Université de technologie de
Belfort-Montbéliard

Projet DS51

Web sémantique, ontologies et extraction des connaissances

Réalisé par :

CHAUSSON Thibault

VIGUIER Léo

Encadré par : CHARIETE Abderrahim

8 juin 2023

Résumé

Dans le cadre de notre formation d'ingénieur, nous devons réaliser l'exploitation de données sur WikiData, mais aussi créer une ontologie avec ces données.

En outre, nous avons eu la charge de réaliser un modèle permettant de reconnaître des animaux provenant du dataset ImageNet[1].

Table des matières

1 Présentation

1.1 Contexte du projet

Nous avons réalisé ce projet dans le cadre de l'unité de valeur DS51 à l'UTBM, qui est une UV du bloc métier Data Science qui dote les étudiants de deux crédits en filière. Pour ce faire, nous sommes constitués en groupe de deux étudiants en informatique.

Le projet se déroule durant la fin du semestre et est décomposé en un unique jalon. Le rendu final est constitué d'un rapport, d'une présentation et de l'ensemble des codes sources.

1.2 Les objectifs du projet

L'usage de l'apprentissage supervisé en Machine Learning offre aux ordinateurs, entre autres, la capacité d'apprendre à identifier automatiquement des objets grâce à des données étiquetées. Les résultats des algorithmes les plus avancés, comme les réseaux de neurones profonds, sont stupéfiants.

Cependant, ces algorithmes demandent un grand nombre d'exemples pour leur phase d'apprentissage. En outre, lorsqu'ils sont employés pour étiqueter de nouvelles données, ils ne peuvent pas expliquer le raisonnement derrière leur décision. Ils peuvent cependant fournir un score qui représente la probabilité que leur décision soit correcte, ou indiquer la portion de la donnée qui a influencé leur décision.

Dans ce projet, nous supposons que les ontologies peuvent en partie surmonter ces deux obstacles scientifiques.

2 Récupération des données

2.1 Importation des données

Comme demandé dans le sujet, nous avons récupéré les données sur le site de la compétition ImageNet Object Localization Challenge de Kaggle. Nous avons ensuite extrait les données qui nous intéressent, à savoir les fichiers avec les identifiants suivant :

- n02114367 (loup)
- n01484850 (requin)
- n01614925 (aigle)
- n02133161 (ours)
- n01537544 (passerin indigo)
- n01443537 (poisson rouge)

2.2 Récupération des noms d'animaux

2.2.1 Récupération des identifiants des animaux sur Wikidata

Pour ce faire, nous allons récupérer la liste des différents animaux existant sur Terre grace à WikiData avec une request SPARQL :

```
SELECT DISTINCT ?animal ?animalLabel WHERE {
  ?animal wdt:P31 wd:Q16521.
  ?animal rdfs:label ?animalLabel filter (lang(?
animalLabel) = "en").
  #OPTIONAL {?animal wdt:P18 ?image.}
} LIMIT 1000000
```

Code source 1 – Requête SPARQL pour récupérer les animaux

Nous téléchargeons le fichier CSV pour les 1000000 premiers résultats et nous le sauvegardons dans le fichier *animals.csv*, pour pouvoir l'exploiter plus tard.

2.2.2 Récupération des identifiants des animaux sur Kaggle

Sur Kaggle, nous récupérons le fichier LOC_synset_mapping.txt, qui contient la liste des identifiants des animaux avec leur nom.

La correspondance entre les 1000 synsets id et leurs descriptions. Par exemple, la ligne 1 indique n01440764 tanche, Tinca tinca signifie qu'il s'agit de la classe 1, que l'identifiant synset est n01440764 et qu'elle contient le poisson tanche.

```
1 n01440764 tench, Tinca tinca
2 n01443537 goldfish, Carassius auratus
```

2.3 Récupération des données depuis une base de données : WikiData

Pour ce faire, nous utiliserons la librairie *wikibaseintegrator*[3] qui permet d'interroger la base de données de WikiData. Nous voulons récupérer l'ensemble des animaux de la base de données, pour ce faire, nous utiliserons : *WikibaseIntegrator* wbi.item.get(entity_id='Q729').

2.4 Récupération des sous-classes des animaux

Nous allons maintenant récupérer les sous-classes des animaux, nous avons choisi de les récupérer sur une profondeur de 2 niveaux. Pour cela, nous allons utiliser la requête SPARQL suivante :

```
1       SELECT ?animal ?animalLabel ?subclass ?subclassLabel ?
    subclass2 ?subclass2Label
2       WHERE
3       {
4         VALUES ?animal {wd:%s}
5         ?animal wdt:P279 ?subclass .
6         OPTIONAL { ?subclass wdt:P279 ?subclass2 . }
7
8         SERVICE wikibase:label { bd:serviceParam wikibase:
    language "en". }
9       }
```

Code source 2 – Requête SPARQL pour récupérer les sous-classes

wd :%s correspond à l'identifiant de l'animal dont nous voulons déterminer les sous-classes.

Nous avons récupéré la propriété *subclass of*, au lieu de *instant of*, car cette deuxième n'était pas toujours présente dans la liste des animaux fournis, contrairement à la première.

L'utilisation de la bibliothèque *wikibaseintegrator* pour interroger la base de données de WikiData, nous permet de ne pas nous faire bloquer par l'API WikiData si nous faisons trop de requêtes.

3 Classification et explication des données

3.1 Classification des données

Nous allons maintenant utiliser les données de la base de données Image Net qui contient des images de différents animaux, pour faire de la classification et de l'explication.

Pour réaliser cette tâche, nous allons utiliser un ensemble d'images issues de la base de données Image Net. Nous allons utiliser les images de 6 classes différentes. Pour chaque classe, nous avons 99% images d'entraînement et 1% images de test. Les classes sont les suivantes :

— n02114367 (loup)
— n01484850 (requin)
— n01614925 (aigle)
— n02133161 (ours)
— n01537544 (passerin indigo)
— n01443537 (poisson rouge)

Nous utiliserons les images d'entraînement pour entraîner un modèle de classification et les images de test pour évaluer la performance du modèle. Nous allons ensuite utiliser les images de test pour expliquer les prédictions du modèle.

Nous diviserons notre travail différentes parties :

1. Récupérer les données sur Kaggle et sélectionner les images des classes qui nous intéressent (https ://www.kaggle.com/c/imagenet-object-localization-challenge)
2. Traiter les données, cela implique de les diviser en classe égale et de les redimensionner, pour ce faire, nous pourrons utiliser des bibliothèques comme OpenCV ou Pillow
3. Diviser les données en données d'entraînement et données de test
4. Le choix du modèle de classification : nous utiliserons des modèles couramment utilisés pour la classification d'images incluant les réseaux de neurones convolutionnels (CNN), qui ont obtenu de bons résultats dans de nombreux domaines
5. Entraîner un modèle de classification, pour ce faire, nous pourrons utiliser des bibliothèques comme Keras ou PyTorch ou TensorFlow
6. Évaluer la performance du modèle sur les données de test, et obtenir l'accuracy du modèle

7. Utiliser les images de test pour expliquer les prédictions du modèle, pour ce faire, nous pourrons utiliser des bibliothèques comme LIME ou SHAP

3.2 Explication des données pour entrainer un modèle

Les images subissent des transformations lors du chargement, afin de les rendre plus facilement 'interprétables' par le modèle. Ces transformations sont les suivantes :

— Redimensionnement de l'image à 220 x 172 pixels. Cela conserve les proportions de l'image, et permet de réduire le nombre de pixels à traiter, ce qui réduit la taille du modèle et le temps d'entraînement
— Normalisation des valeurs des pixels entre 0 et 1
— Transformation en tenseur

3.3 Choix du modèle pour la classification

Le modèle pour classfier les images est un réseau neuronal convolutionnel. Il est assez commun dans le milieu de la classification d'images d'utiliser un tel modèle pour résoudre cette tâche pour de nombreuses raisons.

Tout d'abord, les réseaux neuronaux convolutionnels (CNN en anglais) sont conçus pour extraire des caractéristiques pertinentes à partir d'images. Contrairement aux réseaux neuronaux traditionnels, les CNN tirent parti de la structure spatiale des données d'entrée, en les traitant comme des matrices de pixels, plutôt que de les considérer comme des vecteurs plats. Cette approche permet aux CNN de capturer des motifs locaux tels que des bords, des textures et des formes, ce qui est crucial pour la classification d'images.

En utilisant des couches de convolution, les CNN sont capables de détecter des caractéristiques à différents niveaux d'abstraction. Les premières couches apprennent généralement des filtres simples, tels que des lignes ou des points, tandis que les couches plus profondes peuvent capturer des motifs complexes et des structures hiérarchiques. Cette capacité à apprendre des caractéristiques discriminantes à plusieurs niveaux permet aux CNN de comprendre des images de manière progressive, en capturant à la fois des détails fins et des relations globales.

De plus, les CNN sont équipés de couches de pooling qui réduisent la dimensionnalité des caractéristiques extraites. Cela permet une représentation plus compacte des informations tout en préservant les aspects les plus

saillants des images. La réduction de la dimensionnalité facilite le traitement des données par les couches ultérieures du réseau et améliore l'efficacité du modèle en termes de temps de calcul et de consommation de mémoire.

Après 3 couches de CNN, un « dropout » est effectué. Cette opération supprime une certaine proportion des neurones du modèle, ce qui à pour effet de ralentir l'over-fitting du modèle sur les données d'entraînement.

Enfin, 2 couches linéaires permettent de faire converger le tenseur en entrée vers une des 6 classes que compose cet exercice. Ce modèle ressemble à celui de cette vidéo.

4 Création d'une ontologie

4.1 Création de l'ontologie en Python

Pour ce faire, nous utiliserons le package owlready2 qui permet de créer des ontologies en Python. Les données seront extraites de la base de données de WikiData, nous interrogerons cette base de données par l'API mis à disposition par la librairie wikibaseintegrator.

Dans les étapes précédentes nous avons récupéré le nom des animaux avec les différentes sous-classes qui leurs sont liées. Nous nous sommes rapidement rendu compte que des classes sont en commun avec d'autres animaux. Par exemple, nous avons un animal qui a pour sous-classe « éléphant » et « mammifère » un autre a pour sous-classe « éléphant d'Asie » et « éléphant », donc nous devons créer une classe éléphant d'Asie qui découle d'éléphant qui elle-même découle de mammifère.

De plus, si nous devons ajouter des animaux nous pouvons être confronté au fait qu'un animal n'a pas de classe, donc nous créons une classe générique *SansFamille*.

Pour générer l'ontologie mous suiverons ces étapes :

1. Création d'une IRI
2. Création de la classe *Animalia(Thing)*
3. Création des sous-classes qui découlent de *Animalia* et des sous-classes précédement créées
4. Définition de la classe *Image* avec son nom et si elle est reconnue ou non
5. Définir les exclusions (la notion d'exclusion n'existe pas à proprement parlé dans owlready2)
6. Définition des propriété : *Contient*, ... et des *CaracteristiqueMorphologique* avec par exemple, *plume*, ...
7. Ajout des animaux
8. Ajout des images liées à un animal et si oui ou non l'image est reconnue
9. Enregistrement de l'ontologie

4.2 Visualisation de l'ontologie avec Protégé

Pour comprendre plus finement l'ontologie créée par Python nous utiliserons le logiciel Protégé[2] développé par Stanford.

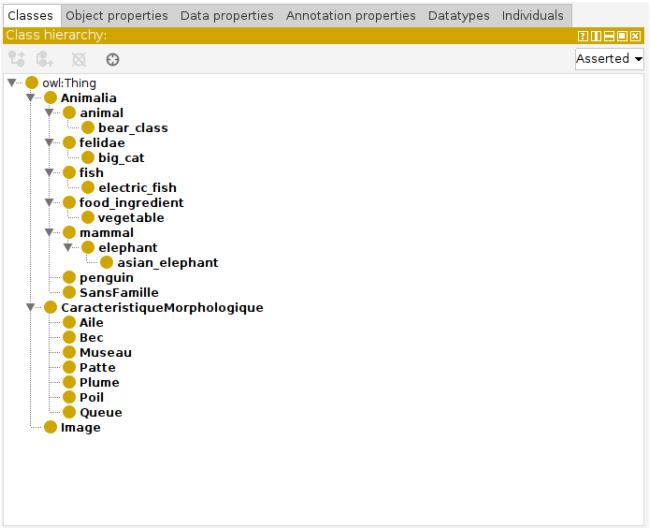

Image 1 – Classe de l'ontologie

Dans la figure 1, nous remarquons l'ensemble des classes créées par notre code source en fonction des informations récupérées sur WikiData, par exemple la classe *big_cat* hérite de *fellidae*, qui hérite elle-même de *Animalla*.

De plus, nous remarquons que l'ensemble des classes sont en minuscule et sans espace, ceci pour simplifier la création de ces-dernières.

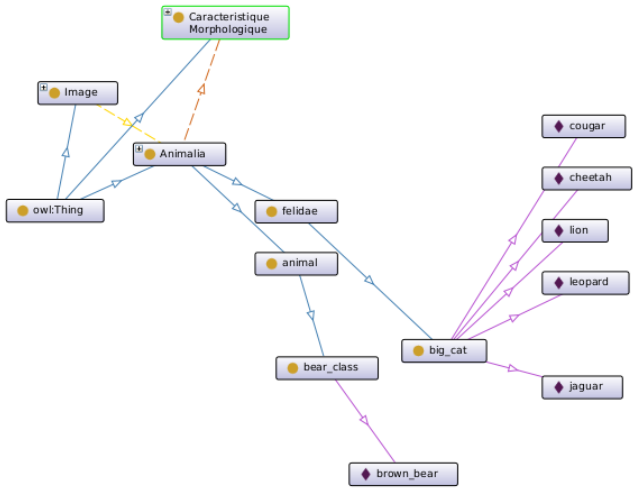

Image 2 – Graphe de l'ontologie

Sur la figure 2, nous visualisons l'ensemble des liens issues des relations entre les animaux. Nous en déduisons que les cougars et les lions doivent se ressembler car ils font partie de la même classe.

```
1       <big_cat rdf:about="#lion">
2         <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
        NamedIndividual"/>
3       </big_cat>
```

Code source 3 – Information d'un animal dans l'ontologie

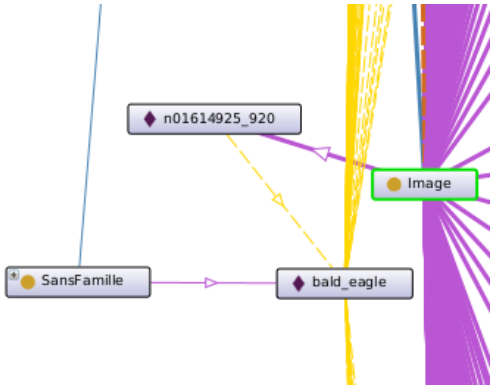

Image 3 – Image de l'ontologie

Il nous à aussi été demandé de montrer la relation entre les images et les animaux, sur la figure 3, il y a la classe *image* qui contient *n01614925_920* qui est relié à *bald_eagle*, et ainsi nous savons que cette image est un Pygargue à tête blanche qui a été ou non reconnu par notre modèle. De plus nous précisons divers information comme l'URL de l'image, son format, ...

```
1       <Image rdf:about="#n01443537_8719">
2         <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
        NamedIndividual"/>
3         <Contient rdf:resource="#goldfish"/>
4         <Reconnue rdf:datatype="http://www.w3.org/2001/XMLSchema
        #boolean">false</Reconnue>
5         <URL rdf:datatype="http://www.w3.org/2001/XMLSchema#
        string">./ILSVRC/Annotations/CLS-LOC/train/n01443537/
        n01443537_8719.xml</URL>
6         <Width rdf:datatype="http://www.w3.org/2001/XMLSchema#
```

```
      string">500</Width>
7       <Height rdf:datatype="http://www.w3.org/2001/XMLSchema#
      string">376</Height>
8       <Depth rdf:datatype="http://www.w3.org/2001/XMLSchema#
      string">3</Depth>
9     </Image>
```

Code source 4 – Information d'une image dans l'ontologie

5 Conclusion

Pour conclure, il est intéressant d'utiliser l'ensemble de ces données pour réaliser des modèles de reconnaissance d'images en utilisant des données accessibles librement sur le Web.

Cette UV nous a permis de mieux comprendre le fonctionnement des ontologie et de la récupération de données grâce à des requêtes SPARQL entre autres. Outre cet aspect, nous avons pu approfondir le sujet passionnant des réseaux neuronaux convolutifs (CNN).

Table des images

Liste des codes sources

Références

[1] *ImageNet Object Localization Challenge*. https://www.kaggle.com/c/imagenet-object-localization-challenge consulté le 15 mai 2023.

[2] *Protégé*. https://protege.stanford.edu consulté le 10 avril 2023.

[3] *wikibaseintegrator*. https://pypi.org/project/wikibaseintegrator/0.11.2/ consulté le 15 mai 2023.