters[
    model.predict(point)]
    return sum([x**2 for x in (point - center)])
```

Puis à partir de cette fonction, calculons la variance intra groupes :

```
    #Compute inertie
    Inert = data3.map(lambda point:error(point)).reduce(lambda x, y: x + y)
    #Compute var Intra
    varIntra=Inert/data3.count()
```

Il peut être intéressant d'exporter le temps de calcul, et ces métriques dans un fichier txt, que nous placerons dans le répertoire output :

```
    #Save summary file
    f = open("../data/bach/output/K_means.txt", 'a')
    f.writelines("model trained in %s seconds \n" %round(tt,3))
    f.writelines("Number of cluster =  %s \n" % k)
    f.writelines("Variance intraclasse =  %s \n" % str(varIntra))
    f.writelines("Inertie = %s \n"  % str(Inert))
    f.close()
```

Enfin nous allons exporter nos données labelisées en csv. Pour cela, nous devons dans un premier temps transformer ces données stoquées pour le moment sur le Hdfs dans un Dataframe SQl puis un Dataframe pandas. Ce passage dans un autre format nous permet entre autre d'utiliser la fonction to_csv().

Afin de créer un Dataframe SQL, nous allons devoir définir un schéma à notre table, nous allons donc extraire du fichier "header.csv" le nom des colonnes et en définir le type. Une fois obtenue cette première dataframe est convertie au format Pandas via to_Pandas().

Mais dans un premier temps, importez la librairie pandas :

```
import pandas
```

Puis exportez vos données en csv dans le repertoire "output" via ce processus :

```
#Concat data with defined clust
concat=data3.zip(clust)

#Import header data
t= sc.textFile("../data/bach/input/header.csv")
header = t.first()

#Parse colnames
schemaString = header.replace('"','')  # get rid of the double-quotes

#Define type of fields
fields = [StructField(field_name, IntegerType(), True) for field_name in
schemaString.split(',')]

#Create Sql schema based on colnames & types
schema = StructType(fields)

#Create DF associated
df= concat.map(lambda p: (int(p[0][0]), int(p[0][1]), int(p[0][2]), int(p[0][3]),
int(p[0][4]) , int(p[0][5]), int(p[0][6]) , int(p[0][7]), int(p[0][8]),
int(p[0][9]), int(p[0][10]),int(p[0][11]), int(p[0][12]),int(p[1]))).toDF(schema)

#to Pandas DF
df2=df.toPandas()

#Save cluster
df2.to_csv("../data/bach/output/BachChoralsClusters.csv")
```

5.4 Représentation graphique via Matplot

Afin d'aller un peu plus loin, nous souhaitons désormais visualiser nos clusters, cependant face à nous un problème technique : MLlib ne fournit pas de moyens de visualiser des données du RDD, et un problème théorique : comment visualiser nos données sur un plan de deux dimensions alors que celles-ci sont décrites par 17 variables ? Nous allons dans un premier temps réaliser une ACP sur nos variables afin de les projeter sur un plan à deux dimensions, puis nous utiliserons la librairie Matplot afin de visualiser ceci.
L'ACP peut être réalisée par MLlib, cependant par un souci de facilité nous utiliserons la librairie Scikit-learn.
Importons donc ces librairies :

```
from sklearn.decomposition import PCA as sklearnPCA
import matplotlib.pyplot as plt
```

Puis réalisons notre ACP :

```
#catch only label column
    label=df2["clust"]
    #the DF without labels
    without=df2.drop("clust",1)
    #PCA on 2 dim
    pca = sklearnPCA(n_components=2)
    #2dimensions Data
    data2D = pca.fit_transform(without)
```

Et enfin traçons notre graphique via Matplot :

```
#Plotting 2D data colored by clust  : if k>5 problem !
colors = ['#581845','#900C3F','#C70039','#FF5733','#FFC300']
col_map=dict(zip(set(label),colors[0:k]))
label_color = [col_map[l] for l in label]

fig, ax = plt.subplots()
ax.scatter(data2D[:,0], data2D[:,1], c=label_color)
plt.show()
```

5.5 Lancez votre application et modifiez vos paramètres

Une fois votre script finit, il ne reste plus qu'à lancer Spark afin de réaliser votre clustering ! Il sera possible alors pour vous de visualiser celui-ci et de consulter les fichiers générés durant la procédure.
Lancez les commandes suivantes depuis votre invite windows :

- Rendez-vous dans votre fichier source Spark :
  cd c :/spark/spark-1.6.1-bin-hadoop2.6/bin
- Puis lancez votre programme de clustering avec k=3 :
  pyspark –driver-memory 2g –executor-memory 2g C :/spark/spark-1.6.1-bin-hadoop2.6/data/bach/app/MonClusteringBach
  3

Si votre programme ne marche pas, utilisez le fichier réponse :
pyspark –driver-memory 2g –executor-memory 2g
c :/spark/spark-1.6.1-bin-hadoop2.6/data/bach/app/BachApp_K-means.py 3
> Le clustering vous semble-t-il cohérent ?

> Allez ensuite dans : C :/spark/spark-1.6.1-bin-hadoop2.6/data/bach/output, Obtenez-vous les fichiers K-means.txt et BachChoralsCluster.csv ? Ouvrez K-means.txt. De combien est votre variance intra-groupe ?

Relancez la procédure pour k=2, 4, 5 et 6, notez à chaque fois la variance obtenue, Quel est le nombre idéal de clusters au sens de la règle du coude ?

6 Induction des règles du Poker

Ce tutoriel a été créé sur la base de cet article comparant différentes méthodes afin d'induire les règles du Poker sur ce même jeu de données sur R :ICI
Dans cette partie nous nous attarderons sur quelques points clefs des applications utilisées et lancerons tour à tour plusieurs algorithmes de machine learning, afin de comparer leurs efficacités dans le cadre de notre problématique.

6.1 Présentation des données

Le poker, tout comme certains jeux de sociétés, n'a pas échappé à la facheuse manie des chercheurs de notre époque à créer des programmes, permettant de battre un humain. Le laboratoire canadien UACPRG (University of Alberta Computer Poker Research Group), en a d'ailleurs fait sa spécialité et en 2015 a réussi à résoudre mathématiquement le jeu du Texas Hold'Em : de ce fait, le logiciel résultant de ces recherches peut alors gagner à coup sûr, et ceci en prenant en compte les configurations de ce jeu, le nombre de joueurs, la mise... [3]
Ainsi introduit, nous allons donc à notre échelle tenter d'induire les règles du poker. Pour cela, nous allons comparer plusieurs méthodes nous permettant ainsi de faire un tour du propriétaire, de continuer à en apprendre sur Spark. Une occasion d'utiliser les librairies disponibles sur MLlib : une classification via un arbre de décision, une méthode de Boosting via descente de gradient, le SVM et la regression logistique. Nous appliquerons ces méthodes au jeu de données déniché sur LIBSVM Data, appelé Poker-hand. Ce jeu de données est scindé en deux : 25 010 mains de poker réservées à l'apprentissage du modèle ainsi que 1 000 000 mains de poker pour la partie test. Chaque main est décrite par 10 attributs : le signe de la carte (Coeur,Carreau,Trèfle,Pic) et le rang de celle-ci (Numerique entre 1 et 13, représentant dans cet ordre : AS,2,3...,Reine, Roi),pour chacune des 5 manches composant un tour. Ainsi que la valeur de la main, qui sera pour le coup notre variable cible :

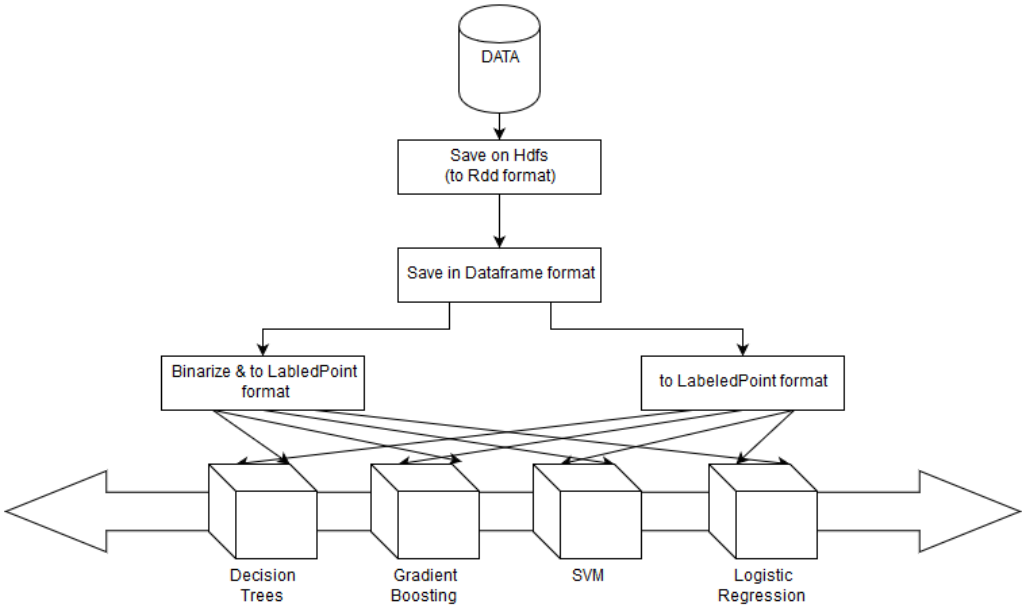
0	1	2	3	4	5	6	7	8	9
rien	une paire	deux paires	brelan	suit	flush	quinte	carré	quinte flush	quinte flush royale

Nous allons donc par le biais de cette problématique, essayer d'induire plusieurs modèles de machine learning. Nous manipulerons donc les trois applications déjà présentes dans le répertoire ../data/poker/app/, nommées PokerApp_GradientBoosting, PokerApp_DecisionsTrees.py, PokerApp_SVM_RegLog.py.

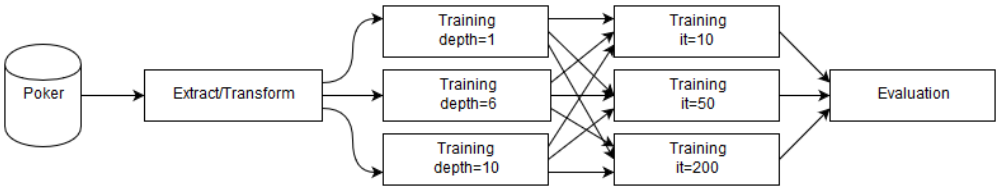
3. "Heads-up limit hold'em poker is solved" writted by Michael Bowling, Neil Burch, Michael Johanson, Oskari Tammelin in Science

6.2 Processus d'analyse

Afin d'avoir une première vue sur le problème et se faire les doigts nous allons lancer dans un premier temps dans la création d'un Arbre de Regression via l'approche CART, afin de prédire nos cibles binaires et discrètes, nous allons ensuite réitérer cette démarche sur trois autres algorithmes de machine learning : Boosting Trees, SVM et Regression Logistique, suite à cela nous comparerons les performances de ceux-ci :



Vous noterez que la création d'une application est très simple et consise, il est donc très facile de parametrer nos modèles, comme le montre par exemple ce schéma :



6.3 Première prise en main et processus d'import des données

Ouvrez le fichier PokerApp_DecisionsTrees.py afin d'obtenir une vue du process que nous utiliserons pour appliquer nos différentes méthodes. Le fichier est composé d'une première partie dans laquelle on importe les librairies utiles, laquelle semble réservée à la génération d'Arbre de décision ? Quel est le rôle de la fonction toLabeledPoint() ? Afin de pouvoir utiliser les données fournies par l'UCI Machine Learning Repository, il nous faut les formater au format LabeledPoint, car la plupart des modèles utilisent ce typage en entrée. Voici à quoi ressemble la bête :

```
        from pyspark.mllib.linalg import SparseVector
        from pyspark.mllib.regression import LabeledPoint

        # Create a labeled point with a positive label and a dense vector.
        LaBetePositive = LabeledPoint(1.0, [1.0, 0.0, 3.0])

        # Create a labeled point with a negative label and a sparse vector.
        LaBeteNegative = LabeledPoint(0.0, SparseVector(3, [0, 2], [1.0, 3.0]))
```

Nos données sont au format ".txt" et sont formatées comme suit :
(rien pour la première main, une paire pour la seconde)

.....
1,1,1,13,2,4,2,3,1,12,0
3,12,3,2,3,11,4,5,2,5,1
.....

Il faudra donc arriver à obtenir quelque chose de cet ordre, c'est à dire :

.....
LabeledPoint(0, SparseVector([1,1,1,13,2,4,2,3,1,12])),
LabeledPoint(1, SparseVector([3,12,3,2,3,11,4,5,2,5]))
.....

Penchons-nous donc sur les fonctions placées avant le main, plus précisément sur toDataFrame() et toLabeledPoint(). La première nous permet de transformer un fichier de données RRD en Dataframe, la seconde nous permet de passer d'un Dataframe en LabeledPoint :

```
from pyspark.sql.functions import col

def toDataframe(data,schema):
    #Split each row ,formatting data & transform to dataframe
    df= data.map(lambda k: k.split(",")).map(lambda p: (int(p[0]), int(p[1]), int(p[2]),
    int(p[3]),    int(p[4]) , int(p[5]), int(p[6]) , int(p[7]), int(p[8]),
    int(p[9]), int(p[10]))).toDF(schema)
    return df

def toLabeledPoint(df):
     #FEATURES
    features= df.map(lambda row: row[0:10])
    #LABELS
    lab= df.map(lambda row: row[10])
    #BIND THE BOTH
    transformedData = lab.zip(features)
    #TO LABELEDPOINT
    Lp = transformedData.map(lambda row: LabeledPoint(row[0],row[1]))
    return Lp
```

Comme nous souhaitons appliquer une binarisation sur nos données, nous avons aussi créer une fonction "filtre" qui modifie chaque target en fonction de sa valeur. Puis une autre fonction qui appliquera le filtre aux données avant de renvoyer un vecteur de LabeledPoints.

```
from pyspark.ml.feature import VectorAssembler
from pyspark.mllib.regression import LabeledPoint

#For filter only if hand have a value
def filterOut(lab):
    if lab >=1 :
            lab=1
    else:
            lab=0
    return lab


def toLabeledPoint2(df):
    features= df.map(lambda row: row[0:10])
    #LABELS
    lab= df.map(lambda row: row[10])
    lab=lab.map(filterOut)
    #BIND THE BOTH
    transformedData = lab.zip(features)
    #TO LABELEDPOINT
    Lp = transformedData.map(lambda row: LabeledPoint(row[0],row[1]))
    return Lp
```

Dans nos trois applications, nous utiliserons donc le même processus de transformation de données, nous pourons ensuite lancer notre classification sur celles-ci :

```
    train_df= toDataframe(train,schema)
    train_df.printSchema()
    test_df= toDataframe(test,schema)
    test_df.printSchema()

    train=toLabeledPoint2(train_df)
    test=toLabeledPoint2(test_df)
    train2=toLabeledPoint(train_df)
    test2=toLabeledPoint(test_df)
```

Maintenant que nous avons soulever le point de l'import des données, nous allons pouvoir lancer nos différents modèles et en mesurer la pertinence.

6.4 Decision Tree Classification

Nous allons donc commencer par lancer notre application générant des arbres de classification (ouvert précédement à titre observatoire), celle-ci va donc créer deux modèles l'un sur les données binaires et l'autre sur la variable cible comprise entre 0 et 9.
Nous définissons à 8 la profondeur maximale de ces arbres, et à 32 le nombre d'individus mimimum sur une feuille (valeur par défault).
Lancez le programme via la commande :
pyspark –driver-memory 2g –executor-memory 2g
c :/spark/spark-1.6.1-bin-hadoop2.6/data/poker/app/PokerApp_DecisionsTrees.py

Lorsque celui-ci a fini de s'éxécuter, allez jeter un oeil dans le dossier "output" du répertoire "poker".
> Qu'y trouvez vous ?
> Que contiennent les fichiers BinaryDecisionTreePredictions.csv et 0-9DecisionTreePredictions.csv ?
> Que contiennent les fichiers 0-9DecisionTree.txt et BinaryDecisionTree.txt ?
> Combien de noeuds possèdent chaque arbre généré ?
> Lequel des deux modèls possèdent le plus bas taux d'erreur ?
> En combien de temps entraîne-t-on nos modèles ?

6.5 Gradient Boosting Classification

Nous réalisons désormais 4 classifications induites par un boosting : deux basées sur des arbres de profondeur 1 (trons) sur nos données binaires et sur les données originales, ainsi que deux autres classifications cette fois ci en définissant notre profondeur maximale à 8. Nous itérons 150 fois dans chacun des cas nos modèles.
Lancez le programme via la commande :
pyspark –driver-memory 2g –executor-memory 2g
c :/spark/spark-1.6.1-bin-hadoop2.6/data/poker/app/PokerApp_GradientBoosting.py

>Combien d'arbres obtient-on pour chaque modèle ?
>Lequels de ces 4 modèles semblent le plus pertinent ?
>Comment pourrait-on augmenter les performances de celui-ci ?

6.6 SVM et Regression Logistique

Afin de clore notre petit tour des méthodes, nous utilisons deux nouveaux algorithmes de MLlib : le SVM et la Regression Logistique. Ces deux algorithmes sont implementés sous la forme d'une descente de gradient. Nous itérons donc 100 fois chacun des modèles entrainés, dans le cas binaire et original de la variable cible. Lancez le programme via la commande :
pyspark –driver-memory 2g –executor-memory 2g
c :/spark/spark-1.6.1-bin-hadoop2.6/data/poker/app/PokerApp_SVM_RegLog.py

Allez ensuite dans le dossier "output" afin de relever les résultats obtenus. Lequels des modèles entaînés (decision trees, boosting, svm, regression logistique) est le plus efficace sur nos données binaires ? Lequel est le plus efficace sur nos données originales ?

6.7 Notes

Nous avons remarqué que l'utilisation de la fonction toLabeledPoint() entraine sur certains PC un dépassement mémoire : dans ce cas vous ne verrez donc aucun résultats dans votre fichier output. Il est donc conseillé de parraleliser le calcul de cette fonction sur plusieurs noeuds afin :

```
train=sc.parallelize(train_df.toLabeledPoint().collect(),4)
test=sc.parallelize(test_df.toLabeledPoint().collect(),4)
```

7 Conclusion

Par le biais de l'application de clustering de morceaux de Bach, vous avez pu apprendre à manipuler pour la première fois Spark : import de données, lancement d'un modèle et paramétrisation, et l'export des résultats. Ceci nous aura permis aussi de réutiliser certaines fonctionnalités de visualisation de données offertes par Matplot (MLlib n'en fournissant pas). Enfin nous avons pu par le biais de l'induction des règles du Poker, introduire un processus de tranformation de données plus complexe et surtout nous avons pu faire le tour de quelques modèles inclus dans la librairies et en comparer la performance dans le cas de données binaire ou qualitative ordinale. Et nous avons pu comprendre que les librairies python peuvent être utiliser en apport des manques actuels de MLlib (lors de certains preprocessing, exports...). Vous avez donc maintenant de bonnes bases pour commencer à utiliser Spark, ce qui est une bonne chose : la solution a le vent en poupe (et depuis 2014, est plus recherchée sur Google que Map-Reduce, hasard ou franc-maçon ?), la communauté d'entreprises sautant le pas s'aggrandit. La boite à outils a en effet de quoi séduire : personnalisable, libre, gestion des pannes,... et s'intègre nativement dans un environnement hadoop grandissant. Cependant, il est vrai que maîtriser l'outil et son framework (ligne de commandes, installation pénible avec de nombreuses dépendances à résoudre, lenteur des calculs sur un PC personnel) demanderons aux utilisateurs d'avoir un bagage en informatique solide. Enfin, ses atouts techniques en font aujourd'hui, avec le retentissant son du big data, un investissement de choix pour les entreprises. La version 2 de Spark est annoncée prochainement [4], ce qui pourrait faire grandir en maturité la solution, affaire à suivre donc... Si vous souhaitez continuer à prendre en main Spark sous Python, n'hesitez donc pas à vous ruer sur les sources et autres supports mis à dispositions dans la partie "pour aller plus loin". En vous remerciant pour votre lecture !

8 Pour aller plus loin...

Ici sont regroupés par thématique principales les differentes références entourant ce sujet, vous pourrez de plus facilement prendre en main les fonctions Spark, grâce aux nombreux exemples fournis dans le repertoire source de Spark : bin/examples/src/main/python

Coté data :

>Data libsvmtools
>Data UCI

Documentations Spark Hadoop :

>Introduction à Map Reduce
>Guide de prise en main Spark Linux (notament un point important la création de l'environement CDH5)
>Instalation d'un cluster Spark sous Linux
>Instalation de CDH5 sous Linux (gestionnaire de mémoire)
>Du data set Hdfs au Rdd

Prise en main :

>Maitriser le csv sans installer de packages
>6 étapes pour bien commencer
>Super cours sur Spark, mais en Scala
>Manipulations basiques de données sous Spark

Ipython :

>Ipython l'API de developement pour Spark
>IPython avec Spark

Streaming Twitter :

>Collecte de tweets et storing

4. http ://fr.slideshare.net/databricks/2016-spark-summit-east-keynote-matei-zaharia

Cassandra et connexion avec autres briques d'Hadoop :

>Streaming big data 1
>Spark streaming with python and kafka >Streaming Big Data 2
>Simple Serveur Cassandra
>Introduction to Spark with Cassandra
>Initialize Cassandra

Connecteur Cassandra :

>https ://github.com/TargetHolding/pyspark-cassandra
>https ://github.com/datastax/spark-cassandra-connector

SOURCES :

Les essentiels :

>La documentation officielle d'Apache
>The Elements of Statistical Learning de Travor Hastie, Robert Tibshirani et Jérome Friedman
>Plateforme documentaire et pédagogique de M Rikotomalala

Support de cours :

>Machine Learning with Spark ML/MLlib du blogger data-scientist chinois ybliang8@gmail.com
>Edvanced Big Data Analytics Lectures - University of Colombia de Ching-Yung Lin
>Wikistat : Introduction à l'utilisation de MLlib de Spark avec l'API pyspark
>Introduction to Boosted Trees Tianqi Chen
>Weighted Least Squares Estimation with Sampling Weights de Hee Choon Shin