Лабораторная работа №8

МЕТОД АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕКСТОВОЙ ИНФОРМАЦИИ

Цель работы: научиться строить автоматические грамматики на основе обучающей выборки.

Порядок выполнения работы

1. Изучение теоретической части работы.
2. Реализация алгоритма автоматического синтеза текстовой информации с помощью синтаксического метода.
3. Оформление отчета по лабораторной работе.

Исходные данные: заданное множество терминальных цепочек.

Выходные данные: грамматика, способная автоматически порождать заданные цепочки, а также бесконечное множество цепочек, схожих по структуре с исходными.

Используя лингвистическую терминологию, процедуру получения решений с помощью обучающей выборки легко интерпретировать как задачу получения грамматики из множества выборочных предложений. Эта процедура называется грамматическим выводом и играет важную роль в изучении синтаксического распознавания образов в связи с ее значением для реализации автоматического обучения. На рис. 1 представлена модель вывода цепочечных грамматик.

Рис. 1. Модель вывода цепочечных грамматик

Задача, показанная на рисунке 1, заключается в том, что множество выборочных цепочек подвергается обработке с помощью адаптивного обучающего алгоритма, представленного блоком. На выходе этого блока в конечном счете воспроизводится грамматика G, согласованная с данными цепочками, т. е. множество цепочек $\{x_i\}$ является подмножеством языка $L(G)$. Пока ни одна из известных схем не в состоянии решить эту задачу в общем виде. Вместо этого предлагаются многочисленные алгоритмы для вывода ограниченных грамматик. Рассмотрим один из алгоритмов, в котором сначала строят нерекурсивную грамматику, порождающую в точности заданные

цепочки, а затем, сращивая нетерминальные элементы, получают более простую рекурсивную грамматику, порождающую бесконечное число цепочек. Алгоритм можно разделить на три части. Первая часть формирует нерекурсивную грамматику. Вторая часть преобразует ее в рекурсивную грамматику. В третьей части происходит упрощение этой грамматики.

Рассмотрим выборочное множество терминальных цепочек (*caaab, bbaab, caab, bbab, cab, bbb, cb*). Требуется получить грамматику, способную автоматически порождать эти цепочки. Алгоритм построения грамматики состоит из следующих этапов.

1 *этап*. Строится нерекурсивная грамматика, порождающая в точности заданное множество выборочных цепочек. Они обрабатываются в порядке уменьшения длины. Правила подстановки строятся и прибавляются к грамматике по мере того, как они становятся нужны для построения соответствующей цепочки из выборки. Заключительное правило подстановки, используемое для порождения самой длинной выборочной цепочки, называется *остаточным правилом*, а длина его правой части равна 2 (это значение выбрано для удобства алгоритма). Остаточное правило дины *n* имеет вид $A \to a_1 a_2,..., a_n,$ где A – нетерминальный символ, а $a_1 a_2,..., a_n$ – терминальные элементы. Предполагается, что остаток каждой цепочки максимальной длины является суффиксом (хвостом) некоторой более короткой цепочки. Если какой-либо остаток не отвечает этому условию, цепочка, равная остатку, добавляется к обучающей выборке.

В нашем примере первой цепочкой максимальной длины в обучающей выборке является *caaab*. Для ее порождения строятся следующие правила подстановки: $S \to cA_1, A_1 \to aA_2, A_2 \to aA_3, A_3 \to ab,$ где A_3 – правило остатка.

Вторая цепочка – *bbaab*. Для ее порождения к грамматике добавляются следующие правила: $S \to bA_4, A_4 \to bA_5, A_5 \to aA_6, A_6 \to ab.$ Поскольку цепочки *bbaab* и *caaab* имеют одинаковую длину, требуется остаточное правило длины 2. Работа первого этапа алгоритма приводит к некоторой избыточности правил подстановки. Например, вторая цепочка может быть также получена введением следующих правил подстановки: $S \to bA_4, A_4 \to bA_2$. Но первый этап алгоритма занимается лишь определением множества правил постановки, которое способно в точности порождать обучающую выборку, и не касается вопроса избыточности. Устранение избыточности выполняется на третьем этапе алгоритма. Для порождения третьей цепочки *caab* требуется добавление к грамматике только одного правила $A_3 \to b.$ Рассмотрев остальные цепочки из обучающей выборки, устанавливаем, что окончательно множество правил подстановки для порождения выборки выглядит так:

$S \to cA_1, S \to bA_4, A_1 \to aA_2, A_1 \to b, A_2 \to aA_3, A_2 \to b, A_3 \to ab, A_3 \to b,$
$A_4 \to bA_5, A_5 \to aA_6, A_5 \to b, A_6 \to ab, A_6 \to b.$

2 *этап*. Здесь, соединяя каждое правило остатка длиной 2 с другим (неостаточным) правилом грамматики, получаем рекурсивную автоматную

грамматику. Это происходит в результате слияния каждого нетерминального элемента правила остатка с нетерминальным элементом неостаточного правила, который может порождать остаток. Например, если A_r – остаточный нетерминал вида $A_r \to a_1 a_2$ и A_n – неостаточный нетерминал вида $A_n \to a_1 A_m$, где $A_m \to a_2$, все встречающиеся A_r заменяются на A_n, а правило подстановки $A_r \to a_1 a_2$ отбрасывается. Так создается автоматная грамматика, способная порождать данную обучающую выборку, а также обладающая общностью, достаточной для порождения бесконечного множества других цепочек. В рассматриваемом примере A_6 может сливаться с A_5, а A_3 может сливаться с A_2, образуя следующие правила подстановки:

$$S \to cA_1,\ S \to bA_4,\ A_1 \to aA_2,\ A_1 \to b, A_2 \to aA_2,\ A_2 \to b,\ A_4 \to bA_5,$$
$$A_5 \to aA_5,\ A_5 \to b.$$

Рекурсивными правилами являются $A_2 \to aA_2$ и $A_5 \to aA_5$.

3 *этап*. Грамматика, полученная на 2 этапе, упрощается объединением эквивалентных правил подстановки. Два правила с левыми частями A_i и A_j эквивалентны, если выполняются следующие условия. Предположим, что, начиная с A_i, можно породить множество цепочек $\{x\}_i$ и, начиная с A_j, можно породить множество цепочек $\{x\}_j$. Если $\{x\}_i = \{x\}_j$, то два правила подстановки считаются эквивалентными, и каждый символ A_j может быть заменен на A_i без ущерба для языка, порождаемого этой грамматикой, т. е. два правила эквивалентны, если они порождают тождественные цепочки языка.

В рассматриваемом примере эквивалентны правила с левыми частями A_1 и A_2. После слияния A_1 и A_2 получаем

$$S \to cA_1,\ S \to bA_4,\ A_1 \to aA_1,\ A_1 \to b,\ A_4 \to bA_5,\ A_5 \to aA_5,\ A_5 \to b,$$

где исключены многократные повторения одного и того же правила. Следующая пара эквивалентных правил – A_1 и A_5. Выполнив преобразования для них, получим

$$S \to cA_1,\ S \to bA_4,\ A_1 \to aA_1,\ A_1 \to b,\ A_4 \to bA_4.$$

Дальнейшее слияние правил невозможно, поэтому алгоритм в процессе обучения строит следующую автоматную грамматику:

$$G = (V_N, V_T, P, S), V_N = (S, A, B), V_T = (a, b, c),$$
$$P: S \to cA, S \to bB, A \to aA, B \to bA, A \to b.$$

Можно проверить, что данная грамматика порождает обучающую выборку, использованную в процессе построения грамматики.