Министерство образования и науки Российской Федерации
Московский физико-технический институт (государственный университет)
Факультет управления и прикладной математики
Вычислительный центр им. А. А. Дородницына РАН
Кафедра «Интеллектуальные системы»

Романов Андрей Михайлович

Методы понижения сложности композиций, получаемых при градиентном бустинге

511656 - Математические и информационные технологии

Магистерская диссертация

Научный руководитель:
профессор
Мучник Илья Борисович

Москва
2013

Содержание

1 Введение

1.1 Задачи обучения по прецедентам

В данной работе будем рассматривать задачи *обучения с учителем*. Имеется множество объектов (ситуаций) X и множество возможных ответов (откликов, реакций) Y. Существует некоторая зависимость между ответами и объектами, но она неизвестна. Такую зависимость будем называть *целевой функцией* (target function) $y^*\colon X \to Y$. Известна только конечная совокупность прецедентов — пар «*объект, ответ*», называемая *обучающей выборкой*, которую будем обозначать как $X^\ell = \{(x_j, y_j)\}_{j=1}^\ell$, где $y_j = y^*(x_j) \in Y$ и $\{x_j\}_{j=1}^\ell \subset X$. На основе этих данных X^ℓ требуется восстановить зависимость, то есть построить алгоритм $a\colon X \to Y$, способный для любого объекта $x_j \in X$ выдать достаточно точный ответ $a(x_j)$. То есть, $a(x)$ должна приближать целевую функцию. Для измерения точности ответов определённым образом вводится *функционал качества* (например MSE или LL, которые далее будут описаны в разделе 1.2).

Признак (feature) f объекта x — это результат измерения некоторой характеристики объекта. Формально признаком называется отображение $f\colon X \to D_f$, где D_f — множество допустимых значений признака. В зависимости от природы множества D_f признаки делятся на несколько типов.

- $D_f = \{0, 1\}$, тогда f — *бинарный* признак;
- D_f — конечное множество, тогда f — *номинальный* признак;
- D_f — конечное упорядоченное множество, тогда f — *порядковый* признак;
- $D_f = \mathbb{R}$, то f — *количественный* признак.

Совокупность признаковых описаний всех объектов выборки X^ℓ, записанную в виде таблицы размера $\ell \times n$, называют *матрицей объектов–признаков* $F = \|f_j(x_i)\|_{\ell \times n}$. Матрица объектов–признаков является стандартным и наиболее распространённым способом представления исходных данных в прикладных задачах.

$$\begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots \\ f_1(x_l) & \dots & f_n(x_l) \end{pmatrix}$$

Методы решения задач обучения по прецедентам сужественно зависят от вида мноджества ответов Y. Если $Y = \{1, \dots, M\}$, то это *задача классификации* (classification) на M непересекающихся классов. В этом случае всё множество объектов X разбивается на классы $K_y = \{x \in X\colon y^*(x) = y\}$, и алгоритм $a(x)$ должен давать ответ на вопрос: «к какому классу принадлежит объект x?». К задачам такого типа относится , например, задача *задача оценивания заёмщиков*. Она решается банками при выдаче кредитов. Объектами в данном случае являются заёмщики — физические или юридические лица, претендующие на получение кредита. В случае физических лиц признаковое описание состоит из анкеты, которую заполняет сам заёмщик, и, возможно, дополнительной информации, которую банк собирает о нём из собственных источников. Примеры бинарных признаков: пол, наличие телефона.

Номинальные признаки — место проживания, профессия, работодатель. Порядковые признаки — образование, занимаемая должность. Количественные признаки — возраст, стаж работы, доход семьи, размер задолженностей в других банках, сумма кредита. Обучающая выборка составляется из заёмщиков с известной кредитной историей. В простейшем случае принятие решений сводится к классификации заёмщиков на два класса: «хороших» и «плохих». Кредиты выдаются только заёмщикам первого класса.

Если $Y = \mathbb{R}$, то это *задача восстановления регрессии* (regression estimation). К задачам такого типа относится , например, задача *прогнозирования потребительского спроса*. Она решается современными супермаркетами и торговыми розничными сетями. Для эффективного управления торговой сетью необходимо прогнозировать объёмы продаж для каждого товара на заданное число дней вперёд. На основе этих прогнозов осуществляется планирование закупок, управление ассортиментом, формирование ценовой политики, планирование промоакций (рекламных кампаний). Исходными данными для прогнозирования являются временные ряды цен и объёмов продаж по товарам и по отдельным магазинам. Для увеличения точности прогнозов необходимо учитывать различные внешние факторы, влияющие на спрос: рекламные кампании, социально-демографические условия, активность конкурентов, праздники, и даже погодные условия. В зависимости от целей анализа в роли объектов выступают либо товары, либо магазины, либо пары «магазин–товар».

1.2 Функционалы качества

Опр. 1.1. ***Функция потерь*** *(loss function) — это неотрицательная функция $\mathcal{L}(a, x)$, характеризующая величину ошибки алгоритма a на объекте x. Если $\mathcal{L}(a, x) = 0$, то ответ $a(x)$ называется корректным.*

Опр. 1.2. ***Функционал качества*** *алгоритма a на выборке X^ℓ:*

$$Q(a, X^\ell) = \frac{1}{\ell} \sum_{j=1}^{\ell} \mathcal{L}(a, x_j). \tag{1}$$

Наиболее часто используются следующие функции потерь, при $Y \subseteq \mathbb{R}$:

- $\mathcal{L}(a, x) = [a(x) \neq y^*(x)]$ — индикатор ошибки (обычно применяется в задачах классификации);
- $\mathcal{L}(a, x) = |a(x) - y^*(x)|$ — отклонение от правильного ответа; функционал Q называется *средней ошибкой* алгоритма a на выборке X^ℓ;
- $\mathcal{L}(a, x) = (a(x) - y^*(x))^2$ — квадратичная функция потерь; функционал Q называется *средней квадратичной ошибкой* алгоритма a на выборке X^ℓ; обычно применяется в задачах регрессии.

Опр. 1.3. ***MSE*** *— средняя квадратичная ошибка.*

$$MSE(a, X^\ell) = \frac{1}{\ell} \sum_{x \in X^\ell} \left(a(x) - y^*(x)\right)^2. \tag{2}$$

Опр. 1.4. ***LinCorr*** *— линейная корреляция.*

$$LinCorr(a, X^\ell) = \frac{\sum\limits_{x\in X^\ell}(a(x)-\overline{a})(y^*(x)-\overline{y^*})}{\sqrt{\sum\limits_{x\in X^\ell}(a(x)-\overline{a})^2 \sum\limits_{x\in X^\ell}(y^*(x)-\overline{y^*})^2}};$$
$$\overline{a} = \frac{1}{\ell}\sum_{x\in X^\ell} a(x), \qquad \overline{y^*} = \frac{1}{\ell}\sum_{x\in X^\ell} y^*(x). \tag{3}$$

Ещё одна функция потерь может быть использована для оценки качества модели при вероятностной постановке задачи обучения. В задачах обучения по прецедентам элементы множества X — это не реальные объекты, а лишь доступные данные о них. Данные могут быть *неточными*, поскольку измерения значений признаков $f_j(x)$ и целевой зависимости $y^*(x)$ обычно выполняются с погрешностями. Данные могут быть *неполными*, поскольку измеряются не все мыслимые признаки, а лишь физически доступные для измерения. В результате одному и тому же описанию x могут соответствовать различные объекты и различные ответы. В таком случае $y^*(x)$, строго говоря, не является функцией. Устранить эту некорректность позволяет *вероятностная постановка задачи*. Вместо существования неизвестной целевой зависимости $y^*(x)$ предположимсуществование неизвестного вероятностного распределения на множестве $X \times Y$ с плотностью $p(x, y)$, из которого случайно и независимо выбираются ℓ наблюдений X^ℓ. Такие выборки называются простыми или *случайными одинаково распределёнными* (independent identically distributed, i.i.d.).

Если наблюдения в выборке X^ℓ независимы, то совместная плотность распределения всех наблюдений равна произведению плотностей $p(x, y)$ в каждом наблюдении: $p(X^\ell) = p(x_1, y_1)\cdots p(x_\ell, y_\ell)$. Подставляя вместо $p(x, y)$ модель плотности $\phi(x, y, \theta)$, получаем функцию правдоподобия (likelihood) $W(\theta, X^\ell)$.

Опр. 1.5. ***Функция правдоподобия***.

$$W(\theta, X^\ell) = \prod_{j=1}^{\ell} \phi(x_j, y_j, \theta). \tag{4}$$

Опр. 1.6. ***Логарифм правдоподобия*** $LL = \frac{1}{\ell}\ln(W)$.

$$LL(\theta, X^\ell) = \frac{1}{\ell}\sum_{j=1}^{\ell} \ln \phi(x_j, y_j, \theta). \tag{5}$$

Получаем, что при вероятностной постановке задачи обучения в качестве функции потерь можно использовать $\mathcal{L}(\theta, x) = \ln \phi(x, y^*(x), \theta)$

Классический метод обучения, называемый *минимизацией эмпирического риска* (empirical risk minimization, ERM), заключается в том, чтобы найти алгоритм a, доставляющий минимальное значение функционалу качества Q на заданной обучающей выборке X^ℓ:

$$\mu(X^\ell) = \arg\min_a Q(a, X^\ell).$$

1.3 Бустинг, общее описание

Бустинг — алгорит машинного обучения (для регрессии или классификации), позволяющий строить модель предсказания в виде композиции из слабых алгоритмов. Он строит модель итерационно, на каждом шаге увеличивая уже построенную композицию алгоритмов. Конкретные реализации алгоритма бустинга могут сильно оличаться друг от друга. Для примера рассмотрим бустинг в задачах классификации и алгоритм *AdaBoost* [6].

Будем рассматривать задачу двуклассовой классификации, $Y = \{-1, 1\}$, X — множество всех объектов. $X^\ell = \{(x_j, y_j)\}_{j=1}^\ell$ — обучающая выборка. Для построения композиции используем некоторое множество базовых алгоритмов $\{b(x)\}$. Каждый алгоритм $b(x)\colon X \to Y$ не отказывается от классификации $\forall x \in X$.

Искомая алгоритмическая композиция имеет вид:

$$a(x) = sign\left(\sum_{j=1}^{N} \alpha_j b_j(x)\right).$$

Опр. 1.7. ***Отступ*** *пары* (y, x) *при использовании алгоритма* $a(x)$.

$$M(x, y) = y\sum_{j=1}^{N} \alpha_j b_j(x). \tag{6}$$

Функционал качества композиции определим в терминах отступов следующим образом:

$$Q_T = \sum_{k=1}^{\ell}\left[y_k \sum_{j=1}^{N} \alpha_j b_j(x_x) < 0\right] = \sum_{k=1}^{\ell}\left[M(x_k, y_k) < 0\right].$$

Q_T — число ошибок алгоритма $a(x)$ на обучающей выборке. Для минимизации функционала Q_T будем использовать две эвристики.

1. При добавлении в композицию нового слагаемого $\alpha_n b_n(x)$ все предыдущие слагаемые $\{\alpha_j b_j(x)\}_{j=1}^{n-1}$ считаем зафиксированными. Оптимируется только базовый алгоритм $b_n(x)$ и коэффициент α_n.

2. Пороговая функция потерь в Q_T заменяется на непрерывно дифференцируемую оценку сверху. Для *AdaBoost* полагаем $[M(x, y) < 0] \leq e^{-M(x,y)}$.

Учитывая вторую эвристику получаем, что:

$$Q_T \leq \tilde{Q}_T = \sum_{k=1}^{\ell} \exp\left(-y_i \sum_{j=1}^{N} \alpha_j b_j(x)\right) \to \min_{\{\alpha\}\,\{b\}}.$$

Пусть $U^\ell = (u_1, \ldots, u_\ell)$ нормированный на единицу вектор весов обьектов из обучающей выборки. Введём обозначения для суммарного веса правильных $P(b; U^\ell)$ и неправильных $N(b; U^\ell)$ классификаций:

- $P(b; U^\ell) = \sum\limits_{k=1}^{\ell} u_k\left[b(x_k) = y_k\right]$;

Алгоритм 1 *AdaBoost*.

Вход: $\{x_j, y_j\}_{j=1}^{\ell}$ — обучающая выборка, T — размер результирующей композиции.
Выход: Множество базовых алгоритмов $\{b_t(x)\}_{t=1}^{T}$ и множество коэффициентов при них $\{\alpha_t\}_{t=1}^{T}$.

1: $w_i = \frac{1}{\ell}\ \forall i \in \{1, \ldots, \ell\}$
2: **для всех** $t \in \{1, \ldots, T\}$
3: $\quad b_t(x) = \arg\min\limits_{b(x)} N(b; W^{\ell})$
4: $\quad \alpha_t = \frac{1}{2} \ln \frac{1 - N(b_t; U^{\ell})}{N(b_t; U^{\ell})}$
5: $\quad w_i = w_i \exp\left(-\alpha_t y_i b_t(x_i)\right),\ \forall i \in \{1, \ldots, \ell\}$
6: $\quad w_i = \frac{w_i}{w_0},\ w_0 = \sum\limits_{j=1}^{\ell} w_j,\ \forall i \in \{1, \ldots, \ell\}$
7: **вернуть** $F(x) = \sum\limits_{t=1}^{T} \alpha_t b_t(x)$

- $N(b; U^{\ell}) = \sum\limits_{k=1}^{\ell} u_k \left[b(x_k) = -y_k\right]$.

Алгоритм 1 *AdaBoost* основываеся на следующей теореме:

Теорема. 1.1. *Пусть базовые алгоритмы не отказываются от классификации, и для любого нормированного вектора весов W^{ℓ} существует базовый алгоритм b который классифицирует выборку чуть лучше, чем наугад: $N(b; U^{\ell}) < \frac{1}{2}$. Тогда минимум функционала $\tilde{Q}_T$ достигается при:*

$$b_T(x) = \arg\min_{b} N(b; U^{\ell});$$

$$\alpha_T = \frac{1}{2} \ln \frac{1 - N(b_T; U^{\ell})}{N(b_T; U^{\ell})}.$$

некоторые важные особенности данного алгоритма:

- На каждой итерации веса обьектов в обучающей выборке модифицируются и нормируются на единицу.
- Большие веса присваиваются обьектам с отрицательными отступами для уделения им большего внимания на следующей итерации.
- Происходит максимизация отступов обьектов.

Теорема. 1.2. *Пусть D — есть распределение пар $X \times Y$, $S = \{x_j, y_j\}_{j=1}^{m}$ — выборка из распределения D. Пусть VC — размерность пространства базовых алгоритмов равна d. Пусть $\mathcal{C}$ — семейство алгоритмов, $\mathcal{C} = \left\{ f \colon x \to \sum\limits_{h \in H} a_h h(x) \;\middle|\; a_h \geq 0,\ \sum\limits_{h} a_h = 1 \right\}$. Тогда, с вероятностью $(1 - \delta)$ $\forall f \in \mathcal{C},\ \forall \theta > 0$ выполнено:*

$$P_D\left[yf(x) \leq 0\right] \leq P_S\left[yf(x) \leq \theta\right] + C\sqrt{\frac{d \log^2(m/d)}{m\theta^2} + \frac{\log 1/\delta}{m}}. \tag{7}$$

Из уравнения (7) видно, что оценка сверху для $P_D\left[yf(x) \leq 0\right]$ не зависит от размера композиции. Таким образом, итеративное увеличение размера композиции не приводит к увеличению значения оценки верхней границы для вероятности ошибочной классификации $P_D\left[yf(x) \leq 0\right]$. Заметим, что композиция $F(x) = \sum\limits_{t=1}^{T} \alpha_t b_t(x)$, которую строит алгоритм *AdaBoost*, принадлежит семейству алгоритмов $\mathcal{C}$. Следовательно теорема 7 является обоснованием высокой эффективности *AdaBoost*. Ещё одним из методов, для которых важна теорема 7, является *градиент бустинг* (будет описан в разделе 1.4).

1.4 Градиентный бустинг

1.4.1 Постановка задачи

Алгоритм градиентного бустинга способен создавать качественные, устойчивые к выбросам модели для регрессии и классификации. Допустим, что мы хотим апроксимировать некоторую функцию $F(x)$, построить её оценку $\hat{F}(x)$. В нашем распоряжении имеется множество пар $\{(y_j, x_j)\}_{j=1}^{\ell}$ — обучающая выборка, где $x_j \in \mathbb{R}^n$ — признаковое описание j-го объекта, а y_j — результат на объекте x_j. Кроме того, мы указываем функцию оптерь $\Psi\left(y, F(x)\right)$. Задачу поиска функции $\hat{F}(x)$ можно поставить следующим образом:

$$\hat{F}(x) = \arg\min_{F(x)} E_{y,x}\Psi\left(y, F(x)\right) = \arg\min_{F(x)} E_x\left[E_y\Psi\left(y, F(x)\right)|x\right]. \tag{8}$$

Если $y \in \mathbb{R}^1$ (задача регрессии), то в качестве $\Psi\left(y, F(x)\right)$ можно использовать $(y-F)^2$ или $|y - F|$. Если $y \in \{-1, 1\}$ (задача классификации), то функци штрафа может принимать вид $\log\left(1 + e^{-2yF}\right)$ [2].

1.4.2 Метод наискорейшего спуска

В [2] предлагается использовать метод наискорейшего спуска для решения задачи (8). Будем рассматривать параметрическую модель $F(x; \mathbf{P})$, где $\mathbf{P} \in \mathbb{R}^K$, $\mathbf{P} = (P_1, \ldots, P_K)$. Обозначим $\Phi(F(x; \mathbf{P})) = E_{y,x}\Psi\left(y, F(x; \mathbf{P})\right)$. Решение ищется пошагово в виде $\mathbf{P}_m = \sum\limits_{j=0}^{m} \mathbf{p}_j$, где $\mathbf{p}_j$ — слагаемое в общем решении, полученное на j-й итерации алгоритма наискорейшего спуска. Градиент функции $\Phi(F(x; \mathbf{P}))$ на m-ом шаге алгоритма имеет вид:

$$g_m = \left[\frac{\partial\Phi(\mathbf{P})}{\partial P_i}\right]_{\mathbf{P}=\mathbf{P}_{m-1}}, \qquad \text{где } \mathbf{P}_{m-1} = \sum_{j=0}^{m-1} \mathbf{p}_j. \tag{9}$$

Новое слагаемое $\mathbf{p}_m$ на m-ом шаге имеет вид:

$$p_m = -\rho_m \mathbf{g}_m, \text{где } \rho_m = \arg\min_{\rho} \Phi\left(\mathbf{P}_{m-1} - \rho\mathbf{g}_m\right).$$

где, ρ_m — градиентный шаг, который предлагается искать методом наискорейшего спуска. Для данного алгритма введём параметр M — *количество итераций*. Эта величина будет определять размер построенной композиции $\mathbf{P} = \mathbf{P}_M = \sum\limits_{j=0}^{M} \mathbf{p}_j$.

1.4.3 Алгоритм решения оптимизационной задачи

Будем действовать по аналогии с методом наискорейшего спуска. Функцию $F(x)$ будем представлять в следующей аддтивной форме:

$$F(x;P) = \sum_{m=0}^{M} \beta_m h(x;a_m), \qquad \text{где } P = \{(\beta_m, a_m)\}_{m=0}^{M}. \tag{10}$$

Функцию $h(x;a_m)$ в (10) будем называть *базовым алгоритмом*. Обычно, $h(x;a_m)$ является некоторой простой функцией от x (в том числе может быть и решающим деревом [7]). Представление (10) функции $F(x;P)$ можно записать в рекурентном виде:

$$F_m(x) = F_{m-1}(x) + \beta_m h(x;a_m). \tag{11}$$

По уравнению (10) видно, что теперь оптимизационная задача (8) состоит в нахождении параметров $\{(\beta_m, a_m)\}_{m=0}^{M}$. Будем решать задачу (8) итерационно. На каждой новой итерации будем строить новую аддитивную добавку $\beta_m h(x;a_m)$ к уже построенному решению. Теперь задача (8) может быть записана в новом виде:

$$(\beta_m, a_m) = \arg\min_{\beta,a} \sum_{j=1}^{\ell} \Psi\left(y_j, F_{m-1} + \beta h(x_j;a)\right). \tag{12}$$

Заметим, что $h(x;a_m)$ — есть некоторый представитель определённого класса функций $\{h(x;a)\}$, который мы заранее выбираем для построения решения. Каждый такой класс обладает индивидуальными особенностями и ограничения. В нашем случаем мы будем использовать каждый конкретный представитель класса $h(x;a_m)$ в виде аналога градиента (9). То есть, на каждой итерации мы будем строить такой базовый алгоритм $h(x;a_m) \in \{h(x;a)\}$, который лучше всего аппроксимирует отрицательный градиент (9). Для построения нового базового алгоритма с таким свойством на m — ом шаге алгоритма бустинга будем находить параметр a_m как решение задачи минимизации (13).

$$a_m = \arg\min_{a,\beta} \sum_{j=1}^{\ell} \left(-g_m(x_j) - \beta h(x_j;a)\right)^2. \tag{13}$$

Далее ищем градиентный шаг и вычисляем значение $F_m(x)$:

$$\rho_m = \arg\min_{\rho} \sum_{j=1}^{\ell} \Psi\left(y_j, F_{m-1} + \rho h(x_j;a_m)\right).$$

$$F_m(x) = F_{m-1}(x) + \rho_m h(x;a_m).$$

В итоге, алгоритм можно представить в виде Алгоритма 2:

Алгоритм 2 Общий алгоритм градиентного бустинга.

Вход: Обучающая выборка $\{(x_j, y_j)\}_{j=1}^{\ell}$, семейство базовых алгоритмов $\{h(x;a)\}$, функция потерь $\Psi(y,x)$.

Выход: Модель прогноза $F(x)$.

1: $F_0(x) = \arg\min\limits_{\rho} \sum\limits_{j=1}^{\ell} \Psi(y_j;\rho)$

2: **для всех** $m \in \{1,\dots,M\}$

3: $\quad \tilde{y}_j = -\left[\frac{\partial\Psi(y_j,F(x_j))}{\partial F(x_j)}\right]_{F(x)=F_{m-1}(x)} \Longrightarrow \{\tilde{y}_j\}_{j=1}^{\ell}$

4: $\quad a_m = \arg\min\limits_{a,\beta} \sum\limits_{j=1}^{\ell} \Psi\left(-g_m(x_j) - \beta h(x_j;a)\right)^2$

5: $\quad \rho_m = \arg\min\limits_{\rho} \sum\limits_{j=1}^{\ell} \Psi\left(y_j, F_{m-1} + \rho h(x_j;a_m)\right)$

6: $\quad F_m(x) = F_{m-1}(x) + \rho_m h(x;a_m)$

7: $F(x) = F_M(x)$

1.5 Решающие деревья

1.5.1 Общий вид деревьев

Решающее дерево ($DecisionTree, DT$) — один из логических алгоритмов классификации или регрессии. Он основывается на поиске конъюнктивных закономерностей. Решающее дерево может быти представлено с помощью графа, который является ориентированным (направленным) деревом.

Деревом в теории графов называется конечный связный граф с множеством вершин $V = \{v_j\}$, в котором нет циклов, и есть одна выделенная вершина $v_0 \in V$, в которую не входит ни одно ребро (такая вершина называется *корнем* дерева). Вершина, не имеющая выходящих ребер, называется *листом*. Остальные вершины называются *внутренними*.

Мы будем рассматривать бинарные решающие деревья. Дерево (граф) называется *бинарным*, если из каждой его внутренней вершины выходит ровно два ребра. Соответственно, для любой внутренней вершины $v \in V$ есть две *дочерние* вершины: L_v — левая и R_v — правая.

Опр. 1.8. ***Бинарное решающее дерево*** *— это алгоритм классификации или регрессии, задающийся бинарным деревом, каждой внутренней вершине $v \in V$ которого соответствует предикат $\beta_v \colon X \to \{0,1\}$. Каждой терминальной вершине $v \in V$ приписан некоторый ответ $A_v \in Y$.*

Обозначим $X = \{x\}$ как пространство объектов, а Y — пространство ответов. Объект $x \in X$, который классифицируется решающим деревом, проходит путь по дереву путь от корня до некоторого листа в соответствии с Алгоритмом 3. Он доходит до вершины v тогда и только тогда, когда конъюнкция $K_v(x)$ выполняется. $K_v(x)$ — составленна из всех придикатов, приписанных внутренним вершинам дерева на пути от корня v_0 до вершины v. Предикаты $\beta_v(x)$ в этой конъюнкции (и, следовательно, в вершинах графа) могут представлять собой индикаторные функции различных событий, например : $[x_j < \theta]$, $[\alpha < x_j < \beta]$, $[x_j \in \{1,2,3\}]$, где x_j —

Алгоритм 3 Классификация (или регрессия) объекта $x \in X$ бинарным DT.

$v = v_0$
пока (вершина v - внутренняя)
  если $\beta_v(x) = 1$ **то**
    $v = R_v$
  иначе
    $v = L_v$
вернуть A_v

j-й признак объекта x.

1.5.2 ODT деревья

ODT — *Oblivious Decision Tree*. Особенность ODT деревъев заключается в том, что для каждой вершины такого дерева количество вершин в левом и правом поддереве одинаково. Пример ODT дерева приведен на Рис. 1.

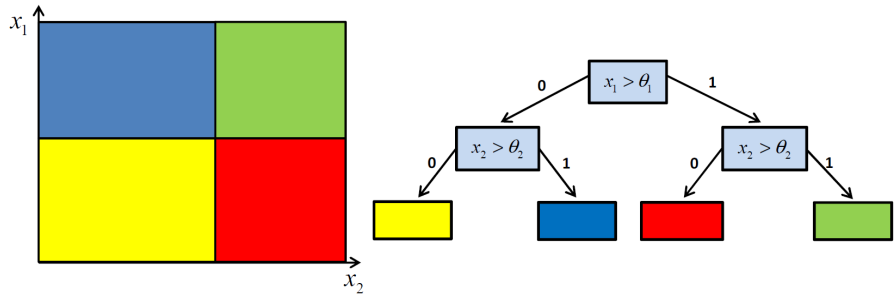

Рис. 1: Пример ODT дерева высоты $h = 2$.

В данной работе будем рассматривать только ODT, имеющие ещё некоторые особенности:

1. На каждом уровне дерева рассматривается только один признак x_j.
2. Во всех вершинах, принадлежащих одному уровню дерева, находится одно и то же условие, имеющее вид $[x_j > \theta]$, где θ — значение порога.

Для более точного формального описания ODT дерева введём следующие обозначения:

- h — высота дерева (длина пути от вершины до любого листа равна h);
- $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^N$ — признаковое описание объекта;
- $T(\vec{x})\colon \mathbb{R}^N \to \{0,1\}^h$ — функция, представленная в виде решающего дерева. $T(\vec{x})$ для каждого $\vec{x}$ задаёт путь от вершины к некоторому листу.

Опишем процесс вычисления функции $T(\vec{x})$. По набору признаков в $\vec{x}$ можно прийти в некоторый лист дерева. По мере двидения по дереву (от вершины к листьям) строится бинарный вектор $\vec{b} \in \{0,1\}^h$. В каждой вершине условие $[x_j > \theta]$

определяет дальнейший шаг вниз по дереву, в левую дочернюю вершину или в правую. Каждый элемент в $\vec{b}$ соответствует одному уровню в решающем дереве, первый элемент воответствует первому уровню дерева, в котором содержится только вершина. Если $[x_j > \theta] = 1$, то элемент ветора $\vec{b}$, соответствующий данному уровню дерева приравнивается едининице, иначе нулю. Таким образом, на k - ом уровне дерева в некоторой вершине, в которую мы пришли для данного объекта $\vec{x}$, мы получаем k - ю координату бинарного вектора $\vec{b}$. Получаем, что вектор $\vec{b}$ описывает некоторый путь в дереве от вершины до листа.

У ODT дерева высоты h есть 2^h листовых вершин. К каждой листовой вершине, соответствующей некоторому бинарному вектору $\vec{y} \in \{0,1\}^h$, припишем число $R(\vec{y})$ — ответ ODT дерева в листе. Таким образом, введем обозначения:

- $R(\vec{y})\colon \{0,1\}^h \to \mathbb{R}$ где $\vec{y} = T(\vec{x}), \vec{y} \in \{0,1\}^h$;
  $R(\vec{y})$ — функция, которая для j - го дерева даёт ответ в листе, соответствующем пути, описываемому бинарным вектором $\vec{y}$.

- $A(\vec{x}) = R\left(T(\vec{x})\right) : \mathbb{R}^N \to \mathbb{R}$;
  $A(\vec{x})$ — ответ решающего дерева на обьекте $\vec{x}$.

1.6 Строение модели MatrixNet

Все деревья в композиции занумеруем от 1 до N (N — общее количество деревьев) в порядке их создания при работе алгоритма градиентного бустинга. Обозначим j как порядковый номер дерева в композиции. Соответственно, к функциям, определенным в пункте (1.5.2) добавим нижний индекс j. Этот индекс будет идентифицировать дерево, которое описывает указываемая функция. В общем виде модель, получаемую при градиентном бустинге, можно представить в виде:

$$F(\vec{x}) = \sum_{j=1}^{N} \alpha_j R_j\left(T_j(\vec{x})\right) = \sum_{j=1}^{N} \alpha_j A_j(\vec{x}). \tag{14}$$

Все деревья в модели $MatrixNet$ имеют одинаковый вес. В соответствии с этим, можно положить, что $\alpha_j = 1\ \forall j \in \{1, \ldots, N\}$. Ведём новые обозначения:

Опр. 1.9. *Вид регрессионной модели $MatrixNet$.*

$$F(\vec{x}) = \sum_{j=1}^{N} R_j\left(T_j(\vec{x})\right) = \sum_{j=1}^{N} A_j(\vec{x}). \tag{15}$$

Заметим, что для построения модели $MatrixNet$ используются только ODT деревья. В первую очередь, это связано с программынми оптимизация, которые можно применить к процедурам постреления, хранения и вычисления функции $F(x)$. Класс ODT деревьев намного уже класса всех деревьев DT. Такой способ выбора базовых алгоритмов приводит к тому, что на этапе построения новой функции $A_j(x)$ нам приходится вибирать из меньшего множества более простых функций, что приводит к некоторого рода регуляризации модели и улучшению обобщающей способности. Помимо этого, в $MatrixNet$ высота строящегося дерева монотонно зависит от его порядкового номера j. Сначала строятся деревья с небольшой высотой, далее высоты

увеличиваются до максимально значения, равного шесть. Вид зависимости $h_j = h(j)$ можно увидеть в пункте (3.3).

1.7 Постановка задачи упрощения композиции

На главной странице Яндекса *www.yandex.ru* каждый пользователь может задавать любые поисковые запросы. Результат, который получает пользователь можно разделить на две части. Первая часть - *поисковая выдача.* Это множество ссылок на интернет ресурсы, вязанные с запросом. Вторая часть - *рекламная выдача.* Это множество текстовых обьявлений, которые могли бы заинтересовать человека, вводившего поисковой запрос. Множество элементов в каждой выдаче должно быть отранжировано специальным образом в соответствии с некоторым числовым рангом, присвоенном каждому элементу. Сам алгоритм ранжирования должен располагать элементы таким образом, что бы удовлетворялись нужды пользователей и самого Яндекс одновременно.

В данной работе рассматриваются некоторые проблемы возникающие при формировании только рекламной выдачи. Перед ранжированием для каждого элемента нужно вычислить его ранг. Для рекламных объявлений в качестве ранга, в данной работе, используется величина CTR.

Опр. 1.10. ***CTR*** *(Click-through rate) — вероятность клика по некоторому элементу.*

Яндекс работает с тысячами рекламодателей, желающих размещать свою рекламу на главной странице. На каждого рекламодателя может приходиться по несколько рекламных объявлений. Соответственно, на каждый поисковой запрос Яндекс требуется вычислять CTR для сотен тысяч рекламных обьявлений, что требует затраты значительных вычислительных рессурсов. Число пользователей Яндекс с каждым годом сильно увеличивается, что приводит к возникновению проблемы производительности при вычислении CTR. Эта проблема является важной, так как время формирования главной старницы существенно влияет на интерес рекламодателей и простых пользователей к поисковику. Задержки в выдаче могут привести к уменьшению количества пользователей и к переходу рекламодателей к конкурентам. Для предсказания CTR в Яндекс ипользуется технология(алгоритм) $MatrixNet$ [4]. Результатом работы этого алгоритма является модель (1.9), представляющая собой сумму N слагаемых. Вычисление значения CTR линейно зависит от величины N. Соответственно, уменьшение значения N влечет за собой увеличение скорости вычисления CTR и, как следствие, увеличение скорости подготовки главной страци. С уменьшением N, уменьшается время отклика поисковика на запрос.

В данном исследовании предложены методы для уменьшения размера композиции (1.9). Рассотрены два случая. В первом случае имеется исходная композиция размера N, после оптимизации на выходе получаем композицию размера $M < N$ при условии, что качество предсказания CTR не должно уменьшиться. Во втором случаем на размер комопзиции после оптимизации заранее накладывается услови $M < M_0$, и требуется построить такую оптимизированную композицию, которая имеет наименьшие потери в качестве предсказания CTR.

2 Способы упрощения композиций

2.1 Виды элементарных экспериментов с моделями MatrixNet

Используем нумерацию деревьев в композиции, приведённую в пункте 1.6. Если выбросить первые k деревьев из линейной комбинации, то молучаем модель:

$$F(\vec{x}) = \sum_{j=k+1}^{N} A_j(\vec{x}). \tag{16}$$

Если выбросить последние k деревьев из линейной комбинации, то молучаем модель:

$$F(\vec{x}) = \sum_{j=1}^{N-k} A_j(\vec{x}). \tag{17}$$

Если удалить одно дерево с номером k, то модель принимает вид:

$$F(\vec{x}) = \sum_{\substack{j=1 \\ j\neq k}}^{N} A_j(\vec{x}). \tag{18}$$

2.2 Упрощение композиции с использованием логистической регрессии с L_1 регуляризацией

В статье "Predictive Learning via Rule Ensembles" [1] рассматриваются различные методы построения линейных моделей для регрессии и классификации. Пусть, $x = (x_1, \ldots, x_n)$ — признаковое описание объекта, S_j - множество всех значений, которые может принимать признак x_j, $x_j \in S_j$ $\forall j \in \{1, \ldots, n\}$. Пусть $S_{jm} \subset S_j$. Базовые алгоритмы в композиции представляют собой решающие деревья, которые можно интерпретировать как наборы правил. Каждое такое правило $r_m(x)$ может быть представлено в виде:

$$r_m(x) = \prod_{j=1}^{n} I\left(x_j \in S_{jm}\right).$$

Основное внимание авторами уделяется методам выделения в композиции наиболее важных правил, приведённого выше вида, которые должны быть хорошо интерпретируемыми. Хочется получать линейную модель, в которой есть только значимые слагаемые, и каждое слагаемое может быть легко понимаемо человеком. Кроме того, предлагаются методы для выявления связей и взаимодействия между переменными $(x_1, \ldots, x_n)$. Получив некоторым методом композицию из решающих деревьев, каждое дерево можно разложить на множество правил. Соответственно, из множества решающих деревьев получается ансамбль правил $\{r_k(x)\}_{k=1}^{K}$. Далее, модель для предсказания записывается в виде:

$$F(\vec{x}) = \hat{\alpha}_0 + \sum_{k=1}^{K} \hat{\alpha}_k r_k(\vec{x}),\ D = \{(\vec{x}_j, y_j)\}_{j=1}^{\ell} - \text{обучающая выборка.}$$

$$\{\hat{\alpha}_k\}_{k=0}^{K} = \arg\min_{\{\alpha_k\}_{k=0}^{K}} \sum_{j=1}^{\ell} L\left(y_j, \alpha_0 + \sum_{k=1}^{K} \alpha_k r_k(\vec{x}_j)\right) + \lambda \sum_{k=1}^{K} |\alpha_k|. \tag{19}$$

Отметим важную вещь, что в формуле 19 присутствует штрафное слагаемое (*lasso*). Для различных функций потерь $L(y, F)$ штраф *lasso* привносит в метод оптимизации, применённый в работе, автоматический отбор привил вида $r_m(x)$ путём обнуления некоторых коэффициентов $\hat{\alpha}_k$. Это свойство активно применяется в работе, приносит свои результаты отбора признаков.

В текущей работе в качестве базового алгоритма так же используются решающие деревья. Они имеют специальный вид для ускорения процесса обучения модели, поэтому не можем рассматривать деревья произвольной высоты и с произвольным видом условий (см. пункт 1.6). Мы абстрагируемся от хорошей интерпретируемости каждого базового алгоритма и заострим внимание на сложности композиции алгоритмов. Так же, в данной работе для нас важно поведение различных функционалов качества модели, таких как: $LL, MSE, LinCorr$. В отличие от работы [1] нам важно сохранить каждое решающее дерево целостным, не разделяя его на отдельные правила $r_m(x)$. В данной работе оно является неделимым элементом композиции. Таким образом, в текущей работе мы тоже будем пользоваться свойствами регуляризации типа *lasso* для отсеивания мало значимых базовых алгоритмов. В качестве функции штрафа возьмём логистическую функцию.

Логистическая регрессия с L_1 регуляризацией имеет свойство отбирать признаки. Будем считать каждое дерево в композиции одним признаком в новой линейной модели, которую мы будем далее обучать. Для обучения данной модели нам нужно уметь создавать обучающую выборку. Заметим, что при обучении начальной (исходной) модели методом градиентного бустинга у нас была обучающая выборка вида:

$$X_{old} = \left\{\left(\vec{x}_j^{old}, y_j^{old}\right)\right\}_{j=1}^{\ell},\ y_j^{old} \in \{0, 1\}.$$
$$\vec{x}_j^{old} \text{ — вектор исходных признаков, } \dim(\vec{x}_j^{old}) = 65.$$

Теперь, построим новое признаковое пространство и используем логистическую регрессию:

- $\vec{x}_j^{new}$ — вектор ответов от деревьев в уже построенной композиции, $\dim \vec{x}_j^{new} = N$;
- N — количество деревьев в композиции.

$$X_{new} = \left\{\left(\vec{x}_j^{new}, y_j^{new}\right)\right\}_{j=1}^{\ell},\ y_j^{new} \in \{-1, 1\}.$$
$$y_j^{new} = \begin{cases} -1, & \text{если } (y_j^{old} = 0); \\ 1, & \text{если } (y_j^{old} = 1). \end{cases}$$

Заметим, что $\vec{x}_j^{new}$ — функция от $\vec{x}_j^{old}$. Введём обозначения:

- $\left(\vec{x}_j^{new}\right) = R_k\left(T_k(\vec{x}_j^{old})\right) = A_k\left(\vec{x}_j^{old}\right)$;
- j — номер объекта в обучающей выборке;

- k — номер признака в новом признаковом описании.

Таким образом, получаем следующую модель:

$$\xi_{j,\vec{x}} = \xi_j(\vec{x}) = R_j\left(T_j(\vec{x})\right) = A_j(\vec{x})\ j = \overline{1,N};$$
$$G(\vec{x}) = \sum_{j=1}^{N} \beta_j \xi_j(\vec{x}) = \left(\vec{\beta}, \vec{\xi}(\vec{x})\right),\ \vec{\beta} = const;$$
$$p(\vec{x}) = \frac{\exp G(\vec{x})}{1+\exp G(\vec{x})} = \frac{1}{1+\exp(-G(\vec{x}))}.$$

Минимизируемый функционал имеет вид:

$$f(\vec{\beta}, C) = C\sum_{j=1}^{\ell} \log\left(1 + \exp\left(-y_j G(\vec{x}_j)\right)\right) + \|\vec{\beta}\|_1 \tag{20}$$

$$G(\vec{x}) = \left(\vec{\beta}, \vec{\xi}(\vec{x}_j)\right),\ \|\vec{\beta}\|_1 = \sum_{j=1}^{N} |\beta_j|$$

$$f(\vec{\beta}, C) \to \min_{\vec{\beta}},\ \vec{\beta}^*_C = \arg\min_{\vec{\beta}} f(\vec{\beta}, C),\ \vec{\beta}^*_C = (\beta^*_{C,1}, \ldots, \beta^*_{C,1}) \tag{21}$$

При обучении новой формулы мы используем ту же самую обучающую выборку, что и при обучении первоначальной модели с использованием MatrixNet. При обучении мы можем регулировать параметр C. Он будет влиять на количество отобранных признаков и на обобщающую способность модели (величину LL на тестовой выборке).

2.3 Упрощение композиции с использованием L_2 регуляризации

2.3.1 Постановка задачи. Неоптимальность композиции с ростом числа деревьев

При построении исходной композиции деревьев методом градиентного бустинга все веса деревьев равны единице. То есть, в процессе работы алгоритма бустинга эти веса не меняются и остаются таковыми в результирующей композиции. В итоге, такая жесткая фиксация весов может оказаться неоптимальной с точки зрения обобщающей способности получающейся модели. Следовательно, появляется возможность улучшить обобщающую способность модели посредством переподбора весов деревьев. Более того, проведённые эксперименты с L_1 регуляризацией показали, что из композиции можно убрать подавляющее большинство решающих деревьев, порядка 90%. Множество важных деревьев – есть некоторое тривиальное подмножество всех деревьев композиции. При применении L_1 регуляризации можно считать, что все деревья независимы. Таким образом, алгоритм градиентного бустинга, в данной ситуации, можно воспринимать как алгоритм, который генерирует некоторое множество строительных блоков. Эти блоки допускается считать независимыми. Далее из этих блоков можно строить новые алгоритмы намного меньшей сложности, качество которых может конкурировать с качеством исходной модели градиентного бустинга.

Теперь, возникает возможность использования L_2 регуляризации. Логарифм функции правдоподобия можно записать в следующем виде:

$$LL(\vec{\beta}) = \sum_{j=1}^{\ell} \log\Big(1 + \exp(-y_j G(\vec{\beta}, \vec{x}_j))\Big)$$

Введём обозначения:

$$G(\vec{\beta}, \vec{x}_j) = \beta_0 + \Big\{\beta_1 \vec{\xi_1}(\vec{x}_j) + \cdots + \beta_N \vec{\xi_N}(\vec{x}_j)\Big\}$$
$$\vec{\beta} = (\beta_0, \beta_1, \ldots, \beta_N)$$
$(N+1)$ - сложность оптимизируемой модели.

$$\left[\begin{array}{l} LL_{train} \Longleftrightarrow \{(y_j, \vec{x}_j)\}_{j=1}^{\ell} - \text{обучающая выборка} \\ LL_{test} \Longleftrightarrow \{(y_j, \vec{x}_j)\}_{j=1}^{\ell} - \text{тестовая выборка} \end{array}\right.$$

Теперь задачу можно записать в следующем виде:

$$\begin{gathered} f(\vec{\beta}, \lambda_2) = LL_{train}(\vec{\beta}) + \tfrac{\lambda_2}{2}\|\vec{\beta}\|_2,\ \|\vec{\beta}\|_2 = \sum_{j=0}^{N} |\beta_j|^2 \\ f(\vec{\beta}, \lambda_2) \to \max_{\vec{\beta}} \Longrightarrow \vec{\beta}^*_{\lambda_2} = \arg\max_{\vec{\beta}} f(\vec{\beta}, \lambda_2 = const) \end{gathered} \tag{22}$$

каждой конкретной модели, в данном случае, нам хочется знать следующее значение: $LL_{test}(\vec{\beta}^*_{\lambda_2^*})$ - оценка обобщающей способности оптимизируемой модели.

$$\lambda_2^* = \arg\max_{\lambda_2} LL_{test}(\vec{\beta}_{\lambda_2}) \tag{23}$$

Фактически $LL_{test}(\vec{\beta}_{\lambda_2^*})$ - есть функция, зависящая только от параметра λ_2, следовательно, можно переобозначить $g(\lambda_2) = LL_{test}(\vec{\beta}_{\lambda_2^*})$. Теперь, задачу (23) можно переписать как:

$$x^* = \arg\max_x g(x) \tag{24}$$

Эксперименты, проведённые с использованием L_1 регуляризации, показали, что $g()$ является унимодальной функцией. Задача (24) - есть задача одномерной оптимизации. Следовательно, для поиска оптимального значения x^* можно, например, использовать метод золотого сечения.

2.3.2 Метод золотого сечения

Псевдокод применённого алгоритма:

Алгоритм 4 Поиск максимума функции на отрезке, методом золотого сечения.

Вход: N - количество итераций, $[a, b]$ - предполагаемый отрезок локализации решения ($x^* \in [a, b]$), ε - точность локализации решения.

Выход: Точка x^*, в которой $g(x)$ достигает максимума.

$\phi = \frac{\sqrt{5}-1}{2}$
пока $|b - a| \geq \varepsilon$
  $x_1 = a + (b - a)(1 - \phi)$
  $y_1 = g(x_1)$
  $x_2 = b - (b - a)(1 - \phi)$
  $y_2 = g(x_2)$
  если $y_1 > y_2$ **то**
    $a = x_1$
  иначе
    $b = x_2$
  если $|b - a| < \varepsilon$ **то**
    $x^* = \frac{a+b}{2}$

3 Экспериментальное исследование

3.1 Задача предсказания клика

Место для показа рекламных баннеров на главной странице Яндекса условно можно разделить на 3 части (слота): *premium, guarantee, dynamic*. Введём определение:

Опр. 3.1. ***Хитом*** *называется множество всех рекламных объявлений, показываемых пользователю, в ответ на его поисковый запрос со всех трёх слотов: premium, guarantee, dynamic.*

Слот *premium* расположен над результатами поисковой выдачи на главной странице Яндекса. Слоты *guarantee* и *dynamic* расположены справа от результатов поисковой выдачи. Слот *guarantee* расположен над слотом *dynamic*.

Для принятия решения о том, какие именно баннеры будут показаны пользователю по данному запросу, строится модель, оценивающая вероятность клика по баннеру. Практический опыт показывает, что нужно строить различные модели для различных слотов. Для слотов *guarantee* и *dynamic* будем пользоваться одной моделью, общей для них. Введём следующие обозначения:

- x — признаковое описание события, показа баннера;
- D — обучающая выборка событий (множество все признаковых описаний объектов);
- $c(x) : x \to \{0, 1\}$ — классификация события $x \in D$;
- $c(x) = \begin{cases} 1, & \text{если при показе был клик,} \\ 0, & \text{если при показе клика не было;} \end{cases}$
- $F(x)$ — оценка (формула прогноза) принадлежности события x к классу 1.

В виде функции качества, которую будем оптимизировать для получения модели, хочется использовать хорошо интерпретируемую функцию. Положим, что каждый документ x классифицируется вероятностно независимо от других в зависимости от функции оценки $F(x)$ следующим образом: вероятность попасть в первый класс равна логистическому преобразованию от функции оценки, то есть $1/(1+\exp(-F(x)))$. Хочется максимизировать вероятность полностью правильной классификации на обучающей выборке.

Опр. 3.2. ***Оценка вероятности клика*** *по баннеру.*

$$p(x) = \frac{1}{1 + exp(-F(x))}. \tag{25}$$

Вероятность правильной классификации для одного события x равна $p(x)^{c(x)}\left(1-p(x)\right)^{1-c(x)}$. Тогда, функцию правдоподобия (4) можно записать следующим образом:

$$W = \prod_{x \in D} p(x)^{c(x)}\left(1-p(x)\right)^{1-c(x)}.$$

Тогда, функция логарифма правдоподобия (5) принимает вид:

$$LL = \frac{1}{|D|}\sum_{x \in D}\left(c(x)\ln\left(p(x)\right) + \left(1-c(x)\right)\ln\left(1-p(x)\right)\right). \tag{26}$$

Так как, $\ln()$ — строго возрастающая функция, то максимизация функции правдоподобия равносильна максимизации $LL()$. В итоге получаем, что задача предсказания клика по баннеру имеет вид:

$$D = \{(x_j, y_j)\}_{j=1}^{\ell},\ y_j = c(x_j),\ y_j \in \{0,1\};$$

$$\sum_{j=1}^{\ell}\left(y_j \ln\left(p(x_j)\right) + \left(1-y_j\right)\ln\left(1-p(x_j)\right)\right) \to \max_F. \tag{27}$$

Статистики качетсва могут быть коррелированы, однако из этого не следует что малое изменение одной статистики влечёт за собой изменение другой. Поэтому, мы получаем дополнительную свободу в ранжировании моделей по их качеству. Кроме того, возможны ситуации, когда какая-либо метрика качества (например, MSE или $LinCorr$) для задачи ведёт себя лучше, чем целевая метрика. Помимо LL для оценки качества формулы прогноза будем использовать величины MSE (2) и $LinCorr$ (3). Для задачи предсказания клика они имеют следующий вид:

$$MSE = \frac{1}{|D|}\sum_{x \in D}\left(c(x) - p(x)\right)^2. \tag{28}$$

$$\begin{gathered} LinCorr = \frac{\sum\limits_{x \in D}(p(x)-\overline{p})(c(x)-\overline{c})}{\sqrt{\sum\limits_{x \in D}(p(x)-\overline{p})^2 \sum\limits_{x \in D}(c(x)-\overline{c})^2}}; \\ \overline{p} = \frac{1}{|D|}\sum_{x \in D} p(x), \qquad \overline{c} = \frac{1}{|D|}\sum_{x \in D} c(x). \end{gathered} \tag{29}$$

Для максимизации функционалов, таких как MSE или LL, используется градиентный бустинг [2, 3], строящийся над решающими деревьями ODT [7]. В данной работе используется алгоритм MatrixNet [4].

3.2 Данные для обучения и тестирования

Лог – множество записей о показах баннеров. На один показ баннера приходится одна запись в логе. Каждая запись содержит некоторое признаковое описание показа. Данные для обучения взяты из лога за неделю с 16 по 22 апреля 2012 года. Количество событий - сто тысяч. Данные для теста взяты из лога за следующую неделю с 23 по 29 апреля 2012 года. Количество событий - сто тысяч. Таким образом, тестовые данные, отобранные таким образом, помогают более качественно проверить качество предсказания. Для обучения и теста брались события из слотов *guarantee* и *dynamic* для одного хита.

3.3 Параметры обучения

На качество обучения LL могут влиять следующие переметры.

1. i – кол-во деревьев в композиции.

2. n – максимальная высота дерева в композиции.

3. w – параметр регуляризации, определяющий длину шага при градиентном спуске.

4. g – влияет на распределение весов деревьев в модели.

$$h_j = min\left\{\left\lfloor\frac{j}{g}\right\rfloor, n\right\}.$$

Оптимальными для обучения MatrixNet, считаем такие параметры, при которых получается максимальное LL на тестовой выборке. Для поиска таких параметров устроим перебор их значений в некотором диапазоне: $i \in I, w \in W$. Остальные параметры имеют значения $\{n = 6, x = 16, g = 1\}$ и не изменяются.

$$I = \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$$
$$W = \{0.01, 0.03, 0.05, 0.1, 0.2, 0.3\}$$

Из всех наборов параметров выберем только два самых лучших (таблица 1), в смысле значения LL.

Параметры	LL	MSE	$LinCorr$
$i = 300$ $w = 0.03$	-0.03564070	0.00605759	0.06425760
$i = 1000$ $w = 0.01$	-0.03560650	0.00605643	0.06565240

Таблица 1: Параметры обучения для двух лучших значений LL.

3.4 Результаты элементарных экспериментов с моделями MatrixNet

3.4.1 Описание величин, отображаемых на графиках

Пусть $Stat$ — имя одной из статистик качества. $Stat \in \{LL, MSE, LinCorr\}$. В каждом эксперименте производятся модификации моделей посредством удаления из них деревьев различными способами. После каждого эксперимента мы получаем множество $\{Stat_k\}_{k=1}^{N}$, где $Stat_k$ - значение функции $Stat$ для изменённой модели, а k — идентификатор модифицированной модели в эксперименте. Конкретный смысл индекса k для каждого эксперимента определяется отдельно и будет описываться в разделе для соответствующих экспериментов. $Stat$ вычисляется на обучающей выборке. $Stat_0$ - значение $Stat$ при использовании неизменённой (изначальной) модели. Так как, величины $\{LL, MSE, LinCorr\}$ имеют разные масштабы изменения, то целесообразно всех их отобразить в один и тот же интервал. В качестве такого интервала будем использовать отрезок $[0, 100]$. Введём обозначения:

$$Stat_{max} = max\left\{Stat_k\right\}_{k=1}^{N}, \qquad Stat_{min} = min\left\{Stat_k\right\}_{k=1}^{N}.$$

Рассмотрим функцию: $f(Stat_k) = \frac{Stat_k}{Stat_0}$, $f\colon \mathbb{R}^- \to \mathbb{R}$ и сделаем над ней линейное преодразование $g(x) = \alpha f(x) + \beta$, такое что:

$$\begin{cases} g(Stat_{max}) = 100; \\ g(Stat_{min}) = 0. \end{cases}$$

Получаем, что нужное преобразование имеет вид $g_k = g(Stat_k) = 100\frac{Stat_{min} - Stat_k}{Stat_{min} - Stat_{max}}$. Заметим, что $g(Stat_k) \xrightarrow{Stat_k \to Stat_{max}} 100$, чем больше $Stat_k$, тем лучше качество классификации.

$$\begin{cases} g = 100 - \text{для лучшего качества;} \\ g = 0 - \text{для худшего качества.} \end{cases}$$

3.4.2 Результаты удаления деревьев, начиная с последнего дерева в модели

Для преобразования модели используем метод (17). Результаты отображены на Рис. 2. В даннном случаем величина k имеет смысл количества деревьев, удалённых из начала композиции. Проследив эти графики в обратном направлении от $j = 300$ до $j = 1$ можно увидеть как ведут себя статистики качества $LL, MSE, LinCorr$ при построении композиции с первого до последнего дерева. Из графика видно, что с ростом композиции все три величины ведут себя в целом монотонно. Можно видеть, что LL очень сильно коррелировано с MSE. LL и MSE ведут себя практически идентично. Так как, используемый алгоритм градиентного бустинга максимизирует значение LL, то графики говорят нам что с ростом композиции алгоритму всё труднее улучшать уже построенное решение. Это видно из того, что LL и MSE выходят на некоторую асимптоту. Однако, $LinCorr$ растёт примерно линейно и на асимптоту не выходит. Различие в поведении величин LL и $LinCorr$ связано с тем что LL — целевая функция для оптимизации, в отличае от $LinCorr$. Поведение LL имеет характерный вид для величины, оптимизируемой методом градиентного бустинга.

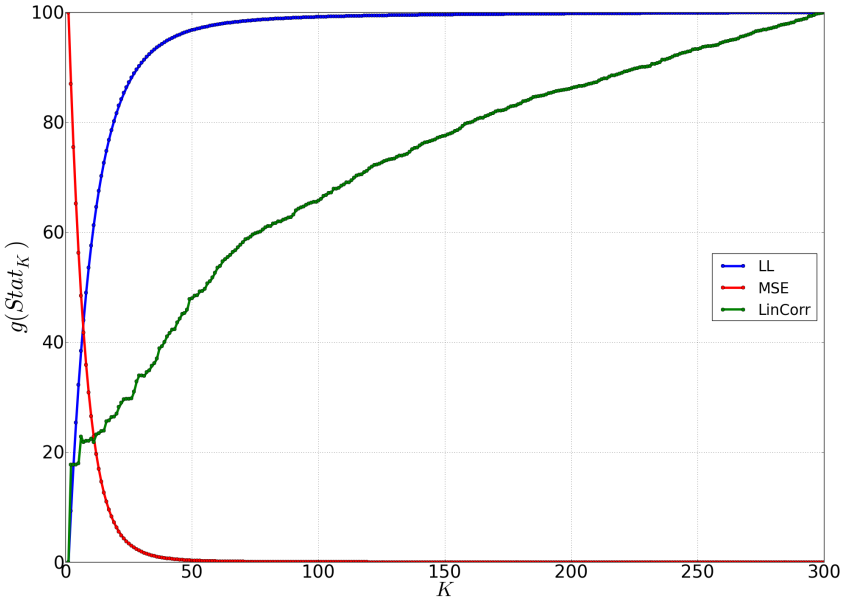

Рис. 2: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE, LinCorr\}$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при удалении деревьев, начиная с последнего.

На Рис. 2 видно, что с увеличением числа деревьев, удаляемых с начала, $g(LL_k)$ монотонно падает. Скорость изменения величины $g(LL_k)$ велика только на интервале $k \in [1, 50]$. Дальнейшее удаление деревьев из модели приводит к уменьшению $g(LL_k)$ до нуля. Заметим, что последние 250 деревьев отвечают за маленький рост $g(LL_k)$. Посмотрим, как ведут себя относительные изменения величины $g(Stat_k)$, $\Delta g_k = |g_k - g_{k+1}|$ для LL и $LinCorr$.

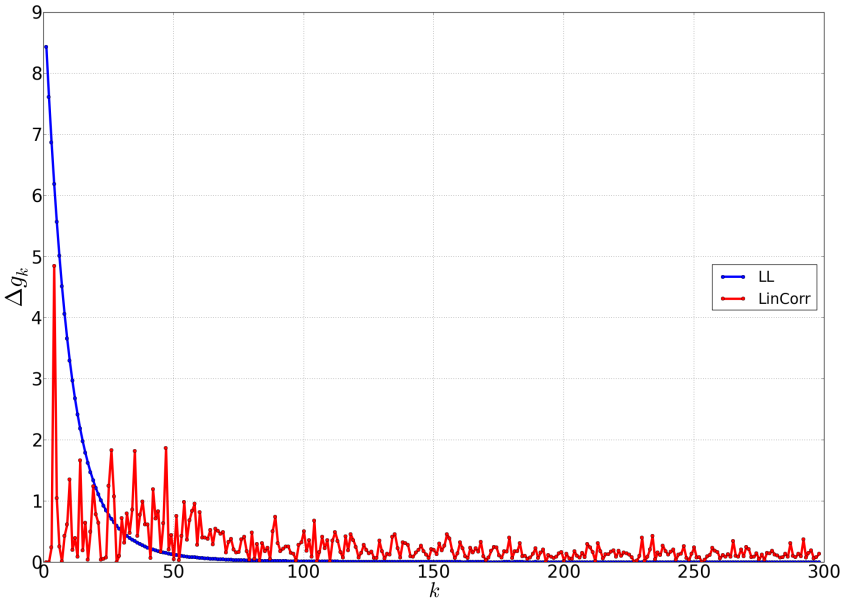

Рис. 3: Поведение $\Delta g(Stat_k)$ для $Stat \in \{LL, LinCorr\}$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при удалении деревьев, начиная с последнего.

На Рис. 3 видно, что для LL, начиная с 50-го дерева (в порядке их создания),

каждое новое дерево в композиции добавляет очень меленький относительный прирост величины $g(LL_k)$, причем, чем больше k , тем меньше $\Delta g(LL_k)$. Для $LinCorr$ такая монотонность не выполняется, тем не менее, величины Δg_k для такой статистики велики только на интервале $k \in [0, 50]$. На основе этого можно сделать предположение, что для данной модели деревья с номерами $k \in [1, 50]$ имеют значительно большее значение в композиции, чем остальные деревья, полученные на более поздних итерациях построения композиции.

Схожие результаты получаются при удалении деревьев из конца композиции из модели, состоящей из тысячи деревьев. Результаты изображены на Рис. 4 и 5. По данным результатам видно, что характер поведения величин $LL, MSE, LinCorr$ такой же как и у модели из трёхсот деревьев. Видно, что с ростом композиции алгоритму всё труднее улучшать уже построенное решение. Последние 90% слагаемых в композиции вносят очень малый прирост величины оптимизируемого функционала по сравнению с первыми 10% деревьев.

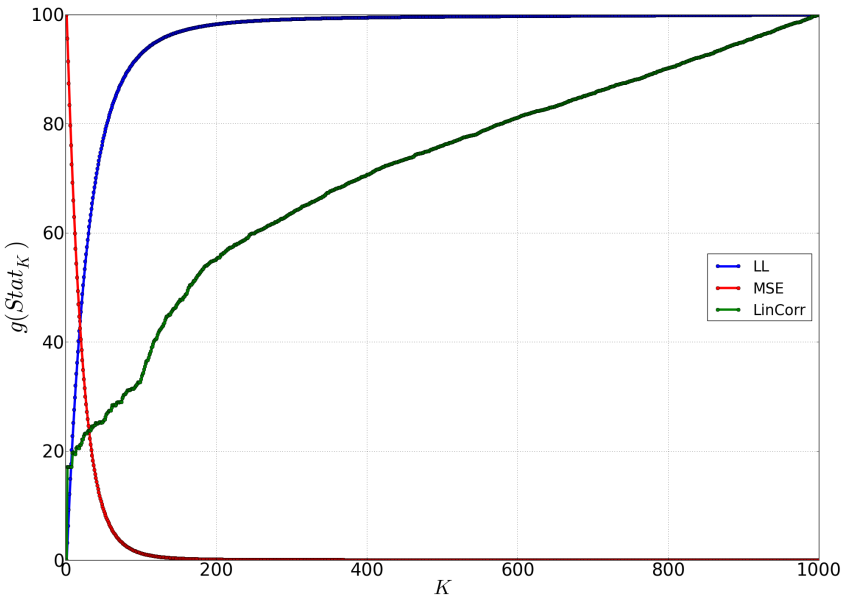

Рис. 4: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE, LinCorr\}$ для модели с параметрами $\{i = 1000\ w = 0.01\}$ при удалении деревьев, начиная с последнего.

3.4.3 Результаты удаления деревьев по одному из середины композиции

Для преобразования модели используем метод (18). Результаты отображены на Рис. 6. В даннном случаем величина k имеет смысл индекса удаляемого дерева. Можно видеть, что аналогично, сильно проявляется влияние первых 50-ти деревьев. На Рис. 6 отображено, что при удалении деревьев из интервала $k \in [1, 50]$, $g_{50} \approx -1$. Данный результат характеризует ценность отдельных деревьев в композиции, первые 50 деревьев имеют большую ценность, что подтверждает нашу гипотезу из пункта (3.4.2) о сильном влиянии первых деревьев в модели на качество классификации.

Схожие результаты получаются при применении элементарных преобразований к модели, состоящей из тысячи деревьев (Рис. 7).

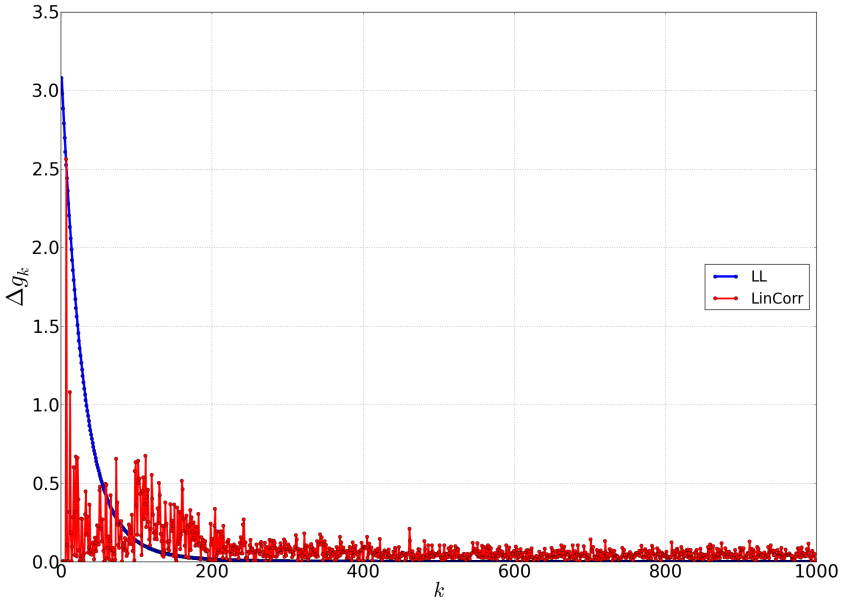

Рис. 5: Поведение $\Delta g(Stat_k)$ для $Stat \in \{LL, LinCorr\}$ для модели с параметрами $\{i = 1000\ w = 0.01\}$ при удалении деревьев, начиная с последнего.

3.4.4 Выводы по результатам проведённых экспериментов

Для моделей различной сложности получаем, что в среднем 10% деревьев из композиции вносят основной вклад в максимизацию LL. Улучшение качества LL после достроения каждого нового дерева монотонно падает на обучающей выборке. Можно сделать предположение, что многие из 90% деревьев, расположенных в хвосте композиции малозначимы во всей композиции и, как показывают приведённые графики, каждое по отдельности, вносят очень малый вклад в улучшение LL. Следовательно, есть потенциальная возможность упростить модель, уменьшив количество деревьев в композиции. При этом, при сокращении арзмеров композиции хочется минимизировать потерю в качестве LL.

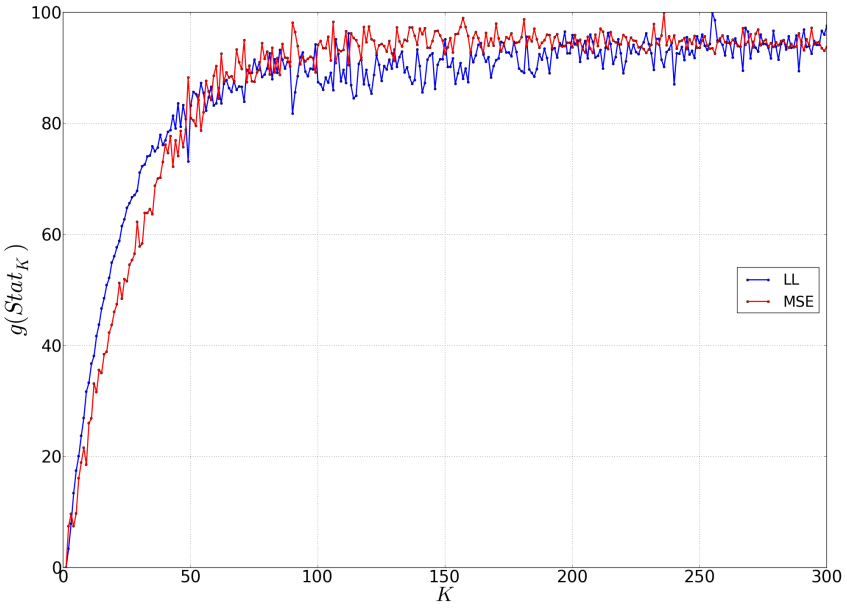

Рис. 6: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE\}$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при удалении деревьев по одному из середины модели.

3.5 Результаты упрощения композиции с использованием L_1 регуляризации

3.5.1 Описание величин, отображаемых на графиках

Исследуем, как будут вести себя различные показатели, в зависимости от параметра C. Введём обозначения:

- $M = M(C)$ — количество отобранных признаков (деревьев).
- N — исходное количество признаков.
- C — параметр регуляризации.
- $M_{\%}(C) = 100\frac{M(C)}{N}$ — количество отобранных признаков по отношению к общему числу признаков (измеряется в процентах).
- LL_0 — качество немодифицированной модели.
- LL_C — качество модели, полученной при решении задачи 21.
- $\Delta_{rel}(a, b) = -100\frac{a-b}{b}$ — измерается в процентах
  при $(a < 0)$ и $(b < 0)$ имеем, что $\{LL_{rel}(a, b) > 0\} \iff \{a > b\}$. Если взять $a = LL_C$ и $b = LL_0$, то $\{LL_{rel}(LL_C, LL_0) > 0\} \iff$ качество оптимизированной модели стало больше чем качество изначальной модели. То есть, во время оптимизации отбор наиболее значимых деревьев не приводёл к падению качества модели.

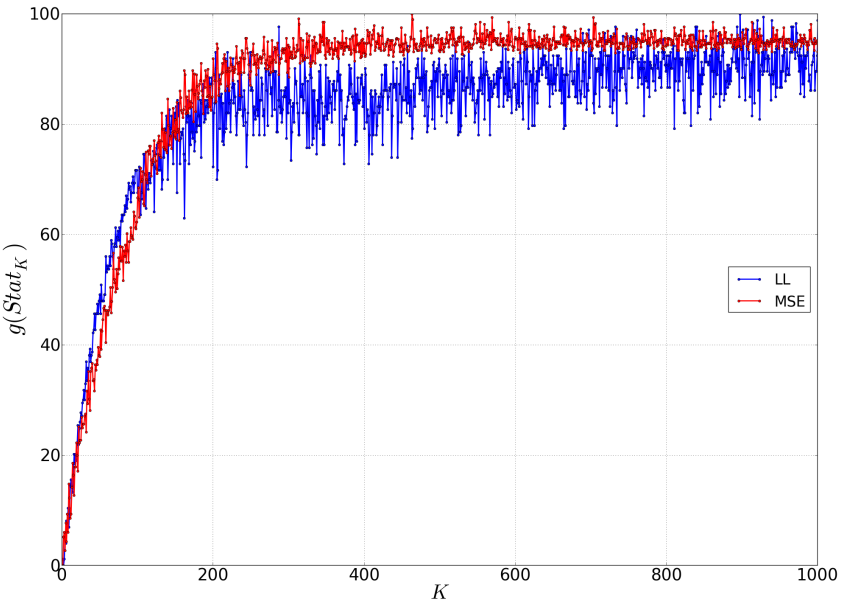

Рис. 7: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE\}$ для модели с параметрами $\{i = 1000\ w = 0.01\}$ при удалении деревьев по одному из середины модели.

3.5.2 Результаты для композиции из трехсот деревьев

Экспериментально исследуем для модели с параметрами $\{i = 300, w = 0.03\}$, как варьирование параметра регуляризации C влияет на обобщающую способность оптимизированной модели (Рис. 8). Будем перебирать параметр C в интервале $[0, 2]$. Для каждого рассмотренного значения C, зная значение $\vec{\beta}_C^*$ вычислим значение LL_C на тестовой выборке и сравним его со значение LL_0, что бы увидеть поведение качества оптимизированной модели.

Красным цветом выделена точка, которая соответствуем параметру C_*, максимизирующему величину LL_C. Для модели с параметрами $\{i = 300, w = 0.03\}$ видно, что график значений $\{\Delta_{rel}(LL_C, LL_0)$ имеет чётко выраженный единственный максимум. Он достигается при значении $C_* = 0.65$, соответственно $LL_{C_*} = -0.0358251$. Получаем, что значение LL_{C_*} всего лишь на 0.5% меньше чем $LL_0 = -0.0356407$.

По виду оптимизируемой функции заметим, что увеличение значения параметра C уменьшает влияние штрафа-*lasso*. Чем больше влияние штрафной добавки, тем сильнее штрафуются большие значения коэффициентов вектора $\vec{\beta}_C^*$. Экспериментальные результаты поведения количества ненулевых коэффициентов вектора $\vec{\beta}_C^*$ отображены на Рис. 9.

Получаем, что для оптимального значения параметра $C_* = 0.65$ в векторе $\vec{\beta}_C^*$ имеется 39 коэффициентов отличных от нуля, что составляет только 13% от общего количества коэффициентов. Это можно интерпретировать так, что для данной модели с параметрами $\{i = 300, w = 0.03\}$ есть возможность удаления из изначальной композиции 261 дерева ценой уменьшения качества LL на 0.5%.

Теперь посмотрим на то, какие из коэффициентов вектора $\vec{\beta}_C^*$ получились не нулевыми (Рис. 10). Бустинг – жадный алгоритм построения композиции алгоритмов. Каждый новый алгоритм в композиции подбирается, так что бы улучшить качество всей предыдущей композиции на обучающей выборке. По результатам из пункта 3.4.3

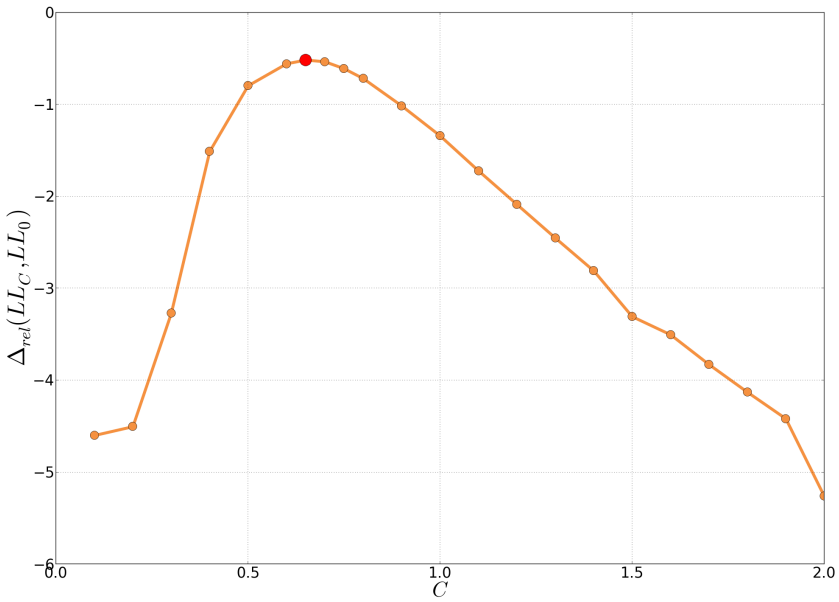

Рис. 8: Поведение качества LL_C оптимизированной модели с параметрами $\{i = 300, w = 0.03\}$ на тестовой выборке.

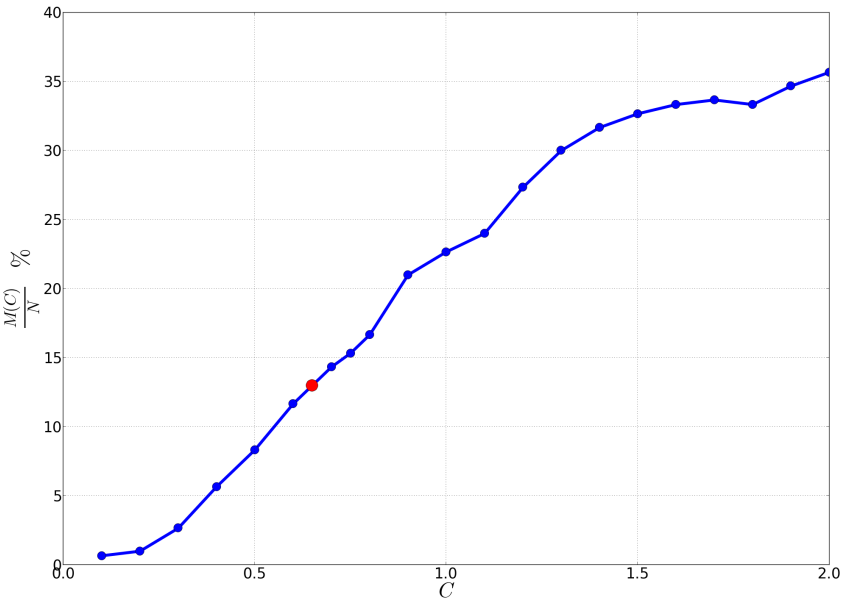

Рис. 9: Поведение количества оставшихся деревьев при оптимизации модели с параметрами $\{i = 300, w = 0.03\}$ методом (21).

можно было бы предположить, что в векторе $\vec{\beta}^*_C$ ненулевыми будут только коэффициенты с небольшими номерами (т.к. деревья с меньшими номерами могут быть более ценными, по работе процедуры градиентного бустинга). Однако, это совсем не так, что показывает Рис. 10. Посмотрим на значения индексов I вектора $\vec{\beta}^*_C$, таких что $I = \{j \mid \beta^*_{j,C} \neq 0\}$. Построим следующую величину $h(\vec{\beta}^*_C, L, k) = \sum_{j=Lk}^{j=L(k+1)-1} \frac{[\beta^*_{C,j} \neq 0]}{L}$. Величина $h(\vec{\beta}^*_C, L, k)$ - есть доля не нулевых коэффициентов вектора $\vec{\beta}^*_C$, индексы

которых лежат в интервале $[Lk, L(k+1)-1]$, где k – номер интервала, L – длина интервала. Возьмём значение $L = 10$.

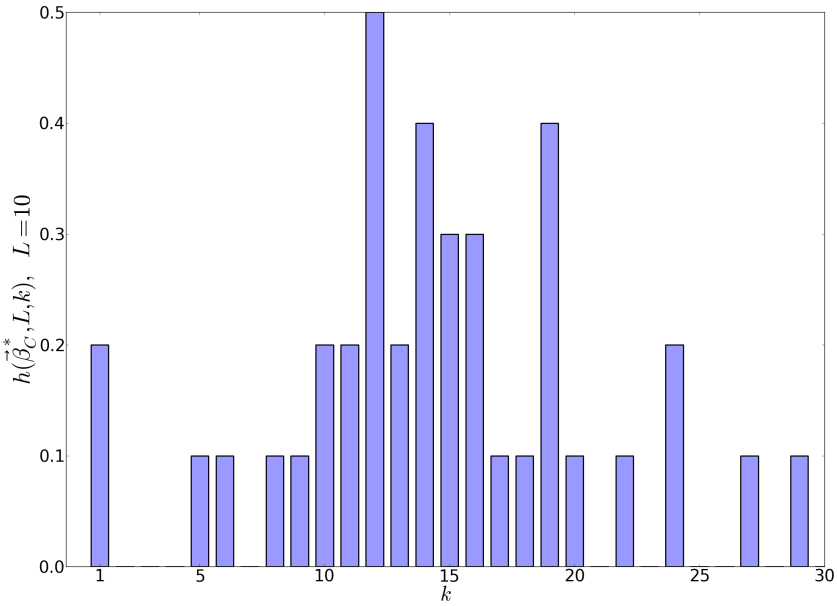

Рис. 10: Распределение ненулевых весов деревьев при оптимизации модели с параметрами $\{i = 300, w = 0.03\}$ методом (21).

Видно, что предложенная процедура логистической регрессии обнуляет совсем не тривиальное множество коэффициентов вектора $\vec{\beta}^*_{C^*}$, соответствующего параметру $C_* = 0.65$. Основная масса ненулевых коэффициентов сосредоточена в середине вектора на интервале $[80, 200]$. Из первых десяти коэффициентов только два отличны от нуля. Более того, из первых сорока коэффициентов два являются не нулевыми. Это эквивалентно тому, что среди первых сорока алгоритмов, построенных градиентным бустингом можно выкинуть 38 алгоритмов, что говорит о неоптимальности жадной стратегии бустинга.

Посмотрим на значения компонент вектора $\vec{\beta}^*_C$ для двух исследуемых моделей, предварительно удалив из $\vec{\beta}^*_C$ элементы с нулевыми значениями. Вектор, полученный после такого преобразования обозначим как $\vec{\beta}^{*,new}_C$ (Рис. 11). Т.е. посмотрим на веса деревьев, оставшихся в композиции, после применения логистической регрессии. Видно, что нет явной зависимости веса дерева от его номера, то есть по сравнению с первоначальной моделью важность деревьев изменилась.

3.5.3 Результаты для композиции из тысячи деревьев

Теперь экспериментально исследуем для модели с параметрами $\{i = 1000, w = 0.01\}$, как варьирование параметра регуляризации C влияет на обобщающую способность оптимизированной модели. Будем перебирать параметр C в интервале $[1, 3]$.

Красным цветом выделена точка, которая соответствуем параметру C^*, максимизирующему величину LL_C. В данном случае, для модели с параметрами $\{i = 1000, w = 0.01\}$ видно, что график значений $\Delta_{rel}(LL_C, LL_0)$ имеет чётко выраженный единственный максимум. Он достигается при значении $C^* = 1.9$, соответственно $LL_{C^*} = -0.35743$. Получаем, что значение LL_{C^*} всего лишь на 0.4% меньше чем

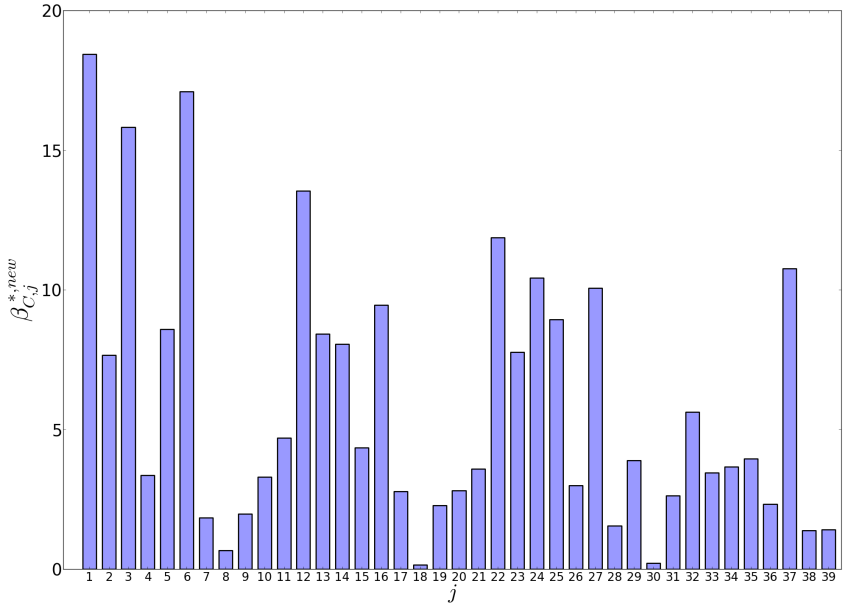

Рис. 11: Значения величин ненулевых весов деревьев при оптимизации модели с параметрами $\{i = 300, w = 0.03\}$

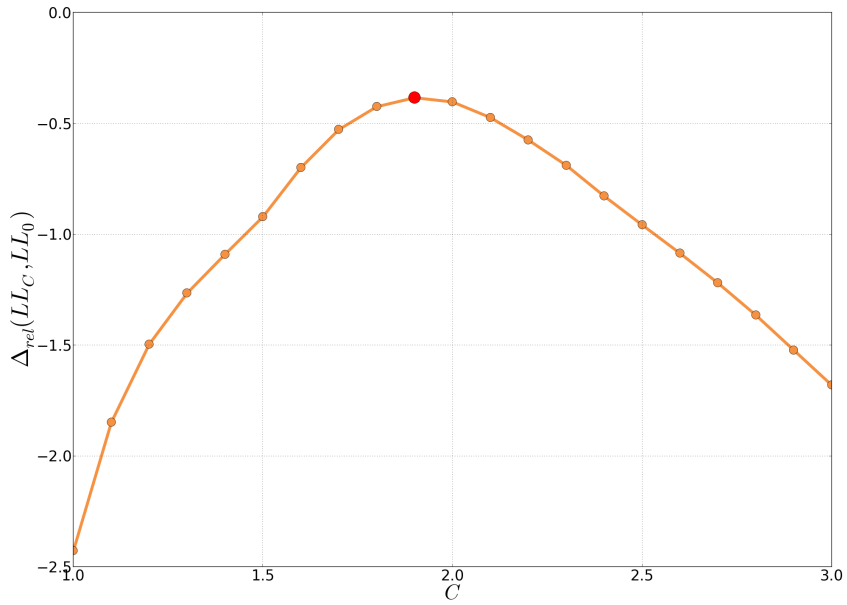

Рис. 12: Поведение качества LL_C оптимизированной модели с параметрами $\{i = 1000, w = 0.01\}$ на тестовой выборке.

$LL_0 = -0.0356065$.

Для данной модели, получаем следующие экспериментальные результаты поведения количества ненулевых коэффициентов вектора $\vec{\beta}^*_C$.

Получаем, что для оптимального значения параметра $C^* = 1.9$ в векторе $\vec{\beta}^*_C$ имеется 57 коэффициентов отличных от нуля, что составляет только 5.7% от общего количества коэффициентов. Это можно интерпретировать так, что для рассматриваемой модели есть возможность удалить из изначальной композиции 943 дерева.

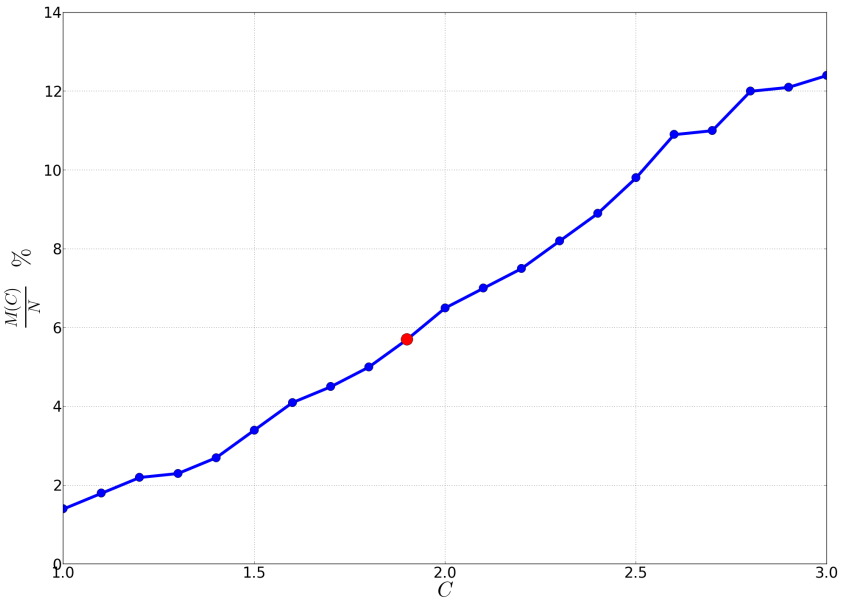

Рис. 13: Поведение количества оставшихся деревьев при оптимизации модели с параметрами $\{i = 1000, w = 0.01\}$ методом (21).

Цена такой трансформации модели - есть уменьшение величины LL на 0.4%.

Теперь посмотрим на то, какие из коэффициентов вектора $\vec{\beta}^*_C$ получились не нулевыми (Рис. 14). Построим величину $h(\vec{\beta}^*_C, L, k)$ при $L = 10$.

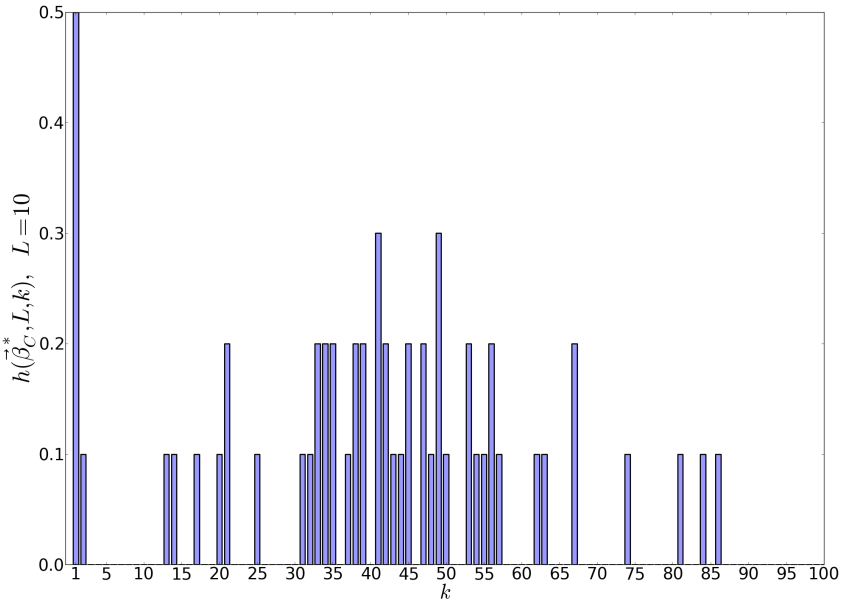

Рис. 14: Распределение ненулевых весов деревьев при оптимизации модели с параметрами $\{i = 1000, w = 0.01\}$ методом (21).

Видно, что предложенная процедура логистической регрессии, по аналогии с предыдущей моделью, обнуляет не тривиальное множество коэффициентов вектора $\vec{\beta}^*_{C^*}$, соответствующего параметру $C^* = 1.9$. Основная масса ненулевых коэффициентов сосредоточена в середине вектора на интервале $[300, 600]$. Однако, теперь резуль-

тат немного другой. Из первых десяти коэффициентов пять отличных от нуля. Из первых ста коэффициентов только шесть являются не нулевыми. Это эквивалентно тому, что среди первых ста алгоритмов, построенных градиентным бустингом можно выкинуть 94 базовых алгоритма, что говорит о неоптимальности жадной стратегии бустинга. Число деревьев в рассматриваемой модели примерно в три раза больше чем в модели с параметрами $\{i = 300, w = 0.03\}$.

Заметим, что и число отобранных деревьев для оптимального значения C^* для данной модели примерно в намного больше числа отобранных деревьев для модели из 300 деревьев (57 против 39, соответственно). Можно сделать вывод, что для фиксированных обучающей и тестовой выборок количество деревьев в модели, оптимизированной методом (21), для оптимального значения C^*, имеет непосредственную связь с размером изначальной композиции.

Посмотрим на значения компонент вектора $\vec{\beta}^*_C$ для исследуемой модели, предварительно удалив из $\vec{\beta}^*_C$ элементы с нулевыми значениями. Т.е. посмотрим на веса деревьев, оставшихся в композиции, после применения логистической регрессии. Результаты отображены на Рис. 15. Видно, что нет явной зависимости веса дерева от его номера, т.е. по сравнению с первоначальной моделью важность деревьев изменилась как для модели из 300 деревьев.

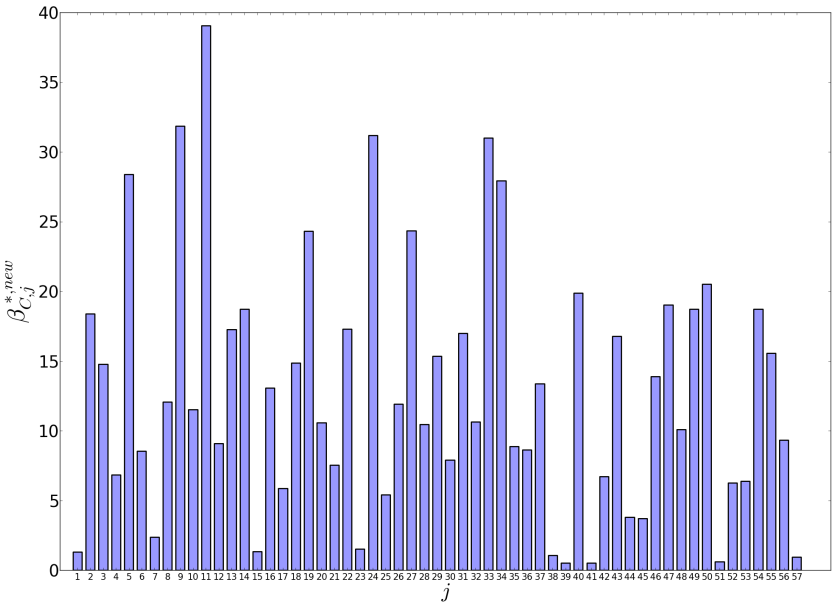

Рис. 15: Значения величин ненулевых весов деревьев при оптимизации модели с параметрами $\{i = 1000, w = 0.01\}$

3.5.4 Численные результаты для метода логистической регрессии с L_1 регуляризацией

В целом, можно видеть, что поведение величин $M_{\%}(C)$ и $\Delta_{rel}(LL_C, LL_0)$ для обеих моделей одинаково. $M_{\%}(C)$ - Монотонно растёт с ростом параметра C. $\Delta_{rel}(LL_C, LL_0)$ - имеет единственный максимум в точке C^*. Можно предположить, что при $C > C^*$ модель начинает переобучаться, а при $C < C^*$ недообучается. Выберем $C = C^*$ и изобразим итоговый результат в виде таблицы 2.

	$M_{\%}(C_*)$	$\Delta_{rel}(LL_{C_*}, LL_0)$	C_*
исходная модель	100	0	—
L_1 - оптимизированная модель	13	-0.5173	0.65

Таблица 2: $M_{\%}(C_*)$ и $\Delta_{rel}(LL_{C_*}, LL_0)$ для модели с параметрами $\{i = 300, w = 0.03\}$.

	train			test		
$Stat$	LL	MSE	$LinCorr$	LL	MSE	$LinCorr$
$Stat_0$	-0.03173	0.005738	0.2044	-0.03564	0.006057	0.0642
$Stat_{C^*}$	-0.03163	0.005699	0.2259	-0.03582	0.006062	0.0610
$\Delta_{rel}(Stat_{C^*}, Stat_0)$	0.31136	0.68137	10.567	-0.51739	-0.06834	-5.122

Таблица 3: $LL, MSE, LinCorr$ для модели с параметрами $\{i = 300, w = 0.03\}$

	$M_{\%}(C)$	$\Delta_{rel}(LL_C, LL_0)$	C_*
исходная модель	100	0	—
L_1 - оптимизированная модель	5.7	-0.38341	1.9

Таблица 4: $M_{\%}(C)$ и $\Delta_{rel}(LL_C, LL_0)$ для модели с параметрами $\{i = 1000, w = 0.01\}$.

	train			test		
$Stat$	LL	MSE	$LinCorr$	LL	MSE	$LinCorr$
$Stat_0$	-0.031414	0.00573	0.2124	-0.03561	0.0060564	0.06565
$Stat_{C^*}$	-0.031314	0.00568	0.2437	-0.03574	0.0060557	0.06701
$\Delta_{rel}(Stat_{C^*}, Stat_0)$	0.3167	0.7939	14.7336	-0.3833	0.0114	2.0619

Таблица 5: Изменение $LL, MSE, LinCorr$ для модели с параметрами $\{i = 1000, w = 0.01\}$.

В итоге получаем, что в новых моделях остаётся всего лишь 5.7% (для $\{i = 300, w = 0.03\}$) и 13% (для $\{i = 1000, w = 0.01\}$) от первоначального количества деревьев. При этом на тесте качество LL уменьшается не более чем на 0.5%, в то время как для модели из 1000 деревьев MSE и $LinCorr$ улучшаются.

3.5.5 Достраивание композиции деревьев, полученной после процедуры логистической регрессии с L_1 регуляризацией

После сокращения композиции возниакет вопрос: можно ли увеличить качество модели путём небольшого увеличения размера модели. Возможность такого улучшения могла бы позволить не уменьшить качество оптимизированной модели.

Градиентный бустинг на каждой новой итерации, при добавлении нового элемента к уже имеющейся композиции, строит этот новый элемент, основываясь на качестве уже построенной композиции. Возьмём модель, полученную просле применения метода (21). Оставим в модели слагаемые, соответствущие не нулевым компонентам в векторе $\vec{\beta}^*_C$. Сколько не нулевых коэффициентов, столько и деревьев получаем в модели. Полученную таким образом модель будем увеличать, применяя градиентный бустинг. Применять эту процедуру будем к обоим рассматриваемым моделям. К моделям с параметрами $\{i = 300\ w = 0.03\}$ и $\{i = 1000\ w = 0.01\}$ достроим 300 и 1000 новых деревьев соответственно.

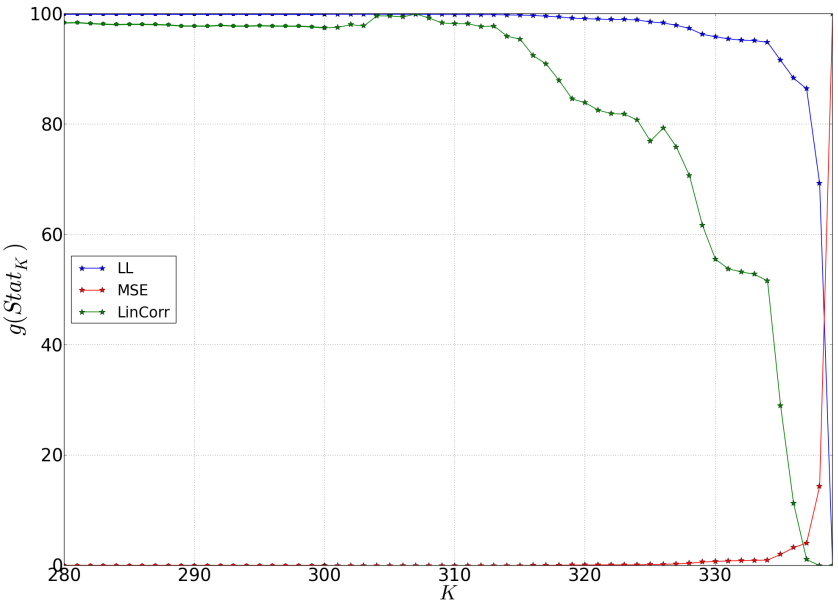

Рис. 16: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE, LinCorr\}$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при достроении новых деревьев градиентным бустингом. $k \in [280, 340]$. Тонкой линией показана часть, соответствующая модели после (21) оптимизации.

Фактически, на Рис. 17 изображено поведение статистик качества для всех новых деревьев, достроенных к уже оптимизированной композиции. Видно, то переобучение модели наступает очень быстро. Хуже всего ведёт себя MSE, переобучение по этой статистике наступает сразу же с добавлением новых деревьев. Немного лучшее поведение показывается для статистик $LL, LinCorr$, однако, улучшение качества модели по этим статистикам совсем незначительное. Введём обозначения:

- величину $Stat_{C^*}$ из раздела (3.5.2) обозначим как $Stat_{L_1}$.
- k^* - индекс дерева в композиции (начиная с начала) для которого соответствующая величина $Stat$ принимает лучшее значение.

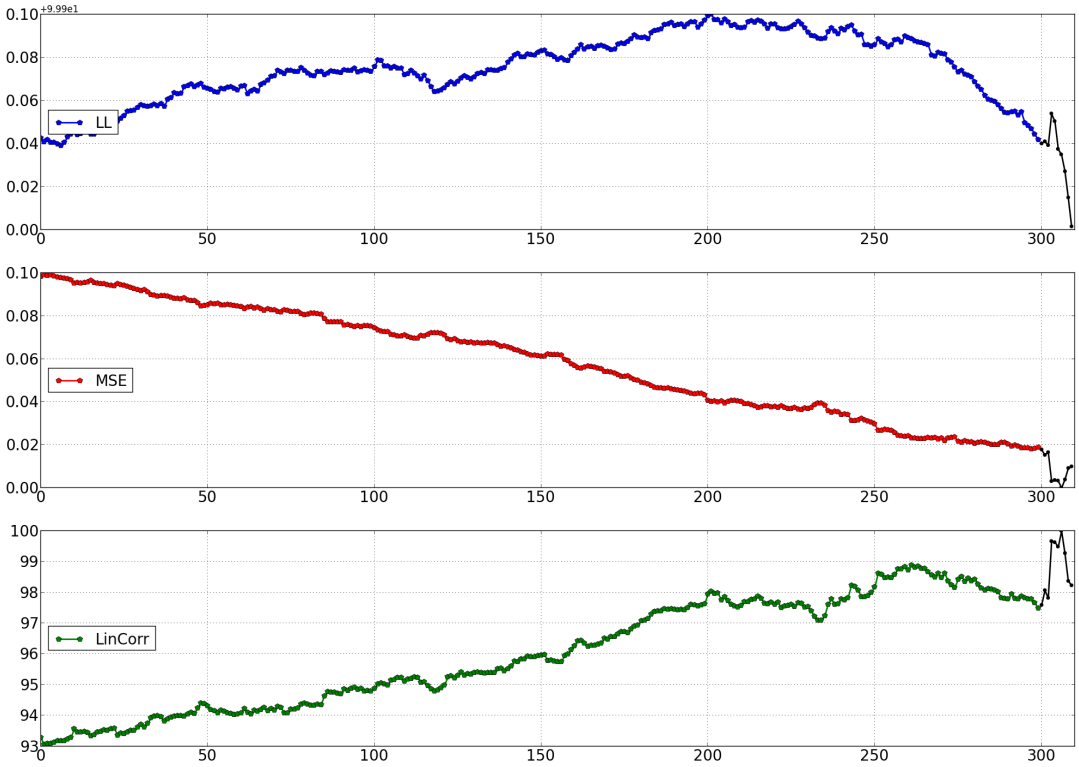

Рис. 17: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE, LinCorr\}$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при достроении новых деревьев градиентным бустингом. $k \in [1, 310]$. Черным цветом показана часть, соответствующая модели после (21) оптимизации.

Результаты приведены в таблице 6:

$Stats$	LL	MSE	$LinCorr$
k^*	138	33	33
$g(Stat_{L_1})$	99.9401	0.0177	97.5904
$g(Stat_{k^*})$	100	0	100
$\Delta_{rel}(Stat_{k^*}, Stat_{L_1})$	0.121981516	-0.046861052	1.248976414

Таблица 6: Результаты достроения новых деревьев для модели $\{i = 300, w = 0.03\}$.

На Рис. 6 видно, что по всем трем статистикикам происходит переобучение. Более того видно, что в L_1 - оптимизированной модели все три статистики доситгают лучшего значенияне на полном количестве деревьев. В результате достроения деревьев удаётся улучшить только статистику - LL. Улучшенье мало и составляет всего лишь 0.12 %. Основным фактором отсутствия улучшения характеристик модели рпи достроении новых деревьев является переобучение. В данной случае переобучаемость методов логистической регрессии и градиентного бустинга складываются, вследствие чего переобучаемость сильно возрастает. В результате, можно сделать вывод, что достроение композиции градиентным бустингом является не эффективным методом улучшения качества модели, оптимизированной методом (21).

Аналогичные результаты мы получаем и для модели с параметрами $\{i = 1000\ w = 0.01\}$.

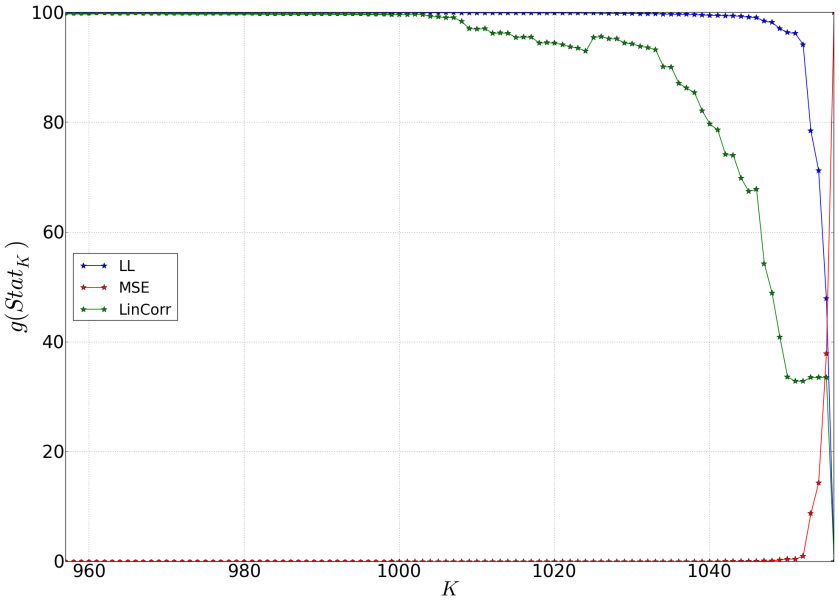

Рис. 18: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE, LinCorr\}$ для модели с параметрами $\{i = 1000\ w = 0.01\}$ при достроении новых деревьев градиентным бустингом. $k \in [957, 1057]$. Тонкой линией показана часть, соответствующая модели после (21) оптимизации.

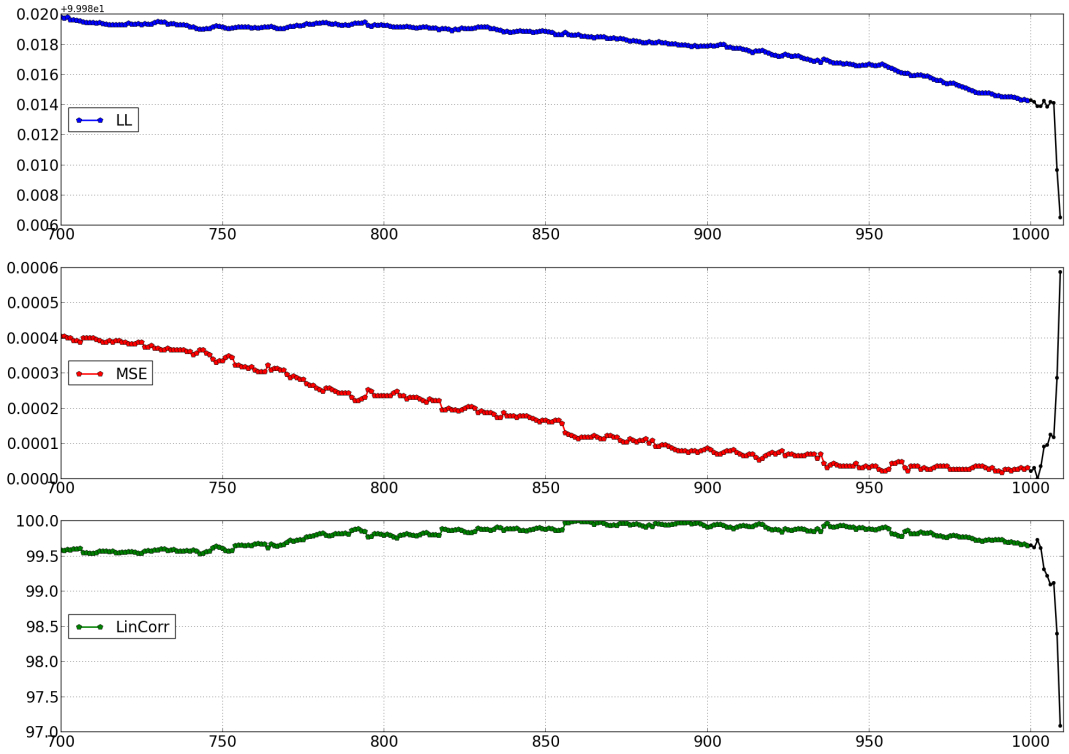

Рис. 19: Поведение $g(Stat_k)$ для $Stat \in \{LL, MSE, LinCorr\}$ для модели с параметрами $\{i = 1000\ w = 0.01\}$ при достроении новых деревьев градиентным бустингом. $k \in [700, 1010]$. Черным цветом показана часть, соответствующая модели после (21) оптимизации.

3.6 Результаты упрощения композиции с использованием L_2 регуляризации

Допустим, что при построении композиции из N деревьев каждая промежуточная композиция из $K < N$ является неоптимальной из-за фиксированных весов всех деревьев и из-за жадности самого алгоритма. Тогда перевзвешиванием этих k деревьев при помощи L_2-регуляризации (22) можно получить новую модель из k деревьев, которая даст некоторых выигрыш в LL на тестовой выборке. Введём обозначения:

- LL^K_{test} - качество на тестовой выборке модели, составленной из первых K деревьев исходной модели.
- $LL^{K,*}_{test}$ - качество на тестовой выборке L_2 - оптимизированной модели, составленной из первых K деревьев исходной модели.
- $\Delta_{LL}(K) = LL^{K,*}_{test} - LL^K_{test}$.

Исследуем зависимость $\Delta_{LL}(K)$. Результаты изображены на Рис. 20.

Получаем, что $\Delta_{LL}(K) > 0\ \forall K = \overline{1, N} \implies LL^{K,*}_{test} > LL^K_{test}$. Таким образом, видно, что перенастройка весов деревьев при помощи логистической регрессии с L_2-регуляризацией позволяет улучшить качество модели. Зависимость $\Delta_{LL}(K)$ является строго убывающей. Значит, чем больше деревьев в композиции, построенной

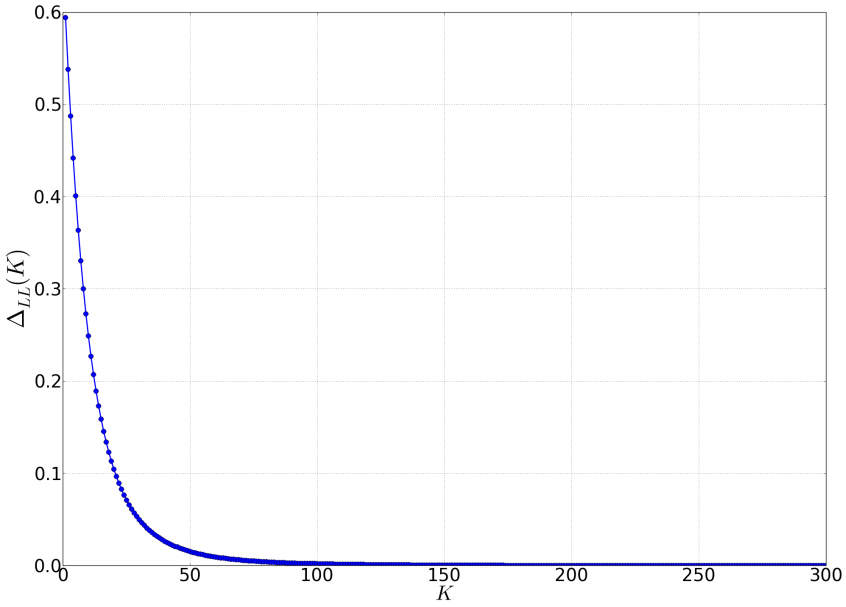

Рис. 20: Поведение $\Delta_{LL}(K)$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при оптимизации методом (22).

градиентным бустингом, тем менее значительным является улучшение качества LL модели. В итоге, для полного числа деревьев имеем $\Delta_{LL}(K = 300) = 0.00003163$, а относительное улучшение LL составляет 0.0887%. То есть, для модели из 300 деревьев происходит очень малое улучшение качества. Тем не менее, при различных ограничениях на количество деревьев в модели можно получить существенный выигрыш. Рассмотрим величину относительного изменения качества модели:

$$\Delta^{\%}_{LL}(K) = -100\frac{LL^{K,*}_{test} - LL^{K}_{test}}{LL^{K}_{test}} = -100\frac{\Delta_{LL}(K)}{LL^{K}_{test}} \tag{30}$$

Значение $\Delta^{\%}_{LL}(K)$ показывает, на сколько процентов качество изначальной модели, состоящей из первых K деревьев, хуже, чем качество этой же модели после оптимизации весов с L_2-регуляризацией (применения метода (22)). Поведение величины $\Delta^{\%}_{LL}(K)$ для модели из трёхсот деревьев показано на Рис. 21.

Подобные графики можно строить для каждой оптимизируемой модели и наглядно видеть, какой выигрыш качества можно получить для каждого из ограничений на количество деревьев в композиции. Приведём величины $\Delta^{\%}_{LL}(K)$ для значений $K \in \{1, 50, 100, 150, 200, 250, 300\}$ в виде таблицы 7.

K	1	50	100	150	200	250	300
$\Delta^{\%}_{LL}(K)$	94.0945	29.3550	5.6408	1.6621	0.5930	0.2338	0.0887

Таблица 7: $\Delta^{\%}_{LL}(K)$ для модели $\{i = 300, w = 0.03\}$.

Так же, важным для нас является вопрос, о том, сколько деревьев можно удалить из модели без потери качества LL при применении данного метода оптимизации

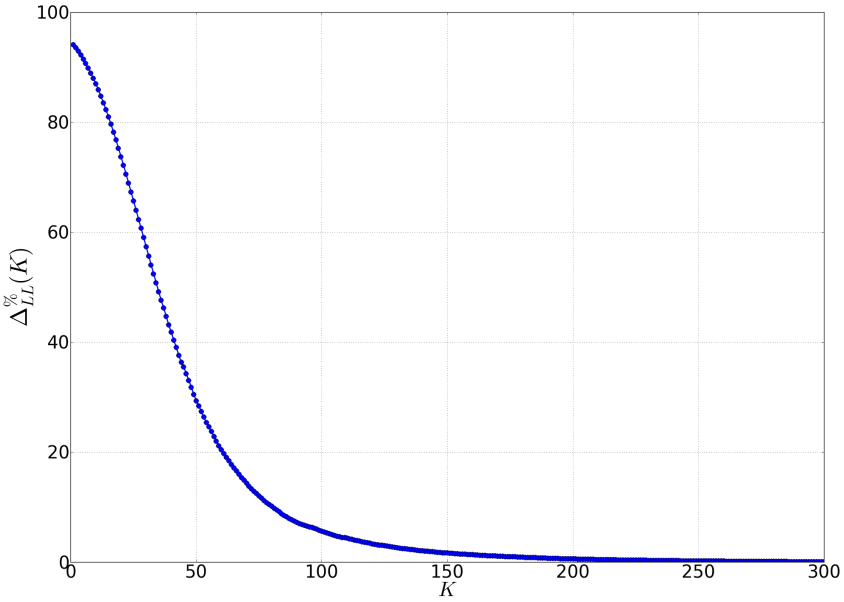

Рис. 21: Поведение $\Delta^{\%}_{LL}(K)$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при оптимизации методом (22).

весов отдельных деревьев. Для каждой модели можно рассмотреть следующую величину:

$$LL^{max}_{test} = \max_{K}\{LL^{K}_{test}\}$$
$$\Delta^{max\ \%}_{LL}(K) = -100\frac{LL^{K,*}_{test} - LL^{max}_{test}}{LL^{max}_{test}}$$

В нашем случае имеем, что $LL^{max}_{test} = \max_{K}\{LL^{K}_{test}\} = LL_{test}$. Следовательно, получаем:

$$\Delta^{max\ \%}_{LL}(K) = -100\frac{LL^{K,*}_{test} - LL_{test}}{LL_{test}}$$

Получаем, что значение $\Delta^{max\ \%}_{LL}(K)$ показывает, на сколько процентов качество изначальной модели (с полным количеством деревьев), хуже, чем качество модели только из K деревьев после оптимизации весов с L_2-регуляризацией. Поведение величины $\Delta^{max\ \%}_{LL}(K)$ для модели из трёхсот деревьев показано на Рис. 22.

Соответственно, обозначим:

$$K^* = \arg\max\{K \mid LL^{K,*}_{test} \geq LL_{test}\} = \arg\max\{K \mid \Delta^{max\ \%}_{LL}(K) \geq 0\} \tag{31}$$

Для данной модели получаем, что $K^* = 224 \Longrightarrow \Delta^{max\%}_{LL}(K^*) = 0.002468\%$. Таким образом, при использовании данного метода для данной модели из 300 деревьев из композиции можно убрать последние 76 деревьев, оставив 224 и не потеряв в качестве. Так же, можно оставить в композиции только первые 108 деревьев, потеряв в качестве $\Delta^{max\%}_{LL}(K = 108) = -1.0131\%$.

3.7 Сравнение L_1 и L_2 регуляризаций

Рассмотрим модель с параметрами $\{i = 300\ w = 0.03\}$. В пункте (3.5.4) в таблице 2 видно, что при использовании L_1 регуляризации максимум LL достигается при

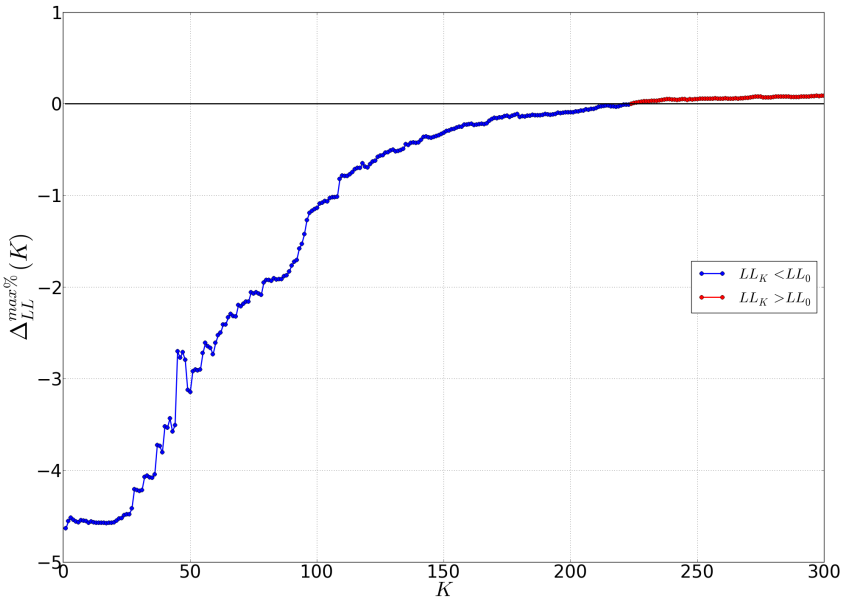

Рис. 22: Поведение $\Delta_{LL}^{max\%}(K)$ для модели с параметрами $\{i = 300\ w = 0.03\}$ при оптимизации методом (22).

$C = 0.65$. При этом в оптимизированной модели выделяется некоторое нетривиальное подмножество деревьев. Обозначим это множество, как J, $|J| = 39$. Теперь оставим в исходной модели только это подмножество деревьев и проведём оптимизацию модели, использую уже L_2 регуляризацию. То есть в новой задаче имеем, что:

$$G(\vec{\beta}, \vec{x}_j) = \beta_0 + \sum_{k \in J} \beta_k \vec{\xi}_k(\vec{x}_j) \tag{32}$$

Введём обозначение :

- $LL_{test}(\vec{\beta}_{\lambda_2^*}) = LL_{L_2}$, где λ_2^* — есть решение задачи (23).

Результаты оптимизаций приведены в таблице 8. Отметим, что множества индексов деревьев в оптимизированных моделях совпадают.

	train			test		
$Stat$	LL	MSE	$LinCorr$	LL	MSE	$LinCorr$
$Stat_0$	-0.03173	0.0057384	0.20436	-0.03564	0.0060576	0.06426
$Stat_{L_1}$	-0.03163	0.0056993	0.22595	-0.03577	0.0060611	0.06015
$Stat_{L_2}$	-0.03132	0.0056942	0.24313	-0.03582	0.0060617	0.06097
$\Delta_{rel}(Stat_{L_1}, Stat_0)$	0.311	0.681	10.567	-0.517	-0.068	-5.122
$\Delta_{rel}(Stat_{L_2}, Stat_0)$	1.305	0.770	18.973	-0.364	-0.057	-6.399
$\Delta_{rel}(Stat_{L_2}, Stat_{L_1})$	0.997	0.089	7.602	0.152	0.011	-1.346

Таблица 8: Сравнение методов (21) и (22) для модели $\{i = 300, w = 0.03\}$.

После использования L_2-регуляризации падение в качестве LL составляет уже 0.36% вместо 0.56% при L_1-регуляризации на тестовой выборке, так же MSE уменьшается на 0.011%. Таким образом, заранее зная множество J (например, после процедуры логистической регрессии с L_1-регуляризацией) можно увеличить обобщающую способность модели посредством применения L_2-регуляризации. Прирост качества на тесте происходит для всех статистик кроме линейной корреляции, она падает на 1.34%.

Теперь абстрагируемся от индексов конкретных деревьев присутствующих в композиции. Будем считать, что важен только размер всей композиции, как показатель скорости вычисления модели. Теперь, руководствуясь только таким быстродействием формул, сравним приведённый метод с L_2-регуляризацией с логистической регрессией с L_1-штрафом, приведённой в пункте (2.2).

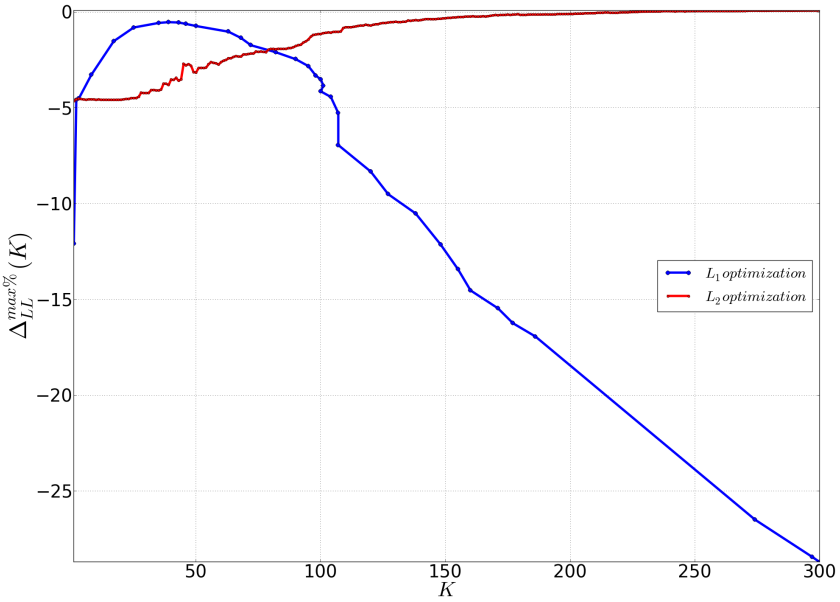

Рис. 23: Поведение $\Delta_{LL}^{max\%}(K)$ на тестовой выборке для модели с параметрами $\{i = 300\ w = 0.03\}$ для методов (21) и (22).

Алгоритм 5 Алгоритм выбора метода уменьшения уменьшения композиции для модели с параметрами $\{i = 300\ w = 0.03\}$.

Вход: Ограничение x на количество деревьев в композиции.

Выход: Точка $\big(K,\ \Delta_{LL,method}^{max\%}(K)\big)$, $method \in \{L_1, L_2\}$ — используемый метод упрощения композиции.

1: **если** $x \in (0, 129)$ **то**
2: используем результаты метода L_1 — регуляризации (21) для точки $\big(K = 39,\ \Delta_{LL,L_1}^{max\%}(K) = -0.52\%\big)$
3: **если** $x \in [129, 300]$ **то**
4: используем результаты метода L_2 — регуляризации (22) для точки $\big(K = x,\ \Delta_{LL,L_2}^{max\%}(x)\big)$

На Рис. 23 изображены $\Delta^{max\%}_{LL,L_1}(K)$ и $\Delta^{max\%}_{LL,L_2}(K)$ — значения величины $\Delta^{max\%}_{LL}(K)$ для моделей полученных при L_1 и L_2 регуляризациях соответственно. Два изображенных графика пересекаются в точке приближенно $K = 80$, $\Delta^{max\%}_{LL}(K) = -2\%$. Как уже отмечалось $39 = \arg\max_K \Delta^{max\%}_{LL,L_1}(K)$, $\Delta^{max\%}_{LL,L_1}(39) = -0.52\%$. Решением уравнения $\Delta^{max\%}_{LL,L_1}(39) = \Delta^{max\%}_{LL,L_2}(K)$ является $K = 129$. Глядя на график можно ответить на вопрос: когда нужно использовать L_1, а когда L_2 регуляризацию (то есть, какой из методов (21) и (22) выбрать). Получаем, что если у нас есть сильное ограничение на количество деревьев в композиции (например, $N < 50$), то нужно воспользоваться логистической регрессией с L_1 регуляризацией. Если есть ограничение $N < 150$, то нужно воспользоваться результатами L_2-регуляризации. Для данной модели для ограничения вида $\{K \leq x\}$ на количество деревьев, имеем следующий алгоритм 5 выбора метода упрощения модели:

4 Выводы

Используемые в данной работе модели, полученные при градиентном бустинге оказались избыточными. Жадность бустинга приводит к тому, что построенные композиции состоят из слишком большого числа деревьев. Многие из этих деревьев можно удалить. Для уменьшения сложности композиций была применена логистическая регрессия с L_1 и L_2 регуляризациями.

При использовании L_1 регуляризации возможно сокращение исходной композиции деревьев в среднем на 90% с 0.5% потери качества LL. После оптимизации в модели остаётся не тривиальное множество деревьев. Из исходной модели удаляются не только деревья, которые создавалиь на поздних итерациях бустинга, но также некоторые деревья из начала и середины композиции. Веса в новой сокращенной модели не имеют монотонной зависимости от номера дерева. После оптимизации значимость деревьев в композиции перераспределилась.

При использовании L_2 регуляризации можно в среднем уменьшить размер исходной модели на треть без потери качества LL. Веса в исходной композиции не оптимальны. Переподбор весов деревьев улучшает качество композиции. Чем больше деревьев в исходной композиции тем меньше эффект улучшения. Более того, композиции, составленные из первых k деревьев исходной композиции так же не оптимальны. Путём применения логистической регрессии с L_2 регуляризацией можно значительно улучшить LL таких композиций. С ростом к величина прироста LL монотонно уменьшается.

Показано, что после оптимизаии к модели не имеет смысла достраивать новые деревья методом градиентного бустинга. Добавление новых слагаемых к оптимизированной композиции приводит к быстрому переобучению.

В эксперименте показано, что модель оптимизированная с использованием L_2 регуляризации имеет большую обобщающую способность чем такая же модель оптимизированная с использованием L_1 регуляризации при одинковых наборах деревьев в итоговой модели. Это позволяет использовать рузличные регуляризации последовательно для улучшения результата. Сначала L_1 регуляризация для отбора деревьев, затем L_2 регуляризация для увеличения обобщающей способности модели, полученной на первом шаге.

Список литературы

[1] Predictive Learning via Rule Ensembles. J. H. Friedman, B. E. Popescu, October 5, 2005.

[2] J. Friedman. Greedy function approximation: A gradient boosting machine. In Techical Report. Dept. of Statistics, Stanford University, 1999.

[3] Jerome H. Friedman, Stochastic gradient boosting, Computational Statistics & Data Analysis, v.38 n.4, p.367-378, 28 February 2002 [doi>10.1016/S0167-9473(01)00065-2].

[4] A. Gulin. Matrixnet. Technical report, http://www.ashmanov.com/arc/searchconf2010/08gulin-searchconf2010.ppt, 2010. (in russian).

[5] A. Broder and V. Josifovski. Introduction to Computational Advertising. http://www.stanford.edu/class/msande239/

[6] Schapire R. The boosting approach to machine learning: An overview // MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA. 2001. http://citeseer.ist.psu.edu/schapire02boosting.html.

[7] Ronny Kohavi and J. Ross Quinlan. 2002. Data mining tasks and methods: Classification: decision-tree discovery. In Handbook of data mining and knowledge discovery, Willi Klösgen and Jan M. Zytkow (Eds.). Oxford University Press, Inc., New York, NY, USA 267-276.