



Ranking long tail queries TS Spring 2020

Learning to rank long tail queries

7 teams · 10 hours ago

[Overview](#)[Data](#)[Notebooks](#)[Discussion](#)[Leaderboard](#)[Rules](#)[Team](#)[My Submissions](#)[Late Submission](#)

Overview

Description

Evaluation

Long tail queries ranking

После получения запроса от пользователя, поисковая система отбирает некоторое число страниц-кандидатов. Для того чтобы показать пользователю страницу результата поиска, страницы-кандидаты упорядочиваются по убыванию их релевантности и показываются только наиболее релевантные.

Вам необходимо реализовать алгоритм машинного обучения ранжированию и с его помощью выбрать 5 наиболее релевантных документов, отсортировать их по убыванию релевантности.

Pipeline

1. Загрузка данных
2. Предобработка данных
 - a. Текст запросов
 - b. Текст документов
 - c. Кликовые данные
3. Извлечение факторов
 - a. Семантические факторы
 - b. Синтаксические факторы
 - c. Кликовые факторы
4. Обучение ранжирующей модели
5. Предсказание

Загрузка данных

- Кликовые данные лежали в hadoop
- Данные о содержании документа скачал на облако mail.ru а затем на кластер и в hadoop
- Данные запросов скачал локально на ноутбук

Предобработка запросов

1. **Преобразование в нижний регистр**
2. **Spellcheck** - исправление опечаток. Использовался Yandex Speller^[1] с помощью REST API.
3. **Lemmatization** - приведение слова к его начальной форме. Использовался Yandex Stemmer^[2] с помощью Python библиотеки pymystem3.
4. **Tokenizer** - разделение предложений на отдельные части (3-граммы, 4-граммы, слова, пары слов). Использовался CountVecorizer из библиотеки sklearn^[3].

[1] Yandex Speller <https://yandex.ru/dev/speller/>

[2] Yandex Stemmer <https://yandex.ru/dev/mystem/>

[3] Sklearn <https://scikit-learn.org/>

Генерация похожих запросов

1. Выделение множества слов используемых в запросах.
2. Поиск синонимов для слов с помощью Yandex Dictionary^[4] REST API и с помощью метода ближайших соседей в пространстве эмбедингов. Использовался предобученный FastText^[5].
3. Подсчёт “похожетси” синонимов как косинусного расстояния в пространстве эмбедингов FastText.
4. Генерация 10 похожих запросов. Вероятность каждого синонима пропорциональна “похожести” на исходное слово. Вероятность итогового запроса равна произведению вероятностей составляющих его слов

[4] Yandex Dicionary <https://tech.yandex.com/dictionary/>

[5] FastText <https://fasttext.cc/>

Преобразование документов

1. **Выделение заголовков.** Если заголовок отсутствует - берутся первые 300 слов. Использовался Hadoop Map-Reduce.
2. **Преобразование в нижний регистр.**
3. **Lemmatization** - приведение слова к его начальной форме. Использовался Yandex Stemmer с помощью Python библиотеки pymystem3.
4. **Tokenizer** - разделение предложений на отдельные части (3-граммы, 4-граммы, слова, пары слов). Использовался Count Vectorizer из библиотеки sklearn.

Семантические факторы

1. Преобразование текстов запросов и заголовков в эмбединги:
 - a. **FastText** (Original^[6], DeepPavlov^[7] on wikipedia, DeepPavlov on Twitter)
 - b. **ELMo** (DeepPavlov on Wikipedia, DeepPavlov on Twitter, DeepPavlov on WMT News)
 - c. **BERT** (DeepPavlov RuBERT, DeepPavlov Conversational RuBERT, Sentence RuBERT)
 - d. **USE** (Google V3^[8], Google V3 Large)
2. Подсчёт косинусной близости

[6] Facebook pre-trained FastText <https://fasttext.cc/docs/en/crawl-vectors.html>

[7] DeepPavlov pre-trained models http://docs.deeppavlov.ai/en/master/features/pretrained_vectors.html

[8] Google pre-trained TensorFlow USE <https://tfhub.dev/google/universal-sentence-encoder-multilingual-large/3>

Синтаксические факторы

1. **TF-IDF, BM-25** на 3-граммах, 4-граммах, словах, парах слов. Использовалась библиотека Gensim^[9]

[9] Gensim python library <https://radimrehurek.com/gensim/>

Кликовые факторы

Использовалась модель из конкурса про пользовательское поведение

Статистики

Imp(d) – число показов документа d по всем выдачам, в которых он встречался.

Click(d) – число кликов на документ d по всем выдачам, в которых он встречался.

CTR(d) – CTR d по всем выдачам.

DocAvgTime(d) - среднее время просмотра документа d.

DocAvgAction(d) - среднее количество активных действий пользователя на документе d

QCTR(q,d) – CTR d по в выдачах по запросу q

FirstCtr(q,d) – CTR, когда d кликается первым в выдаче по запросу q.

LastCtr(q,d) – CTR, когда d кликается последним в выдаче по запросу q.

OnlyCTR(q,d) – CTR, когда кликается только d, по всем выдачам по q.

%DocClicks(q,d) - доля кликов по документу d в выдачах по запросу q.

AvgDocClickTime(q,d) – среднее время до клика на d после показа выдачи по запросу q.

AvgViewTime(q,d) - среднее время просмотра документа d по запросу q.

AvgDocPos(q,d) - средняя позиция d в выдачах по запросу q.

AvgDocClickPos(q,d) - средний номер клика d в выдачах по запросу q.

AvgDocClickInvPos(q,d) - средний номер клика d с конца в выдачах по запросу q.

AvgNumBefore(q,d) – среднее число документов, стоящих в выдаче по запросу q перед d, которые были кликнуты перед d.

AvgNumPast(q,d) – среднее число документов, стоящих в выдаче по запросу q перед d, которые были кликнуты после d.

LastProb(q,d) - вероятность быть последним документом, кликнутым по запросу q

UpProb(q,d) – вероятность клика на документ, находящийся в выдаче по q на позициях выше d.

DownProb(q,d) - вероятность клика на документы, находящиеся в выдаче по q на позициях ниже d.

DoubleProb(q,d) - вероятность того, что по d кликнули два раза подряд.

PastBackProb(q,d) – вероятность того, что к документу вернулись после клика i ссылок.

BeforeProb(q,d) – вероятность того, что после клика на d, пользователь кликал выше него.

...

Для статистики рассматривалось 5 признаков:

- Данная статистика для запроса
- Данная статистика для url
- Данная статистика для host
- Данная статистика для пары запрос - url (10% итог. заполн.)
- Данная статистика для пары запрос - host (13% итог. заполн.)

Регион запроса в работе не учитывался

Итого около 50 признаков для каждой пары запрос-документ

Обработка данных и построение обучающего датасета в две map-reduce джобы

Решение проблемы матчинга запросов

К исходным запросам были добавлены исправленные с помощью spellcheck запросы и до 10 их синонимов к ним с соответствующим весом.

Обучение ранжирующей модели

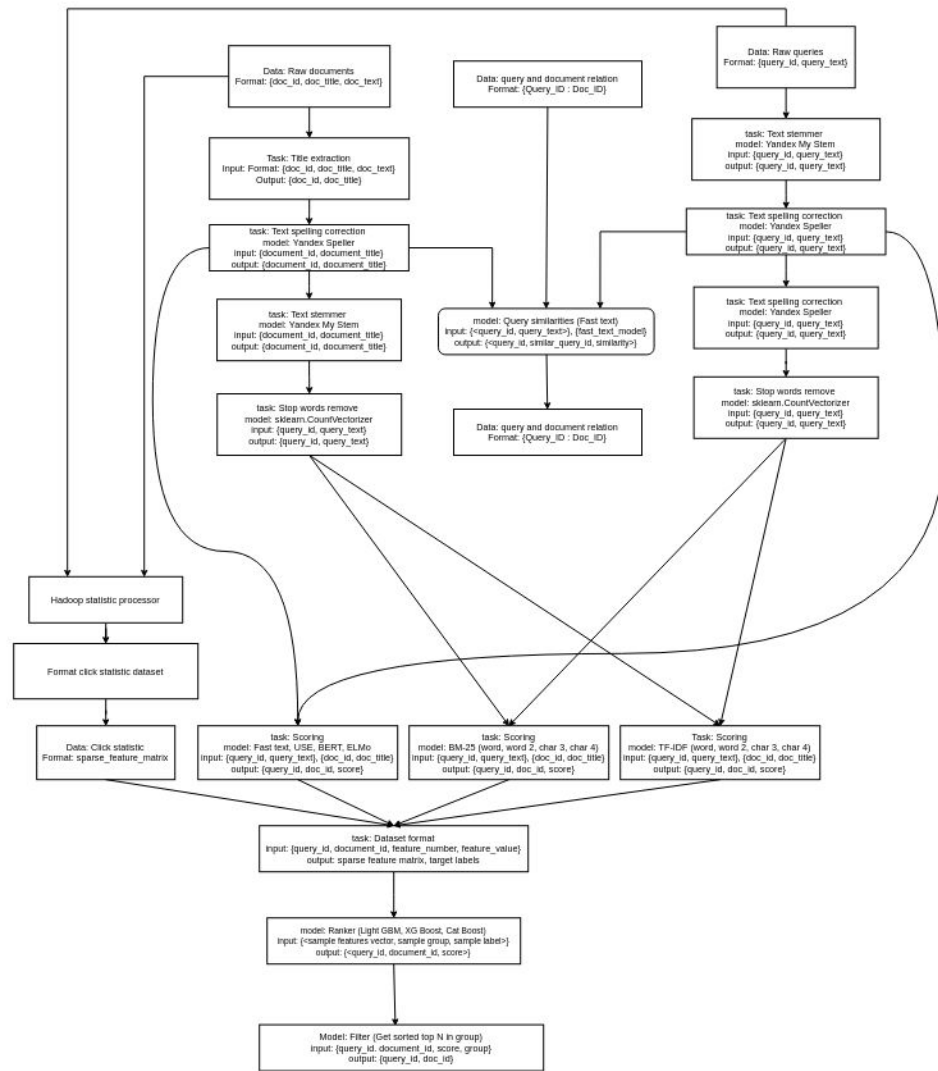
Использовался **Microsoft LightGBM**^[10]

Параметры:

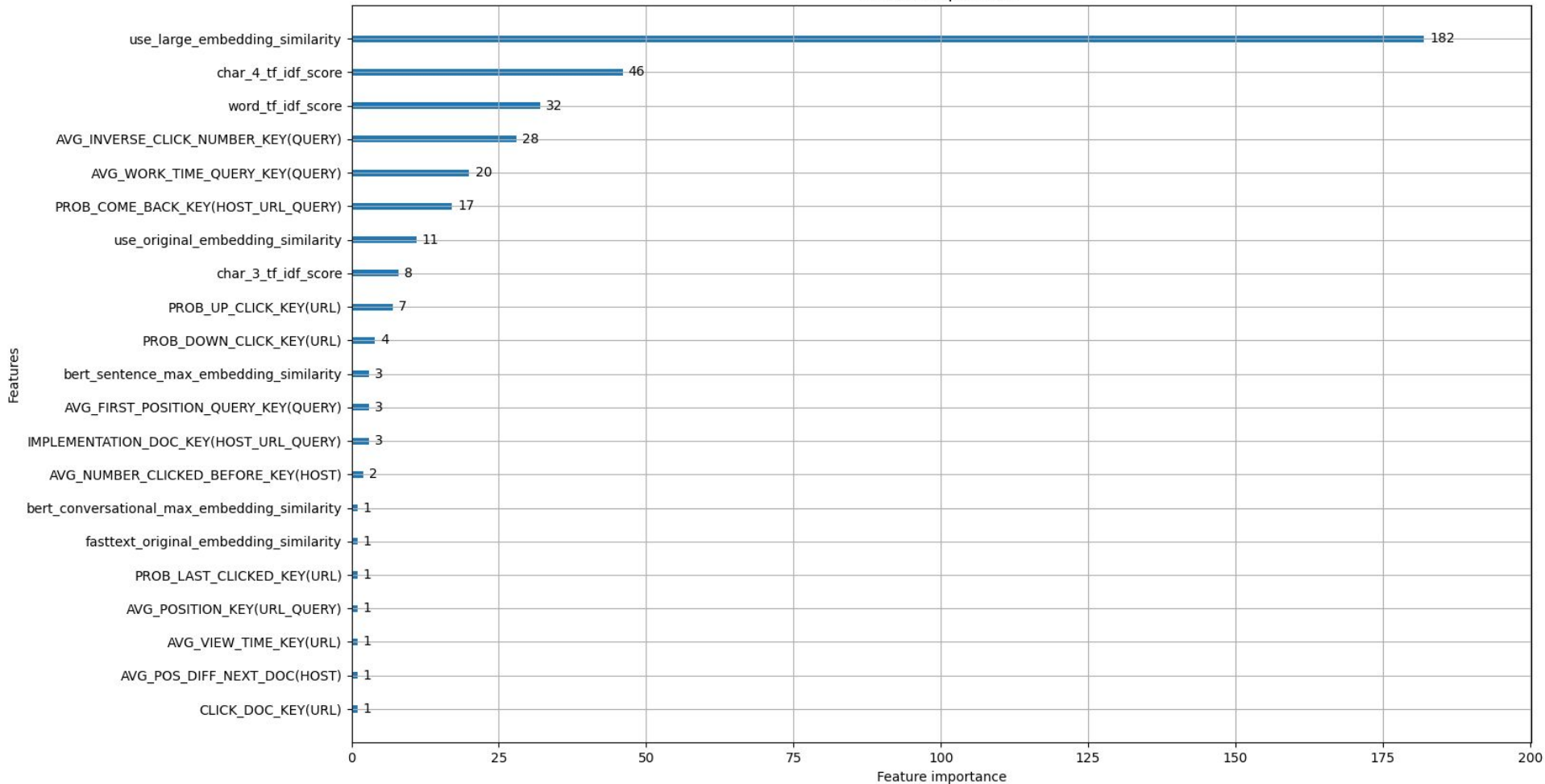
- boosting_type: DART
- learning_rate: 0.04
- n_boost_round: 600
- num_leaves: 31 (default)
- min_data_in_leaf: 20 (default)

[10] Microsoft LightGBM <https://github.com/microsoft/LightGBM>












Итоговый PipeLine (есть на github)



Feature importance



Результаты

#	Δ pub	Team Name	Notebook	Team Members	Score 	Entries	Last
1	—	Rinat Kurbanov		 .	0.79016	30	11h
2	—	Valentine Shilov		 .	0.78966	61	11h
3	—	Podoprikhin Maxim		 .	0.78195	23	20h
4	 1	Сергей Кононов		 .	0.77738	25	15h
5	 1	Pitanov Yelisey		 .	0.77228	36	18h
6	—	SVasilyev		 .	0.77003	14	16h
7	—	Dmitri V. Malov		 ..	0.72717	6	10h
		Baseline			0.69329		

Пробовал, но не получилось

Сглаживание по запросам. Для похожих (косинусное расстояние в пространстве эмбедингов) запросов были посчитаны соответствующие фичи, а затем усреднены по ним предсказания LightGBM.

т.е. Query_1 -> Doc_1, ..., Doc_10. Пусть $\text{cosin_sim}(\text{emb}(\text{Query_1}), \text{emb}(\text{Query_2})) = 0.9$. Тогда посчитаем факторы для (Query_2, Doc_j) j=1,10

Затем считаем $f_{ij} = \text{LightGBM}(\text{Query_i}, \text{Doc_j})$ i={1,2} j=1,10

и ранжируем : $f_{1j} + 0.9 * f_{2j}$ j=1,10

Word2vec

Результат: Не получил прироста качества

SEO метрики

- OpenPageRank - плохое качество
- Alexa Index, Yandex Index, Google Index, очень долго (для 10^5 хостов), хотят деньги

Хотел, но не успел

TF-IDF на Map-Reduce. Решил, что приоритетней сделать другие фичи

Новые знания

Нейронные сети быстрее обучаются и прогоняются на GPU.

На ноутбуке недостаточно ресурсов. Использовал Google Colab (очень удобно, есть интеграция с Google Drive; 12GB RAM; GPU; TPU; нужно постоянное соединение; Лимит на вычислительные ресурсы), Kaggle