ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA TOÁN - TIN HỌC

BÁO CÁO CUỐI KỲ

ĐỀ TÀI: SEQUENCE AWARE RECOMMENDER SYSTEMS: SENSITIVITY PERTURBATIONS OF THE SEQUENCE

MÔN HỌC: XỬ LÝ ĐA CHIỀU

Sinh viên thực hiện:

Nguyễn Duy Đạt - 21110262

Giáo viên hướng dẫn:

TS. Kha Tuấn Minh

Ngày 16 tháng 6 năm 2024

Mục lục

1 Giới thiệu

1.1 Sơ lược về Recommender Systems

Hệ thống đề xuất (Recommender Systems) có lẽ là một trong những ứng dụng thành công nhất của Machine Learning (ML) và Artificial Intelligence (AI) trong hai thập kỷ vừa qua. Các hệ thống này đã được thiết kế, thử nghiệm và triển khai thành công trong nhiều lĩnh vực ứng dụng khác nhau để giới thiệu tới người dùng một danh sách các mặt hàng có liên quan, và đặc biệt hữu ích khi danh mục các mặt hàng có khả năng được đề xuất đủ lớn để người dùng không thể thực hiện đề xuất thủ công. Hệ thống đề xuất nói chung được công nhận là một trong những phương pháp tiếp cận hiệu quả để cung cấp cho người dùng những nội dung được cá nhân hóa. Ví dụ, dịch vụ streaming Netflix hiển thị bảng xếp hạng phim được dự đoán ở cấp độ người dùng (user-level) cho khách hàng của mình để giúp họ quyết định nội dung nào là phù hợp và thú vị hơn để xem. Nhà bán lẻ trực tuyến toán cầu Amazon cung cấp những dự đoán xếp hạng sản phẩm cho người sử dụng dựa trên lịch sử mua hàng trước đó của những khách hàng tương tự, vì mục tiêu cuối cùng của các phương pháp này là đề xuất các mặt hàng phù hợp hơn đối với từng người dùng hoặc các mặt hàng nào sẽ được khách hàng chú ý đến và yêu thích, nên các hệ thống này thường được gọi là hệ thống đề xuất.

Hệ thống đề xuất có nguồn gốc xuất phát từ lĩnh vực truy xuất thông tin (information retrieval), xác định nội dung văn bản trực tuyến có thể phù hợp nhất đối với truy vấn của người dùng đã cho, và sau đó sắp xếp danh sách các tài liệu có liên quan nhất được truy xuất dựa trên các hành vi hoạt động của người dùng. Tuy nhiên, mặc dù cách tiếp cận này được nhiều công ty sử dụng rộng rãi vào những năm 90 để đề xuất các mặt hàng cụ thể, nhưng nó đã nhanh chóng được thay thế bằng các kỹ thuật tiên tiến hơn. Bao gồm Content-based Collaborative Filtering (CF) của Balabanovic và Shoham (1997) và Matrix Factorization (MF) của Koren, Bell, và Volinsky (2009). Hệ thống gợi ý với mục đích tìm hiểu mối quan hệ giữa sở thích của người dùng bằng cách sử dụng các thông tin lịch sử mua hàng của người dùng trong ma trận, chẳng hạn như Singular Value Decomposition (SVD) của Zhou, He, Huang và Zhang (2015), mục tiêu là dự đoán sở thích của người dùng trong tương lai trong ma trận rating user-item. Các kỹ thuật này có những điểm mạnh và điểm yếu riêng, nhiều nhà nghiên cứu, các công ty và chuyên gia về AI đã chọn kết hợp các kỹ thuật theo những cách khác nhau để cung cấp các đề xuất tốt hơn cho người dùng và tăng doanh thu tổng thể, sự tham gia của khách hàng hay hiệu suất của các mô hình. Điều này dẫn đến sự phát triển của các hệ thống

như Hybrid Recommender Systems của Bruke (2002), trong đó các kỹ thuật như đề xuất trọng số (Weighted Recommenders) của Claypool et al. (1999), đề xuất kết hợp (Mixed Recommenders) của Smyth và Cotter (2000) và kết hợp tính năng đề xuất (Feature Combined Recommenders) của Basu, Hirsh và Cohen (1998) có lẽ là các phương pháp phổ biến nhất được sử dụng trong lĩnh vực. Mặc dù các kỹ thuật này thường có thể giải quyết một số vấn đề trong hệ thống đề xuất, chẳng hạn như Cold-start problem khi thêm các items hoặc users mới vào hệ thống, chúng vẫn gặp phải một số vấn đề về hiệu suất trong một số lĩnh vực ứng dụng.

Hệ thống đề xuất truyền thống thường bỏ qua khía cạnh quan trọng về hành vi người dùng: sự thay đổi theo thời gian. Hệ thống đề xuất dựa trên chuỗi (Sequence Aware Recommender Systems) ra đời nhằm giải quyết vấn đề này bằng cách mô hình hóa hành vi người dùng theo chuỗi tương tác, ghi nhớ lịch sử mua sắm, xem phim, nghe nhạc,... để đưa ra dự đoán chính xác hơn về sở thích tương lai.

Ưu điểm vượt trội của hệ thống đề xuất dựa trên chuỗi:

- **Hiểu rõ xu hướng hành vi:** Hệ thống có thể nắm bắt xu hướng thay đổi sở thích của người dùng theo thời gian, từ đó đưa ra đề xuất phù hợp hơn với nhu cầu hiện tại.
- **Cá nhân hóa cao độ:** Việc ghi nhớ lịch sử tương tác giúp hệ thống đề xuất các sản phẩm phù hợp với từng cá nhân một cách chính xác hơn.
- **Hiệu quả trong việc tiếp thị lại:** Hệ thống có thể đề xuất các sản phẩm liên quan đến những sản phẩm mà người dùng đã từng mua hoặc tương tác trước đây, thúc đẩy tỷ lệ mua hàng.

Tuy nhiên, hệ thống đề xuất dựa trên chuỗi cũng tiềm ẩn một số hạn chế nhất định như:

- **Độ phức tạp cao:** Việc mô hình hóa hành vi theo chuỗi đòi hỏi thuật toán phức tạp hơn và tốn nhiều tài nguyên tính toán hơn so với hệ thống đề xuất truyền thống.
- **Vấn đề Độ nhạy cảm với nhiễu:** Hệ thống dễ bị ảnh hưởng bởi những thay đổi nhỏ trong dữ liệu đầu vào, dẫn đến kết quả dự đoán không chính xác.
- **Yêu cầu dữ liệu lớn:** Hệ thống cần có lượng dữ liệu tương tác khổng lồ để học hỏi và mô hình hóa hành vi người dùng một cách chính xác.

Giải quyết vấn đề nhạy cảm với nhiễu là một thách thức lớn đối với hệ thống đề xuất dựa trên chuỗi. Do bản chất mô hình hóa chuỗi, hệ thống này dễ bị ảnh hưởng bởi những thay đổi nhỏ trong dữ liệu đầu vào, dẫn đến kết quả dự đoán không chính xác. Điều này có thể gây ra một số hệ quả tiêu cực như đề xuất không phù hợp, mất doanh thu hay gây tổn hại đến danh tiếng của doanh nghiệp hay người dùng. Để giải quyết vấn đề này, một số giải pháp tiềm năng có thể được áp dụng:

- **Kỹ thuật xử lý nhiễu trong dữ liệu:** p dụng các kỹ thuật lọc nhiễu, xử lý ngoại lệ để đảm bảo chất lượng dữ liệu đầu vào cho hệ thống đề xuất.
- **Phát triển thuật toán mô hình hóa chuỗi robust:** Nghiên cứu và phát triển các thuật toán mô hình hóa chuỗi có khả năng chống nhiễu tốt hơn, ít bị ảnh hưởng bởi những thay đổi nhỏ trong dữ liệu.
- **Kết hợp các phương pháp học máy khác:** Kết hợp các phương pháp học máy khác như học tăng cường (Reinforcement Learning) để tối ưu hóa hiệu quả đề xuất và giảm thiểu ảnh hưởng của nhiễu.

Việc giải quyết vấn đề nhạy cảm với dữ liệu nhiễu là chìa khóa để xây dựng hệ thống đề xuất dựa trên chuỗi hiệu quả, mang lại trải nghiệm người dùng tốt hơn và thúc đẩy doanh thu cho doanh nghiệp.

1.2 Đặt vấn đề

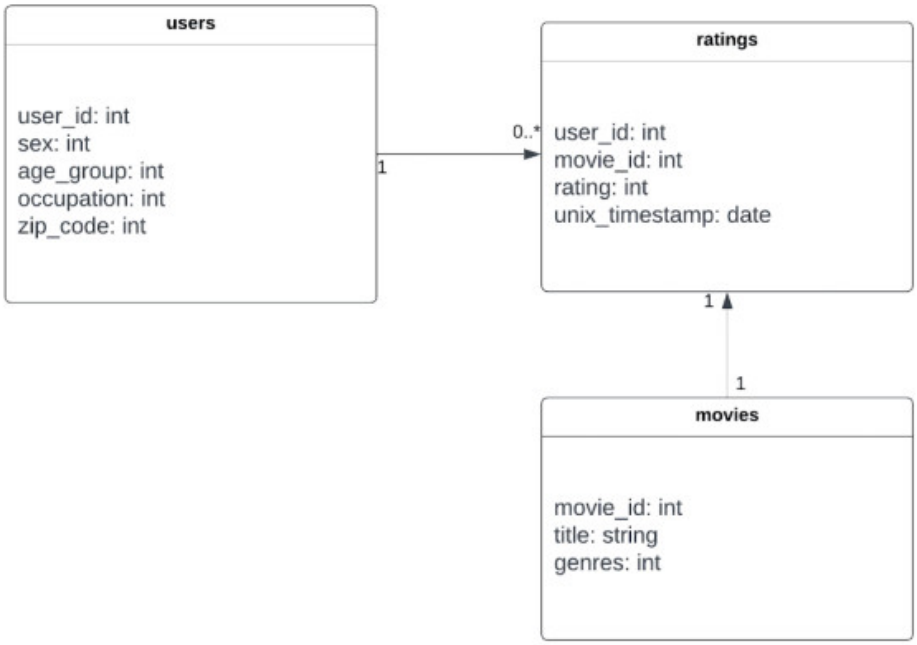
Một giả định quan trọng khi làm việc với dữ liệu là đòi hỏi phải chính xác và không có lỗi. Tuy nhiên, trong thực tế thu thập dữ liệu thường không lý tưởng như vậy, với nhiều vấn đề và tình huống không lường trước sẽ gây nên sai sót. Điều này cũng đúng với các hệ thống gợi ý phim, nơi dữ liệu được thu thập qua phản hồi trực tiếp từ người dùng. Có thể có khoảng cách lớn giữa thời điểm xem phim và thời điểm đánh giá phim, điều này có thể làm giảm độ chính xác của dữ liệu thu thập được.

Một nguyên nhân khác làm xáo trộn dữ liệu là hiện tượng đánh giá ngẫu nhiên các bộ phim, có thể do hiện tượng chia sẻ tài khoản và mật khẩu, một hành vi phổ biến với ước tính khoảng 22.6 % người Mỹ tham gia. Điều này dẫn đến không thể xác định chính xác rằng dữ liệu thu thập được là từ một người dùng duy nhất hay nhiều người.

Báo cáo này nhằm mục đích khám phá ảnh hưởng của các loại sai sót dữ liệu đến hệ thống gợi ý. Bằng cách thử nghiệm các phương pháp xử lý dữ liệu bị lỗi để có thể nhận diện và xử lý những trường hợp dữ liệu bất thường, từ đó cải thiện hiệu quả gợi ý.

2 Data Pre-Processing

Đối với bài báo cáo này, chúng ta sẽ sử dụng bộ dữ liệu MovieLens với tổng cộng một triệu bản ghi, bộ dữ liệu này được đánh giá là đủ lớn để có thể phù hợp với phạm vi mà chúng ta cần tìm hiểu. Để hiểu rõ hơn về cấu trúc của bộ dữ liệu này và thực hiện các kết nối cần thiết, ta có thể xem qua ở Hình 1 sơ đồ UML gồm 3 bảng dữ liệu chính: **users**, **ratings** và **movies**.



Hình 1: Data Description

	user_id	sex	age_group	occupation	zip_code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

Hình 2: Users Description

	user_id	movie_id	rating	unix_timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

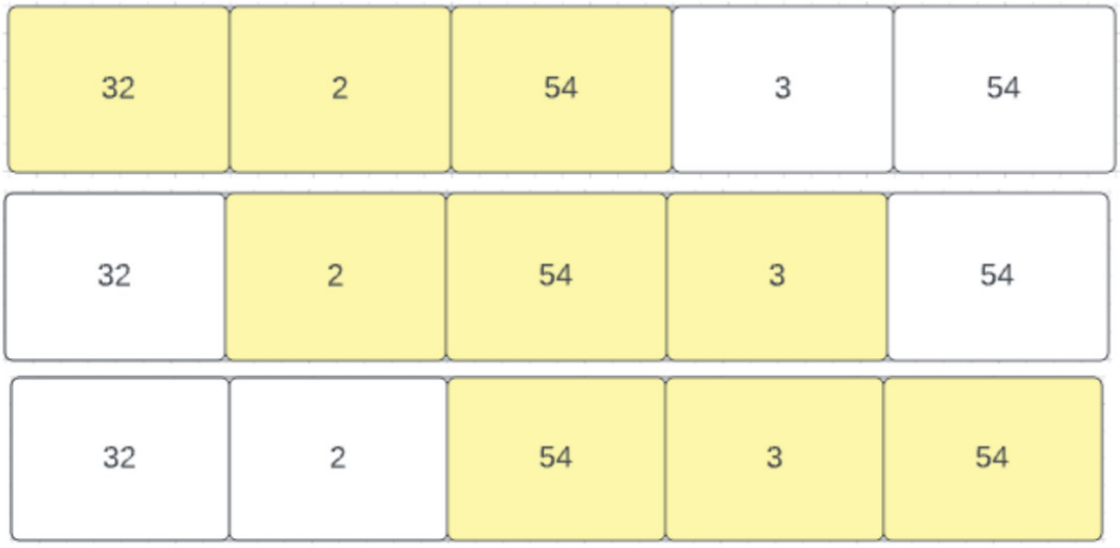
Hình 3: Ratings Description

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

Hình 4: Movies Description

Vì báo cáo này quan tâm đến các chuỗi hành vi nên việc bảo toàn thời gian thực trong dữ liệu trở nên cực kỳ quan trọng. Chúng ta khai thác các tín hiệu tuần tự ẩn chứa bên trong chuỗi hành vi của người dùng bằng cách sử dụng một hàm để tạo ra các chuỗi có thể có độ dài L từ lịch sử xem phim của người dùng. Ví dụ về điều này được minh họa trong Hình 5, một ví dụ với $L = 3$

và *StepSize* = 1, trong đó người dùng này có 5 bộ phim trong lịch sử xem của họ. Nếu chúng ta có một tập hợp người dùng U, và trung bình một người dùng có lịch sử H, chúng là sẽ có khoảng *|U|(H - L + 1)* chuỗi.



Hình 5: Window Creation

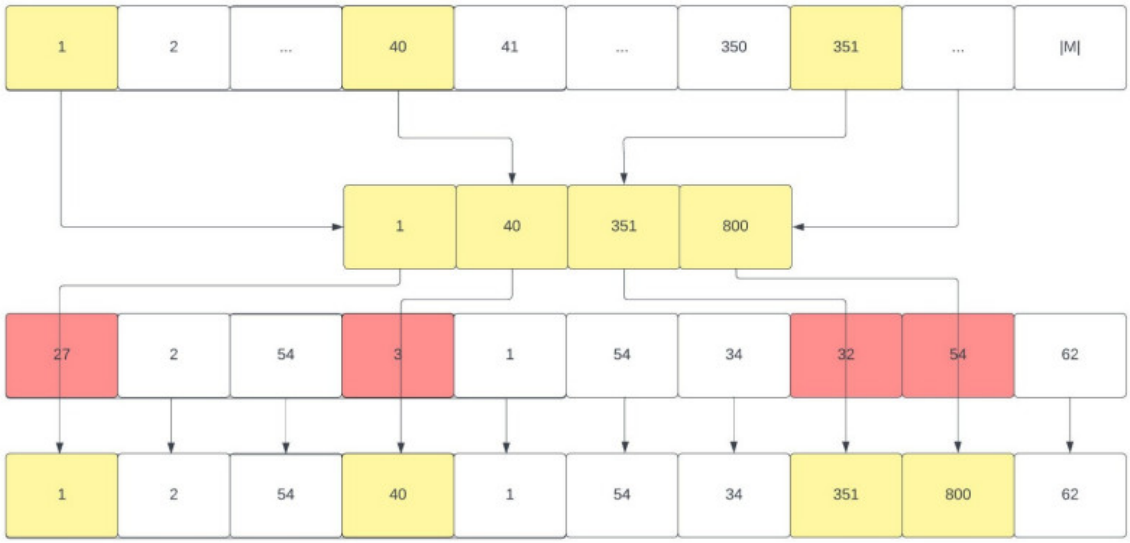
3 Mô phỏng quá trình thực hiện gây nhiễu

3.1 Sequence Order Randomization - Sequence Item Imputation

Đối với bài báo cáo này, chúng ta sẽ thực hiện việc gây nhiễu lên toàn bộ tập dữ liệu, bao gồm cả tập huấn luyện và tập kiểm tra. Các dữ liệu nhiễu này sẽ được áp dụng lên toàn bộ danh sách phim mà mỗi người dùng đã xem để mô phỏng các biến động có thể xảy ra trong thực tế. Trong thực tế, độ nhiễu không chỉ gây ảnh hưởng đến tập huấn luyện hoặc tập kiểm tra mà cả toàn bộ tập dữ liệu. Vì vậy, có thể có những chuỗi bị nhiễu loạn nhẹ và những chuỗi bị nhiễu loạn hoàn toàn. Do đó, ta sẽ khám phá hai loại nhiễu loạn đối với danh sách phim của người dùng: *Sequence Order Randomization* (Hoán vị thứ tự chuỗi) và *Sequence Item Imputation* (Bổ sung mục chuỗi).

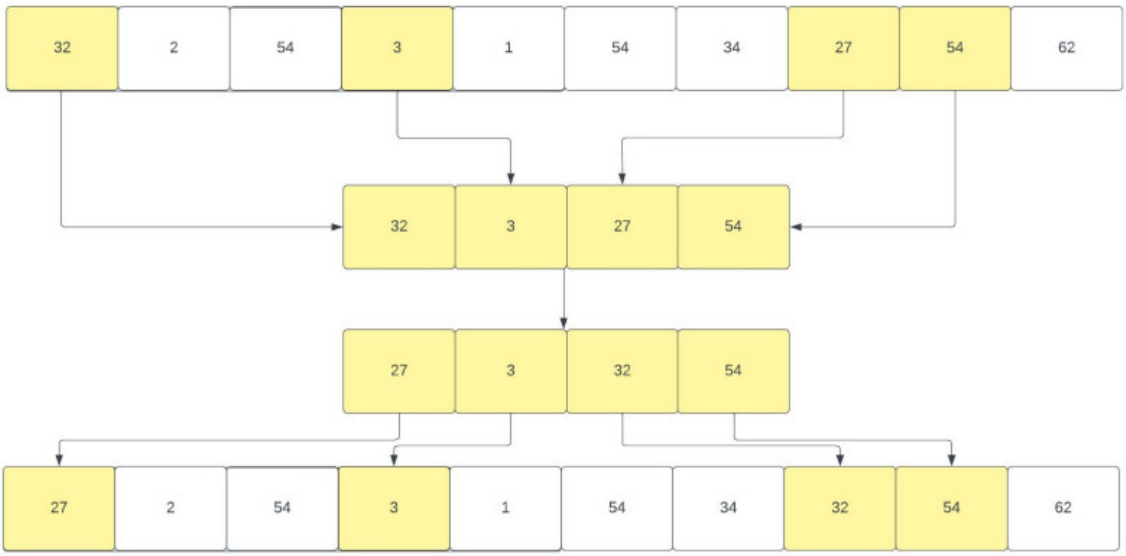
Với *Sequence Item Imputation* (Bổ sung mục chuỗi), minh họa ở Hình 6, ta sẽ chọn ngẫu nhiên một tỷ lệ các chỉ số và thay thế các phim từ một tập hợp các phim mà người dùng chưa xem. Điều này không có nghĩa là tất cả các chuỗi sẽ có cùng mức độ nhiễu. Khi gây nhiễu danh sách phim, một số

chuỗi phim có thể bị thay đổi nhiều hơn hoặc ít hơn so với các chuỗi khác. Điều này tạo ra mức độ nhiễu loạn khác nhau giữa các chuỗi phim. Trước khi chia danh sách phim thành các chuỗi phim, ta sẽ áp dụng nhiễu loạn lên toàn bộ danh sách phim, tức là việc nhiễu loạn không chỉ xảy ra trong các chuỗi riêng lẻ mà trên toàn bộ danh sách phim của người dùng và các đánh giá phim của người dùng sẽ được ngẫu nhiên hóa trong khoảng từ 1 đến 5.



Hình 6: Imputation Procedure

Đối với *Sequence Order Randomization* (Hoán vị thứ tự chuỗi), chọn ngẫu nhiên một tỷ lệ hoặc một số lượng các chỉ số để gây nhiễu, sau đó xáo trộn các giá trị tại các chỉ số này. Quy trình này được minh họa trong Hình 7, chọn bốn chỉ số ngẫu nhiên, xáo trộn các giá trị tại các chỉ số này và sắp xếp lại theo thứ tự mới.



Hình 7: Sequence Order Permutation

3.2 Minh họa bằng Python

3.2.1 Xây dựng hàm Swap Elements:

Hàm này sẽ thay thế các phần tử trong chuỗi bằng các giá trị ngẫu nhiên và điều chỉnh đánh giá của người dùng.

```
def swap_elements(x, t, ratings):
    new_x = x[:]
    new_ratings = ratings[:]
    for idx, value in zip(random.sample(range(len(x)), k=len(t)), t):
        new_x[idx] = value
        new_ratings[idx] = int(random.randint(1, 5))
    return new_x, new_ratings
```

Listing 1: Xây dựng hàm Swap Elements

3.2.2 Xây dựng hàm Random Inputation:

Hàm này sẽ chọn ngẫu nhiên một tỷ lệ các mục và thay thế chúng bằng các mục chưa xem từ một tập hợp các phim.

```
def random_inputation(df, col1, col2, val, percentage=True):
```

```
    inp_movie_ids = []
    inp_ratings = []
    for index, row in df.iterrows():
        user_movies_sequence = row[f'{col1}']
        ratings = row[f'{col2}']
        n = len(user_movies_sequence)

        n_chose = math.floor(n * val) if percentage else val
        movies_to_input = random.choices(row['other_movies'], k=n_chose)
        new_sequence, new_ratings = swap_elements(user_movies_sequence,
   movies_to_input, ratings)
        inp_movie_ids.append(new_sequence)
        inp_ratings.append(new_ratings)
    df['inp_movie_ids'] = inp_movie_ids
    df['inp_ratings'] = inp_ratings
    return df
```

Listing 2: Xây dựng hàm Random Inputation

3.2.3 Xây dựng hàm Shuffle Elements:

Hàm này sẽ xáo trộn các phần tử trong chuỗi để tạo nhiễu.

```
def shuffle(movie_sequence, ratings, count):
    indices_to_shuffle = random.sample(range(len(movie_sequence)), k=count)
    old_indices = indices_to_shuffle.copy()
    random.shuffle(indices_to_shuffle)
    new_movie_sequence = movie_sequence.copy()
    new_ratings_sequence = ratings.copy()
    for index, value in enumerate(indices_to_shuffle):
        old_index = old_indices[index]
        new_movie_sequence[old_index] = movie_sequence[value]
        new_ratings_sequence[old_index] = ratings[value]
    return new_movie_sequence, new_ratings_sequence
```

Listing 3: Xây dựng hàm Shuffle

3.2.4 Xây dựng hàm Random Shuffling:

Hàm này áp dụng việc xáo trộn lên các phần tử trong chuỗi và đánh giá.

```
def random_shuffling(df, col1, col2, val, percentage=True):
    new_sequences = []
    new_rating_lists = []
    for index, row in df.iterrows():
        user_movies_sequence = row[f'{col1}']
        ratings = row[f'{col2}']
        n = len(user_movies_sequence)
        if percentage:
            n_shuffle = math.floor(n * val)
        else:
            n_shuffle = val
        new_seq, new_ratings = shuffle(user_movies_sequence, ratings, n_shuffle)
        new_sequences.append(new_seq)
        new_rating_lists.append(new_ratings)
    df['random_movie_ids'] = new_sequences
    df['random_ratings'] = new_rating_lists
    return df
```

Listing 4: Xây dựng hàm Random Shuffling

3.2.5 Xây dựng hàm Corruption Results:

Hàm này tạo các chuỗi và áp dụng việc gây nhiễu để đánh giá kết quả.

```
def corruption_results(df, col1):
    df[f'{col1}'] = df[f'{col1}'].apply(lambda x: literal_eval(x))
    df['movie_ids'] = df['movie_ids'].apply(lambda x: literal_eval(x))  #

    sequence_length = 8
    step_size = 1
    random.seed(0)
    df[f'{col1}'] = df[f'{col1}'].apply(
        lambda ids: create_sequences(ids, sequence_length, step_size))

    random.seed(0)
```

```
    df.movie_ids = df.movie_ids.apply(
        lambda ids: create_sequences(ids, sequence_length, step_size))

    del df["timestamps"]
    df_p = df[["user_id", f"{col1}"]].explode(
        f"{col1}", ignore_index=True)
    df_t = df[["movie_ids"]].explode(
        "movie_ids", ignore_index=True)
    df_final = pd.concat([df_p, df_t], axis=1)
    df_final = measure_corruption(df_final, col1, 'movie_ids')
    return df_final
```

Listing 5: Xây dựng hàm Corruption Results

3.2.6 Gọi hàm Corruption Results cho các bộ dữ liệu đầu vào:

Áp dụng hàm Corruption Results cho các bộ dữ liệu với các mức độ gây nhiễu khác nhau.

```
res_inp_10 = corruption_results(inputed_ratings_data_10, 'inp_movie_ids')
res_inp_20 = corruption_results(inputed_ratings_data_20, 'inp_movie_ids')
res_inp_30 = corruption_results(inputed_ratings_data_30, 'inp_movie_ids')
res_inp_40 = corruption_results(inputed_ratings_data_40, 'inp_movie_ids')
res_inp_50 = corruption_results(inputed_ratings_data_50, 'inp_movie_ids')
res_ran_10 = corruption_results(random_ratings_data_10, 'random_movie_ids')
res_ran_20 = corruption_results(random_ratings_data_20, 'random_movie_ids')
res_ran_30 = corruption_results(random_ratings_data_30, 'random_movie_ids')
res_ran_40 = corruption_results(random_ratings_data_40, 'random_movie_ids')
res_ran_50 = corruption_results(random_ratings_data_50, 'random_movie_ids')
```

Listing 6: Gọi hàm Corruption Results

```
results_corruption = pd.DataFrame.from_dict({
    "Perturbation": ["Inputed 10%",
                     "Inputed 20%",
                     "Inputed 30%",
                     "Inputed 40%",
                     "Inputed 50%",
                     "Order 10%",
                     "Order 20%",
```

```
                    "Order 30%",
                    "Order 40%",
                    "Order 50%"],
    "Average Corruption": [np.mean(res_inp_10.percentage_change),
                           np.mean(res_inp_20.percentage_change),
                           np.mean(res_inp_30.percentage_change),
                           np.mean(res_inp_40.percentage_change),
                           np.mean(res_inp_50.percentage_change),
                           np.mean(res_ran_10.percentage_change),
                           np.mean(res_ran_20.percentage_change),
                           np.mean(res_ran_30.percentage_change),
                           np.mean(res_ran_40.percentage_change),
                           np.mean(res_ran_50.percentage_change)]
})
results_corruption
```

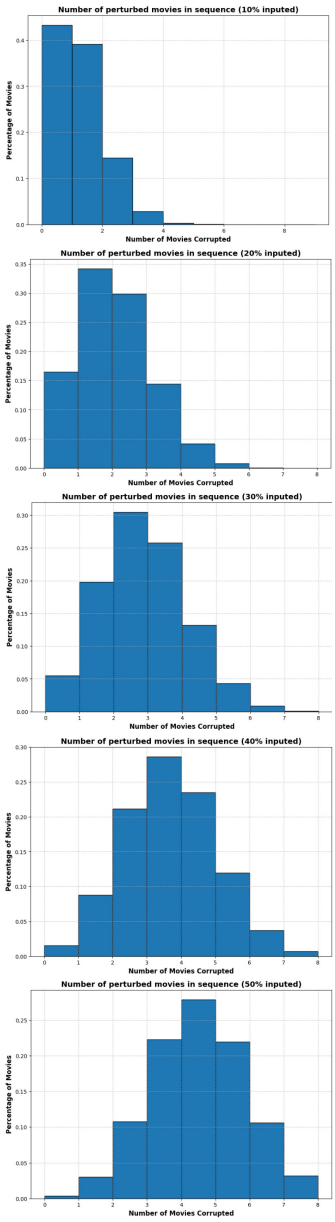
Listing 7: Hiển thị kết quả

Bảng 1 cho thấy mức độ nhiễu trung bình cho tất cả các chuỗi có độ chính xác gần với mức độ nhiễu trên toàn bộ danh sách theo người dùng. Điều này có nghĩa là độ nhiễu loạn được áp dụng nhất quán và chính xác trong các ví dụ. Tuy nhiên, điều quan trọng cần lưu ý là mức độ nhiễu trung bình được đưa ra trong bảng đại diện cho mức tối đa mà các chuỗi dữ liệu có thể bị nhiễu loạn, nhưng trong thực tế có thể thấp hơn, khi áp dụng các phương pháp này vào các tình huống khác nhau mức độ nhiễu thực tế có thể thấp hơn do nhiều yếu tố như cách thức người dùng tương tác, môi trường sử dụng và biện pháp giảm thiểu bổ sung. Nhìn chung, các kết quả này cung cấp cho ta rõ ràng về tác động của các mức độ nhiễu khác nhau và giúp chúng ta định hình được chiến lược cải thiện hiệu suất hệ thống.

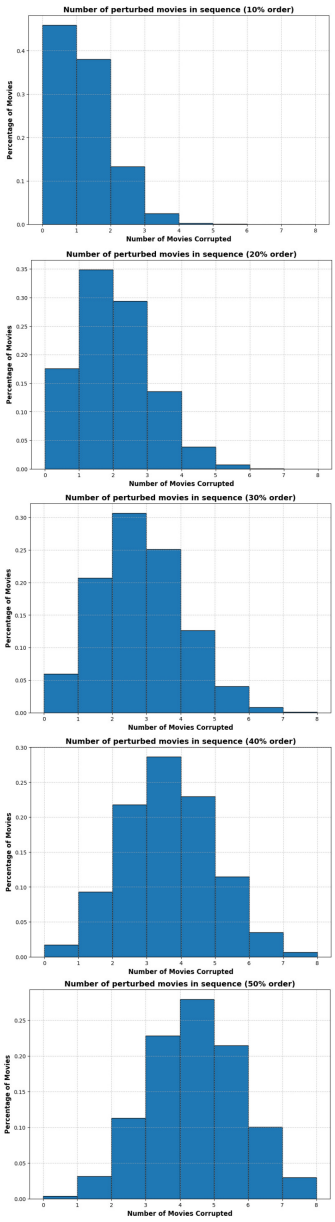
Perturbation	**Average Corruption**
Inputed 10 %	9.750249
Inputed 20%	19.776712
Inputed 30%	29.759100
Inputed 40%	39.771650
Inputed 50%	49.857555
Order 10%	9.170873
Order 20%	19.208164
Order 30%	29.192588
Order 40%	39.204658
Order 50%	49.294998

Bảng 1: Average Sequence Corruption by Perturbation Scheme

Ta trực quan hóa các biểu đồ histogram để thể hiện sự phân bố tỷ lệ phần trăm các chuỗi phim bị nhiễu loạn trong chuỗi. Áp dụng các inputations và orders khác nhau để kiểm tra và đánh giá độ bền vững, tính mạnh mẽ và khả năng xử lý dữ liệu nhiễu của các mô hình dự đoán. Qua đó, giúp cải thiện và tối ưu hóa mô hình để ứng dụng hiệu quả trong các điều kiện dữ liệu thực tế không hoàn hảo.



Hình 8: Sequence Imputation



Hình 9: Sequence Order Perturbation

Sự thay đổi trong phân bố dữ liệu khi tỷ lệ inputation và pertubation tăng lên, số lượng phim bị ảnh hưởng nhiễu trong các chuỗi cũng tăng lên, chuyển dịch đỉnh phân bố từ 0-1 phim (với inputation và pertubation 10%) lên 4-5 phim (với inputation và pertubation 50%). Inputation tăng dần làm tăng mức độ nhiễu loạn , cho thấy rõ ràng sự ảnh hưởng của việc áp dụng nhiễu loạn lên dữ liệu. Các kết quả này cung cấp cơ sở để đánh giá khả năng xử lý của các mô hình trong điều kiện dữ liệu bị nhiễu loạn , giúp cải thiện tính mạnh mẽ và độ tin cậy của chúng.

Các biểu đồ này đã cung cấp một cái nhìn sâu sắc về sự phân bố của nhiễu loạn trong các chuỗi

dữ liệu. Việc có những chuỗi bị nhiễu loạn cao ngay cả ở mức độ nhiễu loạn thấp cho thấy rằng ảnh hưởng của nhiễu loạn không đồng đều và có thể dẫn đến các kết quả khác nhau trong các thử nghiệm hoặc ứng dụng thực tế. Điều này sẽ rất quan trọng để hiểu và quản lý khi thiết kế các hệ thống và thuật toán để đảm bảo hiệu suất ổn định và đáng tin cậy.

4 Mô hình đề xuất embedding chuỗi tích chập (CASER - CNN)

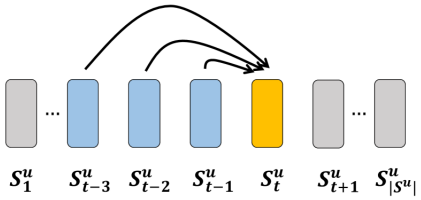
Tùy thuộc vào các ứng dụng cụ thể hoặc tình huống mà thứ tự của các chuỗi có thể được coi là quan trọng. Một ví dụ cho rằng thứ tự của các sự kiện trong quá khứ có thể hoặc không quan trọng đối với nhiệm vụ đề xuất và điều này phụ thuộc vào ngữ cảnh và tập dữ liệu. Thứ tự có thể quan trọng do các ràng buộc ngầm định, chẳng hạn như ta có thể đề xuất *Season 2* của một series phim sau khi *Season 1* kết thúc, nhưng ta hoàn toàn không thể làm ngược lại. Mặc dù nhiều mô hình nhận thức thời gian/chuỗi đã chứng minh được hiệu suất tốt hơn trong những tình huống như thế này, nhưng vẫn chưa có sự đồng thuận về cách lý tưởng để chứng mình rằng các hệ thống đề xuất này đang tận dụng tính chất tuần tự của dữ liệu để cải thiện hiệu suất.

Các hệ thống đề xuất tuần tự chỉ quan trọng khi tập dữ liệu bao gồm các mẫu tuần tự, điều này có thể được đo lường bằng cường độ tuần tự của một tập dữ liệu như trong *Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding*. Sự thay đổi chuỗi gần đây đã trở thành cách đánh giá độ nhạy của các hệ thống đề xuất. Để đánh giá sự thay đổi chuỗi của các hệ thống đề xuất chủ yếu sử dụng *LOO* (leave one out) để đánh giá ảnh hưởng của sự thay đổi lên các hệ thống đề xuất. Tuy nhiên trong bài báo *Rank List Sensitivity of Recommender Systems to Interaction Perturbations[Oh et al., 2022]* còn thêm vào các phương pháp thay thế và bổ sung để đánh giá ảnh hưởng của nhiều thay đổi khác nhau lên độ nhạy của các mô hình này. Bài báo này đã khám phá cách mà các thay đổi nhỏ trong dữ liệu huấn luyện của một người dùng có thể thay đổi dự đoán trong tương tác của những người dùng khác. Điều này sẽ gây ra nhiều vấn đề nghiêm trọng khi hệ thống đề xuất được sử dụng trong các lĩnh vực nhạy cảm hơn, vì những thay đổi dữ liệu của một người dùng có thể làm thay đổi đề xuất của những người dùng khác, đó là lý do tại sao chúng ta cần khám phá mức độ nhạy cảm của các hệ thống đề xuất dựa trên tuần tự này.

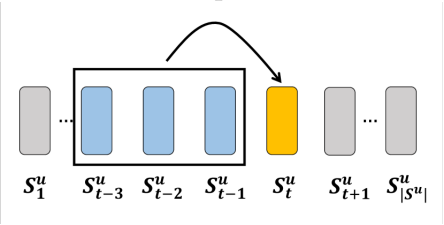
Các cách tiếp cận khác nhau đã được sử dụng để nắm bắt tính tuần tự của các vấn đề này. Một trong những hệ thống đề xuất nhận thức tuần tự có thể được thấy trong bài báo *Behavior Sequence Transformer for E-commerce Recommendation in Alibaba*, cho thấy rằng các hệ thống đề xuất hiên hiện trong doanh nghiệp thường đưa các đặc trưng vào các vector có số chiều thấp và sau đó đưa những đặc trưng này qua một Multi-Layer Perceptron (MLP) (Mạng nơ ron nhiều tầng) để đưa ra các đề xuất cuối cùng. Các mô hình khác như CASER (CNN) cũng đã được sử dụng như một mô hình nhận thức chuỗi, mô hình học các đặc trưng tuần tự sẽ thực hiện tốt hơn trong các nhiệm vụ mà các sản phẩm ngẫu nhiên được giới thiệu trong chuỗi người dùng.

4.1 Top-N Sequential Recommendation

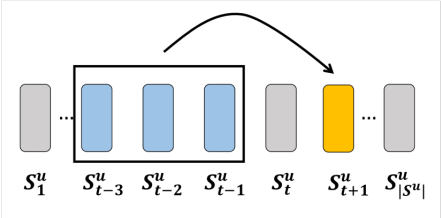
Để mô hình hóa các hành vi tuần tự của người dùng, nghiên cứu của (*Rendle et al., 2010; Liu et al., 2009*) đã xem xét đề xuất tuần tự Top-N, đề xuất N mặt hàng mà người dùng có khả năng tương tác trong tương lai gần. Giả định một tập hợp người dùng $\mathcal{U} = \{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$ và một tập hợp các mặt hàng $\mathcal{I} = \{i_1, i_2, \ldots, i_{|\mathcal{I}|}\}$. Mỗi người dùng u được gắn với một chuỗi các mặt hàng $S_u = (S_{1_u}, \ldots, S_{|S_u|_u})$ trong đó $S_{t_u} \in \mathcal{I}$ chỉ số t cho S_{t_u} biểu thị thứ tự hành động xảy ra trong chuỗi S_u. Dựa trên tất cả các chuỗi của người dùng S_u, mục tiêu là đề xuất cho mỗi người dùng một danh sách các mặt hàng tối đa hóa nhu cầu tương lai của họ, bằng cách xem xét cả sở thích chung và các mẫu hành vi tuần tự của họ. Không giống như đề xuất Top-N thông thường, đề xuất tuần tự Top-N mô hình hóa hành vi của người dùng như một chuỗi các mặt hàng thay vì một tập hợp các mặt hàng. Một ví dụ về ảnh hưởng của các mô hình động ở mức điểm và mức liên hợp, với thứ tự của chuỗi Markov là $L = 3$.



Hình 10: point-level



Hình 11: union-level, no skip



Hình 12: union-level, skip once

4.2 Sự giới hạn của các nghiên cứu trước

Mô hình dựa trên chuỗi Markov (*Rendle et al., 2010; He and McAuley, 2016; Cheng et al., 2013; Wang et al., 2015a*) là phương pháp tiếp cận sớm đối với đề xuất tuần tự Top-N, trong đó một chuỗi Markov bậc L thực hiện các đề xuất dựa trên L hành động trước đó. Chuỗi Markov đầu tiên là một ma trận item-to-item sử dụng Maximum Likelihood Estimation. Factorized Personalized Markov Chains (FPMC) (*Rendle et al., 2010*) được đề xuất bởi Rendle et al. và một biến thể khác (Cheng et al., 2013) cải tiến phương pháp này bằng cách tách ma trận chuyển tiếp này thành hai ma trận con ẩn và có hạng thấp. Tuy nhiên, các phương pháp này gặp phải hai hạn chế chính:

Không mô hình hóa được các mẫu tuần tự mức liên kết:

Như được minh họa trong Hình 10, chuỗi Markov chỉ mô hình hóa các mẫu tuần tự **point-level**, trong đó mỗi hành động trước đó (màu xanh) ảnh hưởng đến hành động mục tiêu (màu vàng) một cách riêng lẻ, thay vì một cách tổng thể (FPMC thuộc trường hợp này). Ảnh hưởng của **point-level** không đủ để mô hình hóa các ảnh hưởng **union-leve**l như trong Hình 11, trong một số hành động

trước đó, theo thứ tự đó và cùng ảnh hưởng đến hành động mục tiêu. Ta có thể cho một ví dụ như, việc mua cả sữa và bơ cùng nhau dẫn đến khả năng cao mua bột mì nhiều hơn là mua sữa hoặc bơ riêng lẻ; mua cả RAM và ổ cứng sau đó mua hệ điều hành, tốt hơn là chỉ mua một trong các thành phần rồi mua hệ điều hành.

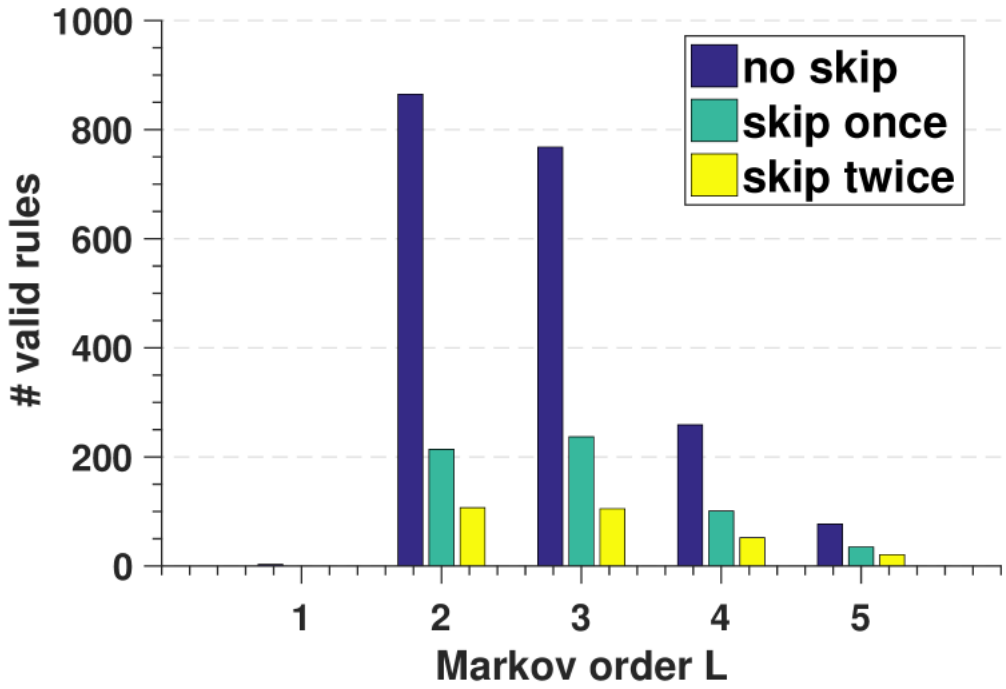
Không cho phép bỏ qua các hành vi:

Các mô hình hiện tại không xem xét hành vi bỏ qua của các mẫu tuần tự như được minh họa trong Hình 12, trong đó ảnh hưởng từ các hành vi trong quá khứ có thể được bỏ qua một vài bước nhưng vẫn mang lại hiệu quả ổn định. Ví dụ, một du khách check-in lần lượt tại sân bay, khách sạn, nhà hàng, quán bar và điểm tham quan. Trong khi check-in tại sân bay và khách sạn không ngay lập tức xảy ra trước check-in tại điểm tham quan, vì chúng có liên kết mạnh mẽ với nhau. Mặt khác, check-in tại nhà hàng hoặc quán bar ít ảnh hưởng đến check-in tại điểm tham quan (vì chúng không nhất thiết xảy ra). Một chuỗi Markov bậc L không mô hình hóa rõ ràng các hành vi bỏ qua này vì nó giả định rằng các bước ở trước có ảnh hưởng ngay lập tức đến bước tiếp theo.

Để cung cấp thêm các dẫn chứng về các ảnh hưởng mức liên kết và hành vi bỏ qua, chúng ta khai thác các quy tắc liên kết tuần tự (*Agrawal and Srikant, 1995; Han et al., 2011*) dưới hình thức sau từ bộ dữ liệu thực tế MovieLens.

$$(S^u_{t-L}, \ldots, S^u_{t-2}, S^u_{t-1}) \rightarrow S^u_t$$

Đối với quy tắc $X \rightarrow Y$ theo hình thức trên, số lần hỗ trợ $\sup(XY)$ là số lần mà X và Y xảy ra theo thứ tự như trong quy tắc, và độ tin cậy $\frac{\sup(XY)}{\sup(X)}$ là tỷ lệ phần trăm các chuỗi trong đó X kéo theo Y trong những chuỗi mà X xảy ra. Quy tắc này đại diện cho ảnh hưởng chung của tất cả các mặt hàng trong X đối với Y. Bằng cách thay đổi bên phải thành S^u_{t+1} hoặc S^u_{t+2}, quy tắc cũng cho thấy ảnh hưởng với một hoặc hai bước bỏ qua. Hình 13 tóm tắt số quy tắc tìm thấy so với *minimun support = 5* và *minimum confidence = 50%* (với các độ tin cậy tối thiểu như 10%, 20% và 30%, các xu hướng này cũng có kết quả tương tự). Hầu hết các quy tắc có thứ tự *L=2* và *L=3* và độ tin cậy của quy tắc càng cao khi thứ tự L càng lớn.



Hình 13: The number of association rules vs L and skip steps. The *minimum support count = 5* and the *minimum confidence = 50%*.

4.3 Giải quyết vấn đề

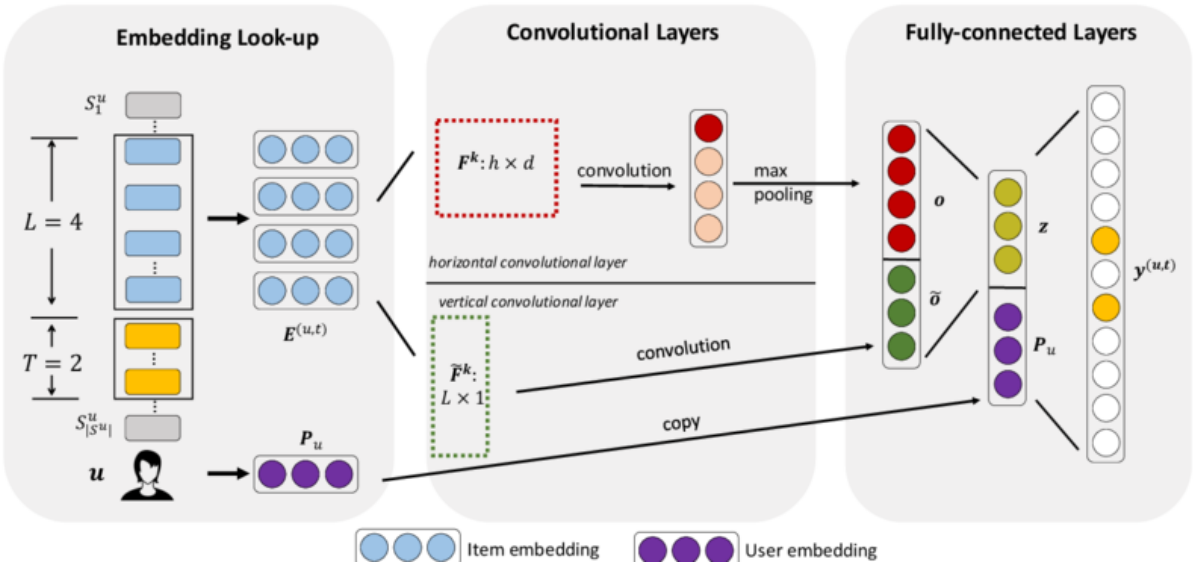
Để giải quyết những hạn chế trên của các công trình hiện có, chúng ta sẽ đề cập đến một mô hình đề xuất có tính đến các nhật ký tương tác của người dùng theo tuần tự thời gian. Đây là một hệ thống đề xuất nhận thức theo chuỗi (*Quadrana et al., 2018*) trong đó đầu vào là môt danh sách có thứ tự và thường được đánh dấu thời gian của các hành động người dùng trong quá khứ. Một số tài liệu gần đây đã chứng minh tính hữu ích của việc kết hợp thông tin này trong việc mô hình hóa các mô hình hành vi theo thời gian của người dùng và khám phá sự thay đổi sở thích của họ.

Mô hình mà chúng ta đề cập đến là CASER (*Tang và Wang, 2018*), viết tắt của Convolutional Sequence Embedding Recommendation Model, sử dụng mạng nơ ron tính chập để nắm bắt các ảnh hưởng động của các hoạt động gần đây của người dùng. Mô hình này tận dụng thành công gần đây của các bộ lọc tích chập của mạng tích chập (CNN) để nhận diện các đặc trưng cục bộ cho nhận diện hình ảnh (*Krizhevsky et al., 2012; Karpathy et al., 2014*) và xử lý ngôn ngữ tự nhiên (*Kim, 2014*). Điểm mới của CASER là biểu diễn các mặt hàng trước đó.

Thành phần chính của CASER bao gồm Tra Cứu Nhúng (*Embedding Look-up*), Các lớp Tích chập (*Convolutional Layer*) và Các lớp Kết nối đầy đủ (*Fully-connected Layers*).

So với các phương pháp hiện tại, CASER cung cấp một số ưu điểm rõ rệt. (1) CASER sử dụng các bộ lọc tích chập ngang và dọc để nhận diện các mẫu hành vi tuần tự ở mức điểm, mức liên kết và hành vi bỏ qua. (2) CASER mô hình hóa cả sở thích chung của người dùng và các mẫu hành vi tuần tự, và tổng quát hóa một số phương pháp hiện đại hiện tại trong một khung đơn nhất. (3) CASER vượt trội so với các phương pháp hiện đại cho đề xuất tuần tự Top-N trên các bộ dữ liệu thực tế.

Trong hệ thống đề xuất nhận thức theo chuỗi, mỗi người dùng được liên kết với một chuỗi các mục từ tập mục. Giả sử $\mathcal{U} = \{u_1, u_2, \ldots, u_{|\mathcal{U}|}\}$ là chuỗi có thứ tự của người dùng u. Mục tiêu của CASER là đề xuất các mục dựa trên việc xem xét sở thích chung (dài hạn) và ý định ngắn hạn của người dùng.



Hình 14: CASER Architecture

4.3.1 Tra cứu Nhúng (Embedding Look-up)

Trích xuất và biểu diễn các mục trong chuỗi người dùng dưới dạng các vector nhúng. Để mô hình hóa hành vi ngắn hạn của người dùng, chúng ta lấy L mục trước đó làm đầu vào và T mục tiếp theo làm mục tiêu từ chuỗi S_{t_u} của người dùng. Một ma trận nhúng đại diện cho các tương tác đó

tại thời điểm t có thể được xây dựng như sau:

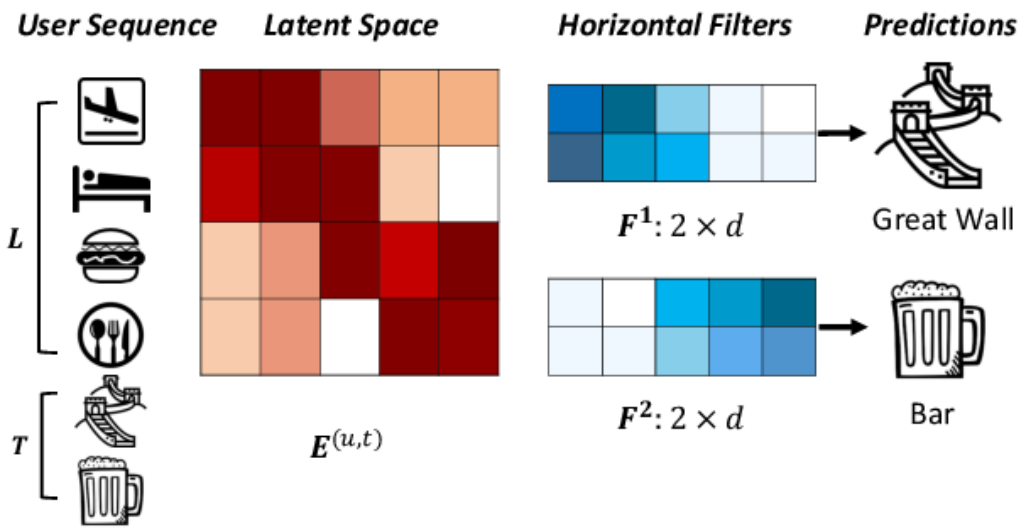
$$E^{(s,t)} = \begin{bmatrix} Q_s^{t-L} \\ \vdots \\ Q_s^{t-2} \\ Q_s^{t-1} \end{bmatrix}$$

- $Q \in \mathbb{R}^{n \times k}$: ma trận những các mục, mỗi mục được biểu diễn bằng một vector những k-chiều.
- q_i: hàng thứ i của ma trận Q.
- $E^{(u,t)} \in \mathbb{R}^{L \times k}$: ma trận những dải diễn cho L mục trước đó của người dùng u tại thời điểm t.

Chúng ta có thể xem ma trận đầu vào $E^{(u,t)}$ như một hình ảnh, là đầu vào của hai thành phần tích chập tiếp theo.

4.3.2 Các lớp Tích chập (Convolutional Layers):

Sử dụng các bộ lọc ngang và dọc để phát hiện các mẫu tuần tự trong chuỗi. Chúng ta sẽ tận dụng sự thành công của bộ lọc tích chập CNN trong việc nắm bắt các đặc trưng cục bộ cho nhận dạng hình ảnh (*Krizhevsky et al., 2012; Karpathy et al., 2014*) và xử lý ngôn ngữ tự nhiên (*Kim, 2014*). Ta sẽ xem ma trận E kích thước $L \times d$ là "hình ảnh" của L mục trước đó trong không gian ẩn và xem các mẫu tuần tự là các đặc trưng cục bộ của "hình ảnh" này. Cách tiếp cận này sẽ cho phép chúng ta sử dụng các bộ lọc tích chập để tìm kiếm các mẫu tuần tự. Hình 15 cho thấy hai "bộ lọc ngang" nắm bắt hai mẫu tuần tự *union-level*. Các bộ lọc này, được biểu diễn dưới dạng ma trận $h \times d$, có chiều cao $h = 2$ và chiều rộng đầy đủ bằng d. Chúng lấy tín hiệu cho các mẫu tuần tự bằng cách trượt qua các hàng của E. Ví dụ, bộ lọc đầu tiên nắm bắt mẫu tuần tự "(sân bay, khách sạn) → Vạn Lý Trường Thành" bằng cách có các giá trị lớn hơn trong các chiều ẩn nơi Sân bay và Khách sạn có giá trị lớn hơn. Tương tự, một "bộ lọc dọc" là một ma trận $L \times 1$ và sẽ trượt qua các cột của E. Chi tiết hơn sẽ được giải thích bên dưới.



Hình 15: Convolutional Filtering

1. **Lớp tích chập Ngang (Horizontal Convolutional Layer)**

   Lớp này, được hiển thị ở phần trên của thành phần thứ hai trong Hình 14, có n bộ lọc ngang $F^k \in \mathbb{R}^{h \times d}, \quad 1 \leq k \leq n, \quad h \in \{1, \ldots, L\}$ là chiều cao của một bộ lọc. Ví dụ, nếu $L = 4$, có thể chọn có $n = 8$ bộ lọc, hai cho mỗi bộ trong $\{1, 2, 3, 4\}$. F^k sẽ trượt từ trên xống dưới trên E và tương tác với tất cả các chiều ngang của E của các mục $i, \quad 1 \leq i \leq L - h + 1$. Kết quả của sự tương tác này là giá trị tích chập thứ i được cho bởi:

$$c_i^k = \phi_c \left(E_{i:i+h-1} \odot F^k \right).$$

   trong đó, ký hiệu $\odot$ biểu thị toán tử tích trong và $\phi_c(.)$ là hàm kích hoạt cho các lớp tích chập. Giá trị này là tích trong giữa F^k và ma trận con được tạo bởi hàng i đến hàng $i - h + 1$ của E, được ký hiệu bởi $E_{i:i+h-1}$. Kết quả tích chập cuối cùng của F^k là vector

$$c^k = \begin{bmatrix} c_1^k & c_2^k & \cdots & c_{L-h+1}^k \end{bmatrix}.$$

   Sau đó, chúng ta sẽ áp dụng một phép pooling max cho c^k để trích xuất giá trị tối đa từ tất cả các giá trị được tạo ra bởi bộ lọc này. Giá trị tối đa nắm bắt đặc trưng quan trọng nhất được trích xuất bởi bộ lọc. Do đó, đối với n bộ lọc trong lớp này, giá trị đầu ra $\mathbf{o} \in \mathbb{R}^n$ là

$$\mathbf{o} = \{\max(c^1), \max(c^2), \ldots, \max(c^n)\}.$$

Các bộ lọc ngang tương tác với mỗi h mục kế tiếp thông qua các nhúng E. Cả các nhúng và các bộ lọc đều được học để giảm thiểu một hàm mục tiêu mã hóa lỗi dự đoán của các mục mục tiêu. Bằng cách trượt các bộ lọc có các chiều cao khác nhau, một tín hiệu đáng kể sẽ được nắm bắt bất kể vị trí của nó. Do đó, các bộ lọc ngang có thể được huấn luyện để nắm bắt các mẫu *union-level* với nhiều kích thước liên kết.

2. **Lớp tích chập Dọc (Vertical Convolutional Layer)**
   Lớp này, được hiển thị ở phần dưới của thành phần thứ hai trong Hình 14. Chúng ta sẽ sử dụng dấu ngã (˜) cho các ký hiệu của lớp này. Giả sử ta có $\tilde{n}$ bộ lọc dọc $\tilde{F}^k \in \mathbb{R}^{L\times 1}, \quad 1 \leq k \leq \tilde{n}$. Mỗi bộ lọc $\tilde{F}^k$ tương tác với các cột của E bằng cách trượt d lần từ trái sang phải trên E, tạo ra kết quả tích chập dọc $\tilde{c}^k$:

$$\tilde{c}^k = \begin{bmatrix} \tilde{c}_1^k & \tilde{c}_2^k & \cdots & \tilde{c}_d^k \end{bmatrix}.$$

Đối với tương tác tích trong, dễ dàng kiểm chứng rằng kết quả này bằng tổng có trọng số qua các hàng của E với $\tilde{F}^k$ làm trọng số:

$$\tilde{c}^k = \sum_{l=1}^{L} \tilde{F}_l^k \cdot E_l$$

trong đó, E_l là hàng thứ l của E. Do đó với các bộ lọc dọc, chúng ta có thể học cách tổng hợp các nhúng của L mục trước đó. Do việc sử dụng chúng là tổng hợp, các bộ lọc dọc có một số khác biệt so với các bộ lọc ngang: (1) Kích thước của mỗi bộ lọc dọc được cố định là $L \times 1$. Điều này là do mỗi cột của E là tiềm ẩn, không có ý nghĩa để tương tác với nhiều cột liên tiếp cùng một lúc.(2) Không cần phải áp dụng phép pooling max qua các kết quả tích chập dọc, vì ta muốn giữ lại tổng hợp cho mỗi chiều tiềm ẩn. Do đó, đầu ra của lớp này là $\tilde{o}$.

4.3.3 Các lớp Kết nối đầy đủ (Fully-connected Layers)

Kết hợp các đặc trưng từ các lớp tích chập và nhúng người dùng để đưa ra dự đoán cuối cùng về mục tiêu tiếp theo. Kết quả của hai lớp tích chập này được nối lại và đưa chúng vào một lớp mạng nơ-ron kết nối đầy đủ để có được các tính năng cao cấp và trừu tượng hơn.

$$z = \phi_a \left(W \begin{bmatrix} \mathbf{o} \\ \tilde{\mathbf{o}} \end{bmatrix} + b \right)$$

trong đó $W \in \mathbb{R}^{d\times(n+\tilde{n})}$ là ma trận trọng số chiếu lớp nối thành một lớp ẩn có chiều d, $b \in \mathbb{R}^d$ là hạng mục lệnh tương ứng và $\phi_a(\cdot)$ là hàm kích hoạt cho lớp kết nối đầy đủ. $\mathbf{z} \in \mathbb{R}^d$ sẽ được gọi là *convolutional sequence embedding*, cái mà mã hóa tất cả các loại đăc trưng tuần tự của L mục trước đó.

Để nắm bắt sở thích chung của người dùng, chúng ta cũng tra cứu nhúng người dùng P_u và nối hai vector có d chiều z và P_u, cùng nhau và chiếu chúng thành một lớp đầu ra với $|L|$ nút, được viết như sau:

$$y^{(u,t)} = W' \begin{bmatrix} z \\ P_u \end{bmatrix} + b'$$

trong đó $b' \in \mathbb{R}^{|L|}$ và $W' \in \mathbb{R}^{|L|\times 2d}$ là bias và ma trận trọng số cho lớp đầu ra, tương ứng. Như được giải thích trong phần trước, giá trị $y_i^{(u,t)}$ trong lớp đầu ra liên quan đến xác suất về mức độ mà người dùng u sẽ tương tác với mục t tại thời điểm t. z nhằm nắm bắt các mẫu tuần tự ngắn hạn, trong trường hợp khi nhúng người dùng P_u để nắm bắt sở thích chung dài hạn của người dùng. Ở đây chúng ta đặt P_u trong lớp ẩn cuối cùng vì một số lý do: (1) Nó có khả năng tổng quát hóa các mô hình khác. (2) Chúng ta có thể huấn luyện trước các tham số của mô hình với các tham số của các mô hình tổng quát khác. Như được nêu trong (*He et al., 2017b*), việc huấn luyện trước như vậy rất quan trọng đối với hiệu suất của mô hình.

4.3.4 Network Training

Để thực hiện huấn luyện, chúng ta sẽ chuyển đổi các giá trị của lớp đầu ra $y^{(u,t)}$,thành xác suất bằng cách:

$$p(\xi_t^u|\xi_{t-1}^u, \xi_{t-2}^u, \ldots, \xi_{t-L}^u) = \sigma(y_{\xi_t^u}^{(u,t)})$$

trong đó, $\sigma(x) = \frac{1}{1+e^{-x}}$ là một hàm *sigmoid*. Ta đặt $C^u = \{L+1, L+2, \ldots, |S^u|\}$ à tập hợp các dấu thời gian mà chúng ta muốn thực hiện dự đóa cho người dùng u. Xác suất của tất cả các chuỗi trong tệp dữ liệu là:

$$p(\xi|\Theta) = \prod_{u,t\in C^u} \sigma(y_{\xi_t^u}^{(u,t)}) \prod_{j\neq\xi_t^u} (1 - \sigma(y_{\xi_t^u}^{(u,t)}))$$

Để nắm bắt thêm các hành vi bỏ qua, chúng ta có thể xem xét các mục tiêu tiếp theo T, $D_t^u = \{\xi_t^u, \xi_{t+1}^u, \ldots, \xi_{t+T}^u\}$ cùng một lúc bằng cách thay thế mục kế tiếp ξ_t^u trong phương trình trên bằng

D_t^u . Lấy logarit của khả năng xảy ra, chúng ta sẽ nhận được hàm mục tiêu còn được gọi là *binary cross-entropy*:

$$\ell = \sum_{u \in U^*} \sum_{t \in T^*} \sum_{i \in D_t^*} -\log(\sigma(y_i^{(u,t)})) + \sum_{j \neq i} -\log(1 - \sigma(y_j^{(u,t)}))$$

4.3.5 Các phương pháp đánh giá

Lưu ý: Đối với các chỉ số đánh giá sau, chúng ta sẽ giới hạn các dự đoán và khuyến nghị thực tế ở các mức $k = [5, 10, 15]$. Ta thực hiện điều này vì trong nhiều vấn đề của hệ thống đề xuất, người dùng thường dừng xem các đề xuất sau khi đã xem qua k mục đầu tiên trong danh sách. Chúng ta sẽ tính toán các chỉ số @k cho mỗi người dùng, sau đó tính trung bình các chỉ số này qua tất cả các người dùng, bởi vì mỗi người dùng chỉ có một chuỗi duy nhất nên để có được một chỉ số tổng quát, chúng ta cần phải tính giá trị trung bình của toàn bộ người dùng.

Với các hệ thống đề xuất nhận thức tuần sự về chuỗi và lợi ích của chúng trong các tình huống với sở thích ngắn hạn và các mẫu tuần tự dài hạn, chúng ta sẽ chọn các chỉ số phù hợp nhất để đo lường hiệu suất của các hệ thống. Precision và Recall là một lựa chọn tốt khi không quan tâm đến thứ tự của khuyến nghị. Mean Average Precision (MAP) hoặc NDCG được ưu tiên hơn khi quan tâm đến thứ tự của khuyến nghị. Đối với mô hình này, chúng ta chọn sử dụng Precision và Recall để cung cấp cái nhìn sâu sắc về tình huống khi thứ tự không quan trọng và NDCG để xem xét tình huống khi thứ tự là quan trọng.

Average per User Precision @k: Precision trả lời cho câu hỏi trong các trường hợp được dự báo là positive thì có bao nhiêu trường hợp là đúng? Và tất nhiên precision càng cao thì mô hình của chúng ta càng tốt trong việc phân loại hồ sơ BAD (BAD chính là nhóm positive). Công thức của Precision như sau:

$$\text{Precision} = \frac{\text{True Positive}}{\text{Total Predicted Positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Average per User Recall@k: Recall đo lường tỷ lệ dự báo chính xác các trường hợp positive trên toàn bộ các mẫu thuộc nhóm positive. Công thức của recall như sau:

$$\text{Recall} = \frac{\text{True Positive}}{\text{Total actual positive}} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Để đánh giá các mô hình đề xuất trên cả tập bị nhiễu và không bị nhiễu, chúng ta sẽ sử dụng **Average per User NDCG@k (Normalized Discounted Cumulative Gain)**. NDCG được tạo ra từ hai thành phần, thành phần thứ nhất là tổng các giá trị liên quan của tất cả các kết quả trong danh sách đề xuất:

$$\text{cumulative gain} = \sum_{j=1}^{n} \text{relevance}_i$$

Thành phần thứ hai là DCG(Discounted cumulative gain):

$$\text{Discounted cumulative gain} = \sum_{j=1}^{n} \frac{2^{\text{relevance}_j} - 1}{\log_2(i+1)}$$

Một vấn đề xảy ra với DCG khi muốn so sánh hiệu suất đề xuất từ một truy vấn này sang một truy vấn khác, chúng ta có thể chuẩn hóa DCG để có được NDCG, chỉ số chúng ta sử dụng để đánh giá mô hình. Ta sẽ lấy các khuyến nghị thực tế và tính DCG cho tập dữ liệu này, sau đó chia DCG dự đoán cho DCG thực tế để có được DCG chuẩn hóa. Chúng ta sẽ lấy chỉ số NDCG@k làm chỉ số cuối cùng. Hàm giảm được đặt là $D(r) = 0$ cho mọi $r > k$, có nghĩa là nếu khuyến nghị dự đoán không nằm trong k khuyến nghị thực tế đầu tiên thì hàm giảm có giá trị bằng không.

4.3.6 Xây dựng mô hình CASER

Mô hình này được sử dụng để cải thiện khả năng dự đoán và đề xuất phim dựa trên các lịch sử tương tác của người dùng, với việc áp dụng các kỹ thuật học sâu hiện đại.

Các bước thực hiện:

- Chuẩn bị dữ liệu
- Tiền xử lý dữ liệu

  - Chuẩn hóa ID người dùng và phim.
  - Chia tập dữ liệu thành tập train và test.

- Xây dựng mô hình

  - Định nghĩa một lớp mạng nơ-ron tích chập (CASER) để xây dựng mô hình hệ thống đề xuất, gồm các thành phần chính (Embedding Layers, Horizontal Convolutional Layers, Vertical Convolutional Layer, Fully Connected Layer, Activation Function and Dropout, Initialize Weights).

- Định nghĩa phương thức *forward* để xác định cách dữ liệu đi qua các lớp của mô hình: (Embedding Lookup, Horizontal Convolution and Pooling, Vertical Convolution and Pooling, Combine Horizontal and Vertical Features, Prediction).

- Huấn luyện mô hình
- Đánh giá mô hình

- Dự đoán cho tập dữ liệu test và tính toán các chỉ số đánh giá như Precision, Recall và NDCG.

```
class Caser(nn.Module):
    def __init__(self, num_users, num_items, model_args):
        super(Caser, self).__init__()
        self.args = model_args

        # init args
        L = self.args['L']
        dims = self.args['d']
        self.n_h = self.args['nh']
        self.n_v = self.args['nv']
        self.drop_ratio = self.args['drop']
        self.ac_conv = activation_getter[self.args['ac_conv']]
        self.ac_fc = activation_getter[self.args['ac_fc']]

        # user and item embeddings
        self.user_embeddings = nn.Embedding(num_users, dims)
        self.item_embeddings = nn.Embedding(num_items, dims)

        # vertical conv layer
        self.conv_v = nn.Conv2d(1, self.n_v, (L, 1))

        # horizontal conv layer
        lengths = [i + 1 for i in range(L)]
        self.conv_h = nn.ModuleList([nn.Conv2d(1, self.n_h, (i, dims)) for i in
    lengths])

        # fully-connected layer
        self.fc1_dim_v = self.n_v * dims
```

```
        self.fc1_dim_h = self.n_h * len(lengths)
        fc1_dim_in = self.fc1_dim_v + self.fc1_dim_h
        # W1, b1 can be encoded with nn.Linear
        self.fc1 = nn.Linear(fc1_dim_in, dims)
        # W2, b2 are encoded with nn.Embedding, as we don't need to compute scores for all items
        self.W2 = nn.Embedding(num_items, dims+dims)
        self.b2 = nn.Embedding(num_items, 1)

        # dropout
        self.dropout = nn.Dropout(self.drop_ratio)

        # weight initialization
        self.user_embeddings.weight.data.normal_(0, 1.0 / self.user_embeddings.embedding_dim)
        self.item_embeddings.weight.data.normal_(0, 1.0 / self.item_embeddings.embedding_dim)
        self.W2.weight.data.normal_(0, 1.0 / self.W2.embedding_dim)
        self.b2.weight.data.zero_()
        self.cache_x = None

    def forward(self, seq_var, user_var, item_var, for_pred=False):
        # Embedding Look-up
        item_embs = self.item_embeddings(seq_var).unsqueeze(1)  # use unsqueeze() to get 4-D
        user_emb = self.user_embeddings(user_var).squeeze(1)

        # Convolutional Layers
        out, out_h, out_v = None, None, None
        # vertical conv layer
        if self.n_v:
            out_v = self.conv_v(item_embs)
            out_v = out_v.view(-1, self.fc1_dim_v)  # prepare for fully connect

        # horizontal conv layer
        out_hs = list()
        if self.n_h:
```

```
            for conv in self.conv_h:
                conv_out = self.ac_conv(conv(item_embs).squeeze(3))
                pool_out = F.max_pool1d(conv_out, conv_out.size(2)).squeeze(2)
                out_hs.append(pool_out)
            out_h = torch.cat(out_hs, 1)  # prepare for fully connect

        # Fully-connected Layers
        out = torch.cat([out_v, out_h], 1)
        # apply dropout
        out = self.dropout(out)

        # fully-connected layer
        z = self.ac_fc(self.fc1(out))
        x = torch.cat([z, user_emb], 1)

        w2 = self.W2(item_var)
        b2 = self.b2(item_var)

        if for_pred:
            w2 = w2.squeeze()
            b2 = b2.squeeze()
            res = (x * w2).sum(1) + b2
        else:
            res = torch.baddbmm(b2, w2, x.unsqueeze(2)).squeeze()

        return res
```

Listing 8: Xây dựng Class CASER

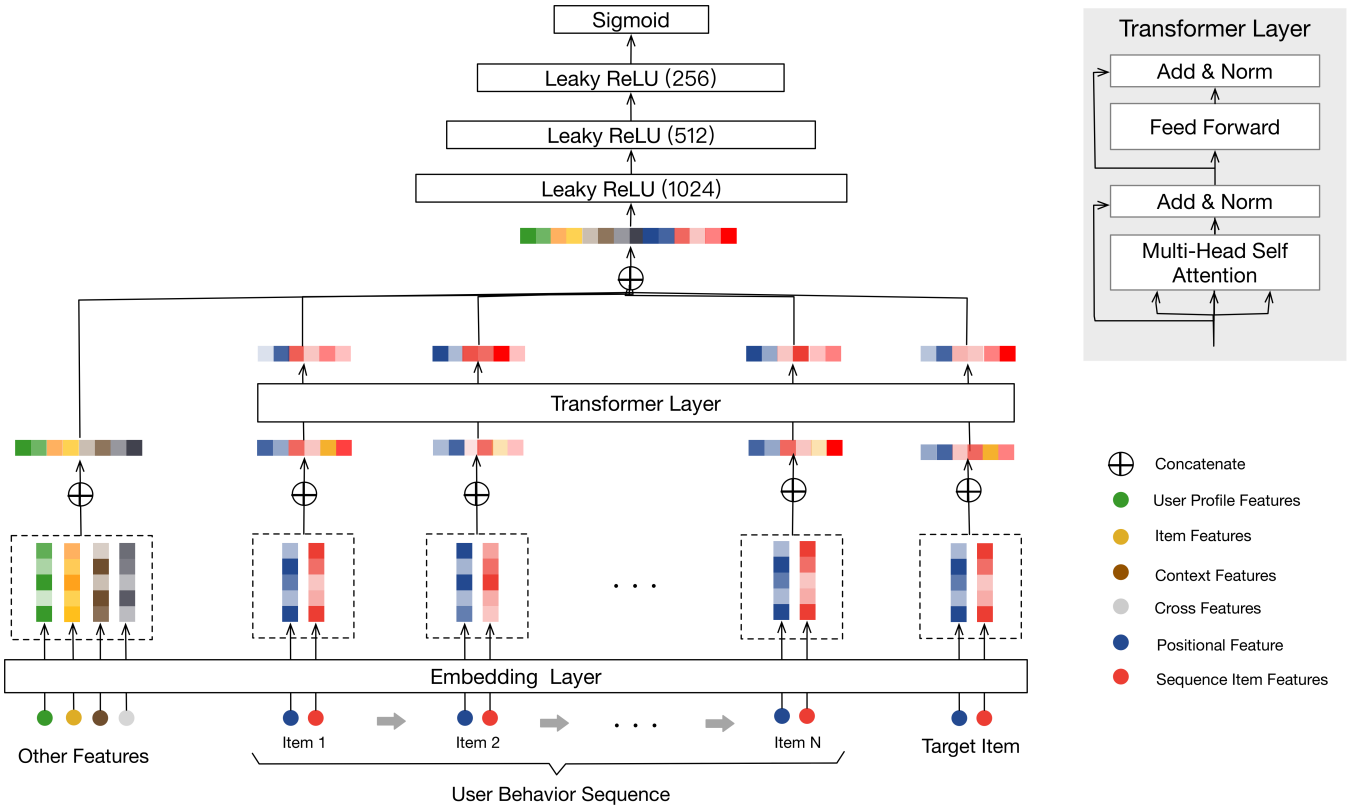
5 Mô hình đề xuất dựa trên thuật toán Behavior Sequence Transformer (BST)

5.1 Mô hình Behavior Sequence Transformer (BST)

Hệ thống đề xuất dựa trên thuật toán Behavior Sequence Transformer hay BST, được triển khai trực tiếp từ bài báo Sequence-aware recommender systems của Massimo Quadrana, Paolo Cremonesi và

Dietmar Jannach. Trong lĩnh vực học sâu, việc sử dụng các vector nhúng (embeddings) và MLP đã trở thành tiêu chuẩn cho các hệ thống đề xuất. Tuy nhiên, các hệ thống đề xuất này thường không tính đến tín hiệu tuần tự trong chuỗi hành vi của người dùng. Các mạng nơ-ron hồi quy (RNN) và sau này là Transformers có thể tận dụng tín hiệu này trong quá trình học. RNN gặp khó khăn trong việc nhớ các phụ thuộc dài hạn và có thể chậm trong quá trình huấn luyện do sử dụng đệ quy và không thể xử lý song song.

Transformers tránh được vấn đề đệ quy, cho phép tính toán song song và giảm thiểu sự suy giảm hiệu suất do các phụ thuộc dài hạn. Trong ví dụ của báo cáo này, lịch sử xem của người dùng được xử lý toàn bộ thay vì từng phim một để tránh các vấn đề phụ thuộc dài hạn. Không có nguy cơ mất thông tin quá khứ. Transformers sử dụng các nhúng vị trí (positional embeddings) để thay thế đệ quy. Ý tưởng là sử dụng các trọng số cố định hoặc được học để mã hóa thông tin liên quan đến một vị trí cụ thể của một token trong chuỗi. Hơn nữa, transformers triển khai cơ chế tự chú ý (self-attention) để tính toán điểm tương đồng giữa các phim trong một chuỗi (các ID phim trong lịch sử xem của người dùng). Ngoài ra, cơ chế chú ý và các nhúng vị trí cung cấp thông tin về mối quan hệ giữa các từ khác nhau.



Hình 16: BST Architecture

Cấu trúc BST nhận đầu vào là lịch sử xem của người dùng, bao gồm xếp hạng phim mục tiêu và "các đặc trưng khác". Đầu tiên, hệ thống nhúng các đặc trưng đầu vào này thành các vector có chiều thấp. Để nắm bắt tốt hơn các mối quan hệ giữa các mục trong chuỗi hành vi, tầng transformer được sử dụng để học một biểu diễn sâu hơn của mỗi mục trong chuỗi. Sau đó, bằng cách kết hợp các vector nhúng của "các đặc trưng khác" và đầu ra của tầng transformer, các mạng nơ-ron ba lớp (MLP) được sử dụng để học các tương tác của các đặc trưng ẩn và cuối cùng học để dự đoán xếp hạng của các phim chưa xem thông qua Mean Squared Error giữa xếp hạng phim dự đoán và xếp hạng phim thực tế.

5.2 Xây dựng mô hình Behavior Sequence Transformer (BST)

Các bước thực hiện:

- Chuẩn bị dữ liệu

- Tiền xử lý dữ liệu

- Thiết lập các tham số cần thiết như độ dài chuỗi (sequence length), step-size và các tham số

khác liên quan đến quá trình huấn luyện.

- Tách dữ liệu thành tập train và tập test.
- Tạo các chuỗi lịch sử xem phim của người dùng với độ dài cố định và bước nhảy là 1.
- Thực hiện các bước định dạng để chuẩn bị dữ liệu đầu vào cho mô hình.

- Xây dựng mô hình BST

- Kế thừa từ *pl.LightningModule*, một lớp trong PyTorch Lightning giúp đơn giản hóa quá trình huấn luyện mô hình.
- Khởi tạo lớp BST: Định nghĩa các lớp embedding, Transformer và Fully connected.
- Định nghĩa hàm forward: Xác định cách dữ liệu đi qua các lớp của mô hình.

- Khởi tạo mô hình và huấn luyện
- Đánh giá mô hình

- Dự đoán cho tập dữ liệu test và tính toán các chỉ số đánh giá như Precision, Recall và NDCG.

```
class BST(pl.LightningModule):
    def __init__(
        self, args=None,
    ):
        super().__init__()
        super(BST, self).__init__()

        self.save_hyperparameters()
        self.args = args
        #--------------------
        # Embedding layers
        ##Users
        self.embeddings_user_id = nn.Embedding(
            int(users.user_id.max())+1, int(math.sqrt(users.user_id.max()))+1
        )
        ###Users features embeddings
        self.embeddings_user_sex = nn.Embedding(
```

```
            len(users.sex.unique()), int(math.sqrt(len(users.sex.unique())))
        )
        self.embeddings_age_group = nn.Embedding(
            len(users.age_group.unique()), int(math.sqrt(len(users.age_group.
unique())))
        )
        self.embeddings_user_occupation = nn.Embedding(
            len(users.occupation.unique()), int(math.sqrt(len(users.occupation.
unique())))
        )
        self.embeddings_user_zip_code = nn.Embedding(
            len(users.zip_code.unique()), int(math.sqrt(len(users.sex.unique()))
)
        )

        ##Movies
        self.embeddings_movie_id = nn.Embedding(
            int(movies.movie_id.max())+1, int(math.sqrt(movies.movie_id.max()))
+1
        )

        ###Movies features embeddings
        genre_vectors = movies[genres].to_numpy()
        self.embeddings_movie_genre = nn.Embedding(
            genre_vectors.shape[0], genre_vectors.shape[1]
        )

        self.embeddings_movie_year = nn.Embedding(
            len(movies.year.unique()), int(math.sqrt(len(movies.year.unique())))
        )
        self.positional_embedding = PositionalEmbedding(8, 9)

        # Network
        self.transfomerlayer = nn.TransformerEncoderLayer(72, 3, dropout=0.2)
        self.linear = nn.Sequential(
            nn.Linear(
                661,
```

```
                1024,
            ),
            nn.LeakyReLU(),
            nn.Linear(1024, 512),
            nn.LeakyReLU(),
            nn.Linear(512, 256),
            nn.LeakyReLU(),
            nn.Linear(256, 1),
        )
        self.criterion = torch.nn.MSELoss()
        self.mae = torchmetrics.MeanAbsoluteError()
        self.mse = torchmetrics.MeanSquaredError()

    def encode_input(self,inputs):
        user_id, movie_history, target_movie_id,  movie_history_ratings,
    target_movie_rating, sex, age_group, occupation = inputs

        #MOVIES
        movie_history = self.embeddings_movie_id(movie_history)
        target_movie = self.embeddings_movie_id(target_movie_id)

        target_movie = torch.unsqueeze(target_movie, 1)
        transfomer_features = torch.cat((movie_history, target_movie),dim=1)

        #USERS
        user_id = self.embeddings_user_id(user_id)

        sex = self.embeddings_user_sex(sex)
        age_group = self.embeddings_age_group(age_group)
        occupation = self.embeddings_user_occupation(occupation)
        user_features = torch.cat((user_id, sex, age_group,occupation), 1)

        return transfomer_features, user_features, target_movie_rating.float()

    def forward(self, batch):
        transfomer_features, user_features, target_movie_rating = self.
    encode_input(batch)
```

```
        positional_embedding = self.positional_embedding(transfomer_features)
        transfomer_features = torch.cat((transfomer_features,
positional_embedding), dim=2)
        transformer_output = self.transfomerlayer(transfomer_features)
        transformer_output = torch.flatten(transformer_output,start_dim=1)

        #Concat with other features
        features = torch.cat((transformer_output,user_features),dim=1)

        output = self.linear(features)
        return output, target_movie_rating

    def training_step(self, batch, batch_idx):
        out, target_movie_rating = self(batch)
        out = out.flatten()
        loss = self.criterion(out, target_movie_rating)

        mae = self.mae(out, target_movie_rating)
        mse = self.mse(out, target_movie_rating)
        rmse =torch.sqrt(mse)
        self.log(
            "train/mae", mae, on_step=True, on_epoch=False, prog_bar=False
        )

        self.log(
            "train/rmse", rmse, on_step=True, on_epoch=False, prog_bar=False
        )

        self.log("train/step_loss", loss, on_step=True, on_epoch=False, prog_bar
=False)
        return loss

    def validation_step(self, batch, batch_idx):
        out, target_movie_rating = self(batch)
        out = out.flatten()
        loss = self.criterion(out, target_movie_rating)

```

```
121         mae = self.mae(out, target_movie_rating)
122         mse = self.mse(out, target_movie_rating)
123         rmse = torch.sqrt(mse)
124 
125         # Store metrics for later aggregation
126         self.log("val/step_loss", loss, on_step=True, on_epoch=False, prog_bar=
    False)  # Log step loss
127         return {"val_loss": loss, "mae": mae.detach(), "rmse": rmse.detach()}
128 
129     def on_validation_epoch_end(self):
130         outputs = self.trainer.logged_metrics  # Access logged metrics
131         avg_loss = outputs["val/step_loss"].mean()  # Calculate average loss
    from step losses
132         avg_mae = outputs["val/mae"].mean() if "val/mae" in outputs else None
133         avg_rmse = outputs["val/rmse"].mean() if "val/rmse" in outputs else None
134 
135         # Log epoch-level metrics
136         self.log("val/loss", avg_loss, on_step=False, on_epoch=True, prog_bar=
    False)
137         if avg_mae is not None:
138             self.log("val/mae", avg_mae, on_step=False, on_epoch=True, prog_bar=
    False)
139         if avg_rmse is not None:
140             self.log("val/rmse", avg_rmse, on_step=False, on_epoch=True,
    prog_bar=False)
141 
142     def test_epoch_end(self, outputs):
143         users = torch.cat([x["users"] for x in outputs])
144         y_hat = torch.cat([x["top14"] for x in outputs])
145         users = users.tolist()
146         y_hat = y_hat.tolist()
147 
148         data = {"users": users, "top14": y_hat}
149         df = pd.DataFrame.from_dict(data)
150         print(len(df))
151         df.to_csv("lightning_logs/predict.csv", index=False)
152 
```

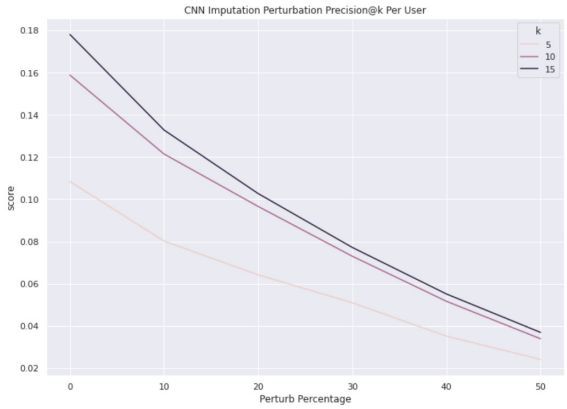
```
    def configure_optimizers(self):
        return torch.optim.AdamW(self.parameters(), lr=0.0005)

    @staticmethod
    def add_model_specific_args(parent_parser):
        parser = ArgumentParser(parents=[parent_parser], add_help=False)
        parser.add_argument("--learning_rate", type=float, default=0.01)
        return parser

    def setup(self, stage=None):
        print("Loading datasets")
        self.train_dataset = MovieDataset("data/train_data.csv")
        self.val_dataset = MovieDataset("data/val_data.csv")
        self.test_dataset = MovieDataset("data/test_data.csv")
        print("Done")

    def train_dataloader(self):
        return torch.utils.data.DataLoader(
            self.train_dataset,
            batch_size=128,
            shuffle=False,
            num_workers=os.cpu_count(),
        )

    def val_dataloader(self):
        return torch.utils.data.DataLoader(
            self.val_dataset,
            batch_size=128,
            shuffle=False,
            num_workers=os.cpu_count(),
        )

    def test_dataloader(self):
        return torch.utils.data.DataLoader(
            self.test_dataset,
            batch_size=128,
            shuffle=False,
```

```
            num_workers=os.cpu_count(),
        )
```

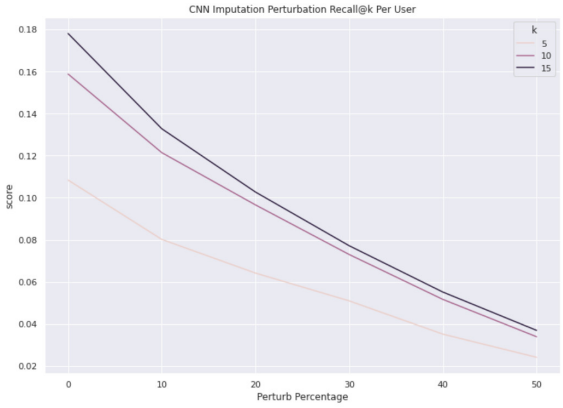
Listing 9: Xây dựng Class BST

6 Kết quả

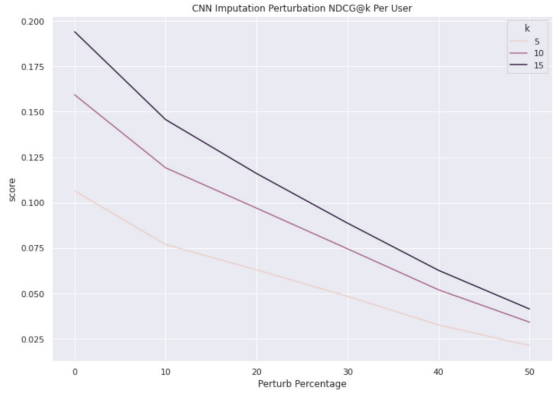
6.1 Trực quan hóa kết quả mô hình Caser:



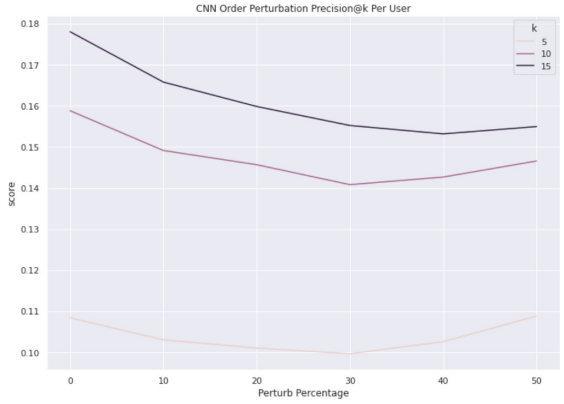
Hình 17: CASER Inputed Data Precision



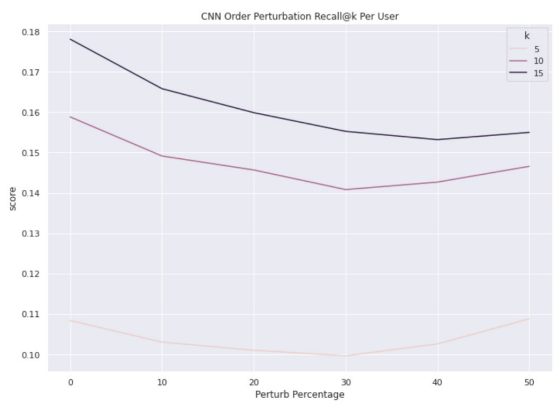
Hình 18: CASER Inputed Data Recall



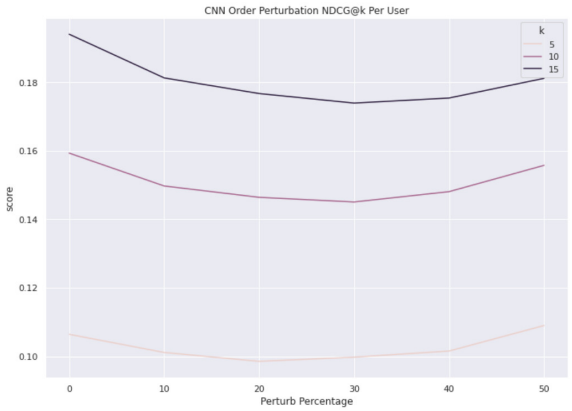
Hình 19: CASER Inputed Data NDCG



Hình 20: CASER Random Data Precision

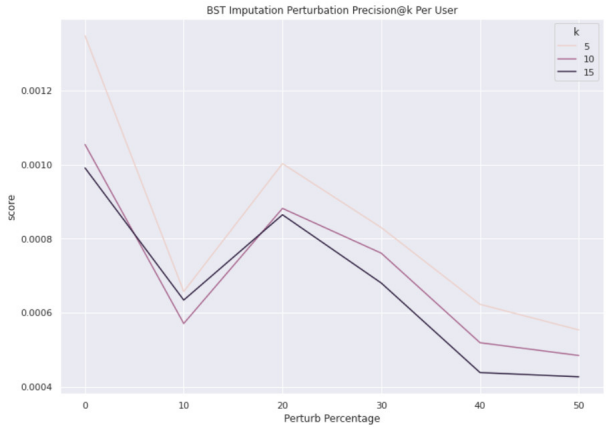


Hình 21: CASER Random Data Recall

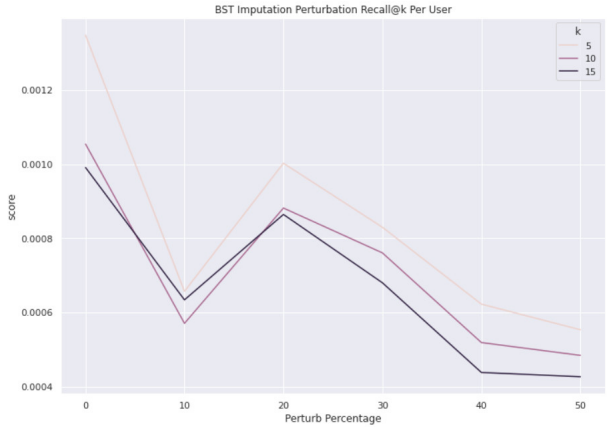


Hình 22: CASER Random Data NDCG

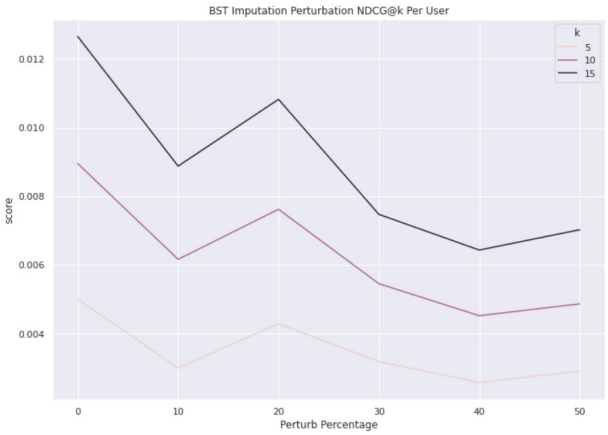
6.2 Trực quan hóa kết quả mô hình BST:



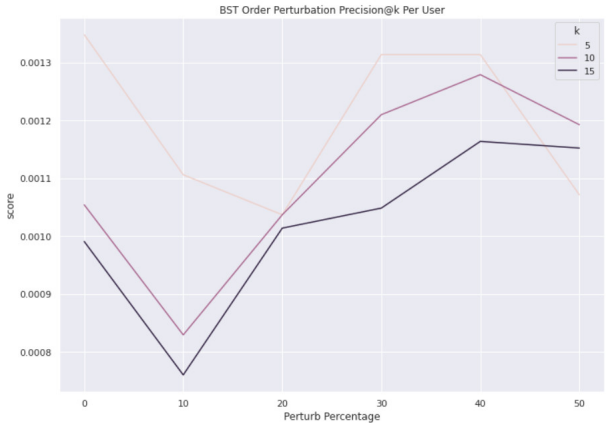
Hình 23: BST Inputed Data Precision



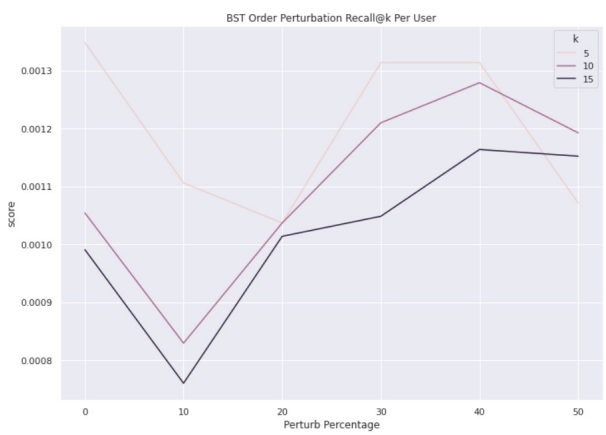
Hình 24: BST Inputed Data Recall



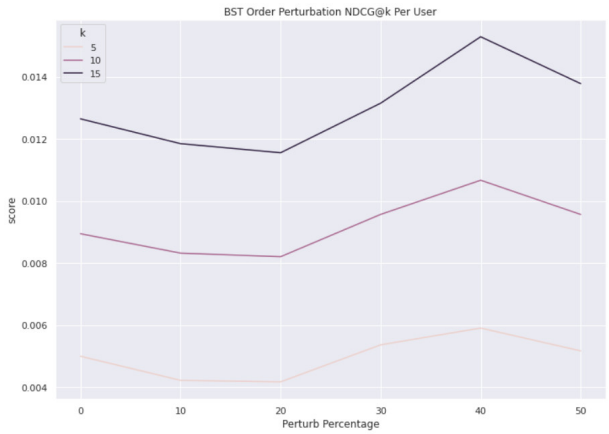
Hình 25: BST Inputed Data NDCG



Hình 26: BST Random Data Precision



Hình 27: BST Random Data Recall



Hình 28: BST Random Data NDCG

6.3 Nhận xét và đánh giá

6.3.1 Phân tích từ các biểu đồ

Mỗi biểu đồ hiển thị sự thay đổi của các chỉ số Precision@k, Recall@k và NDCG@k các biểu đồ này áp dụng cho cả hai mô hình và hai loại nhiễu khác nhau.

Mô hình CASER:

Imputed: khi tỷ lệ nhiễu tăng, các chỉ số như Precision, Recall và NDCG giảm mạnh. Hiệu suất này giảm khi tỷ lệ nhiễu tăng, cho thấy một xu hướng giảm không tuyến tính. Điều này có nghĩa là khi lượng dữ liệu bị thiếu tăng lên, CASER vẫn duy trì được một mức hiệu suất nhất định mặc dù giảm, cho thấy khả năng chống chịu của CASER đối với dữ liệu bị thiếu.

Random: CASER ít bị ảnh hưởng bởi nhiễu random hơn so với nhiễu imputed. Hiệu suất giảm ít nghiêm trọng hơn và ổn định hơn ở mức khoảng 30 đến 40% tỷ lệ nhiễu. Điều này cho thấy CASER có khả năng xử lý tốt hơn khi dữ liệu bị nhiễu random, với hiệu suất duy trì ổn định hơn trong điều kiện nhiễu cao.

Hiệu suất trung bình của CASER:

- **Precision**: Trung bình, CASER đạt điểm precision từ 0.1 đến 0.2. Điều này có nghĩa là với k = 10, mô hình có thể gợi ý đúng từ 1 đến 2 phim cho mỗi chuỗi người dùng.

- **Giải thích**: Khi tăng giá trị k, điểm số tăng lên do xác suất cao hơn cho ít nhất một hoặc hai gợi ý ban đầu của mô hình nằm trong danh sách phim lớn hơn. Điều này cho thấy rằng CASER có khả năng duy trì hiệu suất tốt khi danh sách gợi ý lớn hơn

Mô hình BST:

Imputed: BST có điểm số rất thấp đối với Precision, Recall và NDCG. Các chỉ số này thấp hơn từ 10 đến 100 lần so với CASER, cho thấy sự khác biệt lớn trong khả năng xử lý dữ liệu thiếu giữa hai mô hình. Vì điểm số quá thấp, ngay cả sự cải thiện nhỏ trong gợi ý cũng sẽ làm tăng đáng kể điểm số này. Điều này giải thích tại sao các biểu đồ không cho thấy một mẫu hình rõ ràng khi mỗi mô hình mới được huấn luyện cho từng thay đổi dữ liệu, tạo ra một lượng nhỏ ngẫu nhiên trong các gợi ý.

Random: Khi đối mặt với nhiễu random, BST cho thấy sự biến động lớn và không có mẫu hình rõ ràng. Điều này dẫn đến các mẫu hình phức tạp trong các biểu đồ, cho thấy BST không phù hợp với dữ liệu có tính ngẫu nhiên cao.
Mối quan hệ giữa các giá trị k là không tuyến tính:
Sự khác biệt giữa k = 5 và k = 10 lớn hơn sự khác biệt giữa k = 10 và k = 15 cho các chỉ số Precision, Recall và NDCG. Điều này cho thấy sự gia tăng hiệu suất khi tăng k không phải là tuyến tính, và có những mức độ thay đổi khác nhau ở các giá trị k khác nhau.

6.3.2 Tổng hợp và đánh giá

- **Khả năng chống chịu của mô hình:**

  CASER: CASER cho thấy khả năng chống chịu tốt hơn đối với biến đổi random so với biến đổi imputed. Điều này có thể do cơ chế mạng CNN của mô hình, cho phép nó học được các đặc trưng một cách linh hoạt hơn, từ đó phản ứng hiệu quả hơn với sự ngẫu nhiên trong dữ liệu.

  BST: BST cho thấy khả năng chống chịu rất kém đối với cả hai loại biến đổi, đặc biệt là với biến đổi imputed. Các chỉ số đánh giá của BST rất thấp, cho thấy mô hình không thể xử lý tốt khi dữ liệu bị thiếu hoặc ngẫu nhiên hóa.

- **Phân tích theo K**:

  CASER: Kết quả thực nghiệm cho thấy rằng khi tăng giá trị k (số lượng phim được đề xuất), mô hình có khả năng cung cấp các đề xuất chính xác hơn. Điều này phản ánh khả năng của mô hình trong việc đề xuất một danh sách phim lớn hơn chứa ít nhất một hoặc hai phim phù hợp.

  BST: Kết quả thực nghiệm cho thấy rằng khi tăng giá trị k, BST vẫn không cải thiện đáng kể về hiệu suất. Sự khác biệt giữa các giá trị k khác nhau không rõ ràng, và các biểu đồ không cho thấy mẫu hình nhất quán.

- **Ổn định và độ tin cậy:**

  CASER: Sự giảm hiệu suất không tuyến tính khi tăng biến đổi cho thấy mô hình CASER cần sự ổn định dữ liệu đầu vào để hoạt động tối ưu. Việc này cần được lưu ý khi áp dụng mô hình trong các môi trường có tính bất định cao về dữ liệu.

  BST: BST thiếu sự ổn định và độ tin cậy cần thiết khi đối mặt với các biến đổi trong dữ liệu. Hiệu suất biến động mạnh và không có mẫu hình rõ ràng cho thấy mô hình không phù hợp để áp dụng trong môi trường có tính bất định cao.

- **Hướng phát triển:**

  CASER: Để cải thiện hiệu suất, việc huấn luyện mô hình với các kịch bản dữ liệu biến đổi nhiều hơn và phát triển các cơ chế biến đổi có thể giúp tăng cường độ chính xác và độ tin cậy của mô hình trong các ứng dụng thực tế.

  BST: Để cải thiện hiệu suất, BST cần được điều chỉnh để có khả năng chống chịu nhiễu tốt hơn. Cũng có thể cần kết hợp các phương pháp tiền xử lý dữ liệu để giảm thiểu tác động của nhiễu và cải thiện độ tin cậy của mô hình.

7 Tổng kết

Mô hình CASER (CNN) thể hiện khả năng vượt trội trong việc dự đoán chuỗi các mục tiêu từ lịch sử xem của người dùng, được minh chứng qua các chỉ số đánh giá như Precision@k, Recall@k và NDCG@k. CASER không chỉ dự đoán phim tiếp theo mà người dùng có thể xem, mà còn dự đoán cả chuỗi phim dựa trên đánh giá tốt nhất mà người dùng có thể đưa ra, giúp mô hình phù hợp hơn

với nhiệm vụ đề xuất chuỗi đã sắp xếp. CASER sử dụng hàm mất mát binary cross-entropy trong quá trình huấn luyện, dựa trên đầu ra từ lớp cuối cùng của mô hình là điểm số của người dùng cho mỗi phim, giúp học các xác suất dự đoán và cung cấp các khuyến nghị dựa trên sở thích cá nhân của người dùng. Mặc dù CASER cho thấy khả năng chống chịu tốt với sự xáo trộn ngẫu nhiên của dữ liệu, mô hình này lại thể hiện hiệu suất kém hơn khi xử lý các biến đổi dữ liệu, đặc biệt là khi dữ liệu bị thiếu, làm suy giảm hiệu suất đáng kể. Điều này cần được cân nhắc khi áp dụng mô hình trong môi trường có tính bất định dữ liệu cao.

BST cho thấy khả năng chống chịu rất kém đối với cả hai loại biến đổi, đặc biệt là với biến đổi imputed. Các chỉ số đánh giá của BST rất thấp, cho thấy mô hình không thể xử lý tốt khi dữ liệu bị thiếu hoặc ngẫu nhiên hóa. BST học cách dự đoán xếp hạng của mục tiếp theo, nhưng không trực tiếp học được nhiệm vụ gợi ý chuỗi có thứ tự, dẫn đến hiệu suất kém và không ổn định.

CASER vượt trội hơn BST trong tất cả các chỉ số đánh giá. CASER duy trì được hiệu suất cao ngay cả khi dữ liệu bị xáo trộn ngẫu nhiên, điều này quan trọng đối với các hệ thống gợi ý khi dữ liệu đầu vào không hoàn toàn chính xác hoặc có tính ngẫu nhiên cao. Với khả năng dự đoán chuỗi tốt, CASER cung cấp các gợi ý chính xác và liên quan hơn, nâng cao trải nghiệm người dùng. Khả năng mở rộng và áp dụng rộng rãi của CASER trong các hệ thống gợi ý khác nhau cũng là một lợi thế. Mặc dù CASER yêu cầu nhiều tài nguyên tính toán hơn, nhưng hiệu suất cao của nó có thể bù đắp được chi phí này. Trong khi đó, BST không ổn định và hiệu suất kém khi dữ liệu bị biến đổi hoặc nhiễu loạn, hạn chế khả năng áp dụng trong các tình huống thực tế và yêu cầu cao về độ chính xác và ổn định.

Với bài báo cáo về sự nhạy cảm với dữ liệu nhiễu của chuỗi tuần tự, các mô hình nhận thức chuỗi như CASER (CNN) và BST có nhiều ảnh hưởng quan trọng đối với các hệ thống gợi ý dựa trên chuỗi. Khả năng xử lý dữ liệu biến đổi và nhiễu loạn của hai mô hình này có sự khác biệt đáng kể. Độ chính xác của gợi ý cũng khác nhau giữa hai mô hình. Với khả năng dự đoán chuỗi tốt, CASER cung cấp các gợi ý chính xác và liên quan hơn, từ đó nâng cao trải nghiệm người dùng. Điều này đặc biệt quan trọng trong các hệ thống gợi ý nội dung như phim, âm nhạc hoặc sản phẩm mua sắm. Trái lại, sự thiếu ổn định và độ chính xác thấp của BST có thể dẫn đến gợi ý không phù hợp, làm giảm trải nghiệm người dùng.

Về khả năng mở rộng và áp dụng trong thực tế, CASER có thể mở rộng và áp dụng rộng rãi trong các hệ thống gợi ý khác nhau, phù hợp với các môi trường dữ liệu phức tạp và biến đổi. BST không phù hợp để áp dụng rộng rãi trong các hệ thống gợi ý có yêu cầu cao về độ chính xác và ổn định.

Chi phí tính toán và triển khai của CASER và BST cũng có sự khác biệt. Mặc dù CASER yêu cầu nhiều tài nguyên tính toán hơn, nhưng hiệu suất cao của nó có thể bù đắp được chi phí này. Đối với các ứng dụng yêu cầu chất lượng gợi ý cao, việc đầu tư vào tài nguyên tính toán để sử dụng CASER là xứng đáng. Ngược lại, BST đơn giản và dễ triển khai với chi phí tính toán thấp hơn, nhưng hiệu suất kém của nó khi đối mặt với dữ liệu nhiễu có thể dẫn đến chi phí ẩn cao hơn trong việc quản lý và cải thiện hệ thống.

Hướng mở rộng trong tương lai:

- **Tăng cường khả năng chống chịu với biến đổi dữ liệu:**

  Phát triển các kỹ thuật mới để mô hình có thể nhận diện và điều chỉnh các biến đổi trong dữ liệu một cách hiệu quả hơn, bao gồm cả việc áp dụng các phương pháp học máy chịu đựng nhiễu để mô hình không quá nhạy cảm với các sai lệch trong dữ liệu đầu vào.

- **Tối ưu hóa kiến trúc mạng:**

  Khám phá các cấu trúc mạng nơ-ron phức tạp hơn hoặc các cách tiếp cận mới như mạng nơ-ron chập sâu hơn hoặc kết hợp với các kỹ thuật như Attention để nâng cao khả năng hiểu và xử lý chuỗi dữ liệu.

- **Nghiên cứu về tính bền vững của mô hình:**

  Tiến hành các nghiên cứu để đánh giá sự bền vững của mô hình trước các loại tấn công dữ liệu hoặc khi dữ liệu có sự thay đổi lớn, từ đó tìm ra các giải pháp để tăng cường độ tin cậy và an toàn cho mô hình khi triển khai thực tế.

- **Cải thiện hàm mất mát:**

  Thử nghiệm với các hàm mất mát khác nhau hoặc phát triển một hàm mất mát mới đặc biệt cho các tác vụ đề xuất chuỗi, có thể giúp mô hình học tốt hơn các mối liên kết giữa các mục trong chuỗi và cải thiện độ chính xác của các dự đoán.

- **Tương tác người dùng thời gian thực:**

  Phát triển mô hình để hỗ trợ cập nhật và điều chỉnh dựa trên phản hồi hoặc hành vi của người dùng trong thời gian thực, từ đó làm cho các khuyến nghị trở nên linh hoạt và cá nhân hóa hơn.

- **Áp dụng trong lĩnh vực khác:**

  Khảo sát khả năng áp dụng mô hình CASER cho các loại hình dữ liệu khác hoặc trong các lĩnh vực khác ngoài đề xuất phim, chẳng hạn như trong lĩnh vực thương mại điện tử hoặc nhạc số, nơi việc đề xuất sản phẩm hoặc bài hát có thể được cải thiện bằng cách hiểu rõ hơn về hành vi và sở thích của người dùng.

- **Sử dụng dữ liệu đa dạng hơn:**

  Khai thác và tích hợp các nguồn dữ liệu đa dạng hơn, bao gồm thông tin ngữ cảnh và dữ liệu không cấu trúc, để cải thiện khả năng của mô hình trong việc đưa ra các khuyến nghị chính xác và phù hợp hơn.

  Những hướng phát triển này không chỉ nhằm mục đích tăng cường hiệu suất của mô hình CASER trong các nhiệm vụ cụ thể mà còn mở rộng khả năng áp dụng của nó trong các ứng dụng thực tế, đồng thời đảm bảo tính bền vững và độ tin cậy trong môi trường đa dạng và thay đổi.

Tài liệu

[1] Niko Pajkovic. Algorithms and taste-making: Exposing the netflix recommender system's operational logics. Convergence, 28(1):214–235, 2022.

[2] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. ACM Computing Surveys (CSUR), 51(4):1–36, 2018.

[3] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In Proceedings of the eleventh ACM international conference on web search and data mining, pages 565–573, 2018.

[4] Sejoon Oh, Berk Ustun, Julian McAuley, and Srijan Kumar. Rank list sensitivity of recommender systems to interaction perturbations. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pages 1584–1594, 2022

[5] Wen Chen, Pipei Huang, Jiaming Xu, Xin Guo, Cheng Guo, Fei Sun, Chao Li, Andreas Pfadler, Huan Zhao, and Binqiang Zhao. 2019. POG: Personalized Outfit Generation for Fashion Recommendation at Alibaba iFashion.

[6] R. He and J. McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. In International Conference on Data Mining. IEEE.