Стажировка в VK.
Отчет по профильному заданию на позицию ML-engineer

Вихляев Е.С.

Пермский Государственный Национальный Исследовательский Университет,
ПГНИУ, Пермь
rourkean@yandex.ru

Аннотация В отчете представлено решение задачи ранжирования документов на основе представленного датасета. Задача заключается в обучении модели, которая может ранжировать документы на основе их характеристик в пределах одной поисковой сессии. Описаны шаги подготовки и проверки данных, предпринятые для обеспечения их качества и согласованности. Результирующий датасет используется для обучения машинной модели с использованием scikit-learn и PyTorch. Результирующая модель оценивается с использованием нормализованной накопленной дисконтированной выгоды (NDCG), которая широко используется в задачах поиска и ранжирования, а так же MAP. Обсуждается производительность модели и возможные области для улучшения.

Keywords: ML · Ranking · feature · rank · Data preparation · Data verification · Model training · PyTorch · Scikit-learn · Rank prediction based on features · Ranking metrics · NDCG · ERR · PFound · Model quality assessment.

1 Постановка задачи и определение метрик

1.1 Постановка задачи

Перед нами стоит **задача ранжирования (learning to rank)**. Она заключается в сортировке набора *элементов* из соображения их *релевантности*.

1.2 Определение метрик

Цель метрики качества ранжированния — определить, насколько полученные алгоритмом оценки релевантности $r(e)$ и соответствующая им перестановка π соответствуют истинным значениям релевантности r^{true}.

Заранее следует отметить, что распространенная метрика MAP в нашем случае неприменима, поскольку мы имеем небинарную оценку релевантности.

Normalized Discounted Cumulative Gain (nDCG). Это одна из наиболее часто используемых метрик качества ранжирования. Имеем

$$nDCG_K = \frac{DCG_K}{IDCG_K},$$

где $IDCG_K$ – это максимальное (I — ideal) значение

$$DCG_K = \sum_{k=1}^{K} \frac{r^{\text{true}}(\pi^{-1}(k)}{log_2(k+1)}$$

Excepted reciprocal rank (ERR). Метрика качесвта ранжирования, основанная на каскадной модели. Задается следующей формулой:

$$ERR_K = \sum_{k=1}^{K} \frac{1}{k} P,$$

где ранк понимается по порядку убывания $r(e)$. При расчете вероятностей используются предположения каскадной модели:

$$P = p_k \prod_{i=1}^{k-1} (1-p),$$

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

$$p_k = \frac{2^{r^{\text{true}}(\pi^{-1}(k)} - 1}{2^{max\ r^{true}} - 1}.$$

PFound. Метрика качества ранжирования, использующая похожую каскадную модель:

$$PFound_K = \sum_{k=1}^{K} PLook(k) PRel(k),$$

где

- $PLook(k) = PLook(k-1)(1 - PRel(k-1))(1 - PBreak)$,
- $PRel(i) = 2^{r^{true}(\pi^{-1}(k))-1, r^{true}(\pi^{-1}(k))>00}$,
- $PBreak$ – вероятность того, что пользователь прекратит просмотр по внешним причинам.

2 Получение данных

В нашем случае все просто — мы имеем готовый датасет.

3 EDA

Поскольку мы используем преобразование в скрипты, для начала необходимо переместиться в соответсвующую EDA директорию. Очевидно, переадресация должна быть относительной, чтобы проект адекватно работал и на других устройствах.

3.1 Статистический обзор

Выводим статистический обзор через метод *df.describe()* и *df.shape()*. Исходя из него, можем сделать вывод, что наибольшие средние значения наблюдаются в признаках: *query_ id*, *feature_ 35* (наибольшее значение — 1.000.000), *feature_ 127*, *feature_ 129*, *feature_ 130*, *feature_ 135*.

Затем выводим *df.info()*. Имеем, что в датасете 146 столбцов и 235258 строк. 140 столбцов — типа float, 6 столбцов — целочисленные, типа int.

3.2 Пропущенные значения

Проверяем наличие пропущенных значений NaN с помощью связки методов: *df.isnull().values.any()*. Получаем в итоге False, что означает, что в датасете отсутствуют пропущенные значения.

3.3 Корреляционный анализ

Строим матрицу корреляции всех переменных, кроме query_id (Рис. 1).

Коэффициенты корреляции большей части переменных лежит в районе значения 0.25. Тем не менее, мы видим, что есть несколько кластеров, где корреляция отрицательные. Например, треугольники feature_15 - feature_19 и feature_111 - feature_123 по вертикали.

Так же видим кластер в виде прямоугольника в диапазоне по вертикали (feature_111 - feature_119) и горизонтали (feature_15 - feature_19). Остальные кластеры минимальной корреляции достаточно неоднородны.

В это же самое время, переменные, коэффциенты корреляций которых наиболее высоки — достаточно сильно разбросаны по датасету.

Для увеличения скорости модели, эти кластеры можно удалить, поскольку они не являются для нас полезными.

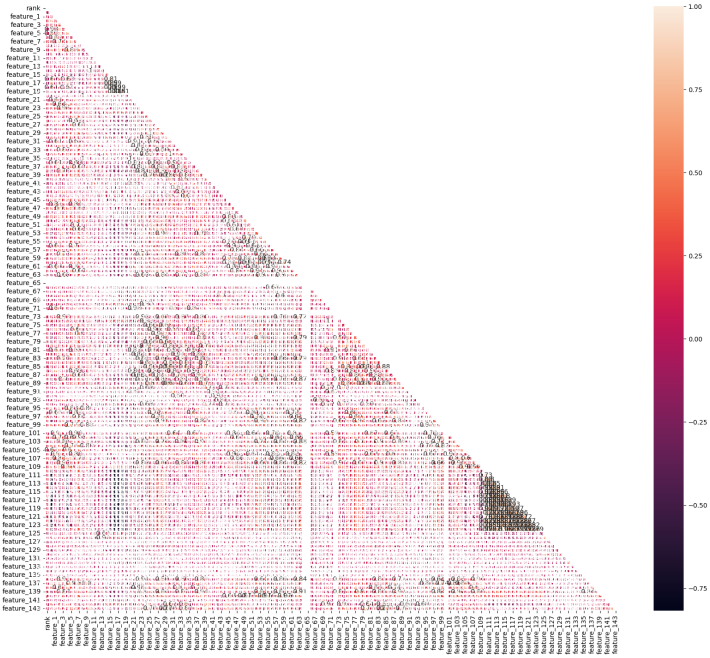

Рис. 1. Матрица корреляции признаков

3.4 График частот

См. ноутбук EDA.

3.5 Гистограммы распределения

См. ноутбук EDA. Как мы можем заметить, далеко не все столбцы имеют нормальное распределение. Точнее говоря, почтчи ни один. Часто встречается нечто похожее на распределение Пуассона, то есть отклонено в стороны, а так же присутствует равномерное распределение.

Делаем вывод, что данные нужно будет масштабировать.

3.6 Диаграммы рассеяния

См. ноутбук EDA. По диаграммам рассеяния можно свидетельствовать о сильной линейной зависимости между переменными. Можно работать с линейными моделями.

3.7 Гистограмма распределения классов

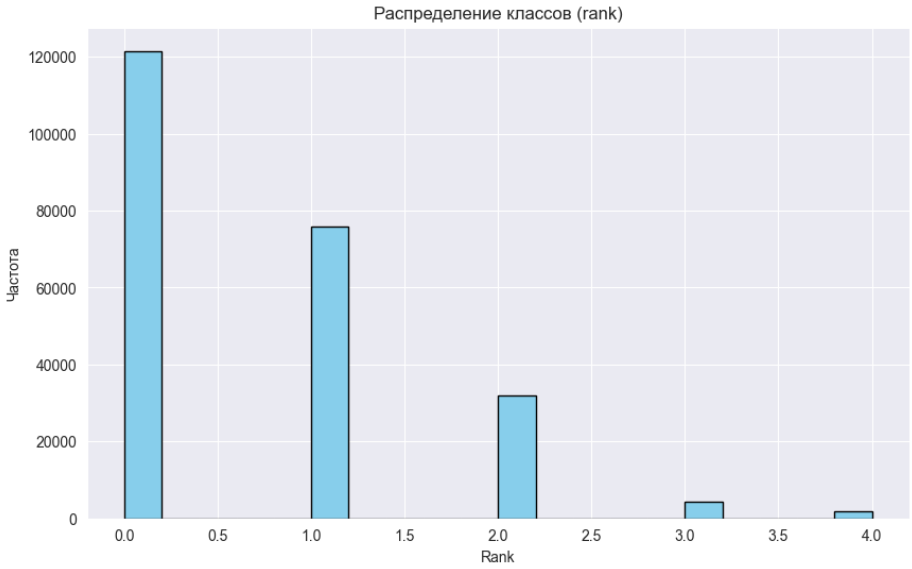

Рис. 2. Гистограмма распределения классов (rank)

Видим, что класс 0 обладает наибольшей частотой, на втором месте класс 1, на третьем класс 2, на четвертом 3 и на последнем 4.

Приходим к выводу, что классы несбалансированны. Есть перепредставленные классы (0, 1), которые могут привести к переобучению. Есть и подпредставленные классы (3, 4), которые сложнее обучить и, следовательно, они могут понизить точность модели.

Вывод: классы необходимо сбалансировать.

3.8 Проверка выбросов

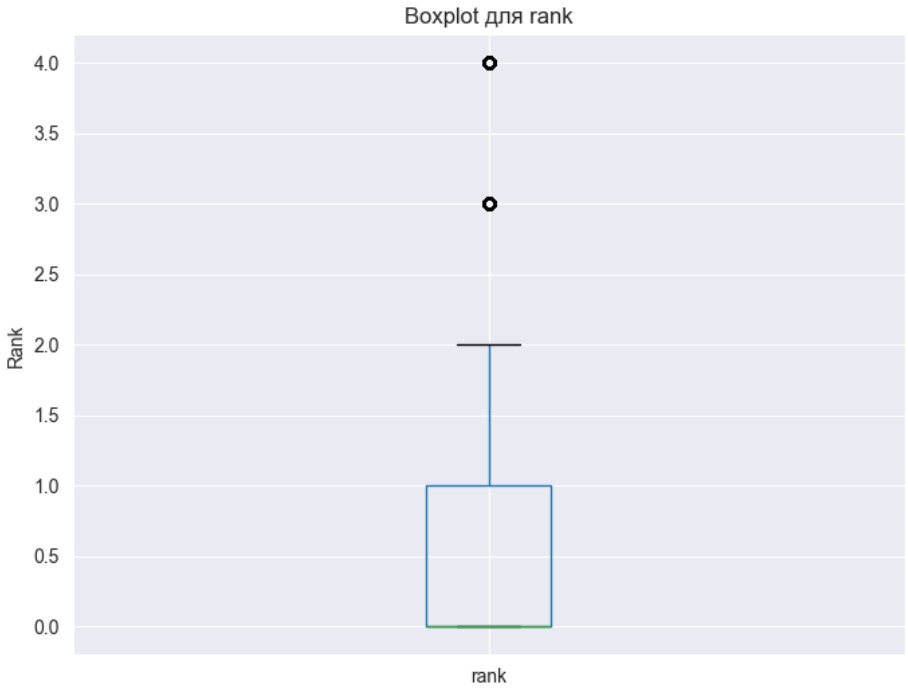

Рис. 3. Boxplot для rank

Очевидно, данные имеют выбросы. Далее проверяем z-оценку для столбца rank с допустимым диапазоном (-3, 3): имеем 1803 выброса.

3.9 Проверка наличия дубликатов

См. ноутбук EDA. Дубликаты в датасете отсутствуют.

3.10 Выводы по EDA

– Пропущенные значения отсутствуют.

- Некоторые кластеры переменных можно удалить.
- Необходимо стандартизировать и масштабировать данные, привести их к нормальному распределнию.
- Можно использовать линейные модели.
- Классы необходимо сбалансировать.
- Необходимо удалить выбросы.
- Необходимо выделить целевую переменную.
- Разделить данные на тестовую и обучающую выборки.

4 Обработка данных

4.1 Удаление ненужных кластеров переменных

На рис. 1) видим, что есть минимум два кластера переменных с отрицательной корреляцией, которые имеет смысл удалить. Они находятся на диагонали матрицы и представляют из себя два треугольника разных размеров. Индексы данных feature лежат в диапазонах (15, 21) и (110, 126). В итоге, из 146 колонок остается 124.

4.2 Стандартизация и масштабирование данных

При стандартизации данных мы должны исключить из этого процесса колонки rank и query_id. Стандартизацию проведем классическим методом StandardScaler().

4.3 Удаление выбросов

Выбросы определим по величине zscore в допустимом диапазоне (-3, 3). Определив строки, лежащие вне этого диапазона — удаляем их из исходного датасета. Из 235258 строк остается 147998 строк.

4.4 Целевая переменная

Очевидно, целевая переменная находится в колонке rank. При этом, поскольку, вероятно, query_id не понадобится при обучении модели, мы исключаем его из множества признаков X.

4.5 Балансировка классов

Анализ классов, проведенный в EDA (см. Рис 1) показывает, что у нас присутствует сильный дисбаланс классов. При обучении это может привести как к переобучению модели, так и к тому, что она будет стремится отдавать предпочтение наибольшему классу, вопреки меньшим классам. Для балансировки были использованы Oversampling и Undersampling. И тот, и другой метод удаляют наименьший класс (4), поскольку он практически не представлен в датасете на фоне остальных классов. При обучении модели это может привести к потери важных факторов, поэтому было принято решение обучить модель на трех различных выборках:

– с дисбалансом классов;
– с Oversampling;
– с Undersampling.

Oversampling увеличил число строк до 325064, в то время как Undersampling уменьшил их до 8564.

4.6 Разделение на тестовую и обучающую выборки

Поскольку у нас достаточно большое количество наблюдений — поделим тестовую и обучающую выборки в классическом соотношении 80/20.

5 Обучение моделей

5.1 XGBoost

XGBoost — одна из самых популярных и эффективных реализаций алгоритма градиентного бустинга на деревьях. Поскольку перед нами стоит задача ранжирования, то мы воспользуемся классом xgboost.XGBRanker.

Поскольку при Oversampling количество признаков в X не увеличивается до 325064, как в y, то этот метод и его результаты далее мы использовать не будем.

Поэтому, модель будет работать с двумя видами данных: не сбалансированными и с Undersampling.

Обучив обе модели, получае следующие диаграммы релевантности:

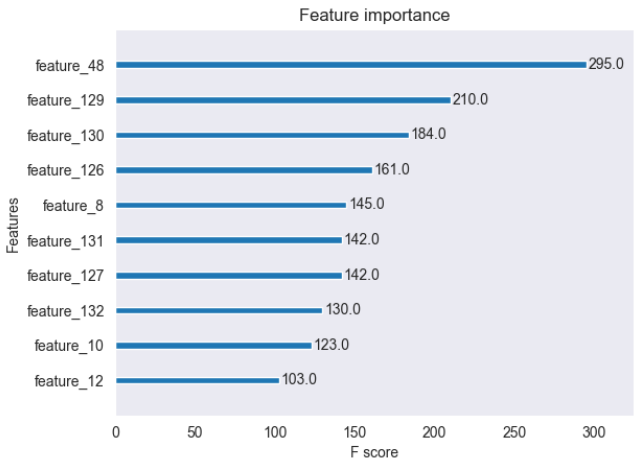

Рис. 4. Диаграмма релевантности для XGBoost без сбалансирования

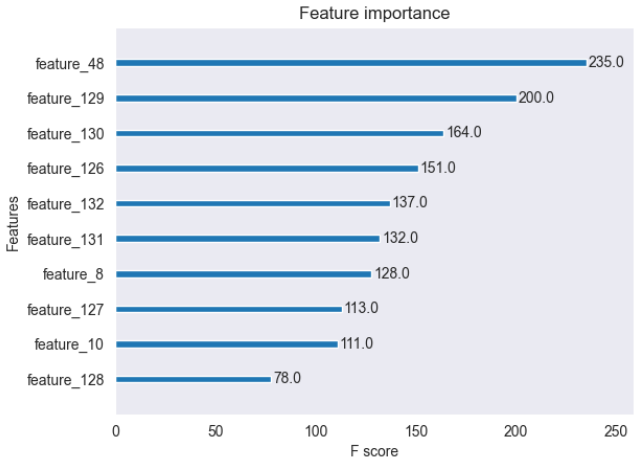

Рис. 5. Диаграмма релевантности для XGBoost с Undersampling

5.2 CatBoost

CatBoost — это библиотека градиентного бустинга, созданная Яндексом. Она использует небрежные (oblivious) деревья решений, чтобы вырастить сбалансированное дерево. Одни и те же функции используются для создания левых и правых разделений (split) на каждом уровне дерева.

Действуем аналогично с XGBoost: обучаем модели на двух выборках с балансировкой и без нее.

6 Оценка моделей

В совокупности мы имеем 4 модели: две модели XGBoost для сбалансированных и несбалансированных данных, и, аналогично, две модели CatBoost.

6.1 NDCG@5

NDCG@5 для XGBoost без сбалансирования: 0.55
NDCG@5 для XGBoost с Undersampling: 0.72

NDCG@5 для CatBoost без сбалансирования: 0.33
NDCG@5 для CatBoost с Undersampling: 0.77

Список литературы

1. Discounted cumulative gain // Wikipedia URL: https://en.wikipedia.org/wiki/Discounted_cumulative_gain (дата обращения: 24.04.2024).
2. Information retrieval // Wikipedia URL: https://en.wikipedia.org/wiki/Information_retrieval Mean_average_precision (дата обращения: 24.04.2024).
3. Olivier Chapelle, Donald Metlzer, Ya Zhang, Pierre Grinspan Expected reciprocal rank for graded relevance // ACM. - 2009. - №18. - С. 621–630.
4. Learning to Rank // XGBoost Documentation URL: https://xgboost.readthedocs.io/en/latest/tutorials/learning_to_rank.html (дата обращения: 30.04.2024).