Томита-парсер

SENNA

Томита-парсер

Технологии Яндекса
https://tech.yandex.ru/tomita/ Система, предназначенная для извлечения структурированных сущностей из текста: фактов, отношений, причинно-следственных связей и др.

- Использует механизм контекстно-свободных грамматик и словари ключевых слов.
- Позволяет добавлять свои расширения.

исходный код:
https://github.com/yandex/tomita-parser/

Основные понятия

- Газеттир — словарь ключевых слов
  (пример статьи: «все города России»)
- Грамматика — множество правил (шаблонов) на языке KC-грамматик
- Факты — таблицы с колонками (полями)

Алгоритм работы парсера

1. Найти вхождения всех ключей из газеттира
2. Найти ключи, которые есть в грамматике (kwtype)
3. Покрыть предложение непересекающимися ключами
4. Отобразить терминалы грамматики на входные слова
5. Интерпретировать на построенном синтаксическом дереве

Пример:

$$S- > Noun$$

```
t = 'механизм контекстно-свободных грамматик'
...
['механизм', 'грамматика']
```

Простейшие правила:

```
S -> Adj Noun
S -> Adj Word<h-reg1>
S -> Adj<gnc-agr[1]> Noun<gnc-agr[1]>
S -> A B C | A B* C+
S -> Adj "слово";
```

Список всех помет:
https://tech.yandex.ru/tomita/doc/dg/concept/all-labels-list-docpage/

Начало работы

- Откуда скачать: https://tech.yandex.ru/tomita/
- Как запустить: https://tech.yandex.ru/tomita/doc/dg/concept/run-parser-docpage/
- Как запустить на Mac iOS:

```
chmod a+x tomita-mac
./tomita-mac config.proto
```

Файлы проекта

config.proto — конфигурационный файл парсера. Сообщает парсеру, где искать все остальные файлы, как их интерпретировать и что делать.

---

dic.gzt — корневой словарь. Содержит перечень всех используемых в проекте словарей и грамматик.

---

mygram.cxx — грамматика

---

facttypes.proto — описание типов фактов

---

kwtypes.proto — описания типов ключевых слов

Файл mydic.gzt – корневой словарь:

```
encoding "utf8";
import "base.proto";              // описания protobuf-типов (TAuxDicA
rticle и прочих)
import "articles_base.proto";     // Файлы base.proto и articles_base.
proto встроены в компилятор.
                                  // Их необходимо включать в начало л
юбого gzt-словаря.
// статья с нашей грамматикой:
TAuxDicArticle "наша_первая_грамматика"
{
    key = { "tomita:first.cxx" type=CUSTOM }
}
```

Описание простейшей грамматики:

```
#encoding "utf-8"
#GRAMMAR_ROOT S
S -> Noun;
```

Сохраняется в специальный файл (first.cxx)

Файл config.proto:

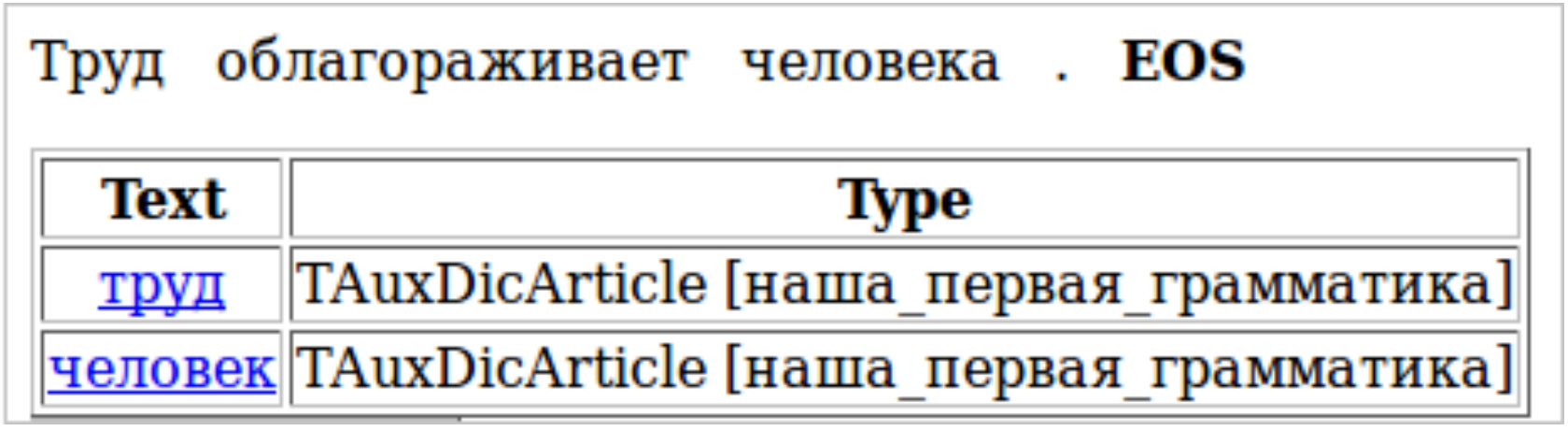
```
encoding "utf8";
TTextMinerConfig {
  Dictionary = "mydic.gzt"; // путь к корневому словарю
  PrettyOutput = "PrettyOutput.html"; // путь к файлу с отладочным
 выводом в удобном для чтения виде
  Input = {
    File = "test.txt"; // путь к входному файлу
  }
  Articles = [
    { Name = "наша_первая_грамматика" } // название статьи в корнев
ом словаре,
                                        // которая содержит запус
каемую грамматику
  ]
}
```

Осталось сделать файл test.txt и проверить:

запуск:

```
./tomitaparser config.proto
```

Вывод печатается в файл PrettyOutput.html.



Труд облагораживает человека . **EOS**

Text	Type
труд	TAuxDicArticle [наша_первая_грамматика]
человек	TAuxDicArticle [наша_первая_грамматика]

Задание

1) Выделить содержание витаминов в продуктах из текста:
   http://chem21.info/info/1069461/
   «Витамин А содержится в моркови»
   «Жиры рыб богаты витамином О»

   База – tutorial1

2) Переложить вывод в таблицу фактов.

   База – tutorial4

SENNA

SENNA (Semantic Extraction using a Neural Network Architecture) – система (а также архитектура нейронной сети), применяющаяся для различных задач анализа текста, в частности, SRL (Semantic Role Labeling).

- Базируется на принципах сверточных нейронных сетей.
- Разработана и представлена в 2007-2008, однако до сих пор является одной из самых удобных и совершенных систем.

SENNA

Из документации:

- SENNA is a software distributed under a non-commercial license, which outputs a host of Natural Language Processing (NLP) predictions: part-of-speech (POS) tags, chunking (CHK), name entity recognition (NER), semantic role labeling (SRL) and syntactic parsing (PSG).
- SENNA is fast because it uses a simple architecture, self-contained because it does not rely on the output of existing NLP system, and accurate because it offers state-of-the-art or near state-of-the-art performance.
- SENNA is written in ANSI C, with about 3500 lines of code. It requires about 200MB of RAM and should run on any IEEE floating point computer.

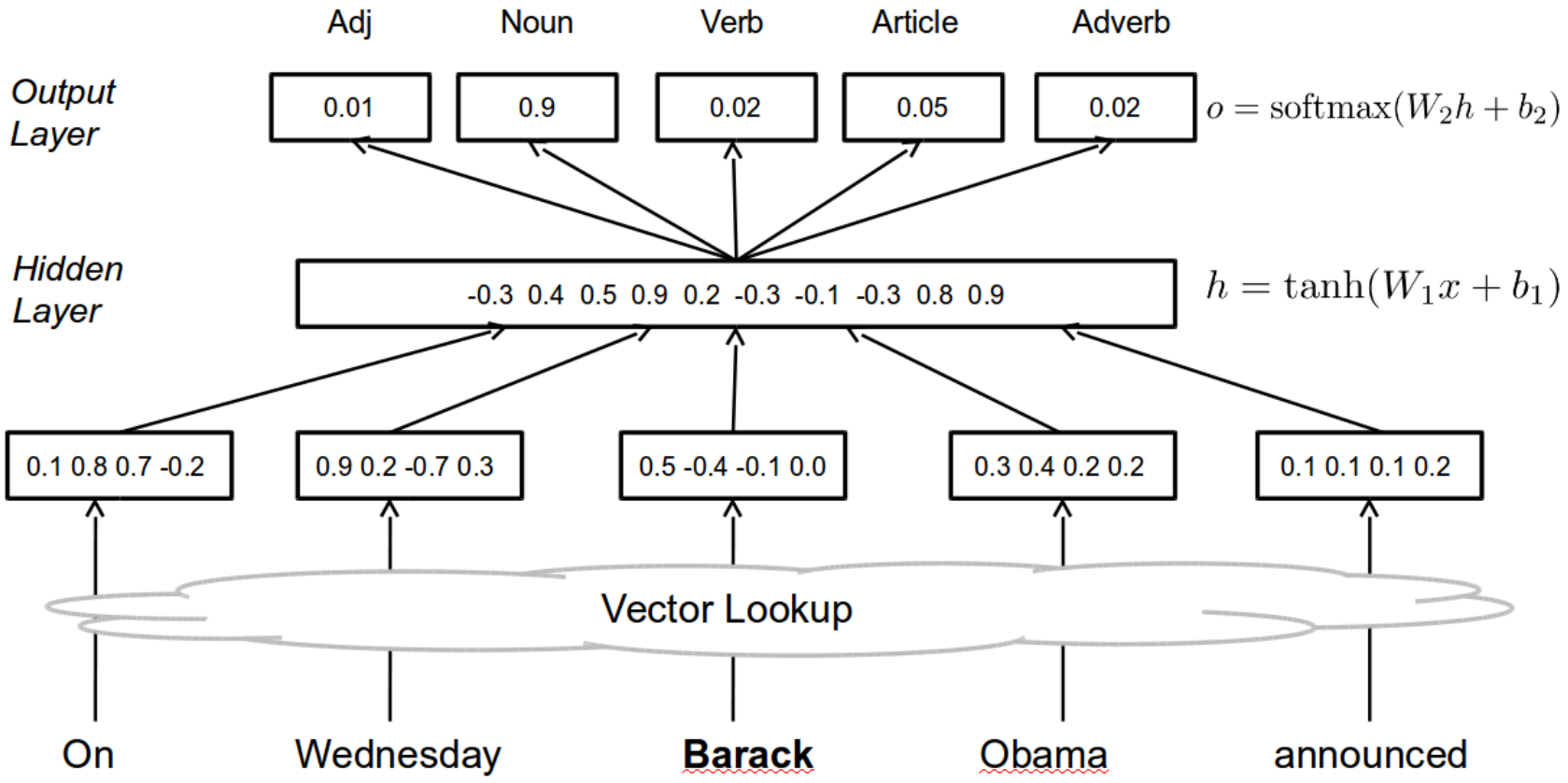
https://ronan.collobert.com/senna/

SENNA

Система применяется для различных задач классификации текстов на уровне слов и предложений (POS-tagging, NER, Chunking). Два способа применения:

- Window-Approach: необходимая информация содержится в контексте слов (NER, POS)
- Sentence-Approach: важно рассматривать предложения целиком (разбор предложений)

SENNA - схема


Adj
Noun
Verb
Article
Adverb
Output Layer
0.01
0.9
0.02
0.05
0.02
$o = \text{softmax}(W_2 h + b_2)$
Hidden Layer
-0.3 0.4 0.5 0.9 0.2 -0.3 -0.1 -0.3 0.8 0.9
$h = \tanh(W_1 x + b_1)$
0.1 0.8 0.7 -0.2
0.9 0.2 -0.7 0.3
0.5 -0.4 -0.1 0.0
0.3 0.4 0.2 0.2
0.1 0.1 0.1 0.2
Vector Lookup
On
Wednesday
Barack
Obama
announced

SENNA - Vector Lookup

Этап получения векторного представления слов (word embedding). На этом этапе могут быть следующие действия:

- Преобразование из формата BoW.
- Преобразования из различных word embedding к формату, требуемому входом сети.

SENNA - слои сети

В сети присутствуют как минимум два слоя:

- Слой для поиска локальных фич среди соседних слов, использует сконкатенированные вектора слов
- Слой для поиска глобальных оптимумов на уровне предложения – дает ответ к основной задаче

Подробное описание архитектуры:
https://ronan.collobert.com/pub/matos/2009_tutorial_nips.pdf,
http://ml.nec-labs.com/software.php?project=senna.

SENNA - пример

1. Select a target word, e.g. **Barack**
2. Create a window of *n* tokens around the target. A window-size=1 would create the tuple (Wednesday, **Barack**, Obama)
   - Use special PADDING token for tokens at sentence border, e.g. (PADDING, **Wednesday**, Barack)
3. Map each token to its word embedding (e.g. 100 dim. word embedding)
4. Concatenate the 3x100 dim. vectors and feed the 300 dim. vector into a dense hidden layer and apply tanh-activation function
5. Take the output of the hidden layer, feed it into a dense layer and apply the softmax-activiation function