Распознавание образов

Универсальный алгоритм пост-обработки результатов распознавания на основе проверяющих грамматик*

К. Б. Булатов, Д. П. Николаев, В. В. Постников

Аннотация. В данной работе рассматривается задача статистической коррекции (пост-обработки) результата распознавания текстовых полей. Обсуждается постановка задачи пост-обработки для текстовых полей с языковой моделью, представленной в виде проверяющей грамматики. Описывается универсальный алгоритм для поиска гипотез, удовлетворяющих языковой модели на основе проверяющей грамматики и описываются его свойства. Алгоритм основан на организации эффективного перебора цепочек альтернативных результатов распознавания в порядке убывания общей оценки и последующей проверке допустимости цепочек с точки зрения проверяющей грамматики.
Ключевые слова: *автоматический ввод документов, распознавание документов, оптическое распознавание символов, статистическая коррекция.*

Введение

Статистическая коррекция, или «пост-обработка» результатов распознавания является одним из важнейших компонентов современных систем оптического распознавания документов. Пост-обработка представляет собой алгоритм априорного исправления ошибок локализации, сегментации и распознавания поля (или группы полей) на основе языковой модели этого поля, с использованием информации о синтаксической и семантической структуре документа.

Существует множество алгоритмов статистической пост-обработки результатов распознавания, отличающихся в используемых языковых моделях распознаваемого объекта, в алгоритмах непосредственного исправления результата распознавания и в областях применимости. Среди наиболее известных и широко используемых методов можно выделить: методы, опирающиеся на скрытые марковские модели (Hidden Markov Models, HMM) [1], конечные автоматы, N-граммные и словарные методы [2], а также механизмы, использующие взвешенные конечные преобразователи (Weighted Finite-State Transducers, WFST) [3].

Располагая информацией о семантической и синтаксической структуре документа и распознаваемого поля, можно построить специализированный алгоритм пост-обработки для каждого конкретного поля. Однако, принимая во внимание необходимость поддержки и развития систем распознавания и сложность их разработки, особый интерес представляют методы и инструменты, позволяющие с минимальными усилиями (со стороны разработчиков системы распознавания) построить достаточно хороший алгоритм пост-обработки, который бы работал с обширным классом документов и полей. Методика настройки и поддержки такого алгоритма была бы

* Исследование выполнено при частичной финансовой поддержке Российским фондом фундаментальных исследований (проекты 13–07–12172 офи_м и № 15–07–06520 офи_м).

унифицирована, а изменяемым компонентом структуры алгоритма были бы только семантика и синтаксис обрабатываемого поля.

1. Универсальный алгоритм на основе взвешенных конечных преобразователей

Достаточно общая модель, позволяющая построить универсальный алгоритм пост-обработки результатов распознавания, описана в работе [3]. Модель опирается на структуру данных взвешенных конечных преобразователей (Weighted Finite-State Transducers, WFST).

WFST представляют собой обобщение взвешенных конечных автоматов — если взвешенный конечный автомат кодирует некоторый взвешенный язык L (т. е. взвешенное множество строк на некоторым алфавитом Σ), то WFST кодирует взвешенное отображение языка L_1 над алфавитом Σ_1 в язык L_2 над алфавитом Σ_2. Взвешенный конечный автомат, принимая строку S над алфавитом Σ, ставит ей в соответствие некоторый вес W, тогда как WFST, принимая строку S_1 над алфавитом Σ_1, ставит ей в соответствие (возможно, бесконечное) множество пар $\left\{\left\langle S_2^1, \mathrm{W}_1\right\rangle,...,\left\langle S_2^k, W_k\right\rangle,...\right\}$, где S_2^i — строка над алфавитом Σ_2, в которую преобразуется строка S_1, а W_i — вес такого преобразования. Веса являются элементами некоторого полукольца (т. е. по сложению не требуется существование противоположных элементов). Каждый переход преобразователя характеризуется двумя состояниями (между которыми осуществляется переход), входным символом (из алфавита Σ_1), выходным символом (из алфавита Σ_2) и весом перехода. Считается, что пустой символ (пустая строка) является элементом обоих алфавитов. Весом преобразования строки X в строку Y является сумма произведений весов переходов по всем путям, на которых конкатенация входных символов образует строку X, а конкатенация выходных символов образует строку Y, и которые переводят преобразователь из начального состояния в одно из терминальных.

Над WFST определяется операция *композиции*, на которую и опирается предложенный в работе [3] метод. Пусть заданы два преобразователя T_1 и T_2, причем T_1 преобразует строку X над алфавитом Σ_X в строку Y над алфавитом Σ_Y, и вес такого преобразования равен W_1, а T_2 преобразует строку Y над алфавитома Σ_Y в строку Z над алфавитом Σ_Z с весом W_2. Тогда преобразователь $T_1 \circ T_2$, называемый композицией преобразователей T_1 и T_2, преобразует строку X в строку Z с весом $W_l \cdot W_2$. Композиция преобразователей является вычислительно затратной операцией, но может вычисляться «лениво», т. е. состояния и переходы результирующего преобразователя могут быть построены в момент, когда к ним необходимо совершить доступ.

Модель алгоритма пост-обработки результатов распознавания на основе WFST опирается на три основных источника информации — модель гипотез (Hypothesis Model, HM), модель ошибок (Error Model, EM) и языковую модель (Language Model, LM). Все три модели представляются в виде взвешенных конечных преобразователей:

1) *Модель гипотез* (*HM*) представляет из себя взвешенный конечный автомат (частный случай WFST — каждая строка преобразуется в себя, вес преобразования совпадает с весом самой строки), принимающий гипотезы, выдвинутой системой распознавания поля. К примеру, пусть результатом распознавания текстового поля является последовательность ячеек $C_1, C_2, ..., C_N$, а каждая ячейка соответствует результату распознавания некоторого символа и представляет собой множество альтернатив и их оценок (весов): $C_i = \left\{\left\langle a_{i1}, q_{i1}\right\rangle,...,\left\langle a_{ik}, q_{ik}\right\rangle\right\}$, где a_{ij} — j-й символ алфавита, q_{ij} — оценка (вес) этого символа. Тогда модель гипотез можно представить в виде конечного преобразователя с $(N+1)$-м состоянием, уложенным в цепочку: 0-е состояние является начальным, N-е состояние является терминальным, переходы осуществляются между $(i-1)$-м и i-м состояниями и соответствуют ячейке C_i. На j-м переходе входным и выходным символом является символ a_{ij}, а вес перехода соотвествует оценке данной альтернативы q_{ij}. На рис. 1 представлен пример модели гипотез, соответствующей результату распознавания строки их трех символов. Алфавитом является множество $\{a,b,c\}$, в качестве оценок альтернатив (и весов переходов) используются вероятностные оценки (т. е. сумма всех переходов, выходящих из одного состояния, равна единице).

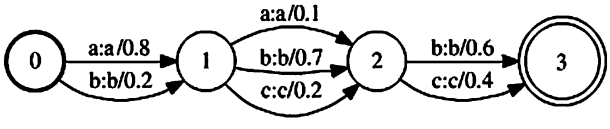

Рис. 1. Пример модели гипотез, представленной в виде WFST [3]. Переходы с нулевым весом опущены

2) *Модель ошибок* (*EM*) является взвешенным конечным преобразователем, кодирующим всевозможные преобразования гипотез, которые можно производить для приведения этих гипотез в соответствие языковой модели. В самом простом случае она может быть представлена в виде WFST с единственным состоянием (которое является и начальным и терминальным). Каждый переход кодирует некоторое изменение символа с соответствующим весом. К примеру, весом замены символа может являться оценка вероятности соответствующей ошибки алгоритма распознавания символов.

3) *Языковой моделью* (*LM*) является представление синтаксиса распознаваемого поля в виде взвешенного конечного автомата. Т. е. языковой моделью является автомат, принимающий все допустимые значения распознаваемого поля (и выставляющий каждому допустимому значению некоторый вес).

После определения модели гипотез, ошибок и языковой модели задача пост-обработки результатов распознавания теперь может быть поставлена следующим образом: рассмотрим композицию всех трех моделей $T = HM \circ EM \circ LM$ (в терминах композиции WFST). Преобразователь T кодирует всевозможные преобразования строк X из модели гипотез *HM* в строки Z из языковой модели *LM*, применяя к строкам X преобразования, закодированные в модели ошибок *EM*. Причем вес такого преобразования включает вес исходной гипотезы, вес преобразования и вес результирующей строки (в случае взвешенной языковой модели). Соответственно в такой модели оптимальной пост-обработке результата распознавания будет соответствовать оптимальный (с точки зрения веса) путь в преобразователе T, переводящий его из начального в одно из терминальных состояний. Входная строка на этом пути будет соответствовать выбранной исходной гипотезе, а выходная строка — исправленному результату распознавания. Оптимальный путь можно искать с помощью алгоритмов поиска кратчайших путей в ориентированных графах.

Преимуществами данного подхода является его общность и гибкость. Модель ошибок, к примеру, может быть без труда расширена таким образом, чтобы учесть удаления и добавления символов (для этого всего лишь стоит добавить в модель ошибок переходы с пустым выходным или входным символом соответственно). Однако у такой модели есть и существенные недостатки. Во-первых, языковая модель здесь должна быть представлена в виде конечного взвешенного конечного преобразователя. Для сложных языков такой автомат может получиться довольно громоздким, и в случае изменения или уточнения языковой модели будет необходимо его перестроение. Также необходимо заметить, что композиция трех преобразователей в качестве результата имеет, как правило, еще более громоздкий преобразователь, а эта композиция вычисляется каждый раз при запуске пост-обработки одного результата распознавания. Ввиду громоздкости композиции, поиск оптимального пути на практике приходится выполнять эвристическими методами [3], такими как A*-search [4].

2. Проверяющие грамматики

Используя модель проверяющих грамматик можно построить более простую модель задачи пост-обработки результатов распознавания, которая будет не насколько общей, как модель на основе WFST, однако легко расширяемой и простой в имплементации.

Проверяющей грамматикой будем называть пару $G = \langle \Sigma, P \rangle$, где Σ — алфавит, а P — предикат допустимости строки над алфавитом Σ, т. е. $P : \Sigma \to \{0,1\}$. Проверяющая грамматика кодирует некоторый язык над алфавитом Σ следующим образом: строка $S \in \Sigma^*$ принадлежит языку, если предикат P принимает на этой строке истинное значение. Стоит заметить, что проверяющая грамматика является более общим способом представления языковой модели, чем конечный автомат. Действительно, любой язык, представимый в виде конечного автомата A, может быть представлен в виде проверяющей грамматики (с предикатом $P(S) \Leftrightarrow S \in \text{acc}(A)$, где $\text{acc}(A)$ — множество строк, принимаемых автоматом A. Однако не все языки, которые можно представить в виде проверяющей грамматики, представимы в виде конечного автомата в общем случае (к примеру, язык слов неограниченной длины над алфавитом $\Sigma = \{X, Y\}$, в которых количество символов *X* больше, чем количество символов *Y*).

Пусть задан результат распознавания (модель гипотез) в виде последовательности ячеек $C_1, C_2, ..., C_N$. Пусть, опять же, каждая ячейка соответствует результату распознавания некоторого символа и представляет собой множество альтернатив и их оценок: $C_i = \{\langle a_{i1}, q_{i1} \rangle, ..., \langle a_{ik}, q_{ik} \rangle\}$, где a_{ij} — *j*-й символ алфавита, q_{ij} — оценка этого символа. Для удобства будем считать, что каждая ячейка содержит *K* альтернатив и все оценки альтернатив принимают положительное значение. Оценкой (весом) строки будем считать произведение оценок каждого из символов этой строки. Если модель языка задана в виде проверяющей грамматики $G = \langle \Sigma, P \rangle$, то задача пост-обработки результата распознавания может быть сформулирована в виде задачи дискретной оптимизации (максимизации) на множестве управлений Σ^N (множестве всех строк длины *N* над алфа-

витом Σ) с предикатом допустимости P и функционалом $F(S \in \Sigma^N) = \prod_{i=1}^{N} q_i(S_i)$, где $q_i(S_i)$ — оценка символа S_i в *i*-й ячейке.

3. Алгоритм пост-обработки с использованием проверяющей грамматики

Опишем алгоритм решения задачи оптимизации, поставленной в предыдущем пункте. Любая задача дискретной оптимизации (т. е. на конечном множестве управлений) может быть решена при помощи полного перебора управлений. Описываемый алгоритм эффективно перебирает управления (строки) в порядке убывания значения функционала до того момента, пока предикат допустимости не примет истинного значения. Зададим как M максимальное количество итераций алгоритма, т. е. максимальное количество строк с максимальной оценкой, на которой будет вычисляться предикат.

Вначале проведем сортировку альтернатив по убыванию оценок, и будем далее считать что для любой ячейки C_i верно неравенство $q_{ij} \geq q_{ik}$ при $j < k$. *Позицией* будем называть последовательность индексов $j_1,...,j_N$, соответствующее строке $\{a_{1j_1},...,a_{Nj_N}\}$. Оценкой позиции, т. е. значением функционала на этой позиции, считаем произведение оценок альтернатив, соответствующих индексам, входящих в позицию: $\prod_{i=1}^{N} q_{ij_i}$. Для хранения позиций потребуется структура данных *PositionBase*, позволяющая добавлять новые позиции (с получением их адреса), получать позицию по адресу и проверять, добавлена ли заданная позиция в базу.

В процессе перечисления позиций мы будем выбирать непросмотренную позицию с максимальной оценкой, после чего добавлять в очередь на рассмотрение *PositionQueue* все позиции, которые можно получить из текущей добавлением единицы к одному из индексов, входящих в позицию. Очередь на рассмотрение *PositionQueue* будет содержать тройки $\langle Q, A, I \rangle$, где Q — оценка непросмотренной позиции, A — адрес просмотренной позиции в *PositionBase*, из которой была получена данная позиция, I — индекс элемента позиции с адресом A, который был увеличен на единицу для получения данной позиции. Для организации очереди *PositionQueue* потребуется структура данных, позволяющая добавлять очередную тройку $\langle Q, A, I \rangle$, а также извлекать тройку с максимальным значением оценки Q.

На первой итерации алгоритма необходимо рассмотреть позицию $S_1 = \{1,1,...,1\}$, обладающую максимальной оценкой. Если предикат P принимает на строке, соответствующей этой позиции, истинное значение, то алгоритм завершается. В противном случае позиция S_1 добавляется в *PositionBase*, а в *PositionQueue* добавляются все тройки вида $\langle Q \cdot q_{i2}/q_{i1}, A(S_1), i \rangle$, для всех $i \in \{1,...,N\}$, где $A(S_1)$ — адрес начальной позиции в *PositionBase*. На каждой последующей итерации алгоритма из *PositionQueue* извлекается тройка $\langle Q, A, I \rangle$ с максимальным значением оценки Q, по адресу исходной позиции A и индексу I восстанавливается рассматриваемая позиция S. Если позиция S уже добавлена в базу рассмотренных позиций *PositionBase*, то она пропускается и из *PositionQueue* извлекается очередная тройка с максимальным значением оценки Q. В противном случае на строке, соответствующей позиции S проверяется значение предиката P. Если предикат P принимает на этой строке истинное значение, то алгоритм завершается. Если предикат P не принимает истинное значение на этой строке, то строка S добавляется в *PositionBase* (с присвоением адреса $A(S)$), в очередь *PositionQueue* добавляются все производные позиции и происходит переход к следующей итерации.

FindSuitableString(*M, N, K, P,* C_1, ... C_N):

для каждого $i \in \{1,...,N\}$:

сортировка C_i по убыванию оценок альтернатив;

инициализация *PositionBase* и *PositionQueue*;

$S_1 = \{1,1,...,1\}$;

если $P(S_1)$, то ответ: S_1, алгоритм завершается;

добавить S_1 в *PositionBase* с адресом $A(S_1)$;

для каждого $i \in \{1,...,N\}$:

если $K > 1$, то:

добавить тройку $\langle Q \cdot q_{i2}/q_{i1}, A(S_1), i \rangle$ в *PositionQueue*;

пока количество позиций в *PositionBase* меньше M:

(*внутренний цикл*) пока не пуста *PositionQueue*:

извлечь из *PositionQueue* тройку $\langle Q, A, I \rangle$ с максимальной оценкой Q;

S_{from} = позиция в *PositionBase* по адресу A;

$S_{curr} = \{S_{from}(1), \mathrm{S}_{from}(2), ..., \mathrm{S}_{from}(I)+1, ..., S_{from}(N)\}$;
если S_{curr} содержится в *PositionBase*,
то: повторить внутренний цикл;
иначе: добавить S_{curr} в *PositionBase*
с адресом $A(S_{curr})$;
если $P(S_{curr})$, то ответ: S_{curr}, алгоритм завершается;
для каждого $i \in \{1, ..., N\}$:
если $K > S_{curr}(\mathrm{i})$:
добавить тройку
$\left\langle Q \cdot q_{i\{\mathrm{S}_{curr}(\mathrm{i})+1\}} / q_{i\{\mathrm{S}_{curr}(\mathrm{i})\}}, A(\mathrm{S}_{curr}), i \right\rangle$
в *PositionQueue*;
выйти из внутреннего цикла;
ответ не найден за M итераций;

Заметим, что за M итераций будет осуществлена проверка предиката не более чем M раз, произойдет не более M добавлений в *PositionBase*, а добавление в *PositionQueue*, извлечение в *PositionQueue*, а также поиск в *PositionBase* произойдет не более $M \cdot N$ раз. Если для реализации *PositionQueue* используется структура данных «куча», а для организации *PositionBase* используеся структура данных «бор», то трудоемкость алгоритма в случае M итераций составляет $O\left(M \cdot \left(p(N) + N^2 + N\log\left(M \cdot N\right)\right)\right)$, где $p(N)$ — трудоемость проверки предиката P на строке длины N.

4. Сокращение перебора с использованием префиксной проверки

Эффективную глубину перебора при сохранении максимального количества итераций алгоритма можно увеличить, введя дополнительный предикат допустимости префикса $\tilde{P}$:

$\tilde{P}(\varepsilon) = 1$, где ε — пустая строка;
$\forall S \in \Sigma^* : P(S) = 1 \Rightarrow \tilde{P}(S) = 1$;
$\forall s \in \Sigma^* : \tilde{P}(s) = 1 \Rightarrow \exists S \in \Sigma^* : P(S) = 1 \wedge \{ s$ — *префикс строки* $S\}$

Используя предикат допустимости префикса $\tilde{P}$ внутренний цикл алгоритма можно модифицировать следующим образом:

(*внутренний цикл*) пока не пуста *PositionQueue*:
извлечь из *PositionQueue* тройку $\langle Q, A, I \rangle$
с максимальной оценкой Q;
S_{from} = позиция в *PositionBase* по адресу A;
$S_{curr} = \{S_{from}(1), \mathrm{S}_{from}(2), ..., \mathrm{S}_{from}(I)+1, ..., S_{from}(N)\}$;
если S_{curr} содержится в *PositionBase*,
то: повторить внутренний цикл;
иначе: добавить S_{curr} в *PositionBase*
с адресом $A(S_{curr})$;
если $P(S_{curr})$, то ответ: S_{curr}, алгоритм завершается;
для каждого $i \in \{1, ..., N\}$:
если $\tilde{P}\left(\{S_{curr}(1), ..., S_{curr}(i-1)\}\right) = 0$ **:**
выйти из внутреннего цикла;
если $K > S_{curr}(\mathrm{i})$:
добавить тройку
$\left\langle Q \cdot q_{i\{\mathrm{S}_{curr}(\mathrm{i})+1\}} / q_{i\{\mathrm{S}_{curr}(\mathrm{i})\}}, A(\mathrm{S}_{curr}), i \right\rangle$
в *PositionQueue*;
выйти из внутреннего цикла;

Генерацию производных позиций можно завершить, если префикс позиции, оканчивающийся в увеличивающемся индексе, не доставляет истинное значение предикату допустимости префикса. Все возможные допустимые позиции, которые можно было бы получить из «отсеченных» производных позиций с недопустимым префиксом, можно также получить из исходной позиции, увеличивая индекс раньше (пока префикс все еще допустим).

Заключение

Была рассмотрена задача пост-обработки результатов распознавания текстовых полей, описан общий подход и представлен алгоритм для поиска гипотез, удовлетворяющих языковой модели распознаваемого поля, представленной в виде проверяющей грамматики. Алгоритм достаточно прост в реализации и универсален, т. к. легко модифицируется для конкретной задачи. Для адаптации алгоритма к конкретной задаче пост-обработки результатов распознавания поля, необходимо лишь реализовать предикат допустимости значения распознаваемого текстового поля. Описанный алгоритм успешно

применяется в существующих системах распознавания для пост-обработки результатов нескольких различных типов полей, таких как даты, поля с проверкой по контрольным цифрам и поля с языковой моделью в виде полного словаря.

Литература

1. *Bouchaffra D.*, *Govindaraju V.*, *Srihari S. N.* Postprocessing of Recognized Strings Using Nonstationary Markovian Models // IEEE Transactions on Pattern Analysis and Machine Intelligence. 1997. V. 21. № 10, P. 990–999
2. *Kukich K.* Techniques for Automatically Correcting Words in Text // ACM computing survey, Computational Linguistics. 1992. V. 24. № 4, P. 377–439 .
3. *Llobet R.*, *Cerdan-Navarro J.-R.*, Perez.-*Cortes J.*, *Arlandis J.* OCR Post-processing Using Weighted Finite-State Transducers // Pattern Recognition (ICPR), 2010. P. 2021–2024.
4. *Hart P. E.*, *Nilsson N. J.*, *Raphael B.* A formal Basis for the Heuristic Determination of Minimum Cost Paths // IEEE Transactions on Systems Science and Cybernetics SSC4. 1968 № 4 (2), P. 100–107

Булатов Константин Булатович. М. н. с. ИСА РАН, аспирант НИТУ МИСиС. Окончил НИТУ МИСиС в 2013 г. Количество печатных работ: 3. Область научных интересов: распознавание образов, машинное обучение, информационные системы, обработка изображений. E-mail: hpbuko@gmail.com.

Николаев Дмитрий Петрович. Зав. сектором ИППИ РАН. К. ф.-м. н. Окончил в 2000 г. МГУ. Количество печатных работ: более 100. Область научных интересов: быстрые алгоритмы обработки изображений. E-mail: dimonstr@iitp.ru

Постников Василий Валерьевич. Зав. лабораторией ИСА ФИЦ ИУ РАН. К. т. н. Окончил в 1990 г. МФТИ. Количество печатных работ: более 40. Область научных интересов: компьютерное зрение. E-mail:vassili.postnikov@gmail.com