January 2017.

XXX

Egor Khairullin

mikari.san@gmail.com

Abstract: The result of many machine learning algorithms are complex and slowly applied models. And further growth in the quality of the such models usually leads to a deterioration in the application times. However, such high quality models are desirable to be used in the conditions of limited resources (memory or cpu time).

1. введение.

Сегодня машинное обучение (МО) используется в различных сферах: от поиска информации в интернете и распознавания речи до оптимизации состава стали и поиска новой физики. Практически во всех случаях для решения поставленной задачи очень ограниченны ресурсы (память, процессор, сеть и т.д.). Для многих алгоритмов МО характерно, что повышение качества модели связанно с повышением уровня потребления ресурсов. Например, при использовании алгоритмов бустинга над деревьями, повышение числа деревьев с десятков до тысяч обычно ведет к значительному росту качества, одновременно увеличивая нагрузку на память и процессор при использовании такой модели. Повышенные требования к аппаратным ресурсам более точных моделей обычно решают за счет различных технических оптимизаций хранения и применения полученной модели: например, с помощью векторных вычислений. Другим подходом является модификация самого процесса обучения, позволяющее получать модели с определенными свойствами, которые позволяют значительно ускорять их применение, не сохраняя необходимый уровень качества. В данной работе мы рассмотрим модификацию процесса обучения предсказательных моделей алгоритмом CatBoost (Gulin, Kuralenok, and Pavlov 2011), которая позволит ускорить использование модели с сохранением необходимого уровня качества.

2. catboost

CatBoost - является реализацией градиентного бустинга над деревьями, в котором используются так называемые “невнимательные” деревья решений (Oblivious Decision Tree). В данном алгоритме значения каждого признака i разбиваются на корзинки с помощью границ B_i. Сами границы определяются заранее с помощью различных статистик. А исходный вектор с вещественными значениями заменяется на вектор с бинарными значениями 0 и 1 ($false$ и $true$): значение i-ого признака f_i заменяется на бинарный вектор g_i, где $g_{ij} = f_i > B_{ij}$, после чего все g_i конкатенируются в один большой вектор. И в получаемых деревьях вместо сравнения некоторого вещественного признака с некоторым вещественным порогом фактически будет находится сравнение с 0 и 1. В матрикснете можно управлять количеством корзинок, передавая соответствующий параметр при обучении. Другой достаточно важный параметр - скорость обучения. Чем она меньше - тем с меньшим вкладом берется каждое новое дерево. Однако, это приходится компенсировать большим количеством деревьев. Данный параметр можно подбирать с помощью кроссвалидации, фиксируя при этом общее количество деревьев.

3. решающий тензор

Одним из самых простых способов ускорения является предварительный расчет. Вектор признаков, получаемый после бинаризации, принимает ограниченное число значений. Поэтому мы можем предпосчитать ответ для всех таких значений.

Пусть n - количество признаков. Рассмотрим некоторую обученную с помощью некоторого алгоритма модель Z, содержащую множество деревьев T. Для удобства будем считать, что значения всех признаков лежат в отрезке $[0, 1]$.

Пусть f_i - набор границ корзинок для i’ого признака, которые использовались внутри набора T. Тогда с помощью плоскостей $X_i = b_{ij}$ для всех i, j мы можем разбить пространство $[0; 1]$ на небольшие гиперпараллелепипеды. Очевидно, что внутри каждого гиперпараллелепипеда предсказание модели будет постоянным. По сути, модель Z позволяет посчитать значение в любой точке нашего гиперкуба. Однако, если мы предпосчитаем значение в каждом кусочке, то мы ускорим применение нашей модели, сведя её лишь к определению нужного кусочка. Асимптотическая сложность применения модели из деревьев - $O(h \cdot t)$, где h - высота деревьев, t - общее количество деревьев. Асимптотическая сложность предпосчитанная сложность гиперкуба же - $O(n \cdot log(b))$, где n - число признаков, а b

- количество разрезов одного признака (такая сложность достигается с помощью использования бинарного поиска). При малом количестве признаков такой гиперкуб может применяться значительно быстрее.

Однако, для хранения такого решающего тензора нужно порядка $O(b^n)$ памяти, что может быть слишком большим. Поэтому использовать предпосчитанный тензор можно лишь при небольших b и n.

В общем случае, размер одного тензора можно определить так:

$$D = (f_1, ..., f_n)$$

$$f_i = \{b_{i1}, ..., b_{ip_i}\}$$

$$S(D) \propto \prod |f_i| = \prod p_i$$

Рассмотрим случай при малых n. Получить небольшое b мы можем двумя основными способами: ограничить число разрезов для каждого признака - тогда мы сможем обучить CatBoost с $t = 5000$, либо сильно ограничить число деревьев - тогда общее количество действительно используемых разрезов будет ровно $h \cdot t$ без учета повторных использований одних и тех же разрезов. В работе (Likhomanenko et al. 2015) использовался первый способ: был обучен и превращен в тензор CatBoost с небольшим количеством разрезов. Потребление памяти у одного гиперкуба может быть чрезвычайно огромным при условии достижения приемлимого качества, поэтому естественным продолжением является построение несколько решающих тензоров.

4. метрика схожести тензоров.

Введем метрику схожести двух тензоров. Определим её как разницу между суммой их размеров и размером их объединения:

$$D^1 = (f_1^1, ..., f_n^1)$$

$$D^2 = (f_1^2, ..., f_n^2)$$

$$D^u = D^1 + D^2$$

$$D^u = (f_1^u, ..., f_f^u), f_i^u = f_i^1 \cup f_i^2$$

$$Sim(D^1, D^2) = S(D^1) + S(D^2) - S(D^1 + D^2)$$

5. задача многокритериальной оптимизации.

Целевыми метриками является Q (качество модели), S (размер) и t (время применения). Однако, можно считать, что время применения пропорционально количеству решающих тензоров M. Входной же вектор должен будет описывать алгоритм получения деревьев и алгоритм объединения их в кубы. Обозначим его V. Тогда мы решаем следующую задачу:

$\min\{Q(V), S(V), M(V)\}$ по всем возможным V.

Прежде чем мы приступим к рассмотрению содержимого вектора X, необходимо отметить, что для практического применения данную задачу можно решать с помощью метода ограничений: обычно на практике есть довольно строгие ограничения на размер модели и доступное время применения.

6. алгоритм объединения конкретного набора деревьев в набор решающих тензоров.

Допустим, у нас есть N деревьев и мы хотим объединить их в M решающих тензоров. Метрикой оптимальности разбиения будем считать занимаемую память. В общем случае, это непростая задача и она требует отдельного изучения. Мы же использовали следующий эвристический подход: 1) Инициализируем M тензоров, поместив в каждый одно произвольно выбранное дерево 2) Выберем любое из оставшихся деревьев 3) Объединим с наиболее схожим тензором 4) Повторяем пункты 2-3 пока не останется неиспользованных деревьев.

7. полный алгоритм построения ансамбля решающих тензоров с помощью catboost.

Тут есть два ключевых подхода:

1) Фиксируем очень грубое разбиение с малым количеством кубов и обучаем сколь угодно большое количество деревьев.

2) Разбиение делаем достаточно детальное, но при этом значительно уменьшаем количество обученных деревьев, тем самым снижая количество реально используемых бинов.

Алгоритм активно использует обе эти идеи.

1) Определяем желаемое число тензоров (M) исходя из временных ограничений

2) Обучаем модель CatBoost с ограниченным количеством разрезов.

3) Объединяем деревья из полученной модели в первый решающий тензор из ансамбля

4) С помощью случайного поиска перебираем ключевые параметры CatBoost

   - Обучаем модель CatBoost, используя модель из 2) как бейзлайн
   - Преобразуем в M-1 решающий тензор
   - Убираем ансамбли, не вписывающиеся в требуемые ограничения по размеру.

5) Выбираем ансамбль с наилучшим качеством на дополнительном тестовом наборе данных.

Такой алгоритм по своему построению позволяет найти оптимальный по Парето ансамбль решающих тензоров.

8. применение алгоритма на датасете

За основу был взят датасет и модель из (Likhomanenko et al. 2015). Количество признаков в датасете ~10, модель была обучена на 5000 деревьях с признаком разбиений 64. Мы сравнивали наш алгоритм построения Ансамбля Решающих Тензоров с тривиальным подходом: обучение малого количества деревьев с высокой скоростью обучения. Для этого мы разделили датасет на три части: обучающая (50%), тестовая (25%) и валидационная (25%). Мы обучали модели на обучающей выборке, затем с помощью тестовой отбирали наиболее качественные модели при заданных ограничениях.

Мы ограничили число тензоров 6 а суммарный размер моделей одним гигабайтом, считая, что для хранения каждого предпосчитанного предсказания необходимо 4 байта (float). Мы перебирали только ключевые параметры CatBoost про целевое число деревьев и скорость обучения.

В качестве метрики была взята площадь под ROC кривой при False Positive Rate до 0.05. Для удобства, мы отнормировали эту метрику, чтобы максимальное её значение (при идеальной модели) было равно 1.

9. обсуждение.

Мы сравнивали модель CatBoost на 5000 деревьев (“best”), быстрые модели CatBoost на 1-125 деревьев (“fast”) и Ансамбли Решающих Тензоров при разных ограничениях на память и разном количестве тензоров.

На графике 1 представлена зависимость качества от числа лукапов. На графике 2 представлена зависимость качества от общего времени применения. На таблице 3 представлены некоторые из обученных в процессе моделей.

TODO: добавить графики из презентации

Видно, что при сравнимом качестве с “fast”, Ансамбли Решающих Тензоров позволяют в 25 раз снизить количество лукапов. А в сравнении с “best” моделью количество лукапов меньше в 1000 раз при небольшой потере в качестве (< 0.006 при значении “best” в 0.6).

Однако, разница во времени работы уже менее значительна - в 2,5 раза и 50 раз соответственно. Это связано с принципами работы кэша и особенностями нашего бенчмарка. Так как Решающий Тензор отребует очень большое количество памяти и не из-за этого вмещается в кэш, почти каждый лукап приводит к чтению из оперативной памяти. Модели CatBoost’а же достаточно малы, из-за чего могут целиком помещаться в кэше. Так как наш бенчмарк состоит только из применений моделей, модели CatBoost’а не вытесняются из кэша. В реальных же приложениях модель CatBoost’а скорее всего не будет единолично владеть кэшом и будет из-за этого ещё медленнее. Решающие тензоры же и так почти не зависят от кэша.

10. заключение.

Ансамбли Решающих Тензоров позволяют упростить внедрение сложных Boosting Decision Trees алгоритмов (таких как CatBoost) в real-time системах без возможности из использовать batching применение. Алгоритм построения Ансамблей позволяет разменивать потребляемую память на скорость применения с небольшой потерей в качестве. Кроме того, использование Ансамбля позволяет значительно повысить качество в сравнении с использованием одиночного Решающего Тензора.

ссылки

Gulin, Andrey, Igor Kuralenok, and Dmitry Pavlov. 2011. “Winning the Transfer Learning Track of Yahoo!’s Learning to Rank Challenge with Yetirank.”

In Yahoo! Learning to Rank Challenge, 63–76.

Likhomanenko, Tatiana, Philip Ilten, Egor Khairullin, Alex Rogozhnikov, Andrey Ustyuzhanin, and Michael Williams. 2015. "LHCb Topological Trigger Reoptimization." In Journal of Physics: Conference Series, IOP Publishing, 082025.