Pop & Rap Artists Collaboration Recommendation Using Spotify Data

Laporan Tugas Kelompok 2 Data Mining

Rocky Arkan Adnan Ahmad - 1806186566
Mahardika Krisna Ihsani - 1806141284

Desember 2021

Abstract

Pada eksperimen ini, kami membangun sistem rekomendasi kolaborasi untuk *artist* pop & rap dengan menggunakan pendekatan *graph mining* dengan menggunakan *spotify API* sebagai sumber data. Kami mendefinisikan setiap *artist* sebagai verteks dalam graf dan *edge* untuk menghubungkan dua artist yang pernah berkolaborasi dimana *edge* bisa berbobot. Bobot *edge* menunjukkan frekuensi kolaborasi dari satu pasang *artist*. Permasalahan tersebut kami definisikan sebagai *link prediction* atau dengan kata lain, diberikan dua verteks beserta fitur-fiturnya, prediksi apakah akan diberikan *edge* atau tidak untuk menghubungkan kedua verteks secara langsung. Kami menggunakan dua jenis pendekatan, yakni dengan memanfaatkan algoritma *machine learning* konvensional dan memanfaatkan arsitektur *graph neural network*. Pada pendekatan *machine learning* tradisional, kami menguji dua jenis fitur yakni fitur *link-based* yang terdiri atas *jaccard similarity*, *adamic-adar measure*, *preferential attachment*, *resource allocation index*, *common neighbors*, dan simrank serta fitur node2vec. Lalu untuk pendekatan *graph neural network*, kami menguji tiga jenis *convolution layer* yakni GAT, GCN dan SAGE. Ekseperimen kami memperoleh hasil bahwa *graph neural network* tidak menghasilkan performa yang secara umum tidak lebih baik dibandingkan dengan pendekatan *machine learning* konvensional. Performa terbaik diperoleh oleh pendekatan *machine learning* konvensional dengan fitur node2vec dengan skor F1 sebesar 0.76-0.77 dan ROC-AUC sebesar 0.8.

Kata Kunci— *artist collaboration recommendation*, *graph mining*, *link prediction*, node2vec, *link-based features*, *graph neural network*

1 *Task Definition*

Task yang kami kerjakan adalah merancang suatu sistem yang bisa memberikan rekomendasi kolaborasi dari pasangan *artist* pop dan rap yang belum pernah berkolaborasi dengan pendekatan *graph mining*.

Kami merepresentasikan setiap *artist* sebagai suatu vertex pada graf. Lalu *edge* pada graf menunjukkan bahwa kedua artist yang dihubungi oleh *edge* tersebut pernah berkolaborasi. *Edge* pada graf bersifat *undirected* dan mempunyai bobot tertentu dimana bobot ini menunjukkan frekuensi kolaborasi kedua artist yang terhubung oleh *edge* tersebut.

Task ini kami bisa definisikan sebagai *link prediction* dari pasangan vertex yang belum terhubung secara langsung. Karena *link prediction* sendiri merupakan persoalan untuk memberikan atau tidak memberikan *edge* dari suatu pasang vertex, maka kita bisa definisikan *task* ini sebagai suatu permasalahan *binary classification* dimana *output* akan bernilai 1 atau *true* apabila pasangan vertex akan diberi *edge* atau rekomendasi dan 0 atau *false* apabila pasang vertex tidak diberi *edge* atau rekomendasi.

2 Metode Pengumpulan Data

Data yang kami gunakan merupakan data lagu yang diperoleh dengan memanfaatkan spotify web API dan diambil dari beberapa *playlist* yang ber-*genre* pop dan rap (beserta variasinya seperti hip-hop). Hal tersebut dilakukan karena spotify web API tidak menyediakan *endpoint* API yang dikhusus untuk mengumpulkan lagu berdasarkan *genre* lagu. Tidak bisa dipungkuri bisa saja terdapat beberapa lagu yang tidak termasuk dalam kedua genre tersebut namun termasuk dalam data kami. Pengumpulan sampel data dilakukan dengan melakukan pengiriman suatu *request* ke *endpoint* https://api.spotify.com/v1/playlists/⟨id_playlist⟩/tracks.

Setelah itu, data yang didapatkan dari proses pengiriman *request* tersebut lalu kami lakukan *filtering* dengan kriteria suatu lagu terdapat lebih dari satu *artist* yang menyanyikan lagu tersebut. Lalu apabila lagu tersebut sudah memenuhi, langkah selanjutnya adalah memformatnya data lagu tersebut menjadi satu *tupple* berbentuk (artist1, artist2, ...artist ken, judul lagu), dimana semua artist akan terurut secara leksikografis. Judul lagu ini berguna untuk menghilangkan duplikasi lagu. Lalu pengurutan artist pertama dan artist kedua dilakukan untuk memudahkan proses perhitungan bobot untuk *edge* yang menghubungkan artist pertama dan artist kedua.

Setelah itu, kami hilangkan beberapa lagu duplikat sehingga tersisa lagu-lagu yang unik untuk digunakan pada *dataset* kami. Karena suatu *tupple* dari *list of tupples* yang diperoleh bisa saja terdiri atas lebih dari dua artist dan juga semua tupple masih mengandung judul lagu dalam sehingga untuk proses selanjutnya, maka langkah selanjutnya adalah memecahnya menjadi beberapa *tupple* yang berformat (artist1, artist2). Sebagai contoh, misalkan terdapat *tupple* (aimer, lilas ikuta, milet, omokage), maka akan diperoleh sekumpulan *tupple* (aimer, lilas ikuta), (aimer, milet), dan (lilas ikuta, milet).

Setelah diperoleh *list of tupple* artist yang berkolaborasi, langkah selanjutnya adalah menjadikan *list* tersebut sebagai *dataframe*. *Dataframe* tersebut digunakan untuk menghitung bobot setiap pasangan *artist* unik dengan menghitung frekuensi kemunculan pasangan tersebut. Lalu menghilangkan pasangan duplikat pada *dataframe*.

Langkah selanjutnya adalah membuat data pasangan *artist* yang tidak berkolaborasi. Langkah ini dilakukan dengan melakukan operasi *cartesian product* dari himpunan artist1 dengan himpunan artist2 lalu dilakukan operasi *set difference* dengan himpunan pasangan artist yang pernah berkolaborasi yang didapat dari langkah sebelumnya. Himpunan pasang artist ini akan dimasukkan ke *dataframe* dan dilakukan *assignment* bobot bernilai 0. Karena jumlah pasangan *artist* yang tidak berkolaborasi jauh lebih banyak dibandingkan dengan pasangan yang pernah berkolaborasi, maka dilakukan *non-repeating sampling* sejumlah banyak pasangan artist yang pernah berkolaborasi agar bisa diperoleh data yang *balanced*.

Tahap terakhir adalah membuat fitur target prediksi untuk menandakan apakah pasangan artist pernah berkolaborasi atau tidak dengan memanfaatkan nilai bobotnya. Nilai tersebut akan bernilai true apabila nilai bobotnya positif dan false apabila nilainya 0.

Graf diperoleh dengan memanfaatkan semua pasangan artist yang pernah berkolaborasi. Lalu untuk setiap pasangannya, akan dibuat *undirected edge* yang menghubungkan artist1 dan artist2 pada pasangan tersebut dengan bobot pasangan tersebut.

Dari pengumpulan data tersebut, diperoleh graf dengan jumlah *node* sebanyak 568 dan *edge* sebanyak 605. Berikut merupakan graf yang kami gunakan untuk eksperimen.

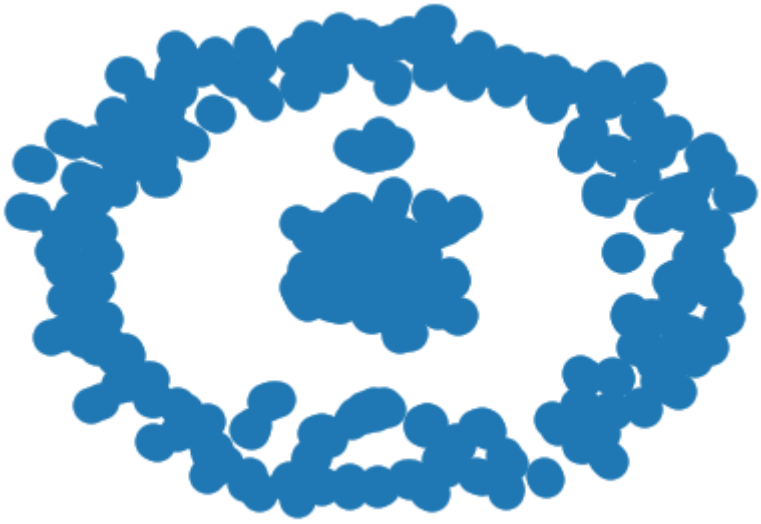


Figure 1: Visualisasi graf kolaborasi artist pop & rap

3 Eksperimen

Kami menguji beberapa pendekatan untuk melakukan *link prediction* yakni dengan menggunakan fitur-fitur yang berkaitan dengan *edge* pada graf seperti *Jaccard Similarity* dan *Adamic-Adar Measure*, memanfaatkan fitur node2vec untuk menghasilkan *embedding* untuk semua vertex pada graf, dan menggunakan *graph neural network* yang memanfaaktan *node-based feature* seperti *degree centrality* dan *closeness centrality*. Semua pendakatan di atas kami evaluasi dengan menggunakan metrik standar untuk *classification task* yakni f1, precision, recall, dan roc-auc,

3.1 *Link Feature-Based Prediction*

Dalam pendekatan ini, dilakukan perhitungan dengan berbagai macam algoritma *link prediction* dan *node similarity*. Hasil dari perhitungan ini akan dijadikan sebagai fitur untuk sebuah *machine learning model*. Lalu, menggunakan fitur-fitur yang diberikan model tersebut akan melakukan klasifikasi untuk melakukan *link prediction*.

Dalam eksperimen ini kami menggunakan model-model berikut, yaitu:

- CatBoost

- Random Forest
- Gradient Boost
- Gaussian Naive Bayes
- K-nearest Neighbor dengan K = 5

Lalu, perhitungan algoritma *link prediction* dan *node similarity* yang digunakan sebagai fitur adalah:

- *Jaccard Similarity*
- *Adamic-Adar Measure*
- *Preferential Attachment*
- *Resource Allocation Index*
- *Common Neighbors*
- *SimRank*

Kami juga melakukan normalisasi untuk nilai dari fitur yang tidak berskala dari 0 sampai 1, seperti *Common Neighbors* dan *Preferential Attachment*. Metode normalisasi yang kami lakukan adalah dengan *MinMax Scaling*.

3.2 Node2Vec

3.2.1 Definisi

Node2Vec merupakan metode untuk menghasilkan *embedding* untuk vertex dari suatu graf. Ide dasar dari algoritma ini adalah melakukan beberapa kali *random walk* dengan menggunakan bias berupa suatu probabilitas transisi dari suatu node ke node lainnya serta dengan jumlah langkah tertentu. Anggaplah sudah dilakukan *random walk* dari verteks t ke verteks v. Lalu akan ditentukan node selanjutnya yang akan dikunjungi (katakanlah sebagai node x). Maka probabilitas transisi untuk mengunjungi node x dari node v apabila dilakukan pengunjungan node t sebelum mengunjungi node v bisa didefinisikan sebagai berikut.

$$\alpha(t,x) = \begin{cases} \frac{1}{p} & \text{jika } d_{t,x} = 0 \\ 1 & \text{jika } d_{t,x} = 1 \\ \frac{1}{q} & \text{jika } d_{t,x} = 2 \end{cases}$$

Perhatikan $d_{t,x}$ merupakan jarak terpendek dari node t ke node v serta p dan q merupakan *hyperparameters* untuk fungsi probabilitas transisi tersebut. Proses *random walk* ini akan menghasilkan sekuens verteks yang dikunjungi dalam sekali jalan. *Random walk* ini akan dijalankan berulang kali dalam jumlah iterasi tertentu. Berikut beberapa *hyperparameters* yang perlu diatur untuk menggunakan metode node2vec.

- p: untuk mengatur probabilitas pengunjungan node yang sebelumnya dikunjungi. Semakin kecil nilai p-nya maka akan semakin besar probabilitas untuk mengunjungi sebelumnya dan sebaliknya.
- q: untuk mengatur probabilitas pengunjungan node yang terhubung secara tidak langsung dari *node* sebelumnya. Semakin kecil nilai q, maka pengunjungan akan cenderung semakin ekspansif serta berlaku sebaliknya.
- jumlah langkah: parameter untuk menentukan berapa langkah yang diperlukan untuk melakukan satu kali *random walk*
- jumlah *walk*: parameter untuk menentukan jumlah iterasi dilakukannya *random walk*

Setelah dilakukan beberapa kali *random walk*, langkah selanjutnya adalah memanfaatkan metode *embedding* word2vec dengan input berupa daftar sekuens verteks yang diperoleh pada proses sebelumnya untuk menghasilkan *node embedding*.

Untuk *link prediction*, diperlukan *embedding* untuk masing-masing pasangan verteks. Kita bisa memanfaatkan *node embedding* yang diperoleh pada proses sebelumnya untuk menghasilkan *embedding* antara dua node dengan menggunakan operator berikut.

- $g(u,v) = \frac{f(u)+f(v)}{2}$ (average)
- $g(u,v) = f(u) * f(v)$ (hadamard)
- $g(u,v) = |f(u) - f(v)|$ (L1)
- $g(u,v) = (f(u) - f(v))^2$ (L2)

Nilai g tersebut akan dijadikan fitur untuk melakukan prediksi. [2]

3.2.2 Pengaturan Eksperimen

Untuk pendekatan node2vec, kami menggunakan model *machine learning* berikut sebagai *classifier*.

- Logistic Regression
- Adaboost
- Random Forest
- MLPClassifier

Semua model di atas kami panggil melalui *library* scikit-learn. Fitur yang digunakan untuk prediktor adalah berupa *node2vec embedding* dari suatu pasangan verteks. Lalu untuk *hyperparameters* yang kami gunakan untuk node2vec sendiri kami melakukan semacam *grid search* untuk menentukan *hyperparameters* yang ideal dengan menggunakan model *baseline* yakni logistic regression. Proses tersebut adalah mencari *hyperparameters* manakah yang akan menghasilkan f1-score terbesar. Dari proses tersebut didapatkan *hyparameters* sebagai berikut.

- p: 2
- q: 1
- jumlah langkah: 20
- jumlah *walk*:8

Proses *data splitting* dilakukan dengan metode *5-fold cross validation* terhadap *dataset*. Proses *data splitting* tersebut dilakukan dengan bantuan *library* sklearn.

3.3 *Graph Neural Network with Node Features*

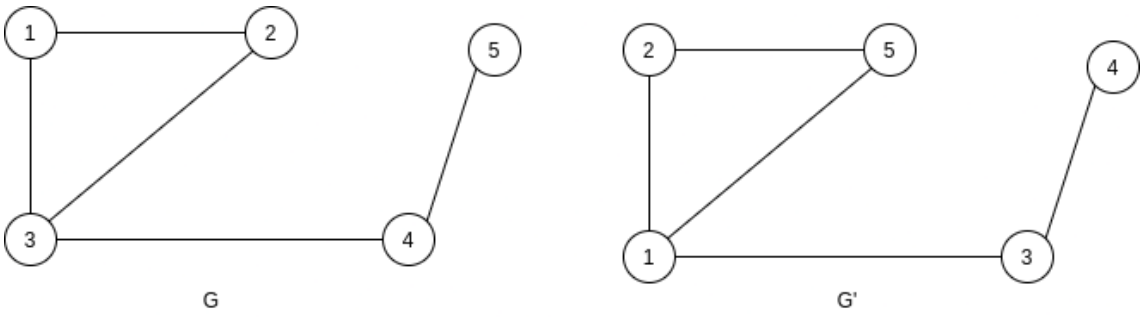
3.3.1 Definisi

Pada metode sebelumnya, kita menggunakan metode *embedding* serta *feature-based*yang cukup dependen dengan *node* yang ada. Sesuai dengan kalimat sebelumnya, permasalahan dari metode-metode sebelumnya adalah metode-metode tersebut cukup dependen terhadap *node* yang ada di graf untuk training sehingga untuk tidak bisa dihasilkan fitur dari *node*

yang tidak termasuk dalam graf *training*. Berangkat dari limitasi tersebut, terdapat metode *node embedding* yang bisa mengatasi permasalahn tersebut dengan memanfaatkan arsitektur *neural network* bernama *graph neural network*. *Graph neural network* (GNN) memanfaatkan *parameter sharing* dan menggunakan operasi konvolusi seperti pada arsitektur *convolutional neural network* (CNN). Namun arsitektur CNN tidak bersifat *permutation invariant* yang terjadi pada masukan graf. Artinya adalah, semua verteks pada graf apabila dilakukan penukaran label pada suatu verteks, verteks dengan posisi tertentu semestinya mempunyai *node embedding* yang sama. Apabila diformulasikan, katakanlah akan dilakukan pemetaan suatu verteks posisi ke-i (v_i) dari graf G serta graf G' ke vektor $\mathbb{R}^d$, dimana d adalah dimensi dari *embedding*, G merupakan graf asli, dan G' merupakan graf yang dilakukan permutasi label verteks. Katakanlah **V** merupakan semua verteks pada graf G, maka *permutation invariance* akan berlaku apabila

$$\forall v_i, f(G, v_i) = f(G', v_i) \text{ dimana } v_i \in \mathbf{V}$$

Sebagai contoh, diberikan gambar berikut dimana graf G dengan graf G' yang merupakan graf hasil permutasi dari G.



Maka *permutation invariance* untuk *node 1* di graf G berlaku apabila hasil pemetaan *node* 1 pada graf G akan sama dengan hasil pemetaan *node* 2 dari graf G'. Syarat tersebut berlaku untuk semua verteks di graf G untuk didapatkan berlaku *permutation invariance*.

Untuk mencapai sifat *permutation invariance*, ide dasar dari GNN adalah menghasilkan *node embedding* dengan mengagregasi informasi *neighborhood* hasil propagasi yang didapatkan dari tetangganya. Suatu *GNN layer* terdiri atas dua tahap yakni *message function* dan *aggregate function*. Proses perhitungan *GNN layer* dimulai melalui *message function* dimana fungsi ini berguna untuk menghasilkan informasi yang akan disampaikan oleh suatu *node* tetangga dari suatu *node*. Setelah masing-masing *node* tetangga menghasilkan informasi melalui *message function*, selanjutnya semua informasi tersebut akan diagregasi

ke *node* tujuan sehingga hasil akhirnya adalah *node embedding* dari *node* tujuan. Jumlah *layer* pada GNN menunjukkan jarak maksimum dari tetangga sehingga apabila kita menggunakan *2-layer* GNN, maka *node embedding* dari suatu *node* dari informasi tetangga-tetangganya yang berjarak 1 hingga 2 *edge*. Ilustrasi berikut menggambarkan *2-layer GNN* untuk menghasilkan suatu *node embedding*.

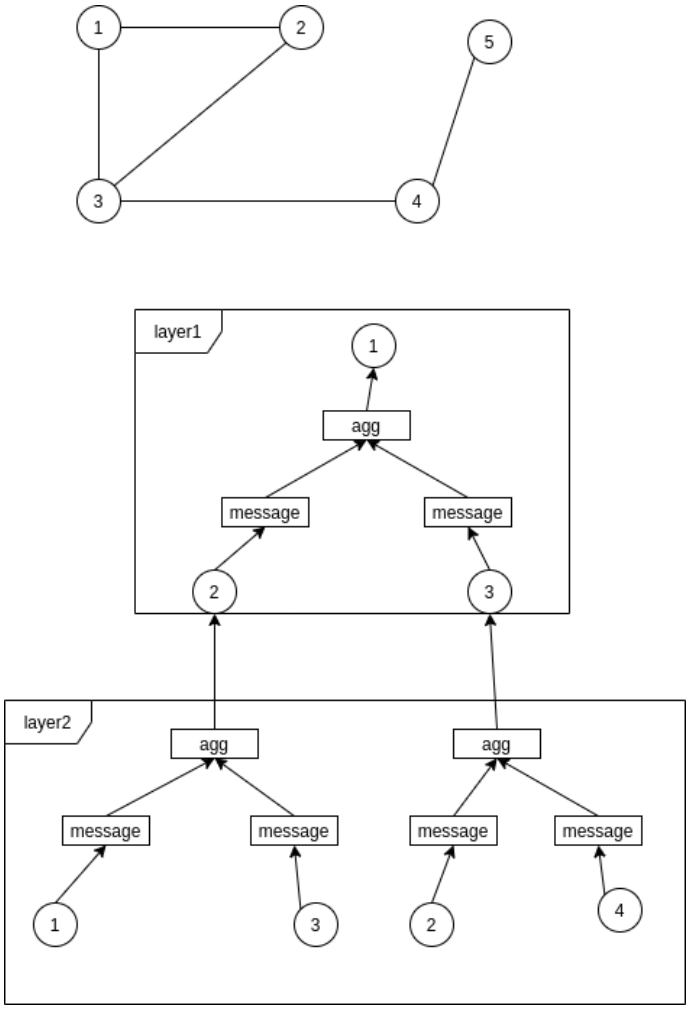


Figure 2: Ilustrasi 2-layer GNN untuk *node embedding node* 1 dari suatu graf

GNN layer ini mempunyai beberapa variasi, di antaranya terdapat GCN, SAGE, dan GAT. Ketiga variasi kami ujikan performanya pada eksperimen ini.

3.3.2 Pengaturan Eksperimen

Untuk eksperimen dengan menggunakan GNN, kami memanfaatkan *library* torch-geometric. *textit library* tersebut digunakan untuk membangun suatu GNN dengan mengutilisasi *framework* pytorch. Berikut merupakan struktur arsitektur GNN yang kami gunakan untuk eksperimen.

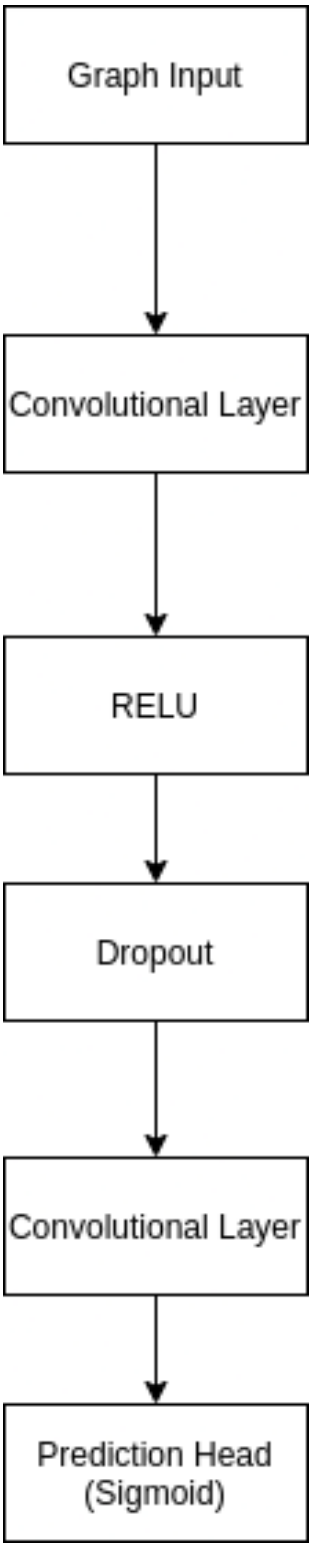


Figure 3: Arsitektur GNN yang kami gunakan

Di situ kami menggunakan *layer Dropout* untuk melakukan proses regularisasi setelah masukan melalui *layer* untuk konvolusi. *Layer* konvolusi inilah yang kami variasikan untuk eksperimen ini.

Task yang digunakan merupakan *binary classification* sehingga *loss function* yang digunakan adalah *binary cross entropy*. Lalu kami menggunakan Adam sebagai *optimizer* untuk pelatihan GNN kami.

Selain menggunakan torch-geometric, kami juga menggunakan *deepsnap* untuk mentransformasikan format networkx menjadi format yang bisa digunakan untuk pelatihan GNN. *Library* tersebut juga digunakan untuk melakukan proses *data splitting* dengan rasio untuk *training set*, *validation set*, dan *test set* masing-masing sebesar 0.8, 0.1, dan 0.1.

Hyperparameters yang kami gunakan untuk proses pelatihan GNN adalah sebagai berikut.

- jumlah epoch: 300
- hidden_dim: 128
- dropout: 0.3

Lalu untuk semua *node* pada *training set, validation set, test set*, kami tentukan *node-based feature* dengan menggunakan *library* networkx. Fitur-fitur tersebut terdiri atas.

- *pagerank*
- *betweenness centrality*
- *degree centrality*
- *load centrality*
- *closeness centrality*

4 Evaluasi

4.1 Metodologi Eksperimen

Untuk semua pendekatan, kami melakukan eksperimen menggunakan data yang kami kumpulkan sebelumnya. Kami melakukan eksperimen berupa *link prediction* terhadap data yang dihilangkan beberapa *edge*-nya. Data dibagi menjadi dua, yaitu *training data* dan *test data*.

Training data adalah data yang dibuat menjadi graf dalam model, lalu model melakukan *link prediction* terhadap edge dari *test data*.

Kami melakukan eksperimen dengan menggunakan KFold *cross-validator*, dengan 5 buah *fold*. Lalu, hasil dari eksperimen dinilai dengan *score* F1 dan ROC-AUC. Lalu, kami juga melakukan analisis eror untuk masing-masing pendekatan. Untuk melakukan analisis eror kami membagi data menjadi dua, yaitu *training data* dan *test data* yang dibagi dengan rasio 65:35. Lalu, *confusion matrix* dibuat dari hasil prediksi model. Analisa eror kami lakukan dari hasil *confusion matrix* tersebut.

4.2 Hasil Eksperimen dan Pembahasan

4.2.1 *Link Feature-Based Prediction*

Kami mendapatkan hasil performa sebagai berikut untuk masing-masing model:

Model	F1	ROC-AUC
CatBoost	**0.7494651996531043**	**0.7732839844974148**
Gradient Boost	0.7465124435533961	0.771678675146046
Random Forest	0.7455815949709315	0.7716835543094166
Gaussian Naive Bayes	0.6550479151595167	0.7296308071556012
K-nearest Neigbor	0.7443969735218093	0.7601024194155477

Dapat dilihat kalau performa F1 dan ROC-AUC terbaik diraih oleh model CatBoost. Gradient Boost dan Random Forest juga cukup setara dengan CatBoost. Ini kemungkinan karena ketiga model adalah merupakan model tipe *ensemble*, sehingga mendapatkan performa yang lebih baik dibandingkan model lainnya yang lebih sederhana. Selain itu, model K-nearest Neighbor sudah bisa mendapatkan performa yang tidak jauh berbeda dengan model *ensemble*, dan Gaussian Naive Bayes mendapatkan performa terendah dengan nilai yang cukup jauh dibanding model lainnya.

Lalu, kami melakukan eror analisis terhadap *confusion matrix* untuk masing-masing model. Hasil *confusion matrix* adalah sebagai berikut:

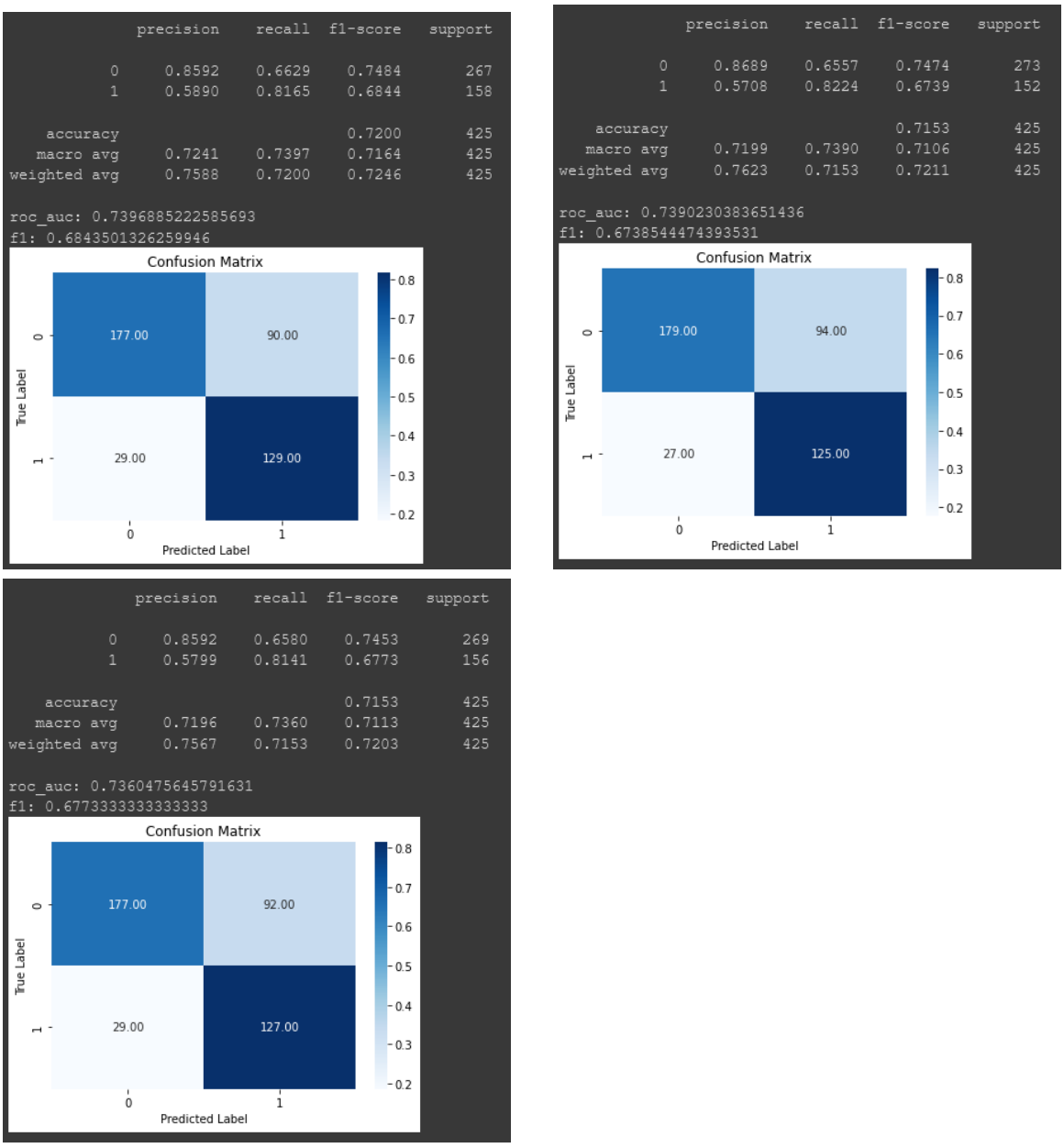


Figure 4: Confusion Matrix (dari kiri ke kanan & atas ke bawah, CatBoost - GradientBoost - Random Forest

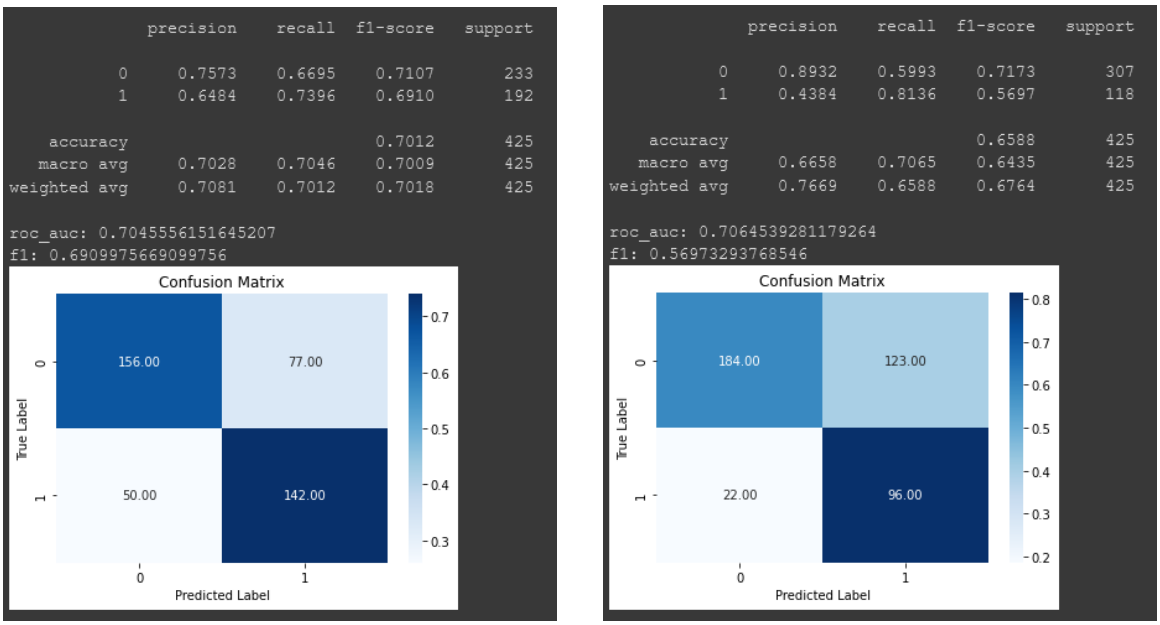


Figure 5: Confusion Matrix (dari kiri ke kanan, K-nearest Neighbor - Gaussian Naive Bayes

Terlihat untuk semua model mempunyai *false positive* yang cukup banyak. Lalu, untuk model *ensemble* dan Naive Bayes mempunyai ciri-ciri yang sama, yaitu untuk label 0 mempunyai *precision* yang tinggi tetapi *recall* yang rendah, lalu untuk label 1 mempunyai *precision* yang rendah tetapi *recall* yang tinggi. Walaupun begitu, keempat model mempunyai *false negative* yang rendah dan total *true positive* dan *true negative* yang cukup tinggi, terutama untuk model *ensemble*.

Disisi lainnya, model KNN mempunyai ciri-ciri yang berbeda dibanding model lainnya. KNN menghasilkan *precision* dan *recall* yang seimbang untuk semua label. Lalu, KNN juga mempunyai *false positive* paling rendah tetapi mempunyai *false negative* paling tinggi. Ini membuat KNN menghasilkan nilai yang paling *balanced* dan mempunyai nilai F1 paling tinggi, tetapi mempunyai nilai ROC-AUC paling rendah.

Lalu, kami juga mengobservasi kenaikan nilai dari model dengan kenaikan *training data*. Berikut adalah plot dari nilai ROC-AUC dan F1 pada model CatBoost dengan besarnya *split size* dari *training data*:

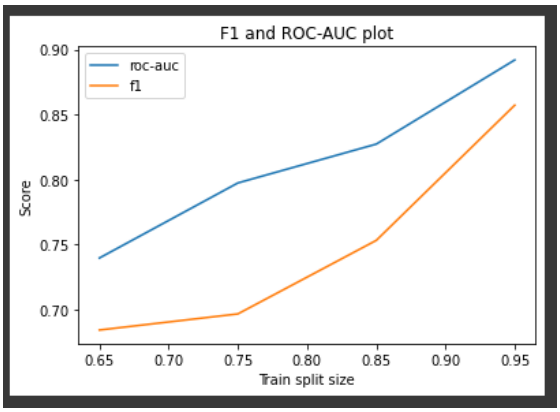


Figure 6: Plot antara *split size* dari *training data* dengan nilai F1 dan ROC-AUC dari model CatBoost

Bisa dilihat kedua nilai berbanding lurus dengan banyaknya *training data*. Sehingga, terlihat semakin banyak *training data* maka nilai dari model akan semakin tinggi. Ini kemungkinan terjadi karena sifat dari graf sendiri dan task *link prediction* kami. Jika sebuah model mempunyai informasi lebih banyak apakah suatu artis sudah pernah kolaborasi sebelumnya, maka tidak aneh kalau model akan semakin baik dalam memprediksi kolaborasi artis. Misalnya, jika dalam *training data* suatu artis diketahui belum pernah kolaborasi dengan siapapun (tidak ada edge), maka cukup sulit untuk memberikan prediksi yang akurat. Namun, jika di dalam *training data* sudah diketahui kalau artis tersebut pernah kolaborasi, maka model bisa lebih baik melakukan prediksi. Contohnya apabila artis 1 berkolaborasi dengan artis 2 dan artis 3, lalu artis 2 pernah berkolaborasi dengan artis 3. Jika *edge* dari artis 1 dihilangkan, maka ketika ingin memprediksi apakah pernah berkolaborasi dengan artis 2 maka bisa mengeluarkan prediksi kalau tidak berkolaborasi. Namun, jika terdapat informasi kalau artis 1 pernah berkolaborasi dengan artis 3, maka prediksi sebelumnya bisa mengeluarkan prediksi pernah berkolaborasi.

Lalu, kami juga melakukan plot terhadap *feature importance* dari model. Berikut adalah plot *feature importance* untuk model CatBoost.

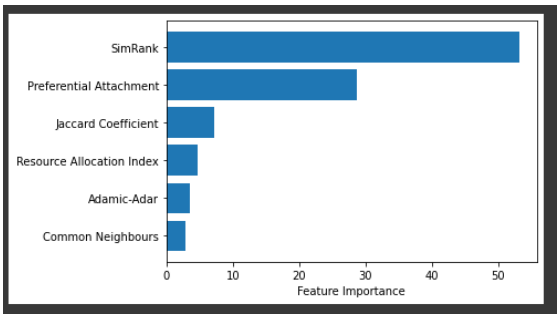


Figure 7: Plot nilai *feature importance* dari CatBoost

Bisa dilihat fitur *SimRank* mempunyai nilai *feature importance* paling tinggi dalam model, disusul oleh *Preferenial Attachment*. Ini menjadikan kedua fitur menjadi dua fitur paling penting dari model. Lalu, untuk ketiga fitur lainnya didapatkan kalau tidak terlalu penting, dengan nilai kurang dari 10%. Tetapi, ketiga fitur tersebut masih berkontribusi walaupun hanya sedikit.

4.2.2 *Node2Vec*

Kami mendapatkan hasil performa sebagai berikut untuk masing-masing model dan juga operator yang digunakan.

Model,Operator	F1	ROC-AUC
Logistic Regression, hadamard	0.6265513540674994	0.7216232637454931
Logistic Regression, average	0.4167929728503544	0.48268836882292254
Logistic Regression, l1	0.7642983469687616	**0.8034444520495153**
Logistic Regression, l2	**0.7745087619570554**	0.8084339485862806
Adaboost, hadamard	0.6210165161177968	0.7208089362543673
Adaboost, average	0.5813838293602005	0.673259166447409
Adaboost, l1	0.6044089210911686	0.7101829013713608
Adaboost, l2	0.6085467993533953	0.7126587984903953
Random Forest, hadamard	0.6148342849664875	0.719370772804701
Random Forest, average	0.6391426659140307	0.7213854881123576
Random Forest, l1	0.581677046299472	0.7011092174010681
Random Forest, l2	0.5839698194741403	0.7020039842313628
MLP, hadamard	0.6419760461242172	0.729793338455312
MLP, average	0.635363580899303	0.706455156389833
MLP, l1	0.722842870526693	0.7809454147688804
MLP, l2	0.759791422170063	0.8007150653916011

Terlihat bahwa performa F1 terbaik diraih oleh model logistic regression dengan operator l2. Sedangkan performa ROC-AUC terbaik diraih oleh model logistic regression dengan operator l1. Terlihat bahwa keduanya mempunyai perbedaan skor yang tidak terlalu signifikan. Secara keseluruhan operator l1 dan l2 menghasilkan nilai performa yang paling baik disusul oleh operator hadamard dan average. Hal yang menarik di sini adalah model *baseline* justru menghasilkan performa yang paling baik dibandingkan dengan model lainnya yang lebih kompleks. Selain itu terlihat bahwa model berbasiskan logistic regression yakni MLP dan logistic regression menghasilkan performa yang lebih baik dibandingkan dengan model *ensemble learning*. Hal ini menunjukkan bahwa pemilihan operator untuk node2vec bisa mempengaruhi performa klasifikasi.

Selanjutnya kami menganalisis *confusion matrix* untuk masing-masing model dengan menggunakan salah satu operator terbaik yakni L2.

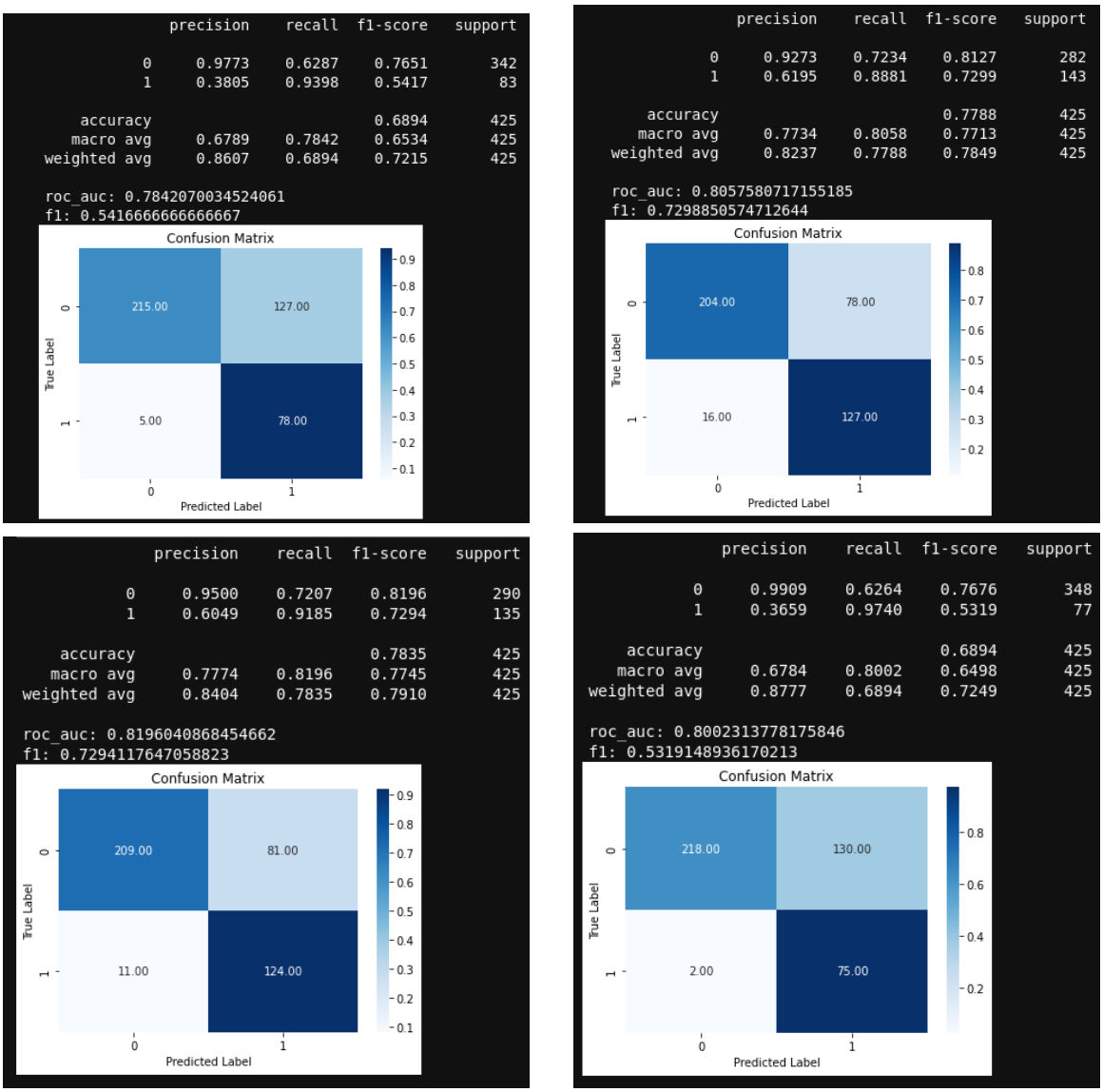


Figure 8: Confusion Matrix (dari kiri ke kanan & atas ke bawah, adaboost-logistic regression-mlp-random forest

Dari gambar terlihat bahwa semua model mempunyai fenomena yang seragam yakni cukup banyak terjadi *false positive*. Selain itu apabila dilihat skor untuk masing-masing labelnya, terjadi nilai *precision* dan *recall* yang saling berbanding terbalik. Pada label 1, terlihat bahwa nilai *recall* tinggi sedangkan *precision* menengah atau rendah. Lalu pada label 0, terlihat bahwa nilai *recall* menengah sedangkan *precision* tinggi. Dari *confusion matrix*

maka bisa disimpulkan bahwa model masih mempunyai kecendrungan untuk merekomendasikan *edge* dari pasangan verteks yang seharusnya tidak direkomendasikan. Di sisi lain, jumlah *false negative* rendah untuk semua model dan juga nilai jumlahan *true negative* dan *true positive* sehingga semua model menghasilkan performa yang lumayan baik dengan menggunakan fitur node2vec sebagai prediktor.

4.2.3 *Graph Neural Network with Node Features*

Untuk pendekatan GNN, kami menguji tiga macam *convolutional layer* yakni GCN, SAGE, dan GAT. Dari ketiga konvolusi diperoleh skor berikut. Terlihat bahwa *convolution layer*

Convolution Layer	F1	ROC-AUC	Precision	Recall
GCN	**0.7491638795986622**	**0.7693808532778356**	**0.7693808532778356**	**0.7491638795986622**
GAT	0.7341772151898733	0.6865244536940687	0.6865244536940687	0.7341772151898733
SAGE	0.7147766323024054	0.734261186264308	0.734261186264308	0.7147766323024054

GCN menghasilkan skor performa terbaik. Namun apabila diperhatikan, perbedaan skor ketiganya tidak terlalu berbeda jauh. Selain itu, pengujian dilakukan dengan pendekatan *train test split* biasa sehingga performa *convolutional layer* terbaik bisa saja bukan GCN pada *data splitting* selanjutnya. Kami tidak menggunakan pendekatan kfold karena kami belum menemukan cara melakukan *kfold validation* pada data yang berbentuk graf. Sedangkan metode sebelumnya menggunakan *kfold* pada data graf yang sudah berbentuk tabular sehingga mudah untuk dilakukan. Terlihat bahwa skor yang dihasilkan oleh *convolutional layer* terbaik tidak lebih baik dibandingkan dengan pendekatan model *machine learning* konvensional terbaik. Berikut merupakan *confusion matrix* untuk semua *confusion layer* yang kami ujikan.

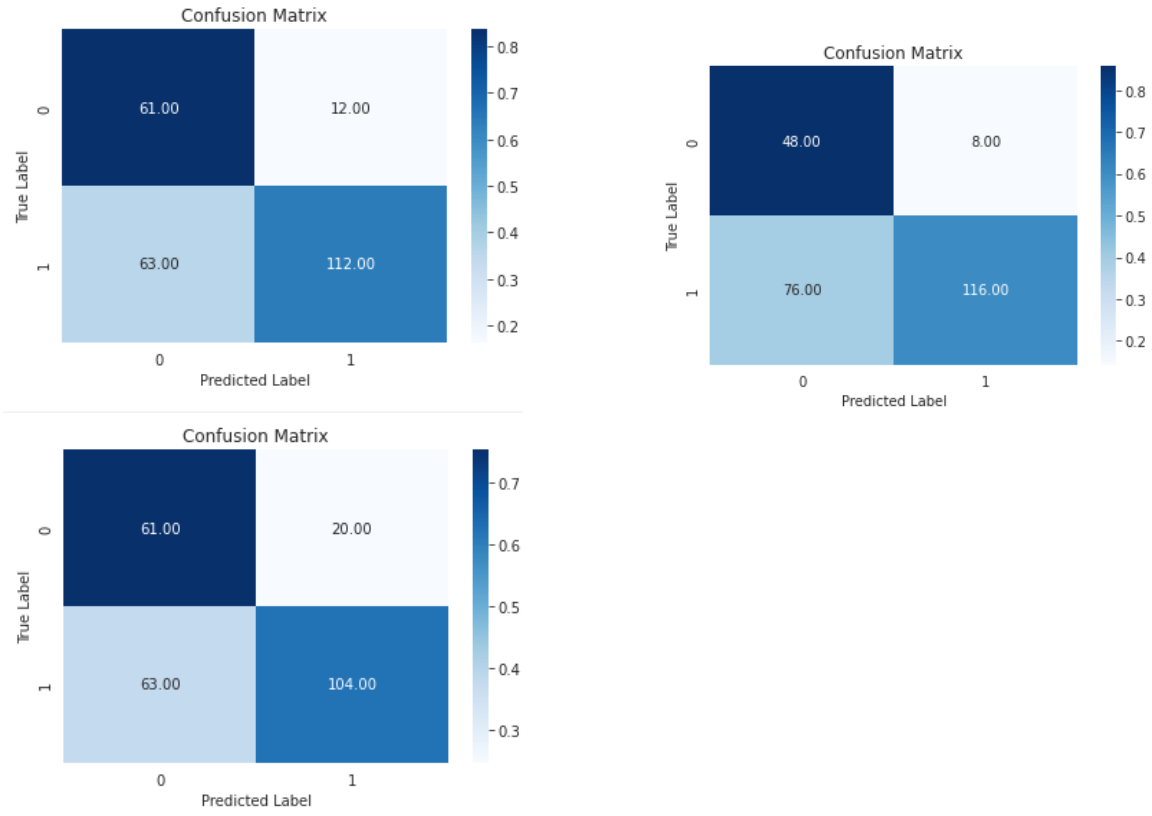


Figure 9: Confusion Matrix (dari kiri ke kanan & atas ke bawah, gcn-gat-sage)

Terlihat bahwa untuk pendekatan GNN mempunyai nilai *false negative* yang besar. Di sisi lain, nilai *false positive* untuk GNN kecil. Terlihat bahwa untuk beberapa *convolution layer*, nilai *true positive* lebih kecil dibandingkan dengan nilai *false negative* dan *true negative*. Ini menunjukkan bahwa pendekatan GNN yang kami gunakan mempunyai kecendrungan untuk memprediksi 0 pada data kami.

Selanjutnya kami menganalisis grafik *training score* dan *validation score* untuk mengecek potensi terjadinya *overfitting*.

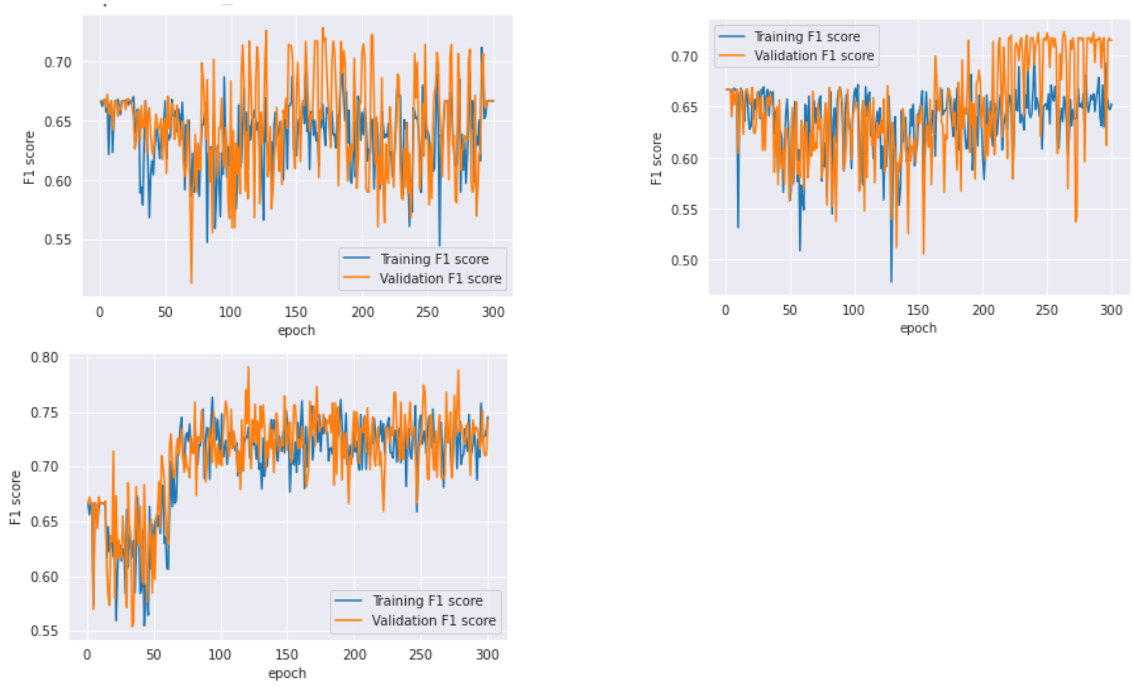


Figure 10: Grafik *training score* dan *validation score* (dari kiri ke kanan & atas ke bawah, gcn-gat-sage)

Terlihat pada grafik semua *convolution layer*, nilai *training score* cenderung tidak lebih tinggi dibandingkan dengan nilai *validation score*. Hal tersebut menunjukkan kemungkinan besar model GNN yang kami gunakan tidak mengalami *overfitting* berkat dengan penambahan *dropout layer* setelah *convolution layer*. Namun, model kami mengalami sedikit *underfitting* apabila dilihat dari skor *confusion matrix* dan *evaluation score* karena skor yang diperoleh masih kurang rendah atau sama dengan model *machine learning* konvensional. Apabila dilihat pada *trend*-nya, hampir semua *convolution layer* kecuali SAGE sudah konvergen pada akhir pelatihan.

4.3 Analisis

Dari hasil yang diperoleh di atas, terlihat bahwa semua metode yang kami gunakan mempunyai *f1-score* yang rata-rata berada di rentang 0.5-0.7. Sedangkan untuk skor *roc-auc score* berada di rentang 0.6-0.8. Hal tersebut merupakan hasil yang cukup baik karena skor prediksi yang dihasilkan sudah lebih baik daripada dilakukan *random guessing*. Lalu apabila diperhatikan pada *confusion matrix*, pendekatan *machine learning* konvensional mem-

punyai sifat prediksi label yang berkebalikan dibandingkan dengan pendekatan *graph neural network*. Pendekatan *machine learning* konvensional menghasilkan jumlah *false negative* yang kecil dan jumlah *false positive* yang besar. Di sisi lain pendekatan *graph neural network* justru menghasilkan *false negative* yang besar dan *false positive* yang kecil. Lalu apabila ditinjau pada skor evaluasinya, pendekatan *graph neural network* menghasilkan yang hampir mirip atau sama dengan pendekatan *machine learning* konvensional. Namun hampir semua variasi *graph neural network* berada di rentang 0.6-0.7 baik pada F1 maupun ROC-AUC sehingga perbedaannya tidak terlalu jauh sedangkan pada pendekatan *machine learning* konvensional, skor performa yang dihasilkan cukup beragam dari yang cukup buruk (0.4) hingga yang mempunyai performa terbaik (F1: 0.77, ROC-AUC: 0.8). Hal yang menarik adalah, *link-feature* yang digunakan justru menghasilkan performa yang paling baik model *machine learning* yang kompleks sedangkan fitur *node2vec* di sisi lain justru menghasilkan performa yang paling baik pada model *machine* yang sederhana. Eksperimen juga membuktikkan bahwa pemilihan *operator* yang tepat untuk node2vec bisa menentukan performa model.

5 Penutup

5.1 Kesimpulan

Setelah dilakukan eksperimen untuk mengkaji berbagai metode untuk *link prediction*, didapat bahwa secara umum semua metode menghasilkan skor performa yang cukup baik. Namun untuk pendekatan *graph neural network*, skor yang dihasilkan justru tidak lebih baik secara umum daripada pendekatan *machine learning* konvensional walaupun persebaran skornya lebih stabil dibandingkan dengan persebaran skor pada *machine learning* konvensional. Dari eksperimen kami, pendekatan terbaik adalah node2vec dengan operator L1 atau L2 dengan menggunakan *logistic regression* sebagai *classifier* dengan skor F1 0.76-0.77 dan ROC-AUC 0.8. Hal tersebut menunjukkan bahwa fitur *node2vec* sendiri cukup *powerful* untuk mengimbangi fitur-fitur *link-based*. Lalu untuk fitur *link-based*, diperoleh bahwa fitur simrank serta *preferential attachment* merupakan fitur yang cukup penting untuk menghasilkan prediksi yang cukup baik. Perlu diperhatikan bahwa eksperimen dilakukan pada data yang relatif sedikit yakni 600-700an data sehingga hasil yang kami peroleh belum bisa digeneralisasi.

5.2 Future Work

Dari eksperimen tersebut kami menyadari terdapat permasalahan-permasalahan yang terjadi yang akan dijadikan sebagai eksperimen selanjutnya dari jumlah data, *graph neural network* yang mengalami *underfitting*, serta limitasi pengetahuan kami mengenai *graph mining* terutama pendekatan *graph neural network*. Oleh karena itu berikut merupakan beberapa *future works* yang bisa dikembangkan dari eksperimen ini.

- Memperluas *scope* untuk artist yang diambil sehingga tidak hanya artist pop & rap saja.
- Menggunakan sumber data lain untuk pengambilan data seperti *knowledge base*.
- Menambahkan fitur-fitur data serta menghilangkan *noise* pada data yang sekiranya bisa memperbaiki performa *graph neural network* terutama memanfaatkan *node embedding* seperti node2vec sebagai fitur pada *graph neural network*.
- Melakukan *hyperparameter tuning* untuk menentukan parameter terbaik untuk semua *machine learning* model baik yang konvensional maupun *graph neural network*.

Daftar Pustaka

Leskovec, J. (2021). Machine learning with graphs course slides.

Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016*.