Министерство образования и науки РФ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина»
Институт фундаментального образования
Кафедра интеллектуальных информационных технологий

К ЗАЩИТЕ ДОПУСТИТЬ

Заведующий кафедрой ИИТ

_____И. Н. Обабков

« ____»_____20___ г.

УСТАНОВЛЕНИЕ ФУНКЦИОНАЛЬНОЙ ЗАВИСИМОСТИ ДАННЫХ ПОСРЕДСТВОМ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Пояснительная записка

Руководитель А. А. Мокрушин

Нормоконтролер Е. М. Потылицина

Студент гр. ФО–411101 А. А. Ибакаева

Екатеринбург – 2015

РЕФЕРАТ

Выпускная квалификационная работа на соискание академической степени бакалавра 46 с., 6 рис., 10 источников.

СИМВОЛЬНАЯ РЕГРЕССИЯ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Объект исследования – алгоритм установления функциональной зависимости.

Цель работы – нахождение функции, наиболее оптимально устанавливающей, согласно некоторым критериям (нормальность, гладкость), зависимость двух наборов данных друг от друга.

Методы исследования: изучение предметной области, формализация задачи, анализ программного обеспечения.

Результаты работы: разработана программа установления функциональной зависимости данных.

Выпускная квалификационная работа выполнена в текстовом редакторе Microsoft Word и представлена в твердой копии.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Символьная регрессия – задача нахождения формулы, которая описывает некую зависимость. Более формально, её можно сформулировать как построение регрессионной модели в виде функциональной зависимости.

Регрессионная модель *f(w, x)* – это параметрическое семейство функций, задающее отображение (формула 1).

$$f: W \times X \to Y, \quad (1)$$

где $w \in W$ – пространство параметров, $x \in X$ – пространство свободных переменных, Y – пространство зависимых переменных.

Функциональная зависимость будет строиться посредством генетического программирования. Данный оптимизационный стохастический алгоритм является модификацией генетического алгоритма. Основное отличие генетического программирования заключается в работе со структурами данных переменного размера. Суперпозиция заданных функций будет представлена в виде дерева, преобразующегося в польскую запись для вычисления математического выражения.

Для решения задачи будет случайным образом создана популяция таких деревьев. Узлы дерева являются математическими функциями, а листы терминальными символами, т.е. константами или независимыми переменными.

Каждое дерево – решение проблемы – будет вычисляться для набора свободных переменных. Если полученный результат достаточно точно совпадет с ожидаемым результатом – зависимой переменной, то необходимая функциональная зависимость считается найденной. В противном случае поиск продолжается дальше.

Следует отметить, что мы заранее не задаем вид и размер необходимой функции. Она меняется случайным образом в ходе выполнения программы путем применения различных генетических операций. Поэтому в результате

может получиться достаточно сложная функциональная зависимость, которую впоследствии можно упрощать и улучшать другими методами.

Актуальность выбранной темы выпускной квалификационной работы обосновывается тем, что символьная регрессия может использоваться для получения эмпирических зависимостей на основе экспериментальных данных. Она широко применяется для решения задач моделирования и прогнозирования. Кроме этого, символьная регрессия с успехом применяется в символьных вычислениях, включая символьное дифференцирование и интегрирование, решение дифференциальных и интегральных уравнений в символьном виде и т.п.

Аналоги. Символьная регрессия отличается от традиционной линейной, квадратичной или полиномиальной регрессии, которые просто находят числовые коэффициенты для функции, вид которой заранее известен.

Проблема. Какая функциональная зависимость существует между двумя разными наборами данных? Другими словами, существует два набора данных. Есть предположение, что они зависят друг от друга. Необходимо найти вид функциональной зависимости.

Цель – нахождение и отработка методики восстановления функциональной зависимости данных.

Для достижения поставленной цели необходимо выполнить следующие задачи:

а) Изучить соответствующую литературу.
б) Разработать алгоритм нахождения функциональной зависимости.
в) Реализовать программу по данному алгоритму.
г) Протестировать программу с реальными данными.
д) Обобщить полученные результаты и сделать соответствующие выводы.

Объектом исследования выступает алгоритм установления функциональной зависимости.

1 Теоретическая часть

1.1 Постановка задачи

Необходимо разработать программу, которая решает задачу символьной регрессии методом генетического программирования. Создаваемая программа должна удовлетворять следующим критериям:

а) На основе входных данных в виде выборки множества значений свободных и зависимых переменных, а также максимально возможной погрешности выдавать результат в виде функции, которая для всех точек начальной выборки принимает значения с суммарной квадратичной ошибкой меньше заданной.

б) Иметь возможность изменения параметров генетического программирования – мощность начальной популяции, максимальная глубина дерева, вероятности скрещивания, репродукции и мутации.

в) Быть удобной в использовании, т.е. предоставлять промежуточные и конечные результаты работы в читабельном виде.

1.2 Обзор аналогов

Существует несколько различных методов аппроксимации функции. Среди них можно выделить:

а) Метод наименьших квадратов (МНК).

б) Регрессия.

1.2.1 Метод наименьших квадратов

В ходе проведения эксперимента можно получить различные значения зависимой переменной *y* при различных значениях свободной переменной *x*.

По этим данным можно построить график зависимости (формула 2).

$$y = f(x) \quad (2)$$

Полученная кривая дает возможность судить о виде функции *f(x)*. Однако постоянные коэффициенты, которые входят в эту функцию, остаются неизвестными. Определить их позволяет метод наименьших квадратов. Экспериментальные точки, как правило, не ложатся точно на кривую. Метод наименьших квадратов требует, чтобы сумма квадратов отклонений экспериментальных точек от кривой (формула 3) была наименьшей.

$$(y_i - f(x_i))^2 \quad (3)$$

Практически этот метод наиболее часто (и наиболее просто) используется в случае линейной зависимости.

Рассмотрим прямую, проходящую через начало координат (формула 4).

$$y = kx \quad (4)$$

Составим величину φ – сумму квадратов отклонений наших точек от прямой линии (формула 5).

$$\varphi = \sum_{i=1}^{n}(y_i - kx_i)^2 \quad (5)$$

Величина φ всегда положительна и оказывается тем меньше, чем ближе к прямой лежат наши точки. Метод наименьших квадратов утверждает, что для k следует выбирать такое значение, при котором φ имеет минимум (формула 6 и формула 7).

$$\frac{d\varphi}{dk} = -2\sum_{i=1}^{n} x_i(y_i - kx_i = 0) \quad (6)$$

или

$$k = \frac{\sum x_i y_i}{\sum {x_i}^2} \quad (7)$$

Среднеквадратичная ошибка определения величины k представлена в формуле 8.

$$S_k = \sqrt{\frac{1}{(n-1)} * (\frac{\sum(y_i - kx_i)^2}{\sum {x_i}^2})}, \quad (8)$$

где – n число измерений.

Рассмотрим теперь несколько более трудный случай, когда точки должны удовлетворить формуле 9 (прямая, не проходящая через начало координат).

$$y = a + bx \text{ (9)}$$

Задача состоит в том, чтобы по имеющемуся набору значений x_i, y_i найти наилучшие значения a и b.

Снова составим квадратичную форму φ, равную сумме квадратов отклонений точек x_i, y_i от прямой (формула 10).

$$\varphi = \sum_{i=1}^{n}(y_i - a - bx_i)^2 \text{ (10)}$$

Далее найдем значения a и b, при которых φ достигает минимума (формула 11 и формула 12).

$$\frac{d\varphi}{da} = -2\sum(y_i - a - bx_i) = 0 \text{ (11)}$$

$$\frac{d\varphi}{db} = -2\sum(y_i - a - bx_i) = 0 \text{ (12)}$$

Совместно решим эти уравнения (формула 13 и формула 14).

$$b = \frac{\sum[(x_i - \bar{x})y_i]}{\sum(x_i - \bar{x})^2} \text{ (13)}$$

$$a = \bar{y} - b\bar{x} \text{ (14)}$$

Теперь можно вычислить среднеквадратичные ошибки определения a и b (формула 15 и формула 16).

$$S_a = \sqrt{\left(\frac{\sum(y_i - bx_i - a)^2}{n-2}\right)\left(\frac{1}{n} + \frac{\bar{x}^2}{\sum(x_i - \bar{x})^2}\right)} \text{ (15)}$$

$$S_b = \sqrt{\frac{\sum(y_i - bx_i - a)^2}{(n-2)\sum(x_i - \bar{x})^2}} \text{ (16)}$$

Преимущество использования метода наименьших квадратов заключается в сведении всех вычислительных процедур к простому вычислению неизвестных коэффициентов, а также в доступности математических выводов. Недостатком данного метода является чувствительность оценок к резким выбросам, встречающимся в исходных данных.

1.2.2 Регрессия

Регрессия – зависимость математического ожидания (например, среднего значения) случайной величины от одной или нескольких других

случайных величин (свободных переменных). Регрессионным анализом называется поиск такой функции, которая описывает эту зависимость. Регрессия может быть представлена в виде суммы неслучайной и случайной составляющих (формула 17).

$$y = f(x) + \nu, \quad (17)$$

где f – функция регрессионной зависимости, а ν – аддитивная случайная величина с математическим ожиданием равным нулю.

Предположение о характере распределения этой величины называется гипотезой порождения данных. Обычно предполагается, что величина ν имеет гауссово распределение с нулевым средним и дисперсией ${\sigma_\nu}^2$.

Задача нахождения регрессионной модели нескольких свободных переменных ставится следующим образом. Задана выборка – множество $\{x_1, \dots, x_N | x \in R^M\}$ значений свободных переменных и множество $\{y_1, \dots, y_N | y \in R\}$ соответствующих им значений зависимой переменной. Эти множества обозначаются как *D*, множество исходных данных $\{(x, y)_i\}$. Задана регрессионная модель – параметрическое семейство функций *f(w, x)* зависящая от параметров $w \in R$ и свободных переменных *x*. Требуется найти наиболее вероятные параметры $\overline{w}$ (формула 18).

$$\overline{w} = \operatorname{argmax}_{w \in R^w} p(y|x, w, f) = p(D|w, f) \quad (18)$$

Функция вероятности p зависит от гипотезы порождения данных и задается Байесовским выводом или методом наибольшего правдоподобия.

Линейная регрессия предполагает, что функция *f* зависит от параметров *w* линейно (формула 19). При этом линейная зависимость от свободной переменной *x* необязательна.

$$y = f(w, x) + \nu = \sum_{j=1}^{N} w_j * g_j(x) + \nu, \quad (19)$$

где $g = [g_1, \dots, g_n]$ – функция из некоторого заданного множества.

Значения параметров в случае линейной регрессии находят с помощью метода наименьших квадратов. Использование этого метода обосновано предположением о гауссовом распределении случайной переменной.

Нелинейной называют регрессию, которая не может быть представлена в виде скалярного произведения (формула 20).

$$f(w, x) = \big(w, g(x)\big) = \sum_{i=1}^{n} w_i g_i(x), \quad (20)$$

где $w = [w_1, \ldots, w_n]$ – параметры регрессионной модели, x – свободная переменная из пространства R^n, y – зависимая переменная, v – случайная величина и $g = [g_1, \ldots, g_n]$ – функция из некоторого заданного множества.

Значения параметров в случае нелинейной регрессии находят с помощью одного из методов градиентного спуска, например алгоритма Левенберга-Марквардта.

Однако в представленных методах происходит поиск коэффициентов модели, структура и сложность которой известны заранее. В отличие от линейной, квадратичной и других видов регрессии символьная регрессия подразумевает как определение коэффициентов, так и построение оптимальной модели.

1.3 Обзор генетического программирования

В природе биологические структуры, которые наиболее успешно борются со своей окружающей средой, выживают и размножаются с более высокой скоростью. Биологи понимают эти структуры как следствие естественного отбора по Дарвину, действующего в среде в течение промежутка времени. Другими словами, в природе структура – это следствие пригодности. То есть пригодность порождает структуру, основываясь на естественном отборе, а также с помощью половой рекомбинации (генетического скрещивания) и мутации.

Когда переменные являются вещественными числами, символьная регрессия включает в себя одновременно и поиск вида функции, и поиск числовых коэффициентов модели. Этим символьная регрессия отличается от традиционной линейной, квадратичной или полиномиальной регрессии,

которые просто находят числовые коэффициенты для функции, вид которой заранее известен.

Поиск математического выражения в символьной форме может рассматриваться в качестве компьютерной программы, которая принимает значения независимых переменных в качестве входных параметров и вычисляет значения зависимых переменных на выходе.

1.4 Генетический алгоритм

В природе эволюция происходит при наличии следующих условий:

а) Организм способен воспроизводить себя;
б) Существует популяция таких способных к размножению особей;
в) Есть некоторое разнообразие организмов;
г) Некоторые различия в способности выжить связаны с этим разнообразием.

В естественной среде разнообразие обеспечивается различием хромосом особей популяции. Это различие преобразуется в изменение структуры организма и его поведения в среде. А это в свою очередь влияет на способность выживания в среде и скорость размножения. Организмы, которые способны лучше выполнять задачи в своей среде, чаще выживают и чаще размножаются, в отличие от менее пригодных особей. Эта концепция естественного отбора и выживания сильнейших была описана Чарльзом Дарвином в книге «О происхождении видов путем естественного отбора» (1859). С течением времени это приводит к тому, что в популяции остаются только особи с такими структурами организма и поведением, которые позволяют им выживать и производить себе подобных. Таким образом, структура особей в популяции меняется из-за естественного отбора. Когда мы видим ощутимые различия в структуре, возникшие из-за разницы в пригодности особей, то говорим, что популяция эволюционировала.

Когда у нас есть популяция особей, то наличие различий, дифференцированно влияющих на способность выжить, почти неизбежно. Поэтому на практике достаточно только первого условия для начала эволюции.

Книга Джона Холланда «Adaptation in Natural and Artificial Systems» (1975) дала основу для наблюдения за всеми адаптивными системами, а затем показала, как эволюционный процесс может быть применен к искусственным системам. Любая проблема адаптации может быть сформулирована в генетических терминах. А после формулировки такая проблема может быть решена генетическим алгоритмом.

Генетический алгоритм симулирует эволюционные процессы Дарвина и природные генетические операции с хромосомами.

Генетический алгоритм является параллельным математическим алгоритмом, который преобразует набор отдельных математических объектов (как правило, символьных строк фиксированной длины), каждый из которых связан с соответствующим значением функции пригодности, в новую популяцию (следующее поколение) с помощью операций, основанных на концепции выживания сильнейших по Дарвину и природных генетических операций (в частности, половой рекомбинации).

Почему генетический алгоритм работает?

На первый взгляд кажется, что тестирование случайным образом созданных строк не даст ничего, кроме значения приспособленности для этих проверяемых точек.

С помощью полученных значений пригодности можно узнать среднюю приспособленность популяции. Это оценка средней пригодности пространства поиска. Она имеет статистическую дисперсию, т.к. это не среднее значение всех точек пространства поиска, а лишь расчет, основанный на тестируемых точках.

После получения среднего значения приспособленности мы иначе смотрим на проверенные строки популяции. Теперь можно увидеть, какие

строки лучше, и насколько лучше остальных они решают заданную проблему.

Далее необходимо решить, что делать дальше.

Одним из вариантов может быть продолжение случайного выбора точек пространства поиска и проверка их пригодности. Но случайный слепой поиск стратегии не адаптивен и не интеллектуален в том смысле, что мы не используем полученную информацию об окружающей среде, чтобы повлиять на направление поиска. Для любой проблемы с нетривиальным пространством поиска невозможно протестировать больше, чем очень малую часть от общего количества точек пространства поиска методом случайного слепого поиска. Пусть есть K^L точек проблемной области, представленных в виде строк размером L над алфавитом размера K. Например, если возможно проверить миллиард точек за секунду, и если слепой случайный поиск продолжается с начала Вселенной (т.е. около 15 млрд. лет), то мы бы нашли только около 10^{27} точек пространства поиска. Пространство поиска 10^{27} – это приблизительно 2^{90} точек, соответствующих двоичной строке с относительно скромной длиной $L = 90$.

Другой вариант состоит в «жадном» использовании лучшего результата тестирования начальной популяции. «Жадная» стратегия предполагает применение этого лучшего результата без тестирования каких-либо других точек пространства. «Жадная» стратегия, в отличие от случайного слепого поиска, является адаптивной и интеллектуальной, потому что использует информацию, полученную на одном этапе поиска, чтобы влиять на направление поиска следующего шага. В целом можно ожидать, что такая стратегия будет в два раза лучше случайного слепого поиска.

Но «жадная» стратегия дает мнимую уверенность, что лучшая точка пространства поиска будет случайно выбрана в маленькую начальную популяцию. В любом интересном пространстве поиска значимого размера маловероятно, что лучшая точка поколения начальной популяции окажется

глобальным оптимумом всего пространства поиска, это маловероятно также для всех ранних поколений. Наша цель состоит в максимизации пользы на протяжении большого промежутка времени, а «жадная» стратегия весьма преждевременна на данном этапе.

Признавая, что мы нашли лучшую точку поколения, не рекомендуется исключать все остальные точки. Мы должны дать преимущество всем точкам, превосходящим среднее значение фитнеса.

Но если мы не будем проверять новые точки пространства, то вернемся к только что отклоненной «жадной» стратегии, т.е. будем наблюдать только за лучшими точками начальной выборки.

Оптимальная адаптивная (интеллектуальная) система должна обрабатывать имеющуюся на данный момент информацию об окружающей среде, чтобы найти оптимальное соотношение между стоимостью освоения новых точек в пространстве и стоимостью применения уже оцененных точек проблемной области. Этот выбор, по сути, должен отражать статистическую дисперсию, связанную с затратами.

Однако в генетическом алгоритме, как и в природе, особи, фактически находящиеся в популяции, имеют второстепенное значение для эволюционного процесса. В природе, если определенная особь доживает до возраста воспроизводства и действительно размножается половым путем, по крайней мере, некоторые из хромосом этой особи сохраняются в хромосомах его потомков следующего поколения популяции. За исключением идентичных близнецов и бесполого размножения редко можно увидеть две точные копии какой-либо особи. Это генетическая характеристика популяции в целом, которая содержится в хромосомах особей популяции, что имеет первостепенное значение. Особи в популяции являются лишь средствами для коллективной передачи генетического профиля и «подопытными кроликами» для тестирования фитнеса.

Мы не знаем, какой именно признак или совокупность признаков отвечают за выживание и скрещивание особей, за производительность

личности в целом. Поэтому будем полагаться на «средних» особей. Если конкретная комбинация атрибутов неоднократно связана с высокой производительностью (потому что особи, содержащие эту комбинацию, имеют высокое значение пригодности), мы можем подумать, что сочетание этих признаков являются причиной наблюдаемой производительности. То же самое справедливо, когда определенная комбинация признаков многократно ассоциируется с низкой или всего лишь средней производительностью. Если же какая-либо комбинация атрибутов обладает и высокой, и низкой производительностью, то ее сложно объяснить в рамках поставленной задачи. Генетический алгоритм реализует этот интуитивно понятный подход к выявлению комбинаций атрибутов, которые отвечают за выполнение сложной нелинейной системы.

Генетический алгоритм предоставляет способ продолжения поиска в проблемной области путем тестирования новых различных точек, которые похожи на точки, уже показавшие приспособленность выше среднего.

При построении новой популяции с использованием имеющейся информации, мы должны помнить, что эта информация не совершенна. Существует вероятность, что особи с пригодностью выше среднего в последующих поколениях утратят свою значимость. Кроме этого, также возможно, что особь с низкой приспособленностью в данном поколении окажется, в конечном счете, связанной с оптимальным решением задачи. Таким образом, мы должны использовать имеющуюся информацию, чтобы направлять наши поиски, но мы должны также помнить, что имеющиеся в настоящее время данные о среде являются неполными.

Представление является ключевым вопросом в генетическом алгоритме, потому что генетические алгоритмы работают непосредственно с кодовым представлением проблемы. Обычный генетический алгоритм, работающий с символьными строками фиксированной длины, способен решить множество проблем. Тем не менее, использование строк заданной длины оставляет много вопросов нерешенными.

Для большинства проблем наиболее естественным представлением решения проблемы является иерархическая компьютерная программа, а не символьная строка заданной длины. Размер и вид иерархической компьютерной программы, которая позволит решить данную проблему, неизвестны заранее, поэтому программа должна иметь возможность изменения размера и вида.

1.5 Генетическое программирование

Многие, казалось бы, разные проблемы в искусственном интеллекте, символьной обработке и машинном обучении можно рассматривать как требующие компьютерной программы, вычисляющей некоторый требуемый результат в зависимости от входных параметров.

Процесс решения этих проблем можно сформулировать как поиск наиболее подходящей индивидуальной компьютерной программы среди всех возможных компьютерных программ. Пространство поиска состоит из всех возможных компьютерных программ, составленных из функций и терминальных символов, соответствующих проблемной области. Генетическое программирование предоставляет способ поиска этих наиболее подходящих индивидуальных компьютерных программ.

Генетическое программирование пробует решить проблему представления в генетических алгоритмах путем увеличения сложности адаптируемых структур. В частности, адаптируемые структуры в генетическом программировании являются общими иерархическими компьютерными программами, динамически изменяющими размер и вид.

В генетическом программировании популяции сотен или тысяч компьютерных программ генетически выведены. Эта селекция осуществляется с помощью принципа выживания сильнейших и воспроизводства наиболее приспособленных особей вместе с генетической рекомбинацией (скрещиванием) организмов путем применения операций,

подходящих для компьютерных программ. Компьютерная программа, которая решает (или приблизительно решает) определенную проблему может возникнуть из комбинации естественного отбора Дарвина и генетических операций.

Генетическое программирование начинается с генерации случайным образом выбранной начальной популяции компьютерных программ из функций и терминалов, соответствующих проблемной области. Функции могут быть стандартными арифметическими операциями, операциями программирования, математическими функциями, логическими функциями или предметно-ориентированными фикциями. В зависимости от конкретной задачи, компьютерная программа может работать с логическими значениями, целыми, вещественными или комплексными числами, векторами, символами. Создание этой начальной популяции в действительности «слепой» случайный поиск в пространстве проблемной области.

Каждая программа в популяции оценивается, насколько хорошо она выполняет свои задачи в проблемной среде. Эта оценка носит название меры пригодности. Ее вид зависит от проблемы.

Для многих задач пригодность естественно измерять ошибкой, погрешностью компьютерной программы. Чем ближе эта ошибка к нулю, тем лучше данная программа. Также приспособленность может быть комбинацией таких факторов, как корректность, экономность и бережливость.

Как правило, каждая компьютерная программа популяции отработает для нескольких значений входных параметров. Тогда пригодность будет считаться в виде суммы или среднего арифметического значений приспособленности всех входных параметров. Например, пригодность компьютерной программы может быть суммой абсолютной величины от разности вычисленного программой значения и корректного решения проблемы. Эта сумма может быть получена из выборки 50 различных входных значений программы.

За исключением случаев, когда проблема мала и проста, она не может быть легко решена путем «слепого» случайного поиска, компьютерные программы нулевого поколения будут иметь очень плохую пригодность. Тем не менее, некоторые особи в популяции будут немного пригоднее остальных. Эти различия в эффективности следует использовать в дальнейшем.

Принципы репродукции и выживания наиболее приспособленных особей и генетические операции половой рекомбинации (скрещивание) используются для создания нового поколения индивидуальных компьютерных программ из текущей популяции.

Операция репродукции включает в себя селекцию, пропорциональную значениям приспособленности, компьютерных программ из текущей популяции и позволяет отобранным особям выжить путем копирования в новую популяцию.

Генетический процесс полового скрещивания двух родителей – компьютерных программ – используется для создания новых потомков от родителей, выбранных пропорционально значениям пригодности. Программы-родители обычно имеют различный размер и форму. Программы-потомки составляются из подвыражений (поддеревьев, подпрограмм) родителей. Эти потомки, как правило, различаются размером и видом от своих родителей.

Интуитивно, если две компьютерные программы несколько эффективны в решении проблемы, то их части, возможно, тоже немного пригодны. Скрещивая случайно выбранные части относительно пригодных программ, мы можем получить новую компьютерную программу, которая даже лучше решает проблему.

После выполнения операций репродукции и скрещивания с текущей популяцией, популяция потомков (новое поколение) помещается в старую популяцию (прошлое поколение).

Каждая особь новой популяции компьютерных программ затем проверяется на пригодность, и процесс повторяется в течение многих поколений.

На каждом этапе этого параллельного, локально управляемого, децентрализованного процесса состоянием процесса будет являться только текущая популяция особей. Движущая сила этого процесса состоит только в наблюдении за пригодностью особей текущей популяции.

Данный алгоритм будет производить популяцию компьютерных программ, которые через много поколений, как правило, демонстрируют увеличение средней пригодности. Кроме того, эти популяции могут быстро и эффективно приспосабливаться к изменениям в окружающей среде.

Как правило, лучшая особь, которая появляется в любом по счету поколении, обозначается как результат генетического программирования.

Важной особенностью генетического программирования является иерархический характер производимых программ. Результаты генетического программирования по своей природе иерархичны.

Динамическая изменчивость также является важным признаком компьютерных программ, созданных генетическим программированием. Было бы трудно и неестественно пытаться заранее уточнить или ограничить размер и форму возможного решения. Более того, такая предварительная спецификация сужает пространство поиска решений и может исключить нахождение решения вообще.

Еще одной важной особенностью генетического программирования выступает отсутствие или сравнительно малая роль предварительной обработки входных данных и постобработки выходных значений. Входные параметры, промежуточные результаты и выходные значения обычно выражаются непосредственно в терминах естественной терминологии предметной области. Элементы функционального множества также естественны для проблемной области.

И наконец, структуры, подвергающиеся адаптации, активны. Они не являются пассивными кодировками решения проблемы. Вместо этого с учетом компьютера, на котором происходит запуск, программы в генетическом программировании – это активные структуры, способные выполниться в их текущем виде.

Парадигма генетического программирования является независимой от проблемной области. Это обеспечивает единый, унифицированный подход к проблеме нахождения решения в виде компьютерной программы.

1.6 Применение генетического программирования для решения задачи символьной регрессии

В обычном генетическом алгоритме и генетическом программировании в качестве адаптируемых структур выступает популяция особей из всего пространства поиска. Генетические методы отличаются от большинства других методов поиска тем, что они включают одновременный параллельный поиск с участием сотен или тысяч точек всего пространства поиска.

Отдельные адаптируемые структуры в генетическом программировании являются композицией функций. Размер, форма и содержание этих функций может быть динамически изменено в ходе выполнения процесса.

Множество возможных структур в генетическом программировании – это множество всех возможных композиций функций, которые могут быть составлены рекурсивно из функционального множества $F = \{f_1, f_2, ..., f_{N\_func}\}$ и множества терминальных символов $T = \{a_1, a_2, ..., a_{N\_term}\}$. Каждая конкретная функция f_i из функционального множества F принимает указанное число $z(f_i)$ аргументов. То есть функция f_i имеет арность равную $z(f_i)$.

Функционально множество может состоять из следующих элементов:

а) арифметические операции;

б) математические функции;

в) логические операции;

г) условные операторы;

д) операторы циклов;

е) другие проблемно-ориентированные функции.

Терминальными символами обычно являются либо переменные «атомы» (представляющие входы, сенсоры, датчики или переменные состояния некоторой системы), либо постоянные «атомы» (такие как число 3 или логическая константа NIL). Иногда в качестве терминальных символов также берутся функции, не принимающие явных аргументов, реальная функциональность таких функций заключается в создании побочных эффектов для состояний системы.

Адаптационные структуры в генетическом программировании отличаются от структур, подвергающихся адаптации, в обычном генетическом алгоритме, который оперирует строками. В обычном генетическом алгоритме структуры представляют одномерные линейные строки фиксированной длины. В вариации Стивена Смита (1980, 1983) обычного генетического алгоритма адаптационные структуры являются одномерными линейными строками переменной длины.

В генетическом программировании терминальное и функциональное множества должны быть выбраны так, чтобы они удовлетворяли требованиями замкнутости и достаточности.

Свойство замкнутости требует, чтобы каждая функция из функционального множества могла принять в качестве аргумента любое значение и тип данных, которые могут быть возвращены любой функцией из функционального множества, а также любой элемент терминального множества. То есть каждая функция из функционального множества должна быть четко определена и замкнута для любой комбинации аргументов, с которыми она может встретиться.

В обычных программах арифметические операции с численными переменными иногда не определены (например, деление на нуль). Многие обычные математические функции иногда не определены (например,

логарифм нуля). Кроме того, возвращаемые некоторыми математическими функциями значения могут входить в список неприемлемых типов данных для проблемной области (например, квадратный корень или логарифм отрицательного числа). Также логическое значение, обычно возвращаемое условным оператором, как правило, не принимается в качестве аргумента арифметическими функциями.

Может показаться, что соблюдение свойства замкнутости для обычной компьютерной программы невозможно или приведет к очень сложной и ограниченной синтаксической структуре. На самом деле это не так. Замкнутость может быть достигнута простым способом для подавляющего большинства задач просто путем тщательной обработки небольшого количества ситуаций.

Если арифметическая операция деления получит в качестве второго аргумента число 0, то свойство замкнутости будет нарушено. Один простой подход гарантирует замкнутость – определение защищенной функции деления. Защищенная функция деления принимает два аргумента и возвращает 1 при попытке деления на 0 (включая деление 0 на 0), а в других случаях возвращает нормальное частное.

Свойство замкнутости желаемо, но не абсолютно необходимо. Если свойство замкнутости не превалирует, то существуют альтернативные способы: исключение особей с нежелательным результатом или система штрафов для таких структур. Вопрос обработки таких ситуаций не уникален для генетических методов, а широко обсуждается в связи с другими алгоритмами. Удовлетворительного решения этой проблемы пока не существует, поэтому мы будет соблюдать свойство замкнутости.

Свойство достаточности требует, чтобы терминальное и функциональное множества могли выразить решение проблемы. Пользователь генетического программирования должен убедиться, что композиция функция и термов приведет к решению проблемы.

Этап определения переменных, у которых достаточно возможностей решить определенную проблему, является общим практически для каждой проблемы в науке.

В зависимости от проблемы этот шаг идентификации может быть очевидным, а может потребовать глубокого понимания предметной области.

Этапы определения примитивных функций и терминальных символов в генетическом программировании эквивалентны аналогичным необходимым этапам в других парадигмах машинного обучения. Эти два шага часто явно не определяются, обсуждаются или признаются исследователями других парадигм. Причиной этого упущения может быть то, что исследователь считает выбор примитивных функций и терминалов присущим формулировке задачи. Такой взгляд особенно понятен, если ученый фокусируется лишь на одном конкретном типе проблемы специфической области.

1.6.1 Начальные структуры

Начальные структуры в генетическом программировании состоят из особей исходной популяции, каждая из которых представляет решение проблемы в виде польской записи.

Создание каждого выражения начальной популяции выполняется в виде дерева со случайно выбранным корнем и упорядоченными ветвями, представляющего данную польскую нотацию.

Начинаем со случайного выбора одной функции из функционального множества F, которая станет корнем дерева. Мы ограничиваем выбор функциональным множеством, поскольку нам необходимо создать иерархическую структуру, а не вырожденную структуру, состоящую из одного терминального символа.

Рисунок 1 демонстрирует начало создания случайного дерева программы. Функция сложения (с двумя аргументами) была выбрана случайно из функционального множества F в качества корня дерева.

Рисунок 1-Создание корня дерева

Когда узел дерева помечается функцией *f* из *F*, то *z(f)* линий, где *z(f)* – количество аргументов функции *f*, выходит из этого узла. Затем для каждой такой линии случайно выбирается элемент из объединенного множества $C = F \cup T$, функций и терминалов, для конечной точки – другого узла – этой линий.

Если в качестве узла была выбрана функция, то дальнейшее создание дерева продолжается рекурсивно так, как было описано выше. Например, на рисунке 2, функция умножения была выбрана из множества $C = F \cup T$ в качестве внутреннего узла (номер два) для конечной точки левой линии корневого узла (функция сложения, номер один). Функция умножения принимает два аргумента, поэтому из второй вершины выходят две линии.

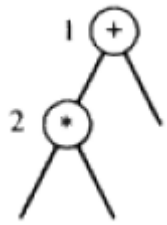

Рисунок 2-Выбор внутреннего узла

Если в качестве любого узла выбирается терминальный символ, то этот узел становится листом, и процесс создания поддерева для этой вершины прекращается. Например, на рисунке 3 терминальный символ *A* был выбран в качестве вершины для левой линии функции умножения. Аналогичным образом, терминалы *B* и *C* стали вершинами двух правых линий функций умножения и сложения соответственно. Этот процесс продолжается рекурсивно слева направо, пока все дерево не создано, как показано на рисунке 3.

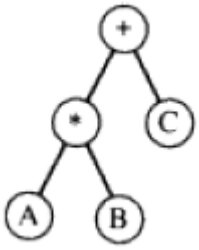

Рисунок 3-Созданное дерево программы

Данный процесс генерации может быть реализован различными способами, приводящими к получению случайных начальных деревьев разного размера и вида. Два основных метода называются «полным» методом и «растущим» методом. Длина дерева определяется, как длина самого длинного пути от корня к листу.

Полный метод создания начальной популяции состоит в генерации дерева, у которого длина каждого пути от листа к корню равна указанной максимальной глубине. Такой вид дерева можно получить путем ограничения выбора функциональным множеством для вершин, у которых глубина меньше заданной. А затем ограничить выбор только терминальным множеством для вершин с максимальной глубиной.

Растущий метод генерации начальной популяции состоит в создании деревьев различной формы. Длина пути от листа до корня не больше заданной максимальной глубины. Это достигается путем случайного выбора внутреннего узла из множества $C = F \cup T$, при этом длина пути от корня к узлу меньше максимальной глубины. А вершины дерева, чья длина равна максимальной, становятся листами и выбираются из терминального множества.

Лучший результат для широкого диапазона проблемы дает объединенный метод, при котором начальная популяция создается чередованием полного и растущего методов. В генетическом программировании мы, как правило, не знаем заранее (или не хотим указывать) размер и форму решения. Объединенный метод генерации деревьев создает широкое разнообразие деревьев различного размера и вида.

Объединенный метод генерации – это смешанный метод, включающий и растущий, и полный методы. Данный способ состоит из создания равного количества деревьев с глубиной, которая находится в интервале от 2 до максимальной заданной глубины. Например, если максимальная глубина равна 6, то 20% деревьев будут иметь глубину 2, 20% деревьев глубину 3, и так далее до глубины 6. Затем для каждого значения глубины 50% деревьев создаются полным методом, а остальные 50% растущим методом.

Заметим, что у всех деревьев, созданных полным методом с заданной глубиной, длина пути от корня к листу одинаковая, равная максимальной глубине, и поэтому эти деревья имеют одинаковую форму. В отличие от этого, у всех деревьев, полученных растущим методом с данным значением глубины, ни один путь от корня дерева до листа не превышает максимальной глубины. Поэтому эти деревья значительно отличаются друг от друга по виду даже при одинаковой максимальной глубине.

Таким образом, объединенный метод создает деревья с большим разнообразием размеров и видов.

Повторяющиеся особи в начальной популяции непродуктивны: они тратят вычислительные ресурсы и приводят к нежелательному сокращению генетического разнообразия популяции. Поэтому желательно, но необязательно, предотвратить появление дубликатов в начальной случайно созданной популяции. В генетическом программировании вероятность появления повторяющихся особей в начальной популяции особенно высока, когда деревья малы. Таким образом, каждое созданное выражение проверяется на уникальность перед добавлением в популяцию. Если новое выражение – дубликат, то процесс повторяется до создания уникальной особи. Иногда (например, для маленьких деревьев) мы должны подставить большее по размеру дерево во время процесса генерации, если исчерпано множество возможных деревьев данного размера.

Разнообразие популяции – это доля особей, у которых нет точной копии во всей популяции. Если выполняется проверка на дублирование при

создании особей, то разнообразие начальной популяции равно 100%. В последующих поколениях появление одинаковых особей при использовании репродукции является неотъемлемой частью генетического процесса.

В противоположность этому в обычном генетическом алгоритме, работающем со строками символов фиксированной длины, каждый из символов строки начальной популяции обычно создается с помощью применения бинарного генератора случайных чисел. Например, бинарные строки длиной 453, используемые Джефферсоном и другими (1991), создаются с помощью двоичного генератора случайных чисел, имеющего пространство поиска размером 2453 (т.е. около 10137). Было бы необычно получить дубликаты среди всего лишь 65 536 индивидуальных строк популяции, когда пространство поиска размером 10137. Поэтому в обычных генетических алгоритмах, как правило, не проводится проверка одинаковых особей.

1.6.2 Пригодность

Приспособленность является движущей силой естественного отбора по Дарвину, и, следовательно, обычных генетических алгоритмов и генетического программирования.

В природе пригодность – это вероятность того, что особь доживет до репродукционного возраста и воспроизведется. Данный показатель может учитываться при расчете числа потомков. В искусственном мире математических алгоритмов мы оцениваем пригодность каким-либо способом, а затем используем значение приспособленности для контроля применения операций, изменяющих структуры в нашей искусственной популяции.

Пригодность может быть вычислена при помощи различных методов, явных и неявных.

Наиболее распространенным подходом вычисления пригодности является создание определенной меры приспособленности для каждой особи

популяции. Данный подход используется подавляющим большинством обычных генетических алгоритмов. Каждой особи популяции присваивается скалярное значение пригодности при помощи некоторой четко и явно определенной процедуры оценки.

Пригодность также может быть вычислена путем совместной эволюции (коэволюции), при которой пригодность игровой стратегии определяется применением этой стратегии против всей популяции (или отобранного числа особей), ведущей противоположную стратегию.

Тот факт, что особи существуют и выживают в популяции, а также успешно воспроизводятся, может свидетельствовать об их приспособленности (как это происходит в природе). Такое неявное определение пригодности часто используется в научных исследованиях (Ray 1990; Holland 1990, 1992). Однако, на данный момент, мы сосредоточимся на более общей ситуации, когда приспособленность вычисляется явно.

Существует 4 меры пригодности:

а) исходная пригодность;

б) стандартизованная пригодность;

в) отрегулированная пригодность;

г) нормированная пригодность.

1.6.2.1 Исходная пригодность

Исходная приспособленность – это измерение пригодности, сформулированное в естественной терминологии проблемы. Например, исходной пригодностью для символьной регрессии является погрешность полученного результата вычисления функции относительно желаемого. Чем меньше погрешность, тем лучше.

Наиболее общим определением исходной приспособленности является ошибка. То есть исходная пригодность отдельного выражения – это сумма расстояний для всех значений независимых переменных между полученным результатом для конкретного значения свободной переменной и желаемым

результатом для этой же переменной. Выражение может быть логическим, целочисленным, вещественным, комплексным, вектором или символьным значением.

Если выражение целочисленное или вещественное, то сумма дистанций вычисляется в виде суммы абсолютных значений разности полученного и необходимого результатов. Когда исходной пригодностью является ошибка, исходная приспособленность *r(i, t)* отдельного выражения *i* в популяции размера *M* любого поколения шага *t* вычисляется по формуле 21.

$$r(i,t) = \sum_{j=1}^{N} |S(i,j) - C(i,j)|, \quad (21)$$

где *S(i, j)* – вычисленное значение выражения *i* для номера *j* значения переменной (всего *N* случаев), *C(j)* – необходимое значение для *j*-ого значения свободной переменной.

Если выражение логическое или символьное, то сумма расстояний будет эквивалентна числу несоответствий. Если выражение комплексное или векторное, то сумма расстояний получается отдельно для каждого компонента структуры.

Если выражение вещественное или целочисленное, то квадратный корень суммы квадратов расстояний может быть использован в качестве альтернативы для измерения приспособленности.

Т.к. исходная пригодность основывается на естественной терминологии проблемы, чем лучше значение, тем оно меньше (когда в качестве меры приспособленности берется ошибка) или больше (когда используется достигнутая выгода).

1.6.2.2 Стандартизованная пригодность

Стандартизованная приспособленность *s(i, t)* пересчитывает исходную пригодность так, чтобы наименьшее численное значение всегда было лучшим результатом. Например, для проблемы оптимального управления, заключающейся в минимизации расходов, меньшее значение исходной

пригодности лучше. Также, если проблема заключается в минимизации ошибки, то меньшее значение исходной приспособленности лучше.

Для таких проблем, когда меньшее значение исходной пригодности является лучшим, стандартизованная пригодность равняется исходной пригодности (формула 22).

$$s(i, t) = r(i, t) \quad (22)$$

Удобно и желательно сделать лучшим значением стандартизованной приспособленности 0.

Если для определенных проблем большее значение исходной пригодности лучше, то стандартизованная приспособленность вычисляется из исходной пригодности. Стандартизованная пригодность равняется разности максимально возможного значения исходной пригодности и данной пригодности (формула 23).

$$s(i, t) = r_{max} - r(i, t) \quad (23)$$

Если верхняя граница неизвестна, а большее значение исходной приспособленности является лучшим, то отрегулированная и нормализованная пригодности могут быть вычислены непосредственно из исходного фитнеса.

1.6.2.3 Отрегулированная пригодность

Кроме того для решения проблемы используется оптимальная регулировка приспособленности. Отрегулированная пригодность $a(i, t)$ вычисляется из стандартизованной пригодности по формуле 24:

$$a(i,t) = \frac{1}{1+s(i,t)}, \quad (24)$$

где $s(i, t)$ – стандартизованная пригодность для особи i поколения t.

Отрегулированная приспособленность находится в промежутке от 0 до 1. Чем больше его значение, тем лучше эта особь в популяции.

Необязательно использовать отрегулированную пригодность в генетическом программировании, но это облегчает решение задачи. Отрегулированная приспособленность хорошо преувеличивает важность

малых различий в значениях стандартизованной пригодности. Таким образом, по мере улучшения популяции большой акцент делается на небольшие различия, что позволяет увидеть разницу между хорошей и очень хорошей особью. Это преувеличение особенно велико, если стандартизованная приспособленность достигает нуля, когда лучшее решение проблемы найдено. Например, если стандартизованная пригодность находится в промежутке от 0 (лучший) до 64 (худший), отрегулированная пригодность двух плохих особей, оцениваемых 64 и 63, будет 0.0154 и 0.0159 соответственно. А для особей с оценкой 4 и 3 отрегулированная приспособленность равна 0.20 и 0.25 соответственно. Данный эффект слабеет (но все равно значителен), когда лучшее значение стандартизованной пригодности не может быть определено.

Необходимо заметить, что для других методов выбора, отличающихся от пропорциональных значениям приспособленности, отрегулированная функция неуместна и не используется.

1.6.2.4 Нормализованная пригодность

Если метод селекции основывается на пропорциональности приспособленности, то понятие нормализованной пригодности также необходимо.

Нормализованная пригодность *n(i, t)* вычисляется с помощью значения отрегулированной приспособленности по формуле 25:

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)} \quad (25)$$

У нормализованной пригодности есть три характеристики:

а) Она находится в промежутке от 0 до 1.

б) Чем больше ее значение, тем лучше особь популяции.

в) Сумма значений нормализованной приспособленности равна 1.

1.6.3 Основные операции изменения структур

В этом разделе описываются две основные операции, используемые для изменения адаптационных структур в генетическом программировании:

а) репродукция Дарвина;

б) скрещивание (половая рекомбинация).

1.6.3.1 Репродукция

Операция воспроизведения для генетического программирования является основной движущей силой естественного отбора Дарвина и выживания наиболее приспособленных особей.

Репродукция состоит из двух этапов. Во-первых, одна особь выбирается из популяции в соответствии с каким-либо способом отбора, основанным на пригодности. Во-вторых, выбранная особь копируется без изменений, из текущей популяции в новую популяцию (т.е. новое поколение).

Существует множество различных методов отбора, основанных на приспособленности. Наиболее популярным является селекция, пропорциональная значениям пригодности.

Если $f(s_i(t))$ – это приспособленность особи s_i в популяции поколения t, то при пропорциональном пригодности отборе вероятность того, что особь s_i будет скопирована в следующее поколение, как результат операции воспроизведения, будет вычисляться по формуле 26.

$$\frac{f(s_i(t))}{\sum_{j=1}^{M} f(s_j(t))}, \quad (26)$$

где $f(s_i(t))$ – это нормализованная пригодность $n(s_i(t))$.

Среди альтернативных методов селекции можно выделить турнирный отбор и селекцию по рангу. При ранговом отборе выбор основывается на ранге (нечисловом значении) значений приспособленности особей в популяции. Данный метод уменьшает потенциально доминирующие эффекты сравнительно высокой пригодности особей популяции путем

создания предсказуемого, ограниченного отбора таких лиц. В то же время ранговая селекция преувеличивает разницу между близко находящимися значениями приспособленности.

В турнирном отборе определенное количество особей (обычно два) выбираются случайным образом из популяции. Затем из них выбирается одна особь с лучшей приспособленностью.

Заметим, что родитель остается в популяции на протяжении всей селекции. То есть, разрешен повторный выбор. Родители могут быть выбраны и, в общем случае, выбираются более одного раза для воспроизведения в текущем поколении.

1.6.3.2 Скрещивание

Операция скрещивания (половая рекомбинация) в генетическом программировании изменяет популяцию путем создания нового потомства, которое состоит из частей, взятых от каждого родителя. Скрещивание начинается с выбора двух особей-родителей и заканчивается созданием двух особей-потомков.

Первый родитель выбирается из популяции таким же способом отбора, основанным на значении пригодности, как и в операции репродукции, т.е. вероятность выбора первого родителя равна его нормированной пригодности. Второй родитель выбирается аналогично первому.

В начале операции скрещивании случайным образом выбирается одна случайная вершина у каждого дерева-родителя. Эта вершина становится точкой скрещивания для этих двух родителей. Следует отметить, что особи-родители, как правило, имеют разный размер.

Фрагмент скрещивания для определенного родителя – это поддерево, корнем которого является точка скрещивания этого родителя, состоящее из всего поддерева родителя, находящего ниже точки скрещивания. Это поддерево иногда может состоять из одного терминального символа.

Первый потомок получается путем удаления фрагмента скрещивания у первого родителя, а затем вставки фрагмента скрещивания второго родителя в точку скрещивания первого родителя. Второй потомок получается симметричным образом.

Если в точке пересечения одного из родителей находится терминальный символ, то поддерево второго родителя вставляется на место терминала в первом родителе (вводя тем самым поддерево вместо одной точки терминала), а терм первого родителя вставляется на место расположения поддерева во втором родителе. Это часто будет давать эффект создания потомства большей глубины.

Если терминальные символы расположены на обеих точках скрещивания, то операция скрещивания просто поменяет местами эти терминалы. Эффект скрещивания в данном случае равен эффекту узловой мутации. Таким образом, мутация иногда является неотъемлемой частью работы кроссинговера.

1.6.4 Второстепенные операции изменения структур

В дополнение к двум основным генетическим операциям репродукции и скрещивания в генетическом программировании есть необязательные вторичные операции: мутация и перестановка.

1.6.4.1 Мутация

Операция мутации вносит случайные изменения в структуры популяции. Мутация – это бесполая операция, изменяющая только одно дерево.

Операция начинается с выбора вершины дерева случайным образом. Эта точка мутации может быть внутренним узлом дерева (функцией) или внешним узлом – листом (терминальным символом). Мутация удаляет значение выбранного узла и все, что находится ниже его по уровню, а затем вставляет случайным образом сгенерированное поддерево в точку мутации.

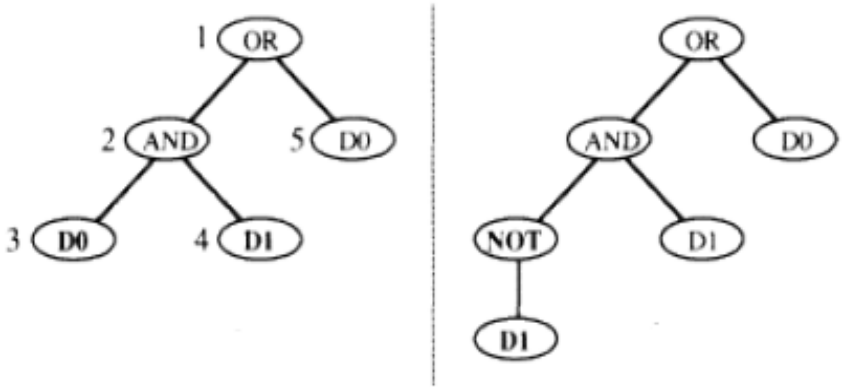

Рисунок 4-Результат Мутации дерева

Для деревьев используются следующие операторы мутации: узловая, усекающая и растущая.

Узловая мутация выполняется следующим образом:

а) Выбрать случайным образом узел, подлежащий мутации, определить его тип.

б) Случайным образом выбрать из соответствующего множества вариантов узлов узел, отличный от рассматриваемого узла.

в) Поменять исходный узел на выбранный узел.

Усекающая мутация производится так:

а) Выбирается узел.

б) Случайным образом выбирается терминальный символ из заданного множества.

в) Обрезается ветвь узла мутации.

г) Вместо обрезанной ветви помещается выбранный терминальный символ.

Растущая мутация выполняется следующим образом:

а) Случайным образом определяется узел мутации.

б) Если узел нетерминальный, то необходимо отсечь ветви, исходящие из него, иначе выбрать другой узел.

в) Вычислить размер (сложность) остатка дерева.

г) Вместо отсеченного дерева вырастить случайным образом новое дерево так, чтобы размер нового построенного дерева не превышал заданный порог.

1.6.4.2 Перестановка

Операция перестановки – это обобщение инверсии для генетического программирования. Происходит случайный выбор внутреннего узла дерева, т.е. функции. Если функция в выбранной вершине имеет *k* аргументов, то перестановка выбирается случайным образом из набора *k!* возможных замен аргументов. После этого аргументы функции меняются в соответствии с выбранной перестановкой (рисунок 5).

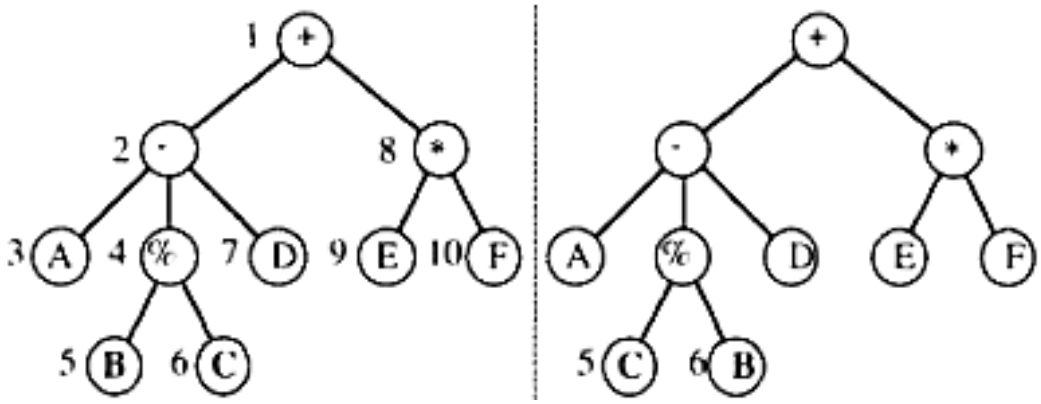

Рисунок 5-Результат перестановки

Стоит отметить, что если выбранная функция коммутативна, то перестановка ее аргументов никак не повлияет на результат.

2 Практическая часть

Была поставлена задача решения символьной регрессии методом генетического программирования. Поиск готовых образцов не дал результатов, найденные программы выдавали результат, точность которого была меньше необходимой. Поэтому было принято решение о разработке собственной программы, предоставляющей необходимый результат.

2.1 Выбор языка программирования

В качестве языка программирования был выбран Python.

Python позволяет использовать эффективные высокоуровневые структуры данных и предлагает простой, но эффективный подход к объектно-ориентированному программированию. Сочетание изящного синтаксиса, динамической типизации в интерпретируемом языке делает Python идеальным языком для написания сценариев и ускоренной разработки приложений в различных сферах и на большинстве платформ.

Интерпретатор Python и разрастающаяся стандартная библиотека находятся в свободном доступе в виде исходников и двоичных файлов для всех основных платформ на официальном сайте Python http://www.python.org и могут распространяться без ограничений.

Python даёт возможность писать компактные и читабельные программы. Программы, написанные на Python, отличаются большей краткостью, чем эквивалентные на C, C++ или Java, по нескольким причинам:

а) высокоуровневые типы данных позволяют вам выражать сложные операции в одной инструкции;
б) группировка инструкций выполняется отступами, а не операторными скобками;
в) нет необходимости в описании переменных или аргументов.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты – динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули.

2.2 Особенности программы

2.2.1 Структура данных

Выборка значений свободных переменных представлена в программе списком, элементы которого являются словарями. Каждый словарь в свою очередь состоит из ключей, обозначающих название переменных, и значений, равных значениям независимых переменных. Значения зависимых переменных собраны в список. Отметим, что важен порядок расположения свободных и зависимых значений, т.к. *i*-ому элементу первого списка соответствует *i*-ый элемент второго списка.

Особи популяции – деревья – представлены собственным классом. Его полями являются структура дерева, отображение дерева и пригодность. Структура дерева – это список списков номеров потомков, т.е. определенная позиция в списке означает номер вершины дерева, которой соответствует список потомков. Отображение дерева представлено в виде словаря, ключом которого является номер вершины, а значением – функция или терминальный символ. Поле пригодности – это вещественное число, обозначающее суммарную квадратичную ошибку определенного дерева. Класс дерева также содержит необходимые для проведения генетических операций поля: позиция скрещивания (номер вершины, которая вместе со своим поддеревом будет заменена при скрещивании) и потомок (дерево,

полученное в результате скрещивания). Данный класс включает в себя функцию нахождения глубины дерева, методы нахождения, удаления и добавления поддерева, необходимые для половой рекомбинации, а также функции замены у дерева случайно выбранного терминального символа или функции на другой терм или функцию, нужные для проведения операции мутации.

Для создания начальной популяции был создан отдельный класс. Полями класса являются глубина дерева (целое число) и созданная особь (дерево). Методы класса позволяют создавать деревья «полным» и «растущим» способами определенной глубины. При создании листов дерева происходит проверка на наличие переменных в данном дереве. Если переменных нет, то значением определенного листа становится переменная, случайным образом выбранная из множества всех переменных. Если в дереве уже есть переменные, то значение данного листа выбирается с равной долей вероятности из множества переменных и констант.

Оператор репродукции представлен собственным классом. Он включает в себе такие поля, как начальная популяция (список деревьев), выбранная популяция (список деревьев) и значения сумм отрегулированной пригодности для каждого набора значений свободных переменных (список действительных чисел). Данный класс также содержит методы получения суммарной отрегулированной пригодности начальной популяции, получения значения пригодности для отдельного дерева и метод отбора наиболее приспособленных особей. Также есть функция нахождения отрегулированной пригодности на основе стандартизованной пригодности и метод проверки наличия в дереве нетерминальных элементов (если в дереве нет функций, то такое дерево не должно попасть в следующую популяцию).

Для проведения операции скрещивания был создан отдельный класс, включающий в себя следующие поля: первый родитель, второй родитель, первый потомок и второй потомок, представленные в виде деревьев. Метод скрещивания случайным образом выбирает позиции у деревьев-родителей,

проверяет функции, соответствующие вершинам этих позиций, на арность, и если она совпадает, то меняет местами определенные поддеревья, в противном случае поиск вершин для замены продолжается рекурсивно. Если глубина рекурсии достигнет своего максимального значения равного 10, происходит выход из данного метода без создания потомков деревьев.

2.2.2 Структура программы

Производится выборка начальных данных: свободных и зависимых значений переменных.

Первоначально создаем начальную популяцию мощностью в 500 особей. Глубина каждого начального дерева не больше 5. Деревья генерируются растущим и полным методами напополам.

Далее выполняется репродукция. Для каждого дерева вычисляется значение его пригодности. В качестве меры приспособленности используется квадратичная ошибка. Это значение нормализуется, и на его основе выбираются наиболее подходящие особи.

На следующем шаге выполняется цикл до момента, когда количество особей нового поколения не станет равным 500. Новое поколение формируется из потомков и лучших особей предыдущего поколения. С вероятностью 0.9 случайным образом выбираются деревья-родители из особей, отобранных на шаге репродукции, и скрещиваются. Если размер потомков не превышает заданный (максимальная глубина дерева равна 16), то полученные деревья помещаются в следующее поколение. С вероятностью 0.1 в новую популяцию добавляется случайно выбранное из наиболее пригодных особей дерево текущего поколения.

После выполняется операция мутации. С вероятностью 0.01 каждая особь может мутировать с помощью узловой, растущей или усекающей мутации.

Последний этап состоит в проверке значения приспособленности каждой особи. Если значение меньше заданной точности, то рассматриваемое

дерево становится одним из решений задачи. В случае если ни одна из особей популяции не достигла такого результата, возвращаемся к стадии репродукции и начинаем сначала с новым поколением. Максимальное количество итераций равно 500.

Из всех полученных решений выбирается функция с минимальной квадратичной ошибкой для второстепенных данных. Даже если пригодность этого решения для основного множества значений не является наилучшей, решающее значение имеет наибольшая схожесть поведения тестовой и полученной функций, поэтому конечное решение выбирается на основе минимальной квадратичной ошибки для второстепенных данных.

2.3 Практические результаты

Для проверки работы программы был проведен эксперимент. Условия эксперимента: определим тестовую функцию, основное множество значений свободных и зависимых переменных, второстепенное множество значений свободных и зависимых переменных. Эксперимент будет считаться успешным, если суммарная квадратичная ошибка найденной программой функции для основной выборки данных будет меньше 1, и суммарная квадратичная ошибка для второстепенной выборки данных также будет меньше 1. Эксперимент будет считаться неуспешным, если многократный запуск программы (не менее 10 раз) не приведет к нахождению функциональной зависимости с требуемым отклонением.

Эксперимент будет выполняться на тестовой машине с процессором Intel(R) Core(TM) i7-3517U 1.90GHz и ОЗУ 4.0 ГБ DDR3.

Вид тестовой функции представлен формулой 27:

$$sin(x * x) * |x + 5.3| + x \quad (27)$$

Множество независимых переменных состоит из переменной *x*, принимающей следующие значения: {-1, -0.3, -0.2, 0, 0.1, 0.5, 0.6, 1, 1.5, 2}.

Ожидаемые зависимые от функции значения: {4.301267, 0.2033, 0.019941, 0, 0.153999, 1.9349, 2.6784, 6.301, 6.7908977, -3.524658}.

Множество второстепенных значений свободной переменной: {1.2, 1.7, 1.9, 0.2, -0.5, -0.8}.

Множество второстепенных значений зависимой переменной: {7.644479000000, 3.442627500000, -1.3505534155622465, 0.4199413380264879, 0.934942963676, 2.8428921923105914}.

В результате работы программы были получена функция с суммарной квадратичной ошибкой по основным данным равной 0.999405090213, а по второстепенным данным равной 0.34031866036260605. Функция представлена формулой 28, а ее график можно увидеть на рисунке 6:

((((x / Sin [Sin [Sin [Sin [Sin [Sin [Abs [x]]]]]]]) + (((Sin [Sin [Cos [x]]] /
/ (-0.0956802175023502)) + 6.150516695690124) +
+ Sin [Sin [Sin [Sin [Abs [x]]]]])) *
* Sin [Sin [Sin [Sin [Sin [Cos [Abs [x]]]]]]]) +
+ (((Cos [Sin [Sin [Cos [x]]]] / Log [Sin [Cos [Sin [Abs [x]]]]]) +
+ Sin [Sin [Sin [Sin [Sin [Sin [Sin [Cos [Abs [x]]]]]]]]] + 5.472573)) +
+ Sin [Sin [Sin [Sin [Sin [Sin [Cos [x]]]]]]]) * Abs[x] (28)

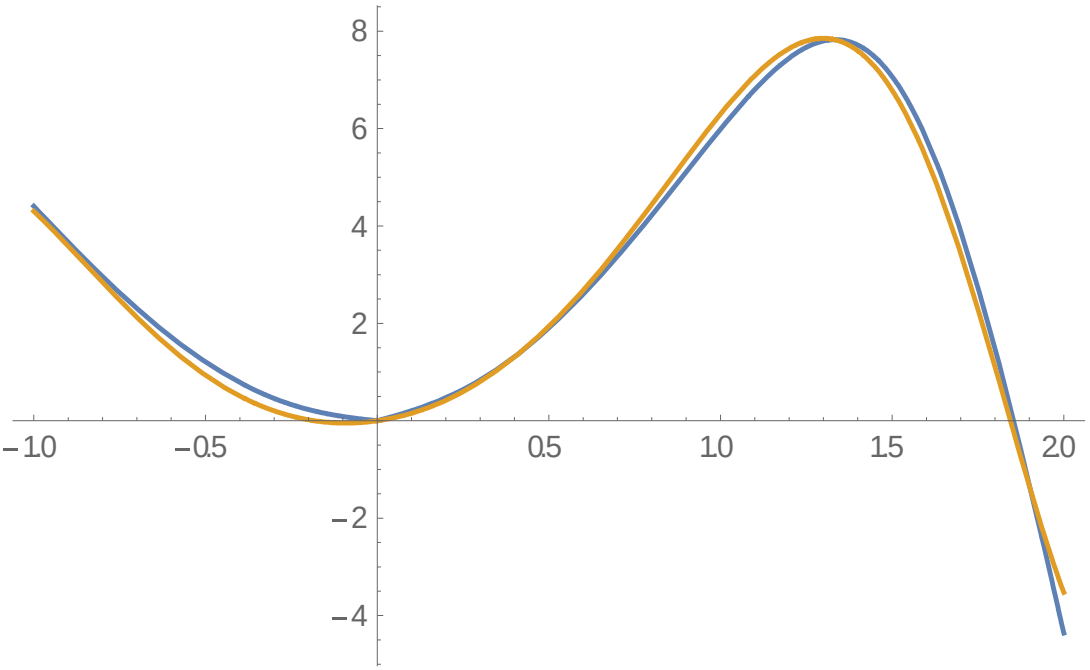

Рисунок 6-График полученной и тестовой функций

На представленном графике (рисунок 6) видно, что полученная функция достаточно точно совпадает с заданной функцией в точках основной и второстепенной выборок данных, т.е. ведет себя как тестовая функция.

Следует отметить, что на результат работы программы сильно влияет выборка значений независимых переменных. Поэтому достаточность входных данных для нахождения функции является самостоятельной задачей и требует отдельного решения.

Также можно заметить, что полученная программой функция имеет достаточно сложный вид, что создает неудобства для ее практического применения.

Данное решение задачи символьной регрессии методом генетического программирования дает результат достаточной точности, которую можно улучшить с помощью других способов аппроксимации функции, например метода наименьших квадратов.

ЗАКЛЮЧЕНИЕ

Было проведено исследование решения задачи символьной регрессии методом генетического программирования.

Рассмотрение генетического алгоритма дало представление об основах эволюционных вычислений и понимание невозможности решения проблемы обычными генетическими алгоритмами ввиду их работы со строками фиксированной длины.

Поэтому было подробно изучено генетическое программирование, сформировано понимание о структуре представления данных в программе, а также операций, применение которых способно привести к решению проблемы.

Далее была разработан алгоритм решения задачи установления функциональной зависимости данных на основе генетического программирования. Подробно рассмотрены генетические операторы, их влияние на промежуточный и конечный результаты работы, и на основании проделанной работы выбраны наиболее подходящие операторы репродукции, скрещивания и мутации.

По разработанному алгоритму была реализована программа. Проверено на практике выполнение и результат всех этапов алгоритма.

Затем было проведено тестирование работы программы для тестовой функции и выборки данных. Результат тестирования продемонстрировал успешность решения задачи установления функциональной зависимости с помощью генетического программирования.

Полученное решение в виде суперпозиции функций дает результат достаточной точности, которую можно улучшить с помощью других способов аппроксимации функции, например метода наименьших квадратов.

Также следует отметить, что на результат работы программы сильно влияет выборка независимых значений. Поэтому достаточность входных

данных для нахождения функции является самостоятельной задачей и требует отдельного решения.

Достоинством использования метода генетического программирования для решения задачи установления функциональная зависимости данных является точность совпадения полученной функции в точках выборки. Если у нас есть дополнительная информация о функции кроме ее точек, мы легко можем использовать ее при применении операции репродукции. Концептуальная простота генетического программирования также является важным преимуществом его практического применения.

В качестве недостатка можно выделить отсутствие эффективных критериев окончания работы программы. Мы не можем предсказать, приведет ли дальнейшее выполнение программы к результату или нет, возможно, мы сможем получить более точный результат, чем уже найденный. Также выполнение программы требует значительных вычислительных ресурсов (увеличение мощности популяции и максимальной глубины дерева) для решения задач высокой точности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Метод наименьших квадратов [Электронный ресурс]. URL: http://teachmen.ru/methods/phys_prac9.php
2. Регрессионный анализ [Электронный ресурс]. URL: http://www.machinelearning.ru/wiki/index.php?title=Регрессионный_анализ
3. Эволюционные вычисления [Электронный ресурс]. URL: http://www.intuit.ru/studies/courses/14227/1284/info
4. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, reading, MA, 1989.
5. Holland J.P. Adaptation in Natural and Artificial Systems.An Introductionary Analysis With Application to Biology? Control and Artificial Intelligence. University of Michigan, 1975.
6. Koza, John R. Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, USA, 1992.
7. Langdon, William B. Genetic programming and data structures. Department of Computer Science, University College London, 1996.
8. Mitchell Melanie. An introduction to Genetic Algorithms. MIT Press, Cambridge, London, 1998.
9. The Python Tutorial [Электронный ресурс]. URL: https://docs.python.org/2/tutorial/index.html
10. Xinjie Yu, Mitsuo Gen. Introduction to Evolutionary Algorithms. Springer, London Limited 2010.