KDD 2020 Notes

Ilnur Shugaepov
VK.com

August 2020

Оглавление

Этот документ содержит мои заметки о некоторых статьях и туториалах с прошедшего KDD. Заметки оформлены в формате конспектов по типу тех, которые можно найти тут https://vk.com/papersreaders.

В документе 11 полноценных конспектов и еще 10 мини конспектов.

Для себя я выделил несколько направлений, на работы в которых я смотрел в первую очередь:

- Advertising — все что связано с рекламой
- Recommender Systems — задачи связанные с рекомендательными системами
- Deep Learning

Также на конференции было много статей и туториалов, посвещенных обучению на графах.

Highlights

Так как конференция проходила в удаленном формате, то для большинства туториалов были предзаписаны видео, поэтому получилось посмотреть записи почти всех туториалов, которые меня заинтересовали.

Ниже список туториалов, которые показались мне наиболее интересными, с кратким описанием.

Advances in Recommender Systems: From Multi-stakeholder Marketplaces to Automated RecSys

https://sites.google.com/view/kdd20-marketplace-autorecsys/
Some text about tutorial

Causal Inference Meets Machine Learning

http://kdd2020tutorial.thumedialab.com
Some text about tutorial

Neural Structured Learning

Some text about tutorial

Scalable Graph Neural Networks with Deep Graph Library

Some text about tutorial

Building Recommender Systems with PyTorch

https://github.com/pytorch/workshops/tree/master/KDD_2020
Some text about tutorial

Deep Learning for Search and Recommender Systems

https://sites.google.com/view/kdd20tutorial-deepsnr
Some text about tutorial

Часть I

Advertising

Predicting conversions in display advertising based on URL embeddings

Reference: http://papers.adkdd.org/2020/papers/adkdd20-qiu-predicting.pdf
Keywords: user behavior, embeddings, lookalike

Какую задачу решают авторы?

Авторы решают задачу предсказания вероятности конверсии пользователя для **конкретного** рекламодателя.

Интересной особенностью данной работы является то, что в качестве данных пользователя используется только история его активности в интернете — последовательность URL'ов, которые посетил пользователь.

Решение можно разбить на две части:

- Обучение векторных представлений для URL'ов
- Обучение модели, которая используя представления для URL'ов, предсказывает вероятность конверсии

Так как две части по сути независимы друг от друга, то в работе рассматриваются разные варианты для решения каждой из подзадач, и то какие из комбинаций решений являются наиболее хорошими.

Как решают?

Рассмотрим сначала решение каждой из подзадач по отдельности, и начнем с того как авторы предлагают получать векторные представления для URL'ов.

URL Representation Schemes

По аналогии с задачами NLP, авторы предлагают рассматривать последовательность URL'ов как документ, предложениями в котором являются последовательности токенов из URL'ов.
Причем каждый URL состоит максимум из трех токенов, например, https://en.wikipedia.org/wiki/Main_Page/Some_Subpage разбивается на следующие токены [en.wikipedia.org, wiki, Main_Page].

Все способы, которые рассматриваются в работе можно поделить на две группы i) one-hot encoding ii) обучение эмбеддинов

One-hot encoding Вектор для URL'а получается усреднением one-hot векторов токенов, на которые разбит URL.
Так как данный способ не позволяет учитывать семантическую похожесть между URL'ами и размер представления линейно зависит от числа токенов, то авторы рассматривают данный способ в качестве baseline решения.

Embedding learning В рамках данного способа авторы предлагают использовать подход похожий на `Word2Vec (Skip-Gram with Negative Sampling)` с той лишь разницей, что обучаются эмбеддинги не для URL'ов целиком, но для токенов.

В качестве данных для обучения используют истории пользовательской активности. В результате, URL'ы, которые часто оказываются рядом в сессиях находятся близко друг к другу в векторном пространстве.

Для того чтобы из токенов получить векторное представление URL'а, в работе рассматривают следующие варианты

- `Token_concat` — конкатенация векторов токенов
- `Token_avg` — усреднение векторов токенов
- `Domain_only` — используется только вектор домена (первый токен)

Conversion Prediction

Для того чтобы обучить, например, логистическую регрессию для предсказания вероятности конверсии, нужно истории пользователей превратить в фичи (вектор фиксированной длины).

Авторы статьи предлагают несколько вариантов того как это можно сделать

- Усреднение векторов эмбеддингов из истории пользователя
- Однослойная нейросеть поверх усреднения векторов
- Так как предыдущие варианты теряют информацию о хронологии, то в качестве третьего варианта авторы предлагают RNN, на вход которой поступают вектора URL'ов из истории пользователя

Общая архитектура представлена на рисунке 1.1. Итоговая модель обучается на данных о конверсиях пользователей, минимизируя `Cross-Entropy Loss`.

Experiments

В рамках экспериментов, авторы оценивают качество разных комбинаций получения векторов URL'ов и получения фичей из истории пользователя.
Качество оценивают на задаче предсказания вероятности конверсии для пяти рекламодателей. В качестве метрики для оценки качества используют `ROC AUC`.

По результатм экспериментов лучше всего себя показала модель `Token_avg/RNN`.

Преимущества подхода

- Довольно прост в реализации

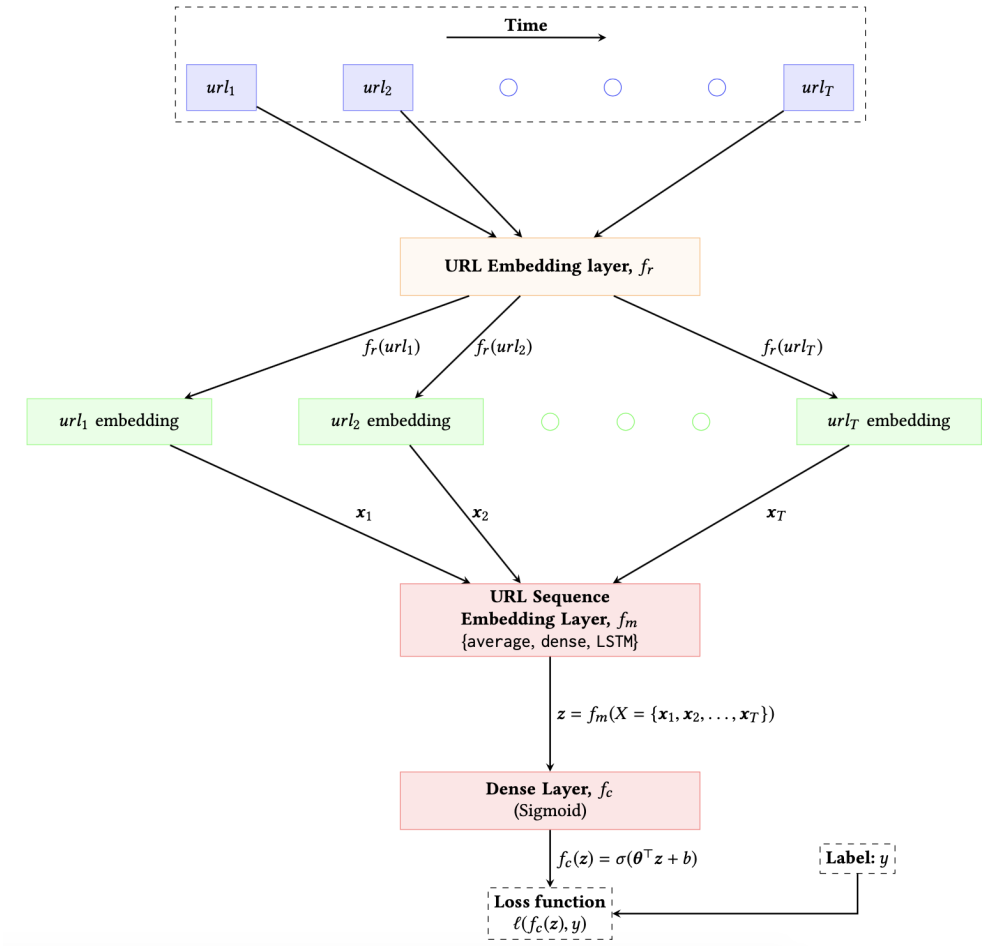

Рис. 1.1: Общая архитектура

Мое мнение

- Хотелось бы увидеть оценку качества на большем числе датасетов. Кажется, что сравнение на пяти рекламодателях это довольно мало для надежной оценки.
- Не смотря на то, что `Token_avg/RNN` оказался лучшим по результатам оффлайн экспериментов, по метрикам видно, что baseline `One_hot/LR` не так уж и сильно отстает по метрикам. Интересно было бы увидеть результаты сравнения данных моделей в онлайне
- Интересно почему авторы не используют дополнительную информацию о пользователях, а ограничились историей серфинга в интернете
- То что предлагают в статье очень похоже на то как я сейчас пробую сделать Lookalike

Key Takeaways:

1. Word2Vec все еще сила в 2020 году

On the Effectiveness of Self-supervised Pre-training for Modeling User Behavior Sequences

Unity

Best paper award from AdKDD
Reference: http://papers.adkdd.org/2020/papers/adkdd20-liao-effectiveness.pdf
Keywords: user behavior, embeddings, CTR prediction

Какую задачу решают авторы?

История активности пользователя (какие сайты посетил, на какие объявления кликал) несет в себе полезные сигналы в том числе о комерческих интересах пользователя.

Использование данной информации может улучшить качество предсказания вероятности клика пользователя по объявлению (CTR prediction), так как позволяет своевременно реагировать на недавние действия пользователя.
Кроме того, в задаче CTR prediction'а очень важно постоянно обновлять модель [3], чтобы можно было реагировать на изменения трендов в данных.

Большинство state-of-the-art моделей на текущий момент - это довольно массивные нейросетевые архитектуры [2, 10, 7], которые довольно тяжело постоянно обновлять, так как это требует много времени и ресурсов, и лишь некоторые модели [10] используют информацию об активности пользователя.

В рамках статьи авторы предлагают нейросетевую архитектуру (см. Рисунок 2.1), состоящую из двух моделей

- **`dense network`** — обрабатывает категориальные и численные признаки связанные с пользователем и объявлением
- **`sequential network`** — обрабатывает активность пользователя

Архитектура позволяет предобучить **`sequential network`** в self-supervised режиме, что позволяет как быстрее переобучать модель для CTR prediction'а, так и добиваться более хороших результатов.

Как решают?

Рассмотрим детальнее что из себя представляет каждая из сетей.

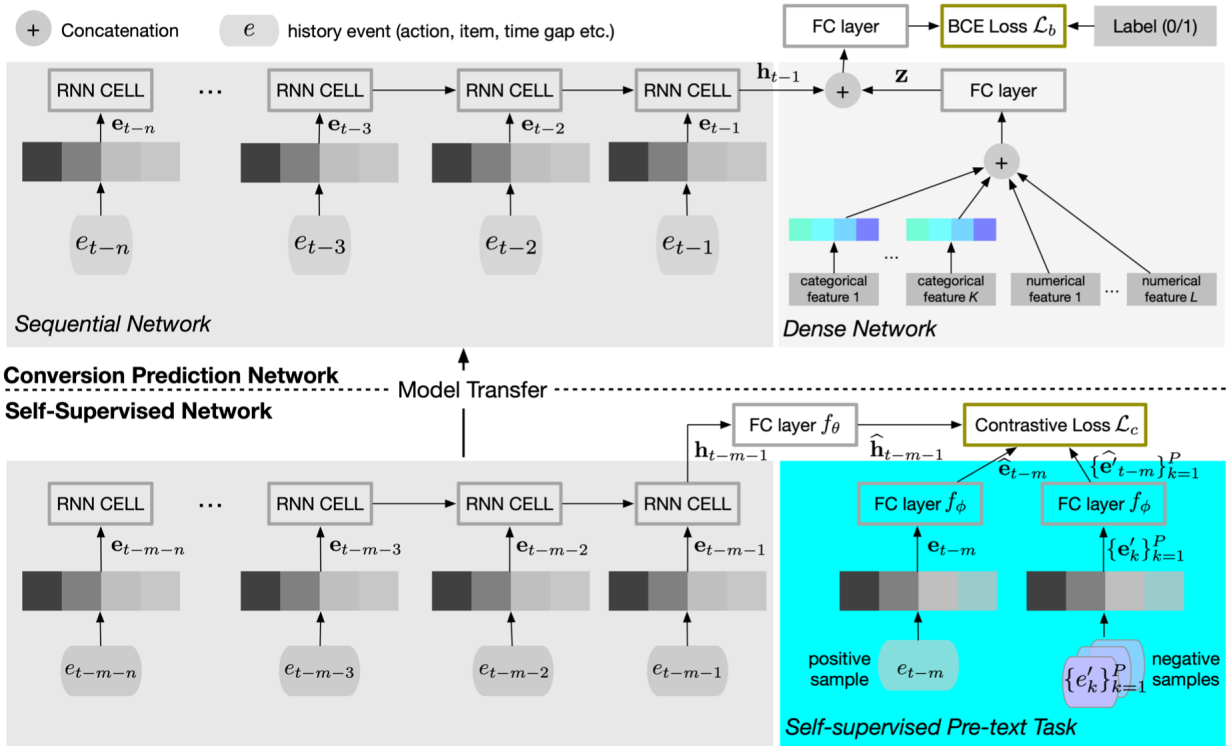

Рис. 2.1: The overview of the proposed conversion prediction and the self-supervised networks. The former consists of the sequential and the dense networks. The latter is learned by the guidance of the proposed pre-text task and it serves as the pre-trained model for the sequential network.

Dense Network

`Dense network` имеет уже довольно привычный для задачи CTR prediction'а вид (см. Рисунок 2.1).

Для категориальных фичей обучаются эмбеддинги, которые конкатенируются с числовыми признаками и подаются на вход нескольким полносвязным слоям.

Remark: в рамках работы в `dense network` подают на вход только ранжируемое объявление.

Sequential Network

Для того чтобы ускорить переобучение модели CTR prediction'а, авторы предлагают предобучать `sequential network` в self-supervised режиме на задаче предсказания следующего события в сессии пользователя (см. Рисунок 2.1 нижняя часть).

Важно заметить, что модель обучают не напрямую предсказывать следующее событие, а обучают различать релевантные события от нерелевантных, то есть для обучения используют `contrastive loss` (Очень похоже на разницу между `CBOW` и `Skip-Gram w/ Negative Sampling` моделями для Word2Vec).

Использование `contrastive loss` имеет следующие преимущества в сравнении с задачей многоклассовой классификации

- Обучение становится существенно быстрее, когда классов очень много. На практике, если

пробовать предсказать следующий посещенный сайт, то классов могут быть миллионы. Ускорение достигается за счет того, что нет необходимости вычислять softmax для расчета итоговых вероятностей.

- Модель более устойчива к шуму в данных для обучения

Итоговая модель

Промежуточные представления полученные с помощью `dense network` и `sequential network` конкатенируются и передаются на вход в полносвязный слой.

Модель обучается оптимизируя классический для задачи CTR-prediction'а лосс - `Binary Cross-Entropy loss`.

За счет того что `sequential network` уже была предобучена, то для обучени итоговой модели нужно как меньше данных, так и меньше времени.

Преимущества подхода

На мой взгляд, самое важное преимущество, о котором почти не написано в статье, заключается в следующем:

Так как модель декомпозирована на `dense network` и `sequential network`, то на этапе ранжирования нет необходимости в том, чтобы запрашивать недавнюю активность пользователя и пропускать ее через `sequential network`.
Промежуточное представление получаемое с помощью `sequential network` можно предпосчитывать и обновлять для пользователя независимо всякий раз, когда мы получаем информацию о новых действиях пользователя, что случается гораздо реже чем ранжирование. И на этапе ранжирования достаточно для пользователя запросить уже посчитанное промежуточное представление.

Это существенно уменьшает время необходимое для ранжирования объявлений, так как фактически на данном этапе используется только `dense network`.

В статье [6] пытались добиться такой же декомпозиции, но итоговое решение получилось очень сложным.

Мое мнение

- Очень хотелось бы в качестве одного из baseline'ов увидеть решение, где в качестве `sequential network` используют просто усреднение вектора объектов из активности пользователя (вектора можно было бы обучть с помощью Word2Vec).
- Также было бы интересно увидеть в качестве `sequential network` и другие архитектуры, например, Bert4Rec.
- Жаль, что совсем не сравнивают свое решение с другими архитектурами на публичных датасетах.

General-Purpose User Embeddings based on Mobile App Usage

Tencent

Reference: [9]
Keywords: user behavior, embeddings, deep learning

Какую задачу решают авторы?

Данныи об использовании пользователем мобильных приложений несут в себе много информации о его интересах (см. Рисунок 3.1).
Список установленных приложений, установки, удаления — все это хорошие индикаторы как долгосрочных так и кратковременных интерсов пользователя.

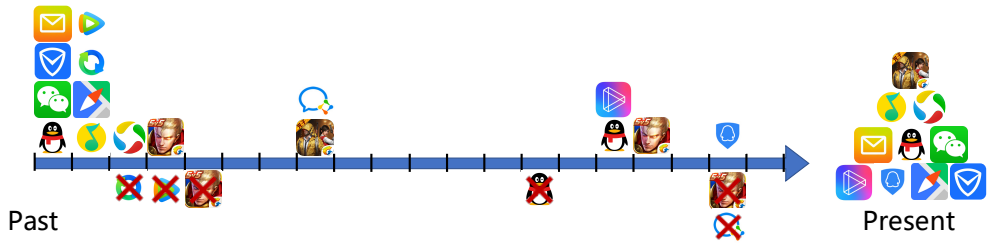

Рис. 3.1: Illustration of retention, installation, and uninstallation. Operations of (un)installation are low-frequency and unevenly distributed over time.

При решении приктических задач, таких как рекомендации, предсказание вероятности клика (CTR prediction), поиск релевантной аудитории (lookalike), можно получить дополнительный прирост качества при использовании информации об использовании пользователем мобильных приложений.
Как правило, данную информацию добавляют в модели путем feature engineering'а, что может быть довольно сложно.

В работе представлена модель, которая по данным об использовании приложений, позволяет построить General-Purpose эмбеддинг для пользователя.
Полученные эмбеддинги можно легко использовать для решения практических задача, избегая сложного feature engineering'а.

Как решают?

Модель описанная в работе довольно сложная, поэтому мне бы хотелось рассмотреть один из бэйслайнов, приведенных в работе, который, не смотря на свою простоту, позволяет получать очень хорошие результаты.

Основная идея Эмбеддинги для пользователя можно получить если скормить список установленных приложений в DAE (см. Рисунок 3.2).

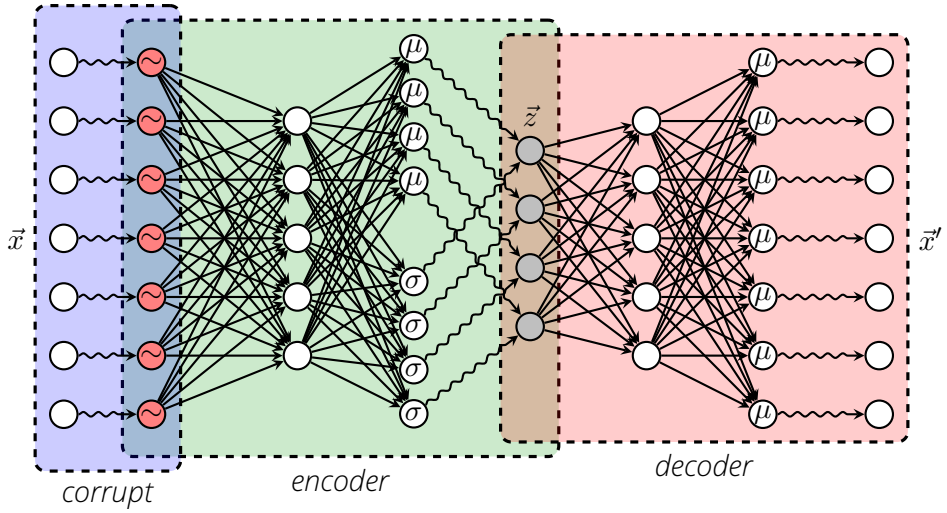

Рис. 3.2: Denoising Autoencoder (DAE)

Data Preprocessing Прежде обучать модель, нужно выполнить предварительную чистку данных, чтобы эмбеддинги пользователей лучше отражали их интересы:

- Игнорируются наиболее популярные приложения, которые есть практически на всех смартфонах, так как факт их наличия на телефоне мало что говорит об интересах пользователя
- Игнорируются предустановленные на телефон приложения
- Также исключаются совсем редкие приложения, которые установлены у маленького числа пользователей

Experiments

В статье представлен ряд экспериментов, подтверждающих полезность полученных эмбеддингов пользователей.
Рассмотрим два эксперимента

Test #1: Next Week's Installation Prediction В рамках первого теста, эмбеддинги пользователя используют для предсказания того какие приложения пользователь установит в течении следующей недели.
Для решения задачи обучают нейросеть, состоящую из нескольких полносвязных слоев, на эмбеддингах пользователей.

Test #2: Look-alike Audience Extension В рамках второго теста, эмбеддинги используются для расширения аудитории.
Данные для обучения — аудитория из миллиона пользователей, примерно 10% из которых - seed (пользователи, которые используют некоторое нишевое приложение).
Задача состоит в том, чтобы научиться искать пользователей, похожих на seed.
Для этого на эмбеддингах пользователей обучается классификатор (XGBoost), который предсказывает вероятность того, что пользователь относится к seed пользователям.

В качестве метрики для оценки качества использовали ROC AUC.

Не смотря на простоту подхода, использование DAE показывает результаты, которые не намного хуже в сравнении с моделью описанной в статье.

Преимущества подхода

- Простой в реализации подход
- Не требует частого переобучения модели целиком, так как новые приложения появляются не так часто
- Позволяет практически в онлайне обновлять эмбеддинги для пользователей при установке/удаление приложений, что позволяет лучше реагировать на изменение интересов пользователя

Hubble: an Industrial System for Audience Expansion in Mobile Marketing

Reference: https://dl.acm.org/doi/pdf/10.1145/3394486.3403295
Keywords: embeddings, lookalike, audience expansion, graphs, knowledge distillation

Какую задачу решают авторы?

Авторы решают довольно популярную в области интернет-рекламы задачу — расширение аудитории.

Часто, у рекламодателя может быть список пользователей, которые положительно отреагировали на предыдущую рекламную кампанию (seed множество).
Для того чтобы показывать новое объявление пользователям, которые с большей вероятностью совершат целевое действие, рекламная система может помочь рекламодателю и найти пользователей, которые похожи на seed множество.

На практике при решении данной задачи разработчики сталкиваются со следующим основным челенджем:

- система должна хорошо масштабироваться на случай большого числа кампаний и пользователей, и выдавать расширенную аудиторию в течении нескольких минут

В рамках работы авторы описывают two-stage подход, где в оффлайне обучают эмбеддинги для пользователей и рекламных кампаний, и в онлайне обучают легковесные модели для расширения аудитории.
Основным недостатков предыдущих работ [1, 4] с two-stage подходом является недостаточно хорошее качество пользовательских эмбеддингов, которые не отражают все многообразие интересов пользователя.

В работе представлен ряд оффлайн и онлайн экспериментов, где показано превосходство нового метода над существующими state-of-the-art подходами к решению задачи расширения аудитории.

Как решают?

Общая архитектура системы Hubble представлена на Рисунке 4.1
Рассмотрим подробнее, что происходит в рамках онлайн и оффлайн фаз.

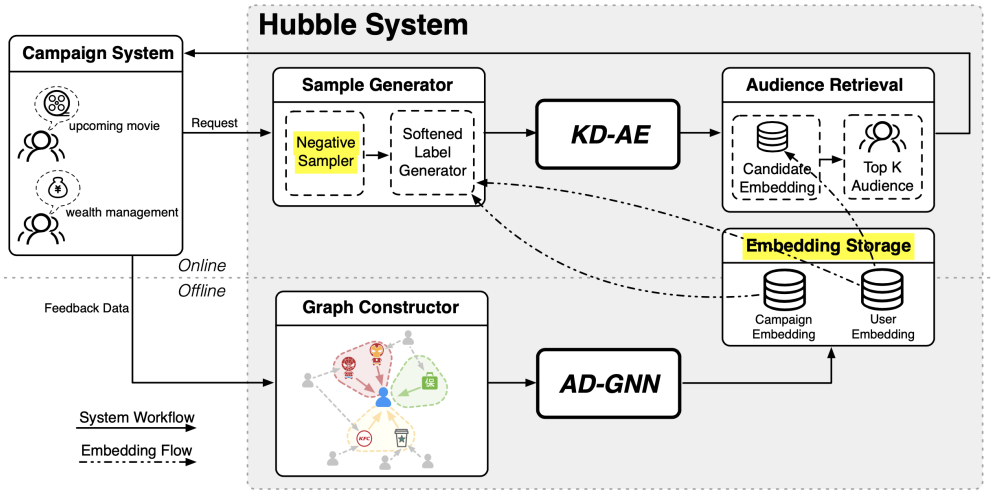

Рис. 4.1: The architecture of Hubble System.

Offline Representation Learning

В оффлайн части решения можно выделить две основные части: User-Campaign граф и модель AD-GNN для обучения эмбеддингов пользователей и кампаний.

User-Campaign граф Данный граф по сути представляет из себя двудольный граф, в котором между пользователем и кампанией есть ребро, если пользователь совершил какое-то целивое действие, например, кликнул по объявлению, в рамках рекламной кампании.

Можно сказать, что данный граф содержит в себе информацию о различных комерческих интересах пользователя.
Если обучить эмбеддинги для вершин User-Campaign граф, то ожидается, что близкие эмбеддинги пользователей будут соответствовать пользователям с похожими интересами.

AD-GNN Не вдаваясь в подробности архитектуры, рассмотрим как обучаются эмбеддинги для пользователей и рекламных кампаний.

Эмбеддинги обучаются на задаче *link-prediction* — пытаемся для пользователя u и кампании c обучить такие вектора $\mathbf{h}_u, \mathbf{h}_c$, что $y_G := \sigma(\mathbf{h}_u \mathbf{h}_c)$ дает нам вероятность того, что между пользователем и кампанией есть ребро в графе.
По сути y_G принимает большое значение для близких векторов, и маленькое для далеких.

В качестве положительных примеров для обучения используются пары (u, c) соотвествующие ребрам в User-Campaign графе, а в качестве негативных такие пары (u', c'), где пользователь u' видел рекламу в рамках кампании c', но не совершил целивое действие.

По сути перед нами задача бинарной классификации, поэтому во время обучения минимизируется обычный Binary Cross-Entropy Loss.

После того как были обучены эмбеддинги для пользователей и кампаний они загружаются в Embedding Storage (см. Рисунок 4.1).

Online Audience Expansion with Knowledge Distillation

Система получает аудиторию $\mathcal{S}_c$, которую нужно расширить. Если для кампании c уже есть эмбеддинг, то есть если она присуствует в User-Campaign графе, то для расширения аудитории достаточно найти пользователей с большим скором $\sigma(\mathbf{h}_u\mathbf{h}_c)$.

На практике, часто создаются новые кампании, для которых еще нет вершины в графе. В данной ситуации обучается классификатор, который предсказывает вероятность того, что пользователь совершит целевое действие.
Для обучения модели в онлайне генерируется множество случайных пользователей $\mathcal{U}_n \subset \mathcal{U} \setminus \mathcal{S}_c$ (см. Рисунок 4.1), и модель обучается на эмбеддингах различать пользователей из $\mathcal{S}_c$ от $\mathcal{U}_n$.

Для того чтобы в онлайн модель передать больше знаний от оффлайн модели AD-GNN, авторы используют *knowledge distillation*, то есть обучают онлайн модель не только на *hard* лэйблах (1 - если пользователь из $\mathcal{S}_c$ и 0 - в противном случае), но и на *soft* лэйблах, которые получают от AD-GNN.
Soft лэйблы получают следующим образом.

1. Выполняется поиск похожих кампаний на кампанию c. Делается это с помощью разбиения пользователей из $\mathcal{S}_c$ на k кластеров и последующим поиском наиболее похожей кампании на каждый кластер. В итоге получаем множество $\{c_i\}_{i=1}^k$ из k кампаний.
2. Soft label для пользователя - средняя похожесть на вектора для кампаний из множества $\{c_i\}_{i=1}^k$.

Итоговый лосс выглядит следующим образом:

$$\begin{aligned}\mathcal{L} = &- \frac{1}{|\mathcal{D}|} \sum_{(u,y_h,y_s)\in\mathcal{D}} y_h \log(\hat{y}) + (1-y_h)\log(1-\hat{y}) \\ &- \frac{\gamma}{|\mathcal{D}|} \sum_{(u,y_h,y_s)\in\mathcal{D}} y_s \log(\hat{y}) + (1-y_s)\log(1-\hat{y}) \\ &+ \frac{\lambda}{2}\|\Theta\|_2,\end{aligned}$$

где $\hat{y} = MLP(\mathbf{h}_u)$, y_h - hard label, y_s - soft label и γ - гиперпараметр, который контролирует степень переноса знаний из AD-GNN в онлайн модель.

В рамках экспериментов авторы показывают, что при $\gamma > 0$ улучшается качество онлайн модели, то есть модель действительно получает дополнительное полезное знание от AD-GNN.

Experiments

Во-первых, авторы показывают, что система работает лучше текущих state-of-the-art решений для задачи расширения аудитории.
Во-вторых, демонстрируют, что предложенная архитектура сети AD-GNN работает лучше других sota решений на задаче *link-prediction*.
И наконец, показывают многократное ускорение по времени в онлайне, которое достигается в основном за счет того, что на вход моделе поступают эмбеддинги пользователей, размерность которых существенно меньше размерности обычного описания с помощью фичей — 64 vs 300k.

Мое мнение

Тут https://vk.com/@papersreaders-finding-users-who-act-alike-transfer-learning-for-expanding можно почитать конспект статьи от Pinterest с KDD'19, в которой описано решение [1].

AutoFIS: Automatic Feature Interaction Selection in Factorization Models for CTR Prediction

Huawei

Reference: https://arxiv.org/abs/2003.11235
Keywords: CTR prediction, factorization machines, neural architecture search
Обзорный коспект про CTR prediction:
https://vk.com/@papersreaders-a-sparse-deep-factorization-machine-for-efficient-ctr-predic

Какую задачу решают авторы?

С появлением Factorization Machines (FM) стало понятно, что добавление в линейную модель взаимодействий второго порядка между признаками существенно улучшает качество модели.
Но также известно, что среди всех возможных пар признаков полезными является лишь небольшая часть (20-50%).
Поэтому добавление в модель всех пар признаков и учет их с одинаковым весом является субоптимальным решением.

Авторы решают задачу поиска хороших пар признаков в FM-like моделях.
Это не первая попытка решить данную задачу для FM-like моделей (см. [8, 5]).
Но в отличие от предыдущих работ, предложенный подход решает эту задачу лучше.

Как решают

Решение состоит из двух фаз: *search stage* и *re-train stage*.
В рамках search stage алгоритм находит хорошие пары признаков, а в рамках re-train stage FM-like модель обучается уже с использованием только отобранных пар.
Рассмотрим чуть подробнее каждую из фаз.

Search stage Модель для поиска хороших пар признаков выглядит следующим образом:

$$l_{\text{AutoFIS}} = \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \alpha_{(i,j)} \langle \mathbf{e}_i, \mathbf{e}_j \rangle ,$$

по сути обычная FM модель, с той лишь разницей, что перед скалярным произведением векторов признаков $\mathbf{e}_i, \mathbf{e}_j$ стоит обучаемый вес $\alpha_{(i,j)}$.

Модель обучается на Binary Cross-Entropy Loss, но с использованием GRDA Optimizer'a, который позволяет получить sparse решение, то есть большая часть коэф α будет равна нулю.
Соотвественно $\alpha_{(i,j)}$ принимает не нулевое значение для важных пар признаков.

Re-train stage После того как были отобраны пары признаков, модель дообучается (тюнятся как веса модели, так и коэффеиценты α, но только для отобранных признаков).

Преимущества подхода

Решение является довольно робастным, то есть запуск search stage с разным seed'ом приводит к похожим результатам.
Найденные пары признаков можно легко использовать при обучении любых других FM-like моделей.

Мое мнение

Интересно, что на сайте https://paperswithcode.com/sota/click-through-rate-prediction-on-criteo AutoFIS не показывает sota результаты.
Более того, незадолго до начала конференции KDD, авторы из Yahoo Research опубликовали работу DeepLight https://arxiv.org/abs/2002.06987, которая показывает sota результаты на датасете Criteo.

Другие работы

Пара работ про совместное ранжирование органического контента и рекламы.

Jointly Learning to Recommend and Advertise

Reference: https://arxiv.org/abs/2003.00097

Ads Allocation in Feed via Constrained Optimization

LinkedIn

Reference: https://dl.acm.org/doi/pdf/10.1145/3394486.3403391

Литература

[1] Stephanie deWet and Jiafan Ou. Finding users who act alike: Transfer learning for expanding advertiser audiences. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2251–2259, 2019.

[2] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[3] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9, 2014.

[4] Yudan Liu, Kaikai Ge, Xu Zhang, and Leyu Lin. Real-time attention based look-alike model for recommender system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2765–2773, 2019.

[5] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference*, pages 1349–1357, 2018.

[6] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. Practice on long sequential user behavior modeling for click-through rate prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2671–2679, 2019.

[7] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

[8] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617*, 2017.

[9] Junqi Zhang, Bing Bai, Ye Lin, Jian Liang, Kun Bai, and Fei Wang. General-purpose user embeddings based on mobile app usage. *arXiv preprint arXiv:2005.13303*, 2020. URL https://arxiv.org/pdf/2005.13303.pdf.

[10] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1059–1068, 2018.

Часть II

Recommender Systems

On Sampled Metrics for Item Recommendation

Google Research

Best research paper award
Reference: http://walid.krichene.net/papers/KDD-sampled-metrics.pdf
Keywords: ranking evaluation, metrics, sampling

Какую задачу решают авторы?

Типичный протокол оценки качества рекомендательных систем выглядит следующим образом:

1. Для отобранного множества пользователей D ранжируем алгоритмом A все множество кандидатов, состоящее из n объектов
2. Для каждого пользователя $\mathbf{x}$ вычисляем $R(A, \mathbf{x})$ — множество позиций релевантных объектов
3. После чего для пользователя считаем метрику M, например, `ROC AUC` или `Precision@K`, `Recall@K`

Итоговое значение метрики получается усредненеим метрик посчитанных по всем пользователям

$$\frac{1}{|D|} \sum_{\mathbf{x} \in D} M(R(A, \mathbf{x})).$$

В ситуации когда n велико часто прибегают к сэмплированию — вместо того чтобы ранжировать все n кандидатов, ранжируют случайную подвыборку из m объектов ($m \ll n$) вместе с релевантными для пользователя объектами.

Ожидается, что метрики посчитанные с сэмплированием позволяют упорядочить алгоритмы ранжирования по качеству так же как и метрики посчитанные без сэмплирования.
Авторы статьи впервые тестируют это предположение и показывают что для большинства используемых метрик оно не верно, даже при многократном сэмплировании и усреднении результатов.

В статье предложены скорректированные варианты привычных метрик, которые позволяют при использовании сэмплирования сортировать алгоритмы по качеству также как если бы сэмплирования не было.

Как решают?

Remark: для простоты можно считать, что для каждого пользователя есть один релевантный объект.

Работу можно разбить на три части

1. Экспериментальная часть, где проверяют предположение, что метрики с сэмплированием и без упорядочивают алгоритмы ранжирования одинаковым образом
2. Теоретическая часть, посвященная скорректированным вариантам метрик
3. Экспериментальная часть, где авторы показывают, что скорректированные варианты метрик работают

Inconsistency of Sampled Metrics

Ключевое определение данной работы

Definition 1 (Consistency): *Let the evaluation data D be fixed. A metric M is consistent under sampling if the relative order of any two recommenders A and B is preserved in expectation. That is, for all A, B,*

$$\frac{1}{|D|}\sum_{\mathbf{x}\in D} M(R(A,\mathbf{x})) > \frac{1}{|D|}\sum_{\mathbf{x}\in D} M(R(B,\mathbf{x}))$$
$$\iff \mathbb{E}\left[\frac{1}{|D|}\sum_{\mathbf{x}\in D} M(\widetilde{R}(A,\mathbf{x}))\right] > \mathbb{E}\left[\frac{1}{|D|}\sum_{\mathbf{x}\in D} M(\widetilde{R}(B,\mathbf{x}))\right]$$

В рамках первой группы экспериментов (см. Рисунок 7.1) показывают как меняются метрики для разных алгоритмов в зависимости от m (число случайно отобранных объектов).

Из экспериментов видно, что единственная консистентная метрика — `ROC AUC`.

Corrected Metrics

Очень рекомендую самостоятельно ознакомиться с данной частью работы, так как ее не очень удобно излогать в виде конспекта.

Выводы

Key Takeaways:

1. Нужно избегать сэмплирования при расчете метрик
2. Если нет возможности избежать сэмплирования, то нужно использовать скорректированные версии метрик
3. Вычисление метрик нужно повторять несколько раз с разным seed'ом для того чтобы уменьшить дисперсию

Все это в лишний раз наводит на мысли о том, что в ряде работ лучшими оказались алгоритмы, которые не обязательно являются лучшими на самом деле, и на результаты экспериментов в статьях всегда надо смотреть с определенной долей скепсиса.

Еще одна работа примерно про тоже On Sampling Top-K Recommendation Evaluation https://dl.acm.org/doi/pdf/10.1145/3394486.3403262 с этой же конференции.

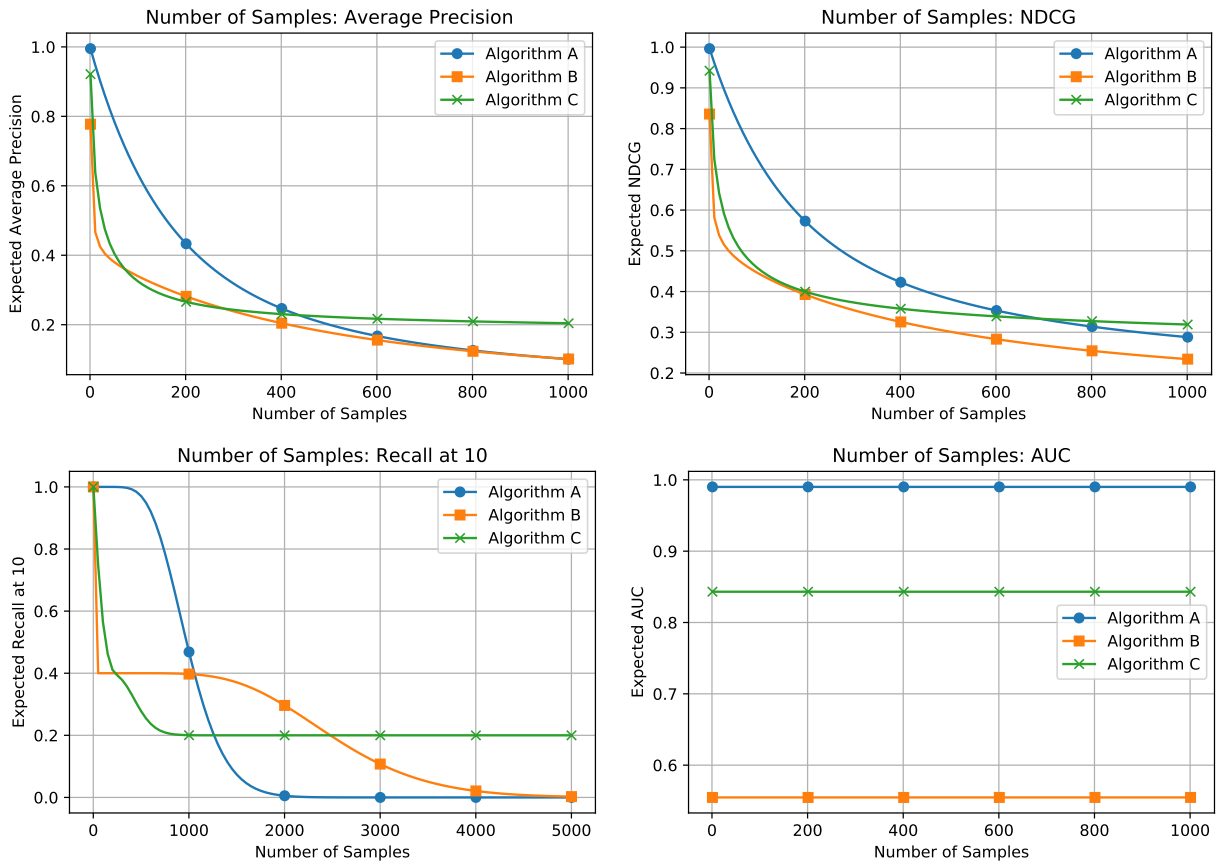

Рис. 7.1: Expected sampling metrics for the running example while increasing the number of samples. For Average Precision, NDCG and Recall, even the relative order of algorithm performance changes with the number of samples. That means, conclusions drawn from a subsample are not consistent with the true performance of the algorithm.

Temporal-Contextual Recommendation in Real-Time

Amazon

Best ADS paper
Reference: https://dl.acm.org/doi/pdf/10.1145/3394486.3403278
Keywords: recommender systems, recurrent neural networks, hybrid model

Какую задачу решают авторы?

Разработчики современных рекомендательных систем сталкиваются со следующими челенджами: Система должна

- Оперативно реагировать на изменение интересов пользователя
- Обучаться на историях большого числа пользователей, состоящих из сотен событий, за разумное время
- Быть эффективной для новых пользователей и объектов (cold-start problem)
- Хорошо масштабироваться на случай ранжирования большого числа объектов

Многие популярные решения не удовлетворяют всем указанным требованиям.
Например, решения, в основе которых лежит факторизация матрицы `Users-Items`, не позволяют оперативно реагировать на изменение интересов пользователя, и не могут строить рекомендации для новых пользователей/объектов.

В статье авторы предлагают решение, которое удовлетворяет всем перечисленным требованиям.
В основе решения — `RNN-like` архитектура, которая наряду с идентификаторами пользователя и объекта использует доступную мета-информацию (признаки пользователя и объекта).
Для того чтобы модель можно было за разумное время обучить на большом каталоге объектов, авторы предлагают использовать `Negative Sampling` [3] (NS) вместо классической многоклассовой классификации.

Как решают?

Не смотря на то, что в статье описано много интересного, я хотел бы уделить внимание только одному моменту в работе — использование NS для ускорения обучения.

Negative Sampling

Сначала ответим на вопрос почему обучение моделей рекомендаций при большом размере каталога товаров может быть медленным.

Рассмотрим следующую постановку задачи рекомендаций: пусть у нас есть история пользователя $\{i_1, \dots, i_k\}$, например, какие товары он уже купил, наша задача научиться предсказывать следующий товар i_{k+1}, который заинтерисует пользователя.

Довольно часто задачу в такой постановке решают сводя ее к задаче *многоклассовой классификации*, где классам соответствуют товары из каталога.
Детальнее, пусть каталог состоит из m товаров,

1. Используя модель, мы получаем скрытое представление $\mathbf{h}_k \coloneqq \mathbf{h}(i_1, \dots, i_k) \in \mathbb{R}^d$, описывающее историю пользователя
2. Полученное представление $\mathbf{h}_k$ домножаем на матрицу $W \in \mathbb{R}^{m \times d}$, чтобы получить скор для каждого товара в каталоге
3. Итоговые вероятности для товаров

$$\hat{y} \coloneqq \mathrm{softmax}(W \cdot \mathbf{h}_k)$$

Модель $\mathbf{h}$ и матрица W обучаются оптимизируя `Cross Entropy Loss`.

Домножение вектора $\mathbf{h}_k$ на матрицу W с последующим softmax'ом занимает много времени, кроме того, матрица W занимает очень много места.

При использовании NS [3] нам не нужно вычислять скоры для всех товаров, а только для релевантного товара и для небольшого случайного множества негативных примеров, так как решаем задачу *бинарной классификации* — учимся отличать релевантные товары от нерелавнтных.
Это позволяет не хранить матрицу W и существенно ускоряет процесс обучения.

Experiments

В рамках экспериментов авторы исследуют то как зависит скорость обучения (#items/sec) при использовании NS от размера каталога.

Авторы замерели скорость обучения и метрики на датасетах MovieLens (1m, 10m, 20m) и на датасете Taobao, при использовании Dense output слоя и при использовании NS.
На датасетах MovieLens модель обучали в течении четырех эпох. На датасете Taobao время на обучение было ограничено шестью часами.

Исходя из результатов (см. таблицу 8.1), можно сделать вывод, что использование NS существенно ускоряет обучение в случае большого каталога объектов (Taobao).
Кроме того, при ограничениях на время обучения, использование NS позволяет добиться лучшего качества за счет того, что модель успевает обработать большее число объектов.

	Output	ml-1m	ml-10m	ml-20m	Taobao
Output size	IS	62	255	362	1087
(m)	Dense	1683	65134	131263	1183451
Throughput	NS	$\underline{23\text{k}}$	$\mathbf{20\text{k}}$	$\mathbf{20\text{k}}$	$\mathbf{7.8\text{k}}$
(#items/sec)	Dense	$\mathbf{23\text{k}}$	$\mathbf{23\text{k}}$	$\mathbf{17\text{k}}$	631
PPL	NS	$\underline{377}$	$\underline{405}$	$\underline{455}$	$\mathbf{17.6\text{k}}$
	Dense	409	439	494	$\mathbf{119\text{k}}$
NDCG	NS	0.128	0.123	0.12	$\mathbf{0.15}$
	Dense	0.123	0.119	0.115	0.08

Рис. 8.1: Model throughput and accuracy. Bold-fonts/underlines show significant/insignificant differences, respectively. NS significantly improved model training for up to 1M unique items (Taobao dataset). Additionally, the dense model for the Taobao dataset requires >16GB memory for BPTT, which is infeasible on many GPUs.

Наряду с привычными метриками для данной задачи, в статье используют еще и **Perplexity**.
Remark: A recommendation system with a PPL of p is equivalent to one that recommends a uniform random selection of p items, one of which is the true next item.

Выводы

Key Takeaways:

1. Если каталог объектов очень большой ($\geq 10^5$), то для обучения модели ранжировани лучше использовать `Negative Sampling`. Это позволяет не только ускорить обучение, но и существенно уменьшить размер модели.

Learning to Cluster Documents into Workspaces Using Large Scale Activity Logs

Google

Refernce: https://dl.acm.org/doi/abs/10.1145/3394486.3403291 [2]
Keywords: user behavior, embeddings, clustering

Какую задачу решают авторы?

Авторы решают задачу кластеризации документов в Google Drive в workspace'ы — отдельные папки, содержащие документы похожие не только по смыслу, но и связанные с конкретными задачами пользователя.

У классических подходов к кластеризации документов есть недостатки связанные с тем, что они могут группировать вместе документы одной тематики, но при этом они не учитывают большое количество других факторов.
Например, если пользователь работает над конкретной задачей, то он вполне может использовать документы с разной тематикой.

В рамках данной работы, авторы не решают задачу кластеризации напрямую, но обучают document similarity модель, которая для двух документов предсказывает относятся они к одному кластеру или нет.
Используя предсказания модели, авторы используют иерархическую кластеризацию для группировки документов в workspace'ы.

Главный вопрос: где взять данные для обучение document similarity модели?
На практике, получение датасета достаточного размера, состоящего из пар документов с разметкой от пользователей, может быть довольно сложно.

Авторы предлагают способ получить разметку для обучения напрямую из логов активности пользователей — сводят задачу к weakly supervised варианту.
Основное предположение, которое позволяет получить weak разметку — если пользователь выполнял действия с документами с небольшой разницей во времени (*co-access*), то скорее всего эти документы относятся к одной задаче, над которой работает пользователь.

Авторы в онлайне сравнивают свое решение с классическими unsupervised подходами для кластеризации документов, и показывают, что предлагаемое решение существенно лучше бэйслайнов.

Как решают?

Рассмотрим сначала детальнее то как авторы предлагают получать weak разметку, а затем на предлагаемую в статье document similarity модель.

Activity-Based Labels and Weak Supervision

Для получения разметки, авторы отталкиваются от предположения о похожести документов, с которыми пользователь работал в течении небольшого промежутка времени.
Два документа считаются co-accessed, и получают $label = 1$, если были открыты с разницей во времени не больше двух минут.

На рисунке 9.1 показано на каких данных обучается document similarity модель.

По сути, вся активность пользователя делиться на две части: первая используется для построения признаков и обучения document similarity модели, а вторая — для получения разметки.

Документы считаются похожими, если они были co-accessed в рамках второй части активности пользователя.

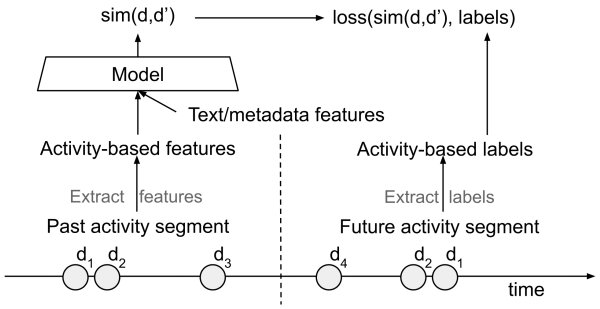

Рис. 9.1: Past and future activity segments used for extracting activity-based features and labels respectively

Document Similarity Model

Рассмотрим подробнее как выглядит модель (см. Рисунок 9.2).

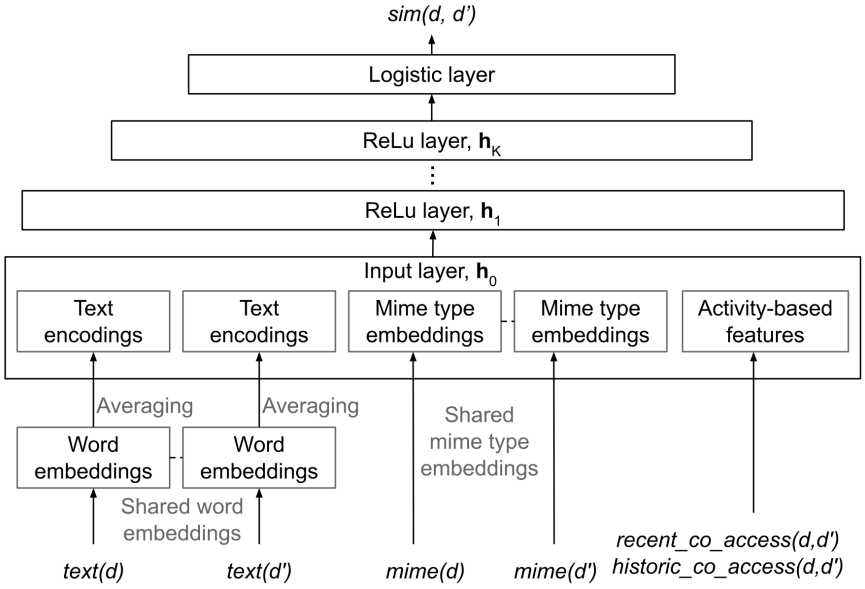

Рис. 9.2: Document Similarity Model

Список признаков, на которых обучается модель:

- Text features
  - $text(d)$ - text content for document d
  - $text(d')$ - text content for document d'
- Metadata features
  - $mime(d)$ - MIME type of the document d
  - $mime(d')$ - MIME type of the document d'
- Activity-based features
  - $recent\_co\_accesses(d, d')$ - number of co-accesses between d, d' in the past 2 weeks
  - $historic\_co\_accesses(d, d')$ - number of co-accesses between d, d' in the past 4 weeks
- Human and activity-based labels
  - $co\_cluster(d, d')$ - human labels on whether d, d' should be clustered together in a workspace
  - $future\_co\_accesses(d, d')$ - number of co-accesses between d, d' in the future week

Авторы показывают, что дополнительное использование Activity-based признаков существенно улучшает качество модели, то есть полезно использовать не только признаки связанные с документами, но и информацию о том как пользователь с ними взаимодействовал.

DSM обучается минимизируя, привычный для задачи бинарной классификации, Binary Cross-Entropy Loss.

Experiments

Авторы удтверждают, что documents similarity модель обученная на weak разметке, по качеству не уступает моделе, обученной на разметке от пользователей.

Кроме того, авторы проводят онлайн эксперимент и оценивают качество предложенного алгоритма кластеризации со следующими бэйслайнами:

- Topicality - для документов обучаются эмбеддинги, которые потом кластеризуются
- Calendar - относит в один кластер документы, прикрепленные к одному событию в календаре
- Favorites - группирует вместе документы, которые пользователь открывает наиболее часто

По результатам экспериментов, предложенный подход работает намного лучше бэйслайнов.

Мое мнение

Красивый и простой способ извлечения weak лэйблов, для обучения модели похожести документов, из большого количества логов активности пользователя.

Другие работы

Embedding-based Retrieval in Facebook Search

Facebook

Reference: https://arxiv.org/abs/2006.11632
Конспект: https://vk.com/@papersreaders-embedding-based-retrieval-in-facebook-search

Исторически поиск в Facebook работал на основе Boolean matching model.
В статье авторы делятся опытом перехода к использованию embedding-based системы на этапе отбора кандидатов перед ранжированием.

В основе предлагаемого решения модель построения эмбеддингов запросов и документов.

Работа наглядно показывает, что даже относительно несложное решение для Embedding-based retrieval показывает существенное улучшение в сравнении с классическим Boolean matching подходом в задаче отбора кандидатов.

Controllable Multi-Interest Framework for Recommendation

Alibaba

Reference: https://arxiv.org/abs/2005.09347
Конспект: https://vk.com/@papersreaders-controllable-multi-interest-framework-for-recommendation

Современные рекомендательные системы используют историю пользователя для построения вектора, который описывает пользователя и используется для поиска объектов-кандидатов при построении рекомендаций.
Однако использование единственного вектора для описания пользователя не позволяет уловить его разнообразные интересы.

Авторы статьи предлагают решение, которое позволяет представить пользователя в виде набора из K векторов, каждый из которых соответствует некоторому интересу пользователя.
Данный подход позволяет делать рекомендации как более точными так и более разнообразными в сравнении с существующими state-of-the-art подходами.

PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest

Pinterest

Reference: https://arxiv.org/pdf/2007.03634.pdf

Статья от Pinterest про прокачку системы рекомендаций пинов для пользователя.

Авторы статьи рассматривают проблемы связанные с представлением пользователя в виде единственного вектора.
Для решения проблем, в статье предлагают представить пользователя в виде набора векторов.

Ключевые отличия от предыдущих работ, предлагающих сделать тоже самое:

1. количество векторов для пользователя не фиксировано
2. вектора для пользователей не обучаются совместно с векторами для пинов

Для того чтобы представить пользователя в виде набора векторов, предлагают делать иерархическую кластеризацию активности пользователя за последнее время (вектора пинов получены black-box моделью).
Каждому кластеру ставят в соответствие его важность.

Для рекомендации релевантных пинов берут 3 наиболее важных кластера и ищут похожие пины с помощью приближенного поиска ближ соседей.

Как и в большинстве последних статей от Pinterest, авторы рассматривают продакшн решение, поэтому достаточно внимания уделяют вопросу о том как все это тащить в прод.

Improving Recommendation Quality in Google Drive

Google

Reference: https://research.google/pubs/pub49272/

Статья про попытки прокачать качество инструмента Quick Access (статья не научная, а просто про опыт и проведенные эксперименты)

В целом, то о чем они пишут очень похоже на наш опыт и на наши эксперименты за последние года полтора.

Эксперименты описанные в статье, которые похожие на наши

- автоматизация пайплайнов переобучения моделей
- попытки затащить DL модели (в статье наоборот пытаются перейти к GBDT)
- изучение того как влияет latency на метрики
- несколько примеров feature engineering'a
- эксперимент с увеличением числа кандидатов для ранжирования
- фича мониторинг

Managing Diversity in Airbnb Search

Reference: [1]

С продуктовой точки зрения, показ пользователю разнообразных (diverse) рекомендаций ведет к улучшению пользовательского опыта.
Однако, современные модели ранжирования обучаются, оптимизируя relevance, что приводит к однообразным результатам в рекомендациях.

В статье авторы численно проверяют предположение о том, что рекомендации довольно часто бывают однообразными, и предлагают метрику Mean Listing Relevance (MLR), которая позволяет оценить как релевантность объектов в выдаче, так и их разнообразие.
В основе предложенной метрики — вычисление расстояний между объектами.

В работе предложено большое количество способов как можно вычислить расстояние между двумя объектами и то как оптимизировать MLR.

В заключении, авторы приводят результаты оффлайн и онлайн экспериментов где показывают рост как привычных для задачи ранжирования метрик, так и метрик связанных с разнообразием.

Disentangled Self-Supervision in Sequential Recommenders

Alibaba

Reference: http://pengcui.thumedialab.com/papers/DisentangledSequentialRecommendation.pdf
Keywords: sequence model, contrastive learning

Новое sota решение в задаче sequential recommendations. Бьет BERT4Rec (на некоторых датасетах очень сильно).
Судя по оптимизируемому лоссу, обучаться модель должна быстрее чем BERT4Rec.
Жаль, что авторы не выложили исходный код.

Литература

[1] Mustafa Abdool, Malay Haldar, Prashant Ramanathan, Tyler Sax, Lanbo Zhang, Aamir Mansawala, Shulin Yang, and Thomas Legrand. Managing diversity in airbnb search. *arXiv preprint arXiv:2004.02621*, 2020. URL https://arxiv.org/pdf/2004.02621.pdf.

[2] Weize Kong, Michael Bendersky, Marc Najork, Brandon Vargo, and Mike Colagrosso. Learning to cluster documents into workspaces using large scale activity logs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2416–2424, 2020. URL https://dl.acm.org/doi/abs/10.1145/3394486.3403291.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Часть III

Deep Learning

Retrospective Loss: Looking Back to Improve Training of Deep Neural Networks

Adobe

Reference: [2]
Keywords: neural networks, loss functions

Какую задачу решают авторы?

В статье демонстрируется простой способ улучшить качество практически любой нейросетевой архитектуры.

Улучшение достигается путем добавления к функции потерь дополнительного слагаемого (retrospective loss).
При оптимизации Retrospective loss вместе с основной функцией потерь, учитываются предыдущие состояния параметров модели таким образом, чтобы как можно быстрее двигаться в направлении к оптимальным параметрам.

В статье изложена мотивация стоящая за retrospective loss, теоретическое обоснование того почему ее использование дает преимущества, и большое количество экспериментов на разных задачах, демонстрирующих улучшение качества моделей.

Как решают?

Рассмотрим сначала как выглядит retrospective loss

> **Definition 2** (Retrospective Loss): *Given network* $g\colon \mathbb{R}^n \to \mathbb{R}^d$ *parametrized by its weights* θ *an input data-label pair* $(\mathbf{x}_i, y_i)$*, the retrospective loss at time step* T *is given by:*
>
> $$\mathcal{L}_{retrospective} = (\kappa + 1)\|g_{\theta^T}(\mathbf{x}_i) - y_i\| - \kappa\|g_{\theta^T}(\mathbf{x}_i) - g_{\theta^{T_p}}(\mathbf{x}_i)\|$$
>
> *where* κ *- scaling co-efficients,* θ^T *- parameters at time step* T *and* T_p *- previous time step.*

Интуиция Минимизируя $\mathcal{L}_{retrospective}$ мы пытаемся сделать так, чтобы на очередном шаге предсказание модели для $\mathbf{x}_i$ было ближе к y_i, чем к предыдущему предсказанию.

Рисунок 11.1 визулизирует данную интуицию.

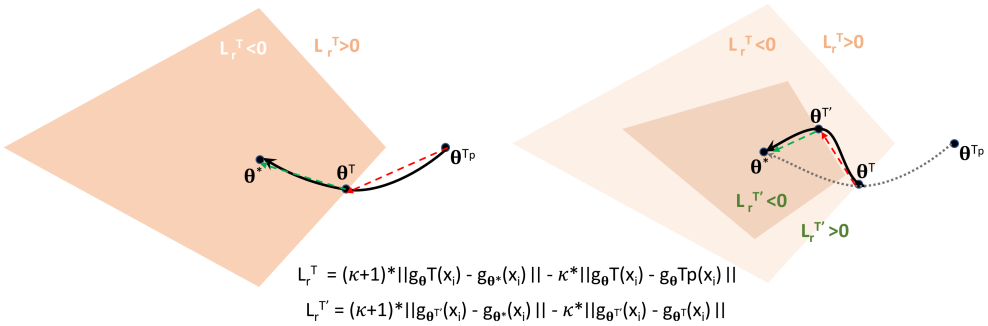

Рис. 11.1: Geometric intuition of the working of the proposed retrospective loss.

По определению, $\mathcal{L}_{retrospective} < 0$ в ситуации, когда предсказание в текущем состоянии, g_{θ^T}, ближе к $y_i := g_{\theta^*}$, чем к предыдущему предсказанию $g_{\theta^{T_p}}$.
В результате, пространство параметров делиться на две части - первая где $\mathcal{L}_{retrospective} < 0$, и вторая где $\mathcal{L}_{retrospective} \geq 0$.

Minimizing retrospective loss pushes the network towards parameters further inside the polytope, thus helping speed up the training process.

Новый loss - просто сумма $\mathcal{L}_{task}$ и $\mathcal{L}_{retrospective}$.

Анализ Главный вопрос: почему добавление $\mathcal{L}_{retrospective}$ позволяет ускорить обучение?

Для ответа на этот вопрос авторы вводят следующее определение

> **Definition 3** (Consistency of Loss Terms for Neural Network Models): *Пусть g_θ нейросеть с параметрами θ и $\theta^* := \arg\min_\theta \mathcal{L}_{task}$ - решение задачи.*
> *Тогда если при добавлении к $\mathcal{L}_{task}$ функции $\mathcal{L}_{add}$, с целью ускорить обучение, оптимум достигается в той же точке, то есть*
> $$\theta^* = \arg\min_\theta(\mathcal{L}_{task} + \mathcal{L}_{add}),$$
> *то функции $\mathcal{L}_{add}$ и $\mathcal{L}_{task}$ называются консистентными.*

Авторы показывают, что при добавление $\mathcal{L}_{retrospective}$ к основному лоссу, минимум функции потерь достигается в тойже точке что и раньше, то есть $\mathcal{L}_{retrospective}$ консистентна с любой $\mathcal{L}_{task}$.

На практике это означает, что при фиксированном числе итераций, использование retrospective loss позволяет ближе подойти к оптимальному решению, в результате мы наблюдаем улучшение метрик.

Преимущества подхода

- Метод очень прост в реализации и практически всегда дает улучшение

Correlation Networks for Extreme Multi-label Text Classification

Reference: [3]
Keywords: multi-label text classification, deep learning, label correlation
Code: https://github.com/XunGuangxu/CorNet

Какую задачу решают авторы?

Задача extreme multi-label text classification (XMTC) состоит в том, чтобы сопоставить тексту набор наиболее релевантных лэйблов.
Пример появления данной задачи на практике: по описанию товара в интернет магазине, найти наиболее подходящие категории, к которым относится товар.
Как и во многих других NLP задачах, Deep Learning подходы дают state-of-the-art результаты.

Как правило, XMTC модели состоят из двух компонент: первая - извлекает информацию из текста и строит некоторое представление, вторая - использует полученное представление для предсказания лэйблов.
Существует большое количество разных подходов к построению первой компоненты, но вторая компонента, чаще всего, представляет из себя полносвязный слой (FC).

Использование простого FC слоя не позволяет учитывать полезные корреляции между лэйблами.
Например, если мы уверены в том, что товару соответствуют лэйблы 'Blues' и 'R&B', то скорее всего также товару должен соотвествовать лэйбл 'Music'.
Знание о корреляции между лэйблами позволяет лучше решать XMTC задачу.

В рамках статьи авторы предлагают Correlation Networks (CorNet) архитектуру, которая выучивает корреляции между лэйблами.
CorNet блок можно добавить в любую XMTC модель поверх label-prediction слоя, для улучшения качества основной модели, без существенного роста размера модели и времени обучения.

Как решают?

Интуиция Пусть у нас есть предсказания лэйблов основной модели - $\mathbf{x}$. Самый простой способ выучить корреляции между лэйблами - добавить еще один FC слой, то есть $\mathbf{y} = W\mathbf{x}$.
Тогда каждый из выходов y_i нового FC слоя представляет из себя линейную комбинацию лэйблов $\{x_1, x_2, \ldots, x_i, \ldots\}$.

Но использование FC слоя очень непрактично с точки зрения потребления памяти, так как матрица W должна иметь размеры $V \times V$, где V - число лэйблов. Хранение такой матрицы чаще всего невозможно.
Кроме того, матрица W должна будет состоять по большей части из нулей, так как большинство лэйблов не скоррелированы.

Исходя из этих соображений, авторы предлагают следующий способ выучить корреляции лэйблов:

$$F(\mathbf{x}) = W_2\delta(W_1\sigma(\mathbf{x}) + \mathbf{b}_1) + \mathbf{b}_2,$$

где σ - sigmoid, δ - ELU function и $W_2 \in \mathbb{R}^{V\times R}, W_1 \in \mathbb{R}^{R\times V}, R \ll V$.

CorNet Итоговая модель CorNet имеет вид $\mathbf{y} = F(\mathbf{x}) + \mathbf{x}$.
Так как матрицы W_2, W_1 имеют небольшой размер, то использование CorNet блока не сильно влияет на размер модели.

Использование CorNet в общем случае случае показано на рисунке 12.1.
То есть можно добавлять несколько CorNet слоев подряд для того чтобы выучивать более сложные корреляции между признаками.

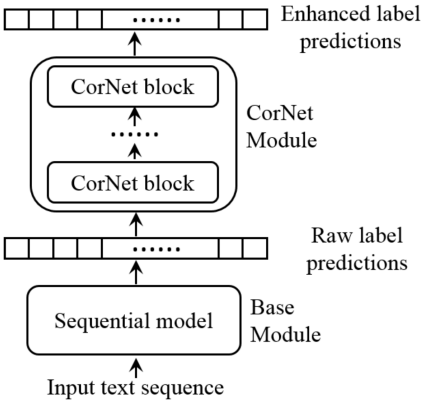

Рис. 12.1: Framework of a CorNet model

Experiments Авторы проводят большое количество экспериментов с разными XMTC моделями на разных датастех, где сравнинвают качество XMTC модели с и без использования CorNet блока.
Из результов экспериментов авторы делают выводы, что использование CorNet блока: практически всегда позволяет значимо улучшить качество XMTC модели, почти не влияет на размер и время обучения.

Выводы

Key Takeaways:

1. Добавление CorNet блока поверх предсказаний основной модели практически бесплатно позволяет улучшить качество модели.
2. Добавление дополнительных CoreNet блоков позволяет дополнительно улучшить качество.

Другие работы

Facebook: Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems

Refernce: https://arxiv.org/pdf/1909.02107.pdf
Keywords: recommendation systems, embeddings, model compression

Современные deep learning модели, которые используются в рекомендательных системах, при обучении используют большое количество категориальных признаков большой размерности.
Как правило, для каждого категориального признака обучается матрица эмбеддингов, которая хранит вектора для каждого возможного значения признака.

Если размерность признака большая, то размер матрицы эмбеддингов может легко доходить до нескольких Гб. Это создает определенные проблемы как при обучении так и при использовании сети.

Один из самых простых способов уменьшить размер модели — hashing trick, значения категориального признака хэшируются в меньшее пространство. Hashing trick очень просто в использовании, но коллизии при хэшировании приводят к потере качества.

В статье представлен способ уменьшения размера модели за счет хранения не одной большой матрицы эмбеддингов для каждого признака, а нескольких небольших матриц.
В рамках экспериментов, авторы показывают, что описанный способ позволяет уменьшить размер модели до 4х раз при этом почти не потеряв в качестве.

Improving Deep Learning For Airbnb Search

Reference: [1]

Продолжение статьи Applying Deep Learning To Airbnb Search https://arxiv.org/pdf/1810.09591.pdf

Довольно интересная статья про переход от GBDT к DL моделям в поиске Airbnb.

В отличии от большинства статей, в данной работе авторы большое внимание уделяют тому какие шишки они набили в процессе перехода к DL и про опыт в целом, а не просто тому какую модель они задеплоили. Не мало места они уделяют обзору того, что хорошо работает в статьях, но у них не заработало.
Особенно мотивирующим получился раздел RETROSPECTIVE.

Литература

[1] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tyler Sax, Lanbo Zhang, Aamir Mansawala, Shulin Yang, Bradley Turnbull, and Junshuo Liao. Improving deep learning for airbnb search. *arXiv preprint arXiv:2002.05515*, 2020. URL https://arxiv.org/pdf/2002.05515.pdf.

[2] Surgan Jandial, Ayush Chopra, Mausoom Sarkar, Piyush Gupta, Balaji Krishnamurthy, and Vineeth Balasubramanian. Retrospective loss: Looking back to improve training of deep neural networks. *arXiv preprint arXiv:2006.13593*, 2020. URL https://arxiv.org/pdf/2006.13593.pdf.

[3] Guangxu Xun, Kishlay Jha, Jianhui Sun, and Aidong Zhang. Correlation networks for extreme multi-label text classification. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1074–1082, 2020. URL https://dl.acm.org/doi/pdf/10.1145/3394486.3403151.

Часть IV

Big Data Infrastructure

To Tune or Not to Tune?
In Search of Optimal Configurations for Data Analytics

Reference: https://dl.acm.org/doi/pdf/10.1145/3394486.3403299
Code: https://github.com/ayat-khairy/tuneful-code
Keywords: spark, configuration tuning, bayesian optimization

Какую задачу решают авторы?

При разработке Spark приложений, разработчики не так часто занимаются вопросом подбора оптимальных параметров (конфигарации) для запуска приложения.
Это может приводить к различным последствиям, например,

- аллокация под приложение слишком большого числа ресурсов, в результате другим приложениям может нехватать ресурсов
- аллокация слишком маленького для задачи числа ресурсов, как следствие приложение работает слишком медленно или вообще не работает из-за нехватки ресурсов

Существуют инструменты для тюнинга конфигураций приложений, которые позволяют избежать подобных ситуаций.
(Тут должен быть блок про предыдущие решения и их недостатки, в работе упоминаются два инструмента: Opentuner[1] и Gunther[2])
Но им часто требуется большое количество (порядка 100) exploration запусков и ими не очень удобно пользоваться.

В статье авторы предлагают фрэймворк Tuneful, который интегрирован в экосистему Spark'а и позволяет относительно быстро тюнить конфигурации приложений.

Как решают

На рисунке 14.1 показано как Tuneful интегрирован в Spark.
Tuneful состоит из трех основных компонент: Significance Analyzer, Cost Modeler и Similarity Analyzer.

[1] http://opentuner.org
[2] https://link.springer.com/chapter/10.1007/978-3-642-40047-6_42

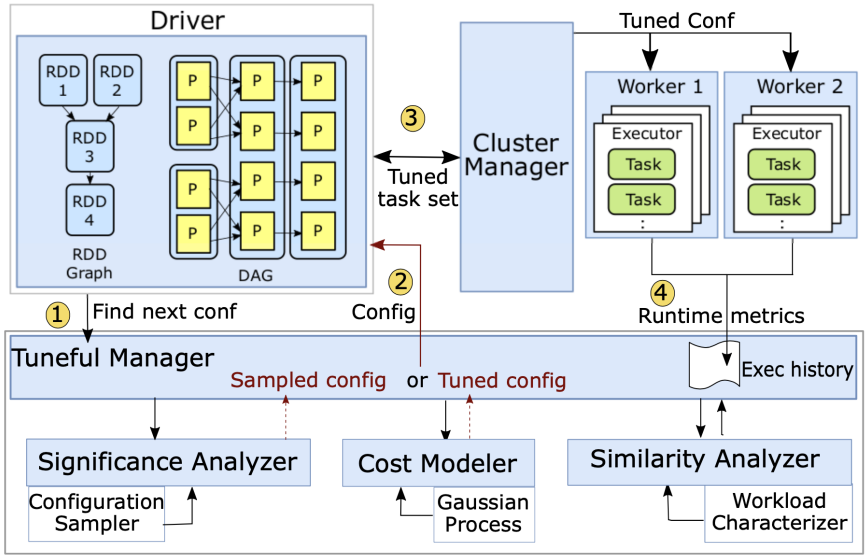

Рис. 14.1: Tuneful–Spark integration: (1) При запуске приложения, Driver запрашивает конфигурацию в Tuneful. (4) После завершения приложения, метрики (время работы, потребление ресурсов итд) отправляются в Tuneful для обновления модели оптимизации.

Кратко работа Tuneful выглядит следующим образом:

Significance Analyzer Также как и в большинстве фрэймворков для оптимизации гиперпараметров ML алгоритмов, процесс оптимизации в Tuneful можно разбить на фазу поиска (exploration) и фазу использования лучшей из найденных конфигураций (exploitation).
В начале фазы поиска Significance Analyzer пытается выделить application-specific важные параметры — запускает приложение с разными конфигурациями, чтобы понять, какие параметры сильнее всего влияют на время работы приложения.

Cost Modeler После того как было выполнено достаточное количество exploration запусков (на практике требуется около 35), Cost Modeler использует информацию об exploration запусках и строит модель, которая позволяет по конфигурации важных параметров предсказывать время работы приложения (неважные параметры при этом остаются фиксированными).
Для построения модели Cost Modeler использует Гауссовские процессы.
Построенная модель используется для получения конфигурации для очередного запуска приложения.
После каждого очередного запуска приложения, Cost Modeler получает информацию о времени работы приложения и обновляет модель.

Когда Tuneful находит достаточно хорошую конфигурацию для приложения, процесс тюнинга останавливается и для последующих запусков используется найденная конфигурация.

Similarity Analyzer Когда Tuneful используется уже продолжительное время (тюнил конфигурации для большого числа приложений), то при тюнинге параметров для нового приложения, Tuneful может использовать информацию о *похожих* приложениях для более удачного старта процесса тюнинга.
Поиском похожих приложений занимается Similarity Analyzer.

Для того чтобы посчитать похожесть между приложениями, авторы описывают их с помощью большого количества метрик, которые собирают во время работы приложения.
На собранных метриках обучают autoencoder, для того чтобы для приложения получить описание в виде вектора меньшей размерности.

Experiments

Авторы проводят большое количество экспериментов для того чтобы оценить качество предложенного фрэймворка.

Для экспериментов используют два кластера: первый из 20 машин и второй из 4х.
В качестве приложений для оптимизации авторы выбрали 5 довольно сильно отличающихся друг от друга приложений.
В качестве бэйслайнов авторы используют Random Search, Opentuner и Gunther.

Авторы показывают, что Tuneful позволяет примерно в 3 раза быстрее найти хорошие конфигурации, и что использование Similarity Analyzer'а позволяет в 3 раза ускорить тюнинг для новых приложений.

Выводы

Key Takeaways:

1. Если приложение запускается суммарно меньше 100 раз, то нет смысла пытаться его тюнить используя специальные фрэймворки из-за большого числа exploration запусков.
2. Если речь идет о приложении, которое, например, запускается по крону каждый день, то оно идеально подходит для тюнинга конфигурации, но нужно быть готовым к тому, что в течении первых 30 запусков почти не будет выйгрыша в суммарном времени работы.

Мое мнение

- В целом, не очень понятно насколько это значимая работа, так как это первая статья, которую я смотрел в области тюнига конфигураций приложений в рамках кластера.

  Думаю, что полезно начать смотреть в этом направлении.
- Немного странным выглядит выгялдит выбор бэйслайнов: почему из числа методов для тюнинга гиперпараметров используется только Random Search, а не какой нибудь более современный фрэймворк?

Lumos: A Library for Diagnosing Metric Regressions in Web-Scale Applications

Microsoft

Reference: https://arxiv.org/abs/2006.12793
Keywords: online metrics, A/B experiments, anomaly detection
Code: https://github.com/microsoft/MS-Lumos

Какую задачу решают авторы?

Неотъемлемой частью любого web-scale приложения является система мониторинга, которая отслеживает большое количество метрик в течении времени, и сообщает об аномалиях.
Быстрое обнаружение аномалий и возможных причин их возникновения очень важно для стабильной работы сервиса и хорошего пользовательского опыта.

Существующие state-of-the-art алгоритмы детекции аномалий имеют довольно высокий false-positive rate и сами по себе не могут дать ответ на вопрос, что могло послужить причиной возникновения аномалии.
Уменьшение доли ложных срабатываний и помощь в поиске истинной причины появления аномалии могут существенно сэкономить время разработчиков.

В данной статье авторы представляют Python фрэймворк Lumos, который решает обозначенные проблемы.