Master 2 Ingénierie Informatique
Université Paris Denis-Diderot

Projet de

Fouilles de Données

EbookMining

Isabelle Richard
Lala Tiana Randriamparany
Victor Oudin

Introduction

Dans le cours de Datamining (*Fouilles de données*), nous devions réaliser un projet d'implémentation des connaissances du cours. Nous avons imaginé plusieurs applications possibles, recommandation d'images, ou de musiques, prévisions des résultats des élections municipales. Finalement, nous avons décidé de concevoir une application de recommandation de livres numériques, e-books.

Nous voulions, dans un premier temps, découvrir comment les fournisseurs de livres sur internet, proposaient leurs produits. Dans un second temps, nous désirions une application permettant de nous recommander des e-books parmi des livres techniques d'informatique (difficile de tous les lire, nous en avons beaucoup).

Une fois choisi ce que nous voulions faire, nous devions trouver « *comment ?* ». Plusieurs composantes du cours de Datamining s'offraient à nous, ce servir des utilisateurs ou du contenu des livres. Si nous nous aidions des utilisateurs, où pouvions nous trouver les données de leurs choix et appréciations, ou sinon comment pouvions nous régler le problème du « C*oldstart* ».

Si nous décidions d'utiliser le contenu des livres, quelles données et métadonnées pouvait-on en tirer ; à priori, le nom du fichier, l’auteur, le titre, le genre, et le texte. Mais quelle exactitude auront ces données, elles peuvent être absentes ou même fausses, à part le texte, toutes peuvent être erronés. Nous devions aussi prendre en compte la masse qu'elles pouvaient représenter, et la possibilité d'une persistance nécessaires de ces données. Il nous fallait aussi déterminer les moyens, méthodes et algorithmes, que nous emploierons, afin de réussir à recommander des livres.

Nous donnerons nos réponses a toutes ces interrogations au fil du rapport, nous aborderont les méthodes employées dans l'implémentation du projet, les algorithmes et libraires utilisés. Nous donnerons ensuite quelque résultats de différents cas d'utilisation de cette implémentations. Pour finir par les options auxquelles nous avons pensées, ainsi que les améliorations futures réalisables.

Méthodes

Nous avons choisi de voir notre projet de Datamining, comme un projet de fouilles de texte (fouilles de données dans un ensemble de textes), et donc de baser entièrement notre recommandation sur la substance principal d'un livre, son texte. Dû, à la grande probabilité de fausseté des métadonnées, nous les utilisons quasiment que pour des affichages. Maintenant, le choix le plus logique s'imposant à nous, était la recommandation par **TF-IDF** (de l'anglais *Term Frequency-Inverse Document Frequency*). Nous rappellerons ce qu'est le **TF-IDF** et la similarité entre vecteurs, puis nos choix d'implémentations.

La recommandation par TF-IDF.

Le **TF** est la fréquence d'un terme ; le nombre d'occurrence d'un mot dans un livre ($|\{m \in L | m=mot\}|$, L : l'ensemble des mots du livre) diviser par le nombre total de mots du livre ($|\{m \in L\}|$).

$$\mathbf{TF}= \frac{|\{m \in L | m=mot\}|}{|\{m \in L\}|}$$

L'**IDF** est la fréquence inverse d'un terme dans les documents ; le logarithme népérien de la division du nombre total de livres ($|\{l \in C\}|$, C : l'ensemble des livres du corpus) par le nombre de livres contenant ce mot ($|\{l \in C \mid m \in l \mid m=mot\}|$).

$$\mathbf{IDF}= ln\left(\frac{|\{l \in C\}|}{|\{l \in C \mid m \in l \mid m=mot\}|} \right)$$

On obtiens la valeur final en multipliant le TF et l'IDF. En appliquant ce calcul pour l'ensemble des mots de l'ensemble des livres du corpus, on obtiens des vecteurs de TF-IDF l'ensemble du corpus, nous permettant ainsi de les pondérer entre eux avec la distance cosinus.

Plus il y a de mots à traiter, plus les vecteurs pour les recommandations sont grands (plus de 80.000 dimensions), et donc longs à calculer. Nous avons donc décidé d'utiliser la similarité-cosinus plutôt que la distance euclidienne pour faire la recommandation, qui nous donne une valeur entre -1 et 1 à la place d'un vecteur de distances de plus de 80.000 dimensions lui aussi.

La **similarité cosinus** (ou **distance cosinus**) est l'angle entre deux vecteurs à n dimensions, est ce calcul ainsi :

$$\mathbf{similarité} = \cos(\theta) = \frac{A \cdot B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

La **recommandation** s'effectue grâce à une matrice de similarités (une matrice de taille corpus * corpus, dont la valeur dans une case [i][j] est la similarité ente le livre i et le livre j). Quand un utilisateur choisi un livre, on copie la ligne correspondant au livre dans la matrice, et on trie les similarités par ordre décroissant. Pour recommander à partir d'un ensemble de livre, quand un utilisateur sélectionne plusieurs livres, il suffit de faire la moyenne des lignes des livres de l'ensemble (ex : $E[0]=(l_1[0]+l_2[0]+...+l_k[0])/k$,$E[1]=(l_1[1]+l_2[1]+...+l_k[1])/k$,..., $E[n]=(l_1[n]+l_2[n]+...+l_k[n])/k$), et de trier cette ligne de moyenne par ordre décroissant de similarité.

Implémentations.

Nous aborderons ici les choix d'implémentations particuliers, et les étapes majeurs du déroulement de l'application. Le langage qui a été choisi est Python. Nous avons choisi d'utiliser ce langage, pour exploiter les nombreux modules disponibles pour le traitement de texte ou de statistiques donner en cours.

Analyse Syntaxique :

Les e-books utilisés sont au format PDF. La lecture d'un document se fait grâce au module Python pyPdf. Ce module permet de récupérer, sous forme de chaînes Unicode, les métadonnées du document, ainsi que son contenu. Ces données sont ensuite nettoyées grâce aux modules Python nltk et string. Dans un premier temps, les chiffres sont supprimés des chaînes, les lettres accentuées sont transformées en caractères non accentuées (exemple: é devient e) et les majuscules sont transformées en minuscules. Ensuite, tout caractère non alphanumérique est transformé en espace, à l'aide de la fonction sub() du module « re ». Enfin, la fonction split() est utilisée pour découper la chaîne en utilisant le caractère d'espacement comme séparateur, permettant ainsi de récupérer la liste des mots du texte. Le nombre de mots du document correspond au nombre d'éléments de cette liste. Le nombre de mots du document correspond au nombre d'éléments de cette liste. Un dictionnaire des occurrences de chaque mot dans le document est ensuite construit. Un filtre est appliqué à la liste avant la construction du dictionnaire : la méthode stopwords.words('french') fournie par le module nltk.corpus, permet d'éliminer les mots les plus courants de la langue française. Cette opération permet de ne pas encombrer la base de données avec des mots comptant peu dans le calcul des TF-IDF des documents. Une fois la nouvelle liste obtenue, le genre et le nombre de chaque mot est retiré avant de l'ajouter au dictionnaire, à l'aide de la méthode FrenchStemmer().stem() du module nltk.stem.snowball. Appliquer cette méthode réduit considérablement le nombre de mots dans la base de données. Enfin, le nombre d'occurrences de chaque mot est calculé et renvoyé sous forme de dictionnaire.

Stockage des données :

Le stockage des données consistera à sauvegarder les différentes informations nécessaires pour effectuer les calculs. Il permet aussi de fournir les fonctions de récupération des données.

Nous avons opté pour le langage SQL pour la gestion de la base de données. Pour ce faire, on a utilisé le module sqlite3 associé à python. Comme notre projet est basé initialement sur une utilisation locale, nous avons opté pour une base de données sous forme de fichier, et qui sera donc sauvegardée dans le répertoire d'exécution du programme.

Les requêtes SQL se chargeront de la création des tables, de l'insertion des données dans les tables, ainsi que la récupération des données sur des critères précis. La base de données est composée de trois tables :

- la table books pour sauvegarder les titres et les auteurs des livres,
- la table word pour le sauvegarde de tous les mots de tous les livres ainsi que pour chaque mot, le nombre de livres qui le contiennent,
- la table TF qui associe à chaque mot de chaque livre son TF.

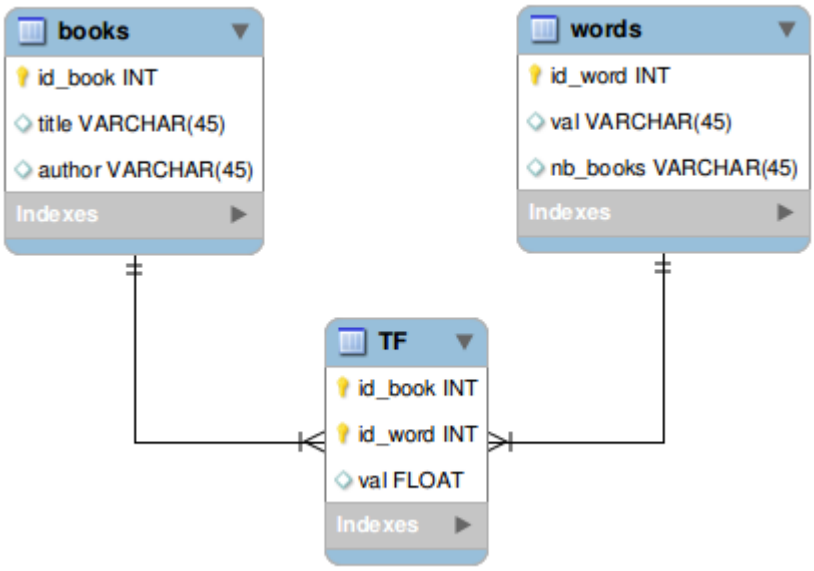

Illustration de la base de données

Déroulement :

Voici étapes par étapes, le déroulement de chaque parties importantes du projet.

Insertion d'un e-book:

1. parser l'e-book;
2. le nettoyer (enlever la ponctuation et autre);
3. compter le nombre de mots dans l'e-book;
4. enlever les stop-words (mais, ou, et, donc, or, ni, car, ...);
5. compter le nombre d'occurrence des mots;
6. calculer le TF des mots;
7. entrer le livre, ses mots et leur TF dans la base de données.

Vecteur de TF-IDF d'un e-book:

1. récupérer le nombre de livres dans la base;
2. calculer l'IDF de chaque mot de la base;
3. extraire les TF de l'e-book;
4. construire le vecteur de TF-IDF.

Matrice de similarité:

1. Pour chaque paire d'e-book calculer la similarité cosinus;
2. enregistrer le résultat dans la matrice symétrique.

Recommandation:

1. choix d'un ou plusieurs e-books par l'utilisateur;
2. copie de la ou des lignes de la matrice de similarité de le ou les e-books;
3. dans le cas de plusieurs e-books additionner les similarité, trier par ordre décroissant de similarité;
4. recommander par, du plus similaire au moins similaire.

Résultats

Voici quelque résultats de mesure de temps obtenus :

- L'analyse syntaxique et l'ajout de 100 livres à la base de données :----------
  environ 1heure
- Le calcul des vecteurs de TF-IDF de 300 livres :----------
  environ 20 secondes
- Le calcul de similarité entre un livre et tous les autres (300) :----------
  environ 5 secondes
- La fabrication de la matrice de similarités de 300 livres :----------
  environ 4 minutes
- La recommandation à partir du choix d'un livre :----------
  immédiat
- La recommandation à partir du choix d'un ensemble de livres :----------
  immédiat

Discussion

Options possibles :

- Feed-back :

Enregistrer un feed-back des utilisateurs pour vérifier le bon fonctionnement de la recommandation, en définissant une valeur de similarité barrière, à partir de laquelle les recommandations au-dessus de celle-ci, non-choisis par l'utilisateur, seront considéré comme des faux-positifs, ou celles en-dessous de celle-ci, choisi, des faux-négatifs.

- Langue :

Offrir la possibilité, lors de la création et de l'ajout à base de données, de choisir la langue du corpus de documents, en l'entrant en paramètre. Ensuite, ce paramètre, serait utilisé lors du stemming et de l'élimination des stopwords.

Améliorations possibles :

- Augmentation de la vitesse d'accès à la base :

Actuellement les écritures dans la base sont séquentielles, ralentissant l'ajout de livres. En rendant ces accès atomiques et asynchrones, en utilisant un SGBD (système de gestion de base de données), il serait alors possible de paralléliser les écritures et lectures, augmentant considérable l'ajout de livres à la base de données et le calcul des TF-IDF.

- Version Web :

Réaliser une version web, un site en php ou javascript, qui recommanderait uniquement dans le corpus en s'appuyant sur la matrice de similarité. La matrice serait donc mise à jour quotidiennement ou hebdomadairement, à l'ajout d'un jeu de livres. On pourrait pousser jusqu'au calcul des vecteurs de TF-IDF et de la matrice dans le site, pour permettre l'ajout en ligne.

- Version Client/Serveur :

Dans cette version, on mettrait la base de données sur un serveur, et l'accès passerai par une application serveur à l'aide d'une application client. Dans cette version, les clients partageraient la même base et demanderaient les similarités au serveur. Pour l'ajout de livre, les clients pourraient analyser les livres en locale, pour n'envoyer au serveur que les infos du livre à ajouter avec son dictionnaire de TF. Le serveur traiterait les demandes d'ajout, en ajoutant le livre et ses TF à la base, recalculerait la matrice similarité, et la diffuserait aux clients.

Conclusion

Grâce à ce projet de Datamining, nous avons pu utiliser concrètement les connaissances du cours, et découvrir un bref aperçus du monde de la recommandation. Nous n'avons pas implémenté le « *feed-back* » mais après plusieurs essais, nous avons trouvé les résultats obtenus cohérents et rationnel, ce-qui nous incite de continuer à nous servir de notre application dans des contextes différents (la recommandation de livres techniques d'informatique). Et pourquoi pas, apporter les options amenés précédemment.

Nous pourrions imaginez aussi une évolution vers de la reconnaissance de livre ou de genre de livre, à l'aide de clustering de la base de données.

Équipe

Le partage du travail s'est fait de la manière suivante :

- Isabelle Richard a géré l'analyse syntaxique des e-books.
- Lala Tiana Randriamparany a géré le stockage et les accès à la base de données.
- Victor Oudin a géré les algorithmiques de similarité et de recommandation.