

УДК 214.16

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 (*NER*). Для решения данной задачи часто используется метод *CRF* (*conditional random fields*) — условные случайные поля. В статье исследуется вопрос решения задачи *NER* для текста на русском языке на основе метода *CRF*. 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* (*named entity recognition*) заключается в выделении определенных непрерывных фрагментов текста (именованных сущностей). Например, имеется новостной текст, в котором необходимо выделить некоторый заранее зафиксированный набор сущностей:

© Аверина М. Д., 2020

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).

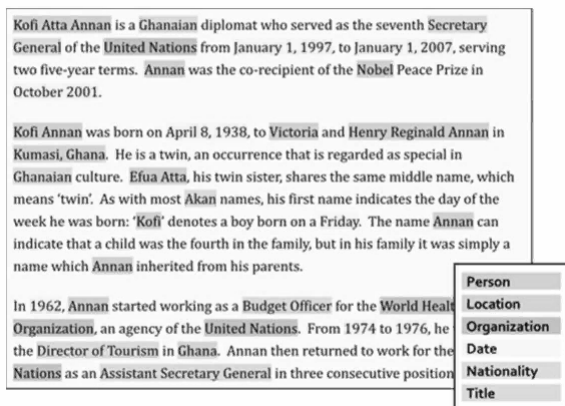


Рис. 1: Пример работы NER.

Данная статья посвящена задаче выделения именованных сущностей для русскоязычных 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 размечена 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.

Одним из наиболее очевидных признаков для решения задачи классификации именованных сущностей является вычленение информации при помощи регулярных выражений. Например, информация о наличии заглавной буквы или знаков препинания 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.

Так же 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. Одним 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 $[\mathbf{w} \mathbf{2} \mathbf{v}]$. 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. Два наиболее часто используемых улучшения — *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 качества 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.

Предсказание тегов при помощи CRF

Наиболее популярный инструмент для решения задачи *NER* является метод *CRF* (*conditional random fields*). Алгоритм *CRF* [HabrCRF] оптимизирует всю цепочку меток целиком, а не каждый элемент по отдельности. 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. Метод *CRF* может учитывать любые особенности и взаимозависимости в исходных данных. Так же он хорошо работает в связке с рекуррентными нейросетями, мо-

делирует совместное распределение на всей последовательности выходов сети одновременно.

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

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

На практике время обучения *CRF* 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 *Precision* (P) и *Recall* (R) дают исчерпывающую оценку качества, но, как правило, приходится балансировать между двумя этими метриками. Если повысить *Recall*, делая решение более «оптимистичным», это приводит к падению *Precision* из-за увеличения числа ложно-положительных ответов. Если же наоборот, то при росте *Precision* происходит одновременное падение *Recall* из-за отбраковки какого-то числа правильных ответов. Поэтому удобно использовать одну величину, так называемую метрику *F1* — среднегармоническое между *Recall* и *Precision* [**F1**]:

$$F1 = 2 \frac{PR}{P + R} \quad (1)$$

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 каче-

ства 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*,
- *lsgd* — стохастический градиентный спуск с регуляризации *L2*,
- *pa* — усредненный персептрон,
- *ap* — passive aggressive,
- *arow* — адаптивная регуляризация.

Алгоритм оптимизации	Долгое время работы	<i>F1</i> на лучшей сущности (<i>best</i>)	<i>F1</i> на худшей сущности (<i>worst</i>)	Примечания
<i>lbfgs</i>	+	1	0.7	Время обучения более 5 часов.
<i>lsgd</i>	+	1	0	Классифицирует все, как самый. объемный класс
<i>pa</i>	—	1	0.45	Время обучения не более 30 минут.
<i>ap</i>	—	0.83	0.24	—
<i>arow</i>	—	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

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

Таблица 2: Сравнительный анализ $F1$.

высокий средний результат (таблица 2). Однако, время обучения такой модели более пяти часов. При необходимости уменьшить время обучения наиболее оптимальным решением будет обучение модели на признаках $(r3, v1)$ с оптимизатором *pa*.

Заключение

Подведем итоги, в статье приведен 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 предсказания 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*.