САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Белоус Михаил Андреевич

Разработка методов машинного обучения

Курсовая работа

Научный руководитель:
к. ф.-м. н. Куралёнок И. Е.

Санкт-Петербург
2014

Оглавление

Введение

Машинное обучение — это подраздел искуственного интеллекта, изучающий алгоритмы, способные обучаться. Задача индуктивного обучения состоит в поиске закономерностей в данных. Данные для обучения представляются в виде множества объясняющих факторов и отклика, который мы ожидаем от программы. Например, имеется текст письма и информация о том, является ли это письмо спамом, задача состоит в том, чтобы программа научилась сама принимать решение, имея только текст письма. Методы машинного обучения применяются во многих областях: в компьютерной лингвистике, распознавании речи, обнаружении спама, ранжировании поисковой выдачи, в финансовом анализе и ядерной физике.

Для оценки качества алгоритма его запускают на данных, которые были от него скрыты в процессе обучения. Таким образом проверяется то, что алгоритм находит закономерности в данных, а не подстраивается под обучающее множество. Явление, когда алгоритм улучшает свою точность на обучающем множестве, но теряет точность на тестовом, называется переобучением. Для сравнения двух алгоритмов их запускают на одинаковом наборе данных и сравнивают по некоторым метрикам, какой алгоритм дает более точный результат на тестовых данных.

Задача, которую я пытаюсь решить — это задача ранжирования, то есть сортировки документов от наиболее подходящих к наименее подходящим. Оценка соотвествия документа к запросу называется релевантностью. Этот алгоритм используется в поисковых системах для определения порядка выдачи. На текущий момент одним из лучших алгоритмов ранжирования является MatrixNet[3], используемый сейчас компанией Яндекс. С этим алгоритмом я и буду сравнивать разработанные алгоритмы.

1. Постановка задачи

Нам дано некоторое множество документов $Learn$, для которых мы знаем их релевантности. Требуется на основе этих данных построить процедуру автоматического ранжирования документов. В работе рассматривается методы индуктивного обучения, то есть такие методы, которые основаны на выявление закономерностей в эмпирических данных, а не на формализации знаний экспертов.
Задача обучения ранжированию состоит в распределении документов, полученных на конкретный запрос, в порядке, максимально близком к идеальному ранжированию, сделанному на основе экспертных оценок[2].

Представление данных

Документы для ранжирования представляются в виде вектора из $\mathbb{R}^n$. Это пространство также называется пространством факторов. Отображение документа в пространство факторов — задача сложная и неоднозначная, ее подробное рассмотрение выходит за рамки данной работы.

Метрики оценки качества

$Test$ — это скрытые от проверяемой программы данные, X — вектор факторов для документов, $target$ — истинная функция оценки для документа, F — оцениваемая функция.
MSE (mean sqaure error, среднее квадратичное отклонение) — метрика оценки качества регрессии. Вычисляется по формуле

$$MSE(X, Test) = \sum_{x \in Test} \frac{(F(x) - target(x))^2}{|Test|} \tag{1}$$

Для оценки качества ранжирования используется мера $pFound$

$$pFound = \sum_{i=0}^{n} pLook_i * pRel_i \tag{2}$$

В этой формуле $pLook_i$ — оценка вероятности того, что пользователь досмотрел до i-ого документа выдачи, а $pRel_i$ — значение релевантности i-ого документа выдачи.

Данные, на которых оцениваются методы

Маленький набор данных: обучающая выборка состоит из 12465 образцов, данные для проверки состоят из 46596 образцов. Каждый образец состоит из 13 бинар-

ных факторов и 20 непрерывных, а также экспертной оценки. Этот набор используется для сравнения по точности.

Большой набор данных: обучающая выборка состоит из более чем 1500000 образцов, каждый предстваляется более чем 700 факторами. В данной работе эти данные используются для сравнение времени работы алгоритмов.

Графический процессор

Графический процессор (Graphical Processing Unit, GPU) подходит для решения задач, допускающих распараллеливание по данным, одновременно обладая средствами выполнения потоков арифметических операций, характеризуемых частыми обращениями к памяти. Благодаря одновременному выполнению множества арифметических операций для нескольких элементов удается скрыть задержки доступа к памяти.

С аппаратной точки зрения графические процессоры состоят из масштабируемого набора многозадачных потоковых мультипроцессоров. Мультипроцессор не просто позволяет запускать сотни вычислительных потоков одновременно, но и требует этого для достижения наибольшей эффективности, запуская потоки группами по 32 элемента, мгновенно осуществляя переключение между ними, так как контекст выполнения содержится внутри мультипроцессора[2].

2. Подходы к решению

2.1. Деревья решений

Дерево решений — это метод построения разбиения пространства на регионы (подпространства) таким образом, что целевая функция имеет в них одно значение. Разбиение строится по набору условий, те точки, которые удовлетворили одинаковому набору условий, попадают в один регион. В данной работе, я использовал алгоритм построения забывчивых деревьев, то есть деревьев, которые выбирают каждое новое разбиение вне зависимости от предыдующих. На каждой итерации выбирается разбиение по некоторому фактору, которое минимизирует разброс целевой функции внутри получившихся после этоло разбиения регионах. Таким образом находим 6 условий и по ним получаем $2^6 = 64$ региона. Наибольшей вычислительной нагрузкой в этом алгоритме является поиск оптимального разбиения. Для ускорения перебираются не все возможные разбиения по каждому фактору, а разбиения по квантилям распределения факторов, то есть разбиения, оставляющие с одной стороны $\frac{1}{k}, \frac{2}{k} ... \frac{k-1}{k}$-тую часть точек. Этот алгоритм используется в качестве базового для построения разбиение по регионам для всех последующих алгоритмов.

2.2. Градиентный бустинг

Пусть есть некоторый алгоритм построения базовых моделей $h(x, y)$, где x — это объясняющие переменные, а y — значения в точке, которую нам нужно научиться аппроксимировать, и функция потери $\Psi(a, b)$. По исходному набору $Learn$ и функции Ψ нужно найти функцию, минимизирующую функцию потери.

$$F^*(x) = \underset{F(x)}{\text{argmin}} \sum_{x,y \in Learn} \Psi(y, F(x)) \tag{3}$$

Градиентный бустинг решает проблему поиска этой функции следующим алгоритмом: пусть на m-ом шаге у нас есть функция F_{m-1}, для каждом точки из обучающего множества посчитаем производную функции потери для текущей функции аппроксимации F_{m-1}.

$$y_i^m = -\frac{\partial \Psi(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \tag{4}$$

Далее обучаем модель $h(x, y_i^m)$ и добавляем её с коэффициентом α к F_{m-1}, коэффициент α и количество слабых моделей, составляющих итоговую модель N, подбираются для каждой задачи. Наиболее трудоемким местом градиентного бустинга

является обучение слабой модели в нашем случае дерева решений.

$$F_m(x) = F_{m-1}(x) + \alpha \cdot (x, y_i^m)(x) \tag{5}$$

Также рекомендуется на каждой итерации обучать модель не на всём множестве, а на случайном подмножестве. Это позволяет отсрочить переобучение[1].

2.3. Линейная аппроксимация листьев деревьев решений

В листьях дерева решений будем вычислять значение целевой функции линейной аппроксимацией факторов, по которым у нас построено дерево. Пусть x_i — значение i-ого фактора, по которому происходило построение дерева. Для удобства введем фиктивный фактор $x_0 = 1$. Тогда значение модели будет вычисляться по формуле

$$F(x, y^l) = \sum_{i=0}^{depth} x_i \cdot y_i^l, x \in R^l \tag{6}$$

Подбираем коэффициенты y^l, минимизирующие сумму квадратов отклонений в точках обучающего множества, попавших в регион.

$$y^l = \underset{y \in \mathbb{R}^{depth+1}}{\text{argmin}} \sum_{x \in Learn, x \in R^l} (F(x, y) - target(x))^2 \tag{7}$$

Продифференцируем по y_i

$$\sum_{x \in Learn, x \in R^l} \sum_{j=0}^{depth} (x_i \cdot x_j \cdot y_i \cdot y_j - x_i \cdot y_i) = 0 \tag{8}$$

Для каждого региона независимая система из $depth + 1$-го уравнения. Решив эту систему найдем y^l.

2.4. Квадратичная аппроксимация деревьев решений с граничными условиями

В листьях дерева решений будем вычислять значение целевой функции квадратичной аппроксимацией факторов, по которым построено дерево. Пусть x_i — факторы, по которым строилось дерево решений, для удобства введем фиктивный фактор $x_0 = 1$.

$$F(x, y^l) = \sum_{i=0, j=0}^{depth} x_i \cdot x_j \cdot y_{i,j}^l, x \in R^l \tag{9}$$

Продифференцируем функцию невязки по $y^l_{i,j}$.

$$\sum_{x \in Learn, x \in R^l} \sum_{i,j,f,g=0}^{depth} x_i \cdot x_j \cdot x_f \cdot x_g \cdot y_{f,g} \cdot y_{i,j} - x_i \cdot x_j \cdot y_{i,j} \tag{10}$$

Значимой проблемой деревьев решений является то, что полученная функция регрессии имеет разрывы при переходе из одного региона в другой. Для квадратичной аппроксимации введем требование равенства полиномов на границах регионов. Пусть l_1, l_2 - граничащие листья, f - номер разделяющего эти регионы измерения, b - граничное условие.

$$1. y^{l_1}_{0,m} + b \cdot y^{l_1}_{f,m} = y^{l_2}_{0,m} + b \cdot y^{l_2}_{f,m}, \forall m \notin \{0, f\} \tag{11}$$

$$2. y^{l_1}_{n,m} = y^{l_2}_{n,m}, \forall m, n \notin \{0, f\} \tag{12}$$

$$3. y^{l_1}_{0,0} + b^2 \cdot y^{l_1}_{f,f} = y^{l_1}_{0,0} + b^2 \cdot y^{l_1}_{f,f} \tag{13}$$

Пусть W — набор условий равенства, введем квадратичное отклонение от условия непрерывности, которое можно записать как $y \cdot w^T \cdot w \cdot y^T$. Коэффициенты квадратичной оптимизации будем искать в виде

$$y = \underset{y \in \mathbb{R}^N}{\text{argmin}} \sum_{x \in Learn} (F(x, y) - target(x))^2 + \lambda \cdot \sum_{w \in W, |w|=1} y \cdot w^T \cdot w \cdot y^T \tag{14}$$

Продифференцировав сумму невязки и квадратичного отклонения от условия непрерывности мы получим одну систему уравнений с $\frac{(depth+1)\cdot(depth+2)}{2} \cdot 2^{depth}$ переменных, для глубины дерева равной 6 размер матрицы будет равен 1792. В связи с трудоемкостью задачи построения этой модели были реализованы следующие оптимизации.

2.4.1. Подсчёт статистик с использованием GPU

Для поиска коэффициентов квадратичной аппроксимации нам нужно подсчитать следующие статистики:

$$\sum_{x \in Learn, x \in R^l} \sum_{i,j=0}^{depth} x_i \cdot x_j \tag{15}$$

$$\sum_{x \in Learn, x \in R^l} \sum_{i,j,f,g=0}^{depth} x_i \cdot x_j \cdot x_f \cdot x_g \tag{16}$$

Если эти статистики подсчитывать, используя CPU, время работы на данных большого пула, будет составлять 37 секунд. Задача подсчёта сумм удобно разбивается на подзадачи. Для подсчёта каждой статистики выделим блок из k (константа зависит от видеокарты) потоков, где i-ый поток будет обрабатывать каждую k-ую точку начиная с i. При этом каждый блок за одну операцию будет подсчитывать статисти-

ку для k точек. В конце работы значения в блоке просуммируем и получим искомое значение статистики. Таким образом можно достичь времени работы 770 миллисекунд.

2.4.2. LQ - разложение с использованием GPU

Для решения системы линейных уравнений следует использовать LQ разложения Хаусхолдера. В этом разложении основной вычислительной нагрузкой является поворот Хаусхолдера относительно вектора w для каждого вектора матрицы.

$$x = x \cdot (1 - 2 \cdot (x, w)) \tag{17}$$

Система уравнений, которую нужно решать, имеет размер 1792 элемента. Время работы реализации на процессоре составляет больше 2-х минут. Если для подсчёта использовать графический процессор следующим образом: для каждого вектора матрицы создать блок из k потоков, i-ый поток будет считать каждое k-ое произведение $w_{i+k \cdot t} \cdot x_{i+k \cdot t}$, далее потоки синхронизируются, и каждый поток получает константу, на которую нужно домножить $x_{i+k \cdot t}$. Таким образом можно осуществить разложение матрицы за 1200 миллисекунд.

2.5. Квадратичная аппроксимация деревьев решения, экспоненциально убывающая в зависимости от расстояния от региона

Значение функции для точки будем вычислять как взвешенную сумму значений квадратичной регрессии в каждом регионе с весом, экспоненциально убывающим от расстояния между точкой и регионом. Для удобства введем фиктивный фактор $x_0 = 1$.

$$F(x, y) = \sum_{l=0}^{2^{depth}} \sum_{i=0,j=0}^{depth} x_i \cdot \cdot x_j \cdot y_{i,j}^l \cdot e^{-dist(R^l, x)} \tag{18}$$

Из формулы регрессии видно, что функция получается непрерывной. Коэффициенты для регрессии будем искать как взвешенную сумму квадратов отклонений с весами, экспоненциально убывающими в соответствии с расстоянием от точки до региона.

$$y^l = \underset{y \in \mathbb{R}^{\frac{(depth+1)\cdot(depth+2)}{2}}}{\text{argmin}} \sum_{x \in Learn} (F(x, y) - target(x))^2 \cdot e^{-dist(R^l, x)} \tag{19}$$

Продифференцируем по $y^l_{i,j}$.

$$\sum_{x \in Learn} \sum_{i,j,f,g=0}^{depth} (x_i \cdot x_j \cdot x_f \cdot x_g \cdot y_{i,j} \cdot y_{f,g} - x_i \cdot x_j \cdot y_{i,j}) \cdot e^{-dist(R^l,x)} = 0 \tag{20}$$

Для каждого региона получается система линейных уравнений размера $\frac{(depth+1)\cdot(depth+2)}{2}$.

2.6. Нежесткие условия в деревьях решений

Пусть условия принадлежности точки к региону c, где c_i имеет вид $x_i \leq c_i$ или $x_i > c_i$. Тогда оригинальное условие принадлежности точки можно расписать

$$\sum_{i=1}^{n} c_i(x) = depth \tag{21}$$

Ослабим это условие, то есть потребуем выполнения лишь части условий, также введем вероятность выполнения условия, определяемую как $P(c_i) = \lambda^{-|y \in learn: x_i > y_i > c_i|}$, для условия $x_i \leq c_i$ и $P(c_i)\lambda^{-|y \in learn: x_i < y_i < c_i|}$, для условия $x_i > c_i$, то есть вероятность экспоненциально падает пропорционально количеству точек, лучше выполняющих условие. Будем считать, что точка принадлежит региону, если вероятность того, что она выполняет хотя бы n_0-ому условий $> 1 - \alpha$.

$$P(\sum_{i=1}^{n} c_i(x) > n_0) > 1 - \alpha \tag{22}$$

Таким образом при подсчёте значения в листе используются точки, близкие к региону.

3. Эксперименты

Матрикс Нет состоящий из 1000 деревьев на маленьком пуле имеет следующий $pFound$ = 0.3073. Модель использующая линейную апроксимацию, так-же состоящие из 1000 деревьев уступила по $pFound$, модель имеет точность всего-лишь 0.3064. Модель состоящая из 1000 деревьем с квадратичной апроксимацией этот метод показал точность 0.3078.

На данных из большого пула, метод квадратичной апроксимации реализованный с помощью GPU, показал сравнимую скорость обучения с matrix net.

Заключение

В ходе курсовой работы были разработаны модификации методов построение регресии на базе забывчивых деревьев решений, включающих 6 разбиений и, соответственно, 64 листа. Из модификаций деревьев строился ансамбль методом градиентного бустинга, который на каждом шаге подбирает модель, наиболее подходящую к градиенту функции потерь.

Была разработана и протестирована модель линейной аппроксимации по значимым факторам в листьях дерева. В классической модели в листьях дерева решений лежат константы, в моей работе рассматривалась замена константы линейной функцией от тех факторов, которые были использованы для построения модели. Коэффициенты регрессии подбирались так, чтобы минимизировать сумму квадратов отклонений значения регрессии от значений в точках из обучающего множества.

- Этот метод имеет меньшу точность, чем matrix net и отстает от него по мере $pFound$ на 2%.

Помимо этого была протестирована модель квадратичной аппроксимации, использующая вместо константы в листьях квадратичную функцию, также зависящую только от тех факторов, которые были использованы для построения изначального разбиения. Коэффициенты квадратичной аппроксимации подбирались так, чтобы минимизировать сумму квадратов отклонения на обучающем множестве и сумму квадратов отклонения от условий непрерывности при переходе между регионами, разбивающими пространство.

- Этот метод был оптимизирован с помощью графического процессора. Было достигнуто ускорение в сравнении с реализацией на CPU в 75 раз .
- Также метод получил прирост в точности на 1.5% относительно базового метода.

Реализована модель, в которой нет жесткого разделения между регионами, а коэффициент влияния региона на точку экспоненциально падает при удалении точки от региона. Значение функции внутрии региона вычисляется квадратичной апрокисимацией значимых факторов.

Была реализована модель нежестких условий деревьев решений, где принадлежность к региону определяется не строгой логикой, а нечеткой, таким образом увеличивается количество точек, попавших в регион.
Для исследования были реализованы программные компоненты на языках C++ и Java. Алгоритмы были протестированы на данных, предоставленных компанией Яндекс. Наилучший результат показал метод квадратичной аппроксимации, превзойдя

по точности базовый метод ранжирования на 1.5%, но на обработки одно точки требует в 28 раз больше времени.

Список литературы

[1] Jerome H. Friendman, http://statweb.stanford.edu/ jhf/ftp/stobst.pdf, 1999.

[2] Игорь Кураленок, Александр Щекалев, http://www.osp.ru/os/2013/08/13037858/, 2013.

[3] Andrey Gulin, Igor Kuralenok, Dmitry Pavlov, Winning The Transfer Learning Track of Yahoo!’s Learning To Rank Challenge with YetiRank, 2011.