Высшая школа экономики

Факультет компьютерных наук

Направление «Прикладная математика и информатика»

Курсовая работа

Обучение формуле Хорна с запросами к несовершенному оракулу.

Студент: А. А. Урусов
Преподаватель: С. А. Объедков

Москва, 2017

1. Постановка задачи.

Рассмотрим сначала оригинальную версию задачи. Пусть есть некоторый набор бинарных переменных переменных $x_1, \ldots, x_n$.

Определение 1. Условие Хорна - это дизъюнкция литералов (то есть переменных или их отрицаний), в которой не более одного раза встречается переменная без отрицания. Таким образом, оно принимает вид $x_{i_1} \wedge x_{i_2} \wedge \ldots x_{i_k} \to a$, где a - либо одна из переменных, либо константа True или False.

Определение 2. Формула Хорна - бинарное выражение вида $L_1 \wedge L_2 \wedge \cdots \wedge L_k$, где L_i - некоторое условие Хорна.

Пусть задана также некоторая формула Хорна $H*$ на этих переменных. Мы можем взаимодействовать с ней посредством оракула, умеющего отвечать на 2 типа запросов:

- Запрос принадлежности. По заданным значениям переменных определить значение $H*$ на этих значениях.
- Запрос эквивалентности. По заданной формуле Хорна H определить, является ли она эквивалентной $H*$. При этом если ответ отрицательный, то также возвращается контрпример x, который может быть положительным $(H(x) = 0, H*(x) = 1)$ или отрицательным $(H(x) = 1, H*(x) = 0)$.

Цель задачи - научиться вычислять $H*$.

Заметим, что если ответ на первый запрос достаточно очевиден (нужно просто подставить в $H*$ заданные значения переменных), то на запрос эквивалентности отвечать довольно сложно (хотя и возможно за полиномиальное время). Поэтому имеет смысл рассмотреть несовершенный оракул, то есть такой, который отвечает только на запросы первого типа.

2. Алгоритм решения задачи с совершенным оракулом.

Алгоритм также описан в [1] и [3].

Предположим сначала, оракул все-таки умеет отвечать на запросы второго типа. Рассмотрим следующий алгоритм нахождения множества импликаций.

Пусть изначально множество импликаций $\mathcal{L} = \emptyset$. Далее, если на очередном шаге наше множество импликаций эквивалентно множеству объектов, то можно просто вернуть его. Иначе, есть некоторый контрпример X. Далее возможны 2 случая:

- X не удовлетворяет $\mathcal{L}$. Тогда этот контрпример положительный. В этом случае для каждой импликации $(A \to B) \in \mathcal{L}$ если $A \subset X$, то заменим эту импликацию на $(A \to B \cap X)$. Таким образом, мы изменяем следствие каждой импликации так, чтобы контрпример ей удовлетворял.

- X удовлетворяет $\mathcal{L}$, то есть контрпример отрицательный. В этом случае мы находим такую импликацию $A \to B$, что $= X \cap A \neq A$, и при этом не входит в множества (последнее проверяется запросом к оракулу), и заменяем ее на $C \to B$. Если же такой импликации не нашлось, то просто добавим в множество импликацию $X \to M$. Таким образом, мы модифицируем множество $\mathcal{L}$ так, что ему не будет удовлетворять X.

Эту последовательность действий будем повторять до тех пор, пока не получим множество импликаций, эквивалентное заданному (это мы проверяем в начале каждого шага). В [3] доказано, что этот алгоритм не только завершается корректно, но и работает полиномиальное время, а именно, ассимптотика этого алгоритма $\tilde{O}(m^2n^2)$, при этом делая $O(mn)$ запросов эквивалентности и $O(m^2n)$ запросов принадлежности, где m - количество импликаций в $H*$, n - количество переменных.

3. Модификация алгоритма для несовершенного оракула.

Вернемся к несовершенному оракулу. Как было сказано выше, он отличается тем, что не допускает запросов на эквивалентность.

В [3] упоминается о невозможности обучения формуле Хорна за полиномиальное время в таких условиях. Однако, мы можем попробовать обучиться формуле Хорна неточно, но с относительно небольшой потерей качества. Для этого смоделируем запросы на эквивалентность следующим образом: на очередном шаге зададим количество итераций $iter$, зависящее от номера шага алгоритма, описанного в предыдущей секции, а также параметров алгоритма, и сгенерируем $iter$ случайных комбинаций значений переменных. Для каждой комбинации $x = (x_1, \ldots, x_n)$ проверим, является ли она контрпримером. Для этого нужно проверить, удовлетворяет ли x импликациям из H (для этого просто пройдем по этим импликациям) и импликациям из $H*$ (для этого достаточно выполнить запрос на принадлежность). Если за заданное количество итераций мы нашли контрпример, то возвращаем его, если не нашли, то считаем, что примера не существует, и заданное множество импликаций эквивалентно.

Осталось понять, как выбрать количество итераций. Зададим сначала 2 параметра $0 < \varepsilon, \delta < 1$. Тогда, согласно [2], если выбрать $iter = \left\lceil \frac{1}{\varepsilon}\left(i + \ln\frac{1}{\delta}\right)\right\rceil$, то имеют место

быть следующие гарантии на качество. Обозначим как Mod(H) множество комбинаций значений переменных, которые удовлетворяют импликациям H. Тогда алгоритм Англуин, описанный в предыдущей секции, с приведенной рандомизированной реализацией запроса эквивалентности гарантирует, что:

$$Pr\left(\frac{|Mod(H)\triangle Mod(H*)|}{2^n} \leq \varepsilon\right) \geq 1-\delta \tag{1}$$

4. Обучение импликациям на основе данных

1. Постановка задачи

В реальной жизни в качестве оракула обычно выступает человек или группа людей, обладающих экспертными знаниями в некоторой области. Однако, во многих случаях доступа к таким людям может не быть по ряду причин. Например, мы можем не обладать полными знаниями о некоторой области, или нахождение этих знаний является слишком долгим в связи с малым количеством экспертов. Однако, мы можем иметь некоторые данные о примерах объектов, о которых пытаемся что-то выучить. В связи с этим появляется задача обучения по выборке. Формально, пусть у нас есть набор объектов X, у каждого из которых есть некоторое количество бинарных признаков.

Определение 3. Замыкание X - это множество $a = x_1 \wedge x_2 \wedge \cdots \wedge x_k, x_i \in X$, то есть множество всевозможных пересечений объектов из X.

Тогда наша задача состоит в нахождении множества импликаций, которому удовлетворяют в точности все объекты из замыкания.

Чтобы применить уже известный алгоритм к такой постановке задачи, нужно описать, как выполняются запросы принадлежности (отметим, что запросы эквивалентности мы уже умеем моделировать через запросы принадлежности, соответственно, их не требуется моделировать снова).

По сути запрос принадлежности заключается в проверке того, что заданный объект x является пересечением некоторого подмножества X' имеющихся объектов. Заметим, что все элементы из X' должны иметь все признаки, которые есть у x. Тогда возьмем множество X'' всех объектов, содержащих все признаки x. Тогда пересечение всех элементов X'', очевидно, содержит все признаки из x. Кроме того, если существует множество X', пересечение элементов которого равно x, то пересечение элементов $X'' \supseteq X'$ тоже равно x. Значит, достаточно проверить, что пересечение всех множеств, содержащих x, равно x.

2. Оптимизации алгоритма

2.1. Ограничение по времени

Поскольку теоретическая оценка на качество довольно грубая, можно предположить, что на практике заданное качество достигается значительно быстрее, чем в момент сходимости. Тогда логично попробовать чем-то ограничить время рандомизированному алгоритму. Это можно сделать двумы способами. Во-первых, можно просто задать некоторый константный порог. Это имеет смысл, поскольку часто алгоритм работает достаточно медленно. Кроме того, для задачи построения импликаций по данным уже существуют точные алгоритмы, которые подробно описаны в [2]. Поэтому логично попробовать ограничить рандомизированный алгоритм некоторой долей времени работы какого-нибудь из них.

2.2. Использование данных для более точного корректирования импликаций

Заметим, что в случае, когда у нас есть данные, мы на самом деле можем использовать их для того, чтобы по данному объекту x понять, что же из него на самом деле следует. Для этого достаточно пересечь все элементы выборки, содержащие x. Будем в алгоритме всегда, когда мы изменяем или добавляем некоторую импликацию, в качестве правой части использовать реальное следствие левой. Заметим, что тогда по построению ни в какой момент не существует положительного контрпримера. Действительно, пусть есть некоторый y, который нарушает одну из наших импликаций $a \to b$ и при этом равен пересечению каких-то объектов из x. Очевидно, все эти объекты содержат a, но тогда они содержат и b, поскольку пересечение всех объектов, содержащих a, содержит и b, значит, пересечение их некоторого подмножества уж тем более содержит b. Значит, y удовлетворяет импликации $a \to b$, что противоречит предположению.

2.3. Использование замыкания

Напомним, что в рандомизированном оракуле мы генерируем случайные объекты, а затем проверяем, являются ли они контрпримерами. Заметим, что при условии использования оптимизации из предыдущего пункта мы уверены, что если контрпример есть, то он отрицательный. Значит, он точно должен удовлетворять нашим импликациям. Тем не менее, многие из сгенерированных объектов не будут удовлетворять этим свойством. Чтобы исправить это, можно замыкать каждый генерируемый объект относительно нашего текущего множества импликаций. Тогда полученный объект достаточно будет проверить на принадлежность соответствующим запросом к оракулу. Тем не менее, замыкание - это довольно сложная операция, поэтому сле-

дует ожидать ухудшения производительности (или качество - если мы ограничиваем время).

5. Тестирование

Для тестирования удобно использовать как раз постановку задачи на основе данных. При этом можно использовать как сгенерированные модельные примеры, так и реальные данные. Начнем с первых. Будем генерировать выборку случайно. При этом будем задавать количество признаков и объектов, а также плотность, то есть вероятность появления отдельно заданного признака у заданного объекта. Кроме того, мы можем задавать параметры нашего алгоритма, а именно, ε, δ, а также использовать различные варианты оптимизаций.

Теперь нужно ввести метрики качества. Во-первых, логично измерять качество работы алгоритма, поскольку он не является точным. В качестве метрики качества можно использовать ту величину, для которой мы гарантируем теоретическую оценку. Поскольку объектов экспоненциально много, честный расчет этой величины занимает очень много времени, поэтому будем использовать рандомизированную оценку этой величины: возьмем 10000 случайных объектов, проверим, сколько из них входят в числитель, и усредним размером выборки.

Теперь вернемся к тому, зачем мы вообще хотим обучаться импликациям. В большой степени это необходимо с целью имитации оператора замыкания. Это означает, что для произвольного набора признаков мы хотим понимать, какими еще признаками обладает объект, обладающий этими. Тогда естественной метрикой будет доля объектов, для которых мы правильно определяем замыкание. Ее тоже можно считать рандомизированно: возьмем выборку из 1000 случайных объектов и проверим, для скольких из них замыкание на основе наших импликаций совпадает с замыканием на основе реальных импликаций, и также усредним размером выборки.

Теперь разберемся с параметрами алгоритма. В первую очередь посмотрим на оптимизации. Есть 3 основных версии алгоритма:

1) Базовый алгоритм. Эта версия не использует никаких оптимизаций, кроме, возможно, ограничения по времени.

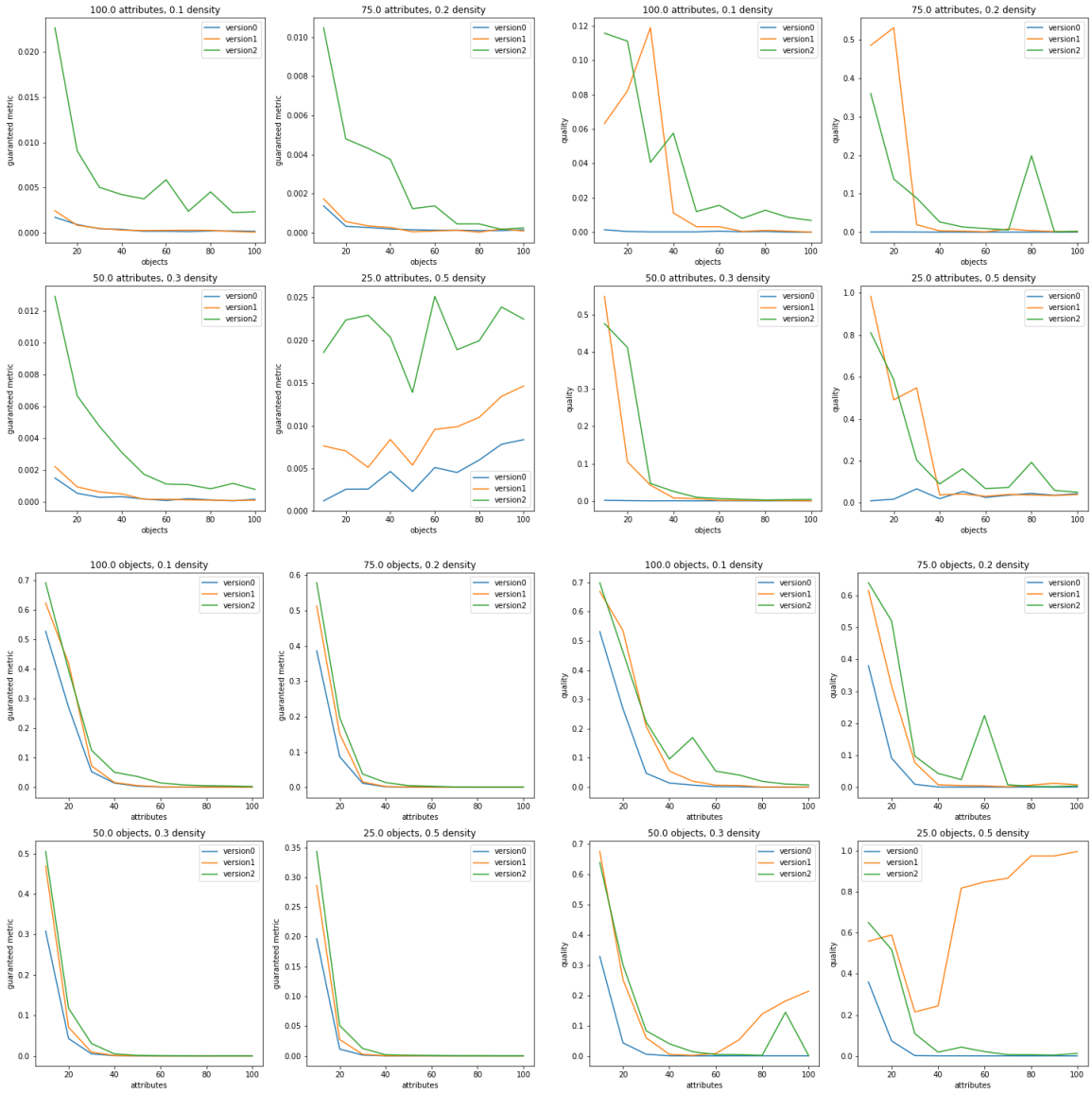
2) Улучшенный алгоритм. Эта версия использует оптимизацию 2 из предыдущей секции для задачи на основе данных.

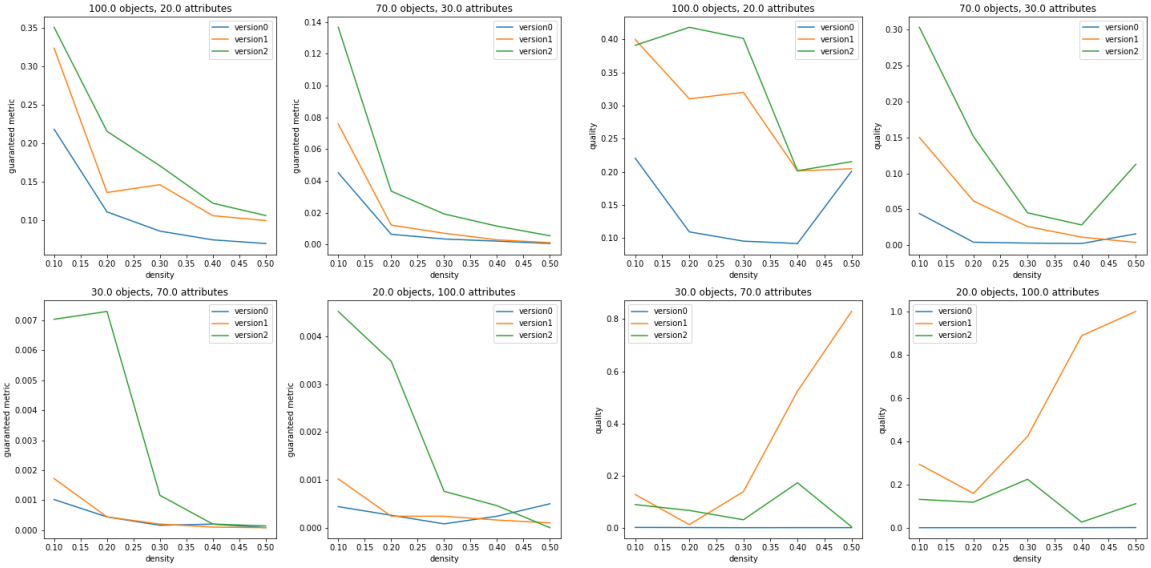
3) Улучшенный алгоритм с улучшенным оракулом. Эта версия использует обе оптимизации 2 и 3 из предыдущей секции. Отметим, что оптимизация 3 частично основана на гарантиях, предоставляемых оптимизацией 2, и, соответственно, имеет мало смысла без нее.

Кроме того, используем первую оптимизацию следующим образом: возьмем из [2] оптимизированный алгоритм Гантера с использованием линейного замыкания, и ограничим половиной времени его работы наш алгоритм.

Наконец, поскольку мы используем рандомизированный алгоритм, имеет смысл каждый эксперимент усреднить по нескольким запускам. В данном случае усреднение производится по 5 запускам.

Посмотрим на зависимости качества работы алгоритмов от параметров выборки.




100.0 objects, 20.0 attributes
70.0 objects, 30.0 attributes
100.0 objects, 20.0 attributes
70.0 objects, 30.0 attributes
30.0 objects, 70.0 attributes
20.0 objects, 100.0 attributes
30.0 objects, 70.0 attributes
20.0 objects, 100.0 attributes
version0
version1
version2
guaranteed metric
quality
density

Coming soon

6. Литература

[1] K. Bazhanov, S. Obiedkov Optimizations in computing the Duquenne–Guigues basis of implications

[2] S. Obiedkov Probabilistic Computation of the Canonical Basis with Membership Queries

[3] D. Angluin, M. Frazier, L. Pitt Learning Conjunctions of Horn Clauses

[4] H. Kautz, M. Kearns, B. Selman Horn approximations of empirical data