Геркулес: интеграция правил для самозагрузочного метода извлечения информации из текстов на естественном языке♣

© Мстислав Масленников

ЗАО «Авикомп Сервисез»
mstislav.maslennikov@avicomp.ru

Аннотация

Извлечение информации из текстов на естественном языке значительно сложнее чем из полуструктурированных текстов, так как требует аннотирования значительного количества тренировочных примеров. В этой работе мы описываем новую самозагрузочную систему ГЕРКУЛЕС, которая комбинирует эвристическое знание от экспертов и обучение на основе правил. Стартуя с небольшого количества заданных эвристических правил, ГЕРКУЛЕС генерирует новые правила, с помощью которых получает новые тренировочные примеры, тем самым повышая качество извлечения информации в рамках самозагрузочной модели обучения. Вся модель строится на основе трёх стадий: генерация базовых правил, уточнение и интеграция правил. Новые правила генерируются на каждой итерации и интегрируются в текущий набор правил. Экспериментальная проверка показывает, что полученная система работает лучше других самозагрузочных систем на наборе неструктурированных текстов MUC7. Другое преимущество ГЕРКУЛЕСа состоит в том, что сгенерированные правила доступны для понимания человеку, что трудноосуществимо в рамках статистических методов машинного обучения.

1. Введение

Извлечение информации, в частности извлечение собственных имён (СИ), является критически важной технологией для поддержки вопросно-ответных систем, систем информационного поиска и систем понимания документов. Заметим, что для полуструктурированных текстов (например, объявлений на работу) характерна избыточность данных одинаковой структуры, что значительно упрощает задачу по извлечению СИ. Поэтому извлечение СИ из текстов на естественном языке значительно сложнее по сравнению с полуструктурированными текстами. Однако наиболее полезная информация содержится в форме текстов на естественном языке, которая не соответствует какому-то строгому шаблону или структуре. Из анализа многих работ по обработке текстов на естественном языке можно сделать вывод, что для корректного извлечения информации требуется большое количество аннотированных данных. Например, [26] сообщили о получении результата F_1 = 94.1% на естественных текстах, используя весь тренировочный корпус документов для обучения. Они утверждают, что небольшое уменьшение количества тренировочных примеров ведёт к значительному ухудшению качества обучения. Согласно [1], для двукратного сокращения ошибок алгоритмам обучения требуется, по крайней мере, двукратное увеличение количества тренировочных примеров.

Таким образом, задача получения большого количества аннотированных примеров зачастую является критически важной. Поэтому требуется автоматизированный метод для извлечения тренировочных примеров. В настоящий момент, самозагрузочный метод автоматизированной аннотации и генерации правил был исследован в различных работах ([2], [15]). Эти исследователи предположили, что вместо аннотации полного корпуса тренировочных текстов можно аннотировать только часть корпуса, а затем обучать систему на неразмеченных текстах. Однако большинство из самозагрузочных систем по-прежнему не достигают разумного результата без интенсивного участия человека [24], например, через *активное обучение* [22]. Наилучший существующий метод [15][1] для текстов на естественном языке, использующий самозагрузочный подход, достигает качества только F_1=87.7% для извлечения людей, F_1=82.3% для

Труды 13[й] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2011, Воронеж, Россия, 2011.

[1] Использует 35 стартовых ключевых слов

извлечения местоположений и F_1=52.7% для организаций.

Другая мотивация нашего исследования состоит в том, что текущие системы не используют достоинства эвристических и сгенерированных правил одновременно. Эвристические правила могут быть созданы экспертами по предметной области для эффективного описания важных закономерностей в текстах корпуса. На практике, создание нескольких эвристических правил обходится дешевле просмотра и аннотирования большого количества текстов. Однако количество таких правил зачастую недостаточно для описания всех ситуаций, из которых нужно извлекать информацию. С другой стороны, компьютер может мгновенно генерировать машинные правила, однако сгенерированные правила зачастую оказываются слишком специфичными и требуют большого количества тренировочных примеров для их обобщения. Согласно [5], из-за недостаточного обобщения правила могут покрывать небольшое количество примеров тренировочного корпуса, из-за чего результат на корпусе для тестирования может значительно ухудшиться. Таким образом, задача интеграции эвристических и сгенерированных правил является полезной и важной для практического извлечения информации.

В этой статье мы представляем новую самозагрузочную систему под названием ГЕРКУЛЕС (HypEr Rule Creation Using Less ExampleS) для получения правил, извлекающих собственные имена из текстов на естественном языке. Наш подход стартует только с нескольких первоначальных эвристических правил. Затем он постепенно извлекает новые тренировочные примеры, на основе которых генерирует новые правила, улучшая таким образом качество системы извлечения информации в рамках самозагрузочной модели. Эксперименты показывают, что ГЕРКУЛЕС превосходит существующие самозагрузочные методы извлечения информации на текстовом корпусе MUC7. Основной вклад нашего исследования состоит в разработке метода генерации правил, который в состоянии интегрировать эвристические правила и автоматически уточнять сгенерированные правила.

Оставшиеся части статьи организованы следующим образом. Раздел 2 описывает самозагрузочную модель и используемые стратегии генерации правил. В разделе 3 мы приводим результаты экспериментов, а в 4-м разделе мы сравниваем наш метод с существующими похожими подходами. Наконец, раздел 5 завершает эту статью.

2. Самозагрузочная модель извлечения информации

2.1 Общая модель

Наша модель состоит из 3 частей: генерация правил, уточнение правил и интеграция правил. ГЕРКУЛЕС начинает самозагрузочный цикл с использования небольшого множества эвристических правил (ЭП) и тренировочных примеров. Одно из ЭП, использованное для класса ЧЕЛОВЕК выглядит как:

HumCap -> Capwords & !список(nationality)

PersonPeriod -> HumCap+ Cap sym_. HumCap

ЧЕЛОВЕК -> список(personTitle) PersonPeriod

Для фрагмента *“Mr. George W. Bush Jr”* это правило извлекает *‘George’* как *HumCap*, ‘W’ как *Cap*, ‘.’ как *sym* и *‘Bush’* как *HumCap*, после чего выражение *‘George W. Bush’* извлекается как *PersonPeriod*. В дальнейшем это правило сопоставляет *‘Mr.’* как *PersonTitle*, и после этого благодаря подчёркиванию цели *PersonPeriod* выражение “*George W. Bush”* извлекается как ЧЕЛОВЕК. Эти ЭП получены из более ранней эвристической системы извлечения собственных имён. Благодаря использованию высокоуровневых ограничений (например, список слов personTitle) эти ЭП не привязываются к одному слову в примерах, поэтому они в состоянии извлечь достаточно большой набор положительных и отрицательных примеров из неразмеченного корпуса, которые можно использовать для генерации новых правил с помощью машины. Новые примеры, полученные во время первого цикла, в случае необходимости могут быть случайным образом выбраны и верифицированы через взаимодействие с пользователем. Однако наши результаты показывают, что это взаимодействие не является необходимым. Единственное усилие, требуемое от человека, состоит в подготовке нескольких первоначальных правил для каждого типа собственных имён.

Чтобы инициализировать систему, первоначальные ЭП добавляются во множество *текущих правил*, на основе которых извлекаются несколько первоначальных примеров и добавляются во множество *надёжных тренировочных примеров* и во множество *неразмеченных примеров* на Рисунке 1. После этого множество надёжных тренировочных примеров используется для тренировки валидатора на основе метода максимальной энтропии (МЭ). Этот валидатор проверяет тип извлечённых из примеров СИ на каждой итерации цикла. Используя множество текущих правил, мы извлекаем собственные имена в примерах из множества *неразмеченных примеров* и начинаем итерацию самозагрузки. Новые примеры пропускаются через МЭ валидатор для проверки, и валидированные примеры добавляются во множество *нормальных тренировочных примеров* для генерации правил. Генерация правил основана на получении новых правил с помощью *генератора правил*, *уточнения правил* низкого качества и *интегрирования* новых *правил* с существующим набором правил. Полученное окончательное множество правил затем снова применяется к неразмеченным примерам для извлечения СИ на следующей итерации.

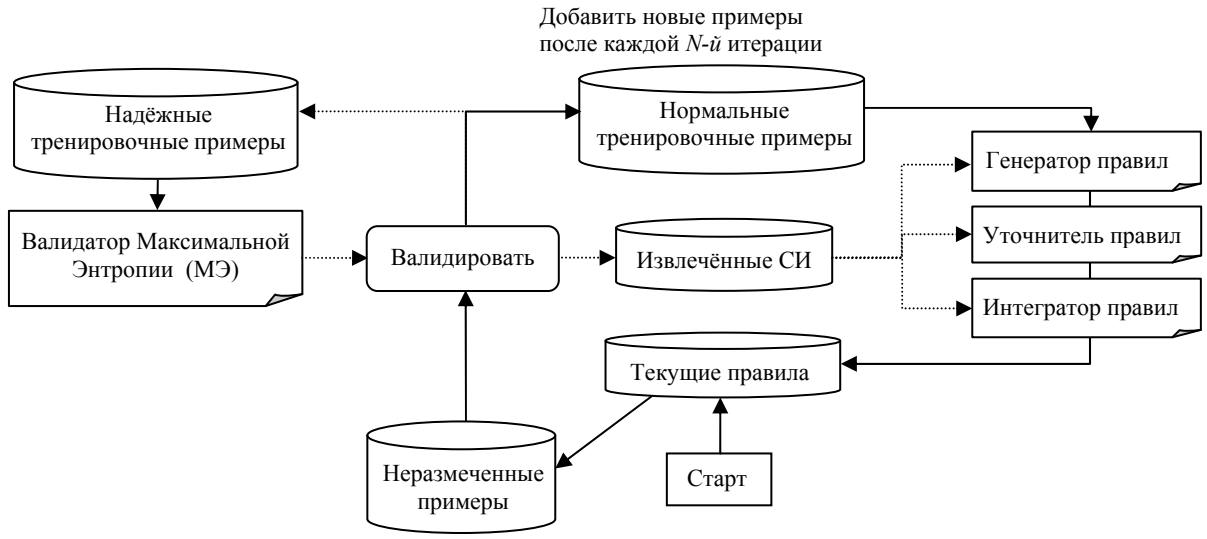

Рисунок 1. Самозагрузочная модель

Добавление всех новых примеров, извлечённых на каждой итерации самозагрузки, вызвало бы быстрое распространение ошибок и привело бы к плохому общему результату [15]. Поэтому не все новые примеры могут быть сразу добавлены к множеству надёжных тренировочных примеров в МЭ модели. Мы решили добавлять новые примеры к множеству надёжных тренировочных примеров лишь после каждых N итераций, где N выбирается динамически на основе точности новых извлечённых примеров. Чтобы оценить точность для примеров каждого класса, нам нужно измерить качество правил, использованных для их извлечения. Для этого мы используем множество уже найденных надёжных тренировочных примеров и измеряем на них качество правил согласно следующей формуле:

$$GO(\{R_1,...,R_n\}) = \frac{1}{n} Точность(R_i) \cdot Полнота(R_i) \quad (1)$$

Упорядочив правила на основе формулы (1), мы выбираем наилучшие m правил для каждого класса таких, что $GO \geq GO^0$, где GO^0 – это предопределённое пороговое значение. Затем мы добавляем новое множество неразмеченных примеров, извлечённых с помощью выбранных m правил для каждого класса, во множество надёжных тренировочных примеров нашей МЭ модели и продолжаем итерирование.

Вся самозагрузочная модель прерывается при выполнении трёх условий: (а) все неразмеченные данные для тренировки были использованы; (б) показатель F_1, базирующийся на надёжных примерах для всех натренированных типов собственных имён, не имеет дальнейшего улучшения в течение трёх последовательных итераций; и (в) превышается фиксированный лимит итераций.

2.2 МЭ Валидатор

Модель максимальной энтропии (МЭ) используется на каждой итерации самозагрузки для валидации извлечённых собственных имён (СИ) и новых тренировочных примеров. Существуют две причины для выбора МЭ модели по сравнению с известными статистическими моделями. Во-первых, МЭ позволяет использовать имеющийся набор знаний для тренировки статистической модели. Согласно [14], модель МЭ может хорошо интегрировать пересекающиеся свойства. Поэтому совместное использование биграммных, фразовых и нетекстовых свойств не приводит к значительным изменениям результатов работы МЭ. Во-вторых, МЭ не нуждается в моделях с обратной обработкой (back-off) для работы с разреженными данными, как это может потребоваться в скрытой марковской модели (СММ). Также, СММ требуют $O(|S|^2)$ переходных параметров (transition parameters) для оценивания [13], поэтому при разреженных данных количество состояний увеличивается, что приводит к серьёзной проблеме.

Основная идея МЭ [19] базируется на понятии энтропии информации: распределение вероятности P для примеров со свойствами $\{x_1 \ldots x_m\}$, принадлежащими классу C_i, кодирует состояние информации с мерой максимальной неопределённости (значением энтропии) и находится в соответствии с существующей моделью информации $\lambda_1 \ldots \lambda_m$. Этот принцип позволяет нам оценить вероятность разметки примера I, принадлежащего классу C_i, согласно формуле:

$$P(C_i \mid I) = \frac{1}{Z(I)} \exp[\lambda_1 f_1(I,C_i) + ... + \lambda_m f_m(I,C_i)] \quad (2)$$

где $f_k(I,C_i)$ является функцией с бинарным значением, которая равна ‘1’ если размеченный пример I содержит свойство x_k для класса C_i.

Для получения информационной модели с

параметрами λ_1 ... λ_m мы применяем метод обобщённого итеративного шкалирования (Generalized Iterative Scaling), описанного в [8], на множестве надёжных тренировочных примеров. $Z(I)$ является нормализующей константой, которую мы получаем в результате суммирования вероятностей $P(C_i|I)$ для всех классов.

После валидации типов СИ, извлечённых из неразмеченных примеров, мы получаем множество примеров с размеченными СИ. Это множество мы используем для оценки сходства СИ на более поздней стадии (см. 2.3.2). Заметим, что мы используем одинаковые свойства для моделирования всех примеров в рамках модели МЭ и для генерации правил. Эти свойства приведёны в Таблице 1.

2.3 Генерация правил

Для генерации правил мы используем 7 базовых лингвистических свойств. Они обобщены в Таблице 1 снизу.

Свойства	***Примеры***
Часть речи (POS)	NN, NP, PP
Лексическое представление	Car, Building
Именная группа	The President of US
Глагольная группа	Happily jogging
Главное существительное	[President]
Главный глагол	[jog]
Цифры	1,2,3

Таблица 1. Базовые лингвистические свойства

Для генерации правил мы также используем упрощённые регулярные выражения, состоящие из определённых пользователем шаблонов и булевских выражений, полученных на основе эвристических правил. Кроме этого, для поддержки генерации правил мы используем списки слов (газетёры), описывающие известные объекты. Примером сгенерированного правила является:

ЧЕЛОВЕК -> список(personTitle) .? Capwords{1,3}

Оно извлекает от 1 до 3 слов с заглавной буквы как ЧЕЛОВЕК, если перед ними расположено слово из списка *personTitle*, такое как *'Mr'*. Свойство Capwords{1,3} в этом правиле – это пример пользовательского шаблона.

Использование определённых пользователем шаблонов в наших правилах значительно улучшает качество сгенерированного набора правил. По сравнению с подходами ([10], [3], [5]), не использующих эвристических правил, ГЕРКУЛЕС начинает обучение на основе шаблонов, отражающих человеческое восприятие контекста собственного имени (СИ).

Для каждого размеченного тренировочного примера мы получаем набор контекстуальных свойств в форме:

$$<F_{-m}>...<F_{-2}><F_{-1}> <F_0> <F_{+1}><F_{+2}>...<F_{+m}>$$

где $<F_i>$ представляет набор свойств для слота i, и m означает ширину рассматриваемого окна из токенов в контексте СИ. В частности, $<F_0>$ содержит набор свойств размеченного СИ.

Процесс генерации правил стартует с измерения статистики свойств на тренировочных примерах. Статистика свойств измеряется на основе оценки вероятности, с которой свойство возникает для данного СИ в определённой позиции контекста. Пусть f_i означает i-е свойство, f_i^k является свойством f_i, возникающем в позиции k контекста СИ, C_j является произвольно выбранным типом СИ, а количество всех типов СИ равно N. Вероятность $P(f_i \mid C_j)$ мы вычисляем как:

$$P(f_i \mid C_j) = \frac{\#\, f_i \text{ в } C_j}{\sum_{l=1}^{N} \#\, f_i \text{ в } C_l} \qquad (3)$$

При оценке свойства мы учитываем также вероятность $P(f_i^k \mid C_j)$ его появления в определённой позиции k в контексте местоположения для выбранного СИ. По аналогии с (4), $P(f_i^k \mid C_j)$ мы подсчитываем как:

$$P(f_i^k \mid C_j) = \frac{\#\, f_i^k \text{ в } C_j}{\sum_{l=1}^{N} \#\, f_i^k \text{ в } C_l} \qquad (4)$$

После этого мы измеряем статистику каждого свойства среди наборов свойств в разных примерах и для разных типов СИ. Для оценки свойства f_i в позиции k выбранного СИ мы учитываем вероятность появления f_i в любой позиции контекста СИ, а также в позиции k. Она определяется на основе следующей формулы, в которой $б$ является весом для вероятностей:

$$S(f_i) = б \times P(f_i \mid C_j) + (1 - б) \times P(f_i^k \mid C_j) \qquad (5)$$

Затем мы объединяем общую информацию во всех тренировочных примерах и выбираем наиболее обобщающие свойства для старта генерации правил. Каждое сгенерированное правило R измеряется в соответствии с нашей модификацией ожидаемой ошибки Лапласа [21]:

$$L_k(R) = \frac{n+1}{n + k \cdot p + 1} \qquad (6)$$

где n (или p) означает количество отрицательных (положительных) примеров, покрытых правилом, и $k \in (0, 1)$. k используется для повышения веса отрицательным примерам по сравнению с формулой из [21]. В наших экспериментах мы установили $k = 0.5$.

Для оптимального качества правил нам нужно перебрать все возможные комбинации свойств и определить результирующую меру Лапласа для сгенерированных правил. Однако такая задача является вычислительно сложной, причём временная сложность растёт экспоненциально при увеличении количества свойств. В качестве компромисса, мы оцениваем меру Лапласа лишь для правил, сгенерированных с использованием свойств с наибольшей статистической встречаемостью среди тренировочного множества, то есть наилучших *n* свойств. В результате мы получаем множество правил с наибольшим покрытием примеров и маленькой мерой Лапласа. Базовый алгоритм для генерации правил похож на представленный в [5] и показан на Рисунке 2.

```
Цикл для каждого примера в стартовых-примерах
 Пока не покрыт(пример)
  Цикл для каждого правила в примере
   измерить(правило)
   если допустимое-правило(правило)
    тогда добавить(правило, множество-правил)
         пометить_добавление(правило,стартовые-примеры)
    иначе цикл по каждому непротестированному-свойству
         если протестировать-добавить(свойство,
              правило,множество-правил)
         тогда пометить_покрытие(правило,стартовые-
примеры)
```

Рисунок 2. Базовый алгоритм генерации правил

При генерации правил возникает проблема неоднозначности между типами СИ. Например, даже для людей трудно понять из фрагмента "*Recently Ford said that…*", принадлежит ли '*Ford*' типу *Organization* или типу *Person*. Поэтому одно лишь слово '*said*' не может быть использовано как хороший признак определения типа СИ. Тем не менее, свойство '*said*' является хорошим индикатором, что предыдущее слово с заглавной буквы является СИ-кандидатом. Чтобы различить типы таких СИ, мы разделили процесс генерации правил на две стадии: (а) генерация якорных и специфичных правил; и (б) разметка специфичных СИ.

2.3.1 Генерация якорных и специфичных правил.

Генерация правил состоит из генерации якорных правил и генерации специфичных правил. При генерации якорных правил мы рассматриваем все типы СИ C_1 … C_N как один тип *C*. Для этого мы собираем примеры всех классов СИ вместе и на их основе генерируем якорные правила, используя обсуждённый выше базовый алгоритм генерации правил на Рисунке 2. Сгенерированные якорные правила состоят из *контекстуальных якорей* и *якорей СИ*. Контекстуальные якоря генерируются на основе контекстных свойств СИ, например слова "*said*". Для генерации якоря СИ мы используем специфические свойства СИ, такие как часть речи *NNP*, или *ЗаглавнаяБуква*. Якорные правила позволяют системе достичь высокой полноты, сохраняя разумную точность. Базируясь на этих правилах, ГЕРКУЛЕС извлекает все кандидаты СИ, не зная заранее их класса. Примером якорного правила является

СИ -> слово (Corp | Corporation)

которое извлекает любое слово, предшествующее *'Corp'* или *'Corporation'*, в качестве СИ-кандидата.

Сгенерированные якорные правила позволяют извлечь большое количество СИ с подтверждёнными границами. На следующей стадии мы используем результаты работы якорных правил для генерации специфичных правил, которые уточняют тип СИ. Чтобы избежать больших ошибок, для генерации специфичных правил мы используем только те якорные правила, которые были отобраны на предыдущей итерации. Генерация специфичных правил проводится также на основе базового алгоритма генерации правил на Рисунке 2.

2.3.2 Разметка специфичных СИ.

В предыдущем примере мы извлекаем слово '*Ford*' в качестве кандидата, используя правило для *Organization*, базирующееся на надёжных тренировочных примерах и имеющее точность P_i. При этом, если значение P_i превышает предустановленное пороговое значение P^0, то мы можем разметить это СИ и добавить в набор извлечённых СИ после валидации МЭ. В случае, если тип СИ остаётся неоднозначным, мы сравниваем его со всеми СИ, извлечёнными на предыдущих и текущей итерации самозагрузки. Сходство между извлечёнными СИ измеряется как:

$$Sim(CИ_X, CИ_Y) = \frac{|T \cap W|}{|T| \cdot |W|} \qquad (7)$$

где *T* является набором слов в первом $CИ_x$, и W является набором слов во втором $CИ_y$. СИ, извлечённые с использованием измерения подобия по формуле (7), не добавляются в набор извлечённых СИ, так как это лишь приблизительное соответствие.

В качестве примера, мы могли ранее извлечь '*Ford Corp.*' и '*Ford Company*' как *Organization* со значениями уверенности *Conf('Ford_Corp'| Organization)* и *Conf('Ford_Company'|Person)*. В этом случае мы измеряем сходство '*Ford*' с каждым из извлечённых СИ для класса *Organization* и подсчитываем комбинированное значение уверенности *Conf('Ford'|Organization)* для '*Ford*' используя:

$$Conf(X \mid CИ_w) = \begin{cases} P_i(X), \text{если } P_i(X) > P^0 \\ \frac{1}{|Y|} \sum_{Y_i \in Y} Sim(X, Y_i) Conf(Y_i), \text{иначе} \end{cases}$$

где Y_i является извлечённым примером *i* типа $CИ_w$ и $P_i(X)$ является точностью правила для извлечения *X*

как $СИ_w$.

Формула выше базируется на эвристике “один смысл в тексте” [25]: два СИ-кандидата с одинаковым написанием имеют одинаковый смысл в одном тексте. В конце вычисления, тип СИ с наибольшим значением $Conf(X|СИ_w)$ приписывается к извлечённому СИ.

2.3.3 Обобщение правил.

Когда эксперты создают эвристические правила, они стараются описать всю предметную область. При этом, даже если компьютеру на вход дать большое количество тренировочных текстов, какие-то из ключевых знаний человека о предметной области могут отсутствовать. Например, для человека очевидно, что все люди подразделяются на мужчин и женщин. То есть, если перед именем какого-то человека стоит слово *‘Mrs’*, то какое-то другое (мужское) имя может начинаться со слова *‘Mr’*. Однако у компьютера таких знаний нет, и потому сгенерированные правила зачастую покрывают меньшее количество тренировочных примеров, чем созданные человеком. Поэтому перед нами стояла дилемма: обобщить ли сгенерированные специфичные правила для покрытия большего количества примеров, или же оставить их неизменными, чтобы не ухудшать точность извлечения примеров с помощью этих правил. Мы решили оставить как первоначальные правила, так и сгенерировать обобщённые правила для дальнейшей интеграции правил на более поздней стадии. Для обобщения специфичных правил мы используем набор общих свойств, определённых динамически с помощью обучения на основе алгоритма, приведённого на Рисунке 3:

```
Цикл по каждому правилу на множестве-правил
  Цикл по свойству в наборе-свойств в тренировочных примерах
    заменяемые-свойства =
          протестировать-заменить(правило,свойство,
                                  надёжные-примеры)
  Цикл по каждому элементу в заменяемых-свойствах
      пометить(правило, элемент ∈ свойство)
      новоеПравило = заменить(правило,элемент,свойство)
      добавить(новоеПравило, множество-правил)
```

Рисунок 3. Алгоритм обобщения правил

На основе этого алгоритма мы генерируем правила, в которых какие-то из свойств заменены для увеличения полноты, при этом лишь незначительно уменьшая точность правила. Например, в нескольких правилах список *femaleTitle* (например *‘Miss’*, *‘Mrs’*) и лексическая форма *‘Mr’* заменяются на *personTitle*. Для обобщения лексических шаблонов в правилах мы также применяем лексические цепочки с использованием тезауруса WordNet.

2.4 Уточнение правил

Другая проблема состоит в том, что некоторые из сгенерированных правил являются слишком общими для извлечения СИ и имеют низкую точность. Источником таких ошибок являются эвристические шаблоны в правилах, которые не подобраны под конкретную предметную область. Примером такого сгенерированного правила является:

Person -> nonCapital Capwords{1,3} (R1)

которое извлекает от 1 до 3 слов с заглавной буквой после незаглавной буквы как СИ с типом *Person*.

Для предложения *“According to Mr Clinton”*, правило *R1* будет неправильно извлекать *“Mr Clinton”* как *Person*, что неверно, так как *‘Mr’* не является частью СИ. Тем не менее, для другого предложения *“According to Bill Clinton”*, правило *R1* работает корректно. Основная причина генерации таких правил состоит в недостатке отрицательных примеров во время тренировочного процесса.

Для решения этой проблемы мы используем множество надёжных примеров, которые имеют предопределённые классы. Так как надёжные примеры уже размечены на предыдущей итерации, на текущей итерации к ним можно применить правила с низкой точностью и извлечь подмножество тренировочных примеров, используемых для уточнения. Извлечённые примеры, которые правильно промаркированы правилом, мы размечаем как положительные. В то же время, любой пример, промаркированный правилом неправильно, размечается как отрицательный.

После получения подмножества требуемых примеров нужно удостовериться, что правила не противоречат друг другу. Для этого мы выполняем прямое и обратное обучение для уточнения правил. Прямое обучение старается решить эту проблему добавлением дополнительных ограничений, тогда как обратное обучение добавляет новые исключения к правилу. Необходимым ограничением для обратного обучения является то, что мы не можем выбирать свойства из первоначальных правил.

2.4.1 Прямое обучение.

Целью прямого обучения является добавление ограничений к правилу без уменьшения его полноты. Правила на этой стадии тренируются с использованием положительных и отрицательных примеров. По сравнению с первоначальной генерацией правил, на этой стадии большее количество отрицательных примеров возникает в результате неправильного распознавания границы СИ. Как и при генерации правил, мы выбираем свойства на основе формулы (5), добавляя выбранные свойства к существующему правилу в конъюнкции с существующими свойствами. Чтобы оценить, насколько улучшилось правило в результате его уточнения, мы используем значение L_k согласно формуле (6). Например, правило R1 в

результате превратилось в такое уточнённое правило:

Person -> nonCapital & PoS(предлог) Capwords{1,3} (R2)

Теперь оно вместо использования любого слова со строчными (*nonCapital*) буквами перед именем человека допускает только те слова со строчными буквами, которые также являются предлогами (например, "*to*"). Мы добавляем уточняющие свойства до тех пор, пока улучшается точность правила и не ухудшается текущее значение полноты, используя базовый алгоритм генерации правил на Рисунке 2. Все измерения точности и полноты базируются на наборе надёжных примеров.

2.4.2 Обратное обучение.

Для обратного обучения мы используем модификацию формулы (5) для подсчёта наиболее перспективных свойств в отрицательных примерах вместо положительных:

$$S(f_i) = б \times P(f_i \mid not\ C_j) + (1 - б) \times P(f_i^k \mid not\ C_j) \quad (8)$$

После этого мы используем найденные свойства для улучшения правил с помощью добавления к ним отрицания или исключения первоначальных свойств. Для этого мы используем базовый алгоритм генерации правил, поменяв на его входе отрицательные примеры на положительные. Свойства в правилах мы выбираем таким образом, чтобы они уменьшали значение ошибок при оценке правила по формуле (6). Например, многие ошибки при работе правила R2 возникают из-за попадания таких слов как *'Mr.'* в собственное имя. Однако ошибочно извлечённые слова в этих примерах хорошо описываются списком *personTitle*. Поэтому на стадии обратного обучения для правила R2 генерируется уточнение в виде добавления списка *list(personTitle)* в качестве отрицательного свойства:

ЧЕЛОВЕК -> nonCapital & PoS(Предлог)
[Capwords{1,3} & !список(personTitle)] (R3)

В итоге, стадия прямого и обратного обучения позволяет улучшить правила, которые при высокой полноте имеют низкую точность. Результирующая точность работы всей системы при этом также улучшилась.

2.5 Интеграция правил

После генерации набора правил и их уточнений нам нужно интегрировать полученные правила с уже существующим списком правил. Одна из проблем при слиянии набора правил состоит в том, что сгенерированные правила могут конфликтовать друг с другом. Более того, конфликтующие правила могут извлекать конфликтующие типы СИ. Поэтому важной задачей является уменьшение неоднозначности правил при их интеграции. Наш алгоритм интеграции правил состоит из двух частей. Сначала для каждого правила из множеств S_1 и S_2 он оценивает их приоритеты. После этого модуль извлечения СИ решает, какое из правил должно быть использовано для извлечения СИ в случаях конфликтов.

Наш алгоритм интеграции правил применяется в двух различных случаях: (а) интеграция эвристических правил со сгенерированными правилами; и (б) интеграция различных наборов сгенерированных правил. Вход в алгоритм состоит из любых двух наборов правил S_1, S_2 и множества тренировочных примеров I. Дополнительно, мы придаём более высокий приоритет эвристическим правилам, созданными людьми. Это сделано заданием более высокого веса для свойств или внешнего знания, найденного в наборе надёжных эвристических правил. Выходом алгоритма является список решений (decision list) L всех правил. Пусть H_i, M_i означают количество свойств для эвристических и сгенерированных правил, соответственно. Предопределённый вес значимости для эвристических и сгенерированных свойств мы обозначаем как ω_h и ω_m, при этом $\omega_h > \omega_m$. Также, пусть N_i является количеством итераций и P_i, R_i являются точностью и полнотой правила $Rule_i$, соответственно. Алгоритм для интеграции правил показан на Рисунке 4:

1. Задать $L = \{\}$, $S = S_1 \cup S_2$
2. Задать вес для правил:
$$W(Rule_i) = (1 + \omega_h H_i + \omega_m M_i)\log(1 + N_i)$$
3. *повторить*
- Ранжировать правила в соответствии с точностью и полнотой правила на примерах I:
$$F_\beta(Rule_i) = \frac{(1+\beta^2)P_i R_i}{\beta^2 P_i + R_i} W(Rule_i)$$
- Добавить лидирующее правило R_0 в L:
$$L = L \cup \{R_0\}, \quad S = S \setminus \{R_0\}$$
-Удалить положительные примеры, покрытые R_0:
$$I = I \setminus I^+(R_0)$$
пока ($F_\beta > F^0$) // F^0 – *пороговое значение*
4. Удалить последнее добавленное правило:
$$L = L \setminus \{R_0\}$$
5. Возвратить L

Рисунок 4. Алгоритм для интеграции

Все правила этого алгоритма проранжированы в соответствии с метрикой $F_в$ на каждой итерации. Правило с наибольшим $F_в$ затем выбирается и добавляется к списку решений (decision list). После этого все примеры, покрытые выбранным правилом, удаляются из тренировочного набора и происходит следующая итерация. На последней итерации мы получаем список правил L, состоящий из отобранных правил входных наборов S_1 и S_2. Этот список затем используется для извлечения СИ.

3. Эксперименты

Тестирование самозагрузочного процесса для генерации правил мы проводили на основе набора

данных MUC-7[2], размеченный типами *Date, Location, Money, Organization, Person и Percent*. Мы провели 5 экспериментов с целью ответить на следующие вопросы: (а) какое количество первоначальных правил является оптимальным; (б) каким образом лучше выбирать первоначальные правила; (в) как изменяется результат при использовании уточнения правил; (г) к какому изменению результата ГЕРКУЛЕСа приводит интеграция правил; и (д) сравнить ГЕРКУЛЕСа с результатами других лучших самозагрузочных систем извлечения СИ.

Базируясь на эмпирических исследованиях, мы обнаружили что из-за некоторых неточных правил в систему могут постоянно добавляться неправильные примеры. На наш взгляд, это ухудшение происходит из-за распространения ошибок в наборе правил. Из-за этих ошибок качество множества примеров также постепенно ухудшается после каждого цикла. Так как мы заметили, что результат системы ухудшается при более 250 итераций, мы решили зафиксировать количество итераций на уровне 250.

3.1 Количество первоначальных правил

Чтобы определить, какое минимальное количество эвристических правил достаточно для запуска Геркулеса, мы сравнили результат для различного количества начальных эвристических правил. Минимальное количество эвристик, которое мы использовали – это 10 эвристических правил для всех классов, полученных с помощью ручного отбора. На Рисунке 5 показаны результаты этого эксперимента, из которых следует, что примерно 30 начальных эвристических правил оптимальны для достижения результата. Все эксперименты на дальнейших стадиях базировались на этом количестве. В этих экспериментах мы учитывали уточнение и интеграцию правил.

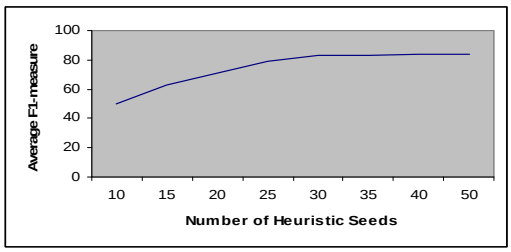

Рисунок 5. Средняя метрика F_1 (Начальные правила)

3.2 Выбор первоначальных правил

Другим важным вопросом является, насколько качество первоначальных правил влияет на итоговый результат. Мы использовали 2 способа выбора первоначальных эвристических правил – базируясь на метрике F_1 или случайно. Наши результаты в Таблице 2 демонстрируют, что выбор начальных правил на основе метрики F_1 приводит к заметно лучшему результату, чем в случае случайного выбора первоначальных правил.

	Случайный выбор			*Выбор на основе F_1*		
Тип СИ	***P***	***R***	***F_1***	***P***	***R***	***F_1***
LOC	0.70	0.63	0.66	0.76	0.96	0.84
ORG	0.41	0.64	0.50	0.64	0.54	0.59
PERSON	0.81	0.79	0.80	0.81	0.89	0.85

Таблица 2. Результаты выбора первоначальных правил

Анализ результатов показывает, что существуют две основных причины, почему выбор первоначальных правил на основе метрики F_1 приводит к лучшему результату. Во-первых, случайный выбор начальных правил может привести к тому, что два правила извлекают похожие примеры. Во-вторых, произвольно выбранные правила могут конфликтовать с правилами для других классов СИ и приводить к конфликтам из-за неоднозначности выбора. По этим причинам при использовании метрики F_1 для выбора начальных правил система извлекает менее похожие тренировочные примеры для разных классов СИ, что приводит к более высокому качеству сгенерированных правил.

3.3 Уточнение правил

В этой секции мы проводим экспериментальную проверку, насколько необходимо проводить уточнение правил. Мы применяли правила, используя и не используя процедуру уточнения правил и измеряли результирующие показатели полноты *R* и точности *P*. Результаты в Таблице 3 показывают, что уточнение правил действительно помогает улучшить точность за счёт полноты. В целом результат по метрике F_1 улучшился после уточнения правил.

	Без уточнения правил			*С уточнением правил*		
Тип СИ	***P***	***R***	***F_1***	***P***	***R***	***F_1***
LOC	0.62	0.83	0.71	0.68	0.74	0.71
ORG	0.47	0.62	0.53	0.53	0.58	0.55
PERSON	0.68	0.86	0.76	0.72	0.75	0.73

Таблица 3. Результаты уточнения правил

3.4 Интеграция правил

В наших экспериментах по интеграции правил мы использовали частичное множество эвристических правил из существующей эвристической системы СИ и множество сгенерированных правил. Наша базовая конфигурация состояла из прямого применения правил и множества вручную составленных приоритетов для типов СИ, которые Геркулес должен был извлечь. Эти приоритеты были получены эвристически, базируясь на наблюдаемых конфликтах между эвристическими правилами в

[2] Доступен по адресу http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2001T02

тренировочных примерах. Например, эвристические правила для типа СИ *Person* более точны на тренировочных примерах, чем правила, которые извлекают ОРГАНИЗАЦИИ. Поэтому в случае конфликтов правил мы отдаём предпочтение типу *Person* по сравнению с типом Организация. Наши результаты в Таблице 4 показывают, что интеграция правил на основе алгоритма, представленного на Рисунке 4, улучшает результат от 4 до 13 процентов согласно метрике F_1.

	Базовая конфигурация			***Конфигурация с интеграцией правил***		
Тип СИ	***P***	***R***	***F_1***	***P***	***R***	***F_1***
LOC	0.68	0.74	0.71	0.76	0.96	0.84
ORG	0.53	0.58	0.55	0.64	0.54	0.59
PERSON	0.72	0.75	0.73	0.81	0.89	0.85

Таблица 4. Результаты интеграции правил

3.5 Сравнение результатов с другими системами

На основе наших результатов можно сделать вывод, что отбор первоначальных данных по метрике F_1, а также включение стадий уточнения и интеграции правил приводят к наилучшему результату ГЕРКУЛЕСа. В целом, его результаты выше существующих самозагрузочных методов, которые тестировались на наборе текстов MUC-7. Таблица 5 показывает наши результаты по сравнению с [15], которая является одной из лучших недавних систем, близко связанной с нашей работой.

	ГЕРКУЛЕС			***Система NIU***		
Тип СИ	***P***	***R***	***F_1***	***P***	***R***	***F_1***
DATE	0.96	0.87	0.91	-	-	-
LOC	0.76	0.96	0.84	0.83	0.82	0.82
MONEY	0.88	0.92	0.90	-	-	-
ORG	0.64	0.54	0.59	0.57	0.49	0.53
PERSON	0.81	0.89	0.85	0.86	0.89	0.88
PERCENT	0.91	0.85	0.88	-	-	-
Overall	*0.83*	*0.84*	*0.83*	-	-	-

Таблица 5. Результаты ГЕРКУЛЕСА

4. **Похожие работы**

Авторы нескольких похожих исследований самозагрузочного извлечения информации (ИИ) изучали, каким образом можно использовать извлекаемые ключевые слова и шаблоны. Riloff [20] предложила использовать эвристические понятийные шаблоны для создания словарей-газетеров, используя заранее классифицированные тренировочные тексты и неразмеченный корпус текстов. Тексты тренировочного корпуса классифицировались как положительные, если в них находится извлекаемое событие. Yangarber et al [24] пытались найти шаблоны для извлечения без использования заранее размеченного корпуса текстов и стартуя генерацию правил с набора первоначальных шаблонов. Одно из основных различий между такими системами и ГЕРКУЛЕСом состоит в способе использования эвристического знания. Вместо использования эвристических правил для выполнения основной задачи извлечения, мы используем эвристические правила только в качестве начальных данных для самозагрузки и также как источник внешнего знания. Использование начальных примеров СИ для старта обучения самозагрузочной системы [15] является наиболее близким к нашей работе. Однако в отличие от нас они не используют эвристические правила на старте тренировочного процесса.

5. **Заключение**

В этой статье мы презентовали новую самозагрузочную систему под названием ГЕРКУЛЕС для задачи извлечения СИ из текстов на естественном языке. Наш подход позволяет нам стартовать лишь с небольшого количества эвристических правил, в то же время, получая сравнимые результаты с существующими системами генерации правил. В отличие от других существующих систем, использующих лишь статистический подход, наша модель по генерации правил позволяет нашей системе автоматически корректировать ошибочные правила самостоятельно. Это сделано с помощью процедуры прямого и обратного уточнения на каждой итерации. Мы также использовали обучение общих правил и уменьшили разницу между сгенерированными и созданными вручную правилами, что не сделано в существующих методах. Более того, наши правила понятны для людей на каждой стадии и могут помочь в больших задачах на понимание документов, информационного поиска и вопросно-ответных задач. Одним из направлений улучшения ГЕРКУЛЕСа, которое мы планируем исследовать в ближайшем будущем – можно ли добавить дополнительные семантические свойства в извлечённые шаблоны с помощью семантических парсеров.

Литература

[1] Blum, A. and Langley, P. 1997. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2): 245–271.

[2] Blum, A. and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proc. of COLT '98*.

[3] Borthwick, A. 1999. A Maximum Entropy Approach to Named Entity Recognition. PhD thesis, New York University.

[4] Califf, M.E. and Mooney, R. J. 1999. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proc. of 16th NCAI, 328-334*.

[5] Ciravegna, F. 2001. An Adaptive Algorithm for Information Extraction from Web-related Texts. In *Proc. of IJCAI-2001 Workshop*.
[6] Cohen, W. W. and Sarawagi S. 2004. Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods. In *Proc. of KDD 2004*, 89-98.
[7] Cohn, D.; Atlas, L.; and Ladner R. 1994. Improving generalization with active learning. *Machine Learning* 15.
[8] Darroch, J. N. and Ratcliff, D. 1972. Generalized Iterative Scaling for Log-Linear Models. *Annals of Mathematical Statistics*, 43(5): 1470-1480
[9] Freitag, D. 1998. Multistrategy Learning for Information Extraction. In *Proc. of the 15th ICML.*
[10] Freitag, D. and McCallum, A. 1999. Information Extraction with HMMs and Shrinkage. *Workshop on Machine Learning for Information Extraction, AAAI*.
[11] Jones, R.; McCallum, A.; Nigam, K.; and Riloff, E. 1999. Bootstrapping for Text Learning Tasks. In *IJCAI-99 Workshop, 52-63.*
[12] Lin, D. K. 1997. Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. In *Proc. of ACL-97.*
[13] McCallum, A..; Freitag, D.; and Pereira, F. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proc. ICML 2000*, 591–598
[14] Nigam, K.; Lafferty, J.; and McCallum, A. 1999. Using maximum entropy for text classification. In *Proc. of the IJCAI-99 Workshop*, 61–67.
[15] Niu, C.; Li, W.; Ding J.; and Srihari, R. K. 2003. A bootstrapping approach to named entity classification using successive learners. In *Proc. of ACL-2003*.
[16] Ng, V. and Cardie, C. 2003. Bootstrapping Coreference Classifiers with Multiple Machine Learning Algorithms. In *Proc. of EMNLP-2003*, ACL.
[17] Peshkin L. and Pfeffer A. 2003. Bayesian Information Extraction Network. In *Proc. of 18th IJCAI*.
[18] Pierce, D. and Cardie C. 2001. Limitations of Co-Training for Natural Language Learning from Large Datasets. In *Proc. of EMNLP-2001*.
[19] Pietra, S. D.; Pietra V. D.; and Laferty J. 1997. Inducing Features of Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4): 380-393.
[20] Riloff E. 1996. Automatically Generating Extraction Patterns from Untagged Text. In *Proc. of 13th AAAI*, 1044-1049.
[21] Soderland S. 1999. Learning Information Extraction Rules for Semi-Structured and Free Text, *Machine Learning*, 34(1-44): 233-272.
[22] Thompson C.A.; Califf M.E.; Mooney J.R. 1999. Active Learning for Natural Language Parsing and Information Extraction. In *Proc. 16th IMLC*, 406-414
[23] Voorhees, E. M. 1993. Using WordNet to disambiguate word sense for text retrieval. In *Proc. of ACM SIGIR*,171-180
[24] Yangarber, R., Lin, W. and Grishman, R. 2002. Unsupervised learning of generalized names. In *Proc. of COLING 2002*.
[25] Yarowsky, D. 1995. Unsupervised word-sense disambiguation rivaling supervised methods. In *Proceedings of ACL-1995*. Cambridge, MA, 189-196.
[26] Zhou G.D and Su J. 2002. Named Entity recognition using an HMM-based chunk tagger. In *Proc. of 40th Annual Meeting of ACL*, 473-480.

Rule Integration for Bootstrapping-Based Information Extraction

© Mstislav Maslennikov

Free text poses serious challenges in comparison with semi-structured text for Information Extraction because it requires significant amount of annotated data for training. We introduce a novel bootstrapping rule-based system which aims to integrate human heuristic knowledge and machine-learned knowledge. Starting with only a few heuristic seed rules and instances, our system gradually mines new rules, new training instances and improves the learning process in a bootstrapping framework.

♣ Автор выражает благодарность Чуа Тат-Сенгу и Виктору Го за помощь в написании этой статьи.