社群網路與推薦系統作業二

F74092269 資訊系-陳冠廷

一、 報告摘要

這次的作業內容為 link prediction 任務，給定三種不同的資料，我們要去預測每個資料可能會存在的邊。因為此次作業的執行方式是透過競賽的方式執行，所以作業變得非常有趣，也讓我有更大的動力去找尋新的演算法來提高最後的表現。

實驗方法我嘗試了一般機器學習和深度學習的方法，不出意外地，讓機器自己學習特徵（也就是深度學習的方式）有非常好的表現。用 Node2Vec 加上 Logistic Regression 只能到達 80%左右的表現；然而使用圖卷積（Graph Convolution Network）學習特徵並結合內積運算可以到達 90%的表現，但是根據不一樣的資料集表現會有較大的落差；最後我引入了 Co-embedding Attributed Networks，利用圖卷積和高斯模型假設，同時學習重建圖與特徵，最後在測試資料集中能達到更好的表現（除了資料集 3 以外），而且在不同資料集的表現沒有過大的差異。

經過多次實驗的結果，資料集 1、資料集 2 使用 CAN 能達到比較好的效果，而資料集 3 使用 GCN 可以得到較好的表現，最後成功在 Public Leader Board 達到第 17 名及本課程中的第 4 名，同時在 Private Leader Board 達到第 16 名和本課程中的第 4 名。

二、 實作步驟與實驗方法

（一） Link Prediction 任務說明

Link prediction 是希望去預測未來圖上是否會存在邊的關係，或者是否有某個邊遺漏在圖上。舉例來說，前者像是推薦系統中的朋友推薦，後者像是知識圖譜上的建立或是關係推測。而這次作業的內容並沒有具體的任務情境，特徵也沒有欄位說明（匿名特徵），所以我是用朋友推薦的概念去想這個任務，當然這並不影響此次作業的實作。

一般的 Link prediction 分成兩種類型：inductive 和 transductive [2]。前者是指訓練、驗證與測試資料為不同的圖，不屬於這次作業的

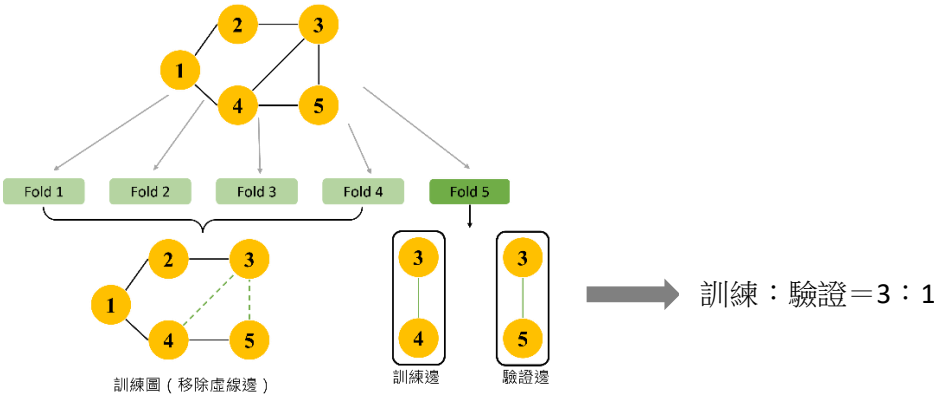
範籌；後者是指這三種資料都是在同一張圖上，也就是這次作業的內容。通常我們會希望正的邊（在訓練圖上沒有但是未來會有）在未來會存在，而負的邊（在訓練圖上沒有在未來也不會有）在未來不應該存在。

我的理解是 Link prediction 同時是分類與生成問題，在訓練過程中是一種二分類問題，我們只會專注於手邊有的正邊和負邊，但是實際上在我們為每一節點生成 embedding 時，每一個邊的值也被確定了（點之間的內積、分類器的分類結果等等），這也就代表我們其實生成了一張原始圖的重構圖。

（二） 資料切割與驗證方式

有別於一般的圖像分類問題，僅要將原始資料集分成訓練、驗證、測試就好，link prediction 需要有很複雜的切割流程，需要考慮到訓練圖與訓練邊和驗證邊，以及可見性的問題（例如：訓練邊和驗證邊都不應在訓練途中）。如圖一所是，我自己定義我的 5-fold 資料集，我把 fold 使用在整張圖上，也就是說每個 fold 都是保留原圖 4/5 的邊當作訓練圖，移除掉佔原圖 1/5 的邊，並將其切割成分類器的訓練資料和驗證資料（我用 sklearn 的 train_test_split 以 3:1 的方式切割）。

然而，圖一沒有呈現到負邊的概念（不好畫），我用字面的方式講述。原資料集中提供了某個邊不存在的資訊，一開始我覺得很多餘，畢竟他對建構原始圖完全沒有幫助，後來在[3]的介紹中才發現他的重要性。我們不能知道原始圖中沒有邊的地方到底哪裡是真的沒有邊的，所以這些資料集中不存在的邊就是要提供負邊的資訊。值得一提的是，負邊的數量遠大於 1/5 的原圖邊數，這樣會導致我們正負樣本比例不均，所以會需要一個好的採樣方式或是對損失函數賦予權重。我的實作方式是將負邊折成 5 個 fold 分別放入每個 Fold 中（在資料集探索處會提到具體的原因）。



圖一、k-fold 拆分原圖

（三）模型介紹

甲、Node2Vec + Logistic Regression

Node2Vec 是一種基於圖的結構和隨機遊走的方式去學習每個節點的表示方法，最後將圖中的每個頂點映射到特定維度的向量表示。我最後將學習出來的表示方法用 pairwise multiplication 來表示任兩節點的邊表示，過程中我也嘗試了 L1-distance, L2-distance, average 的方法，但是都相對差勁。最後，我用簡單的 Logistic Regression 去預測邊存在的機率。

給定 2 個節點特徵u, v維度為D, 有以下集種方式去融合出邊特徵

$$L1(u,v) = [|u_i - v_i|]_{1\times D}$$

$$L2(u,v) = [(u_i - v_i)^2]_{1\times D}$$

$$MUL(u,v) = [u_i * v_i]_{1\times D}$$

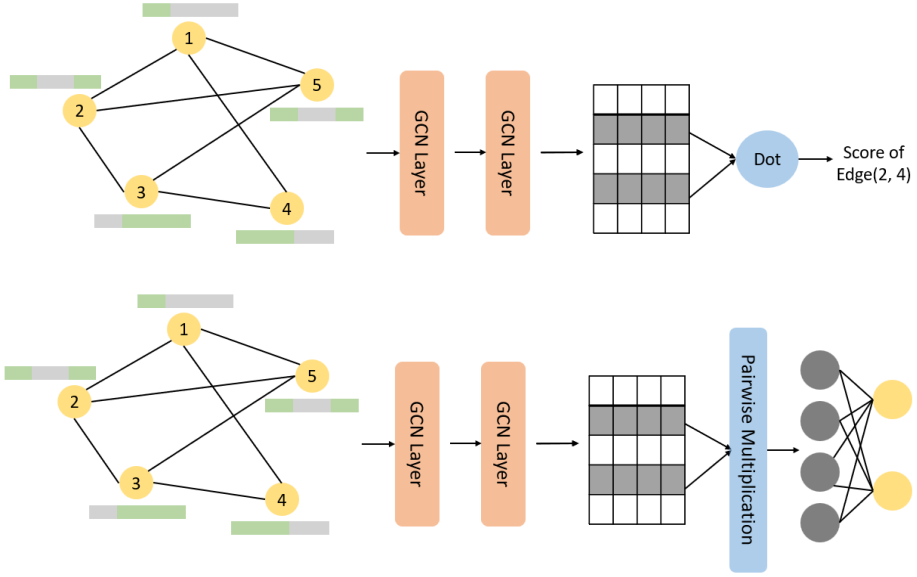
$$AVG(u,v) = \left[\frac{(u_i - v_i)}{2}\right]_{1\times D}$$

實作方法上我分為兩種：(1) 將 node2vec 對圖結構學習出表示後，兩兩表示以 pairwise multiplication 變成邊向量。將 attributes 使用 PCA 降維成連續型的特徵後取 L1-distance。最後把兩個表示串接起來組合成共同特徵，讓 Logistic Regression 去學習。(2) 我把所有的 attribute 都轉化成節點，當 attribute 欄位為 1 時表示節點須與該 attribute 節點連接（像是老師課堂上描述的方式）。最後對整個圖用 node2vec 去學習，取出圖中節點的向量並用 pairwise multiplication 變成邊向量，讓 Logistic Regression 去學習。

Node2Vec 的設定如下：p=1, q=1, embed_size=128, walks=10, context_size=10, walk_len=40, batch_size=128。因為我只是把這個方法當成起手式、baseline 的概念，所以沒有做過多的調參測試。

乙、GCN

使用兩層 GCN 並輸出 128 維度的向量，如下圖分成兩種分類器：(1) 將兩個節點向量內積後表示成邊的分數 (2) 將兩個節點用 pairwise multiplication 轉換成邊的向量後，用簡單的 MLP 分類成兩種類別有邊還是沒邊。



參數的設定如下：learning_rate=1e-5, batch_size=64, epochs=100, hidden_dim=256, out_dim=128。

丙、CAN

CAN 採用 variational autoencoder 的概念，加入 gaussian noise 如李弘毅老師所言，此方法可以增加模糊地帶的預測準確度，也就是我們希望在某段區域內的內容（而不是單點）都被預測成該輸入的標籤，也因此擴大它的泛用性。但是有別於一般 graph autoencoder，CAN 同時也對 attribute 使用 autoencoder，讓特徵層的也能有一樣的效果，最後達到更好的表現

訓練過程中優化以下極為複雜的損失函數就能訓練 CAN 了，GCN 輸出與 Dense 輸出的結果是高斯分布的μ和σ，用這個方法分別重參數化(reparametrize) 新的 embedding 和 attribute。重參數化的方法就是：$z' = \mu + Noise(\sigma) * e^{\sigma}$

$$\mathcal{L}(\theta,\phi;\mathbf{A},\mathbf{X}) = \frac{1}{N^2L}\sum_{l=1}^{L}\Big(\sum_{i,j\in\mathcal{V}}\log p_{\theta_1}(\mathbf{A}_{ij}\mid \mathbf{Z}_i^{\mathcal{V}(l)},\mathbf{Z}_j^{\mathcal{V}(l)})\Big)$$

$$+\frac{1}{NFL}\sum_{l=1}^{L}\Big(\sum_{i\in\mathcal{V},a\in\mathcal{A}}\log p_{\theta_2}(\mathbf{X}_{ia}\mid \mathbf{Z}_i^{\mathcal{V}(l)},\mathbf{Z}_a^{\mathcal{A}(l)})\Big)$$

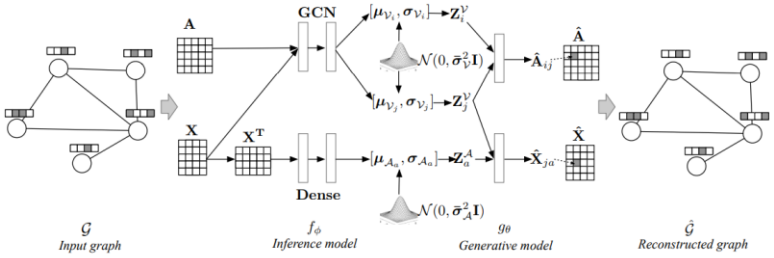
$$+\frac{3}{2N}\sum_{i\in\mathcal{V}}\sum_{d=1}^{D}\Big(1+\log(\frac{\boldsymbol{\sigma}^2_{\mathcal{V}_i}|_d}{\tilde{\boldsymbol{\sigma}}^2_{\mathcal{V}_i}|_d})-\frac{(\boldsymbol{\mu}_{\mathcal{V}_i}|_d)^2}{\tilde{\boldsymbol{\sigma}}^2_{\mathcal{V}_i}|_d}-\boldsymbol{\sigma}^2_{\mathcal{V}_i}|_d\cdot\tilde{\boldsymbol{\sigma}}^2_{\mathcal{V}_i}|_d\Big)$$

$$+\frac{1}{2F}\sum_{a\in\mathcal{A}}\sum_{d=1}^{D}\Big(1+\log(\frac{\boldsymbol{\sigma}^2_{\mathcal{A}_a}|_d}{\tilde{\boldsymbol{\sigma}}^2_{\mathcal{A}_a}|_d})-\frac{(\boldsymbol{\mu}_{\mathcal{A}_a}|_d)^2}{\tilde{\boldsymbol{\sigma}}^2_{\mathcal{A}_a}|_d}-\boldsymbol{\sigma}^2_{\mathcal{A}_a}|_d\cdot\tilde{\boldsymbol{\sigma}}^2_{\mathcal{A}_a}|_d\Big),$$

where

$$\begin{aligned}
\mathbf{Z}_i^{\mathcal{V}(l)} &= \boldsymbol{\mu}_{\mathcal{V}_i}+\boldsymbol{\sigma}^2_{\mathcal{V}_i}\odot\boldsymbol{\epsilon}^{(l)},\text{with }\boldsymbol{\epsilon}^{(l)}\sim\mathcal{N}(0,\mathbf{I}),\\
\mathbf{Z}_j^{\mathcal{V}(l)} &= \boldsymbol{\mu}_{\mathcal{V}_j}+\boldsymbol{\sigma}^2_{\mathcal{V}_j}\odot\boldsymbol{\epsilon}^{(l)},\text{with }\boldsymbol{\epsilon}^{(l)}\sim\mathcal{N}(0,\mathbf{I}),\\
\mathbf{Z}_a^{\mathcal{A}(l)} &= \boldsymbol{\mu}_{\mathcal{A}_a}+\boldsymbol{\sigma}^2_{\mathcal{A}_a}\odot\boldsymbol{\epsilon}^{(l)},\text{with }\boldsymbol{\epsilon}^{(l)}\sim\mathcal{N}(0,\mathbf{I}),
\end{aligned}\qquad(10)$$

參數的設定按照原 github 設定，我只更動了一些 github 上面的一些 bug（像是 argparser、GPU 不能使用的問題），將原始資料轉為 github 可以接受的格式後就可以 train 了。



（四） 最終模型

由於機器學習中希望越多的訓練資料越好，所以 5-fold 只是我調整參數的方式，並檢測模型的穩定性和平均準確度。最後的模型是把表現最好的 fold 的圖拿去訓練，這表示這個 fold 的圖相較於其他 fold 可以涵蓋最多資訊，所以整體的準確度最高，而且邊集合不再拆分成訓練邊和驗證邊，全部當成訓練資料

最後在 Public Leader Board 上，我在 dataset1 和 dataset2 是使用 CAN 模型而 dataset3 則是 GCN 結合內積的結果，最後在公開排行榜獲得班上第 4 名的表現（全部第 17 名），私藏排行榜課程排行不變但是全部排名上升了一位。

三、 實驗結果分析

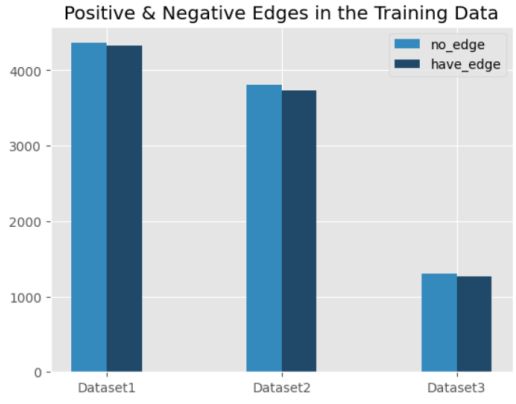
（一） 資料集探索

表一為我針對三個資料集的基本統計結果。可以看出三個資料集都算是小圖（node 數量都不多），而且每個點的平均 degree 約莫為 3 就是只有平均三個邊的概念。然而，三個資料集中以資料集 3 最小僅有 877 個節點、資料集 2 則有較多的節點特徵 3703 種特徵。

這裡要補充說明為甚麼將 negative edge 切五個 fold 放入每個 fold。我們在每個邊集合中會有(0.2*Pos#)個正邊，但是負邊有 Neg#個，這樣就會發生資料不平均的問題。而剛好在這三個資料集中 Pos#與 Neg#大小幾乎一樣，所以我將其拆分成五個 fold 放入每個 fold 的邊集合裡，這樣就約莫可以有等量的正、負邊，在優化過程也不會偏重優化負邊而導致正邊資訊被忽略。

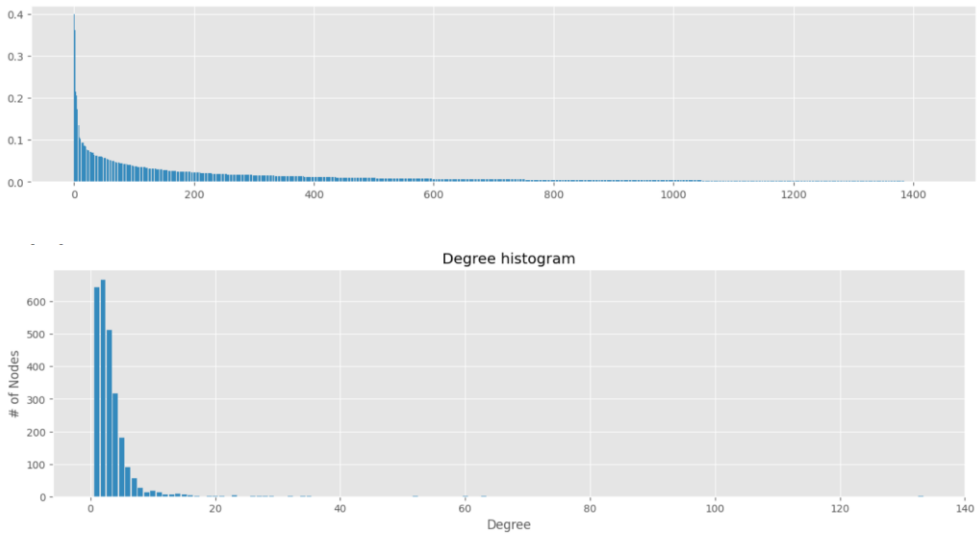
		資料集 1		資料集 2		資料集 3	
Nodes #		2708		3312		877	
Edegs #	**Pos #**	8686	4324	7544	3736	2572	1273
	Neg #		4362		3808		1299
Attributes #		1433		3703		1703	
Average Degree		3.271		2.473		2.920	

表一、三個資料集的基本統計量。



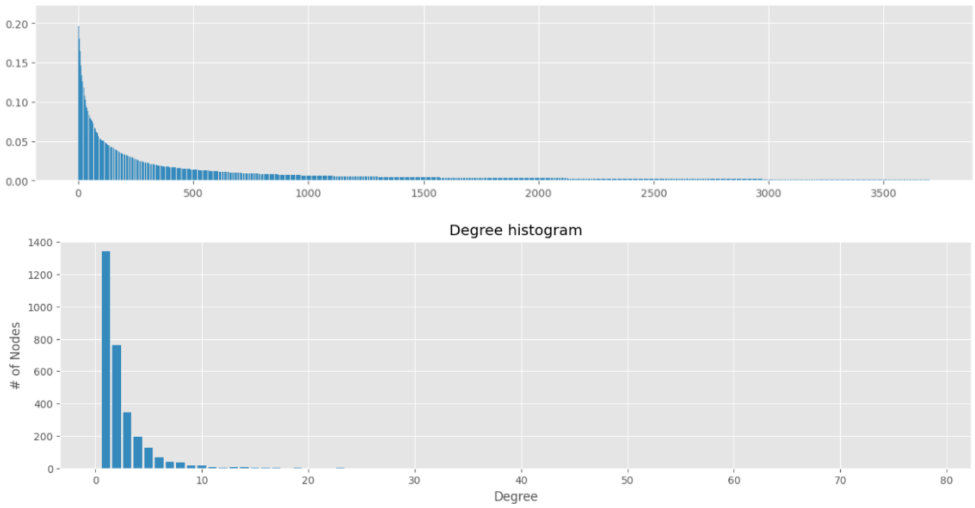
圖、原始圖正、負邊分布

甲、 資料集 1



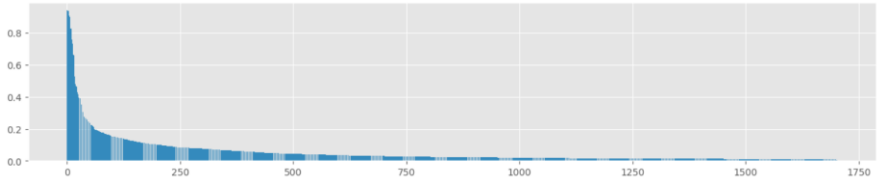
圖、資料集 1 每個特徵的平均值，下圖則是 Degree 的分布關係

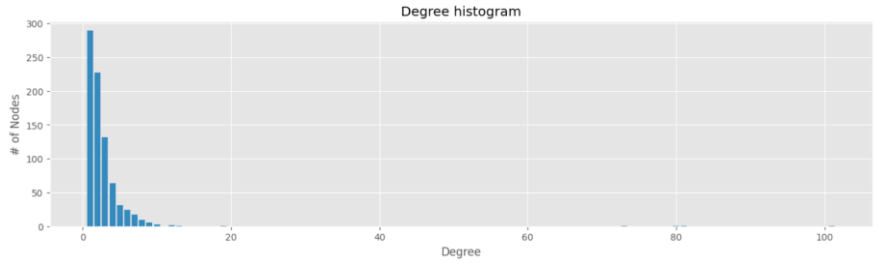
乙、 資料集 2



圖、資料集 2 每個特徵的平均值，下圖則是 Degree 的分布關係

丙、 資料集 3





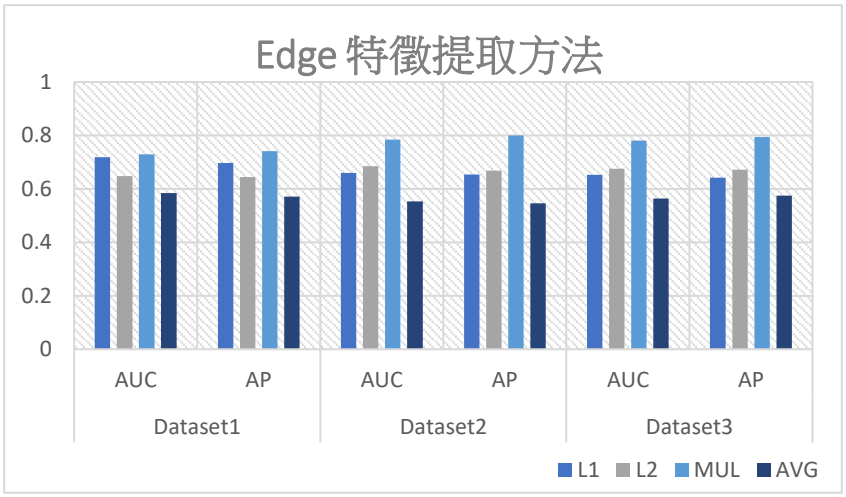
圖、資料集 3 每個特徵的平均值，下圖則是 Degree 的分布關係

綜合三圖可以看到 degree 分布三者都是許多小 degree 的節點所組成的，而資料集 1,2 的特徵值大部分都為 0（非常稀疏的向量），平均每個 user 的 attribute 有 1 的數量少。但是資料集 3 的特徵值中平均每個 user 的 attribute 有 1 的數量比較多，而這些因素也可能是模型最後表現的影響關鍵。

（二） Node2Vec 與基本分類器

甲、 邊的表示運算方法

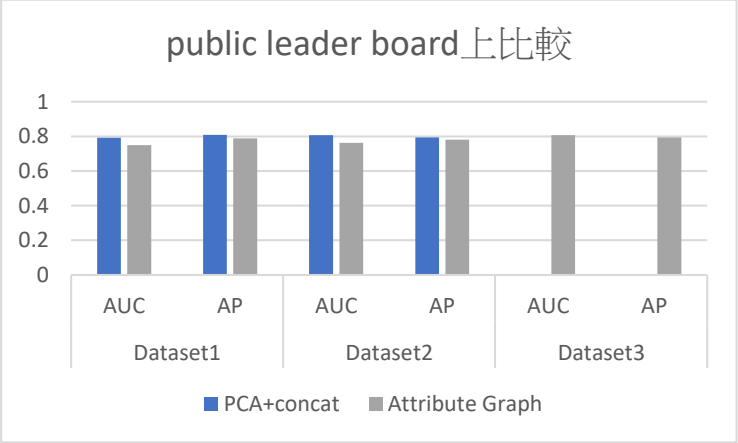
下圖是我實驗不同邊的表示方式在 5-Fold 中的平均表現。我發現使用 pair-wise multiplication 來表示邊可以有最好的成果，而 L1 距離和 L2 距離次之，最直覺的平均法有最差的表現（可以和 pair-wise multiplication 差距超過 20%）。這也說明後續為甚麼在 GCN, CAN 的模型中要使用簡單的內積就好，因為內積其實就是 pair-wise multiplication 的加總結果，可以最大保留乘積特徵。



圖五、不同計算邊的方式在 5-Fold 的平均表現。

乙、 Attributes 結合邊表示

分成兩種方式，一種是將 one-hot attributes 用 PCA 降維後直接接在邊向量上，另一種是將 attributes 當作 node 用 node2vec 演算法學出共同表示方法。實驗數據顯示，前者也就是 PCA 降維在測試集上可以達到更好的表現，但是在資料集 3 會出現 bug（可能是刪除掉 degree 低的邊導致 degree 變 0，所以 node2vec 缺失該節點，我沒有為它 debug 了）。

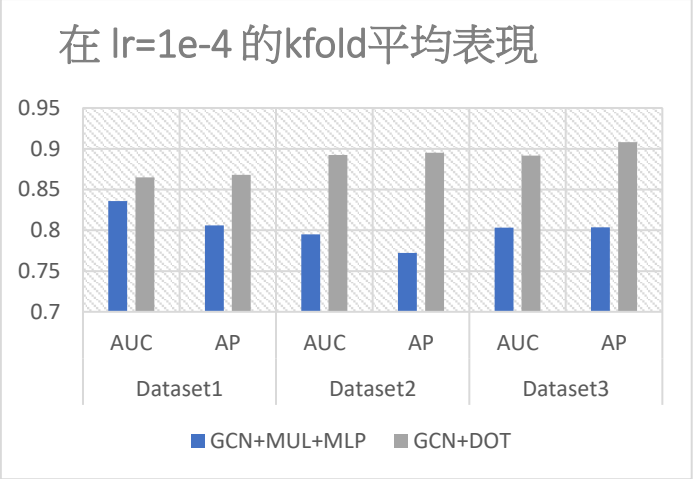


圖、PCA 與 Attribute graph 在測試資料的表現。

（三）GCN 編碼器

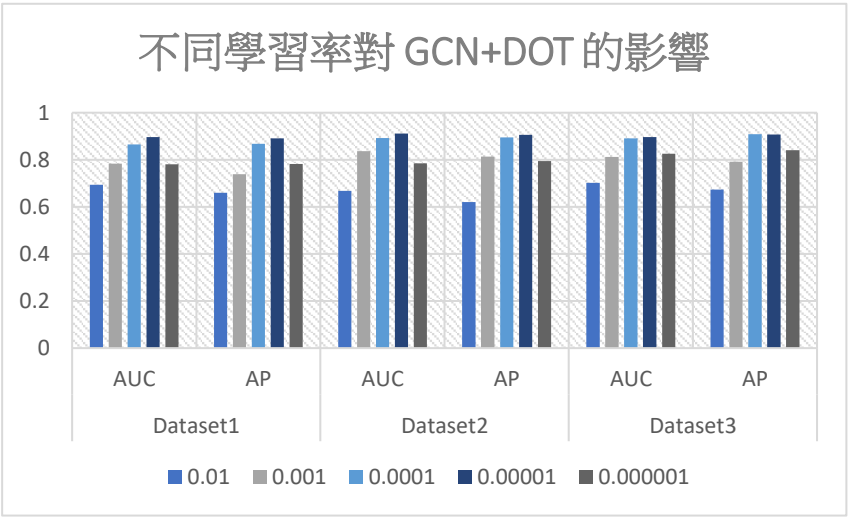
甲、 分類層的選擇

使用一層 MLP 和使用內積就有很明顯的差異（當學習率皆為 1e-4 時），僅使用內積看似簡單的運算卻可以避免掉過複雜而導致的 overfitting。



乙、 學習率的影響

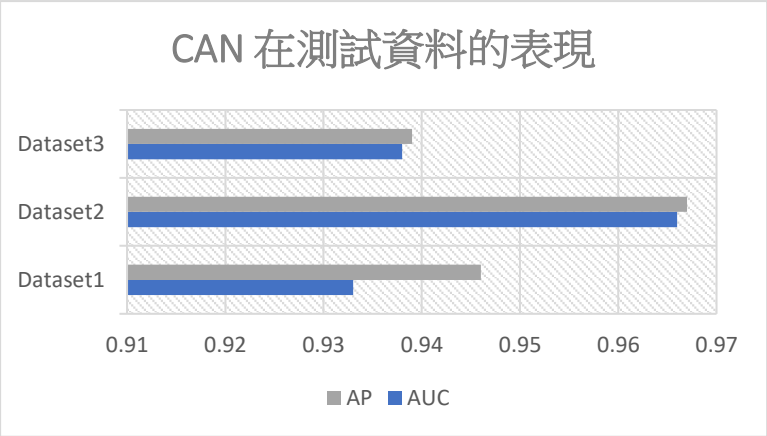
學習率對模型的影響很大，由圖可知對於 GCN 內積模型最佳的學習率為 1e-5。過大的學習率會表現會有明顯下降，可能是沒辦法收斂或是卡在 local minimum 的效應。



圖、不同學習率對 GCN+內積運算的成果影響。

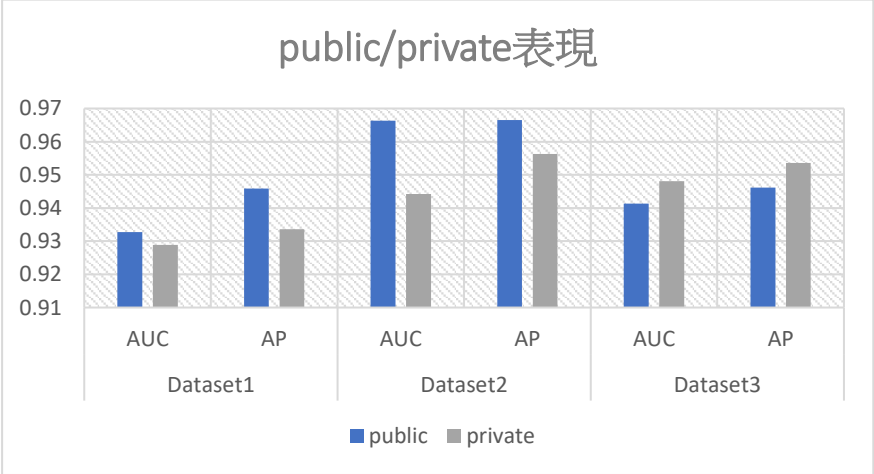
（四） CAN

因為我使用他人的 github 所以要將 k-fold 加入比較困難，所以只有在測試集下比較成果。不論 AUC 或是 AP 都能達到 0.93 以上。



（五） 最終於測試集上的表現

下圖是 leader board 統計結果，private 的表現都有下降的趨勢（除資料 3 外），也應證在小資料下，GCN 傾向做得更好。最後在 private leader board 上升了一名，表示我的訓練方法和模型確實有一定的泛化性。



四、 總結與心得

（一） 結論

針對 Node2Vec 的方法，我發現將 attribute 視為節點並加入圖中的準確度並不比串接節點表示和 attribute 表示好，原因可能是因為我沒有幫 attribute node 和 graph node 設置不同權重，導致它們的地位是一樣的，也就是在 attribute graph 中並沒有區隔它們，最後學出比較不合適

的表示方法。而將特徵串接可以獨立出圖結構特徵和 attribute 特徵，分類器反而能學出其中的轉換關係所以分類分得更好。

然而比較 GCN 和 Node2Vec 方法可以了解到讓模型自己去學習圖特徵能夠更靈活地去表示圖結構，同時他又結合了圖與 attribute 的關係，所以表現能夠比 Node2Vec 單獨學習圖結構更好。此外，從 Node2Vec 轉換到邊向量的比較中可以發現最好的表示方法是將兩向量做 pairwise multiplication；同時比較 GCN 分類器發現僅用簡單的內積 (sum of pairwise multiplication)比用複雜的 MLP 更好。不管在傳統機器學習或是深度學習的方法，運用到節點乘積的結果可以更好的表示邊。

比較 CAN 和 GCN 方法能發現加入高斯分布 Noise 可以增加模型的 generalizability，原因是因為在模糊地帶上，也就是訓練資料沒有的正負邊的資訊時，我們很難保證 GCN 能在那些地方表現的很好，但是 CAN 透過 Noise 的方式讓某段區域內的結果都能被優化到達到更好的表現，同時有將高斯的概念用到 attribute 重構也是提升表現的一大亮點。

在最後的 private leader board 上面，我上升了一名也表示了整體的演算法泛化性蠻強的。而此次作業我的創新點在於：(1) 我將 k-fold 用到 link prediction 上當作調整參數的方法 (2) 比較不一樣的節點到邊的表示，進一步推演出為甚麼後續使用 inner product (3) 使用 CAN 共同學習 attribute 和 graph 進一步優化最終表現。

（二）心得

因為這次作業是用比賽的方式執行，所以我更有動力去做作業，同時也要擔心下一秒會不會被超過。在最後繳交的兩個小時，我用 CAN 模型在 final score 以 0.004 超越第四名，當下就是滿滿的喜悅。

我覺得可以再精進的地方就是使用新的 SOTA 方法看看，可以著重再 Cora 這個資料集的表現，因為他的 attribute、node、edge 數量與此次作業較相近。過程中其實有嘗試使用 2023 最新的 NESS 但是實作起來怪怪的(也沒有付上 code)，所以沒有把它寫到 report 中。

五、 參考資料

[1] Meng, Z., Liang, S., Bao, H., & Zhang, X. (2019, January). Co-embedding attributed networks. In Proceedings of the twelfth ACM international conference on web search and data mining (pp. 393-401).

[2] Zqfeng. (2021, August 12). Graph: Train, Valid, and Test Dataset Split for Link Prediction. Pleiades. https://zqfang.github.io/2021-08-12-graph-linkpredict/

[3] Stellergraph. (2018). Link Prediction with Node2Vec. StellerGraph. https://stellargraph.readthedocs.io/en/stable/demos/link-prediction/node2vec-link-prediction.html

[4] Chen, G. (2022, June 5). CAN-Pytorch. Github. https://github.com/GuanZhengChen/CAN-Pytorch