Гордонов Никита, Ишмуратов Сергей, Каминский Дмитрий, Карасёв Дмитрий

Финальный проект

Deep Learning RecSys

Описание задачи

В нашем проекте мы хотим улучшить результаты по построению рекомендательной системы, полученные в последнем семинаре и домашнем задании, показанные baseline моделями knn и логистическая регрессия. Мы предполагаем, что паттерны в данных описываются более сложной зависимостью, которую не могут найти такие простые модели, и хотим попробовать использовать нейросеть для улучшения результатов, полученных в домашнем задании. Таким образом, перед нами с одной стороны стоит предсказательная задача - мы хотим улучшить метрики, а с другой стороны исследовательская задача - мы проверяем гипотезу о более сложной природе зависимости в данных. Чтобы выполнить вторую задачу, мы приняли решение работать только с user-recipe датасетами, не используя списки ингредиентов, чтобы наша модель обучалась на том же наборе данных, что и модели в домашнем задании, и мы могли объективно сравнить показанные ими результаты. Это важная исследовательская задача, поскольку предсказательные системы повсеместно используются в интернет-продуктах и от качества предоставляемых рекомендаций зависят прибыли компаний. При этом у компаний всегда есть trade-off между качеством модели и ее интерпретируемостью, поэтому мы хотим проверить, насколько deep-learning модель, которая плохо интерпретируется, сможет улучшить целевые метрики.

Архитектура модели.

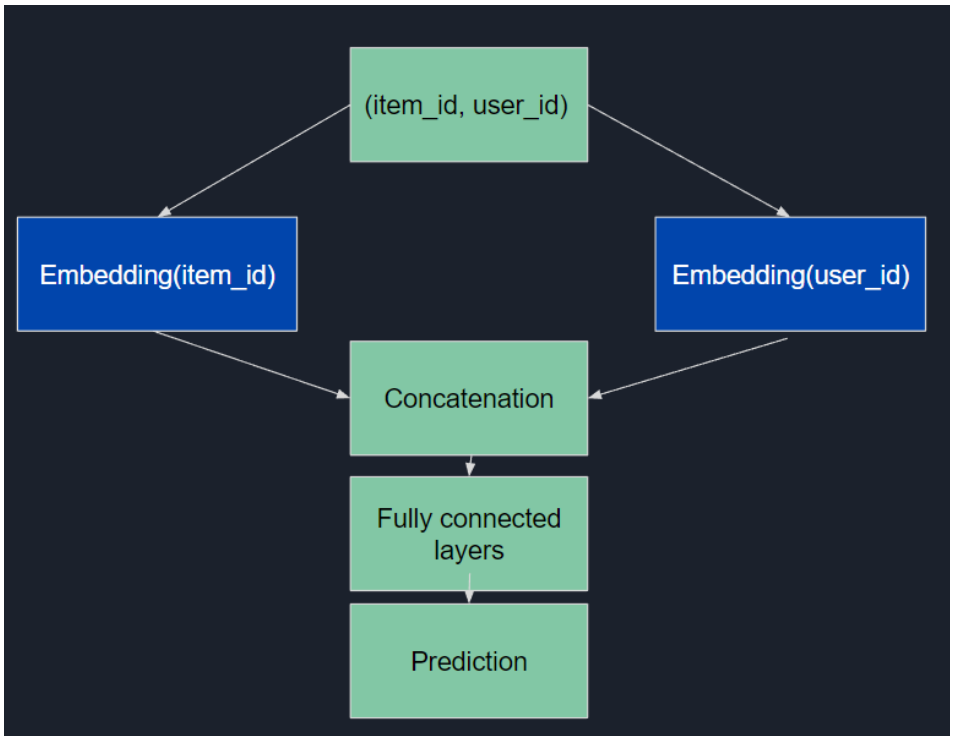
В ходе выбора архитектуры нашей модели мы изучили статьи[1] о современных подходах к построению рекомендательных моделей с помощью глубокого обучения и посмотрели архитектуры моделей, показывающих высокие результаты в соревнованиях на kaggle[2]. В результате, мы остановили свой выбор на FM-семействе моделей (Factorization Machine based Neural Network), которые используют эмбеддинги для представления user и item, а уже передают эти результаты в обычную нейронную сеть. Поскольку для реальных проектов используются сложные и многослойные сети, в которых сложно разбираться и которые сложно перенастраивать, мы решили использовать упрощенный прототип подобной архитектуры, который выполнен с помощью Keras[3]. В модели используется следующая архитектура:

Layer (type)	Output Shape	Param #	Connected to
user (InputLayer)	[(None, 1)]	0	[]
recipe (InputLayer)	[(None, 1)]	0	[]
user_embedding (Embedding)	(None, 1, 15)	15000	['user[0][0]']
item_embedding (Embedding)	(None, 1, 25)	25000	['recipe[0][0]']
reshape_94 (Reshape)	(None, 15)	0	['user_embedding[0][0]']
reshape_95 (Reshape)	(None, 25)	0	['item_embedding[0][0]']
concatenate_47 (Concatenate)	(None, 40)	0	['reshape_94[0][0]', 'reshape_95[0][0]']
dense_94 (Dense)	(None, 128)	5248	['concatenate_47[0][0]']
dense_95 (Dense)	(None, 1)	129	['dense_94[0][0]']

Total params: 45,377
Trainable params: 45,377
Non-trainable params: 0

[1] https://towardsdatascience.com/modern-recommender-systems-a0c727609aa
[2] https://www.kaggle.com/code/morrisb/how-to-recommend-anything-deep-recommender
[3] https://keras.io/api/layers/

Можно упрощенно представить ее с помощью блок схемы:



Индексы user и item поступают каждый в свой эмбеддинг слой, где они переводятся в многомерный вектор (мы предполагаем, что координаты этого вектора несут в себе какие-то скрытые признаки в данных). Затем полученные векторы объединяются с помощью простой конкатенации и передаются в полносвязный слой с ReLU-активацией. Наконец, на основе выхода этих слоев мы делаем предсказание. Отметим, что проблемой такого подхода является необходимость переучивать нейросеть при добавлении полностью новых пользователей или рецептов, однако мы вполне можем сочетать deep learning и, например, knn подходы, делая предсказания для новых пользователей на основе предсказаний deep learning модели для старых пользователей, не переучивая нейросеть.

Настройка гиперпараметров модели и математический аппарат.

Для настройки гиперпараметров модели мы использовали Grid Search, тестируя различные комбинации размерностей исходящих векторов для эмбеддингов и размерности для полносвязного слоя. Всего проверено 18 возможных комбинаций, по 3 эпохи обучения на каждую комбинацию. Лучшая модель затем обучалась 20 эпох. Кроме того мы попробовали использовать вместо MSE cross-entropy loss, однако он показывал сильную тенденцию к предсказанию всех пропущенных значений нулями, поэтому мы отказались от него в пользу MSE.

$$MSE = \frac{1}{N}\sum_{i=0}^{N}(\hat{y}_i - y_i)^2$$

Было выявлено, что оптимизатор ADAM работает лучше, чем классический для таких моделей стохастический градиентный спуск, поэтому мы решили использовать именно его. ADAM работает не только со значениями градиента, но и с его моментами, за счет чего получается лучшее качество. Принцип работы ADAM (Adaptive moment estimation):

$$First\ moment:\ m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$
$$Second\ moment:\ v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$
$$m_0 = v_0 = 0, \qquad usually\ \ \beta_1 = 0.9, \beta_2 = 0.999$$
$$We\ want: \quad E(m_t) = E(g_t)\ \ and\ \ E(v_t) = E(g_t^2)$$
$$But\ \ E(m_t) = E((1-\beta_1)\Sigma_{i=1}^{t}\beta_1^{t-i}g_i) = E(g_i)(1-\beta_1^t) + \epsilon$$
$$The\ estimator\ is\ biased,\ so\ we\ need\ to\ make\ the\ correction$$

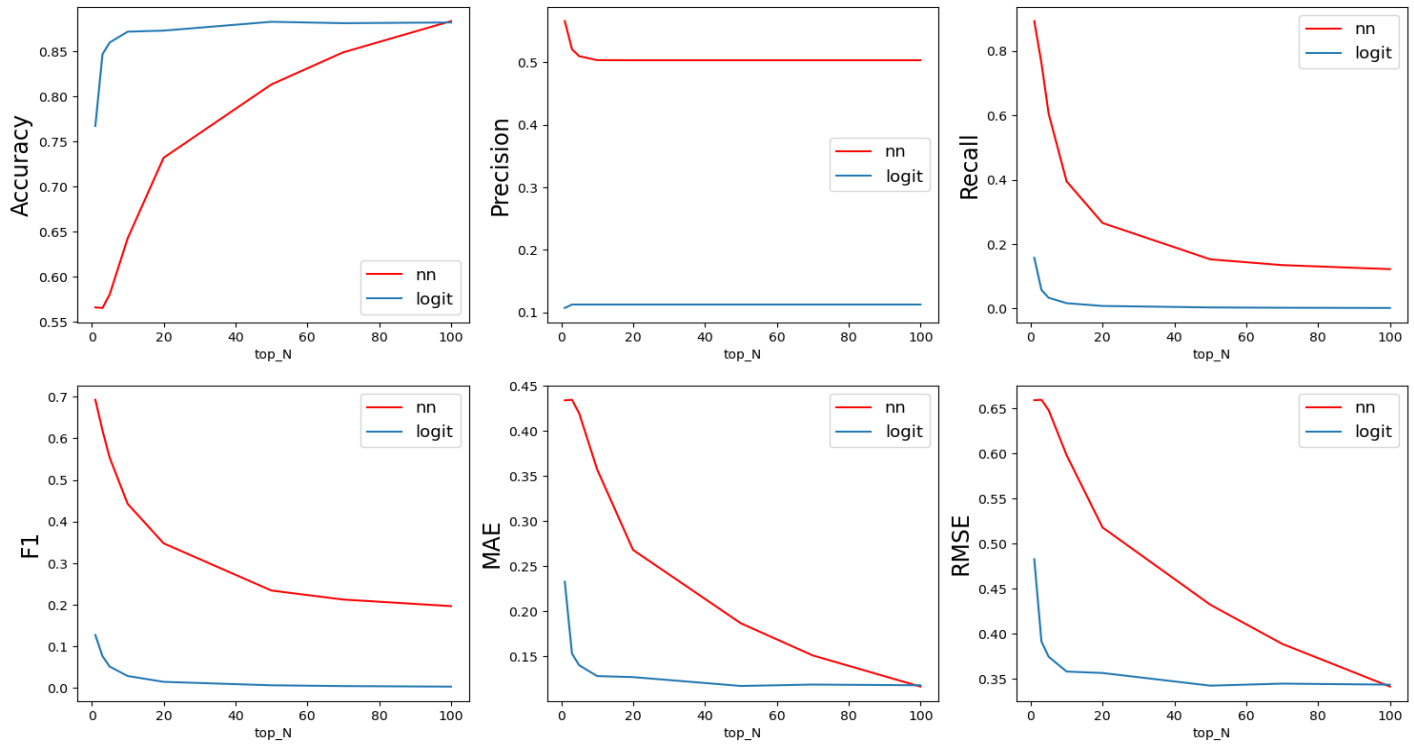
$$Finally,\ \hat{m}_t = \frac{m_t}{1-\beta_1^t}\ and\ \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$Adam\ weight\ update: \quad w_{t+1} = w_t - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

В процессе обучения возникает проблема, что нужно взять частную производную лосса по весам начального слоя, чтобы посчитать градиент. Но посчитать такую производную невозможно, потому что между предсказанием модели и изначальным слоем присутствуют еще несколько слоев со своими весами. Поэтому нейросеть в процессе обучения использует метод Backpropagation:

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial w_{11}^{(3)}} * \frac{\partial w_{11}^{(3)}}{\partial w_{11}^{(2)}} * \frac{\partial w_{11}^{(2)}}{\partial w_{11}^{(1)}}$$

Чтобы получить необходимую производную, нужно посчитать частные производные соседних слоёв и перемножить их.

Метрики для лучшей модели. Сравнение с baseline моделью.



Интерпретация результатов.

Наша модель показала ощутимо более хорошие показатели как по precision, так и по recall, и соответственно и по F1-score, поэтому мы считаем наш вывод о более сложной природе данных, которую нельзя описать линейной моделью, подтвержденной. Необходимо отметить, что именно F1-score является наиболее важной для бизнеса метрикой, так как рекомендательная система должна лучше предсказывать именно единицы, то есть что понравится пользователю, а не то, что ему не понравится, чтобы впоследствии выдать несколько лучших рекомендаций.

Исходя из вышесказанного, считаем, что наша модель показала сильное превосходство над baseline моделью, несмотря на то, что она проигрывает им по лоссам и accuracy, хоть они и не являются важными бизнес-метриками для нашей задачи. Однако, наша модель более сложна в обучении, плохо расширяется на полностью новых пользователей или рецепты и хуже интерпретируется. Но в то же время, сочетая kNN и нашу модель, можно в будущем построить архитектуру, которая будет сочетать плюсы обоих подходов и лишена их минусов.

Список используемых ресурсов

https://keras.io/api/layers/

https://www.kaggle.com/code/morrisb/how-to-recommend-anything-deep-recommender

https://towardsdatascience.com/modern-recommender-systems-a0c727609aa

https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c