Master Mathématiques Appliquées

Modélisation Statistiques et Stochastiques

Rapport de Stage

Optimisation de campagne mail

Estelle RAMBIER

Maître de Stage : Laurent Vézard
Tuteur : Jérémie Bigot

01 septembre 2017

Remerciements

Mes remerciements viennent en premier lieu à Laurent Vézard mon maître de stage. Merci pour ta bienveillance et ta confiance. Je te suis reconnaissante des missions que tu m'as confiées : toujours intéressantes et valorisantes. Un grand merci également pour ta relecture minutieuse de ce rapport.

Merci aussi à Romain Garnier, mon voisin de bureau. Ton aide a été précieuse dans des moments délicats, et ton humour a su me redonner le sourire quand le Shell script me faisaient passer des moments obscures.

Merci plus généralement au reste de l'équipe Data Science Florent et Hubert, pour m'avoir chaleureusement intégrée, et à Véronique Saudrais pour ses nombreux conseils.

Et enfin merci à Stéphane Zittoun de son accueil au sein de son entreprise.

Table des matières

1 Présentation de l'entreprise

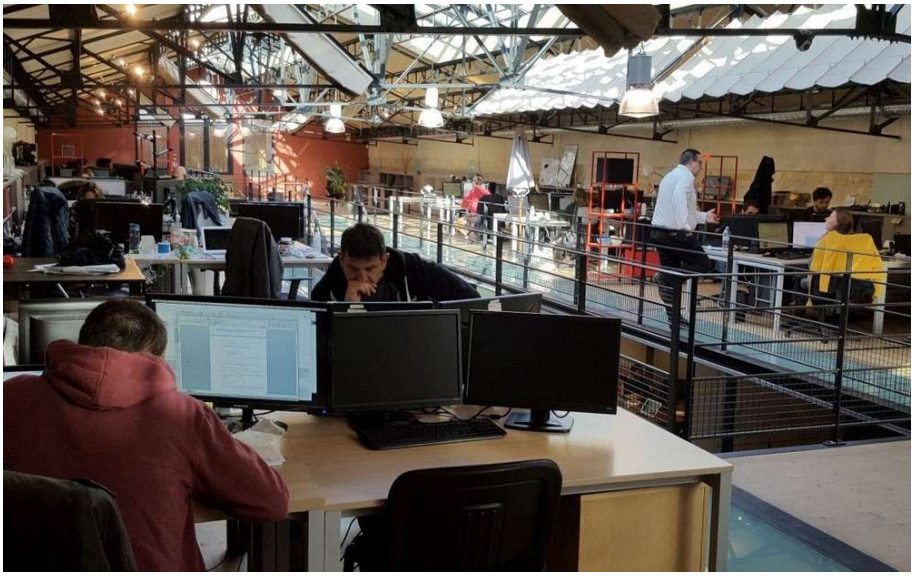


FIGURE 1 – Locaux sur le quai des Chartrons

Le groupe NP6 a été fondé en 1999 à Bordeaux par Stéphane Zittoun. Le groupe est composé de deux compagnies : NP6 Solutions et NP6 Consulting. NP6 Solutions, basée à Bordeaux, est une entreprise de marketing numérique qui édite un logiciel distribué en SaaS (software as a service), permettant à ses clients de piloter leurs campagnes de marketing numérique via une Plateforme marketing d'activation de données. NP6 Consulting est basée à Paris et propose des prestations de consulting scientifique auprès de grands comptes. En 2016, NP6 acquiert Ezakus, pionnier de la Data Management Platform, expert de la qualification et du ciblage d'audience.
Le groupe NP6 est composé d'environ 120 personnes au total. Avec plus d'une centaine de clients, NP6 est un leader européen sur le marché des logiciels de marketing numérique. NP6 Solutions est composé de plusieurs départements : Commerce, Produit, Marketing et communication, Délivrabilité, Production, Support Client, Exploitation, Recherche et Développement et enfin Data Science.

2 Introduction

J'intègre l'équipe Data Science le 06 mars 2017 pour 6 mois. L'équipe est dirigée par Laurent Vézard mon maître de stage, et composée de Romain Garnier, Florent Pigeon et Hubert Alcin.
Les enjeux de l'équipe s'articulent autour de plusieurs sujets, comme par exemple : l'amélioration de leur bibliothèque de Machine Learning, un système de recommandation de produits, un Workflow scalable de scoring. Outre ces sujets plutôt opérationnels, l'équipe traite également des sujets de fond dont le potentiel semble intéressanT, comme par exemple l'intégration des données issues de l'Open Data.

J'ai rédigé dans ce rapport l'ensemble des travaux effectués pendant mon stage. Pour introduire succinctement les thèmes abordés dans les différentes parties : la première partie touche à la mise en place d'un test statistique, la deuxième partie décrit l'usage de méthodes d'apprentissages automatiques. La troisième se concentre sur la construction de la matrice d'entrée des modèles, la quatrième présente mon travail sur un modèle de recommandation produits et enfin la dernière présente une approche visant à déterminer la taille optimale d'échantillons destinés à l'A/B/n testing.

3 Analyse de l'effet d'exposition à une campagne mail

Les clients de NP6 cherchent à déclencher des achats chez les détenteurs de carte de fidélité au travers d'expositions à des e-mails. Une campagne mail est dite performante si d'une part les destinataires du mails ont cliqué sur les liens présents dans ce mail, et si d'autres part, leurs clics sont suivis d'une dépense en magasin. Dans ce cas précis il s'agit de Brico Dépôt. Mon travail s'inscrit dans un contexte de campagne importante pour Brico Dépôt. Mes collègues ont cherché à cibler les individus les plus appétants à commencer un chantier cuisine.
La problématique de mon travail est d'évaluer l'effet de l'exposition de ce mail sur la dépense du client sur les produits catégorisés comme cuisine. Je décrirais d'abord les données ainsi que leurs accès, avant d'exposer le test choisi.

3.1 Les données

Les données sont stockées dans un *cluster* Elastic Search, permettant de stocker un grand nombre de document et requêtage rapide basé sur l'expression de conditions. Elles sont stockées sous forme de documents. Un document par couple client NP6 - prospect. On y trouve toutes les informations collectées par la plate-forme NP6 et par notre client à propose de son prospect.
Un document est formé d'une liste de JSON, format idéal de transport de données. Peu verbeux et simple à décrire dans ses avantages, ce qui le rend lisible aussi bien par la machine que par nous. C'est un format très utilisé dans les formats de retours d'API. Il contient soit un ensemble de clés-valeurs, soit une liste ordonnées d'élément, soit une composition des deux, ce qui est le cas ici. Notre json est composé d'une ligne par individu, composée elle même de 10 objets dont par exemple des infos sur les passages en caisse, les campagnes mails envoyées ou des infos. Par exemple, voici les données du champ **"crm"** qui correspond aux données remplies lors de la souscription à la carte de fidélité :

```
▼ "crm": {
    "long#WeeklyEmailMarketingPressure": 0,
    "email#F849": "geoffreyvallee@hotmail.fr",
    "date#InsertionDate": 1472440680000,
    "string#F813": "10390",
    "string#F814": "32 RUE DE LA MOUE DU MOULIN",
    "string#F811": "CLÉREY",
    "string#F855": "BARBEREY ST SULPICE",
    "string#F812": "FR",
    "string#F856": "Du lundi au samedi de 7h à 19h30 sans interruption.",
    "long#F1422": 1,
    "string#F853": "RN 19<br><br>",
    "string#F810": "GEOFFREY",
    "string#F854": "10600",
    "long#F12629": 2,
    "long#DaySendingsEmailMarketingPressure": 0,
    "string#F852": "03.25.71.25.30",
    "long#F12628": 2,
    "long#F12585": 1,
    "date#F12586": 1472083200000,
    "date#F1419": 675216000000,
    "long#F826": 1,
    "string#CustomerGuid": "02c",
```


FIGURE 2 – Exemple de Json Array sur donnée crm (Customer Relationship Management)

Parmi celles ci, nous disposons aussi des informations sur chaque passage en caisse de chaque individus. Chacun d'eux sont détaillés comme suit. Entre chaque accolade, on a la description d'un produit. Les passages en caisse sont identifié par la clef **"string#tkt_id"** et daté par **"date#tmst"**. Le produit est reconnaissable notamment par son ID à travers la clef **"string#c_materl"**, son prix par **"double#ca_ttc"**, sa famille de produit par **"string#c_famille"**... :

```
▼ "crm_events": [
    {
        "double#qte": 1,
        "string#c_pa_poco": "10390",
        "string#c_pa_pays": null,
        "string#matl_group": "GM801180",
        "string#rpa_tix": "1016",
        "double#marge": 4.66,
        "string#c_materl": "842303",
        "string#c_pa_comm": "00000",
        "double#ca_ttc": 8.2,
        "string#c_famille": "F80120",
        "string#c_plant": "1742",
        "string#consumer": "11499397",
        "double#rt_salhour": 18,
        "date#tmst": 1484067600000,
        "date#calday": 1484002800000,
        "string#tkt_id": "1805ff5010cab2160c95396429a0e678",
        "string#c_rayon": "R80",
        "double#remise_co": 0,
        "string#type_achat": "C"
    }
```


FIGURE 3 – Exemple de stockage d'un produit sur un ticket de caisse

De même, nous stockons les informations sur les e-mail. Cette fois, entre chaque accolade on trouve les informations relatives à une action du client par mail. On y retrouve l'*ID* à travers la clef **"string#unique_key"**, la date de l'action associé à l'utilisateur/ mail **"date#tmst"**. L'action en question est décrite par le champ **"string#type"** par les niveaux suivants : réception **"received_payload_lite"**, ouverture **"open_payload_lite"** ou clic sur le mail **"clic_payload_lite"**.

```
▼ "email_events": [
    {
        "string#type": "received_payload_lite",
        "date#tmst": 1462442400000,
        "string#unique_key": "56269c7dd0cc6bb62639ff3463df443a",
        "string#agency_id": "TAZZ",
        "string#customer_id": "02C",
        "string#action_id": "000KAO",
        "string#target_id": "000QBYYA"
    }
    ,
    {
        "string#type": "open_payload_lite",
        "date#tmst": 1462536000000,
        "string#unique_key": "56269c7dd0cc6bb62639ff3463df443a",
        "string#agency_id": "TAZZ",
        "string#customer_id": "02C",
        "string#action_id": "000KAO",
        "string#target_id": "000QBYYA"
    }
```


FIGURE 4 – Deux exemples de stockage d'information mail

Une seconde chose à relever sur les données est le volume : il y a plus de 2 200 000 individus dans la base, chacun d'entre eux aillant un certain nombre d'historique d'achats et d'envoi de mail. Ceci nécessite un exécution distribuée sur plusieurs serveurs (*Cluster*).

3.2 Méthodologie

Tout d'abord, j'effectue un premier traitement afin de dégrossir le volume et exclure les clients professionnels de Brico Dépôt (à la demande du client). Elastic Search contient un moteur de recherche et d'analyse distribué, capable de résoudre un grand nombre de requête, notamment sur les données telles qu'elles sont stockées ici. Voici un exemple de requête :

```
{"filter":{"bool":{"must":[{"term":
{"agency":"TAZZ"}},{"term":
{"customer":"02C"}},{"range":
{"tags.sampling_selector":
{"gte":0.9999}}}]}}}
```

Un filtre Elastic Search est une arborescence d'opérateurs logiques (terms,

must, and, or, qe...) qui permet d'exprimer des conditions assez fines que les données doivent réunir pour être extraites.

L'idée originale étant de me familiariser avec les outils utilisés par l'équipe, je réaliserai le même traitement d'abord en Python puis en java. Puis j'ai cherché à proposer une approche rigoureuse par test statistique. Celui-ci doit établir si l'exposition à un mail a eu une influence sur les dépenses en magasin sur les catégories de produits visées par l'e-mail. La difficulté réside dans l'extraction des échantillons d'intérêts.

Map/Reduce

Le volume important des données parmi lequel je doit récupérer les informations ne permet pas un traitement traditionnel. Le Map/Reduce est un patron d'architecture de développement informatique, inventé par Google[4]. Celui ci permet des calculs parallèles, qui sont distribués sur un groupement de serveur appelé *cluster*. Le *cluster* de NP6 possède 12 serveurs.

Les termes *map* et *reduce* viennent des deux principales tâches contenues dans l'algorithme. Sur chaque serveur du *cluster* est d'abord appliquée une fonction *map* permettant de générer les données sous forme de tuples : clef, valeur. Puis, à partir des sorties du mapper, la fonction *reduce* combine les tuples pour les réduire en ensemble plus petit. L'exemple suivant résume assez bien les différentes étapes de l'algorithme. Je ne rentrerai pas plus dans les détails.

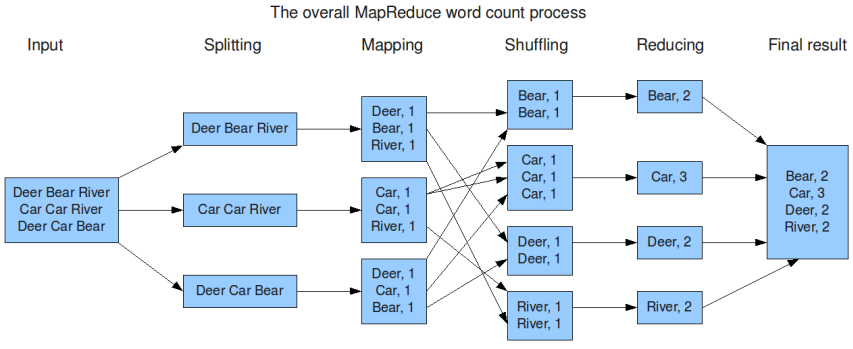


FIGURE 5 – *source :* : http ://www.milanor.net/

Par de simples programmes de comptage en Python et en Java, et épaulée de Romain en Python et Laurent en Java, je me suis familiarisée avec les données et les concepts de map/reduce. Afin de tester, je m'étais constitué un petit échantillon capable de tourner en local et sur lequel je pouvais vérifier mes résultats. C'est ainsi, que j'ai fini par réussir à obtenir ce qui nous intéressait : les montants d'achats sur les produits d'intérêts dans une certaine fenêtre de temps avant et après ouverture/envoi/clic sur une certaine campagne mail.

Le test

On souhaite établir si le montant des achats est plus grand après qu'avant l'exposition au mail. Cela revient à comparer deux populations appariées. Notre choix s'oriente vers le test de Wilcoxon car étant non paramétrique, il nous permet de nous affranchir de l'hypothèse de normalité, malgré le peu de littérature concernant son utilisation avec des données nombreuses. L'idée de base du test des rangs signés de Wilcoxon [1] est d'utiliser la somme des rangs des distances à l'origine des observations positives lorsqu'on ordonne ces n distances : W_n^+. On construit notre échantillon en soustrayant les montants dépensés après l'envoi de l'e-mail moins ceux dépensés avant. Les hypothèses classiques à tester sont la nullité de la médiane :

$$\mathbf{H_0} : m = 0 \qquad \mathbf{H_1} : m > 0 \; ou \; m < 0$$

Cherchant à exhiber une significativité de différence, on cherche plutôt à tester

$$\mathbf{H_0} : m \leqslant 0 \qquad \mathbf{H_1} : m > 0$$

La littérature est unanime : pour les tests unilatéraux il est équivalent de tester $\mathbf{H_0} : m \leqslant 0$ contre $\mathbf{H_1} : m > 0$ et $\mathbf{H_0} : m = 0$ contre $\mathbf{H_1} : m > 0$ [2] [3]. On peux facilement s'en convaincre en regardant la forme de la région de rejet et le comportement de W_n^+ sous les différentes hypothèses.

La librairie Scipy en Python permet de calculer facilement les quantiles des lois de Wilcoxon, de même en R. En Java, les calculs de p-valeur se sont avérés un peu plus corsés. La classe externe existante ne permettait pas le calcul unilatéral et donc pas le contrôle de l'alternative comme en R par exemple. Il a fallu, non sans mal, que je plonge dans le code de la classe

"WilcoxonSignedRankTest" (du *package* Apache Commons Math 3.4) afin de modifier des sorties et ainsi de retourner le calcul unilatéral sous l'alternative m plus grand que zéro. Je retiens que les implémentations à la main nécessite une compréhension dans le cœur du test, impossible de faire l'impasse sur des détails.

3.3 Résultats

Les résultats de la campagne ont fait l'objet d'un *reporting* au client sous forme d'une présentation sur place en présence des principaux interlocuteurs de Brico Dépôt. Dans le but de ne pas noyer le client devant une masse d'indicateur, cet indicateur plus technique n'a pas été présenté.
Mes collègues s'en sont quand même servis pour noter l'impact ou non. Très conservateur sur H0, le test concluait rarement à un effet.

4 Modèle de désabonnement

NP6 propose à ses clients des segmentations de la base donnée des ses clients selon des caractéristiques d'intérêt pour les clients de NP6. Cela consiste à découper la base en ciblant un certain type d'individus. L'idée sous-jacente est de donner la possibilité aux clients de NP6 d'adapter leur communication via l'e-mail aux différents profils de population. Cela peut avoir différentes orientations: limiter la pression marketing sur des population lassées par la réception de mail, booster les ventes de produits en ciblant prioritairement des profils de clients à l'engagement fort (profils de clients intéressés par la réception de mail: les cliqueurs, ouvreurs,...).

Le besoin était de créer un modèle capable de repérer les individus les plus susceptibles de se désabonner appelés "Churner" afin de créer une segmentation. Cette sous population des profils sera ensuite exclues des prochains envois (segmentation dite repoussoir), afin de soulager la pression marketing et ainsi d'éviter de cibler un profil ayant un risque de se désabonner.

4.1 Le modèle

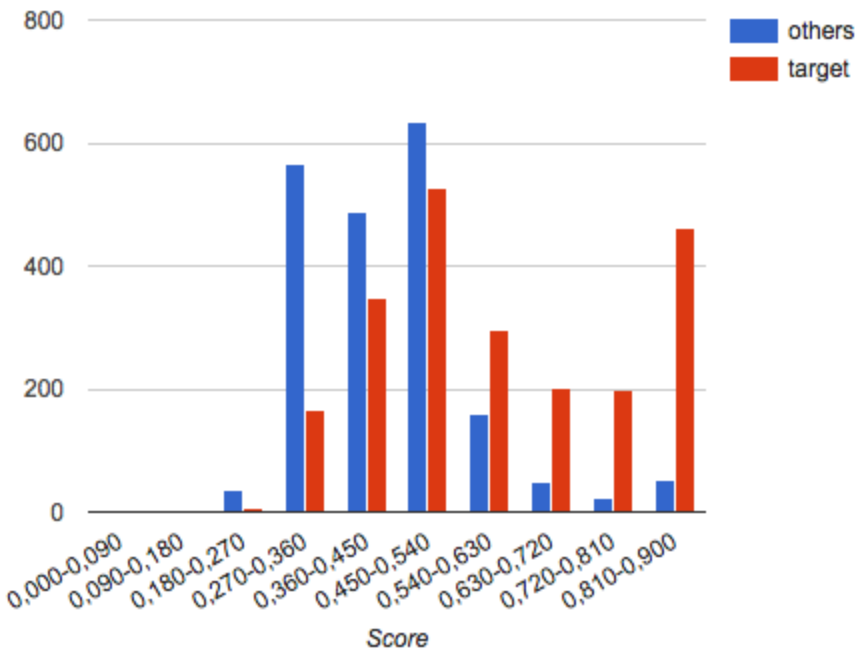
Lors d'une première phase, je devais essayer d'améliorer le modèle construit par mes collègues qui ne permettait pas de bien discriminer les churner (individus les plus susceptibles de se désabonner). Un point important à préciser : l'équipe data science est divisé en deux : les anciens d'Ezakus (racheté par NP6 l'année passée) et les anciens NP6. Des deux côtés, des méthodes ont été développées en Java pour Ezakus et Python pour NP6. La quantité de travail déjà réalisée dans l'élaboration des outils servant à modéliser les problématiques rend le choix entre les deux outils délicat.

La première itération du travail sur la modélisation du Churn a été réalisée en Python (donc du côté NP6). Mon premier objectif sera de comparer les résultats de cette modélisation avec le traitement en Java : la mlLib. La mlLib est le fruit d'un vaste travail rigoureux, développé entièrement par l'équipe Data Science d'Ezakus avant le rachat. La régression logistique y a été codée à la main. Par construction, la mlLib lit une configuration en Json fournie par l'utilisateur. Cette configuration décrit les différentes étapes du processus d'apprentissage du modèle (chargement de la donnée, découpage apprentissage/test, hyper paramètres du modèle, processus de sélection de

variable, processus de validation...). On peux ainsi jouer sur tout les paramètres de la régression : de la pénalisation à la sélection des variables en passant par l'édition d'un Html de mesure de performance du modèle.

Évaluation des modèles

L'évaluation du modèle se fera à travers le *reporting* généré par la mlLib. Via le fichier de configuration de la mlLib, j'ai façonné l'évaluation comme suit. Au sein de la mlLib une première étape divisera le jeu de donnée en 2 : un ensemble de validation et un ensemble test représentant 15% de l'ensemble des données. L'ensemble de validation servira notamment à sélectionner les variables en *backward*, et par validation croisée 4 *folds* élire le meilleur modèle. Puis l'ensemble test évaluera le modèle sélectionné, et éditer le *reporting* en Htlm. Il comprend de nombreux indicateurs. Il y a notamment l'AUC ROC, les cœfficients de la régression devant chaque variables sélectionnées, et une répartition des score. En voici une obtenue avec la meilleure paramétrisation du modèle :



Ce graphe représente la répartition des scores des individus prédis selon leurs statuts réels sur l'ensemble de validation. L'idéal étant d'arriver par le score a bien séparer les deux groupes représentés en rouge pour les churners et bleu pour les autres. Sur cette exemple, on remarque la difficulté du modèle à séparer proprement les deux groupes.

Une problématique émerge: s'il est important de bien repérer les individus susceptibles de se désabonner, il est important d'exclure les individus susceptibles de montrer un intérêt pour le mail, c'est à dire les cliqueurs et les ouvreurs. En effet l'objectif final de ce segment est de permettre d'exclure la population de ce dernier lors des prochains envois. On veux donc éviter d'y inclure des personnes susceptibles d'ouvrir voire même de cliquer sur ces e-mails. Un meilleur AUC sur un modèle n'indique pas nécessairement qu'il est meilleur tel qu'on l'entends : i.e. donnant des scores élevés aux churners et non aux cliqueurs i.e. bien discriminer les deux populations.

N'arrivant à rien de mieux que cette répartition, mes collègues me mettent à disposition les données brutes afin que je construise une nouvelle matrice de données avec des caractéristiques que j'aurai sélectionnées. En d'autres termes, l'idée est d'essayer de dégager des variables (*Features*) discriminant mieux la classe à prédire.

4.2 Construction de la matrice

Ma première confrontation à la donnée brute ne s'est pas faite sans adversité. J'ai mesuré l'importance d'être clair lorsque je demande des données à un collègue. Par défaut de communication, je récupère le tout en format pseudo Json. J'ai découvert plus tard qu'il ne lui aurait pas été plus compliqué de le faire en csv, j'avais juste mal formulé mon besoin.

Essayant dans un premier temps des traitements préliminaires sous Python, quand les différents fichiers représentant 37 millions de lignes se sont avérées emmagasinable par R, j'ai pu rapidement espérer livrer un premier modèle. Cependant, je me heurte rapidement aux limites de R. Ayant mes premières matrices, le Map/Reduce est alors plutôt simple à coder en Python.

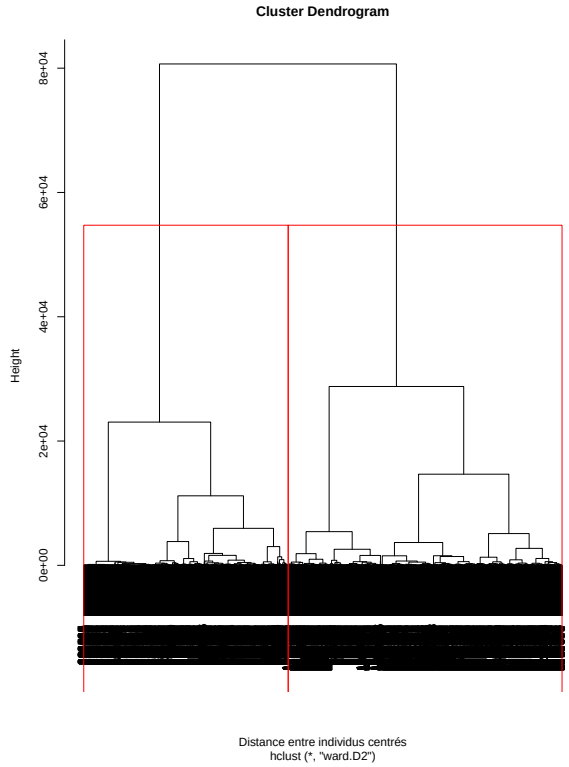
En local tout va bien. J'obtiens l'échantillon de ma matrice de donnée telle que je l'ai souhaitée. Sur le *cluster* tout ne se passe pas aussi bien. Si, sur ma petite extraction de donnée afin de tester en local tout se passe bien, sur l'intégralité de ma base les traitements ne fonctionnent pas. Rencontrant des

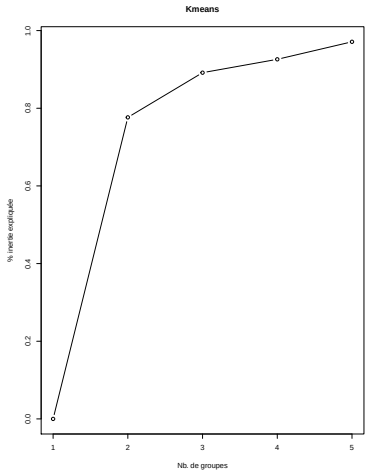
problèmes sur les tris des clés à la sortie du *mapper*, j'intègre donc que le serveur Hadoop ne stock pas les données comme en local. J'ai eue un peu de mal à intégrer que même la matrice d'entrée était stockée en clef valeurs. Ainsi, j'essaie de solutionner mon problème en ajoutant des tris : mon traitement comportait deux *map* et deux *reduce*. Un dernier fichier en Shell script est nécessaire pour orchestrer le tout. Les différents bogues et options du fichier chef d'orchestre ont eu raison de ma volonté. J'ai abandonné l'exécution sur le *cluster* et laissé tourner le code en local.

Et après de nombreuses manipulations en R, j'obtiens ma matrice de donnée avec les variables explicatives telles que je les avais imaginées : des nombres d'envois dans différents intervalles de temps, des taux d'ouvertures et de clics, ainsi que le domaine (Hotmail, Gmail,..) de l'individu et son ancienneté. Du côté du traitement des données, je regrette de ne mettre pas plus accrochée pour arriver à le faire tourner sur le *cluster*, car maintenant, mon modèle n'est pas adapté à une mise en production. Une conséquence est que s'il fonctionne bien, il y aura une étape de transcription en Python ou Java.

4.3 Comportements des churners

Un fois la matrice prête, je me suis lancé dans de rapides analyses de la base. Le but était de prendre en main les données, et d'avoir une idée plus précise sur ce que je devrais trouver.
A travers l'analyse descriptive de la base, on peut exhiber les comportements des churners. Tout d'abord, ils ne constituent pas un groupe homogène : on peut distinguer deux groupes (cf perte d'inertie sur un *kmeans*, règle du coude et CAH).





La comparaison de ces deux groupes délimite des actifs et des non actifs : un des groupes ouvre et clique sur beaucoup plus de mails que l'autre, ceux

sont des individus "jeunes" dans la base. Les individus churners présents depuis longtemps ne sont plus actifs. Il est facile de s'imaginer que ces derniers sont faciles à discriminer des cliqueurs / ouvreurs, les défauts de détection s'apparenteraient plutôt aux churners actifs. Une conjecture, est que si on arrive à diminuer les churners actifs au fur et à mesure, à long terme, on arrivera à fortement diminuer les churners inactif. Les inactifs sont des individus qui ont été beaucoup sollicités et jamais vraiment intéressés, on observe aussi ce comportement chez des individus ni cliqueur, ni churner.

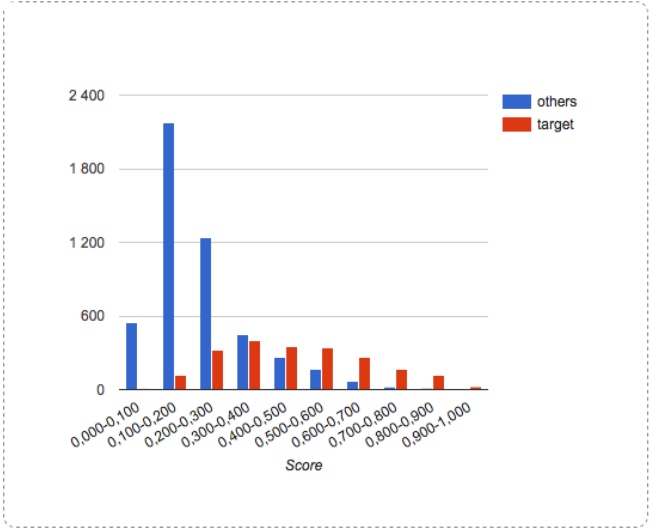
4.4 Mes modèles

Évalution de mes modèles

Dans un premier temps je procéderai exactement comme pour le modèle précédent: division de l'ensemble des données en apprentissage85%/test15%, sélection de variable par validation croisée 4 *folds* sur l'apprentissage, rapport de performance du meilleur modèle sur l'ensemble test.
Ayant cette fois ci le contrôle total sur les données, il devient possible de tester aussi sur des campagnes futures en *off-line*: *i.e* arrêter la collecte des données à un temps t, scorer l'ensemble des individus, et regarder comment fonctionne le modèle sur une campagne envoyée après t.
Mais j'émets tout de même une réserve sur cette façon d'évaluer le modèle. En effet, la règle de décision qui élit un individu comme appétant à se désabonner ne peut par être précise à la campagne près. Révélé par l'analyse de donnée précédente, on sait que des individus se désabonnent bien que identifiés comme cliqueurs et ouvreurs. Si on regarde sur une unique campagne, ils viennent saboter la potentielle bonne discrimination entre cliqueurs-ouvreurs et churner. Par exemple si un churner a cliqué sur la campagne qu'on évalue mais se désabonne sur celle d'après : il sera tagué comme cliqueurs. Il faudrait évaluer sur plusieurs campagnes en enlevant les churners qui ont cliqué sur les précédentes campagnes et qui faussent la juste évaluation du modèle. Comme c'est fait par construction dans la phase de test de la mlLib. Si le test *off-line* est ainsi pessimiste, ce dernier test dans la mlLib est pour les mêmes raisons optimiste puisque on regarde l'ensemble des désabonnements peu importe la campagne.

Qualité du modèle

Sur le plan qualité du modèle, il se comporte plutôt bien sur l'ensemble test. C'est agréable de pouvoir jouer avec les paramètres de la régression aussi facilement qu'offre la configuration de la mlLib. Clairement, sous certaines paramétrisations le modèle a un défaut de sur-apprentissage, conditionné par le paramètre de régularisation. Si on compare ce graphe à celui plus haut obtenu avec l'ancien modèle, on remarque qu'il discrimine mieux les deux catégories d'individus. Ceci est encouragent à essayer des tests *off-line*.



Les figures qui suivent sont les indicateurs clefs des tests *off-line*. Elles sont construites comme suit : on tri par ordre décroissant les individus selon le score donné par le modèle. En abscisse, on lit la part d'individus envoyés sur la base totale, et on lit en ordonnée la part d'individus retrouvés : par exemple, sur la figure 7, lorsqu'on regarde les 4 premiers % des scores, on retrouve 15% des désabonnements de la campagne, 6% des cliqueurs et 3% des ouvreurs. Un bon modèle se caractérise donc par un fort pourcentage de churners parmi les scores élevés avec un bas pourcentage de cliqueurs et d'ouvreurs. La diagonale représentant le pourcentage total de la base. Ainsi, le meilleur modèle en validation croisée donne sur la première campagne ceci :

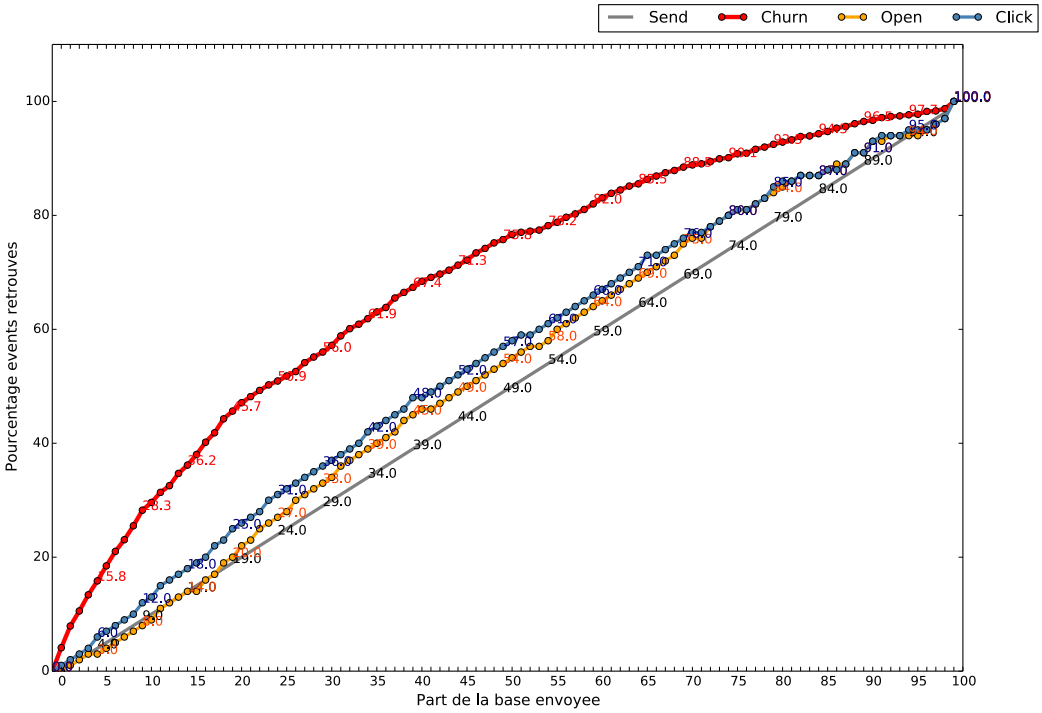


FIGURE 6 – Premier Modèle

En pratique, lors de la création d'un segment repoussoir (comme celui qu'on est en train de construire), on sélectionne que les individus aux scores les plus élevés. On ne veux surtout pas classer en repoussoir des individus qui auraient cliqué. Ainsi, on regarde que les premiers pourcentages de la base. Afin de mesurer les performances, on zoom donc le graphique précédent sur le bas gauche.

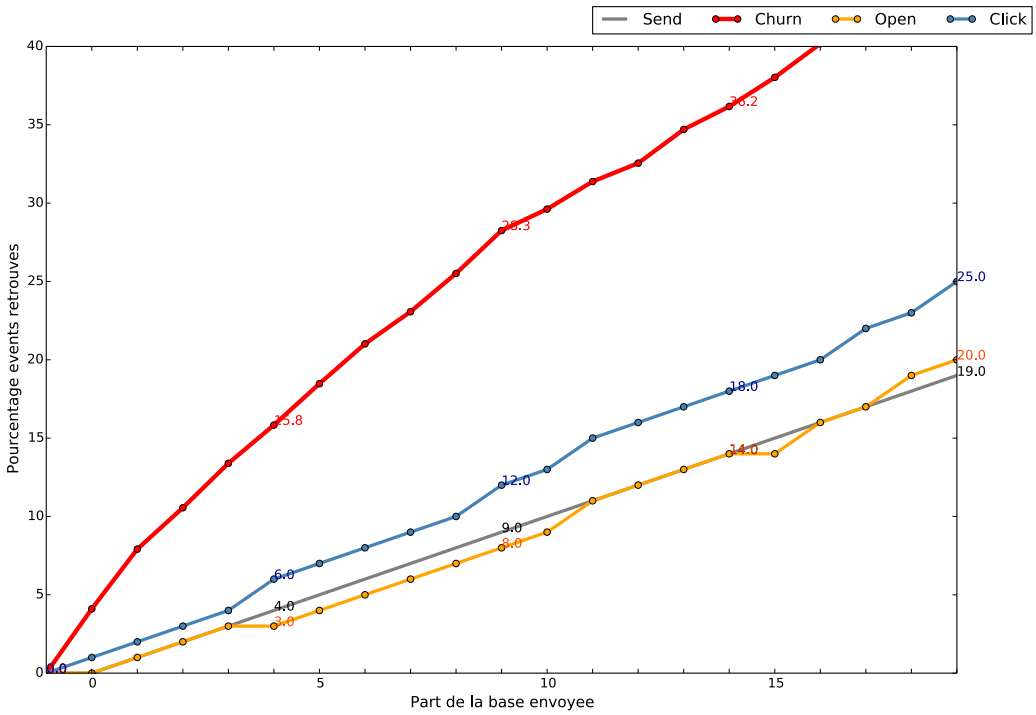


FIGURE 7 – Zoom Figure du dessus

En jouant avec la paramétrisation de la régression et les sélections de variables mon meilleur modèle est le suivant.

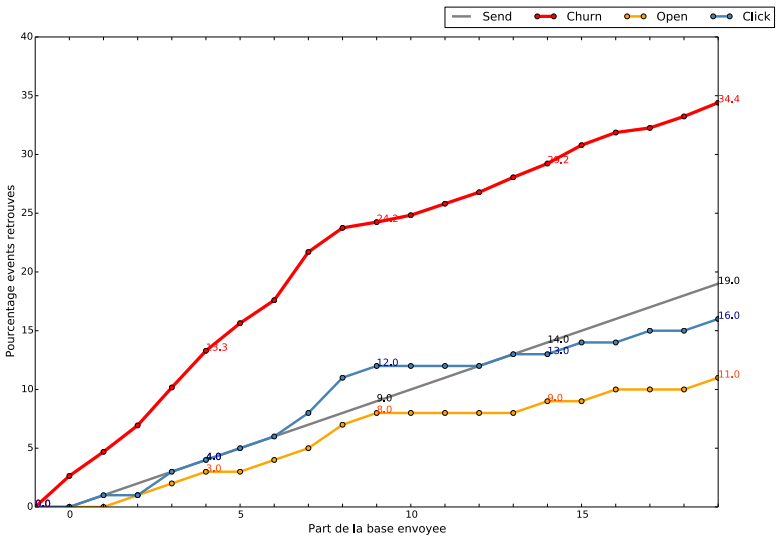


FIGURE 8 – Paramètres optimisés

La sélection rigoureuse des variables gardent toutes les informations sur les nombres d'envoi, le domaine de l'adresse mail destinataire, et le taux de clics dans les plus de 6 jours.

Améliorations

Afin de mieux discriminer les événements, j'explore plusieurs pistes.

J'ai d'abord essayé un peu de *text mining*. Lorsqu'un individu se désabonne, les données brutes m'indiquent sur quel mail il s'est désabonné. Comme je dispose aussi du contenu textuel des mails, je me suis posé la question de savoir si certains mails induisaient plus de désabonnement. La réponse semble être non. J'ai regardé le nombre de désabonnements par mail, et normalisé par le nombre d'envoi. Remarquons que, le tri de ce taux donne le même ordre que le tri du nombre d'envoi (à quelques places près). De plus certaines campagnes mails ont probablement été envoyée après une segmentation choisie de

la base. Mais en s'intéressant au 4 meilleures et 4 moins bonnes campagnes (en terme de désabonnement), et en fermant les yeux devant les biais, les contenus des titres mails associés ne permettent pas de discriminer ces deux populations. On retrouve les mêmes mots clefs dans les deux cas.

Une seconde piste explorée est de modifier mes variables explicatives. Lorsque j'ai construit ma matrice, j'ai choisi des intervalles de temps intuitivement. C'est à dire, par individus : le nombre d'envois, taux d'ouverture et de clic dans les moins et plus de 6 et 15 jours précédents : soit le désabonnement s'il s'agit d'un churner, soit le 15 mars (date d'arrêt de mes données) sinon. J'ai donc essayé d'autres intervalles tels que 10 et 20 jours. On remarquera ici la très forte corrélation entre mes variables explicatives. Ce n'est *a priori* pas un point qui perturberai le fonctionnement de la régression, car les performances se voient légèrement améliorées.

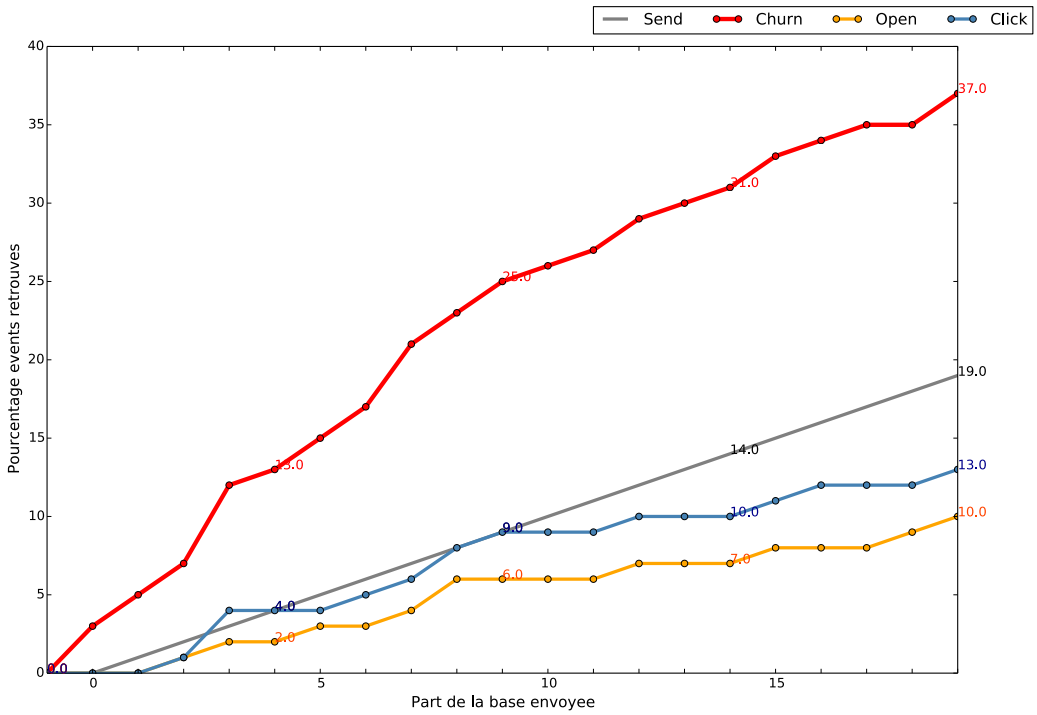


FIGURE 9 – Matrice re-fenêtrée

Une troisième théorie développée fût de faire tourner la régression sur les coordonnées factorielles. La règle du coude me fait choisir les 3 premières composantes principales représentant 60% de variance expliquée. Comme le montre la figure ci dessous, la discrimination entre les individus se fait bien, mais le modèle a plus de mal à retrouver les désabonnement que les modèles précédent.

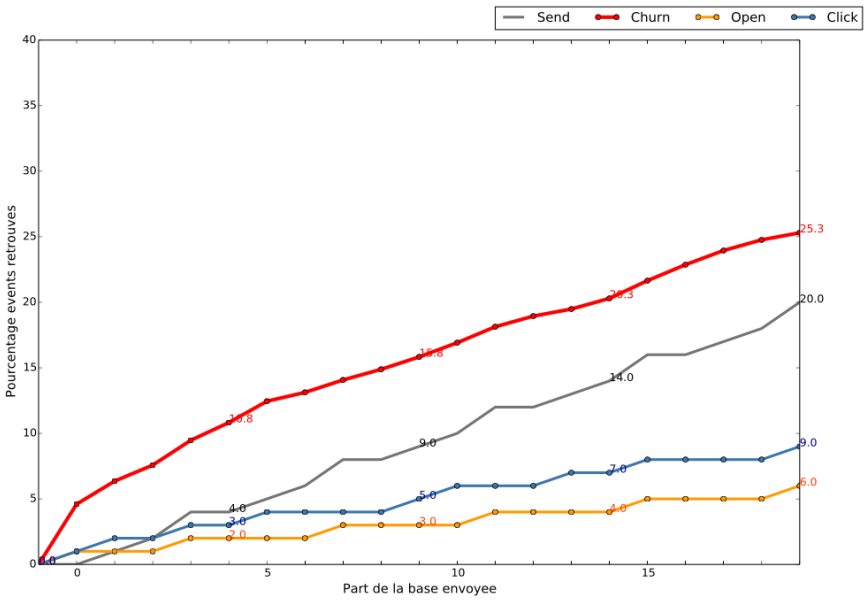


FIGURE 10 – Modèle sur 3 premières composantes principales

Une autre piste d'amélioration était d'inclure dans mon modèle les scores d'ouvertures et de clics des modèles de mes collègues (appétence d'un individus à cliquer/ouvrir). J'ai d'abord essayé des arbitrages tel que la moyenne des scores ou bien le maximum/ minimum. Ceci ne donnait rien de concluant. Ensuite, j'ai tenté de lancer une régression sur les scores : deux variables explicatives le score de clics et mon score de churn. Le modèle nécessite une très forte pénalisation induite par construction. Voici le résultat :

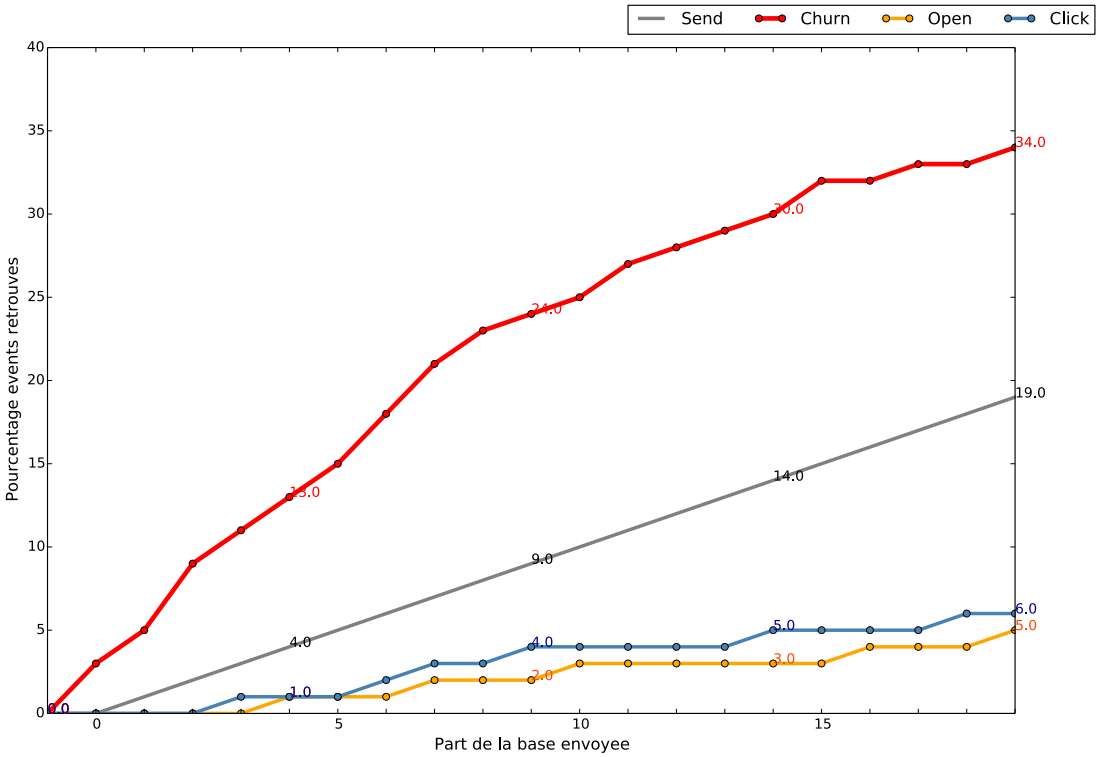


FIGURE 11 – Le meilleur modèle

On trouve dans les 9 premiers pourcentages de mes scores, 24% de churners pour seulement 2 et 4 % d'ouvreurs et de cliqueurs, ce qui correspond à mes meilleurs résultats à ce jour.
Ceci m'a encouragé à essayer des méta-modèles : utilisé le score du modèle comme variables explicatives dans le même modèle. J'ai essayé pas mal de combinaisons mais ça ne fonctionne pas mieux.

4.5 Axes d'amélioration

Parmi les pistes desquels je n'ai pas abouti, le traitement des valeurs manquantes me parait un point pouvant s'avérer concluant. En effet, la matrice

telle qu'elle est construite, ne fait aucune différence entre une valeur 0 sur une variable car l'individu n'a pas fait d'action et un 0 issu du traitement d'une valeur manquante : l'individu n'a pas été sollicité. Cette différence est, à mes yeux, cruciale et représente 14,5% des données. Ce traitement permettrait surtout de refléter correctement la signification des NA, de préserver les aspects des distributions et des relations importantes entre les variables. Il est facile d'imputer la régression logistique car elle ne supporte aucune valeurs manquantes. On peux penser aux arbres qui permettent un traitement élégant. Ou bien s'éloigner de l'apprentissage automatique avec une analyse de survie, mais qui risque de ne pas supporter la charge. La dernière option (la plus simple à mettre en œuvre mais la moins pertinente), est, pour chaque variable, ajouter une variable "is.NA".

Aussi, j'ai pu étudié la dynamique de fuite selon l'historique des comportements face aux mails, il est évident que le désabonnement n'a pas de cause unique. J'aurai aimé avoir accès à des données complémentaires comme les comportements en magasin afin d'enrichir le modèle.

4.6 Conclusions

Le bilan du dernier modèle est positif. Il est bien meilleur que celui qui était réalisé avant mon apport. Le point négatif est qu'il est codé R, ce qui est gênant pour un passage en production. Une intégration passera donc forcement par une translation du code en Java ou en Python selon dans quel projet il sera inclut. Ce qui sera finalement le cas, et un collègue s'en chargera. Mon stage ne sera pas assez long pour constater les premières utilisations clients.

5 Feature Engineering

5.1 Intégration des données de navigation

NP6 dispose de données multi-canal (navigation, information carte de fidélité, historique des actions sur mails ou SMS) stockées en clef valeurs dans un *cluster* Elastic Search nommé en interne ezCore. Lors du rachat, l'idée était de réutiliser les travaux effectués à Ezakus concernant la création de modèles et d'enrichir ces modèles avec les nouvelles sources de données dont NP6 dispose. Ces modèles permettent ensuite de scorer l'audience disponible du client (comme pour le churn). Le *workflow* tel qu'il est aujourd'hui est pensé de façon à construire des modèles à partir de données multi-canal et de stocker les résultats issus du *scoring* dans ezCore. C'est la structure des documents de traiter, la volumétrie et la nécessité à traiter de la donnée provenant de divers canaux qui complique tout ici.
Pour répondre au plus juste à cette problématique d'intégration, je me suis d'abord imbibé de l'architecture du projet avant de me lancer dans l'implémentation en Java et de regarder l'apport dans les modèles prédictifs.

5.1.1 Architecture

L'idée de l'architecture suivante est de pouvoir construire des modèles entièrement codés en Java de façon mécanisée et facilement paramétrable. Le *workflow* s'articule de la façon suivante :

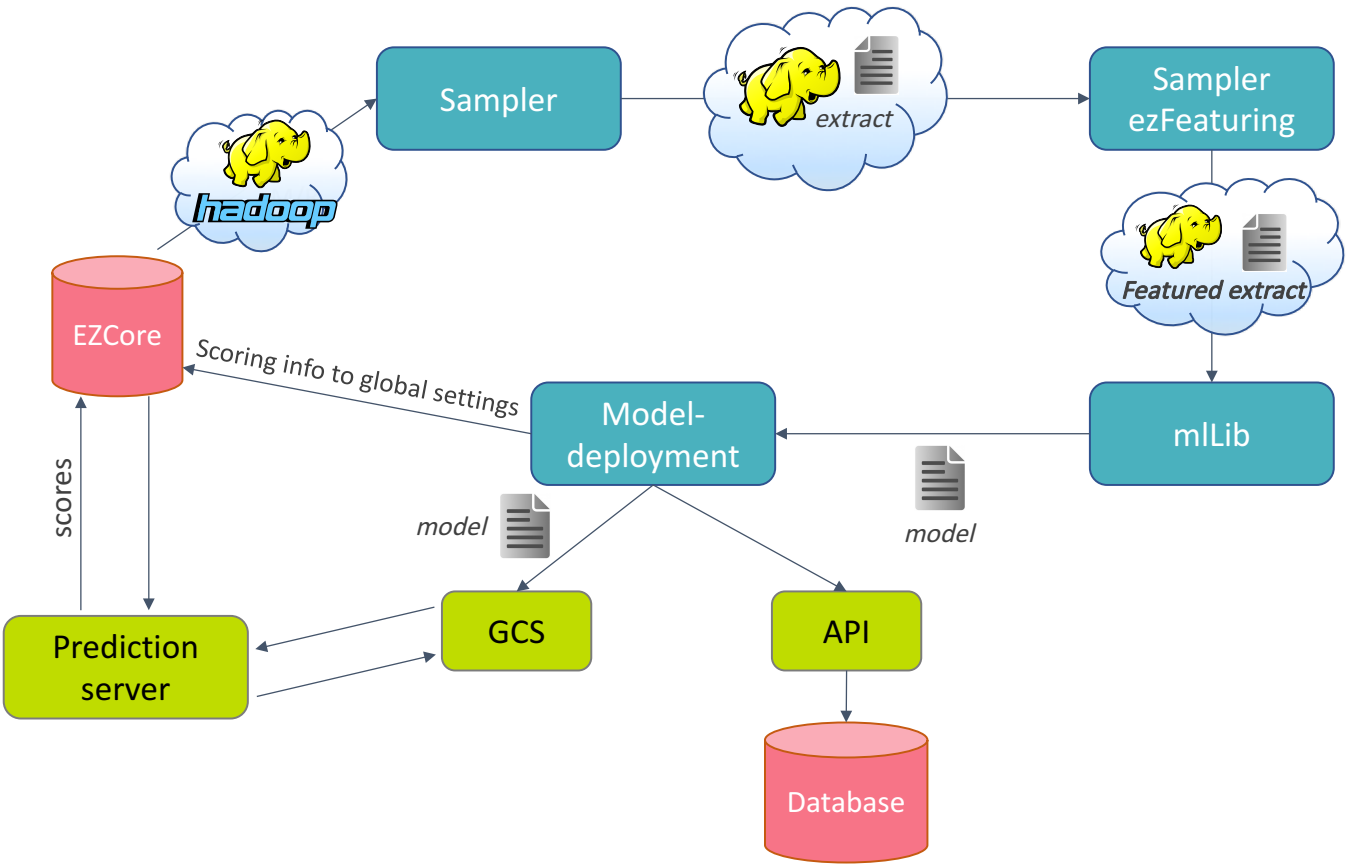


FIGURE 12 – Le workflow

Avec :

Sampler : extraction de la donnée par Map/Reduce. Les critères d'extraction sont exprimés à l'aide de filtres permettant de définir les caractéristiques de la population à extraire. Une fois extraits, les fichiers bruts sont travaillés à l'aide d'un second Map/Reduce, nommé ezFeaturing. Celui ci permet de calculer les *features* (variables explicatives).

mlLib : création des modèles à partir d'un fichier de configuration permettant d'exprimer un scénario d'apprentissage. Les sorties d'une exécution sont le fichier modèle et des rapports contenant les performances calculés sur les échantillons d'apprentissage et de test. Ces rapports, en Html, permettent de juger de la qualité du modèle créé. Ils sont postées sur un serveur Http afin que tous les membres de l'équipe puisse y avoir accès et que la validité du modèle puisse être vérifiée.

Model-deployement : après validation, le modèle est mis en production. Le projet permet de remplir les bases et mettre à disposition des différentes briques de la production le fichier modèle (créé par la mlLib).

Prediction Server (PS) : il s'agit de l'outil qui permet d'obtenir des prédictions. C'est un serveur requêtable via un client embarquable dans les autres projet de la plate-forme ou via des appels Http. Grâce à un appel, le PS récupère dans un dossier local tous les modèles disponibles sur la plate-forme (en particulier ceux qui viennent d'être mis à disposition via le projet Model-deployement (particulièrement les modifications). Il est alors prêt à fournir des prédictions pour ces modèles.

ezCore : L'architecture forme une boucle autour d'ezCore. Celui-ci est utilisé pour extraire la donnée permettant de construire des modèles. Il est également utilisé pour stocker les scores de ces modèles pour chacun des profils connus dans ezCore. L'idée est de se servir des modifications des documents ezCore (*update* d'un document lors de la réalisation d'un nouvel événement concernant le profil) pour recalculer à la volée les scores des modèles. Ainsi, ezCore peut formuler des requêtes vers le PS, traiter les réponses de celui-ci et insérer les scores dans les documents. De plus, lorsqu'un nouveau modèle est mis en production, il faut être capable de lancer une mise à jour forcée de tous les documents afin de calculer les scores sans attendre l'*update* à la volée.

Mon apport se fera sur le traitement des données de navigation qui n'ont pas été touchées depuis le rachat. Les travaux d'extraction et de définition des *features* ont déjà été réalisés chez Ezakus, mais sous un autre format de stockage de la donnée de navigation. Il reste donc à adapter les traitements sur les nouveaux formats de stockage de la donnée de navigation (hit) en

s'assurant que l'implémentation est complètement compatible avec chacune des briques. Ce point est très important. Un erreur pouvant entraîner un arrêt de la plate-forme.

5.1.2 Implémentation

J'ai procédé en deux étapes : j'ai d'abord procédé aux *features engineering* : modifier chacun des traitements afin d'aller taper dans les bons champs (les nouveaux champs), et ajouter pour chaque une méthode permettant de vérifier le bon traitement. Puis dans un second temps, j'ai adapté le *Feature Processing*.

L'ensemble du projet est codé en Java. J'utilise l'ide IntelliJ. L'interface est très complète, et propose de nombreuses fonctionnalités permettant à un débutant de s'en sortir. Pour extraire les *features* de la donnée brute, des méthodes ont été programmées en amont par un collègue. Malgré la quantité de méthodes et de classes intervenant dans les traitements, il est aisé de naviguer entre elles. IntelliJ propose la possibilité d'aller aux déclarations et aux endroits d'appels. Le Graal reste le mode débogue.

Le mode débogue

Je ne serai pas où j'en suis aujourd'hui sans lui. Comme Java est un langage compilé et orienté objet, lorsque le code ne fait pas ce qu'on s'attend à ce qu'il fasse, il est compliqué de savoir d'où cela vient. La notion d'héritage de Java vient rajouter une couche au niveau de complexité. Les classes et méthodes sont par construction imbriquées les unes dans les autres. C'est comme ouvrir un tiroir, dans lequel il y a 10 tiroirs, comprenant chacun d'autres tiroirs.. bref l'enfer lors d'un bogue. Le mode débogue est un *run* spécifique qui prend en compte des points d'arrêt préalablement placés dans le code. Lorsque ces points d'arrêt sont rencontrés à l'exécution, IntelliJ fige momentanément l'exécution pour laisser le temps de voir les valeurs des variables en temps réel d'exécution (et même de les changer). Puis l'exécution peut continuer (ou s'arrêter permettant la correction du bogue). Une infinité de points d'arrêt est possible ce qui permet de marquer à la culotte l'exécution. Le mode débogue est également très utile pour faire un *rétro-*

engeneering: comprendre le rôle d'une partie du code en le voyant en action.

Les tests unitaires

Le plus important hormis le bon traitement des données brutes, est la compatibilité avec chacune des briques du *workflow*. Afin de rendre traitable les nouveaux formats de Hit, j'ai modifié une petite cinquantaine de classe aboutissant à des usages fonctionnels.
Imaginons que dans un an le format change de nouveau. Dans la plate-forme, on aura les anciens hit et les nouveaux. Quelqu'un ré-factorisera mon code pour ajouter les traitements des nouveaux hits. Mais il faudra qu'il s'assure qu'il ne casse pas mon travail, car il se peut que certaines briques fonctionnent encore avec les anciens formats. Ces changements induiraient de re-tester manuellement tous les cas d'utilisation envisageables. Ce qui peut s'avérer laborieux. De plus si on décide d'intégrer un plus grand volume de fonctionnalités avant de refaire les tests, on accroît les risques de bogues. Java propose un outils intéressant : les tests unitaires.

Les tests unitaires sont destinés à tester une seule et unique classe. Cela signifie que si la classe en question fait des appels en base de données ou à un service disposant d'une API, on va utiliser un mock (une classe qui simule le comportement de la classe d'accès à la base de données) afin d'effectuer le test en environnement neutre. Cela implique que les accès en base de données ne sont pas écrits directement dans la classe testée, mais délégués à une classe qui ne fait que ça, et qui est injectée dans la classe lorsqu'elle est construite [8]. Pour une API, le mock permet de contrôler sa réponse dans l'environnement du test unitaire.
Les tests unitaires d'un projet sont exécutés à la compilation de celui-ci. Autrement dit, un test unitaire qui ne s'exécute pas correctement (renvoyant une erreur) rend l'exécution (et donc l'utilisation) du projet impossible. Ainsi, à partir du moment où un test unitaire est construit pour chacune des classes, je peux dormir relativement tranquillement : si une modification de code introduit une régression importante, elle sera quasi systématiquement détectée lors de l'exécution des tests. La limite des tests est qu'ils ne peuvent épuiser l'ensemble des possibilités du programme. Une fois les cas d'école testés, et ceux que mon imagination trouve, il est possible de passer à côté de problèmes non prévus.

5.1.3 Apport dans les modèles

Tout d'abord, notons que pour avoir les données de navigation relatives à un individu, il est nécessaire que celui ci effectue sa navigation sur le site en étant identifié (logué). De cette façon, nous pouvons synchroniser son identifiant de navigation internet via un cookie préalablement posé sur la page. Si l'on prend l'exemple d'un compte client NP6 qui possède aujourd'hui environ 2,2 millions d'individus présents dans la base, ceci représente 16 000 profils seulement. Les modèles en cours de construction sont les modèles de détections de chantier cuisine dont je parlais dans la partie précédente. Deux problèmes se sont posés lorsque Romain m'a aidé à faire tourner les nouveaux modèles utilisant mes variables issues du traitement des hits de navigation. Premièrement, entre temps, une nouvelle version de Java est sortie et a été utilisée sur certains projets format le *workflow* de modélisation. Ces projets auraient nécessité une mise à jour de la version du *cluster* afin que celui-ci soit compatible avec les différents projets utilisant cette nouvelle version de Java. Dans l'attente de cette mise à jour, le *worflow* ne pouvait s'exécuter sur le *cluster*, rendant impossible tout test de mon développement. Et deuxièmement il y a eu d'autres nouvelles intégrations dans le *featuring* qui ont rajouter une couche de bogues. Le temps de mon stage ne sera donc pas suffisant pour observer l'incrément de performance dans les modèles prédictifs après intégration des données de navigation.

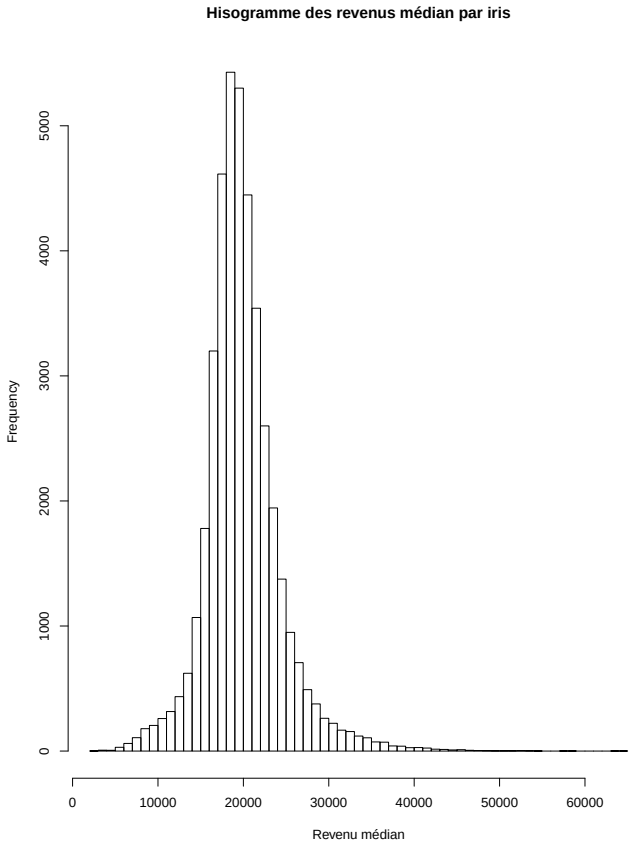
5.2 Prédiction de valeurs manquantes

Grâce aux progrès réalisés en matière d'ouverture des données et de politique dites d'Open Data, l'INSEE a mis à disposition du grand public de nombreuses informations. Ils ont ouvert le découpage de la France en 50 000 Iris, et des informations statistiques sur chacune d'elles. Cherchant à attirer de nouveaux clients et à améliorer les modèles, l'intégration de telles données dans les modèles est à examiner.
Concrètement, l'ouverture de la donnée se traduit par différents jeux de données en csv à télécharger sur le site de l'INSEE. Après une jointure de ces fichiers, on obtient une matrice de 50859 sur 90. Une variable potentiellement intéressante à intégrer aux modèles serait le revenu médian annuel par Iris. Cependant, celui-ci n'est pas toujours renseigné : 8040 données sont manquantes.
Afin d'obtenir une prédiction sur ces lignes manquantes, j'utiliserai le reste des indications sur l'iris : nombre d'habitants, d'hommes, de femmes, de maison, de famille avec deux voitures, de familles à 0,1,2,3,4 enfants, d'abonnements au transport en commun...

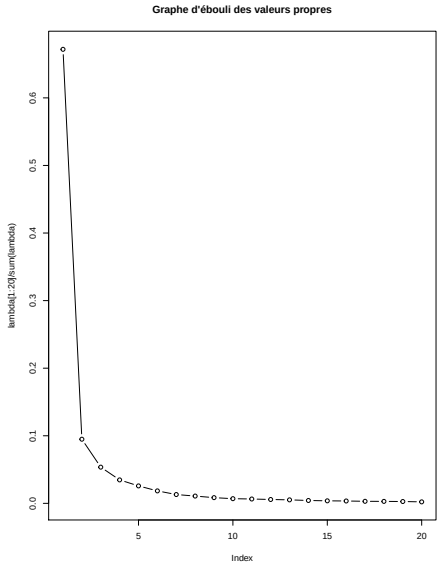
Pour répondre à la problématique de prédiction de ces valeurs manquantes, j'ai dans un premier temps exploré la donnée, avant, dans un second temps, de me lancer dans le choix d'une méthode prédictive.

5.2.1 Analyse préliminaire

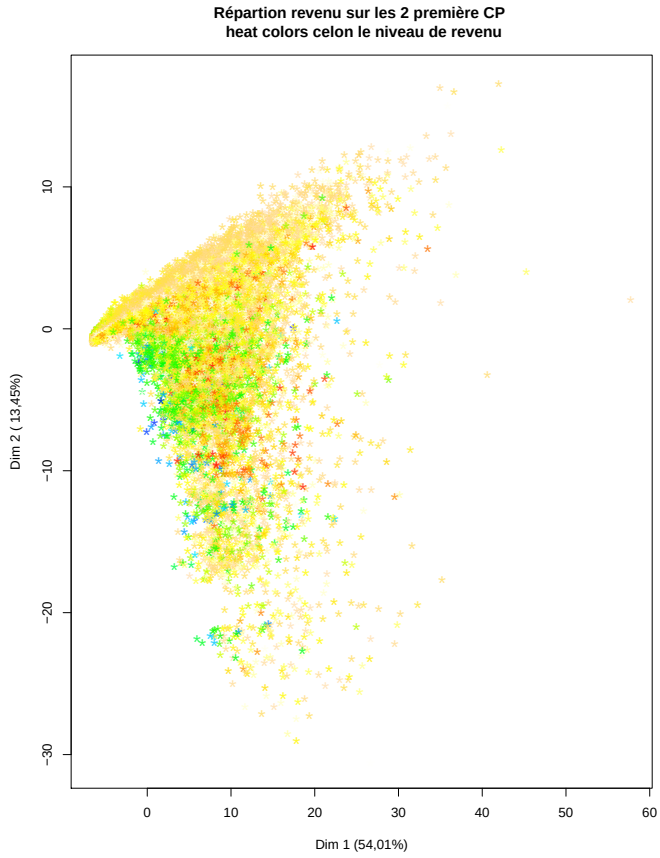
L'exploration statistique de ces matrices m'a fait arriver à plusieurs conclusions : tout d'abord, on pourrait découper la matrice en deux. En effet, par caractérisation des variables, on a une partie dense et une autre creuse. Le choix de la méthode prédictive étant assez corrélé à cette caractéristique, il me paraît important de le relever.
De plus, la répartition des revenus médians semblent pouvoir s'apparenter à une distribution normale.



Je me suis intéressée à la qualité de la représentation en coordonnées factorielles afin d'alléger la dimension de la matrice.



Bien que les deux premières coordonnées comptabilisent 70% de variance expliquée, la discrimination des individus n'est pas évidente : les couleurs sont en fonction du revenu à la façon d'une carte de chaleurs. Les couleurs chaudes (jaunes orange rouge) pour les revenus élevés, et les couleurs froides (noir bleu vert) pour les revenus plus bas.



La séparation de me paraît pas évidente. Cela peux révéler la faiblesse du lien entre les variables explicatives et la variable à expliquer.

On retiendra finalement la matrice sans les données relatives aux transports. On ne constate pas d'amélioration du modèle avec celles ci. On a donc pour l'apprentissage une matrice de 42809 x 63.

5.2.2 Élection d'un modèle

Ce paragraphe expose la démarche de sélection du modèle qui prédira les valeurs manquantes.

Serveur distant et calcul parallèle

Si la volumétrie de la donnée autorise des traitements sous R, les calculs s'avèrent très long pour certaines méthodes. L'équipe dispose d'un serveur

Linux (Mirakulos) sur lequel il est possible de faire tourner des scripts. Théoriquement, cela est simple : il suffit d'accéder à distance au serveur grâce au terminal de commande, puis d'y placer le script et les données, et enfin, de le faire tourner en tache d'arrière plan. En pratique, j'ai rencontré quelques soucis.
Le tout premier obstacle était l'installation de la toute dernière version de R sur Mirakulos. Si il est facile d'installer R, la dernière mise à jour de R était nécessaire à l'usage certaines librairies. Celle ci est absente du repos de la version du système d'exploitation de Debian utilisé ici. Si lorsque il a été installé, c'était la dernière version, depuis il y en a eu d'autre. Le problème est que de nombreuses choses dépende de cette version, la mise à jour n'est pas simple. De plus, comme R est dépendant d'autres logiciels et librairies pas non plus mises à jour, la tâche se complique. Après que Romain se soit arraché les cheveux sur le problème, Alex (responsable de la maintenance des serveurs) de l'infrastructure a finit par contourner le problème et installer tout ce qui manquait à la main. Cela complique légèrement les exécutions de R et les installations des librairies, mais on arrive à nos fins.

Ensuite, comme Mirakulos a plusieurs cœurs, l'étape d'après est d'arriver à faire tourner les calculs en parallèle, très intéressant pour les forêts aléatoires et les réseaux de neurones. Des librairies en R proposent l'implémentation de fonctions permettant de repérer différents cœurs sur un serveur ou encore de fusionner des résultats provenant de différents calculs : "doParallel" et "doMC".

Apprentissage/Test

On cherche le modèle qui prédira au plus juste le revenu médian annuel. Ce sera le modèle dont les différences entre la prédiction et la réalité seront les plus faibles. Je choisi donc de comparer mes modèles selon leurs capacités à minimiser les erreurs absolues : on s'intéressera à la répartition de l'amplitude de celles ci. Par exemple une méthode qui aura bien prédit les données, aura les trois quarts de ces prédictions juste à au plus 1000 euros près. En d'autres termes, on compare des boîtes à moustache de la différence entre chaque prédiction et sa valeur réelle.
Sur chaque méthode, j'ai effectué une validation croisée 10 *folds* afin de respecter le ratio apprentissage/prédiction qu'il faudra reproduire lors de la

prédiction à l'aveugle.

	linear	Lasso	cart	RandomForest	SVMr	nnet	xgboost
Min.	0	0	0	0	0	0	0
1st Qu.	691	683	839	487	439	1075	454
Median	1500	1530	1789	1053	952	2327	973
Mean	2016	2059	2333	1401	1319	3141	1276
3rd Qu.	2704	27720	3169	1903	1729	4347	1756
Max.	28567	21430	33373	23911	25271	23646	23750

Parmi les méthodes testées, la forêt aléatoire, le SVMr et xgboost semblent sortir du lot. Devant le peu de disparité entre leurs résultats, le soin est pris de regarder les performances avec des découpages apprentissage/test différents (on change la graine).
Aussi, je cherche à les optimiser. La forêt est plutôt facile à tuner, les performances précédentes sont sur la meilleure version. Les deux autres méthodes étant plus complexes et avec de nombreux choix de paramétrisation, j'ai effectué des grilles de paramètre. J'ai occupé Mirakulos avec mes scripts en utilisant 100% des cœurs du serveur. Pour xgboost, j'ai pu observer une amélioration avec une paramétrisation bien précise. Pour le Support Vector Machine il semblerai qu'une des meilleures versions soit celle par défaut. Sous certaines paramétrisation, les résultats semblent équivalents, mais j'ai pas réussi à dégager quelque chose de meilleur.

	Xgboost	SVMr	RandomForest
Min.	0	0	0
1st Qu.	472	441	487
Median	1017	952	1052
Mean	1366	1321	1399
3rd Qu.	1848	1736	1902
Max.	24300	25250	24240
R2	0.81	0.8	0.74

Pas facile de les discriminer bien proprement. On peux conjecturer que le Support Vector Machine Regression est légèrement meilleur. On effectuera donc les prédictions avec le Support Vector Machine Régression.

5.2.3 Conclusion

Les prédictions faites, je les ai données au responsable de l'Open Data qui les a intégrées dans les documents Elastic Search.
Comme les prédictions sont à l'année, il est fort probable que un collègue ait besoin de mon travail dans un an ou plus. Je mets donc mon code sur le *Git* de l'entreprise et rédige un rapport interne expliquant ma démarche et mes conclusions.

6 Filtrage Collaboratif

Un projet de la plate-forme au stade de recherche est l'intégration d'un système de recommandation : si un individu achète un article en magasin ou clique sur un produit dans un mail, le système permettrait de produire une liste de produits relatifs à ses produits d'intérêt. Ezakus a déjà développé un moteur de recommandation il y a quelques années pour Cdiscount.

Le but de mon travail est de dégrossir l'intégration robuste de l'ancien modèle à l'architecture de NP6.

6.1 Objectif

A l'issu de ce projet (que je n'aurai pas le temps de finir lors de mon stage), le client doit pouvoir à travers l'éditeur de mail que NP6 propose, intégrer des recommandations produits. Cette intégration concrète implique que le moteur de recommandation doit être appelable par un cURL et répondre un JSON contenant une liste de produits recommandés par individu. Ceci implique d'avoir une idée précise des cheminements nécessaires, c'est à dire de l'architecture complète avant les premières lignes de code.
De plus, cet appel sera effectué pour personnaliser chacun des mails faisant intervenir le module. Il faut donc que les temps de réponse soit cohérents avec cette problématique.

6.2 Plan d'attaque

Le plan d'attaque était d'inclure ce projet au sein de la mlLib : d'une part ceci permettra de faire bénéficier le nouveau modèle de recommandation de tous les outils déjà codés pour les modèles de classification. Et d'autres parts, en repensant à la finalité du projet, comme la mlLib est déjà requêtable à partir du Prediction Server, on gardera une unique librairie et on évitera des étapes laborieuses d'instanciation du PS. Parmi les majeurs difficultés induites, on se heurte à l'implémentation très orientée classification de la mlLib. La généricité de Java est à double tranchant : idéale dans le cas de

projets multiples comme ici, mais néfaste si trop spécifique dans l'implémentation des méthodes.

Je ne partais pas de zéro puisque Justine (une ancienne de l'équipe) avait déjà mis les mains à la pâte. L'architecture cible de son travail était la suivante : (pour mieux comprendre ce schéma gardez en mémoire le premier paragraphe de la partie Feature Engineering et l'intégration des données de navigation)

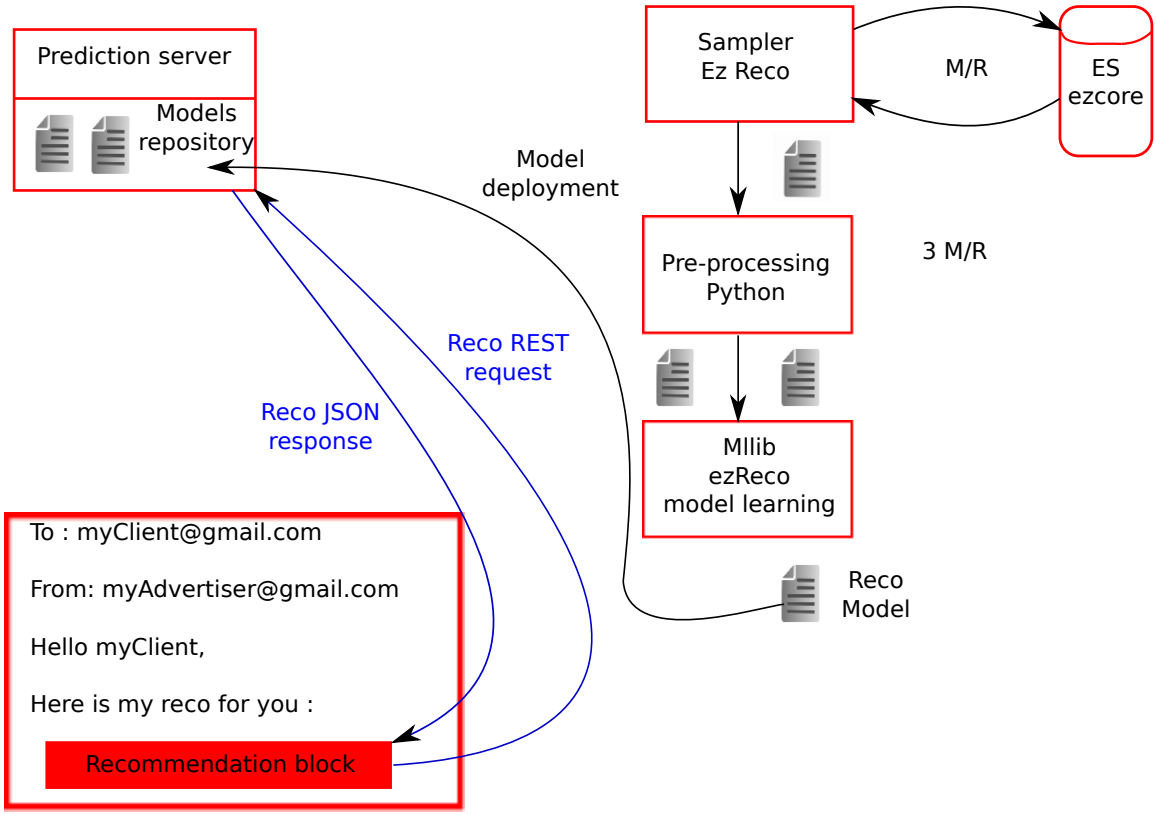


FIGURE 13 – Architecture cible du travail précédent

Elle s'était arrêtée à l'étape *Pre-processing* (les trois Map/Reduce en Python) et une première intégration d'un modèle basique de prédiction. Cette architecture présente quelques failles, ainsi, la demande a un peu changé depuis. Dans un soucis de robustesse et de sérialisation, on souhaite que le

projet tourne à 100% en Java. Et aussi deux autres raisons, premièrement pour faire court, il y aura probablement des filtres sur les produits recommandés à rajouter, il faudra donc changer le format de sortie du sampler. Et deuxièmement, le *pre-processing* implémenté ainsi ne permet pas de découpage apprentissage test.
Mon travail s'articulera selon les étapes suivantes :

étape 1: Modification du format de sortie du sampler.

étape 2: Uniformisation des chargements de format dans la mlLib : afin de lire mon nouveau format et de pouvoir utiliser les méthodes de découpage apprentissages/tests déjà implémentées pour différents formats.

étape 3: Implémentation du *Learner* propre à un modèle de reco, comprend la traduction des trois Map/Reduce Python en Java.

étape 4: Implémentation d'un *Tester*, un moyen de tester la qualité du modèle

étape 5: Édition du rapport en HTML

étape 6: Débogage sur gros jeu de données

6.3 Le modèle de base

Pour répondre à la problématique de recommandation, on souhaite en considérant l'ensemble des produits, définir lesquels peuvent être groupés ensemble car proches au sens d'une distance qu'il nous reste à définit. Ainsi, on pourra à partir d'un passage en caisse (donc une liste de produits) identifier et recommander les produits les plus similaires. Il est donc essentiel de définir ce qu'on appelle la similarité entre les produits.
Mes collègues avaient choisi l'indice de Jaccard. Il s'agit d'une métrique utilisée en statistiques pour comparer la similarité et la diversité entre des échantillons. Il est définit comme le rapport entre le cardinal de l'intersection des ensembles considérés et le cardinal de l'union des ensembles [10].
Le rôle des Map/Reduce est de construire une matrice d'indice, la matrice de similarité. Le premier compte le nombre de commandes par article, et le deuxième le nombre de commandes par paires de produits. En les assemblant, on obtient les composantes de la matrice.
Le modèle de recommandation s'appuie sur l'usage de cette matrice. Il dispose d'une fonction *predict*. Elle est chargée de retourner la prédiction du modèle attribuant une liste de recommandations. Pour une liste de produits

et un nombre de recommandations choisies, elle renvoie l'*ID* des produits associés ayant les indices de similarités les plus élevés.

6.4 Performance du modèle

L'implémentation ne comprenait pas de moyen de tester la performance de l'algorithme. Après un découpage apprentissage/test, j'ai compté le nombre de produits retrouvés par le modèle comme suit. La méthode de test s'apparente à une procédure de *Leave One Out* (LOO) en classification mais sur les articles du ticket. Pour un ticket , je retire un produit et j'effectue une recommandation à partir des autres produits du ticket. Le test est jugé concluant si je retrouve le produit préalablement retiré dans la liste des produits recommandés obtenue via le modèle.

6.5 Implémentation

Je vais pas détailler le développement de ces différentes étapes, et des nombreuses difficultés et nombreux bogues que j'ai eu à surmonter. Après avoir modifié le chargement de la donnée et intégré les différents comptage en Java (anciennement réalisés par les Map/Reduce), j'ai créé des classes abstraites, c'est à dire des classes non instanciées qui permettent d'appeler des méthodes utilisables pour toutes les classes héritant de celles-ci. Parfait pour ma problématique, j'ai pu ainsi me servir des outils de la classification déjà codés, notamment les découpages apprentissage/test, lecture de configuration et édition du rapport Html. Via ces classes abstraites ; j'ai intégré le plus possible mes développement à l'esprit du projet, modifiant le moins possible la structure globale de la mlLib.

6.6 Axes d'amélioration

Le gros défaut de ce modèle est qu'il juge seulement les identifiants des produits. Je pense qu'on verrait un incrément de performance en groupant les produits : par exemple mettre un seul identifiant pour toutes les perceuses et recommander la plus vendue, ou de même pour la faïence. La trop grande diversité d'identifiants noie le bon fonctionnement des prédictions.
De plus les jobs de comptage doivent être revus. Il faudrait intégrer proprement un map-reduce, car le temps de compilation est beaucoup trop long. Sur

quasiment 4 millions de tickets, comptez 18h pour l'ensemble de la construction du modèle à l'édition du rapport Htlm permettant d'évaluer ces performances.

6.7 Conclusion

Pour faire un point sur mon travail réalisé, j'étais enthousiaste à l'idée de contribuer sur un tel projet et je suis parvenu à m'investir à 100% dans ce projet malgré la fin imminente de mon stage. Je suis contente d'avoir réussi à décortiquer la mlLib et d'avoir aboutit l'ensemble des étapes. Bien que l'algorithme derrière mérite d'être optimisé, le modèle tourne !

En commençant ce travail, je n'y voyais rien, je ne savais pas par quel côté commencer parmi les nombreuses classes imbriquées les unes dans les autres. J'ai pu démystifier la programmation orientée objet et ne plus me laisser impressionner par quelques lignes de code. Il n'y a rien qui tient de la magie, si quelque chose ne fonctionne pas, il y a toujours une raison sous-jacente concrète.
Je reviens ici sur la place dans mon stage de la communauté des questions/réponses sur les forums. Être témoin et acteur de ces partages de connaissance a contribué considérablement à ma progression. Dix ans en arrière, mon stage aurait été beaucoup moins multi-tâches. Des expertises précises semblent aujourd'hui ouvertes à n'importe qui ayant une connexion internet.

7 Contribution à la plate-forme : A/B/n testing

NP6 souhaite intégrer à sa plate-forme la possibilité de tester différentes versions de mails. Avant de lancer une campagne mail sur l'ensemble de sa base, un client peut avoir envie de tester différentes versions de mail afin d'élire par exemple quel sujet de mail, ou contenu marketing génère le plus de clics ou d'ouvertures. Il y aura au maximum la possibilité de tester 16 combinaisons de mail.

7.1 Détermination du nombre minimal d'individus par échantillon

Une première question est de savoir, à combien d'individus les différentes versions doivent être envoyées. On cherche ici à garantir une bonne estimation du taux de performance.
Notre population U est finie et composée de N unités d'observations ; chacune de ces unités est une variable indicatrice de la réalisation de l'événement :"l'individu a ouvert le mail" (le raisonnement est complètement isométrique pour "l'individu a cliqué sur le lien du mail"). Ainsi, on s'intéresse à la variable y qui prend la valeur y_k sur l'unité k, y_k à valeur dans $\{0, 1\}$. L'objectif est d'estimer la proportion et donc la moyenne

$$\overline{y} = \frac{1}{N} \sum_{k=1}^{N} y_k = p$$

On a de plus :

$$\sigma_Y^2 = \frac{1}{N} \sum_{k=1}^{N} y_k^2 - \overline{y}^2 = p(1-p)$$

et

$$S_y^2 = \frac{N}{N-1} p(1-p).$$

Ce problème d'estimation de la moyenne est bien connue des plans de sondage [7].L'estimation d'une proportion en est qu'un cas particulier, qui a le bon goût de simplifier considérablement les expressions de variances. Ainsi, dans

le plan aléatoire simple sans remise, on obtient [6] :

$$\begin{aligned}\hat{p} &= \frac{1}{n}\sum_{k=1}^{N} y_k \\ s_y^2 &= \frac{1}{n-1}\sum_{k=1}^{N}(y_k - \hat{p}) \\ &= \frac{n}{n-1}\left(\frac{1}{n}\sum_{k=1}^{N} y_k^2 - \hat{p}^2\right) \\ &= \frac{n}{n-1}\hat{p}(1-\hat{p})\end{aligned}$$

$$\mathbb{V}(\hat{p}) = \frac{N-n}{Nn}\frac{N}{N-1}p(1-p) = \frac{N-n}{(N-1)n}p(1-p),$$

et

$$\hat{\mathbb{V}}(\hat{p}) = \frac{N-n}{Nn}\frac{n}{n-1}\hat{p}(1-\hat{p}) = \frac{N-n}{(N-1)n}\hat{p}(1-\hat{p}).$$

L'objectif est d'atteindre une précision fixée pour le paramètre étudié p qui sera déterminé en fonction des résultats. On voudra donc que p se trouve dans un intervalle de confiance défini autour de $\hat{p}$ avec une probabilité $1-\alpha$:

$$\mathbb{P}\left\{p \in [\hat{p}-\epsilon, \hat{p}+\epsilon]\right\} = 1-\alpha.$$

Or, l'intervalle précédent est celui d'un intervalle de confiance pour l'estimation d'une proportion. Donc il s'écrit aussi [5] :

$$\mathbb{P}\left\{\overline{y} \in \left[\hat{\overline{y}} - z_{1-\alpha/2}\sqrt{\hat{\mathbb{V}}(\hat{p})}, \hat{\overline{y}} + z_{1-\alpha/2}\sqrt{\hat{\mathbb{V}}(\hat{p})}\right]\right\} = 1-\alpha \Leftrightarrow$$

$$\mathbb{P}\left\{\overline{y} \in \left[\hat{\overline{y}} - z_{1-\alpha/2}\ s_y\sqrt{\frac{N-n}{Nn}}, \hat{\overline{y}} + z_{1-\alpha/2}\ s_y\sqrt{\frac{N-n}{Nn}}\right]\right\} = 1-\alpha$$

où $z_{1-\alpha/2}$ est le quantile d'ordre $1-\frac{\alpha}{2}$ d'une variable aléatoire centrée réduite. Ainsi, on a :

$$z_{1-\alpha/2}^2 s_y^2 \frac{N-n}{Nn} \leqslant \epsilon^2$$

$$
\begin{aligned}
\Leftrightarrow \quad & z_{1-\alpha/2}^2 s_y^2 \frac{N-n}{Nn} \leqslant \epsilon^2 \\
\Leftrightarrow \quad & z_{1-\alpha/2}^2 \frac{\cancel{n}}{n-1} \hat{p}(1-\hat{p}) \frac{N-n}{N\cancel{n}} \leqslant \epsilon^2 \\
\Leftrightarrow \quad & z_{1-\alpha/2}^2 \hat{p}(1-\hat{p})(N-n) \leqslant N\epsilon^2(n-1) \\
\Leftrightarrow \quad & z_{1-\alpha/2}^2 \hat{p}(1-\hat{p})N \leqslant N\epsilon^2(n-1) + z_{1-\alpha/2}^2 \hat{p}(1-\hat{p})n \\
\Leftrightarrow \quad & z_{1-\alpha/2}^2 \hat{p}(1-\hat{p})N \leqslant n(N\epsilon^2 + z_{1-\alpha/2}^2 \hat{p}(1-\hat{p})) - N\epsilon^2 \\
\Leftrightarrow \quad & \frac{N(\epsilon^2 + \hat{p}(1-\hat{p})z_{1-\alpha/2}^2)}{\epsilon^2 N + \hat{p}(1-\hat{p})z_{1-\alpha/2}^2} \leqslant n
\end{aligned}
$$

Choix de ϵ

On choisira pour la suite une marge d'erreur ϵ fixée à 10% de la proportion *a priori*. En effet, comme les taux d'ouverture sont autour de 10% et les taux de clics autour de 1%, on ne peux pas définir la même valeur pour les deux. Après discussion avec Laurent et bien que l'usage est de prendre 5%, on choisit un compromis de 10%. On arrive à cette conclusion car il est important d'adapter la théorie à la pratique. Un intervalle de 5% autour de $\hat{p}$ exige des échantillons trop gros pour permettre d'envoyer plusieurs versions. On doit se plier au cahier des charges.

Choix de α

Et on choisira comme usuellement $\alpha = 0.05$.

7.2 Intégration dans la plate-forme

L'intégration dans le produit m'a permis de me confronter aux autres corps de métier du développement. Il n'est pas facile de sensibiliser des collègues peu réceptifs aux arguments de rigueurs sur les tests. La priorité étant de satisfaire le client, la première version de l'A/B test est partie avec la possibilité d'envoyer les mails qu'à très peu de cibles par versions, avec cependant avec un message d'alarme lorsque celui ci est trop bas.
J'ai également tenté d'alerter l'équipe de développement sur la nécessité de dé-doublonner les individus dans le calcul des taux de performances, sans que

cela n'est abouti pour l'instant. Ceci revient à violer la première hypothèse du calcul : chaque individu suit une loi de Bernoulli.
Pour finir, c'était sympa de parler du travail réalisé avec quelqu'un qui n'a pas fait de math depuis le bac. J'ai apprécié l'importance de se ramener à des exemples concrets. J'ai eu du mal à être claire dans mes propos et fini par rédiger la partie de la présentation qui traite du sujet :

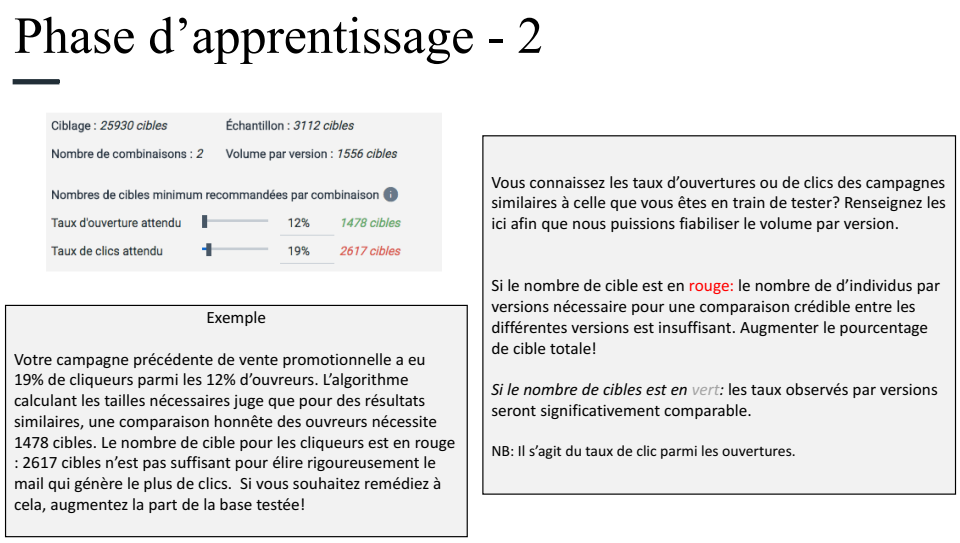


FIGURE 14 – Diapo de présentation de la plate-forme sur la sélection du nombre de cibles

7.3 Algorithme de sélection

De même la première version de cette fonctionnalité ne comprends pas de tirage aléatoire. Il faut comprendre que le produit était urgemment nécessaire à la plate-forme, et le temps a manqué pour tout faire proprement. Mais c'est dans leurs priorités pour la version 2.
Les individus sont actuellement triés par ordre alphabétique. Le conseil de sélection est le suivant. On attribue à chacun d'entre eux une réalisation u_k d'une loi uniforme sur $[0, 1]$, puis les trier par ordre croissant de u_k. Pour i allant de 1 au nombre de combinaisons de l'AB testing désirées, placer dans

l'échantillon i, les individus dont les positions vont de $(i-1) \times n$ à $i \times n$. La requête s'effectuera en interrogeant la base en SQL.

7.4 Le test

Ce paragraphe présente une réflexion à la question du choix entre deux versions de mails.

Je me suis intéressée à deux façons différentes de répondre à cette problématique. Les combinaisons générés par les tests sur différents facteurs m'orientent vers un modèle linéaire généralisé. Il est plus fiable et robuste que des tests de comparaisons multiples. Et j'ai ensuite réfléchi à une seconde solution de comparaisons multiples car beaucoup plus simple à intégrer dans la plate-forme, et, de plus, l'utilisateur aura à terme la possibilité d'enlever des versions qu'il ne souhaite pas tester. Il est peu probable que l'élection de la meilleure version se fasse par un de mes tests, il s'agira plutôt d'un indicateur de confiance sur le choix du meilleur mail.

Régression logistique

Les modèles linéaires généralisés ont été formulés par John Nelder et Robert Wedderburn comme un moyen d'unifier les autres modèles statistiques notamment la régression logistique. Ils proposent une méthode itérative dénommée méthode des moindres carrés repondérés itérativement pour l'estimation du maximum de vraisemblance des paramètres du modèle. L'estimation du maximum de vraisemblance reste populaire et est la méthode par défaut dans de nombreux logiciels de calculs statistiques [9]. Comme les estimateurs du max de vraisemblance sont asymptotiquement gaussiens, une fois l'estimation faite, on connaît leurs lois et l'on peux faire des tests. L'idée de la généralisation est d'utiliser une transformation mathématiques sur la variable à expliquer y appelée fonction de lien, du coup le modèle à ajuster devient :

$$f(E(y)) = a_1x_1 + a_2x_2 + ... + b + \epsilon$$

Où a_i et b peuvent être estimés, x_i variables explicatives, $E(y)$ la moyenne de la variable à expliquer, ϵ un terme d'erreur et f la fonction de lien.

On est dans un cas particulier du modèle linéaire généralisé, ici la variable à analyser suit une loi binomiale. En remarquant que l'événement X_i :

"l'individu i a cliqué sur le mail" suit une loi de Bernoulli :

$$X_i = \begin{cases} 1 & \text{si l'individu i a cliqué sur le mail,} \\ 0 & \text{sinon.} \end{cases}$$

et comme on répète l'observation, en sommant on obtient bien un loi binomiale. Alors la fonction de lien utilisée est la fonction logit : $f(y) = log(\frac{y}{1-y})$ et il s'agit simplement d'une régression logistique. Le test préconisé pour vérifier la significativité des effets est alors un test du Chi 2 (χ^2).
Le point noir de cette méthode est son intégration dans la plate-forme.

Test de proportion sur deux échantillons

Ce test est construit de la façon suivante. Les échantillons testés seront issus d'un tirage aléatoire et leurs effectifs seront supérieurs à 30. On peut ainsi supposer que les probabilités de succès suivent une loi normale dont l'espérance serait la proportion p et dont l'écart-type serait $\sigma = \sqrt{\frac{p(1-p)}{n}}$
Ainsi, si la proportion p_1 est égale que celle d'une autre population p_2 alors la différence des deux suit une loi normale centrée. En divisant par l'écart type, on obtient une loi normale centrée réduite :

$$\frac{\mid p_1 - p_2 \mid}{\sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}} \sim \mathcal{N}(0,1)$$

Tester si l'une est plus grande que l'autre revient alors à vérifier si $\frac{|p_1-p_2|}{\sqrt{\frac{p_1(1-p_1)}{n_1}+\frac{p_2(1-p_2)}{n_2}}}$ s'éloigne d'une distribution normale. De cette façon, on a construit le test :

$$\mathbf{H_0} : p_1 = p_2 \qquad \mathbf{H_1} : p_1 < p_2 \text{ ou } p_1 > p_2$$

On souhaite un niveau de confiance de 95% dans la supériorité d'une des proportions. Au lieu de faire deux fois le test pour savoir en cas de rejet de H0, laquelle est supérieure, on définit la région de rejet bilatéral au niveau 90%, puis on regarde le signe de $p_1 - p_2$:

$$W = \left\{ \frac{\mid p_1 - p_2 \mid}{\sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}} > 1.65 \right\}$$

Je me demandais s'il valait mieux tous les comparer deux à deux, ou chacun à l'ensemble. Dans les deux cas, on doit penser à contrôler l'erreur de l'ensemble en appliquant la correction de Bonferroni ce qui va rendre le test très conservateur. Mais l'intégration dans la plate-forme est plus simple que la méthode précédente.

Conclusions

Au travers de ses missions variées, ce stage a répondu aux attentes que j'avais en le choisissant. Il m'a permis de découvrir l'utilisation d'algorithmes de classifications en chaîne de production : de la collecte de la donnée au *Scoring* des individus.
J'ai eu la chance de participer concrètement aux enjeux de l'équipe en intervenant sur des tâches qui m'ont passionnées. J'ai beaucoup aimé contribuer au workflow de machine learning à travers le moteur de recommandations et le modèle de désabonnement. J'ai eu la sensation d'apporter ma pierre à l'édifice en enrichissant les projets. Ceci m'a donnée confiance en mes compétences et mes capacités de réflexions. J'ai aussi eu un coup de cœur pour la programmation en Java.

Je suis très satisfaite de ma formation. Mon master m'a donné les bonnes briques pour pouvoir intervenir et réfléchir sur les différents aspects théoriques qui m'ont été confiés. Et ma licence de mathématiques m'a apportée rigueur et méthodologie nécessaires à l'appréhension de nouveaux outils et langages de programmation.

Forte de cette expérience, j'ai désormais une idée assez précise sur mon orientation professionnelle. J'aimerais beaucoup par la suite intégrer une entreprise d'acteurs de petites tailles, construire des modèles d'apprentissage automatique et développer en Java.

Références

[1] Frank WILCOXON : *Individual Comparisons by Ranking Methods*, Biometrics Bulletin, Vol. 1, No. 6, pp. 80-83 Dec., 1945.

[2] Philipe BESSE : *Tests statistiques élémentaires*, *https : //www.math.univ−toulouse.fr/besse/Wikistat/pdf/st−l−inf−tests.pdf*.

[3] Delphine FERAL : *Compléments sur la théorie des Tests*, Cours Master MIMSE, Chapitre 2 : Outils de la Statistique non paramétrique et Tests, 2014-2015.

[4] Wikipédia *Map/Reduce* *https : //fr.wikipedia.org/wiki/MapReduce*

[5] Delphine FERAL : *Théorie de l'estimation et des tests statistiques*, Cours Master MIMSE, Chapitre 2 : Estimation paramétrique par Intervalle de Confiance, 2014-2015.

[6] Pascal ARDILLY et Yves TILLÉ : *Exercices corrigés de méthodes de sondage*, Technip, Paris 2002.

[7] Yves TILLÉ : *Sampling algorithms*, Springer, New-York 2006.

[8] Open Classroom *Les tests unitaires* *https : //openclassrooms.com/courses/les−tests−unitaires−en−java*

[9] Wikipédia *Generalized linear model* *https : //en.wikipedia.org/wiki/Generalized_linear_model*

[10] Wikipédia *Indice et distance de Jaccard* *https : //fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard*

Glossaire

Churn Diminution de la quantité d'individus dans une base.

Churner Individu s'étant désabonné.

Cluster Ou grappe de serveurs. Procédé consistant à regrouper plusieurs ordinateurs indépendants appelés nœuds (node en anglais), afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur.

cURL Interface en ligne de commande, destinée à récupérer le contenu d'une ressource accessible par un réseau informatique. Abréviation pour "bibliothèque de requêtes aux URL pour les clients" (ou "voir URL").

Elastic Search Serveur utilisant Lucene pour l'indexation et la recherche des données. Il fournit un moteur de recherche distribué et multi-entité à travers une interface REST (API joignable par appels cURL). C'est un logiciel libre écrit en Java et publié en open source sous licence Apache.

Feature Variable explicative ou caractéristique d'un individu.

Feature Engineering Création des variables explicatives à partir des données brutes qui vont servir de base pour les modèles d'apprentissage.

Feature Processing Traitement compilant le feature engineering.

Git Logiciel de gestion de versions utilisé pour gérer des codes sources, et capables de suivre l'évolution d'un fichier ligne de code par ligne de code.

Hit Le hit de navigation est l'entité de base collectée par NP6 via un cookie déposé préalablement sur une page internet d'un client partenaire. Ce hit contient toutes les informations récupérables issues de la navigation (heure de connexion, moteur de recherché utilisé, type d'appareil ayant servi à la connexion, URL visitée, ...).

Html Format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte (abrégé de HyperText Markup Language).

Mirakulos Nom d'un serveur de NP6.

mlLib Ensemble de la bibliothèque de création des modèles à partir d'un fichier de configuration permettant d'exprimer un scénario d'apprentissage.

Plate-forme Ou système d'exploitation, environnement permettant la gestion et/ou l'utilisation de services applicatifs.

Pre-processing Pré-traitement de la donnée visant à modeler les données brutes.

Scoring Technique qui permet d'affecter un score à un client. Le score obtenu traduit généralement la probabilité qu'un individu réponde à une sollicitation marketing ou appartienne à la cible recherchée. Il mesure donc l'appétence pour l'offre potentielle.

Workflow Processus d'automatisation des tâches permettant un enchaînement automatisé des différentes opérations et étapes de validation d'une tâche plus ou moins complexe (procédure de commande, suivi de projet, campagne email..).