GNN: Link Prediction

Rida Asri

October 22, 2024

1 Introduction

Notre étude se concentre sur la prédiction de liens dans un réseau complexe de villes et d'aéroports.La tâche de prédiction de liens consiste à estimer la probabilité d'existence d'un lien entre deux nœuds donnés dans un réseau, en se basant sur un ensemble de données. Dans notre contexte, ces nœuds représentent des villes, et l'ensemble des données représente les noms des ville , les pays, les populations ... .

Les étapes de prédiction sont décrites ci-dessous :

- Un encodeur crée des embeddings de nœuds en traitant le graphe avec deux couches (ou plus) de convolution.
- Nous ajoutons aléatoirement des liens négatifs au graphe original. Cela rend la tâche du modèle une classification binaire avec les liens positifs provenant des arêtes d'origine et les liens négatifs provenant des arêtes ajoutées.
- Un décodeur fait des prédictions de liens (c'est-à-dire des classifications binaires) sur toutes les arêtes, y compris les liens négatifs, en utilisant des embeddings de nœuds. Il calcule un produit scalaire des embeddings des nœuds à partir de chaque paire de nœuds sur chaque arête. Ensuite, il agrège les valeurs à travers la dimension d'incorporation et crée une seule valeur sur chaque arête qui représente la probabilité de l'existence de l'arête.

2 Methodology

Pour aborder ce problème, nous avons exploré trois architectures de réseaux de neurones graphiques (GNN) modernes :

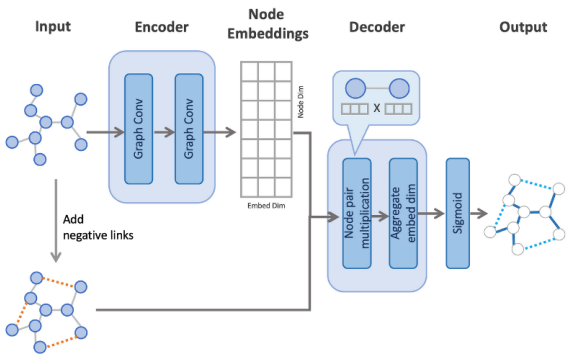

Figure 1: Architecture

2.1 Graph Convolutional Networks (GCN)

Dans ce projet, j'ai utilisé trois modèles de GCN :

2.1.1 Modèle GCN_1

Le modèle GCN_1 est une architecture simple composée de deux couches de convolution. Dans cette architecture, les caractéristiques d'entrée des nœuds sont représentées par `in_channels`. La première couche de convolution, `conv1`, réduit ces dimensions à `hidden_channels` et applique une activation ReLU. La sortie de cette première couche est ensuite passée à la deuxième couche de convolution, `conv2`, qui transforme les dimensions en `out_channels`.

La méthode `encode` prend en entrée les caractéristiques des nœuds et les indices des arêtes, les passant à travers les deux couches de convolution. Les méthodes `decode` et `decode_all` sont utilisées pour calculer la similarité entre les nœuds. La méthode `decode` utilise le produit scalaire des embeddings pour des paires de nœuds spécifiques, tandis que `decode_all` génère une matrice de similarité pour tous les nœuds.

2.1.2 Modèle GCN_2

Le modèle GCN_2 est plus complexe, avec quatre couches de convolution. Comme dans GCN_1, la première couche `conv1` réduit les dimensions des caractéristiques d'entrée à `hidden_channels`. Ensuite, trois couches supplémentaires (`conv2`, `conv3`, `conv4`) transforment les dimensions à travers `hidden_channels2`, `hidden_channels3`, et enfin `out_channels`.

Entre chaque couche de convolution, un mécanisme de régularisation par dropout est appliqué avec un taux de 50%, ce qui aide à prévenir le sur-apprentissage en désactivant aléatoirement certaines caractéristiques lors de l'entraînement. La méthode `encode` passe les caractéristiques à travers les quatre couches de convolution, avec activation ReLU et dropout entre chaque couche. Les méthodes de décodage sont similaires à celles de GCN_1.

2.1.3 Modèle GCN_3

Le modèle GCN_3 présente une architecture qui utilise trois couches de convolution. La première couche, `conv1`, réduit les caractéristiques d'entrée à `hidden_channels`, suivie par `conv2`, qui produit `hidden_channels2`, et enfin `conv3`, qui produit les sorties à `out_channels`.

Un taux de dropout plus élevé de 80% est appliqué après les deux premières couches pour renforcer la régularisation. La méthode `encode` utilise les mêmes principes que les autres modèles, avec ReLU et dropout entre les couches. Les méthodes de décodage sont identiques à celles des modèles précédents.

2.2 Graph Attention Networks (GAT)

Dans ce projet, j'ai utilisé trois modèles de GAT :

2.2.1 Modèle GAT_1

Le modèle GAT_1 est une architecture simple composée de deux couches de convolution. Dans cette architecture, les caractéristiques d'entrée des nœuds sont représentées par `in_channels`. La première couche de convolution, `conv1`, utilise le mécanisme d'attention de type graphe (GAT) pour réduire les dimensions des caractéristiques d'entrée à `hidden_channels` avec un nombre de têtes défini par `heads`. Après cette opération, une activation ELU est appliquée, suivie d'un dropout avec un taux de 30%. La sortie de `conv1` est ensuite passée à la deuxième couche de convolution, `conv2`, qui transforme les dimensions en `out_channels`.

2.2.2 Modèle GAT_2

Le modèle GAT_2 est plus complexe, avec trois couches de convolution. Comme dans GAT_1, la première couche `conv1` réduit les dimensions des caractéristiques d'entrée à `hidden_channels`. Ensuite, une deuxième couche `conv2` transforme ces dimensions vers `hidden_channels2` avec le même mécanisme d'attention, suivie par la troisième couche `conv3`, qui produit les sorties à `out_channels`.

Entre chaque couche de convolution, un mécanisme de régularisation par dropout est appliqué avec un taux de 50%. Les activations ELU sont également utilisées après chaque couche. La méthode `encode` passe les caractéristiques à travers les trois couches de convolution. Les méthodes de décodage sont identiques à celles de GAT_1.

2.2.3 Modèle GAT_3

Le modèle GAT_3 présente une architecture qui utilise quatre couches de convolution. La première couche, `conv1`, réduit les caractéristiques d'entrée à `hidden_channels`, suivie par `conv2`, qui produit `hidden_channels2`, et `conv3`, qui produit `hidden_channels3`. La quatrième couche, `conv4`, transforme les caractéristiques en `out_channels`.

Comme dans GAT_2, un taux de dropout de 50% est appliqué après chaque couche. Les activations ELU sont utilisées entre les couches pour introduire de la non-linéarité. La méthode `encode` suit les mêmes principes que les autres modèles, et les méthodes de décodage sont identiques à celles des modèles précédents.

2.3 Variational Graph Autoencoders (VGAE)

Dans ce projet, j'ai utilisé deux modèles d'encodeurs basés sur des Graph Convolutional Networks (GCN) pour un autoencodeur variationnel graphique (VGAE) :

2.3.1 Modèle Encoder1

Le modèle `Encoder1` est une architecture simple composée de deux couches de convolution. Dans cette architecture, les caractéristiques d'entrée des nœuds sont représentées par `in_channels`. La première couche de convolution, `conv1`, utilise le mécanisme GCN pour transformer les dimensions d'entrée à $2 \times$ `out_channels`. Ensuite, deux couches de convolution, `conv_mu` et `conv_logstd`, produisent respectivement les moyennes et les log-écarts types des embeddings, tous deux de dimension `out_channels`.

La méthode `forward` prend en entrée les caractéristiques des nœuds et les indices des arêtes, et utilise la première couche de convolution suivie d'une activation ReLU. Elle renvoie les moyennes et les log-écarts types des embeddings en utilisant les deux dernières couches de convolution.

2.3.2 Modèle Encoder2

Le modèle `Encoder2` est plus complexe, avec trois couches de convolution. Comme dans `Encoder1`, la première couche `conv1` transforme les dimensions d'entrée à `hidden_channels`. Ensuite, la seconde couche `conv2` applique également le même mécanisme GCN pour transformer les caractéristiques, suivie d'une troisième couche `conv3` qui produit $2 \times$ `out_channels`.

Après ces trois couches, les couches `conv_mu` et `conv_logstd` génèrent respectivement les moyennes et les log-écarts types des embeddings, tous deux de dimension `out_channels`. Les activations ReLU sont appliquées après chaque couche de convolution.

2.3.3 Modèles VGAE

Les modèles `model1` et `model2` sont des instances de VGAE utilisant respectivement `Encoder1` et `Encoder2`. Le premier modèle, `model1`, est construit avec un encodeur d'entrée de dimension 5 et une sortie de dimension 2. Le second modèle, `model2`, utilise un encodeur d'entrée de dimension 5, un nombre de canaux cachés de 64, et une sortie de dimension 2.

3 Prédiction de liens entre les villes

Dans cette section, nous décrivons plus en détail les étapes de prédiction pour la tâche de prédiction de liens entre les villes. Nous avons utilisé un split de 10% pour les nœuds négatifs dans la première étape et un split de 20% dans la deuxième étape afin de comparer les performances des modèles.

3.1 Préparation des données

- **Normalisation des caractéristiques :** Les caractéristiques telles que la population et les coordonnées géographiques sont normalisées.
- **Encodage des caractéristiques catégorielles :** Les caractéristiques catégorielles telles que le pays et le nom de la ville sont encodées.

3.2 Division des données

- **Séparation des liens existants :** Les liens existants sont séparés en ensembles d'entraînement, de validation et de test.
- **Génération de paires négatives :** Des paires négatives (liens non existants) sont générées pour équilibrer le jeu de données. Dans la première étape, nous avons utilisé un split de 10% pour les nœuds négatifs, et dans la deuxième étape, un split de 20%.

3.3 Entraînement du modèle

- Des époques d'entraînement multiples sont effectuées, en optimisant le modèle sur les données d'entraînement et en évaluant ses performances sur un ensemble de validation à chaque époque. Pendant ce processus, des métriques telles que la perte, l'AUC (Area Under the Curve) et la précision sont collectées.

3.4 Test et évaluation

- Le modèle est évalué sur l'ensemble de validation après chaque époque.
- Les nœuds du graphe sont encodés, les liens sont prédits, et deux métriques d'évaluation sont calculées : l'AUC et l'ACC (Average Precision Score).

3.5 Itération et amélioration

- Afin d'évaluer la robustesse de nos modèles, la division des données (d'entraînement, de validation et de test) est changée à chaque itération.
- Nous avons expérimenté différents nombres de couches pour chaque architecture, incluant des modèles peu profonds (qui capturent principalement les interactions locales) et des modèles plus profonds (visant à capturer des relations à plus longue distance).

4 Etude comparative

4.1 Modèle GCN

4.2 Comparaison des modèles GCN avec une séparation des données de 10 %

1Pour cette analyse, les données ont été divisées en ensembles d'entraînement, de validation et de test, en générant des paires de nœuds négatifs. Pour la première étape de la séparation des données, 10 % des nœuds négatifs ont été utilisés, ce qui a permis d'équilibrer le jeu de données et d'améliorer la robustesse de l'évaluation des modèles.

D'après les résultats, le modèle `GCN_1` se distingue comme le meilleur parmi les trois modèles évalués, affichant la valeur d'AUC la plus élevée à 0.943 et une précision (ACC) remarquable de 0.994. Cela indique que `GCN_1` a une très bonne capacité à discriminer entre les classes positives et négatives, ainsi qu'une forte précision dans ses prédictions.

Le `GCN_2` suit avec une AUC de 0.926 et une ACC de 0.992, montrant également des performances solides, mais légèrement inférieures à celles de `GCN_1`.

En revanche, le GCN_3 présente les performances les plus faibles avec une AUC de 0.920 et une ACC de 0.940. Cela suggère que ce modèle est moins efficace pour la discrimination entre les classes et a une précision inférieure par rapport aux deux autres modèles.

4.3 Modèle GAT

4.4 Comparaison des modèles GAT avec une séparation des données de 10 %

2 GAT_1 se distingue avec une valeur d'AUC de 0.895 et une précision (ACC) de 0.988. Cela indique que ce modèle possède une excellente capacité à discriminer entre les classes positives et négatives, ainsi qu'une forte précision dans ses prédictions.

GAT_2 suit de près avec une AUC de 0.889 et une ACC de 0.987. Bien que ses performances soient légèrement inférieures à celles de GAT_1, il montre également une bonne robustesse.

En revanche, GAT_3 présente les performances les plus faibles avec une AUC de 0.878 et une ACC de 0.985. Cela suggère que ce modèle est moins efficace pour la discrimination entre les classes, avec une précision inférieure à celle des deux autres modèles.

4.5 Modèle VGAE

4.6 Comparaison des modèles VGAE avec une séparation des données de 10 %

3 Le modèle **VGAE_2** se distingue avec une valeur d'AUC de **0.820** et une précision (ACC) de **0.968**. Cela indique que ce modèle possède une bonne capacité à discriminer entre les classes positives et négatives, ainsi qu'une forte précision dans ses prédictions.

En revanche, le modèle **VGAE_1** présente des performances légèrement inférieures, avec une AUC de **0.809** et une ACC de **0.965**. Bien que ses résultats soient solides, ils sont inférieurs à ceux de VGAE_2, ce qui suggère que VGAE_1 est un peu moins efficace pour la discrimination entre les classes.

4.7 Comparaison des modèles VGAE et GCN et GAT avec une séparation des données de 20 %

4.7.1 Performance des GCN

4 Les modèles GCN affichent des valeurs AUC variant de 0.929 à 0.935, avec une précision constante de 0.993. Leur moyenne pour l'AUC est de 0.931, ce qui indique une très bonne performance globale.

4.7.2 Performance des GAT

Les modèles GAT montrent des valeurs AUC allant de 0.889 à 0.911, avec une précision légèrement inférieure, oscillant entre 0.986 et 0.990. La moyenne des AUC pour les modèles GAT est de 0.897, ce qui est inférieur à celle des GCN.

4.7.3 Performance des VGAE

Les modèles VGAE ont les performances les plus faibles, avec des valeurs AUC allant de 0.860 à 0.875 et une précision moyenne de 0.977. La moyenne AUC est de 0.868, indiquant un besoin d'amélioration par rapport aux autres modèles.

4.7.4 Conclusion

Les modèles GCN dominent clairement en termes de performances d'AUC et d'ACC, tandis que les modèles GAT offrent des résultats intermédiaires. Les VGAE, bien qu'utiles dans certains contextes, montrent une performance globale inférieure à celle des GCN et GAT.

Table 1: Résultats de l'AUC et de l'ACC pour chaque modèle avec une séparation des données de 10 %

Modèle	AUC	ACC
GCN_1	0.943	0.994
GCN_2	0.926	0.992
GCN_3	0.920	0.94

Table 2: Résultats de l'AUC et de l'ACC pour chaque modèle avec une séparation des données de 10 %

Modèle	AUC	ACC
GAT_1	0.895	0.988
GAT_2	0.889	0.987
GAT_3	0.878	0.985

Table 3: Résultats de l'AUC et de l'ACC pour chaque modèle avec une séparation des données de 10 %

Modèle	AUC	ACC
$VGAE_1$	0.809	0.965
$VGAE_2$	0.820	0.968

Table 4: Résultats de l'AUC et de l'ACC pour chaque modèle avec une séparation des données de 20 %

Modèle	AUC	ACC
GCN_1	0.935	0.993
GCN_2	0.929	0.993
GCN_3	0.931	0.993
GAT_1	0.911	0.990
GAT_2	0.892	0.987
GAT_3	0.889	0.986
$VGAE_1$	0.875	0.978
$VGAE_2$	0.860	0.976

4.8 Analyse Comparative des Modèles avec Différents Splits

Dans cette section, nous comparons les performances des modèles de GCN, GAT et VGAE avec des splits de 10% et 20%.

4.8.1 Performance des GCN

Avec un split de 20%, les modèles GCN affichent des valeurs AUC variant de 0.929 à 0.935, avec une précision constante de 0.993. Leur moyenne pour l'AUC est de 0.931, indiquant une très bonne performance globale. En comparaison, avec un split de 10%, les modèles GCN ont montré une moyenne AUC plus élevée, ce qui suggère qu'une plus grande quantité de données d'entraînement peut améliorer les performances des GCN.

4.8.2 Performance des GAT

Les modèles GAT montrent des valeurs AUC allant de 0.889 à 0.911 avec un split de 20%, tandis qu'avec un split de 10%, la moyenne des AUC pour les modèles GAT était de 0.895. Bien que les performances restent intermédiaires, il est clair que le passage à un split de 20% a conduit à une légère baisse de l'AUC.

4.8.3 Performance des VGAE

Les modèles VGAE affichent des performances les plus faibles, avec des valeurs AUC allant de 0.860 à 0.875 pour un split de 20%. Pour un split de 10%, la moyenne AUC des VGAE était de 0.872. Cela montre que l'augmentation du split a entraîné une légère dégradation des performances des VGAE, indiquant un besoin d'amélioration par rapport aux autres modèles.

4.8.4 Conclusion

En conclusion, les modèles GCN continuent de dominer en termes de performances d'AUC et d'ACC avec un split de 20%, mais il semble qu'une augmentation de la taille du split peut affecter négativement les performances des modèles GAT et VGAE. Les résultats suggèrent que l'optimisation des ratios de split est cruciale pour obtenir les meilleures performances possibles.