Алгоритмы порождения существенно нелинейных моделей

Г. И. РУДОЙ

Аннотация. В работе исследуются алгоритмы порождения и отбора существенно нелинейных моделей (символьная регрессия). Полученные модели применяются для восстановления регрессионной зависимости переменной от данных числовых рядов. Описывается представление моделей в виде матриц смежности. В вычислительном эксперименте приводятся результаты для задачи моделирования волатильности опционов.

1. Введение

В ряде приложений возникает задача восстановления регрессии по набору измеренных данных с условием возможности проинтерпетировать полученные данные экспертом, например <список ссылок на работы по конкретным приложениям>.

Одним из методов, позволяющих получать интерпретируемые модели, является символьная регрессия — процесс, в котором измеренные данные приближаются некоторой математической формулой, например $\sin x^2 + 2x$ или $\log x - \frac{e^x}{x}$. Одна из возможных реализаций этого процесса предложена John Koza [1] [2], использовавшим эволюционные алгоритмы для реализации символьной регрессии. Ivan Zelinka предложил дальнейшее развитие этой идеи [3], получившее название Analytic Programming.

В подобных случаях алгоритм построения требуемой математической формулы выглядит следующим образом: дан набор примитивных функций, из которых можно строить различные формулы (например, степенная функция, +, sin, tan). Начальный набор формул строится либо произвольным образом, либо на базе некоторых предположений эксперта. Затем на каждом шаге производится оценка каждой из формул (например, считается функционал SSE). На базе этой оценки у некоторой части формул случайным образом заменяется одна элементарная функция на другую (например, sin на cos или + на ×), а у некоторой другой части происходит взаимный попарный обмен подвыражениями в формулах.

Среди возможных путей улучшения качества символьной регрессии — анализ информативности различных признаков. Например, в ходе работы эволюционного алгоритма можно выявлять, какие из параметров слабо влияют на качество получающейся формулы, и либо убирать их совсем, либо обеспечивать неслучайность замены элементарных функций или обмена поддеревьев с целью замены этих параметров на другие в предположении, что они, возможно, окажутся более информативными.

Научный руководитель В. В. Стрижов.

Другим вопросом, возникающим при применении подобных эволюционных алгоритмов, является их принципиальная теоретическая корректность: способен ли вообще такой алгоритм породить искомую формулу.

2. Постановка задачи

2.1. **Теоретическая часть.** Пусть дано множество примитивных функций $G = g_1, \ldots, g_{n_g}$. Требуется:

- Построить алгоритм, за конечное время порождающий любую конечную функцию, являющуюся суперпозицией данных примитивных функций.
- Указать способ проверки изоморфности двух функций.

2.2. **Алгоритмическая часть.** Пусть дан набор $(x_i, y_i) \mid i \in 1, \ldots, N, x_i \in R^n, y_i \in R$. Требуется построить аналитическую функцию $f : R^n \to R$ из заданного множества элементарных функций G и доставляющую минимум некоторому функционалу ошибки.

3. Пути решения задачи: теоретическая часть

3.1. **Алгоритм порождения суперпозиций.** Итак, пусть дано множество примитивных функций $G = g_1, \ldots, g_{n_g}$ и множество свободных переменных $X = x_1, \ldots, x_{n_x}$. Сначала опишем итеративный алгоритм, позволяющий за конечное число итераций построить суперпозицию произвольной наперед заданной длины. Для удобства будем исходить из предположения, что множество G состоит только из унарных и бинарных функций, и разделим его соответствующим образом на два подмножества: $G = G_b \cup G_u \mid G_b = g_{b_1}, \ldots, g_{b_k}, G_u = g_{u_1}, \ldots, g_{u_l}$, где G_b — множество всех бинарных функций, а G_u — множество всех унарных функций из G. Потребуем также наличия id в G_b.

Алгоритм 1. *Алгоритм итеративного порождения суперпозиций.*

(1) Инициализируем множество $\mathcal{F}_0 = X$.

(2) Для множества $\mathcal{F}_i$ построим вспомогательное множество U_i, состоящее из результатов применения функций из G_u к элементам $\mathcal{F}_i$:

$$U_i = g_u \circ f \mid g_u \in G_u, f \in \mathcal{F}_i$$

(3) Аналогичным образом построим вспомогательное множество B_i для бинарных функций:

$$B_i = g_b \circ (f, h) \mid g_b \in G_b, f, h \in \mathcal{F}A_i$$

(4) Обозначим $\mathcal{F}_{i+1} = \mathcal{F}_i \cup U_i \cup B_i$ и перейдем к следующей итерации.

Тогда $\mathcal{F} = \cup_0^\infty \mathcal{F}_i$ — множество всех возможных суперпозиций конечной длины, построимых из данного множества примитивных функций.

Заметим, что алгоритм очевидным образом обобщается на множество G, содержащее функции произвольной (но имеющей конечный верхний предел) арности. Для такого обобщения достаточно строить аналогичным образом вспомогательные множества для этих функций.

Алгоритм корректен: любую конечную суперпозицию он действительно породит за конечное же число шагов. Чтобы убедиться в этом, достаточно представить суперпозицию в виде соответствующего графа и рекурсивно пройти вершины к листьям, составляя цепочку соотношений по следующим правилам:

- Если вершина, полученная на i-ом шаге — унарная функция, то это функция от выражения, полученного на $(i-1)$-ом шаге.
- Если вершина, полученная на i-ом шаге — бинарная функция, то это функция от двух выражений, как минимум одно из которых получено на $(i-1)$-ом шаге, а другое — на $(i-1)$-ом или ранее.
- Если это узел со свободной переменной, то он получен на нулевом шаге.

Разворачивая эту цепочку в обратную сторону, можно для каждой суперпозиции получить номер шага алгоритма, на котором она будет получена. Иными словами, для любой суперпозиции мы можем указать конкретный номер итерации, на котором она будет получена, что и требовалось.

Заметим, что алгоритм в таком виде не позволяет получать выражения для численных коэффициентов. Покажем, однако, на примере конструирования множеств U_i и B_i, как исходный алгоритм может быть расширен с учетом таких коэффициентов:

$$U_i = g_u \circ (\alpha f + \beta)$$

$$B_i = g_b \circ (\alpha f + \beta, \psi h + \phi)$$

Иными словами, мы неявно предполагаем, что каждая суперпозиция из предыдущих итераций входит в следующую, будучи умноженной на некоторой коэффициент и с линейной поправкой.

Очевидно, при таком добавлении коэффициентов $\alpha, \beta, \psi, \phi$, зависящих от конкретной комбинации g_u, f или g_b, f, h соответственно, мы не изменяем мощности получившегося множества суперпозиций, поэтому алгоритм и выводы из него остаются корректны.

Очевидно так же, что исходный алгоритм является частным случаем данного при $\alpha = \psi = 1, \beta = \phi = 0$.

В практических приложениях можно поступать аналогичным образом, например, подбирая конкретные значения коэффициентов у получившихся суперпозиций алгоритмом Левенберга-Марквардта.

Заметим так же, что такая модификация алгоритма позволяет нам получить единицу, например, для построения суперпозиций типа $\frac{1}{x}$: $1 = \alpha\ id\ x + \beta \mid \alpha = 0, \beta = 1$.

Отдельно подчеркнем, что численные коэффициенты у различных суперпозиций независимы. Однако, так как на разных итерациях алгоритма мы можем получить, вообще говоря, одну и ту же суперпозицию с точностью до этих коэффициентов, их необходимо не учитывать при тестировании различных суперпозиций на равенство.

Иными словами, коэффициенты зависят *только* от соответствующих функций, участвующих в суперпозиции, но не от номера итерации, на которой эта суперпозиция была получена.

Кроме того, опять же, заметим, что и этот алгоритм очевидным образом обобщается на случай множества G, содержащего функции произвольной арности.

3.2. **Бесконечные суперпозиции.** В предложенных ранее методах [3] построения суперпозиций необходимо было самостоятельно следить за тем, чтобы в ходе работы алгоритма не возникало «зацикленных» суперпозиций типа $f(x,y) = g(f(x,y),x,y)$. Заметим, что в предложенном алгоритме такие суперпозиции не могут возникнуть по построению.

3.3. **Множество допустимых суперпозиций.** Предложенный выше алгоритм позволяет получить действительно все возможные суперпозиции, однако, не все они будут пригодны в практических приложениях: например, $\ln x$ имеет смысл только при $x > 0$, а $\frac{x}{0}$ не имеет смысла вообще никогда. Выражения типа $\frac{x}{\sin x}$ имеют смысл только при $x \neq \pi k$.

Таким образом, необходимо введение понятия множества *допустимых* суперпозиций — то есть, таких суперпозиций, которые в условиях некоторой задачи корректны.

Одним из способов построения только допустимых суперпозиций является модификация предложенного алгоритма таким образом, чтобы отслеживать совместность областей определения и областей значения соответствующих функций в ходе построения суперпозиций. Для свободных переменных это будет, в свою очередь, означать необходимость задания областей значений пользователем при решении конкретных задач.

Заметим, что, хотя теоретически возможно выводить допустимость выражений вида $\frac{x}{\sin x}$ исходя из заданных условий на свободную переменную (например, что $x \in (\frac{\pi}{4}, \frac{\pi}{2})$), в общем случае это потребует решения неравенств в общем виде, что вычислительно неэффективно [1].

3.4. **Множество «минимальных» суперпозиций.** В ходе работы алгоритма могут возникать суперпозиции вида $x + x$ и $2x$, и хотя эти выражения эквивалентны, они представляются различными формулами. Аналогично $x + y$ и $y + x$, отличающиеся порядком следования слагаемых. Таким образом, необходим способ нормализации суперпозиций.

Во-первых, необходимо обеспечивать одинаковый порядок следования операндов, например, упорядочивая их каким-либо образом у коммутирующих бинарных функций.

Во-вторых, необходимо иметь набор правил, позволяющих проверить равенство $x + x$ и $2x$. Иными словами, необходимо иметь набор связей между различными функциями из множества данных примитивных функций. Заметим, что в общем случае эта задача требует введения значительного числа правил и по определению сводится к последовательному переборному их применению к различным подвыражениям суперпозиции.

В связи с этим может оказаться более эффективным иной подход к сравнению суперпозиций: так как по условию практической задачи значения искомой функции

[1] А было бы вычислительно эффективно — все равно, похоже, было бы NP-сложной задачей

даны в конечном числе точек, то для проверки на равенство достаточно вычислить получившиеся суперпозиции в этих точках и сравнить их.[2]

Другим способом, позволяющим избежать разрастания количества правил, может являться использование только «независимых» функций. Например, sin и cos связаны известным тригонометрическим соотношением с точностью до знака, а значит, sin и $\tan = \frac{\sin}{\cos}$ также связаны, как и ряд прочих тригонометрических функций, поэтому предлагается среди примитивных функций оставить лишь sin и стандартные арифметические действия для вывода прочих тригонометрических функций через соответствующие соотношения.

Однако, можно заметить два часто встречающихся шаблона правил, связывающих различные функции:

- Для унарных функций это $f \circ g = h$ (например, $\ln \circ \exp = id$).
- Для бинарных функций это $f(x, g(x, i)) = g(x, s(i))$. Например, $x+xi = x(i+1)$: здесь $f = (+), g = (\times), s(i) = i + 1$.

4. Пути решения задачи: практическая часть

Несмотря на то, что указанный ранее итеративный алгоритм порождения суперпозиций позволяет получить, в принципе, произвольную суперпозицию, для практических применений он непригоден, как и любой алгоритм, реализующий полный перебор, в связи с чрезмерной вычислительной сложностью. Вместо него можно использовать стохастические алгоритмы и ряд эвристик, позволяющих на практике получать за приемлемое время результаты, удовлетворяющие заранее заданным условиям «достаточной пригодности».

В данной работе предлагается следующий алгоритм:

Алгоритм 2. *Алгоритм стохастического порождения суперпозиций.*

(1) Инициализируется начальное множество суперпозиций случайным образом.

Список литературы

[1] John R. Koza. Genetic programming. In James G. Williams and Allen Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29–43. Marcel-Dekker, 1998. Supplement 24.

[2] John R. Koza. Introduction to genetic algorithms, August 15 1998.

[3] Ivan Zelinka, Zuzana Oplatkova, and Lars Nolle. I. ZELINKA et al: ANALYTICAL PROGRAMMING ... ANALYTIC PROGRAMMING – SYMBOLIC REGRESSION BY MEANS OF ARBITRARY EVOLUTIONARY ALGORITHMS, August 14 2008.

Московский физико-технический институт, ФУПМ, каф. «Интеллектуальные системы»

[2] Кстати, может, можно придумать какой-нибудь оптимальный алгоритм поиска расходящихся точек? Ну или эвристику хотя бы, позволяющую перебирать данные точки не в лоб, а более целенаправленно и позволяя находить точки, в которых значения различаются, более быстро.