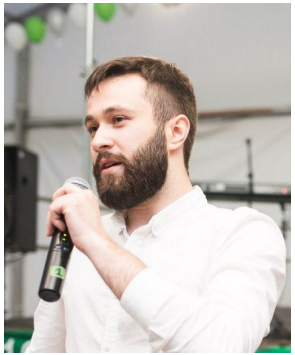
Your First RecSys

Даниил Потапов
Руководитель группы персонализации и рекомендательных систем
МТС BigData



Коллаборативная фильтрация

План

- Основная идея
- SVD
- implicit
  - kNN методы
  - iALS
- LightFM
  - WARP

Основная идея

Получить векторные представления пользователей и объектов на основе данных об их взаимодействиях.

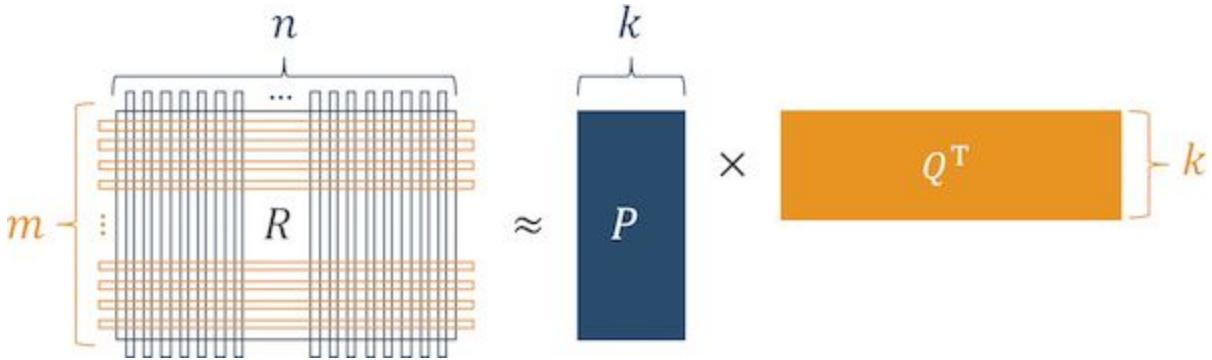
Функция оценки релевантности для пар (пользователь, объект)

$$f_{ui} : Users \times Items \rightarrow Relevance$$

Вектора получаем за счет фиксирования какой-либо функции оценки и решения задачи оптимизации

$$L(R, \hat{R}) \rightarrow min$$

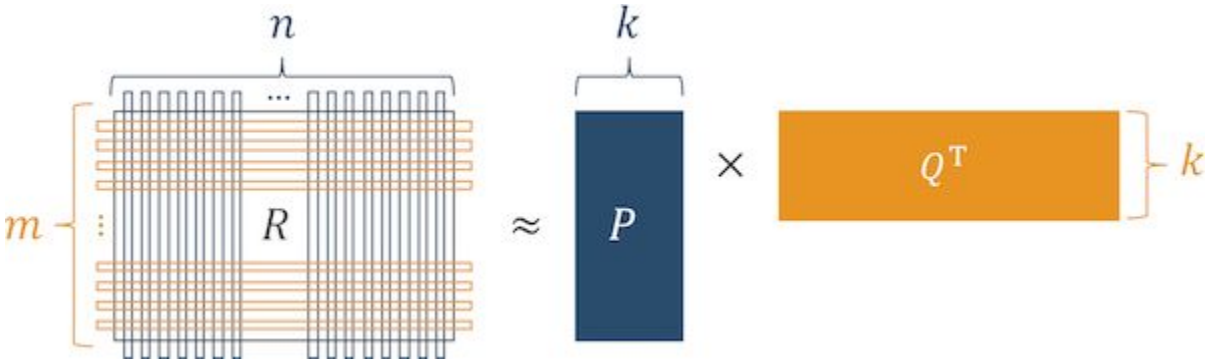
Основная идея



R - матрица взаимодействий
P - матрица векторов пользователей
Q - матрица векторов объектов
k - размерность векторного представления

image source

Основная идея

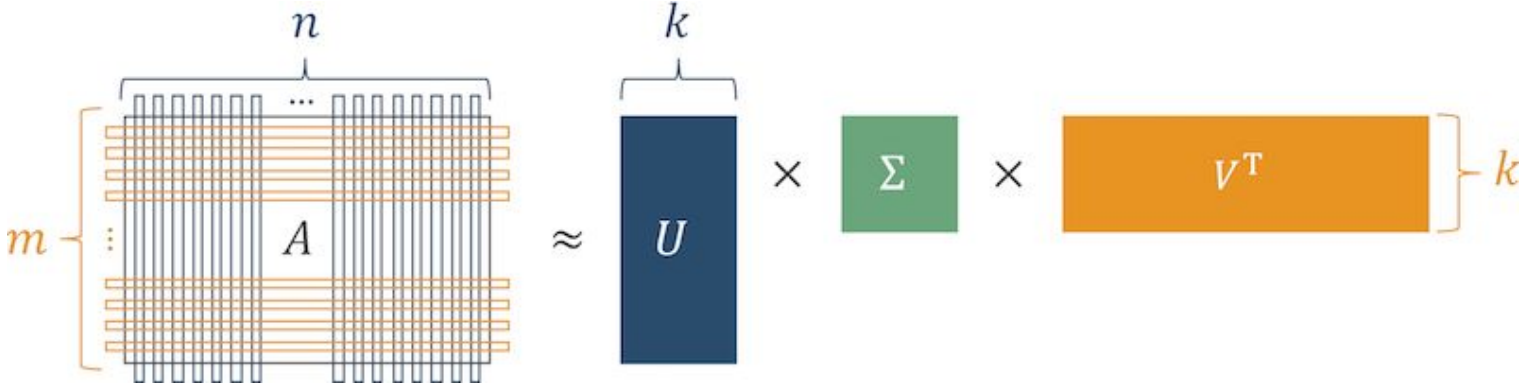


Функция оценки $$\hat{r}_{ui} = \mathbf{p}_u \mathbf{q}_i^T = \sum_d p_{ud} q_{di}$$

Функция ошибки $$L = \sum_{u,i \in S} (r_{ui} - \mathbf{p}_u \mathbf{q}_i^T)^2 + \lambda \left(\sum_u \|\mathbf{p}_u\|^2 + \sum_i \|\mathbf{q}_i\|^2 \right)$$

image source

SVD



U, V - ортогональные матрицы

Σ - диагональная (сингулярные числа)

$$A = U\Sigma V^T$$

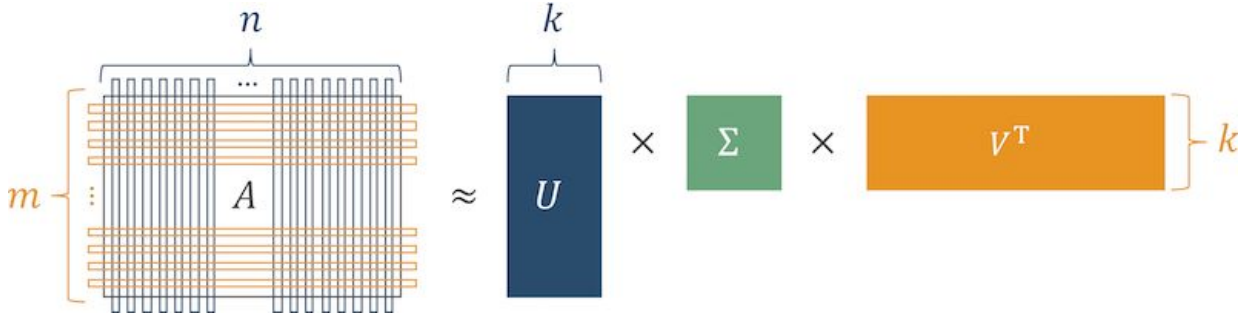
$$U \in \mathbb{R}^{m\times m}$$
$$V \in \mathbb{R}^{n\times n}$$
$$\Sigma \in \mathbb{R}^{m\times n}$$

$$A \approx U_k\Sigma_k V_k^T$$

$$U_k \in \mathbb{R}^{m\times k}$$
$$V_k \in \mathbb{R}^{n\times k}$$
$$\Sigma_k \in \mathbb{R}^{k\times k}$$

image source

SVD



$$\|A - \hat{R}\|_F^2 \to min$$

$$\|X\|_F^2 = \sum_{i,j} x_{ij}^2$$

$$\hat{R} = U_k \Sigma_k V_k^T$$

$$AV_kV_k^T = U\Sigma V^T V_k V_k^T$$

$$AV_kV_k^T = U_k\Sigma_k V_k^T = \hat{R}$$

Пропущенные значения заполняются или интерпретируются как 0

image source

SVD

Особенности:

- Не так хорошо для предсказания значений (в качестве регрессии)
- Подходит для построения топ-N рекомендаций
  - Для каждого пользователя считаем скалярное произведение со всеми объектами
  - Выбираем топ объектов по полученным значениям

Много эффективных реализаций на разных языках

- from scipy.sparse.linalg import svds
- from sklearn.utils.extmath import randomized_svd
- from sklearn.decomposition import TruncatedSVD

implicit

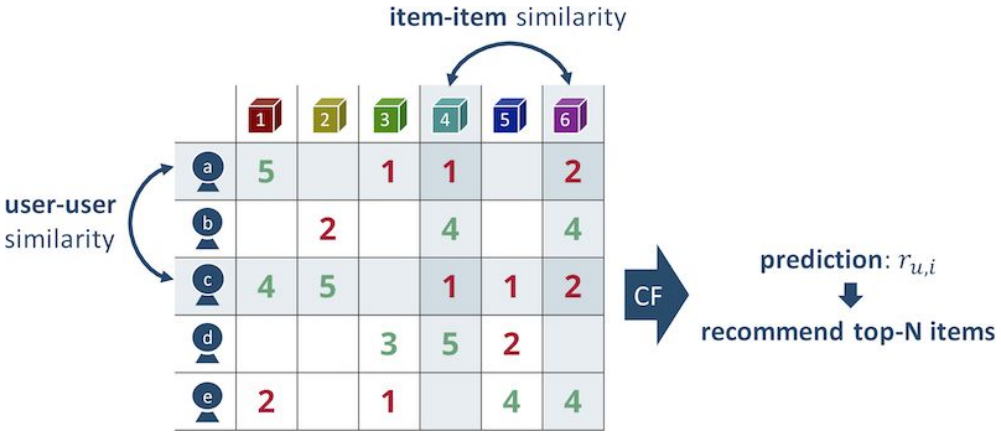
Github - https://github.com/benfred/implicit
Docs - https://implicit.readthedocs.io/en/latest/

Основная фишка - построение моделей на основе неявного таргета (implicit datasets).

Доступные модели:

- Item-to-item kNN
- Logistic matrix factorization
- implicit ALS
- Bayesian Personalized Ranking

implicit kNN



Основная идея:

- В качестве векторных представлений для пользователей и объектов использовать строки и столбцы из матрицы взаимодействий
- С помощью векторов оцениваем схожесть и на основе нее строим рекомендации
  - Косинусное расстояние
  - Корреляция Пирсона

image source

implicit kNN

В implicit доступны три модели и они все Item-ориентированы:

- CosineRecommender
- BM25Recommender
- TFIDFRecommender

Гиперпараметр один для всех - *K*, число соседей. Он влияет на топ, который может выдать модель.

Процесс обучения - поиск *K* ближайших соседей для каждого item и сохранение схожести с ними.

implicit kNN

Построение рекомендаций для пользователя:

- Получение топ-K соседей для каждого объекта, с которым пользователь взаимодействовал
- Объединение всех топов в один с суммированием схожести
- Выдача топ-N самых похожих объектов

implicit kNN

Ключевые методы

- fit
  - *item_user_matrix* - разреженная матрица со взаимодействиями
    Важно именно в такой ориентации подать матрицу (matrix.T)
- recommend
  - *user_id* - номер строки, соответствующий пользователю, для которого строим рекомендации
  - *user_item_matrix* - разреженная матрица с нашими взаимодействиями
  - *N* - топ рекомендаций
  - *filter_already_liked_items* - флаг, для исключения уже известных объектов (*user_item_matrix*)
  - *filter_items* - список столбцов (объектов), которые надо исключить
  - Возвращает список кортежей (tuple) - (столбец, схожесть)

implicit kNN

Обучение модели

```
cosine_model = CosineRecommender(K=10)
cosine_model.fit(train_mat.T) #
```

```
100% |████████████████████████████████| 59599/59599 [00:02<00:00, 22803.72it/s]
```

Построение рекомендаций для одного пользователя

```
recs = cosine_model.recommend(row_id, train_mat, N=top_N, filter_already_liked_items=True)
recs = pd.DataFrame(recs, columns=['col_id', 'similarity'])
recs
```

	col_id	similarity
0	4341	0.297014
1	7353	0.220847
2	36593	0.215622
3	3802	0.188025
4	51215	0.145095
5	49085	0.128586
6	37852	0.102340
7	7873	0.101929
8	46769	0.100504
9	56270	0.100504

implicit kNN

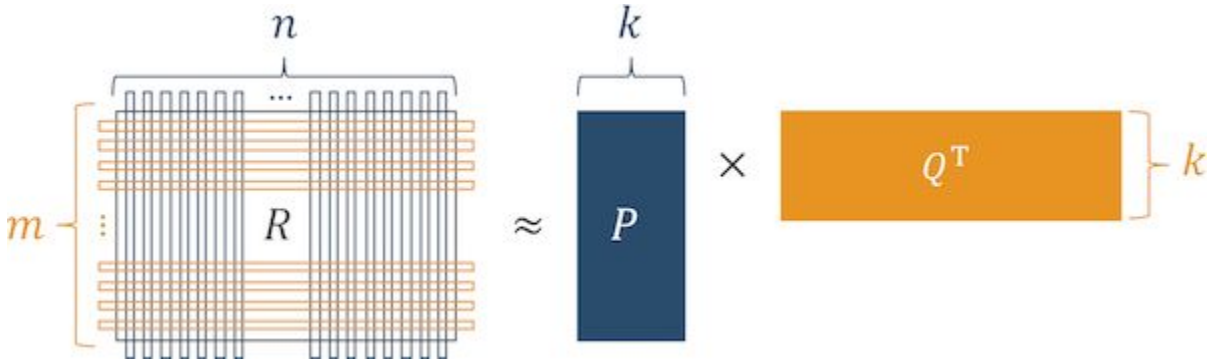
Также есть метод **similar_items**, который выдает топ похожих объектов

- item_id - объект (номер столбца), для которого хотим получить похожие
- N - запрашиваемый топ

```
get_similar_books('пикник на обочине', model)
```

	col_id	similarity	item_id	title
0	23397	1.000000	235407	пикник на обочине
1	8237	0.081634	208935	понедельник начинается в субботу
2	13604	0.075878	86572	град обреченный
3	46038	0.053919	85633	жук в муравейнике
4	9537	0.048289	128823	обитаемый остров
5	11830	0.046667	85653	волны гасят ветер
6	23062	0.043113	138608	сталкер
7	40889	0.038316	287365	большой прикол. анекдоты 31-2016
8	39976	0.035602	94631	трудно быть богом
9	1992	0.031722	35265	записки юного врача

implicit iALS



Функция ошибки $L = \sum_{u,i} c_{ui}(r_{ui} - p_u q_i^T)^2 + \lambda(\sum_u \|p_u\|^2 + \sum_i \|q_i\|^2)$

c_{ui} - степень уверенности (не меньше 1)

r_{ui} - 1/0, факт взаимодействия

image source

МТС

implicit iALS

$$L = \sum_{u,i} c_{ui}(r_{ui} - p_u q_i^T)^2 + \lambda\Big(\sum_u \|p_u\|^2 + \sum_i \|q_i\|^2\Big)$$

$$L = \|C * (R - PQ^T)\|_F^2 + \lambda(\|P\|_F^2 + \|Q\|_F^2)$$

Идея ALS в следующем: фиксируем одну из матриц и оптимизируем другую

$$\frac{\partial L}{\partial p_u} = 0 \Rightarrow p_u = (Q^T Q + Q^T (C^u - I) Q + \lambda I)^{-1} Q^T C^u r_u$$

$$\frac{\partial L}{\partial q_i} = 0 \Rightarrow q_i = (P^T P + P^T \left(C^i - I\right) P + \lambda I)^{-1} P^T C^i r_i$$

implicit iALS

implicit - переход от предсказания значений к предсказанию релевантности, учитывая веса

ALS - метод оптимизации, в котором мы поочередно фиксируем одну матрицу и делаем оптимизационный шаг по другой

Функционал класса идентичен тому, что мы рассматривали у kNN.

Входная матрица интерпретируется как матрица весов c_{ui}

Итоговые предсказания (значения) интерпретации не имеют, они должны использоваться только для ранжирования

implicit iALS

Ключевые гиперпараметры

- *factors* - размерность вектора
  - обычно в районе 16 - 256
  - при использовании GPU должны быть вида 32*N
- *iterations* - кол-во итераций (1 итерация - проход по P и Q)
  - 10 - 200
- *regularization* - регуляризация
  - 0.00001 - 1
- *use_gpu*
- *random_state*

fit метод при повторном вызове делает еще iterations на основе уже имеющихся векторов

implicit iALS

Также в этой модели появляется возможность сделать предсказания для пользователей, которых не было при обучении, но для которых известны несколько взаимодействий.

Это делается через параметр *recalculate_user* в методе recommend. С этим флагом вектор пользователя будет пересчитан на основе матрицы векторов объектов ровно как при обучении самого iALS

$$p_u = \left(Q^T Q + Q^T \left(C^u - I\right) Q + \lambda I\right)^{-1} Q^T C^u r_u$$

LightFM

Github - https://github.com/lyst/lightfm
Docs - https://making.lyst.com/lightfm/docs/home.html

Основная фишка - построение векторов для фичей пользователя и объекта.
В качестве векторных представлений самих пользователей и объектов берется сумма векторов их фичей

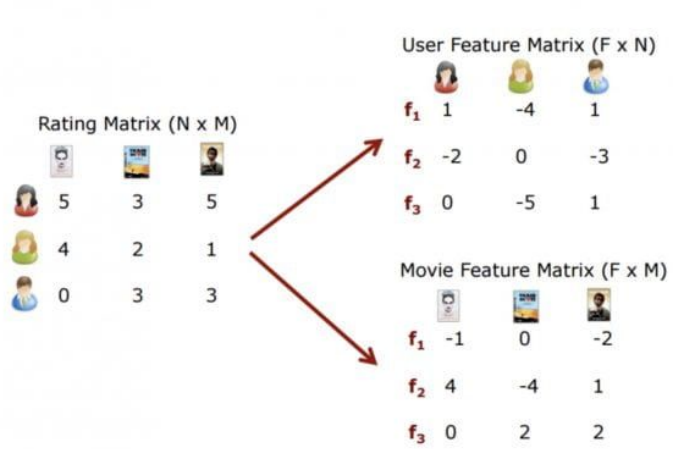


image source

LightFM

Модель одна, с разными loss:

- Logistic
- Bayesian Personalized Ranking
- Weighted Approximate-Rank Pairwise
- k-os WARP

Обучается с помощью SGD. Доступны две вариации:

- adagrad
- adadelta

LightFM Dataset

lightfm.data.Dataset - класс, помогающий собрать разреженные матрицы в нужном виде.

Что требуется на вход (*fit, fit_partial*):

- users - ID пользователей
- items - ID объектов
- user_features - имена фичей пользователей
- item_features - имена фичей объектов

По умолчанию включено построение индикаторных фичей пользователей и объектов.

МТС

LightFM Dataset

`build_interactions` - построение матрицы взаимодействий на основе итератора на список кортежей одного из следующих видов:

- (user_id, item_id)
- (user_id, item_id, weight)

`build_user_features/build_item_features` - построение матрицы фичей на основе итератора на список кортежей одного из следующих видов:

- (user_id, [user_feature_name1, user_feature_name2, ...])
- (user_id, {user_feature_name1: weight})

LightFM Dataset

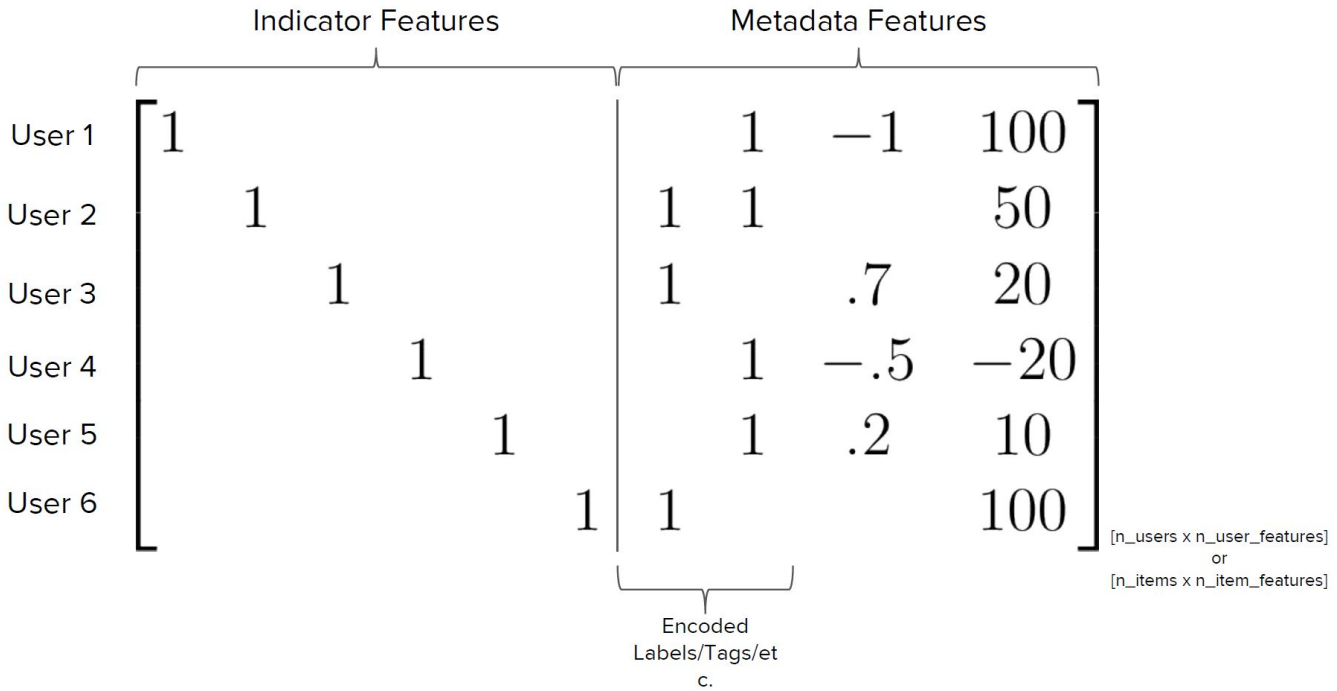


image source

LightFM WARP

Основная идея - переход от проверки качества предсказания значения к предсказанию качества ранжирования.

Достигается это путем сэмплирования негативных примеров и построения функции ошибки на основе соотношения предсказанных значений.

Алгоритм обучения:

1. Берем пользователя *u*
2. Выбираем для него объект *i*, с которым он взаимодействовал
3. Выбираем случайный объект *j*, с которым он не взаимодействовал
4. Считаем скалярные произведения *p = (u, i)* и *n = (u, j)*
5. Сравниваем *p* и *n*
    a. Если p > n, то переходим на шаг 2 (или шаг 1)
    b. Иначе считаем ошибку, правим веса и возвращаемся на шаг 3.

Кол-во сэмплирований - гиперпараметр *max_sampled.*

МТС

LightFM WARP

Pos, Neg - множества позитивных и негативных примеров

$$L_u = \sum_{i \in Pos} L\big(rank(u,i)\big)$$

$$rank(u,i) = \sum_{j \notin Pos} I\big(f(u,i) \geq f(u,j)\big)$$

I - индикаторная функция

$\langle , \rangle$ - скалярное произведение

b - смещения (глобальное, по user, по item)

$$f(u,i) = b_g + b_u + b_i + \langle p_u , q_i \rangle$$

$$L\big(rank(u,i)\big) = \sum_{j \notin Pos} log\big(\lfloor \frac{|Neg|}{Sampled} \rfloor\big) \big|1 - \langle p_u , q_i \rangle + \langle p_u , q_j \rangle\big|_+$$

Sampled - сколько раз нам пришлось сэмплировать негативный пример, чтобы найти такой, для которого скалярное произведение будет больше, чем скалярное произведение с позитивным примером

$|\cdot|_+$ - Hinge loss (max(0, x))

LightFM Model

Ключевые гиперпараметры

- no_components - размерность вектора
- learning_rate - “шаг” при обучении в SGD
- user_alpha/item_alpha - регуляризация по векторам пользователей и объектов
- random_state

Есть loss-зависимые параметры, например для WARP:

- max_sampled - максимальное количество сэмплирований негативных примеров для одной позитивной пары (пользователь, объект). Увеличение приводит к росту качества, но замедлению обучения

LightFM Model

```
lfm_model = LightFM(no_components=64, learning_rate=0.05, loss='warp', max_sampled=5, random_st
ate=23)
```

```
num_epochs = 15
for _ in tqdm(range(num_epochs), total=num_epochs):
    lfm_model.fit_partial(
        train_mat,
        user_features=train_user_features,
        item_features=train_items_features,
        num_threads=4
    )
```

```
100% ████████████████████████████████ 15/15 [01:06<00:00, 4.45s/it]
```

LightFM Model

В отличии от implicit, построение предсказаний в lightfm находится в более сыром виде.

Имеющийся метод model.predict ожидает на вход пары (пользователь, объект) и возвращает массив предсказаний по ним. Никакой дополнительной фильтрации.

Алгоритм построения предсказаний для одного пользователя следующий:

- Получить оценку для всех объектов
  - model.predict
  - user_vector * item_embeddings_matrix
- Получить индексы самых больших скоров
  - np.argpartition(scores, -np.arange(N))[-N:]
- Перевести индекс в наши item_id

Попрактиковаться самим можно здесь:
https://www.kaggle.com/sharthz23/implicit-lightfm

Вопросы можно задавать как на самом kaggle, так и в
- Telegram чате курса
- sharthZ23