Faire du ML avec des données massives : Introduction à Apache Spark (suite)

Table of Contents

Partie III : Construction d'un arbre de décision avec la bibliothèque MLlib de PySpark

L'objectif de cette partie est de familiariser les étudiants à la bibliothèque MLlib de PySpark avec la construction d'un arbre de décision.
Dans ce TP, les étudiants apprendront à charger des données, à les prétraiter et à construire un arbre de décision à l'aide de la bibliothèque MLlib de PySpark.

Rappel sur le principe général des arbres de décision

Les arbres de décision font partie de la catégorie des algorithmes d'apprentissage supervisés. Ils vont nous permettre d'expliquer ou prédire une variable qualitative (classification) ou quantitative (régression).
L'algorithme permettant la construction de l'arbre fonctionne selon le principe de diviser-pour-règner récursif. C'est à dire que son but est de créer des groupes d'observations aussi homogènes que possible au regard de la variable que l'arbre cherche à prédire. Cela se fait en posant une succession de questions binaires (oui/non) sur des variables que l'on aura définit comme pertinentes. Ces variables constitueront les variables explicatives. Les réponses à ces questions constitueront les branches de notre arbre.

L'environnement de travail

Suite aux séances précédentes, PySpark devrait normalement déjà être installé et configuré.

1. Installer les bibliothèques py4j et MLlib.

Vous pouvez faire cela directement dans une cellule jupyter en utilisant la commande :
`pip install mllib` (l'utilitaire pip doit bien être préalablement installé sur votre poste. Le lien suivant peut vous y guider). Lancer la même commande `pip` pour py4j.

2. Dans une cellule, importer les méthodes :
   - LabeledPoint de pyspark.mllib.regression
   - DecisionTree de pyspark.mllib.tree
   - SparkConf et SparkContext de pyspark

- array de numpy

3. Configurer votre contexte spark avec les lignes de codes suivantes :

```
conf = SparkConf().setMaster('local').setAppName('SparkDecisionTree')
sc = SparkContext(conf=conf)
```

Le contexte est l'environnement dans lequel Spark sera exécuté. Cette partie indique à Spark comment il doit prendre en charge la distribution des ressources et des traitements à travers le cluster.
Paramétrer le master (SetMaster) à local signifie que tout sera exécuté localement car nous n'avons pas de cluster sous la main.
La méthode setAppName permet de donner un nom à notre application, lorsque celle-ci est lancée dans une console Spark, nous pourrons voir le nom s'afficher. Vous pouvez mettre le nom que vous souhaitez.

Chargement et exploration des données

Le fichier à partir duquel sera construit l'arbre de décision est le csv 'PastHires.csv'.
L'arbre de décision que nous allons construire doit prédire si un candidat sera embauché ou non en fonction d'un certain nombre de caractéristiques (ou attributs) fournies.

4. Charger les données brutes à partir du fichier CSV fournit dans le TP en utilisant la méthode PySpark appropriée et afficher le RDD.
5. Effectuer une exploration préliminaire des données pour les comprendre
    a. Dans une cellule markdown, décrire les données, que représentent-elles ? décrire les différents attributs ainsi que les valeurs qu'ils prennent.
    b. Nous avons besoin que toutes les valeurs qualititaves soient numériques, décrire dans la cellule markdown, comment vous procéderiez pour transcrire les valeurs qualitatives en valeurs numériques.

Prétraitement des données

1. Supprimer les en-têtes inutiles des données.
    a. Pour faire cela, vous pouvez dans un premier temps, isoler la ligne d'entête dans une variable header à l'aide de la méthode first() puis filtrer sur les lignes différentes du header
2. Convertir les données catégorielles en données numériques.

   Lorsqu'on observe nos attributs, nous voyons que les colonnes qualitatives sont de 2 types : booléennes (ce sont les colonnes Employed ?, Top-tier school, Interned et Hired) et catégorielles (c'est la colonne Level of education)
    a. Écrire une fonction nommée binary qui permet de convertir une variable qualitative booléenne (Y/N) en valeur numérique (0/1)
    b. Écrire une deuxième fonction nommée mapEducation qui permet de convertir une variable qualitative prenant 3 valeurs ('BS', 'PhD' et 'BS') en valeur numérique (1, 2 ou 3). La fonction doit également retourner le chiffre 0 si elle ne reçoit aucune des valeurs précédents
3. Diviser chaque ligne du RDD sur le délimiteur ','.
4. Créer des 'points étiquetés' pour chaque ligne de notre RDD

La bibliothèque MLlib utilise une structure de données particulière appelée 'point étiquetés' en anglais 'labeledPoint'. Ce type de données est spécifique aux algorithmes d'apprentissage dans MLlib et il associe un "label" qui est un réel (la valeur à prédire) à un vecteur (vecteur

d'observations) . Ce "label" correspond soit à la valeur de la variable y quantitative que l'on cherche à prédire en régression, soit à un code de classe : 0.0, 1.0... en classification.

a. Ecrire une fonction nommée createLabeledPoints qui prend en entrée une ligne du RDD et qui retourne une structure de données de type 'point étiqueté' grâce à la méthode LabeledPoint de pyspark.mllib.regression. Attention, tous nos champs doivent être numérisés, vous pouvez donc faire usage des 2 fonctions que vous avez écrites précédemment pour convertir vos colonnes qualitatives en colonnes numériques.
b. Convertir les lignes de votre RDD en labeldPoints et les ranger dans un nouveau RDD nommé trainingData, c'est notre jeu d'apprentissage.

Dans le cadre de cet exercice, nous ne ferons pas de séparation entre jeu de test et jeu d'apprentissage. Nous créerons plutôt un ensemble de test qui ne contiendra qu'un seul candidat.

c. Créer un candidat test, avec 10 ans d'expérience, actuellement employé, 3 employeurs précédents, un diplôme de licence, mais d'une école qui n'est pas de premier plan et où il n'a pas fait de stage.
d. Utiliser la méthode parallelize de Spark qui permet de convertir un tableau numpy en un RDD. Dans notre cas, le tableau numpy ne contiendra qu'une seule ligne. Il peut quand même être transformé en RDD.

Construction de l'arbre de décision

1. Utiliser la méthode pySpark DecisionTree.trainClassifier pour construire un arbre de décision à l'aide des données prétraitées.

Lire la documentation de la méthode (voir lien) et configurer les paramètres de votre arbre de décision (profondeur maximale, critère de division (impurity), etc.).

Effectuer une prédiction

2. A l'aide des méthodes predict() et collect(), effectuer une prédiction sur votre RDD de test. N'hésitez pas à vous appuyer sur la documentation pour vous guider sur la syntaxe.
3. Afficher la prédiction. Quelle classification obtenez-vous ?
4. Afficher l'arbre de décision lui-même à l'aide de la méthode toDebugString()
5. Dans une cellule markdown, décrire le cheminement de l'arbre qui a permis d'arriver à cette prédiction

Pour aller plus loin

1. Au lieu de créer par vous-même les valeurs de tests, séparer le jeu de données en un ensemble de test et d'entrainement
2. Construisez votre arbre sur les données d'entrainement et évaluer les performances de l'arbre de décision sur l'ensemble de test.
3. Utiliser des mesures d'évaluation telles que la précision, le rappel et le f-score pour évaluer la qualité du modèle.

N'hésitez pas à consulter la documentation sur les métriques d'évaluation de pySpark mllib pour vous guider.