MS BGD: Spark TP 2 (2018-2019)

Début du projet et pre-processings

Nous allons commencer le projet guidé de data-science, sur lequel nous travaillerons jusqu'à la fin de ce module. Dans le TP 2, nous allons travailler sur le preprocessing des données. Le preprocessing occupe une place importante dans un projet de data science : il s'agit de faire en sorte que les données soient correctement formatées pour l'entraînement du modèle, et parfois de l'aider en ajoutant de l'information (feature-engineering).

Intro:

- Objectif: Réaliser un modèle capable de prédire si une campagne Kickstarter va atteindre son objectif ou non.

- Données: Les données sont disponibles sur kaggle.com, avec leur description: https://www.kaggle.com/codename007/funding-successful-projects

    - Commencez par lire la description des données et la description de chaque colonne.
    - Téléchargez le dataset pré-cleané train_clean.csv (pour jouer en mode "ultra nightmare": train.csv).

Nous allons utiliser l'IDE IntelliJ qui permet de développer des projets de data science en Spark/scala de façon plus efficace qu'avec le spark-shell. Le Shell reste toutefois utile pour tester des lignes de codes ou pour interagir rapidement avec les données.

I - Setup:

INSTALLATION DE INTELLIJ => Suivre la section **TP 2** dans le Setup.

Charger le template de projet dans IntelliJ

Un template de projet Spark vous est fourni. Commencez par le télécharger, puis décompressez-le. Pour l'importer dans IntelliJ:

- Ouvrez IntelliJ, dans la page d'accueil cliquez sur "import project".
- Sélectionnez le chemin vers le projet décompressé.
- Sélectionnez "import project from external model", et sélectionner SBT
- => click next
- Si "project sdk" est vide cliquez sur new, intelliJ devrait directement ouvrir l'arborescence vers votre installation de java.
- Sélectionnez "use auto import" (Si cette option n'est pas disponible, puisqu'elle a été retirée de la dernière version d'IntelliJ, continuez sans rien cocher)
- => click finish
- SBT project data to import : vérifiez que les deux dossiers sont bien sélectionnés => click OK
- Attendre qu'intelliJ charge le projet et ses dépendances

Soumettre un script à Spark

Pour exécuter un script Spark/scala, il faut ensuite le compiler et en faire un "jar" i.e. un fichier exécutable sur la machine virtuelle java. La compilation d'un script en scala se fait avec SBT (équivalent de maven pour java). Et l'exécutable doit ensuite être lancé sur le cluster Spark via la commande spark-submit.

Pour simplifier les choses un script bash **build_and_submit.sh** vous est fourni dans le template de projet Spark. Pour que ce script fonctionne, vous devez avoir le dossier **spark-2.2.0-bin-hadoop2.7** décompressé dans votre répertoire **HOME.**

(Pour voir plus en détail la procédure complète pour soumettre un Job à un cluster Spark, reportez-vous à la dernière section du document Setup.)

Assurez-vous que le fichier build_and_submit.sh soit exécutable en entrant dans un terminal

```
> cd  /chemin/vers/projet/spark
> chmod +x build_and_submit.sh
```

Il vous suffit ensuite pour lancer votre job spark d'ouvrir un terminal et de faire:

`> cd /chemin/vers/projet/spark` (sauf si vous êtes déjà dans le répertoire où se trouve le script) `> ./build_and_submit.sh Preprocessor` . Cette commande compile le code, construit le jar, puis exécute le job “Preprocessor” sur une instance Spark temporaire sur votre machine (créée juste pour l’exécution du script). Les outputs de votre script (les println, df.show(), etc) sont affichés dans le terminal.

Vous pouvez ouvrir un terminal dans IntelliJ (icône en bas à gauche) pour éviter d’avoir trop de fenêtres ouvertes en même temps. Vous pouvez créer des onglets dans le terminal (icône +): un onglet pour la commande sbt assembly, et un terminal pour le spark-submit.

II - Début du TP:

Allez dans l’arborescence du projet : src/main/scala/com.sparkProject. Vous devez voir deux objets: “Preprocessor” et “Trainer”. **Nous allons coder dans “Preprocessor”**, la partie Trainer servira pour le TP 3. Le but du TP 2 est de préparer le dataset, c’est-à-dire essentiellement nettoyer les données, créer de nouveaux features et traiter les valeurs manquantes.

Vous continuerez à coder dans la fonction “main” qui se trouve dans Preprocessor.

Pour compiler et lancer le script faites la commande `./build_and_submit.sh Preprocessor` dans un terminal, à la racine du projet.

1. Chargement des données

L’ensemble des ressources nécessaires pour les prochaines questions se trouvent dans la documentation de Spark.

a. Charger le fichier csv dans un dataFrame. La première ligne du fichier donne le nom de chaque colonne, on veut que cette ligne soit utilisée pour nommer les colonnes du dataFrame.

b. Afficher le nombre de lignes et le nombre de colonnes dans le dataFrame.

c. Afficher un extrait du dataFrame sous forme de tableau.

d. Afficher le schéma du dataFrame (nom des colonnes et le type des données contenues dans chacune d’elles).

e. Assigner le type “Int” aux colonnes qui vous semblent contenir des entiers.

2. Cleaning

Certaines opérations sur les colonnes sont déjà implémentées dans Spark, mais il est souvent nécessaire de faire appel à des fonctions plus complexes. Dans ce cas on peut créer des UDF (User Defined Functions) qui permettent d'implémenter de nouvelles opérations sur les colonnes. Voir le document spark_notes.

a. Afficher une description statistique des colonnes de type Int (avec .describe().show)
b. Observer les autres colonnes, et proposer des cleanings à faire sur les données: faites des groupBy count, des show, des dropDuplicates. Quels cleaning faire pour chaque colonne ? Y a-t-il des colonnes inutiles ? Comment traiter les valeurs manquantes ? Des "fuites du futur" ???

c. enlever la colonne "disable_communication". cette colonne est très largement majoritairement à "false", il y a 322 "true" (négligeable) le reste est non-identifié.

d. Les fuites du futur: dans les datasets construits a posteriori des évènements, il arrive que des données ne pouvant être connues qu'après la résolution de chaque évènement soient insérées dans le dataset. On a des fuites depuis le futur ! Par exemple, on a ici le nombre de "backers" dans la colonne "backers_count". Il s'agit du nombre de personnes FINAL ayant investi dans chaque projet, or ce nombre n'est connu qu'après la fin de la campagne. Il faut savoir repérer et traiter ces données pour plusieurs raisons:
    i. En pratique quand on voudra appliquer notre modèle, les données du futur ne sont pas présentes (puisqu'elles ne sont pas encore connues). On ne peut donc pas les utiliser comme input pour un modèle.
    ii. Pendant l'entraînement (si on ne les a pas enlevées) elles facilitent le travail du modèle puisque qu'elles contiennent des informations directement liées à ce qu'on veut prédire. Par exemple, si backers_count = 0 on est sûr que la campagne a raté.

    Ici, pour enlever les données du futur on retir les colonnes "backers_count" et "state_changed_at".

e. On pourrait penser que "currency" et "country" sont redondantes, auquel cas on pourrait enlever une des colonne. Mais en y regardant de plus près:
    i. dans la zone euro: même monnaie pour différents pays, il faut donc garder les deux colonnes.
    ii. il semble y avoir des inversions entre ces deux colonnes et du nettoyage à faire en utilisant les deux colonnes. En particulier on peut remarquer que quand country= False le country à l'air d'être dans currency. On le

voit avec la commande: `df.filter($"country"===”False”).groupBy("currency").count.orderBy($"count".desc).show(50)`

Créer deux udfs udf_country et udf_currency telles que:

- Udf_country : si country=False prendre la valeur de currency, sinon si country est une chaîne de caractères de taille autre que 2 remplacer par null, et sinon laisser la valeur country actuelle. On veut les résultat dans une nouvelle colonne **“country2”**.
- Udf_currency: si currency.length != 3 currency prend la valeur null, sinon laisser la valeur currency actuelle. On veut les résultats dans une nouvelle colonne **“currency2”**.

f. Pour une classification, l’équilibrage entre les différentes classes cibles dans les données d’entraînement doit être contrôlé (et éventuellement corrigé). Afficher le nombre d’éléments de chaque classe (colonne **final_status**).

g. Conserver uniquement les lignes qui nous intéressent pour le modèle: **final_status = 0 (Fail) ou 1 (Success)**. Les autres valeurs ne sont pas définies et on les enlève. On pourrait toutefois tester en mettant toutes les autres valeurs à 0 en considérant que les campagnes qui ne sont pas un Success sont un Fail.

3. Ajouter et manipuler des colonnes

Il est parfois utile d’ajouter des “features” (colonnes dans un dataFrame) pour aider le modèle à apprendre. Ici nous allons créer de nouvelles features à partir de celles déjà présentes dans les données. Dans certains cas on peut ajouter des features en allant chercher des sources de données supplémentaires.

Les dates ne sont pas directement exploitables par un modèle sous leur forme initiale dans nos données: il s’agit de timestamps Unix (nombre de secondes depuis le 1er janvier 1970 0h00 UTC). Nous allons traiter ces données pour en extraire des informations pour aider les modèles. Nous allons, entre autres, nous servir des fonctions date dans *org.apache.spark.sql.functions*.

a. Ajoutez une colonne **days_campaign** qui représente la durée de la campagne en jours (le nombre de jours entre “launched_at” et “deadline”).

b. Ajoutez une colonne **hours_prepa** qui représente le nombre d’heures de préparation de la campagne entre “created_at” et “launched_at”. On pourra arrondir le résultat à 3 chiffres après la virgule.

c. Supprimer les colonnes “launched_at”, “created_at” et “deadline”, elles ne sont pas exploitables pour un modèle.

Pour exploiter les données sous forme de texte, nous allons commencer par réunir toutes les colonnes textuelles en une seule. En faisant cela, on rend indiscernable le texte du nom de la campagne, de sa description et des keywords, ce qui peut avoir des conséquences sur la qualité du modèle. Mais on cherche à construire ici un premier benchmark de modèle, avec une solution simple qui pourra servir de référence pour des modèles plus évolués.

d. Ajoutez une colonne **text,** qui contient la concaténation des Strings des colonnes “name”, “desc” et “keywords”. ATTENTION à bien mettre des espaces entre les chaînes de caractères concaténées, car on fera plus un split en se servant des espaces entre les mots.

4. Valeurs nulles

Il y plusieurs façons de traiter les valeurs nulles pour les rendre exploitables par un modèle. Nous avons déjà vu que parfois les valeurs nulles peuvent être comblées en utilisant les valeurs d’une autre colonne (parce que le dataset a été mal préparé). On peut aussi décider de supprimer les exemples d’entraînement contenant des valeurs nulles, mais on risque de perdre beaucoup de données. On peut également les remplacer par la valeur moyenne ou médiane de la colonne. On peut enfin leur attribuer une valeur particulière, distincte des autres valeurs de la colonne.

a. Remplacer les valeurs nulles des colonnes "days_campaign”, "hours_prepa", "goal" par la valeur -1 et par “unknown” pour les colonnes country2 et currency2.

5. Sauvegarder un dataFrame

Sauvegarder le dataFrame au format **parquet** sur votre machine.

Attention ! Lorsqu’on sauvegarde un output en Spark, le résultat est toujours un répertoire contenant un ou plusieurs fichiers. Cela est dû à la nature distribuée de Spark.