Собрание сочинений... гм, булевых

Данные

Имеется 400+ булевых входов x и 13 булевых выходов y. Также есть выборка из условно небольшого числа прецедентов, для которых булевы выходы известны.

В экспериментах:

В экспериментах используется кузнечик и ещё 30 подобных песенок. В каждой песенке по 120 прецедентов. Песенки можно транспонировать, расширяя таким образом выборку до 7920 прецедентов.

Особенность данных:

- Большинство значений всегда нули, т.е. входы можно интерпретировать не как признаки, а как ***сигналы***
- Т.е. в некотором роде можно считать, что отрицание от входной переменной практически бессмысленно
- Важность всех признаков примерна одинакова
- Входной вектор можно также интерпретировать как множество сигналов
- Всегда верно, что при отсутствии сигналов $y = 0$
- Причина того, что $y = 1$, всегда содержится в наборе переменных, принявших значение 1
- Выборка непротиворечива, т.е. векторы x в выборке не повторяются.

Задача

Требуется построить алгоритм, который бы *запоминал* выборку. Что это значит?

Это означает, что алгоритм на обучении должен выдать 100% точность. При этом хочется, чтобы у алгоритма была некоторая обобщающая способность, т.е. он должен в некотором смысле запомнить выборку именно построением правил, по которым y следует из x.

В идеале он должен уметь запоминать новую информацию. То есть чтобы ему можно было подать на вход новые данные, сказать «ещё бывает вот так», и алгоритм расширял свои знания.

Мотивация

Закономерности из реальной жизни мы берём из примеров.

Однако мы не запоминаем все сигналы, пришедшие в мозг, а обобщаем их каким-то простым предикатом. Возможно, упрощение (для хранения фактов в памяти) и обобщение в этом процессе взаимосвязанные вещи.

В результате, запоминание тесно связано с выявлением закономерностей.

Забывание означает, что процесс этот итеративный и субоптимальный. Однако, если один и тот же факт повторять много раз, то он «запомнится» лучше, значит при попытке запомнить прецедент мы строим эмпирическое правило, которое может оказаться неверным или грубым и поэтому «откинуться» в дальнейшем процессе обучения.

Переобучение? Пожалуйста!

Раз формально от нас требуется переобучиться, то давайте возьмём какой-нибудь известный своей склонностью к переобучению алгоритм.

- Деревья в каждом узле проверяют булев признак. Это означает, что будут узлы, соответствующие тому, что признак равен нулю. Это заведомо противоречит природе данных. Дообучение при этом возможно, но новая информация будет разбивать нижние узлы.
- 1NN будет выдавать прогноз по правилу «текущий набор сигналов больше всего похож на вот такой пример из истории, поэтому ответ такой». Такой алгоритм уже подразумевает, что вся выборка хранится в памяти, вместо того, чтобы запомнить правила, по которым была устроена связь между y и x, человек так не делает.

Формальная постановка

Формалисты очень хотели, чтобы задачу формализовали, поэтому есть такой вариант формальной постановки:

$$\sum_i (y_i - f(x_i, \theta)) = 0$$

$$Simplicity(\theta) \to \min_\theta$$

где $f(x, \theta)$ - семейство (или скорее даже класс) алгоритмов, параметризованных θ, а $Simplicity(\theta)$ - некая оценка сложности θ.

Например, если $f(x, \theta)$ - булевы функции, то $Simplicity(\theta)$ можно определить как количество операций в формуле.

Доступная формулировка

Требуется выдавать 100% на обучении, *минимально переобучившись*.

Как мерить степень переобучения?

Чтобы не получить в лоб Вапником-Червоненкисом, скажем, что задача очевидно слишком сложная (оптимизация внутри класса алгоритмов!), поэтому мы будем искать какие-нибудь субоптимальные процедуры.

Простой подход

Будем решать задачу независимо для каждого выхода.

Для одного выхода будем строить булеву формулу.

Будем итеративно двигаться по выборке.

Итерация

Если на очередном объекте текущая формула выдаёт верный ответ, ничего не меняем.

Иначе надо что-то изменить. Идея: давайте что-нибудь добавим. Но что?

Пусть ошибка произошла на множестве сигналов x_1, x_2, x_3, x_4 и требовалось выдать 1, когда формула выдавала ноль.

Мнение любителей булевой алгебры:

Давайте добавим ДНФ этого набора!

$$f(x) = f_{prev}(x) \vee x_1 x_2 x_3 x_4 \bar{x_5} \bar{x_6} \ldots \bar{x_d}$$

Очевидно, что это не комильфо.

Банальное приближение

Воспользовавшись особенностями данных, можно убрать из конъюнкции все отрицания от переменных:

$$f(x) = f_{prev}(x) \vee x_1 x_2 x_3 x_4$$

Уже лучше, но то ли это, что мы хотим?

Модельная задачка

Рассмотрим задачу: пусть $g(x_1, x_2, x_3, x_4)$ - булева функция от 4 переменных, про которую известно, что:

$$g(0,0,0,0) = 0$$

$$g(1,1,1,1) = 1$$

Задача: найти функцию g.

Решение

Решение $\widehat{g}(x_1, x_2, x_3, x_4) = x_1 x_2 x_3 x_4$ выглядит странно; хотя бы тем, что принимает 1 только на одном наборе из 16, хотя в выборке одна единица и один ноль.

А вот $\widehat{g}(x_1, x_2, x_3, x_4) = x_1$, казалось бы, вполне годится.

А что ещё годится?

Оставаясь в рамках «вычёркивания из ДНФ», можно взять конъюнкции двух и более сигналов. Это как раз промежуточные варианты между конъюнкцией из одной переменной и конъюнкцией всех переменных.

Теоретически, разумными решениями будут любые функции, принимающие значение 1 на половине наборов. $x_1 \oplus x_2 \oplus x_3$ например, подойдёт... И, кстати, почему нет, ну изрежет в шахматном порядке гиберкуб, вполне разумная зависимость-то?!?

SVM...

SVM, 1NN и прочее машинное это ваше обучение выдаст функцию голосования:

$$\widehat{g}(x_1, x_2, x_3, x_4) = \sum_{i=0}^{4} x_i > 2$$

или же

$$\widehat{g}(x_1, x_2, x_3, x_4) = \sum_{i=0}^{4} x_i \geq 2$$

(!) При этом значения функции на границе ($\sum_{i=0}^{4} x_i = 2$) остаётся предметом философских дискуссий.

Мораль

Преимущество функции голосования в том, что она симметрична относительно переменных. С другой стороны, непонятно, что на границе.

В любом случае, брался какой-то «предикат» пришедшего набора сигналов, т.е. какая-то функция, которая на этом наборе равна 1.

Пока что остановимся на варианте с элементарными конъюнкциями из пришедших сигналов. И заметим, что эл. конъюнкция из k переменных тем ближе к самому «обобщающему» варианту, чем меньше k.

Тупой алгоритм

Положим, что изначально функция равна тождественному 0.

$$f(x) = 0$$

Пусть на наборе сигналов $A = x_{i_1} \ldots x_{i_m}$ правильный ответ 1, а текущая функция выдаёт 0. Давайте возьмём в качестве предиката просто случайный сигнал x_A. Тогда:

$$f(x) = f_{prev}(x) \vee x_A$$

Если на A правильный ответ 0, а текущая функция выдаёт 1, то поступим аналогично:

$$f(x) = f_{prev}(x) \wedge \overline{x_A}$$

Аналогии?

Кто-то скажет, очень похоже на бустинг...

Скорее, на стохастический градиентный спуск. Берём очередной объект и как-то сдвигаемся в направлении правильного ответа.

Результаты

Конечно же, этот алгоритм будет новыми правилами ломать достижения старых.

За 350 проходов по кузнечику он так и не сошёлся к 0 ошибок, и воспроизвести его не смог.

Неудивительно, т.к. итоговая функция будет выглядеть как-то так:

$$f(x) = (\dots(\dots) \vee x_5 \dots) \wedge \overline{x_5} \dots$$

Эвристика

Давайте потребуем, чтобы **элементарные конъюнкции не повторялись**.

Для генерации предиката очередного множества сигналов мы сначала берём случайный сигнал. Если конъюнкция, состоящая только из одного этого сигнала, уже была, то из множества сэмплируется ещё сигналы и добавляются в текущую элементарную конъюнкцию, пока получившаяся конъюнкция не окажется уникальной.

Пример:

$$f(x) = 0 \vee x_4$$

На новом множестве x_1, x_4, x_7 выдано 1 вместо 0. Случайный сигнал - x_4. Такая конъюнкция уже есть. Поэтому сэмплируется ещё один сигнал, допустим x_1. Конъюнкции $x_1 x_4$ в правиле ещё нет, поэтому итог:

$$f(x) = (0 \vee x_4) \wedge \overline{x_1 x_4}$$

Результаты

Такой алгоритм сходится на кузнечике за 19 итераций.

Результат: 19 iterations learning on kuznechik of conj and disj with history for no repeats.mid

Внезапно

Утверждение. Пусть в выборке $\nexists A, B : A \in B$. Тогда этот алгоритм сходится.

Действительно, в какой-то момент алгоритму придётся использовать конъюнкции всех сигналов, чтобы не повторяться, и он выродится в построение ДНФ.

Свойства

- Если в выборке есть $A, B : A \in B$, то на этапе поиска уникальных конъюнкций такой может вообще не найтись, и тогда алгоритм вообще не знает, что делать. В кузнечике просто до этого «не дошло»
- А на 6 песенках со всеми транспонированными вариациями (всего получилось 18 песен, т.е. 120*18 объектов в выборке) - дошло, да и вообще сходится медленно.

Важность объектов?

Для первых объектов из выборки алгоритм будет брать самые «обобщаемые» правила, то есть конъюнкции из малого числа сигналов, а для последних - уже урезанные, из большого числа. То есть тут есть сильная зависимость от того, в каком порядке ведётся проход по выборке.

(Н.Юдин, 2017): брать на каждой итерации случайный объект из выборки, раз проходов потенциально много. До меня что-то не доходило, надо будет попробовать (TODO).

Попробуем ещё улучшить алгоритм, чтобы он сходился побыстрее.

Проблема перекрытия

Добавляя правило, исправляющее значение в очередной точке A, мы потенциально сбиваем значения в точках, для которых строились все предыдущие правила.

Можно ли добавлять новое правило для A не в конец?

Поиск виновного

Давайте при подстановке A в функцию будем смотреть, какие правила (или даже точнее – предикаты) для этого нового объекта включаются, а какие – нет.

Последнее включившееся на объекте правило назовём **ответственным**.

Пример

Например, для $f(x) = x_1 \wedge \overline{x_5} \vee x_1x_4 \vee x_3x_5$ на объекте $A = \{x_1, x_3, x_4\}$ будет правило $\vee x_1x_4$, т.к. на всех правилах правее предикаты не выполнились.

Заглушка

Утверждение. Самая «левая» позиция для нового правила, при которой значение в новой точке будет гарантированно исправлено, это позиция после ответственного.

Тип (дизъюнкция с предикатом или конъюнкция с отрицанием предиката) последнего включившегося правила определяет итоговое значение на объекте. Поэтому, вставка нового правила до ответственного не меняет значение функции в A, а правее — меняет.

Результат

Вставка правил в «оптимальное» место сократила обучение на кузнечике до 14 итераций. На 6 песенках всё ещё слишком долго.

Оценка правил

Возможность находить «ответственные» за итоговый ответ правила позволяет подсчитывать, насколько хорошо они работают.

Если значение функции на очередном объекте совпало с правильным, давайте поставим ответственному правилу +1, иначе -1.

Качество нового правила разумно инициализировать 1, т.к. оно точно является ответственным на том объекте, для которого создавалось.

Выдавать плюсы-минусы правилам, чьи предикаты сработали, но которые в итоге не были ответственны за итоговый ответ функции, оказывается нелогично:

- вся суть алгоритма в том, чтобы добавлять сильно обобщённые правила, которые будут ошибаться, но будут исправлены последующими «заглушками»
- оценка правила сведётся к статистике по выборке по предикату.

Что делать с оценкой?

Ненависть!

Давайте удалять правила, чья оценка стала слишком низкой (меньше некоторого порога *thr*) как неудачные. Поймать этот момент алгоритмически просто (в этот момент правило было ответственным и в очередной раз совершило ошибку).

Выбор порога?

Разумно, что $thr \leq 0$. Экспериментально выяснилось, что $thr = 0$ даёт самое большое ускорение сходимости.

Исходный код: Discon.ipynb

Используя обе эвристики, на кузнечике сошлось за 12 итераций, а на 6 песенках оно наконец смогло досчитаться.

Результат: 6 transposed songs learn-on-all stubs with history and hating.mid

Однако...

Хочется юзать все 30 песен, а это уже оказывается многовато :(
А больше ускорений сходимости в голову не лезет.

Мотивация

При добавлении нового правила хочется «затирать» как можно меньше значений функции в уже пройденных точках.

При добавлении нового правила мы можем запоминать, за какую точку оно «ответственно» (назовём эти точки **опорными**... где-то я уже это слышал).

$$0 \vee \underset{\substack{x_1\\x_2\\x_3}}{x_1} \wedge \overline{\underset{\substack{x_1\\x_5}}{x_5}} \vee \underset{\substack{x_1\\x_2\\x_4\\x_5}}{x_1 x_4} \vee \underset{\substack{x_3\\x_4\\x_5}}{x_3 x_5}$$

Простое соображение

При добавлении нового правила будем при генерации новой конъюнкции требовать не её новизны, а чтобы она не ломала все остальные правила.

Для этого можно перебирать все опорные точки всех правил и «расширять» конъюнкцию до тех пор, пока она не перестанет на них включаться. Звучит вычислительно сложно...

Не так уж и сложно

Все правила делятся на два типа - «включающие» и «выключающие». Первые в выходах функции меняет часть нулей на единицу, вторые наоборот. Поэтому нужно рассматривать только те опорные точки, на которых правильный ответ противоположен - перебор в два раза сокращается.

Если пользоваться эвристикой заглушки, и вставлять новое правило сразу за ответственным, то все правила правее гарантируют правильный ответ на своих опорных точках. Поэтому можно перебирать только опорные точки правил левее.

Даже алгоритмически за линию

Построение одного правила можно делать за один проход по всем необходимым опорным точкам. Встречая очередную опорную точку, нам нужно расширять конъюнкцию, пока она перестанет на ней выполняться, и при этом гарантированно дальнейшее расширение не изменит этого факта, поэтому возвращаться к этой опорной точке уже не нужно.

Да и вообще...

тут предыдущий алгоритм на 6 песенках еле сходился, а мы тут об алгоритмической сложности думаем...

Идея!

Алгоритм гарантирует, что новые правила не затрут значения в опорных точках всех булевых правил.

Значит, если все объекты выборки сделать опорными, алгоритм будет запоминать её за один заход.

Реализация!

Все объекты, на которых была совершена ошибка, становятся опорными по построению.

Для остальных объектов либо есть ответственное правило, которое и выдало верный ответ, либо ни одно правило не выполнилось. В последнем случае будем считать, что для них ответственным является правило 0, которым функция инициализировалась в начале алгоритма.

Припишем эти объекты ответственным правилам и будем, как и для всех опорных точек, требовать, чтобы новые правила не ломали в них значения.

Результаты

Внезапно это работает.

На 6 песнях:
Результаты: 6 transposed songs greedy conj-disj.mid

На 24 песнях:
Результаты: 24 transposed songs greedy conj-disj.mid

Проблема 0

Однако, иногда на новых данных алгоритм не играл ноты вообще, и похоже, связано это с тем, что в правиле 0 собиралось очень много точек из разных частей пространства, которые не давали последующим правилам обобщаться.

Решение:

Точкам, на которых ни одно правило не выполнилось, создавать заглушку даже несмотря на то, что ответ на них функция выдала верный.

Исходный код: GreedyDiscon.ipynb

Результаты:

- *31 transposed songs greedy-conj-disj tb no zero support points.mid*
- *31 transposed songs greedy-conj-disj tb no zero support points [C-F-E-F].mid*

Недостатки

- Всё ещё непонятно, что делать, если сигналы для конъюнкции закончились.
- Чем дальше объект в выборке, тем уже на нём будет соответствующее ему правило (эвристика заглушки сильно от этого не спасает).
- Правила становятся достаточно узкими (конъюнкция нескольких и более сигналов) достаточно быстро.
- В этом смысле выбранный класс предикатов беден. Они очень быстро вырождаются в длинные конъюнкции, которые редко выполняются.
- В построении правил (т.е. конъюнкций) есть стохастика, и неудачные выборы остаются в правиле навсегда (эвристика ненависти уже не работает, т.к. проход по выборке один).

Мимо пробегала...

доктор физико-математических наук, доцент каферы ММП факультета ВМиК МГУ с 1998 г., главный научный сотрудник Вычислительного Центра им. А.А. Дородницына РАН, профессор Математического факультета Московского Государственного Педагогического Университета (МПГУ) Е.В. Дюкова! А точнее, её приспешники...

И ими было замечено, что ну а вот вы знаете, есть такая штука, как логические корректоры...

И вообще, матрица сравнения K0 / K1, покрытия, пр.

Говорят, так можно получить набор сигналов, по которому классы 0 и 1 гарантированно можно различить. Правда, в предположении о свойствах данных это отберёт нам все сигналы...

ОКИ?

И вообще, все рассуждения выше гнусно попахивают основами кибернетики...

Однако, судя по тому единственному взгляду, который я бросил на список билетов, в день, кхм, зачёта, оки и прочая дискра занимается эквивалентными преобразованиями булевых формул над всякими разными базисами.

Впрочем, возможно это таки имеет отношение к каким-то потайным главам дискретной оптимизации и теории покрытий...

Вспомнив модельную задачку, попробуем взять другой вид предикатов, например:

$$\sum_i x_i \geq thr$$

Переобобщение

Чем ниже *thr*, тем «обобщённее» правило, причём для малых *thr* оно даже как-то... переобобщено. Возможно, если действовать по логике предыдущего алгоритма, то недообобщённость в конце скомпенсируется переобобщённостью в начале O_O

В терминах множеств...

В терминах множеств предикат можно запись так:

$$\mathbb{I}(X) = \left|A \cap X\right| \geq thr_A$$

При выборе нового правила для объекта A достаточно определиться со значением thr_A. Будем пытаться сделать его как можно меньше.

В терминах опорных точек...

Аналогично предыдущим рассуждениям, при минимизации thr_A мы хотим гарантировать, что правило не ломает значения во всех опорных точках.

Для этого мы, как и раньше, должны перебрать все опорные точки правил противоположного типа, располагающихся левее, и гарантировать, что на них предикат не выполнится.

Красивая формула

То есть:

$$\left|A \cap SupPoint\right| < thr_A \qquad \forall SupPoint$$

Отсюда красивая формула:

$$thr_A = \max_{SupPoint} \left|A \cap SupPoint\right| + 1$$

Интерпретация

Предикат условно создаёт некий круг вокруг очередной точки. Чем меньше *thr*, тем больше этот круг. Круг закрашивается в 0 или 1 и кладётся сверху на то, что уже есть.

При этом он не должен перекрыть опорные точки, значения в которых уже зафиксированы, отсюда минимальное *thr*

Так в каком порядке добавлять круги?

Аналитическая задача?

Вот в каком порядке нужно подавать выборку на вход, чтобы, скажем, сумма *thr* была бы минимальной?

Если грубо предположить, что новое правило добавляется на каждом шаге и максимум берётся по всем предыдущим точкам из выборки, то можно сделать следующее: взять матрицу $\{|A_i \cap A_j|\}$ и пытаться переставлять одновременно строки и столбцы так, чтобы сумма чисел в нижнем треугольнике была минимальной.

Чёт не очень решаемая задача...

Исходный код: Greedy linear.ipynb

В текущей версии это всё как-то работает.

Результаты: 31 transposed songs greedy-linear tb.mid

Недостатки

- Проблема, что правило вырождается (превращается в пустой предикат), всё ещё встречается, хотя уже сильно реже.
- При фиксированной выборке можно отсортировать её по количеству сигналов в объектах; в этом случае гарантированно этой проблемы не будет. Однако, мы хотим уметь давать на вход новую информацию.

Обсуждение

Основная беда — всё ещё зависимость от порядка подачи объектов на вход. Первые объекты сильно важнее последних.

Здесь тоже можно попробовать с этим бороться методом Юдина (пройдясь по выборке в разных направлениях или случайно, и потом как-то заансамблировав результаты!). И вот интересный вопрос: а часом не получится ли «в среднем» 1NN?

Обман?

Получается, мы немного сами себя обманули, разрешив алгоритму иметь память, запоминая неограниченное число опорных точек. Теоретически, если для каждого объекта в выборке есть своё правило, то в булевой формуле будет храниться вся выборка.

С другой стороны, построенная формула - это набор правил и исключений из них, и в отличии от 1NN она более интерпретируемая.

Опять они!..

Утверждение. Полученный алгоритм - нейросеть.

Все веса равны 1. Каждое правило - нейрон с соответствующим порогом и связями с теми сигналами, которые есть в соответствующей опорной точке. Дизъюнкция, конъюнкция, отрицание и суперпозиция также выражается в нейросетевых терминах. Функция активации - функция Хевисайда (ну порог, короче).

Next: *так возможно, надо строить именно нейросеть?*