Федеральный исследовательский центр “Информатика и управление”
Российской Академии Наук

На правах рукописи

Трофимов Илья Егорович

Разработка и обоснование методов параллельного покоординатного спуска для обучения обобщенных линейных моделей с регуляризацией

05.13.17 - теоретические основы информатики

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель
д. ф.-м. н. Воронцов К.В.

Москва - 2018

Содержание

Введение

Работа посвящена разработке и теоретическому обоснованию методов параллельного покоординатного спуска для обучения обобщенных линейных моделей с регуляризацией. Разработанные методы применимы для обучения с использованием больших выборок и выполнения на вычислительном кластере.

Актуальность темы. В настоящее время во многих задачах машинного обучения и анализа данных возникают большие обучающие выборки. В качестве примера можно привести задачи поиска в интернете, онлайн рекламы, обработки текстов, анализа показателей датчиков, генетики и т.д. Такие задачи характеризуются большим числом обучающих примеров, высокой размерностью, или и тем и другим одновременно. Обучающие выборки, как правило, разреженные. Желательным свойством также является разреженность полученного решения. Если в этих задачах использовать для обучения только часть имеющихся данных, то качество предсказания, как правило, падает. Поэтому важным направлением исследований является разработка методов машинного обучения, специально предназначенных для больших выборок, а также разработка алгоритмов, позволяющих применять существующие методы на больших выборках.

Обобщенные линейные модели (generalized linear models, GLM) - это класс статистических моделей, в которых предполагается, что зависимая переменная y связана с вектором независимых переменных $\mathbf{x}$ через нелинейную функцию от скалярного произведения с вектором весов $\boldsymbol{\beta}^T\mathbf{x}$. В класс обобщенных линейных моделей входят: логистическая регрессия, пробит регрессия, пуассоновская регрессия, линейная регрессия с квадратичной функцией потерь и некоторые другие менее распространенные модели. Для обеспечения устойчивости к переобучению и стабильной сходимости численных методов обычно используется регуляризация. Наиболее часто используется L_1 или L_2 регуляризация. Также иногда применяются одновременно оба вида регуляризации, данный метод получил название elastic net [1]. Для случая категориальных переменных используется регуляризатор group lasso [2, 3]. Более редким является использование невыпуклых регуляризаторов SCAD [4] или MCP [5].

Обучение GLM сводится к минимизации целевой функции - суммы эмпирического риска и регуляризации. Это является задачей численной оптимизации. Использование L_2 регуляризации - более простое, т.к. соответствующая целевая функция выпуклая и гладкая. Однако случай негладкого L_1 регуляризатора является более сложным. Существует несколько подходов к обучению обобщенных линейных моделей на больших обучающих

выборках с L_1 и L_2 регуляризацией.

Последовательные алгоритмы

Первый подход - онлайн обучение. В онлайн обучении элементы из обучающего множества обрабатываются по одному. Поэтому необязательно хранить обучающую выборку в оперативной памяти, ее можно читать последовательно с диска или получать по сети. Примерами таких методов являются: стохастический градиентный спуск (SGD), RMMP [6], онлайн обучение с усеченным градиентом (online learning via truncated gradient) [7, 8] и метод FTRL-Proximal [9, 10]. С одной стороны, данные методы позволяют получить регрессию приемлемого качества за небольшое число проходов по обучающей выборке. С другой стороны, в них присутствует несколько гиперпараметров (темп обучения, скорость затухания темпа обучения, количество проходов и др.) от которых существенно зависит качество регрессии. На практике оптимальные гиперпараметры подбираются с использованием тестовой выборки или кросс-валидации. Эта задача может быть сложной, так как процедура обучения на большой обучающей выборке обычно занимает продолжительное время. Полезная особенность, отличающей онлайн обучения от остальных методов - это возможность обучения в реальном времени (по мере прихода свежих данных), что позволяет подстраиваться под изменяющееся распределение.

Второй подход - это методы покоординатного спуска. Методы покоординатного спуска по очереди обновляют одну или несколько переменных, стремясь минимизировать целевую функцию или приближение к ней. Методы покоординатного спуска универсальны и позволяют работать как с L_1, так и с L_2 регуляризацией. В статье [11] проведено сравнение большого числа методов для решения задачи линейной классификации с L_1 регуляризацией и сделан вывод, что методы покоординатного спуска работают лучше всего. На практике чаще всего используются следующие алгоритмы этого типа: BBR [12], GLMNET [13], newGLMNET [14]. Все эти методы работают последовательно на одном компьютере. Также их программные реализации требуют загрузки обучающей выборки в оперативную память (RAM).

Третий подход - квазиньютоновские методы Limited Memory BFGS (L-BFGS) [15], TRON [16]. Квазиньютоновские методы эффективно решают задачу оптимизации в случае выпуклой гладкой целевой функции, т.е. применимы для L_2 регуляризации.

Параллельные алгоритмы

В последнее время все чаще встречаются задачи, в которых обучающие выборке настолько большие, что даже описанные методы (предназначенные для выполнения на одном компьютере) работают слишком долго. Для решения этой проблемы необходимо модифицировать описанные выше методы, адаптировав их для параллельного выполнения на одном компьютере с несколькими процессорами или в распределенной системе.

Универсальным способом параллельной оптимизации является метод ADMM [17]. Он применим для задач оптимизации, возникающих при обучении GLM с регуляризацией. Разные варианты ADMM обеспечивают параллелизм как по обучающим примерам, так и по переменным. Обратной стороной универсальности является медленная сходимость метода ADMM.

Алгоритмы онлайн обучения могут выполняться параллельно в распределенных системах с помощью техники, описанной в [18, 19]. Обучающая выборка разделяется между узлами кластера по примерам, на каждой подвыборке обучается независимо обучаются классификаторы и усредняются после каждой эпохи. Усредненный вектор весов используется как начальное приближение на следующей эпохе, и т.д. Данный подход имеет те же плюсы и минусы, что и последовательное онлайн обучение на одном компьютере. Ускорение от количества машин - сублинейное, что является недостатком.

Квазиньютоновские методы Limited Memory BFGS (L-BFGS) [15], TRON [16] могут быть эффективно распараллелены для выполнения на кластере [18, 16] при разбиении обучающей выборки по примерам.

Естественное обобщение методов покоординатного спуска - это выполнение параллельных шагов по нескольким переменным. Именно так работает алгоритм Shotgun [20]. Он основан на параллельных шагах по случайно выбранным переменным. Недостатком этого алгоритма является то, что существует верхний предел по количеству параллельных обновлений, при котором алгоритм сходится. Этот предел зависит от обучающей выборки и его теоретические оценки тяжело вычислимы на практике. В статье [21] используются параллельные шаги по случайным координатам для оптимизации частично сепарабельной целевой функции. Частичная сепарабельность будет иметь место если выборка разреженная. Также в статье [21] проведен теоретический анализ максимального допустимого количества параллельных шагов. В алгоритме GRock [22] множество переменных для обновления выбирается жадно, т.е. шаги выполняются по нескольким переменным, дающим наибольшее уменьшение целевой функции. Проведенные численные эксперименты пока-

зали, что алгоритм GRock сходится быстрее, чем параллельный вариант FISTA [23] и ADMM [17]. Алгоритм Shotgun может быть запущен в распределенной системе с помощью технологии Stale Synchronous Parallel Parameter Server (SSPPS) Ho et al. [24]. Также, как и стандартный Shotgun, данный вариант не всегда сходится.

Альтернативный подход к распараллеливанию покоординатного спуска состоит в том, чтобы, не меняя последовательности вычислений (с точки зрения математики), ускорить их, распределив операции между процессорами. Такой подход описан в статье [25] для Пуассоновской регрессии. В ней вариант метода BBR [12] запускается на сервере с 448 ядрами Graphical Processing Units (GPU). Плюсом данного подхода является гарантированная сходимость и хорошее ускорение. Недостатком является необходимость загружать обучающую выборку в память GPU (которая меньше RAM) а также относительная дороговизна серверов с GPU.

Цель диссертационной работы - разработка и обоснование методов параллельного покоординатного спуска для обучения обобщенных линейных моделей с регуляризацией. Разработка методов, применимых для больших обучающих выборок и выполнения на вычислительном кластере.

Методы исследования. В данной работе использованы, с одной стороны, теоретические методы исследования, опирающиеся на математический анализ и линейную алгебру; с другой стороны, используется метод численного эксперимента на вычислительном кластере.

Основные положения, выносимые на защиту:

1. Метод `d-GLMNET`, выполняющий параллельный покоординатный спуска для минимизации функций риска обобщенных линейных моделей с регуляризацией “elastic net”;

2. Достаточные результаты сходимости и линейной скорости сходимости метода `d-GLMNET`;

3. Метод “асинхронной балансировки нагрузки” для обеспечения эффективного выполнения метода `d-GLMNET` при наличии медленных узлов кластера;

4. Численные эксперименты, доказывающие, что метод `d-GLMNET` получает разреженные решения при использовании L_1-регуляризации;

5. Численные эксперименты, доказывающие, что метод `d-GLMNET` более эффективен, чем общепринятые методы при работе с разреженными обучающими выборками с высокой размерностью признакового пространства.

6. Общедоступная программная реализация метода `d-GLMNET` : `https://github.com/IlyaTrofimov/dlr`.

Научная новизна данной работы заключается в разработке нового метода минимизации функций риска обобщенных линейных моделей с регуляризацией "elastic net". Метод эффективно работает с большими обучающими выборками (big data) с использованием вычислительного кластера. Научной новизной обладают теоретические результаты относительно сходимости метода, а также модификация метода (асинхронная балансировка нагрузки), обеспечивающая эффективное выполнение при неравномерности скорости работы узлов кластера.

Теоретическая значимость состоит в установлении достаточных условий сходимости и линейной скорости сходимости разработанного метода `d-GLMNET`, в том числе, для модификации метода `d-GLMNET`, использующей технику асинхронной балансировки нагрузки.

Практическая значимость определяется тем, что разработанный метод `d-GLMNET` позволяет проводить обучение обобщенных линейных моделей быстрее, чем при использовании общепринятых методов, что позволяет экономить вычислительные ресурсы и получать более точные решения при ограниченном бюджете вычислительных ресурсов.

Теоретические и экспериментальные результаты данной работы используются в курсах "Машинное обучение и большие данные", которые автор читал в 2015-2018 гг. на факультете инноваций и высоких технологий (ФИВТ) МФТИ и Школе анализа данных (ШАД) Яндекса.

Степень достоверности. Достоверность результатов обеспечивается доказательствами теорем и описаниями проведенных экспериментов, допускающими их воспроизводимость. Исходный код программ и выборки, использовавшиеся для вычислительных экспериментов общедоступны.

Апробация работы. Основные положения и результаты работы докладывались автором на конференциях:

- Sixth International Workshop on Data Mining for Online Advertising and Internet Economy, 2012, Пекин;
- 22nd International Conference on World Wide Web, 2013, Рио-де-Жанейро;
- Machine learning and Very Large Data Sets, 2013, Москва;

- Seventeenth International Conference on Artificial Intelligence and Statistics, 2014, Рейкьявик;
- The 4-th International Conference on Analysis of Images, Social Networks and Texts, 2015, Екатеринбург;
- Machine Learning: Prospects and Applications, 2015, Берлин.

Публикации. По тематике исследований опубликовано 5 статей [26, 27, 28, 29, 30], в том числе 2 статьи [29, 30] в изданиях, входящих в список ВАК и 2 статьи, входящие в индекс SCOPUS [28, 29].

Структура работы. Диссертация состоит из введения, 5 глав, заключения, приложения и списка литературы. Материал изложен на 116 стр., содержит 19 рисунков, 5 таблиц, 18 алгоритмов и 103 наименований в списке литературы.

Личный вклад диссертанта является решающим во всех результатах, выносимых на защиту.

Благодарности

Автор благодарит Константина Вячеславовича Воронцова за помощь при подготовке диссертации, Илью Борисовича Мучника за поддержку и ценные советы по организации работы и Александра Вениаминовича Генкина за плодотворное научное сотрудничество. Автор признателен компании Яндекс за предоставление ресурсов для вычислительных экспериментов и лояльное отношение к научным исследованиям.

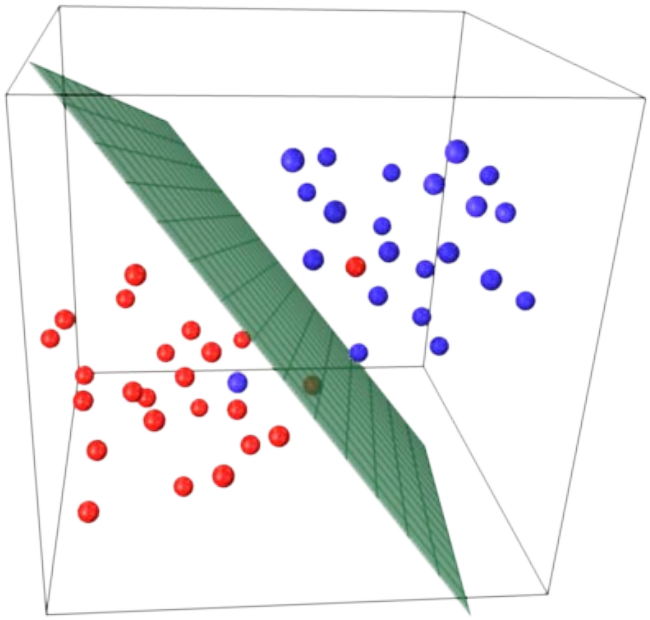


Рис. 1: Линейная бинарная классификация

1 Методы минимизации функций эмпирического риска GLM с регуляризацией

1.1 Обобщенные линейные модели

В данной работе рассматривается задача машинного обучения по прецедентам. Цель этой задачи - научиться предсказывать по вектору признаков $\mathbf{x} \in \mathbb{R}^p$ метку y. Если $y \in \{-1, +1\}$, то получается задача бинарной классификации, если $y \in \mathbb{R}$ - то задача регрессии. Обучение происходит по обучающей выборке, состоящей из пар $\{\mathbf{x}_i, y_i\}_{i=1}^n$. Все вектора $\mathbf{x}_i$ вместе образуют матрицу *объекты-признаки* X. Будем обозначать количество ненулевых элементов в матрице X через nnz.

В случае задачи бинарной классификации строится разделяющая поверхность $\boldsymbol{\beta}^T\mathbf{x} > a$ для некоторого порога a. Объекты по разные стороны от разделяющей гиперплоскости относятся к разным классам (см. рис. 1).

Обобщенные линейные модели - это класс статистических моделей, в которых предполагается, что зависимая переменная Y имеет распределение с мат. ожиданием μ

$$\mathbb{E}[Y] = \mu = g^{-1}(\boldsymbol{\beta}^T\mathbf{x})$$

и дисперсией, зависящей только от мат. ожидания

$$\mathbb{D}[Y] = \phi V(\mu)$$

Функция $g(\cdot)$ называется функцией связи (link function), а величина ϕ - параметром дисперсии. Наиболее часто используемые на практике распределения Y принадлежат к экспоненциальному семейству, т.е. функции плотности распределения имеет вид

$$P(y;\theta,\phi) = \exp\left(\frac{y\theta - b(\theta)}{\phi + c(y,\phi)}\right)$$

Можно показать, что для экспоненциального семейства

$$\mathbb{E}[Y] = b'(\theta)$$
$$\mathbb{D}[Y] = \phi b''(\theta)$$

Функция связи называется *канонической*, если $g(\cdot) = (b')^{-1}(\cdot)$. Все рассматриваемые в этом разделе GLM, кроме пробит-регрессии, имеют каноническую функцию связи. В этом случае имеем

$$g(\mu) = g(\mathbb{E}[Y]) = g(b'(\theta)) = \theta = \boldsymbol{\beta}^T\mathbf{x}$$

Таким образом, распределение Y зависит от $\boldsymbol{\beta}$ только через линейную комбинацию $\boldsymbol{\beta}^T\mathbf{x}$.

$$P(y|\mathbf{x}) = \exp\left(\frac{y\boldsymbol{\beta}^T\mathbf{x} - b(\boldsymbol{\beta}^T\mathbf{x})}{\phi + c(y,\phi)}\right)$$

Перечислим наиболее часто используемые GLM:

- Линейная регрессия, $y \in \mathbb{R}$:

$$P(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi}}\exp\left(\frac{(y - \boldsymbol{\beta}^T\mathbf{x})^2}{2}\right)$$

- Логистическая регрессия, $y \in \{-1, +1\}$:

$$P(y|\mathbf{x}) = \frac{1}{1 + \exp(-y\boldsymbol{\beta}^T\mathbf{x})}$$

- Пробит регрессия, $y \in \{-1, +1\}$:

$$P(y|\mathbf{x}) = \Phi(y\boldsymbol{\beta}^T\mathbf{x})$$

  где $\Phi(\cdot)$ - это функция распределения стандартного нормального распределения

- Пуассоновская регрессия, $y \in \mathbb{N}$:

$$P(y|\boldsymbol{\beta}^T\mathbf{x}) = \frac{\exp(-\lambda)\lambda^n}{n!}, \text{ где } \lambda = \exp(\boldsymbol{\beta}^T\mathbf{x})$$

Обучение GLM выполняется через максимизацию правдоподобия на обучающей выборке

$$\max_{\boldsymbol{\beta}} \prod_{i=1}^{n} P(y_i|\boldsymbol{\beta}^T\mathbf{x}_i)$$

что эквивалентно минимизации минус лог-правдоподобия

$$\min_{\boldsymbol{\beta}} \left\{ -\sum_{i=1}^{n} \log P(y_i|\boldsymbol{\beta}^T\mathbf{x}_i) \right\}$$

Обозначим

$$\ell(y_i, \boldsymbol{\beta}^T\mathbf{x}) = -\log P(y_i|\boldsymbol{\beta}^T\mathbf{x}_i)$$

Функция $\ell(y, \hat{y})$ называется *функцией потерь* и может быть интерпретирована как штраф за отличие истинного y от предсказанного $\hat{y}$. Таким образом, задачи классификации и регрессии свелись к задаче оптимизации

$$\min_{\boldsymbol{\beta}} L(\boldsymbol{\beta}), \qquad L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \ell(y_i, \boldsymbol{\beta}^T\mathbf{x}_i). \tag{1}$$

Данная задача может быть рассмотрена напрямую для заданной функции $\ell(y, \hat{y})$, без статистической интерпретации через обобщенные линейные модели. Функция $L(\boldsymbol{\beta})$ называется также *функцией эмпирического риска*.

Для описанных в этом разделе GLM получаются следующие функции потерь:

- линейная регрессия: $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
- логистическая регрессия: $l(y, \hat{y}) = \log(1 + \exp(-y\hat{y}))$
- пробит регрессия: $l(y, \hat{y}) = -\log(\Phi(y\hat{y}))$
- Пуассоновская регрессия: $l(y, \hat{y}) = \exp(\hat{y}) - y\hat{y}$

Использование всех вышеперечисленных функций потерь приводит к функции эмпирического риска $L(\boldsymbol{\beta})$, выпуклой по $\boldsymbol{\beta}$.

1.2 Виды регуляризации

Задача (1) обычно решается с помощью численных методов оптимизации. Качество решения может быть неудовлетворительным по следующим причинам:

- Решение может значительно изменяться при сколь угодно малых изменениях обучающей выборки (некорректно поставленная задача);

- Численные методы могут работать нестабильно;
- Решение может обладать плохой обобщающей способностью.

Для устранения этих проблем к функции риска добавляют регуляризацию $R(\boldsymbol{\beta})$ и решается задача минимизации *регуляризованного эмпирического риска*

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left(L(\boldsymbol{\beta}) + R(\boldsymbol{\beta}) \right) \tag{2}$$

Перечислим наиболее часто используемые виды регуляризации:

1. L_1 регуляризация

   $$R(\boldsymbol{\beta}) = \lambda_1 \|\boldsymbol{\beta}\|_1 = \lambda_1 \sum_{k=1}^{p} |\beta_k|$$

   Обладает тем свойством, что у решения часть компонент будут нулевыми. Чем больше λ_1 - тем больше нулевых компонент. Таким образом, L_1 регуляризация выполняет также отбор признаков. Кроме того, L_1 регуляризация имеет тенденцию отбирать по одному признаку из группы сильно коррелированных. Функция $R(\boldsymbol{\beta})$ - не гладкая в точке $\boldsymbol{\beta} = 0$.

2. L_2 регуляризация

   $$R(\boldsymbol{\beta}) = \lambda_2 \|\boldsymbol{\beta}\|_2^2 = \frac{\lambda_2}{2} \sum_{k=1}^{p} \beta_k^2$$

   Функция $R(\boldsymbol{\beta})$ - гладкая и не приводит к занулению компонент $\boldsymbol{\beta}$. Имеет тенденцию уменьшать по абсолютному значению коэффициенты $\boldsymbol{\beta}$, значения которых не могут быть достоверно определены по обучающей выборке.

3. Комбинация L_1 и L_2 регуляризации носит название "elastic net" [1]

   $$R(\boldsymbol{\beta}) = \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2$$

   Выполняет отбор признаков также, как и L_1 регуляризация. В тоже время, elastic net регуляризация имеет тенденцию оставлять группу коррелированных коэффициентов целиком. Elastic net регуляризация не гладкая в точке $\boldsymbol{\beta} = 0$.

4. Групповое лассо (group lasso) [2, 3]

   $$R(\boldsymbol{\beta}) = \sum_{g \in \mathcal{G}} \lambda_g \|\boldsymbol{\beta}_g\|_2$$

   где $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^T, \ \ldots \ , \boldsymbol{\beta}_m^T)^T$, используется в случае если необходимо гарантировать отбор признаков группами, например при 1-hot кодировании категориальных переменных,

или если нужно гарантировать инвариантность относительно ортогональных преобразований подпространств групп. Данная регуляризация также не гладкая в точке $\boldsymbol{\beta} = 0$.

5. Smoothly Clipped Absolute Deviation (SCAD) [4] для наглядности обычно определяется через производную

$$R(\boldsymbol{\beta}) = \sum_{k=1}^{p} r_{SCAD}(|\beta_k|, \lambda_1, a)$$

$$r'_{SCAD}(z) = \lambda_1 \left\{ I(z \leq \lambda_1) + \frac{(a\lambda_1 - z)_+}{(a-1)\lambda_1} I(z > \lambda_1) \right\}, \quad z \geq 0, a > 2, \lambda_1 > 0$$

В отличие от L_1 регуляризации, SCAD дает несмещенные оценки для больших по модулю коэффициентов β_k. Данный регуляризатор не выпуклый и не гладкий в точке $\boldsymbol{\beta} = 0$.

6. Minimax Convex Penalty (MCP) [5] также определяется через производную

$$R(\boldsymbol{\beta}) = \sum_{k=1}^{p} r_{MCP}(|\beta_k|, \lambda_1, \gamma)$$

$$r'_{MCP}(z) = \lambda_1 \left(1 - \frac{|z|}{\gamma\lambda_1}\right)_+ \operatorname{sgn}(z), \quad \gamma > 0, \lambda_1 > 0$$

и обладает аналогичными SCAD свойствами.

7. Sparse Laplacian Shrinhage (SLS) [31]

$$R(\boldsymbol{\beta}) = \sum_{j=1}^{p} r_{MCP}(|\beta_k|, \lambda_1, \gamma) + \frac{\lambda_2}{2} \sum_{1 \leq j < k \leq p} |a_{jk}|(\beta_j - s_{jk}\beta_k)^2$$

$$s_{jk} = \operatorname{sgn}(a_{jk})$$

Выполняет регуляризацию и отбор признаков с учетом попарных корреляций между переменными. Матрица (a_{jk}) характеризует похожесть переменных β_j и β_k. Ее можно положить равной, например, коэффициенту линейной корреляции $a_{jk} = (\mathbf{x}_j^T \mathbf{x}_k)/(\|\mathbf{x}_i\|\|\mathbf{x}_k\|)$.

Регуляризаторы L_1, L_2, elastic net, group lasso являются выпуклыми. Поэтому задача (2) является задачей выпуклой оптимизации. Если регуляризатор гладкий (L_2), то решение задачи упрощается, этот случай описан в разделе 1.4. Случай негладкого регуляризатора более сложный с точки зрения оптимизации, подходы к решению описаны в разделе 1.3. Для невыпуклых регуляризаторов SCAD, MCP, SLS можно применять методы, разработанные для L_1 регуляризации, правда уже без гарантий сходимости.

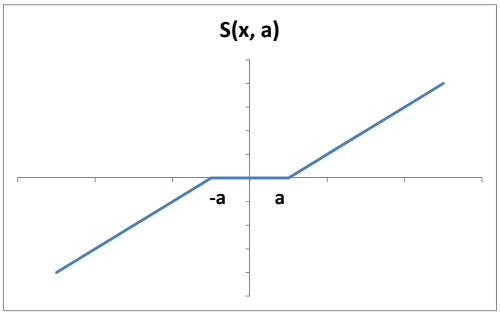


Рис. 2: Функция soft thresholding

1.3 Методы для L_1 регуляризации

1.3.1 Алгоритм Shooting

Давайте сначала опишем простой алгоритм Shooting [32, 33], использующийся для решения задачи LASSO. Исторически Shooting был одним из первых алгоритм покоординатного спуска для задач с L_1 регуляризацией, кроме того он применяется как составная часть более сложных алгоритмов, которые будут использоваться в данной работе. Задача LASSO состоит в минимизации функции

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left(\frac{1}{2} \sum_{i=1}^{n} (y_i - \boldsymbol{\beta}^T \mathbf{x}_i)^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 \right).$$

Алгоритм 1: Shooting

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n$, $\lambda_1 \geq 0$

$\boldsymbol{\beta} \leftarrow 0$

Пока *не выполнено условие останова*

  Цикл $j = 1 \dots p$

    минимизировать $\frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{\beta}^T \mathbf{x}_i - y_i)^2 + \lambda_1 \|\boldsymbol{\beta}\|_1$ по переменной β_k:

      $\beta_k \leftarrow S\left(\frac{\sum_{i=1}^{n} x_{ik}(y_i - (s_i - \beta_k x_{ik}))}{\sum_{i=1}^{n} x_{ik}^2}, \frac{\lambda_1}{\sum_{i=1}^{n} x_{ik}^2} \right)$, где $s_i = \boldsymbol{\beta}^T \mathbf{x}_i$

Возврат: $\boldsymbol{\beta}$

В алгоритме Shooting используется функция $S(\cdot)$ (soft thresholding function)

$$S(x, a) = \operatorname{sgn}(x) \max(|x| - a, 0). \tag{3}$$

График данной функции изображен на рис. 2. Несмотря на простоту, алгоритм Shooting эффективно решает задачу LASSO для выборок с высокой размерностью p. Если объекты

обучающей выборки имеют веса w_i, то функция эмпирического риска будет равна

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left(\frac{1}{2} \sum_{i=1}^{n} w_i (y_i - \boldsymbol{\beta}^T \mathbf{x}_i)^2 + \lambda \|\boldsymbol{\beta}\|_1 \right).$$

В этом случае шаг (3) алгоритма Shooting приобретает вид:

$$\beta_k \leftarrow S \left(\frac{\sum_{i=1}^{n} w_i x_{ik} (y_i - (s_i - \beta_k x_{ik}))}{\sum_{i=1}^{n} w_i x_{ik}^2}, \frac{\lambda}{\sum_{i=1}^{n} w_i x_{ik}^2} \right). \tag{4}$$

Для эффективной реализации алгоритма необходимо хранить вектор скалярных произведений $\mathbf{s} = (\mathbf{x}_i \boldsymbol{\beta}^T)_{i=1}^n$. Количество операций алгоритма Shooting при обновлении переменной β_j пропорционально количеству примеров с ненулевым x_{ij}. Поэтому общая сложность одной итерации - $O(nnz)$. Следовательно, алгоритм Shooting особенно эффективен на разреженных выборках, когда $nnz \ll np$, а большие обучающий выборки, как правило, являются разреженными.

1.3.2 Метод GLMNET

Метод GLMNET [13] предназначен для обучения обобщенных линейных моделей с L_1 и L_2 регуляризацией. Данная задача имеет вид

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \ell(y_i, \boldsymbol{\beta}^T x_i) + \lambda \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2 \right\}. \tag{5}$$

На каждой итерации GLMNET строит квадратичное приближение к функции риска

$$\begin{aligned}
&\sum_{i=1}^{n} \ell(y_i, (\boldsymbol{\beta} + \Delta\boldsymbol{\beta})^T x_i) \approx \\
&\approx \sum_{i=1}^{n} \left\{ \ell(y_i, \boldsymbol{\beta}^T x_i) + \frac{\partial \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}} \Delta\boldsymbol{\beta}^T \mathbf{x}_i + \frac{1}{2} (\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2} (\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \right\} = \\
&= C(\boldsymbol{\beta}) + \frac{1}{2} \sum_{i=1}^{n} w_i (z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2,
\end{aligned}$$

где

$$\begin{aligned}
w_i &= \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2}, \\
z_i &= -\frac{\partial \ell(y_i, \boldsymbol{\beta}^T x_i) / \partial \hat{y}}{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i) / \partial \hat{y}^2}, \\
C(\boldsymbol{\beta}) &= L(\boldsymbol{\beta}) - \frac{1}{2} \sum_{i=1}^{n} z_i^2 w_i.
\end{aligned}$$

Для случая логистической регрессии

$$z_i = \frac{(y_i + 1)/2 - p(\mathbf{x}_i)}{p(\mathbf{x}_i)(1 - p(\mathbf{x}_i))},$$
$$w_i = p(\mathbf{x}_i)(1 - p(\mathbf{x}_i)),$$
$$p(\mathbf{x}_i) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T \mathbf{x}_i}}.$$

В дальнейшем мы будем использовать следующее обозначение для квадратичного приближения к функции риска

$$L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{i=1}^{n} w_i (z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2.$$

Основная идея метода GLMNET - итеративная минимизация квадратичного приближения к функции риска плюс регуляризация

$$\underset{\Delta\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{n} w_i (z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2 + \lambda_1 \|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|_2^2 \right\} \tag{6}$$

с помощью покоординатного спуска. Минимизация квадратичного приближения (6) по переменной $\Delta\beta_j$ аналогично задаче LASSO с весами (4) (отличается только наличием дополнительной L_2 регуляризации) и имеет простой аналитический вид

$$\Delta\beta_j^* = \frac{S\left(\sum_{i=1}^{n} w_i x_{ij} q_i, \lambda_1\right)}{\sum_{i=1}^{n} w_i x_{ij}^2 + \lambda_2} - \beta_j, \tag{7}$$
$$q_i = z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i + (\beta_j + \Delta\beta_j) x_{ij}.$$

Таким образом, метод GLMNET позволяет минимизировать локальные квадратичные приближения к целевой функции без хранения Гессиана в явном виде. Это особенно важно для задач большой размерности.

Формальное описание GLMNET приведено в Алгоритме 2. Данный алгоритм имеет структуру из трех вложенных циклов. На самом глубоком цикле выполняются шаги по отдельным координатам β_j для минимизации квадратичного приближения (6). На промежуточном цикле проверяется сходимость минимизации квадратичного приближения (6), т.е. определяется сколько циклов по всем координатам нужно сделать. Наконец, в самом верхнем цикле производятся шаги алгоритма $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$ и проверяется сходимость исходной задачи (5).

Алгоритм GLMNET может быть значительно ускорен если выполнять шаги только по подмножеству *активных переменных*. В промежуточном цикле только первый раз выполняется проход по всем переменным $\beta_j,\ j = 1\ \ldots\ p$. Последующие проходы выполняются только по подмножеству переменных с ненулевыми β_j. Итерации по подмножеству активных переменных также используются в работах [3, 34].

Алгоритм 2: Алгоритм GLMNET

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n, \lambda_1 \geq 0, \lambda_2 \geq 0$

$\boldsymbol{\beta} \leftarrow 0$

Пока *не выполнено условие останова 1*

  Пока *не выполнено условие останова 2*

    Цикл $j = 1 \ \ldots \ p$

      минимизировать $\frac{1}{2}\sum_{i=1}^n w_i(z_i - \Delta\boldsymbol{\beta}^T\mathbf{x}_i)^2 + \lambda_1\|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|_2^2$

      по $\Delta\beta_j$:

        $\Delta\beta_j = \frac{S\left(\sum_{i=1}^n w_i x_{ij} q_i, \lambda_1\right)}{\sum_{i=1}^n w_i x_{ij}^2} - \beta_j$, где $s_i = \boldsymbol{\beta}^T\mathbf{x}_i$

  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$

Возврат: $\boldsymbol{\beta}$

Алгоритм 3: Вычисление пути регуляризации

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n, K$

Вычислить λ_{max} по формуле (8)

$\boldsymbol{\beta} \leftarrow 0$

Цикл $i = 1 \ \ldots \ K$

  Решить (5) с параметром $\lambda = \lambda_{max} * 2^{-i}$, используя предыдущее $\boldsymbol{\beta}$ как начальное приближение

Возврат: список $\{\lambda, \boldsymbol{\beta}\}$

1.3.3 Путь регуляризации

На практике часто оптимальные параметры регуляризации не известны заранее. В этом случае, можно эффективно решить задачу сразу для множества параметров λ_1, λ_2. Обычно полагают [13]

$$\lambda_1 = \alpha\lambda, \quad \lambda_2 = (1 - \alpha)\lambda$$

для некоторого фиксированного $0 < \alpha \leq 1$ и производится перебор λ по убыванию. Данная процедура называется поиском пути регуляризации (regularization path) и она приведена в Алгоритме 3.

Параметр λ_{max} - это минимальное значение λ, при котором решением (5) будет $\boldsymbol{\beta} = 0$ и находится из условия равенства минимального субградиента $L(\boldsymbol{\beta})$ нулю

$$\lambda_{max} = \frac{1}{\alpha}\max_j \left| \sum_{i=1}^n \frac{\partial \ell(y_i, \boldsymbol{\beta}^T\mathbf{x}_i)}{\partial \hat{y}} x_{ij} \right|. \tag{8}$$

Рис. 3: Путь регуляризации

Пример пути регуляризации приведен на рис. 3. Каждая точка на этом графике - это классификатор, обученный для некоторого параметра λ. По оси OX отложено количество ненулевых компонент $\boldsymbol{\beta}$, по оси OY - качество классификатора на тестовой выборке.

1.3.4 Метод newGLMNET

Несмотря на хорошие результаты полученные на многих выборках, были обнаружены случаи, когда метод GLMNET не сходится. Поэтому Yuan et al. [14] предложили его улучшение - newGLMNET. Полностью приводиться формальное описание newGLMNET не будет, т.к. он сложен и содержит большое число эвристик, ускоряющих сходимость. Метод newGLMNET в целом повторяет GLMNET, за исключением следующих основных отличий:

1. Двухуровневый алгоритм для определения подмножества "активных переменных". Вычисления $\Delta\beta_j$ делаются только для активных переменных.

2. Адаптивный критерий остановки при оптимизации квадратичного приближения (6) На первых итерациях делается грубая оптимизация, на последующих - более точная

3. Шаг $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$, заменяется на

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta},$$

в котором α - это максимальный элемент из последовательности $\{b^j\}_{j=0,1,\ldots}$, удовлетворяющий критерию Армихо

$$f(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}) \leq f(\boldsymbol{\beta}) + \alpha\sigma D,$$

где $0 < b < 1, 0 < \sigma < 1, 0 \leq \gamma < 1$, и

$$D = \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \gamma\Delta\boldsymbol{\beta}^T H \Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta}).$$

Из перечисленных условий только 3-е является критическим для гарантии сходимости метода newGLMNET. В этом случае newGLMNET становится частным случаем семейства методом покоординатного спуска Coordinate Gradient Descent (CGD) [35], для которых доказана теорема о сходимости. В статьях [14, 11] было проведено подробное сравнение существующих методов для логистической регрессии с L_1-регуляризацией и было установлено, что метод newGLMNET работает лучше всего по большому числу критериев: скорость оптимизации целевой функции, скорость увеличения качества на тестовой выборке, разреженность решения.

1.3.5 Онлайн обучение с усеченным градиентом

Онлайн-обучение часто используется для построения регрессий на больших обучающих выборках. При онлайн обучении не требуется загружать всю выборку в оперативную память, т.к. объекты можно обрабатывать по одному. Преимущество методов онлайн обучения для больших обучающих выборок, например метода стохастического градиента, может быть обосновано через approximation-estimation-optimization tradeoff [36].

Для онлайн обучения задача обычно ставится в виде

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \frac{\lambda_1}{n} \|\boldsymbol{\beta}\|_1 \right) \right\},$$

т.е. регуляризатор добавляется отдельно к каждому обучающему примеру. Метод стохастического градиента напрямую не применим для задач с L_1 регуляризацией, т.к. данный регуляризатор не дифференцируем в нуле. Кроме того, шаги стохастического градиента в этом случае

$$\beta_j \leftarrow \beta_j - \eta \left(\frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \beta_j} + \frac{\lambda_1}{n} \operatorname{sgn}(\beta_j) \right)$$

не будут приводить к занулению компонент $\boldsymbol{\beta}$. Поэтому для обобщенных линейных моделей с L_1 регуляризацией в [7] был предложен метод "онлайн обучения с усеченным градиентом" (online learning via truncated gradient). Численные эксперименты показывают, что данный метод обеспечивает хорошее качество классификации и достаточную разреженность решения за существенно меньшее время, чем пакетные алгоритмы. Метод онлайн обучения с усеченным градиентом приведен в Алгоритме 4 и реализован в программе Vowpal Wabbit. На протяжении обучения темп η меняется по сложному правилу, содержащему много эвристик, улучшающих сходимость [37, 38, 39]. Также программа Vowpal Wabbit позволяет задать начальный темп η_0 и степень скорости убывания темпа p. В Алгоритме 4 используется функция soft-thresholding $S(\cdot)$ (3).

Алгоритм 4: Онлайн обучение с усеченным градиентом

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n$, $\lambda_1 \geq 0$, N_{passes}

1 $\boldsymbol{\beta} \leftarrow 0$

2 **Цикл** $k = 1 \ \ldots \ N_{passes}$

3 **Цикл** $i = 1 \ \ldots \ n$

4 **Цикл** $j = 1 \ \ldots \ p$

5 $\beta_j \leftarrow \beta_j - \eta(\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)/\partial \beta_j)$

6 $\beta_j \leftarrow S(\beta_j, \eta\frac{\lambda_1}{n})$

Возврат: $\boldsymbol{\beta}$

Реализация Алгоритма 4 напрямую потребует $O(np)$ операций на каждый проход по выборке (самый верхний цикл). Это является нежелательным при работе с большими обучающими выборками, когда одновременно n и p могут быть большими. Поэтому применяется следующая оптимизация. Заметим, что

$$\frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \beta_j} = \frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \hat{y}} x_{ij},$$

Таким образом, если $x_{ij} = 0$, то нужно сделать только шаг 6. Для эффективной реализации для каждой компоненты β_j запоминается номер итерации τ_{i_1}, когда $x_{i_1 j} \neq 0$ в последний раз. И в ближайшей итерации τ_{i_2}, в которой $x_{i_2 j} \neq 0$, шаг 6 выполняется $\tau_{i_2} - \tau_{i_1}$ раз

$$\beta_j \leftarrow S(\beta_j, (\tau_{i_2} - \tau_{i_1})\eta\lambda_1).$$

Таким образом, количество необходимых операций на один проход по выборке составляет $O(nnz)$.

Стоит отметить, что сходимость метода и эффективная реализация в работе [7] предложены только для постоянного темпа обучения. Эффективная реализация для случая убывающего темпа обучения предложена в [40].

1.4 Методы для L_2 регуляризации

Обучение GLM с L_2 регуляризацией сводится к решению следующей задачи оптимизации

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|^2 \right\}. \tag{9}$$

1.4.1 Онлайн обучение

Задача 9 может быть решена с помощью широкого класса методов онлайн обучения, т.к. для всех рассматриваемых в данной работе GLM целевая функция выпуклая и гладкая. В данном разделе описан алгоритм онлайн обучения из программы Vowpal Wabbit, см. Алгоритм 5. На протяжении обучения темп η меняется по сложному правилу, содержащему много эвристик, улучшающих сходимость [37, 38, 39].

Алгоритм 5: Онлайн обучение для GLM с L_2-регуляризацией

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n$, λ_2 $ge0$, N_{passes}

$\boldsymbol{\beta} \leftarrow 0, \mathbf{w} \leftarrow 0$

Цикл $k = 1 \ \ldots \ N_{passes}$

  Цикл $i = 1 \ \ldots \ n$

    Цикл $j = 1 \ \ldots \ p$

      $\beta_j \leftarrow \beta_j - \eta(\frac{\partial L(\boldsymbol{\beta}^T \mathbf{x}_i, y_i)}{\partial \beta_j} - \frac{\lambda_2}{n} \beta_j)$

Возврат: $\boldsymbol{\beta}$

Сложность одного прохода по выборке составляет $O(nnz)$.

1.4.2 BFGS

Метод BFGS [41, 42] предназначен для решения задачи оптимизации

$$x^* = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} f(x),$$

где $f(\cdot)$ - гладкая выпуклая функция.

Метод BFGS является одним из самых эффективных квази-ньютоновских методов для решения задач оптимизации задачах большой размерности. В контексте машинного обучения он успешно используется при обучении графических моделей и GLM с L_2 регуляризацией [43]. В задачах большой размерности гессиан невозможно хранить в явном

виде. Метод BFGS требует только вычислений значений функции и ее производной, по мере итераций строя низкоранговые приближения к обратному гессиану. Приближение в обратному гессиану обновляется на каждой итерации по формуле

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \tag{10}$$

где

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f(x_{k+1}) - \nabla f(x_k),$$
$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T.$$

Начальное приближение H_0, как правило, выбирается диагональным

$$H_0 = diag\left[\left(\frac{\partial^2 f(x_0)}{\partial x_1^2}\right)^{-1}, \ \dots \ , \left(\frac{\partial^2 f(x_0)}{\partial x_p^2}\right)^{-1}\right].$$

В Алгоритме 6 дано формальное описание метода BFGS.

Алгоритм 6: Метод BFGS

Вход : начальная точка x_0, начальное приближение Гессиана H_0,

$0 < c_1 < c_2 < 1$

1 **Пока** *не выполнено условие останова*

2 Вычислить направление поиска:

3 $p_k = -H_k \nabla f(x_k)$

4 $x_{k+1} = x_k + \alpha_k p_k$, где α_k выбирается таким, чтобы удовлетворить условиям Вольфе:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T(x_k) p_k$$
$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T(x_k) p_k$$

Вычислить H_{k+1} по формуле (10)

5 $k \leftarrow k + 1$

Возврат: x_k

Для выполнения итераций BFGS необходимо хранить последовательноти векторов $\{s_1, \dots, s_k\}$ и $\{y_1, \dots, y_k\}$. Обновление приближения к гессиану (10) и вычисления направления шага $p_k = -H_k \nabla f(x_k)$ требуют только вычисления скалярных произведений векторов размера p. Именно поэтому метод BFGS хорошо подходит для задач большой размерности. Можно показать [42], что если функция $f(\cdot)$ обладает определенными свойствами, то метод BFGS глобально сходится со сверхлинейной скоростью.

1.4.3 Limited Memory BFGS (L-BFGS)

Несмотря на то, что метод BFGS хорошо подходит для решения задач оптимизации высокой размерности, после большого числа итераций для хранений гессиана H_k будет необходимо слишком много места. Поэтому Liu and Nocedal [44] разработали метод Limited Memory BFGS (L-BFGS). Метод L-BFGS хранит матрицы H_k ранга не больше m.

Для вычисления обновлений матрицы H_k в методе L-BFGS достаточно хранить только m последних значений векторов s и y:

$$\{s_{k-1}, \dots, s_{k-m}\}, \quad \{y_{k-1}, \dots, y_{k-m}\}.$$

В этом случае приближение к гессиану будет вычисляться по формуле

$$\begin{aligned} H_k =& (V_{k-1}^T \cdots V_{k-m}^T) H_k^0 (V_{k-m} \cdots V_{k-1}) \\ &+ \rho_{k-m} (V_{k-1}^T \cdots V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_{k-1}) \\ &+ \rho_{k-m+1} (V_{k-1}^T \cdots V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_{k-1}) \\ &+ \cdots \\ &+ \rho_{k-1} s_{k-1} s_{k-1}^T. \end{aligned}$$

В остальных аспектах метод L-BFGS совпадает с BFGS (см. Алгоритм 6). Недостатком L-BFGS является более медленная, чем у BFGS сходимость. Также он плохо работает для задач, в которых собственные значения гессиана сильно отличаются по величине. Чем больше хранится последних векторов s, y (параметр m) тем меньше итераций алгоритма нужно для достижения определенной точности сходимости. В тоже время, с ростом m растет объем вычислений на одной итерации. Обычно выбирается m равное нескольким десяткам. Сложность одной итерации L-BFGS равна $O(mp)$, не считая времени вычисления целевой функции и ее градиента. Для хранения вспомогательных векторов s_k, y_k также нужно $O(mp)$ оперативной памяти. Эффективная реализация L-BFGS для модели вычислений Map/Reduce предложена в работе [45].

1.4.4 Комбинирование онлайн обучения и L-BFGS

Онлайн обучение и метод L-BFGS имеют свои достоинства и недостатки. Онлайн обучение позволяет грубо найти решение за несколько проходов по обучающей выборке. Для некоторых практических задач этого бывает достаточно, особенно, если выполнить большее число проходов просто не хватает времени. В тоже время, методы онлайн обучения

обладают медленной локальной сходимостью и не позволяют найти решение с большой точностью.

Метод L-BFGS - это квазиньютоновский метод. Поэтому он, с одной стороны, медленно сходится в начале, но быстро сходится вблизи оптимума. Поэтому комбинирование онлайн обучения и L-BFGS сочетает сильные стороны обоих методов и хорошо работает на больших обучающих выборках на практике. Эта идея была предложена в работе [18], см. Алгоритм 7.

Алгоритм 7: Комбинирование онлайн-обучения и L-BFGS

Вход : Обучающая выборка, параметр k

1 Выполнить k проходов онлайн обучения по выборке, получить решение $\boldsymbol{\beta}_{online}$

2 Использовать $\boldsymbol{\beta}_{online}$ как начальное приближение в методе L-BFGS

3 **Пока** *не выполнено условие останова*

4 | Выполнить итерацию L-BFGS

5 | $\boldsymbol{\beta}$ - текущее решение

Возврат: $\boldsymbol{\beta}$

2 Параллельные методы для минимизации функций эмпирического риска GLM с регуляризацией

2.1 Модели вычислений и программные архитектуры для распределенного машинного обучения

Важным направлением исследований является разработка методов машинного обучения, специально предназначенных для больших выборок, а также разработка распределенных вычислительных систем, позволяющих применять существующие методы на больших выборках. От таких систем требуется:

1. **Масштабируемость** по количеству обучающих примеров и параметрам модели.

2. **Отказоустойчивость**. При распределенном выполнении на кластере вероятность отказа хотя бы одного узла высока. Распределенная система должна быть устойчива к таким отказам.

3. **Производительность**. Важной проблемой распределенных вычислений является "проблема отстающего", (slow node problem). Суть ее состоит в том, что при необходимости синхронизации между узлами, один медленный узел будет тормозить вычисления в целом. Аналогично отказам - при работе на большом кластере хотя бы один узел с большой вероятностью будет медленным. Распределенная система должна эффективно решать "проблему отстающего". Передача данных по сети не должна быть узким местом вычислительной системы.

4. **Универсальность**. Машинное обучение обычно является частью процесса анализа данных, включающего предобработку данных, вычисление признаков, подбор гиперпараметров обучения, анализ результатов использования модели в прикладной задаче и т.д. Поэтому возможность решения всех задач в одной системе также является весомым аргументом.

Методы машинного обучения, которые чаще всего применяются к большим выборкам из описанных ранее прикладных задач и требуют распределенного обучения: обобщенные линейные модели [18, 28, 46], LDA [47, 48, 46], коллаборативная фильтрация [48, 49], бустинг деревьев решений [50, 51, 52], word2vec [53], глубокие нейросети [54, 55, 56]. В данном разделе рассмотрены следующие подходы к разработке программ для распределенного

машинного обучения: модель вычислений Map/Reduce, передача сообщений по стандарту MPI, архитектура "сервера параметров", система Spark, модели вычислений на графах. Каждый из данных подходов имеет свои достоинства и недостатки, в чем-то ограничивает разработчика и предполагает определенный стиль разработки программ.

2.1.1 Map/Reduce

Модель вычислений Map/Reduce была предложена в статье [57]. В рамках этой модели каждая задача принимает на вход некоторый список пар (key, value), применяет к ним операции map и reduce и возвращает список пар (key, value). Данные обычно хранятся в файлах распределенной файловой системы. Пользователю необходимо определить базовые операции map и reduce:

```
map:    (key1, value1) → list(key2, value2)
reduce: (key2, list(value2)) → list(key3, value3)
```

Операция map принимает на вход одну пару $(key1, value1)$ и возвращает список пар $(key2, value2)$. Операция *reduce* принимает на вход все пары с одним ключом $key2$ и возвращает список пар $(key3, value3)$. Между операциями map и reduce происходит группировка пар с одинаковым значением $key2$. На разных блоках данных вычисления проводятся параллельно. При запуске операций map и reduce программист не контролирует, на каком узле кластера и в какой последовательности на блоках данных будет выполняться программа. В тоже время, планировщик старается выполнять операции локально, т.е. минимизировать передачу данных по сети. Планировщик стремится равномерно распределить нагрузку на узлы: при выполнении операции map - по объему блоков, во время операции reduce - данные для обработки распределяются равномерно по хешу от ключа. При отказе одного узла кластера, выполняющиеся на нем вычисления перезапускаются на других узлах (файлы в распределенной файловой системе реплицируются), что обеспечивает отказоустойчивость. Модель Map/Reduce является индустриальным стандартом для распределенных вычислений и широко используется в интернет-компаниях. Существует несколько проприетарных реализаций модели Map/Reduce: Google MapReduce, Yandex Tables (YT) [58], Disco [59] и т.д. На практике широко используется свободная реализация Apache Hadoop [60]. Кластеры Map/Reduce обычно включают от нескольких десятков до нескольких тысяч узлов. Для решения "проблемы отстающего", в реализациях модели Map/Reduce используется техника спекулятивного выполнения [57]. Она состоит в следующем: когда операции *map* или *reduce* завершились на большей части блоков данных,

то копии задач на незавершившихся блоках запускаются на других узлах. Несмотря на дополнительные вычисления, такая техника часто ускоряет выполнение операций map и reduce на полном объеме данных. Техника спекулятивного выполнения подразумевает, что задачи map и reduce на разных блоках не обмениваются данными между собой в процессе исполнения. С теоретической точки зрения, с помощью модели Map/Reduce можно реализовать алгоритмы, подходящие под “Statistical Query Model” [61]. Это означает, что в алгоритме используются только функции, вычисляемые в виде суммы по обучающей выборке. Например, если алгоритм требует только вычисления аддитивной по объектам функции потерь и ее градиента, то он удовлетворяет данному условию. Для большого числа популярных методов существуют алгоритмы обучения, подходящие под “Statistical Query Model”: линейная и логистическая регрессии, наивный Байес, GDA, PCA, ICA, линейный SVM, обучение нейросети градиентным спуском, EM-алгоритм для смеси гауссиан и т.д. [61]. Несмотря на универсальность модели, отмечается [62, 18], что она хорошо приспособлена для вычисления признаков из больших выборок, но плохо подходит для итеративных алгоритмов машинного обучения. На это есть три основные причины:

1. Каждая итерация алгоритма предполагает запуск новой задачи Map/Reduce, что происходит с задержкой из-за планировщика задач, который ожидает выделения ресурсов.

2. На каждой итерации заново читаются входные данные с диска и пересылаются по сети.

3. Нет встроенных способов хранения состояния между итерациями.

Кроме того, в модели Map/Reduce вычисления на разных блоках данные выполняются параллельно, что не позволяет учитывать структуру данных. Поэтому при большом числе итераций такая реализация будет неэффективной. Вариант решения проблемы 1 предложен в [63] - это “Forward Scheduling”. Идея состоит в том, что мастер-программа, запускающая задачи Map/Reduce, запускает их заранее. После выделения ресурсов и начала фактической работы задач, мастер-программа сообщает им по RPC (в обход планировщика задач) какую конкретно операцию они должны выполнить. Естественно, что такой подход применим, если большое число задач выполняются на одних тех же входных данных. Для решения проблемы 2 были предложены расширения модели Map/Reduce [8, 22], поддерживающие кеширование данных между итерациями. Впрочем, они не получили широкого распространения из-за недостаточной отказоустойчивости. 3-я проблема реша-

ется обычно хранением состояния в HBase или кеш-файлах, которые передаются задачам Map/Reduce перед запуском. Большое число реализаций алгоритмов с помощью модели Map/Reduce описаны в [64, 61, 65]. Несмотря на отмеченные недостатки, MapReduce используется в системах PLANET [63] и H_2O [66] для обучения Gradient Boosted Decision Trees (GBDT). Впрочем, как отмечается в обзоре [67], реализации на MPI и RPC более вычислительно эффективны. В статье [45] предложена эффективная реализация метода оптимизации L-BFGS на базе Map/Reduce.

К **плюсам** реализации методов машинного обучения на Map/Reduce является чрезвычайная распространенность кластеров Map/Reduce, особенно Apache Hadoop. Также программы Map/Reduce обладают хорошей отказоустойчивостью даже при работе на перегруженном задачами (что является нормой) промышленном кластере. Универсальность модели позволяет реализовать широкий класс алгоритмов и проводить весь цикл анализа данных на Map/Reduce. Программы, написанные на Map/Reduce обладают хорошей масштабируемостью. **Минусом** же является, как уже было сказано выше, плохая производительность при выполнении итеративных алгоритмов, в частном случае - алгоритмов машинного обучения.

2.1.2 MPI

Message Passing Interface (MPI) [68] - это стандарт передачи сообщений между процессами в параллельном программировании, широко использующийся при высокопроизводительных вычислениях. Существуют реализации MPI для многих популярных операционных систем и языков программирования. MPI поддерживает двухсторонние (point-to-point) и коллективные (collective) обмены данными. Существуют реализации стандарта как для систем с общей памятью - для обмена данными между процессами на одном компьютере, так и для распределенных систем [69, 70, 18]. MPI определяет только стандарты для передачи данных, не гарантируя отказоустойчивой работы программы в распределенной среде, в отличие от реализаций Map/Reduce. Наиболее часто используемая функция из стандарта MPI в распределенном машинном обучении - это AllReduce, которая эквивалентна последовательному выполнению функций Reduce (из стандарта MPI) и Broadcast. Сигнатура функции AllReduce приведена ниже

```
int MPI_AllReduce(
   void* input_data_p      /* in */,
   void* output_data_p     /* out */,
```

```
int count               /* in */,
MPI_Datatype datatype   /* in */,
MPI_Op operator         /* in */,
MPI_Comm comm           /* in */)
```

Операция AllReduce применяет ассоциативную коммутативную операцию operator покомпонентно к массивам `input_data_p` одинаковой длины count, хранящимся на нескольких узлах. Примеры таких операций: сумма, произведение, максимум, минимум и т.д. Результат операции передается на все машины в массив `output_data_p`. Аналогично функции AllReduce для систем с общей памятью [71], распределенный вариант передает сообщения по структуре остовного бинарного дерева.

Плюсом программ, использующих MPI является более высокая производительность по сравнению с реализациями на Map/Reduce, так как отсутствуют накладные расходы по запуску задачи в итеративных алгоритмах. Программы, использующие MPI обычно пишутся по технике SPMD (Single Program Multiple Data) и получаются небольшим изменением кода последовательной программы, что также является плюсом. **Минусом** же является более низкая отказоустойчивость - при отказе одного из узлов не могут продолжать вычисления остальные. Также программы, написанные с использованием MPI, страдают из-за "проблемы отстающего". Операция AllReduce является моментом синхронизации и не может быть выполнена, пока все машины не завершат предыдущие вычисления. Поэтому для эффективной работы нагрузка на узлы кластера должна быть сбалансирована, что является ответственностью программиста. Примеры успешных проектов, использующих функции стандарта MPI в распределенных системах: Vowpal Wabbit [18], LibLinear [16], d-GLMNET [28], XGBoost [50], LambdaMART [51], pGBRT [52], COTS HPC [55].

2.1.3 Сервера параметров

Сервер параметров - это система для хранения и модификации параметров модели машинного обучения или оптимизации, предназначенная для использования в распределенной среде. При выполнении программ, использующих сервер параметров, все узлы кластера, не считая технических (например, планировщика задач), делятся на две группы: узлы сервера параметров и узлы обработки данных. Сервер параметров состоит одного или более узла кластера и поддерживает две базовые операции - чтения параметров (pull) и записи изменения параметров (push). Узлы обработки данных, в свою очередь, выполняют итерации алгоритма обучения, периодически читая синхронизированный мас-

сив параметров из сервера параметров и отправляя изменения на запись. Такого рода системы обычно обладают следующими свойствами:

1. Параметры хранятся в key-value хранилище.
2. Отказоустойчивость сервера параметров. Данные реплицируются между узлами.
3. Отказоустойчивость обработки данных. При отказе узла обработки данных, вычисления по этому набору данных передаются другому узлу.
4. Поддержка различных типов консистентности обновлений, приходящих от узлов обработки:
   - **Bulk Synchronous Parallel (BSP)**. Все узлы обработки данных должны завершить итерацию, переслать обновления на сервер параметров. Все обновления агрегируются (чаще всего – суммируются) и только после этого начинается следующая итерация.
   - **Asynchronous**. При полностью асинхронной схеме, узлы обработки пересылают обновления в произвольном порядке. Эти обновления выполняются сервером параметров также в произвольном порядке. Соответственно, результат чтения параметров в разные моменты времени не детерминирован.
   - **Stale Synchronous Parallel (SSP)**. Является компромиссом между полностью синхронным и асинхронным вариантами. Разрешается, чтобы рассинхронизация между самым быстрым и самым медленным узлами обработки данных составляла не более фиксированного числа s итераций. При достижении этого отставания, самый быстрый узел приостанавливается.

Асинхронные и частично асинхронные (SSP) типы консистентности призваны ускорить вычисления и решить “проблему отстающего”. В статье [24] вариант синхронизации Stale Synchronous Parallel (SSP) исследован теоретически. Рассматривается случай распределенной оптимизации суммы выпуклых функций

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \frac{1}{T} \sum_{t=1}^{T} f_t(\mathbf{x}),$$

удовлетворяющих критерию Липшица с константой L. Оптимизация делается с помощью шагов градиентного спуска

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{u}_t,$$

где $\mathbf{u}_t = -\eta_t \nabla f_t(\widetilde{\mathbf{x}}_t)$, $\eta_t = \frac{\sigma}{\sqrt{t}}$, $\sigma = \frac{F}{L\sqrt{2(s+1)P}}$. Тогда имеет место следующая оценка

$$\frac{1}{T}\sum_{t=1}^{T} f_t(\widetilde{\mathbf{x}}_t) \leq f(\mathbf{x}^*) + O\left(\sqrt{\frac{2(s+1)P}{T}}\right).$$

Здесь $\widetilde{\mathbf{x}}_t$ - это неточный вектор $\mathbf{x}_t$, в котором учтены не все шаги $\mathbf{u}_t$ за последние s итераций (узлы успели выполнить разное количество итераций), P - степень параллелизма. Это соответствует консистентности SSP с параметром s. Данные теоретические результаты гарантируют сходимость обучения в распрелеленной среде при использовании градиентных методов и консистентности SSP.

Наиболее развитым на текущий момент является архитектура сервера параметров (проект почему-то безымянный), описанная в работе [46]. Проект обладает большим числом возможностей и интересных технических решений. Поддерживаются варианты синхронизации BSP, Asynchronous, SSP, причем параметр s в варианте SSP может меняться динамически для увеличения производительности. На сервер параметров отсылаются только достаточно большие обновления, которые проходят через заданный пользователем фильтр. Обновления передаются в сжатом виде - только ненулевые компоненты векторов. Сервера параметров хранят пары (key, value) с использованием консистентного хеширования (consistent hashing) [72] и репликации по цепочке (chain replication) [73]. Система обладает высокой отказоустойчивостью, подключение новых узлов сервера параметров и узлов обработки данных может осуществляться “на лету”. Для уменьшения объема передаваемых данных, необходимых для репликации, сервера параметров вначале выполняют агрегацию обновлений от нескольких источников, и только после этого - репликацию. В численных экспериментах показаны примеры успешного распределенного машинного обучения с выборками размером больше 100 TB с использованием более 104 ядер и до 6000 серверов. Тестировались следующие методы: логистическая регрессия с L1 регуляризацией, LDA с использованием стохастического вариационного вывода и коллапсированного сэмплирования Гиббса, а также CountMin sketch. Время простоя серверов обработки было совсем небольшим: например только 1.7. Из остальных популярных систем типа “сервер параметров” можно отметить Y!LDA [47], Petuum [74], Distributed Machine Learning Toolkit (DMLTK) [53]. Y!LDA предназначен для распределенного обучение LDA и графических моделей. Особенностью системы Petuum [74] является планировщик STRADS, выполняющий балансировку нагрузки и выделяющий блоки для параллельного выполнения (например, шаги координатного спуска по слабо коррелированным переменным).

К **плюсам** систем машинного обучения, использующих сервер параметров можно от-

нести отличную производительность и масштабируемость. Как обработка данных, так и хранение параметров модели распределены между узлами кластера. Отказы узлов кластера не являются проблемой. При использования асинхронной и SSP консистентности вычислительные системы не подвержены “проблеме отстающего”. **Недостатком** такого рода систем является большая нагрузка на сеть при асинхронном и SSP типах консистентности. Данная проблема частично решается хранением локального кэша для часто обновляемых переменных. При асинхронном стохастическом градиентном спуске алгоритм перестает быть математически эквивалентным последовательному исполнению на одной машине. Ускорение является сублинейным, а сходимость гарантируется только при выполнении консистентности SSP. Кроме того, сервер параметров - это специализированная архитектура для машинного обучения и оптимизации, в которой нельзя выполнить полный цикл анализа данных.

2.1.4 Spark

Система Spark [40] ориентирована на выполнение итеративных алгоритмов и интерактивный анализ данных с помощью команд функционального программирования. Особенностью этих задач является многократное использование одного рабочего множества данных. Основной концепцией Spark является resilient distributed dataset (RDD). RDD - это доступная только для чтения коллекция объектов, хранящаяся распределенно. В отличие от модели Map/Reduce, можно дать указание хранить ее в оперативной памяти. Один RDD можно получить из другого с помощью преобразований (команды функционального программирования), которые отложено выполняются только при совершении определенных действий. Действия - это операции, требующие возврата данных. Для каждого RDD система Spark запоминает последовательность команд, с помощью которых данный RDD был получен, например, из файла HDFS. При отказе одного узла и потери части RDD, последовательность команд воспроизводится и эта часть вычисляется заново. Таким образом достигается отказоустойчивость при работе с RDD, кэшированными в оперативной памяти. Обмен данными между параллельными процессами, обрабатывающими RDD на разных узлах также может происходить с помощью широковещательных переменных (broadcast variables) и аккумуляторов (accumulators). Широковещательные переменные позволяют передать объект всем процессами на разных узлах. Аккумуляторы же позволяют агрегировать данные с помощью коммутативной ассоциативной операции в процессе перебора всех элементов RDD. Численные эксперименты показывают [40]

значительное ускорение итеративных алгоритмов машинного обучения по сравнению с реализацией Map/Reduce на Hadoop. Spark MLLib [75] - это пакет для машинного обучения, написанный на Scala, в котором используются реализации операций линейной алгеброй на C++. Он поддерживает большое число распространенных методов машинного обучения. Также пакет Spark MLLib содержит функции для предобработки данных и конструирования признаков. Авторы отмечают [75] существенное ускорение решения задачи коллаборативной фильтрации с помощью метода ALS по сравнению с реализацией на Hadoop. Система SparkNet [76] предназначена для обучения глубоких нейросетей с помощью комбинирования Spark и Caffe [77]. После преобразования обучающей выборки в RDD и ее распределения по кластеру, на каждом разбиении (partition) запускается несколько итераций обучения нейросети в Caffe. Далее каждые T итераций параметры модели усредняются с помощью встроенной в Spark операции reduce. Усредненные параметры передаются на узлы с помощью техники широковещательных переменных. Авторы [76] считают, что основное преимущество SparkNet - это совместимость с существующими кластерами Spark, а также низкие требования к пропускной способности сети, так как усреднение происходит лишь после нескольких итераций обучения. SparkNet является частным случаем BSP модели вычислений. Для решения “проблемы отстающего”, предлагается усреднять параметры не каждые T итераций, а после фиксированного интервала времени.

Плюсом реализации методов машинного обучения на Spark является совместимость с широко распространенными кластерами Apache Hadoop. Необходимая предобработка данных и конструирование признаков также могут быть выполнены в Spark, что упрощает работу. Программы, написанные на Spark имеют хорошую отказоустойчивость и масштабируемость. Производительность Spark приложений достаточно хорошая, т.к. система изначально была разработана для итеративных алгоритмов и позволяет хранить данные в оперативной памяти. В тоже время, производительность может страдать из-за "проблемы отстающего так как Spark приложения не являются полностью асинхронными и реализуют модель вычислений BSP. **Минусом** является более низкая по сравнению с реализациями на MPI производительность. Отмечается [78], что реализация обучения логистической регрессии методом TRON быстрее с помощью MPI, чем Spark. В тоже время, реализация на Spark более отказоустойчива. В сравнении [79] отмечается, что реализация ансамблей решающих деревьев в Spark MLLib медленнее и требует больше оперативной памяти, чем H2O и XGBoost, сравнение выполнялось на одной машине.

2.1.5 Системы вычислений на графах

Во многих задачах машинного обучения и аналитики данные имеют структуру разреженного графа, например:

- Тематическое моделирование: граф “документы-слова”.
- Коллаборативная фильтрация: граф “пользователи-объекты”.
- PageRank: граф веб-документов и ссылок между ними.
- Классификация и регрессия: матрицу “примеры-признаки”, можно рассматривать как граф. Если элемент матрицы равен нулю, то соответствующего ребра между примером и признаком не будет.
- Нахождение сообществ в социальных сетях: граф связей людей в социальной сети.

Одной из первых моделей вычислений для графов, успешно работающей распределенно была разработанная Google система Pregel [80]. Существует ее свободная реализация Apache Giraph [81, 49]. Впоследствии была предложена система GraphLab [62], а в [48] была описана ее распределенная реализация. В отличие от Pregel, система GraphLab более гибкая и с самого начала направлена на решение задач машинного обучения и анализа данных. Авторы отмечают три основных свойства GraphLab: асинхронность, динамичность, параллельность на графе. В отличие от Pregel, GraphLab не использует синхронизацию BSP. Более гибкие возможности синхронизации позволяют обойти “проблему отстающего”, которая также актуальна при распределенной обработке графов. Объемы вычислений, связанные с каждой вершиной могут существенно отличаться, т.к. степени вершины обычно распределены по степенному закону. Также в GraphLab существует возможность выполнять вычисления параллельно и асинхронно, сохраняя математическую эквивалентность последовательному выполнению. Динамичность означает, что порядок обработки вершин может изменяться динамически в зависимости от заданного пользователем вычисляемого критерия. В модели GraphLab пользователю необходимо задать: В модели GraphLab пользователю необходимо задать:

1. Ненаправленный граф $G = (V, D, E)$. Здесь D - это данные, связанные с каждой вершиной и ребром. Направление вершины можно опционально хранить в данных, связанных с ребром. Структура графа не изменяется в процессе работы.

2. Функцию *update* : $(v, \mathcal{S}_v) \rightarrow (\mathcal{S}_v, \mathcal{T})$. Функция получает на вход вершину $v \in V$ и данные вершин и ребер в ее окрестности S_v, после чего изменяет эти данные. Также функция *update* возвращает набор вершин $\mathcal{T}$ для последующей обработки.

3. Функция *sync*: выполняет агрегацию, вычисляя произвольную коммутативную ассоциативную операцию по всем данным графа. Функция *sync* может использоваться для вычисления критерия останова.

Функция *update* выполняется на вершинах параллельно и асинхронно, поэтому необходимы блокировки данных. GraphLab поддерживает три типа консистентности. Полная консистентность (full consistency) выполняет блокировку на чтение и запись всего региона. В этом случае существует математически эквивалентная последовательная обработка вершин. Консистентность по ребрам (edge consistency) выполняет блокировку на чтение и запись обрабатываемой вершины и соседних ребер. Консистентность по вершинам (vertex consistency) накладывает только блокировку на чтение и запись обрабатываемой вершины. При распределенном выполнении GraphLab стремится максимально равномерно распределить граф по серверам, так, чтобы число ребер между машинами было минимально. Также каждый сервер хранит информацию о призраках (ghosts) - соседних к части графа вершинах и ребрах. Таким образом кешируются данные соседних вершин и ребер с других серверов. GraphLab поддерживает два алгоритма выполнения и синхронизации задач - раскраска графа (chromatic engine) и распределенные блокировки (distributed lock engine). В алгоритме выполнения через раскраску графа, вершины графа раскрашиваются в несколько цветов. За одну итерацию обрабатываются все вершины одного цвета. Алгоритм выполнения с использованием распределенных блокировок более гибок и поддерживает параллельное и распределенное выполнение со всеми описанными ранее типами консистентности. Для гарантирования отказоустойчивости используются снэпшоты Ченди-Лэмпорта [82]. Из остальных систем вычислений на графах стоит отметить GraphX, реализованную на базе Spark. GraphX сравним по производительности с GraphLab и Apache Giraph для задач обработки графов. В данной системе реализованы популярные методы коллаборативной фильтрации и тематического моделирования. Система GraphX достаточно проста, т.к. нет необходимости в специальных механизмах отказоустойчивости и обмена данными между машинами - это обеспечивает Spark. Заметим, что системы вычислений на графах Pregel, GraphLab, GraphX достаточно сильно отличаются между собой. Поэтому анализ достоинств и недостатков приведен для наиболее популярной из них - GraphLab.

К **плюсам** систем вычисления на графах стоит отнести более высокую, по сравнению с реализациями на Hadoop/MapReduce производительность. В статье [48] показано, что GraphLab в 20-60 раз быстрее Hadoop/MapReduce для задач коллаборативной фильтрации (ALS), видео ко-сегментации и Named Entiny Recognition (NER). В тоже время, скорость выполнения примерно совпадает с реализациями на MPI. К **недостаткам** систем вычислений на графах можно отнести их достаточно узкую направленность. Не всякую задачу машинного обучения можно представить в виде алгоритма обработки графа. Полный цикл анализа данных в такой системе не выполним. В работе [74] показано, что сервер параметров Petuum более масштабируем за счет меньшего потребления оперативной памяти, чем GraphLab для задач LASSO и LDA. В работе [83] отмечается, что асинхронный режим GraphLab неэффективен из-за чрезмерного использования сети. А работе [84] отмечается, что GraphLab недостаточно масштабируем по сравнению с сервером параметров из-за механизма снэпшотов. Таким образом, нельзя сделать вывод о превосходстве GraphLab с точки зрения производительности и масштабируемости над системами, предназначенными для итеративных вычислений в оперативной памяти (MPI, сервера параметров, Spark).

2.1.6 Обсуждение

В данном разделе были рассмотрены современные распределенные вычислительные системы для машинного обучения. Пожалуй, наиболее масштабируемой и производительной системой является архитектура “сервера параметров”, для которой описаны успешные примеры обучения с 100 TB данных и кластера из 6000 узлов. Сервера параметров были разработаны специально для распределенного машинного обучения и совместимы с большим числом методов. Явным аутсайдером выглядит Map/Reduce, хорошо подходящий для вычисления признаков из “сырых”, данных, но не предназначенный изначально для итеративных алгоритмов. Реализации на Map/Reduce используются во многих статьях как простой эталон для сравнения, который легко превзойти. С другой стороны, плюсом Map/Reduce является чрезвычайная распространенность поддерживающих его кластеров Hadoop. Остальные системы: MPI, Spark, вычисления на графах - занимают промежуточное положение и обладают своими достоинствами и недостатками. Практически все системы используют асинхронные вычисления для решения “проблемы отстающего”. Однако такие асинхронные алгоритмы не эквивалентны математически последовательным алгоритмам и могут обладать плохой (и недоказанной) сходимостью. Поэтому некоторая синхронизация все-таки необходима: перспективным выглядит консистентность типа

Stale Synchronous Parallel (SSP). Еще одним минусом асинхронных схем является недетермированность. Недетерминированность затрудняет исследования, т.к. точность модели становится более шумной, зависящей от распределения вычислений между узлами кластера. Интересно, что на базе практически любой описанной системы можно реализовать большой спектр популярных методов машинного обучения: обобщенные линейные модели, LDA, коллаборативная фильтрация, бустинг деревьев решений, word2vec, глубокие нейросети. Поэтому при выборе системы зачастую решающим фактором оказывается ее доступность. При условии, что разработчику доступен только один кластер, весомым плюсом является возможность выполнять весь цикл анализа данных в одной системе. С этой точки зрения наиболее интересной является система Spark, сочетающая в себе производительность, масштабируемость и универсальность. Разработка систем для распределенного машинного обучения является активно развивающимся направлением. Ускорение методов машинного обучения за счет вычислений на кластере позволяет на этапе исследования пробовать больше вариантов моделей и решать практические задачи, в которых возникают большие обучающие выборки.

2.2 Методы для L_1 регуляризации

2.2.1 Распределенное обучение с использованием усеченного градиента

Метод “онлайн обучения с усеченным градиентом” описан в разделе 1.3.5. Несмотря на то, что онлайн обучение - это последовательный алгоритм, он может быть распараллелен с помощью приема описанного в [19]. На разных частях выборки независимо (например, на разных узлах кластера) обучаются регрессии с помощью онлайн обучения, получившиеся вектора весов усредняют, и данный цикл повторяется несколько раз. Реализация в Vowpal Wabbit [18] усредняет компоненты β_j с разными весами, см. Алгоритм 8. Обмен данными между машинами выполняется с помощью MPI_AllReduce.

Веса обновляются после каждого обработанного обучающего примера $(\mathbf{x}_i, y_i)$ по формуле:

$$w_j \leftarrow w_j + \left(\frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \beta_j} \right)^2.$$

Заметим, что

$$\left(\frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \beta_j} \right)^2 = \left(\frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \hat{y}} \right)^2 x_{ij}^2.$$

Таким образом, вес переменной j на машине m будет тем больше, чем больше сумма ненулевых x_{ij}^2 на соответствующей машине.

Алгоритм 8: Распределенное онлайн обучение с усеченным градиентом

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n$, разделенная по примерам по M машинам, N_{passes}

Для всех $m = 1 \ \dots \ M$

  $\boldsymbol{\beta}^m \leftarrow 0, \mathbf{w}^m \leftarrow 0$ (вектора размера p)

Цикл $i = 1 \ \dots \ N_{passes}$

  Выполнить параллельно на M машинах:

    Используя $\boldsymbol{\beta}^m$ как начальное приближение, обучить обобщенную линейную регрессию с помощью Алгоритма 4.

    Получить новое решение $\boldsymbol{\beta}^m$, новый вектор весов $\mathbf{w}^m$

  Цикл $j = 1 \ \dots \ p$

    $\beta_j^m \leftarrow \sum_{m=1}^M w_j^m \beta_j^m / \sum_{m=1}^M w_j^m$

    $w_j^m \leftarrow \sum_{m=1}^M (w_j^m)^2 / \sum_{m=1}^M w_j^m$

Возврат: $\boldsymbol{\beta}$

2.2.2 Alternating Direction Method of Multipliers (ADMM)

Метод Alternating Direction Method of Multipliers (ADMM) [17] иногда применяется для задач распределенной оптимизации, возникающих в машинном обучения. Для этого задача должна быть сформулирована в виде задачи выпуклой оптимизации с ограничениями типа равенств

$$\begin{aligned} \operatorname*{argmin}_{x,z} \left(f(x) + g(z)\right), \qquad (11) \\ \text{при ограничении } Ax + Bz = c, \\ \text{функции } f(\cdot), g(\cdot) - \text{выпуклые}, \\ x \in \mathbb{R}^n, z \in \mathbb{R}^m, A \in \mathbb{R}^{p \times n}, B \in \mathbb{R}^{p \times m}, c \in \mathbb{R}^p. \end{aligned}$$

Для решения задачи (11) строится обобщенный Лагранжиан

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|^2, \qquad (12)$$

где $y \in \mathbb{R}^p$ - множители Лагранжа. Сопряженная к (11) задача имеет вид

$$\begin{aligned} \operatorname*{argmax}_y g(y), \qquad (13) \\ g(y) = \inf_{x,z} L_\rho(x, z, y). \end{aligned}$$

Метод ADMM попеременно оптимизирует обобщенный Лагранжиан (12) и сопряженную функцию (13) выполняя следующие итерации

$$x^{k+1} \leftarrow \operatorname*{argmin}_x L_\rho(x, z^k, y^k),$$
$$z^{k+1} \leftarrow \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k),$$
$$y^{k+1} \leftarrow y^k + \rho(Ax^{k+1} + Bz^{k+1} - c).$$

С помощью метода ADMM можно выполнять распределенную оптимизацию с разделением выборки как "по примерам" , так и "по переменным". В первом случае используется техника consensus, во втором случае - sharing. Для случая большой размерности x лучше подходит техника sharing. В технике sharing искомый вектор x представляется как сумма компонент, каждая из которых отвечает подмножеству координат S_i

$$x = \sum_{i=1}^{N} x_i, \quad j \notin S_i \Rightarrow x_{ij} = 0.$$

Задача оптимизации должна быть представлена в виде

$$\operatorname*{argmin}_{x_i,\, i=1\ldots N} \sum_{i=1}^{N} f_i(x_i) + g(\sum_{i=1}^{N} x_i),$$
$$x_i \in \mathbb{R}^n.$$

Для задач машинного обучения функция $f(\cdot)$ соответствует регуляризации, функция $g(\cdot)$ - эмпирическому риску.

Для обобщенных линейный функция с регуляризатором $r(\cdot)$ в постановке sharing задача для ADMM выглядит так

$$\operatorname*{argmin}_{x,z} \left(\ell(\sum_{i=1}^{N} z_i) + \sum_{i=1}^{N} r_i(x_i) \right), \tag{14}$$
$$\text{при ограничении } A_i x_i - z_i = 0, \quad i = 1 \ \ldots \ N,$$
$$x \in \mathbb{R}^n, z \in \mathbb{R}^m, A \in \mathbb{R}^{p\times n}, B \in \mathbb{R}^{p\times m}, c \in \mathbb{R}^p.$$

Полностью шаги ADMM для случая логистической регрессии с L_1 регуляризацией [1] приведены в Алгоритме 9.

[1] Правило обновления $\bar{z}^k$ в [17, раздел 8.3.3] содержит ошибку. Вместо $(\rho/2)$ должно быть $(\rho N/2)$. Алгоритм ADMM для логистической регрессии с L_1 регуляризацией показывал плохие результаты до исправления данной ошибки.

Алгоритм 9: Алгоритм ADMM для обучения GLM с L_1 регуляризацией

Вход : A_i, λ_1, ρ

1 $x_i \leftarrow 0,\ \bar{z} \leftarrow 0,\ \bar{u} \leftarrow 0,\ \overline{Ax} \leftarrow 0$;

2 **Пока** *не выполнено условие останова*

3 Выполнить параллельно на N машинах:

4 $x_i \leftarrow \operatorname{argmin}_{\widetilde{x}_i} \left((\rho/2)\|A_i\widetilde{x}_i - A_i x_i - \bar{z} + \overline{Ax} + u\|^2 + \lambda_1\|\widetilde{x}_i\|_1\right)$

5 $\overline{Ax} \leftarrow \frac{1}{N}\sum_{i=1}^{N} A_i x_i$, с помощью MPI_AllReduce

6 $\bar{z} \leftarrow \operatorname{argmin}_{\bar{z}} \left(\ell(N\bar{z}) + (\rho N/2)\|\bar{z} - \overline{Ax} - u\|^2\right)$

7 $u \leftarrow u + \overline{Ax} - \bar{z}$

Возврат: x_i, для $i = 1\ \ldots\ N$

Алгоритм приведен в т.н. "масштабированной форме" , в которой введена переменная $u = y/\rho$. Также достаточно хранить только среднее значение $\bar{z} = \frac{1}{N}\sum_{i=1}^{N} z_i$. Алгоритм 9 был реализован в программе `dlr` для логистической регрессии с L_1 регуляризацией. Шаг 4 представляет из себя задачу LASSO и решается с помощью одной итерации алгоритма Shooting (см. Алгоритм 1). Шаг 6 - это решение m независимых задач одномерной оптимизации вида (15)

$$\operatorname*{argmin}_t \left(\log(1 + \exp(-t)) + \frac{\rho}{2N}(t - c)^2\right). \tag{15}$$

Как и рекомендовано в [17], такие задачи решаются в два этапа. Сначала находится приближенное решение в таблице с предрасчитанными ответами для диапазона значений c. Потом делается 2 шага по методу Ньютона. Отметим, что Алгоритм 9 также можно рассматривать как распределенный вариант покоординатного спуска, т.к. покоординатный спуск выполняется по переменным x_i на всех машинах на шаге 4.

Скорость сходимости метода ADMM существенно зависит от параметра ρ. Параметр ρ необходимо или подбирать индивидуально для каждой задачи, или адаптивно менять от итерации к итерации [17, раздел 3.4.1].

2.3 Методы для L_2 регуляризации

Для обучения GLM с L_2 регуляризацией необходимо минимизировать выпуклую гладкую целевую функцию (9). Как было сказано в разделе 1.4, для этой задачи на больших обучающих выборках эффективно работают методы онлайн обучения и квазиньютоновские методы. Онлайн обучение может быть распараллелено аналогично случаю L_1 регуляризации (см. раздел 2.2.1) путем усреднения решений с нескольких машин. Что касается квазиньютоновских методов, то из них наиболее часто используются на практике: метод сопряженных градиентов, BFGS, L-BFGS, TRON. Данные методы не хранят Гессиан и требуют только вычислений значения целевой функции $f(\boldsymbol{\beta})$ и градиента $\nabla f(\boldsymbol{\beta})$. Заметим, что минус лог-правдоподобие

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)$$

и его градиент

$$\nabla L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \frac{\partial \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i)}{\partial \hat{y}} \mathbf{x}_i$$

аддитивны по обучающим примерам. Поэтому данные методы могут быть легко распараллелены при разделении обучающей выборки по примерам. Каждая машина вычисляет часть $L(\boldsymbol{\beta})$ и $\nabla L(\boldsymbol{\beta})$ по хранящимся на ней примерам, после чего производится суммирование с помощью MPI_AllReduce. Вектор $\boldsymbol{\beta}$ хранится на всех машинах, поэтому регуляризатор $\frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|^2$ также может быть вычислен. Именно таким образом реализованы параллельные варианты L-BFGS [18] и TRON [16].

В текущей работе для численных экспериментов используется распределенная комбинация онлайн обучения и L-BFGS (см. раздел 1.4.4), реализованная в Vowpal Wabbit. Обмен данными между машинами происходит с помощью MPI_AllReduce.

В статье [18] описаны численные эксперименты, выполненные с такой комбинацией методов на кластере из 1000 машин. Данная система позволила произвести обучение логистической регрессии на датасете размером более 1 ТБ.

3 Параллельный покоординатный спуск: метод d-GLMNET

3.1 Описание

В данной работе предложен метод d-GLMNET, который предназначен для параллельного и распределенного обучения обобщенных линейных моделей. Он может использоваться как для параллельного выполнения на одном компьютере с использованием нескольких процессоров/ядер, так и на компьютерном кластере. В методе d-GLMNET выполняются параллельные шаги по блокам переменных. Предположим, что множество из p признаков разбито на M непересекающихся подмножеств S^m

$$\bigcup_{m=1}^{M} S^m = \{1, \ \dots \ , p\},$$

$$S_m \cap S_k = \varnothing, k \neq m.$$

Будем рассматривать задачу минимизации регуляризованного эмпирического риска с гладкой функций риска $L(\boldsymbol{\beta})$ и с сепарабельным регуляризатором $R(\boldsymbol{\beta}) = \lambda_1 \|\boldsymbol{\beta}\|_1 + (\lambda_2/2)\|\boldsymbol{\beta}\|^2$

$$\operatorname*{argmin}_{\boldsymbol{\beta}} \left\{ L(\boldsymbol{\beta}) + R(\boldsymbol{\beta}) \right\}.$$

Так же, как и в методе GLMNET, рассмотрим минимизацию приближения к функции $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$, в котором $L(\boldsymbol{\beta})$ разложена вплоть до членов 2-го порядка ряда Тейлора

$$L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T H(\boldsymbol{\beta})\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}),$$

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}} L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}). \tag{16}$$

Следующая теорема показывает, что произойдет, если выполнять минимизацию (16) независимо по блокам переменных S^m.

Теорема 1. *Независимая оптимизация квадратичного приближения (16) по блокам переменных $\Delta\boldsymbol{\beta}^m$ эквивалентна оптимизации квадратичного приближения к целевой функции*

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}} \left\{ L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T \widetilde{H}(\boldsymbol{\beta})\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) \right\} \tag{17}$$

с блочно-диагональным $\widetilde{H}(\boldsymbol{\beta})$ приближением гессиана.

$$(\widetilde{H}(\boldsymbol{\beta}))_{jl} = \begin{cases} (\nabla^2 L(\boldsymbol{\beta}))_{jl}, \text{ если } \exists m: \ j, l \in S^m, \\ 0, \quad \text{иначе.} \end{cases} \tag{18}$$

Доказательство. Пусть $\Delta\boldsymbol{\beta} = \sum_{m=1}^{M} \Delta\boldsymbol{\beta}^m$, где $\Delta\boldsymbol{\beta}^m$ - компоненты вектора $\Delta\boldsymbol{\beta}$, соответствующие подмножествам S^m, то есть $\Delta\beta_j^m = 0$ если $j \notin S^m$. Тогда

$$\begin{aligned} L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) &= L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta}^m + \frac{1}{2}\Delta(\boldsymbol{\beta}^m)^T \nabla^2 L(\boldsymbol{\beta}) \Delta\boldsymbol{\beta}^m \\ &= L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta}^m + \frac{1}{2} \sum_{j,k \in S^m} (\nabla^2 L(\boldsymbol{\beta}))_{jk} \Delta\beta_j \Delta\beta_k. \end{aligned}$$

Просуммировав это выражение по m получим

$$\begin{aligned} \sum_{m=1}^{M} L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) &= \sum_{m=1}^{M} \left(L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta}^m + \frac{1}{2} \sum_{j,k \in S^m} (\nabla^2 L(\boldsymbol{\beta}))_{jk} \Delta\beta_j \Delta\beta_k \right) \\ &= ML(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \Delta\boldsymbol{\beta}^T \widetilde{H}(\boldsymbol{\beta}) \Delta\boldsymbol{\beta}. \end{aligned} \tag{19}$$

Из выражения (19) и сепарабельности регуляризатора $R(\boldsymbol{\beta})$ следует, что оптимизация следующего приближения к целевой функции

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}} \left\{ L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \Delta\boldsymbol{\beta}^T \widetilde{H}(\boldsymbol{\beta}) \Delta\boldsymbol{\beta} + R(\boldsymbol{\beta}) \right\}$$

эквивалентна решению M независимых задач

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}^m} \left\{ L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) + \sum_{j \in S^m} R(\beta_j + \Delta\beta_j^m) \;\middle|\; \Delta\beta_j^m = 0 \text{ если } j \notin S^m \right\}, \tag{20}$$

которые могут быть решены параллельно. □

В методе `d-GLMNET` используется более общее приближение

$$L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \Delta\boldsymbol{\beta}^T (\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I)) \Delta\boldsymbol{\beta}$$

и на каждой итерации решается задача

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}} \left\{ L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) \right\}. \tag{21}$$

Необходимость использования параметров $\mu \geq 1, \nu > 0$ будет изложена далее. В данном случае одномерная минимизация по $\Delta\beta_j$ будет иметь вид (см. Приложение 1)

$$\Delta\beta_j^* = \frac{T\left(\sum_{i=1}^{n} w_i x_{ij} r_i + \nu\beta_j, \lambda_1\right)}{\mu \sum_{i=1}^{n} w_i x_{ij}^2 + \lambda_2 + \nu} - \beta_j, \tag{22}$$

$$r_i = z_i - \mu(\Delta\boldsymbol{\beta}^T \mathbf{x}_i + (\beta_j + \Delta\beta_j) x_{ij}).$$

Общая структура метода `d-GLMNET` приведена в Алгоритме 10. В алгоритме `d-GLMNET` есть следующие “степени свободы”:

Алгоритм 10: Общая структура `d-GLMNET`

Вход : обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n$, $\lambda_1 \geq 0$, $\lambda_2 \geq 0$, $\eta_1 \geq 1, \eta_2 \geq 1$
разбиение множества признаков $S^1, \ldots, S^M$

$\boldsymbol{\beta} \leftarrow 0$
$\mu \leftarrow 1$
Пока *не выполнено условие останова*
  Выполнить параллельно на M машинах:
    Выбрать $P^m \subseteq S^m$
    Минимизировать $L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^m)$ по $\Delta\boldsymbol{\beta}^m$ (Алгоритм 11)
    $\Delta\boldsymbol{\beta} \leftarrow \sum_{m=1}^M \Delta\boldsymbol{\beta}^m$
    Найти $\alpha \in (0, 1]$ с помощью линейного поиска (Алгоритм 12)
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}$
  Если $\alpha < 1$ **тогда**
    $\mu \leftarrow \eta_1\mu$
  Иначе
    $\mu \leftarrow \max(1, \mu/\eta_2)$
Возврат: $\boldsymbol{\beta}$

- На каждой итерации можно обновлять не все компоненты $\boldsymbol{\beta}$, а только их часть $P^1 \cup \ldots \cup P^M$, где $P^m \subseteq S^m$. Эта возможность будет использована для решения "проблемы отстающего" (slow node problem), см. раздел 3.7.

- Использование параметра $\mu > 1$ важно для обеспечения разреженности решения в случае L_1 регуляризации, см. раздел 3.2.

- Добавление νI к приближению Гессиана необходимо для гарантии сходимости, см. раздел 3.3.

В качестве критерия останова используется превышение количества итераций заданного максимума или относительное изменение целевой функции между итерациями меньше определенного значения.

Алгоритм 11 описывает способ минимизации квадратичного приближения (20). `d-GLMNET` выполняет один цикл покоординатного спуска по всем компонентам $\boldsymbol{\beta}^m$ для приближенной минимизации (20). Несмотря на то, что GLMNET и newGLMNET выполняют минимизацию квадратичного приближения в несколько циклов, вычислительные эксперименты показали, что одного цикла достаточно.

Алгоритм 11: Минимизация квадратичного приближения на машине m

Вход : $\Delta\boldsymbol{\beta}^m \leftarrow 0$

1 **Цикл** $j \in P^m$

2 $\quad \Delta\beta_j^m \leftarrow \text{argmin}_{\Delta\beta_j^m}\{L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^m)\}$, используется формула (22)

Возврат: $\Delta\boldsymbol{\beta}^m$

Алгоритм 12: Линейный поиск

Вход : $\delta > 0, 0 < b < 1, 0 < \sigma < 1, 0 \leq \gamma < 1$

1 **Если** $\alpha = 1$ *обеспечивает достаточное уменьшение целевой функции (23)* **тогда**

2 $\quad \alpha \leftarrow 1$

3 **Иначе**

4 $\quad$ Найти $\alpha_{init} = \text{argmin}_{\delta<\alpha\leq 1} f(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta})$

5 $\quad$ Правило Армиджо: выбрать α равное наибольшему элементу последовательности $\{\alpha_{init} b^j\}_{j=0,1,\ldots}$ удовлетворяющему

$$f(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}) \leq f(\boldsymbol{\beta}) + \alpha\sigma D \tag{23}$$

$$D = \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \gamma\Delta\boldsymbol{\beta}^T(\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I))\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta})$$

Возврат: α

Как и в других квазиньютоновских алгоритмах, для обеспечения сходимости на каждом шаге нужно выполнять линейный поиск. Алгоритм 12 описывает используемую процедуру линейного поиска. Численные эксперименты показали, что если в качестве α_{init} брать точку минимума функции (2) (шаг 2, Алгоритм 12), то скорость сходимости `d-GLMNET` немного увеличивается. Значение α_{init} находится приближено с помощью 10 итераций метода золотого сечения. Во всех численных экспериментах использовались параметры $b = 0.5, \sigma = 0.01, \gamma = 0$

3.2 Обеспечение разреженности решения

Использование линейного поиска на шаге 8 Алгоритма 10 необходимо для гарантии сходимости, но, в то же время, усложняет нахождение разреженного решения. Из Алгоритма 10 следует, что финальное β_j может быть равно нулю в двух случаях:

- На *всех* итерациях $\Delta\beta_j = 0$.
- На одной из итераций $\Delta\beta_j = -\beta_j$, $\alpha = 1$; на *всех* последующих итерациях $\Delta\beta_j = 0$.

Первый случай является достаточно редким, поэтому для "зануления", переменной β_j на одной из итераций необходимо $\alpha = 1$. Но на практике приращения $\Delta\boldsymbol{\beta}^m$ с разных машин конфликтуют и Алгоритм 12 линейного поиска часто возвращает $\alpha < 1$.

Для обеспечения разреженности решения $\boldsymbol{\beta}$ алгоритм должен выполнять шаг с $\alpha = 1$ достаточно часто. Следующая теорема показывает, что при достаточно большом μ линейный поиск не требуется никогда.

Теорема 2. *Пусть $\Lambda_{max}, \lambda_{min}$ - максимальное и минимальное собственные значения $H(\boldsymbol{\beta})$ и $\widetilde{H}(\boldsymbol{\beta})$ соответственно. Тогда если $\mu \geq \frac{\Lambda_{max}}{(1-\sigma)\lambda_{min}}$, по критерий Армиджо (23) с $\gamma = 0$ будет выполнен для $\alpha = 1$.*

Доказательство. Пусть $g(t) = L(\boldsymbol{\beta} + t\Delta\boldsymbol{\beta})$. Тогда

$$g'(t) = \nabla L(\boldsymbol{\beta} + t\Delta\boldsymbol{\beta})^T \Delta\boldsymbol{\beta},$$
$$g''(t) = \Delta\boldsymbol{\beta}^T \nabla^2 L(\boldsymbol{\beta} + t\Delta\boldsymbol{\beta})\Delta\boldsymbol{\beta}.$$

Получим ограничение сверху на $L(\boldsymbol{\beta} + \Delta\boldsymbol{\beta})$

$$\begin{aligned} L(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) = g(1) = g(0) + \int_0^1 g'(t)dt \leq g(0) + \int_0^1 \left(g'(0) + t \max_{z\in[0,1]} |g''(z)| \right) dt \\ = L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \max_{z\in[0,1]} |\Delta\boldsymbol{\beta}^T \nabla^2 L(\boldsymbol{\beta} + z\Delta\boldsymbol{\beta})\Delta\boldsymbol{\beta}| \\ \leq L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Lambda_{max}\|\Delta\boldsymbol{\beta}\|^2. \end{aligned}$$

В предыдущем неравенстве использовалось $\nabla^2 L(\boldsymbol{\beta}) \preceq \Lambda_{max} I$. Тогда

$$\begin{aligned} f(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) - f(\boldsymbol{\beta}) = L(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) - L(\boldsymbol{\beta}) + R(\Delta\boldsymbol{\beta} + \boldsymbol{\beta}^*) - R(\boldsymbol{\beta}) \\ \leq \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Lambda_{max}\|\Delta\boldsymbol{\beta}\|^2 + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) - R(\boldsymbol{\beta}) = D + \frac{1}{2}\Lambda_{max}\|\Delta\boldsymbol{\beta}\|^2, \end{aligned} \tag{24}$$

где было использовано D из критерия Армиджо (23) для частного случая $\gamma = 0$

$$D = \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) - R(\boldsymbol{\beta}).$$

Пусть $\Delta\boldsymbol{\beta}^*$ минимизирует (21), тогда

$$L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^*) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) \leq L_q^{gen}(\boldsymbol{\beta}, 0) + R(\boldsymbol{\beta}),$$
$$\nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) + \frac{1}{2}(\Delta\boldsymbol{\beta}^*)(\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I))\Delta\boldsymbol{\beta}^* \leq R(\boldsymbol{\beta}).$$

Следовательно, D ограничено сверху

$$\nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^*) - R(\boldsymbol{\beta}) \leq -\frac{1}{2}(\Delta\boldsymbol{\beta}^*)(\mu(\widetilde{H}(\boldsymbol{\beta})) + \nu I)\Delta\boldsymbol{\beta}^*,$$
$$D \leq -\frac{1}{2}(\Delta\boldsymbol{\beta}^*)\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I)\Delta\boldsymbol{\beta}^*.$$

Учитывая, что $\lambda_{min} I \preceq \widetilde{H}(\boldsymbol{\beta}) + \nu I$ получим для $\mu \geq \frac{\Lambda_{max}}{(1-\sigma)\lambda_{min}}$:

$$\frac{1}{2}\Lambda_{max}\|\Delta\boldsymbol{\beta}^*\|^2 \leq \frac{1}{2}(1-\sigma)\mu\lambda_{min}\|\Delta\boldsymbol{\beta}^*\|^2 \leq \frac{1}{2}(1-\sigma)(\Delta\boldsymbol{\beta}^*)^T(\mu(\widetilde{H}+\nu I))\Delta\boldsymbol{\beta}^* \\ \leq -(1-\sigma)D. \tag{25}$$

Подставляя (25) в (24) получим

$$f(\boldsymbol{\beta}+\Delta\boldsymbol{\beta}^*) - f(\boldsymbol{\beta}) \leq D - (1-\sigma)D = \sigma D,$$

что доказывает выполнение критерия Армиджо при $\alpha = 1$. □

На практике тяжело вычислить $\Lambda_{max}, \lambda_{min}$ для произвольной большой обучающей выборки. Также использование параметра $\mu \geq \frac{\Lambda_{max}}{(1-\sigma)\lambda_{min}}$ может привести к слишком маленьким шагам $\Delta\boldsymbol{\beta}$ и медленной сходимости алгоритма. По этой причине в `d-GLMNET` параметр μ изменяется адаптивно, см. Алгоритм 10. В численных экспериментах использовались следующие параметры адаптивного изменения: $\eta_1 = \eta_2 = 2$.

Отметим, что так как линейный поиск всегда возвращает $\alpha = 1$ если $\mu \geq \frac{\Lambda_{max}}{(1-\sigma)\lambda_{min}}$, то при адаптивном изменении μ будет иметь место

$$1 \leq \mu < \frac{\eta_1\Lambda_{max}}{(1-\sigma)\lambda_{min}}.$$

Альтернативная интерпретация задачи (21) при $\mu > 1$ - это минимизация Лагранжиана задачи условной оптимизации

$$\underset{\Delta\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T(\widetilde{H}(\boldsymbol{\beta}) + \nu I)\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta}+\Delta\boldsymbol{\beta}) \right\}$$
$$\text{при ограничении:} \quad \Delta\boldsymbol{\beta}^T(\widetilde{H}(\boldsymbol{\beta}) + \nu I)\Delta\boldsymbol{\beta} \leq r,$$

с множителем Лагранжа $\mu - 1$. Шаги алгоритма ограничены доверительным интервалом r, зависящим от итерации.

3.3 Доказательство сходимости

Метод `d-GLMNET` относится к семейству методов циклического блочно-координатного спуска (CGD), предложенного в статье [35]. Семейство методов CGD предназначено для минимизации суммы двух функций

$$\min_{\boldsymbol{\beta}} F_c(\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + cR(\boldsymbol{\beta}) \tag{26}$$

Алгоритм 13: Метод CGD

Выбрать $\boldsymbol{\beta}^0 \in dom\ R$

Цикл $k = 0, 1, 2, ...$

  Выбрать непустое $\mathcal{J}^k \subseteq \mathbb{N}$ и матрицу $H^k \succ 0$

  Решить (27) используя $\boldsymbol{\beta} = \boldsymbol{\beta}^k, \mathcal{J} = \mathcal{J}^k, H = H^k$ и получить $d^k = d_{H^k}(x^k; \mathcal{J}^k)$

  Выбрать подходящий шаг $\alpha^k > 0$ по правилу Армиджо и положить $\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^k + \alpha^k d^k$

где $L(\boldsymbol{\beta})$ - гладкая (непрерывно дифференцируемая функция) на открытом множестве в $\mathbb{R}^n$ содержащем $dom\ R = \{\boldsymbol{\beta} \mid R(\boldsymbol{\beta}) < \infty\}$, а $R(\boldsymbol{\beta})$ - собственная выпуклая и полунепрерывная снизу функция, константа $c > 0$. Заметим, что $L(\boldsymbol{\beta})$ - непрерывна, т.к. она гладкая.

Шаги метода CGD приведены в Алгоритме 13. На каждой итерации выполняется минимизация

$$d_H(\boldsymbol{\beta}; \mathcal{J}) \stackrel{\text{def}}{=} \underset{d}{\operatorname{argmin}} \left\{ \nabla L(\boldsymbol{\beta})^T d + \frac{1}{2} d^T H d + cR(\boldsymbol{\beta} + d) \mid d_j = 0, \forall j \notin \mathcal{J} \right\} \tag{27}$$

Предложенный в данной работе метод покоординатного спуска `d-GLMNET` является частным случаем CGD. В нем используется $H^k = \mu(\widetilde{H}(\boldsymbol{\beta})_k + \nu I)$ - блочно-диагональное приближение к Гессиану, в которое для обеспечения положительной определенности было добавлено $\nu I,\ \nu > 0$. В случае `d-GLMNET` функция $L(\boldsymbol{\beta})$ - минус лог-правдоподобие, которое является гладкой и выпуклой функцией, определенной на $\mathbb{R}^n$, а $R(\boldsymbol{\beta})$ - это elastic net регуляризатор. Алгоритм линейного поиска (12) выполняется так, чтобы удовлетворить критерию Армиджо.

Для глобальной сходимости метода в соответствии с [35, Теорема 1(e)] достаточно, чтобы

$$(a) \quad e_{min} I \preceq H^k \preceq e_{max} I, \text{ где } e_{max} \geq e_{min} > 0 \tag{28}$$

(b) Последовательность $\{\mathcal{J}^k\}$ выбирается по обобщенному правилу Гаусса-Зейделя

$$\exists\, T > 0 \quad \forall\, k \quad \mathcal{J}^k \cup \mathcal{J}^{k+1} \cup \ \ldots\ \mathcal{J}^{k+T-1} = \mathbb{N}$$

(c) $R(\cdot)$ блочно-сепарабельно по $\mathcal{J}^k$ для всех k, $\sup_k \alpha^k < \infty$.

Также для вычислительной реализации метода необходимо, чтобы линейный поиск сходился за конечное число итераций.

Для этого согласно [35, Лемма 5(b)] достаточно выполнения следующих условий

$$(a) \quad \exists \Lambda > 0 \quad \forall \boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in dom\ R \quad ||\nabla L(\boldsymbol{\beta}_1) - \nabla L(\boldsymbol{\beta}_2)|| \leq \Lambda ||\boldsymbol{\beta}_1 - \boldsymbol{\beta}_2|| \tag{29}$$

$$(b) \quad H^k \succ 0$$

Обновление всех компонент $\boldsymbol{\beta}$ на каждой итерации очевидно является частным случаем правила Гаусса-Зейделя. Используемый в `d-GLMNET` elastic net регуляризатор полностью сепарабельный. Для данного регуляризатора $dom\ R = \mathbb{R}^p$. Поэтому для доказательства сходимости и завершения линейного поиска за конечное число шагов достаточно доказать только (28), (29). Для доказательства (29) заметим, что

$$\|\nabla L(\boldsymbol{\beta}_1) - \nabla L(\boldsymbol{\beta}_2)\| \leq \max_{t \in [0,1]} \|\nabla^2 L(\boldsymbol{\beta}_1 + t(\boldsymbol{\beta}_2 - \boldsymbol{\beta}_1))\| \|\boldsymbol{\beta}_1 - \boldsymbol{\beta}_1\| \leq \Lambda_{max} \|\boldsymbol{\beta}_1 - \boldsymbol{\beta}_1\|$$

Следовательно, линейный поиск про правилу Армиджо сойдется за конечное число шагов если норма Гессиана $\|\nabla^2 L(\boldsymbol{\beta})\|$ ограничена сверху хотя бы в точках $\boldsymbol{\beta}_k, \boldsymbol{\beta}_k + d_k$. Таким образом достаточно доказать только

$$\|\nabla^2 L(\boldsymbol{\beta})\| \leq \Lambda_{max}, \quad \boldsymbol{\beta} = \boldsymbol{\beta}_k, \boldsymbol{\beta}_k + d_k,\ k = 1 \ldots k \tag{30}$$

Теорема 3. *Если у выпуклой функции $g(\boldsymbol{\beta})$, определенной на множестве $\mathcal{D}$ существует единственный минимум $\boldsymbol{\beta}^* \in int\ \mathcal{D}$, то множества уровня ограничены.*

Доказательство. Рассмотрим сферу $S_\varepsilon = \{\boldsymbol{\beta} \mid \|\boldsymbol{\beta} - \boldsymbol{\beta}^*\| = \varepsilon\} \subseteq \mathcal{D}$.
Если $\boldsymbol{\beta}_1 \in S_\varepsilon$, то

$$g(\boldsymbol{\beta}_1) \geq g(\boldsymbol{\beta}^*) + \delta,\ \delta > 0$$

Пусть $\boldsymbol{\beta} \in \mathcal{D}$. Так как $g(\boldsymbol{\beta})$ - выпуклая, то

$$g(\boldsymbol{\beta}^* + t(\boldsymbol{\beta} - \boldsymbol{\beta}^*)) = g(t\boldsymbol{\beta} + (1-t)\boldsymbol{\beta}^*) \leq tg(\boldsymbol{\beta}) + (1-t)g(\boldsymbol{\beta}^*),\ 0 \leq t \leq 1$$

Пусть $t = \varepsilon / \|\boldsymbol{\beta} - \boldsymbol{\beta}^*\|$, тогда $\boldsymbol{\beta}^* + t(\boldsymbol{\beta} - \boldsymbol{\beta}^*) \in S_\varepsilon$. Значит

$$tg(\boldsymbol{\beta}) + (1-t)g(\boldsymbol{\beta}^*) \geq g(\boldsymbol{\beta}^*) + \delta$$

$$tg(\boldsymbol{\beta}) \geq tg(\boldsymbol{\beta}^*) + \delta$$

$$g(\boldsymbol{\beta}) \geq g(\boldsymbol{\beta}^*) + \frac{\delta}{t} = g(\boldsymbol{\beta}^*) + \frac{\delta}{\epsilon} \|\boldsymbol{\beta} - \boldsymbol{\beta}^*\|$$

Следовательно, при $\boldsymbol{\beta} \to \infty$, $g(\boldsymbol{\beta}) \to +\infty$ и множества уровня ограничены. □

Теорема 4. *Если множества субуровня $\mathcal{L}_c$ функции $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ ограничены, а также на каждом множестве $\mathcal{L}_c$ норма Гессиана $\|\nabla^2 L(\boldsymbol{\beta})\|$ ограничена, то выполняется (30).*

Доказательство. Пусть $\boldsymbol{\beta}^0$ - начальное приближение, после чего алгоритм `d-GLMNET` совершает шаги $\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots$. Рассмотрим множество субуровня $\mathcal{L}_c$, где $c = L(\boldsymbol{\beta}^0) + R(\boldsymbol{\beta}^0)$. Докажем, что $d_H(\boldsymbol{\beta}; \mathcal{J})$ из (27) ограничено. Обозначим

$$q_H(d, \boldsymbol{\beta}) \stackrel{\text{def}}{=} \nabla f(\boldsymbol{\beta})^T d + \frac{1}{2} d^T H d + cP(\boldsymbol{\beta} + d)$$

$$\nabla L_{max} = \max_{\boldsymbol{\beta} \in \mathcal{D}} \|\nabla L(\boldsymbol{\beta})\|$$

$$P_{max} = \max_{\boldsymbol{\beta} \in \mathcal{D}} P(\boldsymbol{\beta})$$

Выпуклая функция $P(\boldsymbol{\beta})$ субдифференцируема в любой точке $\boldsymbol{\beta}_1 \in int\ dom\ P$, пусть субградиент в этой точке равен p, тогда

$$P(\boldsymbol{\beta}) \geq P(\boldsymbol{\beta}_1) + p^T(\boldsymbol{\beta} - \boldsymbol{\beta}_1) \geq B + A\|\boldsymbol{\beta}\|$$

где $B = P(\boldsymbol{\beta}_1) - p^T \boldsymbol{\beta}_1, A = -\|p\|$.

$$\begin{aligned} q_H(d, \boldsymbol{\beta}) &\geq -\nabla L_{max}\|d\| + \frac{1}{2} e_{min}\|d\|^2 + B + A\|d\| \\ &\geq \frac{1}{2} e_{min} \left(\|d\| + \frac{A - \nabla L_{max}}{e_{min}} \right)^2 + B - \frac{1}{2} \frac{(A - \nabla L_{max})^2}{e_{min}} \end{aligned}$$

Если d - решение (27), то

$$q_H(d, \boldsymbol{\beta}) \leq q_H(0, \boldsymbol{\beta}) = cP(\boldsymbol{\beta}) \leq cP_{max}$$

Следовательно

$$\frac{1}{2} e_{min} \left(\|d\| + \frac{A - \nabla L_{max}}{e_{min}} \right)^2 + B - \frac{1}{2} \frac{(A - \nabla L_{max})^2}{e_{min}} \leq P_{max}$$

$$\|d\| \leq d_{max} \stackrel{\text{def}}{=} \frac{2}{e_{min}} \left(\sqrt{P_{max} - B + \frac{1}{2} \frac{(A - \nabla L_{max})^2}{e_{min}}} - \frac{A - \nabla L_{max}}{e_{min}} \right)$$

Рассмотрим d_{max}-окрестность множества $\mathcal{L}_c$: $\mathcal{L}_{c,d_{max}} = \{\boldsymbol{\beta} \mid \|\boldsymbol{\beta} - \boldsymbol{\beta}_0\| \leq d_{max}, \boldsymbol{\beta}_0 \in \mathcal{L}_c\}$. Получается, что $\boldsymbol{\beta} + d \in \mathcal{L}_c$. Далее $\boldsymbol{\beta}^1 = \boldsymbol{\beta}^0 + \alpha d$, где $0 < \alpha \leq 1$ выбирается по правилу Армиджо. Поэтому $f(\boldsymbol{\beta}^1) \leq f(\boldsymbol{\beta}^0)$ и $\boldsymbol{\beta}^1 \in \mathcal{L}_c$. Следовательно, для всех $\boldsymbol{\beta}_k \in \mathcal{L}_{c,d_{max}}$, $\boldsymbol{\beta}_k + d_k \in \mathcal{L}_{c,d_{max}}$ для всех k. Так как $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ непрерывно, то множества уровня $\mathcal{L}_c$ - компактны. Поэтому $\|\nabla^2 L(\boldsymbol{\beta})\|$ достигает максимума на $\mathcal{L}_c$ и ограничено сверху. Следовательно, выполняется условие (30).

□

Теорема 5. *Если функция $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ достигает минимум на открытой области определения $\mathcal{D}$, норма Гессиана $\|\nabla^2 L(\boldsymbol{\beta})\|$ непрерывна, то выполняется (30).*

Доказательство. Если $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ достигает минимум на открытой области определения $\mathcal{D}$, то по Теореме 3 множества уровня функции $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ ограничены. Так как $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ непрерывно, то множества уровня $\mathcal{L}_c$ - компактны. Поэтому $\|\nabla^2 L(\boldsymbol{\beta})\|$ достигает максимума на $\mathcal{L}_c$ и ограничено сверху. Следовательно, по Теореме 4 условия (30) выполняются. □

Теорема 6. *Если $\boldsymbol{\beta}_0, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots$ - шаги алгоритма* `d-GLMNET` *и $0 \prec \nabla^2 L(\boldsymbol{\beta}_k) \prec \Lambda$ для всех k, то выполняется условие (28) для блочно-диагонального приближения Гессиана*

$$H = \mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I)$$

где $\widetilde{H}(\boldsymbol{\beta})$ определено в (18).

Доказательство. Для произвольной матрицы N, если $N \succ \lambda I$, то для любого $\mathbf{a} \neq 0$

$$\mathbf{a}^T N \mathbf{a} > \lambda \|\mathbf{a}\|^2$$

Возьмем $\mathbf{a}$ такое, что $a_i = 0$ если $i \notin S^m$. В этом случае

$$\sum_{j,l \in S^m} N_{jl} a_j a_l > \lambda \sum_{j \in S^m} a_j^2$$

суммировав неравенства для всех m получим

$$\sum_m \sum_{j,l \in S^m} N_{jl} a_j a_l > \lambda \sum_m \sum_{j \in S^m} a_j^2$$

$$\mathbf{a}^T \widetilde{N} \mathbf{a} > \lambda \|\mathbf{a}\|^2$$

где $\widetilde{N}$ - блочно-диагональное приближение к N, соответствующее блокам $S^1, S^2, \ldots, S^M$. Аналогичный результат имеет место для $N \prec \lambda I$.

Предположим, что $1 \leq \mu \leq \mu_{max}$. Данные неравенства выполняются для постоянного или адаптивно меняющегося μ (см. Раздел 3.2). Поэтому если $0 \prec \nabla^2 L(\boldsymbol{\beta}_k) \prec \Lambda$ то

$$\nu I \prec \mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I) \prec \mu_{max}(\Lambda + \nu) I$$

□

Теорема 7. *Если функция потерь $\ell(y, \hat{y})$ выпуклая по $\hat{y}$ и ее вторая производная ограничена сверху*

$$\frac{\partial^2 \ell(y, \hat{y})}{\partial \hat{y}^2} < M \tag{31}$$

то для некоторых $\Lambda_{min}, \Lambda_{max} > 0$

$$\Lambda_{min} I \preceq \nabla^2 L(\boldsymbol{\beta}) + \nu I \preceq \Lambda_{max} I \tag{32}$$

Условие (31) и, следовательно, глобальная сходимость, имеет место для линейной, логистической и пробит регрессии, см. Приложение 2.

Доказательство. Так как функция потерь $\ell(y, \hat{y})$ выпуклая, то $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2} \geq 0$ и, следовательно

$$\mathbf{a}^T \nabla^2 L(\boldsymbol{\beta}) \mathbf{a} = \sum_{i=1}^{n} (\mathbf{a}^T \mathbf{x}_i) \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2} (\mathbf{a}^T \mathbf{x}_i) \geq 0$$

С другой стороны

$$\mathbf{a}^T \nabla^2 L(\boldsymbol{\beta}) \mathbf{a} = \sum_{i=1}^{n} (\mathbf{a}^T \mathbf{x}_i) \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2} (\mathbf{a}^T \mathbf{x}_i) \leq \|\mathbf{a}\|^2 \sum_{i=1}^{n} \|\mathbf{x}_i\|^2 M$$

поэтому для некоторых $\Lambda_{min}, \Lambda_{max} > 0$

$$\Lambda_{min} I \preceq \nabla^2 L(\boldsymbol{\beta}) + \nu I \preceq \Lambda_{max} I \tag{33}$$

□

Теорема 8. *Если функция потерь $\ell(y, \hat{y})$ выпуклая по $\hat{y}$ и ее вторая производная ограничена сверху на любом отрезке*

$$\frac{\partial^2 \ell(y, \hat{y})}{\partial \hat{y}^2} < M, \ \hat{y} \in [a, b] \tag{34}$$

(частный случай - $\partial^2 \ell(y, \hat{y}) / \partial \hat{y}^2$ непрерывна), то метод `d-GLMNET` *обладает глобальной сходимостью, кроме того, линейный поиск сходится за конечное число шагов.*

Доказательство. Линии уровня функции $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ - компактны. Если на любом отрезке $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2}$ ограничена, то, следовательно $\|\nabla^2 L(\boldsymbol{\beta})\|$ ограничена на любом ограниченном множестве (см. Теорему 7), в частности, на множествах уровня. Поэтому по Теореме 5 алгоритм `d-GLMNET` сходится.

□

Теорема 9. *Если $L(\boldsymbol{\beta})$ выпуклая дифференцируемая функция, определенная на открытом множестве, то линии уровня функции*

$$\mathcal{L}_c = \{\boldsymbol{\beta} \mid L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|^2 \leq c\}, \ \textit{где } c > c^*$$

$$c^* = \min_{\boldsymbol{\beta}} \left(L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|^2 \right)$$

компактны если $\lambda_2 > 0$.

Доказательство. Функция $L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}\|^2$ выпуклая и непрерывная, т.к. это сумма двух выпуклых функций, а выпуклая функция непрерывна на отрытом множестве. Поэтому линии уровня замкнуты. Необходимо только доказать, что они ограничены. Проведем доказательство от противного. Пусть существует последовательность $\{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ \dots \}$ такая что $\|\boldsymbol{\beta}_k\| \to +\infty$ и $f(\boldsymbol{\beta}_k) \leq c$ для всех k. Так функция $L(\cdot)$ - выпуклая, то

$$L(\boldsymbol{\beta}) \geq L(0) + \nabla L(0)^T \boldsymbol{\beta}$$

и значит

$$\begin{aligned} L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}\|^2 &\geq L(0) + \nabla L(0)^T\boldsymbol{\beta} + \frac{\lambda_2}{2}\|\boldsymbol{\beta}\|^2 \\ &\geq L(0) - \|\nabla L(0)\|\|\boldsymbol{\beta}\| + \frac{\lambda_2}{2}\|\boldsymbol{\beta}\|^2 \\ &\geq \frac{\lambda_2}{2}\left(\|\boldsymbol{\beta}\| - \frac{\|\nabla L(0)\|}{\lambda_2}\right)^2 + const \end{aligned}$$

где слагаемое *const* не зависит от $\boldsymbol{\beta}$ Поэтому $L(\boldsymbol{\beta}_k) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}_k\|^2 \to +\infty$ при $k \to +\infty$, что противоречит предположению $f(\boldsymbol{\beta}_k) \leq c$. □

Альтернативное доказательство сходимости формулирует следующая

Теорема 10. *Если $\partial^2\ell(y, \hat{y})/\partial\hat{y}^2$ непрерывна, то в присутствии L_2 регуляризации алгоритм* `d-GLMNET` *обладает глобальной сходимость и линейный поиск сходится за конечное число шагов.*

Доказательство. Пусть c^* - минимальное значение функции $f(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$, в которой по условию теоремы $\lambda_2 > 0$. Рассмотрим множества уровня

$$\mathcal{L}_c = \{\boldsymbol{\beta} \mid f(\boldsymbol{\beta}) \leq c\}, \text{ где } c > c^*$$

Теорема 9 говорит, что множества уровня функции $L(\boldsymbol{\beta}) + \frac{1}{2}\|\boldsymbol{\beta}\|^2$ компактны, следовательно, множества уровня $\mathcal{L}_c$ также компактны, т.к. $f(\boldsymbol{\beta}) \geq L(\boldsymbol{\beta}) + \frac{1}{2}\|\boldsymbol{\beta}\|^2$. Если $\frac{\partial^2\ell(y,\hat{y})}{\partial\hat{y}^2}$ непрерывна, то и $\nabla^2 L(\boldsymbol{\beta})$ непрерывна. Внутри компактного множества непрерывная функция ограничена

$$\|\nabla^2 L(\boldsymbol{\beta})\| \leq \Lambda, \ \boldsymbol{\beta} \in \mathcal{L}_c$$

Покажем, что линейный поиск закончится через конечное число шагов. Пусть $\boldsymbol{\beta}$ - приближенное решение на текущей итерации, $\Delta\boldsymbol{\beta}$ - шаг. При внимательном рассмотрении [35, Лемма 5(b)] видно, что достаточно выполнения условия (29) только для $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in [\boldsymbol{\beta}, \boldsymbol{\beta} + \Delta\boldsymbol{\beta}]$. Поскольку $\boldsymbol{\beta} \in \mathcal{L}_c$, то и некоторая часть отрезка $[\boldsymbol{\beta}, \bar{\alpha}\Delta\boldsymbol{\beta}] \subseteq [\boldsymbol{\beta}, \Delta\boldsymbol{\beta}]$, где

$0 < \bar{\alpha} \leq 1$, всегда будет содержаться в $\mathcal{L}_c$: $[\boldsymbol{\beta}, \bar{\alpha}\Delta\boldsymbol{\beta}] \subseteq \mathcal{L}_c$, так как $\mathcal{L}_c$ - компакт. Поэтому критерий Армихо будет удовлетворен для некоторого $\alpha \leq \bar{\alpha}$.

При выполнении шагов алгоритма `d-GLMNET`, удовлетворяющих критерию Армиджо, целевая функция $f(\boldsymbol{\beta})$ не возрастает, поэтому всегда $\boldsymbol{\beta} \in \mathcal{L}_c$ и $\|\nabla^2 L(\boldsymbol{\beta})\| < \Lambda$. Согласно Теореме 6 условие (28) будет выполнено для блочно-диагонального приближения Гессиана и обеспечена сходимость.

□

Данный вариант теоремы распространяется, например, на Пуассоновскую регрессию с L_2 регуляризацией или elastic net регуляризацией.

Резюме. Достаточное условие сходимости - это существование единственного минимума целевой функции $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ вместе с ограничением сверху на вторую производную $\partial^2 \ell(y, \hat{y})/\partial \hat{y}^2$ на любом отрезке. В частности, достаточно непрерывности $\partial^2 \ell(y, \hat{y})/\partial \hat{y}^2$ по $\hat{y}$. Это будет выполняться для линейной и логистической регрессии с любой регуляризацией, а также для Пуассоновской регрессии при наличии L_2 или elastic net регуляризации.

3.4 Линейная скорость сходимости

В формулировке теоремы о сходимости используются следующие обозначения:

$\bar{X}$ - множество стационарных точек (локальные экстремумы) функции $f(\cdot)$,

$$dist(x, \bar{X}) \stackrel{\text{def}}{=} \min_{\bar{x} \in \bar{X}} \|x - \bar{x}\| \quad \forall x \in \mathbb{R}^n$$

Предположение 1. *$\bar{X} \neq \varnothing$ и для всех $\zeta \geq \min_x f(x)$ найдутся $\tau > 0$ и $\epsilon > 0$ такие что*

$$dist(x, \bar{X}) \leq \tau ||d_I(x; \mathbb{N})|| \text{ , если } \quad f(x) \leq \zeta, ||d_I(x; \mathbb{N})|| \leq \epsilon$$

где

$$d_H(x) \stackrel{\text{def}}{=} d_H(x; \mathbb{N})$$

Ограниченное правило Гаусса-Зейделя. Пусть $\mathcal{T} \subseteq \{0, 1, \dots\}$, так что

$$0 \in \mathcal{T}, \quad \forall k \in \mathcal{T} \quad \mathbb{N} = \big(\text{объединение непересекающихся множеств } \mathcal{J}^k, \mathcal{J}^{k+1}, \dots, \mathcal{J}^{\tau(k)-1}\big) \tag{35}$$

где $\tau(k) \stackrel{\text{def}}{=} \min\{k' \in \mathcal{T} \mid k' > k\}$. Ограниченное правило Гаусса-Зейделя является частными случаем обычного правила Гаусса-Зейделя. Теорема о линейной скорости сходимости для семейство методов CGD формулируется следующим образом

Теорема. [35, Theorem 2(b)]. Assume that f satisfies (29) for some $\Lambda > 0$. Let $\{x^k\}$, $\{H^k\}$, $\{d^k\}$ be sequences generated by the CGD method satisfying (28), where $\{\mathcal{J}^k\}$ is chosen by restricted Gauss-Seidel rule (35) with $\mathcal{T} \subseteq \{0, 1, \ldots\}$. Then the following results hold.

(b) If F_c satisfies Assumption 1, $P(\cdot)$ is block-separable with respect to $\mathcal{J}^k$ for all k, and α_k is chosen by the Armijo rule with $\sup_k \alpha_{init}^k \leq 1$ and $\inf_k \alpha_{init}^k > 0$ then either $\{F_c(x^k)\} \downarrow -\infty$ or $\{F_c(x^k)\}_{\mathcal{T}}$ converges as least Q-linearly and $\{x^k\}_{\mathcal{T}}$ converges at least R-linearly.

Теорема 11. *Если $\nabla^2 L(\boldsymbol{\beta}) < \Lambda$, $L(\boldsymbol{\beta})$ - строго выпуклая, а также множества уровня $\mathcal{L}_c$ функции уровня $L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$ компактны, то выполняется Предположение 1.*

Доказательство. Данная теорема доказана для логистической регрессии с L_1 регуляризацией в [14, Appendix B]. Внимательный анализ доказательства показывает, что оно может быть легко обобщено на другие строго выпуклые функции потерь $L(\boldsymbol{\beta})$ и выпуклые регуляризаторы $R(\boldsymbol{\beta})$, в т.ч. elastic net. Поэтому для экономии места формальное доказательство опущено. □

Теорема 12. *Предположение 1 верно, если выполняется хотя бы одно из условий:*

1. *$L(\boldsymbol{\beta})$ - строго выпуклая, $\ell(y, \hat{y}) \geq C(y)$ и $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2} < M$.*

2. *$L(\boldsymbol{\beta})$ - выпуклая, присутствует L_2 регуляризатор.*

Доказательство. 1.

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + R(\boldsymbol{\beta}) \geq \sum_{i=1}^{n} C(y_i) + R(\boldsymbol{\beta}) \to +\infty, \text{ если } \|\boldsymbol{\beta}\| \to +\infty$$

Поэтому множества уровня $\mathcal{L}_c$ будут ограниченными и компактными.
Если $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2} < M$, то и $\nabla^2 L(\boldsymbol{\beta}) < \Lambda$.

2. Преобразуем целевую функцию

$$f(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + R(\boldsymbol{\beta}) = \left(L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|^2 \right) + \lambda_1 \|\boldsymbol{\beta}\|$$

и будем считать, что L_2 регуляризация является частью целевой функции.
Если $L(\boldsymbol{\beta})$ - выпуклая, то в соответствии с Теоремой 9 множества уровня функции $L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|^2$ компактны
Шаги алгоритма `d-GLMNET` не изменятся от того, является ли $\frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|^2$ частью $L(\boldsymbol{\beta})$ или регуляризатора $R(\boldsymbol{\beta})$. Действительно, на каждой итерации минимизируется квадратичное приближение (21), где

$$L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T (\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I))\Delta\boldsymbol{\beta}$$

Если $L^*(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}\|^2$, то

$$
\begin{aligned}
&L_q^{*,gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) = \\
&= L^*(\boldsymbol{\beta}) + \nabla L^*(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T(\mu(\widetilde{H}^*(\boldsymbol{\beta}) + \nu I))\Delta\boldsymbol{\beta} \\
&= L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}\|^2 + \nabla L(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + \lambda_2\boldsymbol{\beta}^T\Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T(\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I))\Delta\boldsymbol{\beta} + \frac{1}{2}\mu\lambda_2\Delta\boldsymbol{\beta}^T\Delta\boldsymbol{\beta} \\
&= L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|^2 + \frac{1}{2}(\mu - 1)\|\Delta\boldsymbol{\beta}\|^2
\end{aligned}
$$

Поэтому при $\mu = 1$ решение задачи (21) будет таким же.

Необходимо также удостовериться, что линейный поиск найдет такой же шаг α. В линейном поиске используется параметр

$$
\begin{aligned}
D &= \nabla L(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + \gamma\Delta\boldsymbol{\beta}^T(\mu(\widetilde{H}(\boldsymbol{\beta}) + \nu I))\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta}) \\
&= \nabla L(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta})
\end{aligned}
$$

при $\gamma = 0$. Если мы будем использовать $L^*(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}\|^2$ вместо $L(\boldsymbol{\beta})$, то

$$
\begin{aligned}
D &= \nabla L^*(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta}) \\
&= \nabla L(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + \lambda_2\boldsymbol{\beta}^T\Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta}) \\
&= \nabla L(\boldsymbol{\beta})^T\Delta\boldsymbol{\beta} + (R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|^2) - (R(\boldsymbol{\beta}) + \frac{1}{2}\lambda_2\|\boldsymbol{\beta}\|^2)
\end{aligned}
$$

и останется неизменным.

□

Наконец, проверим выполнение условий [35, Теоремы 2(b)] применительно к алгоритму `d-GLMNET`.

1. Частным случаем ограниченного правила Гаусса-Зейделя является, очевидно $\mathcal{J}^k = \mathbb{N}$, т.е. обновление всех весов на каждой итерации.

2. Теоремы 6, 7 доказывают выполнения Предположения 1 если $\frac{\partial^2\ell(y,\hat{y})}{\partial\hat{y}^2} < M$. Данное условие выполняется для линейной, логистической и пробит-регрессии.

3. Теорема 12 описывает условия, при которых будет выполняться Предположение 2. Вариант 1 выполняется для линейной, логистической и пробит-регрессии при условии строгой выпуклости $L(\boldsymbol{\beta})$. Это будет выполняться если нет линейно зависимых признаков $\mathbf{x}_i$. Вариант 2 выполняется для всех обобщенных линейных моделей с L_2 регуляризатором, в частном случае для Пуассоновской регрессии с L_2 регуляризацией.

4. Регуляризатор elastic net сепарабелен.

5. Алгоритм линейного поиска удовлетворяет условиям $\sup_k \alpha_{init}^k \leq 1$, $\inf_k \alpha_{init}^k \geq \delta > 0$;

6. $F(x) > -\infty$, поэтому имеет место линейная сходимость $\{x^k\}$ и $\{F(x^k)\}$.

3.5 Программная реализация. Запуск на кластере Map/Reduce

В программной реализации метода `d-GLMNET` используются одновременно возможности кластера Map/Reduce и передачи сообщений с помощью стандарта MPI. Алгоритм распределенного покоординатного спуска не был реализован полностью на Map/Reduce, так как данная модель вычислений плохо подходит для итеративных алгоритмов машинного обучения [62, 18]. Использование других моделей вычислений, например Spark [40], выглядит перспективно. Сам метод `d-GLMNET` реализован в программе `dlr`[2], которая запускается на кластере Map/Reduce в режиме стримминга, читая обучающую выборку из HDFS. Процессы программы `dlr`, запущенные на разных машинах, синхронизируются с помощью процедуры MPI_AllReduce. Такая комбинация удобна на практике, т.к. кластеры Map/Reduce широко распространены в компаниях и используются для хранения и обработки больших объемов данным. В тоже время, обмен данными по стандарту MPI существенно ускоряет программу, ведь, как было сказано выше, модель Map/Reduce плохо приспособлена для реализации итеративных алгоритмов машинного обучения. Недостатком данной комбинированной модели является снижение отказоустойчивости. Отказ одной из машин приводит к потери состояния алгоритма невозможности дальнейшего обмена данными с помощью MPI_AllReduce.

Для алгоритмов покоординатного спуска необходимо хранить обучающую выборку в формате инвертированного индекса. В данном формате для каждой переменной β_j хранится список пар $L_j = \{(i, x_{ij}) \mid x_{ij} \neq 0\}$ в порядке возрастания i. Таким образом, элементы матрицы X, относящиеся к одной переменной сгруппированы вместе. Формат инвертированного индекса особенно эффективен для хранения разреженных выборок.

Однако на практике большая часть выборок хранится в формате “по примерам”, в котором сгруппированы вместе элементы матрицы X, относящиеся к одному обучающему примеру. Преобразование из формата “по примерам”, в формат “по переменным”, для больших выборок - непростая вычислительная задача.

Данная задача может быть эффективно решена на кластере Map/Reduce. Псевдокод

[2] `https://github.com/IlyaTrofimov/dlr`

приведен в Алгоритме 14, в котором сразу после преобразования обучающей выборки в формат “по переменным”, запускается программа `dlr` и выполняется обучение.

Алгоритм 14: Преобразование выборки в формат “по примерам” и выполнение обучения на кластере Map/Reduce

Вход : Обучающая выборка $\{\mathbf{x}_i, y_i\}_{i=1}^n$

1 *map*: $(key = i, value = (\mathbf{x}_i,\ y_i)) \rightarrow\ list(key = j,\ value = x_{ij})$,

2 (для всех i, j таких, что $x_{ij} \neq 0$)

3 *reduce*: $(key = j,\ list(value = x_{ij})) \rightarrow$ Программа `dlr`

4 Выполнить обучение с помощью программы `dlr`, найти вектор $\boldsymbol{\beta}$

5 Записать $\boldsymbol{\beta}$ в HDFS

Сначала операция *map* разделяет обучающие примеры по коэффициентам отдельных переменных, после этого шага общее число записей будет равно nnz. Между *map* и *reduce* происходит группировка записей с одним ключом, т.е. отвечающих одному β_j. Hadoop запускает несколько задач *reduce*, каждая из которых получает на обработку подмножество записей $(key = j,\ list(value = x_{ij}))$, выбираемых случайно по хешу от ключа. Таким образом, признаки разделяются случайно между машинами, что полезно для численной оптимизации. Операция *reduce* выполняется по технологии стриминга[3], вызывая программу `dlr`.

Операция *reduce* получает на вход подмножество записей $(key = j,\ list(value = x_{ij}))$, т.е. часть обучающей выборки, отвечающей некоторому подмножеству признаков S^m. Обозначим данную часть обучающей выборки через X^m

$$X^m = \{L_j \mid j \in S^m\}, \quad L_j = \{(i, x_{ij}) \mid x_{ij} \neq 0\}$$

После получения данных, программа `dlr` преобразует их в бинарный формат (таблица 1) и сохраняет в файл. Специальный формат файла для обучающей выборки позволяет читать его только последовательно при выполнении шагов по переменным.

Таблица 1: Формат бинарного файла для X^m

feature_id	(example_id, value)	(example_id, value)	...	feature_id	(example_id, value)	...

[3] `http://hadoop.apache.org/docs/r1.2.1/streaming.html`

Алгоритм 15: Распределенный покоординатный спуск

Вход : Обучающая выборка, λ_1, λ_2, разбиение признаков $S^1, \ldots, S^M$

1 **Пока** *не выполнено условие останова*

2 Выполнить параллельно на M машинах:

3 Прочитать последовательно часть обучающей выборки X^m

4 Вычислить $\Delta\boldsymbol{\beta}^m$ и $X^m\Delta\boldsymbol{\beta}^m$ для весов из $P^m \subseteq S^m$

5 Суммировать вектора $X^m\Delta\boldsymbol{\beta}^m$ с помощью MPI_AllReduce:

6 $X\Delta\boldsymbol{\beta} \leftarrow \sum_{m=1}^{M} X^m\Delta\boldsymbol{\beta}^m$

7 Вычислить размер шага α используя линейный поиск (Алгоритм 12)

8 $\boldsymbol{\beta}^m \leftarrow \boldsymbol{\beta}^m + \alpha\Delta\boldsymbol{\beta}^m$

9 $X\boldsymbol{\beta} \leftarrow X\boldsymbol{\beta} + \alpha X\Delta\boldsymbol{\beta}$

Возврат: $\boldsymbol{\beta}$

3.6 Алгоритм программы dlr

Алгоритм 15 описывает эффективную реализацию распределенного покоординатного спуска в программе `dlr`. Машина с индексом m хранит часть X^m обучающей выборки, отвечающей подмножеству весов S^m. Будем использовать обозначение

$$\boldsymbol{\beta} = ((\boldsymbol{\beta}^1)^T, \ldots, (\boldsymbol{\beta}^M)^T)^T$$

Алгоритм 15 обладает следующими особенностями:

1. Вектор весов $\boldsymbol{\beta}$ хранится распределенно на всех M машинах; машина с индексом m хранит $\boldsymbol{\beta}^m$.

2. Программа хранит в оперативной памяти только вектора $\mathbf{y}$, $X\boldsymbol{\beta}$ [4], $X\Delta\boldsymbol{\beta}$, $\boldsymbol{\beta}^m$, $\Delta\boldsymbol{\beta}^m$.

   Размер занимаемой оперативной памяти на машине m равен $3n + 2|S^m|$.

3. Программа синхронизирует вектор $X\boldsymbol{\beta}$ между всеми машинами после каждой итерации. Синхронизация выполняется с помощью суммирования $X^m\Delta\boldsymbol{\beta}^m$ на шаге 6. Общий объем передаваемых данных равен Mn. Ограничение сверху на время передачи на каждой итерации - $O(n \ln M)$. Логарифм возникает из-за того, что во время процедуры MPI_AllReduce машины обмениваются данными по структуре бинарного дерева.

[4] Для частного случая логистической регрессии можно хранить вектор $\exp(X\boldsymbol{\beta})$ вместо $X\boldsymbol{\beta}$ для ускорения вычислений.

4. На каждой итерации обновляется подмножество весов $P^m \subseteq S^m$. В разделе 3.7 описываются две возможные стратегии выбора подмножеств P^m.

5. На шаге 4 могут быть выполнены различные варианты покоординатных обновлений. Вариант, используемый в `d-GLMNET` описан в разделе 3.1. Вычисления выполняются за один проход по части обучающей выборки X^m и имеют сложность $O(nnz)$, что хорошо подходит для больших разреженных выборок.

6. Программа `dlr` читает обучающую выборку последовательно с диска, а не из оперативной памяти. Это медленнее в случае небольших выборок, но делает программу более масштабируемой. Также это соответствует типичному использованию Map/Reduce кластеров: большие диски, много задач запускается параллельно разными пользователями. Каждая задача может обрабатывать большой объем данных, но она занимает небольшой объем оперативной памяти.

7. Выполнение линейного поиска на шаге 7 требует нахождения α такого, что

$$f(\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + R(\boldsymbol{\beta})$$
$$f(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}) \leq f(\boldsymbol{\beta}) + \alpha\sigma D$$
$$D = \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) - R(\boldsymbol{\beta})$$

   Предполагаем, что регуляризатор $R(\cdot)$ - блочно-сепарабельный, например, L_1, L_2, group lasso, SCAD, и т.п.

   (a) Если регуляризатор $R(\cdot)$ - сепарабельный, то величина D [5] также сепарабельна

$$D = \sum_{m=1}^{M} \left\{ \nabla L(\boldsymbol{\beta}^m)^T \Delta\boldsymbol{\beta}^m + R(\boldsymbol{\beta}^m + \Delta\boldsymbol{\beta}^m) - R(\boldsymbol{\beta}^m) \right\}$$

   Ее компоненты могут быть вычислены независимо на всех машинах, а потом сложены с помощью MPI_AllReduce.

   (b) Необходимо вычислять лог-правдоподобие $L(\boldsymbol{\beta}+\alpha\Delta\boldsymbol{\beta})$ и регуляризатор $R(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta})$ для произвольного $\alpha \in (0, 1]$. Так как вектора $X\boldsymbol{\beta}$ синхронизирован между машинами, то лог-правдоподобие может быть вычислено на каждой машине

$$L(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}) = \sum_{i=1}^{n} \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i + \alpha\Delta\boldsymbol{\beta}^T \mathbf{x}_i)$$

[5] Слагаемого с Гессианом нет, т.к. используется вариант линейного поиска в котором $\gamma = 0$

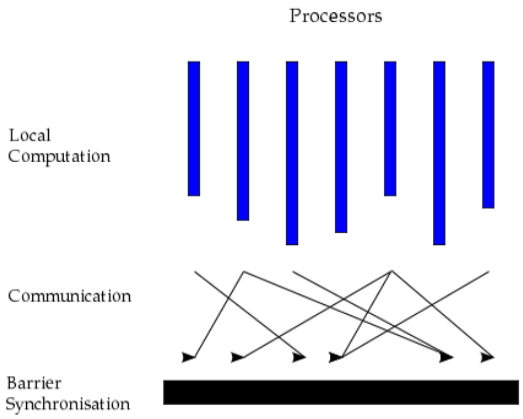


Рис. 4: Bulk Synchronous Parallel

Каждая машина вычисляет часть регуляризатора $R(\beta^m)$ независимо, после чего они суммируются с помощью MPI_AllReduce [6].

8. В Алгоритме 14 и программе `dlr` используется стиль программирования Single Program Multiple Data (SPMD), т.е. все машины выполняют одинаковый код и среди них нет выделенного мастера. Некоторые операции, например линейный поиск на шаге 7 избыточно выполняются на всех машинах.

3.7 Асинхронная балансировка нагрузки

В программе `dlr` поддерживается два варианта выбора подмножества признаков P^m. Первый вариант - всегда брать $P^m = S^m$. В этом случае каждая машина должна выполнить фиксированный объем вычислений, после этого на шаге 6 машины обмениваются данными и синхронизируются. Такой алгоритм работы является частным случаем модели вычислений Bulk Synchronous Parallel (BSP) [85], которая применима для итеративных алгоритмов. В модели BSP каждая итерация состоит из трех фаз (см. рис. 4).

На первой фазе процессоры получают на вход общие данные от предыдущей итерации и производят необходимые вычисления. Во второй фазе процессоры обмениваются данными между собой, вычисляя конечный результат итерации. Наконец, на третьей фазе происходит синхронизация, т.е. ожидание того, чтобы все процессоры завершили необходимые вычисления.

Модель BSP удобна для программирования, но ее реализация будет медленно работать на практике, если один из процессоров выполняет свой блок задач существенно медлен-

[6] Используется реализация из проекта Vowpal Wabbit
`https://github.com/JohnLangford/vowpal_wabbit`

нее, чем остальные. Время выполнения одной итерации определяется самым медленным процессором из-за необходимости синхронизации. В контексте распределенных вычислений такая ситуация называется "проблемой отстающего" (straggler problem, slow machine problem) и, чем больше процессоров работает параллельно, тем существенней эта проблема. Заметим, что один из процессоров может работать медленнее по целому ряду причин: неравномерность распределения объема вычислений, перегруженность другими задачами, более медленное оборудование, ошибки настройки оборудования, и т.п.

Существуют разные подходы к решению "проблемы отстающего". Во-первых можно использовать асинхронную передачу данных между процессорами [86] или машинам [84]. Техника Stale Synchronous Parallel Parameter Server (SSPPS) [24] обеспечивает частичную асинхронность, позволяя самой медленной машине отставать от самой быстрой не более чем на фиксированное число итераций. Архитектура для распределенных вычислений Fugue [87] позволяет машинам проводить более точную оптимизацию, ожидая медленную машину. Метод Y!LDA [47] для обучения тематических моделей на больших выборках поддерживает глобальное состояние в сервере параметров, а каждая машина обновляет его асинхронно.

В реализациях модели Map/Reduce иногда используется техника спекулятивного выполнения [57]. Когда большая часть операций *map* или *reduce* завершились, не завершившиеся операции перезапускаются на тех же блоках данных на других машинах. Несмотря на дополнительные вычисления, такая техника часто ускоряет выполнение задач *map/reduce* на полном объеме данных. Техника спекулятивного выполнения не применима для алгоритмов, поддерживающих состояние. В алгоритме `d-GLMNET` это вектора $X\boldsymbol{\beta}, \boldsymbol{\beta}^m$.

Время, которая машина затрачивает на одну итерацию, во-первых сильно зависит от размера X^m, во-вторых - от загруженности машины другим задачами. При практическом использовании программы `dlr` было обнаружено, что самые медленные машины - далеко не всегда отвечают за самую большую часть X^m.

Для решения этой проблемы был разработан алгоритм асинхронной балансировки нагрузки - Asynchronous Load Balancing (ALB). В рамках этого алгоритма подмножество P^m выбирается адаптивно. В программе есть дополнительный поток (thread), который проверяет, сколько машин уже выполнили обновления по всем переменным из S^m (Алгоритм 15, шаг 4). Если доля таких машин больше, чем κM, где $0 < \kappa < 1$, то все машины прерывают процесс оптимизации и переходят к шагу синхронизации (Алгоритм 15, шаг 6).

Обновление весов из S^m выполняются циклически, поэтому на следующей итерации

машина продолжит выполнять шаги по переменным, начиная со следующей из S^m. Таким образом, быстрые машины могут сделать больше одного цикла по переменным их S^m, т.е. обновить один вес несколько раз. В численных экспериментах использовался параметр $\kappa = 0.75$.

Будем называть модификацию метода `d-GLMNET` с асинхронной балансировкой нагрузки - `d-GLMNET-ALB`. Потенциальным недостатком метода является то, что очень медленная машина может не произвести обновления всех переменных, за которые она отвечает, даже после большого числа итераций. Впрочем, такая крайняя ситуация не встречалась в численных экспериментах.

Метод `d-GLMNET-ALB` обладает глобальной сходимостью, при условии, что $P^1, \ldots, P^M$ покрывают $\{1, \ldots, p\}$ каждые T итераций. Данное условие выполняется на практике. Однако линейная скорость сходимости не может быть доказана, т.к. порядок обновление переменных $P^1, \ldots, P^M$ - недетерминированный, и не соответствует ограниченному правилу Гаусса-Зейделя [35].

Алгоритм 16: Асинхронная балансировка нагрузки для покоординатного спуска

Вход : обучающая выборка, разбиение признаков $S^1, \ldots, S^m$, $0 < \kappa < 1$

$j^1 \leftarrow 0, \ldots, j^m \leftarrow 0$
Пока не выполнено условие останова:
  Выполнить параллельно на M машинах:
    Запустить поток 1:
      $d^m \leftarrow 0$
      Выполнять в цикле:
        $i^m \leftarrow S^m[j^m]$
        Вычислить шаг $\Delta\beta_{i^m}$
        $d^m \leftarrow d^m + 1$
        $j^m \leftarrow (j^m + 1)\ mod\ |S^m|$
    Запустить асинхронно поток 2:
      Ожидать 100 мс
      Если $d^m \geq |S^m|$ на более чем κM машинах, то прервать поток 1.
  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}$

Возврат: $\boldsymbol{\beta}$

На рис. 5 приведена гистограмма среднего времени одной итерации для выборки $yandex\_ad$ при параллельном обучении на 64 машинах. Несмотря на то, что большая часть машин

выполняет итерацию за примерно одинаковое время, некоторые машины более медленные, а одна - существенно. На рис. 6 показана аналогичная гистограмма для программы *dlr* с ALB. Видно, что в гистограмме нет выбросов (существенно отстающих машин).

Рис. 7 показывает гистограмму количества циклов по подмножествам переменных S^m, которое было выполнено на первой итерации Алгоритма 16. Видно, что часть машин "отстающие выполнили меньше 100%, а другая часть - "опережающие" успели выполнить цикл несколько раз. Таким образом, Алгоритм ALB, с одной стороны, успешно справляется с проблемой отстающих машин, с другой стороны - позволяет полностью задействовать ресурс быстрых машин и избегать простоя.

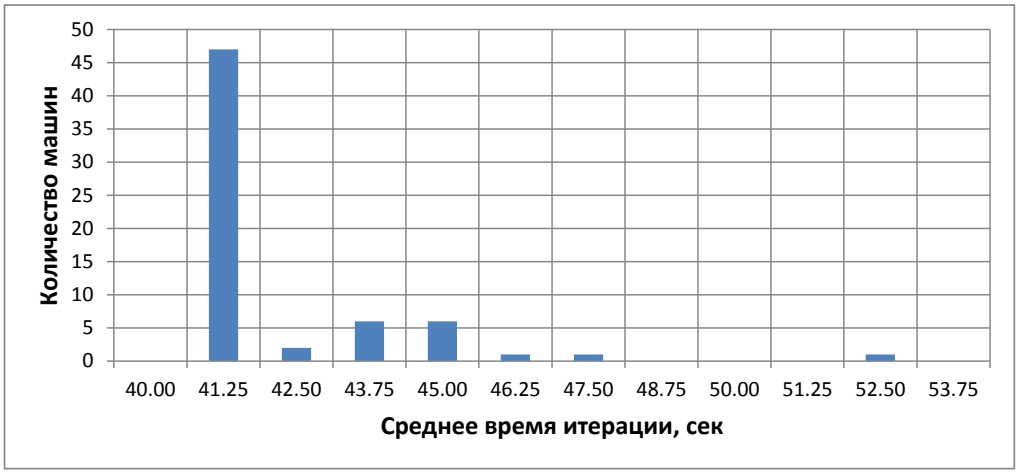


Рис. 5: Распределение времени итерации алгоритма `d-GLMNET`

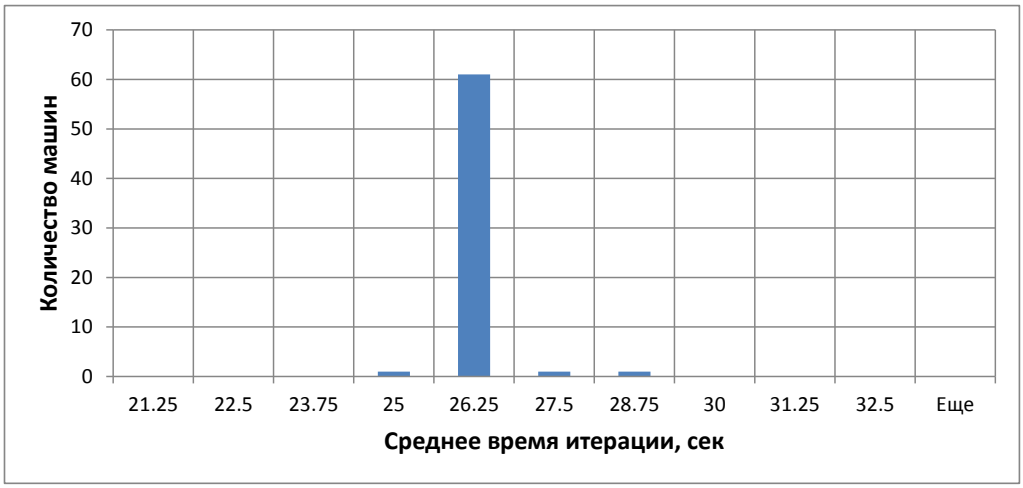


Рис. 6: Распределение времени итерации алгоритма `d-GLMNET-ALB`

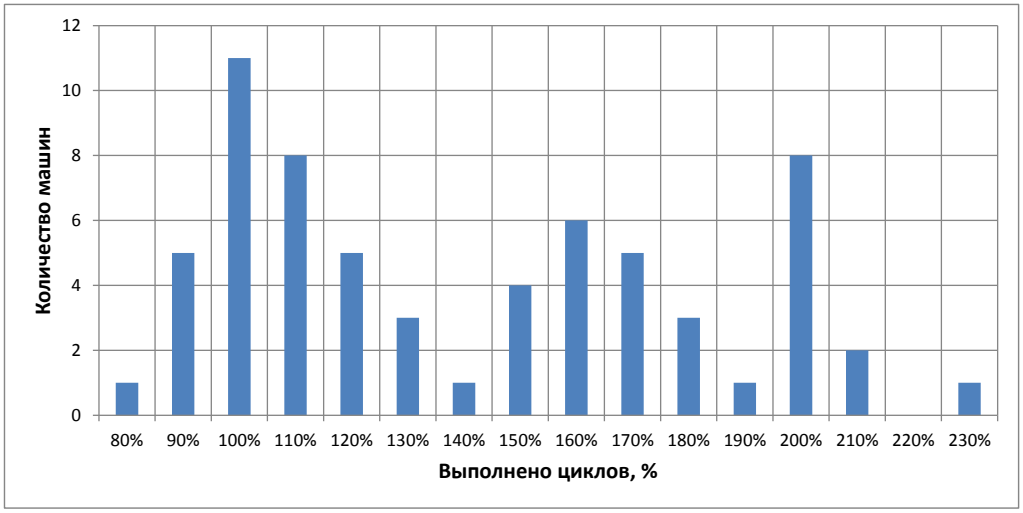


Рис. 7: Гистограмма выполненных циклов обновления весов, алгоритм `d-GLMNET-ALB`

Таблица 2: Оценка сложности методов

Метод	Сложность итерации	Передача по сети
Онлайн обучение с усеченным градиентом	$O(nnz)$	$O(p)$
L-BFGS	$O(nnz)$	$O(p)$
d-GLMNET	$O(nnz)$	$O(n)$
ADMM, sharing	$O(nnz)$	$O(n)$

4 Численные эксперименты с методом d-GLMNET

4.1 Сравнение методов

В ходе численных экспериментах метод d-GLMNET сравнивался со следующими методами:

1. Онлайн обучение с усеченным градиентом (для L_1, L_2 регуляризации)
2. L-BFGS (для L_2 регуляризации)
3. ADMM, вариант sharing (для L_1 регуляризации)

В таблице 2 приведено сравнение сложности вычислений используемых методов обучения. Каждая итерация (один проход по выборке) занимает $O(nnz)$ и хорошо подходит для больших разреженных данных. В тоже время, объем данных, передаваемых по сети отличается. При распределенном обучении с усеченным градиентом и L-BFGS по сети передается вектор $\boldsymbol{\beta}$ и вектор весов $\mathbf{w}$ для вычисления средневзвешенного $\boldsymbol{\beta}$. В алгоритме d-GLMNET вектора скалярных произведений $\Delta\boldsymbol{\beta}^T\mathbf{x}_i$, в алгоритме ADMM с техникой sharing - вектор $\mathbf{z}$. Также, методы d-GLMNET и ADMM, основанные на распределенном покоординатном спуске, в принципе не передают вектор $\boldsymbol{\beta}$ между машинами, что удобно при большой размерности $\boldsymbol{\beta}$.

4.2 Использованные наборы данных

В данном разделе описаны численные эксперименты, выполненные с методом d-GLMNET для важного на практике частного случая: решение задачи бинарной классификации с помощью логистической регрессии с L_1 или L_2 регуляризацией. Для численных экспериментов использовались датасеты, приведенные в таблице 3. Датасеты *epsilon*, *webspam*, *dna*

Таблица 3: Характеристики использованных датасетов

dataset	size	#examples (train/validation/test)	#features	nnz (train)	avg nonzeros
epsilon	12 Gb	0.4×10^6 / 0.05×10^6 / 0.05×10^6	2000	8.0×10^8	2000
webspam	21 Gb	0.315×10^6 / 0.0175×10^6 / 0.0175×10^6	16.6×10^6	1.2×10^9	3727
dna	71 Gb	45×10^6 / 2.5×10^6 / 2.5×10^6	800	9.0×10^9	200
yandex_ad	56 Gb	57×10^6 / 2.35×10^6 / 2.35×10^6	35×10^6	5.7×10^9	100

- публичные, они использовались в Pascal Large Scale Learning Challenge 2008 [7]. Датасет *yandex_ad2* - не публичный, содержит коммерческие данные компании Яндекс.

- **epsilon** - Синтетический датасет. Для него использовалась предобработка и разделение на обучение и тест из
  `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`
- **webspam** - Бинарная классификация веб-спама (некачественных веб-страниц)
  `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`
- **dna** [8] - Задача бинарной классификации сплайсинга в генетике. Для него была сделана такая же предобработка, как и в соревновании Pascal Large Scale Learning Challenge (см. `ftp://largescale.ml.tu-berlin.de/largescale/dna/`) и после этого выполнено разделение на обучение и тест.
- **yandex_ad** - Предсказание вероятности клика по поисковой рекламе Яндекса.

Эксперименты выполнялись на кластере, состоящем из серверов Intel(R) Xeon(R) CPU E5-2660 2.20GHz, 32 GB RAM, соединенных гигабитным Ethernet. Для всех датасетов вычисления проводились на 16 серверах, на каждом сервере запускался один экземпляр сравниваемой программы.

4.3 Адаптивное изменение параметра μ

Адаптивное изменение параметра μ в алгоритме `d-GLMNET` предназначено для обеспечения разреженности решения (см. раздел 3.2). Для иллюстрации эффективности данного

[7] `http://largescale.ml.tu-berlin.de/`

[8] По техническим причинам этот интересный датасет использовался только в одном численном эксперименте

Таблица 4: Параметры алгоритмов

Датасет	λ_1	**Vowpal Wabbit**		**ADMM**	Q^*
		η	p	ρ	
epsilon	2	0.5	0.5	0.25	1.05959×10^5
webspam	1 / 64	0.1	0.5	0.0625	2.00582×10^3
yandex_ad	2	0.5	0.5	1	1.69864×10^7

подхода было проведено обучение логистической регрессии с L_1 регуляризацией на датасете *yandex_ ad*, построены графики зависимости субоптимальности целевой функции, качества на тесте, а также разреженности решения от времени - рис. 8. Графики показывают, что адаптивное изменение μ немного ускоряет сходимость алгоритма и скорость достижения качества на тесте, но, в тоже время, разреженность решения становится существенно лучше.

Во всех дальнейших экспериментах с L_1 регуляризацией использовалось адаптивное изменение μ, а в экспериментах с L_2 регуляризацией - постоянное $\mu = 1$ (т.к. разреженность решения не обязательна).

4.4 Эксперименты с L_1 регуляризацией

В численных экспериментах сравнивалось три метода для решения задачи логистической регрессии с L_1 регуляризацией:

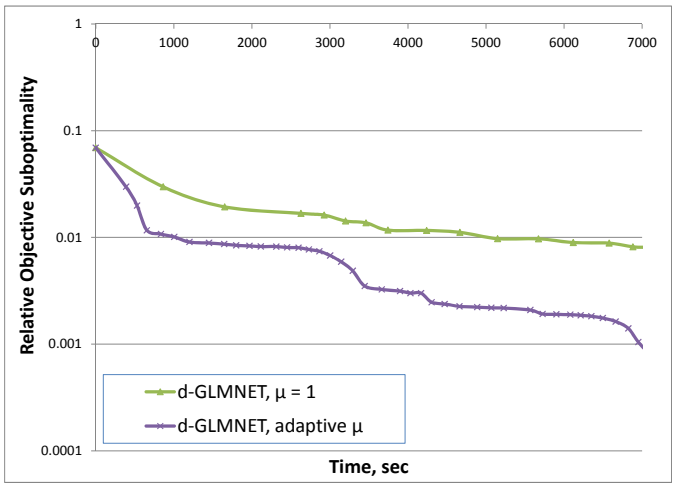
1. Методы `d-GLMNET`, а также вариант с асинхронной балансировкой нагрузки `d-GLMNET-ALB`
2. Распределенное онлайн обучение с усеченным градиентом (см. раздел 2.2.1)
3. Распределенный покоординатный спуск с помощью ADMM (см. раздел 2.2.2)

4.4.1 Численный эксперимент 1

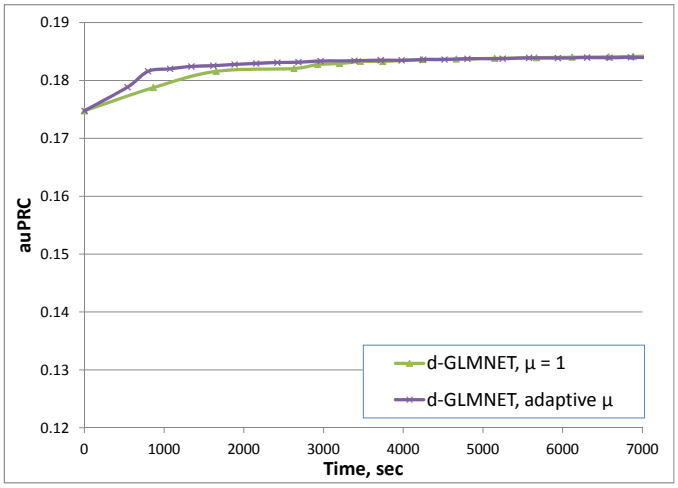
Эксперименты проводились по следующей методике:

1. Сначала для всех датасетов определялся оптимальный параметр регуляризации λ_1, максимизирующий качество на валидационной выборке.
2. После этого приближено находилось оптимальное значение целевой функции Q^* (36) с помощью программы *liblinear* [9] для датасетов *epsilon* и *webspam*, программы `dlr`

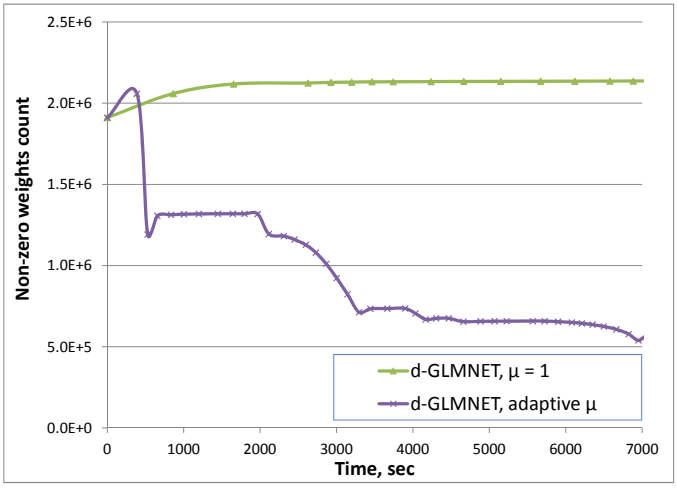
[9] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`



(a) Субоптимальность целевой функции



(b) Качество на тестовой выборке (площадь под кривой Precision-Recall)



(c) Количество ненулевых компонент $\boldsymbol{\beta}$

Рис. 8: Сравнение постоянного $\mu = 1$ и изменяющегося адаптивно μ для датасета *yandex_ad*, L_1 регуляризация

для остальных.

$$Q^* = \min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} \log(1 + \exp(-\boldsymbol{\beta}^T \mathbf{x}_i)) + \lambda_1 \|\boldsymbol{\beta}\|_1 \right\}. \quad (36)$$

3. Для программы Vowpal Wabbit[10] проводился поиск оптимальных гиперпараметров совместно в диапазонах $\eta \in [0.1, 0.5]$, $p \in [0.5, 0.9]$ и выполнялось 50 проходов онлайн-обучения. Выбиралась комбинация гиперпараметров, которая обеспечивала минимальное значение целевой функции (целевая функция вычислялась после каждого прохода).

4. Для алгоритма ADMM перебирался параметр $\rho \in [4^{-3}, \ \ldots \ , 4^3]$. Для дальнейших экспериментов использовалось ρ, минимизирующее целевую функцию за 10 итераций.

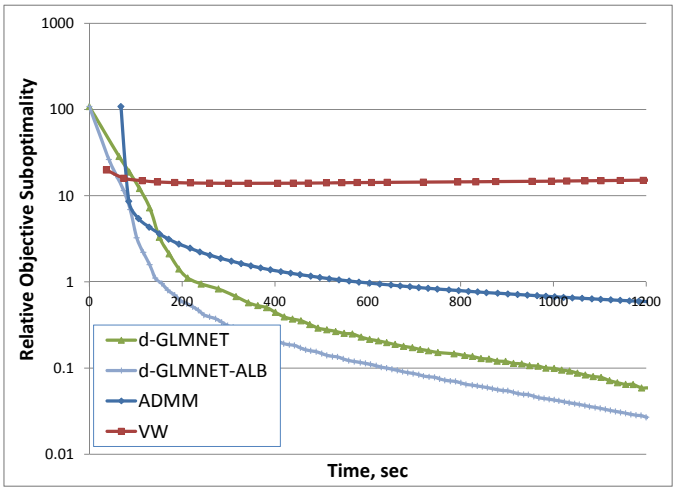
5. Программные реализации алгоритмов `d-GLMNET` и ADMM (в программе `dlr`), онлайн обучение с усеченным градиентом (в программе Vowpal Wabbit) запускались на кластере 9 раз. Для сравнения выбирался запуск с медианным временем выполнения.

Найденные оптимальные гиперпараметры алгоритмов приведены в таблице 4. Заметим, что алгоритм `d-GLMNET` не имеет свободных гиперпараметров (за исключением регуляризации), что упрощает его практическое использование. В процессе работы сравниваемых алгоритмов измерялась точность на тестовом множестве (auPRC - площадь под кривой Precision-Recall), значение целевой функции в виде $(Q - Q^*)/Q^*$ а также количество ненулевых компонент (весов) вектора $\boldsymbol{\beta}$. Результаты приведены на рис. 9, 10, 11.

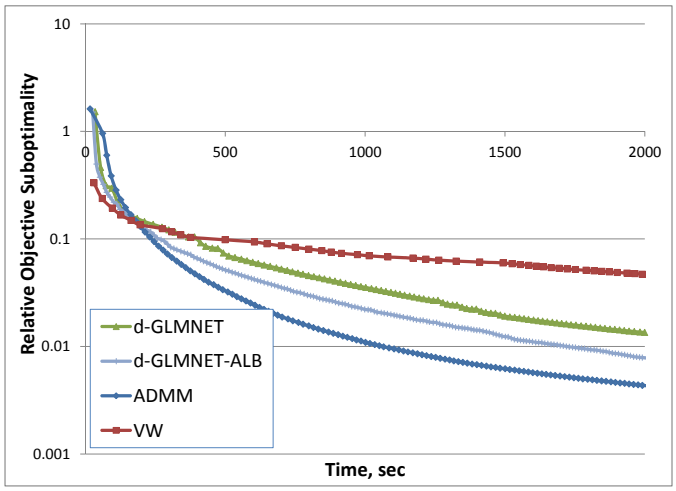
4.4.2 Выводы из эксперимента 1

Рис. 9, 10 показывают, что алгоритм `d-GLMNET` быстрее оптимизирует целевую функцию и достигает качества на тестовой выборке на датасетах *webspam*, *yandex_ad*, чем остальные алгоритмы. Отличительной особенностью этих датасетов является большое число признаков и высокая разреженность. Метод ADMM, в целом, показал себя хорошо, и он был немного быстрее, чем `d-GLMNET` и `d-GLMNET-ALB` на датасете *epsilon*. Vowpal Wabbit достигает такого же или меньшего качества на тесте, но плохо оптимизирует целевую функцию. Численные эксперименты с L_1 регуляризацией показывают, что асинхронная балансировка нагрузки ускоряет или, как минимум, не замедляет алгоритм `d-GLMNET`

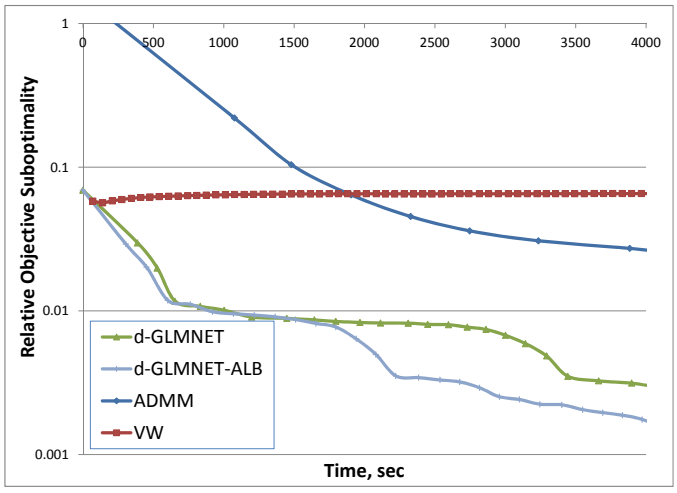
[10] Параметр регуляризации λ (36) связан с аргументом *--l1 arg* программы Vowpal Wabbit соотношением $arg = \lambda/n$, где n - это размер обучающей выборки



(a) webspam

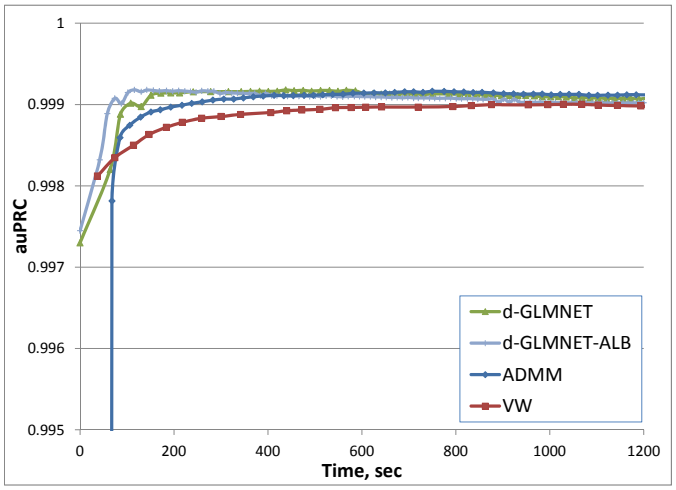


(b) epsilon

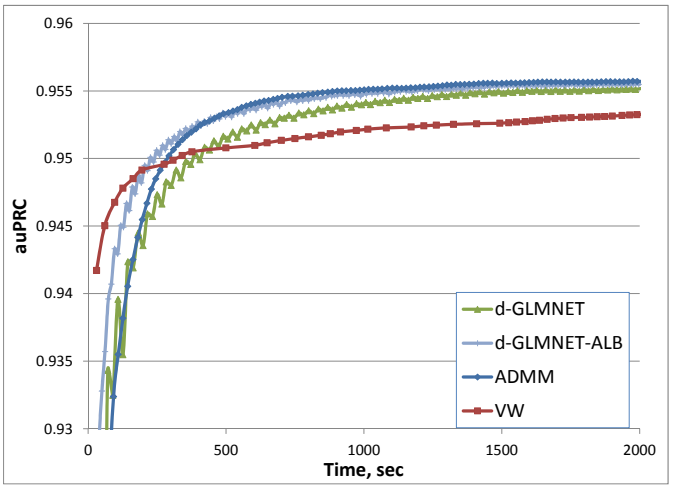


(c) yandex_ad

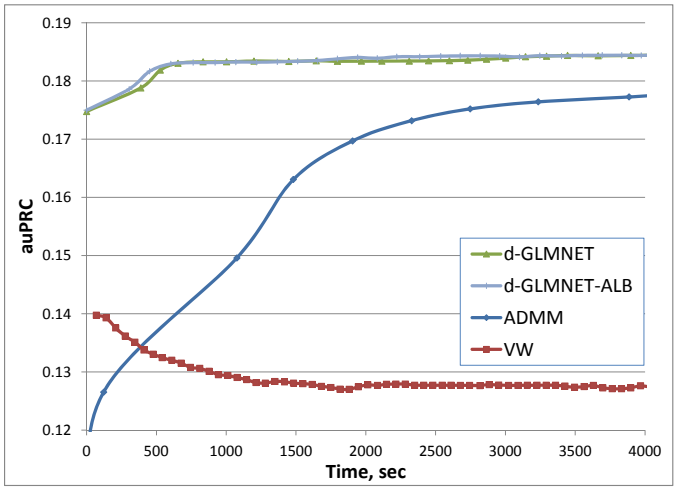
Рис. 9: L_1-регуляризация: субоптимальность целевой функции



(a) webspam

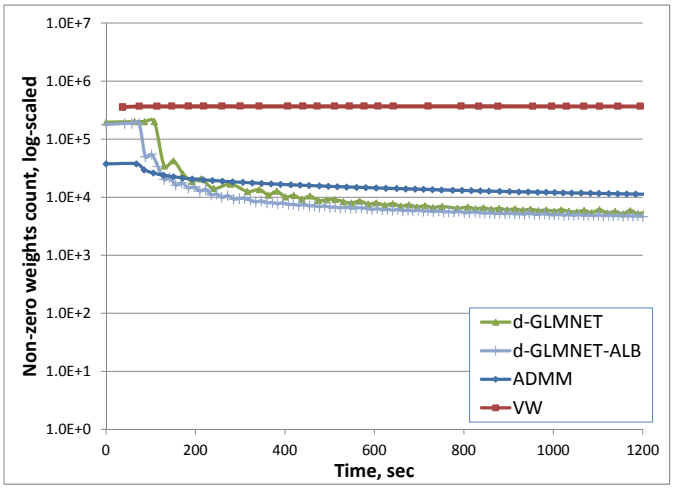


(b) epsilon

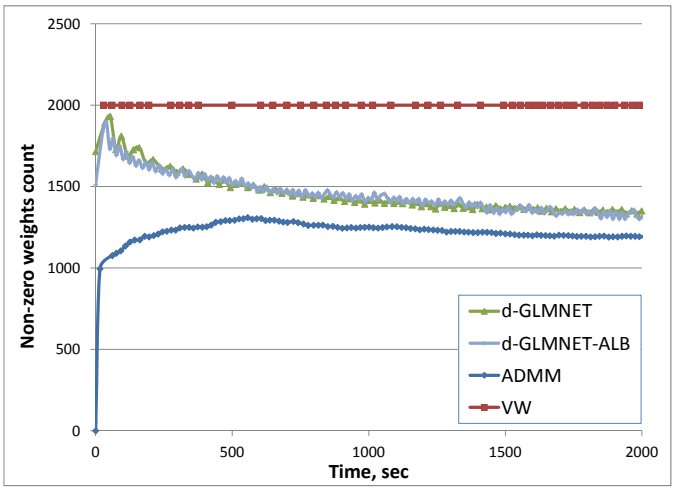


(c) yandex_ad

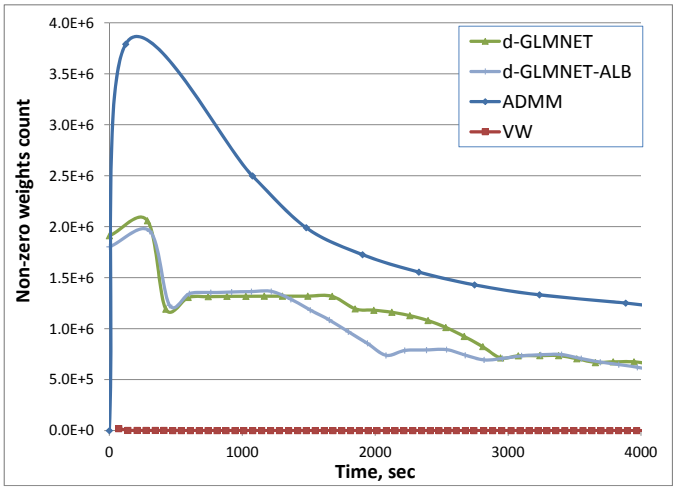
Рис. 10: L_1-регуляризация: качество на тестовой выборке (площадь под кривой Precision-Recall)



(a) webspam



(b) epsilon



(c) yandex_ad

Рис. 11: L_1-регуляризация: количество ненулевых весов $\boldsymbol{\beta}$

Рис. 11 показывает зависимость кол-ва ненулевых весов от времени. Разреженность решения, полученного `d-GLMNET` лучше, чем у ADMM для датасетов *webspam* и *yandex_ad* и немного хуже на *epsilon*. Решения, полученные с помощью Vowpal Wabbit или чрезмерно разреженные или не разреженные совсем, в сравнении с другими алгоритмами.

4.4.3 Численный эксперимент 2

Как показывают результаты эксперимента 1, алгоритм `d-GLMNET-ALB` быстрее оптимизирует целевую функцию логистической регрессии с L_1 регуляризацией для фиксированного параметра регуляризации λ_1, чем алгоритмы ADMM и онлайн обучение с усеченным градиентом на разреженных датасетах высокой размерности. Но онлайн обучение с усеченным градиентом иногда находит решения со степенью разреженности, достаточно сильно отличающейся от `d-GLMNET-ALB`. Поэтому имеет смысл провести сравнение качества алгоритмов для всего пути регуляризации.

Использовалась следующая схема эксперимента:

1. С помощью `d-GLMNET-ALB` вычислялся путь регуляризации для 20 значений λ_1 с помощью Алгоритма 3. Для каждого решения вычислялось количество ненулевых весов и точность на тестовом множестве.

2. Для всех полученных значений $\lambda \in [\lambda_{max}2^{-1}, \lambda_{max}2^{-2}, ..., \lambda_{max}2^{-20}]$ перебирались гиперпараметры онлайн-обучения совместно в диапазонах $\eta \in [0.1, 0.5]$, $p \in [0.5, 0.9]$ и выполнялось 50 проходов онлайн-обучения.

   Для каждой комбинации $(\eta, p, \text{номер прохода})$ вычислялось количество ненулевых весов и точность на тестовом множестве.

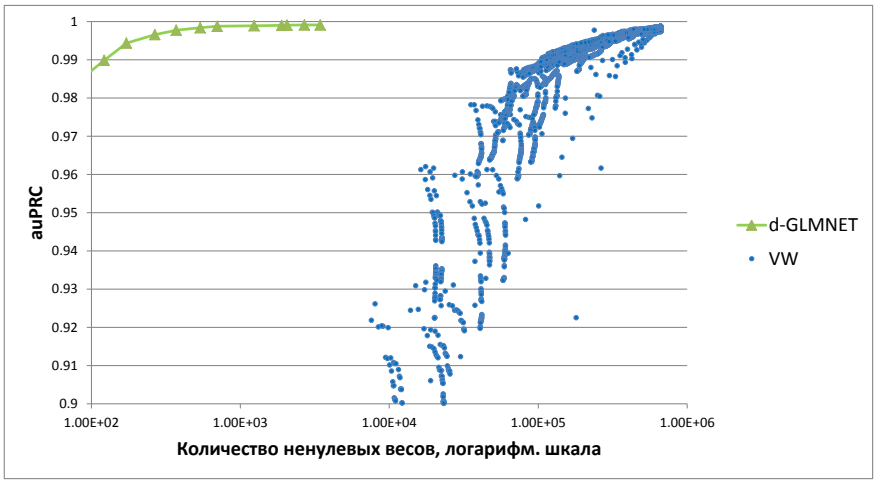
   Результаты показаны на рис. 12.

4.4.4 Выводы из эксперимента 2

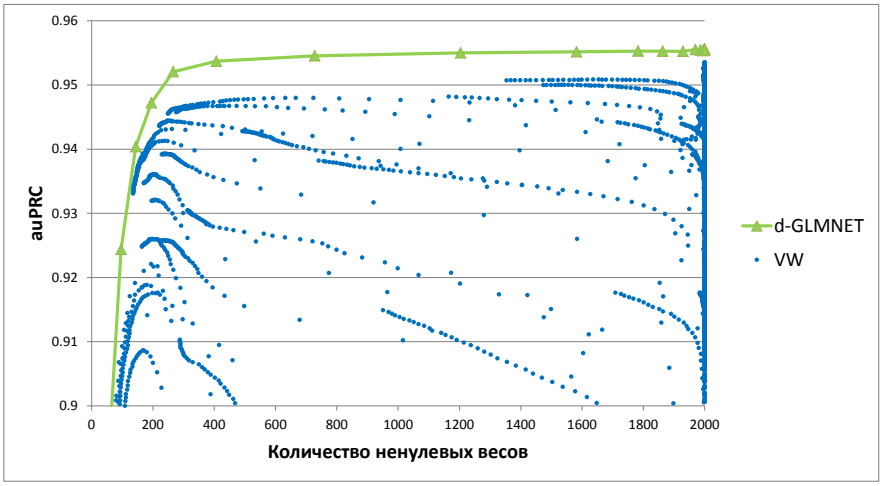
Из рис. 12 видно, что для всех датасетов для каждой степени разреженности алгоритм `d-GLMNET-ALB` обеспечивает более высокое качество классификации на тестовой выборке.

4.4.5 Численный эксперимент 3

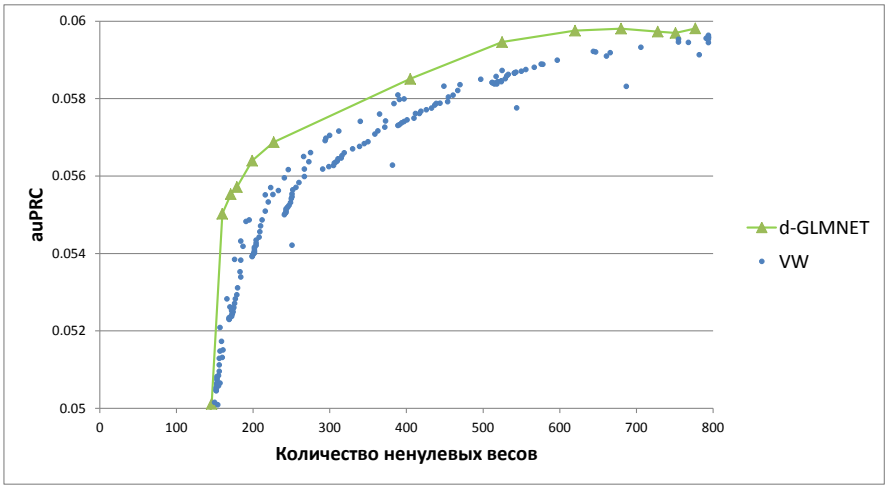
Ускорение от распараллеливания в зависимости от количества используемых машин является важной характеристикой алгоритма. Данный эксперимент был проведен для



(a) webspam



(b) epsilon



(c) dna

Рис. 12: L_1-регуляризация: путь регуляризации, качество на тестовой выборке (площадь под кривой Precision-Recall)

Таблица 5: Параметры алгоритмов

Датасет	λ_2	Q^*
epsilon	0.25	1.04625×10^5
webspam	0.25	1.3740×10^4
yandex_ad	16	1.6554862×10^7

алгоритма `d-GLMNET-ALB`. Ускорение вычислялось по времени, необходимому для достижения значения целевой функции $\leq 1.025Q^*$. Результаты показаны на рис. 13.

4.4.6 Выводы из эксперимента 3

Видно, что для каждого датасета существует некоторый предел достигаемого ускорения. Это связано с тем, что при распараллеливании алгоритма `d-GLMNET-ALB` на все большее число машин блочно-диагональное приближение Гессиана становится менее точным, $\Delta\beta^m$ с разных машин конфликтуют все чаще и поэтому приходится делать маленькие шаги. Также при увеличении числа машин растет время на передачу данных.

4.5 Эксперименты с L_2 регуляризацией

В численных экспериментах сравнивалось два метода для решения задачи логистической регрессии с L_2 регуляризацией:

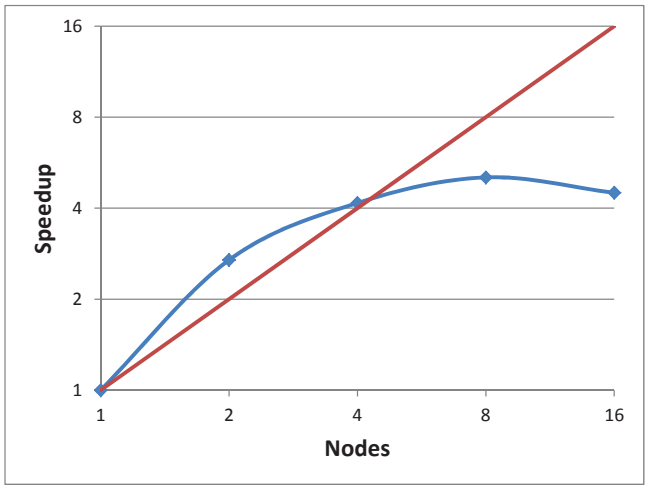
1. Метод `d-GLMNET`, а также модификация с асинхронной балансировкой нагрузки `d-GLMNET-ALB`
2. Онлайн обучение, комбинированное с L-BFGS (см. раздел 1.4.4). Для онлайн обучения выполнялся один проход по выборке ($k = 1$). Использовались реализации онлайн обучения и L-BFGS из проекта Vowpal Wabbit.

4.5.1 Численный эксперимент 1

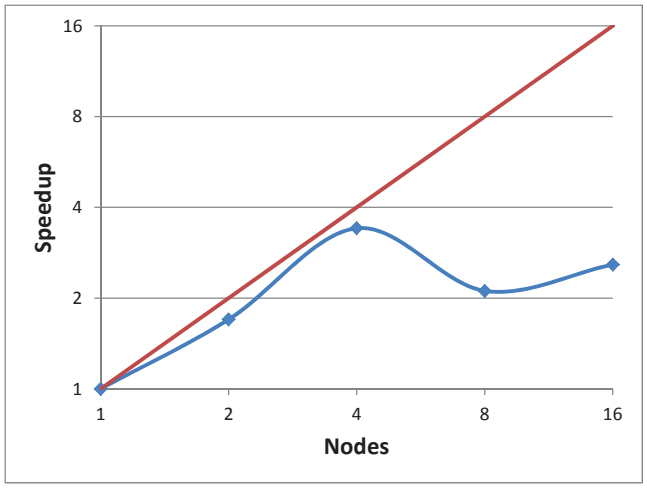
Эксперименты проводились по следующей методике:

1. Сначала для всех датасетов определялся оптимальный параметр регуляризации λ_2 по валидационной выборке.
2. После этого приближенно находилось оптимальное значение целевой функции Q^* (37) с помощью программы *liblinear* [11] для датасетов *epsilon* и *webspam*, программы

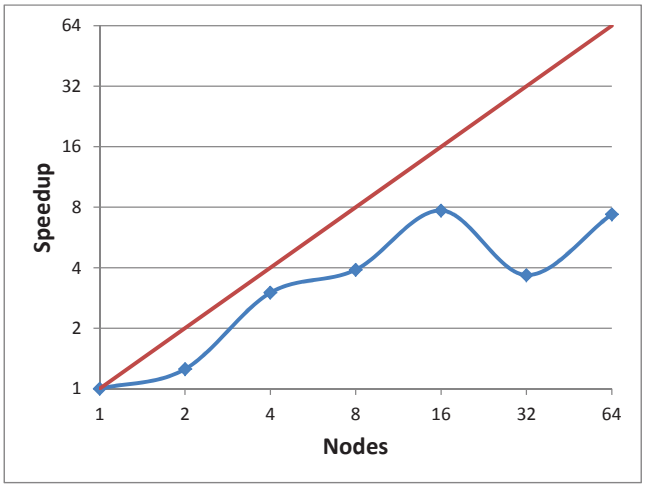
[11] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`



(a) webspam



(b) epsilon



(c) yandex_ad

Рис. 13: L_1-регуляризация: ускорение в зависимости от количества машин. Красная линия показывает идеальное (линейное) ускорение.

`dlr` для датасета *yandex_ ad*.

$$Q^* = \min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} \log(1 + \exp(-\boldsymbol{\beta}^T \mathbf{x}_i)) + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2 \right\} \tag{37}$$

3. Программные реализации алгоритмов запускались на кластере 9 раз. Для сравнения выбирался запуск с медианным временем выполнения.

Найденные λ_2 и Q^* приведены в таблице 5. В процессе работы сравниваемых алгоритмов измерялась точность на тестовой выборке (auPRC - площадь под кривой Precision-Recall), значение целевой функции $(Q - Q^*)/Q^*$. Результаты приведены на рис. 14, 15.

4.5.2 Выводы из эксперимента 1

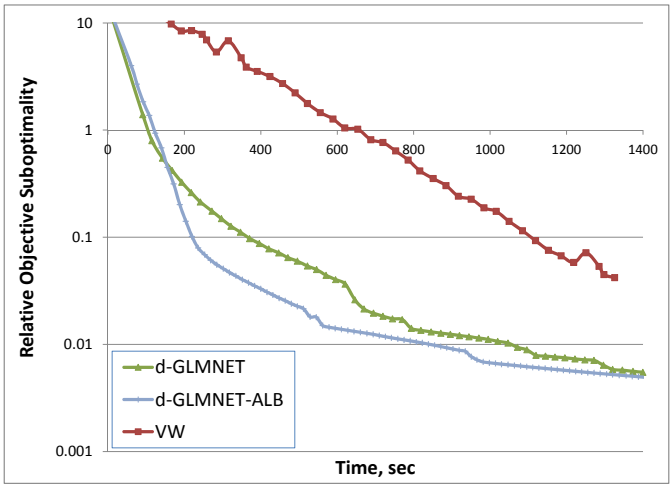
Как видно из рис. 14, 15, методы `d-GLMNET` и `d-GLMNET-ALB` быстрее оптимизирует целевую функцию и достигают лучшего качества классификации на датасетах *webspam* и *yandex_ ad*. Отличительной особенностью этих датасетов является большое число признаков и высокая разреженность. На датасете *epsilon* с относительно небольшим числом признаков, который не является разреженным, методы `d-GLMNET` и `d-GLMNET-ALB` проигрывают с сравнении с онлайн обучением, комбинированным с L-BFGS. Численные эксперименты с L_2 регуляризацией показывают, что асинхронная балансировка нагрузки ускоряет или, как минимум, не замедляет алгоритм `d-GLMNET`.

4.5.3 Численный эксперимент 2

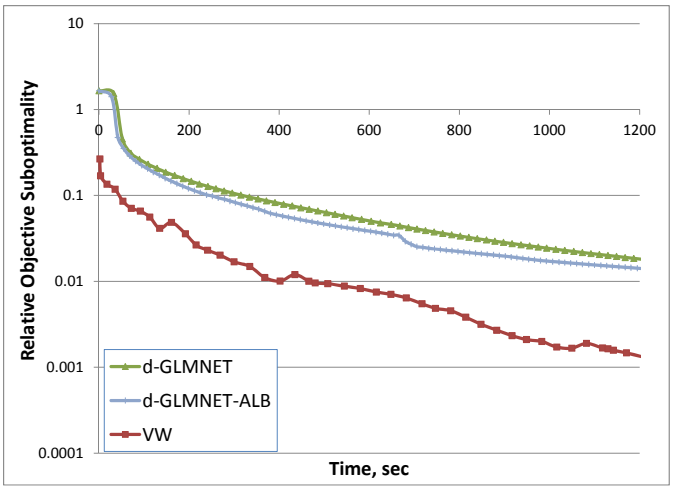
Во втором численном эксперименте изучалось ускорение алгоритма `d-GLMNET-ALB` в зависимости от количества используемых машин. Программные реализации алгоритмов запускались на $1, 2, 4, ..., 16$ машинах на датасетах *webspam* и *epsilon* и на $1, 2, 4, ..., 64$ машинах на датасете *yandex_ ad*. Ускорение вычислялось по времени, необходимому для достижения значение целевой функции $\leq 1.025Q^*$.

4.5.4 Выводы из численного эксперимента 2

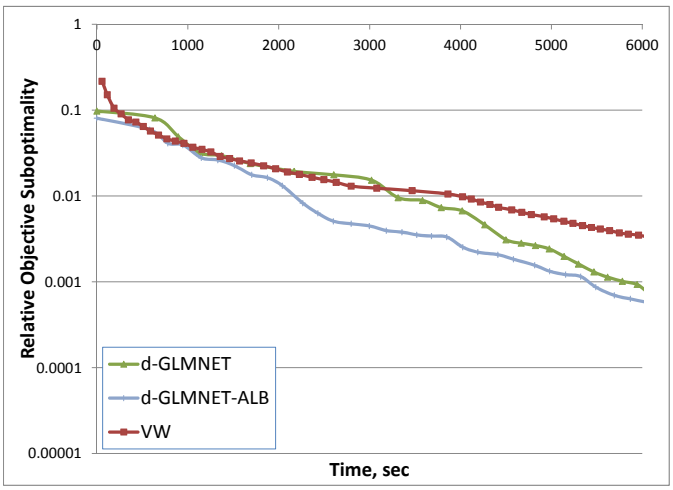
Как показывают рис. 16, алгоритм `d-GLMNET-ALB` ускоряется при увеличение числа машин, достигая определенного лимита. В целом, графики ускорения аналогичны случаю L_1 регуляризации.



(a) webspam

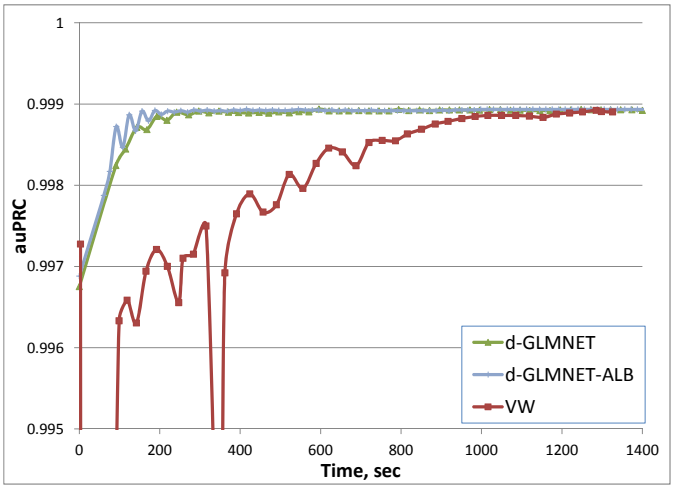


(b) epsilon

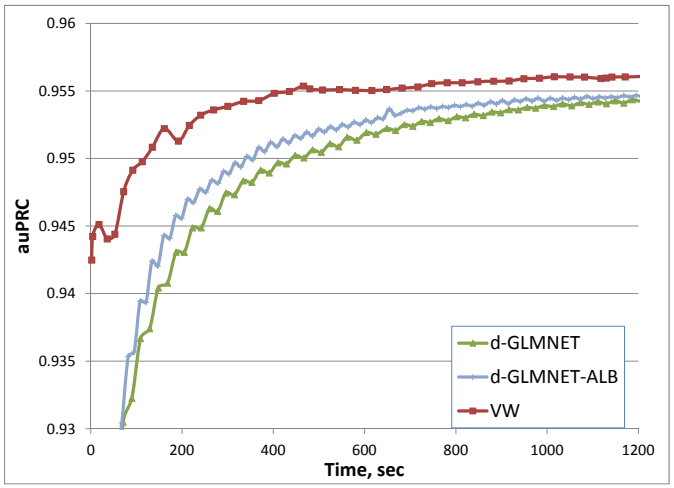


(c) yandex_ad

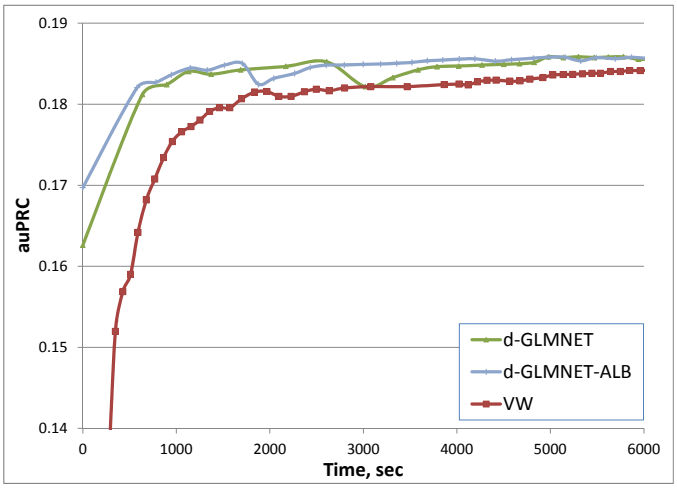
Рис. 14: L_2-регуляризация: субоптимальность целевой функции



(a) webspam

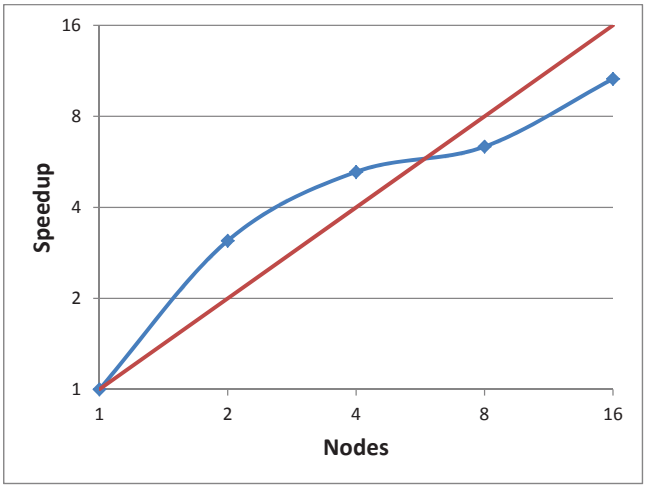


(b) epsilon

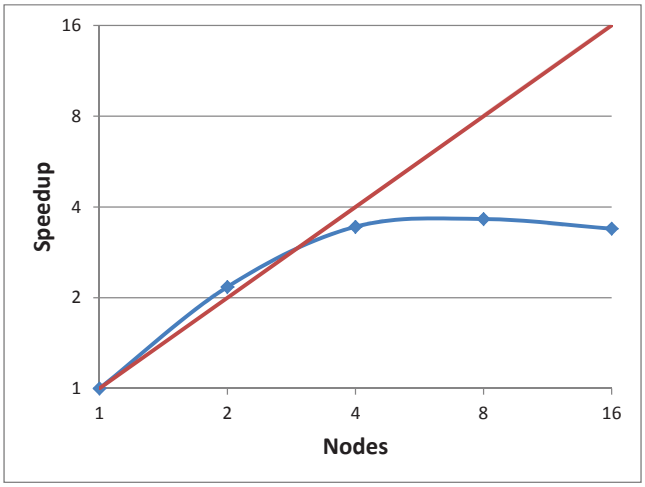


(c) yandex_ad

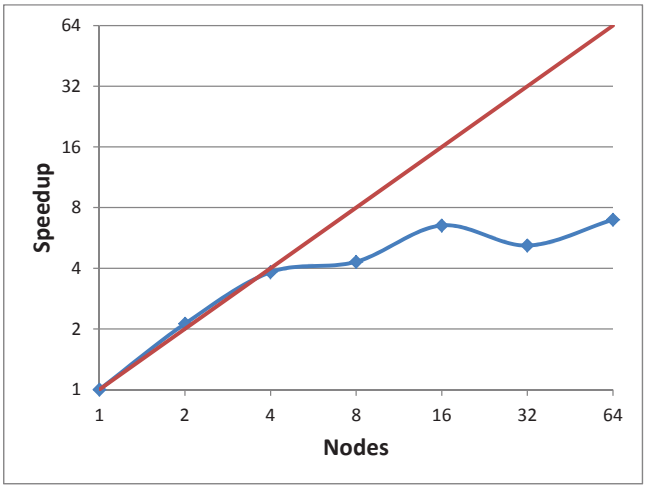
Рис. 15: L_2-регуляризация: качество на тестовой выборке (площадь под кривой Precision-Recall)



(a) webspam



(b) epsilon



(c) yandex_ad

Рис. 16: Ускорение в зависимости от количества машин. Красная линия показывает идеальное (линейное) ускорение.

5 Комбинирование логистической регрессии и деревьев решений для предсказания вероятности клика в поисковой рекламе

При оптимизации работы *поисковых систем* (ПС) в интернете возникает большое число задач, для решения которых нужно выполнять машинное обучения на больших выборках.

Основная цель поисковой системы - предоставлять ответ по запросу пользователя, сформулированному в виде короткой фразы - «запроса». Основной ответ ПС - это ссылки на веб-страницы, содержащие информацию, удовлетворяющую текущий интерес пользователя. Такие веб-страницы называются *релевантными*. Построение качественной поисковой системы - сложная задача, требующая как большого объема программного обеспечения, так и разработки качественных математических методов для его эффективной работы. Основные поисковые системы (Яндекс, Google, Bing) в настоящий момент выдают по запросу пользователя два основных типа ответа: ссылки на релевантные веб-страницы и *поисковую рекламу*. Ссылки на веб-страницы, как правило, сопровождаются небольшими текстовыми описаниями (сниппеты). Поисковая реклама также является текстовой. Пример страницы с результатами поиска системы Яндекс приведен на рис. 17. Поисковая система Яндекс показывает большую часть рекламных объявлений справа от результатов поиска, а часть объявлений - над поиском (спецразмещение).

ПС выбирает для показа пользователю наиболее релевантные ссылки, пытаясь ответить на его информационный запрос. Поэтому ссылки ранжируются по предсказанной с помощью машинного обучения релевантности. В тоже время, поисковая реклама является основным источником доходов ПС. Поэтому необходимо отбирать и ранжировать поисковую рекламу учитывая, с одной стороны, уместность ее показа, а с другой стороны - ожидаемую прибыль ПС.

Наиболее популярный способ монетизации (получения дохода) поисковой рекламы - модель «pay-per-click». В рамках этой модели рекламодатель платит в том случае, когда пользователь кликает на рекламу и, следовательно, переходит по ссылке на сайт рекламодателя. Существуют и другие модели - «pay-per-impression» (оплачивается показ) и «pay-per-action» (оплачивается действие, выполненное на сайте рекламодателя). В данной работе рассматривается только модель «pay-per-click».

В рамках модели «pay-per-click» для организации эффективной продажи рекламы

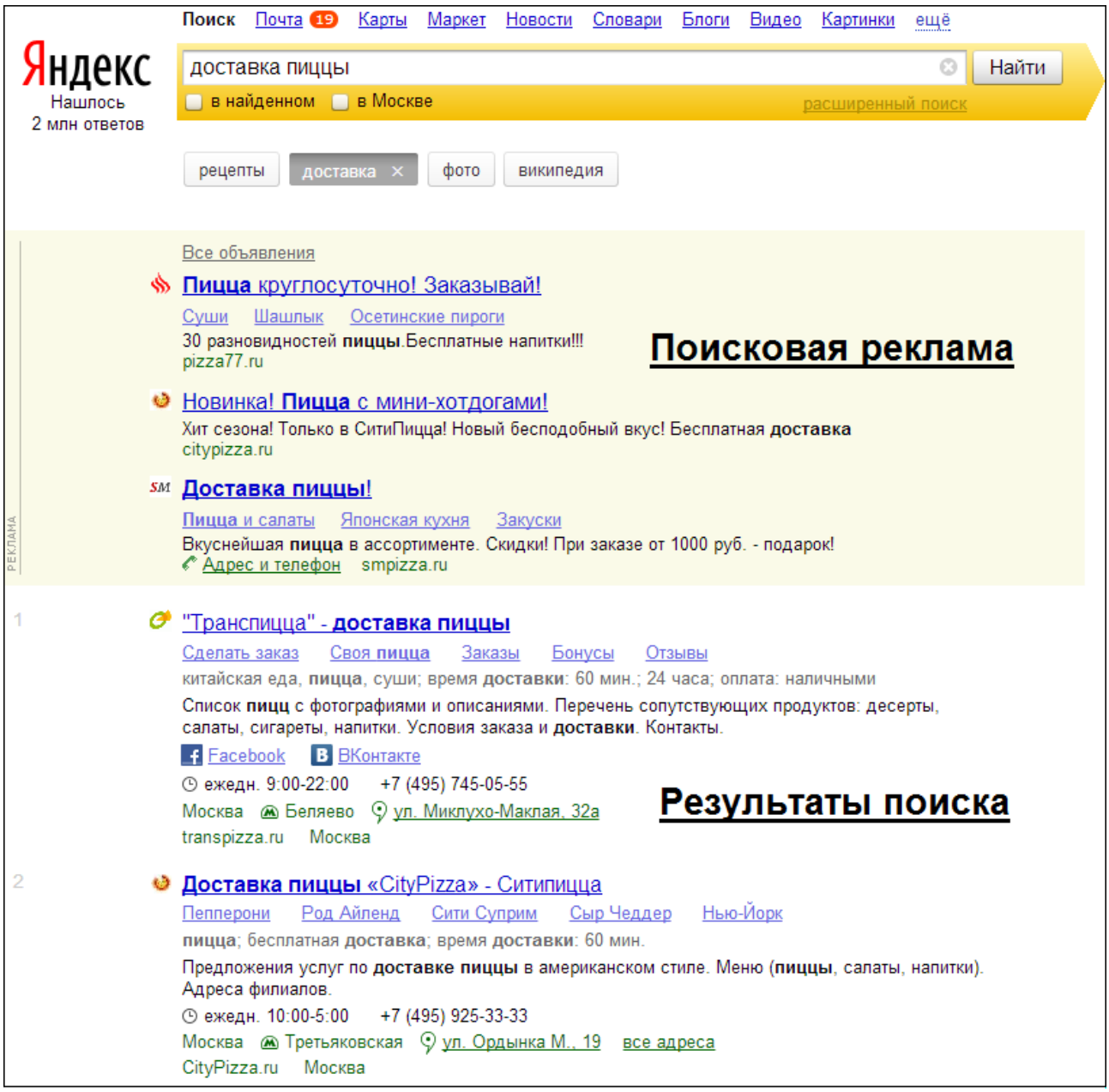


Рис. 17: Страница результатов поисковой системы Яндекс

нужно уметь оценивать вероятность клика. Данную величину обычно называют CTR (click-through ratio). Рекламодатель сообщает поисковой системе свою ставку - Bid, т.е. максимальную плату за клик. Таким образом произведение

$$CTR \times Bid$$

можно рассматривать как оценку математического ожидания прибыли ПС от показа рекламного объявления. Предсказание вероятности клика является наиболее важной проблемой в онлайн рекламе, решаемой с помощь машинного обучения. Алгоритмические и математические проблемы онлайн рекламы подробно освещены в [88].

5.1 Алгоритм отбора объявлений для показа в ПС Яндекс

Отбор объявлений для показа объявлений рядом с результатами поиска происходит с помощью проведения аукциона по *ключевым фразам*. Аукцион по ключевым фразам описан в [89, 88]: рекламодатель выбирает несколько ключевых фраз, описывающих продаваемй продукт или услугу. Когда пользователь вводит запрос, ПС сравнивает его со списком всех ключевых фраз и отбирает близкие по смыслу. Существуют различные правила определения «близости»: точное совпадение, включение, совпадение с точностью до синонимов, и т.д. Рекламодатель настраивает разрешенные методы определения «близости» по своему усмотрению. В процессе сравнения слова запроса и ключевой фразы приводятся к нормальной форме (лемматизируются). Основным методом сравнения запроса с ключевой фразы в ПС Яндекс является включение ключевой фразы в запрос. Например, по запросу **[заказ пиццы в москве недорого]** могут быть показаны объявления с ключевыми фразами **[заказ пиццы в москве]** или **[пицца недорого]**.

Обозначим через CTR_i предсказанную вероятность клика по объявлению, Bid_i - указанную рекламодателем ставку и $CPM_i \stackrel{\text{def}}{=} CTR_i \times Bid_i$. ПС отбирает объявления для показа с помощью Алгоритма 17.

Таким образом, качество прогноза CTR_i играет решающую роль при отборе объявлений для показа. Выбор правильных объявлений увеличивает прибыль рекламной системы и качество ее работы с точки зрения пользователя, ведь объявления с высокой вероятность клика более интересны пользователю. Заметим, что сумма, которую рекламодатель заплатит в случае клика по его объявлению не превышает его ставки и определяется ставками и CTR конкурентов. Такой алгоритм называется обобщенным аукционом второй цены [89]. Пороги T_i позволяют настраивать баланс между агрегированными характеристика-

Алгоритм 17: Алгоритм отбора объявлений для показа

Вход : База всех объявлений с указанными ставками Bid_i, формула вычисления CTR

1 Отобрать объявления-кандидаты на показ, в соответствии с правилами вычисления близости запроса и ключевой фразы

2 Вычислить для каждого объявления прогноз вероятности клика CTR_i

3 Оставить только объявления, у которых $CPM_i \geq T_i$, где T_i - это порог, зависящий как от объявления, так и от запроса

4 Сортировать объявления по убыванию CPM_i и отобрать не более 3 объявлений с максимальным CPM_i

5 Сортировать объявления по убыванию Bid_i

6 Вычислить платеж рекламодателя c_i как наименьшую ставку, достаточную для того, чтобы он занимал ту же позицию при показе при отборе данным алгоритмом

Возврат: список объявлений для показа

ми рекламы : средний CTR, средняя стоимость клика, средняя релевантность. Методика подбора порогов T_i описана в [90].

5.1.1 Задача предсказания вероятность клика

Существует два основных подхода к предсказанию вероятности клика. Первый подход - это предсказывать вероятность клика для пары «ключевая фраза - объявление». В данном подходе выбираются пары «ключевая фраза - объявление», по которым объявление показывалось достаточное число раз, чтобы можно было вычислить эмпирическую частоту клика. Такие пары «ключевая фраза - объявление» составляют обучающую выборку. Данный подход имеет два недостатка. Во-первых, приходится обучаться на смещенной обучающей выборке, в которой нет пар «ключевая фраза - объявление» с малым числом показов, хотя они являются наиболее интересными для прогноза. Во-вторых, данный подход не позволяет учитывать весь контекст показа объявления: информацию о пользователе и запросе. В рамках данного подхода предсказание вероятности клика выполняется с помощью логистической регрессии [91] или деревьев решений [92].

Второй подход - это использование «сырых данных» непосредственно из лога показов рекламы. В этом случае переменная для предсказания - это $y_i \in \{0, 1\}$, где 1 означает,

что клик был, 0 - не было. Вектор независимых переменных $\mathbf{x}_i$ содержит всю доступную информацию для прогноза: параметры пользователя, запроса, объявления, время, и т.д. В рамках этого подхода вероятность клика предсказывается с помощью бустинга решающих правил [93], графических моделей [94, 95], бустинга решающих деревьев [26], нейросетей [96]. Данный подход не имеет вышеупомянутых недостатков, но он требует обучения на большей выборке. В настоящей работе используется второй подход к предсказанию вероятности клика.

Для обучения модели использовался лог показов рекламы ПС Яндекс. Лог показов рекламы содержит факты показов рекламы пользователям с отметками был клик или нет, а а также с большим количеством параметров пользователя, запроса и объявления. Таким образом, пары $(\mathbf{x}_i, y_i)$ могут быть получены из лога показов. Как уже было сказано выше, y_i описывает факт клика, а $\mathbf{x}_i$ - это вектор из 54 вещественнозначных признаков, разбитых на следующие группы:

- **Параметры пользователя**: описывают паттерны поведения пользователя и его интересы;
- **Контекст**: дата и время, номер страницы выдачи поисковой системы;
- **Запрос**: включает числовые факторы, такие как количество букв в запросе, количество слов в запросе и т.п.;
- **Ключевая фраза, Заголовок объявления, Текст объявления**: количество слов в ключевой фразе, заголовке, тексте объявления, доля заглавных букв в заголовке объявления, и т.п.
- **Запрос** $\times$ **Ключевая фраза** : количество слов, входящих с запрос но не входящих в ключевую фразу;
- **Запрос** $\times$ **(Заголовок объявления, Текст объявления)**: TF*IDF запроса и текста, TF*IDF запроса и заголовка и т.п.;
- **Рекламодатель**: статистика кликов и показов домена рекламодателя, а также эмпирическая частота кликов по домену рекламодателя (клики / показы);
- **Кликовая статистика пары «ключевая фраза - объявление»**: статистика кликов и показов пары «ключевая фраза - объявление», а также эмпирическая частота кликов (клики показы);

5.1.2 Предсказание вероятности клика с помощью бустинга деревьев решений

Предсказание вероятности клика выполняется в ПС Яндекс с помощью алгоритма MatrixNet [97, 98]. Алгоритм MatrixNet является вариантом бустинга деревьев решений (Gradient Boosting Machine, GBM) [99] с использованием стохастического бустинга [100]. Данный алгоритм очень универсален и успешно применяется для решения разнообразных задач классификации и регрессии. Его основные особенности:

- высокая устойчивость к переобучению;
- автоматический учет взаимодействия высокого порядка между входными переменными;
- возможность приближать разрывные функции;
- в большинстве случаев не требуется предварительной обработки входных переменных.

MatrixNet имеет несколько эвристических улучшений относительно оригинального метода GBM:

- используются «забывчивые» (oblivious) деревья решений, в которых на одном уровне работает одно решающее правило;
- регуляризация значений в листьях вместо ограничений на минимальное число объектов в листе;
- высота дерева постепенно увеличивается в процессе итераций от 1 до заданного заранее значения по формуле

$$height(h(\mathbf{x}, a_m)) = \min(1 + [m/10], H_{max})$$

  где m - номер итерации. В численных экспериментах параметр H_{max} всегда устанавливался равным 10.

MatrixNet позволяет работать с различными функциями потерь: квадратичной, логистической и специальными функциями потерь для задач ранжирования.

5.1.3 Использование категориальных признаков и признаков на основе текстов

Кроме упомянутых ранее признаков представляет большой интерес применение следующих разреженных признаков высокой размерности:

- Категориальные (номинальные) признаки: UserID, BannerID, DomainID, OrderID, PhraseID и т.п.
- Текстовые признаки типа "мешок слов"(bag of words): текст запроса, текст ключевой фразы, заголовок баннера, текст баннера.

Типичным представлением этих признаков в алгоритмах машинного обучения является one-hot кодировка. Например, если UserID может принимает N различных уникальных значений, то каждому обучающему примеру ставится в соответствие ставится вектор $\mathbf{z}^{user} \in \mathbb{R}^N$. В векторе $\mathbf{z}^{user}$ все компоненты равны нулю, кроме одной - соответствующей значению UserID в данном примере, она равна 1.

Аналогичное кодирование используется для "мешков слов". Например, если запрос пользователя - **[обзор моделей цифровых камер]**, то ему соответствует вектор $\mathbf{z}^{query} \in \mathbb{R}^D$, где D - размер словаря. Все элементы вектора $\mathbf{z}^{query}$ равны нулю, кроме соответствующих словам **обзор, модель, цифровой, камера** которые равны 1. Заметим, что слова запроса были *лемматизированы*, т.е. приведены в нормальную форму. Вес слова не обязательно равен 1, он может зависеть, например, от частоты слова [101].

Можно рассматривать также квадратичные комбинации признаков. Для рассмотренных выше векторов $\mathbf{z}^{user} \in \mathbb{R}^N$ и $\mathbf{z}^{query} \in \mathbb{R}^D$ их квадратичной комбинацией будет вектор $\mathbf{z}^{user,query} \in \mathbb{R}^{ND}$ такой что

$$z_{kl}^{user,query} = z_k^{user} z_l^{query}$$

Таким образом, можно составить вектор признаков на основе категориальных признаков и признаков типа "мешок слов а также их квадратичных комбинаций.

$$\mathbf{z} = (\mathbf{z}^{user}, \mathbf{z}^{query}, \ \dots \ , \mathbf{z}^{user,query}, \ \dots \ )$$

Разреженный вектор признаков $\mathbf{z}$ будет иметь высокую размерность.

5.1.4 Комбинирование логистической регрессии и бустинга деревьев решений

Как показано в статье [26], предсказание вероятности клика с помощью MatrixNet эффективно работает если входные признаки - вещественнозначные и их не слишком много

- несколько сотен. Если пространство входных признаком имеет очень высокую размерность, то перебор всех признаков, необходимый для построения одного дерева решений будет занимать слишком много времени. В алгоритме бустинга над решающими деревьями для достижения хорошего качества прогноза нужно подбирать значительное число деревьев - порядка тысячи [26]. В тоже время, для задач высокой размерности с большими обучающими выборками эффективно работают обобщенные линейные модели с регуляризацией, в частности логистическая регрессия. Поэтому логично постараться объединить оба подхода: использование деревьев решений для небольшого числа веществennозначных признаков и логистической регрессия для подпространства бинарных признаков высокой размерности.

Данную комбинацию можно осуществить двумя основными способами:

1. Использовать совместную вероятностную модель [27, 102]:

$$P(y = +1|\mathbf{x}, \mathbf{z}) = \frac{1}{1 + \exp(-\sum_{m=1}^{M} \rho_m h(\mathbf{x}, a_m) - \boldsymbol{\beta}^T \mathbf{z})}$$

где $h(\mathbf{x}, a_m)$ - это дерево решений с параметрами a_m. Данная модель подбирается в два этапа с помощью максимизации правдоподобия на обучающей выборке

$$LL = \sum_{i=1}^{n} ([y_i = +1] \log(P(y = +1|\mathbf{x}_i, \mathbf{z}_i) + [y_i = -1] \log(1 - P(y = +1|\mathbf{x}_i, \mathbf{z}_i))$$

Сначала фиксируется $\boldsymbol{\beta} = 0$ и подбирается слагаемое $\sum_{m=1}^{n} \rho_m h(\mathbf{x}, a_m)$ с помощью метода MatrixNet (см. раздел 5.1.2). Обозначим прогноз MatrixNet на обучающем примере i через

$$s_i = \sum_{m=1}^{M} \rho_m h(\mathbf{x}_i, a_m)$$

На втором этапе подбираются параметры $\boldsymbol{\beta}$, что сводится к обучению логистической регрессии. К логистической регрессии для улучшения точности прогноза добавляется регуляризация $R(\boldsymbol{\beta})$

$$P(y_i = +1|\mathbf{z}_i) = \frac{1}{1 + \exp(-s_i - \boldsymbol{\beta}^T \mathbf{z}_i)}$$

$$LL(\boldsymbol{\beta}) = \sum_{i=1}^{n} ([y_i = +1] \log(P(y_i = +1|\mathbf{z}_i)) + [y_i = -1] \log(1 - P(y_i = +1|\mathbf{z}_i)))$$

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left(LL(\boldsymbol{\beta}) + R(\boldsymbol{\beta})\right) \tag{38}$$

Фактически логистическая регрессия обучается на остатках от MatrixNet.

Отметим, что при использовании данного подхода при обучении MatrixNet всегда приходится одновременно производить обучение и логистической регрессии. Оказывается, что это неудобно с практической точки зрения, т.к. операция обучения MatrixNet производится достаточно часто с исследовательскими целями - тестирование параметров базовых регрессий, тестирование новых входных сигналов и т.д.

2. Альтернативный подход состоит в том, чтобы обучить логистическую регрессию независимо

$$P(y_i = +1|\mathbf{z}_i) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^T \mathbf{z}_i)}$$

после чего использовать ее прогноз $P(y|\mathbf{z})$ как входной признак в MatrixNet, причем MatrixNet обучается на другой выборке.

Такой подход оказывается более удобным на практике. После того, как логистическая регрессия обучена, ее прогнозы могут быть вычислены на любой другой выборке, на которой производятся эксперименты с методом MatrixNet.

5.2 Численные эксперименты. Применение метода `d-GLMNET`

В ходе численных экспериментов было произведено сравнение двух методов обучения логистической регрессии на разреженном датасете. Параметры обучающей и тестовой выборок:

- 69×10^6 примеров
- Признаки: GenPos (позиция объявления в блоке), OrderD (ид. рекламной кампании), SearchQueryLemmaH (лемматизированные слова запроса), BannerBMCategoryID (категория баннера), PhraseID (ид. ключевой фразы). А также квадратичные комбинации: SearchQueryLemmaH × OrderID, SearchQueryLemmaH × BannerBMCategoryID, SearchQueryLemmaH × PhraseID.

  Всего размерность пространства признаков была 185×10^6.

Поскольку пространство признаков имеет большую размерность, то из-за ограничений по объему занимаемой оперативной памяти, возникающих при использовании модели в сервисах Яндекса в реальном времени, необходим *отбор признаков*.

Сравнивалось два метода:

- **Метод, используемый в Яндексе**.

1. Предварительно выполняется отбор признаков по частоте встречаемости в обучающей выборке: оставляются признаки, которые были ненулевыми более чем в n_{min} обучающих примерах. Тестировались параметры фильтрации признаков $n_{min} \in \{2, 3, 4, 5, 7, 10, 15, 30, 60, 120\}$.
2. После этого обучалась логистическая регрессия с L_2 регуляризацией (параметр регуляризации был подобран на отдельной валидационной выборке). Использовался метод L-BFGS, комбинированный с онлайн-обучением, реализация в программе Vowpal Wabbit.

- **Метод** `d-GLMNET`

  Поскольку метод `d-GLMNET` эффективно делает L_1-регуляризацию, то можно одновременно выполнять обучение модели и отбор признаков. Тестировалось два варианта регуляризации:

  1. L_1 регуляризация, $\lambda_1 \in \{2^0, 2^1, \ldots, 2^4\}$
  2. elastic net регуляризация, $\lambda_1 \in \{0, 2^{-1}, 2^0, \ldots, 2^3\}, \lambda_2 = 16$

Обучение выполнялось на 32 машинах Intel(R) Xeon(R) CPU E5-2660 2.20GHz, 32 GB RAM, соединенных гигабитным Ethernet.

Каждый вариант обучения (изменялся способ фильтрации признаков или регуляризация) изображен на рис. 18 в виде точки. По оси OX отложено кол-во ненулевых весов решения, по оси OY - ошибка не тестовой выборке - Negative Log-Likelihood.

Как следует из рис. 18, метод `d-GLMNET` показывает наилучшие результаты, в том смысле, что для фиксированной степени разреженности решения, ошибка на тестовой выборке минимальна. Вариант обучения, использовавшийся в Яндексе на момент численного эксперимента - фильтрация с $n_{min} = 5$. Таким образом, выбирая параметры elastic net регуляризации можно найти варианты, в которых относительно использующегося в Яндексе метода:

- Ошибка такая же, но решение в 7 раз более разреженное: 26×10^6 против 3.7×10^6 ненулевых компонент $\boldsymbol{\beta}$.
- Ошибка меньше на 0.26%, что является хорошим улучшением [26].

Разреженность решения важна на практике из-за ограниченности памяти серверов, на которых модель выполняет прогнозы в реальном времени. Время обучения составляло 2-3 ч. при различных параметрах для обеих программ.

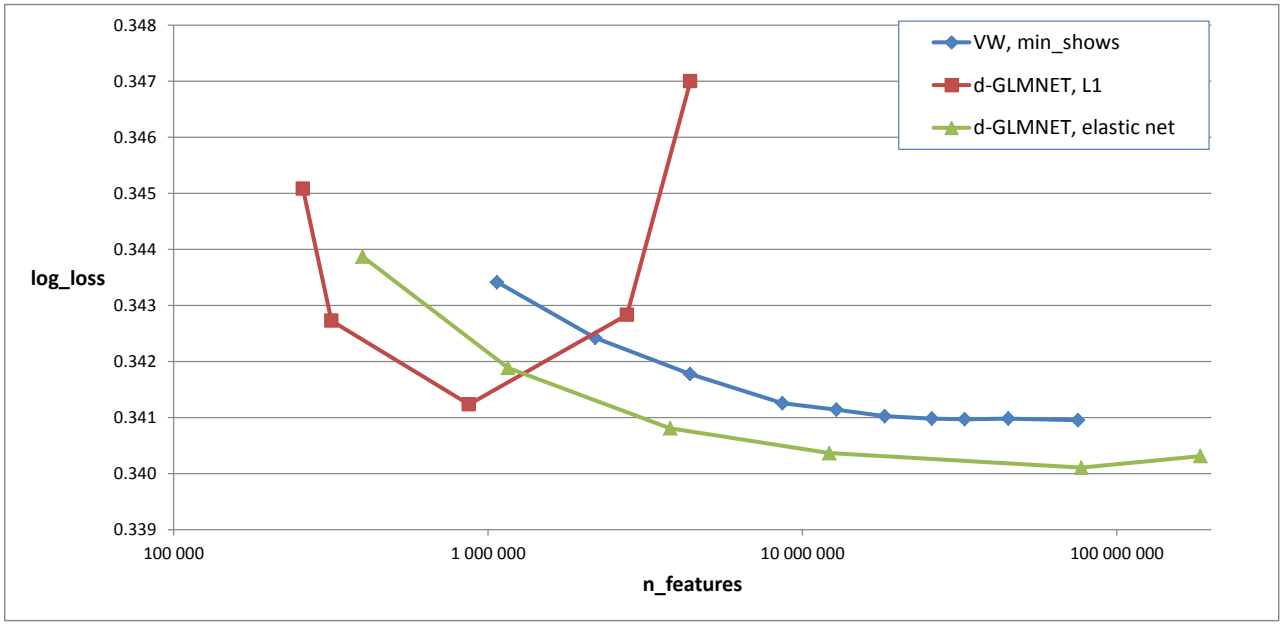


Рис. 18: Сравнение разреженности и ошибки на тесте методов обучения логистической регрессии

Заключение

Основные результаты диссертации:

1. Предложен новый метод минимизации функций риска обобщенных линейных с регуляризацией “elastic net” - `d-GLMNET`;
2. Получены достаточные условия сходимости метода `d-GLMNET`;
3. Получены достаточные условия *линейной скорости* сходимости метода `d-GLMNET`;
4. Доказана возможность получать разреженные решения с помощью метода `d-GLMNET` при использовании L_1-регуляризации;
5. Предложен метод “асинхронной балансировки нагрузки” для обеспечения эффективного выполнения при наличии медленных узлов кластера;
6. Проведены численные эксперименты, доказывающие, что метод `d-GLMNET` более эффективен, чем общепринятые методы при работе с разреженными обучающими выборками с высокой размерностью признакового пространства;
7. Проведен численный эксперимент, показывающий что метод `d-GLMNET` более эффективен для решения задачи прогнозирования вероятности клика по рекламе в поисковой системе “Яндекс”, чем использующийся альтернативный метод;
8. Разработана программная реализация метода `d-GLMNET`, которая общедоступна по адресу: `https://github.com/IlyaTrofimov/dlr`.

Приложения

1 Вывод шага метода d-GLMNET

Приведем решение задачи одномерной минимизации функции

$$\underset{\Delta\beta_j}{\operatorname{argmin}} \left\{ L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) + R(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) \right\} \tag{39}$$

для случая регуляризации elastic net

$$R(\boldsymbol{\beta}) = \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{1}{2}\lambda_2 \|\boldsymbol{\beta}\|_2^2$$

Мы имеем:

$$\begin{aligned} L(\boldsymbol{\beta} + \Delta\boldsymbol{\beta}) &= \sum_{i=1}^{n} \ell(y_i, (\boldsymbol{\beta} + \Delta\boldsymbol{\beta})^T x_i) \approx \\ &\approx \sum_{i=1}^{n} \left\{ \ell(y_i, \boldsymbol{\beta}^T x_i) + \frac{\partial \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}} \Delta\boldsymbol{\beta}^T \mathbf{x}_i + \frac{1}{2}(\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2} (\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \right\} = \\ &= C(\boldsymbol{\beta}) + \frac{1}{2} \sum_{i=1}^{n} w_i (z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2 \end{aligned}$$

где

$$w_i = \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2}, \qquad z_i = -\frac{\partial \ell(y_i, \boldsymbol{\beta}^T x_i)/\partial \hat{y}}{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)/\partial \hat{y}^2}, \qquad C(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) - \frac{1}{2} \sum_{i=1}^{n} z_i^2 w_i$$

Также очевидно, что

$$\begin{aligned} \nabla L(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \frac{\partial \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}} \mathbf{x}_i \\ \nabla^2 L(\boldsymbol{\beta}) = H(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2} \mathbf{x}_i \mathbf{x}_i^T \end{aligned}$$

Модифицированное квадратичное приближение

$$L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \Delta\boldsymbol{\beta}^T (\mu(H(\boldsymbol{\beta}) + \nu I)) \Delta\boldsymbol{\beta}$$

будет иметь вид

$$\begin{aligned} L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) &= L(\boldsymbol{\beta}) + \sum_{i=1}^{n} \left\{ \frac{\partial \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}} \Delta\boldsymbol{\beta}^T \mathbf{x}_i + \frac{\mu}{2} (\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \frac{\partial^2 \ell(y_i, \boldsymbol{\beta}^T x_i)}{\partial \hat{y}^2} (\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \right\} + \frac{\nu}{2} \|\Delta\boldsymbol{\beta}\|^2 \\ &= L(\boldsymbol{\beta}) + \frac{1}{2} \sum_{i=1}^{n} \left\{ \mu w_i (-\frac{2 z_i}{\mu} (\Delta\boldsymbol{\beta}^T \mathbf{x}_i) + (\Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2) \right\} + \frac{\nu}{2} \|\Delta\boldsymbol{\beta}\|^2 \\ &= \frac{1}{2} \sum_{i=1}^{n} \left\{ \mu w_i (\frac{z_i}{\mu} - \Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2 \right\} + C'(\boldsymbol{\beta}) + \frac{\nu}{2} \|\Delta\boldsymbol{\beta}\|^2 \end{aligned}$$

где $C'(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) - \frac{1}{2}\sum_{i=1}^{n}\frac{z_i^2 w_i}{\mu}$. Обозначим $\widehat{\boldsymbol{\beta}} = \boldsymbol{\beta} + \Delta\boldsymbol{\beta}$. Тогда

$$
\begin{aligned}
&L_q^{gen}(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) + \lambda_1\|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{\beta} + \Delta\boldsymbol{\beta}\|_2^2 = \\
&= \frac{1}{2}\sum_{i=1}^{n}\left\{\mu w_i(\frac{z_i}{\mu} - \widehat{\boldsymbol{\beta}}^T\mathbf{x}_i + \boldsymbol{\beta}^T\mathbf{x}_i)^2\right\} + C'(\boldsymbol{\beta}) + \frac{\nu}{2}\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|^2 + \lambda_1\|\widehat{\boldsymbol{\beta}}\|_1 + \frac{1}{2}\lambda_2\|\widehat{\boldsymbol{\beta}}\|_2^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}\left\{\mu w_i(q_i - \widehat{\beta}_j x_{ij})^2\right\} + \frac{\nu}{2}(\widehat{\beta}_j - \beta_j)^2 + \lambda_1|\widehat{\beta}_j|_1 + \frac{1}{2}\lambda_2\widehat{\beta}_j{}^2 \\
&+ C'(\boldsymbol{\beta}) + \sum_{k\neq j}\left\{\lambda_1|\widehat{\beta}_k|_1 + \frac{1}{2}\lambda_2\widehat{\beta}_k{}^2\right\} \\
&= \frac{1}{2}\sum_{i=1}^{n}\left\{\mu w_i(q_i^2 - 2q_i\widehat{\beta}_j x_{ij} + \widehat{\beta}_j{}^2 x_{ij}^2)\right\} + \frac{\nu}{2}(\widehat{\beta}_j{}^2 - 2\beta_j\widehat{\beta}_j + \beta_j^2) + \lambda_1|\widehat{\beta}_j|_1 + \frac{1}{2}\lambda_2\widehat{\beta}_j{}^2 \\
&+ C'(\boldsymbol{\beta}) + \sum_{k\neq j}\left\{\lambda_1|\widehat{\beta}_k|_1 + \frac{1}{2}\lambda_2\widehat{\beta}_k{}^2\right\} \\
&= \frac{1}{2}A\widehat{\beta}_j{}^2 - B\widehat{\beta}_j + \lambda_1|\widehat{\beta}_j| + C''(\boldsymbol{\beta}, \widehat{\boldsymbol{\beta}})
\end{aligned}
$$

Где были введены обозначения

$$
\begin{aligned}
q_i &= \frac{z_i}{\mu} - \sum_{k\neq j}\widehat{\beta}_k{}^T x_{ik} + \boldsymbol{\beta}^T\mathbf{x}_i \\
A &= \mu\sum_{i=1}^{n} w_i x_{ij}^2 + \nu + \lambda_2 \\
B &= \mu\sum_{i=1}^{n} w_i q_i x_{ij} + \nu\beta_j
\end{aligned}
$$

и функция $C''(\boldsymbol{\beta}, \widehat{\boldsymbol{\beta}})$ не зависит от $\widehat{\beta}_j$ и может не учитываться при решении задачи (39). Таким образом, необходимо минимизировать функцию

$$
\underset{\widehat{\beta}_j}{\operatorname{argmin}}\left\{\frac{1}{2}A\widehat{\beta}_j{}^2 - B\widehat{\beta}_j + \lambda_1|\widehat{\beta}_j|\right\} \tag{40}
$$

Утверждение 1. *Если $A > 0, \lambda_1 \geq 0$, то минимум функции*

$$
g(a) = \frac{1}{2}Aa^2 - Ba + \lambda_1|a|
$$

достигается в точке

$$
a^* = \frac{S(B, \lambda_1)}{A} \tag{41}
$$

где $S(\cdot,\cdot)$ - функция soft-thresholding.

Доказательство. Очевидно, что функция $g(a)$ - строго выпуклая, поэтому ее минимум существует и единственный. Минимальный субградиент в точке $a = 0$ равен $S(B, \lambda_1)$, поэтому минимум в точке $a = 0$ будет достигаться только если $|B| < \lambda_1$.

Если $B > \lambda_1$, то $g(a) < g(-a)$ при $a > 0$, следовательно минимум достигается при $a > 0$. Точка минимума находится из условия равенства нулю производной в области $a > 0$

$$g'(a^*) = Aa^* - B + \lambda_1 = 0$$

Следовательно, $a^* = (B - \lambda_1)/A$.

Аналогично для случая $B < -\lambda_1$ получим $a^* = (B + \lambda_1)/A$. Все эти случаи описываются общей формулой (41). □

Таким образом, решение задачи (40) имеет вид

$$\widehat{\beta}_j = \frac{S(\mu \sum_{i=1}^n w_i q_i x_{ij} + \nu\beta_j, \lambda_1)}{\mu \sum_{i=1}^n w_i x_{ij}^2 + \nu + \lambda_2}$$

Преобразуем

$$\mu q_i = z_i - \mu \sum_{k \neq j} \widehat{\beta}_k^T x_{ik} + \mu \boldsymbol{\beta}^T \mathbf{x}_i = z_i - \mu \widehat{\boldsymbol{\beta}}^T \mathbf{x}_i + \mu \boldsymbol{\beta}^T \mathbf{x}_i + \mu \widehat{\beta}_j x_{ik}$$

$$= z_i - \mu \Delta \boldsymbol{\beta}^T \mathbf{x}_i + \mu(\beta_j + \Delta\beta_j) x_{ik}$$

Поэтому решение исходной задачи (39)

$$\Delta\beta_j = \frac{S(\sum_{i=1}^n w_i r_i x_{ij} + \nu\beta_j, \lambda_1)}{\mu \sum_{i=1}^n w_i x_{ij}^2 + \nu + \lambda_2} - \beta_j$$

$$r_i = z_i - \mu \Delta \boldsymbol{\beta}^T \mathbf{x}_i + (\beta_j + \mu\Delta\beta_j) x_{ij}$$

2 Ограничения сверху на вторые производные функций риска

- **Квадратичная**: $\ell(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2} = 1$

- **Логистическая**: $\ell(y, \hat{y}) = \log(1 + \exp(-y\hat{y}))$. Для логистической функции риска $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2} = p(\hat{y})(1 - p(\hat{y}))$ где $p(\hat{y}) = 1/(1 + e^{-\hat{y}})$ и, следовательно $\frac{\partial^2 \ell(y,\hat{y})}{\partial \hat{y}^2} \leq \frac{1}{4}$.

- **Пробит**: $\ell(y, \hat{y}) = -\log(\Phi(y\hat{y}))$, где $\Phi(\cdot)$ - это куммулятивная функция нормального распределения. Обозначим $p(\hat{y}) = \frac{1}{\sqrt{2\pi}} \exp\left(-\hat{y}^2/2\right)$. Достаточно выполнить доказательство только для $y = 1$, потому что $\frac{\partial^2 \ell(-1,\hat{y})}{\partial \hat{y}^2} = \frac{\partial^2 \ell(1,-\hat{y})}{\partial \hat{y}^2}$. Имеем

$$\frac{\partial^2 \ell(y, \hat{y})}{\partial \hat{y}^2} = \frac{\hat{y}p(\hat{y})}{\Phi(\hat{y})} + \frac{p^2(\hat{y})}{\Phi^2(\hat{y})}$$

Когда $\hat{y} \geq 0$, то вторая производная ограничена сверху

$$\frac{\hat{y}p(\hat{y})}{\Phi(\hat{y})} + \frac{p^2(\hat{y})}{\Phi^2(\hat{y})} \leq 2\hat{y}p(\hat{y}) + 4p^2(\hat{y}) \leq 2p(1) + 4p(0)$$

потому что $\Phi(\hat{y}) \geq \Phi(0) = 1/2$ и $\hat{y}p(\hat{y})$ достигает максимум при $\hat{y} = 1$. Когда $\hat{y} \in (-1, 0)$, то вторая производная ограничена. Случай $\hat{y} \leq -1$ немного более сложный. Из [103] мы имеем

$$\frac{|\hat{y}|p(\hat{y})}{1 + \hat{y}^2} < \Phi(\hat{y}) < \frac{p(\hat{y})}{|\hat{y}|}$$

тогда

$$\frac{1}{\Phi(\hat{y})} < \frac{1 + \hat{y}^2}{|\hat{y}|p(\hat{y})}$$
$$\frac{\hat{y}}{\Phi(\hat{y})} < \hat{y}\frac{|\hat{y}|}{p(\hat{y})}$$

и, следовательно

$$\frac{\partial^2 \ell(y, \hat{y})}{\partial \hat{y}^2} = \frac{\hat{y}p(\hat{y})}{\Phi(\hat{y})} + \frac{p^2(\hat{y})}{\Phi^2(\hat{y})} < \hat{y}|\hat{y}| + \left(\frac{1 + \hat{y}^2}{|\hat{y}|}\right)^2 = -\hat{y}^2 + \frac{1 + 2\hat{y}^2 + \hat{y}^4}{\hat{y}^2} = 2 + \frac{1}{\hat{y}^2} \leq 3$$

Таким образом, для всех вариантов $\hat{y} \geq 0$, $\hat{y} \in (-1, 0)$, $\hat{y} \leq -1$ вторая производная ограничена сверху.

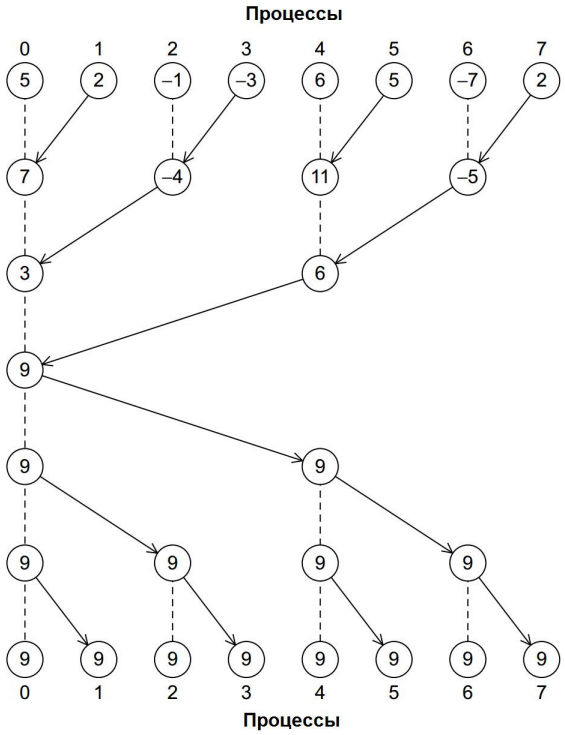


Рис. 19: MPI AllReduce

3 MPI AllReduce

На рис. 19 приведена схема работы алгоритма передачи данных `MPI AllReduce`. Данная схема может использоваться как для передачи данных между процессами на одном узле, так и в распределенной среде. Процессы обмениваются данными по структуре бинарного дерева. Алгоритм состоит из двух этапов: `Reduce` и `Broadcast`. На этапе `Reduce` данные передаются по дереву от листьев к корню, попутно в узлах дерева выполняется необходимая операция над данными. После этого в корне находится результат агрегации. Далее на этапе `Broadcast` результат агрегации передается вниз от корня к листьям, распространяясь, таким образом по всем процессам. Если M - число узлов кластере, размер массива для обработки в одном процессе - a, то объем передаваемых данных будет $O(aM)$, верхнее ограничение на время передачи - $O(a \ln M)$.

4 Обобщение до распределенного обучения бустинга

Программная реализация алгоритма `d-GLMNET` может быть обобщена для использования при обучении *аддитивных моделей (additive models)*, использующихся в бустинге. Рассмотрим, например, алгоритм Gradient Boosting Machine (GBM) [99]. Если имеются пары $(\mathbf{x}, y)$ подчиняющиеся совместному распределению $P(\mathbf{x}, y)$, то GBM пытается построить функцию $F^*(\mathbf{x})$, которая минимизирует математическое ожидание некоторой функции потерь $L(y, \hat{y})$

$$F^*(\mathbf{x}) = \underset{F(\mathbf{x})}{\operatorname{argmin}} \mathbb{E}_{y,\mathbf{x}} L(y, F(\mathbf{x}))$$

Функция $F^*(x)$ строится как сумма «базовых регрессий» (weak learner)

$$F_K(\mathbf{x}) = \sum_{k=0}^{K} \beta_k h(\mathbf{x}, a_k)$$

Базовые регрессии $h(\mathbf{x}, a_k)$ - это обычно деревья решений. Вектор a определяет структуру дерева. Базовые регрессии подбираются последовательно на каждом шаге с помощью приближенной минимизации

$$(\beta_k, a_k) = \underset{\beta,a}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{k-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i, a))$$

которая выполняется следующим образом

$$a_k = \underset{a}{\operatorname{argmin}} \sum_{i=1}^{n} (z_i - h(\mathbf{x}_i, a))^2$$

$$\rho_k = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{k-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i, a_k))$$

где

$$z_i = -\frac{\partial L(y_i, F_{k-1}(\mathbf{x}_i))}{\partial \hat{y}}$$

После этого функция обновляется

$$F_k(\mathbf{x}) = F_{k-1}(\mathbf{x}) + \nu \rho_k h(\mathbf{x}, a_k)$$

где $\nu < 1$ - это параметр *регуляризации* (shrinkage). На каждой итерации для подбора «базовой рергессии» используется случайное подмножество обучающей выборки (стохастический бустинг [100]). Стандартной долей случайно выбираемых объектов является $0,5$.

Параллельная реализация бустинга, являющаяся обобщением `d-GLMNET` приведена в Алгоритме 18 (можно сравнить с исходным Алгоритмом 15). В нем используется обозначение $\mathbf{x} = ((\mathbf{x}^1)^T, \ \ldots \ , (\mathbf{x}^M)^T)$, т.е. вектор признаков разбивается на блоки соответственно

Алгоритм 18: Распределенный бустинг

Вход : Обучающая выборка, разбиение признаков $S^1, \ldots, S^M$, параметр $\nu < 1$

$\hat{\mathbf{y}} \leftarrow 0$, *вектор предсказаний, длина* n

Пока *не выполнено условие останова*

  Выполнить параллельно на M машинах:

    Вычислить $f^m(\mathbf{x}^m)$ с помощью шага алгоритма GBM по части обучающей выборки X^m

    Суммировать вектора $(f^m(\mathbf{x}_i^m))_{i=1}^n$ с помощью MPI_AllReduce:

      $\Delta\hat{\mathbf{y}} \leftarrow \sum_{m=1}^M (f^m(\mathbf{x}_i^m))_{i=1}^n$

    Вычислить размер шага α используя линейный поиск

      $\alpha \leftarrow \text{argmin}_\rho \sum_{i=1}^n L(y_i, \hat{y}_i + \rho\Delta\hat{y}_i)$

    $F(\mathbf{x}) \leftarrow F(\mathbf{x}) + \nu\alpha \sum_{m=1}^M f^m(\mathbf{x}^m)$

    $\hat{\mathbf{y}} \leftarrow \hat{\mathbf{y}} + \nu\alpha\Delta\hat{\mathbf{y}}$

Возврат: $\boldsymbol{\beta}$

разбиению $S^1, \ldots, S^M$. Обучающая выборка X разделена по машинам соответственно ("вертикальное" разбиение). Каждая машина обучает базовые регрессии $f^m(\mathbf{x}^m)$ зависящие только от подмножества переменных S^m. На каждой итерации синхронизируется состояние - вектор текущих предсказаний $\hat{\mathbf{y}}$. Объем передаваемых по сети данных равен Mn. Тестирование данной реализации является интересной темой для дальнейшего исследования.

Список иллюстраций

Список таблиц

Список алгоритмов

Список литературы

[1] Zou Hui, Hastie Trevor. Regularization and variable selection via the elastic net // Journal of the Royal Statistical Society: Series B (Statistical Methodology). — 2005. — Apr. — Vol. 67, no. 2. — P. 301–320. — Access mode: http://doi.wiley.com/10.1111/j.1467-9868.2005.00503.x.

[2] Yuan Ming, Lin Yi. Model selection and estimation in regression with grouped variables // Journal of the Royal Statistical Society: Series B (Statistical Methodology). — 2006. — Vol. 68, no. 1. — P. 49–67.

[3] Meier Lukas, Van De Geer Sara, Bühlmann Peter. The group lasso for logistic regression // Journal of the Royal Statistical Society. Series B: Statistical Methodology. — 2008. — Vol. 70, no. 1. — P. 53–71.

[4] Fan Jianqing, Li Runze. Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties // Journal of the American Statistical Association. — 2001. — Dec. — Vol. 96, no. 456. — P. 1348–1360. — Access mode: http://www.tandfonline.com/doi/abs/10.1198/016214501753382273.

[5] Zhang Cun Hui. Nearly unbiased variable selection under minimax concave penalty. — 2010. — Vol. 38. — P. 894–942. — ISBN: 9040210063. — 1002.4734.

[6] Balakrishnan Suhrid, Madigan David. Algorithms for Sparse Linear Classifiers in the Massive Data Setting // Journal of Machine Learning Research. — 2007. — Vol. 1. — P. 1–26.

[7] Langford John, Li Lihong, Zhang Tong. Sparse Online Learning via Truncated Gradient // Journal of Machine Learning Research. — 2009. — Vol. 10. — P. 777–801.

[8] Lipton Zachary C, Elkan Charles. Efficient Elastic Net Regularization for Sparse Linear Models // arXiv preprint arXiv:1505.06449. — 2015.

[9] McMahan H Brendan. Follow-the-Regularized-Leader and Mirror Descent : Equivalence Theorems and L1 Regularization // AISTATS' 11. — 2011.

[10] Ad Click Prediction: a View from the Trenches / H Brendan McMahan, Gary Holt, D Sculley et al. // KDD' 13. — Chicago, Illinois, USA, 2013.

[11] A Comparison of Optimization Methods and Software for Large-scale L1-regularized Linear Classification / Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, Chih-Jen Lin // Journal of Machine Learning Research. — 2010. — Vol. 11. — P. 3183–3234.

[12] Genkin Alexander, Lewis David D, Madigan David. Large-Scale Bayesian Logistic Regression for Text Categorization // Technometrics. — 2007. — Aug. — Vol. 49, no. 3. — P. 291–304.

[13] Friedman Jerome, Hastie Trevor, Tibshirani Rob. Regularization Paths for Generalized Linear Models via Coordinate Descent // Journal of Statistical Software. — 2010. — Vol. 33, no. 1.

[14] An Improved GLMNET for L1-regularized Logistic Regression / Guo-Xun Yuan, Chia-Hua Ho, Cho-Jui Hsieh, Chih-Jen Lin // Journal of Machine Learning Research. — 2012. — Vol. 13. — P. 1999–2030.

[15] A comparison of numerical optimizers for logistic regression : Rep. ; Executor: Thomas P Minka : 2007.

[16] Distributed Newton Methods for Regularized Logistic Regression / Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, Chih-Jen Lin // Advances in Knowledge Discovery and Data Mining. — Springer, 2015. — P. 690–703.

[17] Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers / Stephen Boyd, Neal Parikh, Eric Chu et al. — 2010. — Vol. 3. — P. 1–122. — ISBN: 1935823719358.

[18] A reliable effective terascale linear learning system : Rep. ; Executor: Alekh Agarwal, O Chapelle, M Dudík, John Langford : 2011. — arXiv:1110.4198v2. — http://arxiv.org/abs/1110.4198.

[19] Parallelized Stochastic Gradient Descent. / M Zinkevich, Markus Weimer, AJ Smola, L Li // NIPS' 10. — 2010. — Access mode: `https://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.pdf`.

[20] Parallel Coordinate Descent for L1-Regularized Loss Minimization / Joseph K Bradley, Aapo Kyrola, Danny Bickson, Carlos Guestrin // ICML' 11. — Bellevue, WA, USA, 2011. — arXiv:1105.5379v1.

[21] Parallel Coordinate Descent Methods for Big Data Optimization : Rep. ; Executor: Peter Richtárik, Martin Takáč : 2012. — P. 1–35. arXiv:1212.0873v1. — http://arxiv.org/abs/1212.0873.

[22] Peng Zhimin, Yan Ming, Yin Wotao. Parallel and Distributed Sparse Optimization // STATOS' 13. — 2013.

[23] Beck Amir, Teboulle Marc. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems // SIAM Journal on Imaging Sciences. — 2009. — Jan. — Vol. 2, no. 1. — P. 183–202. — Access mode: http://epubs.siam.org/doi/abs/10.1137/080716542.

[24] More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server / Qirong Ho, James Cipar, Henggang Cui et al. // NIPS' 13. — 2013.

[25] Massive parallelization of serial inference algorithms for a complex generalized linear model. / Marc a Suchard, Shawn E Simpson, Ivan Zorych et al. // ACM transactions on modeling and computer simulation : a publication of the Association for Computing Machinery. — 2013. — Jan. — Vol. 23, no. 1. — P. 1–18. — arXiv:1208.0945v1.

[26] Trofimov Ilya, Kornetova Anna, Topinskiy Valery. Using boosted trees for click-through rate prediction for sponsored search // Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy / ACM. — 2012. — P. 2.

[27] Trofimov Ilya. New Features for Query Dependent Sponsored Search Click Prediction // WWW. — Rio de Janeiro, Brazil, 2013.

[28] Trofimov Ilya, Genkin Alexander. Distributed coordinate descent for l1-regularized logistic regression // International Conference on Analysis of Images, Social Networks and Texts / Springer. — 2015. — P. 243–254.

[29] Trofimov Ilya, Genkin Alexander. Distributed coordinate descent for generalized linear models with regularization // Pattern Recognition and Image Analysis. — 2017. — Vol. 27, no. 2. — P. 349–364.

[30] Трофимов И. Распределенные вычислительные системы для машинного обучения // Информационные технологии и вычислительные системы. — 2017. — Т. 1, № 3. — С. 56–69.

[31] The sparse Laplacian shrinkage estimator for high-dimensional regression / Jian Huang, Shuangge Ma, Hongzhe Li, Cun-Hui Zhang // Annals of statistics. — 2011. — Vol. 39, no. 4. — P. 2021.

[32] Fu W.J. Penalized Regressions : The Bridge Versus the Lasso // J. Comput. Graph. Statist. — 1998. — Vol. 7, no. 3. — P. 397–416.

[33] Pathwise coordinate optimization / Jerome Friedman, Trevor Hastie, Holger Höfling, Robert Tibshirani // The Annals of Applied Statistics. — 2007. — dec. — Vol. 1, no. 2. — P. 302–332. — Access mode: http://projecteuclid.org/euclid.aoas/1196438020.

[34] Sparse multinomial logistic regression: Fast algorithms and generalization bounds / Balaji Krishnapuram, Lawrence Carin, Mario AT Figueiredo, Alexander J Hartemink // Pattern Analysis and Machine Intelligence, IEEE Transactions on. — 2005. — Vol. 27, no. 6. — P. 957–968.

[35] Tseng Paul, Yun Sangwoon. A coordinate gradient descent method for nonsmooth separable minimization // Mathematical Programming. — 2009. — Aug. — Vol. 117, no. 1-2. — P. 387–423.

[36] Bottou Leon, Bousquet Olivier. The Tradeoffs of Large Scale Learning. — 2005.

[37] Karampatziakis Nikos, Langford John. Online importance weight aware updates // UAI. — 2010. — http://arxiv.org/abs/1011.1576. arXiv:1011.1576v4.

[38] McMahan H. Brendan, Streeter Matthew. Adaptive Bound Optimization for Online Convex Optimization. — 2010. — 1002.4908.

[39] Ross Stephane, Mineiro Paul, Langford John. Normalized online learning // Twenty-Ninth Conference on Uncertainty in Artificial Intelligence. — 2013. — arXiv:1305.6646v1.

[40] Spark : Cluster Computing with Working Sets / Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin et al. // HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. — 2010. — P. 10.

[41] Nocedal Jorge. Updating quasi-Newton matrices with limited storage // Mathematics of computation. — 1980. — Vol. 35, no. 151. — P. 773–782.

[42] Wright Stephen J, Nocedal Jorge. Numerical optimization. — Springer New York, 1999. — Vol. 2.

[43] Sutton Charles, Mccallum Andrew. An Introduction to Conditional Random Fields for Relational Learning // Graphical Models. — 2002. — Vol. 7.

[44] Liu Dong C, Nocedal Jorge. On the limited memory BFGS method for large scale optimization // Mathematical programming. — 1989. — Vol. 45, no. 1-3. — P. 503–528.

[45] Chen Weizhu, Wang Zhenghao, Zhou Jingren. Large-scale L-BFGS using MapReduce // Advances in Neural Information Processing Systems. — 2014. — P. 1332–1340.

[46] Scaling distributed machine learning with the parameter server / Mu Li, David G Andersen, Jun Woo Park et al. // 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). — 2014. — P. 583–598. — https://github.com/dmlc/ps-lite.

[47] Smola Alexander, Narayanamurthy Shravan. An architecture for parallel topic models // Proceedings of the VLDB Endowment. — 2010. — Vol. 3, no. 1-2. — P. 703–710.

[48] Distributed GraphLab: a framework for machine learning and data mining in the cloud / Yucheng Low, Danny Bickson, Joseph Gonzalez et al. // Proceedings of the VLDB Endowment. — 2012. — Vol. 5, no. 8. — P. 716–727. — http://graphlab.org.

[49] Stylianou Maria. Apache Giraph for applications in Machine Learning & Recommendation Systems. — 2014. — http://people.inf.ethz.ch/jaggim/meetup/5/slides/ML-meetup-5-Stylianou-Giraph.pdf.

[50] Chen Tianqi, Guestrin Carlos. XGBoost: A scalable tree boosting system // arXiv preprint arXiv:1603.02754. — 2016.

[51] Svore Krysta M, Burges CJ. Large-scale learning to rank using boosted decision trees // Scaling Up Machine Learning: Parallel and Distributed Approaches. — 2011. — Vol. 2.

[52] Parallel boosted regression trees for web search ranking / Stephen Tyree, Kilian Q Weinberger, Kunal Agrawal, Jennifer Paykin // Proceedings of the 20th international conference on World wide web / ACM. — 2011. — P. 387–396.

[53] Microsoft. Distributed Machine Learning Toolkit. — 2016. — http://www.dmtk.io.

[54] FireCaffe: near-linear acceleration of deep neural network training on compute clusters / Forrest N Iandola, Khalid Ashraf, Mattthew W Moskewicz, Kurt Keutzer // arXiv preprint arXiv:1511.00175. — 2015.

[55] Deep learning with COTS HPC systems / Adam Coates, Brody Huval, Tao Wang et al. — 2013.

[56] Large scale distributed deep networks / Jeffrey Dean, Greg Corrado, Rajat Monga et al. // Advances in neural information processing systems. — 2012. — P. 1223–1231.

[57] Dean Jeffrey, Ghemawat Sanjay. MapReduce : Simplified Data Processing on Large Clusters // OSDI' 04. — San Francisco, 2004.

[58] YT — a new framwork for distributed computations : Rep. ; Executor: Maxim Babenko : 2013. — https://events.yandex.ru/lib/talks/1091/.

[59] Nokia Research Center. Disco Project. — 2008. — http://discoproject.org/.

[60] White Tom. Hadoop: The definitive guide. — "O'Reilly Media, Inc. 2012.

[61] Map-reduce for machine learning on multicore / Cheng Chu, Sang Kyun Kim, Yi-An Lin et al. // Advances in neural information processing systems. — 2007. — Vol. 19. — P. 281.

[62] Graphlab: A new framework for parallel machine learning / Yucheng Low, Joseph Gonzalez, Aapo Kyrola et al. // UAI' 10. — Cataline Island, California, 2010. — arXiv:1006.4990v1.

[63] Planet: massively parallel learning of tree ensembles with mapreduce / Biswanath Panda, Joshua S Herbach, Sugato Basu, Roberto J Bayardo // Proceedings of the VLDB Endowment. — 2009. — Vol. 2, no. 2. — P. 1426–1437.

[64] Leskovec Jure, Rajaraman Anand, Ullman Jeffrey David. Mining of Massive Datasets. — Cambridge University Press, 2014.

[65] Bekkerman Ron, Bilenko Mikhail, Langford John. Scaling up machine learning: Parallel and distributed approaches. — Cambridge University Press, 2011.

[66] Gradient Boosted Machines with H2O : Rep. ; Executor: Cliff Click, Jessica Lanford, Michal Malohlava et al. : 2015. — http://h2o.gitbooks.io/gbm-with-h2o/.

[67] Scaling Up Decision Tree Ensembles : Rep. ; Executor: Misha Bilenko et al. : 2011. — http://hunch.net/~large_scale_survey/TreeEnsembles.pdf.

[68] Forum Message Passing Interface. MPI: A Message-Passing Interface Standard, Version 3.0 // ACM SIGOPS operating systems review / High Performance Computing Center Stuttgart (HLRS). — 2012.

[69] Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation / Edgar Gabriel, Graham E. Fagg, George Bosilca et al. // Proceedings, 11th European PVM/MPI Users' Group Meeting. — Budapest, Hungary, 2004. — September. — P. 97–104.

[70] Rabit developers. Rabit: Reliable Allreduce and Broadcast Interface. — 2016. — https://github.com/dmlc/rabit.

[71] Pacheco Peter. An introduction to parallel programming. — Elsevier, 2011.

[72] Chord: A scalable peer-to-peer lookup service for internet applications / Ion Stoica, Robert Morris, David Karger et al. // ACM SIGCOMM Computer Communication Review. — 2001. — Vol. 31, no. 4. — P. 149–160.

[73] Van Renesse Robbert, Schneider Fred B. Chain Replication for Supporting High Throughput and Availability. // OSDI. — Vol. 4. — 2004. — P. 91–104.

[74] Petuum: A framework for iterative-convergent distributed ML / Wei Dai, Jinliang Wei, Jin Kyu Kim et al. — 2013. — http://www.petuum.com.

[75] Apache Foundation. Spark MLLib. — 2016. — http://spark.apache.org/mllib/.

[76] SparkNet: Training Deep Networks in Spark / Philipp Moritz, Robert Nishihara, Ion Stoica, Michael I Jordan // arXiv preprint arXiv:1511.06051. — 2015. — https://github.com/amplab/SparkNet.

[77] Caffe: Convolutional architecture for fast feature embedding / Yangqing Jia, Evan Shelhamer, Jeff Donahue et al. // Proceedings of the 22nd ACM international conference on Multimedia / ACM. — 2014. — P. 675–678.

[78] Large-scale logistic regression and linear support vector machines using Spark / Chieh-Yen Lin, Cheng-Hao Tsai, Ching-Pei Lee, Chih-Jen Lin // Big Data (Big Data), 2014 IEEE International Conference on / IEEE. — 2014. — P. 519–528.

[79] Pafka Szilard. Simple/limited/incomplete benchmark for scalability, speed and accuracy of machine learning libraries for classification. — 2016. — https://github.com/szilard/benchm-ml.

[80] Pregel: a system for large-scale graph processing / Grzegorz Malewicz, Matthew H Austern, Aart JC Bik et al. // Proceedings of the 2010 ACM SIGMOD International Conference on Management of data / ACM. — 2010. — P. 135–146.

[81] Ching Avery, Kunz C. Giraph: Large-scale graph processing infrastructure on Hadoop // Hadoop Summit. — 2011. — Vol. 6, no. 29. — P. 2011. — http://giraph.apache.org.

[82] Chandy K Mani, Lamport Leslie. Distributed snapshots: determining global states of distributed systems // ACM Transactions on Computer Systems (TOCS). — 1985. — Vol. 3, no. 1. — P. 63–75.

[83] An experimental comparison of pregel-like graph processing systems / Minyang Han, Khuzaima Daudjee, Khaled Ammar et al. // Proceedings of the VLDB Endowment. — 2014. — Vol. 7, no. 12. — P. 1047–1058.

[84] Parameter Server for Distributed Machine Learning / M Li, Li Zhou, Zichao Yang et al. // Big Learning Workshop. — 2013. — P. 1–10. — Access mode: http://www.istc-cc.cmu.edu/publications/papers/2013/ps.pdf.

[85] Valiant L.G. A Bridging Model for Parallel Computation // Communications of the ACM. — 1990. — Vol. 22, no. 8. — P. 103–111.

[86] HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent / Feng Niu, Benjamin Recht, Christopher Re, Stephen J. Wright. — 2011. — P. 21. — 1106.5730.

[87] Fugue : Slow-Worker-Agnostic Distributed Learning for Big Models on Big Data / Abhimanu Kumar, Alex Beutel, Qirong Ho, Eric P Xing // AISTATS. — Vol. 33. — 2014.

[88] Introduction to Computational Advertising : Rep. ; Executor: Andrei Broder, Vanja Josifovski : 2012.

[89] Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords : Rep. / National Bureau of Economic Research ; Executor: Benjamin Edelman, Michael Ostrovsky, Michael Schwarz : 2005.

[90] Chervonenkis Alexey, Sorokina Anna, Topinsky Valery A. Optimization of ads allocation in sponsored search // Proceedings of the 22nd international conference on World Wide Web companion / International World Wide Web Conferences Steering Committee. — 2013. — P. 121–122.

[91] Richardson Matthew, Dominowska Ewa, Ragno Robert. Predicting clicks: estimating the click-through rate for new ads // Proceedings of the 16th international conference on World Wide Web / ACM. — 2007. — P. 521–530.

[92] Optimizing display advertisements based on historic user trails / Neha Gupta, Udayan Khurana, T Lee, Sandeep Nawathe // SIGIR Workshop on Internet Advertising. — 2011.

[93] Dembczynski Krzysztof, Kotlowski Wojciech, Weiss Dawid. Predicting ads click-through rate with decision rules // Workshop on targeting and ranking in online advertising. — Vol. 2008. — 2008.

[94] Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine / Thore Graepel, Joaquin Q Candela, Thomas Borchert, Ralf Herbrich // Proceedings of the 27th International Conference on Machine Learning (ICML-10). — 2010. — P. 13–20.

[95] A novel click model and its applications to online advertising / Zeyuan Allen Zhu, Weizhu Chen, Tom Minka et al. // Proceedings of the third ACM international conference on Web search and data mining / ACM. — 2010. — P. 321–330.

[96] Baqapuri Afroze Ibrahim, Trofimov Ilya. Using Neural Networks for Click Prediction of Sponsored Search // arXiv preprint arXiv:1412.6601. — 2014.

[97] Matrixnet : Rep. ; Executor: A Gulin : 2010. — http://www.ashmanov.com/arc/searchconf2010/08gulin-searchconf2010.ppt.

[98] Gulin Andrey, Kuralenok Igor, Pavlov Dmitry. Winning The Transfer Learning Track of Yahoo!'s Learning To Rank Challenge with YetiRank. // Yahoo! Learning to Rank Challenge. — 2011. — P. 63–76.

[99] Friedman Jerome H. Greedy function approximation: a gradient boosting machine // Annals of statistics. — 2001. — P. 1189–1232.

[100] Friedman Jerome H. Stochastic gradient boosting // Computational Statistics & Data Analysis. — 2002. — Vol. 38, no. 4. — P. 367–378.

[101] Shaparenko Benyah, Çetin Özgür, Iyer Rukmini. Data-driven text features for sponsored search click prediction // KDD, ADKDD Workshop. — New York, USA : ACM Press, 2009. — P. 46–54. — Access mode: `http://portal.acm.org/citation.cfm?doid=1592748.1592755`.

[102] Text-based online advertising: L1 regularization for feature selection : Rep. ; Executor: Ilya Trofimov : 2013. — http://www.slideshare.net/ssuserb10599/ss-26831908.

[103] Normal distribution : Rep. ; Executor: Wikipedia for schools : 2013. — http://schools-wikipedia.org/wp/n/Normal_distribution.htm.