Kaggle Coupon Purchase Prediction

Projet de Fin d'Etudes
Année 2015-2016

Jacques Peeters
Génie Informatique et Statistique
Polytech Lille

Tuteur : Cristian Preda – Professeur de statistiques à Polytech Lille

Table des matières

1. Introduction

J'ai choisi comme Projet de Fin d'Etudes de participer à la compétition « Coupon Purchase Prediction » sur Kaggle. La compétition était proposée par Ponpare, le leader Japonais de vente de coupons de réduction, avec un choix très varié allant de séances de Yoga aux restaurants de sushi. Groupon est son équivalent en France. Investir dans une nouvelle expérience n'est pas gratuite. La peur de gaspiller de l'argent et du temps peut entraver un achat. En utilisant l'historique d'achats et de visites, la compétition vise à prédire quels coupons un client va acheter sur la période de temps suivante et ainsi les aider à trouver les produits qui leur correspondent.

2. Description des données

Les données disponibles sont les suivantes :

- L'historique, « weblog.csv », de 2.9M visites et achats entre le 01/07/2011 et le 23/06/2012 avec user_id, coupon_id, date, purchase_flag,
- 22.9k utilisateurs, « userlist.csv » avec les variables user_id, age, sexe,
- 19.4k coupons, « couponlist.csv », d'entraînement, couponTrain, et 310 de test, couponTest, avec leur description de 25 variables (genre, location géographique, prix, dates de disponibilités, ...),

Il faut donc prédire quels couponTest les utilisateurs vont acheter sur la période du 24/06/2012 au 30/06/2012. Les couponTrain ne peuvent être achetés durant cette période. Les couponTest ne peuvent être achetés au cours de la période d'entraînement, mais certains peuvent avoir été visités.

Du fait du temps limité que je bénéficie pour la réalisation de mon projet et la rédaction du mémoire je me suis permis de récupérer le jeu de données traduit en anglais depuis le japonais pour les locations géographique et le genre des coupons (hôtel, restaurant, etc). Les ID des coupons et utilisateurs étaient des chaînes de 32 caractères. Ces chaînes ont été converties en entier. Je possède donc également les fichiers « coupon_id_hash.csv » et « user_id_hash.csv » qui me permettent de repasser aux ID conformes à la compétition et à la lecture de mes fichiers résultats par Kaggle. Je suis néanmoins conscient que le travail de nettoyage des données représente un point clé du travail d'analyse.

3. Choix de la solution proposée

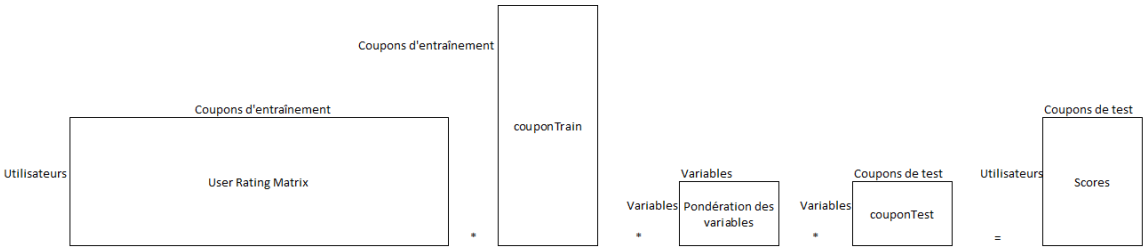
Ne possédant pas d'historiques d'achats sur les couponTest, j'ai choisi de calculer leur similarité, ressemblance, avec les couponTrain qui eux constituent mon historique d'achats et de visites. J'ai choisi comme calcul de similarité une modification du « cosine similarity ». J'ai validé ce choix puisque cette technique a été utilisée par d'autres compétiteurs. J'ai jeté un œil aux solutions d'autres compétiteurs afin d'identifier des points à améliorer sur mon propre script. Mais je souhaite insister sur le point que j'ai bien réalisé moi-même mon propre programme et ai apporté plusieurs améliorations. Améliorations que je détaillerai plus tard dans ce document.

3.1. Solution détaillée

L'idée du programme est la suivante:

- attribuer une note pour chaque paire (utilisateur, coupon d'entraînement), 1 s'il y a eu achat, 0 sinon. On obtient une matrice « User Rating Matrix » de taille 19.4k*22.9k.
- calculer les préférences pour chaque (utilisateur, variables de coupon) par User Rating Matrix * couponTrain. Nous avons une matrice de taille 19.4k * Variables avec Variables le nombre de variables pour décrire un coupon.
- calculer la similarité cosinus modifiée entre les (utilisateur, coupon de test) en donnant un "poids" à chaque variable.
- prédire les 10 coupons avec la plus forte similarité/score pour chaque utilisateur.

Voici un résumé des calculs matriciels effectués :



3.2. Précision sur la similarité cosinus modifiée

Ci-dessous la définition issue de Wikipédia : *« La similarité cosinus (ou mesure cosinus) permet de calculer la similarité entre deux vecteurs à n dimensions en déterminant le cosinus de l'angle entre eux.*

Soit deux vecteurs A et B, l'angle $\cos\theta$ s'obtient par le produit scalaire et la norme des vecteurs :

$$\cos\theta = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Comme la valeur $\cos\theta$ est comprise dans l'intervalle [-1, 1], la valeur -1 indiquera des vecteurs résolument opposés, 0 des vecteurs indépendants (orthogonaux) et 1 des vecteurs similaires (colinéaires de coefficient positif). Les valeurs intermédiaires permettent d'évaluer le degré de similarité.»

Deux ajustements majeurs sont à réaliser. Tout d'abord les variables n'ayant pas toutes la même importance, il est évident qu'elles doivent être pondérées pour un matrice de poids notée W. Par exemple, il est très probable que la région géographique ait une importance plus forte que la réduction accordée. Pour pouvoir profiter d'un coupon, même avantageux, il faut nécessairement qu'il puisse être utilisé à proximité de son foyer. Deuxièmement, l'objectif est de recommander les 10 meilleurs coupons et non de connaître exactement la similarité, par conséquent on peut supprimer le dénominateur et conserver la relation d'ordre des coupons « préférés ».

La formule de la similarité cosinus modifiée est donc simplement la suivante :

$$modifiedCos = A * W * B$$

3.3. Feature engineering

La définition du feature engineering par Wikipédia est la suivante : *« Le feature engineering est le procédé pour lequel de nouvelles variables sont créées à partir des données existantes pour permettre l'application des algorithmes de machine learning. »*

Les variables ils sont initialement qualitatives ou quantitatives. Or la similarité cosinus ne peut s'appliquer que sur des variables quantitatives puisqu'elle repose sur des multiplications de matrices. Pour cela les variables qualitatives sont transformées en variables booléennes, c'est-à-dire que chaque modalité devient un nouvelle variable avec 0 ou 1 comme valeur. Une variable qualitative à N modalités est ainsi redécoupée en N variables booléennes. Ainsi les manipulations matricielles sont possibles.

Une transformation log10 est appliquée aux variables prix initial et prix après réduction. Pour justifier cette transformation, nous supposons que la différence perçue par les clients entre un coupon à 20€ et un second à 40€ est supérieure à celle entre un coupon à 1000€ et un second à 1020€. La perception des prix n'est pas linéaire.

Des variables comme la date de début de disponibilité et la fin ne sont pas très intéressantes. Ce qui est intéressant est la durée de disponibilité, il faut donc les remplacer. Il en va de même pour la période de validité des coupons. Ex : disp = disp_end – disp_from.

Il y a aussi des variables très redondantes. Par exemple il y a deux variables qualitatives pour décrire le genre des coupons, deux autres pour la préfecture/ville. Dans chaque cas, uniquement une des deux variables n'est conservée.

Les neuf variables conservées sont les suivantes :

- genre_name, le genre (hôtel, restaurant, massage, …),
- price_rate, le pourcentage de réduction,
- catalog_price, le prix avant réduction,
- discount_price, le prix après réduction,
- disp_period, la période de disponibilité
- valid_period, la période de validité
- usable_date_DAY, les jours disponibles (lundi, mardi, …)
- large_area_name, la zone géographique large,
- small_area_name, la zone géographique réduite.

4. Evaluation locale des paramètres

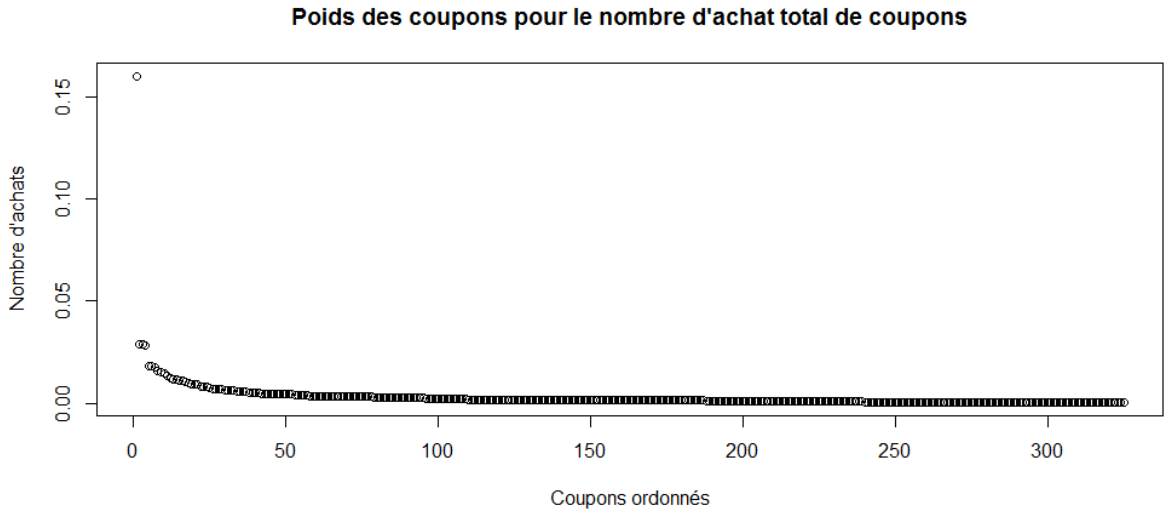
Avant de soumettre notre fichiers de recommandations à Kaggle, pour qu'il évalue la précision de nos recommandations sur son fichier de test privé, il est important d'ajuster les variables de notre modèle. Le nombre de soumissions lors des compétitions en cours est limité à deux par jours. Cela permet d'éviter le « reverse engineering ». Kaggle évalue la performance sur son propre fichier de test privé, l'overfitting est donc forcément à éviter.

4.1. Description

- Placer les 7 derniers jours de weblog dans weblogTest, le reste dans weblogTrain,
- Filtrer les coupons weblogTest ayant été vus et/ou achetés pendant weblogTrain,
- Constituer les couponTrain et couponTest,
- Identifier les achats des utilisateurs pendant weblogTest,
- Appliquer l'algorithme sur les données d'entraînements,
- Mesurer la performance MAP@10 en fonction de la pondération des variables.

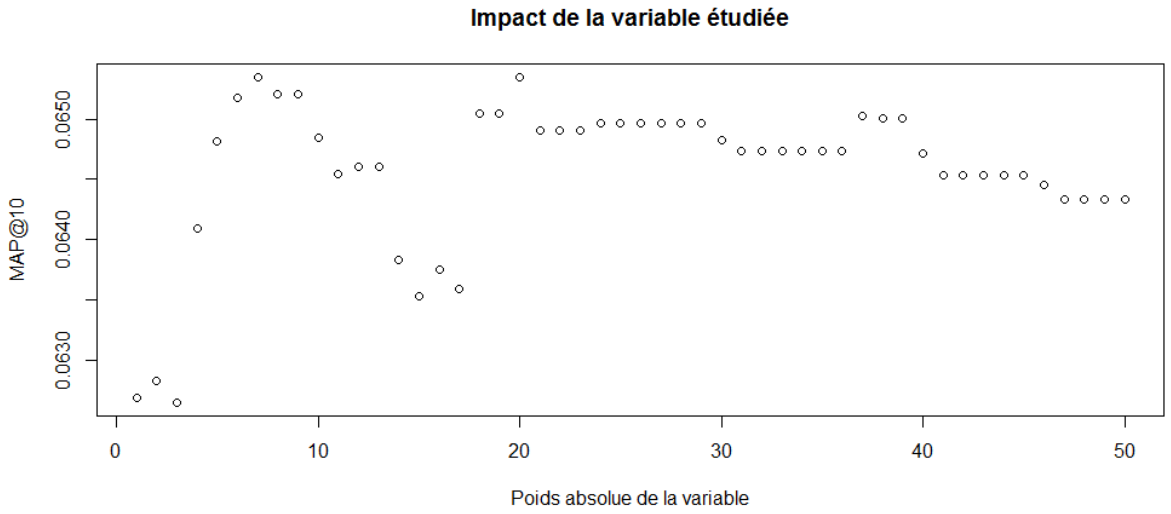
4.2. Remarques

Après optimisations des variables puis upload sur Kaggle j'ai remarqué que mes premiers essais n'étaient pas fructueux. J'ai procédé à une analyse des coupons achetés durant la partie test.



On aperçoit des coupons « extrêmes ». Ce sont des coupons gratuits ou bénéficiants d'offres exceptionelles. Ces coupons favorisent l'overfitting et je soupconne que ce type de coupons aient été préablement éliminés du test privé Kaggle. J'ai ainsi décidé de supprimmé les 4 premiers coupons qui représentaient 24% des achats de mon échantillon test. Mon échantillon test passait donc de 2314 achats à 1746.

Malgré ces précautions les résultats locaux ont conservés des irrégularités.



L'exemple ci-dessus montre la variation de la performance en fonction du poids absolue de la variable géographique. Le maximum est en 7, mais la variation avec les points aux alentours est très grande. C'est probablement une irrégularité du test local qui n'est pas présente dans l'échantillon test de Kaggle. Par conséquent, il est risqué de choisir 7 comme poids pour cette variable (risque d'overfitting). Il est plus sage de préfer une valeur comprise entre 20 et 30 et ainsi garantir une plus grande stabilité. Pour les raisons détaillées précédement il est compliqué de paramétrer automatiquement les variables. Ce sont donc des choix personels suite à l'analyse des résultats locaux pour les neuf variables conservées précédement décrites.

Pour obtenir des résultats plus régulier il aurait probablement fallu :

- Eliminer les coupons « extrêmes » de l'échantillon d'apprentissage,
- Réaliser différent échantillons d'apprentissage et de test (Bootstrap Aggregating) pour des semaines de test différentes.

5. Evaluation de la précision

5.1. Présentation

La précision des prédictions est évaluée à l'aide de la méthode Mean Average Precision @ 10 (MAP@10). Voici la formule proposée par Kaggle :

$$MAP@10 = \frac{1}{|U|}\sum_{u=1}^{|U|}\frac{1}{min(m,10)}\sum_{k=1}^{min(n,10)}P(k)$$

où |U| le nombre d'utilisateurs, P(k) la précision pour les k premiers coupons, n le nombre de coupons correctement prédits, et m le nombre de coupons achetés pour l'utilisateur donné sur la période testée. Si m=0 l'évaluation pour l'utilisateur est considérée nulle.

5.2. Remarque

L'ordre des prédictions est très important pour les utilisateurs, ils regardent en priorité les premières suggestions. Par conséquent, il est logique de recommander les coupons cibles en

premiers et non dans les dernières positions de nos 10 recommandations. Cette métrique en tiens compte.

5.3. Exemples

Voici quelques exemples pour aider à la compréhension.

u	m	n	Rang des coupons bien prédits	Average Precision
1	2	2	1 et 2	(1/1 + 2/2) / 3 = 0.67
2	2	2	1 et 10	(1/1 + 2/10) / 3 = 0.40
3	0	0	.	0

Nous avons donc le résultat suivant pour ces trois utilisateurs :

Mean Average Precision	(0.67 + 0.40 + 0) / 3 = 0.35

6. Améliorations effectuées

Comme précisé précédemment, j'ai observé d'autres solutions proposées, notamment celles qui s'appuyaient sur la technique de similarités cosinus. Mon script, entièrement réalisé par mes soins, se différencie en de nombreux points. Voici les améliorations majeures de mon script:

- la matrice de note (User Rating Matrix) qui me permet de facilement ajuster les notes si je souhaite inclure le nombre de visites, la quantité d'achat, ...
- mes recherches sur les manipulations de données à l'aide du package dplyr pour un gain de lisibilité et de performance,
- mes recherches sur la lecture et manipulations de données avec data.table pour améliorer la rapidité d'exécution,
- utilisation du package Matrix pour permettre la gestion des matrices sparses afin d'éviter les problèmes de taille mémoire,
- package dummies pour faciliter la création des variables binaires et la lisibilité.

Ci-dessous un petit BenchMark entre un script exemple proposé sur Kaggle et celui conçu par mes soins :

	Script exemple	Script personnel	Gain relatif
Lecture des données	02:14	00:06	22
Algorithme	07:21	00:27	16
Total	09:35	00:33	17

A précision constante, le gain de rapidité est très confortable. Cette amélioration d'exécution permet d'éviter de perdre du temps pour obtenir les résultats locaux et ainsi ajuster son modèle au plus vite.

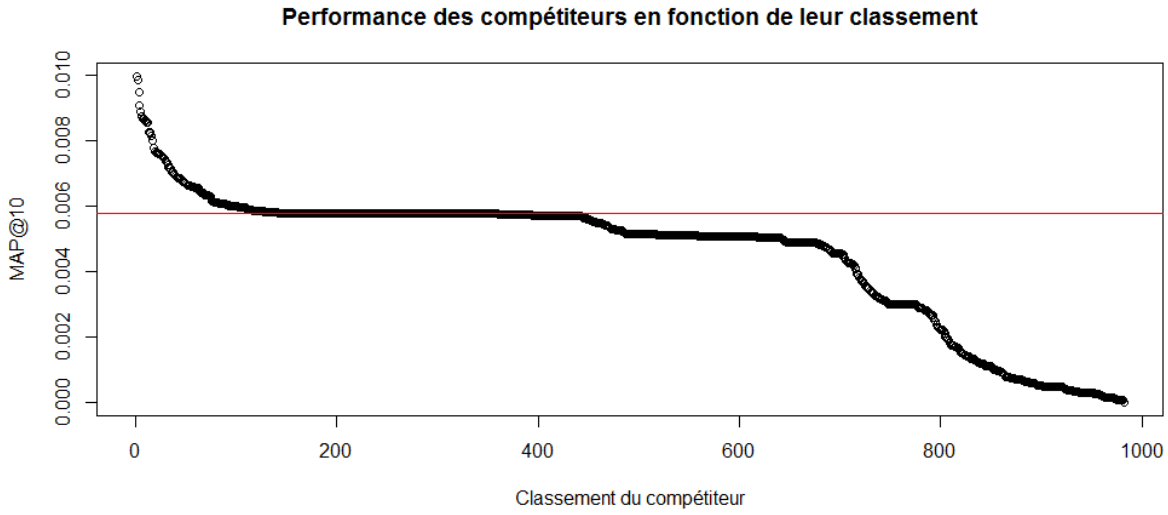
7. Résultat

Après plusieurs ajustements de mes variables en local, ainsi qu'une séparation entre les utilisateurs homme et femme j'ai obtenu un score de 0.005779 sur le classement privé, ce qui correspond à un classement de 150ème sur 981 participants.

152	↑217	udit saini 2	0.005777
153	↑246	Amine Benhalloum	0.005773
-		**Asterix**	**0.005779**
Post-Deadline Entry			
152	↑217	udit saini 2	0.005777
153	↑246	Amine Benhalloum	0.005773

C'est une performance honorable mais je suis juste au-dessus d'un palier de compétiteurs. Les meilleurs compétiteurs arrivent à obtenir des scores bien meilleurs. Cela s'explique par deux raisons principales :

- beaucoup de compétiteurs se sont ré-apropriés la technique de la similarité cosinus,
- cette technique ne peut obtenir de résultats bien meilleurs.



8. Difficultés rencontrées

Durant ce projet, j'ai fait face à un bug majeur dans mon script qui m'était invisible. Le script fonctionnait de façon cohérente mais les résultats MAP@10 étaient très mauvais. J'ai perdu un temps fou à comprendre pourquoi. Je pensais que le problème venait de la pondération de mes variables et ai donc essayé de les ajuster. J'ai identifié la source du problème lorsque j'ai testé de recommander des coupons au hasard, de façon random. Je me suis rendu compte que j'obtenais des résultats similaires. Un bug était donc présent dans mon code. Un bug très difficile à identifier que voici :

```
107  # Les fonctions
108  # Création de la matrice sparse en fonction des notes des utilisateurs
109  fURM = function(URM){
110    sURM = sparseMatrix(URM$user_id, URM$coupon_id, x = URM$rating)
111    col = unique(weblogF$coupon_id) #On ne garde que les colonnes intéressantes
112    col=sort(col) #Il faut ordonner les colonnes à conserver
113    sURM=sURM[,col] # sinon la commande suivante marche mal, je l'ai appris douloureusement
114    return(sURM)
115  }
```

Pour permettre la multiplication matricielle entre mon User Rating Matrix et couponTrain, je supprimais les colonnes vides de mon User Rating Matrix. Or pour cela, il faut que les colonnes soient ordonnées. C'est pour cela que j'ai rajouté la ligne 112. Sans cette ligne, les mauvaises colonnes étaient supprimées, malheureusement aucun message d'erreur n'apparaissait. Je pense que ce problème était difficilement identifiable, mais cela m'a permis d'apprendre.

Excepté ceci, j'estime avoir réussi à progresser de façon linéaire dans mon travail.

9. Conclusion

Ce Projet de Fin d'Etudes m'a été extrêmement bénéfique. Tout d'abord cela m'a permis d'énormément progresser sur ma pratique de R, un logiciel que j'affectionne beaucoup. J'ai d'ailleurs pu valoriser mon travail lors de mes entretiens de stage. J'ai également publié le code sur mon GitHub personnel afin qu'il constitue une vitrine pour mon CV.

Oui il y a encore beaucoup de points à améliorer, mais ce projet constitue une étape dans ma progression. J'ai découvert Kaggle, qui offre certes la possibilité de prendre part à des compétitions intéressantes mais qui est surtout riche en information. Une mine d'informations importante à suivre pour découvrir ce qui se fait de mieux en machine learning. J'ai ainsi découvert la technique des « Boosted Trees » et compte donc m'essayer prochainement à son implémentation dans un de mes prochains script R grâce au package xgboost.

Par ailleurs, en parallèle de ce projet, j'avais mon semestre d'études à Polytechnique Milan et ma recherche de stage. Un semestre qui s'est bien déroulé mais durant lequel, il m'a fallu fournir une charge de travail plus conséquente que celle à Lille. Je pense également avoir largement dépassé la charge de travail initialement allouée à mon PFE.

Mon code, commenté avec soin, ainsi que les jeux de données nécessaires sont disponibles dans le fichier « PFE_JacquesPeeters.rar ».