УДК 004.912

Задача извлечения именованных сущностей в русскоязычном тексте

М. Д. Аверина
Ярославский государственный университет им. П. Г. Демидова
E-mail: maverina518@gmail.com

Аннотация

В статье рассматривается задача извлечения именованных сущностей из текста. Для решения данной задачи часто используется метод *CRF* — условные случайные поля. В статье исследуется вопрос решения задачи извлечения именованных сущностей для текста на русском языке на основе метода условных случайных полей. При этом были использованы различные подходы к признаковому описанию текста, был проведен сравнительный анализ моделей при использовании различных признаков и методов оптимизации. Лучшая модель показала качество *F1* равное $75\,\% - 99\,\%$ в зависимости от сущности.

Ключевые слова: распознавание именованных сущностей, *NER*, условные случайные поля, *CRF*, токенизация, нормализация слов, *word2vec*, *fastText*, *F1*.

Введение

Одна из важнейших задач – сбор и анализ статистических данных нормативных документов является достаточно трудоемкой для специалистов. На данный момент, в условиях повсеместного внедрения электронного документооборота, данная задача особенно актуальна. Автоматизация процесса анализа текстов – задача распознавания именованных сущностей (named entity recognition, *NER*) [**base**] позволит оптимизировать работу многих специалистов как по временным, так и по качественным показателям.

Задача извлечения сущностей

Задача NER заключается в выделении определенных непрерывных фрагментов текста (именованных сущностей). Например, имеется новостной текст, в котором необходимо выделить некоторый

заранее зафиксированный набор сущностей: персоны, локации, организации, даты и т.д. Соответственно алгоритм должен определить, что участок текста «1 января 1997 года» является датой, «Кофи Аннан» – персоной, а «ООН» – организацией (рис.1).

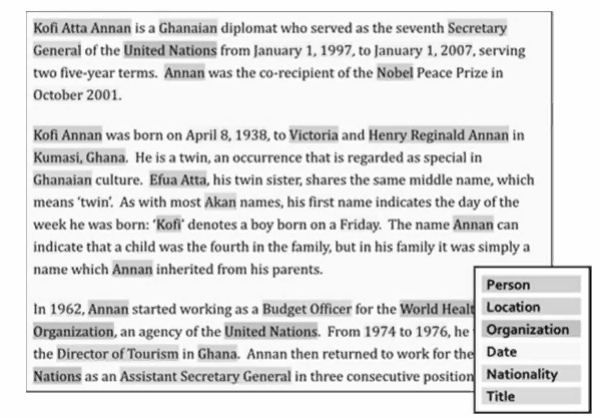

Рис. 1: Пример работы NER.

Данная статья посвящена задаче выделения именованных сущностей для рускоязычных текстов. В качестве тестовых данных была использована открытая база судебной статистики [**CourtsData**]. Такой выбор был сделан в силу большого количества доступных документов, а также соответствующего задаче характера текстов (наличие множества различных имён, дат, наименований, сумм и т.д.)

На основе данной базы была сформирована и размеччена выборка из 500 файлов, которая была размечена на инструменте BRAT. BRAT (brat rapid annotation tool) — онлайн-инструмент для разметки письменных текстов. Были выделены следующие сущности: номер документа (doc num), суд (court), судья (judge), дата суда (date court), истец (plaintiff), ответчик или представитель (defendant), решение суда (court decision), статья или тип штрафа (payment fine), сумма выплаты (payment amount), срок обжалования (appeal time).

Предварительная обработка данных

Первым этапом решения задачи является предобработка данных, которая включает в себя разбиение текста на слова, удаление

ненужных символов, и извлечение признаков слов. Токенизация — это процесс разбиения текстового документа на отдельные слова, которые называются токенами. Для начала весь текст был разбит на токены [**ner**], при этом удалены символы, не несущие смысловой нагрузки, и слова приведены к нижнему регистру.

Одим из наиболее очевидных признаков для решения задачи классификации именованных сущностей является вычленение информации при помощи регулярных выражений. Например, информация о наличие заглавной буквы или знаков припинания после слова заносится в «словарь символов». «Словарь символов» — структура данных, которая хранит в себе информацию о находящихся рядом знаках препинания и о регистре букв слова. Например почта *«v1@mail.com,»* будет разбита на *«v1@»* и *«mail.com,»*, где @ идентифицируется как слово-спецсимвол, а запятая заносится в «словарь символов». Символ @, позволяет классифицировать строку как почтовый адрес. Используемые нами спецсимволы: @, #, №, %, /.

Приведем список признаков в «словаре символов»:

- первая буква большая, остальные маленькие;
- все буквы маленькие;
- все буквы большие;
- первая буква маленькая;
- наличие запятой или точки в конце или начале слова и т.д.

Также в качестве признака можно использовать «само слово». Стоит отметить, что данный признак не является релевантным, если в обучающей выборке часто упоминается одна и та же фамилия или одно и тоже название организации, поскольку алгоритм «зацепляется» за конкретное слово. Например для тестов судов, фамилия судьи встречается в тексте несколько раз.

Большинство слов в тексте имеет падеж или склонение, что в свою очередь усложняет работу алгоритма. Одим из способов решения данной проблемы является нормализация слов — приведение их к стандартному виду (например, лосями → лось, мыла → мыть, зеленого → зеленый). При этом число и род можно использовать как признаки. Также часть речи (существительное, предлог и т.д.), является неплохим признаком при распознавании сущностей. Отметим, что в случае слов-спецсимволов, можно каждому символу задать уникальные значения частей речи. Например, для символа % морфологией будет *PERCENT*.

Другим распространенным подходом к вычислению признаков является векторизация слов — представление слова в виде вектора [**w2v**]. После предобработки текста слова можно закодировать векторами чисел, которые в дальнейшем удобно использоваться при обучении. Word2Vec работает с большими текстовыми данными и по определенным правилам присваивает каждому слову уникальный набор чисел — семантический вектор. В последнее время все более популярным становится подход к векторизации текста, при котором Word2Vec дополняется различными улучшениями. Наиболее часто используемым улучшением — fastText. FastText исправляет недостаток Word2Vec: если обучение модели начинается с прямого кодирования одного D-мерного вектора, то игнорируется внутренняя структура слов. Вместо прямого кодирования слов, fastText предлагает изучать N-граммы символов и представлять слова как сумму векторов N-грамм.

Для улучшения качетсва обучения, можно учитывать не только признаки текущего токена, но и соседних токенов, тем самым учитывая контекст. В дальнейшем для используемых признаков будем использовать обозначения: «список символов» — r, «само слово» — v, часть речи — m, нормализация — n, Word2Vec — w, FastText — f. А цифра после буквы будет обозначать количество рассматриваемых соседей влево и вправо. Например, *f3* означает, что использовалось 7 признаков FastText — для текущего слова и для 3-х соседей с каждой стороны.

Прогнозирование тегов при помощи CRF

Наиболее популярный инструмент для решения задачи NER является метод CRF (conditional random fields) [**HabrCRF**]. Алгоритм оптимизирует всю цепочку меток целиком, а не каждый элемент по отдельности, учитывает любые особенности и взаимозависимости в исходных данных, а также он хорошо работает в связке с рекуррентными нейросетями, моделирует совместное распределение на всей последовательности выходов сети одновременно.

Линейный CRF хорошо подходит для решения задач сегментации и разметки последовательности, например: автоматическое выделение ключевых слов из текстов, выделение именованных сущностей (классификация сущностей), анализ тональности, автоматическое распознавание речи.

Процесс обучения имеет большую вычислительную сложность, а именно $O(mNTQ2nS)$ где:

- m — количество тренировочных итераций;
- N — количество обучающих последовательностей;
- T — средняя длина обучающей последовательност;
- Q — количество выходных классов;
- n – количество признаков в обучающей матрице;
- S – время работы алгоритма оптимизации на каждом шаге.

На практике время обучения даже выше за счет всевозможных дополнительных операций таких, как сглаживание, преобразование данных из одного формата в другой. Отметим, что при увеличении количества признаков, например, за счет соседних слов, время обучения значительно увеличивается.

Метрика F1

После того как CRF обучен необходимо оценить качество его работы. Метрики точности (Precision, P) и полноты (Recall, R) дают исчерпывающую оценку качества, но, как правило, приходится балансировать между двумя этими метриками. Если повысить полноту, делая решение более «оптимистичным», это приводит к падению точности из-за увеличения числа ложно-положительных ответов. Если же наоборот, то при росте точности происходит одновременное падение полноты из-за отбраковки какого-то числа правильных ответов. Поэтому удобно использовать одну величину, так называемую метрику *F1* — среднегармоническое между полнотой и точностью:

$$F1 = 2\frac{PR}{P+R} \tag{1}$$

Для оценки качества полученных результатов было решено использовать F1, но Precision и Recall вычислять, путем объединения соседних слов с одним тегом в одну сущность. Поскольку одна сущность может состоять из нескольких токенов, то стоит при оценке качества учитывать ее целиком.

Результаты

Рассмотрим результаты обучения (400 файлов) и тестирования (100 файлов) на данных судебных протоколов: наилучшего качества классификации можно добиться за счет подбора параметров

классификатора. Необходимо выбрать наиболее эффективный и быстрый оптимизатор для CRF. В таблице 1 приведен анализ оптимизаторов, при одном фиксированном наборе признаков. Ниже приведены алгоритмы оптимизации CRF.

- *lbfgs* — градиентный спуск с использованием метода *L-BFGS*;
- *l2sgd* — стохастический градиентный спуск с регуляризации *L2*;
- *pa* — усредненный персептрон;
- *ap* — passive aggressive;
- *arow*— адаптивная регуляризация.

Алгоритм оптимизации	Долгое время работы	*F1* на лучшей сущности	*F1* на худшей сущности	Примечания
lbfgs	+	1	0.7	Время обучения более 5 часов.
l2sgd	+	1	0	Классифицирует все, как самый. объемный класс
pa	–	1	0.45	Время обучения не более 30 минут.
ap	–	0.83	0.24	–
arow	–	0.79	0.31	–

Таблица 1: Сравнительный анализ алгоритмов оптимизации.

Как видно из таблицы 1 лучший результат показали алгоритмы оптимизации *pa* и *lbfgs*. Стоит отметить, что *lbfgs* более трудоемкий по сравнению с *pa*. В дальнейших тестах для выбора оптимальных признаков будут использоваться данные два оптимизатора.

Заметим, что наименьший разброс F1 принимает при признаках $(r3, v3)$ с оптимизатором *lbfgs*, при этом демонстрируя достаточно высокий средний результат (таблица 2). Однако, время обучения такой модели более пяти часов. При необходимости уменьшить вре-

мя обучения наиболее оптимальным решением будет обучение модели на признаках $(r3, v1)$ с оптимизатором *pa*.

Алгоритм / Признаки	*pa*	*l2sgd*	*lbfgs*
$(r3, v3)$	–	–	doc num 0.993 defendant 0.751 5.55.36
$(r3, v1)$	judge 0.906 defendant 0.455 0.07.58	judge 0.867 defendant 0.116 0.4.11	doc num 0.973 defendant 0.685 3.32.49
$(w1, v1, m1)$	court 0.827 defendant 0.331 0.05.20	judge 0.751 court 0.013 0.3.14	judge 0.971 defendant 0.541 3.03.01
$(r3, m3)$	judge 0.692 payment 0.412 0.33.58	date court 0.746 court 0.126 0.12.11	judge 0.818 defendant 0.285 3.58.24

Таблица 2: Сравнительный анализ F1.
Сверху вних в ячейках указаны: F1 на лучшей сущности, F1 на худшей сущности, время обучения модели.

Заключение

Подведем итоги, в статье приведет подход решения задачи NER, в котором был приведен набор различных наборов признаков для текста, а также проведен сравнительный анализ алгоритмов оптимизации и признаков, который показал, что алгоритм *pa* с признаками $(r3, v1)$ показал наилучший результат по времени и F1 в совокупности. Алгоритм *lbfgs* с признаками $(r3, v3)$ продемонстрировал лучший результат по F1, но оказался трудоемким и долгообучаемым. Стоит отметить, что лучший результат показал алгоритм *pa* по времени. Но для более точного прогнозирования будет необходимо использовать *lbfgs*.

Для дальнейшего улучшения результатов планируется выделять еще новые признаки, а также можно попробовать использовать другие методы классификации и усовершенствование архитектуры классификатора на основе biLSTM и BERT.