

Рекомендательные системы

Урок 1. Введение, примеры задач, бизнес- и ML-метрики

Домашнее задание

- 1. Приведите еще примеры метрик для оценки рекомендаций/ранжирования (можно взять из интернета, или ваши знания)
- 2. Доделать все функции, где стоит комментарий "сделать дома"

1. Задание

Приведите еще примеры метрик для оценки рекомендаций/ранжирования (можно взять из интернета, или ваши знания)

2. ML-метрики качества

1. Hit rate

Hit rate = (был ли хотя бы 1 релевантный товар среди рекомендованных)

Hit rate@k = (был ли хотя бы 1 релевантный товар среди топ-k рекомендованных)

2. Precision

Precision $\equiv P@k$ = (# of recommended items that are relevant) / (# of recommended items)

$$Precision = \frac{TP}{TP + FP}$$
$$Precision = \frac{|hitset|}{N}$$

where

$$|hitset| = |test \cap topN|$$

Precision@k $\equiv P@k$ = (# of recommended items @k that are relevant) / (# of recommended items @k)

Money Precision@k = (revenue of recommended items @k that are relevant) / (revenue of recommended items @k)

3. Recall

Recall - доля рекомендованных товаров среди релевантных = Какой % купленных товаров был среди рекомендованных

- Обычно используется для моделей пре-фильтрации товаров (убрать те товары, которые точно не будем рекомендовать)

Recall $\equiv R$ = (# of recommended items that are relevant) / (# of relevant items)

Recall@k $\equiv R@k$ = (# of recommended items @k that are relevant) / (# of relevant items)

Money Recall@k = (revenue of recommended items @k that are relevant) / (revenue of relevant items)

Метрики ранжирования

AP@k

AP@k - average precision at k

$$AP@k = \frac{1}{r} \sum [recommended_{relevant_i}] * precision@k$$

- r - кол-во релевантных среди рекомендованных
- Суммируем по всем релевантным товарам
- Зависит от порядка рекомендаций

MAP@k

MAP@k (Mean Average Precision@k)

Среднее AP@k по всем юзерам

- Показывает средневзвешенную точность рекомендаций

$$MAP@k = \frac{1}{|U|} \sum_u AP_k$$

|U| - кол-во юзеров

AUC@k

Normalized discounted cumulative gain (NDCG@k)

$$DCG = \frac{1}{|r|} \sum_u \frac{[bought\ fact]}{discount(i)}$$

$discount(i) = i$ if $i \leq 2$,

$discount(i) = \log_2(i + 1)$ if $i > 2$

DCG - дисконтированный накопленный прирост

(!) Считаем для первых k рекомендаций

(!) - существуют вариации с другими $discount(i)$

i - ранг рекомендованного товара

|r| - кол-во рекомендованных товаров

$$NDCG = \frac{DCG}{ideal\ DCG}$$

$$ideal\ DCG@k = \sum_{k=1}^K \frac{1}{\log_2(k + 1)}$$

Mean Reciprocal Rank (MRR@k) -

Средний взаимный ранг

- Считаем для первых k рекомендаций
- Найти ранг первого релевантного предсказания k_u
- Посчитать reciprocal rank = $\frac{1}{k_u}$

$$MRR = mean(\frac{1}{k_u})$$

Метрики на основе ранговой корреляции

Отдельно стоит выделить метрики качества ранжирования, основанные на одном из коэффициентов **ранговой корреляции**. В статистике, ранговый коэффициент корреляции — это коэффициент корреляции, который учитывает не сами значения, а лишь их ранг (порядок).

Рассмотрим два наиболее распространенных ранговых коэффициента корреляции: коэффициенты Спирмена и Кендэлла.

Ранговый коэффициент корреляции Кендэлла (Kendall- τ)

[http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient] (http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient%5D).

$$\tau = \frac{C^+ - C^-}{\frac{1}{2}N(N - 1)}$$

where C^+ is the number of concordant pairs (согласованных пар), and C^- is the number of discordant pairs (не согласованных пар) in the data set.

где C^+ — число совпадений, C^- — число инверсий, N — объем выборки

Ранговый коэффициент корреляции Спирмена (Spearman's- ρ)

[http://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient] (http://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient%5D).

Второй — ранговый коэффициент корреляции Спирмена — по сути является ни чем иным как корреляции Пирсона, посчитанной на значениях рангов.

$$\rho = \frac{1}{n_u} \frac{\sum_i (r_{ui} - \bar{r})(\hat{r}_{ui} - \hat{\bar{r}})}{\sigma(r)\sigma(\hat{r})}$$

Метрики на основе каскадной модели поведения

До этого момента мы не углублялись в то, как пользователь (далее мы рассмотрим частный случай объекта — пользователь) изучает предложенные ему элементы. На самом деле, неявно нами было сделано предположение, что просмотр каждого элемента **независим** от просмотров других элементов — своего рода «наивность». На практике же, элементы зачастую просматриваются пользователем поочередно, и то, просмотрит ли пользователь следующий элемент, зависит от его удовлетворенности предыдущими. Рассмотрим пример: в ответ на поисковый запрос алгоритм ранжирования предложил пользователю несколько документов. Если документы на позиции 1 и 2 оказались крайне релевантны, то вероятность того, что пользователь просмотрит документ на позиции 3 мала, т.к. он будет вполне удовлетворен первыми двумя.

Подобные модели поведения пользователя, где изучение предложенных ему элементов происходит последовательно и вероятность просмотра элемента зависит от релевантности предыдущих называются каскадными.

Expected reciprocal rank

Expected reciprocal rank (ERR) — пример метрики качества ранжирования, основанной на каскадной модели. Задается она следующей формулой:

$$ERR@k = \sum_{k=1} K \frac{1}{|k|} P(\text{объект остановится на элементе с рангом } k)$$

где ранг понимается по порядку убывания веса ранжированных элементов $r(e)$. Самое интересное в этой метрике — вероятности. При их расчете используются предположения каскадной модели:

$$P(\text{объект остановится на элементе с рангом } k) = p_k \prod_{i=1}^{k-1} (1 - p_i),$$

где p_k — вероятность того, что пользователь будет удовлетворен объектом с рангом k .

PFoundk

PFound — метрика качества ранжирования, предложенная нашими соотечественниками [http://romip.ru/romip2009/15_yandex.pdf] (http://romip.ru/romip2009/15_yandex.pdf%5D) и использующая похожую на каскадную модель:

2. Задание

2. Доделать все функции, где стоит комментарий "сделать дома"

```
B [1]: import pandas as pd
import numpy as np
```

```
B [18]: recommended_list = [143, 156, 1134, 991, 27, 1543, 3345, 533, 11, 43] #id товаров

# user1
bought_list = [521, 32, 143, 991]
```

1. Hit rate

```
B [19]: def hit_rate(recommended_list, bought_list):
    bought_list = np.array(bought_list)
    recommended_list = np.array(recommended_list)

    flags = np.isin(bought_list, recommended_list)
    print(flags)
    return (flags.sum() > 0) * 1

def hit_rate_at_k(recommended_list, bought_list, k=5):
    # сделать в домашней работе
    # pass
    return hit_rate(recommended_list[:k], bought_list)
```

```
B [23]: hit_rate(recommended_list, bought_list)

[False True True False]
```

Out[23]: 1

2. Precision

```
B [6]: recommended_list = [143,156, 1134, 991, 27, 1543, 3345, 533, 11, 43] #id товара
bought_list = [521, 32, 143, 991]
prices_resommended = [400, 60, 40, 40 , 90, 200, 50, 100, 40, 15]
```

```
B [7]: def precision(recommended_list, bought_list):

    bought_list = np.array(bought_list)
    recommended_list = np.array(recommended_list)

    flags = np.isin(bought_list, recommended_list)
    return flags.sum() / len(recommended_list)

def precision_at_k(recommended_list, bought_list, k=5):
    # сделать в домашней работе
    return precision(recommended_list[:k], bought_list)

def money_precision_at_k(recommended_list, bought_list, prices_recommended, k=5):

    recommend_list = np.array(recommended_list[:k])
    prices_recommend = np.array(prices_recommended[:k])

    flags = np.isin(recommend_list, bought_list)

    precision = np.dot(flags, prices_recommend).sum() / prices_recommend.sum()

    return precision
```

```
B [8]: precision(recommended_list, bought_list)
```

Out[8]: 0.2

```
B [9]: precision_at_k(recommended_list, bought_list, 5)
```

Out[9]: 0.4

```
B [10]: k=5
money_precision_at_k(recommended_list, bought_list, prices_resommended, k)
```

Out[10]: 0.6984126984126984

3. Recall

```
B [11]: recommended_list = [143, 156, 1134, 991, 27, 1543, 3345, 533, 11, 43] #id товара
bought_list = [521, 32, 143, 991]
prices_resommended = [400, 60, 40, 40 , 90, 200, 50, 100, 40, 15]
```

```
B [12]: def recall(recommended_list, bought_list):

    bought_list = np.array(bought_list)
    recommended_list = np.array(recommended_list)

    flags = np.isin(bought_list, recommended_list)

    return flags.sum() / len(bought_list)

def recall_at_k(recommended_list, bought_list, k=5):
    # pass
    # сделать дома
    return recall(recommended_list[:k], bought_list)

#def money_recall_at_k(recommended_list, bought_list, prices_recommended, prices_bought, k=5):
def money_recall_at_k(recommended_list, bought_list, prices_recommended, k=5):
    # pass
    # сделать дома

    if k >= len(recommended_list):
        k = len(recommended_list)-1

    # bought_list = np.array(bought_list)
    recommend_list = np.array(recommended_list[:k])
    prices_recommend = np.array(prices_recommended[:k])

    flags = np.isin(bought_list, recommend_list)

    recall = np.dot(flags, prices_recommend).sum() / prices_recommend.sum()

    return recall
```

```
B [13]: recall(recommended_list, bought_list)
```

```
Out[13]: 0.5
```

```
B [14]: recall_at_k(recommended_list,bought_list, 10)
```

```
Out[14]: 0.5
```

```
B [15]: money_precision_at_k(recommended_list, bought_list, prices_resommended, 10)
```

```
Out[15]: 0.4251207729468599
```

Метрики ранжирования

AP@k

```
B [16]: recommended_list = [143, 156, 1134, 991, 27, 1543, 3345, 533, 11, 43] #id товаров
bought_list = [521, 32, 143, 991]
```

```
B [17]: def ap_k(recommended_list, bought_list, k=5):

    bought_list = np.array(bought_list)
    recommended_list = np.array(recommended_list)[:k]

    relevant_indexes = np.nonzero(np.isin(recommended_list, bought_list))[0]
    if len(relevant_indexes) == 0:
        return 0

    amount_relevant = len(relevant_indexes)

    for index_relevant in relevant_indexes:
        print(precision_at_k(recommended_list, bought_list, k=index_relevant + 1) )
    sum_ = sum([precision_at_k(recommended_list, bought_list, k=index_relevant + 1) for index_relevant in relevant_indexes])

    return sum_/amount_relevant
```

```
B [18]: ap_k(recommended_list, bought_list, k=5)
```

```
1.0
0.5
```

```
Out[18]: 0.75
```

MAP@k

```
B [19]: # теперь список из 3 пользователей
recommended_list_3_users = [[143, 156, 1134, 991, 27, 1543, 3345, 533, 11, 43],
                             [1134, 533, 14, 4, 15, 1543, 1, 99, 27, 3345],
                             [991, 3345, 27, 533, 43, 143, 1543, 156, 1134, 11]
                             ]

bought_list_3_users = [[521, 32, 143], # юзер 1
                       [143, 156, 991, 43, 11], # юзер 2
                       [1,2] # юзер 3
                       ]
```

```
B [20]: def map_k(recommended_list, bought_list, k=5):

    # сделать дома

    result = 0
    N = len(bought_list_3_users)
    for i in range(N):
        print(f"ap@k({i})={ap_k(recommended_list[i], bought_list[i], k=5)}")
        result += ap_k(recommended_list[i], bought_list[i], k=5)

    return result/N
```

```
B [21]: map_k(recommended_list_3_users, bought_list_3_users, k=5)
```

```
1.0
ap@k(0)=1.0
1.0
ap@k(1)=0
ap@k(2)=0
```

```
Out[21]: 0.3333333333333333
```

Normalized discounted cumulative gain (NDCG@k)

B [46]: `import math`

B [47]: `# по желанию`

`# def ndcg_at_k():`
`# pass`

B [126]: `# по желанию`

```
def idial_dcg_at_k(recommended_list, bought_list, k):
    bought = np.array(bought_list)
    recommended = np.array(recommended_list[:k])
    discount = []

    for i in range(k):
        if i < 2:
            discount.append( 1/(i+1) )
        else:
            discount.append( 1/math.log2(i+1))

    print(discount)
    return sum(discount)/k

def dcg_at_k(recommended_list, bought_list, k):
    bought = np.array(bought_list)
    recommended = np.array(recommended_list[:k])
    discount = []

    for i in range(k):
        if recommended_list[i] in bought_list:
            if i < 2:
                discount.append( 1/(i+1) )
            else:
                discount.append( 1/math.log2(i+1) )
        else:
            discount.append(0)

    print(discount)
    return sum(discount)/k

def ndcg_at_k(recommended_list, bought_list, k):
    dcg_atk= dcg_at_k(recommended_list, bought_list, k)
    idial_dcg_atk = idial_dcg_at_k(recommended_list, bought_list, k)

    return dcg_atk/(k*idial_dcg_atk)
```

B [127]: `recommended_list = [143, 156, 1134, 991, 27, 1543, 3345, 533, 11, 43] #id товаров`
`bought_list = [521, 32, 143, 991]`

B [128]: `k=5`
`# print(recommended_list, bought_list, k)`
`# print(1/math.Log2(3), 1/math.Log2(4),1/math.Log2(5))`
`idial_dcg_at_k(recommended_list, bought_list, k)`

`[1.0, 0.5, 0.6309297535714575, 0.5, 0.43067655807339306]`

Out[128]: `0.6123212623289701`

B [129]: `dcg_at_k(recommended_list, bought_list, k=5)`

`[1.0, 0, 0, 0.5, 0]`

Out[129]: `0.3`

B [130]: `ndcg_at_k(recommended_list, bought_list, k)`

`[1.0, 0, 0, 0.5, 0]`

`[1.0, 0.5, 0.6309297535714575, 0.5, 0.43067655807339306]`

Out[130]: `0.0979877781342908`

Mean Reciprocal Rank (MRR@k)

```
B [23]: # теперь список из 3 пользователей
recommended_list_3_users = [[143, 156, 1134, 991, 27, 1543, 3345, 533, 11, 43],
                             [1134, 533, 14, 4, 15, 1543, 1, 99, 27, 3345],
                             [991, 3345, 27, 533, 43, 143, 1543, 156, 1134, 11]
                             ]

bought_list_3_users = [[521, 32, 143], # юзер 1
                      [143, 156, 991, 43, 11], # юзер 2
                      [1, 2] # юзер 3
                      ]
```

```
B [24]: def inverse_rank(recommended_list, bought_list, k=1):
        # Находим 1/ранг первого релевантного предсказания для пользователя
        rank=[( 1/(i+1) ) for i in range(len(recommended_list[:k])) for j in range(len(bought_list)) if recommended_list[i] == bought_list[j]]

        if len(rank) > 0:
            return rank[0]
        else:
            return 0

def reciprocal_rank(recommended_list_users, bought_list_users, k=1):
    # Находим список рангов

    rec_rank = []
    for i in range(len(bought_list_users)):
        rec_rank.append(inverse_rank(recommended_list_users[i], bought_list_users[i], k))
    rec_rank
    return rec_rank

def mean_reciprocal_rank(recommended_list_users, bought_list_users, k=1):

    # сделать дома

    # Mean Reciprocal Rank

    rec_rank = reciprocal_rank(recommended_list_users, bought_list_users, k)

    mrr_at_k = np.sum(rec_rank)/len(bought_list_users)

    return mrr_at_k
```

```
B [25]: reciprocal_rank(recommended_list_3_users, bought_list_3_users, k=10)
```

```
Out[25]: [1.0, 0, 0]
```

```
B [26]: mean_reciprocal_rank(recommended_list_3_users, bought_list_3_users, k=10)
```

```
Out[26]: 0.3333333333333333
```

<https://habr.com/ru/company/econtenta/blog/303458/> (<https://habr.com/ru/company/econtenta/blog/303458/>)

Evaluation <https://github.com/ocelma/python-recsys/blob/master/doc/source/evaluation.rst> (<https://github.com/ocelma/python-recsys/blob/master/doc/source/evaluation.rst>)