Pusan National University
Computer Science and Engineering
Technical Report 2024-10

질병 연관 유전자 발굴을 위한 머신러닝 기법 설계

202155534 김이경

201913123 박화성

202055650 이윤재

지도교수 송길태

목 차

1. 서론 1

1.1. 연구 배경 1

1.2. 연구 목표 2

2. 연구 배경 3

2.1. 데이터 선정 3

2.2. 데이터 분석 3

2.3. 데이터 시각화 5

2.4. GCN 모델 7

2.5. 이기종 그래프 8

2.6. Pytorch Geometric 8

3. 연구 내용 9

3.1. 데이터 전처리 9

3.2. 모델 구성 13

3.3. 모델 고도화 14

4. 연구 결과 분석 및 평가 17

4.1. 학습 및 평가 결과 17

4.2. 시연 18

5. 결론 및 향후 연구 방향 19

6. 과제 일정 및 역할 20

7. 참고 문헌 21

1. 서론

1.1. 연구 배경

인간의 유전자에서 질병과의 관계를 파악하는 것은 질병의 발병가능성을 예측하고 초기에 대응하기 위해 중요한 생물학적 문제이다. 여러 임상 실험을 통해 유전자와 질병 간의 관계를 파악하거나 유전자 데이터 분석을 통해 유전자와 질병의 관계를 밝혀내기도 한다. 하지만 한 유전자가 여러 질병에 걸쳐 발현되거나 환경적 요인으로 인한 질병은 관계를 찾아내기 어렵다. 또한, 인간의 전체 유전자 수 중 확인된 유전자가 한정되어 있기 때문에 직접적으로 연관성을 찾아내는 방식은 한계가 존재한다. 이를 해결하기 비교적 최근 연구에서는 유전자 간의 유사성을 통해 질병과의 연관을 예측하는 방식을 도입했다. 즉, 기존에 알려진 질병 유전자와 유사한 특성을 가지는 후보 유전자는 질병과 관련될 가능성이 크다는 것에 기반에 유전자와 질병의 새로운 연관성을 찾아낼 수 있다.

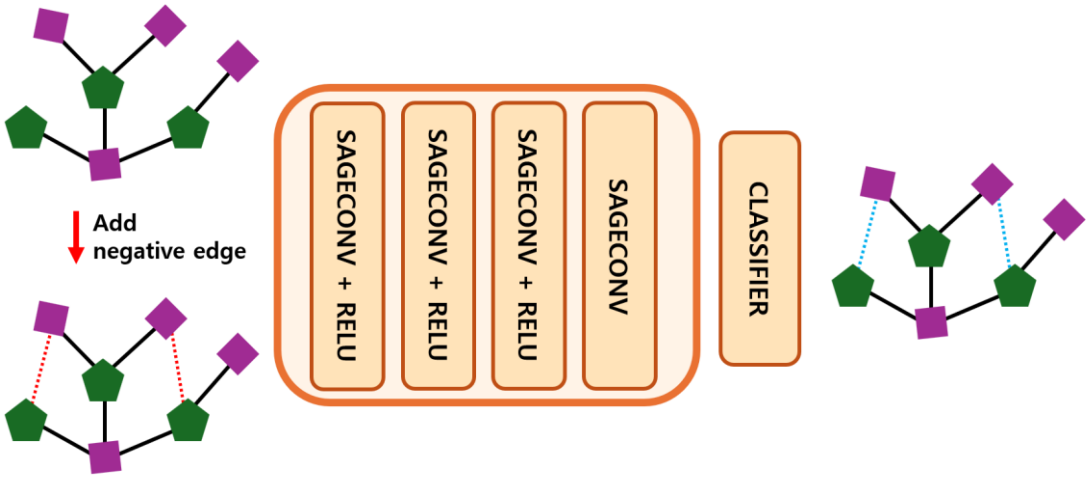
질병 연관 유전자를 예측하기 위한 모델로 Collaborative Filtering, Matrix Factorization, Graph Convolution Network를 사용하는 경우를 볼 수 있었다. 이러한 모델을 통해 질병과 유전자 간의 새로운 연관관계를 예측해 질병-유전자 연관성 연구의 효율 상승과 의료기술의 발전을 이룰 수 있을 것이다.

Credit: Zayna Sheikh, Broad Communications

1.2. 연구 목표

본 과제에서는 DisGeNET의 데이터베이스를 통해 질병과 유전자, gda를 수집하고 이를 바탕으로 이기종 그래프(Heterogenous Graph)를 구성한다. 이후, GCN 기법을 적용해 모델을 생성하여 질병과 유전자의 관계를 학습하고 하이퍼 파라미터의 조정, 최적화 기법 적용 등 모델 고도화를 위해 다양한 노력을 수행해 최적화된 모델을 도출해 내는 것을 목표로 한다.

모델의 전체적인 구조는 아래와 같다.



2. 연구 배경

2.1. 데이터 선정

DisGeNET은 인간 질병과 관련된 유전자에 대한 정보를 제공하며 인간 질병의 기초적 조사, 질병 유전자 분석, 약물 치료 효과 등 다양한 의료 연구 목적으로 사용되는 플랫폼이다. DisGeNET에서는 21,671개의 유전자와 30,170개의 질병, 장애, 비정상적 표현 간의 연관성 데이터와 유전자 변이 와 질병 간의 연관성 데이터를 제공한다. 본 과제에서는 질병과 유전자의 연관성 예측을 위해 DisGeNET에서 제공하는 질병 데이터, 유전자 데이터, gda(유전자-질병 관계) 데이터를 사용한다.

2.2. 데이터 분석

(1) disease association - 질병과 관련된 여러 가지 특징들을 제공해 주는 데이터

diseaseId	- UMLS[1]에 의해 질병마다 부여된 Id - 각 질병은 자신의 고유한 Id 값을 가짐
diseaseName	- 질병마다 부여된 질병의 이름으로 종복을 허용하지 않음 - 해당 데이터는 30,170종류의 질병을 포함
diseaseType	- 질병을 3가지 타입으로 분류 - 각 질병은 한 가지의 타입으로만 분류됨 - disease / phenotype / group
diseaseClass	- MeSH[2]에 의해 부여된 질병 클래스 - 각 질병은 복수의 질병 클래스를 가질 수 있음 - 해당 데이터에는 26개의 class가 존재
diseaseSemanticType	- UMLS에 의해 부여된 질병의 의미 유형 - 각 질병은 하나의 의미 유형을 가짐 - 해당 데이터에는 33개의 유형이 존재
NofGenes	- 질병과 관련된 유전자의 수
NofPmids	- 질병과 관련된 출판물의 수

[1] Unified Medical Language Systems

[2] Medical Subject Headings

(2) gene association - 유전자와 관련된 여러 가지 특징들을 제공해 주는 데이터

geneId	- NCBI[3]에 의해 부여된 유전자의 ID - 각 유전자는 고유한 유전자 ID를 가짐 - 해당 데이터는 21,671개의 질병을 포함
geneSymbol	- 유전자 기호로 유전자를 약어로 표기 - 유전자는 하나의 유전자 기호만을 가짐 - 동일한 유전자를 갖는 서로 다른 유전자가 존재 가능 - 해당 데이터는 21,666개의 유전자 기호가 존재
DSI	- 유전자의 질병 특이성 지수(Disease Specificity Index) - 0~1 값을 가지며 관련된 질병의 수가 많을수록 낮은 값을 가짐
DPI	- 유전자에 대한 다면 발현 지수(Disease Pleiotropy Index) - 0~1 값을 가지며 관련된 질병의 수가 많을수록 높은 값을 가짐
PLI	- 유전자의 기능 상실 불내증 확률
protein_class	- DTO[4]에 따른 단백질 분류 - 해당 데이터에는 21개의 단백질 클래스가 존재
protein_class_name	- 단백질 클래스에 부여된 단백질 클래스의 이름
NofDiseases	- 유전자와 관련된 질병의 수
NofPmids	- 유전자와 관련된 출판물의 수

(3) gene disease association - 유전자와 관련된 질병들을 나타내는 데이터

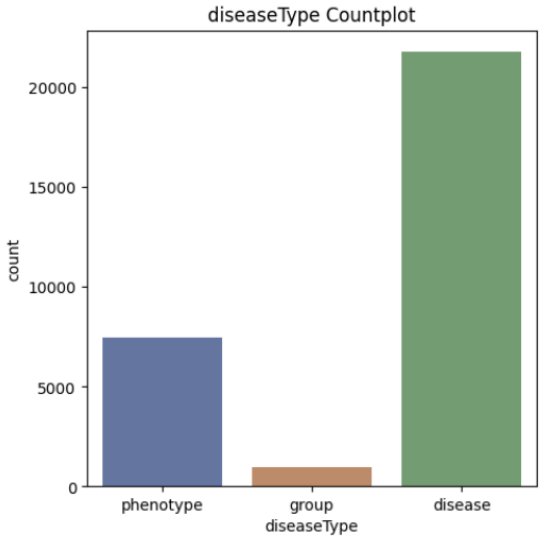
gda	- DisGeNET에서 제공하는 gda 관련 링크
gene	- NCBI에서 제공하는 유전자에 대한 정보
geneSymbol	- 유전자에 해당하는 유전자 기호
disease	- Linked Life Data에서 제공하는 질병에 대한 정보
diseaseName	- 질병의 이름

[3] National Center for Biotechnology Information

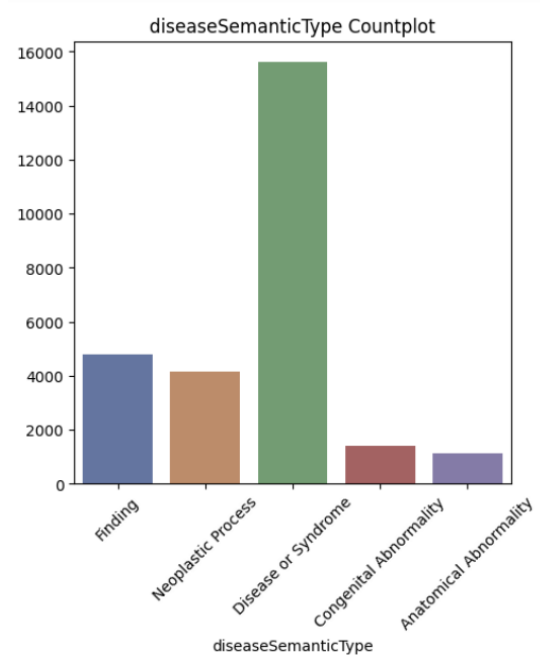
[4] Drug Target Ontology

2.3. 데이터 시각화

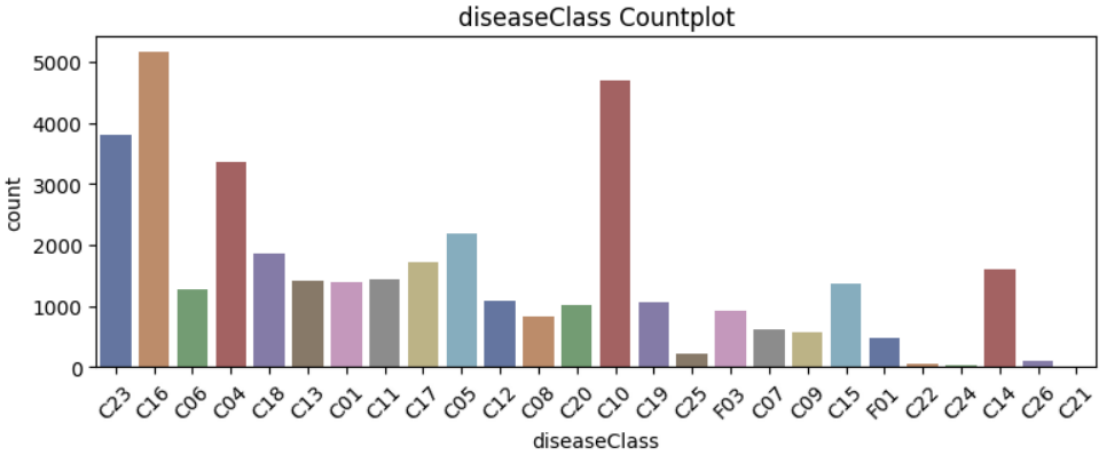
(1) **disease association** - 질병과 관련된 여러 가지 특징들을 제공해 주는 데이터



- diseaseType에 대한 Countplot[5]
- 대부분 질병의 유형은 disease임을 알 수 있다.



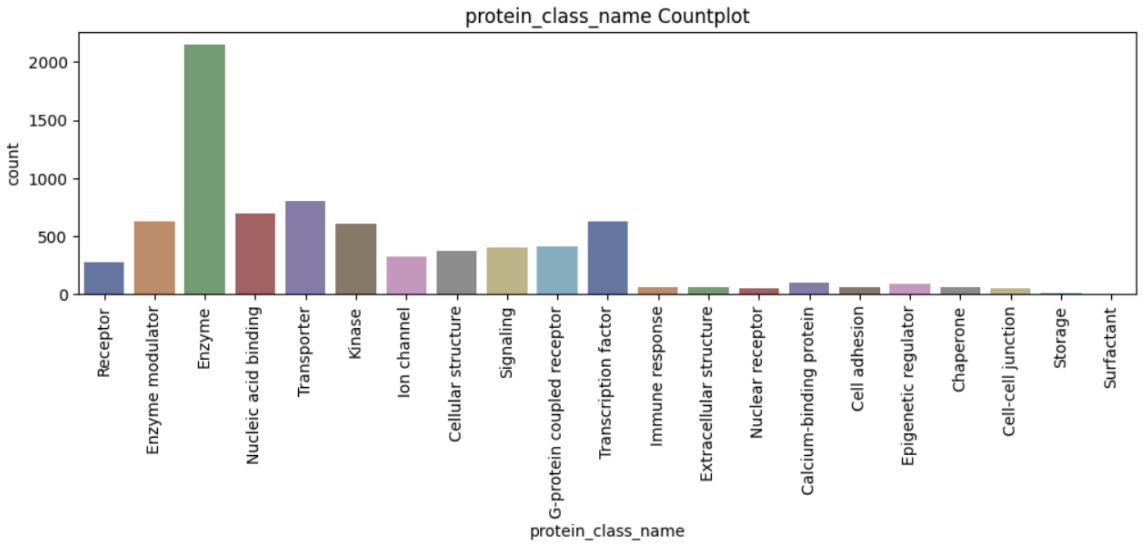
- diseaseSemanticType에 대한 countplot
- 상위 5개의 diseaseSemanticType만을 시각화



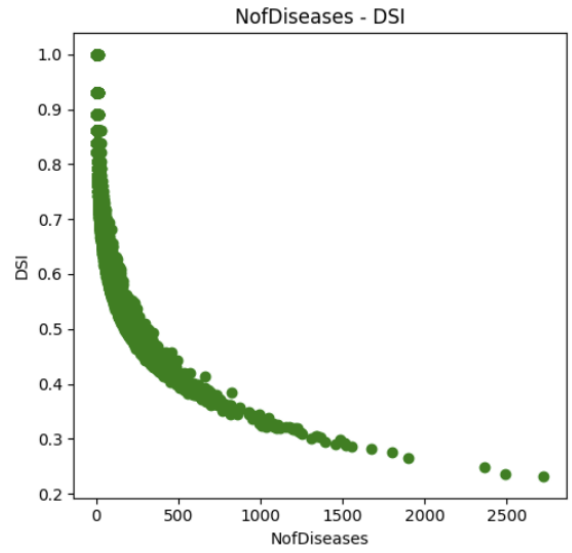
질병은 26개의 disease class로 분류되며, C16, C10, C04, C23에 해당하는 질병이 가장 많음을 알 수 있다.

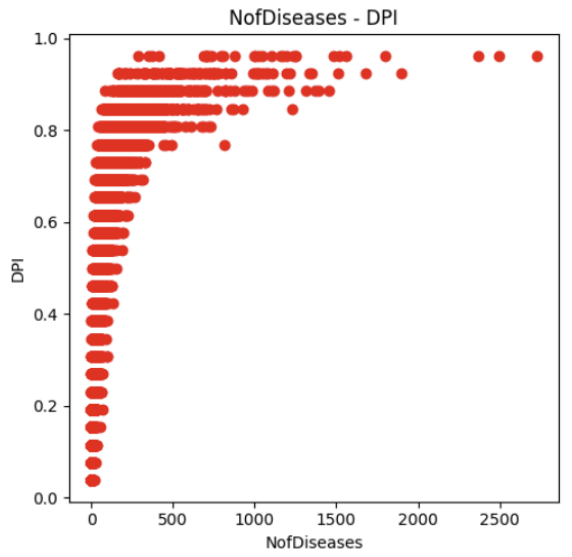
[5] 범주형 변수의 빈도수를 시각화

(2) **gene association** - 유전자와 관련된 여러 가지 특징들을 제공해 주는 데이터



유전자는 21개의 protein class로 분류되며 Enzyme protein class를 갖는 유전자가 가장 많음을 알 수 있다.



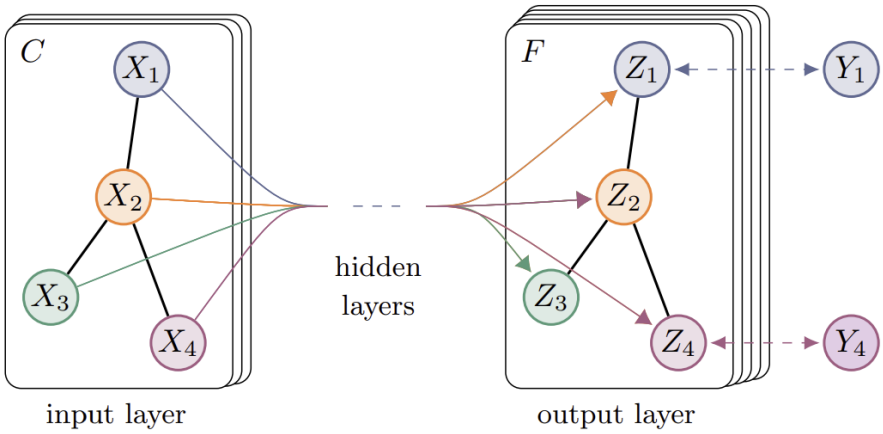


- NofDisease - DSI
- NofDisease가 큰 유전자는 낮은(0에 가까운)DSI 값을 갖는다.

- NofDiseases - DPI
- NofDisease가 큰 유전자는 큰(1에 가까운) DPI 값을 갖는다.

완벽한 비례-반비례 관계를 맺는 것은 아니지만 어느 정도 관계가 있음을 알 수 있다.

2.4. GCN 모델



GCN(Graph Convolution Network)는 Graph 형태의 데이터에서 합성곱을 통해 유의미한 정보를 뽑아내는 신경망 모델이다. GCN는 그래프 구조를 통해 각 노드의 특징을 효과적으로 학습하며, 주로 반지도 학습(Semi-supervised learning)에 사용된다. GCN의 입력은 다음과 같은 행렬들로 구성된다.

1) Feature matrix

모든 노드의 피처를 모아둔 행렬이다. $X \in R^{NxF}$ 형태로 N은 노드의 수, F는 각 노드의 특징 수를 나타낸다. 따라서 하나의 행이 한 노드의 피처를 나타낸다.

2) Adjacency Matrix

노드 간의 연결 여부를 나타내는 행렬로서 그래프의 구조를 표현한다. A_{ij}가 1이면 노드 i와 노드 j가 연결됨을 나타낸다. 이전 상태의 정보를 유지하기 위해 대각선 값을 모두 1로 설정한다.

GCN의 각 계층에서 노드는 합성곱의 과정을 거쳐 Feature Matrix를 변환한다. 하나의 계층을 거치면 각 노드는 연결된 간선으로 자신의 노드 정보를 전파하고 정보를 받은 노드는 자신의 노드의 정보를 업데이트한다. 이때 노드의 수는 일정하지만, 필터의 형태에 따라 피처의 수가 달라질 수 있다. Adjacency matrix의 값은 변하지 않고 연산에 필요한 연관성 정보를 제공하는 데 사용되고, 학습의 목적에 따라 negative edge를 추가하는 등 변환할 수 있다.

반지도 학습은 레이블이 있는 데이터와 없는 데이터를 함께 사용하여 모델을 학습시키는 방식으로, 상대적으로 극히 희소하게 파악된 질병-유전자 관계 정보를 통해 아직 알려지지 않은 수많은 질병-유전자 관계 탐색에 활용된다.

레이블이 있는 데이터는 손실 함수를 사용하여 예측된 레이블과 실제 레이블 간의 차이를 최소화하면 GCN 레이어를 통해 그래프 전체로 전파되어 레이블이 없는 데이터는 embedding 학습에도 기여한다.

2.5. 이기종 그래프

일반적인 그래프가 모든 노드와 간선의 타입을 구별하지 않고 하나의 타입으로 표현하는 것과 다르게, 이기종 그래프는 2개 이상의 노드 타입, 간선 타입을 가질 수 있는 그래프이다. 이기종 그래프는 서로 다른 데이터가 다른 피처 타입을 가지고 있더라도 하나의 그래프로 구성할 수 있고 타입에 따라 다르게 처리할 수 있다는 장점이 있다.

본 과제에서는 질병과 유전자를 각각 나타내는 2가지 타입의 노드와 두 노드 사이의 연관성을 나타내기 위한 하나의 간선 타입을 가지는 이기종 그래프를 생성하고 이를 입력값으로 사용해 모델을 생성했다.

2.6. Pytorch Geometric

과제에서 주로 사용하는 라이브러리는 Pytorch Geometric으로, Pytorch를 기반으로 그래프 신경망을 쉽게 작성하고 학습할 수 있는 기능을 제공하는 라이브러리다. 과제에서는 라이브러리에서 제공하는 HeteroData로 이기종 그래프를 구성하고 HeteroConv, SAGEConv, GATConv 등의 다양한 함수를 사용해 최적의 모델을 찾아냈다.

3. 연구 내용

3.1. 데이터 전처리

(1) disease association

diseaseId	feature matrix에서 각 노드를 구별하기 위해 사용된다.
diseaseClass	- 각 항목이 2개 이상의 레이블을 가지는 열이므로 one-hot encoding을 통해 희소행렬을 형성한다. - 이때 결측치 값은 모든 값이 0인 벡터로 설정한다.
diseaseType, diseaseSemanticType	범주형 데이터이므로 각 항목에 맞게 특정 값을 가지도록 label encoding을 수행한다.
diseaseName	"diseaseId"로 대체할 수 있으므로 제거한다.

```
disease = pd.read_csv(data_path + "disease_associations.tsv",
delimiter="\t", encoding='cp949')

del disease["diseaseName"]

disease['diseaseClass']
        = disease["diseaseClass"].fillna("").str.split(";")
disease = disease.fillna("None")
# multi one-hot encoding 진행
mlb = MultiLabelBinarizer()
one_hot_encoded = mlb.fit_transform(disease['diseaseClass'])

# PCA를 통해 diseaseClass의 영향 축소
pca = PCA(n_components=2)
disease_class_feature = pca.fit_transform(one_hot_encoded)
disease_class_feature = pd.DataFrame(disease_class_feature)
disease = pd.concat([disease, disease_class_feature], axis=1)

# 라벨인코딩을 적용할 변수의 리스트 생성
features = ["diseaseType", "diseaseSemanticType" ]

# fit()과 transform()으로 라벨인코딩을 수행
for feature in features:
    le = LabelEncoder()
    le = le.fit(disease[feature])
    disease[feature] = le.transform(disease[feature])
```

```
# 학습에 사용될 데이터에 id값이 적용되지 않도록 분리
disease_id = disease["diseaseId"]
del disease["diseaseId"]
del disease["diseaseClass"]

# 데이터를 더 잘 표현할 수 있도록 5차원에서 32차원으로 변환
data = torch.Tensor(disease.values)
linear_layer = nn.Linear(6, 32)
transfomed_disease = linear_layer(data)
```

(2) gene association

geneId	feature matrix에서 각 노드를 구별하기 위해 사용된다.
protein_class, protein_class_name	과도한 결측치 값을 가지므로 제거한다.
DSI, DPI, PLI	결측치 항목만 제거한다.
geneSymbol	geneId로 대체될 수 있어서 제거한다.

```
gene = pd.read_csv(data_path + "gene_associations.csv")

del gene["geneSymbol"], gene["protein_class_name"], gene["protein_class"]

# gda의 geneId와 비교하기 위해 문자열로 변경
gene["geneId"] = gene["geneId"].apply(str)

# "DSI", "DPI", "PLI" 각각의 결측치 항목 삭제
gene = gene.dropna(subset=["DSI", "DPI", "PLI"]).reset_index(drop=True)

# 학습에 사용될 데이터에 id값이 적용되지 않도록 분리
gene_id = gene["geneId"]
del gene["geneId"]

# 데이터를 더 잘 표현할 수 있도록 5차원에서 32차원으로 변환
data = torch.Tensor(gene.values)
linear_layer = nn.Linear(5, 32)
transformed_gene = linear_layer(data)
```

(3) gda (Gene-Disease Association)

Feature matrix에서 구별자로 사용될 각 아이디만을 추출하고 기존의 유전자, 질병 데이터에 맞게 gda 데이터의 아이디를 가공한다.

```
# gda 데이터에서 유전자, 질병 id 추출
def get_diseaseId(x):
  index = x.find("id/")
  if index == -1:
    return -1
  return x[index+3:]

def get_geneId(x):
  index = x.find("ncbigene/")
  if index == -1:
    return -1
  return x[index+9:]

gda["geneId"] = gda["gene"].apply(get_geneId)
gda["diseaseId"] = gda["disease"].apply(get_diseaseId)
gda = gda[["geneId", "diseaseId"]]

# gda 내 유전자-질병 쌍 중복 제거
gda = gda.drop_duplicates(["geneId", "diseaseId"])
```

geneId	diseaseId
3067	C0007102
3096	C0007103
3912	C0009324
3912	C0009363
58484	C0009763

(4) Heteroreneous Graph

Pytorch Geometric의 이기종 데이터를 이용해 그래프를 구성하였다. 32차원의 유전자와 질병을 노드로 하고 둘 간의 관계를 간선으로 하는 이기종 그래프를 생성하였다.

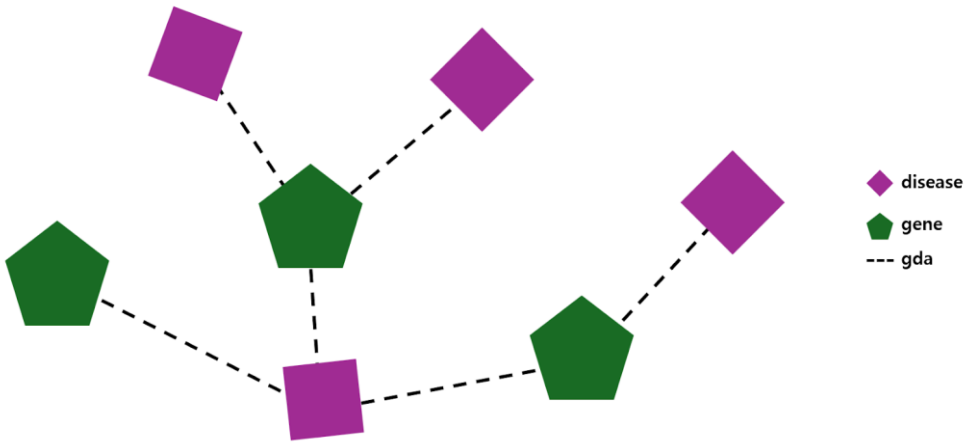
```
data = HeteroData()

# 유전자, 질병 노드 추가
data["gene"].x = torch.tensor(gene.loc[:,gene.columns!="geneId"] ··· )
data["disease"].x = torch.tensor(disease.loc[:, disease.columns! ··· )

# id를 인덱스로 변환
gda_index = pd.DataFrame({"geneIndex" : gda["geneId"] ··· ).transpose()

# 유전자-질병, 질병-유전자 엣지 추가
data["gene", "associations", "disease"].edge_index ··· (gda_index.values)
data = T.ToUndirected()(data)
```

diseaseId: C0000727

1.779948	0.44154	0.445494	0.139612	-1.25676	-0.14796	0.953746	1.248745
-0.22627	0.128925	-0.48005	0.021263	0.102218	0.735923	0.088815	-0.50367
0.385097	-0.64757	-1.39814	-0.65307	-0.71003	-1.51873	0.085731	0.011298
-0.28948	-0.74908	0.750271	-0.73553	-0.85915	-0.8268	0.528756	-0.02157

geneId: 1

0.489631	-0.27692	-0.24577	-0.20863	0.282036	0.148487	-0.69432	0.135623
-0.28332	-0.16985	-0.14878	-0.03597	0.026235	0.399022	-0.34318	-0.48572
-0.59752	0.591642	0.04807	0.050002	0.345588	0.43645	0.1435	0.276908
0.323187	0.334313	0.352494	0.042661	-0.43507	-0.17164	-0.11938	-0.0749



3.2. 모델 구성

```
class GCN(nn.Module):
  def __init__(self, h_dim):
    super().__init__()
    self.conv1 = SAGEConv((-1,-1), h_dim)
    self.conv2 = SAGEConv((-1,-1), h_dim)
  def forward(self, x_dict, edge_index_dict):
    x_dict = self.conv1(x_dict, edge_index_dict).relu()
    x_dict = self.conv2(x_dict, edge_index_dict)
    return x_dict

class Classifier(nn.Module):
  def forward(self, x_dict, edge_label_index):
    edge_feat_gene = x_dict["gene"][edge_label_index[0]]
    edge_feat_disease = x_dict["disease"][edge_label_index[1]]
    return (edge_feat_gene * edge_feat_disease).sum(dim=-1)

class Model(nn.Module):
  def __init__(self, h_dim):
        super().__init__()
        self.gcn = GCN(h_dim)
        self.gcn = to_hetero(self.gcn, data.metadata())
        self.classifier = Classifier()
```

GCN의 합성곱 층으로는 GraphSAGE를 프레임워크의 개념을 사용하였다. GraphSAGE는 대규모 그래프에서 일반화 문제를 해결하기 위한 방식이다. SAGEConv 계층을 2번 통과함으로써 이웃 노드의 정보로 자신의 노드 정보를 업데이트하고 이를 통해 그래프상에서 노드 간의 연관성을 표현하도록 학습한다. 계층을 모두 거친 그래프에 Classifier를 적용해 벡터화함으로 노드의 정보를 토대로 하나의 값을 출력한다.

```
Epoch: 001, Loss: 0.5880, Train: 0.9503, Val: 0.9493, Test: 0.9534
Epoch: 002, Loss: 0.5397, Train: 0.9737, Val: 0.9746, Test: 0.9779
Epoch: 003, Loss: 0.5077, Train: 0.9840, Val: 0.9877, Test: 0.9869
Epoch: 004, Loss: 0.4828, Train: 0.9885, Val: 0.9931, Test: 0.9913
Epoch: 005, Loss: 0.4623, Train: 0.9906, Val: 0.9956, Test: 0.9937
```

베이스 모델과 8월까지 수집한 데이터로 테스트를 수행해 본 결과 정확도가 0.99라는 과도하게 높은 정확도를 얻을 수 있었다. 유전자와 질병 사이의 연관 관계를 나타내는 간선 데이터 부족으로 인한 과적합을 그 원인으로 판단했고 이를 해결하기 위해 추가적인 연구를 수행하였다.

3.3. 모델 고도화

(1) 추가 데이터 수집

베이스 모델을 구축하며 추가적인 데이터 수집의 필요성을 느낄 수 있었다. 이를 위해 DisGeNET에서 제공하는 API를 이용해 gda 데이터를 추가 수집했다.

```
                                        ⋮
  # Parse response content in JSON format
  response_parsed = json.loads(response.text)
                                        ⋮
  df = pd.json_normalize(response_parsed["payload"])
  if(response_parsed["paging"]["totalElements"] > 0):
    # gda 데이터 수집을 위해 유전자와 질병의 아이디 정보만 가져온다.
    gda_result = df[["geneNcbiID","diseaseUMLSCUI"]]
    gda_df = pd.concat([gda_df, gda_result])
    time.sleep(1.3)
gda_df.to_csv("gda_0.csv",index=False)
```

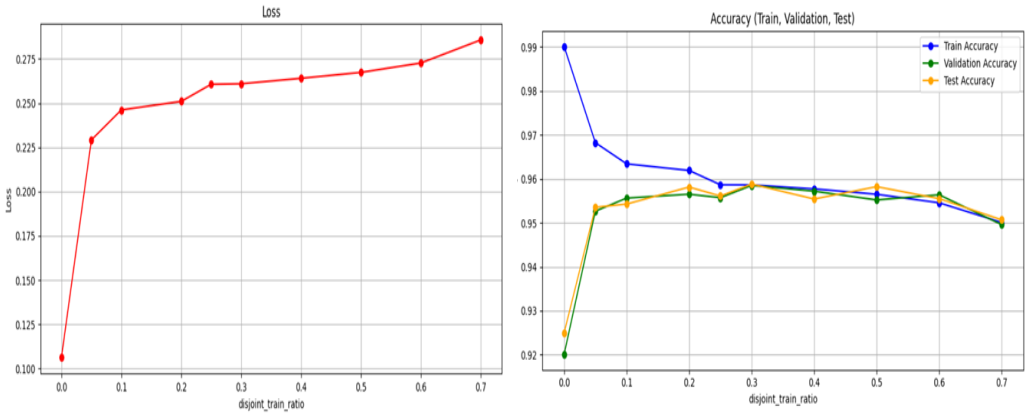
위의 과정을 통해 약 70,000개의 gda를 추가 확보하였다.

(2) 하이퍼파라미터 튜닝

앞서 구현한 베이스 모델을 바탕으로 하이퍼파라미터를 조정하며 모델에 적합한 값을 찾아 모델의 정확성을 높이고자 했다.

- **disjoint_train_ratio**

  학습에 사용되는 간선 중 메시지 전달을 위한 데이터와 레이블만을 위한 데이터의 비율을 설정하는 옵션이다.

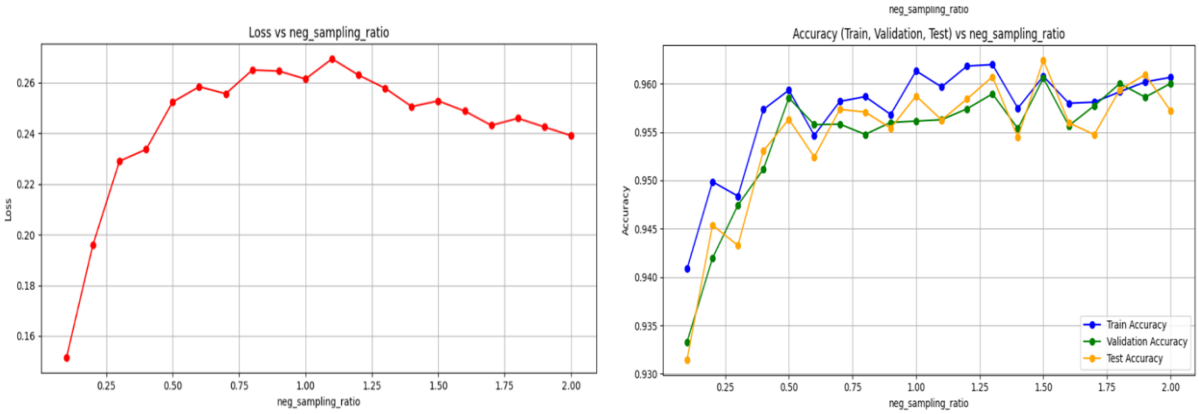


값이 0에 가까울수록 학습 데이터셋의 정확도는 높아지는 데 반해 검증, 테스트 데이터셋의 정확도는 낮아지는 과적합을 보인다. 과적합을 방지할 수 있고 비교적 높은 정확도를 보이는 0.3으로 설정하였다.

- **neg_sample_ratio**

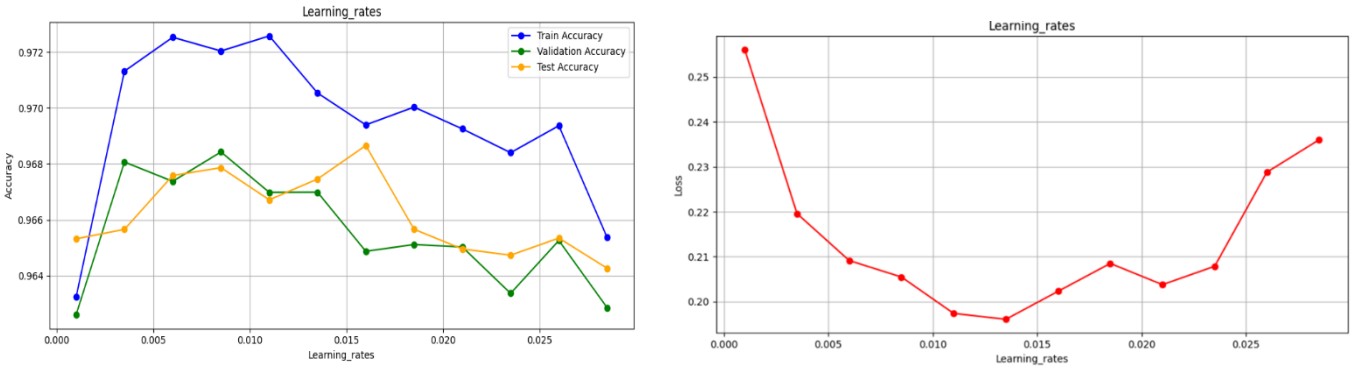
neg_sample_ratio는 positive sample에 대한 negative sample의 비율로, gda 내 연관 정보가 존재하는 간선을 positive sample, 테스트를 위해 연관성이 없음을 나타내기 위해 추가된 edge를 negative sample이라 볼 수 있다.

아래의 과정을 통해 1.25의 값이 가장 적합하다고 판단했다.



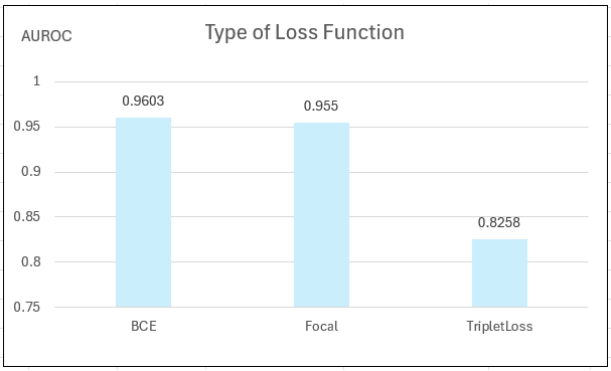
- **Learning rate**

가장 적합한 learning rate를 찾기 위해서 다양한 learning rate를 적용하였고 가장 낮은 loss와 높은 acc 값을 가지는 0.01을 도출해 낼 수 있었다.



- **loss function**

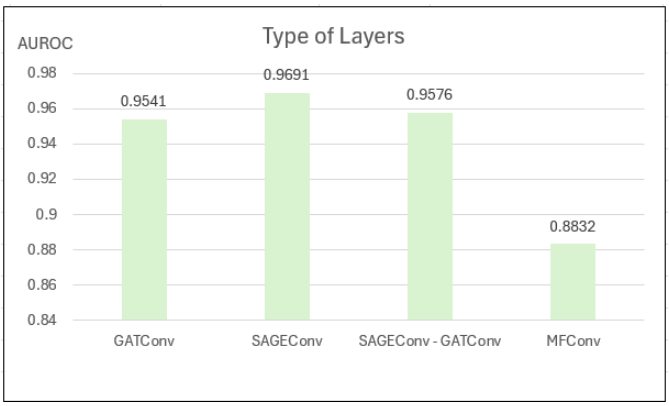
  link prediction을 위한 이진 분류 loss function 중 "Binary cross entropy"가 적합함을 확인하였다.

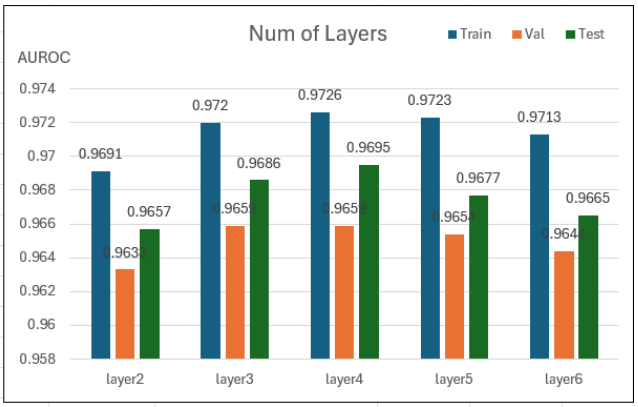


(3) 모델 계층 구성

기본 모델 구성인 2 layer에서 모델 학습에 적합한 layer를 찾기 위해서 다양한 합성곱 계층에 대해 실험해 본 결과 "SAGEConv"가 적합함을 알아낼 수 있었다.

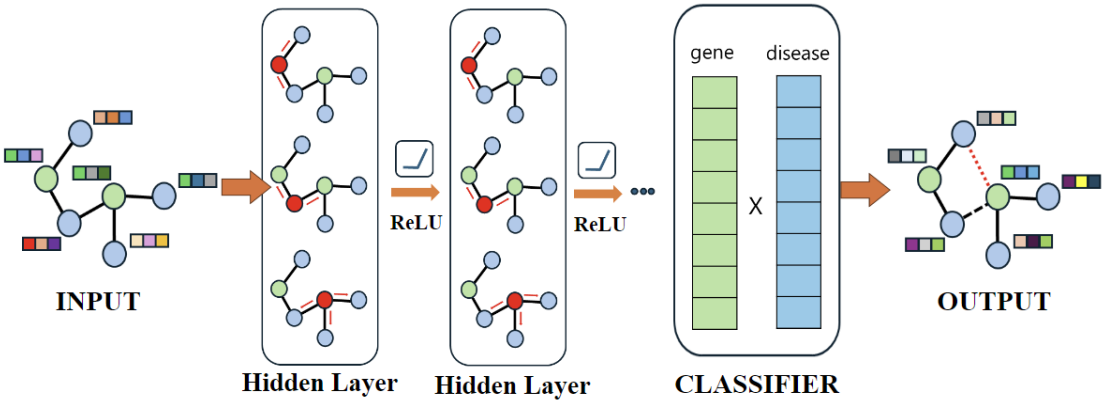
이를 통해 layer를 추가로 쌓아가면서 학습시켜 본 결과 4 layer가 가장 학습이 잘 되었지만, layer 수에 따른 차이는 크지 않은 것을 확인했다.





4. 연구 결과 분석 및 평가

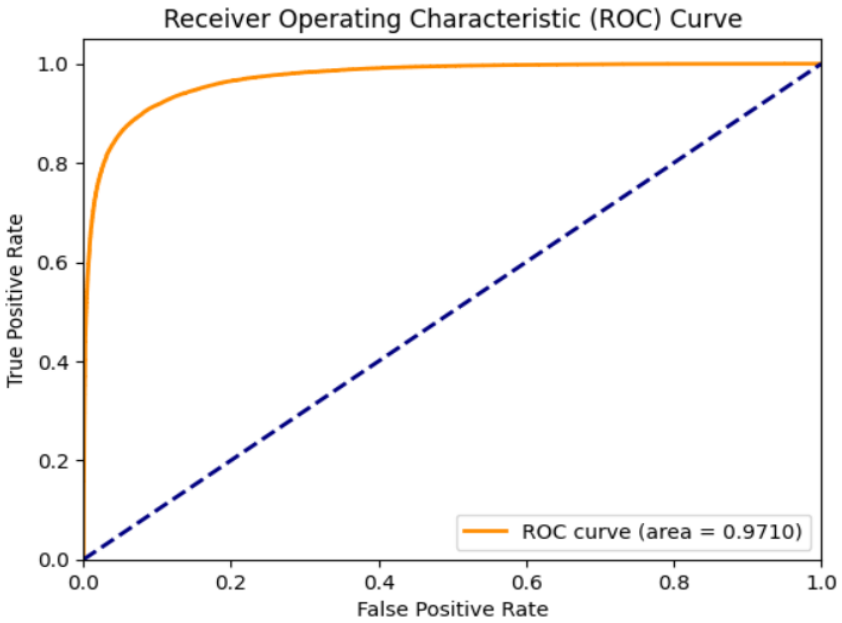
4.1. 학습 및 평가 결과



과제를 통해 최종적으로 얻어낸 모델은 위와 같은 과정으로 학습이 이루어진다.

```
Epoch 20, Train Loss: 0.2742, Train acc: 0.885656 Train AUROC: 0.9540, Val AUROC: 0.9518, Test AUROC: 0.9534
Epoch 40, Train Loss: 0.2432, Train acc: 0.901209 Train AUROC: 0.9636, Val AUROC: 0.9627, Test AUROC: 0.9633
Epoch 60, Train Loss: 0.2338, Train acc: 0.905245 Train AUROC: 0.9680, Val AUROC: 0.9670, Test AUROC: 0.9677
Epoch 80, Train Loss: 0.2187, Train acc: 0.911288 Train AUROC: 0.9707, Val AUROC: 0.9684, Test AUROC: 0.9691
Epoch 100, Train Loss: 0.2087, Train acc: 0.915003 Train AUROC: 0.9729, Val AUROC: 0.9688, Test AUROC: 0.9695
```

최종 모델의 학습 결과는 Train Accuarcy : 0.915, Train AUROC : 0.9729 로 준수한 값이 나왔고, 평가 결과도 Val AUROC : 0.9688, Test AUROC : 0.9695로 큰 과적합 없이 모델 학습이 잘 되었다고 평가 내릴 수 있었다.



모델의 성능을 평가하기 위한 ROC curve를 그려 보았고 AUROC 값이 0.9710이 나왔다. 이를 통해 우리의 모델이 link prediction을 위한 이진 분류에 적합함을 확인할 수 있었다.

4.2. 시연

실제로 모델이 학습함에 따라 Link Prediction이 잘 생성되었는지 확인해 보았다

```
Predicted links for gene 2 :
Disease C2239176: Prediction: 0.9994, Ground Truth: 1.0
Disease C0002395: Prediction: 0.9972, Ground Truth: 1.0
Disease C0030567: Prediction: 0.9965, Ground Truth: 1.0
Disease C0022660: Prediction: 0.9903, Ground Truth: 1.0
Disease C0009375: Prediction: 0.9892, Ground Truth: 1.0
Disease C0023890: Prediction: 0.9849, Ground Truth: 1.0
Disease C2609414: Prediction: 0.9704, Ground Truth: 1.0
Disease C4749335: Prediction: 0.4580, Ground Truth: 0.0
Disease C0004606: Prediction: 0.1271, Ground Truth: 1.0
Disease C3805969: Prediction: 0.0402, Ground Truth: 0.0
Disease C3272931: Prediction: 0.0358, Ground Truth: 0.0

Predicted links for Disease C0030567 :
Gene 57010: Prediction: 0.9805, Ground Truth: 0.0
Gene 197322: Prediction: 0.9694, Ground Truth: 0.0
Gene 5681: Prediction: 0.8229, Ground Truth: 0.0
```

Gene 2에 대한 결과에서, GT(Ground Truth) 값이 1인 기존 gda 데이터를 제외하고, GT 값이 0인 새로운 gda 데이터가 생성됨을 확인할 수 있었다.

Disease C0030567, 즉 Parkinson disease에 대해 Ground Truth 값이 0인 경우만을 확인한 결과, 3개의 Gene에서 예측값이 0.5를 넘었다. 이를 통해 새로운 gda가 생겼음을 예측할 수 있었다.

이와 같은 결과를 통해, 우리의 모델은 예측값을 기반으로 GDA의 중요도에 차등을 둘 수 있으며, 새롭게 예측된 GDA를 확인함으로써 실험의 효율적인 방향성을 제시할 수 있을 것으로 보인다.

5. 결론 및 향후 연구 방향

이번 과제를 통해 질병 연관 유전자 발굴을 위한 머신러닝 기법을 설계하고 다양한 방식으로 모델의 정확도를 높이기 위한 방법을 탐색했다. 모델의 신뢰성을 높이기 위해 추가적인 GDA 데이터 수집 및 학습 데이터 내의 negative edge 비율 조정, 레이어 개수 조정 등 다양한 방법을 적용했고 최적의 모델을 도출해 낼 수 있었다. 이를 통해 질병-유전자의 관계를 탐구하는 연구에서 특정 질병과 관련성이 높은 유전자를 제공해 연구의 가이드라인을 제공하고, 의학 기술의 발전에 도움이 될 것이라 기대한다.

현재 모델에서는 질병과 유전자만을 바탕으로 학습을 진행했기 때문에 연구에 적용되기에는 단순한 모델이라고 생각된다. 이후, 환경적 요인, 단백질, RNA 등의 노드를 추가하고 질병-질병, 유전자-유전자 등 다른 종류의 관계 데이터를 제공하는 데이터베이스를 탐색하고 추가 수집해 더욱 복합적인 그래프를 구성할 예정이다. 추가 데이터를 통해 더욱 정교하고 세밀한 모델을 얻어낼 수 있을 것이다.

6. 과제 일정 및 역할

개발 구분	추진내용	추진일정											
		5월		6월		7월		8월		9월		10월	
		上	下	上	下	上	下	上	下	上	下	上	下
계획	착수보고서 작성												
분석	주요 기술 이해												
	선례 연구 이해 및 분석												
설계	데이터탐색												
	학습 모델 기법 연구												
	개발 환경 구축												
개발	데이터셋 전처리												
	베이스모델 구축												
	중간보고서 작성												
고도화	추가 데이터 수집												
	하이퍼파라미터 조정												
마무리	최종보고서 작성												
	결과물 업로드 및 후속 처리												

조원	역할 분담
김이경	- 머신러닝 기법 학습, 주요 라이브러리 학습 - 데이터 전처리, 베이스 모델 구축
박화성	- 머신러닝 기법 학습, 데이터 전처리 - 하이퍼파라미터 조정, 시연 프로그램 구현
이윤재	- 머신러닝 기법 학습, 아키텍처 설계 - 데이터 수집 및 추가 수집, 데이터 시각화

7. 참고 문헌

[1] Peng Han, Peng Yang, Peilin Zhao, Shuo Shang, Yong Liu, Jiayu Zhou, Xin Gao, Panos Kalnis, “GCN-MF: Disease-Gene Association Identification By Graph Convolutional Networks and Matrix Factorization”, 2019
[2] Thomas N. Kipf, Max Welling, “Semi-Supervised Classification with Graph Convolutional Networks”
[3] Tomonori Masui, “Graph Neural Networks with PyG on Node Classification, Link Prediction, and Anomaly Detection”, https://towardsdatascience.com/graph-neural-networks-with-pyg-on-node-classification-link-prediction-and-anomaly-detection-14aa38fe1275 (Oct 7, 2022)
[4] PyG Documentation [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/index.html
[5] DisgeNet [Online]. Available: https://www.disgenet.org
[6] National Library of Medicine - National Institutes of Health [Online]. Available: https://www.nlm.nih.gov