

January 2017.

XXX

Egor Khairullin

mikari.san@gmail.com

Abstract: Результатами многих алгоритмов машинного обучения является довольно сложная и медленно применяющаяся модель. Однако такие модели хочется применять в условиях ограниченных ресурсов и пропускать через них как можно больше событий.

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 ограничены ресурсы (память, процессор, сеть и т.д.). Для многих алгоритмов МО характерно, что повышение качества модели связано 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 Matrixnet (Gulin, Kuralenok, and Pavlov 2011).

2. matrixnet

Матрикснет - алгоритм МО, придуманный в Яндексе в 20XX году. Является реализацией градиентного бустинга деревьями, однако используются так называемые “невнимательные” деревья решений (Oblivious Decision Tree). Особенностью является регулируемое число разбиение для каждого фактора, определяемое на основании различных статистик до непосредственного обучения. Другой достаточно важный параметр - скорость обучения. Этот параметр позволяет выбирать между высоким качеством (большое число деревьев) и большой скоростью применения (маленькое число деревьев). За основу был взят датасет и модель из (статья про матрикснет для черна). Количество факторов в датасете ~ 10 , бейзлайн-модель была обучена на 5000 деревьях с фактором разбиений 64. Очевидное решение исследуемой проблемы: сильно зависить скорость обучения и получить модель с очень небольшим количеством деревьев (меньше 10) хотя и позволяет достичь приемлимой скорости применения, приводит к сильной потере качества.

3. гиперкуб

Одним из самых простых способов ускорения является предпосчет. Пользуясь конечным множеством значений у каждого дерева (для Oblivious Decision Tree это не более, чем 2^h различных значений, где h - высота дерева), можно предпосчитать все значения для целого набора деревьев, тем самым ускорив вычисления в момент применения.

Пусть n - количество факторов, T - деревья, полученные в результате обучения некоторого алгоритма. Для удобства будем считать, что значения всех факторов лежат в $[0; 1]$. Пусть B_i - набор разрезов для i -ого фактора, которые использовались внутри набора T . Тогда с помощью плоскостей $X_i = B_{ij}$ для всех i, j мы можем разбить гиперкуб $[0; 1]$ на небольшие гиперпараллелепипеды. Очевидно, что внутри каждого кусочка предсказание модели будет постоянным. По сути, наша модель 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 факторов, а b - количество разрезов одного фактора (такая сложность достигается с помощью использования бинарного поиска). При малом количестве факторов такой гиперкуб может применяться значительно быстрее. Однако, для

хранения такого гиперкуба нужно порядка $O(b^n)$ памяти, что может быть слишком большим. Поэтому использовать предпочитанный гиперкуб можно лишь при небольших b и n . Рассмотрим случае при малых n . Получить небольшое b мы можем двумя основными способами: ограничить число разрезов для каждого фактора - тогда мы сможем обучить матрикснет с $t = 5000$, либо сильно ограничить число деревьев - тогда общее количество действительно используемых разрезов будет равно $h \cdot t$ без учета повторных использований одних и тех же разрезов. В работе (ссылка на статью про матрикснет для церна) использовался первый способ: был обучен и превращен в “гиперкуб” матрикснет с небольшим количеством разрезов. Потребление памяти у одного гиперкуба может быть чрезвычайно огромным при условии достижения приемлимого качества, поэтому естественным продолжением является построение несколько гиперкубов. Тут возникает две связанные задачи: 1) Как эффективно разбить N деревьев в M гиперкубов, минимизируя суммарный размер гиперкубов. 2) Как обучать деревья, позволяющие эффективно разбивать их в N деревьев.

3. объединение деревьев в несколько гиперкубов.

Допустим, у нас есть N деревьев и мы хотим объединить их в M гиперкубов. При $M = 1$ задача тривиальна и рассматривалась выше. Рассмотрим $M > 1$. Метрикой оптимальности разбиения будем считать занимаемую память.

Для решения данной задачи мы использовали несколько эвристических методов:

- 1) Вначале у нас N гиперкубов из 1 дерева. На каждой итерации находим и сливаем два наиболее схожих гиперкуба (схожих - то есть итоговая метрика вырастет слабее всего)
- 2) Вначале у нас M гиперкубов и в каждом одно произвольно выбранное дерево. Выбираем любое из оставшихся деревьев и добавляем в наиболее схожий с ним гиперкуб
- 3) Аналогично 2-ому, но выбираем самое лучшее дерево. Эти методы, очевидно, не дают гарантированно лучшего варианта, но позволяют получить вариант, который лучше случайного разбиения. Мы в итоге брали самый лучший из вариант из всех трех.

4. обучение подходящих деревьев.

Для получения гиперкубов можно использовать деревья, полученные простым запуском матрикснета. Важными параметрами тут будут:

- 1) Фактор разбиения фичей (X)
- 2) Число деревьев (T)
- 3) Скорость обучения (W)
- 4) Итоговое качество (Q)
- 5) Число гиперкубов (M)
- 6) Итоговый размер гиперкубов (S)

Оптимизационная задача была поставлена таким образом: максимизируем Q , при $S \leq S_T$, $M \leq M_T$, где M_T и S_T определяются исходя из требуемой скорости применения модели и максимально доступной памяти.

Исследовалось несколько следующих частных способов: 1) “Rude”: очень небольшой X , большое T , $M = 1$. На выходе получается из-за очень малого X один достаточно компактный гиперкуб. 2) “Small”: большой X , $T < 100$, $M < 10$. Из-за большого X деревья получаются очень разные и значительно хуже склеиваются, из-за чего приходится сильно ограничивать число деревьев, используемых для склейки в гиперкубы. 3) Аналогично 2, но взяв в качестве бейзлайна при обучение пункт 1.

Для поиска оптимальных решений мы использовали поиск по сетке с параметрами T и W . Для каждой пары строился оптимальный набор гиперкубов. Среди всех полученных наборов отбирался лучший по качеству, укладывающийся в потребление памяти.

5. результаты.

На графике 1 представлены результаты. По оси x указано ограничение на суммарный размер модели S , а по оси y - итоговое качество модели Q .

- 1) Baseline - обычный матрикснет с $T = 5000$, $X = 64$, изображен линией для удобства, хотя потребление памяти у обычной модели очень мало
- 2) Rude - при $T = 5000$, $X = 5$
- 3) Small - при $T < 100$, $X = 64$ при различных M .
- 4) Rude + Small По графику видно, что способ 4 дает наибольшее итоговое качество. Таким образом итогом можно порекомендовать в качестве первого гиперкуба брать rude схему, а при недостаточном качестве добавлять Small гиперкубы, полученные из матрикснета, обученного на небольшом количестве деревьев.

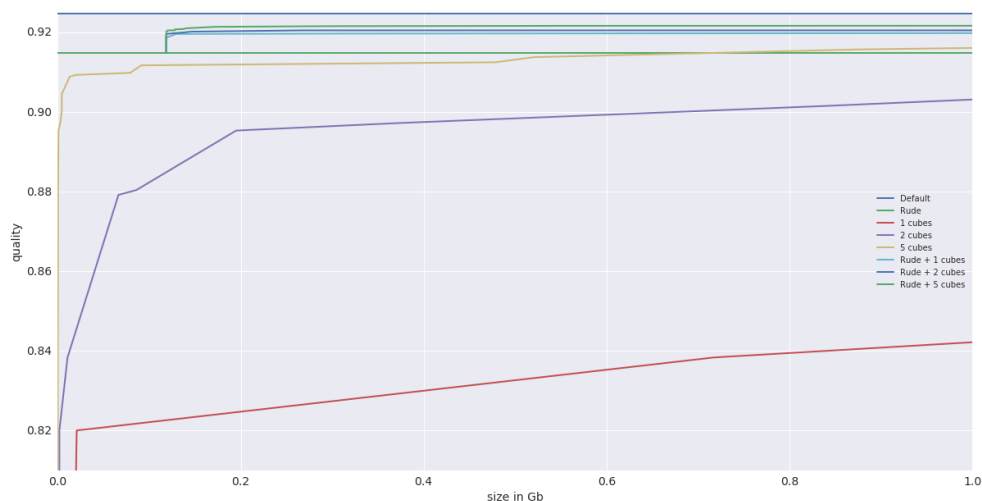


Figure 1: График 1. Качество против размера

Стоит отметить, что при малом X матрикснет строит не самые лучшие разбиения факторов. Поэтому в дальнейших исследованиях необходимо рассмотреть способы нахождения более оптимальных разбиений, чем получается с помощью простых статистических методов.

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.