Data Scientist. Рекомендательные системы.

Брюханов Д. В.

План работы:

1. Постановка задачи, исходные данные и что с ними нужно сделать.
2. Подсчет топ 10 самых популярных товаров для каждого пользователя и оценка этого подсчета – как идеал, к которому мы будем стремиться.
3. Roadmap работы, какие подходы и какие библиотеки использовались.
    3.1 Иплисит ALS и kNN
    3.2 LightFM
    3.3 Нейросеть
4. Подведение итогов и вывод.

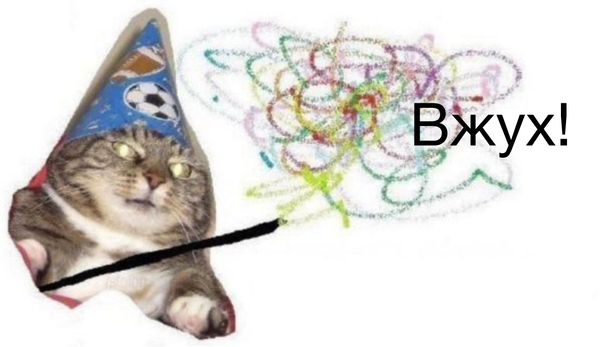
Постановка задачи:

Нам нужно разработать рекомендательную систему подбора товаров, исходя из индивидуальных особенностей каждого пользователя. Представим задачу графически.

Нам нужно:

Вот это

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order	product_id	add_to_cart_order	reordered
0	2539329	1	1	2	8	NaN	196	1.0	0.0
1	2539329	1	1	2	8	NaN	14084	2.0	0.0
2	2539329	1	1	2	8	NaN	12427	3.0	0.0
3	2539329	1	1	2	8	NaN	26088	4.0	0.0
4	2539329	1	1	2	8	NaN	26405	5.0	0.0



Превратить вот в это

	user_id	product_id
0	1	10326 10258 13032 13176 38928 41787 17122 1303...
1	2	17758 17872 47209 5907 32052 47766 48210 35917...
2	3	8021 42557 28373 1005 12845 18599 1005 22035 1...
3	7	42265 14332 9598 39121 33740 519 19678 29894 2...
4	13	42248 32850 1689 18168 18168 42248 41480 5652 ...

Получив при этом скор (MAP@10) >= 0,2094

Но, стремится мы будем сюда □ > 0,25

Roadmap экспериментов:

Разбор алгоритмов:

1. **«Наивный алгоритм»**

С ним все просто: мы группируем данные по пользователю и считаем кол-во купленных за всю историю продуктов. В результате будем иметь для каждого пользователя такую картину:

	product_id	amount
0	33754	17
1	21903	14
2	24838	14
3	17704	13
4	46667	13

Повторим эту операцию 100 000 раз (для каждого пользователя) и преобразовав к итоговому виду, получим скор 0.27841.

Это будет наш эталон, к нему мы будем стремиться, однако мы не можем назвать такой подход рекомендательной системой, ведь мы ничего нового не предлагаем пользователю, а принимаем в учет только те товары, про которые он сам прекрасно знает и, более того, любит, т к это его топ.

2. **Implicit ALS (Alternating Least Squares)**

Хорошая удобная библиотека, главная фишка заключается в том, что можно прямо «из коробки» передать топ из скольки товаров мы хотим порекомендовать каждому покупателю. Перед тем, как обучать алгоритм есть 2 главных параметра его настройки: т н факторы (factors) и итерации (iterations). От различных комбинаций этих двух параметров варьируется скор. Однако это изменение не столь существенное, если сравнивать его с влиянием, что оказывает подготовка данных: категориальное кодирование. (~0.006 против ~0.025). Было опробовано несколько комбинаций факторов и итераций. Оптимальными показали себя factors=30, iterations=8. Итоговый скор с ними 0.05377. Это существенно НЕ ДОТЯГИВАЕТ до заданных целей.

3. **Implicit kNN (k Nearest Neighbours)**

Метод из той же библиотеки, но основанный на другом принципе: он сравнивает величину с k ближайшими соседями для выявления сходств. Такой подход сработал лучше всего при 10 соседях, дав 0.12458 точности предсказаний. Однако до заданной планки это также существенно НЕ ДОТЯГИВАЕТ.

4. **LightFM**

Другая библиотека, также основанна на принципе разряженных матриц, однако при создании модели позволяет добавлять вспомогательные фичи для юзеров и айтемов, представляющих строки и столбцы такой матрицы. Кроме этого можно настраивать веса образцов, задавать функцию потерь и выбирать количество эпох обучения, совсем как в нейросетях. Система однако, может работать и по одной лишь матрице, только вот скор такая система по метрике MAP@10 дает наихудший, практически нулевой. Экспериментировать с ней дальше я не стал, не увидел смысла в добавлении фичей, т к вряд ли бы они улучшили скор на несколько порядков. Вердикт: НЕ ДОТЯГИВАЕТ.

Несколько слов о разряженных матрицах.

Ключевым элементов в работе вышеописанных библиотек является т н разряженная матрица (sparse matrix). Мне такую удобнее было всего получать при помощи библиотеки scipy.sparse. Чтобы получить эту самую матрицу, сначала нам требуется преобразовать входные данные. Делается это буквально в одну строчку. Нам нужно посчитать кол-во добавлений каждого товара в корзину для каждого пользователя. Выглядеть такая группировка будет следующим образом:

	user_id	product_id	add_to_cart_order
0	1	196	10
1	1	10258	9
2	1	10326	1
3	1	12427	10
4	1	13032	3
...	...	...	...
9459060	206209	43961	3
9459061	206209	44325	1
9459062	206209	48370	1
9459063	206209	48697	1
9459064	206209	48742	2

Далее закодируем значения этих столбцов с помощью cat.codes и создадим из них 2 разряженных матрицы: одну «айтем-юзер» и другую «юзер-айтем». Первую будем использовать для обучения алгоритма, а вторую для предсказаний. Можно так же транспонировать одну матрицу и использовать для того и другого, но лучше срабатывает первый метод.
После того как матрицы готовы остается только обучить на одной из них алгоритмы и после предсказать(порекомендовать) на другой требуемый топ товаров для желаемого пользователя или списка пользователей.

Звучит это просто и легко, но…

Интересная проблема

В начале все было нормально и библиотека ставилась с помощью !pip install implicit, но в один прекрасный день этот способ просто перестал работать

Клонирование с Github уменьшало точность в разы. Однако все работало и выполнялось точно так же

!sudo -H python3 -m pip install implicit --no-cache --force-reinstall --log ./implicit.txt – помогла только эта строчка! И тут я понял, что

Алгоритм на основе нейронной сети.

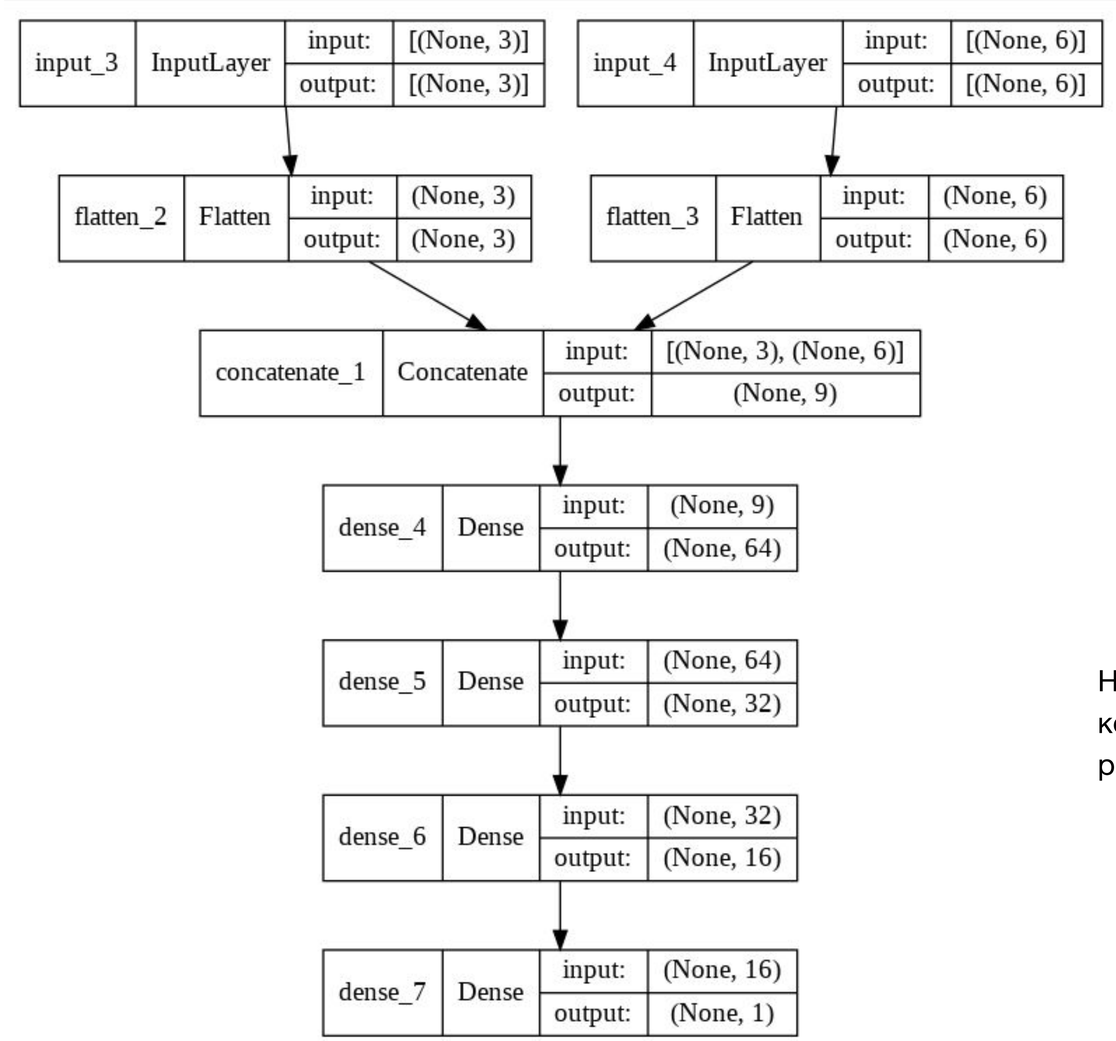
- В качестве вишенки на торте и чего-то по-настоящему рабочего было решено воспользоваться нейронной сетью.

- Исходный набор признаков был преобразован и дополнен, и в итоге получилось 3 признака для пользователей и 6 для продуктов. В качестве таргета был взят факт повторного заказа того или иного продукта конкретным юзером. Весь сет был разбит на трэйн и тест выборки по принципу разницы между первым и последним номера заказа. Если разница была больше 1, то такие строки уходили в трэйн, если меньше – то в тест.

- Идея была в том, чтобы по величине сырой сигмоиды после предсказаний судить о «силе» или весе того или иного продукта для конкретного пользователя, отсортировав все эти продукты по убыванию значения сигмоиды, можно было получить топ N. Не всегда это был топ 10, иногда больше, иногда меньше, ведь с некоторыми продуктами некоторые пользователи могли вовсе не взаимодействовать.

- Итоговый сет из 10 продуктов добивался случайным образом из исторического топ10 для каждого юзера. Также он добивался и рандомом из всех возможных продуктов, но добивать из топ10 была выгодней для скора приблизительно на 0.015. Повторы естественно исключались.

Архитектура нейросети:



2 входа отдельно для продуктовых и пользовательских признаков

Превращаем данные в одномерный тензор

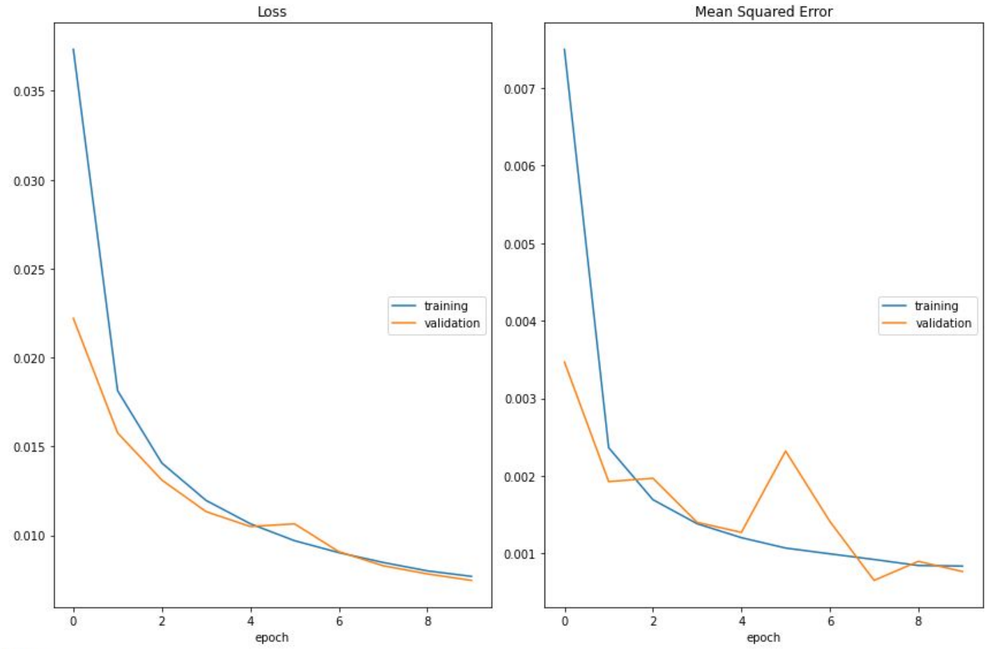
Объединение

Некоторый стек полносвязных слоев. Он менялся в ходе экспериментов. Менялось кол-во слоев, выходное пространство первого слоя и последующих, значение регуляризации l2, общего для всех слоев.

Интересные наблюдения.

Приведу две наиболее удачные конфигурации сетей: одна дала наилучшие финальные рекомендации, другая наивысшую точность предсказаний (accuracy):

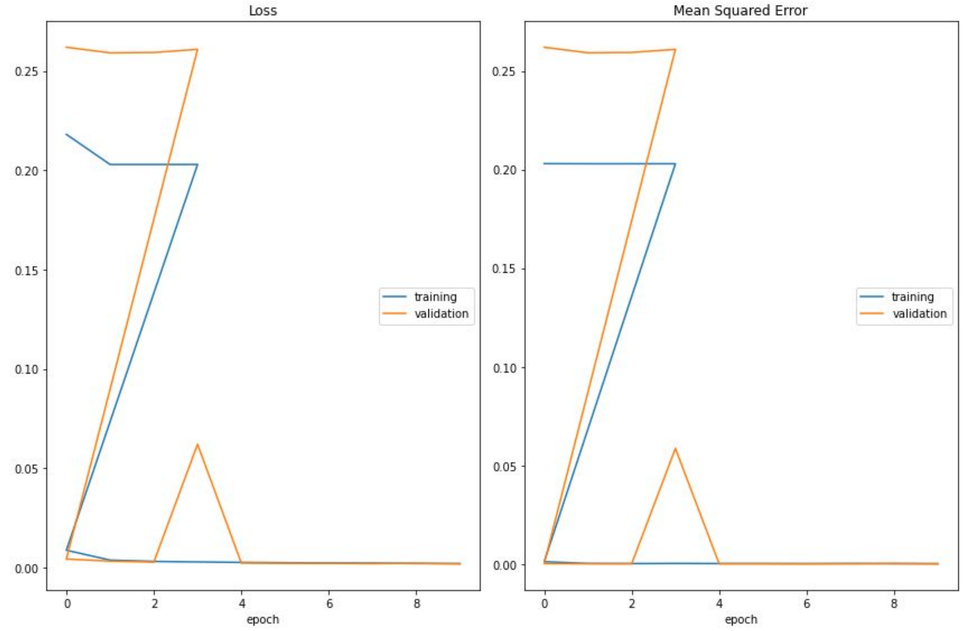
Лучшие предсказания (MAP@10 - 0.27080, accuracy – 0.70)




```
Loss
       training              (min:    0.008, max:    0.037, cur:    0.008)
       validation            (min:    0.007, max:    0.022, cur:    0.007)
Mean Squared Error
       training              (min:    0.001, max:    0.007, cur:    0.001)
       validation            (min:    0.001, max:    0.003, cur:    0.001)
56702/56702 [==============================] - 256s 5ms/step - loss: 0.0077 - mse: 8.3894e-04 - val_loss: 0.0075 - val_mse: 7.6820e-04
```

Архитектура показана на пред. слайде, Adam, lr=1e-3, reg. l2=0.01

Лучшая точность (MAP@10 - 25571, accuracy - 0.97)




```
Loss
       training              (min:    0.002, max:    0.218, cur:    0.002)
       validation            (min:    0.002, max:    0.262, cur:    0.002)
Mean Squared Error
       training              (min:    0.000, max:    0.203, cur:    0.000)
       validation            (min:    0.000, max:    0.262, cur:    0.000)

Epoch 00010: val_mse did not improve from 0.00009
56702/56702 [==============================] - 251s 4ms/step - loss: 0.0020 - mse: 2.2548e-04 - val_loss: 0.0018 - val_mse: 1.1837e-04
```

4 dense(128, 64, 32, 16), Adam, lr=1e-3, reg. l2=0.001

Некоторые технические моменты.

Вид итогового датафрейма, отсортированного по величине выхода сигмоиды:

	user_id	product_id	predictions
0	1	196	0.975069
1098	1	12427	0.974666
1309	1	10258	0.970747
1408	1	25133	0.967029
1590	1	46149	0.929090
...	...	...	...
692610	206209	31477	0.695112
844490	206209	38730	0.683582
620478	206209	6567	0.670001
885094	206209	22920	0.654735
68584	206209	14197	0.507085

1079141 rows × 3 columns

Как дополнялись предсказания:

```
swap =  random.choices(top_10.tolist()[i], k=10)

for i in fetch.index:
  while len(fetch['product_id'][i]) != 10:
    if len(fetch['product_id'][i]) < 10:
      fetch['product_id'][i].append([x for x in swap if x not in fetch[
'product_id'][i]]) # здесь исключаются повторы
    elif len(fetch['product_id'][i]) > 10:
      fetch['product_id'][i].remove(fetch['product_id'][i][-1]) # удаля
ем последний (наименее популярный элемент)
```

Набор списка из 10 продуктов для пользователя:

```
fetch = rec_df.groupby('user_id')['product_id'].apply(list).reset_index()
```

Вывод:

- Исходя из опыта работы с Implicit создается впечатление, что это просто не под Windows, минимум под Docker, хотя Colab теоретически должно быть все равно, однако же работало это ниже ожидаемого уровня по данной задаче. kNN должен был показать 0.2, судя по опыту решения этой задачи на просторах интернета.
- LightFM-у как минимум нужны фичи, но проблем с установкой не было, разве что набирать предсказания не так удобно, как в Implicit.
- Ну а нейросеть в очередной раз показала себя как ультимативный универсальный оптимизатор, однако пайплайн с ней самый нетривиальный, я бы даже сказал изысканный, и подготовка данных самая долгая.
- Поэтому подводя итог, хочется сказать что: если важна скорость и допустимо сделать продукт «на коленке», то выбор падает однозначно на Implicit, если же важна точность – то тогда нейросеть.

Спасибо за внимание!