Outils Data : Comment améliorer leur accessibilité pour les populariser ?

Enzo Bonnal

2018

Maître d'apprentissage : Bruno Kauffmann
Tutrice enseignante : Fatiha Zaïdi

Mémoire de première année de Master d'Informatique pour la Science des Données

Université Paris Saclay

Entreprise d'Accueil : Orange

Sommaire

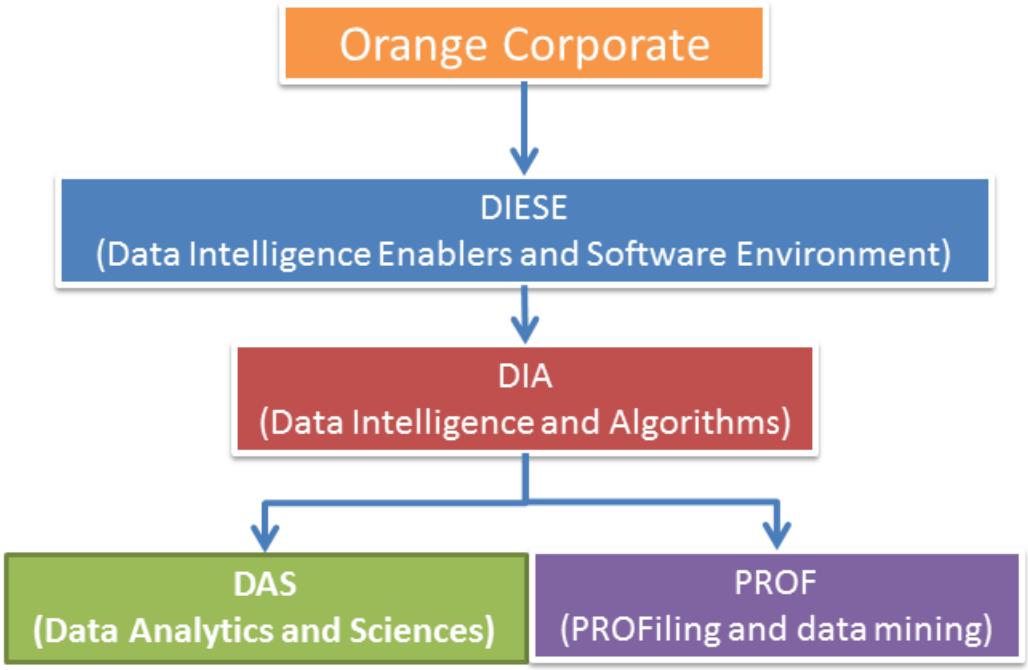
Sommaire

Remerciements

Avant toute chose je tiens à remercier Bruno Kauffmann pour son soutient tout au long de cette première année d'apprentissage et pour ses conseils précieux au quotidien, notamment pour la rédaction de ce mémoire.
Je tiens également à remercier Fatiha Zaidi pour son suivi en tant que tutrice universitaire. Je remercie également toute l'équipe d'enseignants du master ISD qui nous fait, avec patience, profiter de son expertise.
Un grand merci à Nebojsa et Alexis aux côtés de qui j'ai pu travailler et progresser tout au long de l'année. Un grand merci également à tous ceux qui, en plus des personnes déjà citées, participent aux rires quotidiens : Pierre, Ali, Eole, Shengda, Ilies, Riu et Megah ainsi qu'à tous les autres membres de DIESE que j'ai pu croiser pour leur gentillesse générale.

Introduction

Environnement



J'effectue mon alternance dans la direction DIESE (Data Intelligence Enablers and Software Environment), dans les locaux de Châtillon baptisés "Orange Gardens". DIESE est une direction d'Orange Coporate qui est l'entité qui chapeaute tout le groupe Orange, notamment Orange France et les autres filiales d'Orange dans le monde ou encore Orange Business Service. L'objectif de DIESE est de fournir l'ensemble des ressources techniques, des services, des expertises et de l'accompagnement permettant à ces différentes entités de :

1. Traiter massivement l'ensemble des données disponibles,
2. Mesurer en prédictif et en temps réel l'expérience client,

3. Déployer en continu et en production tous les développements,
4. Fournir des environnements de travail simples efficaces et favorisant l'agilité,
5. Garantir une sécurité renforcée et de bout en bout des données.

Au sein de DIESE j'ai rejoint le département DIA (Data Intelligence and Algorithms) qui contient cinq équipes dont deux sont notables pour la suite de ce mémoire :

- Mon équipe : DAS (Data Analytics and Sciences), dirigée par mon maître d'apprentissage Bruno Kauffmann. C'est une équipe de Data Scientists et Data Engineers qui ont un rôle d'expertise sur les projets de Data Science de toutes les entités du groupe.
- PROF (PROFiling and data mining) : C'est une équipe composée de chercheurs et d'ingénieurs de recherche dans le domaine du machine learning.

J'évolue donc dans un contexte très riche, entouré d'experts enthousiastes.

Mission

Au sein de DAS mon rôle est d'assister les membres de l'équipe dans leurs divers projets. J'ai pendant cette première année pu participer à deux projets qui ont abouti avec succès :

- Un outil de pseudonymisation de données structurées
- Une library Python autour d'un outil de machine learning interne

J'ai participé à toutes les étapes de production d'un produit : de la caractérisation du besoin à la mise en production.

L'objectif dans mes deux projets a été d'améliorer l'accessibilité à des technologies afin de les rendre utilisables par le plus grand nombre. Dans le premier projet, il a été question de mettre à la portée de presque n'importe qui la puissance du framework *Apache Hadoop MapReduce* pour pseudonymiser des données. Le deuxième projet a quant à lui permis de rendre plus souple d'utilisation et plus attractif pour un certain public un logiciel de Machine Learning nommé *Khiops* et développé par l'équipe PROF présentée précédemment.

Durant cette première année j'ai découvert des aspects du métier de Data Engineer. Des missions avec une composante Data Science me seront sans doute confiées lorsque mon bagage de statistiques et de machine learning me le permettra au cours de l'année prochaine.

1

Outil de pseudonymisation de données structurées

Entrons dans le vif du sujet avec une première partie détaillant le premier projet auquel j'ai participé. J'ai travaillé sur ce projet de septembre à février sous les conseils et en collaboration avec mon collègue Data Engineer Nebojsa Topolscak, projet suivi de près comme à son habitude par mon maître d'apprentissage Bruno Kauffmann.

1.1 Contexte et besoin

Rappelons rapidement les définitions des termes *anonymisation* et *pseudonymisation* d'après un article du site du CNRS [1]:

L'anonymisation des données détruit toute possibilité de pouvoir identifier à quel individu appartiennent les données personnelles. Ce processus consiste à modifier le contenu ou la structure des données en question afin de rendre la ré-identification des personnes quasi impossible, même après traitement.

La pseudonymisation est le traitement de données à caractère personnel de telle façon que celles-ci ne puissent plus être attribuées à une personne concernée précise sans avoir recours à des informations supplémentaires, pour autant que ces informations supplémentaires soient conservées séparément et soumises à des mesures techniques et organisationnelles afin de garantir que les données à caractère personnel ne sont pas attribuées à une personne physique identifiée ou identifiable.

DIESE a été souvent sollicité pour de l'assistance à la pseudonymisation mais dans le contexte de nouvelle réglementation que constitue la mise en application le 25 Mai dernier

du Règlement Général sur la Protection des Données, les demandes devenaient de plus en plus fréquentes. Dans la majorité des cas ces demandes portaient sur la pseudonymisation de données structurées[1] de type CSV potentiellement massives.

Les clients potentiels sont des départements de DIESE ou des entités de filiales d'Orange qui ne disposent pas d'outil pour réaliser une pseudonymisation ou qui n'ont pas la capacité technique pour traiter de gros volumes. Nous avons commencé par répondre aux requêtes au cas par cas de façon personnalisée mais cette démarche n'était pas satisfaisante puisque trop chronophage dans le cas d'une forte demande. L'idée de Bruno a alors été de donner un outil aux clients pour les rendre autonomes. Le cahier des charges que doit remplir l'outil est le suivant:

- Être distribué pour passer à l'échelle
- Traiter tout type de fichier CSV
- Être facilement déployable sur les plateformes des clients
- Être suffisamment simple pour être utilisé par des employés peu techniques

1.2 État de l'art

Nous allons faire ici un bref tour d'horizon des technologies disponibles pour nous rendre compte que notre cahier des charges nécessitait le développement d'un nouvel outil.

Les solutions propriétaires
De nombreuses solutions propriétaires existent mais elles présentent souvent les mêmes désavantages. De plus, le fait qu'elles soient propriétaires est déjà problématique en soi. Nous allons en voir deux exemples assez représentatifs :

TeskaLabs a mis au point une solution nommée TurboCat.io() [3] qui présente les avantages d'être très modulable et compatible avec une très grande variété d'environnements. Il n'est cependant pas distribué et ne peut donc pas passer à l'échelle du *Big Data*.

La deuxième solution étudiée est Talend [2] qui est une solution puissante et tout en un qui gère le stockage, la pseudonymisation et le traitement des données mais qui ne permet pas non plus un traitement distribué. De plus cette solution est assez lourde à mettre en place, ce qui est rédhibitoire pour un déploiement dans les filiales.

[1] par opposition aux données brutes de type logs

Les solutions open source
Les solutions open source les plus pertinentes sont l'utilisation de projets de la fondation *Apache* tels que *Hive* ou *Pig* qui sont des surcouches d'*Hadoop* assez haut niveau.

Cependant ces technologies ne sont pas assez abstraites pour qu'un collaborateur peu technique les appréhende facilement. En effet, au-delà de la syntaxe non triviale, l'utilisation de ces technologies nécessite la définition de *User Defined Functions* (UDF) pour réaliser toutes les actions dont nous avons envie de disposer dans un outil de pseudonymisation, comme le *hashing* qui n'est par exemple pas nativement supporté dans *Pig*.

Avec la même complexité d'utilisation nous pourrions envisager l'écriture de scripts *Python* utilisant *PySpark*.

1.3 Résolution de la problématique

Nous allons maintenant passer en revue les différentes étapes qui nous ont menés jusqu'à l'élaboration de la solution opérationnelle baptisée *Pseudonymizer*.

1.3.1 Les premiers jobs customisés

Ma première tâche au sein de l'équipe a été de participer à l'apport d'une réponse au cas par cas aux besoins de pseudonymisations. La solution était d'écrire un job Hadoop MapReduce en Java 7 personnalisé pour chaque type de traitement. La méthode *map* avait cette allure :

```
public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
// skip header (first line)
if (key.get() == 0) {return;}
//split line on separator FIELD_SEPARATOR
String[] splittedLine = value.toString().split(FIELD_SEPARATOR);
//Line process :
StringBuffer modifiedLine=new StringBuffer();
//Here we take the first 5 characters of the first field in clear and we hash
    the other part with SHA256
modifiedLine.append(splittedLine[0].substring(0,5) +
HasherSha256.getHashAsBase64(seed,splittedLine[0].substring(5,
    splittedLine[0].length())));
//We add a seperator
modifiedLine.append(FIELD_SEPARATOR);
//And so on for every field to process :
[...]
//the map returns modifiedLine by writing it to MapReduce context
```

```
context.write(new Text(modifiedLine), NullWritable.get());
}
```

Cette méthode devait être réécrite pour toute nouvelle demande en dialoguant avec le client afin de se mettre d'accord sur les actions à effectuer. Ensuite une fois notre job écrit et empaqueté dans un .jar, nous l'uploadions et l'exécutions sur le cluster de stockage et de calcul pour le Big Data géré par notre direction DIESE et nommé *DIOD*. Le client pouvait ensuite accéder à ses données pseudonymisées sur la plateforme.

Ce processus avait le gros désavantage de solliciter notre équipe pour un service à faible valeur ajoutée, potentiellement chronophage et pouvant être source d'erreurs d'incompréhension. Ces différents points ont donc justifié l'élaboration de l'outil qui fait l'objet de la section suivante.

1.3.2 Du premier prototype d'outil à la version industrielle

1.3.2.1 Le fichier de configuration

Bruno a lancé l'idée de l'utilisation d'un *fichier de configuration*. L'utilisateur doit y renseigner les actions qu'il désire exécuter sur ses données et l'outil doit faire le reste du travail automatiquement.

Il y a eu de nombreuses discussions auxquelles j'ai participé pour décider de la syntaxe à adopter pour ce fichier. Il fallait trouver une syntaxe simple et intuitive même pour un non développeur. Différentes tentatives ont été faites pour équilibrer la balance entre intuitivité et possibilités. Nous avons finalement décidé de nous inspirer de la syntaxe des fichiers de configuration du framework de traitement de logs *Apache Flume*.

Voici un example de script basique pour notre outil:

```
in.path = "WORK/Data/movies" # définition du chemin vers le dossier d'input
in.header = 1                # indique la présence d'une ligne de header
out.path = "WORK/Pseudo/movies_pseudonymized" # chemin d'output
out.separator = "\t"         # définition du séparateur en output
log.path = "WORK/Pseudo/movies_job_logs" # chemin pour les logs du traitement

mon_sel_de_hachage = "toto" # définition d'une variable contenant "toto"

#  la 1ère colonne de l'output est une copie de la 7ème colonne d'input :
out1 = in7

# la 2ème colonne de l'output est le hash de la 3ème colonne d'input :
```

```
out2 = in3.hash(mon_sel_de_hachage)
```

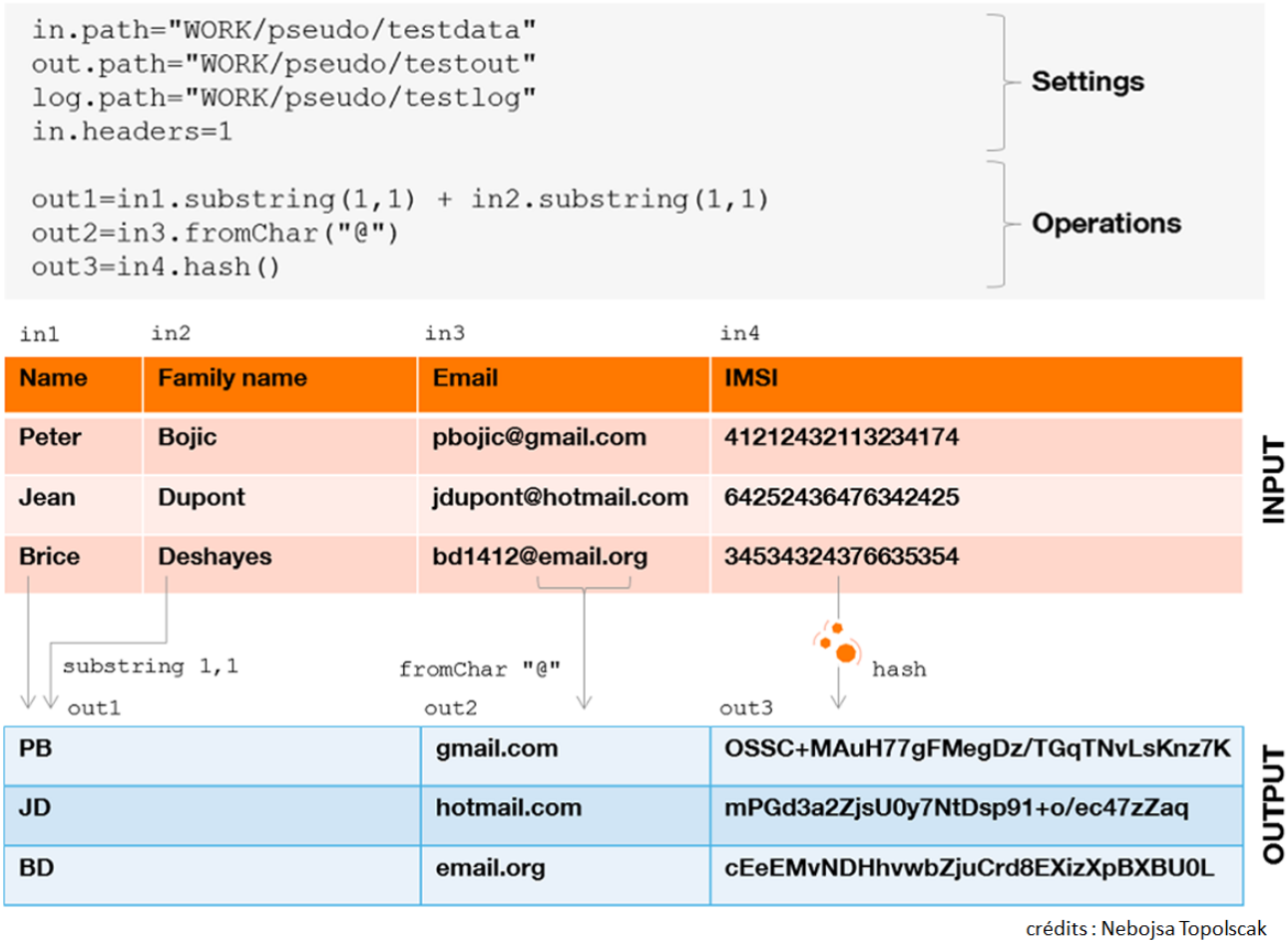
On voit donc qu'il y a globalement trois types de lignes :

- Les lignes qui permettent de spécifier un paramètre du job: les chemins et les séparateurs dans cet exemple
- Les lignes qui permettent d'assigner une chaine de caractères à une variable : `mon_sel_de_hachage="toto"`.
- Les lignes qui permettent de définir le contenu d'une colonne du fichier résultat : `out1=in7`.

De nombreux opérateurs sont disponibles. Ils s'appliquent tous sur une chaine de caractères et renvoient également une chaine de caractères ce qui rends possible les cascades d'opérations. La seule exception est l'opérateur de concaténation + qui permet de joindre deux chaines. Avec ces opérateurs on peut facilement réaliser de nombreux traitements. Par exemple si la première colonne d'input contient une adresse email et que l'on souhaite hasher ce qu'il y a à droite du caractère @ et garder le reste on va écrire :

```
in1.toChar("@").hash() + "@" + in1.fromChar("@")
```

La figure suivante montre un autre exemple de fichier de configuration basique ainsi que le traitement des données qui lui est associé:



crédits : Nebojsa Topolscak

Une gestion des erreurs est également implémentée, prenons un exemple : Nous pouvons obtenir une sous-chaine de caractères entre la $5^{\text{ème}}$ et la $7^{\text{ème}}$ position des champs de la colonne 1 à l'aide de l'opération `in1.substring(5,7)`. Quel doit être le comportement de l'opération si le premier champ d'une des lignes ne contient par exemple que trois caractères ? Faut-il renvoyer une chaine vide ? Remplacer l'output par une autre chaine personnalisée ? Par un tag d'erreur ? Ou bien sauter cette ligne en la considérant comme invalide ? La réponse est que c'est à l'utilisateur de choisir et qu'il pourra le spécifier dans un paramètre optionnel à la fin de chaque opération. En reprenant l'exemple précédent, si l'on souhaite sauter la ligne dans le cas d'une erreur on écrira : `in1.substring(5,7,skipLine)`.

Cette syntaxe simple cache tout l'aspect technique à l'utilisateur mais reste néanmoins très modulable et puissante. Il y a une vingtaine d'options différentes spécifiables ainsi qu'une dizaine d'opérations.

Fonctionnalité avancée : Manipulation de tables de correspondances
Prenons ici un cas concret : Un service souhaite réaliser une étude pour déterminer des profils de fraudeurs. Le *Pseudonymizer* sera un outil efficace pour hasher les champs sensibles en

amont de l'étude afin que les Data Scientists ne manipulent pas de données en clair. Seulement il est primordial de pouvoir faire marche arrière une fois que leur modèle de détection sera opérationnel et que des identifiants hashés de fraudeurs seront remontés.

Une fonctionnalité existe pour combler ce besoin : la gestion de tables de correspondance. Ces tables établissent une correspondance entre les valeurs en clair et les hash. Étant donné un hash on pourra ainsi remonter vers l'identifiant si besoin. Ces tables sont des fichiers CSV qui devront être stockés dans un endroit hautement sécurisé et séparément des données pseudonymisées. Dans le fichier de configuration il suffit d'utiliser l'opération `.createHashMap(seed,map)` à la place de `.hash(seed)` pour activer la création d'une table de correspondance lors d'une opération de hash. Le retour en arrière se fait quant à lui à l'aide de l'opération `.lookup(map)`.

1.3.2.2 Déploiement et utilisation

L'outil est facile à déployer pour un client puisqu'il ne comporte que deux éléments : une archive .jar contenant l'application ainsi qu'un petit script bash pour lancer l'outil proprement. Une fois son *fichier de configuration* complété, l'utilisateur n'a ensuite qu'à lancer le script bash *pseudo* avec le .jar et le *fichier de configuration* en paramètres :

```
pseudo pseudonymizer-1.0.jar /path/configfile.txt
```

Dès que le traitement est terminé, l'utilisateur peut retrouver quelques logs sur le déroulement du traitement dans le dossier log qu'il a spécifié dans le fichier de configuration. En voici un exemple:

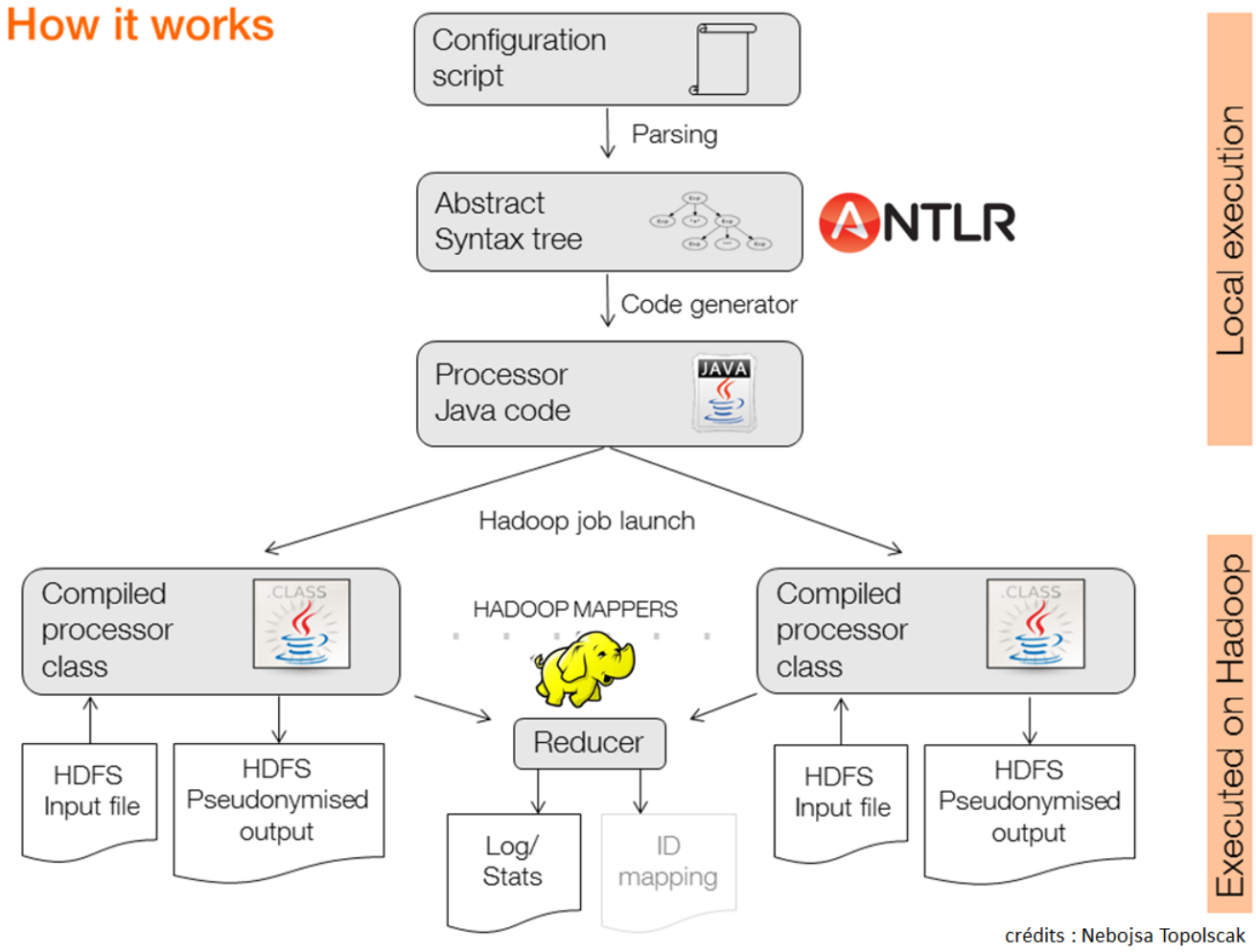
```
STAT:hdfs://PATH/name.csv duration:976ms, linesTotal:100000, invalidLines:3,
headersSkipped:0, fieldErrorsTotal:0, linesWithFieldErrors:0,
invalidLinesBuffer:3/10(not saturated)

hdfs://PATH/name.csv: INVALID_LINE 1: 130, unknown, 5
hdfs://PATH/name.csv: INVALID_LINE 2: 5, unknown, 4
hdfs://PATH/name.csv: INVALID_LINE 3: 268, unknown, 3
```

Il peut y retrouver la durée du traitement ainsi que le compte de ses lignes ou des erreurs rencontrées. Un petit échantillon des lignes qui ont été considérées comme invalides est affiché pour donner une idée du problème à l'utilisateur.

1.3.2.3 Fonctionnement de l'outil

L'outil va ensuite se servir de ce fichier de configuration pour construire un job *MapReduce* en suivant ces trois étapes :

1. Lexer et Parser le fichier de configuration pour créer du code source Java en mémoire capable de traiter une ligne d'input et de ressortir une ligne d'output. Des messages d'erreur précis sont affichés si des fautes syntaxiques ou des incohérences sont repérées et l'outil arrête immédiatement son exécution sans aller plus loin.
2. Compiler *à la volée* et toujours en mémoire cette chaine de caractères contenant le code source
3. Créer une instance de la classe compilée obtenue et s'en servir pour traiter les lignes dans le *Mapper* du job MapReduce



1.3.2.4 Implémentation

Lexing et Parsing du fichier de configuration
Les premières versions de l'outil avaient des fonctionnalités assez limitées et ne permettaient par exemple pas de chainer des opérateurs ni d'effectuer des concaténations. J'ai donc dans un premier temps réalisé "à la main" le lexing et le parsing du fichier de configuration puisque la syntaxe était très basique. Cependant, au fur et à mesure que nous ajoutions des fonctionnalités à l'outil le code devenait impossible à maintenir. Nous sommes donc passé à l'utilisation d'un outil de parsing implémenté en Java nommé *ANTLR*. Il permet de spécifier une grammaire et d'ensuite parcourir facilement l'arbre de syntaxe abstraite généré. Le code est devenu complètement industriel et l'ajout de nouvelles fonctionnalités est devenu trivial. La grammaire ANTLR utilisée est disponible en Annexe 4.1.

I/O de fichiers CSV
La robustesse des IO est garantie par l'utilisation du package CSV d'Apache. L'outil gère grâce à lui la plupart des format csv, notamment le format d'export de csv d'*Excel*.

Choix du langage et du framework de traitement distribué
Le langage Java a été choisi pour profiter de l'expertise de mon collègue Nebojsa. Le choix du framework *Apache Hadoop MapReduce* plutôt qu'*Apache Spark* a été quant à lui motivé par la plus grande maîtrise bas niveau qu'offre *MapReduce* et par le fait que notre outil n'aurait pas bénéficié des optimisations qu'offre *Spark* puisqu'il ne fait appel qu'à une seule phase de *Map*.

Choix de l'unité de division des données
Lorsque l'on réalise un traitement *MapReduce*, l'unité de division des données est classiquement la ligne. Ce découpage implique naturellement que l'ordre des lignes de l'output ne correspond pas à celui de l'input et ceci était perçu comme un problème pour les clients. Il a alors été décidé de réalisé le *split* avec pour unité le fichier. Ce choix a ralenti l'outil d'un facteur 1.5 à 2 mais était nécessaire.

Structuration
Nous avons développé le *Pseudonymizer* en 6 packages qui contiennent en tout 25 classes, et un package de tests.

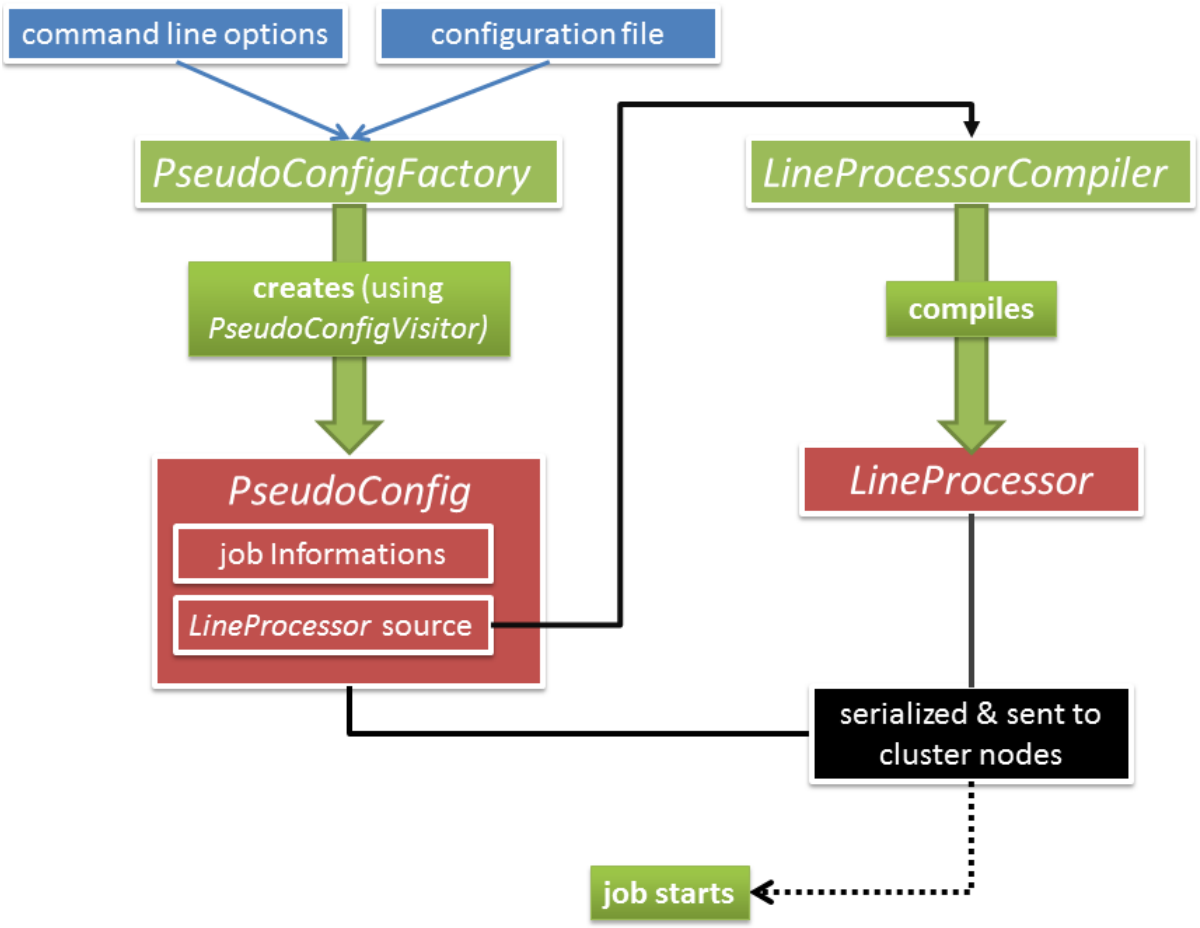
Le premier package nommé *antlr* contient toutes les classes et interfaces générées par le framework *ANTLR* et permettant le lexing et le parsing du fichier de configuration.

Le package *util* contient quant à lui les classes utilitaires permettant le hashage ou la gestion des tables de correspondance.

Les quatre autres packages et leurs interconnexions sont décrits dans le tableau suivant au travers des classes principales qu'ils contiennent :

Classe	package	action
PseudoConfigVisitor	codegen	génère le code source Java du *LineProcessor* à partir de l'arbre de syntaxe abstraite du fichier de configuration
PseudoConfigFactory	config	Factory permettant de créer une instance de *PseudoConfig* à l'aide du *PseudoConfigVisitor*
PseudoConfig	config	permet de stocker toutes les informations tirées du fichier de configuration durant son parsing. Contient le code source *Java* du *LineProcessor*.
StreamProcessor	processor	permet de traiter, à l'aide du *LineProcessor*, les lignes d'un stream de caractères issu des fichiers d'input et d'écrire dans un stream d'output vers le *HDFS*
LineProcessor	processor	classe compilée à la volée par le *LineProcessorCompiler* à partir du code source généré par le *PseudoConfigVisitor*
FullFileRecordReader	mr	étend la classe *RecordReader* du framework MapReduce pour permettre de splitter les données avec le fichier pour unité plutôt que la ligne
PseudoMapper	mr	envoie les logs ainsi que les paires valeur-hash au *PseudoReducer*. Mets en action le *StreamProcessor*.
PseudoReduce	mr	Agrège les logs dans un fichier unique et rempli les tables de correspondance à partir des paires valeur-hash

Voici un schéma des rôles des classes en amont du lancement du job MapReduce :



Nous avons un *design pattern* de *'Factory'* pour la classe *ConfigFileFactory*:

```
public class PseudoConfigFactory {
public static PseudoConfig createInstance(String configFileContent,
    CommandLine cline) throws PseudoConfigException{
[...]
return pseudoConfig;
}
}
```

Travail de documentation

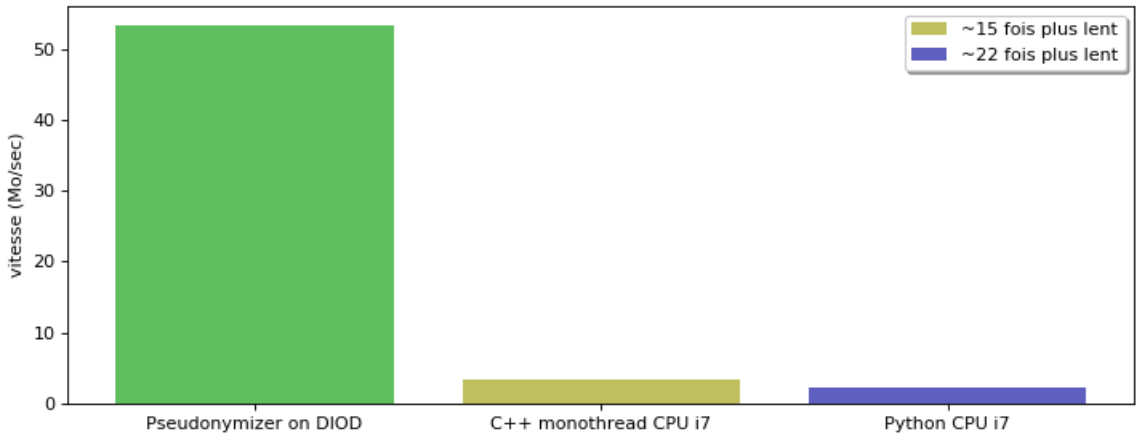
L'élaboration d'une documentation utilisateur complète était une nécessité pour que l'outil soit utilisé. En effet même si il faut réaliser une présentation de l'outil au client pour motiver son adoption, il est primordial qu'il puisse être autonome, notamment pour la découverte des différentes fonctionnalités avancées. J'en ai rédigé une bonne partie et elle est accessible sur le wiki interne d'Orange. Vous pouvez la retrouver en Annexe **??**.

1.4 Premiers résultats

Le *Pseudonymizer* est souvent une alternative à une pseudonymisation non distribuée effectuée sur une machine. J'ai donc réalisé un benchmark pour comparer les performances de notre outil sur notre plateforme à celles d'un traitement écrit en *C++* ou dans un script *Python* et exécuté sur un processeur *i7* récent. La pseudonymisation est réalisée sur un dossier de 416 fichiers de 100 Mo sur trois colonnes. L'action correspond aux lignes de configuration suivantes:

```
out1 = in2.toChar("(")
out2 = in1 + in2.hash
out3 = in2.substring(3, error)
out4-6 = in1-3
```

Le traitement contient des actions variées : recherche de caractère, concaténation, hashing, obtention de sous-chaine et recopie de 3 champs. Voici le résultat obtenu :



Note: Les performances sur notre *cluster* dépendent grandement de son état d'utilisation. Ainsi on peut décupler la vitesse du traitement en lançant l'outil sur une plage horaire peu exploitée. Ce benchmark n'est donné qu'à titre indicatif.

L'outil commence à être utilisé en France par un département d'Orange Corporate ainsi que par un département d'Orange Côte d'Ivoire. J'ai demandé un retour au premier utilisateur de l'outil qui réalisait auparavant ses taches de pseudonymisation à l'aide d'un prétraitement *Hive* : Il trouve que la syntaxe est assez complète pour son utilisation et que l'intérêt majeur de l'outil est qu'il permet dans son cas de découpler le chargement et le traitement des données sur la plateforme.
Il m'est arrivé dans ce contexte d'endosser le rôle de support pour conseiller notre client sur

des aspects techniques liés à l'outil et sur la migration de leur ancienne solution vers notre outil.

1.5 Évolutions futures

1.5.1 Chiffrement

Le principal désavantage de notre méthode de pseudonymisation réversible à base de tables de correspondances entre valeur en clair et valeurs hashées est le coup de stockage de ces tables. Permettre de chiffrer un champ pourrait donc s'avérer très pratique sur les jeux de données particulièrement massifs.

Il faudra forcément utiliser un mécanisme de chiffrement déterministe pour ne pas rendre impossible les agrégations selon le champ chiffré et perdre une grande partie de l'information contenue dans le jeux de données. Un chiffrement asymétrique pourrait également être très intéressant d'un point de vu sécurité puisque la clé publique manipulée par celui qui pseudonymise les données n'a pas une grande valeur tandis que la clé privée n'est utilisée qu'en cas de retours en arrière, potentiellement peu fréquents, et peut être stockée dans un endroit plus sûr.

Il faudra prévoir l'utilisation d'un sel comme dans le cas du hashage pour prévenir les attaques par *dictionnaire* : une attaque par dictionnaire est une forme de brute force où l'attaquant connaissant l'algorithme utilisé va créer une table de correspondance avec des millions d'entrées, par exemple des millions de numéros de téléphone, afin de pouvoir les ré-identifier dans les données pseudonymisées.

1.5.2 Traitement de données non et peu structurées

Une autre future amélioration concerne le traitement des logs. Les logs sont des données qui ne sont le plus souvent pas pré-structurées sous forme de colonnes. Ils nécessitent donc des mécanismes supplémentaires pour pseudonymiser les informations sensibles et identifiantes qu'ils contiendraient.

La plus basique des fonctionnalités pour réussir le traitement est l'utilisation d'expressions régulières. Grâce à elles nous serions capables de reconnaitre de nombreux patterns simples comme les numéros de téléphones ou les identifiants *IMEI* des téléphones portables.

Cependant il n'est par exemple pas possible avec une expression régulière de repérer des adresses de domicile dans un texte ne suivant aucun pattern précis. Nous avons pour cela planifié d'utiliser un modèle de *champs aléatoires conditionnels* (CRF) pré-entrainé par les

membres de NST, une autre équipe d'Orange Corporate. Ce modèle peut reconnaitre des adresses, des emails, des noms de familles et des prénoms. Nous pourrions également réaliser un autre entrainement afin de détecter des numéros de cartes bancaires ou encore permettre à l'utilisateur de fournir à l'outil un modèle de son choix.

Une fois ces ajouts réalisés l'outil sera à même de couvrir une très grande partie des besoins du groupe, notamment la pseudonymisation de *verbatim* issus de conversations avec les divers chatbots ou avec l'assistant d'Orange nommé *Djingo*.

1.5.3 Version streaming

Une version streaming de l'outil est également planifiée pour permettre de connecter l'outil directement à l'ingestion des données sur la plateforme, par exemple par le biais d'un serveur *Apache Kafka*, et d'éviter l'accumulation et le traitement de grosses quantités de données en une seule fois. Il s'agira de traitements en *micro-batch* réalisés à l'aide d'*Apache Spark*. Les changements pour adapter l'outil ne sont pas si lourds puisqu'il faudra remplacer la partie *Hadoop MapReduce* par du *Spark Streaming* mais que toutes les couches plus hautes comme le parsing du fichier de configuration et sa compilation en *Java* restent inchangées.

2

Library Python autour d'un outil de machine learning

Abordons à présent le deuxième et dernier projet auquel j'ai pris part cette année, sous les conseils et en collaboration avec mon collègue chercheur en Data Science Alexis Bondu depuis le mois de Mars.

2.1 Contexte et besoin

2.1.1 Qu'est-ce que Khiops ?

Khiops est un outil de machine learning développé par l'équipe PROF depuis plus d'une décennie. Voici quelques caractéristiques :

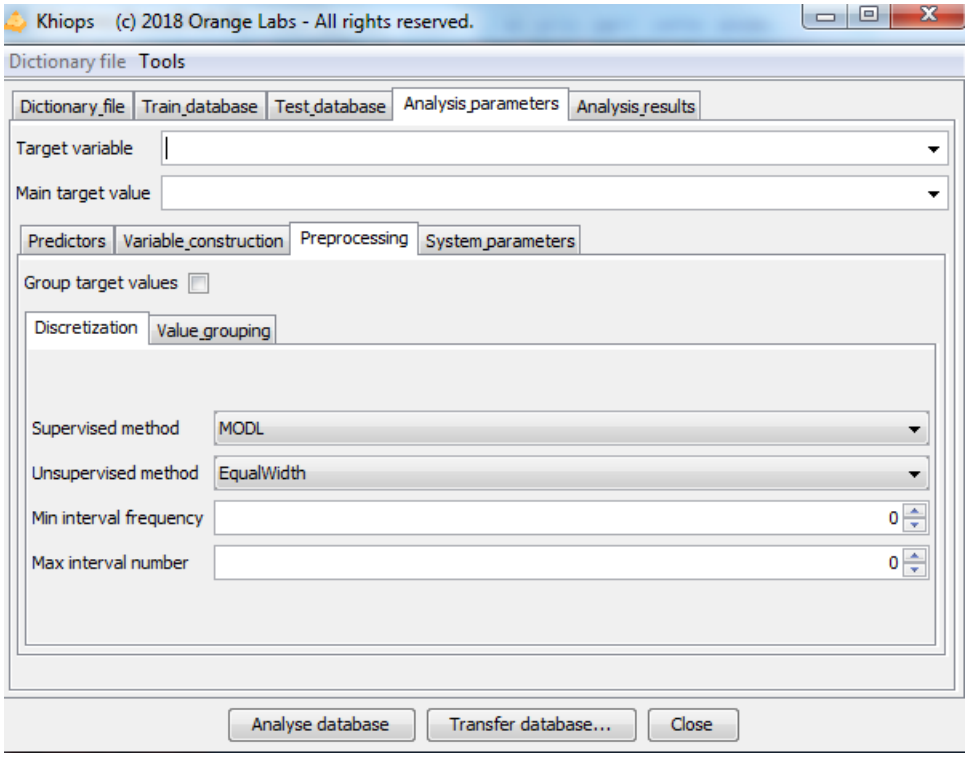
- Il permet la classification, la regression et le clustering
- Il passe bien à l'échelle
- C'est un algorithme sans hyper-paramètres qui gère naturellement la balance entre biais et variance afin de ne pas tomber dans un sous-apprentissage ou sur-apprentissage
- Il permet d'apprendre sur des jeux de données composées de plusieurs tables

Note concernant le mode multitable de Khiops
Khiops va extraire des tables secondaires des agrégations les plus pertinentes possibles pour l'apprentissage.

Prenons un exemple : Imaginons que nous avons une table principale contenant des informations (âge, adresse, offre souscrite...) sur les clients possédant une carte SIM Orange et une table secondaire référençant les appels des clients. On souhaite savoir si les clients vont quitter Orange dans l'année qui suit ou bien rester à partir de ces données, afin de leur faire parvenir des offres ciblées attractives. Dans ce contexte *Khiops* va par exemple détecter que le nombre d'appel de plus de 11 minutes effectués par un client durant le trimestres passé est une valeur très informative et l'utiliser lors de l'apprentissage.

2.1.2 Les limites ergonomiques de l'outil

Le mode d'utilisation classique de *Khiops* se fait par le biais d'une interface graphique basées sur du Java Swing générée automatiquement :



Les résultats sont quant à eux visualisables à l'aide d'un autre outil dédié. Les utilisateurs avertis de *Khiops* ont la possibilité d'utiliser un langage de script propre à l'outil pour gagner du temps.

Ces modes d'utilisation sont difficiles à prendre en main pour un néophyte habitué à manipuler les algorithmes de machine learning et leurs prédictions dans son langage de programmation.

Ces limites ergonomiques gâchent et cachent la puissance de *Khiops* aux yeux des potentiels nouveaux utilisateurs qui ne veulent pas dépenser du temps dans la prise en main d'un nouvel outil. Il semblait essentiel de pouvoir rendre plus attractif *Khiops* en offrant un autre mode d'utilisation plus familier.

2.2 Solution apportée : PyKhiops

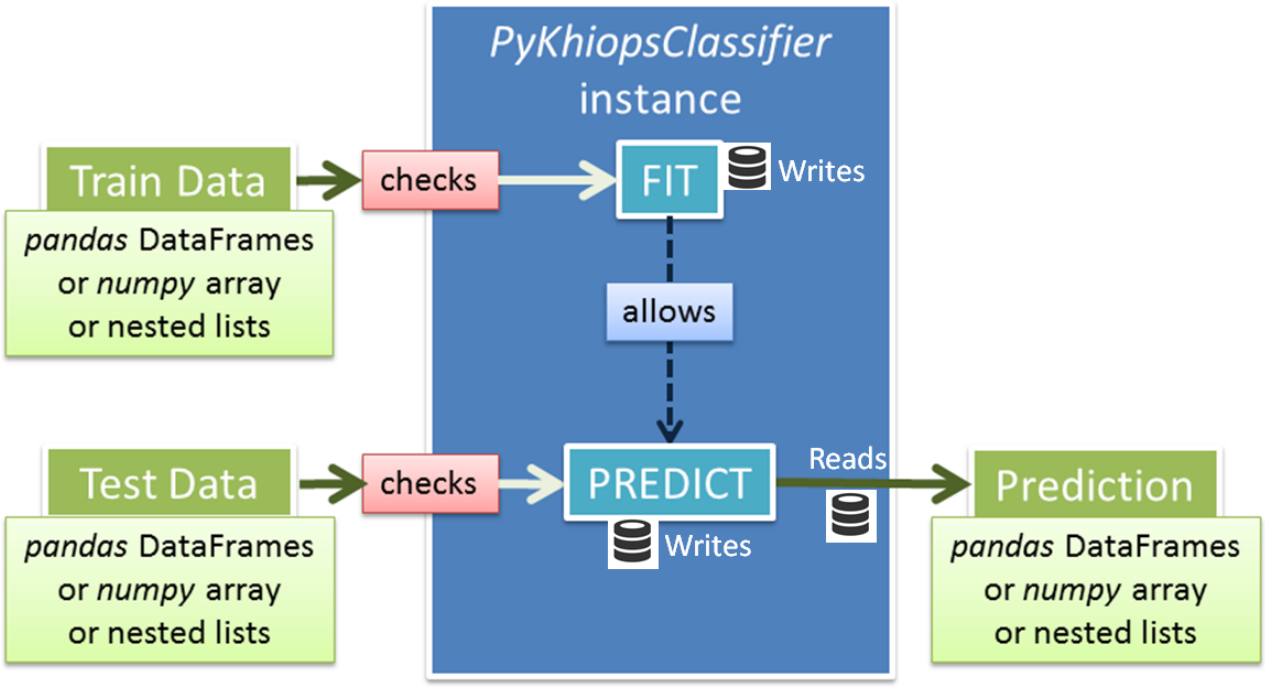
Alexis a alors réalisé dans ce contexte une première version de wrapper qui permet d'utiliser Khiops dans le langage *Python* avec une syntaxe adoptant les conventions du framework très répandu *scikit-learn*. Marc Boullé le créteur de *Khiops* a par la suite fait mué ce projet de wrapper en une librairie complète permettant d'appeler toutes les fonctionnalités de *Khiops* depuis un script *Python*.

Dans ce projet mon travail a été, sous la direction d'Alexis, de robustifier et d'industrialiser le module se chargeant de la partie wrapper *scikit-learn*. J'ai ainsi :

1. Recodé l'ensemble du module pour qu'il adopte toutes les conventions *pythoniques*.
2. Refactorisé le code en fonctions élémentaires pour améliorer sa maintenabilité.
3. Ajouté de nombreuses sécurités de typage et diverses vérifications de cohérence d'utilisation. Ce n'est pas *pythonique* mais c'est essentiel pour que l'utilisateur ait des erreurs claires et qu'il n'ait pas une mauvaise image de *Khiops* à cause du wrapper.
4. Rédigé un set conséquent de tests unitaires
5. Réalisé une documentation complète adoptant les conventions de *scikit-learn*. Toute la documentation est factorisée dans la version de développement du module, c'est à dire que chaque paramètre de fonction n'est décrit qu'une seule fois pour une maintenabilité facilité. L'exécution de cette version de développement va générer une version *release* du module avec une documentation lisible classique.

Actuellement le module est finalisé et il sera rendu disponible nativement au sein de la librairie *Python* complète dans la prochaine version de *Khiops*.

Voici un schéma qui représente le fonctionnement de la classe de classification *PyKhiopsClassifier* :



Dans son utilisation basique de classification, le rôle du wrapper est :

1. de recevoir les données d'entrainement
2. de réaliser des vérifications de typage et de dimension qui peuvent s'avérer complexes, notamment lorsque le mode *multitable* est utilisé.
3. d'écrire les données sur disque
4. d'appeler *Khiops* sur ces données pour obtenir un modèle entrainé
5. de pouvoir ensuite recevoir des données de test
6. de réaliser les mêmes vérifications que précédemment mais en vérifiant en plus la cohérence avec les données d'entrainement
7. d'écrire les données sur disque
8. d'appeler *Khiops* sur ces données pour obtenir des prédictions à l'aide du modèle précédemment appris
9. de récupérer ces prédictions depuis le disque et de les rendre à l'utilisateur sous la forme d'un objet *Python* en cohérence de type avec les objets qu'il a fourni jusqu'à présent

Voici un exemple de script utilisant le module :

```
from PyKhiops import PyKhiopsClassifier, acc
import pandas as pd

X_train, y_train, X_test, y_test = [charge données avec pandas]

pk = PyKhiopsClassifier()
pk.fit(X=X_train, y=y_train)
y_pred = pk.predict(X=X_test)
print(acc(y_pred, y_test)) # on regarde l'accuracy de la prédiction
```

La syntaxe pour du *multitable* a été pensée pour être très simple et tirer parti de l'indexation des DataFrames du module *pandas* :

```
#On souhaite ici prédire l'age de clients àpartir d'une table principale et
#d'une table secondaire d'appels passés et d'une table de factures hors
    forfait :
from PyKhiops import PyKhiopsClassifier, acc
import pandas as pd

clients_train, appels_train, hors_forfaits_train, age_train,
clients_test, appels_test, hors_forfaits_test, age_test = [charge données avec
    pandas]

pk = PyKhiopsClassifier()
pk.fit(X=clients_train, X_list=[appels_train, hors_forfaits_train] y=age_train)
age_pred = pk.predict(X=clients_test, X_list=[appels_test, hors_forfaits_test])
print(acc(age_pred, age_test))
```

Ce module rend donc très familière l'utilisation de *Khiops* pour n'importe quel Data Scientist. Le module permet également d'utiliser la classe *PyKhiopsRecoder* qui est un wrapping de la fonctionnalité de *Khiops* de recodage de jeux de données.

2.3 Évolutions futures

La librairie est déjà utilisée par les différents stagiaires de l'équipe et quelques améliorations sont prévues pour mon module comme par exemple l'ajout d'une classe PyKhiopsRegressor pour compléter les possibilités d'apprentissage supervisé du wrapper.

Il se pourrait également la fonctionnalité de coclustering de *Khiops* soit prochainement wrappée pour *Spark* en *Scala*. Il sera alors pertinent d'ajouter à la librairie *PyKhiops* un module *PySparkKhiops*...

3

Enseignements tirés et progression personnelle sur cette 1ère année

Durant cette année, je me suis vraiment senti progresser durant chacune de mes sessions en entreprise. J'ai gagné en expérience technique autant qu'en expérience du monde de l'entreprise.

3.1 Vie d'entreprise

J'en sais désormais plus sur le fonctionnement d'un grand groupe tel qu'Orange et sur les bonnes pratiques à adopter pour être efficace dans un tel environnement. Il faut être à l'écoute de ses collaborateurs et ne pas se précipiter. Le temps dépensé à prendre du recul sur une tâche et à en discuter est très largement rattrapé par la suite. J'ai la chance de pouvoir être entouré de collègue experts et patients qui m'apportent leur conseils dès que nécessaire.

3.2 Technique

Je vais ici essayer de récapituler brièvement les progressions techniques que j'ai effectuées lors de mes sessions d'entreprise de cette année.

3.2.1 Java

Aux côtés de mon collègue Nebojsa j'ai énormément progressé en développement *Java* et en génie logiciel.

J'ai appris à écrire un code modulaire en structurant le code en classes ayant un rôle naturel et utilisant si nécessaire des *design patterns* pertinents. Le code est ainsi facilement maintenable et lisible, ce qui est essentiel lorsque nous sommes plusieurs à collaborer sur le même projet. J'ai également appris qu'il faut éviter à tout prix de réinventer la roue. Il est très rentable de passer du temps à rechercher des solutions existantes pour ensuite les adapter.

J'ai également intégré des détails d'optimisation tels que l'utilisation d'énumérations ou de *StringBuffer* pour construire des chaines de caractères efficacement. J'ai gagné en compétence sur le framework *Apache Hadoop*, que ce soit au niveau *HDFS* ou *MapReduce*. L'utilisation dans un contexte industriel d'une grammaire de langage de script à l'aide d'*ANTLR* a aussi été enrichissante.

3.2.2 Python

Grâce notamment à Eole et Alexis j'ai beaucoup progressé en développement *Python* jusqu'à être capable de produire un code industriel. J'en suis presque arrivé à connaître par coeur les Python Enhancement Practices...

J'ai appris à gérer la balance entre optimisations fines et lisibilité. Un code *Python* se doit dans la plupart des cas d'être lisible et compréhensible en l'absence de commentaires, quitte à sacrifier quelques millisecondes lors de l'exécution.

J'ai également dû me familiariser avec énormément de détails des packages *numpy* et *pandas* ainsi qu'avec le module *unittest*.

Conclusion

Cette première année m'a conforté dans l'idée que l'alternance est vraiment un processus d'apprentissage qui me correspond. Je me suis tout autant enrichi au cours des sessions universitaires qu'au cours des sessions en entreprise. Pouvoir mettre en application dans un contexte professionnel ce que l'on apprend est vraiment valorisant et permet de se remettre en question en permanence.

Mes deux projets de l'année m'ont énormément appris techniquement sur des aspects variés s'étalant sur tout le cycle de production d'un outil industriel.

Je vais profiter de ma dernière année de Master en apprentissage pour parfaire mon bagage technique et gagner un maximum de compétences et d'expérience afin de prétendre à devenir un Data Engineer efficace !

Bibliographie

[1] CNRS. Rgpd : choisir entre l'anonymisation ou la pseudonymisation des données personnelles, Juin 2018. http://www.cil.cnrs.fr/CIL/spip.php?article3056.

[2] Talend. Solution talend, 2017. https://fr.talend.com/blog/2017/07/11/gdpr-et-gestion-des-donnees-cinq-piliers-pour-reussir-avec-talend/.

[3] Teskalabs. Solution turbocat.io, 2017. https://www.teskalabs.com/products/turbocat.io/.

4

Annexe

4.1 Grammaire du fichier de configuration du Pseudonymizer

Voici le fichier .g4 décrivant notre grammaire pour le framework *ANTLR*.

```
grammar Pseudo;

configFile: assignment+ EOF;

assignment:
IN_FIELDS '=' INT                          #AssignInFields
|IN_PATH '=' STRING                        #AssignInPath
|IN_SEPARATOR '=' STRING                   #AssignInSeparator
|IN_HEADERS '=' INT                        #AssignInHeaders
|OUT_KEEPHEADERS '=' INT                   #AssignOutKeepHeaders
|OUTXHEADER '=' STRING                     #AssignOutXHeader
|IN_NAMES '=' varlist                      #AssignInNames
|IN_ENCODING '=' STRING                    #AssignInEncoding
|IN_CSVTYPE '=' csvType = (K_EXCEL|K_TSV|K_RFC)#AssignInCsvType
|OUT_PATH '=' STRING                       #AssignOutPath
|OUT_SEPARATOR '=' STRING                  #AssignOutSeparator
|OUT_SEED '=' outSeed = (STRING|K_RANDOM)  #AssignOutSeed
|OUT_ERROR '=' STRING                      #AssignOutError
|OUT_ENCODING '=' STRING                   #AssignOutEncoding
|OUT_CSVTYPE '=' csvType =(K_EXCEL|K_TSV|K_RFC)#AssignOutCsvType
|LOG_PATH '=' STRING                       #AssignLogPath
|MAP_COLLISION '=' collisionCheck=(K_ON|K_OFF) #AssignCollisionCheck
|VAR '=' concat                            #AssignVar
|OUTVAR '=' concat                         #AssignOutVar
|mapVar=MAPVAR '.path' '=' STRING          #AssignMapPath
|OUTRANGE '=' INRANGE                      #AssignRange
```

```
;

concat: tok | (tok '+' concat);

tok:   tokvar(f_substring2|f_substring3
|f_last|f_tochar|f_fromchar
|f_hash0|f_hash1
|f_createhashmap1|f_createhashmap2
|f_lookuporcreatehashmap1|f_lookuporcreatehashmap2|f_lookup)*?;

tokvar:
VAR      #TokvarVar
|INVAR   #TokvarInvar
|STRING  #TokvarString
;

varlist: VAR (',' VAR)*;

f_substring2:
'.substring(' arg1=INT (',' arg2=(K_ALL|K_ERROR|K_SKIP|STRING))?')';
f_substring3:
    '.substring('arg1=INT','arg2=INT(','arg3=(K_ALL|K_INTERSECTION|K_ERROR|K_SKIP|STRING))?')
f_last:
'.last('arg1=INT(','arg2=(K_ALL|K_INTERSECTION|K_ERROR|K_SKIP|STRING))?')';
f_tochar:
'.toChar(' arg1=STRING (',' arg2=(K_ALL|K_ERROR|K_SKIP|STRING))?')';
f_fromchar:
'.fromChar(' arg1=STRING (',' arg2=(K_ALL|K_ERROR|K_SKIP|STRING))? ')';
f_hash0:
'.hash'|'.hash(' ')';
f_hash1:
'.hash(' arg1=STRING ')';
f_createhashmap1:
'.createHashMap(' arg1=MAPVAR ')';
f_createhashmap2:
'.createHashMap(' arg1=MAPVAR ',' arg2=STRING ')';
f_lookuporcreatehashmap1:
'.addToHashMap(' arg1=MAPVAR')';
f_lookuporcreatehashmap2:
'.addToHashMap(' arg1=MAPVAR ',' arg2=STRING')';
f_lookup:
'.lookup(' arg1=MAPVAR (',' arg2=(K_ERROR|K_SKIP|STRING))?')';

K_ALL:          'all';
K_INTERSECTION: 'intersection';
K_ERROR:        'error';
K_SKIP:         'skipLine';
K_RANDOM:       'random';
```

```
K_EXCEL:          [eE][xX][cC][eE][lL];
K_TSV:            [tT][sS][vV];
K_RFC:            [rR][fF][cC];
K_ON:             [oO][nN];
K_OFF:            [oO][fF][fF];

IN_FIELDS:            'in.fields';
IN_PATH:              'in.path';
IN_SEPARATOR:         'in.separator';
IN_HEADERS:           'in.headers';
OUT_KEEPHEADERS:      'out.keepHeaders';
IN_NAMES:             'in.names';
IN_ENCODING:          'in.encoding';
IN_CSVTYPE:           'in.csvType';
OUT_PATH:             'out.path';
OUT_SEPARATOR:        'out.separator';
OUT_SEED:             'out.seed';
OUT_ERROR:            'out.error';
OUT_ENCODING:         'out.encoding';
OUT_CSVTYPE:          'out.csvType';
LOG_PATH:             'log.path';
MAP_COLLISION:        'map.collisionCheck';
OUTXHEADER:           'out'[0-9]+'.header';

BOOL:       ('true'|'false');
INRANGE:    'in' [0-9]+ '-' [0-9]+;
OUTRANGE:   'out' [0-9]+ '-' [0-9]+;
INVAR:      'in' [0-9]+ ;
OUTVAR:     'out' [0-9]+;
MAPVAR:     'map'[0-9]+;
STRING:     '"' ('\\"'|~'"')* '"' ; // backslash-quote is an escaped quote
INT:        [0-9]+ ;
VAR:        ([a-z]|[A-Z])([a-z]|[A-Z]|[0-9]|'-'|'_')*; //1st char must be alpha
EOL:        '\r'? '\n' -> channel(HIDDEN);
WS:         (' ' | '\t')+ -> channel(HIDDEN);
COMMENT:    '#'.*?( EOL|EOF) -> channel(HIDDEN);
```