HAI824 -TP2

RAMBAL JULIEN , M1-IASD

March 12, 2023

1 Introduction :

Dans ce travail il nous était demandé de réaliser un modèle capable de prédire un lien de type owl:sameAs entre deux entités de graphes de conaissance

1.1 Contexte :

Une base de données se raffine au cours de sa vie , il est donc impératif d'avoir un modèle capable de redécouvrir des relations à partir d'une donnée déjà traité. Il est également primordial de convenir d'un format de stockage le plus universel possible pour encourager des potentiels contributeurs à utiliser notre dataset

1.2 Sujet du travail :

Le dataset est l'ontologie DOREMUS proposée par Manel Achichi, Pasquale Lisena, Konstantin Todorov et al en Octobre 2018 .
Les relations préexistantes et à déduire sont disponibles sur doremus-playground , les fichiers sont présentés sous le format ttl et se nomme : target + source et refDHT.

1.3 Plan :

Nous divisons ce TP en 3 étapes que sont le pre-processing des données , créations du dataset et l'apprentissage

1.3.1 Pre-processing :

Dans cette partie nous transformerons le fichier rdf en dataframe exploitable et nous profiterons de cette transformation pour comprendre les spécificités du dataset.

Les outils utilisés seront les modules python : rdfLib , ntlk et transformers .

1.3.2 Modèle :

Fort de la connaissance que nous avons apprises à l'étape précédente nous choisirons un graphe hétérogène créer à partir de la librairie DGL .

1.3.3 Apprentissage :

Finalement nous évaluerons les performances du modèle et débattrons sur les possibilités d'évolution .

2 Pre-processing

La première observation est que les données sont hétérogènes , nous avons réfléchi à 3 options pour répondre à cette spécificité :

- Utilisé le Graphe RDF fournis
- Utilisé un graphe homogène mais dont les nœuds et les relations sont associés à l'embedding de leur type
- Utilisé un graphe hétérogène avec embedding

L'avantage de la première solution est qu'elle nous épargnera des lourdeurs dans la phase de pre-processing l'inconvénient est que les ressources que nous avons trouvé sur le sujet sont peu documentées ou datées et la communauté à laquelle nous pourrions avoir accès est réduite .

Le graphe homogène traité avec GCN quant à lui demande des ressources de calculs conséquentes surtout si le graphe contient beaucoup de type de nœuds et de relations différentes . L'avantage est que de nombreux outil actuellement développé existe et que le sujet est "tendance" .

Finalement le graphe hétérogène demande moins de temps de calculs (on s'épargne l'embedding des nœuds parents et des relations mais l'on conserve l'embedding des noeuds "valeurs") mais cette solution est plus spécifique est donc moins documenté que le graphe homogène .

2.1 RDF :

Nous avons tout d'abord développer la première option , l'idée proposée a été de calculer l'embedding du noeud à partir de son voisinage notre expertise nous contraignait à ne faire qu'un embedding sur le 1-voisinage

2.1.1 Observation :

L'analyse nous a dans un premier temps conforté dans un 1-voisinnage , en effet pour les 2 graphes du dataset le degré moyen d'un noeud parent est de 3.15 (en comptant la relation type) et son écart-type est de 1.4 . Cela dit parmi ces voisins en moyenne la moitié (1.5) sont un autre noeud parent .
Finalement on conclu qu'un noeud typique du graphe est de la forme :

id_noeud ; type ; $proba(id) = 0.75 | proba(valeur) = 0.25$

Par conséquent un modèle qui se repose sur le 1-voisinnage est insuffisant il faut donc un k voisinnage

2.1.2 Conclusion :

L'approche RDF implique de trouver un k satisfaisant mais aussi une pondération des relations/edges du graphes cohérent avec le dataset .

Nous avons investigués plusieurs outils pour répondre à cette problématique , principalement dgl-ke cela dit de nombreux pépins techniques nous ont poussé à considérer sont grand frère dgl .

2.2 DGL Homogène :

Le format attendu pour les modèles DGL est très divers , l'outil nous étant nouveau nous avons choisi une approche conventionnelle et donc faire la transformation $RDF \rightarrow csv$ manuellement. Nous avons donc partagé notre dataset en triplet (head;relation;tail) pour ensuite constituer les noeuds et edges du graph .

Les noeuds parents auront pour features un vecteur qui représente l'encodage BERT de la premières phrase de la description de leur type tandis que les noeuds valeurs auront faute de mieux pour feature l'encodage de : leur type (s'ils en ont un) et leur valeur en string .

Les relations quant à elle auront elles aussi un vecteur qui représente leur type à partir de l'encodage de sa description .

Finalement nous obtiendront un ensemble de triplet (id1;id_relation;id2) et un ensemble de triplet (id;id_type;feature)

2.2.1 Observation :

Sur un total de 5602 noeuds (source : 2597 , target : 3005) a cette instant du TP nous estimons qu'au moins 229 noeuds parents ne possèdent aucune relation mise à part "type" ce qui représente 4% du dataset.L'intégralité de ces noeuds parents vides sont des noeuds "E21_Person"

Tous les types du dataset ne sont pas renseignés sur les fichiers erlang ou doremus que nous avons pré-téléchargé , par exemple P165_incorporates .
Deplus de nombreuses descriptions tout comme les labels sont manquants .En complément nous notons aussi que certains noeuds sont issus de "http://data.doremus.org/vocabulary" et plus généralement sont décrit dans vocabularies .

Nous notons également des orthographes différentes pour dénommer la même chose (cf. Scope Note) .

Concernant les valeurs des noeuds de valeurs nous notons qu'ils sont constitués d'un tableau d'objet json :

$$@id|@lang|@type|@value$$

Ce tableau représente les différentes dénominations d'une même entité . nous avons conclu que réduire ce tableau à une seule valeur serait bénéfique [1]

2.2.2 Solution mis en oeuvre :

Nous allons si la description et label sont manquants prendre le nom comme description , en cas de multiples valeurs nous avons choisi de conservé la valeur du tableau qui est la plus similaire à l'ensemble des autres valeurs du tableau grâce à l'embedding BERT multi langue .

2.2.3 Conclusion :

L'encodage des noeuds est assez long bien qu'il soit réduit à cause d'un faible nombre de type d'entité distincte et par le pré-téléchargement de la majorité des types/relations .

[1] habituellement nous souhaitons avoir une augmentation de la donnée pour faire apprendre le modèle mais ici comme nous le verrons il faut diminuer autant que possible la taille des features de chaque noeuds

3 Dataset :

Nous adjoignons au Dataset obtenu par la phase de pre-processing précédente le dataset décris dans le fichier "refDHT.rdf" qui est un ensemble d'éléments tous constitué par une paire d'entités et une relation binaire . Nous allons comme précédemment le transformé en un ensemble de triplet symbolisant des edges (id1;id_relation;id2)

4 Apprentissage :

4.1 Graphe Hétérogène :

Nous avons en entrée 10675 edges dont 238 viennent de refDHT et représentes des relations d'égalités sémantiques , nous avons également un total de 9386 noeuds dont 4013 sont des noeuds de valeurs .

Le premier résultat de l'apprentissage avec un nombre limité de features (uniquement sur les noeuds de valeurs) a été très décevants . La loss ne s'améliorait pas et l'accuracy était à 17% .

Nous avons par la suite compris que le type de chaque noeuds devait impérativement être pris en compte mais les résultats restés assez médiocres : une loss constante (environ 0.88) et l'accuracy 60%

Pour la suite nous prévoyons de faire entrer dans le dataset de nouveau graph ou de nouvelle relation d'égalité car nous en avons seulement 238 ce qui est peu compte tenu des 10K edges du graphe . Mais aussi est surtout de nous attarder davantage sur le fonctionnement interne de DGL car les améliorations possibles impliquent une compréhension approfondie des GCNN .

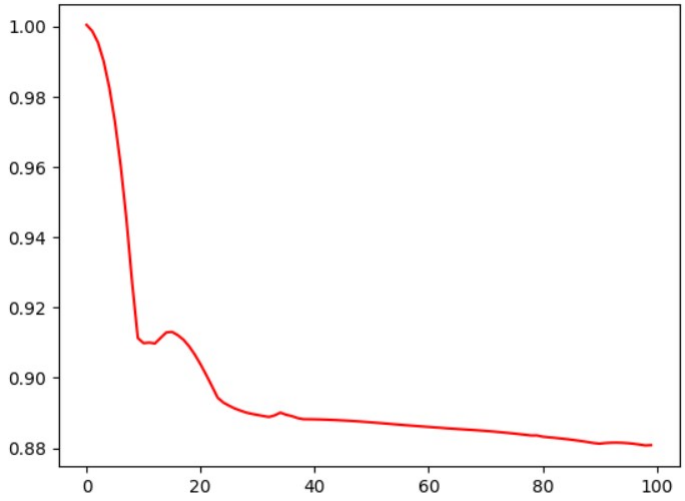

Figure 1: Loss par epoch

5 Conclusion :

En conclusion bien que les résultats soient décevants de nombreuses possibilités nous ont été ouvertes par notre introduction a de multiples outils tels que DGL mais aussi la librairie transformers de huggingface que nous avons découvert sur ce TP .
La manipulation de csv très volumineux et du langage python pour son traitement nous a permis de cerner de nouvelles problématique et d'établir une méthodologie pour y répondre .
Finalement le volé statistique très succin sur les modèles de machine learning nous a donné l'envie de nous y consacrer plus amplement pour le prochain TP .