BOISDON Hugues

DERVILLE Thomas

Rapport Projet Datamining

**But du projet** :

L’objectif du projet était de créer un système de recommandation de monuments historiques. A partir d’une base de données composés d’une centaine de monuments et à l’aide d’une application récupérant des images « likées » par l’utilisateur, le système a pour but de recommander d’autres monuments qui correspondent à ses préférences.

**Source des données et licences :**

Les données des monuments proviennent toutes de Wikidata et ont été obtenues à l’aide du service de requête intégré. Toutes les images sont licenciées sous le CC BY-SA 4.0.

Les données concernant le style architectural et le pays ont été obtenues via l’API WikiBaseIntegrator qui permet de récupérer les données liées à un identifiant WikiData sans avoir à repasser par une requête SPARQL.

**Taille des données :**

Les données sont stockées sous deux formats différents : un fichier data au format JSON faisant environ 80 Ko, un autre fichier JSON users d’environ 1Ko et un dossier images contenant les images liées aux monuments de la base de données, dossier faisant environ 276 Mo.

**Informations stockées pour chaque image :**

label : Contient le nom (en anglais afin d’avoir le moins de valeurs inconnues) du monument.

link : Contient le lien wikimedia vers l’image principale du monument (image qui apparaît sur la page wikipédia du monument en général)

entityID : Contient l’identifiant WikiData du monument. On l’utilise pour les requêtes WikiBaseIntegrator.

format : Contient le format de l’image principale (jpeg, png,…)

path : Contient le chemin relatif vers l’image contenu dans le dossier image stocké localement.

size : Contient la taille de l’image sous forme d’une liste à deux membres sous le format [ X, Y ].

tags : Contient une liste possédant les différents tags liés au monument entre autres le siècle, le pays et le/les styles architecturaux liées au monument. Principalement utilisé dans le système de recommandation afin de différencier les différents monuments.

date : Contient la date de construction du monument. On la récupère directement depuis la requête SPARQL afin de réaliser une première filtration en ne récupérant que des monuments possédant une date.

style : Contient le/les qualifier(s) WikiData des styles architecturaux possédés par le monument. Pour la même raison que pour date on récupère ces informations directement depuis la requête.

orientation : Contient l’orientation de l’image (paysage ou portrait). Est utilisé dans le système de visualisation.

artist : Contient le nom de l’auteur de la photo.

dateTaken : Contient la date à laquelle la photo a été prise. Est utilisé dans le système de visualisation.

appareil : Contient le nom de l’appareil avec lequel la photo a été prise.

dColors : Contient les couleurs prédominantes d’une image. Utilisé pour l’algorithme k-means.

dProportions : Contient les proportions des couleurs prédominantes. Utilisé pour l’algorithme k-means.

**Informations concernant les préférences de l’utilisateur :**

Les informations concernant l’utilisateurs sont stockées dans un fichier json nommé users. On y stocke le nom de l’utilisateur, ses likes et ses dislikes sous forme d’indices. Par défaut tous les indices sont stockés dans les dislikes et lorsque l’utilisateur like une image on « déplace » son indice dans la table like.

**Visualisation des données :**

Les données qu’on choisit de visualiser sont celles sur l’année à laquelle la photo a été prise ainsi que l’orientation de la photo. On pourrait améliorer le système en visualisant la fréquence des différents tags voire les types de tags (année, pays, style) en ajoutant ces critères.

**Exploration de données et Apprentissage machine :**

Pour la partie apprentissage machine on utilise la librairie sklearn afin d’entraîner l’algorithme de recommandation en fonction des différentes images likées par l’utilisateur.

Pour la partie exploration de données on utilise divers moyens de récupérer les données et les analyser. Tout d’abord les requêtes SPARQL permettent de récupérer un grand nombre d’objets à partir de certains critères, ici l’objet est un monument avec une image, une date de construction/d’ouverture et au moins un style architectural.

On utilise aussi WikiBaseIntegrator qui permet d’accéder à toutes les informations d’un objet lié à un qualifier WikiData, cela permet de choisir ce que l’on veut récupérer plus précisément qu’avec une requête SPARQL (un seul item au lieu de balayer la base de données entière de WikiData), on peut alors facilement récupérer le pays d’un monument sans avoir à repasser par une requête SPARQL.

**Auto-évaluation :**

Annotation des données : Du fait de la requête SPARQL initiale, on a déjà un premier filtre qui permet de « sauter » quelques vérifications qui seraient normalement nécessaires comme le type des données reçues (int, str, …) avant de les utiliser dans les diverses fonctions du système. Un exemple notable est un monument n’ayant pas de date de construction précise donc il possède bien la propriété P571 mais la valeur liée est inconnue. Le monument passe alors le filtre indirect de la requête SPARQL ce qui induit des erreurs dans le programme sans les vérifications sur la présence d’une clé « year » dans le JSON contenant les données récupérées.

On part donc du principe que l’on connaît le type des données contenus dans la dataframe sont connues et vérifiées à l’avance.

Système de recommandation : On pourrait améliorer le système en prenant en compte les différents tags ainsi que les couleurs des bâtiments à l’aide des k-means en plus des tests effectués par la librairie sklearn. De plus le système n’est pas entièrement fonctionnel au niveau de l’affichage de la recommandation.

Application : Différents onglets pourraient être rajoutés en plus de ceux déjà existants notamment des statistiques sur les images likées par l’utilisateur (lien avec la visualisation de données). De plus différentes fonctionnalités de gestion des utilisateurs pourraient être implémentés comme la suppression d’utilisateur, l’ajout de mot de passe pour se connecter, ….

**Séances pratiques, exercices et pistes d’améliorations :**

Les premiers TPs sont un peu « simples » dans le sens où le code est déjà fourni dans le code Jupyter ce qui ne pousse pas à tester du code autre que celui fourni en cherchant à résoudre les questions. Il suffirait de supprimer les lignes de codes répondant aux questions afin que nous puissions chercher à résoudre les questions par nous-mêmes avant de voir les réponses.

Aussi nous n’avons pas eu tant de temps que ça pour travailler sur le projet pendant les séances de TP qui étaient surtout monopolisées par la compréhension et l’application du cours. Le gros du travail devait être réalisé en autonomie mais nous n’avions pas beaucoup de temps pour poser des questions aux intervenants qui étaient très monopolisés à la fin du module.

**Conclusion :**

Ce projet était très intéressant puisqu’il nous permettait de réaliser notre premier projet de Datamining mais aussi de système de recommandation avec. Ainsi nous avons pour voir tout le chemin que pouvaient parcourir les données entre leur source (base de données ouvertes sur internet type WikiData) et leur origine (l’utilisateur) et comment elles étaient récupérées, traités, analysées puis triées avant d’être proposées à l’utilisateur en fonction de ses besoins.

Notre seul regret est le manque de temps qui ne nous a pas permis de réaliser toutes les idées que nous avions pour le système de recommandation en lui-même.