دانشکده فنی – گروه مهندسی کامپیوتر

گزارش پروژه ی کارشناسی

عنوان : سیستم توصیه گر فیلم با استفاده از ادغام روش شبکه های عصبی عمیق و collaborative filtering

نام و نام خانوادگی: دلارام شیخ بهایی

شماره دانشجویی: ۹۷۱۱۱۳۰۳۲

استاد راهنما: سرکار خانم دکتر آزاده طباطبایی

تابستان ۱۴۰۱


بسم الله الرحمن الرحیم

چکیده

اینترنت به عنوان منبعی برای حجم انبوه داده‌ها و اطلاعات محسوب می‌شود. در عین حال، کالاها و خدمات متنوعی نیز از طریق اینترنت در دسترس عموم مردم هستند. در این اقیانوس، نیاز به ابزاری برای پالایش، اولویت‌بندی و تحویل موثر اطلاعات مورد نیاز و مرتبط به هر کاربر به او محسوس است. این کار به عنوان راهکاری مُسَکن‌وار برای مسئله وجود سرریز اطلاعات (Information Overload) در نظر گرفته شده است. امروزه، سرریز اطلاعات مشکلات متعددی را برای کاربران اینترنت به وجود آورده است. ریکامندر سیستم در صدد است تا این مشکل را با جستجو در میان حجم انبوهی از اطلاعاتی حل کند که همه روزه به صورت پویا تولید می‌شوند و محتوا و خدمات شخصی‌سازی شده برای هر کاربر را در اختیار او قرار دهد.

در این مقاله به توضیح روش Neural collaborative filtering در سیستم های توصیه گر می پردازیم و پس از توضیح روش ها به تشریح کد پروژه میپردازیم و بعد از آن نمودار ها را بررسی میکنیم

کلید واژه ها :

سیستم توصیه گر ، شبکه عصبی ، collaborative filtering ، ماشین لرنینگ ، یادگیری عمیق

فهرست

۱–مقدمه

تماشای فیلم‌های سینمایی و سریال‌های تلویزیونی از جمله علاقمندی‌های بسیاری از افراد است. ولیکن، بحث انتخاب فیلم محبوب برای هر شخصی، بسیار وابسته به علایق فردی او است. برای مثال، ممکن است فردی از دیدن فیلم «سرنوشت شگفت‌انگیز آملی پولن (Amélie) «لذت ببرد. اما آیا این مساله به خاطر موسیقی فیلم است؟ از ماهیت هنری فیلم نشات می‌گیرد؟ به دلیل نقش‌آفرینی «اودره ژوستن توتو (Audrey Justine Tautou)» در این فیلم است؟ و یا همه گزینه‌های بالا؟ به همان میزان که ممکن است سلیقه هر فرد در انتخاب فیلم بی‌همتا و غیر قابل پیش‌بینی باشد، احتمال دارد که کسی دیگر در جهان سلیقه‌ای بسیار مشابه با او داشته باشد.

این دقیقا همان ایده نهفته در پس توصیه‌گرهای Collaborative Filtering است. collaborative filtering روشی برای توصیه فیلم به کاربر بر مبنای یافتن دیگر کاربران با سلایق مشابه است. با بهره‌گیری از این روش می‌توان کاربران با سلیقه مشابه را بر پایه فیلمی که هر دو دیده‌اند و به طور مشابه به آن امتیازدهی کرده‌اند یافت.

۲-روش های استفاده شده برای پیاده سازی ریکامندر سیستم فیلم

- collaborative filtering

collaborative filtering از شباهت های بین کاربران و آیتم ها به طور همزمان برای ارائه توصیه ها استفاده می کند مدل های collaborative filtering می توانند یک آیتم را براساس علایق کاربر B که مشابه به کاربر A است، توصیه کنند. به علاوه، می تواند ویژگی ها(embeding) را به صورت خودکار یاد بگیرد .

یک سیستم توصیه فیلم را در نظر بگیرید که در آن داده های آموزشی شامل یک ماتریس بازخورد (رتبه بندی) است که در آن:

هر ردیف نشان دهنده یک کاربر است.
هر ستون نشان دهنده یک آیتم (یک فیلم)است.
بازخورد فیلم ها به یکی از دو دسته تقسیم می شود:

صریح:کاربران با ارائه یک رتبه بندی عددی مشخص می کنند که چقدر یک فیلم خاص را دوست دارند.
ضمنی: اگر کاربر یک فیلم را تماشا کند، سیستم استنباط می کند که کاربر علاقه مند است.
برای ساده سازی فرض می کنیم که ماتریس بازخورد دودویی است؛ یعنی مقدار ۱ نشان دهنده علاقه به فیلم است.

هنگامی که کاربر از صفحه اصلی بازدید می کند، سیستم باید فیلم ها را براساس هر دو توصیه کند:

شباهت به فیلم هایی که کاربر در گذشته دوست داشته است
فیلم هایی که کاربران مشابه آن ها را دوست داشتند

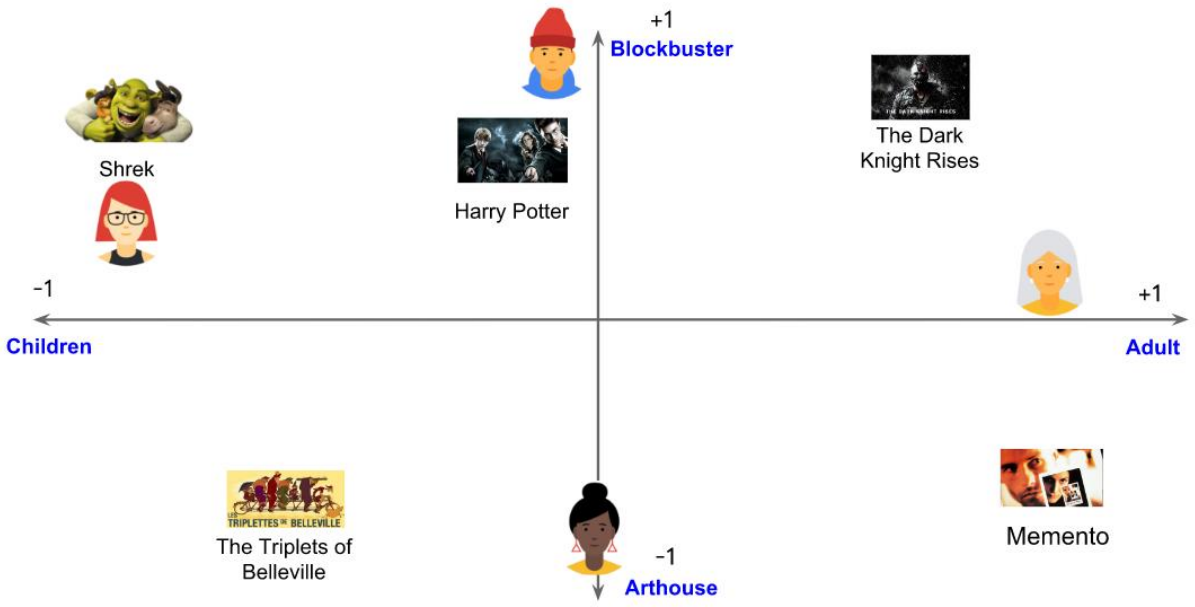
Embedding یک بعدی

فرض کنید به هر فیلم یک بازه بین -۱ و ۱ اختصاص می دهیم که مشخص می کند فیلم برای کودکان (مقادیر منفی) یا بزرگسالان (مقادیر مثبت) است. فرض کنید به هر کاربر یک بازه بین -۱ و ۱ اختصاص می‌دهیم که علاقه کاربر به فیلم‌های کودکان (نزدیک به -۱) یا فیلم‌های بزرگسالان (نزدیک به +۱) را توصیف می‌کند. ضرب داخلی user embedding و movie embedding باید برای فیلم هایی که انتظار داریم کاربر دوست داشته باشد، بیشتر (نزدیک به ۱) باشد.

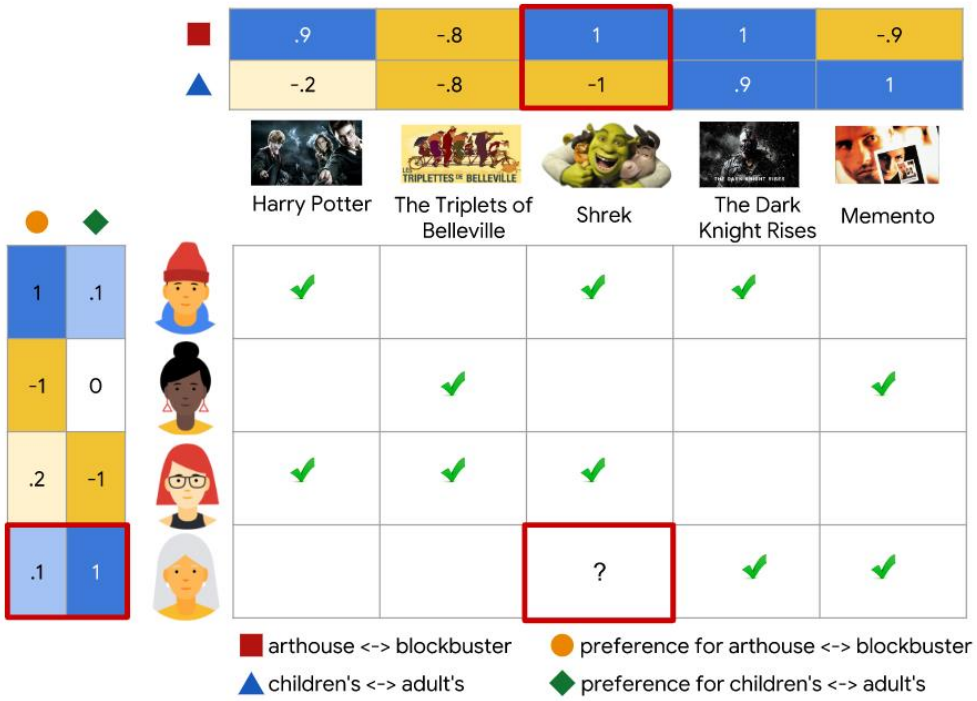
در نمودار زیر، هر تیک فیلمی را مشخص می کند که کاربر خاصی تماشا کرده است. کاربران سوم و چهارم ترجیحاتی دارند که به خوبی توسط این ویژگی توضیح داده شده است – کاربر سوم فیلم را برای کودکان و کاربر چهارم فیلم را برای بزرگسالان ترجیح می دهد. با این حال، اولویت های کاربران اول و دوم به خوبی توسط این ویژگی توضیح داده نمی شود.

Embedding دو بعدی

یک ویژگی برای توضیح ترجیحات همه کاربران کافی نبود. برای غلبه بر این مشکل، ویژگی دوم را اضافه می کنیم درجه ای که هر فیلم یک بلاک باستر یا یک فیلم هنری است. با ویژگی دوم، حالا می توانیم هر فیلم را با embedding دو بعدی زیر نمایش دهیم



ما دوباره کاربران خود را در همان فضای embedding قرار می دهیم تا ماتریس بازخورد را به بهترین شکل توضیح دهیم: برای هر جفت (کاربر، آیتم)، می خواهیم ضرب داخلی user embedding و movie embedding نزدیک به ۱ باشد وقتی کاربر فیلم را تماشا کرده، و در غیر این صورت به ۰.



ماهیت colabrotive این رویکرد زمانی آشکار می شود که مدل embedding ها را یاد می گیرد. فرض کنید بردارهای embedding برای فیلم ها ثابت باشند. سپس، مدل می تواند یک بردار embedding را یاد بگیرد تا کاربران ترجیحات خود را به بهترین نحو توضیح دهند. در نتیجه، embedding کاربران با اولویت های مشابه نزدیک به هم خواهد بود. به طور مشابه، اگر embeddings برای کاربران ثابت باشد، آنگاه می توانیم embeddings فیلم را یاد بگیریم تا ماتریس بازخورد را به بهترین شکل توضیح دهیم. در نتیجه، embeddings فیلم های مورد علاقه کاربران مشابه در فضای embedding نزدیک خواهد بود.

برای ساخت مدل فیلترینگ مشارکتی، از دو متد عمده استفاده می‌شود:

- متد مبتنی بر حافظه(memory-based)
- متد مبتنی بر مدل(model-based)

متد مبتنی بر حافظه

در این متد دو رویکرد اصلی وجود دارد:

۱-در این رویکرد، ما کاربران را گروه‌بندی می‌کنیم و از تعامل آیتم-کاربر یکی از کاربران، سایر تعاملات را پیش‌بینی می‌کنیم.

۲-در این رویکرد، ما آیتم‌ها را گروه‌بندی می‌کنیم و بر اساس امتیازاتی که کاربر به آیتم‌های موجود در یک گروه می‌دهد، علایق او را پیش‌بینی می‌کنیم.

متد مبتنی بر مدل

این متد، مبتنی بر تکنیک‌های یادگیری ماشین و داده‌کاوی است. هدف ما آموزش مدل‌هایی است که پیش‌بینی‌هایی انجام دهند. به عنوان مثال می‌توان از داده‌های موجود درباره تعاملات کاربر-آیتم استفاده کرد و ۵ آیتم احتمالی مورد علاقه‌ی کاربر را پیش‌بینی نمود.

matrix factorization یک مدل embedding ساده است. با توجه به ماتریس $A \in R^{m \times n}$ بازخورد ، که در آن m تعداد کاربران (یا پرس و جوها)و n تعداد آیتم ها است، مدل موارد زیر را یاد می گیرد:

ماتریس embegging کاربر $U \in \mathbb{R}^{m \times d}$ ، که در آن سطر i مقادیر embedding برای کاربر i است.

ماتریس embedding آیتم $V \in \mathbb{R}^{n \times d}$ ، که در آن ردیف j مقادیر embedding برای آیتم j است.

embedding ها یاد میگیرند که ضرب داخلی دو ماتریس embedding با تقریب خوبی میزان بازخورد هر کاربر به هر فیلم را نشان میدهد

- شبکه‌های عصبی

شبکه عصبی چیست؟

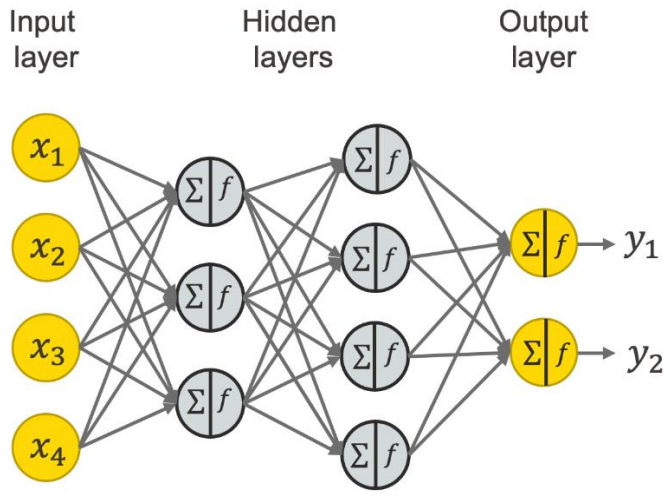
شبکه عصبی توسط ورودی ها آموزش داده می شود و شامل سه لایه ورودی و پنهان و خروجی است و هر کدام از عصب ها دارای مقدار آستانه و تابع فعال سازی میباشند که به ما خروجی می دهند نتیجه ای که به دست می اوریم با خروجی که انتظار داریم مقایسه می شود که این دو مقدار باید نزدیک به هم باشند مدل یاد میگیرد که وزن ها و مقدار آستانه را طوری تنظیم کند که خروجی درست دریافت کند.

شبکه عصبی مصنوعی در واقع دسته ای الگوریتم است که برای شناسایی و تشخیص الگوها به کار می رود.

شبکه عصبی عمیق(Dnn)

هرچه تعداد لایه هاو عصب ها در هر لایه پنهان بیشتر باشند مدل پیچیده تر می‌شود وقتی این شبکه های عصبی که شامل بیشتر از سه لایه از عصب های لایه های ورودی و خروجی اند به آنها شبکه عصبی عمیق گفته می شود و به یادگیری آنها یادگیری عمیق گفته می شود که به وسیله این شبکه های عصبی عمیق مسائل بسیار پیچیده در زمینه پیش بینی و دسته بندی به مسائل ساده حل می شود. در واقع یادگیری عمیق یک تابع است که ورودی را به خروجی تبدیل میکند شبکه عصبی عمیق ارتباط داده های ورودی و خروجی را پیدا می کند منظور از عمیق بودن شبکه عصبی چند لایه بودن این شبکه هاست لایه های شبکه ی عصبی از گره ها تشکیل شده اند که یک گره مانند مغز انسان مکانی برای انجام محاسبات است دریک گره داده های ورودی در یک وزن (weight) ضرب می‌شود. هر چقدر این وزن بیشتر باشد تاثیر داده زیاد تر می شود پس از آن، مجموع داده‌های ضرب‌شده در وزنشان محاسبه می‌گردد. در آخر هم برای

رسیدن به خروجی، مجموع به دست آمده، از یک تابع فعال‌ساز (Activation Function) عبور می‌کند و خروجی میگیرد.



اجزای تشکیل دهنده ی شبکه عصبی

یک شبکه عصبی شامل موارد زیر است:

نورون

یک نورون مصنوعی یک عملکرد ریاضی است. این یک یا چند ورودی را می گیرد که در مقادیری به نام وزن ضرب می شوند و با هم جمع می شوند. این مقدار سپس به یک تابع غیر خطی منتقل می شود که به عنوان یک تابع فعال سازی شناخته می شود تا به خروجی نورون تبدیل شود.

وزن

وزن پارامتری در یک شبکه عصبی است که داده های ورودی را در لایه های پنهان شبکه تغییر می دهد. شبکه عصبی مجموعه ای از گره ها یا نورون ها است. درون هر گره مجموعه ای از ورودی ها ، وزن و مقدار بایاس وجود دارد.اغلب اوقات یک شبکه عصبی در لایه های پنهان شبکه قرار دارد. وزن و بایاس) معمولاً w و b نامیده می شوند (پارامترهای قابل یادگیری مدل یادگیری ماشین هستند. وقتی ورودی ها بین نورون ها منتقل می شوند ، وزن ها به همراه بایاس بر روی ورودی ها اعمال می شوند.

تابع

nورودی به صورت X1 تا Xn به وزن‌های مربوطه از Wk1 تا Wkn داده شده‌اند. ابتدا وزن‌ها در ورودی‌های خودشان ضرب، و سپس با مقدار bias جمع می‌شوند. حاصل را u می‌نامیم.

$$u=\sum w \times x+b$$

سپس تابع فعال‌سازی برروی u اعمال می‌شود، یعنی f(u) ، و در نهایت مقدار خروجی نهایی را به عنوان

$$Yk = f(u)$$

از نورون دریافت می‌کنیم.

بهترین تعریف یک شبکه‌ی عصبی را شخصی به نام لیپینگ یانگ ارائه می‌دهد. او شبکه‌ی عصبی را اینگونه تعریف کرده‌است:

شبکه‌های عصبی شامل تعدادی نورون مصنوعی اند که اطلاعات را بین یکدیگر تبادل می‌کنند، و هر کدام دارای وزن هایی می باشند که بر پایه تجربه‌ی شبکه به وجود می اینـد. نورون‌ها نقطه‌ی فعال‌سازیی دارند که اگر مجموع وزن و داده‌های ارسال شده به آن‌ها از آن نقطه عبور کنند، آن‌ها فعال می‌شوند. نورون‌هایی که فعال شده اند باعث یادگیری می‌شوند.

۳- توضیح کد پروژه

بخش اول – توصیف داده‌ها

برای سیستم توصیه‌گر از مجموعه داده « MovieLens» استفاده شده است. این مجموعه داده شامل ۱۰۰،۰۰۰ امتیازدهی برای ۹،۰۰۰ فیلم مختلف توسط ۱۰۰۰۰ کاربر است. داده‌ها متعلق سال ۲۰۱۸ هستند. بنابراین هیچ فیلمی با تاریخ انتشار پس از سال ۲۰۱۸ در آن وجود ندارد. داده‌های امتیاز از مقیاس ۰,۵ تا ۵,۰ هستند. هر کاربر در این مجموعه داده حداقل به ۲۰ فیلم گوناگون امتیاز داده است. در پروژه دو فایل داده «Ratings»و «Movies» به کار رفته است. این داده‌ها در «دیتافریم (DataFrame) «کتابخانه «pandas»پایتون خوانده می‌شوند.این دو دیتافریم در زیر ارائه شده‌اند:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

پنج سطر اول دیتافریم Movies

	userId	movieId	y	timestamp
0	1	1	4.0	2000-07-30 20:45:03
1	1	3	4.0	2000-07-30 20:20:47
2	1	6	4.0	2000-07-30 20:37:04
3	1	47	5.0	2000-07-30 21:03:35
4	1	50	5.0	2000-07-30 20:48:51

پنج سطر اول دیتافریم Ratings – ستون y همان ستون ratings است

یک جنبه مهم از داده‌ها این حقیقت است که داده‌های امتیازدهی بسیار «خلوت (Sparse)» هستند. این امر بدین دلیل به وقوع می‌پیوندد که بسیاری فیلم وجود دارد ولی هر کاربر تنها به بخش کوچکی از فیلم‌ها امتیاز داده است. بنابراین، اگر یک ماتریس کامل با ۱۰۰۰۰۰ سطر مربوط به کاربران ساخته شود و ۹۰۰۰ستون مربوط به فیلم‌ها وجود داشته باشد، بیشتر مقادیر «NaN» هستند.

خواندن دیتا

```
# Products
dtf_products = pd.read_excel("data_movies.xlsx", sheet_name="products")

dtf_products = dtf_products[~dtf_products["genres"].isna()]
dtf_products["product"] = range(0,len(dtf_products))
dtf_products["name"] = dtf_products["title"].apply(lambda x: re.sub("[\(\[].*?[\)\]]", "", x).strip())
dtf_products["date"] = dtf_products["title"].apply(lambda x: int(x.split("(")[-1].replace(")","").strip())
                                                    if "(" in x else np.nan)

## add features
dtf_products["date"] = dtf_products["date"].fillna(9999)
dtf_products["old"] = dtf_products["date"].apply(lambda x: 1 if x < 2000 else 0)

dtf_products
```

دیتـا را از فایـل اکسـل بـه وسـیله ی read_excel میخـوانیم و سـپس بـه مرتـب کـردن داده هـا در دیتـافریم و اسـتخراج ویژگـی هـا و ایجـاد سـتون هـای جدیـد میپـردازیم مـثلا در عنـوان فـیلم کـه بـه صـورت title(date) هست تـاریخ را از عنـوان جـدا میکنـیم و در دو سـتون جدیـد قـرار میـدهیم و همچنـین سـتون old را ایجـاد میکنـیم که مشخص میکند فیلم قدیمی است یا خیر

درنهایت دیتا فریم جدید به صورت زیر خواهد بود

	movieId	title	genres	product	name	date	old
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	0	Toy Story	1995.0	1
1	2	Jumanji (1995)	Adventure|Children|Fantasy	1	Jumanji	1995.0	1
2	3	Grumpier Old Men (1995)	Comedy|Romance	2	Grumpier Old Men	1995.0	1
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	3	Waiting to Exhale	1995.0	1
4	5	Father of the Bride Part II (1995)	Comedy	4	Father of the Bride Part II	1995.0	1

به همین ترتیب برای کاربران هم عمل میکنیم

```
# Users
dtf_users = pd.read_excel("data_movies.xlsx", sheet_name="users").head(10000)

dtf_users["user"] = dtf_users["userId"].apply(lambda x: x-1)

dtf_users["timestamp"] = dtf_users["timestamp"].apply(lambda x: datetime.fromtimestamp(x))
dtf_users["daytime"] = dtf_users["timestamp"].apply(lambda x: 1 if 6<int(x.strftime("%H"))<20 else 0)
dtf_users["weekend"] = dtf_users["timestamp"].apply(lambda x: 1 if x.weekday() in [5,6] else 0)

dtf_users = dtf_users.merge(dtf_products[["movieId","product"]], how="left")
dtf_users = dtf_users.rename(columns={"rating":"y"})

dtf_users
```

بـــرای دیتـــافریم ratings از روی ســـتون timestamp مقـــادیری مثـــل daytime و weekend را اســـتخراج میکنیم و در یک ستون جدید میگذاریم ، ایـن مقـادیر بیـانگر آن اسـت کـه در چـه سـاعتی در روز فـیلم مشـاهده شـده و آیا این روز آخر هفته بوده یا خیر .

و همچنـین سـتون product کـه ردیـف و شـماره ی هـر فـیلم در دیتـافریم movies را مشـخص میکنـد را بـه دیتافریم ratings اضافه میکنیم

این اطلاعات به عنوان ویژگی در آموزش سیستم ما اثر خواهد داشت

درنهایت دیتا فریم جدید به صورت زیر خواهد بود

	userId	movieId	y	timestamp	user	daytime	weekend	product
0	1	1	4.0	2000-07-30 20:45:03	0	0	1	0
1	1	3	4.0	2000-07-30 20:20:47	0	0	1	2
2	1	6	4.0	2000-07-30 20:37:04	0	0	1	5
3	1	47	5.0	2000-07-30 21:03:35	0	0	1	43
4	1	50	5.0	2000-07-30 20:48:51	0	0	1	46

سپس از دیتافریم ها ستون هایی که مورد نیاز است را انتخاب میکنیم

برای دیتافریم product ستون های زیر را انتخاب میکنیم

```
# select only useful columns
dtf_products = dtf_products[["product","name","old","genres"]].set_index("product")
dtf_products.head()
```

product	name	old	genres
0	Toy Story	1	Adventure|Animation|Children|Comedy|Fantasy
1	Jumanji	1	Adventure|Children|Fantasy
2	Grumpier Old Men	1	Comedy|Romance
3	Waiting to Exhale	1	Comedy|Drama|Romance
4	Father of the Bride Part II	1	Comedy

برای دیتافریم users ستون های زیر را انتخاب میکنیم

```
dtf_users = dtf_users[["user","product","y"]]
dtf_users.head()
```

	user	product	y
0	0	0	4.0
1	0	2	4.0
2	0	5	4.0
3	0	43	5.0
4	0	46	5.0

و در نهایت دیتافریم context که شامل ویژگی های اضافی مورد استفاده در سیستم است

```python
# extract context
dtf_context = dtf_users[["user","product","daytime","weekend"]]
dtf_context.head()
```

	user	product	daytime	weekend
0	0	0	0	1
1	0	2	0	1
2	0	5	0	1
3	0	43	0	1
4	0	46	0	1

نرمال کردن و مرتب سازی داده ها

در این قسمت برای دیتافریم product ما ژانر های هر فیلم را که به | به هم متصل هستند را از هم جدا میکنیم و در ستون های جداگانه قرار میدهیم و برای هر فیلم با ۰ یا ۱ مشخص میکنیم که فیلم ما شامل چه ژانر هایی می شود

```python
tags = [i.split("|") for i in dtf_products["genres"].unique()]
columns = list(set([i for lst in tags for i in lst]))
columns.remove('(no genres listed)')
print(columns)
```

دیتا فریم ما به این صورت در خواهد آمد

product	name	old	genres	Mystery	Children	Comedy	Adventure	Thriller	Drama	Horror	...	Fantasy	Western	War	Sci-Fi	Film-Noir	Romance	Animation
0	Toy Story	1	Adventure|Animation|Children|Comedy|Fantasy	0	1	1	1	0	0	0	...	1	0	0	0	0	0	1
1	Jumanji	1	Adventure|Children|Fantasy	0	1	0	1	0	0	0	...	1	0	0	0	0	0	0
2	Grumpier Old Men	1	Comedy|Romance	0	0	1	0	0	0	0	...	0	0	0	0	0	1	0
3	Waiting to Exhale	1	Comedy|Drama|Romance	0	0	1	0	0	1	0	...	0	0	0	0	0	1	0
4	Father of the Bride Part II	1	Comedy	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0

ساخت ماتریس اصلی

ماتریس نهایی را به طوری میسازیم که سطر ها user ها ، ستون ها product ها و مقادیر آن ratings ها یا همان y باشد

```
tmp = dtf_users.copy()
dtf_users = tmp.pivot_table(index="user", columns="product", values="y")
missing_cols = list(set(dtf_products.index) - set(dtf_users.columns))
for col in missing_cols:
    dtf_users[col] = np.nan
dtf_users = dtf_users[sorted(dtf_users.columns)]
dtf_users
```

در اینجا برای درایه هایی که مقدار رای را ندارند ، nan قرار میدهیم و در نهایت ماتریس به شکل زیر خواهد بود

product	0	1	2	3	4	5	6	7	8	9	...	9731	9732	9733	9734	9735	9736	9737	9738	9739	9740
user																					
0	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
61	NaN	4.0	NaN	NaN	NaN	4.5	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
62	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63	4.0	NaN	3.5	NaN	NaN	4.5	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
64	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
65	4.0	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

بخش دوم – پیش پردازش

در مرحله ی پیش پردازش ما داده های خود را برای پردازش آماده میکنیم که این مرحله شامل دو بخش است : scalling و partitioning

Scalling : در این مرحله با استفاده از minMaxScaler داده های rating را در یک رنج مشخصی قرار میدهیم، در اینجا بین ۰,۵ تا ۱

```
dtf_users = pd.DataFrame(preprocessing.MinMaxScaler(feature_range=(0.5,1)).fit_transform(dtf_users.values),
                         columns=dtf_users.columns, index=dtf_users.index)
dtf_users
```

و حالا تمام مقادیر rating ما به تناسب، در این بازه قرار میگیرند

Partitioning : در این مرحله تمام دیتا را به دو بخش train و test به نسبت ۸۰ به ۲۰ بخش بندی میکنیم

```
split = int(0.8*dtf_users.shape[1])
split
# Train
dtf_train = dtf_users.loc[:, :split-1]
print("non-null data:", dtf_train[dtf_train>0].count().sum())
dtf_train
# Test
dtf_test = dtf_users.loc[:, split:]
print("non-null data:", dtf_test[dtf_test>0].count().sum())
```

حالا یک کاربر برای نمونه انتخاب میکنیم

```
# Select a user
i = 1
train = dtf_train.iloc[i].to_frame(name="y")
test = dtf_test.iloc[i].to_frame(name="y")
```

از مجموعه ی تست همه ی product ها را میگیریم اما مقدار y را مخفی میکنیم

```
tmp = test.copy()
tmp["y"] = np.nan
train = train.append(tmp)
train.tail()
```

product	y
9736	NaN
9737	NaN
9738	NaN
9739	NaN
9740	NaN

حال باید وزن هایی که کاربر به هر ویژگی می دهد را تخمین بزنیم. ما بردار کاربر-محصولات و ماتریس محصولات-ویژگی ها را داریم.

```
# Model
usr = train[["y"]].fillna(0).values.T
prd = dtf_products.drop(["name","genres"],axis=1).values
print("Users", usr.shape, " x  Products", prd.shape)
```

ماتریس کاربران - محصولات: با استفاده از fillna مقادیر null را در ستون y دیتافریم train با صفر جایگزین میکنیم و مقادیر آن را با استفاده از T که نوعی دسترسی به متد transpose است، سطر و ستون های آن را جابجا میکنیم تا روی مورب اصلی خودش منعکس شود

ماتریس محصولات - ویژگی ها : با استفاده از متد drop ، ستون name , genres را حذف میکنیم

```
## usr_ft(users,fatures) = usr(users,products) x prd(products,features)
usr_ft = np.dot(usr, prd)
## normalize
weights = usr_ft / usr_ft.sum()
## predicted rating(users,products) = weights(users,fatures) x prd.T(features,products)
pred = np.dot(weights, prd.T)

test = test.merge(pd.DataFrame(pred[0], columns=["yhat"]), how="left", left_index=True, right_index=True).reset_index()
test = test[~test["y"].isna()]
```

ضرب داخلی ماتریس کاربر-محصولات و محصولات- ویژگی ها را حساب میکنیم و سپس نرمالایز میکنیم

سپس برای مقدار پیش بینی ضرب داخلی وزن های هر کاربر به ویژگی و ماتریس ویژگی – محصولات را حساب میکنیم و در آخر نتیجه را در ستون yhat قرار میدهیم

و دیتا فریم ما برای یک کاربر مشخص به صورت زیر خواهد بود

	product	y	yhat
271	8063	0.812500	0.364068
513	8305	1.000000	0.441084
584	8376	0.777778	0.077697
674	8466	0.800000	0.215465
717	8509	0.500000	0.477922
758	8550	0.875000	0.356925
889	8681	1.000000	0.327055
1036	8828	0.500000	0.033233

محاسبه ی دقت

```python
# Evaluate
def mean_reciprocal_rank(y_test, predicted):
    score = []
    for product in y_test:
        mrr = 1 / (list(predicted).index(product) + 1) if product in predicted else 0
        score.append(mrr)
    return np.mean(score)
```

Mean Reciprocal Rank معیاری برای ارزیابی سیستم هایی است که لیست رتبه بندی شده ای از پاسخ ها را به پرس و جوها برمی گرداند. برای یک پرس و جو ,Reciprocal Rank به صورت ۱/rank است که rank جایگاه پاسخ با بالاترین رتبه است. اگر در پرس و جو پاسخ صحیحی برگردانده نشد، Reciprocal Rank ۰ است.

$$\mathrm{MRR} = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{\mathrm{rank}_i}$$

برای هر مقدار پیش بینی شده در مجموعه test ، Reciprocal Rank محاسبه میشود که اگر موجود بود ، یک بر روی index+1 که Reciprocal Rank میباشد را برگرداند و در غیر این صورت ۰ را برگرداند و در نهایت میانگین همه ی rr ها را حساب کند

```
top = 5
y_test = test.sort_values("y", ascending=False)["product"].values[:top]
print("y_test:", y_test)

predicted = test.sort_values("yhat", ascending=False)["product"].values[:top]
print("predicted:", predicted)

true_positive = len(list(set(y_test) & set(predicted)))
print("true positive:", true_positive, "("+str(round(true_positive/top*100,1))+"%)")
print("accuracy:", str(round(metrics.accuracy_score(y_test,predicted)*100,1))+"%")
print("mrr:", round(mean_reciprocal_rank(y_test, predicted),2))
```

در این قسمت مجموعه ی y_test که مقادیر دیتافریم test ما هست را ایجاد میکنیم به این صورت که ابتدا بر اساس میزان رای ، به صورت نزولی آن را مرتب میکنیم و ۵ تا از فیلم هایی که کاربر بیشترین رای را به آن ها داده است را میگیریم

و برای مجموعه ی predicted که شامل مقادیر پیش بینی شده است دقیقا به همین ترتیب عمل میکنیم و بر حسب yhat ۵ مقدار اول را میگیریم

و سپس مقادیر مشترک ، دقت و mrr را برای این دو مجموعه حساب میکنیم و نتیجه ی زیر حاصل میشود

```
y_test: [8305 8681 8550 8063 8466]
predicted: [8509 8305 8063 8550 8681]
true positive: 4 (80.0%)
accuracy: 0.0%
mrr: 0.26
```

بخش سوم – طراحی مدل Neural Collaborative Filtering

حالا به سراغ الگوریتم اصلی میرویم و مدل خود را بسازیم

ایده اصلی این است که از لایه embedding یک شبکه عصبی برای ایجاد ماتریس های کاربران و محصولات استفاده کنیم.ورودی ها جفت کاربر–محصول هستند و خروجی رتبه بندی است.

	user	product	y
0	1	8063	0.812500
1	1	8305	1.000000
2	1	8376	0.777778
3	1	8466	0.800000
4	1	8509	0.500000

Neural Collaborative Filtering غیر خطی بودن شبکه های عصبی و matrix factorization را ترکیب می کند. این مدل به گونه ای طراحی شده است که با استفاده از آن نه تنها برای Collaborative Filtering ، بلکه برای یک شبکه عصبی عمیق کاملاً متصل، از فضای embedding نهایت استفاده را ببرد. همچنین باید الگوها و ویژگی هایی را که ممکن است matrix factorization از دست بدهد را دارا باشد

```
# Input layer
xusers_in = layers.Input(name="xusers_in", shape=(1,))
xproducts_in = layers.Input(name="xproducts_in", shape=(1,))
```

ورودی ها را برای فیلم ها و کاربران با استفاده از متد input در keras مشخص میکنیم

Input() به ما اجازه می‌دهد یک مدل Keras را فقط با دانستن ورودی‌ها و خروجی‌های مدل بسازیم.

به عنوان مثال، اگر a، b و c تانسورهای Keras باشند، انجام این موارد ممکن می شود:

model = Model(input=[a, b], output=c)

Shape(1,) مشخص میکند که ورودی مورد انتظار دسته ای از بردارهای یک بعدی خواهد بود.

```
# A) Matrix Factorization
## embeddings and reshape
cf_xusers_emb = layers.Embedding(name="cf_xusers_emb", input_dim=usr, output_dim=embeddings_size)(xusers_in)
cf_xusers = layers.Reshape(name='cf_xusers', target_shape=(embeddings_size,))(cf_xusers_emb)
## embeddings and reshape
cf_xproducts_emb = layers.Embedding(name="cf_xproducts_emb", input_dim=prd, output_dim=embeddings_size)(xproducts_in)
cf_xproducts = layers.Reshape(name='cf_xproducts', target_shape=(embeddings_size,))(cf_xproducts_emb)
## product
cf_xx = layers.Dot(name='cf_xx', normalize=True, axes=1)([cf_xusers, cf_xproducts])
```

حالا مدل matrix factorization را میسازیم

در ابتدا embedding users را میسازیم. متد Embedding اعداد صحیح مثبت (شاخص ها) را به بردارهای متراکم با اندازه ثابت تبدیل می کند. که پارامتر های آن، input_dim که به معنی اندازه ی ورودی ماست و output_dim که همان اندازه ی ثابت بردار dense یا متراکم ماست و xuser_in را به عنوان ورودی به آن میدهیم

سپس reshape را انجام میدهیم . لایه ی reshape لایه ای است که ورودی ها را به شکل داده شده تغییر شکل می دهد.

خروجی این لایه برای کاربر و محصول در نهایت در هم ضرب داخلی میشوند و مدل matrix factorization ما را میسازد

```
# B) Neural Network
## embeddings and reshape
nn_xusers_emb = layers.Embedding(name="nn_xusers_emb", input_dim=usr, output_dim=embeddings_size)(xusers_in)
nn_xusers = layers.Reshape(name='nn_xusers', target_shape=(embeddings_size,))(nn_xusers_emb)
## embeddings and reshape
nn_xproducts_emb = layers.Embedding(name="nn_xproducts_emb", input_dim=prd, output_dim=embeddings_size)(xproducts_in)
nn_xproducts = layers.Reshape(name='nn_xproducts', target_shape=(embeddings_size,))(nn_xproducts_emb)
## concat and dense
nn_xx = layers.Concatenate()([nn_xusers, nn_xproducts])
nn_xx = layers.Dense(name="nn_xx", units=int(embeddings_size/2), activation='relu')(nn_xx)
```

حالا مدل شبکه عصبی را دقیقا با مراحلی مشابه بالا میسازیم با این تفاوت که خروجی لایه ی reshape برای کاربران و محصولات ، با استفاده از متد Concatenate به هم متصل میکند. این متد ورودی ها را می گیرد و آن ها را در طول یک بُعد مشخص به هم متصل می کند. ورودی ها باید در همه ابعاد به جز بُعد Concatenate ، اندازه یکسانی داشته باشند.

و سپس متد Dense ، لایه ی dense همان لایه ی پنهان است که در آن هر گره به تمامی گره های لایه ی همبند بعدی متصل است و ورودی هایش را از لایه ی قبلی میگیرد.

در واقع لایه ی dense عملیات زیر را انجام میدهد

output = activation(dot(input, kernel) + bias)

kernel همان ماتریس وزن است و bias بردار بایاس است که توسط لایه ساخته میشوند

لایه dense روی لایه خروجی حاصل ضرب داخلی ورودی و ماتریس وزن را انجام می دهد.

Unit در آن ابعاد فضای خروجی را مشخص میکند

Activation تابع فعال سازی ما برای شبکه ی عصبی است که در اینجا relu انتخاب شده است

```
# Merge A & B
y_out = layers.Concatenate()([cf_xx, nn_xx])
y_out = layers.Dense(name="y_out", units=1, activation='linear')(y_out)

model = models.Model(inputs=[xusers_in,xproducts_in], outputs=y_out, name="Neural_CollaborativeFiltering")
model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mean_absolute_percentage_error'])
model.summary()
```

در نهایت مدل های A و B را با هم ادغام میکنیم و مدل خروجی نهایی را به دست می آوریم

ورودی و خروجی مدل و نام آن را مشخص میکنیم و با استفاده از متد compile مدل را برای آموزش configure میکنیم

Optimizer یا بهینه ساز ،تابع یا الگوریتمی است که ویژگی های شبکه عصبی مانند وزن و نرخ یادگیری را تغییر می دهد. بنابراین، به کاهش ضرر کلی و بهبود دقت کمک می کند.

Adam یک الگوریتم بهینه سازی جایگزین برای گرادیان کاهشی تصادفی برای آموزش مدل های یادگیری عمیق است.

Learning rate: به طور خاص، نرخ یادگیری یک پارامتر قابل تنظیم است که در آموزش شبکه‌های عصبی استفاده می‌شود که دارای یک مقدار مثبت کوچک است، اغلب در محدوده بین ۰,۰ و ۱,۰. نرخ یادگیری سرعت انطباق مدل با مسئله را کنترل می کند که در اینجا نرخ یادگیری ما ۰,۰۰۱ است

تابع ضرر یا loss : تابعی است که مقادیر هدف و خروجی پیش بینی شده را مقایسه می کند. اندازه گیری می کند که شبکه عصبی چگونه داده های آموزشی را مدل سازی می کند. هنگام آموزش، هدف ما به حداقل رساندن این ضرر بین خروجی های پیش بینی شده و هدف است که تابع ضرر ما در اینجا mean absolute error است که جلوتر به توضیح آن میپردازیم

Metric یا متریک :تابعی است که برای ارزیابی در مورد عملکرد مدل استفاده می شود.توابع متریک شبیه توابع ضرر هستند، با این تفاوت که نتایج حاصل از ارزیابی یک متریک هنگام آموزش مدل استفاده نمی شود.

حالا که مدل را ساختیم آن را train و سپس test میکنیم

```
# train
training = model.fit(x=[train["user"], train["product"]], y=train["y"],
                     epochs=100, batch_size=128, shuffle=True, verbose=0, validation_split=0.3)
model = training.model
```

تابع fit مدل را برای تعداد ثابتی از دوره‌ها یا epoches (تکرار روی یک مجموعه داده) آموزش می‌دهد.

X: دیتای ورودی

Y : دیتای هدف

Epoch : تعداد دوره ها یک فراپارامتر است که تعداد دفعاتی را که الگوریتم یادگیری در کل مجموعه داده آموزشی کار می کند را تعیین می کند. یک دوره به این معنی است که هر نمونه در مجموعه داده آموزشی فرصتی برای به روز رسانی پارامترهای مدل داخلی داشته است. یک دوره از یک یا چند دسته تشکیل شده است.

batch_size : تعداد نمونه هایی است که در یک زمان به شبکه ارسال می شود

validation_split: روشی برای پیش‌بینی تناسب یک مدل با یک مجموعه آزمایشی فرضی در زمانی که یک مجموعه آزمایش صریح در دسترس نیست

```
# test
test["yhat"] = model.predict([test["user"], test["product"]])
test
```

Predict : پیش بینی های خروجی را برای نمونه های ورودی ایجاد می کند

ارزیابی

```
y_test = test["y"]
predicted = test["yhat"]

print("Mean Absolute Error (Σ|y-pred|/n):", "{:,.0f}".format(metrics.mean_absolute_error(y_test, predicted)))
print("Mean Absolute Perc Error (Σ(|y-pred|/y)/n):", str(round(np.mean(np.abs((y_test-predicted)/predicted)), 2))+"%")
```

برای ارزیابی از mean absolute error استفاده میکنیم که به این صورت از که میانگین مقدار واقعی و مقدار پیش بینی شده را برای داده ها محاسبه میکند و فرمول آن مطابق زیر است

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

MAE = mean absolute error

y_i = prediction

x_i = true value

n = total number of data points

با فراخوانی مدل برای یک یوزر مشخص، مانند قبل عمل میکنیم و نتیجه ی نهایی به صورت زیر خواهد بود

```
--- user 1 ---
y_test: [8305 8681 8550 8063 8466]
predicted: [8828 8681 8550 8466 8305]
true positive: 4 (80.0%)
accuracy: 40.0%
mrr: 0.26
```

همان طور که مشاهده میشود دقت مدل نسبت به روش قبل افزایش یافته است

مدل Collaborative Filtering

در این قسمت به پیاده سازی مدل collabrotive filtering ساده میپردازیم تا بتوانیم آن را با مدل Neural Collaborative Filtering که ساخته ایم مقایسه کنیم

مراحل ساخت مدل اکثرا مشابه ها مدل ساخته شده با روش Neural Collaborative Filtering است که در ادامه توضیح خواهم داد

```
# Users (,1,embedding_size) and Reshape (,embedding_size)
xusers_in = layers.Input(name="xusers_in", shape=(1,))
xusers_emb = layers.Embedding(name="xusers_emb", input_dim=usr, output_dim=embeddings_size)(xusers_in)
xusers = layers.Reshape(name='xusers', target_shape=(embeddings_size,))(xusers_emb)

# Products (,1,embedding_size) and Reshape (,embedding_size)
xproducts_in = layers.Input(name="xproducts_in", shape=(1,))
xproducts_emb = layers.Embedding(name="xproducts_emb", input_dim=prd, output_dim=embeddings_size)(xproducts_in)
xproducts = layers.Reshape(name='xproducts', target_shape=(embeddings_size,))(xproducts_emb)

# Product (,1) values[-1,1]
xx = layers.Dot(name='xx', normalize=True, axes=1)([xusers, xproducts])

# Predict ratings (,1)
y_out = layers.Dense(name="y_out", units=1, activation='linear')(xx)

# Compile
model = models.Model(inputs=[xusers_in,xproducts_in], outputs=y_out, name="CollaborativeFiltering")
model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mean_absolute_percentage_error'])
model.summary()
```

همان طور که میبینید مانند قبل لایه ی input و embedding را برای کاربران و محصولات میسازیم و سپس آن ها را reshape میکنیم

تفاوت این مدل با مدل قبلی در این است که در ادامه ما ضرب داخلی xuser و xproduct را محاسبه میکنیم و به عنوان ورودی به متد dense میدهیم که اینبار تابع فعال سازی این متد خطی یا linear است و به این صورت ، پس از ساختن مدل به همان شیوه ی قبلی ما مدل Collaborative Filtering را داریم که با تست کردن آن به خروجی زیر میرسیم

```
--- user 1 ---
y_test: [8305 8681 8550 8063 8466]
predicted: [8305 8376 8828 8681 8550]
true positive: 3 (60.0%)
accuracy: 20.0%
mrr: 0.29
```

نتیجه گیری

از مقایسه ای این دو مدل نتیجه میگیریم که مدل Neural Collaborative Filtering با مقادیر

true positive : ۸۰٪

accuracy : ۴۰٪

از مدل Collaborative Filtering با مقادیر

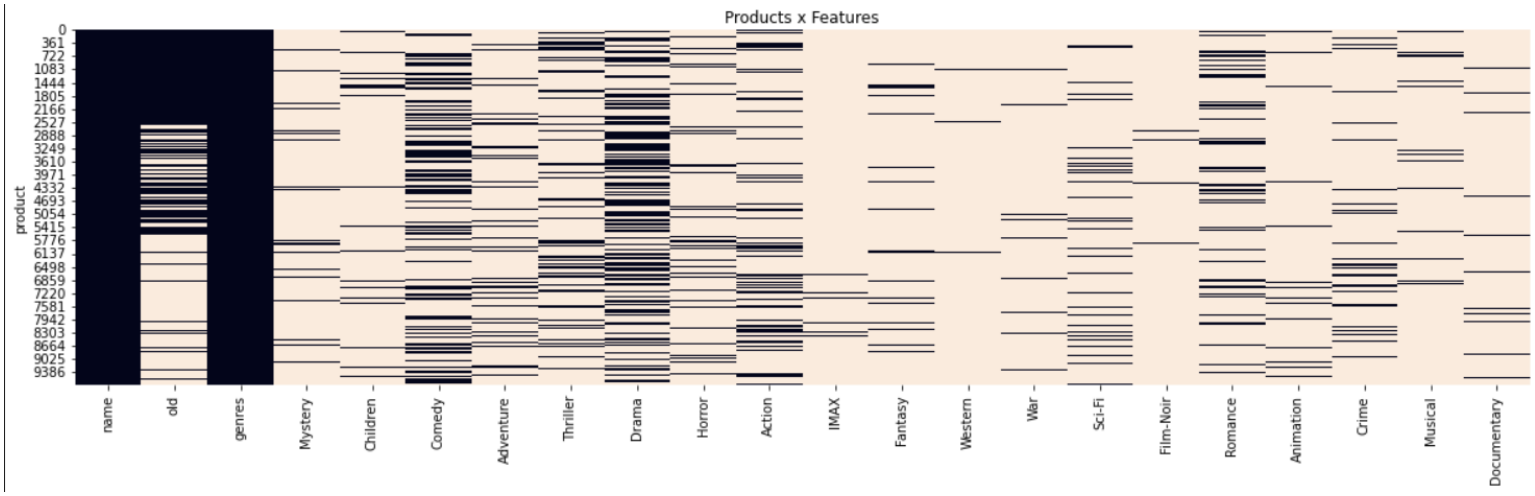
true positive : ۶۰٪

accuracy : ۲۰٪

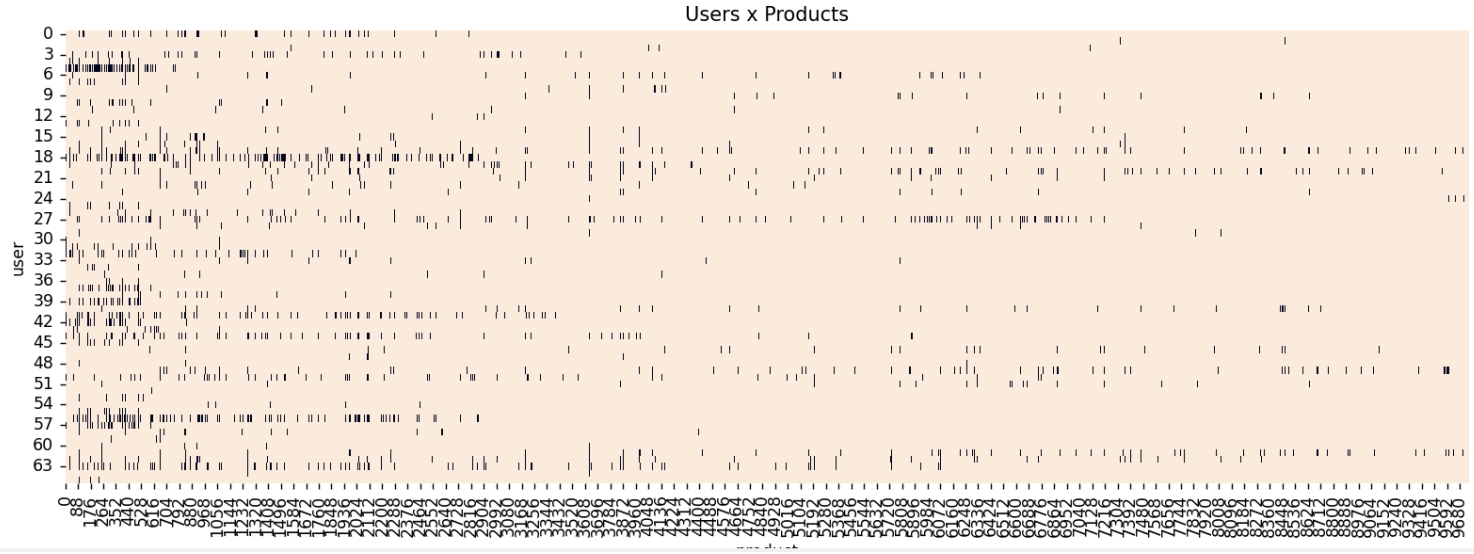
عملکرد و نتیجه ی بهتری دارد

۴- نمودار ها و مدل ها

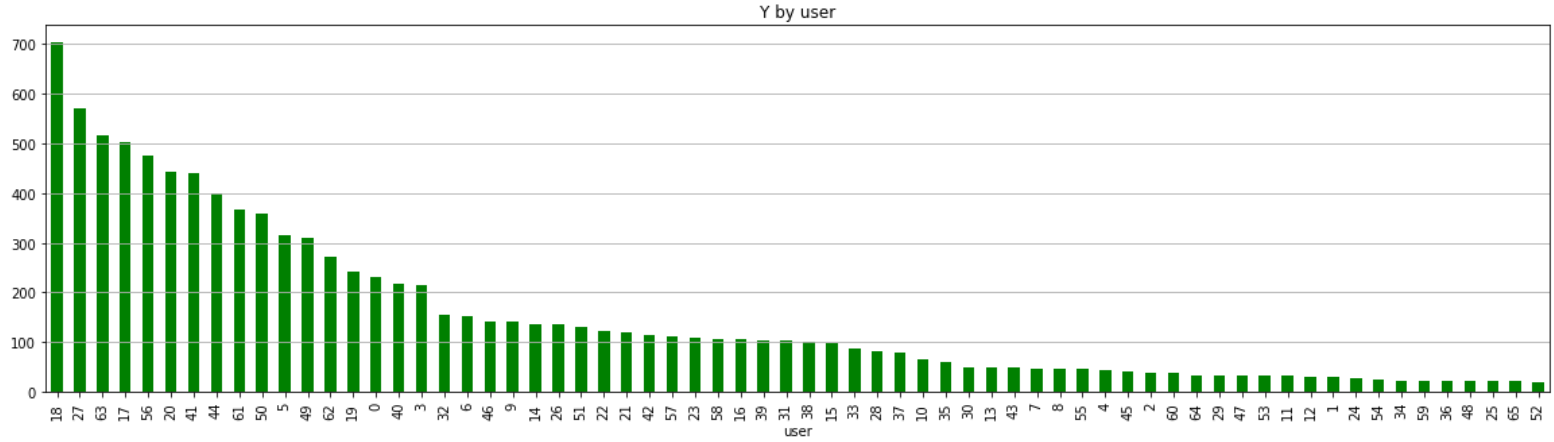
در این قسمت به توضیح نمودار های رسم شده در پروژه که با استفاده از matplotlib ترسیم شده است میپردازیم



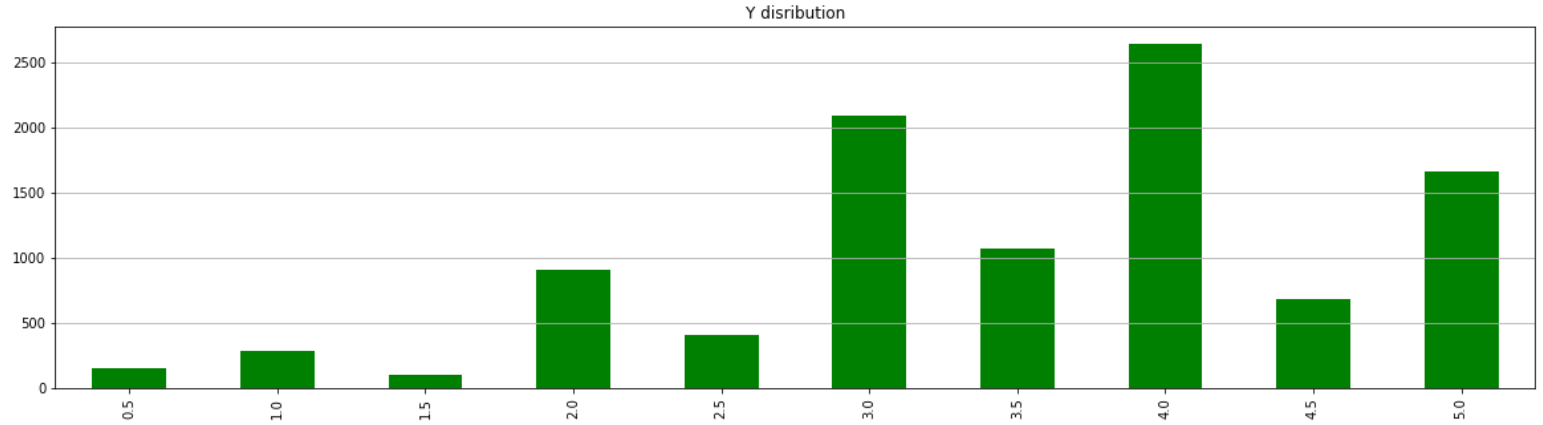
نمودار محصولات-ویژگی ها که برای هر فیلم ویژگی های آن که شامل ژانر های فیلم ، قدیمی بودن یا نبودن و نام آن را مشخص میکند و به این ترتیب فضاهای پر و خالی در دیتافریم ما نیز مشخص میشود



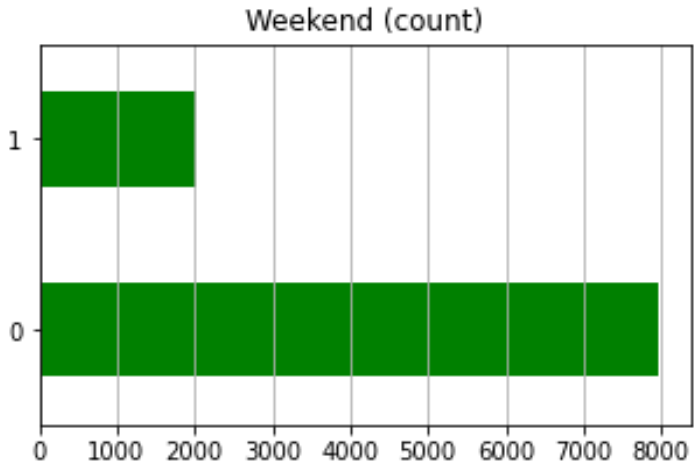
شمای ماتریس اسپارس که users-products که سطر های آن را user ها و ستون های آن را فیلم ها مشخص میکنند و مقدار ها هر درایه ی آن میزان رای هر کاربر به هر فیلم است



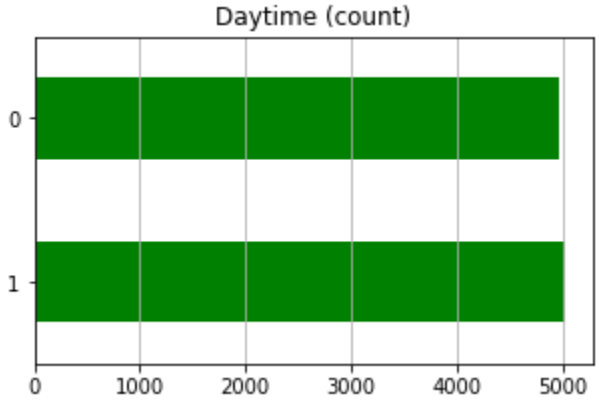
نمودار تعداد رای های داده شده توسط هر user



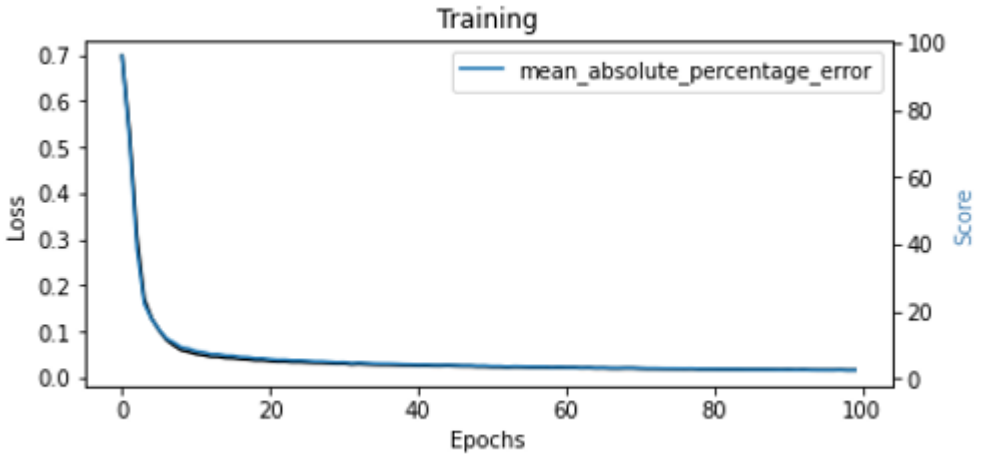
نمودار توزیع رای ها ، نشان میدهد که هر رای به چه تعدادی در مجموعه وجود دارد



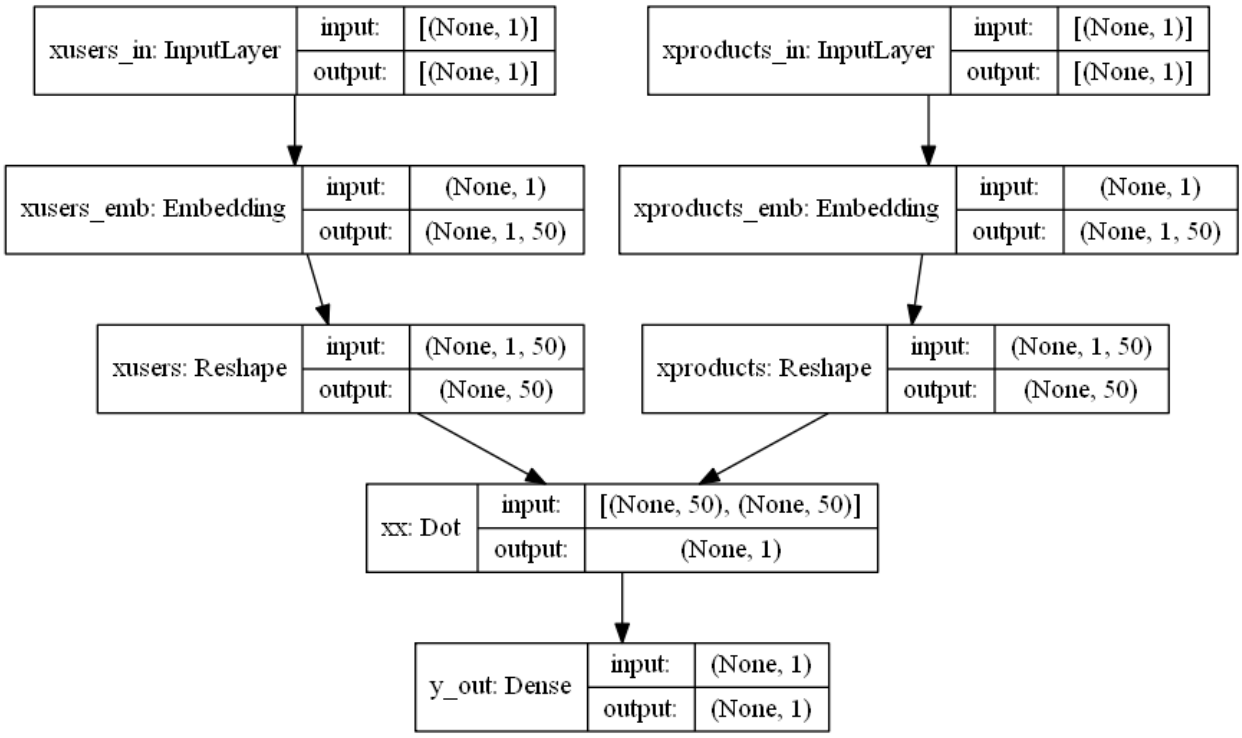
نمودار نشان دهنده ی اینکه چه تعدادی از رای هایی که به فیلم ها داده شده در آخر هفته بوده است



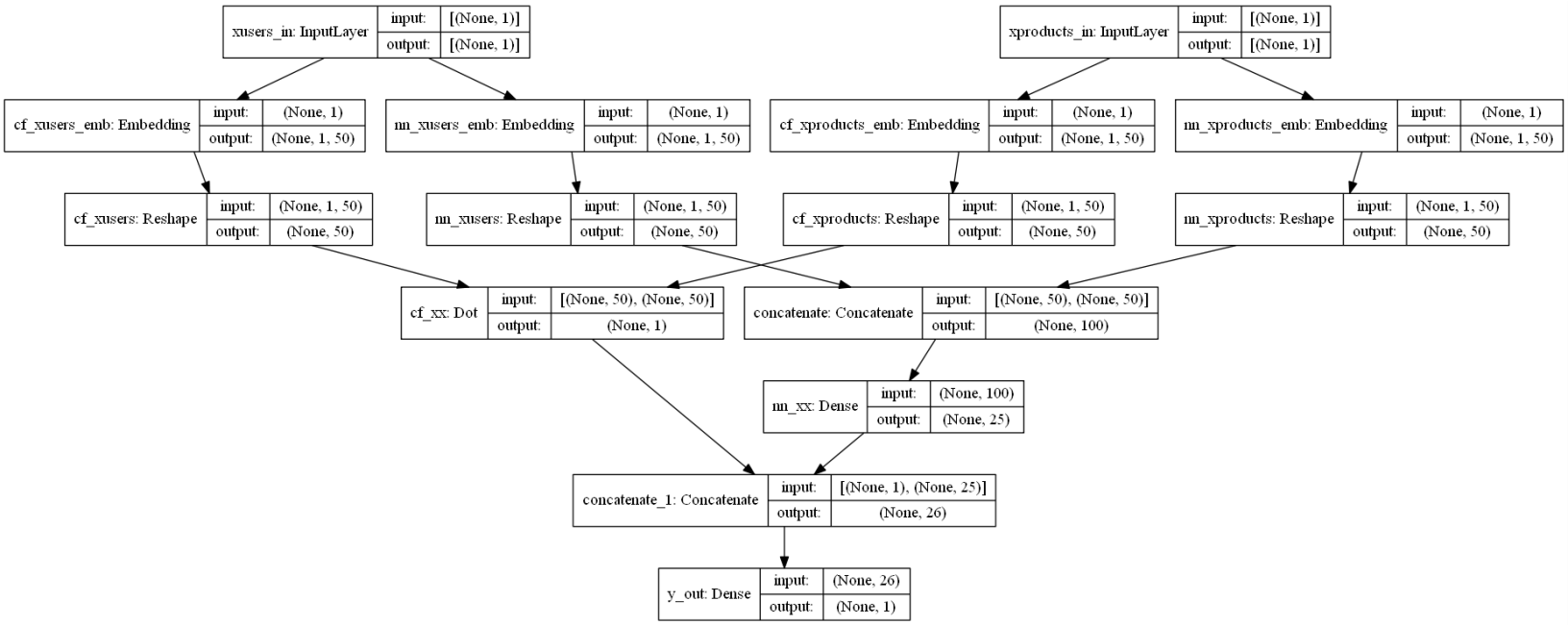
نمودار نشان دهنده ی اینکه چه تعدادی از رای هایی که به فیلم ها داده شده در روز و چه تعداد در شب بوده است



نمودار training مشخص میکند که به ازای هر دوره که روی داده ها پردازش صورت میگیرد میزان loss که بر اساس mean absolute percentage error می باشد چقدر است که همانطور که میبینیم به ازای هر epoch این مقدار کمتر شده که از شیب زیاد به شیب کم و تقریبا ثابت می رسد



مدل colaborartive filtering که مراحل ساخت آن را که قبل تر در قسمت کدنویسی توضیح داده شد مشخص میکند



neural collaborative filtering مدل

۵- جمع بندی

در این پروژه سعی شد تا با روشی موثر و کارآمد ، یک نوع جدیدی از سیستم های توصیه گر با روشی ترکیبی پیاده سازی شود تا بهترین نتیجه حاصل شود

و همچنین نمودار ها و مدل های تصویری برای تحلیل راحت تر پروژه اضافه شده است

برای پیاده سازی این پروژه من علاوه بر مطالعه بر روی روش های معمول پیاده سازی سیستم های توصیه گر، مطالعات و تحقیقاتی هم بر روی شبکه های عصبی و یادگیری عمیق داشته ام

در نهایت کمال تشکر و قدردانی را از استاد راهنمای محترم این پروژه ، سرکار خانم دکتر آزاده طباطبایی دارم

- دلارام شیخ بهایی

۶- منابع

- An Efficient Deep Learning Approach for Collaborative Filtering
- Collaborative Filtering
- Neural Network
- Keras