CSE5002 Project

> Name : 谢岳臻   SID : GTM11913008

Problem Statement

> Key Points : What is the problem to solve?

This project can be simplified to classify the nodes in the graph, which is mainly a multi-class classification problem in graph. Given the feature of each nodes, and the whole graph (the nodes are connected with edges), then we can predict the nodes' labels.

Data Processing

> Key Points : How do you process the data (including topology and attributes) before feeding to a classifier?

Given the feature of each nodes $X \in \mathbb{R}^{n\times 6}$, edges are expressed with a adjlist file. We can easily change the adjlist to adjacency matrix or edges.

Raw Features

The raw features contains six features, and each features can describe as followed

Describe\Index	degree	gender	major	second_major	dormitory	high_school
mean	1.440060	1.537474	9.067963	7.519351	208.175382	18515.809137
std	0.930884	0.548758	7.566342	10.132974	85.977516	18476.478081
min	1	0	0	0	0	0
max	6	2	44	44	290	61426

The minimize value 0 is the **absent value**, we need to full them. I choose to use the `SimpleImputer` in `sklearn`. Fit the learner in the training set, and use it both in training set and testing set.

```
sim = SimpleImputer(missing_values=0, strategy='mean')
sim.fit(data[:4000])
data = sim.transform(data)
```

It is easy to see that each feature is different, with the 5th feature having an extremely large mean and variance. If the unprocessed data is directly to the classifier, the 5th feature will play a large role in the discourse. Therefore, each feature needs to be **normalized** before input to the classifier.

In this project, I use the *StandardScaler* in *sklearn*.

$$z = \frac{(x-u)}{s}$$

However, **gender**, **major**, **second_major**, **school** should not compare with the values. So, we can use **one-hot** encoding to get feature. Since the **high school** has too many values, I only one-hot encoding **gender**, **major**, **second_major** three features. The feature after one-hot encoding will be $X \in \mathbb{R}^{n \times 94}$.

Feature	number of classes
degree	6
gender	3
major	43
second_major	44
dormitory	64
high_school	2505

In the experiment section, I will show the result both without one-hot encoding and with one-hot encoding.

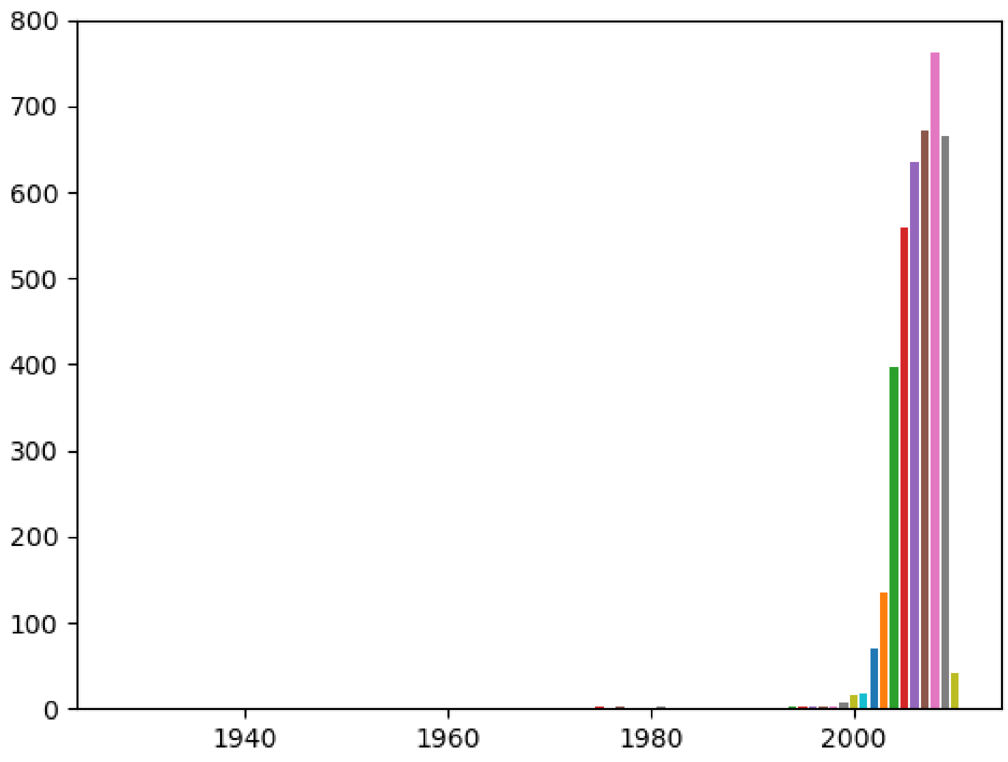
Edges

Transforming the adjlist into **edge** and **adjacency** matrix makes it easy to input to subsequent processing, mainly to facilitate input to the neural network.

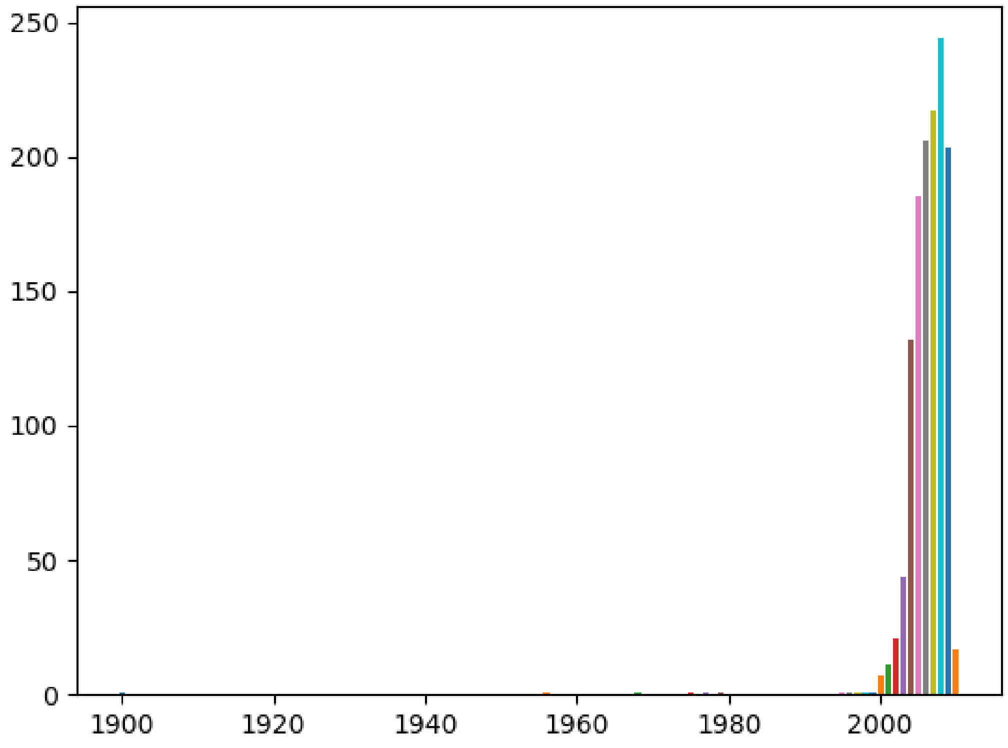
Labels

The distribution of the labels can be visualized in the following diagram:

Training set :



Testing set :



It is easy to see that the number of people in each category (year) is extremely unevenly distributed, so direct use of the classifier for classification may make the classifier more biased towards classes with a high number in the training set.

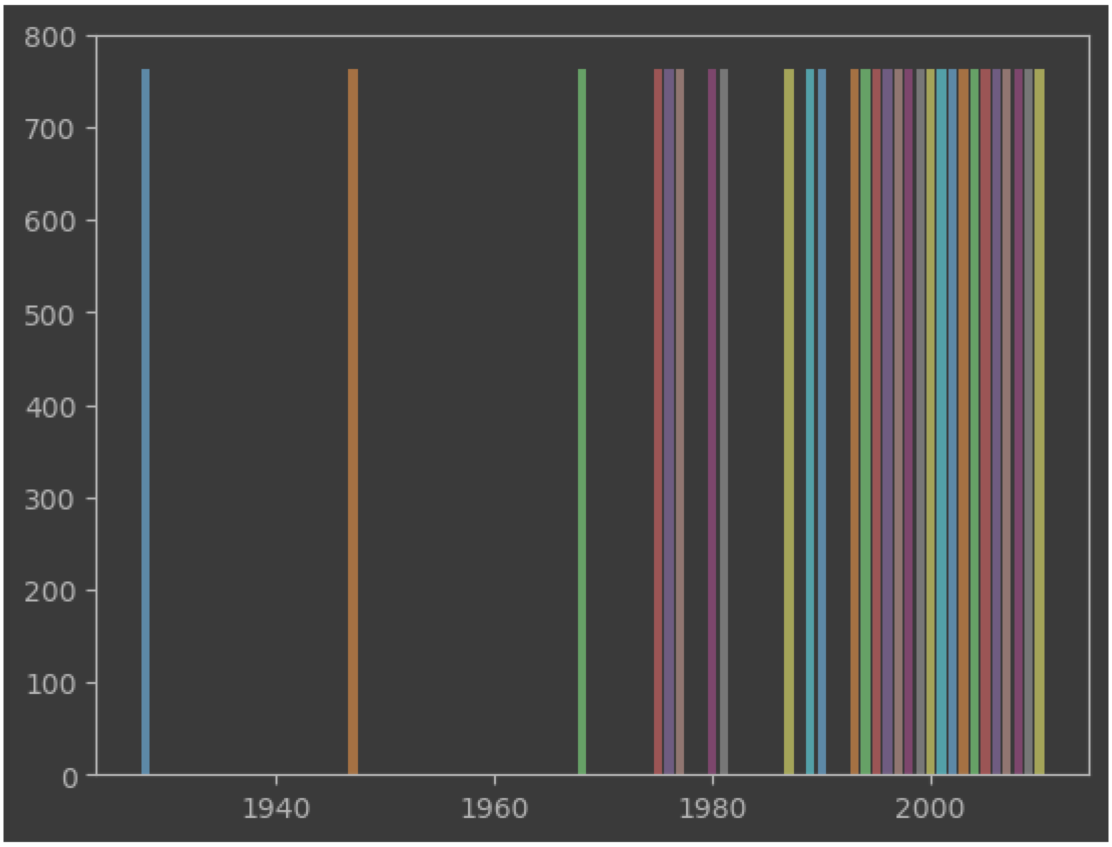
In addition to that, on the test set, years appearing that are not in the training set can be marked as error points, but they still have to be calculated when calculating the accuracy. The following years are not appeared in training set.

- 1900
- 1956
- 1979

Then, the years should be mapped into a sequences of number, instead of spares years data. Using `labels_uni, indices = np.unique(labels, return_inverse=True)` , we can get the unique labels and the `inv_list` to transform back to the years data. The mapping can see as followed.

Mapping Lables	Years	Mapping Lables	Years	Mapping Lables	Years
0	1928	5	1977	10	1990
1	1947	6	1980	11	1993
2	1968	7	1981	12	1994
3	1975	8	1987	13	1995
4	1976	9	1989	14....	....

Considering the extreme imbalance of samples and the fact that there are many classes with only one sample, the number of samples is balanced using **resampling**. This may leads to over-fitting.



Model

After careful consideration, the models used can be classified into the following three categories

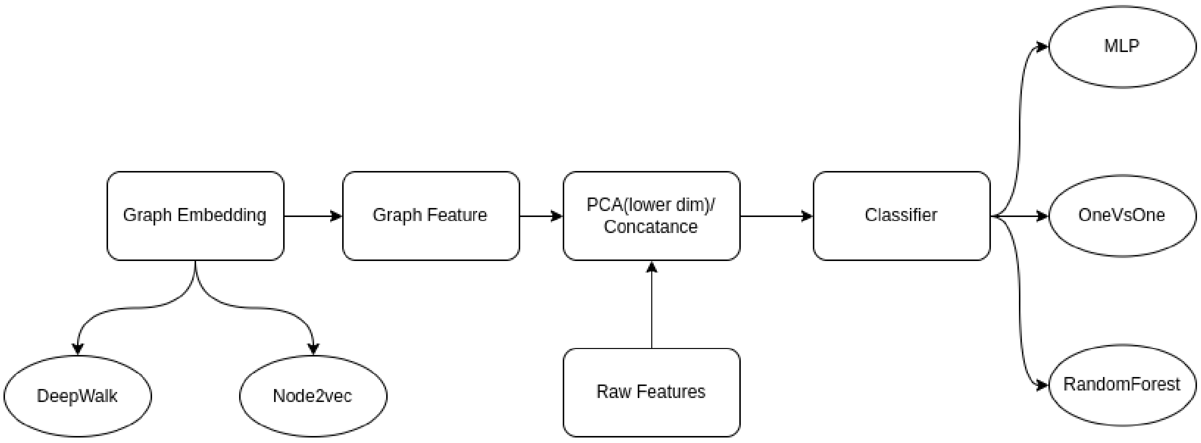
- Unsupervised Graph Embedding + Classifier
    - Two features : Graph Features, Raw Features
    - Combination ways :
        - Concatenate
        - Self-attention (weighted)
- Supervised Graph Embedding + Classifier
- GNN (**Graphic Nuaral Network**)

Unsupervised Graph Embedding + Classifier

A graph embedding determines a fixed length vector representation for each entity (usually nodes) in our graph. These embeddings are a lower dimensional representation of the graph and preserve the graph's topology. And the lower dimensional representation can combine with the given raw features, then pass them to the classifier to predict the labels.

Depending on the way of combining, the methods can be divided into **direct concatenation** and the use of **self-attention** mechanisms.

The whole pipeline can seen as followed :



- Graph Embedding
    - Lower dimensional representation of the graph
    - DeepWalk
        - Using short random walks to learn representations for vertices in graphs
    - Node2Vec
        - Extention of DeepWalk, combining with DFS and BFS, better than DeepWalk
- Combination
    - Fuse two features, A, B
    - Concatenate $[A, B]$
    - Self-attention $W \times [A, B]$
- PCA
    - Reducing the dimensionality of features
- Classifier
    - Predict the Labels
    - MLP

        - Fully connected class of feed-forward artificial neural network (ANN)
    - OneVsOne
        - Fitting one classifier per class pair
        - Change the multi-class classification to binary classification
    - RandomForest
        - Ensemble learning method => Bagging, a combination of strong learner
        - Operates by constructing a multitude of decision trees

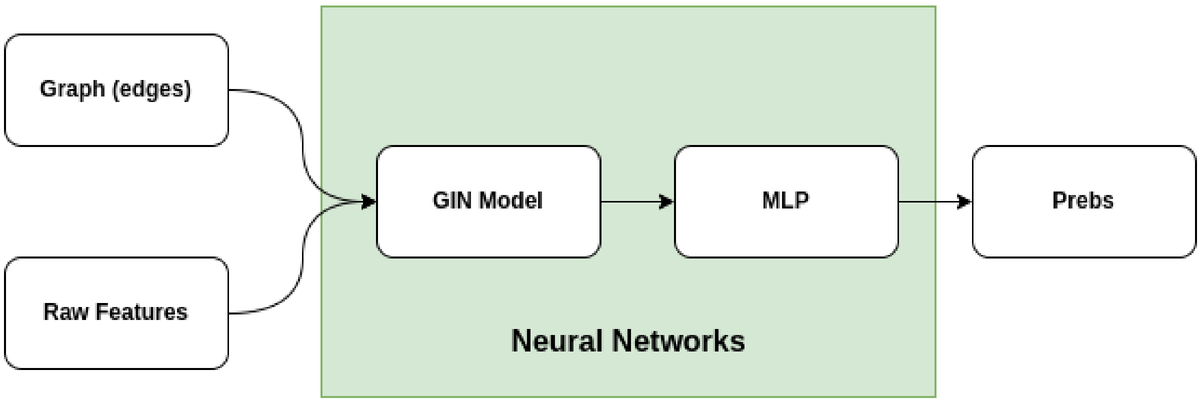
These methods can be combined in many ways to predict labels.

Supervised Graph Embedding + Classifier

Using neural networks to extract graph features, unlike unsupervised graph embedding, this can be trained as a layer in a neural network. In this project, the **GIN model** is used for exploration.

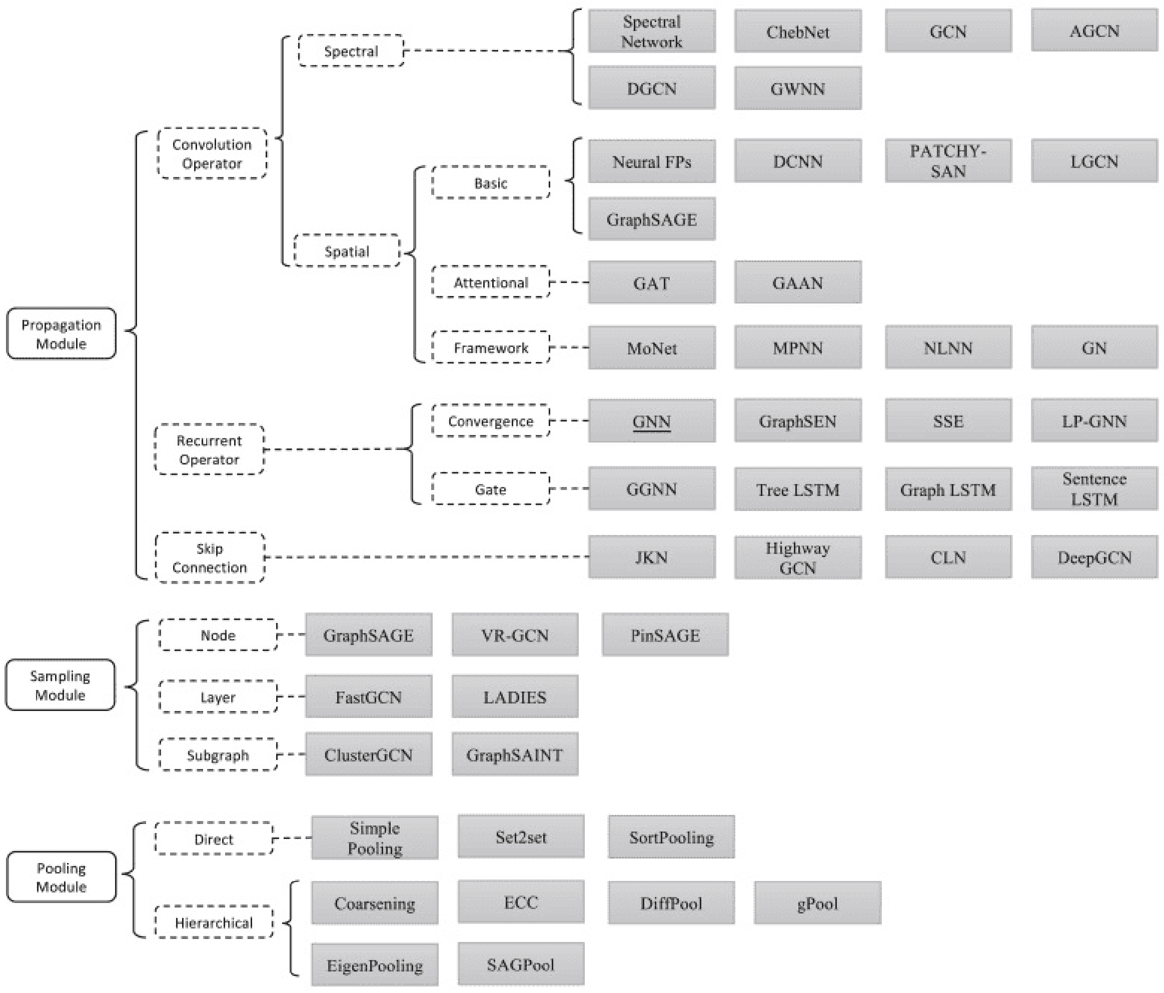
> Graph Isomorphism Network (GIN) generalizes the WL test and hence achieves maximum discriminative power among GNNs.

After the layer of GIN, we can simply use the MLP to predict the labels and compare with the unsupervised graph embedding ways. The whole pipeline :



GNN (Graphic Nuaral Network)

GNN has various models, I have select three classical GNN to test, **GAT**, **GCN** and **GraphSAGE**.



GCN

Graph Convolutional Networks (GCN) is the most cited paper in the GNN literature and the most commonly used architecture in real-life applications.

GCN is the first to propose a **convolutional approach** to fusion of graph structure features, providing a new perspective. GCN is computationally well understood, essentially the same as the CNN convolution process, a weighted summation process, that is, the neighbor points through the degree matrix and its adjacency matrix, calculate the weight of each edge, and then weighted summation.

GAT

GCN does convolution, and the weights on the edges are fixed every time when need to do fusion. Add an attention mechanisms to the fusion and let the model learn by itself, this is what GAT doing.

It can be said that, **GAT** is the extension of **GCN**.

GraphSAGE

GCN does convolutional fusion based on the full graph, and the gradient is updated based on the full graph. If the graph is large and there are more neighboring nodes at each point, the efficiency of such fusion is inevitably very low.

> **Transductive Learning VS. Inductive Learning**

> Transductive learning : means that the data to be predicted **can be seen** by the model during training. To explain further, it means that before training, the structure of the graph is already fixed, and the structure of the points or edge relationships you want to predict should already be in this graph.
>
> Inductive learning : means that the data to be predicted is **not seen** by the model during training, which is how we normally do algorithmic models, and the data is separated during training and prediction, which means that the graph structure can be not fixed as mentioned above, and new nodes are added.

GraphSAGE is a model of inductive learning. GraphSAGE proposes to randomly pick subgraphs to sample and update node embedding through subgraphs, so that the structure of the picked subgraphs itself is changed, and thus the model learns a way of sampling and aggregating parameters, which effectively solves the unseen nodes This effectively solves the problem of unseen nodes and avoids the dilemma of updating the node embedding of the whole graph together during training, which effectively increases the scalability.

It can be said that, **GAT** is the extension of **GCN**.

Therefore, it can be expected that the performance of GAT and GraphSAGE is better than that of GCN.

Final Model

I choose the **GraphSAGE**, because

- In the comparison experiment, it has the second best performance, only lower than Node2vec + OVO
- Simple and effective, speed much faster than Node2vec + OVO
- My roommate has already helped me to realize it, and it is very simple to start experimenting...

Evaluate Metric

Accuracy

Accuracy can be used to measure the accuracy of a model, a very general but very common indicator of the model's performance

$$ACC = \frac{\sum \mathbb{I}(y = \hat{y})}{|y|}$$

Cross Entropy Loss

It is mainly used to measure the difference between two probability distributions, and is a **loss function** commonly used in classification problems.

$$Loss = -\hat{y}\log(\mathrm{softmax}(y))$$

Negative Log-Likelihood Loss

Minimize the difference between the model prediction results and the true label to make the model's prediction results closer to the true results. Similar to the Cross Entropy Loss

$$\text{CrossEntropyLoss}(x, y) = \text{NLL\_Loss}(\log \text{Softmax}(x), \text{y})$$

Experiments Details

Environment Setup

All the experiments are based on *python* , mainly uses *pytorch*, *sklearn* and *pytorch-geometric*. The experiments are performed both on my personal computer with AMD Ryzen 5 3600X (12) @ 3.800GHz and NVIDIA GeForce RTX 2060 SUPER, and a server with Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz and 6 TITAN V.

All the requirement libraries are in README.md in code folds, you can also check the requirement on github repo.

DeepWalk

Given the following parameters

```
number_walks = 8
representation_size = 128
walk_length = 40
window_size = 10
```

Node2Vec

Given the following parameters

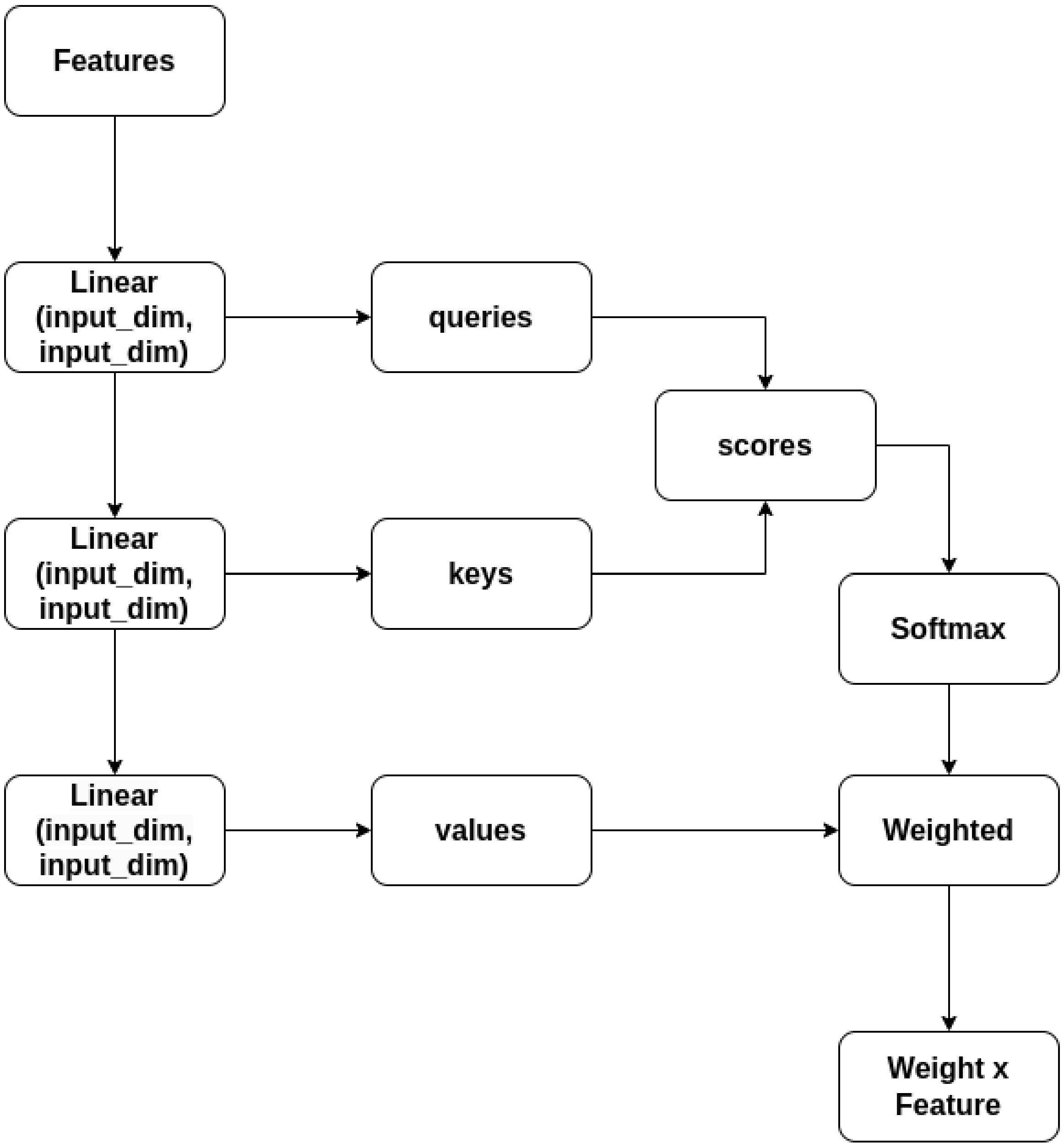
```
dimensions=128
walk_length=30
num_walks=200
window=10
```

Self-Attention

The Self-Attention module takes an input tensor `x` with shape `(batch_size, seq_length, input_dim)` and returns a weighted representation of the input sequence with the same form.

The attention mechanism is implemented using **dot-product attention**, where the query, key, and value vectors are learned through linear transformations of the input sequence.

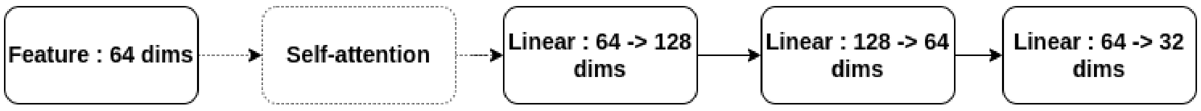
The attention scores are then calculated as the dot product of the queries and keys, and the attention is applied by **multiplying the values** by the attention scores. The result is a weighted representation of the input sequence that considers each element’s importance.


Features
Linear (input_dim, input_dim)
queries
scores
Linear (input_dim, input_dim)
keys
Softmax
Linear (input_dim, input_dim)
values
Weighted
Weight x Feature

Input_dim is set to 64.

MLP

MLP with three layers, the dimensions are 128, 64, 32 respectively.



The final output of the 32-dimensional vector represents the likelihood of the 32 classes.

MLP uses Cross Entropy Loss to train.

OneVsOne(OVO)

Each classifier is `SVC` , SVC classification is a nonlinear classifier that separates different classes of data by constructing a nonlinear decision boundary. And can use `C` parameter, to prevent overfitting.

Using `CalibratedClassifierCV` to cross-validation the training process

```
clf = SVC(C=1)
clf = CalibratedClassifierCV(clf, method='sigmoid')
clf = OneVsOneClassifier(clf)
clf.fit(X_train, Y_train)
```

RandomForest

Also use cross-validation to train the model

```
acc = cross_val_score(estimator=RandomForestClassifier(n_estimators=100),
X=X_train, y=Y_train, cv=10)
print("average accuracy :", np.mean(acc))
print("average std :", np.std(acc))
```

GIN

Using GIN + MLP to predict, and use `CrossEntropyLoss` totrain the whole model

```
self.gnn = GIN(in_channels, hidden_channels, num_layers, dropout=dropout,
jk='cat')
self.classifier = MLP([hidden_channels, hidden_channels, out_channels],
norm="batch_norm", dropout=dropout)
```

Using Adam optimizer, learning rate set to `0.01`, weight_decay set to `5e-4`.

GCN, GAT, GraphSAGE

All using Adam optimizer, learning rate set to `0.01`, weight_decay set to `5e-4`. Using `Negative Log-Likelihood Loss` to train the whole model

GCN

Using two layers Graph Convolution Layers.

```
self.conv1 = GraphConvolution(nfeat, nhid)
self.conv2 = GraphConvolution(nhid, nclass)
```

GAT

Using two layers GAT Convolution Layers.

```
self.gat1 = GATConv(feature, hidden, heads=heads)
self.gat2 = GATConv(hidden*heads, classes)
```

GraphSAGE

Using exist code.

Experiment Result

> Due to the limitation of time, all the experiment is test only once, maybe random factors interfering. (Guarantee that the error is within a certain range)

Unsupervised Graph Embedding + Classifier

Final Result shows in the table(**fill the absent values, Without one-hot encoding, Without PCA**):

	DeepWalk + MLP	**DeepWalk + MLP + Self-attention**	**DeepWalk + OVO**	**DeepWalk + RandomForest**
Training Acc (Best)	0.9980541225450267	0.9979183636528193	0.9937098379943886	0.9726236969698212
Testing Acc	0.7958397534668722	0.7719568567026195	0.8166409861325116	0.7334360554699538
F1-Score macro	0.36652479405515703	0.349767461882951	0.3683636207761497	/
F1-Score micro	0.7958397534668721	0.7719568567026194	0.8166409861325117	/
F1-Score weighted	0.7948680343657579	0.7683764252376482	0.8108238228750033	/

	Word2vec + MLP	**Word2vec + MLP + Self-attention**	**Word2vec + OVO**	**Word2vec + RandomForest**
Training Acc (Best)	0.9690469725767038	0.9992759525748937	0.9936645850303195	0.9774654283484472
Testing Acc	0.8066255778120185	0.47226502311248075	**0.8451463790446841**	0.7773497688751926
F1-Score macro	0.3421681816754324	0.21645115198706744	0.38721768837829007	/
F1-Score micro	0.8066255778120185	0.47226502311248075	0.845146379044684	/
F1-Score weighted	0.8058452844859023	0.5625662668107738	0.8417633226943736	/

With PCA(only test a few model):

	DeepWalk + MLP	**DeepWalk + MLP + Self-attention**	**DeepWalk + OVO**	**DeepWalk + RandomForest**
Training Acc (Best)	0.9961987510181917	0.9979183636528193	0.9913114308987239	0.9723975550452796
Testing Acc	0.7773497688751926	0.7318952234206472	0.8020030816640986	0.7372881355932204
F1-Score macro	0.3354470491354979	0.32459283449645937	0.35432221301417494	/
F1-Score micro	0.7773497688751926	0.7318952234206472	0.8020030816640985	/
F1-Score weighted	0.7759898313239914	0.7330884970859773	0.796546634589864	/

Without filling the absent values(only test a few model):

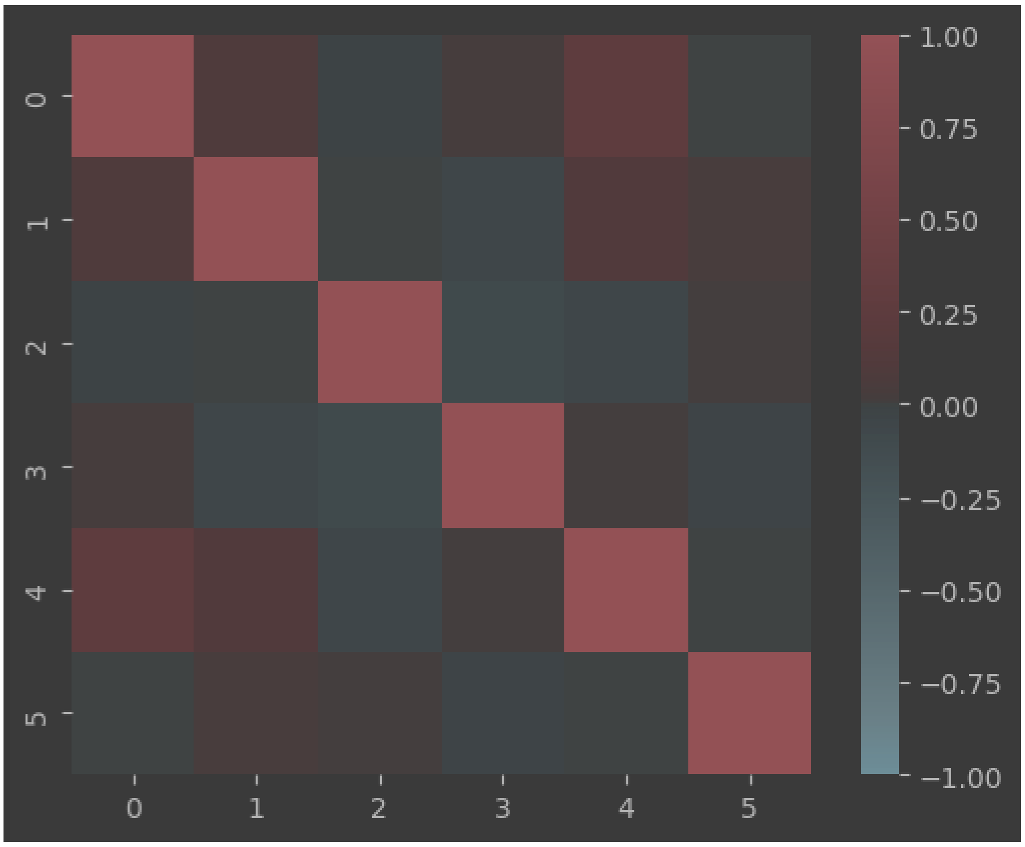
	DeepWalk + MLP	DeepWalk + OVO	DeepWalk + RandomForest
Training Acc (Best)	0.9964250158385374	0.9941171146710109	0.9734835278959583
Testing Acc	0.7827426810477658	0.8181818181818182	0.7295839753466872
F1-Score macro	0.33643382340403954	0.3500109165059251	/
F1-Score micro	0.7827426810477658	0.8181818181818182	/
F1-Score weighted	0.7815303566710782	0.8120838047769431	/

With the one-hot encoding(only test a few model):

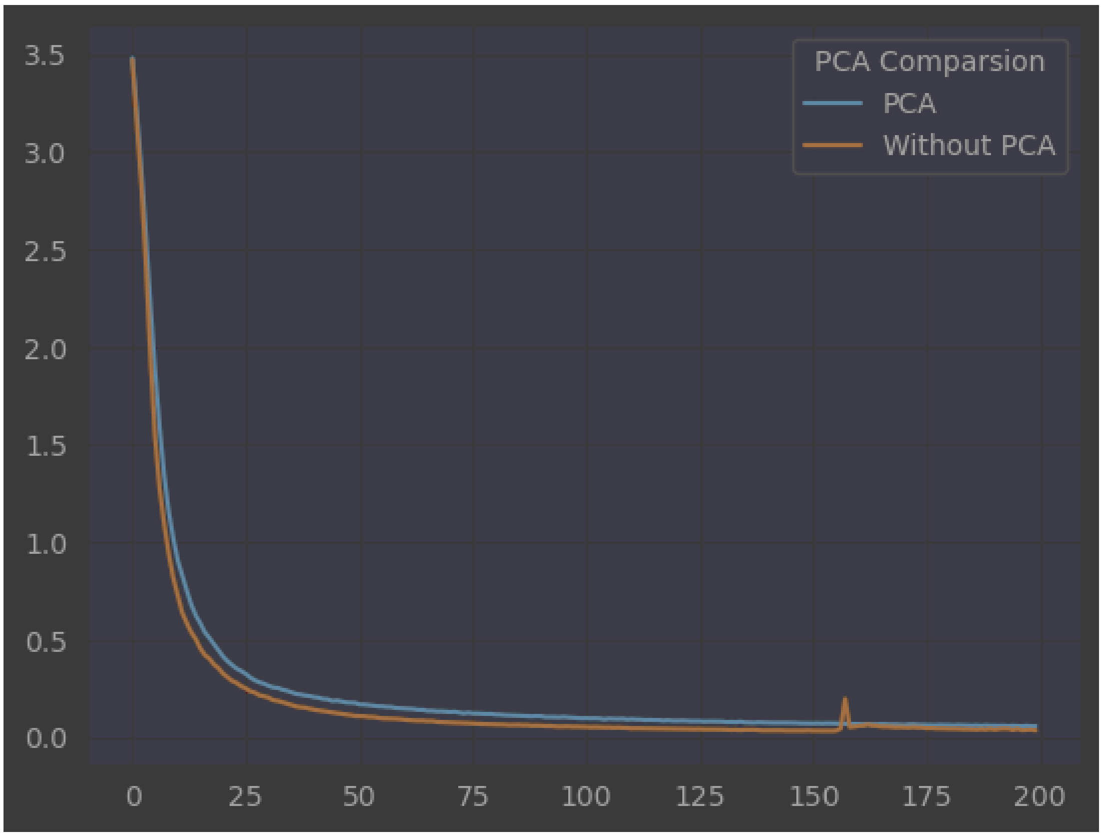
	DeepWalk + MLP	DeepWalk + OVO	DeepWalk + RandomForest
Training Acc (Best)	0.9962440039822609	0.9933478142818355	0.9730762266253439
Testing Acc	0.7673343605546996	0.7896764252696457	0.7380585516178737
F1-Score macro	0.3451645347878547	0.32217034079367596	/
F1-Score micro	0.7673343605546995	0.7896764252696455	/
F1-Score weighted	0.7653331867218369	0.7826998698150947	/

> Why PCA doesn't work well here?

PCA **cannot** improve the accuracy if using the raw feature, which only has 6 features. The correlation matrix also shows that each feature is already conditional independent.



Directly use PCA will loss some features, decrease the accuracy. It's clear in the following graph, PCA and without PCA converge almost the same times, however, the loss of the model without PCA lower.



Supervised Graph Embedding + Classifier

	GIN + MLP
Training Acc (Best)	0.94
Testing Acc	0.7326656394453005
F1-Score macro	0.2675807840727791
F1-Score micro	0.7326656394453005
F1-Score weighted	0.7301604203236546

GNN (Graphic Nuaral Network)

	GCN	GAT	GraphSAGE
Training Acc (Best)	0.71475	0.83925	0.99
Testing Acc	0.6856702619414484	0.7604006163328197	0.83
F1-Score macro	0.2506670435538773	0.3445996457775545	/
F1-Score micro	0.6856702619414484	0.7604006163328197	/
F1-Score weighted	0.667507186945356	0.7542515091665348	/

Limitation and Future work

Limitation :

- GNN only test three classical networks, not test SOTA
- Resampling will cause over-fitting, and cannot use validation set while training (resampling over the training set)
- Labels in datasets should performance better after one-hot encoding, need to do specific optimization

Future work :

- Testing more models
- Using another ways to balance the datasets, for example, using cost-matrix.

Conclusion

Using PCA, one-hot encoding, and filling missing values does not result in a performance gain. In graph embedding using unsupervised learning (Node2vec), the learners using OneVSOne classifiers perform the best, achieving 84.5% accuracy on the test set. Among graph embeddings with supervised learning and GNN, GraphSAGE performs the best, achieving 83% accuracy. Although not as accurate as unsupervised learning, it has a speed advantage. Training Node2vec and OneVSOne usually takes about 20 min, while GraphSAGE takes only about 5 min.

Therefore, using GraphSAGE is the best choice.

Ablation Study

> Conduct experiments to discuss whether using both two sources of information is better than using a single source of information.

Only used the node features, not using the topology data.

Just not concatenate with the graph feature, directly pass the features into classifier.

	MLP	**OVO**	**RandomForest**
Training Acc (Best)	0.8435605032129604	0.8093492623766857	0.9083668005628965
Testing Acc	0.24807395993836673	0.26194144838212635	0.29044684129429893
F1-Score macro	0.06441342142312516	0.0698234377839078	/
F1-Score micro	0.24807395993836673	0.26194144838212635	/
F1-Score weighted	0.25729241882084886	0.26743419838739074	/

It's easy to see that although the accuracy in training set reach a high accuracy, it cannot work well in testing set. And all the model performance worse than that with the graph data.