

هدف این پروژه تبدیل داده‌های مربوط به ارتباط بین داروها و ژن‌ها به نمایش عددی (feature vector) است. این نمایش عددی به ما امکان می‌دهد از الگوریتم‌های یادگیری ماشین استفاده کنیم و وظایفی مانند پیش‌بینی ارتباط‌های جدید دارو-ژن (Link Prediction)، خوشه‌بندی داروها یا تحلیل شباهت بین داروها و ژن‌ها را انجام دهیم.

برای رسیدن به این هدف، مراحل مشخصی انجام شد که در ادامه توضیح داده می‌شود.

## مرحله ۱: پاکسازی داده‌ها (Data Cleaning)

چه کار کردیم؟

- فایل اولیه ChG-Miner\_miner-chem-gene.tsv را گرفتیم.
- با اسکریپت cleandata.py ستون‌ها را نرمال‌سازی کردیم تا به صورت استاندارد Drug و Gene باشند.
- ردیف‌های ناقص و تکراری حذف شدند.
- شناسه‌های دارو (DrugBank ID) و ژن (UniProt ID) بررسی و اعتبارسنجی شدند.
- خروجی نهایی به نام clean\_dataset.tsv ذخیره شد.

چرا مهم بود؟

- داده‌ی خام اغلب نویز دارد و نامرتب است.
- مدل‌های یادگیری نیازمند داده‌ی تمیز و استاندارد هستند.
- اگر داده‌ها درست پاکسازی نمی‌شدند، گراف به‌درستی ساخته نمی‌شد و embedding‌هایی که روی آن ساخته می‌شدند بی‌معنی بودند.

## مرحله ۲: ساخت گراف دارو-ژن (Graph Construction)

چه کار کردیم؟

- از فایل clean\_dataset.tsv یک گراف دودویی (bipartite graph) ساختیم.
- نودها دو نوع تعریف شدند:
  - داروها (DrugBank ID)
  - ژن‌ها/پروتئین‌ها (UniProt ID)
- هر یال نشان‌دهنده‌ی یک ارتباط دارو-ژن واقعی بود.

چرا مهم بود؟

- برای تولید embedding شبکه‌ای باید یک گراف داشته باشیم.
- گراف ساختار تعاملات دارو-ژن را نشان می‌دهد.
- جایگاه هر دارو و ژن در این شبکه اطلاعات کلیدی درباره‌ی نقش آن‌ها به ما می‌دهد.

---

## مرحله ۳: تولید Node Embedding شبکه‌ای (Graph Embedding)

چه کار کردیم؟

- الگوریتم DeepWalk/Node2Vec را روی گراف اجرا کردیم.
- از هر نود چندین random walk تولید شد (طول مسیر ۵).
- این مسیرها مانند جمله‌های متنی به مدل Word2Vec داده شدند.
- Word2Vec برای هر نود یک بردار ۱۲۸ بعدی تولید کرد.
- خروجی در فایل embeddings.tsv ذخیره شد.

چرا مهم بود؟

- embedding شبکه‌ای موقعیت هر نود را در گراف یاد می‌گیرد.
- داروهایی که به ژن‌های مشابه متصل هستند embedding های مشابه می‌گیرند.
- این embedding ها اولین نمایش عددی از ساختار شبکه‌ای را به ما دادند و پایه‌ای برای مراحل بعدی شدند.

---

## مرحله ۴: افزودن ویژگی‌های شیمیایی (SMILES Features)

چه کار کردیم؟

- یک فایل جداگانه شامل SMILES (نمایش ساختاری داروها) برای حدود ۵۰۰۰ دارو داشتیم.
- این فایل با دیتاست اصلی merge شد تا هر دارو، در صورت وجود، ستون SMILES داشته باشد.
- با کتابخانه RDKit از SMILES ها fingerprint مولکولی (ECFP4) استخراج کردیم.
- fingerprint ها بردارهای باینری ۲۰۴۸ بعدی بودند که زیرساختارهای شیمیایی داروها را نمایش می‌دادند.
- خروجی در فایل drug\_fp.tsv ذخیره شد.

چرا مهم بود؟

- embedding شبکه‌ای فقط ساختار ارتباطات در گراف را یاد می‌گیرد.
- اما داروها ویژگی‌های ذاتی شیمیایی هم دارند.
- اضافه کردن fingerprint شیمیایی باعث شد مدل هم شبکه و هم شیمی دارو را ببیند و دانش غنی‌تری از داروها داشته باشد.

---

## مرحله ۵: ادغام ویژگی‌ها (Feature Fusion)

چه کار کردیم؟

- برای هر دارو دو embedding داشتیم:
  1. embedding شبکه‌ای (emb\_deepwalk.tsv)
  2. embedding شیمیایی (drug\_fp.tsv)
- این دو embedding را با هم ترکیب (fusion) کردیم:

- Concatenation (cat): دو بردار را پشت سر هم گذاشتیم → خروجی `emb_fused_cat.tsv`.
- Summation (sum): بردارها را جمع/میانگین گرفتیم → خروجی `emb_fused_sum.tsv`.

چرا مهم بود؟

- هر منبع ویژگی بخشی از واقعیت را نشان می‌دهد (شبکه‌ای + شیمیایی).
- با ادغام، یک بردار قوی‌تر می‌سازیم که اطلاعات بیشتری دارد.
- `concat` همه‌ی اطلاعات را نگه می‌دارد (ولی ابعادش بزرگتر می‌شود).
- `sum` ابعاد را کوچکتر نگه می‌دارد، اما ممکن است جزئیات ظریف از بین برود.

## مرحله ۶: وضعیت ویژگی‌ها برای داروها و ژن‌ها

داروها:

- `embedding` شبکه‌ای (برای همه داروها)
- `embedding` شیمیایی (فقط برای داروهایی که SMILES داشتند)
- بردار ادغام‌شده (`fusion`)

ژن‌ها/پروتئین‌ها:

- فقط `embedding` شبکه‌ای (چون داده‌ی زیستی دیگری در دسترس نبود).

چرا این انتخاب درست است؟

- داروها داده‌ی شیمیایی داشتند، بنابراین از آن‌ها استفاده کردیم.
- ژن‌ها داده‌ی شیمیایی نداشتند، پس همان `embedding` شبکه‌ای کافی بود.

## مرحله ۷: تعدیل داده‌ها (Balancing)

یکی از چالش‌های اساسی در تحلیل داده‌های زیستی، نامتوازن بودن (Imbalance) داده‌هاست. در مسئله‌ی ما که بر پایه‌ی گراف دارو-ژن طراحی شده، تعداد ارتباط‌های واقعی (مثبت) بین داروها و ژن‌ها بسیار کمتر از تمام ترکیب‌های ممکن دارو-ژن است. این موضوع باعث می‌شود اگر مدل یادگیری را مستقیماً روی این داده‌ها آموزش دهیم، به شدت به سمت پیش‌بینی کلاس غالب (یعنی عدم ارتباط یا همان منفی) سوگیری پیدا کند. برای رفع این مشکل، باید داده‌ها را متعادل (balanced) کنیم؛ یعنی تعداد نمونه‌های مثبت و منفی را برابر یا نزدیک به برابر قرار دهیم. این کار به مدل اجازه می‌دهد الگوهای واقعی تعاملات دارو-ژن را یاد بگیرد.

برای پیاده‌سازی این بخش، اسکریپت `make_balanced_dataset.py` طراحی شد. این اسکریپت ابتدا لایه‌های مثبت (ارتباط‌های واقعی دارو-ژن) را از دیتاست اصلی می‌گیرد. سپس با انتخاب تصادفی جفت‌های دارو-ژن که در دیتاست مثبت وجود نداشتند، نمونه‌های منفی ساخته می‌شود. تعداد این منفی‌ها بر اساس نسبت انتخاب‌شده (برای مثال ۱:۱) تعیین می‌گردد. در ادامه، داده‌های مثبت و منفی با هم ترکیب شده و به صورت تصادفی (`shuffle`) مرتب می‌شوند تا ترتیب خاصی نداشته باشند. خروجی این بخش فایل `balanced_pairs.tsv` است که شامل سه

ستون اصلی یعنی دارو، ژن و برجسب (label) است؛ به طوری که ۱ نشان‌دهنده‌ی ارتباط واقعی و ۰ نشان‌دهنده‌ی ارتباط غیر واقعی است.

در صورتی که embedding نودها در دسترس باشد، اسکرپیت قادر است برای هر جفت دارو-ژن یک بردار ویژگی نیز تولید کند. این کار با ادغام embedding (concatenation) دارو و ژن انجام می‌شود و خروجی در فایل `balanced_features.tsv` ذخیره می‌گردد. این فایل آماده‌ی استفاده در مدل‌های یادگیری است زیرا علاوه بر ستون‌های دارو، ژن و برجسب، شامل ویژگی‌های عددی نهایی (`f0, f1, ...`) برای هر جفت نیز می‌باشد.

مرحله‌ی **Balancing** اهمیت زیادی دارد زیرا از سوگیری مدل به سمت کلاس غالب جلوگیری می‌کند، باعث می‌شود مدل الگوهای واقعی تعامل دارو-ژن را یاد بگیرد و معیارهای ارزیابی مانند **Precision**، **Recall** و **F1-Score** معنی‌دارتر و دقیق‌تر شوند. در نتیجه، این مرحله یک دیتاست آموزشی متوازن و قابل‌اعتماد برای مدل‌سازی فراهم می‌کند و پایه‌ی محکمی برای پیش‌بینی تعاملات جدید دارو-ژن به وجود می‌آورد.

## مرحله ۸: کاهش ابعاد (Dimensionality Reduction)

بعد از ادغام ویژگی‌های شبکه‌ای و شیمیایی، به‌خصوص با روش **Concatenation**، بُعد بردار ویژگی‌ها بسیار زیاد شد. به عنوان مثال، embedding شبکه‌ای ۱۲۸ بعدی و fingerprint شیمیایی ۱۰۲۴ یا ۲۰۴۸ بعدی بودند. وقتی این دو کنار هم قرار گرفتند، طول بردارها به بیش از ۱۱۰۰ یا حتی ۲۲۰۰ بعد رسید. چنین ابعادی مشکلاتی به وجود می‌آورد:

- محاسبات یادگیری ماشین بسیار سنگین و زمان‌بر می‌شوند.
- خطر **Overfitting** بالا می‌رود، چون مدل روی نویز و جزئیات غیرمهم حساس می‌شود.
- بسیاری از ویژگی‌ها تکراری یا کم‌اهمیت هستند و فقط باعث شلوغی داده می‌شوند.
- تفسیر و تحلیل داده‌ها نیز سخت‌تر می‌شود.

برای حل این مشکل از روش **PCA (Principal Component Analysis)** استفاده شد. این روش ابتدا داده‌ها را استاندارد می‌کند (میانگین صفر، واریانس یک) و سپس محورهای جدیدی به نام مؤلفه‌های اصلی **Principal Components** می‌سازد. این محورها به گونه‌ای انتخاب می‌شوند که بیشترین واریانس داده را در کمترین بعد ممکن حفظ کنند. به این ترتیب، داده‌هایی که مثلاً ۱۱۵۲ بعد داشتند، می‌توانند به ۱۲۸ یا حتی ۱۰۰ بعد کاهش پیدا کنند، در حالی که بخش زیادی از اطلاعات اصلی همچنان باقی بماند.

انتخاب تعداد بعد نهایی به دو روش انجام شد:

1. انتخاب مستقیم تعداد مؤلفه‌ها
2. بر اساس درصد واریانس، به طوری که مثلاً ۹۵٪ اطلاعات حفظ شود و PCA خودش تعداد مؤلفه‌ها را تعیین کند.

خروجی این مرحله یک فایل embedding جدید مثل `emb_fused_cat_pca.tsv` بود که شامل همان ستون‌های متادیتا مانند `node` و `type` و ستون‌های جدید ویژگی (`p0, p1, ...`) می‌شد.

این مرحله بسیار مهم بود زیرا:

- سرعت آموزش مدل‌ها را افزایش داد.
- خطر **Overfitting** را کاهش داد چون ویژگی‌های نویزی یا تکراری حذف شدند.
- کیفیت و دقت مدل بهبود پیدا کرد چون مدل روی اطلاعات اصلی تمرکز کرد

- داده‌ها به شکل فشرده‌تر و ساده‌تری آماده‌ی ورود به مرحله‌ی مدل‌سازی شدند.

ما از PCA استفاده کردیم چون سریع، ساده، پرکاربرد و مناسب داده‌های **embedding** عددی هست. روش‌های دیگه یا خیلی کند بودن (t-SNE, UMAP)، یا پیچیده و سنگین (Autoencoder)، یا مناسب نوع داده‌ی ما نبودن.

می‌ریم سراغ بخش اصلی که آموزش مدل هست:

### Random forest:

در مرحله‌ی اول من از یک مدل **Random Forest** استفاده کردم. برای این کار ابتدا داده‌های نهایی که شامل ویژگی‌های استخراج‌شده و کاهش‌یافته بودند فایل `balanced_features_pca.tsv` را بارگذاری کردم. سپس داده‌ها را به دو بخش ۸۰٪ برای آموزش و ۲۰٪ برای آزمون تقسیم کردم تا مطمئن شوم مدل علاوه بر یادگیری روی داده‌های آموزش، توانایی تعمیم روی داده‌های جدید را هم دارد.

مدل Random Forest با ۲۰۰ درخت تصمیم (`n_estimators=200`) و عمق حداکثر ۲۰ (`max_depth=20`) آموزش داده شد. همچنین برای جلوگیری از سوگیری به دلیل احتمالی نامتوازن بودن داده‌ها، پارامتر `class_weight="balanced"` تنظیم گردید. پس از آموزش، مدل روی داده‌های آزمون ارزیابی شد و نتایج زیر به‌دست آمد:

- **Accuracy:** حدود ۹۳٪
- **Precision:** حدود ۹۵٪
- **Recall:** حدود ۹۱٪
- **F1-score:** حدود ۹۳٪
- **ROC-AUC:** حدود ۹۱٪

این نتایج نشان داد که مدل توانایی بالایی در شناسایی ارتباط‌های واقعی دارو-ژن دارد و همزمان نرخ خطا در پیش‌بینی‌های مثبت (False Positive) پایین است. تعادل خوب بین Precision و Recall نیز نشان‌دهنده‌ی عملکرد پایدار و متوازن مدل در این مسئله است.

```
Accuracy : 0.9303170409511229
Precision: 0.9499309392265194
Recall    : 0.9085204755614267
F1-score  : 0.9287643484132343
ROC-AUC   : 0.9808416252362364
(base)
```

### • تناسب با داده‌های tabular جدولی:

داده‌های ما به شکل feature vector هستند embedding شبکه‌ای + ویژگی‌های شیمیایی Random Forest. برای این نوع داده‌ها عالی عمل می‌کند.

### • توانایی کار با ویژگی‌های زیاد و پرنویز:

در داده‌های ما ممکن است بعضی ابعاد embedding مهم نباشند یا نویزی باشند Random Forest. به دلیل ماهیت درختی‌اش به‌خوبی می‌تواند ویژگی‌های مهم را شناسایی کند و نسبت به نویز مقاوم است.

### ● کاهش خطر: Overfitting:

چون Random Forest از تعداد زیادی درخت تصمیم استفاده می‌کند و خروجی را میانگین می‌گیرد، نسبت به مدل‌های تک‌درختی کمتر دچار overfitting می‌شود.

### ● قابلیت تفسیر:

این مدل می‌تواند **Feature Importance** بدهد و نشان دهد کدام ویژگی‌ها در تصمیم‌گیری مهم‌تر بوده‌اند. این موضوع در تحلیل زیستی کاربردی است.

## SVM:

## مدل دوم (Support Vector Machine) با RBF Kernel

در ادامه‌ی کار، پس از اجرای مدل Random Forest، از یک مدل SVM با **kernel شعاعی (RBF)** استفاده شد. ورودی این مدل فایل `balanced_features_pca.tsv` بود که در آن ویژگی‌ها پس از کاهش بعد با PCA ذخیره شده بودند. همانند مدل قبلی، داده‌ها به دو بخش آموزش (**Train**) و آزمون (**Test**) تقسیم شدند تا بتوان عملکرد مدل را به صورت منصفانه ارزیابی کرد.

نتایج به دست آمده:

- **Accuracy: 0.942** یعنی 94.2٪ از نمونه‌ها به درستی طبقه‌بندی شدند.
- **F1-score: 0.942** نشان‌دهنده تعادل بسیار خوب بین Precision و Recall است.
- **ROC-AUC: 0.987** نشان‌دهنده توانایی بسیار بالای مدل در تمایز بین ارتباط‌های واقعی و غیرواقعی.
- **PR-AUC: 0.987** نشان‌دهنده کیفیت عالی مدل در بازیابی نمونه‌های مثبت.

```
(.venv310) (base) shadizargarzadeh@Shadis-MacBook-Pro Step 2 % python3 train_svm.py --in balanced_features_pca.tsv --save-model svm_pca.joblib
(.venv310) (base) shadizargarzadeh@Shadis-MacBook-Pro Step 2 %

=== SVM (RBF) ===
Accuracy: 0.9422 | F1: 0.9420 | ROC-AUC: 0.9871 | PR-AUC: 0.9871
Confusion matrix:
[[2866 162]
 [188 2840]]
Classification report:
      precision    recall  f1-score   support

      0       0.9384       0.9465       0.9425       3028
      1       0.9460       0.9379       0.9420       3028

   accuracy       0.9422       0.9422       0.9422       6056
  macro avg       0.9422       0.9422       0.9422       6056
weighted avg       0.9422       0.9422       0.9422       6056

[OK] Saved model -> svm_pca.joblib
[INFO] Train size = 24222, Test size = 6056
(.venv310) (base) shadizargarzadeh@Shadis-MacBook-Pro Step 2 %
```

## SVM (Support Vector Machine) علت انتخاب

1. قدرت در داده‌های با بعد بالا:  
SVM به‌ویژه با **kernel RBF** در داده‌هایی که تعداد ویژگی‌ها زیاد است (مثل **embedding** های ما)، عملکرد بسیار خوبی دارد.
2. توانایی جداسازی داده‌های غیرخطی:  
روابط بین دارو و ژن پیچیده و غیرخطی هستند SVM با **kernel RBF** می‌تواند مرزهای تصمیم‌گیری غیرخطی را یاد بگیرد.

3. کارایی بالا در دیتاست‌های نسبتاً کوچک/متوسط:  
در پروژه ما تعداد نمونه‌ها خیلی بزرگ نیست (چند ده هزار جفت دارو-ژن)، SVM در چنین شرایطی معمولاً از شبکه‌های عمیق بهتر عمل می‌کند.
4. نتایج پایدار و شناخته‌شده در بیوانفورماتیک:  
SVM یکی از پرکاربردترین الگوریتم‌ها در مسائل زیستی (مثل طبقه‌بندی ژن و دارو) است و نتایج آن معمولاً قوی و پایدار است.

### Random Forest with Cross Validation:

برای ارزیابی دقیق‌تر مدل Random Forest، به جای یک بار تقسیم Train/Test، از روش **5-Fold Stratified Cross Validation** استفاده شد. در این روش داده‌ها به ۵ بخش مساوی تقسیم شدند، هر بار یکی از بخش‌ها به عنوان داده‌ی آزمون استفاده شد و مدل روی چهار بخش دیگر آموزش دید. در پایان، میانگین و انحراف معیارهای ارزیابی محاسبه شد.

- **Accuracy:  $0.926 \pm 0.006$** 
  - یعنی مدل به طور میانگین حدود ۹۳٪ نمونه‌ها را به درستی طبقه‌بندی کرده است.
  - انحراف معیار پایین (۰/۶٪) نشان می‌دهد عملکرد مدل پایدار است و وابسته به یک تقسیم خاص از داده‌ها نیست.
- **Precision:  $0.948 \pm 0.007$** 
  - یعنی از پیش‌بینی‌های مثبت مدل، حدود ۹۵٪ درست بوده‌اند.
  - این نشان می‌دهد خطای False Positive بسیار پایین است.
- **Recall:  $0.902 \pm 0.008$** 
  - یعنی مدل توانسته حدود ۹۰٪ از تمام نمونه‌های مثبت واقعی را پیدا کند.
  - این معیار مهمی در مسائل زیستی است چون نمی‌خواهیم تعامل واقعی دارو-ژن از دست بره.
- **F1-score:  $0.924 \pm 0.006$** 
  - میانگین هماهنگ Precision و Recall → نشان‌دهنده‌ی تعادل عالی بین این دو.
- **ROC-AUC:  $0.980 \pm 0.002$** 
  - بسیار بالا (نزدیک ۱)، یعنی مدل قدرت تفکیک بسیار بالایی بین کلاس مثبت و منفی دارد.
  - انحراف معیار بسیار کم → پایدار بودن مدل.

1. مقادیر Precision و Recall نزدیک به هم هستند، که یعنی مدل هم توانایی بالایی در شناسایی مثبت‌های واقعی دارد (Recall) و هم پیش‌بینی‌های مثبتش قابل اعتمادند. (Precision)
2. مقدار بالای ROC-AUC (0.98) نشان می‌دهد مدل تقریباً به بهترین حد ممکن در جداسازی مثبت‌ها از منفی‌ها رسیده است.
3. انحراف معیارهای پایین در همه معیارها نشان می‌دهند مدل روی همه‌ی تقسیم‌بندی‌های مختلف داده‌ها به طور مشابه خوب عمل کرده و نتیجه تصادفی یا وابسته به یک Split خاص نیست.

## SVM with Cross Validation:

نتایج گزارش شده

- **Accuracy =  $0.9495 \pm 0.0036$**   
→ یعنی مدل در حدود 95% کل نمونه‌ها رو درست پیش‌بینی کرده، با واریانس خیلی کم (پایداری خوب بین فولدها).
- **F1-score =  $0.9493 \pm 0.0036$**   
→ این مقدار نزدیک به Accuracy هست، یعنی تعادل خوبی بین Precision و Recall وجود داره.
- **ROC-AUC =  $0.9884 \pm 0.0010$**   
→ مقدار خیلی بالاست (نزدیک به 1). این نشون می‌ده که مدل توانایی خیلی خوبی در تفکیک کلاس مثبت و منفی داره.
- **Average Precision (AUPRC) =  $0.9882 \pm 0.0007$**   
→ این متریک برای داده‌های نامتوازن خیلی مهمه. مقدار نزدیک به 0.99 یعنی مدل به‌خوبی تونسته نمونه‌های مثبت واقعی رو تشخیص بده.

## GNN:

• در ایپاک‌های ابتدایی (مثلاً ایپاک 25):

- **AUPRC  $\approx 0.52$**
  - **AUROC  $\approx 0.80$**
- مدل تازه شروع کرده و هنوز قدرت تفکیک خوبی روی داده‌ها نداره.

با پیشرفت ایپاک‌ها، **AUPRC** به شکل پیوسته رشد می‌کنه و از 0.52 به حدود 0.85 می‌رسه. این نشون می‌ده که مدل داره به‌مرور نمونه‌های مثبت واقعی رو خیلی بهتر تشخیص می‌ده.

**AUROC** در کل روند پایدار بوده (حدود 0.80 تا 0.81) یعنی توانایی مدل در تفکیک کلی مثبت‌ها از منفی‌ها ثابت و خوب باقی مونده



## بهترین عملکرد ثبت شده

**AUPRC = 0.8592 •**

**AUROC = 0.8124 •**

این نتایج اهمیت زیادی دارند چون:

- **AUPRC بالا** ( $\approx 0.86$ ) یعنی مدل در داده‌های نامتوازن (که مثبت‌ها کم هستند) دقت خیلی خوبی در تشخیص مثبت‌ها دارد. این متریک از Accuracy یا حتی AUROC مهم‌تره چون روی داده‌های بایسته (imbalanced) تمرکز روی مثبت‌ها حیاتی‌تره.
- **AUROC حدود 0.81** نشون می‌ده که مدل در تفکیک کلی مثبت و منفی هم توانمند هست، هرچند بیشترین بهبود نسبت به baseline در AUPRC دیده می‌شه.

```
70 full_nodes = test_attn_nodes
71
OUTPUT TEROSHDL: LOG REPORT TEROSHDL: TIMING PROBLEMS DEBUG CONSOLE TERMINAL PORTS GITLENS zsh - Step 2
Epoch 025 | loss=0.5261 | val AUPRC=0.8577 AUROC=0.8072
Epoch 030 | loss=0.5169 | val AUPRC=0.8553 AUROC=0.8033
Epoch 035 | loss=0.5076 | val AUPRC=0.8573 AUROC=0.8061
Epoch 040 | loss=0.4986 | val AUPRC=0.8566 AUROC=0.8047
Epoch 045 | loss=0.4852 | val AUPRC=0.8575 AUROC=0.8065
Epoch 050 | loss=0.4744 | val AUPRC=0.8570 AUROC=0.8052
Epoch 055 | loss=0.4592 | val AUPRC=0.8577 AUROC=0.8073
Epoch 060 | loss=0.4522 | val AUPRC=0.8567 AUROC=0.8059
Epoch 065 | loss=0.4382 | val AUPRC=0.8570 AUROC=0.8078
Epoch 070 | loss=0.4263 | val AUPRC=0.8602 AUROC=0.8166
Epoch 075 | loss=0.4220 | val AUPRC=0.8613 AUROC=0.8194
Epoch 080 | loss=0.4099 | val AUPRC=0.8592 AUROC=0.8142
Epoch 085 | loss=0.4009 | val AUPRC=0.8603 AUROC=0.8171
Epoch 090 | loss=0.3935 | val AUPRC=0.8646 AUROC=0.8238
Epoch 095 | loss=0.3818 | val AUPRC=0.8684 AUROC=0.8297
Epoch 100 | loss=0.3773 | val AUPRC=0.8677 AUROC=0.8275
TEST - AUPRC=0.8592 | AUROC=0.8112
Saved predictions to predictions_test.csv
(.venv310) (base) shadizargarzadeh@Shadis-MacBook-Pro Step 2 %
```

MLP:

- در طول آموزش تا اپیاک 29،  $\text{train accuracy} \approx 0.97$  و  $\text{val accuracy} \approx 0.97$  بوده.
  - معیارهای نهایی روی داده تست:
    - $\text{Accuracy} = 0.9625$
    - $\text{Precision} = 0.9532$
    - $\text{Recall} = 0.9661$
    - $\text{F1-score} = 0.9596$
    - $\text{ROC-AUC} = 0.9934$
  - **ROC-AUC خیلی بالا (0.993)**
- یعنی MLP توانسته مرز تصمیم‌گیری بسیار دقیق بین کلاس مثبت و منفی ایجاد کند. این مقدار بالاتر از GNN ( $\approx 0.81$ ) است.
- **دقت کلی ( $\text{Accuracy} \approx 0.96$ )**
- این نشون می‌ده مدل روی کل داده‌ها، هم مثبت و هم منفی، بسیار خوب عمل کرده.
- **Precision و Recall نزدیک به هم ( $0.95-0.96$ )**
- یعنی هم خطای False Positive کم بوده، هم False Negative. تعادل عالی بین precision و recall باعث شده F1-score هم بالا بره.

## • GNN

- $AUPRC \approx 0.86$
- $AUROC \approx 0.81$
- تمرکز بیشتر روی کشف ساختار گراف و ارتباط بین دارو-ژن ها.
- قوی‌تر در داده‌های نامتوازن (جایی که مثبت‌ها کم هستند).

## • MLP

- $ROC-AUC \approx 0.99$
- $Accuracy \approx 0.96$
- Precision, Recall, F1 همگی بالا. ( $\approx 0.95-0.96$ )
- عملکرد بسیار خوب در سطح کلی داده، حتی بهتر از GNN از نظر ROC-AUC.

## MLP With Cross Validation:

مدل **MLP** با دو لایه پنهان ۲۵۶ و ۱۲۸ نرون، dropout برابر ۰/۲، ۵۰ اپیاک و batch size برابر ۳۲ آموزش داده شد. نرخ یادگیری ۰/۰۰۱ و الگوریتم Adam برای بهینه‌سازی استفاده شد. این مدل با ۵-فولد کراس‌ولیدیشن ارزیابی گردید.

نتایج نشان داد که مدل عملکرد بسیار قوی و پایدار دارد. **دقت (Accuracy)** برابر با ۰/۹۶۰۵ به دست آمد که یعنی حدود ۹۶ درصد کل نمونه‌ها درست پیش‌بینی شدند. **Precision** برابر با ۰/۹۵۳۶ بود که نشان می‌دهد بیشتر پیش‌بینی‌های مثبت مدل درست بوده و خطای مثبت کاذب پایین است. **Recall** برابر با ۰/۹۶۸۳ به دست آمد که یعنی مدل توانست تقریباً همه‌ی نمونه‌های مثبت واقعی را شناسایی کند و خطای منفی کاذب بسیار کم بود. ترکیب این دو معیار باعث شد **F1-score** به ۰/۹۶۰۹ برسد که بیانگر تعادل عالی بین Precision و Recall است.

در نهایت، **ROC-AUC** برابر با ۰/۹۹۲۰ به دست آمد. این مقدار نزدیک به ۱ نشان می‌دهد که مدل تقریباً مرز تصمیم‌گیری بی‌نقصی بین کلاس مثبت و منفی پیدا کرده است.

```
(base) shadizargazadeh@Shadis-MacBook-Pro Step 3 % python MLPWithCross.py \
--features_path "/Users/shadizargazadeh/Desktop/ci final proj/Step 3/balanced_features_pca.tsv" \
--k 5 \
--epochs 50 \
--batch_size 32 \
--patience 5 \
--hidden_layers 256,128 \
--dropout 0.2 \
--lr 0.001 \
--out_dir "./cv_runs_step3"

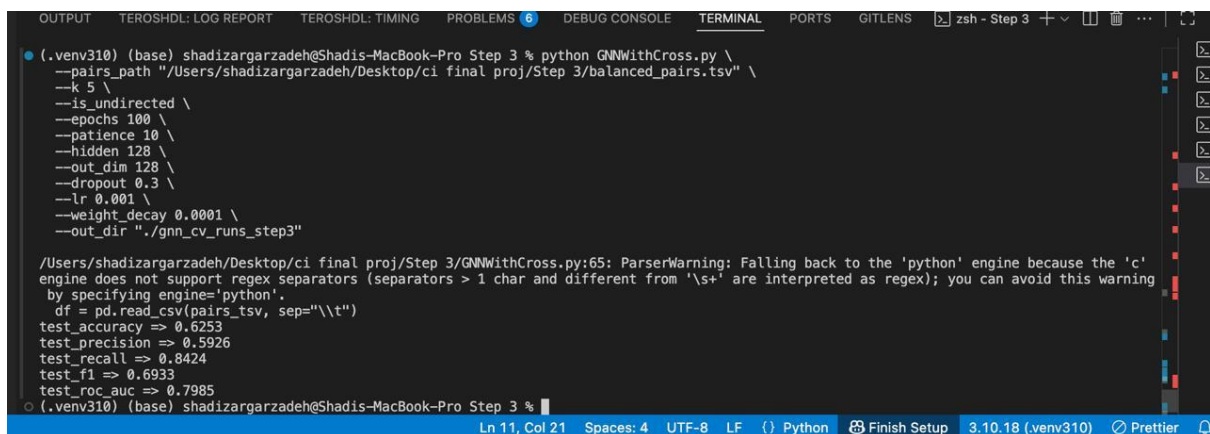
/Users/shadizargazadeh/Desktop/ci final proj/Step 3/MLPWithCross.py:47: ParserWarning: Falling back to the 'python' engine because the 'c'
engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning
by specifying engine='python'.
  df = pd.read_csv(path, sep="\t")
test_accuracy => 0.9605
test_precision => 0.9536
test_recall => 0.9683
test_f1 => 0.9609
test_roc_auc => 0.9920
```

## GNN with Crossvalidation:

مدل **GNN** با دو لایه کانولوشن گراف (GCNConv)، ساینز مخفی ۱۲۸، dropout برابر ۰/۳، ۱۰۰ اپیاک و نرخ یادگیری ۰/۰۰۱ آموزش داده شد. وزن منظم‌ساز (weight decay) برابر با ۰/۰۰۰۱ در نظر گرفته شد و مدل با ۵-فولد کراس‌ولیدیشن مورد ارزیابی قرار گرفت.

نتایج نشان داد که دقت کلی (**Accuracy**) برابر با ۰/۶۲۵۳ بوده است. این مقدار پایین است و نشان می‌دهد که مدل تنها حدود ۶۲ درصد نمونه‌ها را درست پیش‌بینی کرده است. **Precision** برابر با ۰/۵۹۲۶ به دست آمد، به این معنا که تنها حدود ۵۹ درصد از پیش‌بینی‌های مثبت واقعاً مثبت بوده‌اند و مدل دچار خطای مثبت کاذب بالا شده است. در مقابل، **Recall** برابر با ۰/۸۴۲۴ بود که نشان می‌دهد مدل توانسته بخش بزرگی از نمونه‌های مثبت واقعی را شناسایی کند و در پیدا کردن مثبت‌ها موفق‌تر از دقت در مثبت‌های پیش‌بینی‌شده بوده است. همین اختلاف باعث شد که **F1-score** به ۰/۶۹۳۳ برسد، که متوسط و نشان‌دهنده تعادل نه‌چندان خوب بین Precision و Recall است.

در نهایت، **ROC-AUC** برابر با ۰/۷۹۸۵ به دست آمد که نشان می‌دهد توانایی مدل در تفکیک کلی کلاس مثبت و منفی متوسط بوده و با فاصله زیادی از عملکرد نزدیک به ایده‌آل مدل‌های دیگر قرار دارد.



```
(.venv310) (base) shadizargarzadeh@Shadis-MacBook-Pro Step 3 % python GNNWithCross.py \
--pairs_path "/Users/shadizargarzadeh/Desktop/ci final proj/Step 3/balanced_pairs.tsv" \
--k 5 \
--is_undirected \
--epochs 100 \
--patience 10 \
--hidden 128 \
--out_dim 128 \
--dropout 0.3 \
--lr 0.001 \
--weight_decay 0.0001 \
--out_dir "./gnn_cv_runs_step3"

/Users/shadizargarzadeh/Desktop/ci final proj/Step 3/GNNWithCross.py:65: ParserWarning: Falling back to the 'python' engine because the 'c'
engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning
by specifying engine='python'.
  df = pd.read_csv(pairs_tsv, sep="\t")
test_accuracy => 0.6253
test_precision => 0.5926
test_recall => 0.8424
test_f1 => 0.6933
test_roc_auc => 0.7985
(.venv310) (base) shadizargarzadeh@Shadis-MacBook-Pro Step 3 %
```

