АЛГОРИТМЫ ПОРОЖДЕНИЯ ДОПУСТИМЫХ СУПЕРПОЗИЦИЙ СУЩЕСТВЕННО НЕЛИНЕЙНЫХ РЕГРЕССИОННЫХ МОДЕЛЕЙ [1]

В. В. Стрижов [2], Г. И. Рудой [3]

Аннотация

При восстановлении нелинейной регрессии предлагается рассмотреть набор индуктивно порожденных моделей с целью выбора оптимальной модели. В работе исследуются индуктивные алгоритмы порождения допустимых существенно нелинейных моделей. Предлагается алгоритм, порождающий все возможные суперпозиции заданной сложности за конечное число шагов, и приводится его теоретическое обоснование. В вычислительном эксперименте приводятся результаты для задачи моделирования волатильности опционов.

Ключевые слова: *Символьная регрессия, нелинейные модели, индуктивное порождение, волатильность опционов.*

1 Введение

В ряде приложений [1–3] возникает задача восстановления регрессии по набору измеренных данных. При этом предполагается, что модель должна иметь возможность быть проинтерпретированной экспертом в контексте предметной области.

Одним из методов, позволяющих получать интерпретируемые модели, является символьная регрессия [4–6], согласно которой измеряемые данные приближаются некоторой математической формулой, например, $\sin x^2+2x$ или $\log x-\frac{e^x}{x}$. При решении регрессионной задачи данные приближаются различными формулами, являющимися произвольными суперпозициями функций из некоторого заданного набора. Одна из возможных реализаций этого метода предложена Джоном Коза [7, 8], использовавшим эволюционные алгоритмы для реализации символьной регрессии. Иван Зелинка предложил дальнейшее развитие этой идеи [9], получившее название аналитического программирования.

Алгоритм построения требуемой математической модели 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 согласно функции ошибки либо другого функционала [10] качества. 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.

[1] Работа выполнена при поддержке РФФИ, грант №10-07-00422.

[2] Вычислительный центр РАН, strijov@ccas.ru

[3] Московский физико-технический институт, 0xd34df00d@gmail.com

Получаемая формула является математической моделью [11] исследуемого процесса или явления — то есть, это математическое отношение, описывающее основные закономерности, присущие этому явлению.

Целью данной работы является теоретическое обоснование алгоритмов индуктивного порождения моделей и анализ этих алгоритмов. Другим вопросом, возникающим при применении подобных эволюционных алгоритмов, является их принципиальная теоретическая корректность: способен ли вообще такой алгоритм породить искомую формулу.

Алгоритм индуктивного порождения моделей, предложенный в настоящей работе, решает некоторые типичные проблемы предложенных ранее методов, упомянутые, например, в [9], а именно:

- Порождение рекурсивных суперпозиций, суперпозиций, содержащих несоответствующее используемым функциям число аргументов, и т. д. — в предложенном алгоритме эти проблемы не возникают по построению.
- Несовпадение области определения некоторой примитивной функции и области значений ее аргументов (возможно, тоже некоторых суперпозиций).
- При ограничении числа примитивных функций, участвующих в суперпозиции, а также при соответствующем задании множества примитивных функций исключается проблема слишком сложных суперпозиций.

Во второй части данной работы формально поставлена задача построения алгоритма индуктивного порождения моделей. Затем, в третьей части строится искомый алгоритм для частного случая непараметризованных моделей и доказывается его корректность, а затем алгоритм обобщается на случай моделей, имеющих параметры. В четвертой части описываются вспомогательные технические приемы, использованные в практическом алгоритме порождения моделей, описанном в пятой части. Результаты вычислительного эксперимента приведены в шестой части настоящей работы.

2 Постановка задачи

Пусть дана регрессионная выборка:

$$D = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \ldots, N\}, \mathbf{x}_i \in \mathbb{X} \subset \mathbb{R}^n, y_i \in \mathbb{Y} \subset \mathbb{R}\},$$

где N — объем регресионной выборки (число объектов), $\mathbf{x}_i$ — вектор значений независимых переменных i-ого объекта, y_i — значение зависимой переменной у i-ого объекта, $\mathbb{X}$ — множество значений независимых переменных, лежащее в $\mathbb{R}^n$, $\mathbb{Y}$ — множество значений зависимой переменной.

Требуется выбрать параметрическую функцию $f : \Omega \times \mathbb{R}^n \to \mathbb{R}$ из порождаемого множества $\mathcal{F} = \{f_r\}$, где Ω — пространство параметров, доставляющую минимум некоторому функционалу ошибки, определяемому ниже.

То есть, если множество всех суперпозиций:

$$\mathcal{F} = \{f_r \mid f_r : (\boldsymbol{\omega}, \mathbf{x}) \mapsto y \in \mathbb{Y}, r \in \mathbb{N}\},$$

то требуется найти такой индекс $\hat{r}$, что функция f_r среди всех $f \in \mathcal{F}$ доставляет минимум функционалу качества S при данной регрессионной выборке D:

$$\hat{r} = \arg\min_{r \in \mathbb{N}} S(f_r \mid \hat{\boldsymbol{\omega}}_{\boldsymbol{r}}, D), \tag{1}$$

где $\hat{\boldsymbol{\omega}}_r$ — оптимальный вектор параметров функции f_r для каждой $f \in \mathcal{F}$ при данной регрессионной выборке D:

$$\hat{\boldsymbol{\omega}}_{\boldsymbol{r}} = \arg\min_{\boldsymbol{\omega} \in \Omega} S(\boldsymbol{\omega} \mid f_r, D). \tag{2}$$

В качестве функционала качества S используется SSE:

$$S(\boldsymbol{\omega}, f, D) = \sum_{i=1}^{N} (y_i - f(\boldsymbol{\omega}, \mathbf{x}_i))^2 \mid (\mathbf{x}_i, y_i) \in D. \tag{3}$$

Сформулируем также постановку теоретической задачи. Для этого сначала введем понятие суперпозиции функций.

Если множество значений $\mathbb{Y}_i$ функции f_i содержится во множестве определения $\mathbb{X}_{i+1}$ функции f_{i+1}, то есть

$$f_i : \mathbb{X}_i \to \mathbb{Y}_i \subset \mathbb{X}_{i+1}, i = 1, 2, \ldots, \theta - 1,$$

то функция

$$f_\theta \circ f_{\theta-1} \circ \cdots \circ f_1, \theta \geq 2,$$

определяемая равенством

$$(f_\theta \circ f_{\theta-1} \circ \cdots \circ f_1)(\mathbf{x}) = f_\theta(f_{\theta-1}(\ldots(f_1(\mathbf{x})))), x \in \mathbb{X}_1,$$

называется *сложной функцией* [12] или *суперпозицией функций* $f_1, f_2, \ldots, f_\theta$.

Таким образом, получаем

Определение 1. *Суперпозиция функций — функция, представленная как композиция нескольких функций.*

Пусть $G = \{g_1, \ldots, g_l\}$ — множество данных порождающих функций, а именно, для каждой $g_i \in G$ заданы:

- сама функция g_i (например, sin, cos, $\times$),
- арность функции и порядок следования аргументов,
- домен ($\mathrm{dom} g_i$) и кодомен ($\mathrm{cod} g_i$) функции,

- область определения $\mathcal{D}g_i \subset \mathrm{dom}g_i$) и область значений $\mathcal{E}g_i \subset \mathrm{cod}g_i$.

Требуется построить упомянуютую функцию f как суперпозицию порождающих функций из заданного множества G.

Поясним различие между последними двумя пунктами. Например, $\mathrm{dom}f$ показывает, значения из какого множества принимает функция f (целые числа, действительные числа, декартово произведение целых чисел и $\{0,1\}$, и т. п.). Область определения же показывает, на каких значениях из $\mathrm{dom}f$ функция f определена и имеет смысл. Так, для функции $f(x_1, x_2) = \log_{x_1} x_2$:

$$\mathrm{dom}f = \mathbb{R} \times \mathbb{R},$$

$$\mathrm{cod}f = \mathbb{R},$$

$$\mathcal{D}f = \{(x_1, x_2) \mid x_1 \in (0;1) \cup (1;+\infty), x_2 \in (0;+\infty)\},$$

$$\mathcal{E}f = (-\infty;+\infty).$$

Требуется также:

- построить алгоритм $\mathfrak{A}$, за конечное число итераций порождающий любую конечную суперпозицию данных примитивных функций,
- указать способ проверки изоморфности двух суперпозиций.

Заметим, что мы не требуем для примитивных функций свойства их непорождаемости в наиболее общей формулировке типа принципиальной невозможности породить в ходе работы искомого алгоритма суперпозицию, изоморфную некоторой функции из G. Такое требование является слишком ограничивающим. В частности, невозможно было бы иметь в G одновременно, например, функции id, exp и log, так как $\mathrm{id} \equiv \log \circ \exp$.

В дальнейшем будем также считать, что суперпозиция, соответствующая единственной свободной переменной ($f(\mathbf{x}) = x_i$), полностью эквивалентна функции вида $\mathrm{id}x_i$.

3 Алгоритм индуктивного порождения допустимых суперпозиций

Условимся считать, что каждой суперпозиции f сопоставлено дерево Γ_f, эквивалентное этой суперпозиции и строящееся следующим образом:

- В вершинах V_i дерева Γ_f находятся соответствующие порождающие функции $g_s, s = s(i)$.
- Число дочерних вершин у некоторой вершины V_i равно арности соответствующей функции g_s.

- Порядок смежных некоторой вершине V_i вершин соотвествует порядку аргументов соответствующей функции $g_{s(i)}$.
- В листьях дерева Γ_f находятся свободные переменные x_i либо числовые параметры ω_i.
- Порядок вершин V_i в смысле уровня вершин определяет порядок вычисления примитивных функций: дерево вычисляется снизу вверх. То есть, сначала подставляются конкретные значения свободных переменных, затем вычисляются значения в вершинах, все дочерние вершины которых — свободные переменные, и так далее до тех пор, пока не останется единственная вершина, бывшая корнем дерева, содержащая результат выражения.

Таким образом, вычисление значения выражения f в некоторой точке с данным вектором параметров ω эквивалентно подстановке соответствующих значений свободных переменных x_i и параметров ω_i в дерево Γ_f выражения.

Заметим важное свойство таких деревьев: каждое поддерево Γ_f^i дерева Γ_f, соответствующее вершине V_i, также соответствует некоторой суперпозиции, являющейся составляющей исходной суперпозиции f.

Для примера рассмотрим дерево, соответствующиее суперпозиции $f = \sin(\ln x_1) + \frac{x_2^3}{2}$ (см. рис 1).

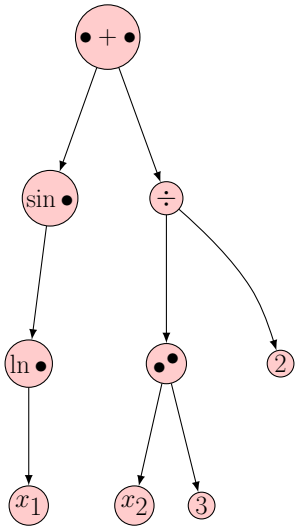


Рис. 1: Дерево выражения $\sin(\ln x_1) + \frac{x_2^3}{2}$

Здесь точками обозначены аргументы функций. Как видно, корнем дерева является вершина, соответствующая операции сложения, которая должна быть выполнена в последнюю очередь. Операция сложения имеет два различных поддерева, соответствующих двум аргументам этой операции. Заметим также, что здесь не использованы операции типа «разделить на два» или «возвести в куб». Вместо этого используются

операции деления и возведения в степень в общем виде, а в данном конкретном дереве соответствующие аргументы зафиксированны соответствующими константами.

3.1 Алгоритм порождения суперпозиций

Сначала определим понятие *глубины суперпозиции*:

Определение 2. *Глубина суперпозиции f — максимальная глубина дерева Γ_f.*

Теперь опишем итеративный алгоритм $\mathfrak{A}^*$, порождающий суперпозиции, не содержащие параметров. Описанный алгоритм породит любую суперпозицию конечной глубины за конечное число шагов.

Пусть дано множество примитивных функций $G = \{g_1, \ldots, g_l\}$ и множество свободных переменных $X = \{x_1, \ldots, x_n\}$.

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 $\mathfrak{A}^*$ итеративного порождения суперпозиций.*

1. Перед первым шагом зададим начальные значения множества $\mathcal{F}_0$ и вспомогательного индексного множества $\mathcal{I}$, служащего для запоминания, на какой итерации впервые встречена каждая суперпозиция:

$$\mathcal{F}_0 = X,$$

$$\mathcal{I} = \{(x, 0) \mid x \in X\}.$$

2. Для множества $\mathcal{F}_i$ построим вспомогательное множество U_i, состоящее из суперпозиций, полученных в результате применения функций $g_u \in G_u$ 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 $g_b \in G_b$:

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

4. Обозначим $\mathcal{F}_{i+1} = \mathcal{F}_i \cup U_i \cup B_i$.

5. Для каждой суперпозиции f из $\mathcal{F}_{i+1}$ добавим пару $(f, i+1)$ в множество $\mathcal{I}_f$, если суперпозиция f еще там не присутствует.

6. Перейдем к следующей итерации.

Тогда $\mathcal{F} = \cup_{i=0}^{\infty}\mathcal{F}_i$ — множество всех возможных суперпозиций конечной длины, которые можно построить из данного множества примитивных функций.

Вспомогательное множество $\mathcal{I}$ позволяет запоминать, на какой итерации была впервые встречена данная суперпозиция. Это необходимо, так как каждая суперпозиция, впервые порожденная на i-ой итерации, будет порождена еще раз и на любой итерации после i. Одной из возможностей избежать необходимости в этом множестве является построение $\mathcal{F}_{i+1}$ как $\mathcal{F}_{i+1} = U_i \cup B_i$ (без $\mathcal{F}_i$), а множества U_i и B_i строить следующим образом:

$$U_i = \{g_u \circ f \mid g_u \in G_u, f \in \cup_{j=0}^{i}\mathcal{F}_j\},$$
$$B_i = \{g_b \circ (f, h) \mid g_b \in G_b, f, h \in \cup_{j=0}^{i}\mathcal{F}_j\}.$$

Алгоритм $\mathfrak{A}^*$ очевидным образом обобщается на случай, когда множество G содержащит функции произвольной (но конечной) арности. Действительно, для такого обобщения достаточно строить аналогичным образом вспомогательные множества для этих функций, а именно, для множества функций G_n арности n построим вспомогательное множество H_i^n вида:

$$H_i^n = \{g \circ (f_1, f_2, \ldots, f_n) \mid g \in G_n, f_j \in \mathcal{F}_i\}.$$

В этих обозначениях $U_i \equiv H_i^1$, а $B_i \equiv H_i^2$.

Тогда множество $\mathcal{F}_{i+1} = \mathcal{F}_i \cup_{n=0}^{n_{max}} H_i^n$, где n_{max} — максимальное значение арности функций из G.

Теорема 1. *Алгоритм $\mathfrak{A}^*$ действительно породит любую конечную суперпозицию за конечное число шагов.*

Доказательство. Чтобы убедиться в этом, найдем номер итерации, на котором будет порождена некоторая произвольная конечная суперпозиция f. Чтобы найти этот номер, пронумеруем вершины графа Γ_f по следующим правилам:

- Если это вершина со свободной переменной, то она имеет номер 0.
- Если вершина V соответствует унарной функции, то она имеет номер $i + 1$, где i — номер дочерней для этой функции вершины.
- Если вершина V соответствует бинарной функции, то она имеет номер $i + 1$, где $i = \max(l, r)$, а l и r — номера, соответственно, первой и второй дочерней вершины.

Нумеруя вершины графа Γ_f таким образом, мы получим номер вершины, соответствующей корню графа. Это и будет номером итерации, на которой получена суперпозиция f.

Иными словами, для любой суперпозиции мы можем указать конкретный номер итерации, на котором она будет получена, что и требовалось. □

В предложенных ранее методах [9] 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 $\mathfrak{A}^*$ такие суперпозиции не могут возникнуть по построению.

3.2 Порождение параметризованных моделей

Алгоритм в таком виде не позволяет получать выражения, содержащие численные параметры $\boldsymbol{\omega}$ суперпозиции $f(\boldsymbol{\omega}, \mathbf{x})$. Покажем, однако, на примере конструирования множеств U_i и B_i, как исходный алгоритм $\mathfrak{A}^*$ может быть расширен с учетом таких параметров путем введения параметров:

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

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

Будем обозначать этот расширенный алгоритм как $\mathfrak{A}$. Здесь параметры α, β зависят только от комбинации g_u, f (или g_b, f, h для $\alpha, \beta, \psi, \phi$). Соответственно, для упрощения их индексы опущены.

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

Очевидно, при таком добавлении параметров $\alpha, \beta, \psi, \phi$ мы не изменяем мощности получившегося множества суперпозиций, поэтому алгоритм и выводы из него остаются корректны. В частности, исходный алгоритм является частным случаем данного при $\alpha \equiv \psi \equiv 1, \beta \equiv \phi \equiv 0$.

$\alpha, \beta, \psi, \phi$ являются параметрами модели. В практических приложениях можно оптимизировать значения этих параметров у получившихся суперпозиций, например, алгоритмом Левенберга-Марквардта [13, 14].

Заметим также, 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 параметры $\boldsymbol{\omega}$ у различных суперпозиций различны. Однако, так как на разных итерациях алгоритма мы можем получить, вообще говоря, одну и ту же суперпозицию с точностью до этих параметров, их необходимо не учитывать при тестировании различных суперпозиций на равенство.

Заметим, что и этот алгоритм очевидным образом обобщается на случай множества G, содержащего функции произвольной арности.

3.3 Количество возможных суперпозиций

Посчитаем количество суперпозиций, получаемых после каждой итерации алгоритма $\mathfrak{A}$. Очевидно, с учетом вышеупомянутых оговорок касательно сравнения параметризованных суперпозиций, это количество равно количеству для алгоритма $\mathfrak{A}^*$.

Итак, пусть дано n независимых переменных: $|X| = n$, а мощность множества G распишем через мощности его подмножеств функций соответствующей арности: $|G_1| = l_1, |G_2| = l_2, \ldots, |G_p| = l_p$. На нулевой итерации имеем $P_0 = n$ суперпозиций.

На первой итерации дополнительно порождается:

$$P_1 = l_1 n + l_2 n^2 + \cdots + l_n n^p = \sum_{i=1}^{p} l_i P_0^i,$$

и суммарное число суперпозиций после первой итерации:

$$\hat{P}_1 = P_1 + P_0 = \sum_{i=1}^{p} l_i P_0^i + P_0.$$

Как было замечено ранее, суперпозиции, порожденные на k-ой итерации, будут также порождены и на любой следующей после k итерации, поэтому суммарное число суперпозиций после второй итерации будет равно:

$$\hat{P}_2 = \sum_{i=1}^{p} l_i \hat{P}_1^i.$$

И вообще, после k-ой итерации будет порождено:

$$\hat{P}_k = \sum_{j=1}^{p} l_i \hat{P}_{k-1}^i.$$

Оценим порядок роста количества функций, порожденных после k-ой итерации.

Теорема 2. *Пусть в множестве примитивных функций G содержится l_p функций арности $p > 1$ и ни одной функции арности $p + k \mid k > 0$, и имеется $n > 1$ независимых переменных. Тогда справедлива следующая оценка количества суперпозиций, порожденных алгоритмом $\mathfrak{A}$ после k-ой итерации:*

$$|\mathcal{F}_k| = \mathcal{O}(l_p^{\sum_{i=0}^{k-1} p^i} n^{p^k}).$$

Доказательство. Оценим сначала порядок роста для случая, когда есть лишь одна m-арная функция и n свободных переменных.

После первой итерации алгоритма будет порождено $n^m + n$ суперпозиций. После второй — $(n^m + n)^m + n^m + n$, что можно оценить как $(n^m)^m = n^{m^2}$. И вообще, после k-ой итерации количество суперпозиций можно оценить как n^{m^k}.

Видно, что для оценки скорости роста количества порожденных суперпозиций можно учитывать только функции с наибольшей арностью.

Рассмотрим теперь случай, когда имеется не одна функция арности m, а l_m таких функций. Тогда на первой итерации порождается $l_m n^m + n$ суперпозиций, на второй:

$$l_m(l_m n^m + n)^m + l_m n^m + n \approx l_m^{m+1} n^{m^2},$$

на третьей, с учетом этого приближения

$$l_m(l_m^{m+1} n^{m^2})^m = l_m l_m^{m(m+1)} n^{m^3} = l_m^{m^2+m+1} n^{m^3}.$$

И вообще, скорость роста количества порожденных суперпозиций можно оценить как:

$$|\mathcal{F}_k| = \mathcal{O}(l_m^{\sum_{i=0}^{k-1} m^i} n^{m^k}).$$

Таким образом, получаем оценку для случая, когда в множестве G содержится l_p функций арности p и ни одной функции арности $p + k \mid k > 0$:

$$|\mathcal{F}_k| = \mathcal{O}(l_p^{\sum_{i=0}^{k-1} p^i} n^{p^k}).$$

□

3.4 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.

Определение 3. *Допустимая суперпозиция f — такая суперпозиция, значение которой определено для любой комбинации значений свободных переменных , область значений $\mathbb{X}$ которых определяется конкретной задачей, $\mathbb{X} \subset \mathbb{R}^n$ где n — число свободных переменных.*

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 $\mathbb{X}$ 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.

Таким образом, можно сформулировать очевидное *достаточное условие недопустимости* суперпозиции:

Определение 4. *Достаточное условие недопустимости суперпозиции f: в соответствующем дереве Γ_f хотя бы одна вершина V_i имеет хотя бы одну дочернюю вершину V_j такую, что область значений функции $g_{s(j)}$ шире, чем область определения функции $g_{s(i)}$:*

$$\exists i, j : V_i \in \Gamma_f, V_j \in \Gamma_f \wedge \exists \kappa : \kappa \in \mathcal{E}g_{s(j)} \wedge \kappa \notin \mathcal{D}g_{s(i)}.$$

Говоря, что область значений функции f шире области определения функции g, мы имеем ввиду, что существует по крайней мере одно значение функции f, не входящее в область определения функции g.

Подчеркнем, что, хотя свободные переменные могут принимать, например, все значения из $\mathbb{R}$, выбором множества $\mathbb{X}$ можно обеспечить возможность использования их в качестве аргументов функциям с более узкой, чем $\mathbb{R}$, но не менее узкой, чем $\mathbb{X}$, областью определения, если это не противоречит данной регрессионной выборке.

Для построения множества допустимых суперпозиций достаточно построить множество всех возможных суперпозиций при помощи алгоритма $\mathfrak{A}$, а затем удалить из этого множества все суперпозиции, не удовлетворяющие сформулированному признаку.

3.5 Применимость в задачах классификации

Предложенный алгоритм $\mathfrak{A}$ может быть применен и для решения задач классификации.

Выделим подмножества $G_\mu \subset G$, соответствующие различным дискретным cod: $g \in G$:

$$G_\mu = \{g \mid \mathrm{cod} g = \mathbb{Y}_\mu\},$$

где $\mathbb{Y}_\mu$ — различные дискретные множества, соответствующие различным наборам классов.

Тогда суперпозиции, область значений которых соответствует $\mathbb{Y}_\mu$ при фиксированном μ, и являются порожденными алгоритмом $\mathfrak{A}$ классификаторами для класса $\mathbb{Y}_\mu$. Таким образом, достаточно отобрать из всех порожденных суперпозиций f те, которые имеют в корневой вершине дерева Γ_f функцию $g \in G_\mu$.

4 Алгоритм Левенберга-Марквардта и мультистарт

Алгоритм Левенберга-Марквардта ($\mathcal{LM}$) [13, 14] предназначен для решения задачи минимизации функции, представляющей из себя сумму квадратичных членов. В частности, он используется для оптимизации параметров нелинейных регрессионных моделей в предположении, что в качестве критерия оптимизации используется среднеквадратичная ошибка модели на обучающей выборке:

$$S(\boldsymbol{\omega}) = \sum_{i=1}^{N} [y_i - f(\boldsymbol{\omega}, \mathbf{x}_i)]^2 \to \min,$$

где $\boldsymbol{\omega}$ — вектор параметров перпозиции f.

$\mathcal{LM}$ может рассматриваться как комбинация методов Гаусса-Ньютона и градиентного спуска.

Перед началом работы алгоритма задается начальный вектор параметров $\boldsymbol{\omega}_0$. На каждой итерации этот вектор заменяется новой оценкой, $\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k + \boldsymbol{\delta}_k$. Для определения $\boldsymbol{\delta}_k = \boldsymbol{\delta}$ используется линейное приближение функции:

$$\mathbf{f}(\boldsymbol{\omega} + \boldsymbol{\delta}, \mathbf{X}) \approx \mathbf{f}(\boldsymbol{\omega}, \mathbf{X}) + \mathbf{J}\boldsymbol{\delta},$$

где $\mathbf{J}$ — якобиан функции $\mathbf{f}$ в точке $\boldsymbol{\omega}$.

Приращение $\boldsymbol{\delta}$ в точке $\boldsymbol{\omega}$, доставляющей минимум S, равно нулю, поэтому для нахождения последующего значения приращения $\boldsymbol{\delta}$ приравняем нулю вектор частных производных S по $\boldsymbol{\omega}$. То есть, в векторной нотации:

$$S(\boldsymbol{\omega} + \boldsymbol{\delta}) \approx \|\mathbf{y} - \mathbf{f}(\boldsymbol{\omega}) - \mathbf{J}\boldsymbol{\delta}\|^2.$$

Дифференциирование по $\boldsymbol{\delta}$ и приравнивание нулю приводит к следующему уравнению для $\boldsymbol{\delta}$:

$$(\mathbf{J}^T\mathbf{J})\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\omega})].$$

Левенберг предложил заменить $(\mathbf{J}^T\mathbf{J})$ на $(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})$, где λ — некоторый параметр регуляризации. Марквардт дополнил это предложение с целью более быстрого движения по тем направлениям, где градиент меньше. Для этого вместо $\mathbf{I}$ используется диагональ матрицы $\mathbf{J}^T\mathbf{J}$, и искомое уравнение на $\boldsymbol{\delta}$ выглядит как:

$$(\mathbf{J}^T\mathbf{J} + \lambda\mathrm{diag}(\mathbf{J}^T\mathbf{J}))\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\omega})].$$

Решая это уравнение, получаем окончательное выражение для $\boldsymbol{\delta} = \boldsymbol{\delta}_k$:

$$\boldsymbol{\delta}_k = (\mathbf{J}^T\mathbf{J} + \lambda\mathrm{diag}(\mathbf{J}^T\mathbf{J}))^{-1}\mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\omega})].$$

4.1 Выбор λ

Для определения параметра регуляризации λ в настоящей работе применяется следующая эвристика.

В начале работы $\mathcal{LM}$ задается некоторое значение λ_0, например, 0.01, и фиксируется коэффициент $\nu > 1$. Затем, на каждой итерации алгоритма вычисляется значение функционала ошибки для $\lambda = \lambda_i$ и $\lambda = \frac{\lambda_i}{\nu}$. В случае, если хотя бы одно из этих значений доставляет функционалу ошибки меньшее значение, чем до этой итерации, то λ_{i+1} принимается равным этому значению. Иначе λ_i умножается на ν до тех пор, пока значение функционала ошибки не уменьшится.

4.2 Мультистарт

Как и всякий подобный алгоритм оптимизации, $\mathcal{LM}$ находит лишь локальный минимум. Для решения этой проблемы применяется метод *мультистарта*: случайным образом задается несколько начальных приближений, и для каждого из них запускается $\mathcal{LM}$. Если найдено несколько различных локальных минимумов, то выбирается тот из них, в котором значение $S(\boldsymbol{\omega})$ меньше всего.

5 Алгоритм итеративного стохастического порождения суперпозиций

Несмотря на то, что указанный ранее итеративный алгоритм порождения суперпозиций позволяет получить за конечное число шагов 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 данном разделе описывается примененный в настоящей работе алгоритм.

Сначала опишем вспомогательный алгоритм случайного порождения суперпозиции:

Алгоритм 2. *Алгоритм случайного порождения суперпозиции $\mathcal{RF}$.*
Вход:

- *Набор пороговых значений* $0 < \xi_1 < \xi_2 < \xi_3 < 1$.
- *Максимальная глубина порождаемой суперпозиции* Td.

Алгоритм работает следующим образом. Генерируется случайное число ξ на интервале $(0; 1)$, и рассматриваются следующие случаи:

- $\xi \leq \xi_1$: результатом алгоритма является некоторая случайно выбранная свободная переменная.
- $\xi_1 < \xi \leq \xi_2$: результатом алгоритма является числовой параметр.
- $\xi_2 < \xi \leq \xi_3$: результатом алгоритма является некоторая случайно выбранная унарная функция, для определения аргумента которой данный алгоритм рекурсивно запускается еще раз.
- $\xi_3 < \xi$: результатом алгоритма является некоторая случайно выбранная бинарная функция, аргументы которой порождаются аналогичны образом.

При этом, порождение тривиальных суперпозиций (свободных переменных и параметров) запрещено: на самом первом шаге пороговые значения масштабируется таким образом, чтобы всегда породилась унарная или бинарная функция. Аналогично при превышении значения Td пороговые значения масштабируются таким образом, чтобы был порожден узел, соответствующий свободной переменной или параметру, и алгоритм завершился.

В ходе работы предлагаемого алгоритма каждой суперпозиции f ставится в соответствие ее *качество* Q_f (иногда будем говорить, что суперпозиция *оценивается*), рассчитываемое исходя из функции ошибки S_f этой суперпозиции на выборке D и ее сложности C_f — числа узлов в соответствующем графе Γ_f, по следующей формуле:

$$Q_f = \frac{1}{1 + S_f} \left(\alpha \hat{Q} + \frac{1 - \alpha \hat{Q}}{1 + \exp(C_f - \tau)} \right), \tag{4}$$

где $\hat{Q}$ — минимальная приспособленность суперпозиции из критерия останова, α — некоторый коэффициент, $0 \ll \alpha < 1$, а τ — коэффициент, характеризующий желаемую сложность модели. Второй множитель в данной формуле выполняет роль штрафа за слишком большую сложность суперпозиции.

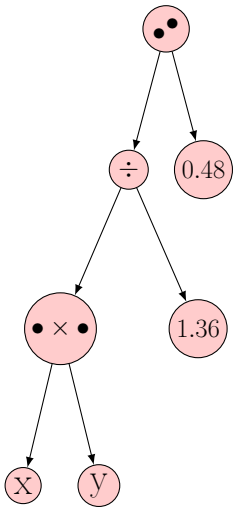
Таким образом, чем лучше результаты суперпозиции, тем ближе значение ее приспособленности к 1, и, наоборот, чем хуже — тем ближе к 0.

Итак, теперь опишем сам алгоритм:

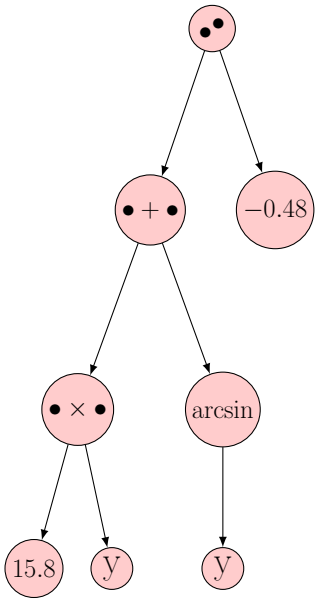
Алгоритм 3. *Итеративный алгоритм стохастического порождения суперпозиций.*
Вход:

- *Множество порождающих функций G, состоящее только из унарных и бинарных функций.*
- *Регрессионная выборка D.*
- N_{max} *— максимальное число одновременно рассматриваемых суперпозиций.*
- I_{max} *— максимальное число итераций алгоритма.*
- $\hat{Q}$ *— минимальная приспособленность суперпозиций.*
- *Прочие параметры, используемые в* (4) *и алгоритме 2.*

1. Инициализируется начальный массив $\mathcal{X}_f$ суперпозиций. А именно, порождается N_{max} суперпозиций алгоритмом 2.
2. Оптимизируются параметры $\boldsymbol{\omega}$ суперпозиций из $\mathcal{X}_f$ алгоритмом $\mathcal{LM}$.
3. Вычисляется значение Q_f для каждой еще не оцененной суперпозиции f из $\mathcal{X}_f$: для нее рассчитывается значение функции ошибки S_f согласно (3) на выборке D, и ставится в соответствие значение Q_f в соответствии с (4).
4. Массив суперпозиций $\mathcal{X}_f$ сортируется согласно их приспособленности.
5. Наименее приспособленные суперпозиции удаляются из массива $\mathcal{X}_f$ до тех пор, пока его размер не станет равен N_{max}.
6. Отбирается некоторая часть наименее приспособленных суперпозиций из $\mathcal{X}_f$ (в данной работе — $\frac{1}{3}$ от числа всех суперпозиций). У этой части происходит случайная замена одной функции или свободной переменной на другую: генерируются две случайные величины, одна из которых служит для выбора вершины дерева Γ_f, которую предстоит изменить, а другая — для выбора нового элемента для этой вершины. Замена такова, чтобы сохранилась структура суперпозиции, а именно — в случае замены функции сохраняется арность, а свободная переменная заменяется только на другую свободную переменную. При этом исходные суперпозиции сохраняются в массиве $\mathcal{X}_f$.
7. Повторяются шаги $3 - 4$.
8. Производится случайный обмен поддеревьями наиболее приспособленных суперпозиций. Вершины, соответствующие этим поддеревьям, выбираются случайным образом. При этом исходные суперпозиции сохраняются в массиве $\mathcal{X}_f$.
9. Повторяются шаги $2 - 4$.



Граф суперпозиции $\left(\frac{1.36}{xy}\right)^{0.48}$



Граф суперпозиции $(15.8y + \arcsin y)^{-0.48}$

Таблица 1: Графы результирующих суперпозиций

10. Проверяются условия останова: если либо число итераций больше I_{max}, либо в массиве $\mathcal{X}_f$ есть хотя бы одна суперпозиция с приспособленностью больше, чем $\hat{Q}$, то алгоритм останавливается, и результатом является наиболее приспособленная суперпозиция, иначе переход к шагу 2.

Заметим, что выборка D не делится на обучающую и контролирующую — контроль качества оставляется различным стандартным методикам типа скользящего контроля.

Аналогично алгоритму $\mathfrak{A}$, предложенный стохастический алгоритм также может быть применен и для решения задач классификации.

6 Вычислительный эксперимент

В вычислительном эксперименте восстанавливается регрессионная зависимость волатильности опциона от его стоимости и сроков исполнения [15, 16]. Используются исторические данные о волатильности опционов Brent Crude Oil. Срок действия опциона — полгода, с 02.01.2001 по 26.06.2001, тип — право на продажу базового инструмента. Базовым инструментом в данном случае является нефть. Использовались ежедневные цены закрытия опциона и базового инструмета.

Данный инструмент имеет низкую волатильность, вследствие чего среди данных нет выбросов. В данных имется пропуски, так как опционы с ценами, далекими от цен базового инструмента, не торговались сразу после выпуска опционов.

В ходе предобработки данных выяснено, что для больших значений волатильности зависимость преобладает существенно неоднозначный характер, поэтому для облегчения аналитического описания моделировалась зависимость цены от волатильности и времени. Искомая зависимость для ряда получившихся суперпозиций легко восстанавливается аналитически нахождением обратных формул.

Использованные параметры алгоритма 3: $N_{max} = 200, I_{max} = 50, \hat{Q} = 0.95, \tau = 10, \alpha = 0.05$. При отсутствии улучшения результатов в течение нескольких итераций подряд алгоритм 3 также завершался.

В таблице 2 приведены некоторые из наилучших суперпозиций, порожденных в результате работы алгоритма 3. Указан номер итерации i, на которой суперпозиция была впервые получена, сама суперпозиция, среднеквадратичная ошибка (S_f) и сложность в смысле количества узлов в соответствующем графе выражения. Числовые коэффициенты в приведенных формулах и значения функционала S_f искусственно округлены до $2-3$ значащей цифры. Результаты отсортированы в порядке возрастания сложности получившихся суперпозиций.

i	Суперпозиция	S_f	C_f
9	$\left(\frac{1.36}{xy}\right)^{0.48}$	≈ 0.0182	7
14	$(15.8y + \arcsin y)^{-0.48}$	≈ 0.0208	8
13	$(yx^{0.882} + \arcsin y)^{-0.482}$	≈ 0.0178	10
8	$0.125\frac{y}{(y^2)^{0.8+y}}$	≈ 0.0171	11
14	$\frac{\frac{3.86\cdot 10^{11}+y}{y\frac{1.227\cdot 10^{11}}{x^y}-2.46\cdot 10^8}}{y\cos\left(\frac{\frac{-5.89\cdot 10^{-3}+y}{y-5.47\cdot 10^{-3}}}{\frac{y\cos y}{y}}\right)}}{y^y\cos y + xy}$	≈ 0.0092	42

Таблица 2: Результаты вычислительного эксперимента

В таблице 1 представлены графы первых двух упомянутых в таблице суперпозиций. На рисунках 2 и 3 отображены изометрическая проекция и проекция на одну из плоскостей для суперпозиции $\left(\frac{1.36}{xy}\right)^{0.48}$.

7 Заключение

В работе исследованы индуктивные алгоритмы порождения допустимых существенно нелинейных суперпозиций. Предложен переборный алгоритм, порождающий все возможные суперпозиции заданной сложности за конечное число шагов, и приведено его теоретическое обоснование.

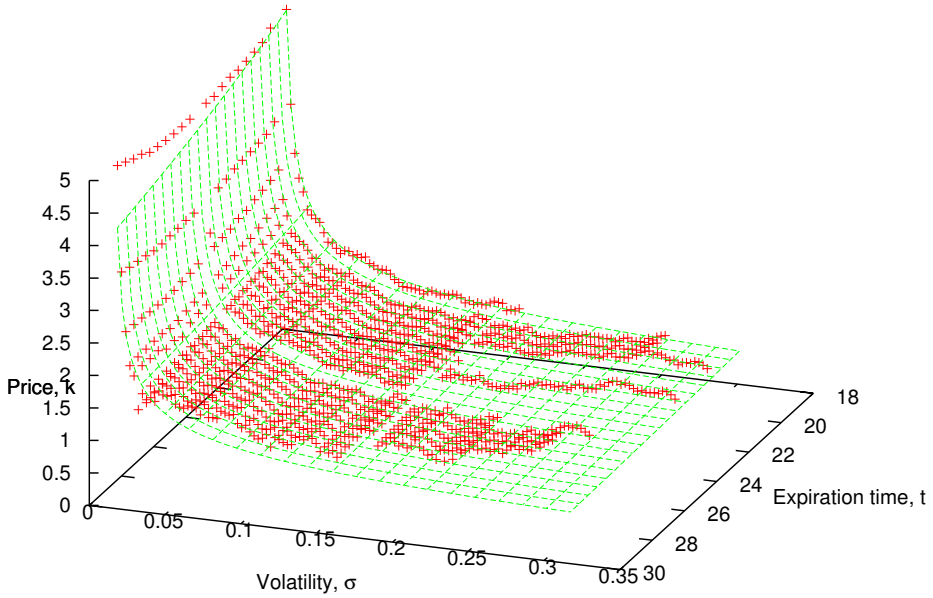


Рис. 2: Изометрическая проекция результирующей суперпозиции

Алгоритм индуктивного порождения моделей, предложенный в настоящей работе, решает некоторые типичные проблемы предложенных ранее методов, упомянутые, например, в [9], а именно:

- Порождение рекурсивных суперпозиций, суперпозиций, содержащих несоответствующее используемым функциям число аргументов, и т. д. — в предложенном алгоритме эти проблемы не возникают по построению.
- Несовпадение области определения некоторой примитивной функции и области значений ее аргументов (возможно, тоже некоторых суперпозиций).
- При ограничении числа примитивных функций, участвующих в суперпозиции, а также при соответствующем задании множества примитивных функций исключается проблема слишком сложных суперпозиций.

Описан стохастический алгоритм индуктивного порождения существенно нелинейных суперпозиций и приведены результаты его работы для задачи моделирования волатильности опционов. Предложенный алгоритм также может оказаться полезным в задачах классификации.

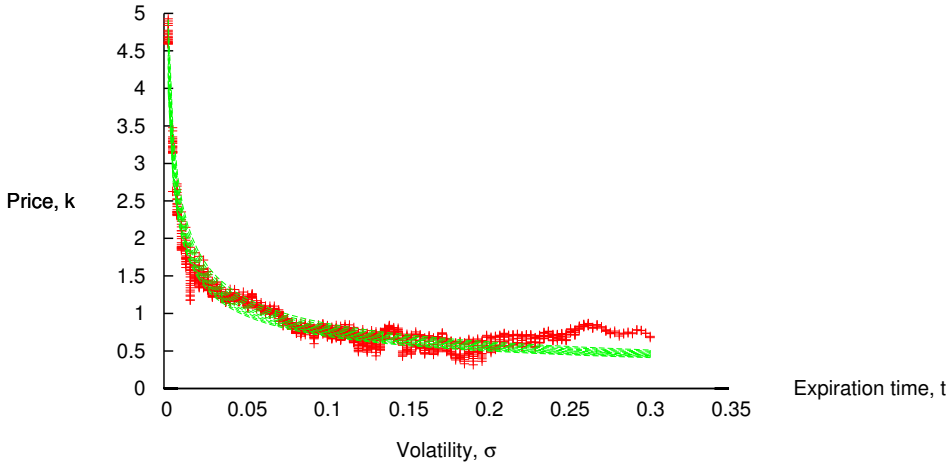


Рис. 3: Проекция результирующей суперпозиции

Список литературы

[1] John Duffy and Jim Engle-Warnick. Using symbolic regression to infer strategies from experimental data. In Shu-Heng Chen, editor, *Evolutionary Computation in Economics and Finance*, volume 100 of *Studies in Fuzziness and Soft Computing*, chapter 4, pages 61–84. Physica Verlag, 2002 2002.

[2] P. Barmpalexis, K. Kachrimanis, A. Tsakonas, and E. Georgarakis. Symbolic regression via genetic programming in the optimization of a controlled release pharmaceutical formulation. *Chemometrics and Intelligent Laboratory Systems*, 107(1):75–82, 2011.

[3] Michael Schmidt and Hod Lipson. Symbolic regression of implicit equations. In Rick L. Riolo, Una-May O'Reilly, and Trent McConaghy, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 5, pages 73–85. Springer, Ann Arbor, 14-16 May 2009.

[4] J. W. Davidson, D. A. Savic, and G. A. Walters. Symbolic and numerical regression: experiments and applications. In Robert John and Ralph Birkenhead, editors, *Developments in Soft Computing*, pages 175–182, De Montfort University, Leicester, UK, 29-30 June 2000. 2001. Physica Verlag.

[5] Claude Sammut and Geoffrey I. Webb. Symbolic regression. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, page 954. Springer, 2010.

[6] Vadim Strijov and Gerhard-Wilhelm Weber. Nonlinear regression model generation using hyperparameter optimization. *Computers & Mathematics with Applications*, 60(4):981–988, 2010.

[7] 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.

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

[9] 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.

[10] Тырсин А.Н. Об эквивалентности знакового и наименьших модулей методов построения линейных моделей. *Обозрение прикладной и промышленной математики*, 12(4):879–880, 2005.

[11] Ю. Н. Павловский. *Имитационные модели и системы*. Фазис, 2000.

[12] А. Б. Иванов и др. В. И. Битюцков, М. И. Войцеховский. *Математическая энциклопедия*, volume 4. Советская Энциклопедия, 1984.

[13] D. W. Marquardt. An algorithm for least-squares estimation of non-linear parameters. *Journal of the Society of Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[14] J. J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In *G.A. Watson*, Lecture Notes in Mathematics 630, pages 105–116. Springer-Verlag, Berlin, 1978. Cited in Åke Björck's bibliography on least squares, which is available by anonymous ftp from `math.liu.se` in `pub/references`.

[15] T. Daglish, J. Hull, and W. Suo. Volatility surfaces: Theory, rules of thumb, and empirical evidence. *Quantitative Finance*, 7(5):507–524, 2007.

[16] В. В. Стрижов and Р. А. Сологуб. Индуктивное порождение регрессионных моделей предполагаемой волатильности для опционных торгов. *Вычислительные технологии*, 14(5):102–113, 2009.