2024 전기 졸업과제 중간보고서

질병 연관 유전자 발굴을 위한 머신러닝 기법 설계

7조 서울대구부산

목차

1. 과제 배경 및 목표

a. 기존 목표

유전자와 질병과의 관계를 파악하는 것은 질병의 원인을 분석하고 빠르게 대응하기 위한 중요한 생물학적 문제이다. 이러한 관계를 파악하기 위한 많은 노력이 있었지만 복잡한 유전자 발현 시스템으로 인해 유전자-질병 관계 파악에 있어서 많은 어려움을 겪고 있다. 이를 해결하기 위해 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 것을 의미한다.

위의 특성을 바탕으로 새로운 유전자-질병 관계를 예측하는 모델을 생성하는 것이 본 과제의 목표이다. 이때 유전자와 질병은 복잡한 고차원 관계가 있으므로, 이를 반영할 수 있는 그래프 구조의 GCN을 적용해 예측 모델을 생성할 계획이다.

b. 추가 요구사항

Heterogeneous Graph 개념을 도입해 질병 노드와 유전자 노드를 분리해 그래프를 생성한다. 생성한 그래프를 바탕으로 GCN을 적용해 모델을 생성한다. 모델의 결과 벡터 값이 설정한 threshold 값보다 큰 경우 두 노드는 연관성이 있는 것으로 판단해 노드 간의 새로운 연관성 정보를 제공한다. 이를 통해 질병과 유전자 사이의 연관성 연구에 도움을 주는 것을 최종 목표로 한다.

2. 데이터 분석

a. 데이터 선정 - 착수보고서 내용

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. DisGeNET에서 는 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 (disease association), 유전자 데이터 (gene association), 질병과 유전자 사이의 관계를 나타내는 데이터(gene disease association)를 사용한다.

b. 데이터 특징

(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

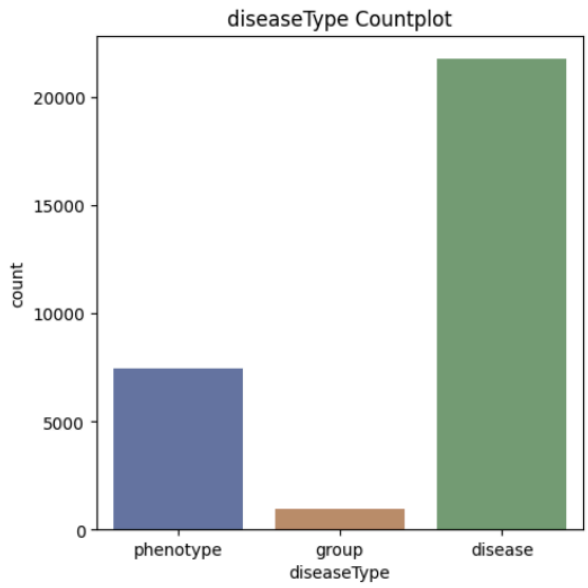
(3) **gene disease association** - 유전자와 관련된 질병들을 나타내는 데이터

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

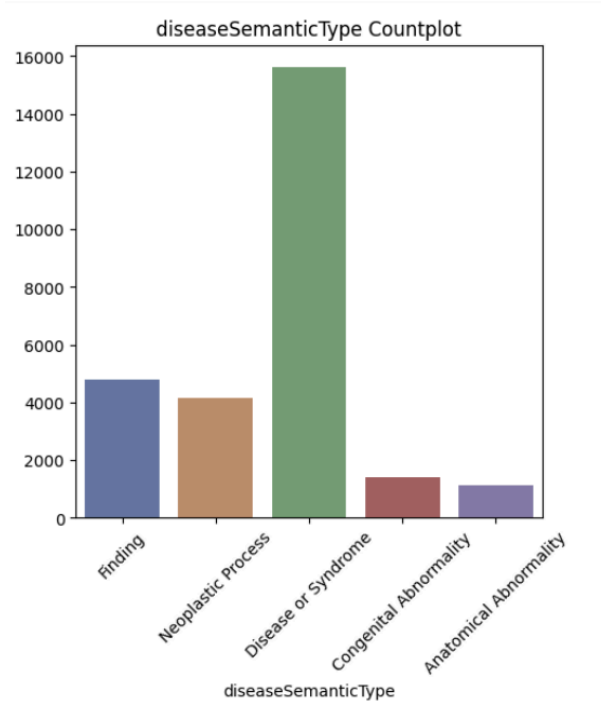
[3] National Center for Biotechnology Information
[4] Drug Target Ontology

C. 데이터 시각화

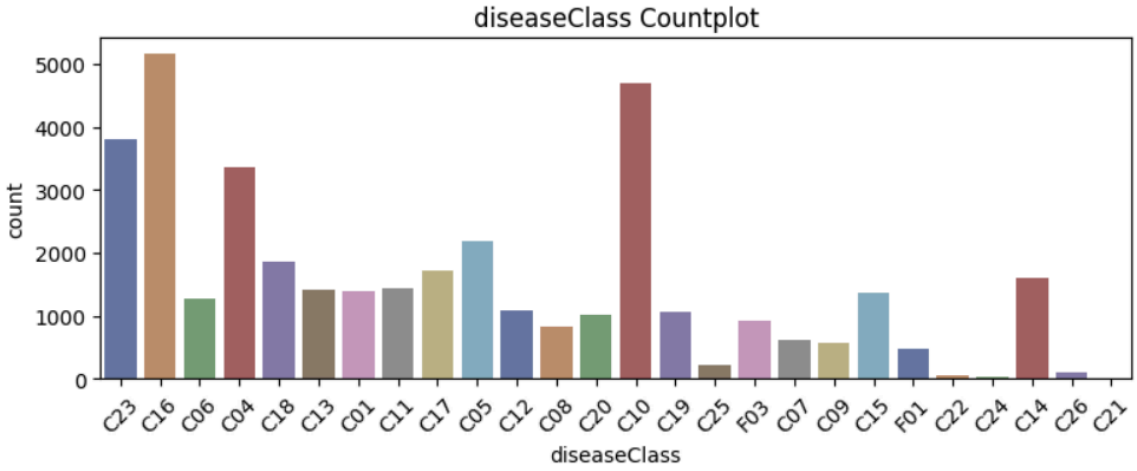
(1) **disease association** - 질병과 관련된 여러 가지 특징들을 제공해 주는 데이터



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



- diseaseSemanticType에 대한 countplot
- 상위 5개의 diseaseSemanticType만을 시각화

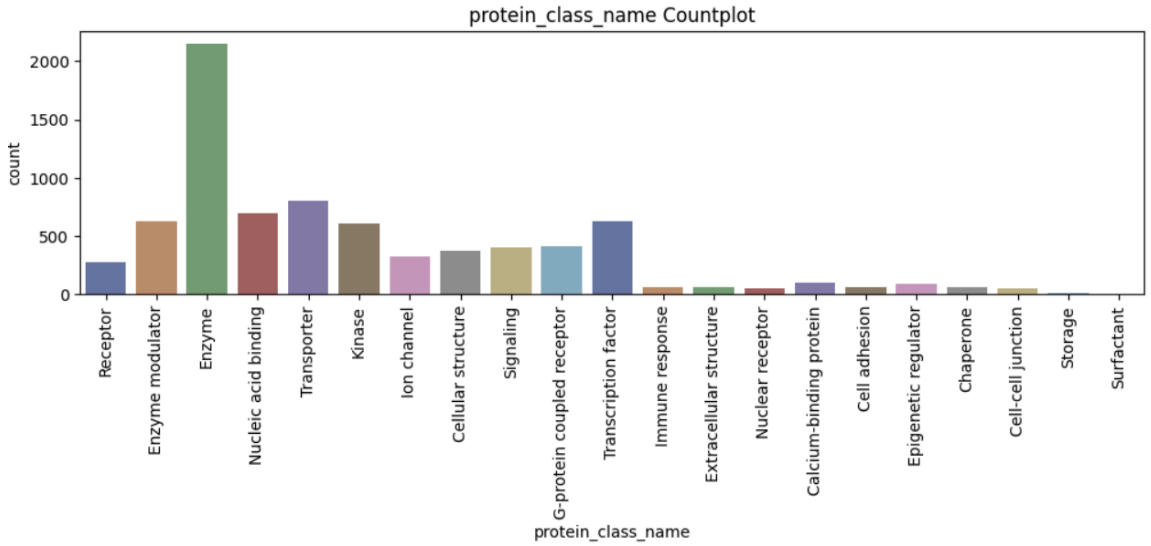


질병은 26개의 disease class로 분류되며, C16, C10, C04, C23에 해당하는 질병이 가장

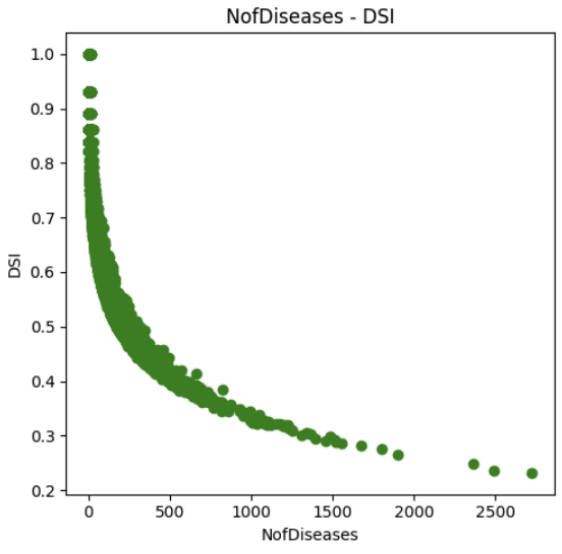
[5] 범주형 변수의 빈도수를 시각화

많음을 알 수 있다.

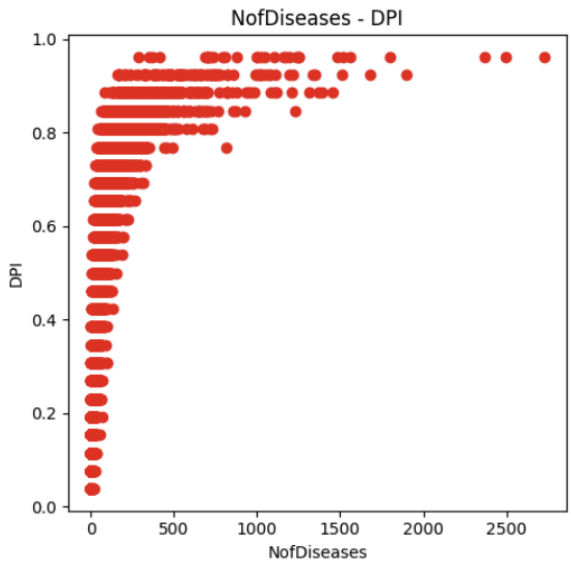
(2) **gene association** - 유전자와 관련된 여러 가지 특징들을 제공해 주는 데이터



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.

3. 모델 개요

a. 데이터 전처리

(1) disease association

diseaseId	feature matrix에서 각 노드를 구별할 수 있는 node_id로 사용하도록 구별한다.
diseaseClass	- 각 항목이 2개 이상의 label을 가지는 column이므로 one-hot encoding을 통해 sparse matrix를 형성한다. - 이때 결측치 값은 모든 값이 0인 벡터로 설정한다.
diseaseType	categorical data이므로 각 항목에 맞게 특정 값을 가지도록 label encoding을 수행한다.
diseaseSemanticType	categorical data이므로 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'])

# diseaseClass의 차원 수가 너무 커져 PCA를 통해 조절
pca = PCA(n_components=2)
disease_class_feature = pca.fit_transform(one_hot_encoded)
disease_class_feature = pd.DataFrame(disease_class_feature)
```

```
# 라벨인코딩을 적용할 변수의 리스트를 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])

# data index 표시를 위해 "diseaseId" 항목 따로 저장
disease_id = disease["diseaseId"]
del disease["diseaseId"]
del disease["diseaseClass"]

disease = pd.concat([disease, disease_class_feature], axis=1)
# nn.Linear는 파이토치에서 사용되는 선형 변환(linear transformation)을 수행
data = torch.Tensor(disease.values)

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

```
torch.Size([30170, 32])
tensor([[ 0.1646,  0.6068, -0.0835,  ..., -0.3748,
          [ 0.6155,  0.2542,  0.5965,  ...,  0.1180,
          [ 0.6002,  0.1298, -0.5760,  ..., -0.1959,
          ...,
          [ 0.1161,  0.2095, -0.0638,  ...,  0.4466,
```

(2) **gene association**

geneId	feature matrix에서 각 노드를 구별할 수 있는 node_id로 사용하도록 구별한다.
protein_class	과도한 결측치 값을 가지므로 제거한다.

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)

# data index 표시를 위해 따로 저장
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)

# Apply the linear transformation
transformed_gene = linear_layer(data)
```

```
torch.Size([15576, 32])
tensor([[   0.2381,   -0.5834,    4.0440,  ...,   -7.4566,    4.5886,
          -16.3350],
        [ -14.3020,  -13.8706,   28.0730,  ...,  -56.4210,   12.9030,
         -100.5000],
        [ -32.6973,  -31.7654,   37.8062,  ...,  -71.9893,   -5.0874,
         -108.7309],
        ...,
        [   0.7363,   -0.3135,    0.6414,  ...,   -0.1353,    0.4572,
           -1.0584],
        [  -5.9608,   -6.2368,   14.7620,  ...,  -29.3362,    8.8582,
          -54.4969],
        [   0.5101,   -0.4898,    1.0577,  ...,   -0.9306,    0.6639,
```

(3) **GDA(Gene-Disease Association)**

Feature matrix에서 구별자로 사용될 각 Id만을 뽑아내고 기존의 Gene, Disease data에 맞게 GDA의 ID data를 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

b. Heterogeneous Graph

Heterogeneous Graph는 다양한 타입의 노드와 엣지를 가질 수 있는 그래프이다. 본 과제에서는 Gene과 Disease 각각에 대한 노드 타입을 가지고 GDA(Gene-Disease Associations)의 양방향 관계를 표현하기 위해 associations, rev-associations 두 가지의 엣지 타입을 가지는 그래프를 구성해 학습을 진행한다.

c. Pytorch Geometric

Pytorch Geometric 라이브러리는 Pytorch를 기반으로 그래프 신경망을 쉽게 작성하고 학습할 수 있는 기능을 제공하는 라이브러리다. 과제에서는 라이브러리에서 제공하는 HeteroData로 이종그래프를 구성하고 HeteroConv, SAGEConv, GATConv로 모델을 구성한다.

d. 모델 구현 과정 및 설명

i. 사용된 기법

GCN의 graph convolution layer로는 GraphSAGE를 프레임워크의 개념을 사용했다. GraphSAGE는 대규모 그래프에서 일반화 문제를 해결하기 위한 방식이다. 그래프 구조의 mini-batch를 생성해 대규모 그래프의 복잡성을 줄이고 일반화된 모델을 제공한다.

ii. 코드

현재 GCN 알고리즘을 이용한 베이스 모델 구축을 완료한 상태이다.

```
data = HeteroData()

# 유전자, 질병 노드 추가
data["gene"].x = torch.tensor(gene.loc[:, gene.columns!="geneId"].values,
dtype=torch.float32)
data["disease"].x = torch.tensor(disease.loc[:,
disease.columns!="diseaseId"].values, dtype=torch.float32)

# id를 인덱스로 변환
gda_index = pd.DataFrame({"geneIndex" :
gda["geneId"].map(gene.set_index("geneId").index.get_loc),
                          "diseaseIndex" :
gda["diseaseId"].map(disease.set_index("diseaseId").index.get_loc)}).transpose(
)

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

torch-geometric의 HeteroData를 이용해 그래프를 구성했다. 전처리된 질병과 유전자 데이터를 그래프의 노드로 추가하고 gda의 각 id를 노드 인덱스로 변경해 엣지를 생성했다.

```
# 데이터 분할
train_data, val_data, test_data = T.RandomLinkSplit(
    num_val=0.1,
    num_test=0.1,
    is_undirected=True,
    disjoint_train_ratio=0.3,
    neg_sampling_ratio=1.0,
    add_negative_train_samples=True,
    edge_types=[("gene", "associations","disease")],
    rev_edge_types=[("disease", "rev_associations", "gene")]
)(data)
```

그래프를 train, validation, test 데이터 셋으로 분리한다. 8:1:1의 비율로 분리하고 disjoint_train_ratio를 설정해 message passing이 아닌 단순 라벨만을 제공하는 엣지를 추가해 모델의 과적합을 방지한다.

```
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()

  def forward(self, data):
      x_dict = {
          "gene":data["gene"].x,
          "disease":data["disease"].x
      }

      x_dict = self.gcn(x_dict, data.edge_index_dict)
      output = self.classifier(
          x_dict,
          data["gene", "associations", "disease"].edge_label_index,
      )

      return output
```

SAGEConv Layer를 2번 통과함으로써 이웃 노드의 정보로 자신의 노드 정보를 업데이트하고 이를 통해 그래프상에서 노드 간의 연관성을 표현하도록 학습한다. Layer를 모두 거친 그래프에 Classifier를 적용해 벡터화함으로 노드의 정보를 토대로 하나의 값을 출력한다.

iii. 발전 방향

(1) 데이터 관련 발전 방향

- 작성한 베이스 모델의 높은 정확도는 유전자와 질병의 수에 비해 유전자와 질병 사이의 관계를 나타내는 GDA 데이터 수가 적기 때문이라고 생각한다. 이를 해결하기 위해 추가적인 GDA 데이터를 수집하여 이를 해결하고자 한다.
- 현재 전처리 과정에서 유전자 데이터는 결측치가 많아 일부 데이터를 제거하여 데이터 손실이 발생했다. 이를 보완하기 위해 데이터 손실을 최소화하는 결측치 처리 방식을 고안해 학습을 수행하며 모델에 가장 적합한 데이터 셋을 찾아갈 예정이다.

(2) 모델 관련 발전 방향

- Adam, RMSprop 등의 최적화 기법을 도입해 모델의 성능을 높이기 위한 노력을 수행할 예정이다.
- GridSearchCV 라이브러리를 적용해 다양한 하이퍼파라미터를 조합하고 교차검증을 수행해 과적합을 최적화할 수 있는 적절한 하이퍼파라미터 값을 도출해 낼 것이다.

8월 말부터 위와 과정을 통해 신뢰도 높은 모델을 제공할 예정이다.

4. 과제 추진 계획 변경 및 진척도

a. 과제 일정

개발 구분	추진 내용	추진 일정 (월별, 상/하 구분)											
		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~~						■	■					
	모델 학습 및 클리닝						■	■	■	■	■		
테스트	모델 테스트 및 최적화								■	■	■	■	
마무리	최종보고서 작성										■	■	■
	결과물 업로드 및 후속 처리												■

b. 조원 별 진척도

조원	역할 분담
김이경	머신러닝 기법 학습, 데이터셋 수집, 데이터셋 전처리
박화성	머신러닝 기법 학습, 데이터셋 수집, 데이터셋 전처리
이윤재	머신러닝 기법 학습, 모델 생성을 위한 라이브러리 학습, 데이터 시각화

5. 참고 자료

[1]https://github.com/heartcored98

[2]https://pytorch-geometric.readthedocs.io/en/latest/index.html

[3]https://towardsdatascience.com/graph-neural-networks-with-pyg-on-node-classification-link-prediction-and-anomaly-detection-14aa38fe1275