GNN - Node Classification

Bio-inspired machine learning

Y. Abida, B. Vadurel
M2-IA - 20/10/2024
Université Lyon 1 Claude Bernard

1. Problème posé

Dans ce rapport, nous nous intéressons au problème de classification des nœuds dans un graphe. Le jeu de données que nous utilisons contient des informations sur les aéroports dans le monde, chaque nœud possède la longitude/latitude, la population locale, le pays et la ville de l'aéroport. Les arêtes représentent les aéroports qui sont reliés par un trajet.

L'objectif sera de cacher l'information du pays dans chacun des nœuds et de le prédire.

2. Preprocessing des données

2.1. Encoding des variables textuelles

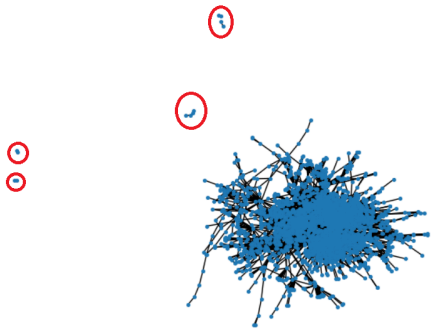
Dans le graphe, chaque nœud possède deux données textuelles : city_name et country, afin de pouvoir les exploiter nous appliquons un label encoding dessus, c'est-à-dire on associe à chacune des valeurs possibles.

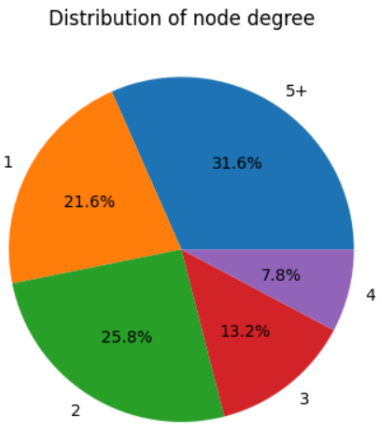
2.2. Masquage de la donnée population

Nous observons la présence de données aberrantes dans la donnée population, comme cela rajoute du bruit dans le modèle, nous préférons la retirer des nœuds avant d'apprendre dessus.

2.3. Filtrage des noeuds isolés

En visualisant le graphe, on observe la présence de nœuds isolés du reste.





En effet, plusieurs nœuds ont un degré faible, comme nos modèles utilisent de la convolution cela peut nuire à leur apprentissage, nous décidons donc de retirer les nœuds qui ont un degré inférieur à 3 pour nos prédictions.

2.4. Z-score

Le Z-score est une mesure statistique qui calcule la distance d'une donnée par rapport à la moyenne en fonction des écarts-types. En analysant les Z-scores, nous pouvons identifier les valeurs qui s'écartent le plus de la distribution normale.

En appliquant un filtrage pour éliminer les nœuds isolés, le nombre de nœuds passent de 3363 nœuds à 1769 nœuds. Ensuite, nous avons appliqué le filtrage avec le calcul du Z-score, en conservant les nœuds avec un Z-score inférieur à 3. Les nœuds avec un Z-score supérieur à 3 ont été considérés comme des *outliers* (données aberrantes) et on été supprimés. Ce qui réduit donc le nombre de nœuds à 1734.

2.5. Split du jeu de données

Une fois le graphe converti en jeu données, nous le séparons en 3 : un jeu d'entraînement, de validation et de test en suivant la répartition 0.8/0.1/0.1.

3. Optimisations

3.1. Early Stopping

Afin d'éviter le sur-apprentissage sur le jeu de données, nous avons mis en place l'early stopping pour nos GNN.

3.2. Optimisation bayésienne

Nous avons implémenté l'optimisation bayésienne pour explorer des plages d'hyper paramètres de manière adaptative. Nous avons donc cherché les meilleurs hyper paramètres suivants :

- Learning rate de l'optimiseur (Adam)
- Weight decay de l'optimiseur
- Nombre de neurones par couches

4. Architectures utilisées

4.1. Graph Convolutional Network (GCN)

Dans un premier temps, nous avons testé le GCN avec une seule couche de convolution comme base de nos tests, puis nous avons fait évoluer le modèle en y ajoutant de nouvelles couches pour finalement obtenir le modèle suivant :

- 2 couches de Convolution (GCNConv)
- 2 couches linéaires fully connected
- Du pooling entre chaque convolution
- ReLU comme fonction d'activation
- Adam comme Optimizer

Nous observons que le tuning d'hyper paramètres ne permet pas d'avoir d'améliorations significatives sur les performances du modèle.

4.2. GraphSage

Nous avons aussi testé GraphSage avec l'architecture suivante :

- 2 couche de Convolution (SAGEConv)
- 3 couches linéaires fully connected
- ReLU comme fonction d'activation
- Adam comme Optimizer

4.3. Graph Attention Network (GAT)

4.3.1. Les différents modèles

Le modèle GAT introduit un mécanisme d'attention qui permet à chaque nœud d'attribuer des poids différents à ses voisins. Ce mécanisme d'attention est particulièrement utile pour les graphes où les nœuds peuvent avoir des relations hétérogènes.

Modèle avec un seul head :

- Architecture :
  - 2 ou 3 couches de convolution attentionnée (GATConv)
  - Fonctions d'activation ELU après chaque couche
  - Couche finale avec un softmax logarithmique pour la classification
  - Dropout appliqué à chaque étape pour régulariser et éviter le surapprentissage

Modèle avec plusieurs heads :

- Architecture :
  - 2 couches de convolution attentionnée (GATConv) avec plusieurs heads concaténés
  - Des couches linéaires fully connected
  - Fonctions d'activation ELU après chaque couche
  - Couche finale avec un softmax logarithmique pour la classification
  - Dropout appliqué à chaque étape pour régulariser et éviter le surapprentissage

4.3.2. Performance du modèle

Après un premier entraînement initial sur nos modèles GAT à un seul head, l'accuracy ne dépassait pas 20%.

Nous avons entraîné les model avec 300 neurones par layers et un dropout de 0.3
Accuracy : 0.18, Precision : 0.035, RMSE : 114

4.3.3. Optimisation

4.3.3.1. Optimisation Bayésienne

Nous avons appliqué l'optimisation bayésienne sur nos modèles. En plus des hyperparamètres classiques, nous avons ajouté le taux de *dropout* et le nombre de *heads* pour les modèles GAT multi-head, afin d'optimiser davantage la performance.
Les meilleurs hyperparamètres trouvés pour le modèle **GAT_Forward1** sont :

Hyperparamètre	Valeur
lr	0.0010312531082429088
wd	2.7069760917328855e-05
dropout	0.0015058484884345191
nb_neurons	305

4.3.3.2. Observation sur les optimisations

Après avoir effectué plusieurs essais sur notre meilleur modèle avec les meilleurs hyperparamètres sur nos données non filtrés, nous avons réussi à atteindre de bonnes accuracy, variant entre 70% et 90%, avec une moyenne de 73.9%.

En utilisant ces mêmes hyperparamètres sur des données filtrées, nous avons également eu des bonnes accuracy, variant entre 67% et 77%, avec une moyenne de 74.1%.
Ces variations d'accuracy sont dues à l'initialisation aléatoire des poids des modèles.

5. Comparatif des modèles

Dans la page ci-après, vous trouverez un comparatif des modèles que l'on a pu testé. Le premier tableau contient les scores pour les modèles avec le filtrage des données et le second contient les scores obtenus sans le filtrage des données. Les méthodes que l'on a employés pour le filtrage des données sont décrites dans les parties 2.2, 2.3 et 2.4.

	Accuracy	**Precision**	**RMSE**
GCN_Course	0.405	0.354	73.685
GCN	0.610	0.566	61.65
GraphSage	0.189	0.036	100.580
GAT_Forward1	0.754	0.735	46.780
GAT_Forward2	0.677	0.655	49.444
GAT_Forward3	0.659	0.633	52.790
GAT_Forward4	0.580	0.583	53.523
GAT_Multihead1	0.713	0.720	54,367
GAT_Multihead2	0.742	0.741	50.839

Tableau 1 : Score obtenus pour chaque modèle avec filtrage des données

	Accuracy	**Precision**	**RMSE**
GCN	0.181	0.032	115.724
GAT_Forward1	**0.811**	**0.800**	**43.59**
GAT_Forward2	0.6777	0.655	49.444
GAT_Forward3	0.563	0.548	72.596
GAT_Forward4	0.522	0.512	69.127
GAT_Multihead1	0.751	0.7655	49.971
GAT_Multihead2	0.726	0.728	52.986

Tableau 2 : Score obtenus pour chaque modèle sans filtrage des données

Parmi tous les modèles testés, celui qui obtient les meilleures performances est le GAT_Forward1 sans le filtrage des données. GAT_Forward1 a pour particularité de posséder 3 couches de GATConv, pour le reste, il suit l'architecture décrite en partie 4.3.1.

6. Ablation study

Pour l'ablatation, on va donc s'intéresser à notre meilleur modèle : GAT_Forward1. On va donc se proposer d'enlever chacune des couches du modèle une à une et observer l'impact que ça a sur les performances de celui-ci.

Couche supprimée	Accuracy	Precision	RMSE
1 GATConv	0.636	0.615	61.371
2 GATConv	0.439	0.336	70.951
ELU	0.268	0.102	97.627
SoftMax	0.7419	0.7322	55.345
Dropout	0.598	0.6144	64.129

On observe une baisse de performance du modèle pour chaque couche retirée, tout particulièrement pour la fonction d'activation ELU. On peut donc en conclure que chacune des couches du réseau sont importantes pour lui permettre de faire de bonnes prédictions.