Cheaper Elsewhere

Hamza Jeddad
Mickaël Lagattu
Mohammed Souheil Lassouad
Noubair Matah

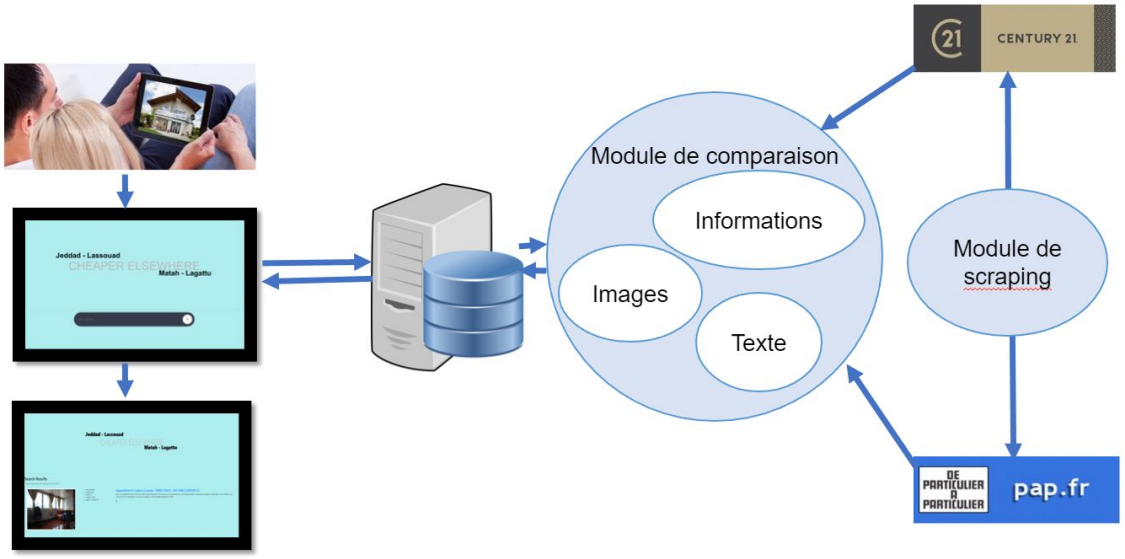
Sommaire

I - Introduction

L'achat d'un bien immobilier n'est pas un acte à prendre à la légère. Pour les ménages qui décident de devenir propriétaires, c'est un engagement déterminant qui durera plusieurs années, voire décennies. Avec la multiplication des sites de mise en relation de particuliers, en plus de la multitude d'agences immobilières bien établies, les recherches de la maison ou de l'appartement parfait se révèlent particulièrement fastidieuses. Il n'est pas rare qu'un même bien immobilier se retrouve mis en vente sur plusieurs sites à des prix différents.

II - Schéma d'utilisation

Le fonctionnement et l'utilisation du site est résumé dans le schéma suivant :



Les utilisateurs qui ont repéré une annonce immobilière qui les intéresse peuvent aller sur la page d'accueil du site et entrer leur lien. Le site interroge la base de données pour en extraire des annonces pour le même bien qui auraient été enregistrées au préalable. Le site affiche ensuite les résultats.
D'un autre côté, tous les jours à minuit, un module de scraping se déclenche et va collecter les annonces présentes sur les sites immobiliers. Ces annonces sont envoyés vers un module de comparaisons qui va utiliser trois critères (Informations extraites, similarités de texte et comparaison d'images) pour déterminer les annonces relatives au même bien immobilier. Les annonces récoltées et leurs liens sont enregistrés dans la base de données.

III - Description technique des différentes parties

1) Comparaison des images

La classe GlobalComparator fait appel à la classe “imageComparator” et “detection” pour calculer une partie du score global de similitude d’une annonce avec la requête.

.La classe "SSIM" : (Structural Similarity) indice de mesure de similarité entre deux images. Cet indice est peu sensible au changement de qualité par compression où à l’ajout d’un logo. Le constructeur prend en argument deux noms d’images (ex: “image1.jpg” et “image2.jpg”). La méthode SSIM(image1,image2).compare_images() retourne le couple (SSIM, MSE) avec MSE = Mean Square Error, qui est l’erreur associé à la mesure de similitude entre les deux images.

.La classe "histogram" : mesure moyenne des couleurs présentes dans l'image. L'avantage de cette mesure est qu'elle donne une corrélation quand deux images d'un seul lieu sont prises de deux angles différents. Le constructeur prend en argument deux noms d’images (ex: “image1.jpg” et “image2.jpg”). La méthode histogram(image1, image2).correlation() retourne une corrélation entre 0 et 1.

.La classe "detection" : définie dans un fichier à part, elle fait appel à la librairie ImageAI (https://github.com/OlafenwaMoses/ImageAI) et retourne une liste des objets contenus dans l'image. Le constructeur prend en argument un nom d’une image (ex : “image.jpge”). La méthode detection(image).getObjects() retourne une liste de dictionnaires, chacun contenant l’objet détecté, 4 coordonnées du rectangle contenant cet objet dans l’image et un pourcentage de probabilité. Exemple d’output:
[{'percentage_probability': 62.83443570137024, 'name': 'vase', 'box_points': array([187, 219, 211, 255])}, {'percentage_probability': 85.70202589035034, 'name': 'tv', 'box_points': array([219, 142, 385, 268])}]

Script de test et exemples d’utilisation:

```
"test SSIM"
print("\nTest SSIM")
image1 = "with_logo.jpg"
```

```
image2 = "without_logo.jpg"
print("(SSIM, MSE) : ", SSIM(image1,
image2).compare_images())


# print(SSIM(image1, image2))
if SSIM(image1, image2).compare_images()[0] > 0.9:
    print("Exactly the same images, possibly with a logo
superposed : ", image1, "and", image2)

print("")

"Histogram test"
print("\nTest Histogramme")
files = []
for e in os.listdir(data_path):
    if '.jpg' in e: files.append(e)
near_histo=dict()
for i in range(len(files)):
    near_histo[files[i]]=set()
    others=[e for e in files if e!=files[i]]
    for x in others:
        if histogram(files[i],x).correlation()>=0.8:
            near_histo[files[i]].add(x)
print("Images with similar histograms : ",near_histo)

print("")
image="century.jpg"
print("Objets :",detection(image).getObjects())
```

2) Comparaisons textuelles :

Les annonces des biens immobiliers sont , dans la plupart des sites, accompagnées d'une description textuelle. Dans celle-ci, l'annonceur essaye de mettre en valeur son bien, en ajoutant des informations supplémentaires (station de métro/ supermarchés les proches

...) par rapport aux celles de base (surface, nombre de pièces ...) et/ou en reformulant ces dernières. Vu que les agences aident les particuliers qui veulent vendre leurs biens à rédiger cette descriptions, une description pour le même bien peut varier d'un bien à l'autre ce qui rendre la comparaison textuelle une tâche compliquée.

La comparaison de textes peut se faire de plusieurs méthodes :

2.1 . Coefficient de Jaccard :

Jacc(texte1, texte2) = (# des mots différents en commun)/(# total des mots différents dans les deux textes = card(set(texte1) ∩ set(texte1)) / card(set(texte1) U set(texte1))

Avantages :

- Ne nécessite aucun training en amont.

Inconvénients :

- Ne prend pas en considération les poids des mots dans phrases
- Ne prend pas en considération les similarités sémantiques entre deux mots différents
- Ne prend pas l'ordre en considération

2.1 . Sac de mots (Bag of word) :

Chaque texte est représenté comme un vecteur sparse dont la i-ème composante correspond aux nombre d'occurrences du i-ème mot (par rapport au vocabulaire) dans le texte. On peut choisir la similarité du cosinus comme une similarité entre les deux textes :

Cos_sim(vec1,vec2) = vec1.vec2/(||vec1|| ||vec2||)

Avantages :

- Prend les occurrences en considération.

Inconvénients :

- Ne prend pas en considération les similarités sémantiques entre deux mots différents
- Ne prend pas l'ordre en considération

2.1 . Word2Vec :

Les méthodes citées ci-dessus, ne prennent pas en considération le sens des mots utilisés dans la description. Utiliser un modèle pré-entrainé de représentation paraît utile pour cet effet. Chaque mot du vocabulaire a une représentation dans un espace vectoriel de dimension finie. On représente chaque phrase par une moyenne des dits vecteurs pondérée des tf-idf des mots et on calcule la similarité du cosinus à la fin.

3) Scrapping

Dans le MVP mis en place, nous avons décidé de tester notre idée sur deux site d'annonces immobilières (PAP et Century21) et donc par conséquent nous avons mis en place deux modules de scrapping.

Les deux modules fonctionnent,à quelque détails près,de la même manière.

Les URLs des pages de chaque annonce ne présentent pas une logique ou une régularité qui permette de deviner l'URL de l'annonce d'un autre bien.

En revanche, les pages affichant plusieurs annonces présentent quant à elle une structure assez simple dans leur URL permettant de choisir le numéro de la page souhaitée.

Ainsi l'opération de scrapping peut être divisée en quatre étapes :

1. On scrapp les liens des différents annonces sur la première page affichant les annonces
2. Pour chaque lien d'annonce, on scrapp la page de l'annonce correspondante et on extrait les informations voulues.
3. On incrémente l'URL de la page d'annonce pour passer à la suivante et on reproduit le même process.

Les requêtes HTTP pour obtenir le code source de chaque page sont faites à l'aide de la fonction get du module requests. Un web agent a été ajouté pour diminuer les chances de se faire bloquer par le site (surtout pour PAP). Par ailleurs, nous avons mis en place un temps d'attente aléatoire entre les différentes requêtes pour maximiser encore une fois les chances de réaliser le scrapp sans se faire bloquer.

Parser le code source et en extraire les informations voulues a été réalisé grâce au module BeautifulSoup.

Pour chaque annonce, le résultat du scrapp est une liste contenant plusieurs informations : le prix de vente, la surface du bien(totale et habitable),le type d'appartement,l'arrondissement, nombre de pièces, le texte de l'annonce, les adresses

locales des images du bien (elles sont téléchargées pendant le scrapp) et finalement l'identifiant de l'annonce

4) Serveur et front

Cette partie sert principalement de colonne vertébrale au programme, en permettant d'orchestrer les tâches qui assurent les fonctionnalités de notre programme. Ainsi, dans le fichier __init__.py, on retrouve la fonction qui est exécutée au lancement du serveur. Cette fonction initialise les différents modules nécessaires (Flask, mongoDB, Bootstrap), définit les url des pages qui seront accessibles aux utilisateurs et lance le module de Scraping dans un thread à part, via la classe Scrapper dans le fichier scrapp_launcher.py.

Au niveau des technologies utilisées, nous avons choisi Flask pour gérer l'aspect serveur. Plus léger et facile d'utilisation que Django, il était plus adapté au projet. La base de données en MongoDB nous permettait de mieux gérer les liens de similarité entre les annonces que les bases de données relationnelles. Enfin, après avoir parlé de React pour réaliser la front, nous avons préféré conserver un framework plus classique avec HTML, CSS et Bootstrap, pour ne pas perdre trop de temps à maîtriser une technologie qui ne constitue pas le coeur du site.

Pour générer les pages, nous avons opté pour un design de type « factory », vu dans le cours de java de 2^{e} année. Ce design permet de créer une instance d'une classe PageFactory à laquelle on demandera des instances des classes des pages en question. Les classes représentant les pages sont dans le fichier pages.py. On y définit trois pages : la page d'accueil, sur laquelle on demande un lien, la page de résultats, et une page d'erreur, qui n'est finalement plus utilisée dans la version finale du programme, car on a préféré renvoyer une page sans résultats plutôt qu'une page d'erreur en cas de lien erroné.

Les pages elles-mêmes, c'est-à-dire ce qui sera affiché à l'écran de l'utilisateur, est fait de HTML et CSS en utilisant Bootstrap, en utilisant l'API Jinja2 qui permet de générer relativement facilement le contenu à l'intérieur des pages. Les codes bootstrap utilisés ont été principalement pris sur le site bootsnipp.com, qui propose justement des exemples de code Bootstrap faits pour être intégrés facilement aux sites.

Lorsque l'utilisateur rentre un lien pour une agence, ce lien doit être traité pour en extraire l'identifiant qui correspond à l'annonce pointée par celui-ci. Ce traitement se fait dans la classe LinkProcess. Une fois l'extraction de l'identifiant effectuée, on recherche cette annonce dans la base de données, et on en retire les annonces enregistrées comme similaires.