POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

LOG6308 : Système de recommandations Application iOS : WebRecommendation

Projet de session - Rapport

Thomas Neyraut
ECOLE POLYTECHNIQUE DE MONTREAL

Table des matières

Introduction

Dans le cadre du projet de session du cours LOG6308 – Système de recommandations, nous avons conçu et implémenté un système de recommandations sur iOS. Ce document présente l'intégralité du projet réalisé. Dans une première partie, nous présentons le constat depuis lequel nous sommes partis pour faire émerger ce projet, ainsi que les objectifs que nous nous sommes fixés. Dans une seconde partie, nous présentons l'ensemble des fonctionnalités de l'application iOS. Une troisième partie présente le fonctionnement du système de recommandations. Dans la quatrième partie, nous présentons l'ensemble des implémentations que nous avons réalisées. Pour finir dans une cinquième et dernière partie, nous présentons les difficultés que nous avons rencontrés ainsi que les travaux futurs que nous allons effectuer sur le sujet. L'intégralité de nos implémentations sont accessibles sur le GitHub à l'adresse suivante : https://github.com/tneyraut/RecommendingWeb. Pour comprendre l'organisation des fichiers dans les dossiers du projet, veuillez-vous référer au fichier README.

1) Constat et objectifs

L'idée de ce projet nous est venu d'une constatation assez simple. De plus en plus d'utilisateurs abandonnent les ordinateurs au profit des tablettes, et de plus en plus d'utilisateurs utilisent une tablette pour naviguer sur le web. Personnaliser son navigateur web prend aussi un certain temps. Il faut sauvegarder dans ses favoris les différentes pages web que l'on apprécie, organiser ses favoris pour pouvoir s'y retrouver quand on a beaucoup de pages web qui y sont sauvegardées. Si l'utilisateur décide de ne pas passer par les favoris, il perd à chaque fois du temps à taper dans le champ de recherche le nom du site qu'il souhaite visiter. Sinon, il doit conserver chacun des sites qu'il visite régulièrement dans un onglet actif. On peut alors se retrouver avec un nombre d'onglets d'ouverts très important et ne plus arriver à s'y retrouver. Nous avons aussi remarqué qu'il est difficile de trouver de nouveaux sites web qui pourraient nous intéresser.

Nous avons donc eu l'idée de concevoir une application iOS intégrant une navigation web et effectuant deux types de recommandations à l'utilisateur :

- Recommandation du site web que l'utilisateur devrait souhaiter visiter,
- Recommandation de nouveaux site web.

Ainsi, nos objectifs au cours de ce projet ont été les suivants :

- Implémenter une application iOS intégrant une navigation web,
- Implémenter sur l'application iOS un système de recommandation recommandant à l'utilisateur le site web qu'il devrait vouloir visiter à l'instant considéré,
- Implémenter sur l'application iOS un système de recommandation recommandant à l'utilisateur des nouveaux sites web qui devraient l'intéresser à l'instant considéré.

2) Fonctionnalités de l'application iOS

Les fonctionnalités qu'intègrent l'application iOS sont les suivantes :

- Navigation web,
- Recommandation du site web qui devrait être visité par l'utilisateur.

Cependant, l'application iOS n'intègre pas la fonctionnalité suivante : « Recommandation de nouveaux sites web qui devraient intéresser l'utilisateur ». Cette fonctionnalité repose sur une approche collaborative user-user. De ce fait, elle nécessite l'utilisation de calculs matriciels complexes qui ne sont pas directement implémentés en Swift sur la plateforme iOS comme ils le sont en R. Nous avons donc cherché à implémenter les différents calculs matriciels nécessaires à l'emploi d'une telle méthode mais nous avons manqué de temps pour mener à terme cette implémentation. Nous avons donc choisi d'implémenter cette fonctionnalité en R. Ce choix est de toute manière le meilleur puisque cette recommandation nécessite l'utilisation d'une méthode collaborative. Il est donc inconcevable de fournir à chaque appareil (iPhone et iPad) l'ensemble des données des utilisateurs pour des raisons évidentes de place de stockage sur les iPhones et iPad, mais aussi pour des raisons de confidentialités. Aucun utilisateur ne devrait disposer sur son appareil les données d'autres utilisateurs. Au final, cette recommandation pour chaque utilisateur devra être calculé sur un serveur distant disposant de toutes les données des utilisateurs et son résultat devra être transmis à l'appareil de l'utilisateur considéré. Le diagramme des cas d'utilisations ci-dessous (figure 1) présente les différents acteurs ainsi que les différentes fonctionnalités du système. Les fonctionnalités présentes en rouge n'ont pas encore été développées jusqu'à présent.

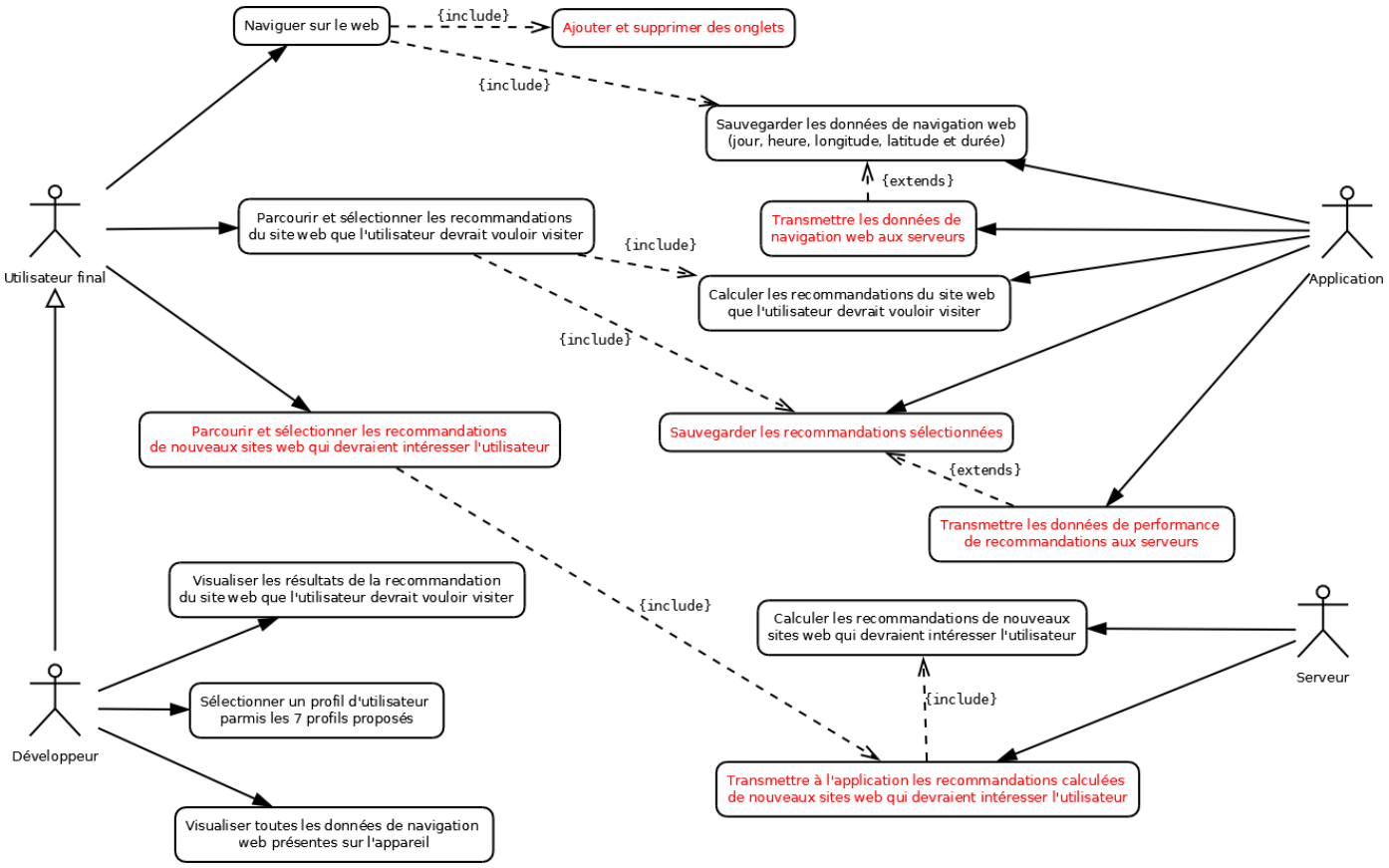

Figure 1. Diagramme des cas d'utilisation de l'application

La figure 2 ci-dessous présente l'unique interface qui compose l'application finale pour les utilisateurs. Cette view est composé de trois éléments graphiques :

- L'élément graphique principal est la WebView qui affiche par défaut la page Google à chaque nouveau lancement de l'application,
- Le second élément graphique important ici est la collectionView qui affiche les différentes recommandations (site web qu'il devrait vouloir visiter) faites à l'utilisateur. Chaque cellule représente un site web et l'utilisateur peut accéder au site web en sélectionnant la cellule correspondante. Les sites web sont ordonnés de la gauche vers la droite, du site web le plus probable à visiter au site web le moins probable,
- Le dernier élément graphique intéressant ici est la barre de recherche présente tout en haut de l'interface. Cette barre de recherche permet à l'utilisateur d'accéder à un site web directement ou à faire une recherche Google sans passer par le site web de Google.

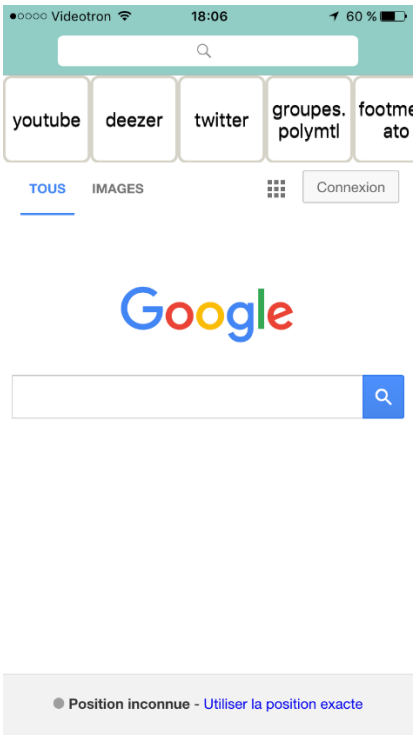

Figure 2. Interface graphique de l'application finale utilisateur

Pour des soucis de tests une version test de l'application a été développé. Cette version comporte quelques fonctionnalités supplémentaires mais qui n'apparaitrons pas dans la version utilisateur finale :

- Visualisation de l'ensemble des données des profils utilisateurs présents sur l'application test,
- Visualisation des résultats détailles des recommandations pour chaque utilisateur.

Ainsi, la version test de l'application est composée quatre interfaces au lieu d'une. La figure 3 ci-dessous présente l'interface d'accueil de cette version test. Chaque cellule de cette interface représente un profil d'utilisateur différent que nous expliciterons plus loin. En cliquant sur l'une des cellules, on débouche sur l'interface unique présentée ci-dessous qui composera la version finale. Tout comme la version finale, il est impossible de sortir de cette view (unique view) à moins de quitter l'application.

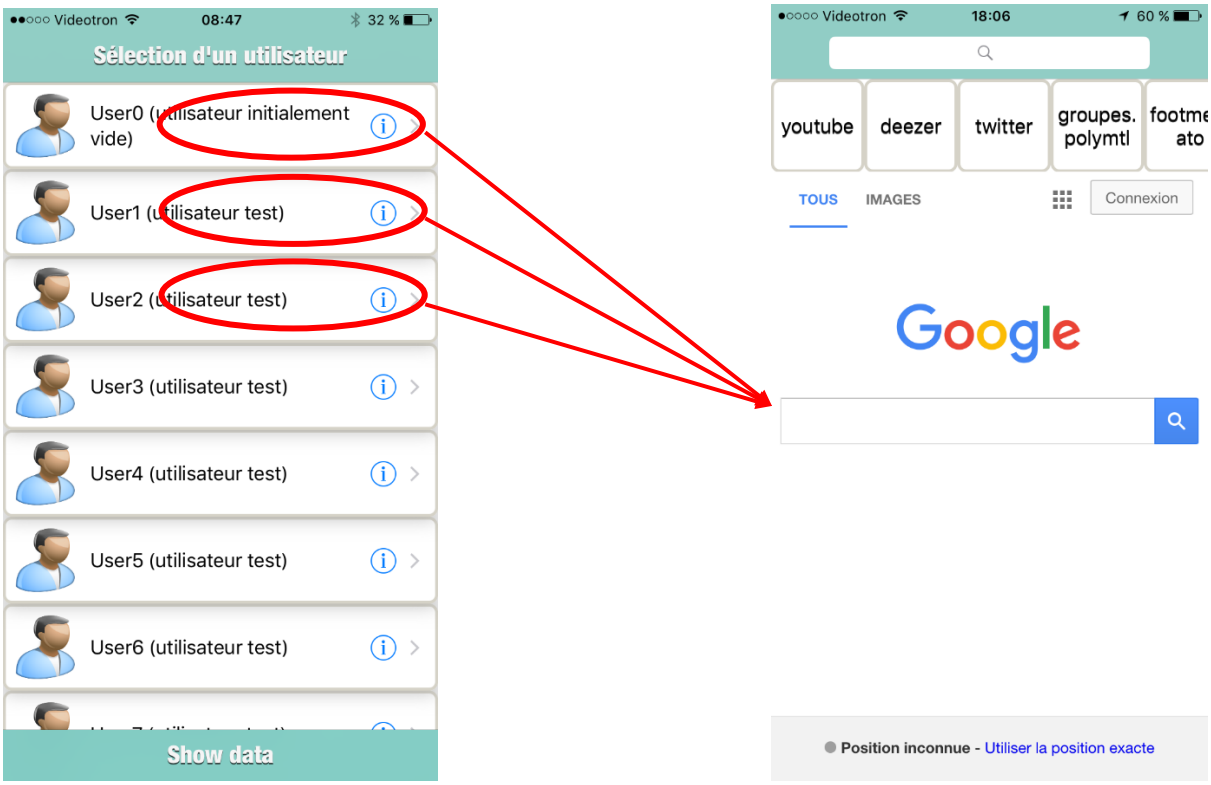

Figure 3. Interface d'accueil de la version test de l'application et accès à l'interface unique de la version finale de l'application

La figure 4 ci-dessous présente une des view de la version test de l'application et comment y accéder depuis la view d'accueil. Cette nouvelle view est une view dans laquelle l'ensemble des données des profils utilisateur présents sur l'appareil est présenté.

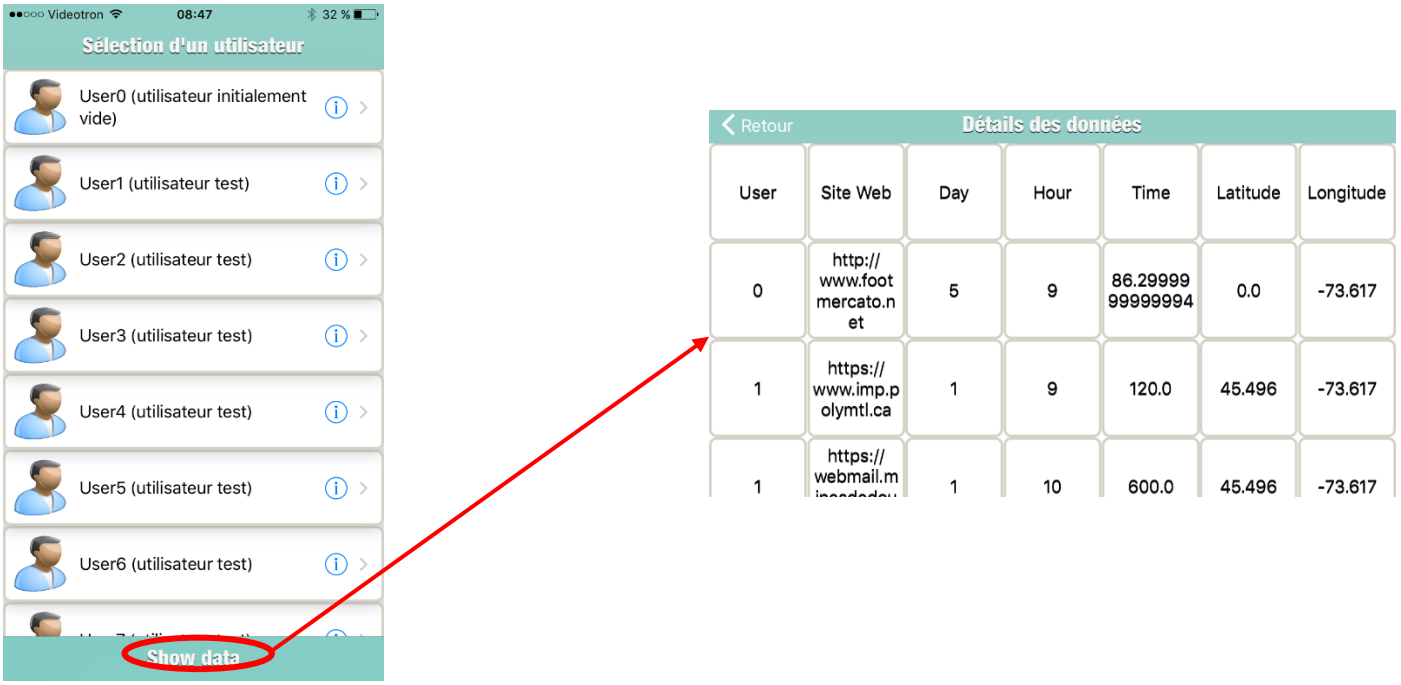

Figure 4. Accès à l'interface des détails des données présentes sur l'appareil

La figure 5 ci-dessous présente la dernière view supplémentaires de la version test de l'application et comment y accéder. Cette view présente les résultats de la recommandation pour l'utilisateur sélectionné. Les sites web sont ordonnées de haut en bas du premier site web que l'utilisateur devrait vouloir visiter à l'instant présent jusqu'au dernier qu'il devrait vouloir visiter à l'instant présent. Plusieurs détails sont fournis en plus avec notamment le score de recommandation calculé par l'application ainsi que le temps global passé par l'utilisateur sur le site web.

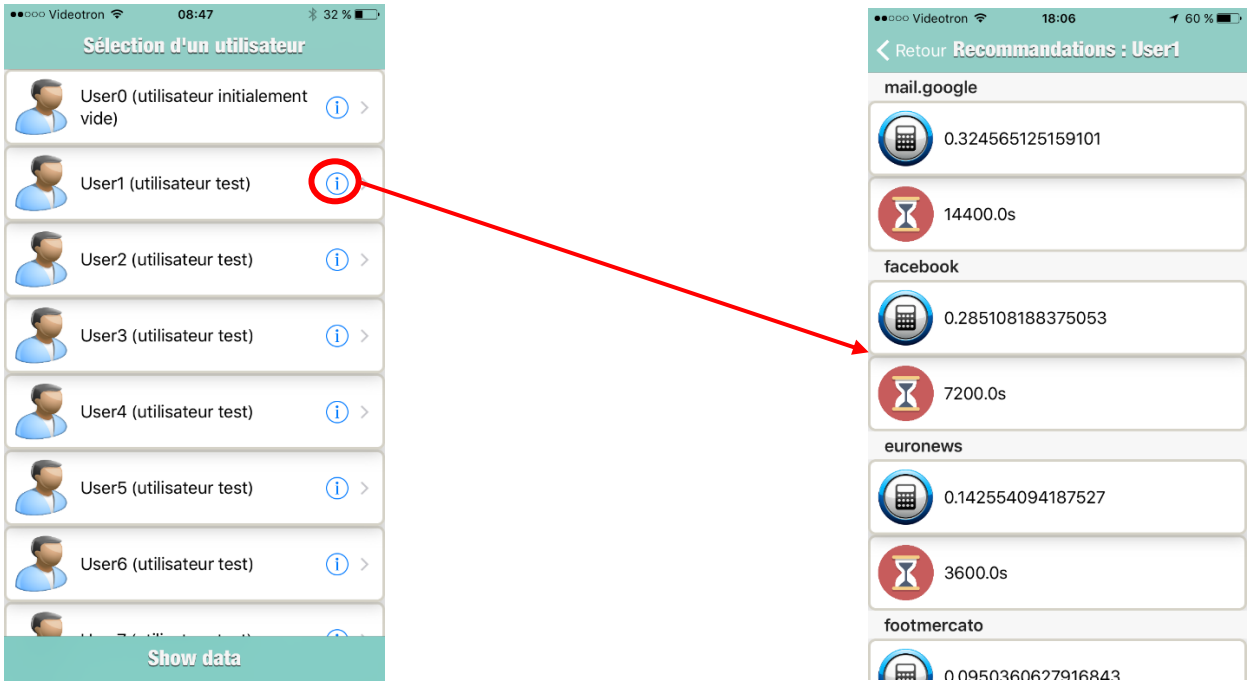

Figure 5. Accès à l'interface des résultats et des détails des recommandations des sites web que l'utilisateur sélectionné devrait vouloir visiter

3) Fonctionnement du système de recommandations

Nous nous étions fixé l'objectif de concevoir un système de recommandations avec deux types de recommandations distincts. En effet, notre application iOS doit :

- Recommander le site web que l'utilisateur devrait vouloir visiter à l'instant considéré,
- Recommander de nouveaux sites web qui devraient intéressés l'utilisateur.

Dans une première sous-partie, nous présentons le fonctionnement de notre système de recommandations pour le premier type de recommandations. Dans une seconde sous-partie, nous présentons le fonctionnement du système de recommandation pour la seconde recommandation.

a) Fonctionnement pour la première recommandation

L'objectif de cette recommandation est de prédire le site web qui devrait être visité par l'utilisateur quel que soit l'instant considéré. Pour cela, notre système de recommandations prend en considération un ensemble d'éléments de contexte pour effectuer ses recommandations :

- Le jour actuel (quel jour sommes-nous ?) : lundi, mardi, mercredi, jeudi, vendredi, samedi ou dimanche,
- Le créneau horaire actuel (dans quel créneau horaire sommes-nous ?) : 0h-5h, 5h-9h, 9h-12h, 12h-14h, 14h-19h, 19h-24h,
- La position de l'utilisateur (où sommes-nous ?) : latitude et longitude.

Grâce à l'ensemble de ses éléments de contexte, notre système de recommandations va récupérer certaines données historiques des précédentes navigations web effectuées par l'utilisateur. L'application enregistre toutes les données des navigations web de l'utilisateur sur 28 jours d'utilisation. Quand l'utilisateur débute un 29^{ième} jour d'utilisation de l'application, les données du 1^{ier} jour d'utilisation sont supprimées de l'application et remplacer par les données d'utilisation du 29^{ième} jour d'utilisation. Les données de navigations web de l'utilisateur sauvegardées par l'application sont les suivantes :

- Nom du domaine du site web visité,
- Jour de la visite : lundi, mardi, mercredi, jeudi, vendredi, samedi ou dimanche,
- Heure du début de la visite : 0h, 1h, 2h, …, 23h, 24h,
- Localisation de l'utilisateur : latitude et longitude,
- Temps passé sur le site.

Il est important de noter que notre application ne sauvegarde pas les navigations effectuées sur le site de Google (Google est de toute manière accessible directement au début de chaque lancement de l'application).

Quand l'utilisateur va ouvrir l'application, celle-ci va calculer automatiquement la recommandation en fonction des éléments de contexte présentés précédemment. Cette recommandation s'effectue en deux étapes :

- Calcul de la recommandation avec les éléments de contexte : Jour actuel et créneau horaire actuel,
- Calcul de la recommandation avec l'éléments de contexte : Position de l'utilisateur (latitude et longitude).

Pour la première étape, nous utilisons une régression linéaire sur le contexte. Nous calculons un score de recommandations s_r pour chaque site web visité qui est égale au produit entre le facteur de régression α et le temps de visité sur le site web considéré t_v : s_r = α * t_v. La figure 5 ci-dessous représente le fonctionnement de cette étape. L'algorithme récupère en entrée le jour d que nous sommes (lundi, mardi, …) ainsi que l'heure actuelle h. Les notations c-1 et c+1 correspondent respectivement au créneau horaire précédent le créneau horaire c, et au créneau horaire suivant le créneau horaire c. De même, les notations d-1 et d+1 correspondent au jour précédent le jour d, et au jour suivant le jour d. Par exemple, pour d = Lundi et c = 0h-5h :

- d-1 = Dimanche avec c = 0h-5h,
- d+1 = Mardi avec c = 0h-5h,
- c-1 = 19h-24h avec d = Dimanche,
- c+1 = 5h-9h avec d = Lundi.

La figure 1 en annexe présente un exemple complet d'une exécution de cet algorithme avec comme élément de contexte : jour = lundi et heure = 10h (créneau horaire correspondant : 9h-12h).

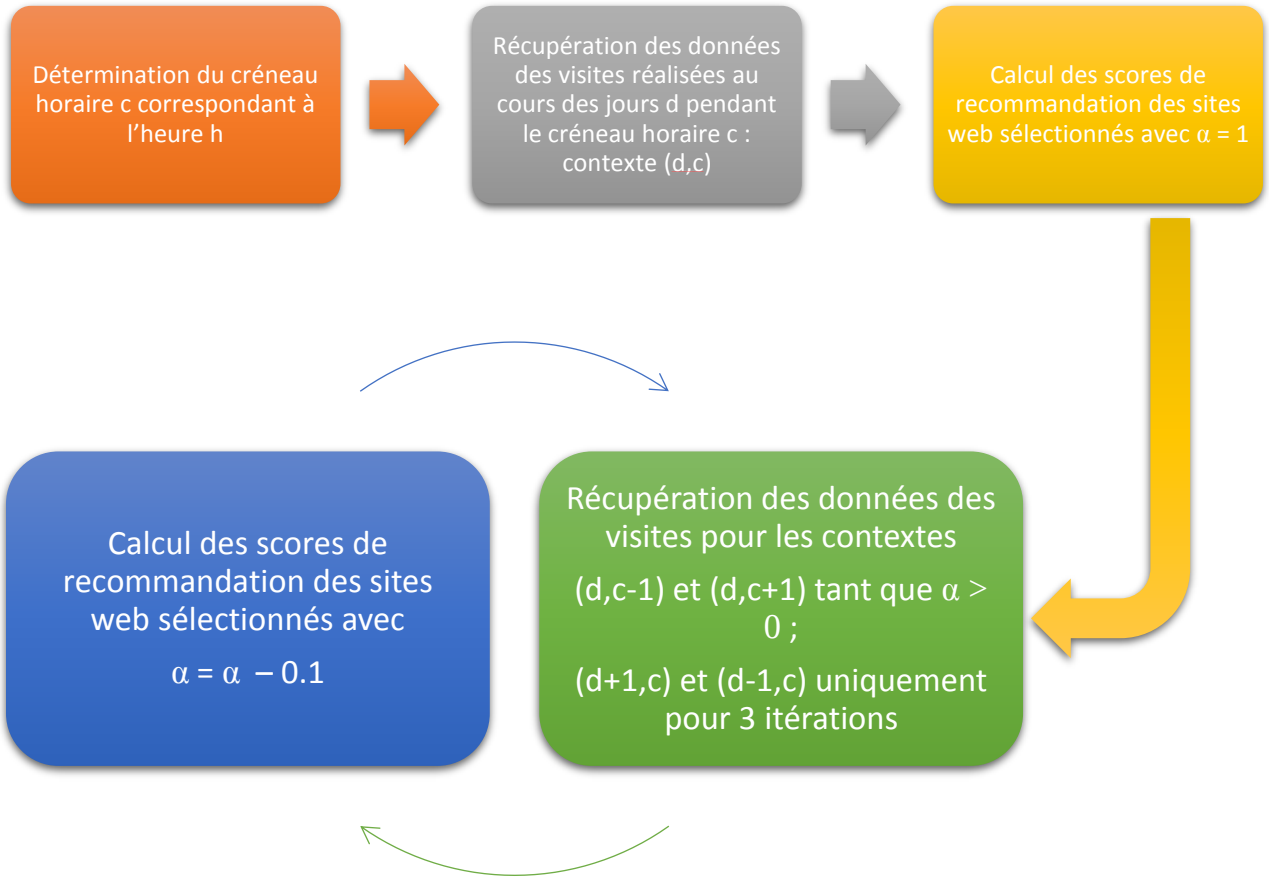

Figure 6. Fonctionnement de l'algorithme de recommandations pour les éléments de contexte du jour et de l'heure

Une fois cette première recommandation calculée, nous récupérons les sites web que l'utilisateur a visité dans le passé à sa position actuelle. Nous calculons ensuite pour chacun des sites web son score de recommandations s_r correspondant au produit entre un facteur α (égale à 2 dans l'implémentation) et le temps de visite t_v : s_v = 2 * t_v. De par le coefficient α égale à 2, nous mettons en avant dans cette recommandations les sites web que l'utilisateur a déjà visité à l'endroit où il se situe.

Par la suite, nous réunissons les deux recommandations calculées, l'une avec les éléments de contexte de jour et d'heure, et l'autre avec les éléments de contexte de position (latitude et longitude). Les scores de recommandations s_r sont sommés pour les sites web apparaissant dans les deux listes de recommandations. Classiquement, on fait une fusion (union) des deux listes. On trie ensuite la liste obtenue par ordre décroissant et on normalise les scores de recommandations. La liste obtenue sera ensuite affichée dans l'ordre sur l'interface graphique de l'application.

b) Fonctionnement pour la seconde recommandation

L'objectif de cette recommandation est de recommander de nouveaux sites web à l'utilisateur considéré en fonction du contexte. En effet tout comme la première recommandation, cette recommandation dépend du contexte, et plus exactement des éléments de contexte suivants :

- Le jour actuel (quel jour sommes-nous ?) : lundi, mardi, mercredi, jeudi, vendredi, samedi ou dimanche,
- Le créneau horaire actuel (dans quel créneau horaire sommes-nous ?) : 0h-5h, 5h-9h, 9h-12h, 12h-14h, 14h-19h, 19h-24h.

Pour répondre à cette recommandation, nous avons employé une approche user-user. Dans notre cas, les sites web sont films, et leurs scores de recommandations pour les différents utilisateurs, les notes qu'ils ont reçu de ses mêmes utilisateurs. Ainsi pour effectuer les calculs de cette recommandation, nous devons dans un premier temps calculer les scores de recommandations pour tous les sites web. Nous calculons donc la première recommandation pour tous les utilisateurs avec uniquement les éléments de contexte du jour et de l'heure considéré, sans l'aspect localisation et aussi sans régression du contexte. Un site web qui n'a pas été visité par un utilisateur pour le jour et l'heure considéré a alors un score de recommandation (note) égale à 0 (l'utilisateur n'a pas vu le film et donc ne l'a pas noté). Les notes sont toutes comprises entre 0 et 1 du fait de la normalisation des scores de recommandations.

Une fois toutes les notes calculées pour tous les utilisateurs et tous les sites web, nous calculons la distance euclidienne entre l'utilisateur considéré et chacun des autres utilisateurs. Nous sélectionnons par la suite les 20 utilisateurs les plus proche de l'utilisateur considéré (ayant une distance euclidienne la plus faible). Nous effectuons ensuite le calcul classique de cette approche avec les valeurs centrées et la constante de normalisation. Ainsi, nous obtenons une liste de sites web à recommander à l'utilisateur considéré.

4) Explication de l'implémentation

Dans cette partie, nous présentons les différentes implémentations que nous avons effectuées. Dans un premier temps, nous présentons l'implémentation de l'application iOS que nous avons réalisée. Dans un second temps, nous présentons l'implémentation en R de nos deux recommandations. Pour finir, nous présentons le programme Java qui nous a permis de générer les données tests que nous avons utilisées.

L'ensemble de nos implémentations sont disponibles sur notre GitHub à l'adresse suivante : https://github.com/tneyraut/RecommendingWeb

a) Implémentation de l'application iOS

La première chose que nous avons effectué au cours de ce projet a été l'implémentation de l'application iOS. Nous avons choisi d'implémenter l'application en Swift et non en Objective-C. Ce choix est un choix purement basé sur l'avenir puisque le langage Objective-C est voué à disparaitre. On peut aussi noter que le langage Swift est beaucoup moins lourd, plus logique et bien plus proche des autres langages de programmation que l'est le langage Objective-C. Nous avons aussi choisi d'implémenter totalement l'application en ligne de code, et donc sans story-board. Le diagramme de classes ci-dessous (figure 7) présente l'implémentation. Notre implémentation est composée des classes suivantes :

- AppDelegate.swift : Ce fichier est généré automatiquement par l'IDE Xcode, c'est le point d'entrée de l'application. Cette classe contient l'équivalent de la méthode main du programme classique. Dans cette classe, nous instancions le navigationController de l'application ainsi que sa première view,
- UsersTableViewController.swift : Cette classe définit l'interface (présente uniquement dans la version test de l'application) permettant de sélectionner l'utilisateur que l'on souhaite considérer pour les tests,
- DetailRecommandationTableViewController.swift : Cette classe définit l'interface (présente uniquement dans la version teste de l'application) permettant de visualiser les détails des résultats de la première recommandation,
- DetailsDataCollectionViewController.swift : Cette classe définit l'interface (présente uniquement dans la version teste de l'application) permettant de visualiser grossièrement l'ensemble des données de navigation sauvegardées sur l'appareil,
- MainViewController : Cette classe définit l'interface principale de l'application et qui est l'unique interface de la version utilisateur finale de l'application. Cette interface permet à l'utilisateur final de naviguer sur le web et lui propose les résultats de la première recommandation,
- RecommendationCollectionViewController.swift : Cette classe définit l'interface exposant à l'utilisateur les résultats de la première recommandation. Cette interface est intégrée directement à l'interface définit par la classe MainViewController,
- Recommendation.swift : Cette classe permet à l'application d'effectuer les différentes recommandations,
- Data.swift : Cette classe permet de sauvegarder les données de navigation et d'y accéder pour effectuer les recommandations,
- Timer.swift : Cette classe static permet de calculer le temps de navigation passé sur chaque site web visité par l'utilisateur,
- UserData.swift : Cette classe permet de charger les données de navigation des utilisateurs tests,
- TimeSolt.swift : Cette classe permet de représenter les différents créneaux horaires et les données qui leurs sont associées,
- WebSite.swift : Cette classe permet de représenter les différents sites web et les données qui leurs sont associées,
- TableViewCellWithImage.swift et CollectionViewCellWithLabel.swift : Ces deux classes permettent de définir la composition et le design de certains éléments graphiques utilisés dans le code de l'application.

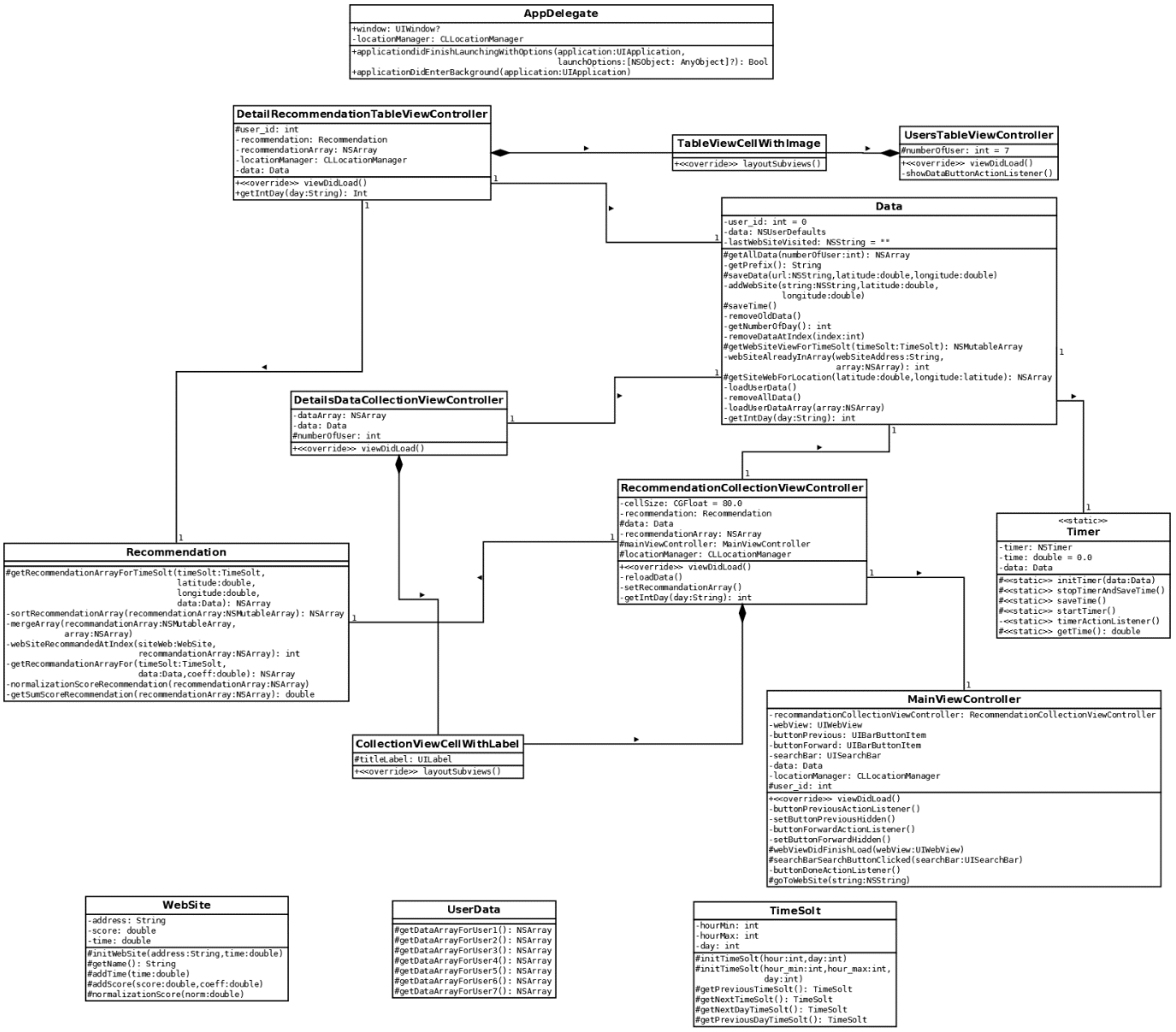

Figure 7. Diagramme de classes de l'application iOS

Pour pouvoir exécuter l'application sur un iPhone ou un iPad réel, il faut disposer d'une machine MacOS avec l'IDE Xcode installé. Il est impossible d'exécuter l'application sur des devices simulées car l'application fait appel à la localisation de l'appareil et les simulateurs ne simulent pas cet aspect d'une device.

Il faut noter que, les données de navigation pour un site web donné sont comptabilisées et sauvegardées si ce site web ne fait pas parti de la black-liste. En effet, certains sites web sont black-listés (par exemple : Google) car cela n'a aucun intérêt de les recommander. Concernant les sauvegardes du temps de visite passé par l'utilisateur sur un site web, notre implémentation répond parfaitement au différent cas d'utilisation. Si l'utilisateur arrête l'exécution de l'application (kill) alors l'application va sauvegarder le temps de visite passé sur le site en cours. L'application va aussi sauvegarder le temps de visite passé sur le site courant si l'utilisateur décide de quitter l'application (mise en background de l'application) sans arrêter l'exécution. Si l'utilisateur relancera l'application alors que celle-ci est en

background (retour en foreground) alors le timer va reprendre à zéro et le temps de visite va commencer à être de nouveau compté pour le site courant si bien sûr celui-ci ne fait pas parti de la black-liste. Enfin, quand l'application était en background et retourne en foreground, la liste des recommandations va être mise à jour car potentiellement, le contexte a changé.

b) Implémentation des recommandations en R

Comme les calculs matriciels ne sont pas aisément réalisable en Swift et que l'intégration de R sur iOS est tout bonnement impossible, nous avons implémenté les calculs de notre seconde recommandation sur un script R (fichier script_r.R). Nous en avons aussi profité pour y implémenter les mêmes calculs que nous avons implémenté sur l'application iOS pour la première recommandation.

Le script R est donc décomposé en plusieurs parties. Nous récupérons dans un premier temps les différents ensembles de données nécessaires à nos calculs. Nous définissons par la suite les différents éléments de contextes :

- L'utilisateur considéré,
- L'heure considérée,
- Le jour considéré,
- La localisation considérée (latitude et longitude).

Ensuite, nous implémentons les calculs concernant la recommandation du site web que l'utilisateur devrait vouloir visiter selon le contexte. Ces calculs sont exactement les mêmes que nous avons implémentés sur l'application iOS et tels qu'ils sont expliqués dans la troisième partie de ce rapport.

Nous implémentons ensuite la seconde recommandation. L'objectif est ici de recommander de nouveaux sites web aux utilisateurs. Les calculs implémentés sont tels qu'ils ont été expliqués dans la troisième partie de ce rapport. Nous avons employé pour cette recommandation une approche user-user. Par rapprochement avec ce que nous avons effectué comme travaux dans le cadre du cours LOG6308, les films sont ici les sites web et les notes des utilisateurs aux films sont les scores de recommandations obtenus par la première recommandation. Les scores ont une valeur entre 0 et 1. Si la valeur d'un score pour un utilisateur et un site web est égale à 0 cela signifie que l'utilisateur n'a pas visité ce site web dans le contexte considéré (l'utilisateur n'a pas vu le film et donc ne l'a pas noté).

c) Implémentation du programme Java DataCreator

Pour pouvoir tester nos deux recommandations, nous avons eu besoin de données tests. Pour cela, nous avons décidé d'implémenter un petit programme Java permettant de gérer les données nécessaires. Ce programme est composé de 5 classes :

- Data.java : Cette classe définie les données de navigation enregistrées par l'application iOS et exploitable par celle-ci ou par le script R,
- TimeSolt.java : Cette classe permet de représenter les différents créneaux horaires,
- Day.java : Cette classe permet de représenter les différents jours,
- UserProfile.java : Cette classe permet de définir différents profils d'utilisateur,
- DataCreator.java : Cette classe est la classe principale du programme.

La classe UserProfile définit un total de six profils d'utilisateur. Un profil d'utilisateur est défini par :

- Un nombre de sites web visités (int nbWebSiteVisited),

- Un pourcentage de temps passé sur le web (int pourcentageTempsPasse),
- Pour chaque créneau horaire de chaque jour, un nombre maximum de sites web visités (tableau maxNbWebSiteVisitedDuringDayTimeSolt),
- Les jours pour lesquels la navigation web de l'utilisateur est la même (tableau daySimilarArray),
- Les créneaux de chaque jour pour lesquels la navigation web de l'utilisateur est la même (tableau timeSoltSimilarArray).

La classe UserProfile définit un dernier profil d'utilisateur mais pour celui-ci, le comportement de l'utilisateur est défini totalement aléatoirement. Le diagramme de classes ci-dessous (figure 8) présente l'implémentation du programme.

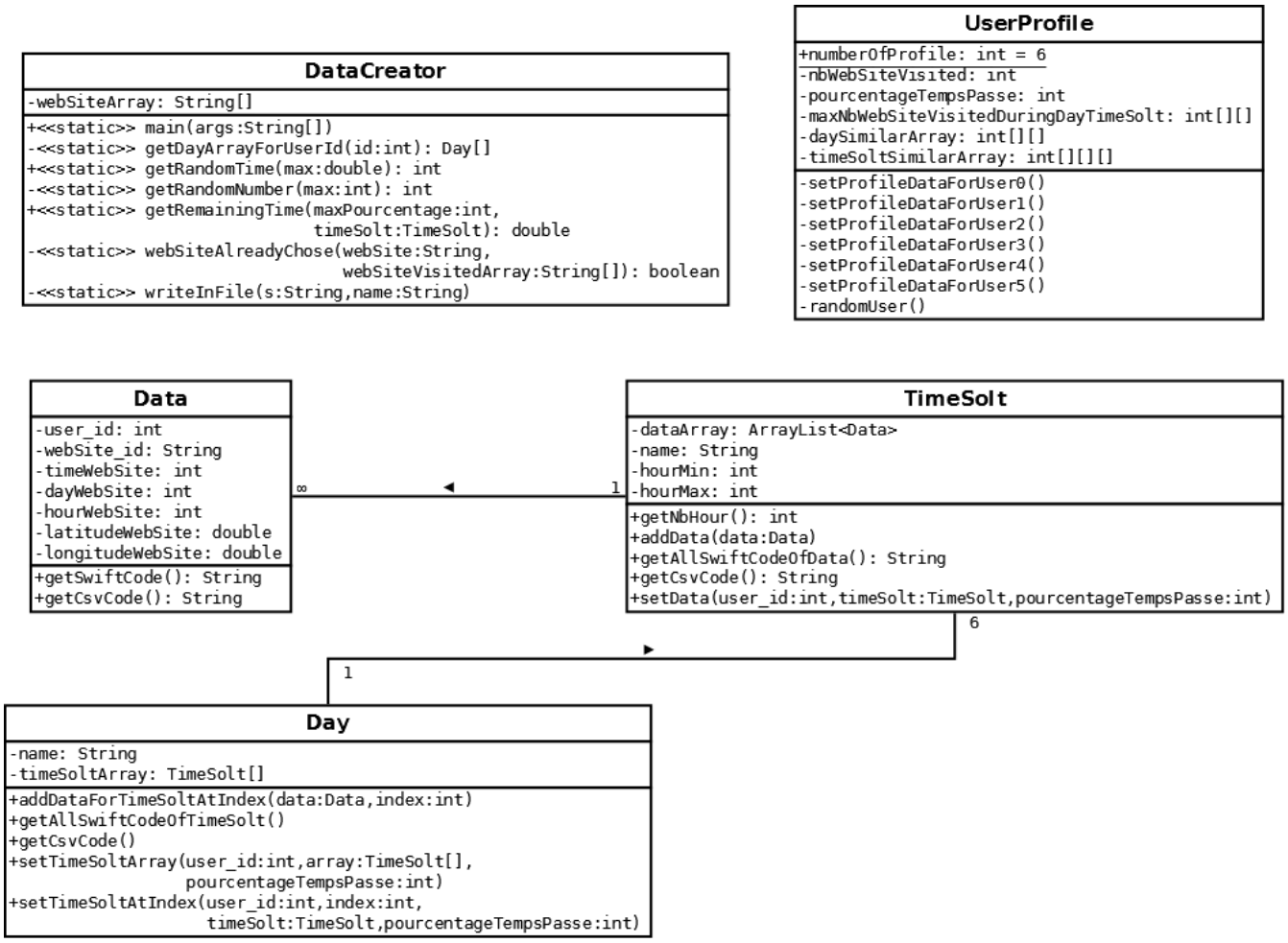

Figure 8. Diagramme de classes du programme Java

Pour pouvoir utiliser ce programme Java, il faut utiliser les commandes suivantes :

- javac file_name.java
- java file_name id

Le paramètre id doit être compris entre -1 et (le nombre de profil utilisateur – 1) inclus. Quand le paramètre id est positif, le programme va générer des données de navigation pour un unique utilisateur de profil type du profil référant à valeur du paramètre id. Si le paramètre id est égale à -1, alors le

programme va générer des données de navigation pour un utilisateur de chaque profil d'utilisateur. Le programme va aussi générer totalement aléatoirement des données de navigation pour 100 utilisateurs. Ainsi, si le paramètre id est égalé à -1, le programme Java va générer des données de navigation pour 106 utilisateurs. Nous effectuons donc la seconde recommandation sur un panel de 106 utilisateurs.

Ce programme java génère trois fichiers de données :

- data.txt : ce fichier comporte le code Swift correspondant aux données tests pour qu'elles soient intégrées à l'application iOS,
- data.csv : ce fichier comporte les données tests pour qu'elles puissent être utilisées par le script R,
- item.csv : ce fichier comporte les données relatives aux sites web pour qu'elles puissent être utilisées par le script R.

5) Difficultés rencontrées et travaux futurs

Nous avons rencontré au cours de ce projet de nombreuses difficultés. Dans un premier temps, nous avons dû concevoir un programme Java suffisamment élaboré pour générer des données tests de profils utilisateur différents et suffisamment complexe. Nous aurions pu concevoir un programme Java encore plus complexe pour obtenir des données tests encore plus diverses mais nous pensons que cela ne nous aurait pas apporté beaucoup plus de choses.

La plus grande difficulté que nous avons rencontrée et que le langage Swift ne permet d'effectuer aisément des calculs matriciels comme le permet le langage R. Nous avons essayé d'implémenter les calculs matriciels nécessaires aux calculs de notre seconde recommandation mais nous n'avons pu eu le temps d'aboutir jusqu'au bout. Nous n'avons pas pu non plus surpasser notre difficulté dans l'intégration de R à iOS. Nous avons donc effectué les calculs de la seconde recommandation dans un script R en dehors de l'application iOS. L'un de nos futurs travaux consistera alors à intégrer les calculs de cette seconde recommandation à un ensemble de serveurs distants. L'application iOS devra périodiquement fournir les données de navigation des utilisateurs aux serveurs. Les serveurs calculeront alors en retour les résultats de la seconde recommandation pour chaque utilisateur, et les enverrons à chaque utilisateur correspondant. L'application iOS pourra ainsi recommander des nouveaux sites web aux utilisateurs.

Nous chercherons aussi à améliorer le design de l'interface de l'application iOS mais aussi ses fonctionnalités en y ajoutant la possibilité d'avoir plusieurs onglets web. Il est important de noter qu'il nous a été difficile de véritablement conclure sur les performances de nos deux types de recommandations. En effet, avec les travaux que nous avons menés, nous ne sommes pas en mesure de véritablement pouvoir conclure sur les performances de notre système. Pour cela, nous allons mener dans les prochaines semaines un test réel de minimum un mois sur un ensemble d'utilisateurs témoins. Pour récupérer efficacement les données de performances de nos recommandations, nous allons implémenter un processus de communication entre l'application et les serveurs. L'idée est de sauvegarder sur les serveurs la position des recommandations qui ont été sélectionné par les utilisateurs. Concernant la première recommandation, plus la proportion des recommandations sélectionnées se situe dans les toutes premières positions de la recommandation, plus la recommandation est performante. Pour la seconde recommandation, il faudra calculer le nombre de sites web recommandés avec leur position de recommandation qui ont été visités par les utilisateurs et qui ont été revisités plusieurs fois par la suite. Plus ce nombre sera important avec le temps passé lors des revisites, plus cette seconde recommandation

aura été efficace. Pour finir, nous chercherons à améliorer nos deux systèmes de recommandations en fonction des résultats de performance que nous aurons obtenus.

En résume dans les semaines à venir, nous allons :

- Intégrer aux serveurs les calculs de la seconde recommandation implémentés dans le script R,
- Implémenter un processus de communication périodique entre l'application iOS et les serveurs distants pour fournir les données de navigation des utilisateurs aux serveurs et fournir les résultats de la seconde recommandation aux utilisateurs,
- Ajouter à l'interface de l'application iOS la possibilité d'avoir plusieurs onglets web,
- Implémenter un processus de communication entre l'application iOS et les serveurs pour récupérer les performances de la première recommandation,
- Tester l'application sur un ensemble d'utilisateurs témoins pendant minimum un mois,
- Améliorer les recommandations suites aux résultats de performance des deux recommandations durant la session test.

Conclusion

En conclusion, ce projet nous a permis de mettre en pratique les différentes notions que nous avons acquises au cours de la session dans le cours LOG6308 – Systèmes de recommandations. Nous avons conçu et implémenté en partie un système complexe composé de deux types de recommandations. Il nous reste cependant encore du travail à effectuer pour compléter et tester plus en détails notre système.

Figure 1. Exemple d'une exécution de l'algorithme de recommandations du site web que l'utilisateur devrait souhaiter visiter en fonction du jour et de l'heure actuel (jour = lundi et heure = 10h)