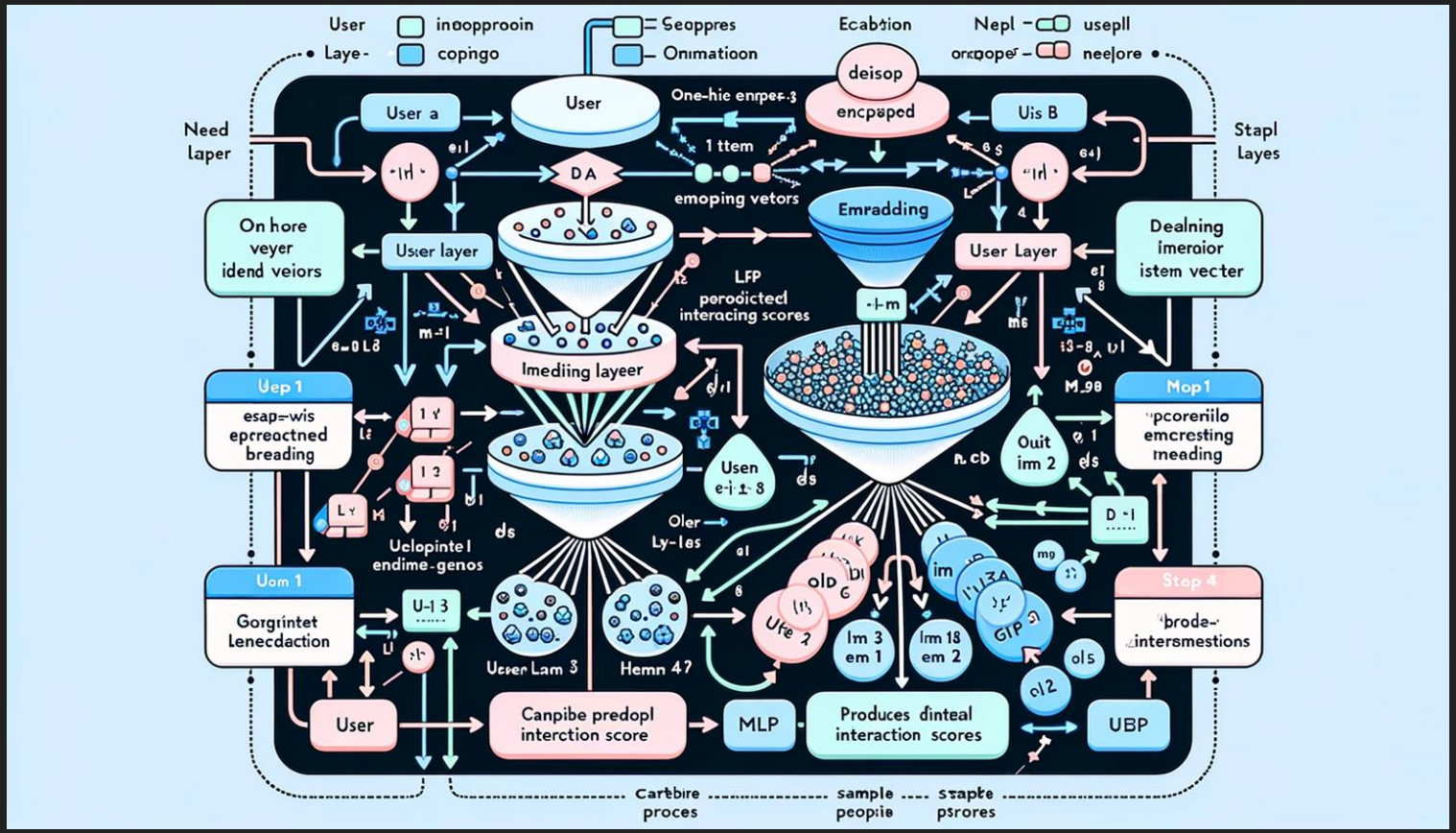
Neural Collaborative Filtering

Przemysław Rośleń 180150
Jakub Sachajko 179976

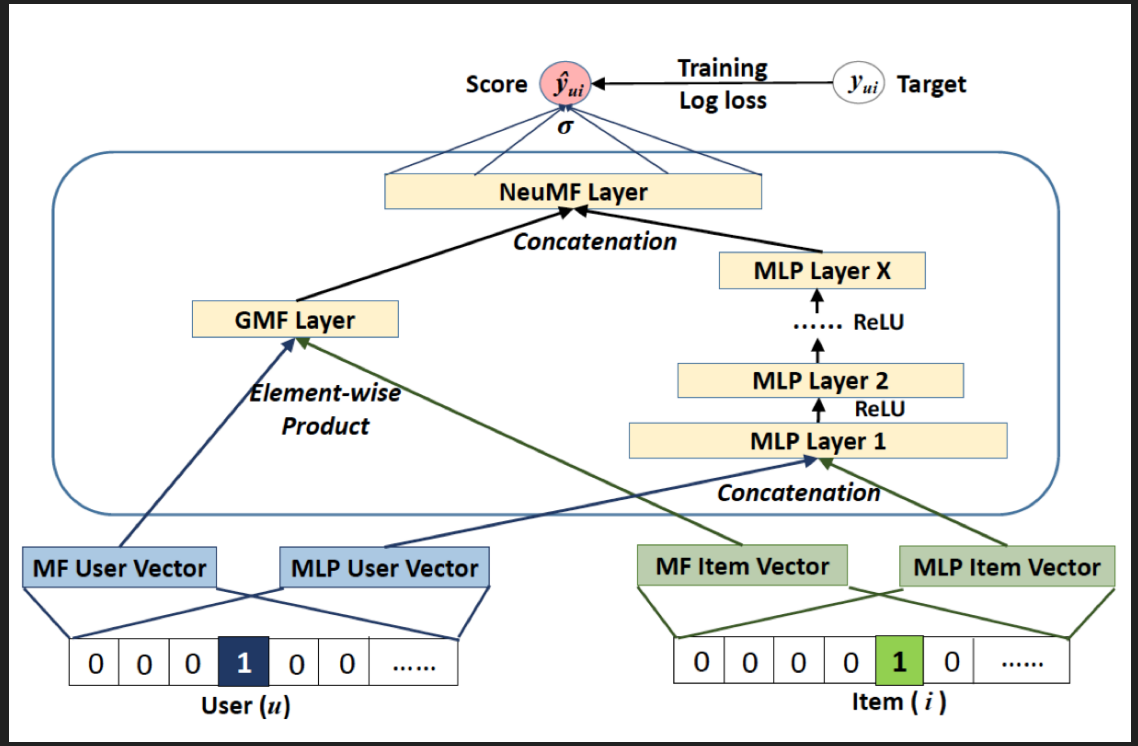
Plan prezentacji

1. **Wyjaśnienie algorytmu NCF + przykład**
2. **Analiza kodu**
3. **3 różne warianty algorytmu NCF**
4. **Zbiory danych**
5. **Analiza eksperymentów**
6. **Wyniki eksperymentów**
7. **Wnioski**

Neural Collaborative Filtering w oczach GPT-4



Opis Algorytmu



Przebieg algorytmu NCF dla przykładowych danych

Przykładowe dane wejściowe:

- Users: {User A, User B}
- Items: {Item 1, Item 2} Items: {Item 1, Item 2}
- Interactions: {User A likes Item 1, Item 2, and Item 3; User B likes Item 4}

Kroki algorytmu:

1. Input Layer: Represent users and items as one-hot encoded vectors:
    - User A: [1, 0], User B: [0, 1]
    - Item 1: [1, 0, 0, 0], Item 2: [0, 1, 0, 0], Item 3: [0, 0, 1, 0], Item 4: [0, 0, 0, 1]
2. Embedding Layer:
    - Convert sparse vectors to dense embeddings
    - User A Embedding: [0.4, 0.6], User B Embedding: [0.7, 0.3]
    - Item 1 Embedding: [0.5, 0.5], Item 2 Embedding: [0.6, 0.4], Item 3 Embedding: [0.3, 0.7], Item 4 Embedding: [0.2, 0.8]

Przebieg algorytmu NCF dla przykładowych danych c.d

3. Neural CF Layers:

   - Process embeddings through neural network layers, minimizing the cost function

   - GMF Pathway:
     i. Element-wise product of user and item embeddings
     ii. For example User A and Item 1: [0.4 * 0.5, 0.6 * 0.5] = [0.2, 0.3]

$$L_{sqr} = \sum_{(u,i) \in \mathcal{Y} \cup \mathcal{Y}^-} w_{ui}(y_{ui} - \hat{y}_{ui})^2,$$

   - MLP Pathway:
     i. Concatenate user and item embeddings and pass through MLP layers
     ii. For example User A and Item 1: Concatenated Vector [0.4, 0.6, 0.5, 0.5]

   - Output Layer (NeuMF):
     i. Combine outputs from GMF and MLP, applying a final activation function (e.g., sigmoid, tanh) for prediction
     ii. Predicted Score: Indicates likelihood of User A liking Item 1, and similarly for other user-item pairs

3. Final Prediction:
   - The score would reflect the model's prediction of whether a user likes a specific item, based on learned patterns from the input data.

Analiza Kodu

- Struktura repozytorium składa się z pięciu plików o rozszerzeniu python:
  - Dataset.py
  - evaluate.py
  - GMF.py
  - MLP.py
  - NeuMF.py
- Metoda ewaluacji “leave-one-out”, która została wspomniana w artykule nie została zaimplementowana w kodzie
- Kod został napisany w prosty sposób, przez co zrozumienie go jest bardzo proste, a sam kod jest przejrzysty
- Kod jest napisany poprawnie
- pozwala na modyfikowanie aż 15 hiperparametrów
- Zaimplementowane modele są w dużym stopniu konfigurowalne, dzięki czemu dają ogromne pole do eksperymentowania z różnymi wartościami hiperparametrów

Analiza Kodu - hiperparametry

```
def parse_args():
    parser = argparse.ArgumentParser(description="Run NeuMF.")
    parser.add_argument('--path', nargs='?', default='Data/',
                        help='Input data path.')
    parser.add_argument('--dataset', nargs='?', default='ml-1m',
                        help='Choose a dataset.')
    parser.add_argument('--epochs', type=int, default=100,
                        help='Number of epochs.')
    parser.add_argument('--batch_size', type=int, default=256,
                        help='Batch size.')
    parser.add_argument('--num_factors', type=int, default=8,
                        help='Embedding size of MF model.')
    parser.add_argument('--layers', nargs='?', default='[64,32,16,8]',
                        help="MLP layers. Note that the first layer is the concatenation of user and item embeddings.
    parser.add_argument('--reg_mf', type=float, default=0,
                        help='Regularization for MF embeddings.')
    parser.add_argument('--reg_layers', nargs='?', default='[0,0,0,0]',
                        help="Regularization for each MLP layer. reg_layers[0] is the regularization for embeddings.")
    parser.add_argument('--num_neg', type=int, default=4,
                        help='Number of negative instances to pair with a positive instance.')
    parser.add_argument('--lr', type=float, default=0.001,
                        help='Learning rate.')
    parser.add_argument('--learner', nargs='?', default='adam',
                        help='Specify an optimizer: adagrad, adam, rmsprop, sgd')
    parser.add_argument('--verbose', type=int, default=1,
                        help='Show performance per X iterations')
    parser.add_argument('--out', type=int, default=1,
                        help='Whether to save the trained model.')
    parser.add_argument('--mf_pretrain', nargs='?', default='',
                        help='Specify the pretrain model file for MF part. If empty, no pretrain will be used')
    parser.add_argument('--mlp_pretrain', nargs='?', default='',
                        help='Specify the pretrain model file for MLP part. If empty, no pretrain will be used')
    return parser.parse_args()
```

GMF

```
def get_model(num_users, num_items, latent_dim, regs=[0,0]):
    # Input variables
    user_input = Input(shape=(1,), dtype='int32', name = 'user_input')
    item_input = Input(shape=(1,), dtype='int32', name = 'item_input')

    MF_Embedding_User = Embedding(input_dim = num_users, output_dim = latent_dim, name = 'user_embedding',
                                  init = init_normal, W_regularizer = l2(regs[0]), input_length=1)
    MF_Embedding_Item = Embedding(input_dim = num_items, output_dim = latent_dim, name = 'item_embedding',
                                  init = init_normal, W_regularizer = l2(regs[1]), input_length=1)

    # Crucial to flatten an embedding vector!
    user_latent = Flatten()(MF_Embedding_User(user_input))
    item_latent = Flatten()(MF_Embedding_Item(item_input))

    # Element-wise product of user and item embeddings
    predict_vector = merge([user_latent, item_latent], mode = 'mul')

    # Final prediction layer
    #prediction = Lambda(lambda x: K.sigmoid(K.sum(x)), output_shape=(1,))(predict_vector)
    prediction = Dense(1, activation='sigmoid', init='lecun_uniform', name = 'prediction')(predict_vector)

    model = Model(input=[user_input, item_input],
                  output=prediction)

    return model
```

MLP

```
def get_model(num_users, num_items, layers = [20,10], reg_layers=[0,0]):
    assert len(layers) == len(reg_layers)
    num_layer = len(layers) #Number of layers in the MLP
    # Input variables
    user_input = Input(shape=(1,), dtype='int32', name = 'user_input')
    item_input = Input(shape=(1,), dtype='int32', name = 'item_input')

    MLP_Embedding_User = Embedding(input_dim = num_users, output_dim = layers[0]/2, name = 'user_embedding',
                                   init = init_normal, W_regularizer = l2(reg_layers[0]), input_length=1)
    MLP_Embedding_Item = Embedding(input_dim = num_items, output_dim = layers[0]/2, name = 'item_embedding',
                                   init = init_normal, W_regularizer = l2(reg_layers[0]), input_length=1)

    # Crucial to flatten an embedding vector!
    user_latent = Flatten()(MLP_Embedding_User(user_input))
    item_latent = Flatten()(MLP_Embedding_Item(item_input))

    # The 0-th layer is the concatenation of embedding layers
    vector = merge([user_latent, item_latent], mode = 'concat')

    # MLP layers
    for idx in xrange(1, num_layer):
        layer = Dense(layers[idx], W_regularizer= l2(reg_layers[idx]), activation='relu', name = 'layer%d' %idx)
        vector = layer(vector)

    # Final prediction layer
    prediction = Dense(1, activation='sigmoid', init='lecun_uniform', name = 'prediction')(vector)

    model = Model(input=[user_input, item_input],
                  output=prediction)

    return model
```

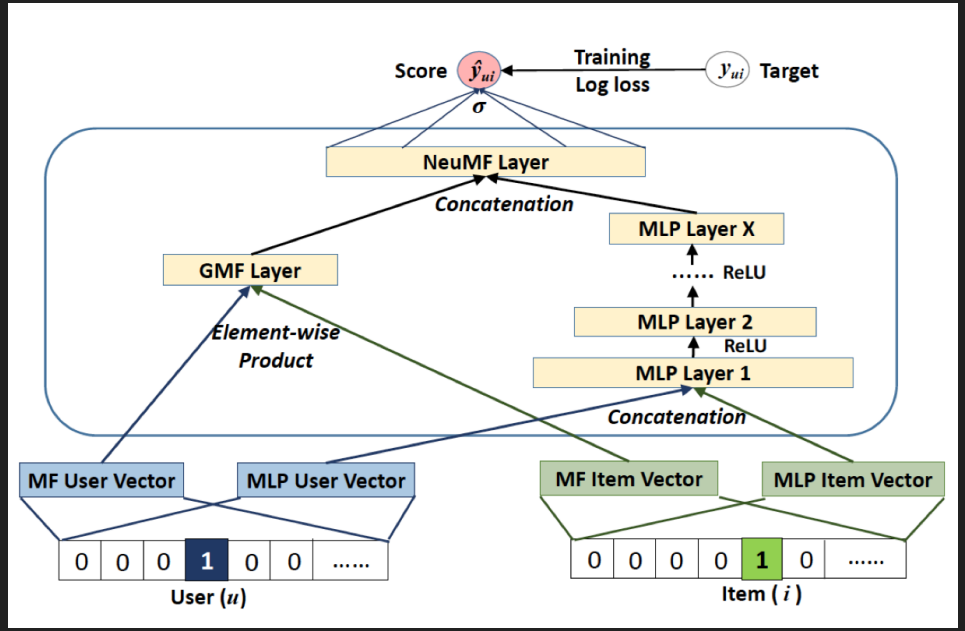
Różnice między GMF a MLP

- GMF opiera się na klasycznym podejściu użytkownika i przedmiotu są przedstawione jako wektory, a preferencje są wyznaczane poprzez iloczyn skalarny tych wektorów.
- GMF jest bardziej ograniczone, co sprawia, że może być mniej skuteczne w modelowaniu złożonych zależności.
- Efektywna obliczeniowo

- Model MLP składa się z warstw ukrytych z neuronami, które przetwarzają reprezentacje użytkownika i przedmiotu.
- MLP jest bardziej elastyczne, ponieważ może modelować nieliniowe zależności między użytkownikami a przedmiotami.
- Kosztowna obliczeniowo

NeuMF

- Podejście Hybrydowe
- Łączy szukanie liniowych wzorców (GMF) i nieliniowych wzorców (MLP)
- Kosztowna obliczeniowo



NeuMF

```
def load_pretrain_model(model, gmf_model, mlp_model, num_layers):
    # MF embeddings
    gmf_user_embeddings = gmf_model.get_layer('user_embedding').get_weights()
    gmf_item_embeddings = gmf_model.get_layer('item_embedding').get_weights()
    model.get_layer('mf_embedding_user').set_weights(gmf_user_embeddings)
    model.get_layer('mf_embedding_item').set_weights(gmf_item_embeddings)

    # MLP embeddings
    mlp_user_embeddings = mlp_model.get_layer('user_embedding').get_weights()
    mlp_item_embeddings = mlp_model.get_layer('item_embedding').get_weights()
    model.get_layer('mlp_embedding_user').set_weights(mlp_user_embeddings)
    model.get_layer('mlp_embedding_item').set_weights(mlp_item_embeddings)

    # MLP layers
    for i in xrange(1, num_layers):
        mlp_layer_weights = mlp_model.get_layer('layer%d' %i).get_weights()
        model.get_layer('layer%d' %i).set_weights(mlp_layer_weights)

    # Prediction weights
    gmf_prediction = gmf_model.get_layer('prediction').get_weights()
    mlp_prediction = mlp_model.get_layer('prediction').get_weights()
    new_weights = np.concatenate((gmf_prediction[0], mlp_prediction[0]), axis=0)
    new_b = gmf_prediction[1] + mlp_prediction[1]
    model.get_layer('prediction').set_weights([0.5*new_weights, 0.5*new_b])
    return model
```

MovieLens 1 Million

- Zbiór danych zawierający oceny filmów zebranych od użytkowników platformy MovieLens. Głównym celem zbierania tego zbioru danych było umożliwienie badania i rozwijania algorytmów rekomendacji filmów.
- Został utworzony i jest zarządzany przez GroupLens Research, które jest grupą badawczą z University of Minnesota.
- Zbiór Movielens 1M zawiera milion ocen filmów dokonanych przez użytkowników. Każda ocena jest przypisana do określonego użytkownika i filmu. Oprócz tego, zbiór zawiera informacje o użytkownikach i filmach, takie jak identyfikatory, tytuły, gatunki itp.
- Oceny w zbiorze oznaczają subiektywne preferencje użytkowników wobec konkretnych filmów. Są to liczby całkowite reprezentujące oceny przypisane przez użytkowników danym filmom.

MovieLens - szczegółowa analiza

- Wartości dla podstawowych charakterystyk zbioru danych:
  - Ilość ocen: 1 000 209
  - Ilość użytkowników: 6 040
  - Ilość filmów: 3 706

- Średnie ilości ocen na użytkownika/produkt:
  - Średnia ilość ocen na użytkownika: Około 166
  - Średnia ilość ocen na film: Około 257

- Gęstość macierzy ocen:
  - Gęstość macierzy ocen można obliczyć jako ilość ocen / (ilość użytkowników * ilość filmów).
  - Dla Movielens 1M gęstość ta wynosi około 4,47%.

- Dodatkowe informacje zawarte w zbiorze:

  - Oprócz ocen, zbiór zawiera informacje o użytkownikach (np. wiek, płeć) i filmach (np. gatunki). Te dodatkowe informacje mogą być wykorzystywane do bardziej zaawansowanych metod rekomendacyjnych, takich jak spersonalizowane rekomendacje uwzględniające preferencje użytkowników.

Pinterest (pinterest-20)

- Brak informacji o celu zbierania tego zbioru danych.
- Pinterest-20 to zestaw danych pochodzący z Pinterest, platformy mediów społecznościowych skoncentrowanej na udostępnianiu i odkrywaniu pomysłów za pomocą obrazów. Ten zestaw danych to podzbiór zawierający interakcje użytkowników z pinami na Pinterest.
- Interakcje w zbiorze oznaczają subiektywne preferencje użytkowników wobec konkretnych postów/zdjęć na platformie Pinterest. Są one reprezentowane w postaci liczb w systemie binarnym, gdzie 1 oznacza, że dana interakcja wystąpiła, a 0 oznacza brak interakcji.

Pinterest - szczegółowa analiza

- Wartości dla podstawowych charakterystyk zbioru danych:
  - Ilość interakcji: 1 500 809
  - Ilość użytkowników: 55 187
  - Ilość pinów: 9 916

- Średnie ilości interakcji na użytkownika/produkt:
  - Średnia ilość interakcji na użytkownika: 27
  - Średnia ilość interakcji na pina: 151

- Gęstość macierzy ocen:
  - Gęstość macierzy ocen można obliczyć jako ilość interakcji / (ilość użytkowników * ilość pinów).
  - Dla zbioru Pinterest-20 gęstość ta wynosi około 0,27%.

- Dodatkowe informacje zawarte w zbiorze:

  - Brak

Analiza Eksperymentu - metody ewaluacji i miary

- Na początku skorzystano z metod ewaluacji opartych na strategii "leave-one-out"
- Dla każdego użytkownika, jego ostatnia interakcja została wzięta jako zestaw testowy, a pozostałe dane zostały użyte do treningu.
- Ze względu na czasochłonność oceny wszystkich elementów dla każdego użytkownika, zastosowano strategię losowego próbkowania i ewaluacji 100 elementów, które nie były wcześniej interakcjonowane przez danego użytkownika.
- skorzystano z następujących miar oceny:
  - Hit ratio (HR@10 oraz HR@K)
  - Normalized discounted cumulative gain (NDCG@10 oraz NDCG@K)

gdzie K: 1-10

- Dodatkowo, porównano zaproponowane metody NCF (GMF, MLP oraz NeuMF) z następującymi metodami:
  - ItemPop
  - ItemKNN
  - BPR
  - eALS

Analiza Eksperymentu - zbiór danych

- Wykorzystano dwa zestawy danych:
  - MovieLens
  - Pinterest

- MovieLens to zbiór danych zawierający milion ocen filmów, podczas gdy Pinterest to zbiór danych zawierający interakcje użytkowników z pinami na Pinterest

- Oba zbiory danych zostały poddane filtrowaniu, aby pozostawić tylko użytkowników z co najmniej 20 interakcjami

- Dane MovieLens zostały dodatkowo przekształcone z danych o ocenach filmów (explicit feedback) na dane implicit feedback, gdzie każda interakcja została oznaczona jako 0 lub 1, w zależności od tego, czy użytkownik ocenił film

- Zbiór Pinterest w oryginale posiadał dane typu implicit feedback, więc nie wymagał tego przekształcenia

- zastosowano następujące podziały danych na zbiór treningowy i testowy:
  - leave-one-out (nie zaimplementowana w kodzie)
  - ewaluacja po 100 losowych itemów

Analiza Eksperymentu - podsumowanie

- Wybrano odpowiednie zbiory danych, na których przeprowadzono wszystkie konieczne tranformacje, aby dostosować je do rozwiązywanego problemu
- Podczas przeprowadzania eksperymentów obrano konkretną metodę ewaluacji, którą następnie zmieniono na bardziej odpowiednią dla zastosowanych zbiorów danych (zbyt duża złożoność czasowa metody “leave-one-out”
- Zdefiniowano konkretne miary oceny systemu, które są adekwatne do problemu przewidywania niezaobserwowanych interakcji użytkowników z produktami, które w następnej kolejności posłużą do rankingu pozycji
- Dokonano porównania uzyskanych wyników z rezultatami innych znanych algorytmów dedykowanych tego typu problemom
- Wniosek: Naszym zdaniem protokół oceny algorytmu odpowiada problemowi, który autorzy artykułu chcieli rozwiązać

Wykonanie kodu

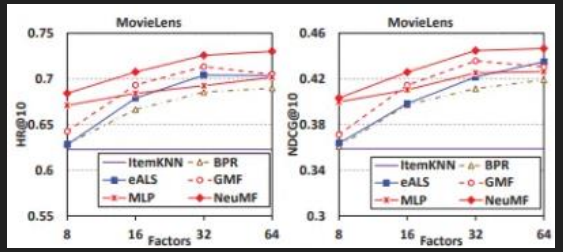
- Aby odpalić program należy kierować się instrukcją w pliku README.md dołączonym do projektu na githubie
- Do repozytorium dołączony jest Dockerfile oraz datasety MovieLens oraz Pinterest.
- Procedura uruchamiania programu:
  - Zaczynamy od zbudowania Dockerfile'a : `` `docker build --no-cache=true -t ncf-keras-theano .` ``
  - Następnie uruchamiamy dowolny z czterech możliwych modeli: GMF, MLP, NeuMF oraz z pre-trainingiem za pomocą odpowiedniej komendy
  - Przykładowa komenda dla MLP: `docker run --volume=$(pwd):/home ncf-keras-theano python MLP.py --dataset ml-1m --epochs 20 --batch_size 256 --layers [64,32,16,8] --reg_layers [0,0,0,0] --num_neg 4 --lr 0.001 --learner adam --verbose 1 --out 1`
- Na wyniki GMF oczekuje się około 16 minut
- Na wyniki MLP oczekuje się około 38 minut
- Na wyniki NeuMF (bez pre-training) oczekuje się około 39 minut

Pliki konfiguracyjne

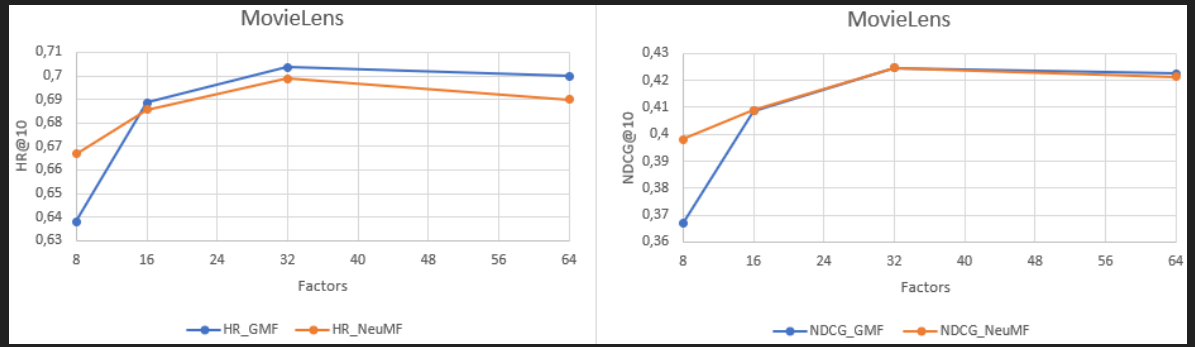
- Dockerfile - plik odpowiadający za konfigurację środowiska uruchomieniowego programu m.i.n:
  - utworzenie odpowiednich katalogów
  - wersja ubuntu
  - wersja pythona
  - wersja Kerasa
  - wersja Theano
  - wersja h5py
  - wersja numpy

- Pliki z kodem dla poszczególnych modeli pełniące również funkcję plików konfiguracyjnych umożliwiając inicjalizację hiperparametrów dla poszczególnych modeli

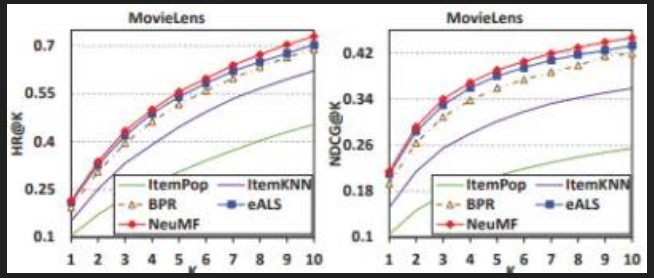
Eksperyment 1.1 - różne liczby cech ukrytych



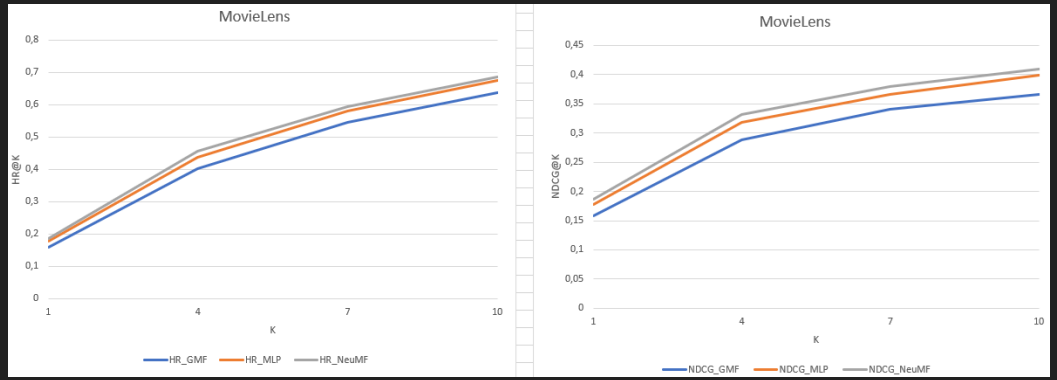
VS



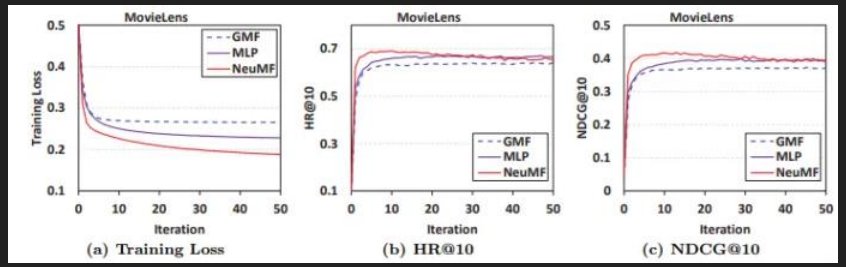
Eksperyment 1.2 - różne wartości parametru TopK



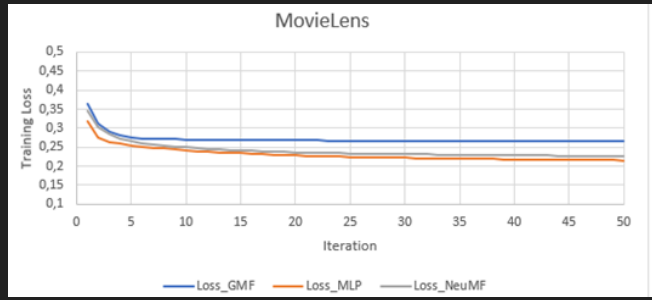
VS

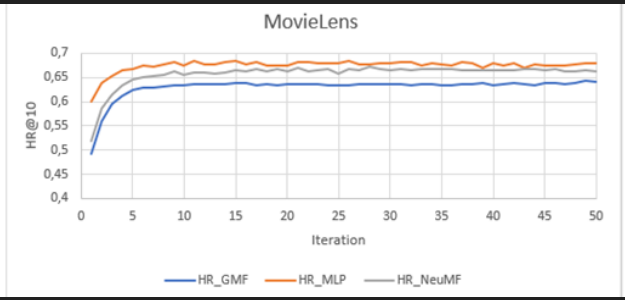


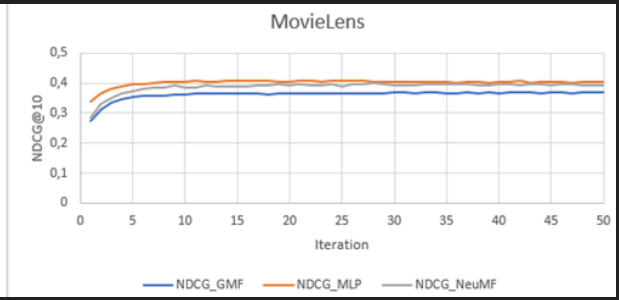
Eksperyment 2.1 - 50 iteracji



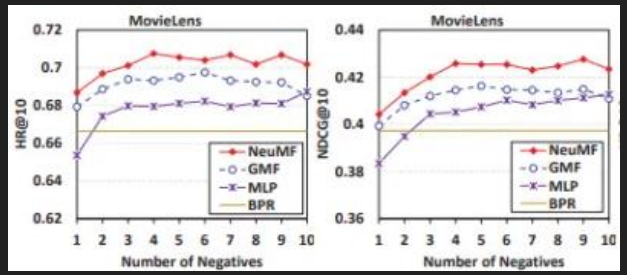
VS



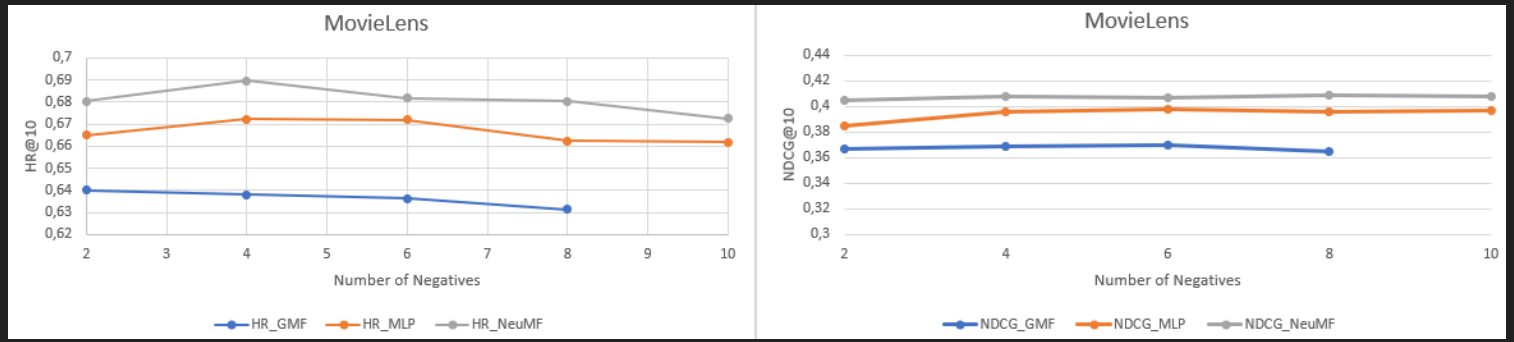




Eksperyment 2.2 - różne liczby negatywnych próbek



VS



Wyniki - analiza

- w większości przypadków, uzyskane przez nas wyniki są zbliżone do oryginalnych wyników przedstawionych w artykule

- uzyskane przez nas wyniki są w przypadku wszystkich eksperymentów nieco gorsze od oryginalnych

- może to wynikać z:
  - innych wartości pozostałych hiperparametrów (nie podanych przez autorów artykułu)
  - mniejszej liczbie epok treningowych niż w oryginalnych eksperymentach (nie podanych przez autorów artykułu)

Wnioski

- Algorytm przedstawiony został zaimplementowany poprawnie i w dużej mierze zgodnie z opisem w artykule
- W artykule były jednak opisane pewne funkcje, które nie zostały zaimplementowane w kodzie (m.i.n. leave one out cross-validation)
- Jeżeli chodzi o opis eksperymentów brakowało informacji o wartościach wszystkich hiperparametrów, a nie tylko tego poddawanego danemu eksperymentowi
- W celu ulepszenia eksperymentu można przykładowo spróbować:
    - przetestować modyfikacje innych hiperparametrów, które nie były modyfikowane w przedstawionych eksperymentach
    - zastosować regularyzację Lasso do kwadratowej funkcji straty, aby zminimalizować ryzyko uzyskania dużej wariancji parametrów wyniku. Przykładowa zmodyfikowana funkcja kosztu:

$$L_{lasso}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m}|\hat{\beta}_j|.$$

Źródła

1. https://arxiv.org/pdf/1708.05031v2.pdf
2. https://github.com/hexiangnan/neural_collaborative_filtering
3. https://towardsdatascience.com/neural-collaborative-filtering-96cef1009401
4. https://chat.openai.com/

Dziękujemy za uwagę