

УДК 214.16

**Задача извлечения 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, и был проведен сравнительный анализ моделей при использовании различных признаков и методов оптимизации. Лучшая модель показала качество F1-меры 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 (сущности в тексте). Например, есть новостной текст, в котором необходимо выделить некоторый заранее зафиксированный набор сущностей (персоны,

локации, организации, даты и т.д.). Таким образом необходимо определить, что участок текста «1 января 1997 года» является датой, «Кофи Аннан» – персоной, а «ООН» – организацией (1).

## News NER example

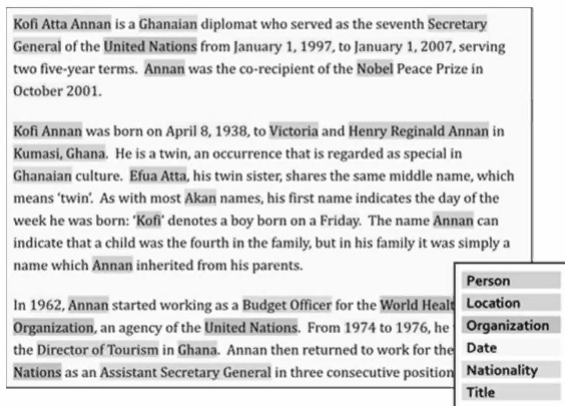


Рис. 1: Результаты работы NER.

При разработке системы распознавания сущностей в качестве тестовых данных была использована выборка из 500 файлов открытой русскоязычной базы судебной статистики [CourtsData]. Такой выбор был сделан в силу большого количества доступных документов, а также соответствующего задаче характера текстов (наличие большого количества имён, дат, наименований, сумм и т.д.)

### Предварительная обработка данных

Первым этапом является предобработка данных: разбиение на слова, удаление ненужных символов, извлечение признаков слов. Токенизация — процесс разбиения текстового документа на отдельные слова, которые называются токенами. Для начала, весь текст необходимо токенизировать [Ner], при этом удаляются не несущие смысл символы и пробелы между цифрами. После весь текст приводится к нижнему регистру, а наличие заглавной буквы заносится в «словарь символов» (см. ниже).

При обработке данных необходимо выделить символы в начале и конце токена. Например почта "*v1@mail.com*," будет разбита на "*v1@*" и "*mail.com*," где @ идентифицируется как слово-спецсимвол, а запятая добавляется 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.

По аналогии с ранее упомянутыми признаками "само слово" ( $v$ ) тоже может использоваться в качестве признака. Данный признак эффективен в паре со «словарем символов» при разнообразных данных во время обучения. Стоит отметить, что он плохо подходит, если в обучающей выборке часто встречается одна и та же фамилия или одно и тоже название организации, поскольку классификатор «затачивается» на определенном слове. Например, когда фамилия судьи одна и та же.

Большинство слов в тексте используется в определенном падеже или склонении, это усложняет работу классификатора. Одним из способов решения данной проблемы является нормализация - приведение слов к «исходному» виду (лосями → лось, мыла → мыть, зеленого → зеленый). После нормализации число и род можно отнести к отдельным признакам ( $n$ ). Также возможно использовать часть речи ( $m$ ) (существительное, предлог и т.д.), как признак для распознавания сущностей. В случае спецсимволов, после токенизации необходимо задавать каждому такому символу уникальные значения частей речи. Например, для символа % морфологией будет *PERCENT*.

Также слова можно представлять в векторном пространстве. Процесс конвертации текста в векторы называется векторизацией [ $\mathbf{w2v}$ ]. Теперь после предобработки текста, необходимо представить его в числовом виде, то есть закодировать текстовые данные числами, которые в дальнейшем могут использоваться в обучении. Технология *Word2Vec*( $w$ ) работает с большими текстовыми данными и по определенным правилам присваивает каждому слову уникальный набор чисел — семантический вектор. В последнее вре-

мя все более популярным становится подход к векторизации текста, при котором *Word2Vec* дополняется различными улучшениями. Два наиболее часто используемых улучшения — это *GloVe* и *fastText(f)*. *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.

## Предсказание тегов при помощи CRF

Наиболее популярный способ для классификации именованных сущностей — *CRF* (*conditional random fields*) [**HabrCRF**]. *CRF* оптимизирует всю цепочку меток целиком, а не каждый элемент в этой цепочке. 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.

Сложность процесса обучения *CRF* большая, а именно  $O(mNTQ2nS)$  где:

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

*CRF* может учитывать любые особенности и взаимозависимости в исходных данных. Один из лучших методов для *NER*, очень долго обучается, хорошо работает в связке с рекуррентными нейросетями, моделирует совместное распределение на всей последовательности выходов сети одновременно.

На практике вычислительная сложность обучения (время обучения) *CRF* даже выше за счет всевозможных дополнительных операций таких, как сглаживание, преобразование данных из формата в формат и т.д. Отметим, что при увеличении количества признаков (например, учитывание признаков соседних слов) время обучения значительно увеличится.

## Метрика F1

После того как *CRF* обучен необходимо оценить качество его работы. Например, метрики *Precision(P)* и *Recall(R)* дают исчерпывающую характеристику классификатора. Но, как правило при построении классификаторов приходится все время балансировать между двумя этими метриками. Если повысить *Recall*, делая классификатор более «оптимистичным», это приводит к падению *Precision* из-за увеличения числа ложно-положительных ответов. Если же наоборот классификатор делать более «пессимистичным», то при росте *Precision* это вызовет одновременное падение *Recall* из-за отбраковки какого-то числа правильных ответов. Поэтому удобно для характеристики классификатора использовать одну величину, так называемую метрику *F1* (среднегармонической между *Recall* и *Precision*):

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

Для оценки результатов была выбрана использовать меру *F1*, но *precision* и *recall* будем считать, объединяя соседние слова с одним тегом в одну тегированную сущность, поскольку одна сущность может состоять из нескольких токенов, то стоит при оценивании качества учитывать ее целиком.

## Результаты

Рассмотрим результаты тестирования 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 фиксированном наборе признаков.

*CRF* может учитывать любые особенности и взаимозависимости в исходных данных.

- *lbfgs* - градиентный спуск с использованием метода L-BFGS,
- *l2sgd* - стохастического градиентного спуска с регуляризации L2,
- *pa* - усредненный персептрон,
- *ap* - Passive Aggressive ,
- *arow* - адаптивная регуляризация.

Алгоритм оптимизации	Долгое время работы	Лучший результат	$F1$ на худшей сущности	Примечания
<i>lbfgs</i>	+	1	0.7	время обучения более 5 часов
<i>l2sgd</i>	+	1	0	классифицирует все, как самый объемный класс
<i>pa</i>	-	1	0.45	время работы 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*.

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

## Заключение

Подведем итоги. Нами была решена задача морфологической классификации. В данной статье проведен сравнительный анализ алгоритмов оптимизации, различных наборов признаков. 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 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 F-меры.

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*