TP 2 : 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 ce tp 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).

Introduction

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

Données: Les données sont disponibles sur kaggle.

Commencez par lire la description des données et la description de chaque colonne en cliquant sur *train.csv* dans la partie *Data Sources*. Téléchargez le dataset pré-cleané *train_clean.csv* présent dans le dossier *data* (pour jouer en mode "ultra nightmare" utilisez plutôt le fichier *train.csv*).

Début du TP

- **Chargement des données**

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

1. Chargez le fichier *train_clean.csv* dans un *DataFrame*. La première ligne du fichier donne le nom de chaque colonne (aka le header), on veut que cette ligne soit utilisée pour nommer les colonnes du dataFrame :

2. Affichez le nombre de lignes et le nombre de colonnes dans le DataFrame :

3. Affichez un extrait du DataFrame sous forme de tableau :

4. Affichez le schéma du DataFrame, à savoir le nom de chaque colonne avec son type :

5. Assignez le type *Int* aux colonnes qui vous semblent contenir des entiers :

- **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 *UDFs* (*User Defined Functions*) qui permettent d'implémenter de nouvelles opérations sur les colonnes. Voir la partie User Defined Functions du fichier spark_notes.md pour comprendre comment ça fonctionne.

```
dfCasted
  .select("goal", "backers_count", "final_status")
  .describe()
  .show
```

Observez les autres colonnes, posez-vous les bonnes questions : quel cleaning faire pour chaque colonne ? Y a-t-il des colonnes inutiles ? Comment traiter les valeurs manquantes ? A-t-on des données dupliquées ? Quelles sont les valeurs de mes colonnes ? Des répartitions intéressantes ? Des "fuites du futur" (vous entendrez souvent le terme *data leakage*) ??? Proposez des cleanings à faire sur les données : des *groupBy-count*, des *show*, des *dropDuplicates*, etc.

1. Enlevez la colonne *disable_communication*. Cette colonne est très largement majoritairement à *false*, il n'y a que 322 *true* (négligeable), le reste est non-identifié :

➢ *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 total de personnes 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 :

- pendant l'entraînement (si on ne les a pas enlevées) elles facilitent le travail du modèle puisqu'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.
- au moment d'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.

Ici, pour enlever les données du futur on retire les colonnes *backers_count* et *state_changed_at* :

```
val dfNoFutur = df2.drop("backers_count", "state_changed_at")
```

➢ *Colonnes currency et country*

On pourrait penser que les colonnes *currency* et *country* sont redondantes, auquel cas on pourrait enlever une des colonnes. Mais c'est oublier par exemple que tous les pays de la zone euro ont la même monnaie ! Il faut donc garder les deux colonnes.

Il semble y avoir des inversions entre ces deux colonnes et du nettoyage à faire. On remarque en particulier que lorsque `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)
```

Créez deux udfs nommées *udf_country* et *udf_currency* telles que :

- *cleanCountry* : 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 résultats dans une nouvelle colonne *country2*.

- *cleanCurrency* : si `currency.length != 3` currency prend la valeur *null*, sinon laisser la valeur currency actuelle. On veut les résultats dans une nouvelle colonne *currency2*.

On a montré ici l'utilisation d'udfs, mais de façon générale toujours privilégier les fonctions déjà codées dans Spark car elles sont optimisées.

Pour une classification, l'équilibrage entre les différentes classes cibles dans les données d'entraînement doit être contrôlé (et éventuellement corrigé). Affichez le nombre d'éléments de chaque classe (colonne *final_status*). Conservez uniquement les lignes qui nous intéressent pour le modèle, à savoir lorsque final_status vaut 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.

- **Ajouter et manipuler des colonnes**

Il est parfois utile d'ajouter des *features* (colonnes dans un DataFrame) pour aider le modèle lors de son apprentissage. 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 liées aux dates de l'objet functions.

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

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.

Supprimez 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.

Mettre les colonnes *name*, *desc*, et *keywords* en minuscules.

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 par la suite un split en se servant des espaces entre les mots.

- ***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.

Remplacez les valeurs nulles des colonnes *days_campaign*, *hours_prepa*, et *goal* par la valeur -1 et par *"unknown"* pour les colonnes *country2* et *currency2*.

- **Sauvegarder un DataFrame**

Sauvegarder le DataFrame final au format parquet sur votre machine :

```
monDataFrameFinal.write.parquet("/path/ou/les/donnees/seront/sauvegardees")
```

```
scala> dfContry.write.parquet("C:/Cours_Caplogy_Data/Cours_Spark_Scala/TP2/sauvegarde")
```

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.