

**《数据挖掘》实验报告**

实验题目 药物副作用预测（分类任务）

学院系别 信息学院

专业名称 人工智能

学生姓名 刘宇菲

学生学号 31520241154517

任课教师 张仲楠

2025年6月28日

目 录

[第一章 概述 1](#_Toc202312832)

[第二章 总体设计 1](#_Toc202312833)

[第三章 算法详细介绍 1](#_Toc202312834)

[3.1分子图建模 1](#_Toc202312835)

[3.2模型结构：**ImprovedGCN** 3](#_Toc202312836)

[第四章 实验步骤及结果 4](#_Toc202312837)

[4.1实验软硬件环境 4](#_Toc202312838)

[4.2实验步骤 4](#_Toc202312839)

[4.3实验结果与分析 7](#_Toc202312840)

[参考文献 8](#_Toc202312841)

# 第一章 概述

药物副作用预测（Drug Side Effect Prediction）是药物研发中的关键任务，旨在根据药物分子的结构信息预测其在人体使用过程中可能引发的非预期生理反应。该任务采用多标签分类方法，通过机器学习模型学习药物结构与副作用标签之间的对应关系。准确的副作用预测可在药物研发早期发现潜在安全风险，降低临床试验失败率和撤药风险。

本实验基于公开的 SIDER 数据集，构建图神经网络模型对药物副作用进行多标签分类预测，使用 AUROC 和 AUPR 作为主要评估指标。

# 第二章 总体设计

本实验流程主要包括以下步骤：

1. **图构建**：使用 SMILES 表示的药物分子结构，解析为分子图。每个分子图节点为原子，边为化学键，节点/边特征通过 RDKit 提取。标签为 27 维 0/1 多标签向量，对应不同器官系统的副作用。
2. **模型设计**：采用PyTorch Geometric中的GINEConv构建图神经网络模型。使用图卷积神经网络（GCN）提取图表示。三种全局池化（mean, max, sum）融合全局特征。多层全连接网络用于最终分类。
3. **训练与评估**：微平均 AUROC（Area Under ROC Curve），微平均 AUPR（Area Under Precision-Recall Curve）
4. **模型保存与结果输出**。

# 第三章 算法详细介绍

3.1分子图建模

使用 RDKit 将 SMILES 转换为分子图：

* 节点特征包括原子编号、杂环性、混成轨道、是否在环中等 15 维特征。
* 边特征包括键类型、芳香性、是否在环中。

    def **smiles\_to\_graph**(self, smiles, label):

        mol = **Chem**.MolFromSmiles(smiles)

        if mol is None:

            return None

        mol = **Chem**.AddHs(mol)

        x = []

        for atom in mol.GetAtoms():

            features = [

                atom.GetAtomicNum() / 100.0,

                atom.GetDegree() / 10.0,

                atom.GetFormalCharge(),

**int**(atom.GetHybridization()) / 10.0,

**int**(atom.GetIsAromatic()),

                atom.GetMass() / 100.0,

**int**(atom.IsInRing()),

                atom.GetTotalNumHs() / 10.0,

**int**(atom.GetChiralTag()) / 10.0,

**len**([x for x in atom.GetNeighbors()]) / 10.0,

**float**(atom.GetAtomicNum() in [6, 7, 8, 9, 15, 16, 17, 35, 53]),  *# 常见药物原子*

**int**(atom.IsInRingSize(3)),

**int**(atom.IsInRingSize(4)),

**int**(atom.IsInRingSize(5)),

**int**(atom.IsInRingSize(6)),

            ]

            x.**append**(features)

        edge\_attr = []

        edge\_indices = []

        for bond in mol.GetBonds():

            i = bond.GetBeginAtomIdx()

            j = bond.GetEndAtomIdx()

            bond\_type = bond.GetBondType()

            bond\_features = [

**float**(bond\_type) / 10.0,

**int**(bond.GetIsAromatic()),

**int**(bond.IsInRing()),

            ]

            edge\_indices.**extend**([[i, j], [j, i]])

            edge\_attr.**extend**([bond\_features, bond\_features])

        if **len**(edge\_indices) == 0:

*# 如果没有边，创建自环*

            num\_atoms = **len**(x)

            edge\_indices = [[i, i] for i in **range**(num\_atoms)]

            edge\_attr = [[0.0, 0.0, 0.0] for \_ in **range**(num\_atoms)]

        edge\_index = **torch**.**tensor**(edge\_indices, dtype=**torch**.long).**t**().**contiguous**()

        edge\_attr = **torch**.**tensor**(edge\_attr, dtype=**torch**.float)

        x = **torch**.**tensor**(x, dtype=**torch**.float)

        y = **torch**.**tensor**(label, dtype=**torch**.float)

**data** = **Data**(x=x, edge\_index=edge\_index, edge\_attr=edge\_attr, y=y)

        return **data**

3.2模型结构：**ImprovedGCN**

* 输入层：15 维原子特征
* 图卷积层：5 层 GCNConv（每层后接 BatchNorm、ReLU 和 Dropout）
* 池化操作：global\_mean\_pool，global\_max\_pool和global\_add\_pool。
* 全连接分类器：输入为池化拼接结果（hidden\_dim×3），三层全连接层，含 ReLU 与 Dropout，最终输出维度为 27（类别数）。*# Model*

*# Model*

class **ImprovedGCN**(**nn**.**Module**):

    def **\_\_init\_\_**(self, input\_dim, hidden\_dim, output\_dim, num\_layers=3, dropout=0.2):

**super**(**ImprovedGCN**, self).**\_\_init\_\_**()

        self.num\_layers = num\_layers

        self.dropout = dropout

        self.convs = **nn**.**ModuleList**()

        self.convs.**append**(**GCNConv**(input\_dim, hidden\_dim))

        for \_ in **range**(num\_layers - 1):

            self.convs.**append**(**GCNConv**(hidden\_dim, hidden\_dim))

        self.batch\_norms = **nn**.**ModuleList**()

        for \_ in **range**(num\_layers):

            self.batch\_norms.**append**(**nn**.**BatchNorm1d**(hidden\_dim))

        self.classifier = **nn**.**Sequential**(

**nn**.**Linear**(hidden\_dim \* 3, hidden\_dim \* 2),  *# 3种池化的拼接*

**nn**.**ReLU**(),

**nn**.**Dropout**(dropout),

**nn**.**Linear**(hidden\_dim \* 2, hidden\_dim),

**nn**.**ReLU**(),

**nn**.**Dropout**(dropout),

**nn**.**Linear**(hidden\_dim, output\_dim)

        )

    def **forward**(self, data):

        x, edge\_index, batch = data.x, data.edge\_index, data.batch

        for i in **range**(self.num\_layers):

            x = self.convs[i](x, edge\_index)

            x = self.batch\_norms[i](x)

            x = **F**.**relu**(x)

            x = **F**.**dropout**(x, p=self.dropout, training=self.training)

        x1 = **global\_mean\_pool**(x, batch)

        x2 = **global\_max\_pool**(x, batch)

        x3 = **global\_add\_pool**(x, batch)

        x = **torch**.**cat**([x1, x2, x3], dim=1)

        out = self.classifier(x)

        return out

# 第四章 实验步骤及结果

4.1实验软硬件环境

 **硬件平台**：

CPU: Intel i7-12700H

GPU: NVIDIA RTX 2080 Ti

 **软件环境：**

操作系统：Windows 10

Python 版本：3.10

核心库版本：

PyTorch：2.7.1+cu126

RDKit：2025.3.3

Scikit-learn：1.7.0

Pandas / NumPy

(详见requirement.txt文件)

4.2实验步骤

4.2.1 构建Dataset类

* 使用smiles\_to\_graph函数提取图结构

class **SIDERDataset**(**Dataset**):

    def **\_\_init\_\_**(self, csv\_path):

        df = **pd**.**read\_csv**(csv\_path)

        self.smiles\_list = df.iloc[:, 0].**tolist**()

        self.labels = df.iloc[:, 1:].values.**astype**(**np**.float32)

        self.num\_classes = self.labels.shape[1]

        self.data\_list = [self.**smiles\_to\_graph**(smiles, label) for smiles, label in **zip**(self.smiles\_list, self.labels)]

        self.data\_list = [**d** for d in self.data\_list if d is not None]

    def **\_\_len\_\_**(self):

        return **len**(self.data\_list)

    def **\_\_getitem\_\_**(self, idx):

        return self.data\_list[idx]

4.2.2 模型初始化与训练

* 初始化 ImprovedGCN 模型（隐藏层维度为 256，5 层卷积）。
* 使用训练集训练 100 个 epoch。
* 每轮验证集上评估 AUROC 和 AUPR，保存最佳模型。

def **train**(model, loader, optimizer, criterion):

    model.train()

    total\_loss = 0

    for data in loader:

        data = data.to(device)

        optimizer.zero\_grad()

        out = model(data)

        target = data.y.view(out.shape[0], -1)

        loss = criterion(out, target)

        loss.backward()

        optimizer.step()

        total\_loss += loss.item()

return total\_loss / **len**(loader)

def **model\_train**():

    train\_dataset = **SIDERDataset**('../Dataset/2\_SIDER/SIDER\_train.csv')

    test\_dataset = **SIDERDataset**('../Dataset/2\_SIDER/SIDER\_test.csv')

    input\_dim = 15

    output\_dim = train\_dataset.num\_classes

    train\_loader = **DataLoader**(train\_dataset, batch\_size=32, shuffle=True)

    test\_loader = **DataLoader**(test\_dataset, batch\_size=32)

**model** = **ImprovedGCN**(input\_dim, 256, output\_dim, num\_layers=5, dropout=0.3).**to**(device)

    optimizer = **torch**.**optim**.**Adam**(**model**.**parameters**(), lr=1e-5, weight\_decay=1e-4)

    scheduler = **torch**.**optim**.**lr\_scheduler**.**ReduceLROnPlateau**(optimizer, mode='max', factor=0.8, patience=5)

**criterion** = **nn**.**BCEWithLogitsLoss**()

    best\_auroc = 0.0

    patience\_counter = 0

    max\_patience = 15

    for epoch in **range**(1, 100):

        loss = **train**(**model**, train\_loader, optimizer, **criterion**)

        auroc, aupr, y\_true, y\_score = **evaluate**(**model**, test\_loader)

        scheduler.**step**(auroc)

**print**(f"Epoch {epoch:03d}: Loss={loss:.4f} | AUROC={auroc:.4f} | AUPR={aupr:.4f}")

        if auroc > best\_auroc and auroc > 0.7:

            best\_auroc = auroc

**torch**.**save**(**model**.**state\_dict**(), "best\_model.pth")

**print**(f"New best model saved with AUROC: {best\_auroc:.4f}")

            patience\_counter = 0

        else:

            patience\_counter += 1

**print**(f"Best AUROC achieved: {best\_auroc:.4f}")

**draw\_curves**(y\_true, y\_score)

4.2.3可视化结果

* 绘制 ROC 曲线与 PR 曲线图。

def **draw\_curves**(y\_true, y\_score):

**plt**.**figure**(figsize=(12, 5))

*# ROC Curve*

**plt**.**subplot**(1, 2, 1)

    fpr, tpr, \_ = **roc\_curve**(y\_true.ravel(), y\_score.ravel())

**plt**.**plot**(fpr, tpr, label='micro-average ROC')

**plt**.**plot**([0, 1], [0, 1], 'k--')

**plt**.**title**("Micro-average ROC Curve")

**plt**.**xlabel**("FPR")

**plt**.**ylabel**("TPR")

**plt**.**legend**()

*# PR Curve*

**plt**.**subplot**(1, 2, 2)

    precision, recall, \_ = **precision\_recall\_curve**(y\_true.ravel(), y\_score.ravel())

**plt**.**plot**(recall, precision, label='micro-average PR')

**plt**.**title**("Micro-average PR Curve")

**plt**.**xlabel**("Recall")

**plt**.**ylabel**("Precision")

**plt**.**legend**()

**plt**.**tight\_layout**()

**plt**.**savefig**("curves\_micro.png")

**plt**.**show**()

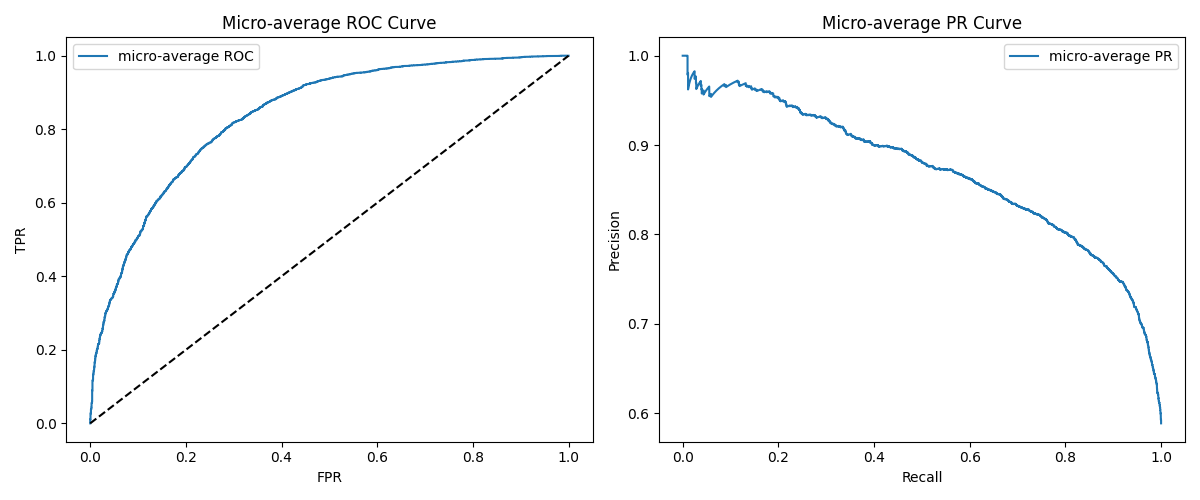
4.3实验结果与分析

* 对比分析：

|  |  |  |
| --- | --- | --- |
| Model Type | AUROC | AUPR |
| 普通MLP | 0.7344 | 0.7658 |
| 普通的GCN模型 | 0.8002 | 0.8342 |
| Ours | **0.8373** | **0.8705** |

相比基础MLP或普通GCN模型，ImprovedGCN能够利用提升预测效果。

* ROC 与 PR 曲线



* 参数实验（学习率）：

|  |  |  |
| --- | --- | --- |
| Learning-rate | AUROC | AUPR |
| 0.0001 | **0.8373** | **0.8705** |
| 0.00001 | 0.8331 | 0.8524 |

尝试了0.0001，0.00001，两种学习率之后发现，0.0001的结果更占优势。

# 参考文献

[1] Kuhn, M., Letunic, I., Jensen, L. J., & Bork, P. (2016). The SIDER database of drugs and side effects. Nucleic Acids Research, 44(D1), D1075–D1079.

[2] Kipf, T. N., & Welling, M. (2016). Semi-Supervised Classification with Graph Convolutional Networks. arXiv preprint arXiv:1609.02907.

[3] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4–24.