Лабораторная работа 5. Рекомендательные системы

Рекомендательные системы – подвид систем машинного обучения, основной целью которых является формирование рекомендаций – персонализированных выборок данных, основывающихся на информации о пользователе, в том числе:

- маркетинговом профиле (возраст, пол, место жительства, трудовая деятельность и т.д.);
- указанных напрямую тем, которые интересуют пользователя;
- запросах пользователя на поиск;
- пользовательской активности (просмотр страницы, установка приложения и т.п.);
- обратной связи от пользователя («лайк», рейтинг, отзывы и т.п.).

Основными этапами построения рекомендательных систем являются выборка кандидатов, оценка и повторное ранжирование. Выборка кандидатов осуществляется на основе фильтрации по содержимому, фильтрации по коллаборации или нейросетевой фильтрации.

В лабораторной работе предлагается построить выборку кандидатов с помощью фильтрации по коллаборации с использованием нейросетевых моделей глубокого обучения. В отличие от фильтрации по содержимому, фильтрация по коллаборации позволяет формировать рекомендации также на основании интересов похожих друг на друга пользователей. Для создания матрицы обратной связи, отражающей заинтересованность пользователей в различных продуктах, можно использовать функцию `pivot_table` библиотеки pandas:

```
import pandas as pd
util_df=pd.pivot_table(data=df, values='rating', index='userId', columns='movieId')
```

В переменной df в данном случае записан исходный датасет, в котором каждая строка соответствует отдельной оценке пользователя некоторого элемента. Матрица обратной связи строится таким образом, что значение колонки `'rating'`, становится значением в ячейке матрицы, а колонки `'userId'` и `'movieId'` ставятся в соответствие строкам и столбцам матрицы, соответственно. Пример матрицы приведен на рисунке 1. Ячейки, для которых рейтинги пользователей отсутствовали, приняли значения NaN. Заменить эти значения на 0 можно с помощью функции:

```
util_df.fillna(0)
```

movieId	0	1	2	3	4	5	6	7	8	9	...
userId											
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...
4	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
...	...	...	...	...	...	...	...	...	...	...	...

Рисунок 1. Пример матрицы обратной связи (выделено красным: пользователь с индексом 2 оценил фильм с индексом 4 на оценку/рейтинг 4)

Для решения задачи факторизации, то есть представления матрицы обратной связи в виде 2 множителей меньшей размерности (которые часто называются «вложениями»), в данной лабораторной работе предлагается использовать модель, построенную на основе библиотеки Keras. Для начала необходимо разделить датасет на обучающую и тестовую выборку:

```
users = df.userId.unique()
movies = df.movieId.unique()

userid2idx = {o:i for i,o in enumerate(users)}
movieid2idx = {o:i for i,o in enumerate(movies)}
df['userId'] = df['userId'].apply(lambda x: userid2idx[x])
df['movieId'] = df['movieId'].apply(lambda x: movieid2idx[x])

train,test=train_test_split(df[['userId','movieId']],df[['rating']],te
st_size=0.20,shufle=True)
```

В данном примере используются 2 вложения, представляющие пользователей и элементы, которые они оценивают. Размерность новых векторов вложений можно задать произвольной, в примере ниже используется значение `n_latent_factors=64`. В данном случае строятся две параллельные нейронные сети, результаты работы которых будут представлять полученные «вложения» пользователя и отдельного элемента данных, после чего для их агрегации используется скалярное произведение. Поскольку такая сеть имеет нелинейную структуру, она может быть задана с использованием функционального API Keras на основании классов Input и Model. В функциональном API каждый слой модели может быть вызван как функция,

аргументом которой является входы с предыдущего слоя. Обратите внимание также на использование специального слоя dot, который реализует операцию скалярного произведения.

```
from keras.layers import Dropout, Dense, Flatten, Activation,
Input, Embedding
from keras.layers.merge import dot
from keras.models import Model

n_movies=len(df['movieId'].unique())
n_users=len(df['userId'].unique())

user_input=Input(shape=(1,),name='user_input',dtype='int64')
user_embedding=Embedding(n_users,n_latent_factors,name='user_emb
edding')(user_input)
user_vec =Flatten(name='FlattenUsers')(user_embedding)
movie_input=Input(shape=(1,),name='movie_input',dtype='int64')
movie_embedding=Embedding(n_movies,n_latent_factors,name='movie_
embedding')(movie_input)
movie_vec=Flatten(name='FlattenMovies')(movie_embedding)
sim=dot([user_vec,movie_vec],name='Simalarity-Dot-Product',
axes=1)
model = Model([user_input, movie_input],sim)
model.compile(loss='mse')
```

Обучение модели:

```
History = model.fit([train.userId,train.movieId],train.rating,
batch_size=128, epochs = 50, validation_data =
([test.userId,test.movieId],test.rating), verbose = 1)
```

Для оценки производительности модели результаты ее обучения можно отобразить в виде графика:

```
from pylab import rcParams
rcParams['figure.figsize'] = 10, 5
import matplotlib.pyplot as plt
plt.plot(History.history['loss'] , 'g')
plt.plot(History.history['val_loss'] , 'b')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.grid(True)
plt.show()
```

В рассмотренном примере вся обработка заключалась в получении новых представлений и объединении их на основе операции скалярного произведения, однако, этот процесс можно сделать менее тривиальным и использовать в качестве выходной метрики значение, подобранное на основе нейросетевого анализа. Архитектура нейронной сети может быть задана в данном случае следующим образом:

```
user_input=Input(shape=(1,),name='user_input',dtype='int64')
user_embedding=Embedding(n_users,n_latent_factors,name='user_emb
edding')(user_input)
user_vec=Flatten(name='FlattenUsers')(user_embedding)
user_vec=Dropout(0.40)(user_vec)

movie_input=Input(shape=(1,),name='movie_input',dtype='int64')
movie_embedding=Embedding(n_movies,n_latent_factors,name='movie_
embedding')(movie_input)
movie_vec=Flatten(name='FlattenMovies')(movie_embedding)
movie_vec=Dropout(0.40)(movie_vec)

sim=dot([user_vec,movie_vec],name='Simalarity-Dot-Product',
axes=1)
nn_inp=Dense(96,activation='relu')(sim)
nn_inp=Dropout(0.4)(nn_inp)
nn_inp=Dense(1,activation='relu')(nn_inp)
nn_model = Model([user_input, movie_input],nn_inp)
nn_model.summary()
```

Задание к лабораторной работе 5.

1. Создайте рекомендательную систему для предложенного датасета двумя предложенными способами. Датасет: https://www.kaggle.com/datasets/rajmehra03/movielens100k
2. Продемонстрируйте, что система работает корректно.
3. Измените параметры моделей из примера для получения лучшего результата.
4. С помощью информации, приведенной в 4 и 5 лабораторных работах, а также в документации, поясните, что делает каждый из созданных вами слоев.
5. Представьте ваше исследование в виде Блокнота JupyterLab, содержащего все пункты задания.