ЛЕКЦИЯ 10

План лекции:

1. Модель построения цепочечных грамматик.
2. Алгоритм вывода двумерных грамматик.

10.1 Обучение и грамматический вывод

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. Тем не менее, область грамматического вывода находится еще в начальной стадии развития. На рисунке 1 представлена модель вывода цепочечных грамматик.

Рис. 1 – Модель вывода цепочечных грамматик

Задача, показанная на рисунке 10.1, 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 одни из алгоритмов, в котором сначала строится нерекурсивная грамматика, порождающая 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 часть. 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 $A->a_1a_{2...}a_n$. Где A – нетерминальный символ, а $a_1a_{2...}a_n$ – терминальные элементы. Предполагается, что остаток каждой цепочки максимальной длины является суффиксом (хвостовым концом) 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:
$S->cA_1, A_1->aA_2, A_2->aA_3, A_3->ab,$ где A_3 – правило остатка. Вторая цепочка – bbaab. Для ее порождения к грамматике добавляются следующие правила:
$S->bA_4, A_4->bA_5, A_5->aA_6, A_6->ab$. Поскольку цепочка bbaab и caaab имеют одинаковую длину, требуется остаточное правило длины 2. Работа первой части алгоритма приводит к некоторой избыточности правил подстановки. Например, вторая цепочка может быть также получена введением следующих правил подстановки: $S->bA_4, A_4->bA_2$. Но первая часть алгоритма занимается лишь определением множества правил постановки, которое способно в точности порождать обучающую выборку, и не касается вопроса избыточности. Устранение избыточности выполняется в третьей части алгоритма. Для порождения третьей цепочки caab требуется добавление к грамматике только одного правила $A_3->b$. Рассмотрев остальные цепочки из обучающей выборки, устанавливаем, что окончательно множество правил подстановки для порождения выборки выглядит так:
$S->cA_1, S->bA_4, A_1->aA_2, A_1->b, A_2->aA_3, A_2->b, A_3->ab, A_3->b, A_4->bA_5, A_5->aA_6, A_5->b, A_6->ab, A_6->b.$

2 часть. Здесь, соединяя каждое правило остатка длины 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 $A_r->a_1a_2$ *и* A_n – неостаточный нетерминал вида $A_n->a_1A_m$, *где* $A_m->a_2$, все встречающиеся A_r заменяются на A_n, а правило подстановки $A_r->a_1a_2$ 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:
$S->cA_1, S->bA_4, A_1->aA_2, A_1->b, A_2->aA_2, A_2->b, A_2->b, A_4->bA_5,$

A_5->aA_5, A_5->b, A_5->b.
Рекурсивными правилами являются A_2 ->aA_2 *и* A_5->aA_5.

3 часть. Грамматика, полученная во 2 части, 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 A1 и A2. После слияния A_1 и A_2 получаем:
S->cA_1, S->bA_4, A_1->aA_1, A_1->b, A_4->bA_5, A_5->aA_5, A_5->b. Где исключены многократные повторения одного и того же правила. Теперь ясно, что эквиваленты A_1 и A_5, выполним преобразования для них и получим:
S->cA_1, S->bA_4, A_1->aA_1, A_1->b, A_4->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->cA, S->bB, A->aA, B->bA, A->b.

Можно легко проверить, что данная грамматика порождает обучающую выборку, использованную в процессе ее вывода.

10.2 Вывод двумерных грамматик

Основная сложность применения этих грамматик заключается в точном определении правил двумерного соединения. Рассмотрим алгоритм, отражающий построение двумерной грамматики в процессе обучения. Алгоритм допускает, что соответствующие двумерные позиционные дескрипторы задаются учителем. Суть алгоритма в следующем: имеется множество непроизводных элементов и позиционных дескрипторов, начиная с непроизводных элементов и, применяя дескрипторы, строятся более сложные структуры. Когда процесс завершается, выводится грамматика с использованием шагов построения структур. Рассмотрим пример вывода грамматики. На рис. 2 изображены простые непроизводные элементы, позиционные дескрипторы и выборочный образ, который является комбинацией непроизводных элементов. Для упрощения системы обозначений назовем окружность выборочного образа «объектом 1», левый глаз – «объектом 2», правый глаз – «объектом 3», нос – «объектом 4», рот – «объектом 5». Начиная с непроизводных элементов и последовательно применяя дескрипторы, можно построить различные сложные объекты.

Процесс построения рассмотрим на примере выборочного образа. Первым сложным образом является объект 6: I(2,1), т.е. объект 2, находящийся внутри объекта 1.Этому условию удовлетворяет выборочный образ.

Следующие объекты также соответствуют выборочному образу: объект 7: I(3,1), объект 8: I(4,1), объект 9: I(5,1), объект 10: L(2,3), объект 11: A(4,5).

На следующем шаге из порожденных объектов строятся более сложные структуры: объект 12: I(10,1), объект 13: A(10,4), объект 14: A(10,5), объект 15: A(10,11).

Следующий уровень сложности достигается дальнейшей комбинацией ранее порожденных объектов:

объект 16: I(13,1), объект 17: A(14,1), объект 18: A(15,1), объект 19: A(13,5).

Объект 18 является полным терминальным описанием исследуемого образа, т.е. объект 18 – это объект 15, находящийся внутри объекта 1, представляющего собой окружность. Объект 15 – это объект 10, расположенный над объектом 11. Кроме того, объект 10 – это один глаз, расположенный слева от другого, а объект 11 – это нос, расположенный надо ртом. Таким образом, объект 18 представляет собой искомый образ лица.

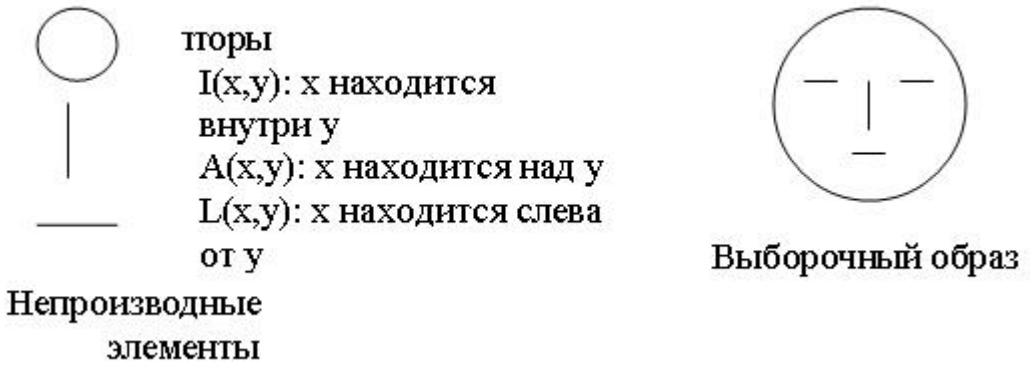

Рис. 2 – Исходные данные для построения грамматики

Грамматика, порождающая выборочный образ, легко восстанавливается по шагам, ведущим к построению объекта. Грамматика выглядит так:
$G=(V_N, V_T, P, S)$. $V_N=(S,B,C,D)$, $V_T=(h,v,c)$,
P: S->I(B,c), B->A(C,D), C->L(h,h), D-A(v,h).

Множество правил подстановки является правилами построения образа. Если предположить, что S – это лицо, то правила подстановки представляют следующее. Лицо – это некоторый объект B, расположенный внутри окружности. Объект B представляет собой некоторый объект C, расположенный над другим объектом D, причем C – горизонтальный отрезок, расположенный слева от другого горизонтального отрезка (глаза), а D – вертикальный отрезок, расположенный над горизонтальным (нос и рот).

Если задано несколько выборочных образов, грамматика выводится для каждого из них. Затем грамматики объединяются, а эквивалентные правила сливаются. Получающаяся в результате грамматика способна порождать всю обучающую выборку полностью. Эта процедура выглядит так же, как и схема сращивания для цепочечных объектов.

Главное в выводе двумерных грамматик: промежуточные объекты, порожденные в этом примере, не исчерпывают всех возможностей. Однако здесь ставилась задача порождения одного или более множества шагов процесса построения, приводящего от непроизводных элементов к выборочному образу. При этом желательно порождать как можно меньше промежуточных объектов. Еще один вопрос – это определение позиционных дескрипторов и в конечном итоге – правила двумерного соединения структур.