Un système de recommandation pour le panier moyen

Angéllica C. Araujo[1]

[1]Rio de Janeiro - RJ, Rio de Janeiro Brésil

angellica.c.a@gmail.com

1. Dénition

1.1. Vue d'ensemble du projet

L'identification de corrélations dans les données d'achat séquentielles est appelée analyse du panier moyen. Son usage le plus commun est de faciliter la prévision des prochains articles qui peuvent intéresser un utilisateur. Ainsi, l'objectif final de ce type d'analyse, portant sur les habitudes d'achat et de consommation, est d'obtenir des informations sur le comportement d'achat d'un utilisateur [Kaur and Kang 2016].

Les algorithmes de recommandation sont également utilisés pour personnaliser l'achat et la vente en ligne. Par exemple, Amazon utilise des algorithmes qui adaptent le contenu d'une page internet aux préférences connues de l'utilisateur [Amazon 2003]. Ces outils informatiques jouent un rôle important dans l'industrie, puisqu'ils permettent d'adapter l'offre à la demande et aux préférences individuelles des consommateurs.

Bien qu'ils soient très étudiés, ces outils doivent suivre les transformations propre à leur modèle de calcul. Des études récentes, comme celles qui ont été commanditées par Netflix, ont montré l'intérêt d'explorer l'association entre différentes variables dans les résultats de recommandations [Basilico and Raimond 2017]. Cet article se base sur les avancées régulières en Machine Learning, et leurs applications à plusieurs filières industrielles, dans le but d'améliorer les résultats d'un outil de recommandation spécifiquement adapté aux secteurs de la vente au détail et de l'agroalimentaire. Le défi Kaggle [Kaggle 2017] a également inspiré ce projet, et a constitué une source de données et de métriques à des fins de comparaison.

1.2. Problème

Soit $C = \{c^1, c^2, c^3, ..., c^n\}$ un ensemble de clients, $I = \{i^1, i^2, i^3, ..., i^n\}$ un ensemble d'articles et $E_c = \{E_c^1, E_c^2, E_c^3, ..., E_c^n\}$ l'ensemble des évènements clients. Un évènement client se définit comme $E_c^t, b \subset I, \forall c, t$ où pour chaque ensemble b d'articles achetés a lieu une transaction qui comprend un article ou plus, pour chaque utilisateur dans un temps t. L'algorithme proposé doit prendre en compte l'historique de consommation d'un utilisateur E_c^t et prévoir quels articles cet utilisateur va le plus probablement vouloir acheter, en plus d'estimer la probabilité d'un tel achat.Ainsi, ce problème peut être considéré comme une tâche de modélisation de classification automatique.

Le présent travail se fixe les objectifs suivants:

1. Pré-traiter l'ensemble des commandes et des produits pour obtenir assez d'informations sur les paniers b par client
2. Entraîner les modèles de classification en prenant en compte les caractéristiques liées aux évènements de consommation E_c du client c

3. La tâche de classification peut prendre en considératon plusieurs valeurs différentes comme caractéristiques temporelles comme données d'entrée
4. Pour chaque modèle, produire comme sortie un vecteur de probabilité de consommation par article
5. Analyser les résultats obtenus pour chaque modèle avec des métriques d'évaluation

La solution proposée vise à montrer que d'autres caractéristiques que, dans le cas étudié, l'article et les données de l'utilisateur présentent un intérêt et peuvent avoir un impact positif. De cette manière, la précision des résultats produits par ce type d'outils informatiques peut être améliorée. Ainsi, le système de recommandation que nous proposons inclura un modèle de classification afin d'effectuer les tâches d'entraînement et de prédiction.

1.3. Métriques

Nous utiliserons le F1 Score tel que proposé par le défi Kaggle [Kaggle 2017]. Les fractions de rappel (recall) et de précision (precision) sont prises en comptes dans le calcul, ce qui nous fournit une métrique dont le résultat prend en compte les deux fractions [Prentice-hall]. La formule générale pour le F1 Score est la suivante:

$$F_\beta = (1 + \beta^2) \left(\frac{precision.recall}{(\beta^2.precision)+recall} \right)$$

La formule appliquée dans le cas présent est une adaptation de cette formule. Elle est appelée F-score, ou encore F-mesure. Soit $\beta = 1$, we have the following formula as basis for the calculation of score and validation of the results of the work:

$$F_1 = (2) \left(\frac{precision.recall}{precision+recall} \right)$$

Le rappel (recall) est une mesure la proportionnalité entre les résultats positifs et les résultats identifiés comme positifs. De cette manière, le rappel (recall) nous permet d'évaluer à quel point le résultat final est complet. La précision (precision) nous permet d'analyser à quel point l'exactitude du résultat final, puisqu'elle montre le rapport entre les résultats vrais positifs et les résultats identifiés comme positifs (incluant les faux positifs) [Prentice-hall].

2. Analyse

2.1. Exploration des données

Le dataset Instacart est un ensemble de données ouvert au public qui inclut toutes les commandes en ligne de l'Instacart Online Grocery Shopping. Il est disponible depuis 2017. L'ensemble de données contient plus de 3 millions de commandes faites par plus de 200 000 utilisateurs d'Instacart. Chaque utilisateur a réalisé entre 4 et 100 commandes, et chaque commande contient une séquence des articles achetés. De plus, le jour de la semaine et l'heure de la journée où la commande a été passée sont connus, ainsi que le temps écoulé entre les commandes.

Le jeu de données est un ensemble de fichiers .csv. Dans le cadre de ce travail, nous nous focalisons sur les données fournies par les fichiers produits commandés (*order products*) et les fichiers commandes (*orders*). Les fichiers de commande de produits indiquent quels produits ont été achetés et dans quel ordre, et l'attribut commandé à nouveau (reorder) indique que le client a déjà effectué une commande incluant ce produit.

Nous utilisons les fichiers de commande pour sélectionner l'ensemble d'entraînement et de test, en plus de constituer nos paniers [Kaggle 2017]. Ainsi, notre travail reçoit comme entrée de commandes les données suivantes [Instacart 2017]:

orders (3.4m rows, 206k users):

- order_id: identification de la commande
- user_id: identification du client
- eval_set: ensemble d'évaluation auquel cette commande appartient (antérieur, entraînement ou test)
- order_number: numéro de séquence de la commande pour un utilisateur donné (1 = premier, n = énième)
- order_dow: jour de la semaine où la commande a été effectuée
- order_hour_of_day: heure de la journée à laquelle la commandée a été effectuée
- days_since_prior: jours écoulés depuis la dernière commande, avec un maximum de 30 (non applicable pour order$_number = 1$)

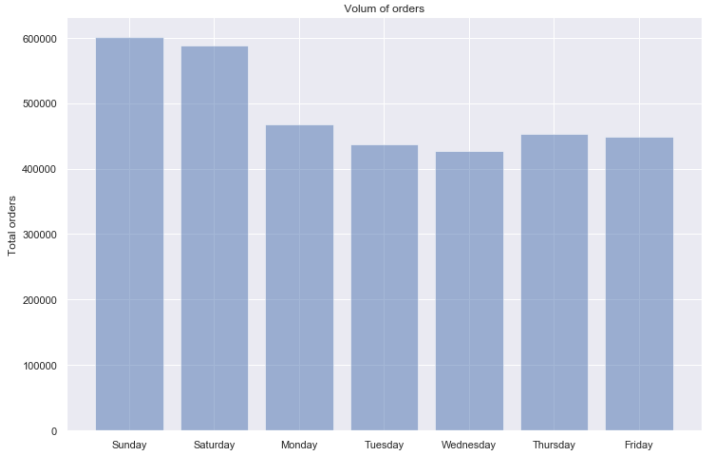
order products (antérieur, entraînement et test) (30m+ lignes):

- order_id: clé étrangère
- product_id: clé étrangère
- add_to_cart_order: commande dans laquelle chaque produit a été ajouté au panier
- reordered: 1 si le produit a déjà été commandé par l'utilisateur dans le passé, sinon 0

2.2. Visualisation exploratoire

L'analyse exploratoire que nous réalisons sur ces données correspond à la distribution des variables catégorielles *order_dow* et *order_hour_of_day*. La 1 illustre la distribution des valeurs pour la variable *order_dow* dans l'ensemble de données.

Figure 1. Volume des commandes par jour de la semaine



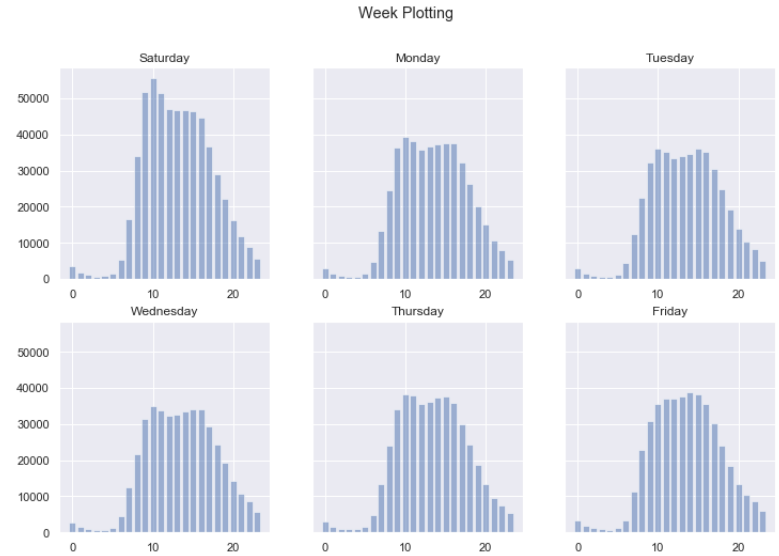
La figure 1 montre une légère différence dans la fréquence des commandes entre les catégories de valeur samedi (*Saturday*) et dimanche (*Sunday*). Étant donné la nature du problème sur lequel nous travaillons, ceci est un résultat attendu, les consommateurs ayant en général plus de temps pour faire des achats pendant le week-end.

La seconde étape de notre analyse exploratoire est effectuée sur la variable *order_hour_of_day*. La figure 2 montre la fréquence des achats par heure de la journée. La figure 3 montre la même analyse appliquée aux autres valeurs de *order_dow*.

Figure 2. Volume des commandes pour le jour de la semaine : Sunday (dimanche)

La catégorie *Sunday* (dimanche) concentre la plus haute fréquence de données, elle est donc montrée comme exemple dans l'ensemble des graphiques de distribution. La gure 2 montre un grand volument de ventes entre les valeurs *9* et *17*, ce qui indique que les matins et les après-midis concentrent un grand volume de ventes pendant le week-end.

Figure 3. Vue hebdomadaire du volume de commandes



Nous devons maintenant vérifier si cette tendance reste la même pendant toute la semaine. Malgré la basse fréquence des données pendant le reste de la semaine, on peut

constater que la tendance à observer une haute fréquence de ventes entre *9* et *17* reste constante. Cette information est pertinente car une variation dans cette tendance pourrait signifier la présence d'anomalies et avoir un impact négatif sur le modèle.

2.3. Algorithmes et techniques

Pour ce travail, nous avons choisi d'appliquer un modèle de classification. Notre intention est de mettre au point une routine qui puisse produire un vecteur de probabilités sur les articles qui appartiennent à l'ensemble I. L'objectif principal est de calculer la probabilité $Pr(c$ qu'un individu soit intéressé par i dans une séquence temporelle $t_c + 1|E_c)$.

Maintenant que nous avons une hypothèse formulée, puisque nous sommes en présence d'un problème de classification qui prend en compte la connaissance antérieure pour évaluer la probabilité de notre hypothèse, nous pouvons appliquer des outils mathématiques orientés vers la prédiction d'occurrences de probabilités [Science 2018].

Comme entrée, notre solution reçoit une évènement de consommation E_c, défini par une séquence d'articles choisis. Ensuite, nous testons les résultats d'une classification naïve bayésienne (Naive Bayes Classier) et d'une forêt aléatoire (Random Forest model) pour prédire le prochain article qu'un consommateur c, a le plus de chances de vouloir choisir, afin de complémenter le panier d'achats comme un système de recommandation [Science 2018].

L'algorithme de type forêt aléatoire a des caractéristiques qui sont plus pertinentes que d'autres algorithmes de classification dans le cas du problèmes que nous traitons. La forêt aléatoire permet de prédire si un résultat appartient à une classe spécifique grâce à une valeur de probabilité. De plus, étant donné qu'il se repose sur l'apprentissage par arbre de décision, ce modèle est difficilement sujet au surapprentissage

Afin d'obtenir la meilleure performance possible de l'algorithme forêt aléatoire, il est nécessaire, avant même d'exécuter le modèle, de pratiquer une extraction de caractéristiques (feature engineering) dans le but d'ajouter des données de meilleure qualité à l'ensemble d'entraînement. Lorsque les premières options seront épuisées et continueront à chercher de meilleures valeurs de performance, so parameter seront réglés pour optimiser le modèle de classification.

Les hyperparamètres sont des valeurs qui sont définites avant la phase d'entraînement. Dans le cas présent, nous travaillons sur le nombre d'arbres de décision dans la forêt et sur le nombre de caractéristique utilisées pour séparer les nœuds terminaux, appelés feuilles (leaves).

- n_estimators
- min_samples_split
- min_samples_leaf
- max_features
- max_depth
- bootstrap

2.4. Test de performance

Tout comme les paramètres de succès du défi Kaggle, dont nous nous inspirons, nous utilisons en premier recours la valeur de 0.4091449 omme maximum idéal, basé sur

le panel Leaderboard de Kaggle. Nous considérons acceptable un résultat situé dans l'intervalle entre 0.4074450 et 0.4047891, ce qui correspond aux 50 meilleurs résultats de la compétition [Kaggle 2017].

En deuxième recours, nous comparons les résultats obtenus par les deux classeurs. Notre objectif est de vérifier lequel des deux modèles a une meilleure performance et produit des résultats plus exacts. Par exemple, nous pouvons structurer notre classification naïve bayésienne en nous basant sur des caractéristiques décrivant une règle de décision d'achat des consommateurs pour le produit i, puis comparer les résultats avec le modèle de forêt aléatoire, qui utilise une sélections aléatoire de caractéristiques basée sur les évènements antérieurs [Science 2018].

3. Méthodologie

3.1. Pré-traitement des données

Le pré-traitement des données est une approche générale de l'amélioration des modèles de machine learning, comme le modèle de forêt aléatoire. Dans cette sections, nous expliquons comment nous utilisons l'extraction de caractéristiques pour améliorer les résultats moyens pour la précision (*Precision*), le rappel (*Recall*) et *F1-Score* [Koehrsen 2018].

- Procéder à l'extraction de caractéristiques pour extraire les informations sur le comportement des utilisateurs (*User Behavior*)
- Procéder au pré-traitement des données pour des ensembles de produits
- Combiner les nouvelles caractéristiques en tant qu'ensemble de données *Baskets*
- Appliquer *Baskets* dans les modèles d'entraînement

La section Visualisation Exploratoire, nous avons expliqué qu'il existe des tendances dans les fréquences d'achats pendant la semaine, avec des plus hautes fréquences d'achats enregistrées pendant le week-end. Étant donné que les attributs d'achat sont associés à des valeurs importantes, nous commençons par comptées les données continues liées à l'intérêt des consommateurs pour un produit dans une perspective générale, comme suit:

- total_orders: nombre total de commande par consommateurr
- count_items_on_cart: nombre total d'articles dans le panier par consommateur
- general_item_reorder_freq: fréquence d'achat par produit

L'ensemble d'entraînement est basé sur l'ensemble de commandes antérieures, c'est à dire sur les commandes dont la *eval_set* est égale à *prior*. Avant de calculture les valeurs pour les caractéristiques ci-dessus, il a été nécessaire de combiner d'autres ensembles de données afin de pouvoir procéder aux calculs nécessaires. Ainsi, la première séquence de regroupement de données est:

- grouper l'ensemble *order_products* avec l'ensemble *orders* avec l'attribut *order_id* pour à la fois les valeurs *prior* et *train* de *eval_set*
- grouper l'ensemble *order_products prior* avec l'ensemble *products* avec l'attribut *product_id*

Nous avons utilisé la bibliothèque *Pandas* pour entreprendre les étapes de prétraitement des données décrites. Nous montrons ci-dessous une partie de la routine implémentée pour le regroupement des données de commandes, utilisateurs et produits.

```
import pandas

# Load data
orders = pandas.read_csv("orders.csv")
order_products_train = pandas.read_csv("order_products__train.csv")
order_products_prior = pandas.read_csv("order_products__prior.csv")
products = pandas.read_csv("products.csv")
```

Le processus de regroupement des ensembles de données a été effectué en suivant les méthodes merge et concat de la bibliothèque Pandas. De cette manière, nous avons pu regrouper les données grâce au critère relation, et rassembler des données sans critère antérieur, comme la concaténation des données.

```
orders_products_train =
pandas.merge(orders[orders['eval_set']=='train'], order_products_train,
    on='order_id',
    how='right')

orders_products_prior =
pandas.merge(orders[orders['eval_set']=='prior'],
    order_products_prior,
    on='order_id',
    how='right')

frames = [orders_products_train, orders_products_prior]
orders_products = pandas.concat(frames)
```

L'ensemble *order_products* nous ermet de savoir si un produit a déjà été commandé plus d'une fois par un consommateur grâce à l'attribut *reordered* et nous permet de connaître la quantité d'articles dans un panier grâce à l'attribut *add_to_cart_order*. Les informations concernant l'utilisateur ont été obtenues grâce à l'ensemble d'attributs orders.

La première partie de l'extraction de caractéristiques est d'extraire les informations pertinentes relatives aux utilisateurs ou aux achats de l'ensemble de données. Par exemple, nous considérons important de connaître la fréquence à laquelle chaque utilisateur commande un produit de nouveau.

```
# 2.1 : Select reordered products
products_reordered = orders_products[orders_products['reordered']==1]

# 2.2 : Count reordered frequency for each user
count_items_on_cart =
    products_reordered
    .groupby(['user_id'])['add_to_cart_order']
    .aggregate('count')

count_items_on_cart =
```

```
    count_items_on_cart
    .reset_index(name='count_reorder_by_user')
```

Le processus d'extraction de caractéristiques ne concerne pas seulement l'ajout de caractéristiques dans l'ensemble de données. Après avoir inséré les caractéristiques mentionnées ci-dessus dans l'ensemble d'entraînement, nous avons supprimé les caractéristiques suivantes : *eval_set*, *add_to_cart_order*, *reordered*, *days_since_prior_order*.

Ces caractéristiques ont été utilisées soit pour calculer les nouvelles caractéristiques, soit n'ont aucune corrélation pertinente avec les données sur l'intérêt des consommateurs pour les produits. Enfin, la figure 4 montre les 10 premières lignes de l'ensemble d'entraînement.

Figure 4. Dix premières lignes du nouvel ensemble d'entraînement

order_id	product_id	user_id	order_number	order_dow	order_hour_of_day	general_item_reorder_freq	count_items_on_cart	total_orders
1	49302	112108	4	4	10	101	9	3
1	11109	112108	4	4	10	3192	9	3
1	10246	112108	4	4	10	12498	9	3
1	49683	112108	4	4	10	67313	9	3
1	43633	112108	4	4	10	312	9	3
1	13176	112108	4	4	10	315913	9	3
1	47209	112108	4	4	10	170131	9	3
1	22035	112108	4	4	10	45639	9	3
816049	49302	47901	14	4	6	101	22	13
816049	22035	47901	14	4	6	45639	22	13
816049	20574	47901	14	4	6	7622	22	13

Afin de vérifier si nous sommes sur la bonne voie, nous procédons à l'extraction d'un rapport préliminaire avant et après les modifications dans l'ensemble d'entraînement. Le tableau 3.1 montre les premiers effets dans les résultats apreès l'insertion des nouvelles caractéristiques. Pour les trois indicateurs, nous avons obtenu un gain de 0.03. De plus, nous avons amélioré la justesse (accuracy) de 0.644 à 0.668.

Table 1. Moyenne et valeurs totales avant et après l'extraction de caractéristiques

avg / total	Before	After
Precision	0.64	0.67
Recall	0.64	0.67
F1-Score	0.64	0.67

Malgré l'impact positif obtenu sur les résultats précédents, il est crucial de s'assurer que toutes les données sont évaluées de manière adéquate. Par exemple, après quelques séries de combinaisons entre les ensembles de données et les fonctions d'agrégation, nous ne pouvons pas garantir que les n-uplets (tuples) ont été associés correctement.

Ainsi, pendant la phase d'assemblage de *user behavior*, nous avons décidé de réaliser un test avec des ID égaux à 1 afin de vérifier que le total d'achats dans l'historique d'un consommateur reste le même jusqu'au moment de la formation de l'ensemble de données utilisé comme entrée dans les modèles de prédiction.

La variable *count_user_1_orders* quantifie le totale des requêtes d'utilisateur 1 and l'ensemble de données *orders*. Après la fusion des ensembles de données *order_products* nous devons nous assurer qu'il n'y a aucune duplication superflue des achats enregistrés. Pour ceci, nous utilisons la variable *count_user_1_orders_products* pour calculer la commande totale de l'utilisateur 1 après la fusion.

La figure 5 montre les valeurs de chacune des deux variables. Ainsi, nous sommes en mesure de vérifier qu'avant et après la fusion entre les ensembles de données le total des requêtes pour l'utilisateur 1 est resté le même.

Figure 5. Test sur le total des commandes avant et après la fusion des ensembles pour l'utilisateur 1

count_user_1_orders	int	1	11
count_user_1_orders_products	Series	(11,)	Series object of pandas.core.series module

Le résultat attendu pour l'ensemble de données *User Bahavior* est une combinaison d'informations générales sur le consommateur qui lie le produit au consommateur. Ainsi, pour chaque utilisateur, nous nous attendons à une corrélation entre le nombre de traces dans *User Bahavior* et les commandes totales. Ceci est illustré par la figure 6, qui montre à peine plus de 11 traces pour l'utilisateur 1. Ce résultat était attendu puis This is illustrated in gure 6, where we can see a little more than eleven records for user 1; expected result since *User Bahavior* a request assembles at least one or more products.

Figure 6. Test sur l'ensemble de données User Behavior pour l'utilisateur 1

Index	user_id	count_reorder_by_user	count_orders_by_user	product_id	count_reorder_product_by_user
0	1	51	11	196	10
1	1	51	11	10258	9
2	1	51	11	12427	9
3	1	51	11	13032	3
4	1	51	11	13176	1
5	1	51	11	25133	8
6	1	51	11	26088	2
7	1	51	11	26405	2
8	1	51	11	38928	1
9	1	51	11	39657	1
10	1	51	11	46149	3
11	1	51	11	49235	2

Une fois que nous avons rassemblé les paramètres liés au comportement d'achat de l'utilisateur, et les paramètres pertinents à l'identification des produits, il a été possible de rassembler des informations dans le dernier ensemble de données utilisé pour le processus d'entraînement du modèle.

```
baskets = pandas.merge(general_user_behavior,
products_preferences,
on=['user_id'],
how='right',
validate='one_to_many')
baskets = pandas.merge(product_user_behavior,
baskets,
on=['user_id', 'product_id'],
how='right')
baskets = baskets.drop(['eval_set', 'order_number'], axis=1)
```

3.2. Implémentation

Le processus d'implémentation se résume à:

1. Phase d'entraînement
2. Phase d'extraction des métriques
3. Phase d'application

A ce stade, le classeur reçoit comme entrée les données de sortie de la routine de pré-traitement. L'implémentation de l'étape d'entraînement a été effectuée dans le fichier python train.py. Les routines implémentées dans le fichier peuvent être effectuées comme suit:

1. Importer les bibliotèques nécessaires qui contiennent les classeurs
2. Importer les méthodes de pré-traitement et effectuer la routine tel que décrit dans la section précédente
3. Partager une partie des données entre entraînement et test : cette étape peut etre effectuée grâce à la méthode *train_set_split*
4. Instancier le classeur forêt aléatoire
5. Entraîner le classeur à l'aide des données d'entraînement obtenues à l'étape 3
6. Tester la prédiction sur l'ensemble de test obtenu à l'étape 3
7. Calculer les métriques de performance
8. Si les métriques ne donnent pas satisfaction, vérifier les hyperparamètres et retourner à l'étape 4

Nous présentons ci-dessous une pré-visualisation du code décrit aux étapes 1, 2, 3 et 4. Pour l'implémentation du modèle de classification forêt aléatoire nous avons choisi une version fournie par la bibliothèque *sklearn*. La routine de séparation entre l'ensemble de test et l'ensemble d'entraînement a été obtenue de la même bibliothèque.

Nous avons choisi la bibliothèque *sklearn* car elle permet d'implémenter une version consolidée du classeur, et fournit une syntaxe lisible par le code dans la phase d'entraînement, ce qui nous a permis de nous concentrer sur des tâches telles que la validation et l'amélioration des résultats.

```
from sklearn.model_selection import train_test_split
# extract X_train and y_train
X_train, X_test, y_train, y_test =
    train_test_split(baskets_x, baskets_y, test_size=0.2)

# Random Forest Classification training
```

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 24,
                max_features= 6,
                random_state = 42)
classifier.fit(X_train,y_train)

# Use the resulting model to predict over the testing baskets
prediction_result = classifier.predict(X_test)
```

3.3. Raffinement

Dans la phase de raffinement, nous cherchons à améliorer les résultats obtenus pendant le processus d'extraction de caractéristiques. Nous pouvons considérer les résultats obtenus pendant ce processus comme de bons résultats ; nous souhaitons cependant les améliorer encore plus. Dans ce but, nous optimisons le modèle de forêt aléatoire à l'aides de ressource obtenues dans la bibliothèque Scikit-Learn.

Le processus de réglage de l'hyperparamètre doit être effectué avant le processus d'entraînement d'un modèle, l'analyste de données peut donc s'en charger. Afin de déterminer les valeurs optimales des hyperparamètres, nous effectuons plusieurs tests en alternant entre les valeurs de l'hyperparamètre selon différentes combinaisons.

Dans le cas de la forêt aléatoire, la configuration de l'hyperparamètre inclut le nombre total d'arbres de décision dans la forêt et le nombre de caractéristiques considéré idéal pour chaque arbre idéal lors de la séparation des feuilles. Dans la mesure où ce processus effectué sur l'ensemble d'entraînement est expérimental, un surapprentissage peut avoir lieu. Nous utilisons donc une méthode de validation croisée fournie par Scikit-Learn pour définir les intervalles optimaux des hyperparamètres.

```
{
    'n_estimators': 200,
    'min_samples_split': 5,
    'min_samples_leaf': 4,
    'max_features': 'auto',
    'max_depth': 10,
    'bootstrap': True
}
```

4. Résultats

4.1. Évaluation et validation du modèle

À la fin du processus de développement, nous avons mis de côté une partie de l'ensemble de données afin de procéder à la phase de testage. Ceci a été effectué après l'extraction des caractéristiques et avant le processus de réglage dee hyperparamètres.

Nous avons défini les paramètres finaux pour la modèle de forêt aléatoire en nous appuyant sur des méthodes disponibles dans la bibliothèque Scikit-Learn library, en suivant la petite partie du code qui suit:

```
from sklearn.model_selection import RandomizedSearchCV
RandomizedSearchCV(estimator = random_classifier,
                               param_distributions = random_grid,
```

```
                                n_iter = 100, cv = 3,
                                verbose=2,
                                random_state=42,
                                n_jobs = -1)
```

Afin de vérifier les premiers résultats, il nous a été nécessaire de choisir un des deux modèles sélectionnés. Afin de prendre la meilleure décision possible, nous utilisons les premiers résultats présentés dans le rapport de classification, avant le réglage des hyperparamètres, dans les figures 7 et 8.

Figure 7. Premier rapport de classification pour 20k exemples lors de l'entraînement du classeur type forêt aléatoire

```
Accuracy on test set: 0.61825
             precision    recall  f1-score   support

          0       0.08 |    0.05      0.06      1000
          1       0.72      0.81      0.76      3000

avg / total       0.56      0.62      0.59      4000
```

Figure 8. Premier rapport de classification pour 20k exemples lors de l'entraînement du classeur type bayésien

```
Accuracy on test set: 0.60325
             precision    recall  f1-score   support

          0       0.33      0.68      0.45       937
          1       0.86      0.58      0.69      3063

avg / total       0.73      0.60      0.63      4000
```

4.2. Justication

Nous avons utilisé la formule du F1-Score formula, expliquée dans la section métriques, pour valider notre modèle type forêt aléatoire le comparer aux résultats du leaderboard Kaggle. Notre proposition diffère de celle du défi Kaggle sous plusieurs aspects, notamment car notre modèle ne vise pas à répondre à une commande déjà effectuée, mais cherche plutôt à désigner le produit le plus intéressant pour une utilisateur donné.

L'extraction de caractéristiques et le processus d'entraînement ont été effectués dans la suite Anaconda, en utilisant les ressources de Spyder, sur un ordinateur MacBook Pro. Malgré les ressources disponibles, il nous a été nécessaire de réaliser une réduction de l'ensemble de données afin de rendre l'étude faisable, puisque le le noyau n'arrivait plus à prendre en charge l'entraînement des grandes quantités de données dans l'ensemble de données original.

Même si la sortie de données de notre méthode n'est pas exactement la même que celle du défi Kaggle, elle est validée d'une manière comparable, c'est à dire en vérifiant si un produit a été commandé à nouveau ou non pour chaque requête spécifique.

Dans le cas de notre étude, l'identifiant unique de la commande ne fait pas partie de l'ensemble de données d'entrée puisque nous entraînons le comportement et les préférences de consommation. Le tableau ci-dessous montre les trois premiers modèles du leaderboard de Kaggle et des résultats du F1-Score de chacun d'entre eux.

Nous avons essayé de corréler les modèles appliqués, mais il a seulement été possible d'identifier le second modèle. Nous avons ensuite ajouté le score obtenu par notre modèle, après l'extractionde caractéristiques, avant et après l'amélioration du modèle.

Position	Kaggle Score	Kaggle Model	Random Forest Score	Tuned Random Forest Score
1	0.4091	NA	0.59	0.70
2	0.4082	XGBoost		
3	0.4081	NA		

5. Conclusion

5.1. Visualisation

Le tableau ci-dessous illustre un des enregistrements de prédiction de consommation du produit Plain Bagelettes pour l'utilisateur 91. La sortie finale du résultat montre la valeur de probabilité d'association avec chaque classe. De cette manière, elle montre les meilleures intervalles de recommandation (ou pas) d'un produit lorsqu'il existe une succession de valeurs proches les unes des autres.

Product	User	Weekday	Hour	Not Recommended (%)	Recommended (%)
Plain Bagelettes	91	Friday	13	0.46	0.53

5.2. Réflexion

Notre solution proposée au problème posé se résume de la manière suivante:

1. Définir formellement le problème décrit plus haut dans ce document.
2. Chercher des articles et des ressources de recherche.
3. Denir le langage et la bibliothèque à utiliser.
4. Choisir l'IDEs pour l'implémentation du code.
5. Structurer les étapes nécessaires à la résolution du problème.
6. Dénir les modèles pour les phases d'entraînement.
7. Effectuer les étapes de résolution du problème.
8. Systématiser les résultats afin d'évaluer et d'améliorer les modèles.

Sans surprise, les étapes les plus compliquée du développement de ce projet ont été les étapes liées au pré-traitement des caractéristiques et à l'extraction de caractéristiques (étape 7). L'ensemble de données Instacart est sans aucun doute l'un des plus complets, voire le seul, dans le domaine des données liées aux achats dans le secteur alimentaire.

Cette grande quantité d'informations doit être traitée avec le plus grand soin. Pendant l'implémentation de l'étape 7, nous avons réalisé des vérifications ponctuelles afin de nous assurer que l'ensemble d'entraînement final ne présentait pas d'informations contradictoires ou de doublons.

5.3. Améliorations

Nous pourrions appliquer l'entraînement à une plus grande partie de l'ensemble de données pour améliorer les résultats des métriques du F1-Score si nous avions plus de matériel hardware à notre disposition.

Nous proposons la configuration suivante pour une meilleure exécution dans le travail avec des ensembles de données tels que celui d'Instacart:

- Core i7 8700k
- 16gb RAM
- GeForce 1070ti
- Motherboard Asus Z370-F gaming

Reference

Amazon (2003). Amazon.com recommendations.

Basilico, J. and Raimond, Y. (2017). Déjà vu: The importance of time and causality in recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 342–342, New York, NY, USA. ACM.

Instacart (2017). The instacart online grocery shopping dataset 2017 data descriptions.

Kaggle (2017). Instacart market basket analysis.

Kaur, M. and Kang, S. (2016). Market basket analysis: Identify the changing trends of market data using association rule mining. *Procedia Computer Science*, 85:78 – 85. International Conference on Computational Modelling and Security (CMS 2016).

Koehrsen, W. (2018). Gathering more data and feature engineering.

Prentice-hall, M. Probability and statistics for engineers and scientists, by r.e. walpole, r.h. myers, and s.l.

Science, T. D. (2018). The random forest algorithm.