Apprentissage fédératif pour la prédiction du churn : une évaluation

Sébastien Godard *, Nicolas Voisine**
Tanguy Urvoy**, Vincent Lemaire**

*Université de Bretagne Sud, **Orange Labs Lannion

Résumé. Les smartphones sont omniprésents dans notre quotidien. Ils constituent une ressource informatique à portée de la main avec un accès direct à une quantité considérable d'informations personnelles. Ils représentent une source de données très précieuse pour les opérateurs de télécommunication, mais la nature très décentralisée de ces données et les attentes évidentes des clients en matière de respect de la vie privée requièrent de nouvelles approches en apprentissage statistique. L'apprentissage ubiquitaire (ou *ubiquitous datamining*) qui intègre des terminaux avec la capacité de traiter localement leurs propres données, est une alternative intéressante à la centralisation de masse en traitement de données. L'apprentissage fédératif, (ou *federated learning*) est une réalisation de l'ubiquitous datamining, qui permet de déployer certains modèles d'apprentissage automatique sur des terminaux "autonomes" tels les smartphones. Cet article propose une évaluation détaillée de ce type d'apprentissage distribué dans le cadre de la prédiction de l'attrition (ou *churn*) sur des données issues d'un opérateur télécom.

1 Introduction

Omniprésents dans notre quotidien, les smartphones représentent une véritable ressource informatique à portée de la main. Pour de nombreuses personnes, il s'agit des appareils informatiques les plus utilisés. Fréquemment sollicités, ils ont accès à une quantité considérable de données issues de leurs puissants capteurs (appareils photos, microphones, GPS, etc.), ainsi qu'à des objets connectés auxquels ils sont reliés. Cette riche source de données est potentiellement porteuse de nombreuses promesses pour les opérateurs télécoms et leurs clients : elle permet de contrôler au plus près la qualité des services et leur ergonomie, elle permet aussi d'envisager des services innovants par apprentissage statistique. Mais ces données sont aussi, et surtout, des données très personnelles et très sensibles.

L'entrée en vigueur du RGPD (règlement général sur la protection des données European Union, 2016) redéfinit complètement la politique de gestion des données. Il n'est plus question de recueillir massivement les données de clients sans objectif précis de service. Le RGPD fixe le cadre juridique relatif à la protection des données personnelles au sein de l'Union Européenne. Responsabilisant davantage les entreprises, le RGPD donne de nouvelles obligations aux opérateurs de services quant à la gestion des données, rendant notamment leur centralisa-

tion beaucoup plus encadrée. Le respect de la vie privée est plus que jamais une problématique importante au cœur du traitement des données.

L'objectif de cette étude expérimentale est de proposer et d'évaluer une solution d'apprentissage statistique décentralisée, destinée à être déployée sur les smartphones [1] . Le fait de s'affranchir de la centralisation des données clients, est un apport en termes de protection de la vie privée qui permet de se conformer aux normes du RGPD. La solution que nous proposons est grandement inspirée par une publication récente qui introduit une réalisation du domaine de l'ubiquitous datamining : le *federated learning* (May et Saitta, 2010; McMahan et Ramage, 2017). L'objectif est de réaliser une instanciation d'un des concepts clés du federated learning : l'apprentissage d'un modèle de classification (dans cet article un réseau de neurones), de façon décentralisée et fortement distribuée en partageant les paramètres de modèles statistiques plutôt que les données sensibles. Les analyses que nous présentons sont basées sur des simulations à partir de données réelles d'opérateur de télécommunication.

2 Federated learning

2.1 Définition du concept

Le federated learning est une approche décentralisée de machine learning. Il s'inscrit dans le cadre de l'ubiquitous datamining (Park et Kargupta, 2002; Bhaduri et al., 2008). Il s'applique à un environnement ubiquitaire, dans lequel les données sont fortement distribuées et proviennent de nombreuses sources mobiles et hétérogènes. Ces mêmes sources sont dotées de capacités de calcul et de mémoire qui, bien que limitées, offrent la possibilité de participer activement à l'apprentissage de modèles statistiques. L'idée centrale de cette approche est de partager le modèle plutôt que les données, pour un calcul et une analyse sécurisés. Ainsi, un modèle statistique est défini, commun à l'ensemble des agents intervenant dans le système. L'apprentissage du modèle global est réalisé de façon collaborative en sollicitant directement les capacités de calcul des appareils sources des données. Chaque appareil calcule des statistiques locales à partir de ses propres données. Les seules informations échangées dans le système sont des résultats d'algorithmes intermédiaires correspondant à ces opérations, peu sensibles par nature. Ainsi, les données ne font l'objet d'aucune centralisation. Elles restent hébergées localement sur les appareils qui les ont générées : c'est une démarche plus respectueuse de la vie privée. Elle permet également de répondre à la problématique de la sécurité, limitant la surface d'attaque à l'appareil seul.

2.2 Principale méthode

L'apprentissage distribué de réseaux de neurones est, à ce jour, la principale méthode d'application du federated learning (McMahan et al., 2016; McMahan et Ramage, 2017). Un modèle unique est défini et partagé à l'ensemble des agents. Reposant sur une architecture client/serveur, la tâche d'apprentissage de ce modèle est réalisée par un ensemble d'appareils,

1. Il ne serait pas aisé actuellement de faire tourner des réseaux de neurones à grande échelle sur des téléphones. Cet aspect est peu étudié dans l'article (et dans la littérature à notre connaissance). Cependant il faut noter qu'il est déjà possible de le faire via Tensorflow lite (Abadi et al., 2015) et que les smartphone actuels ont une puissance de calcul comparable à un petit PC, de plus les constructeurs les équipent de plus en plus de GPU. Ces aspects ne sont pas étudiés dans l'article et seront investigués ultérieurement.

communément appelés clients. Un serveur central coordonne l'apprentissage. Le serveur sélectionne aléatoirement des clients, auxquels il transmet les poids actuels du modèle global (A), comme le représente la Figure 1. Chacun de ces clients, en prenant comme paramètres initiaux ces poids, entraîne indépendamment le modèle à partir de ses propres données (B). Le serveur agrège les résultats en effectuant la moyenne (pondérée par la taille des jeux de données locaux) des poids résultants de chaque entrainement indépendant (C). Il actualise ainsi les poids du modèle global. Ces étapes constituent un "tour de communication". Elles sont répétées itérativement afin de réaliser l'apprentissage du modèle.

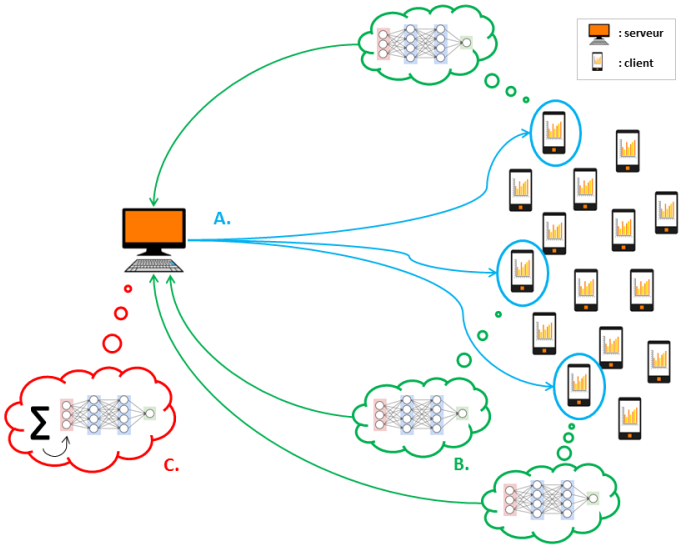

FIG. 1 – *Apprentissage fédéré d'un modèle de réseau de neurones.*

2.3 Algorithme

L'ensemble des paramètres du modèle "fédératif" [2] sont listés ci-dessous. Ces paramètres sont dictés par la partie centralisée du système d'apprentissage global. L'algorithme 1 montre comment ces paramètres sont utilisés au sein d'une structure algorithmique globale telle que nous l'avons utilisée pour nos expériences.

Paramètres du modèle :
— T : nombre de tours de communication ($t = 1, 2, ..., T$).
— K : nombre total de clients.
— C : fraction de clients à sélectionner lors de chaque tour t.
— B : taille des batchs locaux (B = ∞ correspond à un batch intégral)

2. Le modèle de réseaux de neurones est lui décrit dans la partie expérimentation

— E : nombre d'itérations effectuées par les modèles locaux.
— η : taux d'apprentissage local (learning rate).
— L_R : paramètre contrôlant la mise à jour des poids du modèle global.

Algorithme 1 *Les K clients sont indexés par k ; B est la taille du minibatch local, E est le nombre d'époques locales, and η est le taux d'apprentissage.*

Le serveur exécute :
initialiser w_0
pour chaque tour de communication $t = 1, 2, ...$ **faire**
  $m \leftarrow max(C \cdot K, 1)$
  $S_t \leftarrow$ (ensemble aléatoire de m clients)
  pour chaque client $k \in S_t$ **en parallèle faire**
    $w_{t+1}^k \leftarrow ClientUpdate(k, w_t)$
  fin pour
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
fin pour

ClientUpdate(k, w) : *// Exécution du client k*
$\beta \leftarrow$ (diviser P_k en batch de taille B.)
pour chaque epoque local i de 1 à E **faire**
  pour batch $b \in \beta$ **faire**
    $w \leftarrow w - \eta \bigtriangledown l(w; b)$
  fin pour
fin pour
renvoyer w au serveur

Le serveur initialise [3] les poids w_0 du modèle, avant chaque tour de communication t. Le nombre m de clients que le serveur doit sélectionner à chaque tour, parmi les K clients, est défini par le coefficient C (par définition, si $C = 0$, $m = 1$). A chaque tour, le serveur sélectionne aléatoirement un ensemble S_t de m clients et demande aux clients de calculer une mise à jour des poids du modèle (ClientUpdate). Dans cette procédure un clients sélectionnés calcule une mise à jour des poids du modèle à partir de ses propres données P_k, avec pour poids initiaux les poids courants du modèle. Les paramètres B et E contrôlent l'apprentissage local, correspondant respectivement à la taille des batchs locaux et au nombre d'itérations à effectuer sur les données locales. Le serveur actualise ensuite les poids du modèle en effectuant une moyenne pondérée par la quantité de données de chaque client participant.

Par rapport à l'algorithme décrit dans l'article (McMahan et al., 2016), nous avons ajouté un paramètre au niveau serveur afin de "contrôler" la mise à jour des paramètres du modèle global. A l'issue de chaque tour de communication, le serveur effectue la moyenne pondérée des poids résultant des entrainements locaux effectués par les clients. Au lieu de faire de cette moyenne pondérée les nouveaux poids du modèle global, nous n'effectuons qu'une mise à jour partielle. Pour ce faire, les poids résultant de l'entrainement distribué sont moyennés avec les poids courants du modèle global (voir Équation 1).

3. Ces poids sont issus du tour précédent.

$$w_{t+1} \leftarrow L_R * \frac{1}{n} \sum_{k=1}^{K} n_k * w_{t+1}^k + (1 - L_R) * w_t \quad (1)$$

Le nouveau paramètre, L_R, contrôle le poids de la mise à jour. Nous avons fixé sa valeur par défaut à 0.5, valeur pour laquelle les paramètres courants et les paramètres issus de l'entrainement distribué ont le même poids. Ce paramètre agit comme un moment d'inertie : il apporte de la stabilité dans l'apprentissage global du modèle.

Afin de tester ce type d'apprentissage distribué, tel que défini ci-dessus, nous avons utilisé une approche client/serveur au cours des expérimentations. Les données ont été réparties en jeux de données "locaux" indépendants représentant les clients. Le rôle de serveur était de coordonner l'apprentissage du modèle en étapes successives. Nous avons considéré un ensemble de clients fixés, disposant chacun d'une base de données locale fixée. La partie expérimentale présentée plus bas dans l'article a été intégralement réalisée en langage Python avec TensorFlow (Abadi et al., 2015).

3 Les données de churn (attrition)

Le concept marketing de *churn*, ou d'attrition, désigne la résiliation d'un contrat ou d'un forfait dans le cas des télécoms, et plus généralement la perte de clients. L'intérêt avec ce genre de données est de prédire les clients susceptibles de résilier leur offre. Ainsi, il est possible de prendre les devants en menant auprès de ces clients des actions marketing dans le but de les conserver.

Afin de ne pas centraliser les données un jeu de données est construit à partir de deux fichiers de logs anonymisés qui restent localisés sur le terminal du client. Le premier fichier contient des logs de type parcours client, fournissant des informations sur les interactions du client avec les canaux de communication Orange. À chaque ligne sont notamment relevées la date et l'heure précise ainsi que le motif de l'interaction. Ces informations de logs sont ensuite transformées sous forme vectorielles, à la manière d'images, tel que suggéré dans (Castanedo et al., 2014).

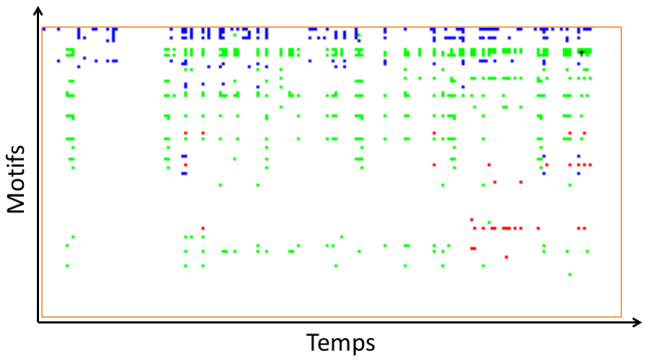

FIG. 2 – *Exemple de représentation des données de churn.*

Un exemple de ces images est donné en figure 2. Les abscisses correspondent à des intervalles de temps et les ordonnées aux valeurs des différents motifs. A chaque pas de temps (en X) correspond un vecteur (en Y). Chaque composante de ce vecteur est une représentation d'un compte ou d'un agrégat calculé sur les logs (par exemple nombre de contacts vers le service client). Le code couleur du pixel est une représentation "imagée" de la valeur de cette composante. Le second fichier contient l'information cible, à savoir la résiliation ou non du contrat du client. Il nous permet de construire la variable réponse du jeu de données. C'est une variable binaire pour laquelle un label positif indique une résiliation (churn). Les données d'entrainement sont composées de 61 251 exemples pour l'apprentissage et de 15 313 exemples pour le test.

4 Protocole d'apprentissage des modèles locaux

4.1 Modèles locaux neuronaux utilisés

Nous avons repris l'architecture de modèle qui a été définie dans le cadre des travaux de Castanedo et al. (2014). Il s'agit d'un modèle convolutif, constitué de 3 couches de convolution (respectivement (100,3), (1,3), (1,3)) avec 32 noyaux de profondeur et une fonction d'activation ReLU. Elles sont toutes suivies d'une couche de MaxPooling (1,3), puis de Dropout (0.5), et d'une BatchNormalization. Ensuite, une couche de linéarisation, suivie d'une couche dense de 100 unités avec une fonction d'activation ReLU, suivie d'un Dropout (0.5), précèdent une couche de sortie Sigmoid.

4.2 Critère de performances

L'AUC est utilisée pour évaluer les performances de classement (*ranking*) du modèle global, à partir des données de test. Le choix d'utiliser ce critère d'évaluation plutôt que la précision (*accuracy*) est notamment motivé par la nature "déséquilibrée" que présentent les données. En effet, seulement 5.4% des données étant labellisées "churn", un modèle trivial qui répondrait systématiquement "non-churn" obtiendrait une précision 84%.

4.3 Performances de références

Pour avoir une valeur de référence des résultats qui sont utilisés avec le modèle où les données sont distribuées nous avons entrainé un modèle "classique" avec les données centralisées (même données et même format de prétraitement). Ce modèle obtient une AUC en test de 82%. Nous nous référerons donc à cette valeur afin de comparer les performances observées avec l'approche d'apprentissage fédéré.

4.4 Régularisation et Early Stopping

Lors de l'apprentissage du modèle de référence nous avons constaté un problème de surapprentissage et donc des performances statistiques en test éloignées de celles en apprentissage. Afin de remédier à ce problème nous avons utilisé le dropout (Srivastava et al., 2014) et la batchNormalization (Ioffe et Szegedy, 2015). Le dropout est une technique employée

pour éviter le sur-apprentissage, qui consiste à désactiver aléatoirement une certaine proportion de neurones (ou de connections) lors de l'apprentissage du modèle. La BatchNormalization consiste à normaliser les sorties des couches du réseau.

Toujours dans le but de limiter le sur-apprentissage (au niveau des modèles locaux) nous avons contrôlé l'AUC après chaque itération d'apprentissage au sein d'une procédure "d'early stopping" utilisant un ensemble de validation prélevé au sein de l'ensemble d'apprentissage. Nous conservons de ce fait le modèle le plus performant en validation, qui sera retourné au serveur.

L'intérêt de l'early stopping est de stopper l'apprentissage dès lors que celui-ci ne progresse plus. Cela trouve encore plus son sens dans le cadre de l'apprentissage fédéré, puisque les clients ne disposant chacun que d'un nombre limité d'exemples des données, ceux-ci pourraient être rapidement appris "par cœur" par le modèle. L'early stopping permet aussi de s'affranchir d'une charge de calcul inutile.

Note : Certains clients n'ont qu'un seul label présent dans leurs données [4], ce qui ne permet pas d'évaluer l'AUC localement. On doit donc se contenter de maximiser la précision (*i.e.* le taux de bonnes prédictions du label présent). Le partage des poids (et le paramètre d'inertie L_R) permettent de compenser ce problème en stabilisant le processus d'apprentissage global.

5 Résultats

Nous présentons dans cette section les résultats selon plusieurs axes d'analyse : le parallélisme, la charge locale sur chaque client et la distribution des données.

5.1 Parallélisme multi-client

iid_weighted	B=12	
C	T	V
0	78	baseline
0,1	39	x 2,0
0,2	36	x 2,2
0,5	34	x 2,3
1	34	x 2,3

FIG. 3 – *Effet du parallélisme multi-client sur l'apprentissage global en termes d'accélération du nombre de tours de communication (pour atteindre 0.78 d'AUC).*

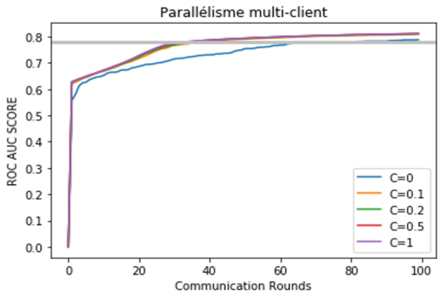

FIG. 4 – *Effet du parallélisme multi-client sur l'apprentissage global : convergence de l'AUC versus C et du nombre de tours de communication (T).*

Le parallélisme multi-client est contrôlé par le coefficient C. Il s'agit de la fraction de clients que le serveur doit sélectionner lors de chaque tour de communication. Afin d'étu-

4. Un client est rarement "churner" et "non-churner" au même moment.

dier l'effet du parallélisme multi-client sur l'apprentissage global du modèle, nous avons entrainé de façon distribuée notre modèle, pour différentes valeurs de C. Nous avons fixé les valeurs des autres paramètres à : K=100 ; B=12 ; E=1 ; L_R=0.5 ; η = 0.001. Nous avons réalisé ces tests à partir d'une distribution des données IID en utilisant un dictionnaire de poids ("class_weight") afin de rééquilibrer la représentativité des données. Cette distribution est nommée IID_weighted. L'attribut "class_weight" est utilisé pour pondérer la fonction de perte pendant l'entraînement. Cela permet ainsi de donner un poids différents aux observations au cours de l'entrainement du modèle, en fonction de leur classe. Pour définir la valeur des poids mappés à chaque indice de classe, on utilise la fonction "compute_class_weight" de scikit-learn, permettant d'estimer les poids des classes pour les jeux de données unbalanced.

Les Figures 3 et 4 présentent sous forme de tableau ou de courbes le nombre T de tours de communication nécessaires avant d'attendre 0.78 d'AUC en test. La colonne V (dans la figure 3) correspond au facteur d'accroissement de vitesse pour atteindre ce taux, relativement au modèle de référence où C=0 (un unique client par tour). On constate que le modèle est deux fois plus rapide pour atteindre le seuil d'AUC en test en passant d'un (C=0) à dix clients (C=0.1) sollicités par tour de communication. En revanche, au-delà de ce nombre, le gain de performances obtenu en augmentant le nombre de clients sollicités s'estompe, comme on peut l'observer sur la Figure 4. Afin de tirer bénéfice du parallélisme multi-client, tout en conservant un bon compromis entre performances statistiques et charge de calcul, fixer la valeur de C à 0.1 semble être un choix judicieux.

Note de lecture des résultats : le gain en temps d'apprentissage qui se limite à un facteur 2 n'est pas mesuré en temps lors de la comparaison au "non parallèle", on mesure ici le nombre de tours de communication C pour différentes hypothèses de parallélisation.

5.1.1 Charge locale de calcul

Deux paramètres locaux influent sur la charge de calcul imputée aux clients : la taille des batchs B et le nombre d'itérations (epochs) E. Afin d'accroître la charge locale de calcul (performance), on peut à la fois réduire la valeur de B et augmenter celle de E. Nous avons au préalable réalisé une distribution "IID_weighted" des données, c'est-à-dire une distribution IID en recourant à un dictionnaire de poids ("class_weight") afin de rééquilibrer la représentativité des données. Nous avons donc réalisé diverses expériences en faisant varier à la fois les valeurs de B et de E, les autres paramètres étant fixées : K=100, C=0.1, η=0.001, L_R=0.5. Nous avons réalisé ces expériences pour B=12, B=51 et B=612 (full-batch), avec E=1, puis E=10 avec utilisation d'un early stopping.

On observe sur la Figure 5 que le modèle est plus performant lorsque l'on réduit la taille des batchs et que l'on augmente le nombre d'itérations réalisées en local.

5.2 Distribution des données

Dans le contexte de l'ubiquitous datamining, les données sont susceptibles d'être hétérogènes en fonction de leur source. Nous avons donc utilisé divers modes de répartition des données afin d'étudier l'influence de l'hétérogénéité de la distribution sur l'apprentissage global. Nous nous intéressons à l'hétérogénéité des données (IID ou non-IID), et au déséquilibre de leur répartition en termes de quantité. Pour réaliser une répartition équilibrée, les données sont réparties en quantités équitables entre les différents clients. Une répartition déséquilibrée

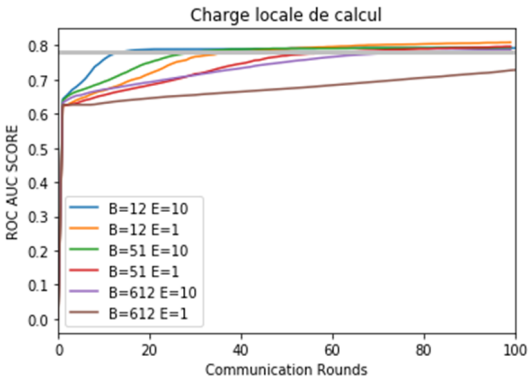

FIG. 5 – *Effet de la charge locale de calcul sur l'apprentissage global.*

consiste à affecter des quantités de données différentes aux clients (voir Figure 6). Pour ce faire, la quantité de données affectées à chaque client est définie par un tirage pseudo-aléatoire. Une répartition équilibrée (au sens uniforme) en quantité consiste à affecter le même nombre de données à chaque client, contrairement à une répartition déséquilibrée (unbalanced) pour laquelle cette quantité est définie pseudo-aléatoirement.

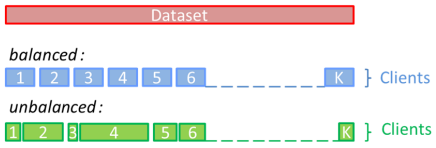

FIG. 6 – *Distribution balanced / unbalanced*

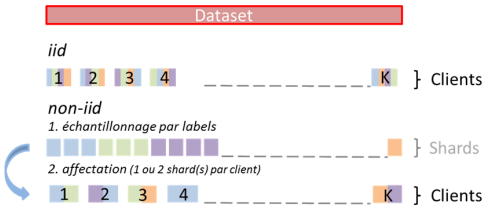

FIG. 7 – *Distribution IID / non-IID.*

Pour obtenir une répartition IID, les données doivent être mélangées suivant les classes avant d'être affectées aux clients. Le jeu de données est ensuite fractionné. À l'inverse, la répartition la plus hétérogène (non-IID) consiste à affecter des données d'un unique label à chaque client. Pour ce faire, nous procédons à un tri des données suivant les labels, puis nous divisons les données en échantillons ("shards"). Un "shard" est un sous-échantillon du jeu de données complet de label unique. Nous affectons ensuite aléatoirement un échantillon à chaque client (voir Figure 7). Ainsi, chaque client non-IID ne possède que des données d'un unique label dans son jeu local. Les clients IID possèdent 2 labels.

Ce type de distribution peut sembler modéliser une vision "extrême" du problème. Dans un cas tel que celui des données provenant des smartphones, on peut s'attendre à une surreprésentation de certaines classes dues à un usage propre à l'utilisateur. Toutefois, cela n'implique pas nécessairement la non-représentation totale d'autres classes parmi ses données.

Une distribution non-IID “1 shard” a été réalisée en triant préalablement les données suivant les labels, avant de constituer 100 sous-échantillons (shards) de 612 exemples chacun. Chaque sous-échantillon constitue le jeu de données d’un client. Ainsi, en moyenne, un unique label est représenté dans chaque jeu d’entrainement client. Une distribution “partiellement IID” (25% IID) a été effectuée en combinant les deux modes de distribution précédents. Ainsi, 25% des données ont été affectées de manière IID, les données restantes étant réparties de la même sorte que pour la distribution “1 shard”. Le choix de ce coefficient modéré s’explique par le faible nombre d’exemples labellisés “churn”. Il permet à plusieurs clients d’avoir une représentation de “churn” plus importante. Utilisé un coefficient plus fort reviendrait à s’approcher d’une distribution non-IID (sous formes de “shards”), puisque la quasi-totalité des clients ne possèderaient pratiquement que des exemples de la même classe.

Ensuite, en raison du côté très déséquilibré des données, nous avons réalisées ces mêmes répartitions en tentant de rééquilibrer la représentativité des classes. Pour le mode de répartition IID, nous avons donc utilisé un dictionnaire de poids (“class_weight”), afin de repondérer les données localement lors de l’entrainement du modèle. Nous appelons cette expérience “IID_weighted”.

Nous avons donc entrainé de manière fédérée notre modèle à partir des différents modes de distributions décrits ci-dessus. Afin de comparer les résultats obtenus en fonction des distributions, les valeurs des paramètres du modèle ont été fixées : K=100 ; C=0.1 ; B=12 ; E=10* (avec early stopping) ; η = 0.001 ; L_R=0.5.

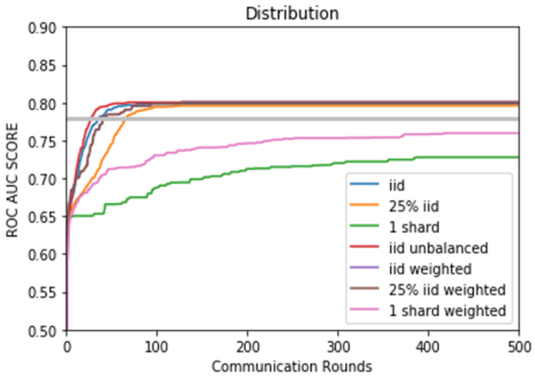

FIG. 8 – *Apprentissage distribué en fonction de la distribution des données.*

La figure 8 présente les résultats obtenus de l’apprentissage fédéré des données de churn en fonction du type de distribution. Les courbes représentent pour chaque modèle l’AUC de test en fonction du nombre de tours de communication. La ligne horizontale grise représente le seuil de 0.78 d’AUC en test. On observe que plus la répartition des données présente un caractère non-IID, plus les performances sont moindres. Cela implique que les performances du modèle sont plus aléatoires. Cette pondération au niveau global permet même à la distribution de type partiellement IID de surpasser les performances maximales de classification obtenues avec la distribution IID. Aussi, on remarque que l’hétérogénéité de la quantité de données détenues par chaque client ne nuit pas aux performances du modèle, dépassant même celles du modèle de référence (à partir de la distribution “IID”).

6 Discussion et Conclusion

Les expérimentations réalisées dans le cadre de cette étude nous ont permis de simuler le federated learning modifié pour les problèmes de classification supervisée déséquilibré. Nous avons mis en œuvre cette approche sur un problème et des données d'opérateur provenant du Groupe Orange.

On en retire un certain nombre d'enseignements :

— Les performances statistiques constatées au cours de ces expérimentations sont intéressantes, atteignant les performances obtenues par un apprentissage centralisé.

— Nous avons pu constater l'intérêt de tirer profit du parallélisme multi-client, ne nécessitant toutefois pas de solliciter l'ensemble des ressources à disposition. L'optimisation de la charge locale de calcul, mobilisant les ressources locales, permet d'accélérer significativement l'apprentissage du modèle global.

— L'hétérogénéité en termes de quantité de données détenues par chaque client ne nuit pas aux performances du modèle, lui profitant même dans certains cas. Concernant l'hétérogénéité de la représentativité des classes dans les données des clients, nous avons pu constater qu'elle pouvait contraindre l'apprentissage du modèle global, nécessitant davantage de temps ce qui peut dégrader les performances prédictives. Toutefois, sauf dans les cas les plus extrêmes, des performances de classification restent satisfaisantes.

Il faut néanmoins noter que les expérimentations réalisées dans le cadre de cette étude n'ont permis que de simuler l'aspect distribué de l'apprentissage. Ces travaux n'ont pas en effet fait l'objet d'une réelle mise en application, ne permettant donc pas d'appréhender pleinement la problématique des autres coûts : coûts de communication dans un environnement réel, contraintes matérielles et logicielles. Le federated learning demeure aujourd'hui au stade expérimental et n'a encore été appliqué que dans des problèmes d'apprentissage supervisé. Son utilisation en apprentissage non-supervisé pourrait être une piste de recherche intéressante.

Enfin même si pour le nouveau règlement européen sur la protection des données personnelles l'apprentissage distribué va dans le bon sens (car il n'y pas de recueil explicite et centralisé des données personnelles) il ne garantit pas totalement le respect de la vie privée car certaines informations sensibles pourraient tout de même être extraites des modèles partagés. Il offre tout de même une première étape technique vers des méthodes plus robustes telles que la *local differential privacy* Duchi et al. (2014).

Références

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, et X. Zheng (2015). TensorFlow : Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bhaduri, K., K. Das, K. Liu, H. Kargupta, , et J. Ryan (2008). Distributed data mining bibliography. https ://www.csee.umbc.edu/ hillol/DDMBIB/ddmbib.pdf.

Castanedo, F., G. Valverde, J. Zaratiegui, et A. Vazquez (2014). Using deep learning to predict customer churn in a mobile telecommunication network. `http://www.wiseathena.com/pdf/wa\_dl.pdf`, last visited November 2018.

Duchi, J. C., M. I. Jordan, et M. J. Wainwright (2014). Privacy aware learning. *J. ACM 61*(6), 38 :1–38 :57.

European Union (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union L119*, 1–88.

Ioffe, S. et C. Szegedy (2015). Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *ICML*, Volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456.

May, M. et L. Saitta (2010). *Ubiquitous knowledge discovery : challenges, techniques, applications*, Volume 6202. Springer.

McMahan, B. et D. Ramage (2017). Federated learning : Collaborative machine learning without centralized training data. `https://research.googleblog.com/2017/04/federated-learning-collaborative.html`, last visited November 2018.

McMahan, H. B., E. Moore, D. Ramage, S. Hampson, et al. (2016). Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv :1602.05629*.

Park, B.-H. et H. Kargupta (2002). Distributed data mining : Algorithms, systems, and applications. pp. 341–358.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, et R. Salakhutdinov (2014). Dropout : A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*(1), 1929–1958.

Summary

Smartphones are ubiquitous in our daily lives. They form an easy-to-reach computing resource with a direct access to a considerable amount of personal information. They represent a highly valuable source of data for telecom operators, but their highly decentralized nature and the evident customer's expectations regarding privacy require new statistical learning approaches. Ubiquitous datamining, by including the device's ability to process their own data locally constitutes an interesting alternative to massively centralized data analysis. Federated learning is a realization of ubiquitous datamining intended to deploy the training of neural networks on smartphones. We propose here an experimental evaluation of this distributed learning approach on mobile operator data.