2021

Лекция 15. Распределенные алгоритмы

МЕТОДЫ МАШИННОГО ОБУЧЕНИЕ

ПАПУЛИН С.Ю. (papulin.study@yandex.ru)

Дополнительные материалы доступны по ссылке: https://github.com/mlmethods

Содержание

1. Среднее значение и стандартное отклонение

Среднее значение

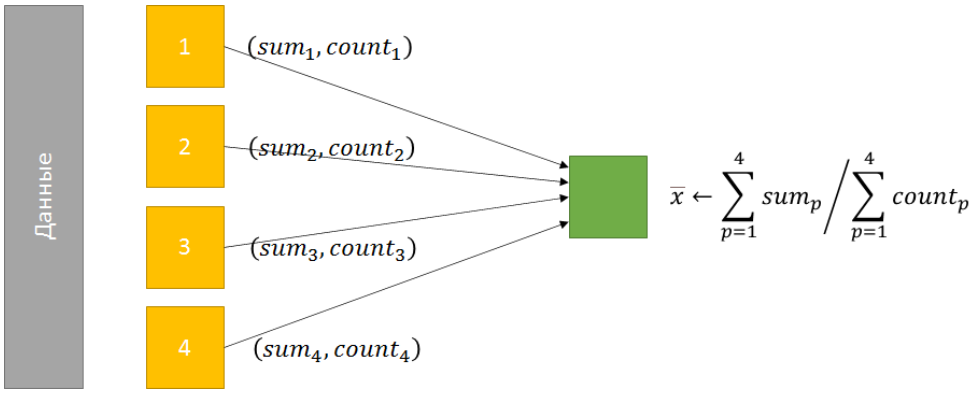
Пусть у нас есть набор данных D, который содержит n элементов. Каждый элемент x – вещественное число, то есть $x \in \mathbb{R}$. Тогда среднее значение можно вычислить следующим образом

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Предположим, что набор данных D разбит на P непересекающихся частей (подмножества). Например, значения D распределены между P вычислительными узлами так, что нет возможности собрать все данные на одном узле и рассчитать среднее значение. Часть данных будем обозначать как D^p, где p – индекс части. В этом случае необходимо вычислить сумму значений и количество элементов каждой части из P. Тогда среднее значение можно вычислить следующим образом

$$\overline{x} = \frac{\sum_{p=1}^{P} sum_p}{\sum_{p=1}^{P} count_p}.$$

Ниже приведены схематичное представления и алгоритм распределенного вычисления среднего значения.



Алгоритм 1. Distributed Mean	
1	for p in $1..P$ in parallel:
2	$sum_p \leftarrow 0$
3	$count_p \leftarrow 0$
4	for x in D^p:
5	$sum_p \leftarrow sum_p + x$
6	$count_p \leftarrow count_p + 1$
7	collect$\big((sum_1, count_1), .., (sum_P, count_P)\big)$
8	$\overline{x} \leftarrow \sum_{p=1}^{P} sum_p \Big/ \sum_{p=1}^{P} count_p$

Стандартное отклонение

В общем виде стандартное отклонение вычисляется как

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}.$$

В данном выражении для расчета используется среднее значение $\overline{x}$. Это означает, что сперва необходимо вычислить $\overline{x}$. Преобразуем исходную формулу так, чтобы избавиться от предварительного вычисления среднего

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2 - \left(\frac{1}{n}\sum_{i=1}^{n}x_i\right)^2}.$$

Таким образом, алгоритм может вычислить стандартное отклонение за один проход по данных D.

Если данные разбиты на P частей, как рассматривалось ранее, то такой подход требует получения дополнительных данных от каждой части для расчета стандартного отклонения. Помимо суммы (sum_p) и количества элементов ($count_1$) каждой части, вычисляется сумма квадратов ($ssum_p$). После этого вычисляется общее количество элементов, сумма и сумма квадратов от всех частей

$$n = \sum_{p=1}^{P} count_p,$$

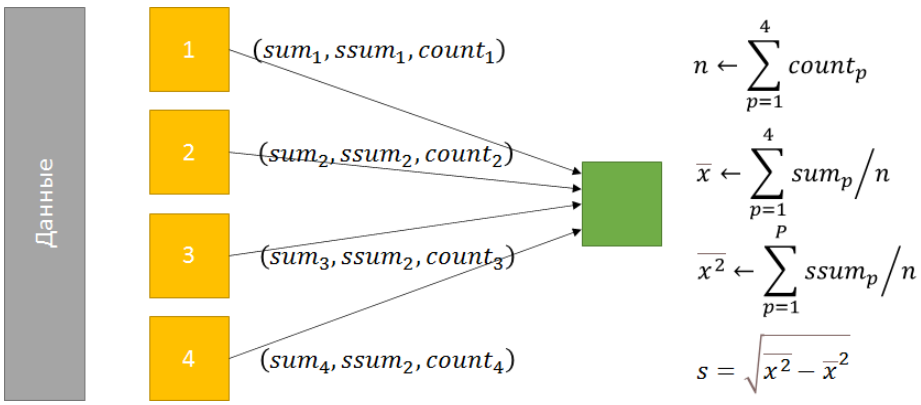
$$\overline{x} = \sum_{p=1}^{P} sum_p \Big/ n,$$

$$\overline{x^2} = \sum_{p=1}^{P} ssum_p \Big/ n.$$

В итоге стандартное отклонение примет вид

$$s = \sqrt{\overline{x^2} - \overline{x}^2}$$

Ниже приведен пример при разбиении данных на четыре части



Общий алгоритм распределенного вычисления стандартного отклонения приведен ниже

Алгоритм 2. Distributed Standard Deviation	
1	for p in $1..P$ in parallel:

2	$sum_p \leftarrow 0$
3	$ssum_p \leftarrow 0$
4	$count_p \leftarrow 0$
5	for x in D^p:
6	$sum_p \leftarrow sum_p + x$
7	$ssum_p \leftarrow ssum_p + x^2$
8	$count_p \leftarrow count_p + 1$
9	collect$\big((sum_1, ssum_1, count_1), .., (sum_P, ssum_P, count_P)\big)$
10	$n \leftarrow \sum_{p=1}^{P} count_p; \overline{x} \leftarrow \sum_{p=1}^{P} sum_p/n; \overline{x^2} \leftarrow \sum_{p=1}^{P} ssum_p/n$
11	$s = \sqrt{\overline{x^2} - \overline{x}^2}$

2. Косинусное сходство

Исходные данные:

$$X = \begin{bmatrix} - & x_1 & - \\ \vdots & \ddots & \vdots \\ - & x_n & - \end{bmatrix}^{n \times p} = \begin{bmatrix} | & \cdots & | \\ c_1 & \ddots & c_p \\ | & \cdots & | \end{bmatrix}^{n \times p}$$

Найти:

$$\text{cosine}(i,j) = \frac{{c_i}^T c_j}{\|c_i\|\|c_j\|} = \frac{\sum_{k=1}^{n} x_{ki} x_{kj}}{\|c_i\|\|c_j\|},$$

где

$$c_i = \begin{bmatrix} x_{1i} \\ \vdots \\ x_{ni} \end{bmatrix} \text{ и } c_j = \begin{bmatrix} x_{1j} \\ \vdots \\ x_{nj} \end{bmatrix}$$

Решение:

Внешнее произведение векторов:

$$x_i {x_i}^T = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{bmatrix} [x_{i1} \quad \cdots \quad x_{ip}] = \begin{bmatrix} x_{i1}x_{i1} & \cdots & x_{i1}x_{ip} \\ \vdots & \ddots & \vdots \\ x_{ip}x_{i1} & \cdots & x_{ip}x_{ip} \end{bmatrix}$$

Тогда внешнее перемножение матриц можно представить как

$$X^T X = \sum_{i=1}^{n} x_i {x_i}^T = \begin{bmatrix} \sum_{i=1}^{n} x_{i1}x_{i1} & \cdots & \sum_{i=1}^{n} x_{i1}x_{ip} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_{ip}x_{i1} & \cdots & \sum_{i=1}^{n} x_{ip}x_{ip} \end{bmatrix} = \begin{bmatrix} {c_1}^T c_1 & \cdots & {c_1}^T c_p \\ \vdots & \ddots & \vdots \\ {c_p}^T c_1 & \cdots & {c_p}^T c_p \end{bmatrix}$$

Ниже приведен алгоритм распределенного вычисления косинусного сходства, который состоит из двух стадий – map (трансформация данных) и reduce (агрегирование)

Алгоритм 1. Cosine Similarity. Map		
1	for m in $1..M$ in parallel:	M map-задач
2	for x in D^m:	D^m – наблюдения в части m
3	for (x_j, x_k) in all_pairs_x:	$x = (x_1, \dots, x_p)$ – одно наблюдение
4	$v \leftarrow x_j \cdot x_k$	
5	emit$((j,k) \rightarrow v)$	$key \rightarrow value$

Алгоритм 2. Cosine Similarity. Reduce		
1	for r in $1..R$ in parallel:	R reduce-задач с разделением по ключу (partitioned by key)
2	receive$(\|\|c_1\|\|, \dots, \|\|c_p\|\|)$	Заранее вычисленные значения
3	receive pairs from map part	
4	for $(i,j) \rightarrow [v_1, \dots, v_K]$ in $pairs$:	v_k – ненулевые значения из map части
5	$c_i^T c_j \leftarrow \sum_{k=1}^{K} v_k$	
6	$cosine(i,j) \leftarrow \frac{c_i^T c_j}{\|\|c_i\|\|\|\|c_j\|\|}$	

3. Градиентный спуск

Функция потерь в общем виде:

$$L(y, x, \theta) = \sum_{(x,y)\in D} l(y, x, \theta)$$

Для линейной регрессии:

$$l(y, x, \theta) = (y - x^T\theta)^2$$

Градиент от функции потерь:

$$\nabla L(y, x, \theta) = \sum_{(x,y)\in D} (y - x^T\theta)x$$

Алгоритм 1. GD	
1	initialize(θ)
2	for i in $1..T$:
3	$grad \leftarrow 0$
4	for (x,y) in D:
5	$grad \leftarrow grad + (y - x^T\theta)x$
6	$\theta \leftarrow \theta - \eta \cdot grad$
7	return θ

Общий вид записи итерации в градиентном спуске:

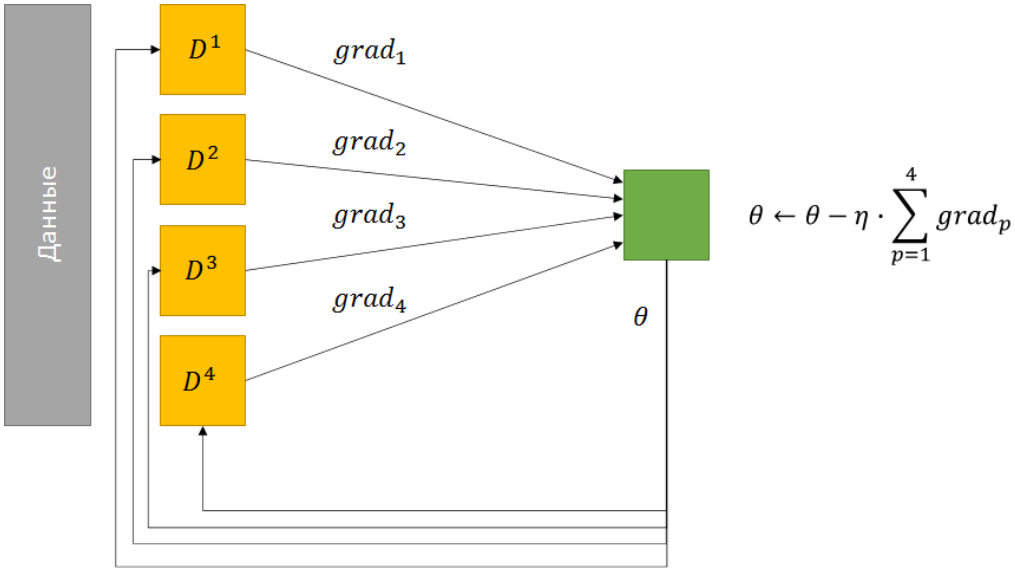
$$\theta \leftarrow \theta - \eta \cdot \nabla L(y, x, \theta)$$

Распределенный градиентный спуск

Алгоритм 2. Distributed GD	
1	initialize(θ)
2	for i in $1..T$:
3	broadcast(θ)
4	for p in $1..P$ in parallel:
5	$grad_p \leftarrow 0$
6	for (x, y) in D^p:
7	$grad_p \leftarrow grad_p + (y - x^T\theta)x$
8	collect($grad_1, .., grad_P$)
9	$\theta \leftarrow \theta - \eta \cdot \sum_{p=1}^{P} grad_p$
10	return θ

Общий вид записи итерации в распределенном градиентном спуске:

$$\theta \leftarrow \theta - \eta \cdot \sum_{p=1}^{P} \sum_{(x,y)\in D^p} (y - x^T\theta)x$$



$$\nabla L(y, x, \theta) = \sum_{(x,y)\in D} (y - x^T\theta)x$$
$$= \sum_{(x,y)\in D^1} (y - x^T\theta)x + \sum_{(x,y)\in D^2} (y - x^T\theta)x + \sum_{(x,y)\in D^3} (y - x^T\theta)x$$
$$+ \sum_{(x,y)\in D^4} (y - x^T\theta)x$$

4. Стохастический градиентный спуск

Градиент от функции потерь одного наблюдения в линейной регрессии:

$$\nabla l(y, x, \theta) = (y - x^T\theta)x$$

Алгоритм 1. SGD	
1	initialize(θ)
2	for i in $1..T$:
3	shuffle(D)
4	for (x, y) in D:
5	$\theta \leftarrow \theta - \eta\nabla l(y, x, \theta)$
6	return θ

Общий вид записи итерации в стохастическом градиентном спуске:

$$\theta \leftarrow \theta - \eta\nabla l(y, x, \theta)$$

Стохастический градиентный спуск с мини-пакетами

Алгоритм 2. SGD: mini-batch		
1	initialize(θ)	
2	for i in $1..T$:	эпохи
3	shuffle(D)	выборка без возврата
4	batches $\leftarrow$ split$(D, batch_{size})$	
5	for $\mathcal{B}$ in batches:	
6	grad $\leftarrow 0$	
7	for (x, y) in $\mathcal{B}$:	
8	grad $\leftarrow$ grad $+ \nabla l(y, x, \theta)$	
9	$\theta \leftarrow \theta - \eta \cdot$ grad	
10	return θ	

Общий вид записи итерации в стохастическом градиентном спуске с мини-пакетами:

$$\theta \leftarrow \theta - \eta \cdot \sum_{(x,y)\in\mathcal{B}} \nabla l(y, x, \theta)$$

Параллельный стохастический градиентный спуск

Алгоритм 3. SGD: parallel	
1	for p in $1..P$ in parallel:
2	initialize(θ^p)
3	shuffle(D^p)
4	for (x, y) in D^p:
5	$\theta^p \leftarrow \theta^p - \eta\nabla l(y, x, \theta^p)$

6	collect$(\theta^1, .., \theta^P)$
7	$\theta = \frac{1}{P}\sum_{i=0}^{P} \theta^i$
8	return θ

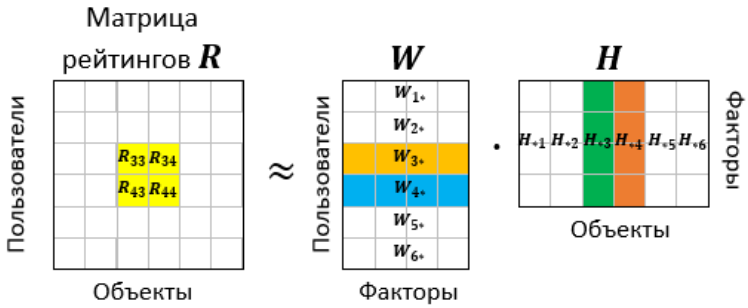
Стохастический градиентный спуск с общей памятью

Алгоритм 4. SGD: HogWild!		
1	initialize(θ)	
2	for p in $1..P$ in parallel:	
3	shuffle(D^p)	
4	for (x, y) in D^p:	
5	$\Delta\theta \leftarrow -\eta\nabla l(y, x, \theta)$	
6	for i in $1..M$ with $\Delta\theta_i \neq 0$:	
7	$\theta_i \leftarrow \theta_i + \Delta\theta_i$	Атомарная операция
8	return θ	

Алгоритм 5. SGD: Spark mini-batch	
1	initialize(θ)
2	for i in $1..T$:
3	broadcast(θ)
4	for p in $1..P$ in parallel:
5	$\mathcal{B}^p \leftarrow$ sample$(D^p, batch_{size})$
6	$\text{grad}_p \leftarrow 0$
7	for (x, y) in $\mathcal{B}^p$:
8	$\text{grad}_p \leftarrow \text{grad}_p + \nabla l(y, x, \theta)$
9	collect$(\text{grad}_1, .., \text{grad}_P)$
10	$\theta \leftarrow \theta - \eta \cdot \sum_{p=1}^{P} \text{grad}_p$
11	return θ

$$\theta \leftarrow \theta - \eta \cdot \sum_{p=1}^{P} \sum_{(x,y)\in\mathcal{B}^p} \nabla l(y, x, \theta)$$

5. Распределённая факторизация матрицы рейтингов



Задача оптимизации

$$\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{H}} = \arg\min_{\boldsymbol{W},\boldsymbol{H}} L(\boldsymbol{W}, \boldsymbol{H})$$

Функция потерь

$$L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{(i,j)\in\Omega} L_{ij}(\boldsymbol{W}, \boldsymbol{H})$$

$$L_{ij}(\boldsymbol{W}, \boldsymbol{H}) = l\big(\boldsymbol{R}_{ij}, \boldsymbol{W}_{i*}, \boldsymbol{H}_{*j}\big)$$

Пример функции потерь (без регуляризации)

$$L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{(i,j)\in\Omega} \big(\boldsymbol{R}_{ij} - \boldsymbol{W}_{i*}\boldsymbol{H}_{*j}\big)^2$$

Факторизация неполной матрицы:

- Градиентный спуск
- Стохастический градиентный спуск
- Чередование наименьших квадратов (Alternating Least Squares – ALS)

Метод наименьших квадратов с чередованием

$$L(\boldsymbol{W}, \boldsymbol{H}) = \sum_{(i,j)\in\Omega} \big(\boldsymbol{R}_{ij} - \boldsymbol{W}_{i*}\boldsymbol{H}_{*j}\big)^2$$

$$\frac{\partial L(\boldsymbol{W}, \boldsymbol{H})}{\partial \boldsymbol{W}_{i*}} = 0$$

$$\frac{\partial L(\boldsymbol{W}, \boldsymbol{H})}{\partial \boldsymbol{H}_{*j}} = 0$$

$$\boldsymbol{R}_{i*} - \boldsymbol{W}_{i*}\boldsymbol{H}^{(n)} = \boldsymbol{0}$$

$$\boldsymbol{R}_{*j} - \boldsymbol{W}^{(n+1)}\boldsymbol{H}_{*j} = \boldsymbol{0}$$

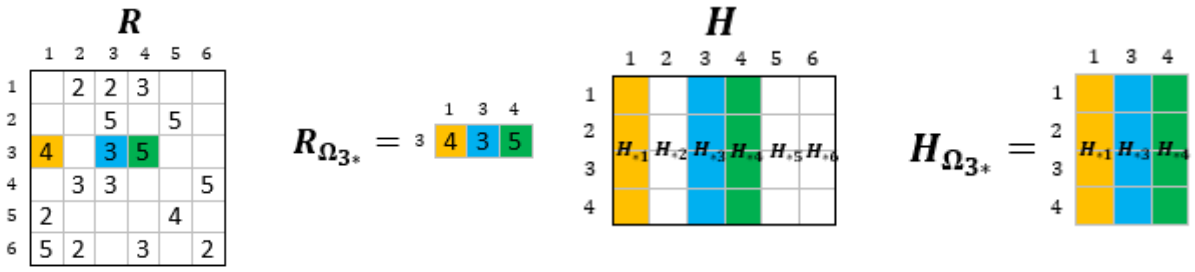
Шаг 1. Вычисление $\boldsymbol{W}$ при фиксированном $\boldsymbol{H}$

Метод наименьших квадратов для задачи линейной регрессии:

$$\widehat{\boldsymbol{\theta}} = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{y}$$

Для матрицы рейтингов

$$\boldsymbol{W}_{i*}^{(n+1)^{\boldsymbol{T}}} = \left(\boldsymbol{H}_{\boldsymbol{\Omega}_{i*}}^{(n)}\boldsymbol{H}_{\boldsymbol{\Omega}_{i*}}^{(n)^{\boldsymbol{T}}}\right)^{-1}\boldsymbol{H}_{\boldsymbol{\Omega}_{i*}}^{(n)}\boldsymbol{R}_{\boldsymbol{\Omega}_{i*}}{}^{\boldsymbol{T}}$$

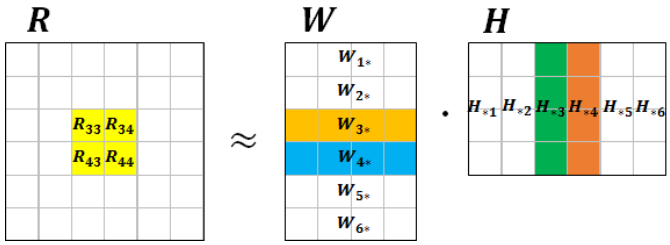


Шаг 2. Вычисление $\boldsymbol{H}$ при фиксированном $\boldsymbol{W}$

$$\boldsymbol{H}_{*\boldsymbol{j}}^{(n+1)} = \left(\boldsymbol{W}_{\boldsymbol{\Omega}_{*\boldsymbol{j}}}^{(n+1)^{\boldsymbol{T}}}\boldsymbol{W}_{\boldsymbol{\Omega}_{*\boldsymbol{j}}}^{(n+1)}\right)^{-1}\boldsymbol{W}_{\boldsymbol{\Omega}_{*\boldsymbol{j}}}^{(n+1)^{\boldsymbol{T}}}\boldsymbol{R}_{\boldsymbol{\Omega}_{*\boldsymbol{j}}}$$

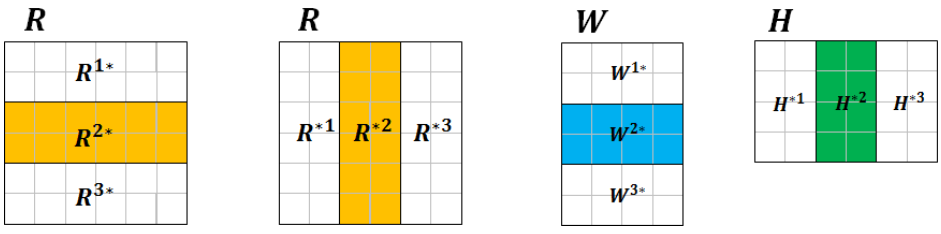
Распределенный ALS

Факторизация матрицы рейтингов на одном вычислительном узле

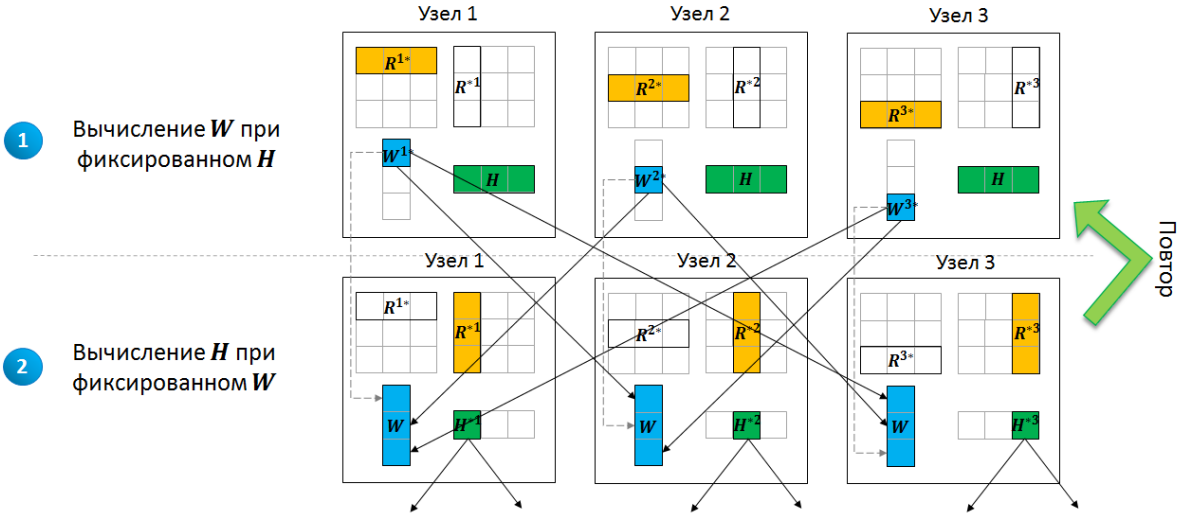


Распределенная факторизация матрицы рейтингов

На каждом узле хранятся часть рейтингов по горизонтали и вертикали (R^{i*} и R^{*j}) и матрицы W и H. При этом на каждом узле зафиксированная матрица хранит полностью в неизменном виде и обновляется только часть используемой в качестве неизвестных параметров.



Схематично алгоритм обучения приведены на рисунке ниже



1 $H \leftarrow$ средний рейтинг для первой строки и малые случайные значения для остальных
2 Цикл: критерий остановки
3 $W \leftarrow$ вычисление при фиксированном H распределено на d узлах
4 Обновление W на всех узлах
5 $H \leftarrow$ вычисление при фиксированном W распределено на d узлах
6 Обновление H на всех узлах

Список литературы

1. Zadeh, Reza & Carlsson, Gunnar. (2013). Dimension Independent Matrix Square using MapReduce.
2. CME 323: Distributed Algorithms and Optimization, Spring 2015 http://stanford.edu/~rezab/dao. Instructor: Reza Zadeh, Databricks and Stanford. Lecture 10, 4/29/2015. Scribed by Jan Dlabal, Fang-Chieh Chou, Yu-Wei Lin, Yi-Hong Kuo.
3. Lars Schmidt-Thieme. Lecture Note. Big Data Analytics. Distributed Machine Learning Algorithms: Distributed Stochastic Gradient Descent, University of Hildesheim, Germany
4. Niu, Feng & Recht, Benjamin & Ré, Christopher & Wright, Stephen. (2011). HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient
5. R. Gemulla, P. J. Haas, E. Nijkamp, Y. Sismanis «Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent», IBM Research Report RJ10481, March 2011 Revised February, 2013
6. Yunhong Zhou, Dennis Wilkins, Robert Schreiber, Rong Pan «Large-Scale Parallel Collaborative Filtering for the Netflix Prize» AAIM '08 Proceedings of the 4th international conference on Algorithmic Aspects in Information and Management. PP 337–348, Springer-Verlag Berlin, Heidelberg, 2008