

به نام خدا

درس تحلیل کلان داده ها

تمرین سری سوم

سنا موحدین ۹۹۳۱۰۸۰

## بخش اول)

سوال ۱:

.a

concept drift به وقوع تغییر در توزیع داده‌ها یا رابطه‌های میان متغیرها در یک محیط زمان‌بندی شده اشاره دارد. به عبارت دیگر، در یک سیستم مدل‌سازی شده، ممکن است داده‌ها به مرور زمان تغییر کنند، و به همین دلیل مدلی که از داده‌های قدیمی یاد گرفته شده است ممکن است دیگر به طور دقیق مفهوم جدید را نتواند توصیف کند. این تغییرات ممکن است به شکل تدریجی و پیوسته رخ دهند یا به صورت ناگهانی و ناگهانی اتفاق بیافتد.

الگوریتم CVFDT (Concept-adapting Very Fast Decision Tree) یک الگوریتم یادگیری برای داده‌های جریانی است که قابلیت تطبیق با تغییرات در مفهوم داده‌ها را دارد. CVFDT از چندین روش برای مقابله با concept drift استفاده می‌کند. ۳ تا از مهمترین این راه حل‌ها به این صورت هستند:

۱. chunking: داده‌های ورودی را به صورت پنجره‌ای یا چانک (Chunk) پردازش می‌کند. این به الگوریتم اجازه می‌دهد تا تغییرات در توزیع داده‌ها را از طریق مقایسه آمارهای مختلف مانند توزیع کلاس بین چانک‌های متوالی تشخیص دهد. هنگامی که تغییرات معنی‌دار مشاهده می‌شود، الگوریتم فرآیندی را آغاز می‌کند تا ساختار درخت تصمیم را به منظور تطبیق با مفهوم جدید تغییر دهد.

۲. adaptive tree growth: به صورت پویا درخت‌های تصمیم را رشد می‌دهد در حالی که داده‌های جدید وارد می‌شوند. هنگامی که تغییر مفهوم مشاهده می‌شود، شاخه‌های جدید ایجاد می‌شود یا شاخه‌های موجود قطع می‌شود تا ساختار درخت تصمیم را برای سازگاری با توزیع داده‌های جدید تغییر دهد. این مکانیسم رشد تطبیقی به CVFDT کمک می‌کند تا دقت خود را در حضور تغییر مفهوم حفظ کند.

۳. CVFDT-selective retraining: قسمت‌های مختلفی از درخت تصمیم که تحت تاثیر تغییر مفهوم هستند را به جای آموزش مجدد کلیه درخت انتخاب می‌کند. این رویکرد آموزش انتخابی میزان هزینه محاسباتی را کاهش می‌دهد و به CVFDT این امکان را می‌دهد که به سرعت به مفهوم‌های جدید در داده‌های جریانی تطبیق یابد بدون نیاز به منابع گسترشده. با تمرکز بر روی بخش‌های مهم‌تر درخت، CVFDT مدل خود را به توزیع داده‌های فعلی به روز می‌کند.

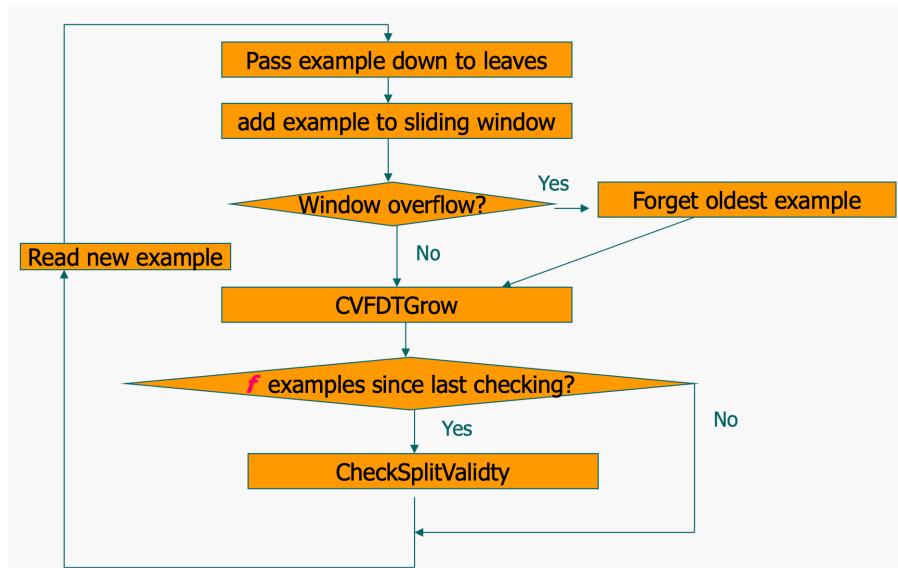
b. به طور خلاصه، هر دو الگوریتم VFDT و CVFDT برای ساخت درخت تصمیم از داده‌های جریانی طراحی شده‌اند. ولی با اضافه کردن مکانیسم‌های مقابله با concept drift در جاهایی که به مرور زمان تغییر داریم، مناسب‌تر است. پس تفاوت اصلی بین آن‌ها در توانایی مقابله با تغییر مفهوم در داده‌های جریانی است.

به طور دقیقتر می‌توانیم بگوییم با توجه به این که VFDT قابلیت تطبیق با تغییرات در توزیع داده‌ها را ندارد، ممکن است دچار افت دقت شود زمانی که مفهوم در داده‌های جریانی تغییر کند. از سوی دیگر، CVFDT یک توسعه از VFDT است که به طور خاص به مقابله با تغییر مفهوم می‌پردازد. این الگوریتم مکانیسم‌هایی را برای تطبیق ساختار درخت تصمیم به تغییرات در توزیع داده‌ها اضافه می‌کند، این باعث می‌شود که درخت تصمیم همیشه دقیق و به روز باشد هنگامی که مفهوم در داده‌های جریانی تغییر می‌کند.

الگوریتم VFDT درخت‌های تصمیم را بر اساس داده‌های مشاهده شده ساخته می‌کند، اما ساختار خود را به طور فعال به منظور سازگاری با تغییرات در مفهوم زیرین تغییر نمی‌دهد.

ولی CVFDT ساختار درخت تصمیم خود را به طور پویا به تغییرات در مفهوم تطبیق می‌دهد. این الگوریتم از تکنیک‌هایی مانند chunking، رشد درخت تطبیقی و آموزش انتخابی برای شناسایی و پاسخ به تغییر مفهوم استفاده می‌کند، این باعث می‌شود که درخت تصمیم همیشه دقیق و به روز باشد هنگامی که توزیع داده‌ها تغییر می‌کند.

c. مراحل الگوریتم CVFDT را می‌توان به این صورت نشان داد:



به طور دقیق‌تر، الگوریتم به این صورت عمل می‌کند:  
altemate tree  
های برای هر گره در HT به صورت خالی شروع می‌شوند.  
نمونه‌هایی از جریان را به طور نامحدود پردازش می‌کنیم. برای هر نمونه‌ی (X,Y):  
(X,Y) را با استفاده از HT به مجموعه‌ای از برگ‌ها منتقل می‌کنیم و تمام alternate tree های گره‌های (X,Y) از آن عبور می‌کنند.

(X,Y) را به پنجره کشویی (sliding window) نمونه‌ها اضافه می‌کنیم.  
اگر پنجره کشویی سریز شد، قدیمی‌ترین نمونه‌ها را حذف می‌کنیم.  
بخش CVFDT Grow صورت می‌گیرد.  
اگر f نمونه از آخرین بررسی alternative tree های دیده شده است، check split validity انجام می‌شود.  
در نهایت وقتی برای همه‌ی (X,Y) ها فرایند انجام شد، HT برگدانده می‌شود.

## سوال: ۲

a. یادگیری تدریجی روشی از یادگیری ماشین است که در آن یک مدل هوش مصنوعی دانش خود را به تدریج و بدون فراموش کردن اطلاعات کسب شده قبلی یاد می‌گیرد و افزایش می‌دهد. در اصل، الگوهای یادگیری انسان را با به دست آوردن اطلاعات جدید در طول زمان تقليید می‌کند، در حالی که دانش قبلی را حفظ و توسعه می‌دهد. یادگیری تدریجی در سناریوهایی که داده‌ها به ترتیب به دست می‌آیند یا در جایی که ذخیره همه داده‌ها برای پردازش امکان‌پذیر نیست بسیار مهم است. برای مثال، یک مدل فیلتر ایمیل اسپم را در نظر بگیرید. با batch learning مدل با مجموعه بزرگی از ایمیل‌ها به طور همزمان آموزش داده می‌شود و سپس برای ایمیل‌های بعدی اعمال می‌شود. اگر ماهیت ایمیل‌های اسپم تغییر کند، ممکن است مدل شکست بخورد، مگر آنکه در دسته‌ای از ایمیل‌های جدید که ویژگی‌های اسپم بهروز شده را شامل است دوباره آموزش داده شود. همچنین یک فیلتر اسپم مبتنی بر یادگیری تدریجی خود را با رسیدن ایمیل‌های جدید تطبیق می‌دهد و به تدریج درک خود را از اسپم به روزرسانی می‌کند. اگر استراتژی‌های اسپم تغییر کند، این نوع مدل می‌تواند به طور بالقوه یاد بگیرد که سبک‌های اسپم جدید را بدون نیاز به مجموعه جدیدی از داده‌های آموزشی تشخیص دهد.

از مزایای یادگیری تدریجی می‌توانیم به این موارد اشاره کنیم: استفاده بهینه از منابع، سازگاری به شکل بلاذرنگ، یادگیری کارآمد، یادگیری از داده‌های غیر ثابت

این روش محدودیت‌هایی هم دارد، مانند: فراموشی فاجعه‌بار، مشکل در مدیریت رانش مفهومی و خطر بیش‌برازش در ابتدای توضیحات مثالی از کاربرد این روش را گفتیم. علاوه بر آن می‌توانیم به استفاده از آن در ماشین‌های خودران و سیستم‌های توصیه‌ی اخبار هم اشاره کنیم.

برای پیاده‌سازی یادگیری تدریجی از الگوریتم‌های مختلفی می‌توان استفاده کرد. در ادامه به توضیح مختصراً از چند الگوریتم کاربردی‌تر در این متد می‌پردازیم:

○ SGD: یک انتخاب محبوب برای یادگیری تدریجی (Incremental Learning) است. پارامترهای مدل را با استفاده از یک نمونه در یک زمان یا یک دسته کوچک از نمونه‌ها به روز می‌کند. این رویکرد به مدل اجازه می‌دهد تا در حین آنکه دسته‌ای پس از دیگری را پردازش می‌کند، به صورت تدریجی یاد بگیرد SGD. به طور گسترده در برنامه‌های مختلف، از رگرسیون خطی ساده تا مدل‌های یادگیری عمیق پیچیده، استفاده می‌شود. برای مثال، در توسعه یک سیستم تعمیر و نگهداری پیش‌بینی‌کننده برای یک کارخانه تولیدی، SGD می‌تواند برای آموزش تدریجی یک مدل با داده‌های حسگر، تنظیم پارامترهای مدل با ورود داده‌های جدید استفاده شود. به این ترتیب، مدل می‌تواند خرابی‌های احتمالی تجهیزات را با دقت بیشتری در طول زمان پیش‌بینی کند.

○ SVM: SVM‌های آنلاین اقتباسی از الگوریتم سنتی SVM برای مدیریت یادگیری تدریجی (Incremental Learning) هستند. آن‌ها با به روزرسانی مدل SVM با ورود هر دسته جدید از داده کار می‌کنند و آن را برای جریان‌های داده یا برنامه‌های کاربردی در مقیاس بزرگ که در آن آموزش مجدد مدل با هر نمونه جدید غیر عملی است مناسب می‌کند. برای مثال، یک SVM آنلاین می‌تواند در یک تسلیک طبقه‌بندی متن برای یک خبرگزاری در مقیاس بزرگ استفاده شود که در آن مقالات باید به موضوع‌های مختلف به شکل بلاذرنگ طبقه‌بندی شوند SVM. می‌تواند به صورت تدریجی از هر مقاله جدید بیاموزد و دقت طبقه‌بندی خود را در طول زمان بهبود بخشد.

○ درختان تصمیم تدریجی: درختان تصمیم نوعی الگوریتم یادگیری ماشین هستند که می‌توانند از یادگیری تدریجی نیز پشتیبانی کنند. الگوریتم‌های درخت تصمیم تدریجی، مانند درخت Hoeffding یا درخت تصمیم بسیار سریع (VFDT)، درخت تصمیم را به صورت تدریجی می‌سازند و از روش‌های آماری برای تصمیم‌گیری زمان تقسیم گره‌ها استفاده می‌کنند. مثلاً وقتی یک

شرکت مخابراتی می‌خواهد ریزش مشتری را در زمان واقعی پیش‌بینی کند، آن‌ها می‌توانند از یک درخت تصمیم تدریجی برای یادگیری از هر تعامل با مشتری استفاده کنند و به تدریج توانایی مدل را برای پیش‌بینی مشتریانی که احتمالاً از بین می‌روند بهبود می‌بخشند.

- مدل‌های یادگیری عمیق تدریجی: مدل‌های یادگیری عمیق، به‌ویژه شبکه‌های عصبی بازگشتی (RNN) و انواع خاصی از شبکه‌های عصبی کانولوشنال (CNN)، می‌توانند برای یادگیری تدریجی سازگار شوند. این مدل‌ها از داده‌های جدید با بهروزرسانی تدریجی وزن خود یاد می‌گیرند که به آن‌ها اجازه می‌دهد تا داده‌های streaming یا محیط‌هایی را که در طول زمان تغییر می‌کنند مدیریت کنند. برای مثال، یک پلتفرم تجارت الکترونیک می‌تواند از یک مدل یادگیری عمیق تدریجی برای ارائه توصیه‌های محصول در زمان واقعی به کاربران خود استفاده کند. این مدل از هر تعامل کاربر یاد می‌گیرد و وزن‌های خود را به صورت تدریجی به‌روزرسانی می‌کند تا ترجیحات کاربران را بهتر دریافت کند و توصیه‌های دقیق‌تری ارائه کند.

b. الگوریتم Adaptive Random Forest (ARF) یک روش یادگیری مجموعه‌ای است که مفاهیم رندوم فارست و یادگیری آنلاین را ترکیب می‌کند. این الگوریتم برای پردازش جریان داده طراحی شده است، جایی که نمونه‌های جدید به صورت پیوسته و در زمان واقعی وارد می‌شوند. الگوریتم رندوم فارست سنتی با انتخاب تصادفی زیرمجموعه‌هایی از داده‌های آموزشی و ویژگی‌ها، مجموعه‌ای از درخت‌های تصمیم ساخت می‌کند. هر درخت در مجموعه به طور مستقل رایی برای برچسب کلاس یک نمونه ورودی می‌دهد و پیش‌بینی نهایی توسط رأی گیری اکثریت تعیین می‌شود. در ARF، الگوریتم به تغییرات توزیع داده‌ها در طول زمان سازگاری پیدا می‌کند و به همین دلیل برای سناریوهای داده‌های جریانی مناسب است. این را با معرفی عوامل سایه‌پذیری انجام می‌دهد که کنترل کننده اهمیت نمونه‌های قدیمی در حالی که نمونه‌های جدید وارد می‌شوند.

- به طور کلی نحوه کار الگوریتم ARF به این صورت است:
  ۱. یک مجموعه خالی از درخت‌های تصمیم مقداردهی اولیه می‌شود.
  ۲. برای هر نمونه ورودی:
    - از مجموعه درخت‌ها نمونه ای انتخاب می‌شود.
    - برای هر درخت انتخاب شده:
    - اگر درخت برچسب کلاس نمونه را به درستی پیش‌بینی کند، آمارها و عوامل سایه‌پذیری درخت به روز می‌شود.
    - اگر درخت برچسب کلاس نمونه را به طور نادرست پیش‌بینی کند، یک درخت جدید ایجاد شده و به مجموعه درخت‌ها اضافه می‌شود.
    - به صورت اختیاری، بر اساس fading factor، درخت‌های قدیمی از مجموعه حذف خواهند شد.
  ۳. پیش‌بینی‌ها با جمع آوری رأی‌های تمام درخت‌ها در مجموعه انجام می‌شود.

- می‌توانیم در کتابخانه‌ی scikit-multiflow برای صدا زدن AdaptiveRandomForestClassifier از کلاس

```
1 from skmultiflow.lazy import AdaptiveRandomForestClassifier
2 from skmultiflow.data import FileStream
3
4 # Create an instance of the AdaptiveRandomForestClassifier
5 arf = AdaptiveRandomForestClassifier(n_estimators=10)
6
7 # Create a stream of data (e.g., from a file)
8 stream = FileStream("your_data_file.csv")
9
10 # Prepare the stream for learning
11 stream.prepare_for_use()
12
13 # Run the ARF algorithm on the data stream
14 while stream.has_more_samples():
15     X, y = stream.next_sample()
16     arf.partial_fit(X, y, classes=stream.target_values)
17
18 # Once the learning is complete, you can use the model to make predictions
19 # For example, on a new instance 'new_instance':
```

استفاده کنیم. یک مثال ساده از استفاده‌ی آن به این صورت است:

در این کد پس از آنکه با یک تعداد تخمینی AdaptiveRandomForestClassifier ایجاد کردیم و همچنین دیتا را لود کردیم، ابتدا با استفاده از `prepare_for_use` دیتا را آماده‌ی استفاده می‌کنیم. در ادامه بخش اصلی را داریم. الگوریتم ARF با استفاده از یک حلقه while که روی نمونه‌های موجود در جریان تکرار می‌شود، روی جریان اعمال می‌شود. برای هر نمونه، مرحله یادگیری، با فراخوانی `arf.partial_fit(X, y, classes=stream.target_values)` آموزش دیده برای پیش‌بینی نمونه‌های جدید استفاده می‌توانیم استفاده کنیم.

c. در این ۲ بخش، همان‌طور که گفته شده از توابع آماده در `skmultiflow` استفاده می‌کنیم. پس از خواندن دیتا به صورت به این صورت عمل می‌کنیم: (با توجه به این که گفته شده دیتا `SEAGenerator`. آماده شده، با خواندن آن به این روش، `data stream` خواهیم داشت.)

```
csv_file_path = 'stream_data.csv'
stream = FileStream(csv_file_path)
stream.prepare_for_use()
```

دیتای ما اندازه‌اش ۴ میلیون است، نصف آن را برای بخش `train` قرار می‌دهیم. اندازه‌ی کل سمپل‌ها را هم مشخص می‌کنیم، دراستفاده از این توابع، برای تست از کل سمپل‌ها منهای دیتای آموزش که تعریف کردیم، استفاده می‌کند.

در ادامه دو مدل را طبق فرمتی که باید تعریف می‌کنیم و در نهایت، از معیارهای `accuracy` و `kappa` (مریبوط به بخش d) برای ارزیابی استفاده می‌کنیم. نکته‌ی قابل توجه دیگر این است که پس از این که یکی از مدل‌ها را روی دیتا اعمال کردیم، `stream` را برای آموزش دیدن توسط مدل بعدی ابتدا `restart` می‌کنیم.

نتایج بدست آمده:  
در این داده‌ها،

```
Prequential Evaluation
Evaluating 1 target(s).
Pre-training on 2000000 sample(s).
Evaluating...
#####
[100%] [187.02s]
Processed samples: 4000000
Mean performance:
Hoeffding Tree - Accuracy      : 0.9993
Hoeffding Tree - Kappa          : 0.9984
```

نتایج برای دو مدل به هم نزدیک و خوب است، ولی زمان اجرای ARF بهوضوح خیلی بیشتر بود.

```
Prequential Evaluation
Evaluating 1 target(s).
Pre-training on 2000000 sample(s).
Evaluating...
#####
[100%] [11616.78s]
Processed samples: 4000000
Mean performance:
Adaptive Random Forest - Accuracy      : 0.9982
Adaptive Random Forest - Kappa          : 0.9959
```

## توضیح معیارهای استفاده شده: Accuracy

دقت یک معیار ساده است که برای ارزیابی عملکرد یک طبقه‌بندی کننده با اندازه‌گیری نسبت نمونه‌های درست پیش‌بینی شده از کل نمونه‌ها در جریان داده استفاده می‌شود. در EvaluatePrequential<sup>۱</sup> در کتابخانه scikit-multiflow، دقต به صورت پیوسته محاسبه می‌شود زیرا مدل به پردازش داده‌های ورودی می‌پردازد. این ارزیابی بلدرنگ کمک می‌کند تا بفهمیم مدل در جریان داده‌ها چقدر خوب عمل می‌کند، که این امر در سناریوهای یادگیری آنلاین که داده‌ها به صورت متوالی می‌رسند و مدل باید به طور پویا سازگار شود، بسیار مهم است.

با این حال، accuracy می‌تواند گمراه کننده باشد، به ویژه در مجموعه داده‌های نامتعادل که در آن یک کلاس به طور قابل توجهی از دیگر کلاس‌ها بیشتر است. در چنین مواردی، ممکن است مدل با پیش‌بینی کلاس اکثربیت به تنها یی دقت بالایی نشان دهد و پیش‌بینی‌های کلاس اقلیت را نادیده بگیرد. بنابراین، اتكا صرف به دقیق است تصویر کاملی از عملکرد مدل ارائه ندهد و نیاز به استفاده از معیارهای اضافی مانند کاپا برای به دست آوردن بینش‌های بیشتر ضروری است.

## Kappa

کاپا، و به طور خاص کاپای کوهن، یک معیار آماری است که توافق بین برچسب‌های پیش‌بینی شده و واقعی کلاس‌ها را با در نظر گرفتن احتمال توافق به صورت تصادفی محاسبه می‌کند. در روش EvaluatePrequential<sup>۲</sup>، کاپا برای ارزیابی دقیق‌تر از عملکرد طبقه‌بندی کننده به ویژه در مجموعه داده‌های نامتعادل استفاده می‌شود. این معیار دقیق‌تر از مشاهده شده را با دقیق‌تر از (تصادفی) مقایسه می‌کند و به این ترتیب دیدگاه متعادل‌تری از قدرت پیش‌بینی مدل ارائه می‌دهد.

مقدار کاپا از ۱ تا ۱ متغیر است، که در آن ۱ نشان‌دهنده توافق کامل، ۰ نشان‌دهنده عدم توافق فراتر از شанс و مقادیر منفی نشان‌دهنده توافق کمتر از شанс است. با ترکیب کاپا در ارزیابی، scikit-multiflow به کاربران این امکان را می‌دهد که بفهمند مدل آنها چقدر بهتر از حدس تصادفی عمل می‌کند و به این ترتیب درک بهتری از اثربخشی آن در کلاس‌های مختلف ارائه می‌دهد. این امر کاپا را به یک معیار ارزشمند در ارزیابی قابلیت‌های پیش‌بینی واقعی مدل در محیط‌های داده جریانی تبدیل می‌کند.

## بخش دوم)

سوال ۱:

$$\begin{array}{ll}
 \text{average for } A = \frac{r+r+r+1+1}{5} = 1,1 & \text{average for } D = \frac{r+r+1}{3} = r \\
 \text{average for } B = \frac{r+1+r}{3} = 1,44 & \text{average for } E = \frac{1+r+r}{3} = 1,44 \\
 \text{average for } C = \frac{r+r+r+r}{4} = 1,1 & \text{average for } F = \frac{1+1+r}{3} = 1,33 \\
 \end{array} \quad (a)$$

$(x - \bar{x})$  بمحضه

	$u_1$	$u_2$	$u_3$	$u_4$	$u_5$
A	1,1	1,1	1,1	-1,1	-1,1
B	0,144	-1,44		1,144	
C	1,1	-0,1		-0,1	0,1
D		0	1		-1
E			-1,44	1,144	0,144
F	-0,144	-0,144			0,44

\* Pearson : ( $S_{xy}$  = items rated by both user  $x$  and  $y$ )

$$\text{sim}(x,y) = \frac{\sum_{S \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{S \in S_{xy}} (r_{xs} - \bar{r}_x)^2} \sqrt{\sum_{S \in S_{xy}} (r_{ys} - \bar{r}_y)^2}}$$

$\bar{r}_x, \bar{r}_y$ : avg. rates of  $x$  and  $y$

$$\text{sim}(A,D) = \frac{1,1 \times 0 + 1,1 \times 1 + (-1,1) \times (-1)}{\sqrt{1,1^2 + 1,1^2 + (-1,1)^2} \sqrt{0^2 + 1^2 + (-1)^2}} = 0,144 \quad \text{sim}(B,D) = 0$$

$$\text{sim}(C,D) = \frac{-0,1 \times 0 + 0,1 \times 1 + (-1) \times (-1)}{\sqrt{0,1^2 + (0,1)^2} \sqrt{0^2 + 1^2 + (-1)^2}} = -0,144$$

$$\text{sim}(E,D) = \frac{1 \times (-1,44) + (-1) \times 0,144}{\sqrt{1^2 + 1^2} \sqrt{(-1,44)^2 + 0,144^2}} = -0,144$$

$$\text{sim}(F,D) = \frac{-0,144 \times 0 + 0,44 \times (-1)}{\sqrt{(-0,144)^2 + 0,44^2} \sqrt{0^2 + 1^2}} = -0,144$$

رسانی از  $B$  و  $A$  در یک مسیر

$$r_{xi} = \frac{\sum_{y \in N} S_{xy} \cdot r_{yi}}{\sum_{y \in N} S_{xy}} \rightarrow \frac{0,144 \times (-1) + 0 \times 1}{0,144 + 0} = -1 \quad (b)$$

## سوال ۲:

مزایا:

- برای هر نوع آیتمی جواب می‌دهد: نیازی به feature selection نداریم.
- تنها به تعامل بین یک کاربر و آیتم‌های او بسته نکرده و نظرات کاربران مشابه کاربر مدنظر را برای پیش‌بینی و ارائه پیشنهاداتش در نظر می‌گیرد.
- می‌تواند آیتم‌های جدیدی پیشنهاد دهد و مشکل content based که محدود به تاریخچه‌ی آیتم‌های کاربر بود را ندارد.
- مقیاس پذیر یا همان scalable است و در دیتاست‌های بزرگ به راحتی قابل به کارگیری است.
- adaptive و با تغییر سلیقه‌ی کاربر در طول زمان هماهنگ است. با توجه به آپدیت‌های صورت گرفته recommendation را می‌تواند تغییر دهد تا بهبود یابند.

معایب:

- cold start دارد و باید منتظر بماند تا تعداد کافی یوزر وارد سیستم شوند و فعالیت کنند.
- در این روش از ماتریس user/rating استفاده می‌کنیم که این یکی از مزایای این روش است، ولی در کنار مزیتی که دارد، عیب هم هست و در اینجا sparse بودن ماتریس مشکل است و پیدا کردن یوزرهایی که به آیتم‌های مشترک نظر داده‌اند سخت است.
- نمی‌تواند آیتمی که تا به حال به آن امتیاز داده نشده را پیشنهاد دهد. این مشکل به first rater معروف است.
- این روش نسبت به آیتم‌های محبوب خیلی بایاس است و آن‌ها را در بسیاری از نتایج می‌آورد، درحالی که شاید آیتم‌هایی که محبوب نیستند گزینه‌های بهتری باشند.

سوال ۳:

(a) بخش

توضیح کد بخش pearson:

این بخش برای یافتن کاربرانی است که بیشترین شباهت با یک کاربر مشخص (کاربر ۱۲۶) بر اساس امتیازدهی آنها به فیلم‌ها دارد. در ابتدا، یک جدول متقاضی با نام `ratings\_pivot` ایجاد می‌شود که در آن هر ردیف نمایانگر یک کاربر(`userId`)، هر ستون نمایانگر یک فیلم(`movieId`)، و مقادیر سلول‌ها نمایانگر امتیاز‌هایی است که کاربران به فیلم‌ها داده‌اند.

برای اندازه‌گیری شباهت بین کاربران، تابع `pearson\_similarity` تعریف می‌شود. این تابع ضریب همبستگی پیرسون بین دو کاربر را محاسبه می‌کند با ابتدا شناسایی فیلم‌هایی که هر دو کاربر امتیاز داده‌اند.(`common\_ratings`) در صورتی که تعداد این فیلم‌ها کمتر از ۲ باشد که نشان‌دهنده داده‌های کافی برای محاسبه ارتباط قابل اعتماد نیست، تابع مقدار `NaN` برمی‌گرداند. در غیر این صورت، این تابع ضریب همبستگی پیرسون را محاسبه می‌کند. تابع به این صورت تعریف می‌شود:

■ Pearson correlation coefficient

$$sim(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \bar{r}_y)^2}}$$

$\bar{r}_x, \bar{r}_y \dots \text{avg rating of } x, y$

در مرحله بعدی، کد شباهت پیرسون بین کاربر ۱۲۶(`user\_id`) و سایر کاربران دیگر در مجموعه داده محاسبه می‌کند. این کار با استفاده از حلقه `apply` و تابع `lambda` بر روی تمام کاربران در `ratings\_pivot` انجام می‌شود و سپس شباهت با خود کاربر ۱۲۶ حذف می‌شود. نتایج شباهت‌ها در `pearson\_similarities` ذخیره می‌شود.

در نهایت، ده کاربر که بیشترین شباهت پیرسون با کاربر ۱۲۶ را نشان می‌دهند شناسایی شده و چاپ می‌شوند.

توضیح کد بخش cosine sim:

در این تابع هم از کد آماده‌ی مربوط به محاسبه کسینوسی استفاده شده و ۱۰ کاربر شبیه‌تر برگردانده شده است.

در کل شباهت پیرسون خیلی بهتر عمل می‌کند و نتایج دقیق‌تری ارائه می‌دهد. به همین دلیل در ادامه برای بخش‌های بعدی از شباهت پیرسون استفاده خواهیم کرد. خروجی‌های این بخش:

Pearson:	
userId	
61	1.0
146	1.0
237	1.0
252	1.0
259	1.0
272	1.0
293	1.0
309	1.0
487	1.0
511	1.0

Cosine sim:	
userId	
379	0.813373
94	0.706430
507	0.690587
512	0.684872
81	0.679452
485	0.679053
179	0.674915
498	0.671849
470	0.665665
340	0.652847

## (b) بخش

در این بخش، از روش user-user CF برای پیش‌بینی امتیازهای فیلم برای کاربر ۱۲۶ پیاده‌سازی شده است. این روش بر اساس شباهت کاربران به یکدیگر از نظر الگوی امتیازدهی به فیلم‌ها عمل می‌کند. در ابتدا، تابع `predict\_user\_rating` تعریف شده است که با استفاده از میانگین وزن دار امتیازهای کاربران مشابه، امتیاز پیش‌بینی شده را برای یک فیلم خاص (`movie\_id`) برای کاربر مورد نظر (`user\_id`) محاسبه می‌کند. این تابع به ازای هر کاربر مشابه، امتیازاتی که آن‌ها به فیلم مورد نظر داده‌اند را استخراج کرده و با استفاده از شباهت پیرسون، امتیاز پیش‌بینی شده را محاسبه می‌کند.

در تابع `user\_user\_cf`، که برای تولید پیشنهادهای فیلم برای کاربر `user\_id` استفاده می‌شود، ابتدا امتیازهای فیلم‌های قبلاً داده شده توسط کاربر `user\_id` از جدول `ratings\_pivot` استخراج می‌شوند. سپس برای هر فیلمی که کاربر `user\_id` هنوز امتیاز نداده است، از تابع `predict\_user\_rating` برای پیش‌بینی امتیاز استفاده می‌شود. پس از محاسبه امتیازهای پیش‌بینی برای همه فیلم‌ها، ۶ فیلم برتر برگردانده می‌شود. فرمولی که در این بخش از آن استفاده شده به این صورت است:

$$\hat{r}_{ui} = \frac{\sum_{v \in S(u)} s_{uv} \cdot r_{vi}}{\sum_{v \in S(u)} |s_{uv}|}$$

به طوری که:

- $\hat{r}_{ui}$  is the predicted rating for user  $u$  on item  $i$ .
- $S(u)$  is the set of users similar to user  $u$ .
- $s_{uv}$  is the similarity between user  $u$  and user  $v$ .
- $r_{vi}$  is the rating of user  $v$  on item  $i$ .

خروجی:

```
Top 6 recommended movies using user-user CF:
Movie 912 (5.0)
Movie 1172 (5.0)
Movie 1250 (5.0)
Movie 1283 (5.0)
Movie 1304 (5.0)
Movie 1584 (5.0)
```

## (c) و (d) بخش

تابع item-item-cf برای هر دو یکی است. صرفاً یک مود تعریف می‌کنیم که اگر  $\bullet$  بود از حالت غیر baseline و در غیر این صورت از baseline استفاده کند. حالا توضیح این توابع را به طور مشخص‌تر می‌گوییم:

در این کد، ماتریس شباهت آیتم‌ها با استفاده از پیرسون محاسبه شده است که در تابع `predict\_item\_rating` برای پیش‌بینی امتیازها به کار می‌رود. ابتدا، ماتریس `item\_sim\_df` با استفاده از شباهت پیرسون بین آیتم‌ها ساخته می‌شود. سپس، تابع `predict\_item\_rating` برای پیش‌بینی امتیاز یک کاربر (`user\_id`) به یک آیتم (`item\_id`) تعریف شده است. این تابع، این امتیازهای کاربر را از جدول `ratings\_pivot` استخراج کرده و آیتم‌های مشابه با آیتم هدف را شناسایی می‌کند. برای هر آیتم

مشابه که کاربر به آن امتیاز داده، ضریب شباهت و امتیاز کاربر به آن آیتم محاسبه شده و در جمع وزنی (`numerator`) و جمع مطلق ضریبها (`denominator`) به کار می‌رود. در نهایت، امتیاز پیش‌بینی شده به صورت نسبت `numerator` به `denominator` محاسبه می‌شود. اگر `denominator` صفر باشد، مقدار `NaN` برگردانده می‌شود.

تابع `predict\_item\_rating\_with\_baseline` نسخه بهبود یافته‌ای از تابع قبلی است که از برآورد پایه برای بهبود دقت پیش‌بینی‌ها استفاده می‌کند. این تابع ابتدا برآورد پایه (`bxi`) را محاسبه می‌کند که شامل میانگین کلی امتیازها(`mu`)، بایاس کاربر(`bx`)، و بایاس آیتم(`bi`) است. سپس مشابه تابع قبلی، آیتم‌های مشابه با آیتم هدف را شناسایی کرده و ضریب شباهت و امتیاز‌های مرتبط را برای محاسبه جمع وزنی و جمع مطلق ضریبها به کار می‌برد. اما در اینجا، امتیاز آیتم مشابه از برآورد پایه مربوطه کسر می‌شود تا تاثیر بایاس‌ها لحاظ گردد. در نهایت، امتیاز پیش‌بینی شده به صورت جمع برآورد پایه و نسبت `denominator` به `numerator` محاسبه می‌شود. اگر `denominator` صفر باشد، مقدار برآورد پایه برگردانده می‌شود.

تابع `item\_item\_cf` برای پیشنهاد فیلم‌ها بر اساس فیلترینگ همکارانه آیتم-آیتم طراحی شده است. اگر `mode` برابر با صفر باشد، تابع ساده‌تر پیش‌بینی امتیاز فراخوانی می‌شود و اگر `mode` برابر با یک باشد، نسخه با برآورد پایه استفاده می‌شود. امتیاز‌های پیش‌بینی شده در یک سری ذخیره می‌شوند. فرمول‌های استفاده شده در این دو بخش به این صورت هستند:

بدون baseline‌ها:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} \text{sim}(i,j) \cdot r_{uj}}{\sum_{j \in N(i)} |\text{sim}(i,j)|}$$

- $\hat{r}_{ui}$  is the predicted rating of user  $u$  for item  $i$ .
- $N(i)$  is the set of items similar to item  $i$  that user  $u$  has rated.
- $\text{sim}(i, j)$  is the similarity between item  $i$  and item  $j$ .
- $r_{uj}$  is the actual rating of user  $u$  for item  $j$ .

به طوری که:

با baseline‌ها:

$$\hat{r}_{ui} = \mu + b_u + b_i + \frac{\sum_{j \in N(i)} \text{sim}(i,j) \cdot (r_{uj} - \mu - b_u - b_j)}{\sum_{j \in N(i)} |\text{sim}(i,j)|}$$

- $\mu$  is the global average rating.
- $b_u$  is the bias of user  $u$  (how much user  $u$ 's ratings deviate from the global average).
- $b_i$  is the bias of item  $i$  (how much item  $i$ 's ratings deviate from the global average).
- Other terms are as defined above.

به طوری که:

خروجی‌ها:

```
Top 6 recommended movies using item-item CF:  
Movie 2268 (3.59011168088767)  
Movie 2 (3.5243121063177916)  
Movie 1101 (3.502890351669596)  
Movie 3578 (3.5008311549944136)  
Movie 2706 (3.500130255277038)  
Movie 2571 (3.4877977492085033)
```

```
Top 6 recommended movies using item-item CF (using baseline):  
Movie 741 (4.425681217089712)  
Movie 44195 (4.41897691784027)  
Movie 1272 (4.355954629885956)  
Movie 904 (4.346099139826445)  
Movie 1252 (4.333022965431732)  
Movie 2761 (4.298575343505807)
```

(e) بخش

در این بخش برای ترکیب نتایج، از یک میانگین وزنی استفاده می‌کنیم. همان‌طور که دیدیم، نتایج در حالت user-user به همین دلیل برای نتایج آن، وزن بیشتری قرار می‌دهیم. سپس در بین تعدادی از برترین نتایج هر متدها، مشترک‌ها را پیدا کرده و با وزن‌های تعریف شده، نتایج را حساب می‌کنیم. در نهایت، توصیه‌های ترکیبی به یک سری pandas تبدیل می‌شوند و بهترین ۶ توصیه بر اساس بالاترین امتیازات انتخاب و نمایش داده می‌شوند.

خروجی:

```
Top 6 recommended movies using combination of user-user and item-item CF:  
Movie 1584 (3.5)  
Movie 912 (3.5)  
Movie 1172 (3.5)  
Movie 1250 (3.5)  
Movie 1283 (3.5)  
Movie 1304 (3.5)
```