Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Иванов Олег Юрьевич

Использование методов машинного обучения для повышения эффективности систем управления базами данных

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
к.ф.-м.н.
Д.П. Ветров

Москва, 2016

Содержание

Аннотация

В данной работе рассматривается устройство оптимизатора запросов реляционных систем управления базами данных и предлагается метод устранения выявленных недостатков. Показывается, что причиной наибольших ошибок оптимизатора запросов является именно оценка количества кортежей, удовлетворяющих некоторому заданному набору условий. Существующие подходы к этой проблеме обладают рядом недостатков, уменьшающих их практическую значимость. В работе предлагается метод сведения оценки количества кортежей к задаче машинного обучения. Проводится теоретическое и экспериментальное исследование этого метода. Показывается, что предложенный метод существенно увеличивает точность предсказания количества кортежей, что приводит к увеличению производительности систем управления базами данных.

1 Введение

На сегодняшний день большая часть программных систем занимается обработкой информации. Веб–сервисы, социальные сети, банковские системы, программы для ведения учета и бухгалтерии на предприятиях — все они вынуждены хранить и обрабатывать большие объемы информации. При этом к хранению и обработке информации во всех этих программных системах предъявляются схожие требования: высокая надежность, скорость доступа и изменения информации, возможность хранить информацию на разных машинах, создавать резервные копии и так далее. Для того, чтобы получившийся программный продукт удовлетворял этим требованиям, в большинстве современных программных продуктов используются универсальные системы управления базами данных (СУБД).

Система управления базами данных организует хранение данных произвольного характера во внешней памяти, проверку целостности этих данных и восстановление их после сбоев, осуществляет доступ и обработку этих данных в соответствии с получаемыми запросами.

В данной работе рассматриваются реляционные СУБД. Важной особенностью запросов в таких СУБД является их декларативность. Это означает, что в запросе формулируется, какие операции нужно совершить над данными. Физический же способ совершения этих операций должна выбрать СУБД.

Например, для запроса SQL `SELECT * FROM users WHERE age < 25` СУБД может просмотреть все записи таблицы `users` и отобрать нужные, а может использовать индекс — построенное заранее дерево поиска — в котором записи таблицы `users` отсортированы по возрасту.

Способ выполнения запроса SQL называется планом выполнения этого запроса. У одного запроса может быть множество различных планов выполнения, и время их выполнения может отличаться на несколько порядков. Поэтому выбор наиболее эффективного плана выполнения запроса очень важен для производительности СУБД. Часть СУБД, осуществляющая выбор плана выполнения запроса, называется оптимизатором запросов СУБД.

В данной работе рассматривается стоимостной оптимизатор запросов, используемый в большинстве современных реляционных СУБД. В работе исследуется устройство этого типа оптимизаторов запросов и его недостатки. Было показано, что наибольшие ошибки оптимизатор запросов допускает из-за неправильной оценки количества отбираемых условиями кортежей. Это в свою очередь происходит из-за предположения о независимости условий в запросе, то есть доля кортежей, удовлетворяющих всем условиям запроса, считается равной произведению долей кортежей, удовлетворяющих каждому условию в отдельности. Устройство стоимостных оптимизаторов более подробно описано в разделе 1.2.

Например, для запроса `SELECT * FROM users WHERE age < 10 AND married = TRUE` доля пользователей, которые младше 10 лет и состоят в браке, будет считаться равной произведению доли пользователей младше 10 лет и доли пользователей в

браке. Естественно, это ошибка, потому что среди пользователей младше 10 лет никто не состоит в браке.

Существуют разные подходы к решению этой проблемы. Наиболее популярным в сообществе разработчиков СУБД подходом являются многомерные гистограммы, которые позволяют напрямую считать долю пользователей, удовлетворяющих набору условий. Очевидной проблемой такого подхода является невозможность построения гистограммы по достаточно большому количеству столбцов и сложность определения тех столбцов, по которым нужно строить гистограммы.

Существует несколько работ в области машинного обучения, посвященных данной проблеме, однако каждая из этих работ имеет свои недостатки, которые уменьшают практическую ценность полученных в ней результатов. Все работы, посвященные данной проблеме, и их недостатки разобраны в разделе 1.3.

В разделе 2.1 предложено сведение задачи оценки количества кортежей, отбираемых с помощью данных условий, к задаче машинного обучения. В разделе 2.2 описываются полученные теоретические гарантии для предложенного сведения. В разделе 2.5 описаны эвристики, позволяющие улучшить предложенные алгоритм. В разделе 3 проведено экспериментальное сравнение разных алгоритмов машинного обучения для поставленной задачи, в том числе сравнение с методами из других работ.

Полученные результаты показывают применимость предложенного подхода для улучшения оптимизатора запросов реляционных СУБД, и, следовательно, для увеличения производительности реляционных СУБД в целом.

1.1 Определения и обозначения

Системой управления базами данных (СУБД) называется совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Реляционной системой управления базами данных (РСУБД) называется СУБД, управляющая реляционными базами данных.

Реляционной базой данных называется база данных, основанная на реляционной модели данных.

Реляционнной моделью данных называется модель, где данные представленны в виде набора отношений.

Отношением в реляционной модели данных называется пара из схемы отношения и тела отношения.

Схемой H отношения называется конечное множество упорядоченных пар вида $(A_i,\ T_i)$, где A_i — имя атрибута, а T_i — имя типа (домена), $i = 1, \ldots, n$. По определению требуется, чтобы все имена атрибутов в заголовке отношения были различными (уникальными).

Доменом в реляционной модели данных называется тип данных, то есть допустимое множество значений переменной этого типа.

Телом отношения называется множество кортежей t. Кортеж t, соответствующий заголовку H, — множество упорядоченных троек вида $< A_i, T_i, v_i >$, по одной такой тройке для каждого атрибута в H, где v_i — допустимое значение типа (домена) T_i. Так как имена атрибутов уникальны, то указание домена в кортеже обычно излишне. Поэтому кортеж t, соответствующий заголовку H, нередко определяют как множество пар (A_i, v_i).

В качестве модели реляционной СУБД в данной работе будет рассматриваться СУБД PostgreSQL с открытым исходным кодом.

SQL–запросом называется предложение на языке SQL, описывающее, какие операции требуется совершить над базой данных.

Планом выполнения SQL–запроса называется последовательность операций, необходимых для получения результата SQL–запроса. Абстрактным представлением такого плана выполнения является корневое дерево физических операций, пример которого показан на рис. 1. Дуги в дереве операций представляют потоки данных между операциями. Далее будем использовать термины «дерево физических операций» и «план выполнения» (или просто план) в одном и том же смысле.

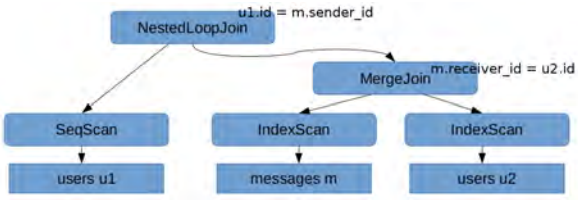


Рис. 1: Пример плана выполнения SQL–запроса

Листовыми вершинами плана будем называть листья соответствующего плану дерева операций, а внутренними вершинами плана — все остальные его вершины.

Частичным планом или подпланом выполнения SQL–запроса будем называть дерево физических операций, являющееся поддеревом некоторого плана выполнения этого SQL–запроса.

Оптимизатором запросов СУБД будем называть ту её часть, которая отвечает за построение по SQL–запросу плана его выполнения.

Большая часть современных реляционных СУБД использует стоимостной оптимизатор запросов.

Стоимостным называется оптимизатор запросов, который каждому плану или частичному плану выполнения запроса ставит в соответствие некоторую стоимость (вещественное число), а затем производит точную или приближенную оптимизацию этой стоимости, то есть в пространстве всех возможных планов выполнения данного SQL–запроса ищет план с минимальной стоимостью.

Стоимостью плана называется оценка количества ресурсов, требуемых для его выполнения. В случае, когда на сервере не выполняются другие задачи и запросы, оцениваемое время выполнения запроса прямо пропорционально количеству потраченных на него ресурсов. Поэтому можно считать, что стоимость плана — это его время выполнения в некоторых условных единицах.

Заметим, что стоимость плана может оцениваться только в контексте конкретной базы данных — один и тот же план может требовать разного количества ресурсов при разных размерах, содержимом или соотношении размеров таблиц в базе данных.

Стоимостью выполнения плана (англ. total cost) называется оценка количества ресурсов, требуемых для того, чтобы план выдал все кортежи результата его выполнения.

Стоимостью запуска плана (англ. start-up cost) называется оценка количества ресурсов, требуемых для того, чтобы план выдал первый кортеж результата его выполнения.

Стоимостью выполнения вершины плана будем называть стоимость выполнения поддерева с корнем в этой вершине.

Аналогично определим стоимость стоимость запуска вершины плана.

Разделение стоимостей на стоимость запуска и стоимость выполнения не выполняется для всех реляционных СУБД. Оно используется в СУБД PostgreSQL для корректной обработки подзапросов, например, для запроса внутри предиката EXISTS нужно оптимизировать стоимость запуска, а для запроса с заданным параметром LIMIT оптимизируется линейная комбинация стоимости запуска и стоимости выполнения. Однако эта модификация стандартной схемы стоимостного планировщика не важна в данной работе, поэтому в дальнейшем мы будем рассматривать стандартную модель, в которой нет понятия «стоимость запуска плана», а понятия «стоимость выполнения плана» и «стоимость плана» эквивалентны.

Количество кортежей, выдаваемых вершиной плана, называется мощностью (англ. cardinality) этой вершины.

Каждая вершина плана может содержать некоторые условия, которые проверяются для всех кортежей, обрабатыаемых этой вершиной. Вершина плана выдает только кортежи, удовлетворяющие всем условиям в ней.

Выборочностью (англ. selectivity) вершины называется отношение её мощности к количеству обработанных в ней кортежей. Это вещественное число от 0 до 1, поскольку некоторые обработанные в вершине кортежи не удовлетворяют всем условиям запроса и потому не выдаются в результате её работы. Мы также будем называть выборочность вершины совместной выборочностью всех условий вершины, по аналогии с теорией вероятности.

Выборочностью условия в некоторой вершине называется отношение количества обработанных кортежей, удовлетворяющих этому условию, к общему количеству кортежей, обработанных в этой вершине. По аналогии с теорией вероятности, мы также будем называть это отношение маргинальной выборочностью условия.

Целью данной работы является

- выявление недостатков стоимостных оптимизаторов запросов в реляционных СУБД и определение среди них наиболее критичных для выбора оптимального плана,

- предложение, экспериментальное и теоретическое изучение методов машинного обучения, позволяющих преодолевать эти недостатки
- определение направлений для дальнейшего исследования

1.2 Устройство оптимизатора запросов реляционных СУБД

В этой работе в качестве модели реляционной СУБД используется СУБД с открытым исходным кодом PostgreSQL. Однако стоимостной планировщик запросов основан на один и тех же принципах во всех реляционных СУБД, поэтому полученные результаты применимы и в них.

В этом разделе будут упоминаться детали реализации, необходимые для воспроизводимости экспериментов. Для всех таких деталей в тексте явно указано, что они специфичны именно для СУБД PostgreSQL.

Для почти любого SQL-запроса существует несколько возможных планов его выполнения. Поэтому задача выбора плана выполнения запроса появилась вместе с первыми реляционными СУБД. К её решению существует два подхода: эвристическая оптимизация (англ. rule-based [6]) и стоимостная оптимизация (англ. cost-based [1]). Стоимостная оптимизация запросов использовалась в первой реляционной СУБД System R [25] и применяется сейчас практически во всех наиболее популярных реляционных СУБД. Эвристическая оптимизация не получила такого широкого распространения. Подробнее об оптимизации запросов в различных СУБД можно почитать здесь [5].

Основная идея стоимостной оптимизации запросов заключается в том, что для каждого плана выполнения запроса мы можем оценить стоимость его выполнения. Тогда задача выбора оптимального плана превращается в задачу минимизации функции стоимости плана в пространстве всех возможных планов выполнения данного запроса:

$$J(x) \to \min_{x \in all\_plans(query)}$$

Для решения задачи минимизации могут использоваться различные алгоритмы. Однако мало алгоритмов дискретной оптимизации работают в некотором подпространстве пространства деревьев. Легко показать, что количество планов растет как минимум экспоненциально от количества объединяемых в запросе отношений, поэтому полный перебор всех планов не подойдет в качестве решения.

В System R в качестве алгоритма оптимизации использовалось динамическое программирование по подмножествам [5]. Этот алгоритм гарантирует нахождение точного минимума оптимизируемой функции и работает существенно быстрее полного перебора. Однако динамическое программирование по подмножествам все равно требует $O(2^n)$ памяти и имеет вычислительную сложность $O(3^n)$, где n — количество объединяемых в запросе отношений. Поэтому в PostgreSQL для маленьких n используется динамическое программирование, а для больших n используется генетический алгоритм [20].

В большинстве реальных SQL–запросов объединяется не очень большое количество отношений, а метод динамического программирования по подмножествам в таком случае гарантирует оптимальность решения. Поэтому в данной работе мы сосредоточимся на функции определения стоимости плана. Для определения стоимости плана последовательно решаются две подзадачи: сначала для каждой вершины находится её мощность, затем на основе моделей операторов в вершине вычисляется её стоимость.

Для нахождения мощности вершины требуется совместную выборочность всех условий в вершине умножить на количество кортежей, которые она будет обрабатывать. Количество обрабатываемых кортежей — это либо произведение количеств кортежей, выдаваемыми сыновьями данной вершины, если вершина соответствует операции объединения двух отношений, либо количество строк в таблице, если вершина соответствует сканированию таблицы. Совместная выборочность условий в вершине вычисляется как произведение маргинальных выборочностей условий в этой вершине (т. е. предполагается независисмость условий). Маргинальные выборочности условий в вершине эффективно находятся с помощью гистограмм, индексов и различных эвристик [1].

Стоимость выполнения вершины оценивается через количество обрабатываемых в вершине кортежей, стоимости выполнения её сыновей и модель оператора в вершине. В СУБД PostgreSQL модели оператора являются линейными регрессиями с пятью весами, соответствующими различным операциям, выполняемым оператором: последовательного доступа к странице, случайного доступа к странице, обработке кортежа на процессоре, обработка кортежа из индекса на процессоре, выполнение какой-либо операции на процессоре. Например, для вершины, соответствующей оператору сортировки в памяти, стоимость выполнения будет равна сумме стоимости выполнения единственного сына и числа $1.39 \cdot 2 \cdot c_o \cdot N \log N$, где c_o — стоимость применения оператора на процессоре, $2 \cdot c_o$ — стоимость одного сравнения пары элементов, 1.39 — средняя константа при асимптотике времени выполнения, $N \log N$ — асимптотика времени выполнения.

В алгоритме динамического программирования не оценивается сразу весь план. Для каждого подмножества отношений строится план наименьший стоимости, объединяющий их с учетом всех условий, перечисленных в запросе. Поэтому по факту оракул отвечает на два типа запросов:

1. для оператора сканирования таблицы: по количеству записей в таблице и набору условий требуется оценить количество кортежей, удовлетворяющих указанным условиям;

2. для оператора объединения двух отношений: по количеству кортежей в объединяемых отношениях и набору условий объединения требуется оценить количество кортежей в результирующем отношении.

1.3 Обзор литературы

Существует ряд работ [22], [18], [12], [24], [19], посвященных предсказанию времени выполнения запроса с использованием методов машинного обучения. Однако в них решается не задача предсказания времени выполнения любого плана, а задача предсказания времени выполнения плана, выбранного оптимизатором запросов. То есть в этих работах строится оракул $J(x)$ не на всех возможных планах, а только на существенно меньшем подмножестве

$$\left\{ \underset{x \in all\_plans(query)}{\operatorname{argmin}} J(x) \middle| query \in Queries \right\}$$

Естественно, такую задачу решать легче, чем поставленную перед нами. Более того, на практике мы не сможем выполнить большинство планов за разумное время, чтобы получить хорошую обучающую выборку. Поэтому данные методы неприменимы к поставленной задаче.

Есть ряд работ [27], [23], [29], [30], основанных на сэмплировании. Они, как и в предыдущем абзаце, не решают задачу построения оракула, но предлагают подход для исправления ошибок оракула для плана, выбранного оптимизатором. Они используют принцип сэмплирования из данных, что показывает хорошие результаты на запросах, содержащих мало операций объединения отношений, но может приводить к слишком большой дисперсии для сложных запросов.

Перейдем к обзору статей, применимых к нахождению функции стоимости плана. Наиболее важной здесь является статья [15], в которой устанавливается, что оценка количества кортежей в вершинах совершается с большими ошибками, и именно эти ошибки чаще всего ведут к выбору неоптимального плана. Зависимость плана от стоимостных моделей операторов намного меньше. Аналогичные результаты были независимо получены в данном исследовании и будут приведены в разделе 3.2.

Однако в некоторых работах рассматриваются способы улучшения моделей операторов без улучшения оценок количества кортежей в вершинах. Например, в работе [23] предлагается использовать метод наименьших квадратов для настройки весов различных действий в стоимостной модели PostgreSQL, а в работе [18] строится своя линейная модель с большим количеством признаков.

Несмотря на то, что наибольшим недостатком стоимостного оптимизатора запросов в реляционных СУБД является именно оценка количества кортежей в вершине, исчерпывающих решений проблемы до сих пор предложено не было.

В СУБД–сообществе популярным решением проблемы коррелированных условий являются многомерные гистограммы [10], [21], [16], [3], [8], [31], [11]. Их недостатками являются рост потребляемых ресурсов и памяти с увеличением размерности, сложность определения, для каких столбцов нужно строить многомерные гистограммы, а какие являются независимыми или не упоминаются совместно в запросах. Отметим, что в работе [21] также рассказывается о возможности применения SVD–разложения для определения выборочности условий на отрезках для вещественных данных, однако это недостаточно общий подход, поэтому в существующих СУБД

он реализован не был. Работа [31] содержит также совмещение сэмплирования и многомерных гистограмм.

Работ по применению машинного обучения к задаче определения количества кортежей в вершине, или, что эквивалентно, определению выборочности набора условий, сейчас существует немного.

В работе 1998 года [17] предлагается использовать нейросети для оценки выборочностей условий, однако в ней рассматривается только задача построения функции выборочности для определенных пользователем типов и функций.

В работе 2001 года [9] предлагается оценивать совместную выборочность условий используя вывод в автоматически сгенерированных по данным байесовским сетям. Для построения хорошо описывающих данные сетей используется принцип максимизации правдоподобия, а для выбора структуры сети — различные эвристики. Отсюда следует большой недостаток метода, указанный авторами в статье — невозможность применения его на достаточно больших данных, которые часто встречаются в современных СУБД. Также задача автоматического построения байесовских сетей по данным не считается решенной, поэтому качество и стабильность работы предложенного авторами решения вызывает сомнения. В работе не было проведено экспериментов по использованию данного метода для выбора плана выполнения запросов в СУБД.

Работа 2009 года [13] посвящена применению формулы Байеса для определения совместной выборочности условий с использованием совместной статистики на данных. Эта работа содержит ряд предположений и эвристик, которые далеко не всегда выполняются на реальных данных. В работе не было проведено экспериментов по использованию данного метода для выбора плана выполнения запроса в СУБД. Также в данной работе не исследованы запросы, в которых сканируется более чем одна таблица.

В работе 2015 года [4] было предложено использовать нейронные сети для оценки совместной выборочности набора условий на отрезках. В этой работе нейронные сети выполняют ту же функцию, что и многомерные гистограммы, однако лишены проклятия размерности. Также в данной работе не исследованы запросы, в которых сканируется более чем одна таблица. В работе отсутствует информация о количестве нейронов в слоях предлагаемой авторами нейронной сети, что не позволяет в точности повторить их эксперименты. Существенным недостатком работы является отсутствие информации о том, сколько времени и запросов требуется для обучения нейронной сети на предложенной ими на базе данных. Естественно, при достаточно долгом обучении на фиксированных данных и достаточно большом количестве нейронов в слоях можно получить идеальные предсказания выборочности. Однако практическая применимость нейронных сетей для задачи предсказания выборочности определяется ресурсоемкостью обучения и необходимым для него количеством входных данных. Никаких данных по этим вопросам авторы в своей статье не приводят.

2 Предложенный метод оценки совместной выборочности

2.1 Формализация задачи

Наиболее критичной для производительности СУБД частью является оценка совместной выборочности условий, что будет показано в разделе 3.2. Поэтому именно для этой задачи мы будем искать решения с использованием методов машинного обучения.

Ключевым моментом данной работы является введение признакового пространства. На вход оракулу подается набор условий и их маргинальных выборочностей, требуется оценить их совместную выборочность. Будем считать признаками логарифмы маргинальных выборочностей условий. Условия, отличающиеся только в константах, будем считать эквивалентными. Можно рассматривать данное допущение как типичный прием машинного обучения — hashing trick — примененный для уменьшения размерности пространства. Однако за этим стоит более мощная мотивация: мы предполагаем, что вся необходимая для предсказания информация о константах условия содержится в его маргинальной выборочности. Можно показать это строго для простых условий вида a $<$ const: здесь по выборочности условия мы можем восстановить значение константы, то есть потери информации не происходит.

Вообще говоря, полученное признаковое пространство не является конечномерным, поскольку потенциально существует бесконечное множество различных условий в SQL–запросах. Однако на практике количество различных условий ограничено и невелико. Тем не менее, при разработке методов следует либо заранее считать, что метод будет работать только с запросами, содержащие только условия из обучающей выборки, либо использовать методы с возможностью добавления признаков «на лету».

Мотивацией выбора логарифмов выборочностей служит то, что после логарифмирования ошибка выборочности будет равна ошибке логарифма мощности множества кортежей в вершине: $\log Cardinality = \log Selectivity + \log Number\_of\_tuples$. Также при логарифмировании линейная модель с единичными весами соответствует стандартной для СУБД модели выборочности, где все условия полагаются независимыми.

Также заметим, что в этой задаче присутствует обратная связь: распределение запросов к оракулу изменяется в зависимости от возвращаемых им результатов, поскольку он является составной частью процесса оптимизации. Поэтому нельзя просто собрать статистику по выполнению запросов, применить её и получить гарантированное улучшение результата. В результате обучения на статистике, собранной по планам выполнения запросов, мы получаем изменившийся оптимизатор запросов, который будет выбирать другие планы. Этих планов и их комбинации условий не было в нашей обучающей выборке поэтому наши предсказания для них могут сколь угодно сильно отличаться от истинных. Однако если мы на каждой итерации будем добавлять статистику по новым планам в обучающую выборку, то

рано или поздно этот процесс сойдется к некоторому аналогу точки локального оптимума. Это будет показано в теореме 1.

Таким образом мы получаем итеративный алгоритм обучения 2.1.

Algorithm 2.1 Алгоритм итеративного обучения модели

Вход: Q — конечный набор запросов, на котором происходит обучение

Вход: $\mathbb{A}$ — множество всех возможных регрессоров

Вход: $\mathbb{X}$ — множество всех возможных наборов условий с маргинальными выборочностями

Вход: $Execute : Q \times \mathbb{A} \to 2^{\mathbb{X}\times\mathbb{R}[0,1]}$ — функция из СУБД для выполнения запроса $q \in Q$, использующая для поиска оптимального плана регрессор $R \in \mathbb{A}$ и возвращающая наборы условий с маргинальными выборочностями и истинными совместными выборочностями $(x, t) \in \mathbb{X}\times\mathbb{R}[0, 1]$ для каждой вершины выполненого плана

Вход: $Learn : 2^{\mathbb{X}\times\mathbb{R}[0,1]} \to \mathbb{A}$ — алгоритм машинного обучения, возвращающий по собранной статистике $S \in 2^{\mathbb{X}\times\mathbb{R}[0,1]}$ обученный регрессор $R \in \mathbb{A}$

Выход: Обученный регрессор $R \in \mathbb{R}$

Инициализировать регрессор R_0
$S_0 \leftarrow \varnothing$
для $i = 1, 2, \ldots$
  $S_i \leftarrow S_{i-1}$
  $R_i \leftarrow R_{i-1}$
  для всех $q \in Q_i$
    $S_i \leftarrow Execute(q, R_i) \cup S_i$
    $R_i \leftarrow Learn(S_i)$
  если $R_{i-1} = R_i$ **то**
    выход

2.2 Теоретические гарантии

Теорема 1. *Пусть в алгоритме 2.1 выполняются следующие условия:*

- *процедура обучения Learn такова, что* $\forall(x, t) \in S \to Learn(S)(x) = t$, *т. е. обученный с её помощью регрессор идеально точно работает на объектах обучающей выборки;*
- $\forall i \in \{0, 1, \ldots\}$ *обучающая выборка* S_i *обладает свойством однозначности:* $\nexists(x_1, t_1), (x_2, t_2) \in S_i : x_1 = x_2$ *и* $t_1 \neq t_2$, *то есть для любого набора условий и их маргинальных выборочностей существует единственная истинная совместная выборочность.*

Тогда:

- *Для фиксированного конечного множества запросов Q, не изменяющих данные, вышеописанный алгоритм 2.1 сойдется (то есть регрессор R_i, и, следовательно, выбираемые оптимизатором планы перестанут меняться, алгоритм остановится) за конечное число шагов к обучающей выборке S и регрессору R, который мы назовем «локальным оптимумом».*
- $\forall (x,t) \in S \to R(x) = t$, *т. е. полученный регрессор R будет идеально точно оценивать количество кортежей в каждой вершине каждого когда–либо выполненного в ходе обучения плана.*

Доказательство. Для фиксированного конечного Q количество различных планов ограничено. Поэтому ограничено количество различных объектов в обучающей выборке $|S_i| < S_{max}$. Значит, это количество рано или поздно перестанет увеличиваться. Пусть алгоритм не сойдется за конечное число шагов. Тогда поскольку $\forall i \in 1, 2, \ldots\ S_{i-1} \in S_i$ и $|S_i| < S_{max}$, то для некоторого k будет выполнено $S_k = S_{k+1}$. Тогда $R_k = Learn(S_k) = Learn(S_{k+1}) = R_{k+1}$, поэтому алгоритм остановится. Противоречие. Значит, алгоритм должен сойтись за конечное число шагов.

Поскольку алгоритм обучения таков, что $\forall (x,t) \in S \to Learn(S)(x) = t$, то регрессор будет идеально точно оценивать количество кортежей в каждой вершине каждого когда–либо выполненного в ходе обучения плана. ■

Замечание 1.1. *На практике сходимость метода наступает через несколько итераций.*

Замечание 1.2. *Также на практике обычно не строго выполняются предположения про регрессор, неизменность данных и конечное множество запросов.*

Замечание 1.3. *Отметим, что в данном алгоритме регрессор всё равно будет делать предсказания выборочности для тех наборов условий, которые никогда не были в обучающей выборке, и эти предсказания могут быть сколь угодно неточными. Именно поэтому мы не можем гарантировать глобальную оптимальность выбираемых планов.*

Теорема 2. *Пусть выполнены условия теоремы 1.*

Пусть также к результатам регрессора добавляется независимый для разных объектов случайный шум, такой что в итоге выборочность распределена на отрезке $[0,1]$ и для любого объекта и любой точки $x \in [0,1]$ плотность вероятности выборочности в x больше некоторого положительного ε.

Также потребуем, чтобы для любого плана, получаемого в результате ответов оракула $s_1, s_2, \ldots, s_n$ на объекты $x_1, x_2, \ldots, x_n$, существовали $\varepsilon_{l,1}\varepsilon_{r,1}$, $\varepsilon_{l,2}, \varepsilon_{r,2}, \ldots, \varepsilon_{l,n}, \varepsilon_{r,n} \geqslant 0$, такие что $\varepsilon_{l,1} + \varepsilon_{r,1}, \varepsilon_{l,2} + \varepsilon_{r,2}, \ldots, \varepsilon_{l,n} + \varepsilon_{r,n} > 0$ и при любых ответах оракула $\hat{s_1} \in [s_1 - \varepsilon_{l,1}, s_1 + \varepsilon_{r,1}]$, $\hat{s_2} \in [s_2 - \varepsilon_{l,2}, s_2 + \varepsilon_{r,2}]$, $\ldots$, $\hat{s_n} \in [s_n - \varepsilon_{l,n}, s_n + \varepsilon_{r,n}]$ набор запросов к оракулу не изменялся и в итоге был выбран тот же план.

Тогда процесс обучения сойдется глобально (то есть ответы регрессора стабилизируются и будут идеально точны для всех возможных условий в вершинах) за конечное число шагов с вероятностью 1.

Доказательство. По условию теоремы, для любого плана, который может быть выбран, существуют такие $s_1, \varepsilon_{l,1}, \varepsilon_{r,1}, s_2, \varepsilon_{l,2}, \varepsilon_{r,2} \ldots, s_n, \varepsilon_{l,n}, \varepsilon_{r,n}$ что для любая последовательность ответов оракула $\hat{s_1} \in [s_1 - \varepsilon_{l,1}, s_1 + \varepsilon_{r,1}], \hat{s_2} \in [s_2 - \varepsilon_{l,2}, s_2 + \varepsilon_{r,2}], \ldots, \hat{s_n} \in [s_n - \varepsilon_{l,n}, s_n + \varepsilon_{r,n}]$ приводит к его выбору. По условию теоремы добавляемый шум таков, что вероятность получить последовательность ответов $\hat{s_1} \in [s_1 - \varepsilon_{l,1}, s_1 + \varepsilon_{r,1}], \hat{s_2} \in [s_2 - \varepsilon_{l,2}, s_2 + \varepsilon_{r,2}], \ldots, \hat{s_n} \in [s_n - \varepsilon_{l,n}, s_n + \varepsilon_{r,n}]$ не менее $p = \prod_{i=1}^{n} \varepsilon \cdot (\varepsilon_{l,i} + \varepsilon_{r,i})$. Поскольку это число в течение всего процесса обучения постоянно и больше нуля, то рано или поздно этот план будет выполнен с вероятностью $\lim_{m \to \infty} 1 - (1-p)^m = 1$.

Таким образом, введенный подобным образом шум гарантирует, что любой план, который может быть выбран, будет выполнен через какое–то конечное число шагов с вероятностью 1. Поэтому за конечное число шагов будут исследованы все возможные условия в вершинах, и тогда регрессор будет выдавать абсолюно верные предсказания для всех этих условий по первой теореме. ■

Следствие 2.1. *Пусть выполняются условия теоремы, стоимостная модель операторов идеальна, а метод оптимизации точен. Тогда после глобальной сходимости регрессора все выбираемые оптимизатором планы будут оптимальны.*

Замечание 2.1. *На практике выполняется предположение об «устойчивости» метода оптимизации, то есть о том, что при небольшом изменении ответов оракула выбранный план не изменится. Таким образом показывается важность задачи исследования пространства планов: при исследовании можно получить намного более сильные гарантии.*

Замечание 2.2. *На практике полученные гарантии имеют такую же ценность, как способность метода Монте–Карло находить глобальный минимум. Количество итераций с использованием приведенного в теореме шума до сходимости слишком велико для того, чтобы иметь практическую ценность. Однако теорема призвана показать, что*

1. *даже достаточно простой метод является достаточно мощным, чтобы гарантировать глобальную сходимость;*

2. *существуют методы исследования пространства планов, позволяющие гарантировать глобальную сходимость, то есть заметно усилить теоретические гарантии предыдущей теоремы;*

3. *разработка более интеллектуальных методов исследования пространства планов перспективна, поскольку позволяет получать гарантии глобальной сходимости.*

2.3 Параметрические методы

В качестве представителя параметрических методов, позволяющего добавлять признаки «на лету», мы выбрали гребневую линейную регрессию [14] с регуляризатором вида $\lambda \sum_i (w_i - 1)^2$.

Другим представителем параметрического метода являются нейронные сети. Они интересны для нашей задачи, потому что нейронная сеть теоретически может сколь угодно точно приближать распределение любой сложности при достаточном размере скрытого слоя. С другой стороны, нейронная сеть является параметрическим методом, поэтому для её обучения могут применяться стохастические методы оптимизации, что позволяет не хранить всю обучающую выборку.

Заметим также, что предложенный в работе [4] способ применения нейронных сетей является частным случаем предложенной разделе 2.1 постановки задачи для листовых вершин плана. Поэтому проведенные в разделе 3.6 эксперименты можно рассматривать как сравнение с предложенным в работе [4] методом.

2.4 Непараметрические методы

В качестве представителя параметрических методов, позволяющего добавлять признаки «на лету», мы выбрали регрессионную модификацию метода k ближайших соседей [2], где предсказание делается на основе взвешенных значений k ближайших соседей. Веса соседей вычисляются как обратное расстояние от объекта до них и затем нормируются. В качестве функции расстояния используется квадрат евклидового расстояния между объектами в признаковом пространстве.

$$\tilde{w_{(i)}} = \frac{1}{1 + dist(x, x_{(i)})}$$

$$w_{(i)} = \frac{\tilde{w_{(i)}}}{\sum\limits_{j=1}^{k} \tilde{w_{(j)}}}$$

$$\hat{y} = \sum_{i=1}^{k} w_{(i)} y_{(i)}$$

Единица в знаменателе введена для численной устойчивости метода и не влияет на его результаты. При большом количестве объектов необходимо использовать эффективные структуры данных для быстрого поиска ближайших соседей.

Другим исследуемым методом является градиентный бустинг над решающими деревьями [7]. Его существенным недостатком является невозможность дообучения. Единственным способом учитывать новую поступающую информацию является обучение нового регрессора с нуля над обновленной обучающей выборкой. Это требует хранения всей обучающей выборки, поэтому метод отнесен в данной работе к непараметрическим. Также дообучение занимает достаточно много времени.

Минусом непараметрических методов является необходимость хранить все объекты обучающей выборки, что приводит к их практической неприменимости.

Использованный в данной работе способ устранения этого недостатка приведен в разделе 2.5. Заметим однако, что проблема оптимизации памяти и времени дообучения различных методов машинного обучения в изменяемом признаковом пространстве интересна сама по себе и может служить темой отдельного исследования.

2.5 Используемые эвристики

В предложенных непараметрических методах есть очевидная проблема: мы запоминаем все объекты. Естественно, даже на слабо нагруженных СУБД очень скоро алгоритм соберет гораздо больше объектов, чем мы можем хранить в памяти, а на сильно нагруженных СУБД этих объектов будет даже больше, чем можно обрабатывать без существенного снижения производительности системы. Однако далеко не все объекты содержат новую или полезную информацию. Отсюда возникает задача отбора объектов. В данной работе для решения этой задачи была применена следующая эвристика: если предсказание на объекте хорошее, то есть предсказанный логарифм выборочности отличается от истинного значения не более чем на $\delta = 0.1$, то считается, что этот объект не содержит новой информации и потому не добавляется в обучающую выборку.

Таким образом, мы получаем алгоритм 2.2.

Algorithm 2.2 Алгоритм итеративного обучения модели

Вход: Q — конечный набор запросов, на котором происходит обучение

Вход: $\mathbb{A}$ — множество всех возможных регрессоров

Вход: $\mathbb{X}$ — множество всех возможных наборов условий с маргинальными выборочностями

Вход: $Execute : Q \times \mathbb{A} \to 2^{\mathbb{X}\times\mathbb{R}[0,1]}$ — функция из СУБД для выполнения запроса $q \in Q$, использующая для поиска оптимального плана регрессор $R \in \mathbb{A}$ и возвращающая наборы условий с маргинальными выборочностями и истинными совместными выборочностями $(x, t) \in \mathbb{X}\times\mathbb{R}[0, 1]$ для каждой вершины выполненого плана

Вход: $Learn : 2^{\mathbb{X}\times\mathbb{R}[0,1]} \to \mathbb{A}$ — алгоритм машинного обучения, возвращающий по собранной статистике $S \in 2^{\mathbb{X}\times\mathbb{R}[0,1]}$ обученный регрессор $R \in \mathbb{A}$

Вход: $\delta \in \mathbb{R}_+$

Выход: Обученный регрессор $R \in \mathbb{R}$

Инициализировать регрессор R_0
$S_0 \leftarrow \varnothing$
для $i = 1, 2, \ldots$
  $S_i \leftarrow S_{i-1}$
  $R_i \leftarrow R_{i-1}$
  для всех $q \in Q_i$
    $S_{i,q} \leftarrow \left\{(x, t)\middle|(x, t) \in Execute(q, R_i) \text{ и } |R_i(x) - t| > \delta\right\}$
    $S_i \leftarrow S_{i,q} \cup S_i$
    $R_i \leftarrow Learn(S_i)$
  если $R_{i-1} = R_i$ **то**
    выход

3 Вычислительные эксперименты

3.1 Исходные данные и условия эксперимента

Для экспериментов в этой работе используются две базы данных.

StrongCor. Эта синтетическая база данных предложена в работе [4] специально для определения качества оценки выборочности условий. База данных состоит из одной таблицы, содержащей 15 столбцов и 40000 строк. Таблица генерируется следующим образом:

$$x_{i,1} = \text{randInt}(0, 100)$$
$$x_{i,n} = (x_{i,n-1} + \text{randInt}(2, 3)) \bmod 100$$

$\text{randInt}(a, b)$ возвращает равновероятно случайное целое число от a до b включительно.

В этом наборе данных будем генерировать запросы по следующему правилу:

1. сгенерируем равновероятно случайное подмножество размера c из первых C столбцов: $columns \subseteq 1, \ldots, C, |columns| = c$;
2. для каждого столбца $columns_i$ сгенерируем отрезок $[a_i, b_i] \subseteq [0, 100]$:
   (a) сначала равновероятно генерируем длину отрезка $l_i \in \{0, 1, \ldots, 101\}$;
   (b) затем равновероятно генерируем первую точку $a_i \in \{0, 1, \ldots, 100 - l_i + 1\}$;
   (c) получаем отрезок $[a_i, a_i + l_i - 1]$;
3. получаем запрос
   `SELECT * FROM X WHERE`
   $a_1 \leqslant x_{columns_1}$ `AND` $x_{columns_1} \leqslant b_1$
   `AND` $a_2 \leqslant x_{columns_2}$ `AND` $x_{columns_2} \leqslant b_2$
   `AND ...AND` $a_C \leqslant x_{columns_C}$ `AND` $x_{columns_C} \leqslant b_C$.

Запросы в наборе данных StrongCor построены таким образом, что план содержит только одну вершину, поэтому в этом экспериментах с этими данными одна итерация обучения, один запрос, один план выполнения запроса и одна вершина плана являются синонимами.

Все тесты проводились на 11 заранее сгенерированных последовательностях из 30000 запросов. Ошибки усредняются по этим последовательностям. Также на некоторых графиках проводится дополнительно усреднение по окну. Также на графиках цветом отображено среднеквадратичное отклонение в точке, взятое по всем 11 последовательностям.

TPC–H. Для экспериментов в данной работе используется известный тест производительности для СУБД TPC–H [28]. Этот тест создан для оценок производительности СУБД и аппаратного обеспечения, поэтому максимально приближен к реальной нагрузке на СУБД.

Тесты производительности для СУБД TPC–H содержит генератор базы данных различных размеров, а также генератор запросов. Запрос генерируется методом подстановки случайно сгенерированных значений в один из выбираемых пользователем шаблонов запросов. Всего тесте производительности TPC–H есть 22 шаблона запросов. Таким образом моделируется реальные условия, когда все запросы к базе данных различны, но при этом есть ограниченное и небольшое количество структур этих запросов.

Все эти запросы не изменяют данные в базе, поэтому при отсутствии машинного обучения один и тот же запрос будет требовать примерно одинаковое количество ресурсов вне зависимости от того, когда он был выполнен. На самом деле, реальное время выполнения запроса существенно зависит от состояния кэшей СУБД, которое в свою очередь зависит от недавно выполнявшихся запросов. Однако оптимизатор запросов не учитывает такие детали. На практике для «прогревания» кэшей требуется один или несколько раз выполнить по запросу для каждого шаблона. В данной работе перед началом машинного обучения два раза выполняется по запросу для каждого шаблона, затем обученный регрессор сбрасывается и начинается измерение значений, связанных с ходом обучения.

В данной работе в тесте TPC–H была сгенерирована база данных размером 1 Гб.

В ходе экспериментов было установлено, что запросы в этом тесте генерируются таким образом, что совместная и магринальная выборочность набора условий практически не меняется. Это служит мотивацией рекомендовать к использованию в СУБД под подобной нагрузкой непараметрические методы, поскольку они сразу «запомнят» выборочности и начнут выдавать очень хорошие прогнозы.

Поэтому эксперименты на этом наборе данных служат в основном не для тестирования качества предсказания совместной выборочности, а для исследования стоимостного оптимизатора СУБД при типичной для СУБД нагрузке и для оценки практических результатов применения предложенного метода.

Особенности проведения экспериментов. Эксперименты, где исследуется скорость обучения различных методов, проводятся следующим образом: генерируются запросы и последовательно выполняются в СУБД. Для каждого запроса вычисляется ошибка предсказания выборочности его вершин оптимизатором запросов, затем этот запрос добавляется в обучающую выборку и переобучается регрессор. Для параметрических методов, позволяющих не хранить выборку и использовать стохастический градиентный спуск (SGD), просто совершается несколько итераций стохастического градиентного спуска по новым объектам.

Последовательный запуск по одному сгенерированному запросов каждого типа будет называть эпохой. Процесс обучения состоит из нескольких эпох.

Эксперименты проводились на ноутбуке со следующими характеристиками:

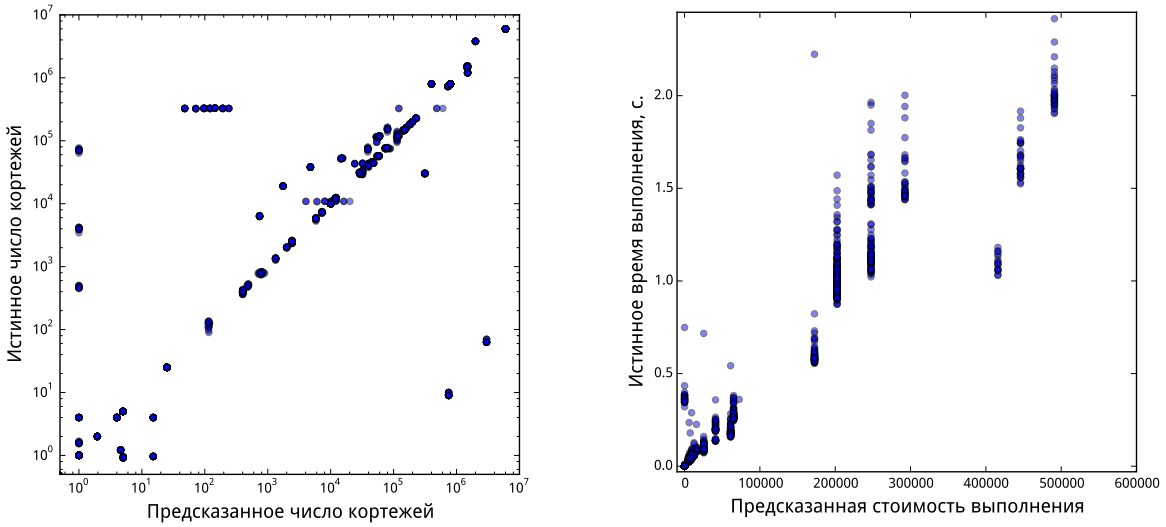
- Core i7
- 8Gb RAM
- Linux Ubuntu 14.04

В качестве реляционной СУБД использовалась СУБД с открытым исходным кодом PostgreSQL версии 9.6dev.

Использовались реализации градиентного бустинга над решающими деревьями и линейных моделей из библиотеки sklearn [1]. Использовалась реализация нейронной сети из библиотеки keras [2].

3.2 Оценка качества решения подзадач в стандартном оракуле стоимостной оптимизации

В этом разделе сравнивается качество предсказания выборочности и качество предсказания стоимости вершины плана.



а) каждая точка на графике соответствует вершине плана

б) каждая точка на графике соответствует вершине плана, в котором количество кортежей предсказано с относительной точностью до 10%

Рис. 2: Результаты работы стоимостного оптимизатора запросов

Каждая точка на графиках 2 соответствует одной вершине плана некоторого запроса. Для каждой вершины предсказано количество отобранных в ней кортежей

[1] http://scikit-learn.org/

[2] http://keras.io/

и стоимость ее выполнения, а затем измерены реальное количество отобранных кортежей и время выполнения. На рис. 2б отображены только те вершины, для которых количество кортежей предсказано с относительной точностью 10%, поэтому по ней можно судить о качестве стоимостных моделей операторов.

На рис. 2а видно, что результат решения первой подзадачи (оценка количества кортежей) может отличаться от истинного на несколько порядков. Из рис. 2б следует, что при правильной оценке количества кортежей стоимостная модель операторов достаточно адекватно оценивает время выполнения плана, поскольку видна сильная корреляция со временем выполнения, отсутствуют отклонения на порядок или более от прямой пропорциональности.

Таким образом, узким местом оптимизатора запросов является именно оценка количества кортежей в каждой вершине плана. Причиной этой ошибки является зависимость условий в запросах. Поэтому для запросов с зависимыми между собой условиями оптимизатор часто ошибается в оценках стоимости планов и, в результате, выбирает далеко не оптимальный план выполнения. К аналогичным результатам независимо пришла другая группа исследователей в работе [15].

3.3 Сравнение модификаций метода k ближайших соседей

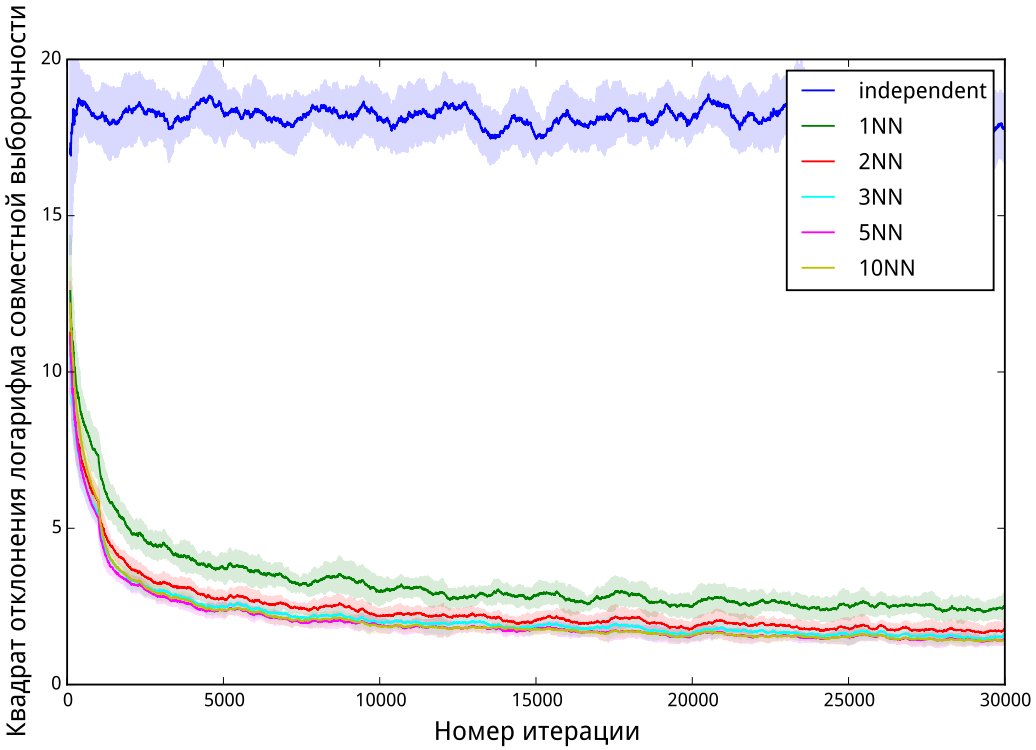


Рис. 3: Точность предсказания выборочности методом k ближайших соседей на StrongCor, усредненная с окном 1000, без эвристики отбора объектов, $c = C = 3$

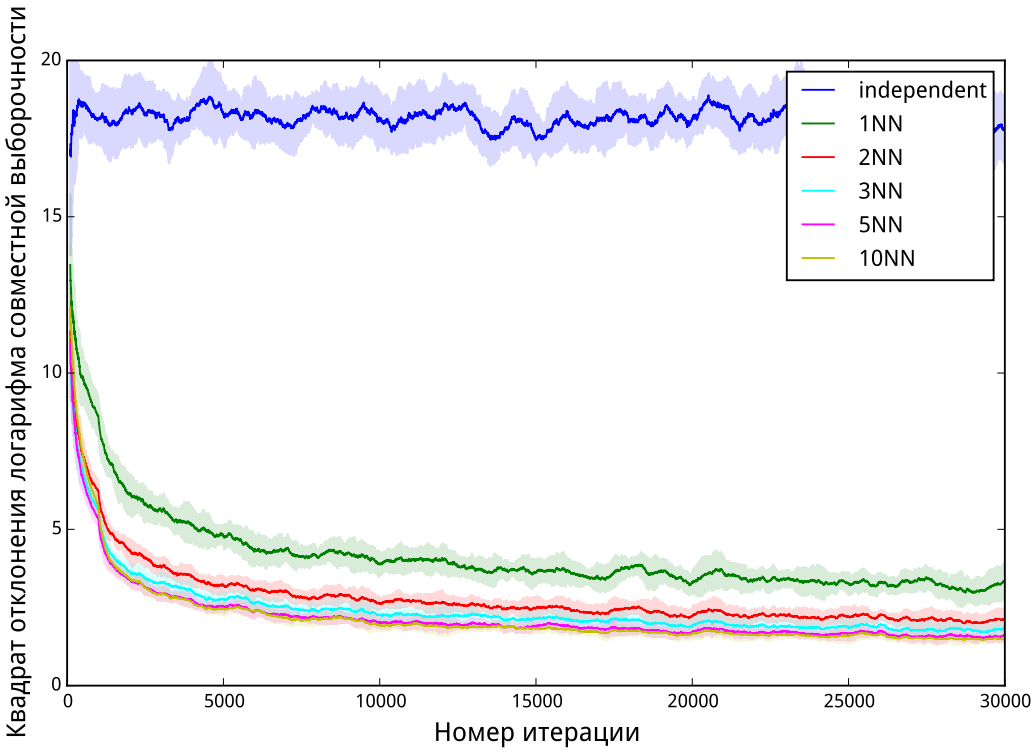
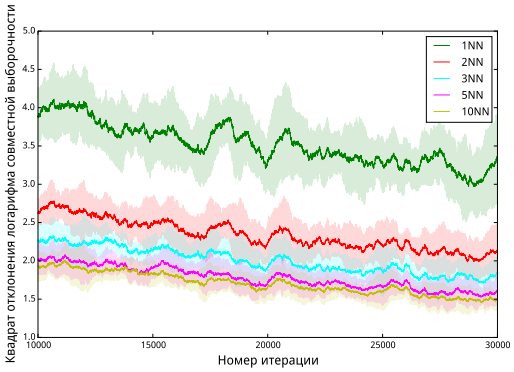
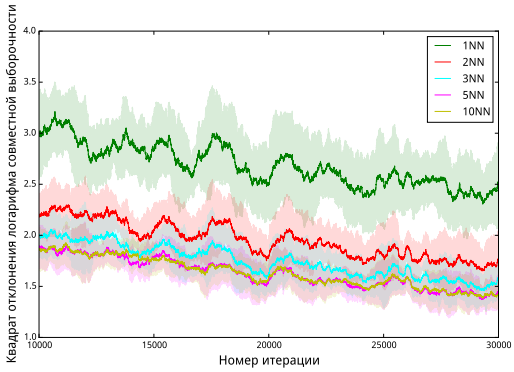


Рис. 4: Точность предсказания выборочности методом k ближайших соседей на StrongCor, усредненная с окном 1000, с эвристикой отбора объектов, $c = C = 3$



а) с эвристикой отбора объектов



б) без эвристики отбора объектов

Рис. 5: Точность предсказания выборочности методом k ближайших соседей на StrongCor, усредненная с окном 1000, $c = C = 3$

На рис. 3, 4 и 5 видно, что использование эвристики отбора объектов слабо ухудшает точность предсказания, но при этом позволяет не сохранять примерно половину объектов. На рис. 6 отображена зависимость между k и долей отбрасываемых объектов.

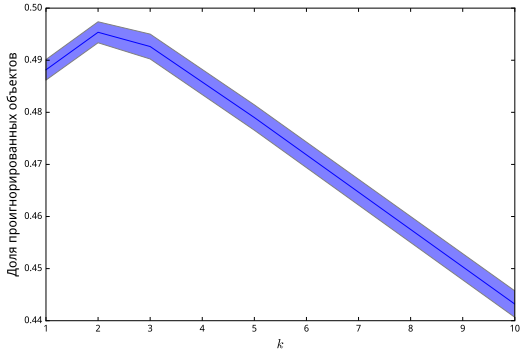


Рис. 6: Доля объектов, отброшенных эвристикой отбора объектов, при использовании метода k ближайших соседей на StrongCor $c = C = 3$

На рис. 4 мы видим, что метод k ближайших соседей предсказывает выборочность существенно лучше, чем стандартный оптимизатор с предположением о независимости условий. Видно, что метод ближайшего соседа работает заметно хуже, чем метод k ближайших соседей. На рис. 5 видно, что при увеличении k качество предсказания возрастает незначительно, в отличие от вычислительной сложности алгоритма. Поэтому по соображениям вычислительной сложности, качества предсказания и доли отбрасываемых эвристикой объектов разумнее всего использовать $k = 2$ или $k = 3$. В этой работе в дальнейшем будет использоваться $k = 3$.

3.4 Сравнение модификаций градиентного бустинга над решающими деревьями

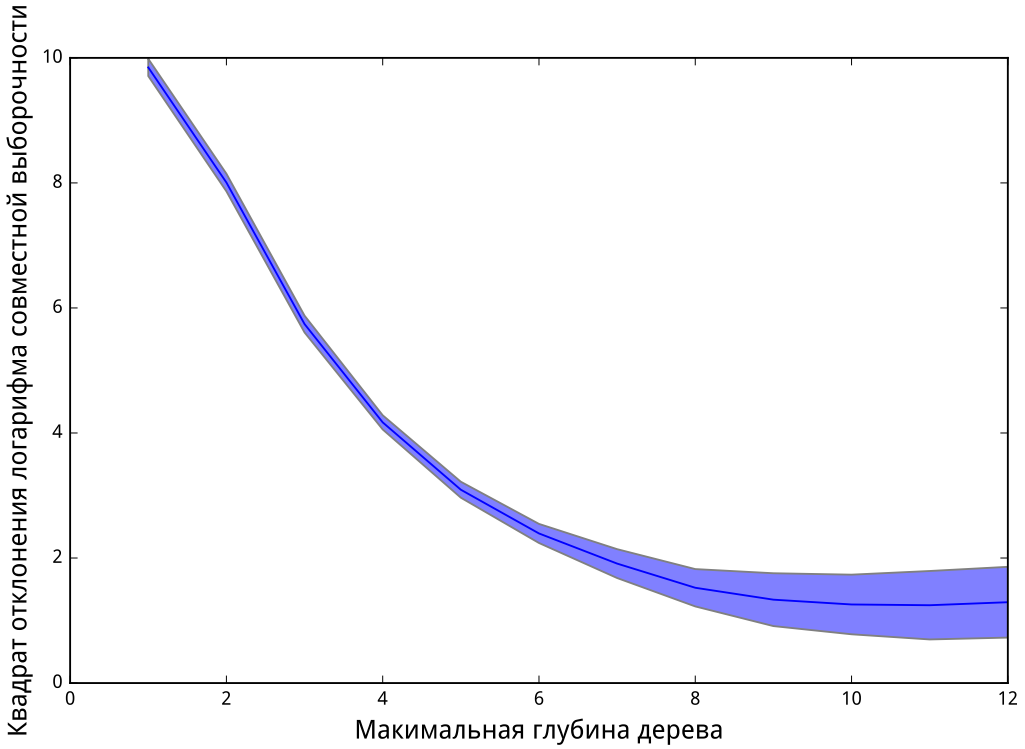
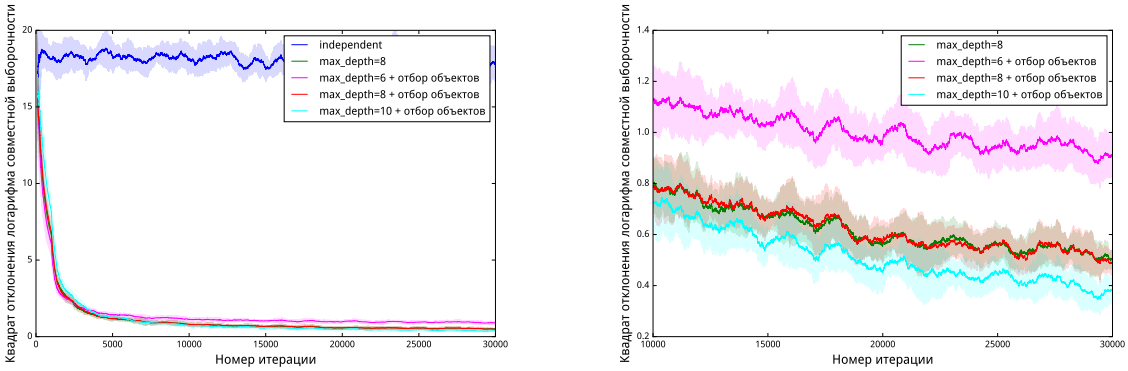


Рис. 7: Точность предсказания выборочности градиентным бустингом над решающими деревьями на StrongCor без эвристики отбора объектов, $c = C = 3$

На рис. 7 видно, что оптимальной максимальной глубиной деревьев является 10. На рис. 8а показана динамика обучения этой модификации градиентного бустинга. На ней видно, что глубина 10 ненамного лучше глубины 8, хотя обучается примерно в 5 раз медленнее. На рис. 8б, видно что эвристика отбора объектов не ухудшает качество работы. В эксперименте с градиентным бустингом над решающими деревьями с максимальной глубиной 8 было установлено, что эвристика позволяет не сохранять 0.39 от всех объектов, поэтому её использование целесообразно с этим методом.

В дальнейшим в экспериментах мы будем использовать градиентный бустинг над решающими деревьями с максимальной глубиной 8 и эвристикой отбора объектов.



а) б)

Рис. 8: Точность предсказания выборочности градиентным бустингом над решающими деревьями на StrongCor, $c = C = 3$

3.5 Сравнение модификаций линейной регрессии

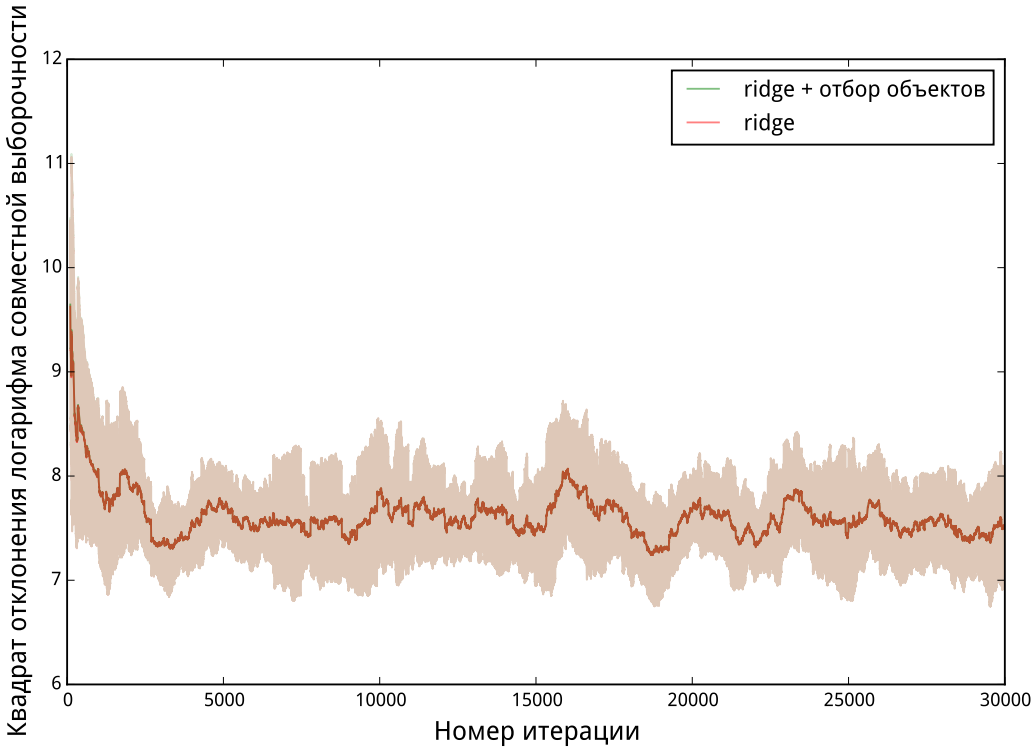


Рис. 9: Точность предсказания выборочности линейной моделью на StrongCor, усредненная с окном 1000, $c = C = 3$

На рис. 9 видно, что эвристика отбора объектов не ухудшает качество работы. Это можно объяснить тем, что если на объекте хорошее предсказание, то градиент по нему будет мал при текущих параметрах, поэтому точка оптимума изменится

слабо. Однако она позволяет не сохранять примерно 0.03 от всех объектов. Поэтому для данной модели её использование нецелесообразно.

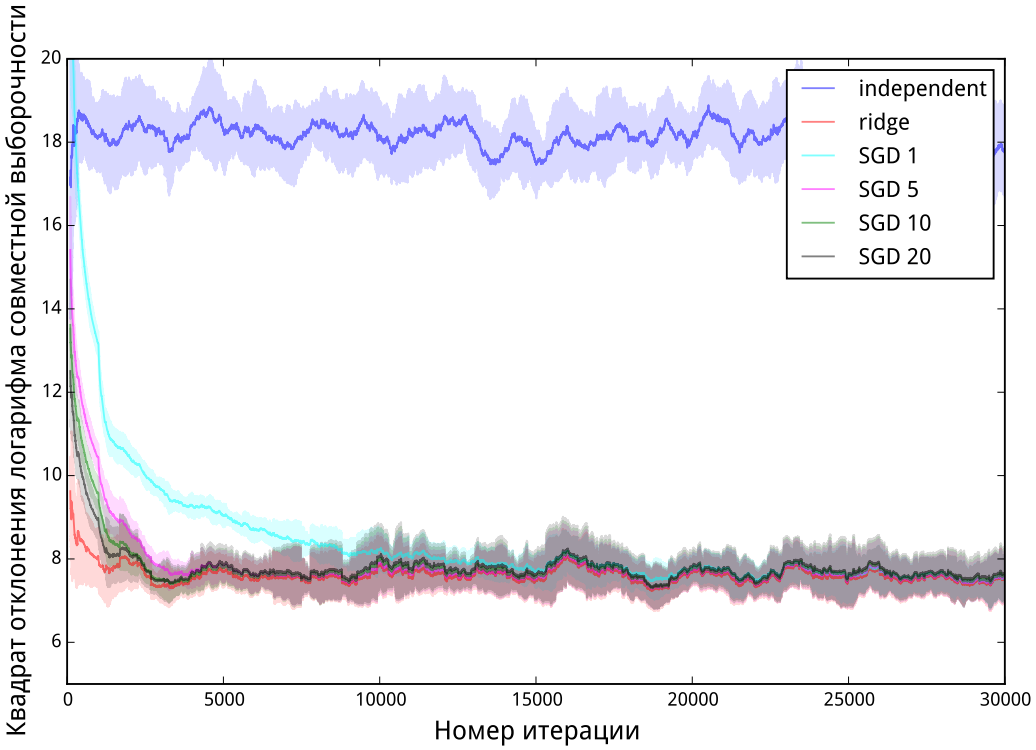


Рис. 10: Точность предсказания выборочности линейной моделью на StrongCor, усредненная с окном 1000, $c = C = 3$

На рис. 10 показана динамика обучения различных линейных моделей. Число после стохастического градиентного спуска показывает, сколько шагов по градиенту делается для одного объекта. Например, SGD 20 обозначает стохастический градиентный спуск, в котором для каждого объекта делается 20 итераций спуска по нему. Также видно, что метод градиентного спуска хотя и обучается медленнее, но сходится к тому же решению, что и метод гребневой регрессии (ridge). При этом метод стохастического градиентного спуска позволяет не хранить обучающую выборку, чем выгодно отличается от прочих методов.

В дальнейшем для экспериментов мы будем использовать стохастический градиентный спуск с 10 итерациями по объектам в качестве компромисса между скоростью обучения и его вычислительной сложностью, поскольку скорость спуска для 10 и для 20 итераций получается примерно одинаковой. К сожалению, результат, к которому сходится линейная модель, заметно хуже результатов других методов, что свидетельствует о том, что линейная модель плохо описывает закономерности в данных.

3.6 Сравнение модификаций нейронных сетей

Нейронные сети также могут обучаться стохастически, что позволяет не хранить всю обучающую выборку. Однако во всех техниках стохастического обучения нейронных сетей рекомендуется производить шаг не по одному объекту, а по некоторому батчу. Поэтому будем хранить последние B объектов, где B — размер батча. Когда в обучающую выборку приходит новый объект, происходит шаг стохастической оптимизации по батчу. Если в батче после этого более B объектов, то самый старый объект выкидывается.

Для экспериментов в данной работе использовалась полносвязнная нейронная сеть с одним скрытым слоем и сигмоидной функцией активации. Для настройки параметров сети применялся алгоритм RMSProp [26].

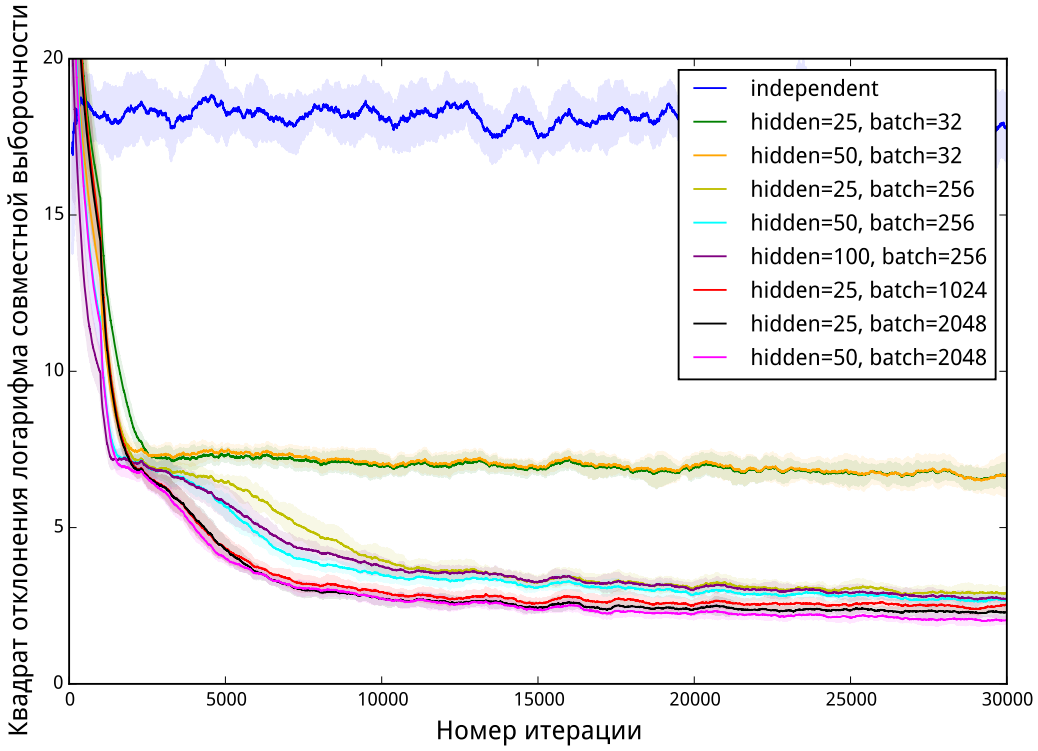


Рис. 11: Точность предсказания выборочности нейронной сетью на StrongCor, усредненная с окном 1000, $c = C = 3$

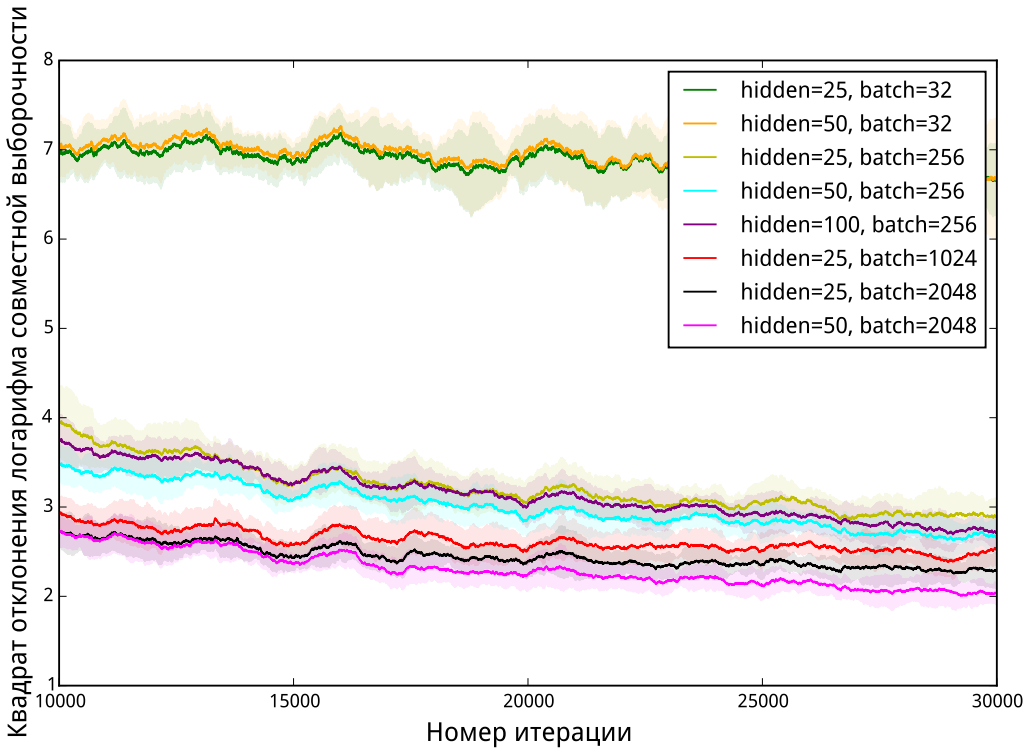


Рис. 12: Точность предсказания выборочности нейронной сетью на StrongCor, усредненная с окном 1000, $c = C = 3$

На рис. 11, 12 видно, что с увеличением размера батча увеличивается оптимальное количество нейронов на скрытом слое. Также видно, что чем больше батч, тем лучше результат обучения. Из соображений компромисса между точностью предсказания и скоростью работы будем использовать нейронную сеть с 50 нейронами на скрытом слое и размером батча 256.

3.7 Сравнение различных методов

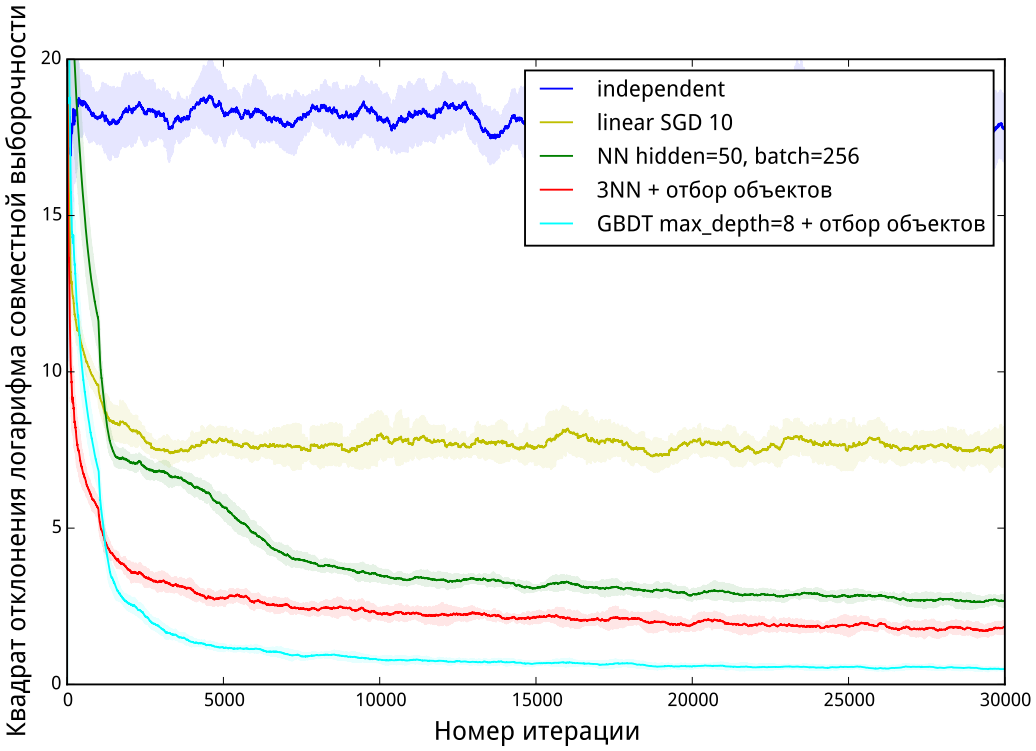


Рис. 13: Точность предсказания выборочности на StrongCor, усредненная с окном 1000, $c = C = 3$

На рис. 13 отображены отобранные представители каждого из методов. Видно, что наилучшие результаты показывает метод градиентного бустинга над решающими деревьями, однако он имеет ряд недостатков: сложность обучения и необходимость хранения обучающей выборки.

Нейронная сеть и метод k ближайших соседей показывают примерно одинаковые результаты. Нейронная сеть потенциально может показывать более хорошие результаты, однако для этого требуется больше времени. Также в ней есть больше параметров, которые необходимо настраивать: размер батча, размер скрытого слоя, особенности обучения.

Линейная модель, в отличие от вышеперечисленных методов, не может описывать сколь угодно сложные распределения. Видно, что линейная модель сошлась к точке оптимума, однако даже этой точке она плохо описывает истинную зависимость в данных.

3.8 Эксперименты на промышленных данных

Можно взять описанный выше алгоритм и запустить его на наборе данных TPC–H. Это приведет к тому, что один из выбранных планов будет настолько плох, что невозможно будет дождаться конца его выполнения. Это связано с тем,

что регрессору приходится находить совместную выборочность для таких наборов условий, которых нет в обучающей выборке. В таких случаях велика вероятность ошибки регрессора. Более того, на рис. 14 видно, что изначально некоторые регрессоры работают даже хуже, чем стандартная модель, предполагающая независимость условий. Из–за этой ошибки и получается план, который может выполняться очень долго.

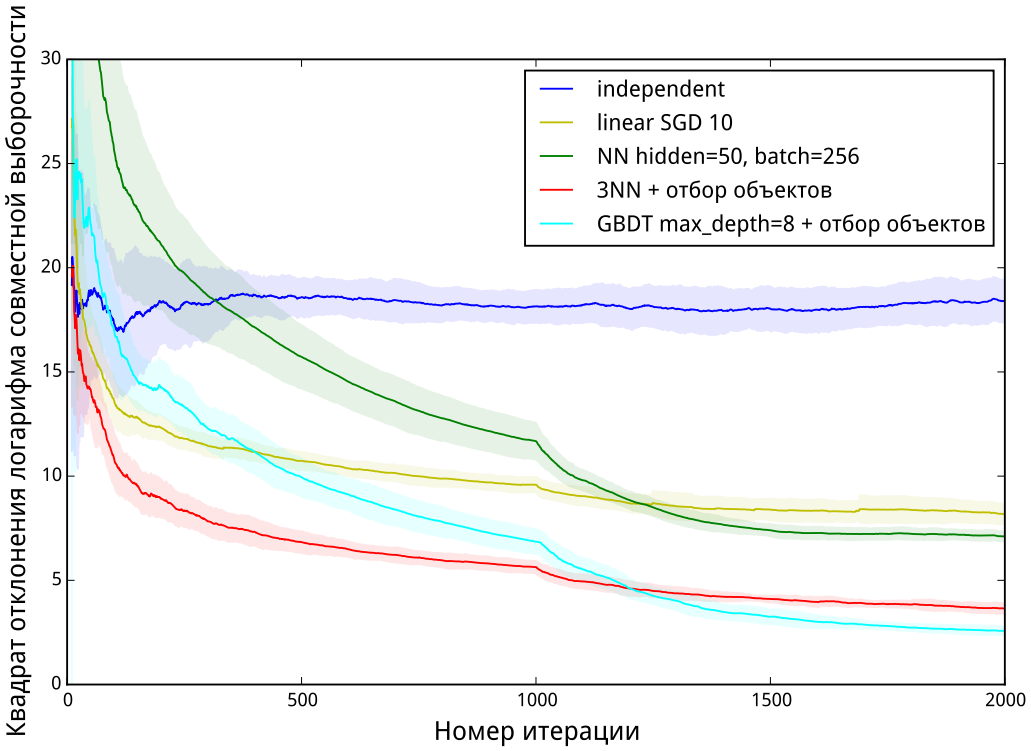


Рис. 14: Точность предсказания выборочности на StrongCor, усредненная с окном 1000, $c = C = 3$

Нельзя просто прервать выполнение плохого плана. Если информация из него никак не будет учтена, то в дальнейшем регрессор может продолжить выдавать ответы, ведущие к выбору того же плана. Есть два способа борьбы с этой проблемой.

Первый способ заключается в том, чтобы адаптировать методы машинного обучения под работу с незавершенными планами. В таком случае в выборке будет информация не только о том, что совместная выборочность условий равна какому-то числу, но и о том, что совместная выборочность условий превышает какое-то число.

Второй способ использует априорную информацию о свойствах методов. На рис. 14 видно, что сначала наилучшее качество предсказания дает стандартная модель, предполагающая независимость, затем наилучшие предсказания получаются с помощью модели k ближайших соседей, затем наилучший результат показывает градиентный бустинг над решающими деревьями. Возникает идея взять взвешенную комбинацию предсказаний нескольких методов и затем постепенно менять веса в этой комбинации.

В следующем эксперименте возьмем в качестве регрессора взвешенную линейную комбинацию градиентного бустинга над решающими деревьями и стандартной модели. Напомним, что регрессор предсказывает логарифм совместной выборочности. В качестве веса при предсказании градиентного бустинга возьмем меру уверенности в том, что у него было достаточно информации для обучения $w = \frac{n}{m+n}$, где n — количество таких же наборов условий в обучающей выборке, m — некоторая константа, регулирующая скорость убывания веса у стандартной модели. В последующих экспериментах было использовано $m = 10$. Тогда итоговое предсказание $\hat{y} = w \cdot \hat{y}_{GB} + (1 - w) \cdot \hat{y}_{independent}$. Возможны и другие способы задания весов над различными алгоритмами.

Будем последовательно генерировать и запускать все запросы из TPC–H, которые выполняются менее 10 секунд с помощью стандартной модели в СУБД PostgreSQL.

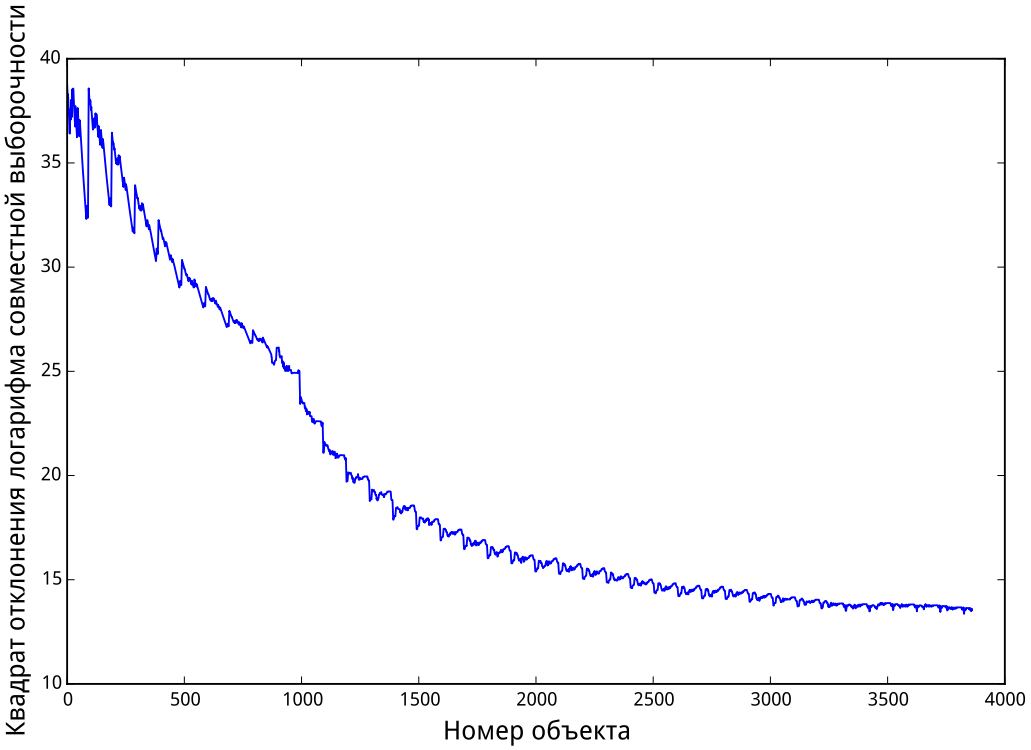


Рис. 15: Динамика точности предсказания выборочности взвешенной комбинацией стандартной модели и градиентного бустинга нал решающими деревьями на TPC–H в течении 50 эпох, усредненная с окном 1000

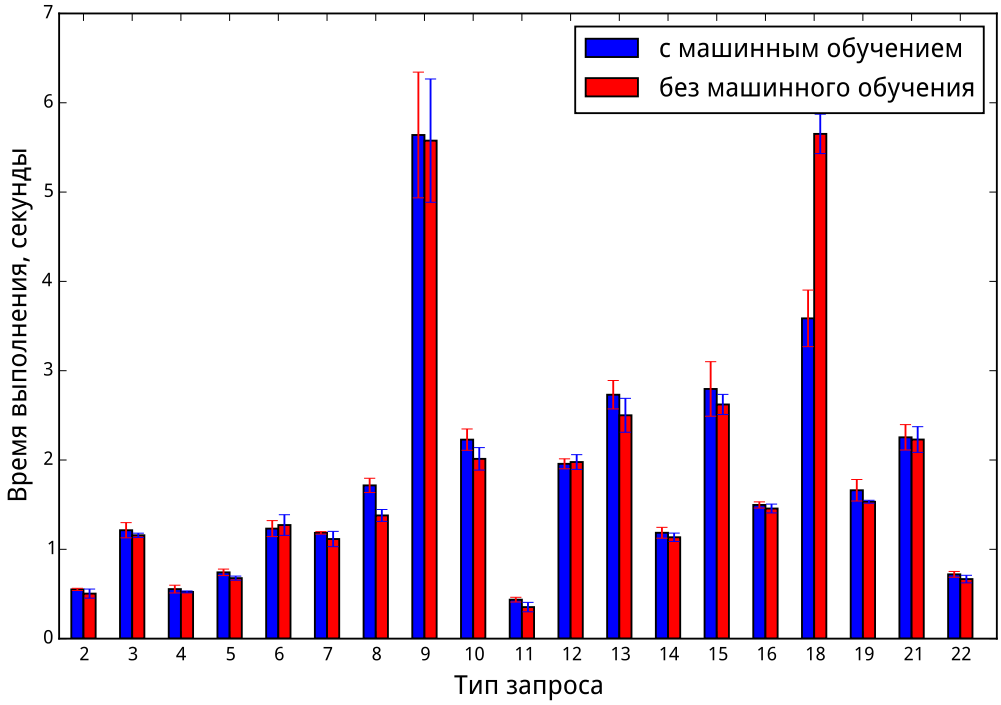


Рис. 16: Изменение времени выполнения запросов в результате использования предложенной модели через 5 эпох обучения

После трех таких итераций алгоритм сходится. На рис. 15 видно, что качество предсказания повышается со временем, а на рис. 16 видно, что все типы запросов начали выполняться чуть дольше, зато один из запросов стал выполняться в полтора раза быстрее. Небольшое замедление на большинстве типов запросов объясняется тем, что применение методов машинного обучения требует больше времени, чем простое перемножение чисел. Также существенный вклад в замедление вносит коммуникация со скриптом на python, в котором и происходит вычисление совместной выборочности.

Также в ходе экспериментов с промышленными данными были выявлены проблемы с тем, что оптимизатор запросов PostgreSQL может произвольно перемещать условия по подпланам и доверять предсказаниям регрессора по тем наборам условий, которых нет и никогда не будет в обучающей выборке. Например, на рис. 17 отображены два плана, которые соответствуют одному и тому же подзапросу. Для оценки мощности этого подплана оптимизатор может использовать предсказания для первого подплана, но запускать на выполнение второй.

Для этого плана делаются предсказания:

Этот план оказывается в обучающей выборке:

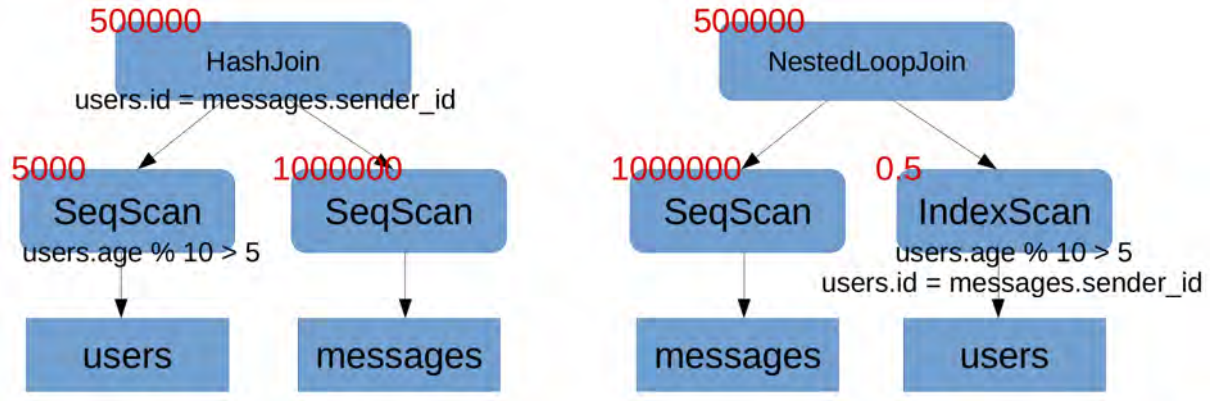


Рис. 17: Проблема неполноты обучающей выборки

3.9 Обсуждение и выводы

Три метода показали свою эффективность в задаче оценки совместной выборочности на модельных данных.

Полученные результаты объясняются тем, что характер зависимости совместной выборочности от маргинальных выборочностей определяется базой данных. Поскольку распределение данных в базе может быть произвольным, зависимость совместной выборочности от маргинальных выборочностей может быть сколь угодно сложной. Поэтому хорошие результаты показывают только методы, которые могут моделировать сколь угодно сложные зависимости, то есть непараметрические методы и нейронные сети.

Отметим, что каждый из трех лучших методов имеет свои достоинства и недостатки. Градиентный бустинг над решающими деревьями показывает наилучший результат, но требует много ресурсов для дообучения. Метод k ближайших соседей показывает наилучший результат в условиях малого количества информации и позволяет легко добавлять объекты в обучающую выборку, однако для быстрого поиска ближайших объектов по большой выборке требуются специальные алгоритмы и структуры данных. Наконец, нейронные сети являются параметрическим методом, поэтому можно использовать стохастические методы оптимизации и не хранить обучающую выборку.

Исследование на промышленных данных показало, что регрессор должен не только достаточно хорошо оценивать выборочность, но и слабо отклоняться от стандартной модели в тех случаях, когда недостаточно данных для того, чтобы сделать уверенный прогноз. Была предложена эвристика, с помощью которой можно создать регрессор, удовлетворяющий данному условию, на основе любого стандартного регрессора, причем при достаточно большом количестве данных предложенный регрессор переходит в стандартный.

Эксперимент на промышленных данных показал, что предложенный метод не только делает более точные предсказания совместной выборочности, но и тем самым увеличивает производительность СУБД.

Также эксперименты на промышленных данных показывают, что предложенный метод не всегда стабильно работает на внутренних вершинах плана. Первой причиной этого является недостаточная полнота обучающей выборки — некоторые сочетания условий никогда не выполнялись в СУБД, поэтому нам неизвестна их истинная совместная выборочность. Решать проблему можно с помощью принудительного запуска некоторых планов, хотя это и является достаточно ресурсоемким решением. Вторая причина этой нестабильности — особенности реализации оптимизатора PostgreSQL, а именно перемещение условий по подпланам. Эту проблему можно решить либо изменив оптимизатор PostgreSQL так, чтобы он больше соответствовал теоретическим моделям, либо, предпочтительнее, расширить формулировку задачи и учитывать не только условия в вершине, но также и всё поддерево этой вершины.

Таким образом, для задачи оценки совместной выборочности во внутренних вершинах плана предложенные методы существенно улучшают предсказание, но иногда могут работать не очень стабильно. Для листовых вершин такого эффекта нет, поэтому задачу оценки совместной выборочности в листовых вершинах плана можно считать полностью решенной в этой работе.

3.10 Темы для дальнейшей работы

- Разработка и исследование методов машинного обучения, учитывающие специфику задачи, то есть изменяемое признаковое пространство, необходимость дообучения, возможно, незавершенные планы.
- Разработка методов исследования признакового пространства с целью нахождения точки глобального оптимума или более хороших локальных оптимумов в процессе обучения.
- Построение более сложных композиций базовых алгоритмов для увеличения стабильности и скорости обучения метода во внутренних вершинах.
- Переход от предложенной формализации задачи к более сложным, где помимо условий в вершине учитываются условия в её поддереве.

4 Заключение

В данной работе была рассмотрена задача увеличения эффективности СУБД с применением методов машинного обучения. Было рассмотрено устройство стоимостного оптимизатора запросов в реляционных СУБД. В рамках данного исследования было экспериментально подтверждено, что слабым местом стоимостных оптимизаторов запросов является именно оценка совместной выборочности для зависимых условий. Далее была рассмотрена задача оценки совместной выборочности для зависимых условий. Было введено признаковое пространство для этой задачи, рассмотрены параметрические и непараметрические методы машинного обучения для её решения. По результатам экспериментов некоторые из предложенных методов показали свою эффективность. Были сделаны выводы об области применимости различных методов. С помощью предложенных методов удалось повысить качество предказания выборочности, в результате чего для некоторых запросов оптимизатор стал выбирать более эффективный план выполнения.

Список литературы

[1] Access path selection in a relational database management system / P. G. Selinger, M. M. Astrahan, D. D. Chamberlin et al. // Proceedings of the 1979 ACM SIGMOD international conference on Management of data / ACM. "— 1979. "— Pp. 23–34.

[2] *Altman N. S.* An introduction to kernel and nearest-neighbor nonparametric regression // *The American Statistician*. "— 1992. "— Vol. 46, no. 3. "— Pp. 175–185.

[3] *Bruno N., Chaudhuri S., Gravano L.* Stholes: A multidimensional workload-aware histogram. "— 2001.

[4] Cardinality estimation using neural networks / H. Liu, M. Xu, Z. Yu et al. // Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering. "— CASCON '15. "— Riverton, NJ, USA: IBM Corp., 2015. "— Pp. 53–59.

[5] *Chaudhuri S.* An overview of query optimization in relational systems // Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. "— PODS '98. "— New York, NY, USA: ACM, 1998. "— Pp. 34–43.

[6] *Freytag J. C.* A rule-based view of query optimization // Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data. "— SIGMOD '87. "— New York, NY, USA: ACM, 1987. "— Pp. 173–180.

[7] *Friedman J. H.* Greedy function approximation: a gradient boosting machine // *Annals of statistics*. "— 2001. "— Pp. 1189–1232.

[8] *Furtado P., Madeira H.* Summary grids: Building accurate multidimensional histograms. "— 1999.

[9] *Getoor L., Taskar B., Koller D.* Selectivity estimation using probabilistic models // *SIGMOD Rec.* "— 2001. "— May. "— Vol. 30, no. 2. "— Pp. 461–472.

[10] *Gunopulos D., Tsotras V. J., Domeniconi C.* Selectivity estimators for multidimensional range queries over real attributes // *The VLDB Journal*. "— 2005. "— Vol. 14. "— Pp. 137–154.

[11] *Halim F., Karras P., Yap R. H.* Fast and effective histogram construction // Proceedings of the 18th ACM Conference on Information and Knowledge Management. "— CIKM '09. "— New York, NY, USA: ACM, 2009. "— Pp. 1167–1176.

[12] *Hasan R., Gandon F.* A machine learning approach to sparql query performance prediction // Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01. "— WI-IAT '14. "— Washington, DC, USA: IEEE Computer Society, 2014. "— Pp. 266–273.

[13] *Heimel M., Markl V., Murthy K.* A bayesian approach to estimating the selectivity of conjunctive predicates. // BTW / Citeseer. "— 2009. "— Pp. 47–56.

[14] *Hoerl A. E., Kennard R. W.* Ridge regression: Biased estimation for nonorthogonal problems // *Technometrics*. "— 1970. "— Vol. 12, no. 1. "— Pp. 55–67.

[15] How good are query optimizers, really? / V. Leis, A. Gubichev, A. Mirchev et al. // *Proc. VLDB Endow.* "— 2015. "— Nov.. "— Vol. 9, no. 3. "— Pp. 204–215.

[16] Improved histograms for selectivity estimation of range predicates / V. Poosala, P. J. Haas, Y. E. Ioannidis, E. J. Shekita. "— 1996. "— Pp. 294–305.

[17] *Lakshmi M. S., Zhou S.* Selectivity estimation in extensible databases - a neural network approach // Proceedings of the 24rd International Conference on Very Large Data Bases. "— VLDB '98. "— San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. "— Pp. 623–627.

[18] Learning-based query performance modeling and prediction / M. Akdere, U. Çetintemel, M. Riondato et al. // Data Engineering (ICDE), 2012 IEEE 28th International Conference on. "— 2012. "— April. "— Pp. 390–401.

[19] *Malik T., Burns R. C., Chawla N. V.* A black-box approach to query cardinality estimation. // CIDR. "— 2007. "— Pp. 56–67.

[20] *Mitchell M.* An Introduction to Genetic Algorithms. "— Cambridge, MA, USA: MIT Press, 1998.

[21] *Poosala V.* Selectivity estimation without the attribute value independence assumption. "— 1997. "— Pp. 486–495.

[22] Predicting multiple metrics for queries: Better decisions enabled by machine learning / A. Ganapathi, H. Kuno, U. Dayal et al. // Proceedings of the 2009 IEEE International Conference on Data Engineering. "— ICDE '09. "— Washington, DC, USA: IEEE Computer Society, 2009. "— Pp. 592–603.

[23] Predicting query execution time: Are optimizer cost models really unusable? / W. Wu, Y. Chi, S. Zhu et al. // Data Engineering (ICDE), 2013 IEEE 29th International Conference on. "— 2013. "— April. "— Pp. 1081–1092.

[24] Robust estimation of resource consumption for sql queries using statistical techniques / J. Li, A. C. König, V. Narasayya, S. Chaudhuri // *Proc. VLDB Endow.* "— 2012. "— Jul.. "— Vol. 5, no. 11. "— Pp. 1555–1566.

[25] System r: Relational approach to database management / M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin et al. // *ACM Trans. Database Syst.* "— 1976. "— Jun.. "— Vol. 1, no. 2. "— Pp. 97–137.

[26] *Tieleman T., Hinton G.* Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude // *COURSERA: Neural Networks for Machine Learning.* "— 2012. "— Vol. 4. "— P. 2.

[27] Towards predicting query execution time for concurrent and dynamic database workloads / W. Wu, Y. Chi, H. Hacígümüş, J. F. Naughton // *Proc. VLDB Endow.* "— 2013. "— Aug.. "— Vol. 6, no. 10. "— Pp. 925–936.

[28] TPC BENCHMARK H (Decision Support) Standard Specification / Transaction Processing Performance Council. "— 2006.

[29] Uncertainty aware query execution time prediction / W. Wu, X. Wu, H. Hacigümüs, J. F. Naughton // *CoRR.* "— 2014. "— Vol. abs/1408.6589.

[30] *Wu W., Naughton J. F., Singh H.* Sampling-based query re-optimization // *CoRR.* "— 2016. "— Vol. abs/1601.05748.

[31] *Yu X., Koudas N., Zuzarte C.* Advances in Database Technology - EDBT 2006: 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006 / Ed. by Y. Ioannidis, M. H. Scholl, J. W. Schmidt et al. "— Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. "— Pp. 460–477.