Jspell.pm – um módulo de análise morfológica para uso em Processamento de Linguagem Natural

Alberto Manuel Simões
José João Almeida
Departamento de Informática
Universidade do Minho

Neste documento é nosso propósito apresentar as características presentes no analisador morfológico `jspell` e quais as suas consequências ao nível de aplicações de processamento de linguagem natural.

Como ferramenta que é frequentemente integrada em *software* mais específico, apresentamos um módulo Perl desenvolvido com o objectivo de facilitar a interligação do analisador morfológico com pequenas aplicações desenvolvidas em linguagens de *scripting*.

Devido à constante necessidade de melhoramento de dicionários, e em particular dos analisadores morfológicos, discutimos as propriedades que estes devem conter para facilitar o seu processamento e enriquecimento automático.

Introdução

Um analisador morfológico é uma ferramenta capaz de, dada uma palavra, indicar as características morfológicas (o género, número, categoria gramatical, ...) de todas as suas análises.

Tradicionalmente, os analisadores morfológicos contêm um dicionário de base (palavras e suas características morfológicas) e um conjunto de regras de formação de novas palavras. Esta ideia está na base do funcionamento do analisador morfológico `jspell`[1].

Breve história do jspell

O `jspell` foi desenvolvido com base no corrector ortográfico *open-source* `ispell`[2]. Embora não seja um analisador morfológico, o `ispell` incluía já possibilidade de definição e uso de regras morfológicas elementares. Em vez de usar um dicionário em extensão, define-se um conjunto de regras morfológicas (tabela de afixos que permite, a partir de uma palavra, obter várias flexões e derivações) e associa-se a cada palavra o conjunto de regras morfológicas que lhe são aplicáveis. Esta associação permite que um dicionário seja substancialmente mais pequeno do

Actas do XVII Encontro Nacional da Associação Portuguesa de Linguística, Lisboa, APL, 2002, pp. 485-495

que a listagem de todas as palavras dele obtidas e permite também uma maior riqueza de tratamento de palavras desconhecidas.

Estrutura do dicionário

O dicionário do `jspell` é constituído por entradas. Cada entrada corresponde a um lema[1], uma descrição morfológica e um conjunto de regras de flexão e derivação.

Consideremos o seguinte exemplo:

```
considerar/CAT=v,T=inf,TR=_/XYPLcv/
o/CAT=art,G=m,N=s/p/
seguinte/CAT=a_nc,G=2,N=s/pm/ comentário...
exemplo/CAT=nc,G=m,N=s/p/
```

Como se pode ver, cada uma das linhas é composta por quatro partes, separadas por uma barra de divisão (carácter '/'). O primeiro campo é o nosso lema, o segundo é uma lista de propriedades morfológicas, segue-se um conjunto de identificadores de regras de derivação ou flexão. O último campo permite a inclusão de comentários.

Analisemos a palavra *exemplo*:

- CAT=nc – a categoria é *nome comum*;
- G=m – o género é *masculino*;
- N=s – o número é *singular*;
- p – indica que a palavra pode formar plural (*exemplos*) usando a definição da regra 'p' (que mostramos abaixo).

Muitos outros tipos de regras morfológicas podem ser adicionadas como sejam identificação de formação do feminino, conjugação de verbos regulares, conjugação pronominal, formação em *mente* ou *idade*.

Para uma descrição mais completa do formato, e para ver uma descrição do tratamento de excepções, dos mecanismos de definição de abreviaturas ou de outros pormenores, consultar o manual que acompanha o código fonte.

Estrutura das regras de afixos

Os identificadores das regras de derivação ou flexão estão definidos num ficheiro separado, chamado de *regras de afixos*. Estas regras explicam como se forma novas palavras a partir do lema.

[1] O objectivo deste artigo não é definir o conceito de lema., que no entanto, vai ser usado frequentemente ao longo deste documento. Neste contexto, consideramos lema uma palavra de onde se pode obter outras palavras, por flexão ou derivação, e que não pode ser obtida de qualquer outro lema.

Por exemplo, a formação de plurais segundo a regra "p" é descrita do seguinte modo:

```
flag *p:
  [^Ã][^LSMRNZX]>       S     ; "N=p"  # -> s          ex. pato, patos
  Ã E           >       S     ; "N=p"  # ãe -> ães     ex. mãe, mães
  Ã O           >    -ÃO,ÕES  ; "N=p"  # ão -> ões     ex. leão, leões
  A L           >     -L,IS   ; "N=p"  # al -> ais     ex. animal, animais
  O L           >    -OL,ÓIS  ; "N=p"  # ol -> óis     ex. anzol, anzóis
  [^V] E L      >    -EL,ÉIS  ; "N=p"  # el -> éis     ex. papel, papéis
   ...
```

Cada uma das regras permite especificar a formação de um plural para palavras com terminações diferentes. Da mesma forma que especifica o modo como a palavra é alterada, também permite indicar que propriedades são mudadas. Neste caso, o número passa a plural (N=p).

Uma única regra pode descrever a formação de muitas palavras. Por exemplo, as regras de flexão dos verbos associam características como sejam a pessoa, o número e o tempo respectivo:

```
[^\-] A R   >  -AR,O      ; "P=1,N=s,T=p"
A R         >  -R,S       ; "P=2,N=s,T=p"
[^\-] A R   >  -AR,A      ; "P=3,N=s,T=p"
A R         >  -R,MOS     ; "P=1,N=p,T=p"
A R         >  -R,IS      ; "P=2,N=p,T=p"
A R         >  -R,M       ; "P=3,N=p,T=p"
 ...
```

O exemplo anterior permite a conjugação de verbos terminados em *ar* para o presente do indicativo. Note-se que neste caso quer a pessoa, número e tempo verbal são alterados.

Como caso particular, a categoria gramatical pode ser também alterada, como se pode verificar neste exemplo de transformação de adjectivos em advérbios:

```
flag +m:          ; "CAT=adj,N=s"       #mente
O            -O,AMENTE    ; "CAT=adv"   # abertamente
U               AMENTE    ; "CAT=adv"   # cruamente
[^UO]            MENTE    ; "CAT=adv"   # anteriormente
 ...
```

Funcionalidade do jspell

O `jspell` permite várias formas de utilização, desde a herdada do `ispell`, como corrector ortográfico, até como biblioteca da linguagem de programação **C**, **Prolog** ou até **Perl**, como veremos nas próximas secções.

Em particular, e em funcionamento interactivo com o utilizador, podemos analisar os seguintes exemplos:

```
$  jspell -d português -a

@(#) International Jspell Version 1.00b1, 11/07/2001

coreana
* coreana 0 :lex(coreano, [CAT=adj,N=s,G=m], [], [G=f], [])
```

Neste primeiro exemplo, após a primeira linha que nos indica que versão do `jspell` que estamos a executar, temos duas outras linhas, uma em que indicamos a palavra que queremos analisar (*coreana*) e a seguinte onde o `jspell` nos dá informação sobre a palavra.

Neste caso, aparece-nos um asterisco que indica que a palavra existe no dicionário, gerada a partir do lema *coreano* (que é um adjectivo, masculino singular) mudando o género para feminino.

```
cordiais
 * cordiais 0 :lex(cordial, [CAT=adj,N=s,G=_], [], [N=p], [])
```

Este outro exemplo, indica-nos que *cordiais* pode ser obtida do adjectivo singular *cordial* (sem género) alterando o número para plural.

```
cordeais
* cordeais 0 :lex(cordear, [CAT=v,T=inf,TR=t], [], [P=2,N=p,T=p], [])
```

Mais interessante é pedir informação acerca de um tempo verbal. Neste caso, *cordeais* é uma forma do verbo *cordear*, na segunda pessoa, plural no presente do indicativo.

```
crodeais
& crodeais 0 :
   CORDEAIS= lex(cordear, [CAT=v,T=inf,TR=t], [], [P=2,N=p,T=p], []),
    RODEAIS= lex(rodear,  [CAT=v,T=inf,TR=_], [], [P=2,N=p,T=p], [])
```

Finalmente, este exemplo permite-nos ver o que acontece quando pedimos ao `jspell` dados sobre uma palavra que não existe. Neste caso, é colocado um '&' antes da palavra desconhecida. São procuradas palavras aproximadas à que procuramos e é dada a informação correspondente a essas palavras. Este comportamento pode ser alterado usando determinadas opções permitindo não calcular palavras aproximadas, ou procurá-las de um modo mais restrito.

jspell.pm: Interface com Perl

O Perl[4] é uma linguagem pertencente a uma categoria a que designamos geralmente de *scripting*. Apresenta grande versatilidade quando usado em processamento de texto e no processamento de linguagem natural.

Por esta razão, foi desenvolvido um módulo Perl para interface entre o analisador morfológico e programas desenvolvidos nesta linguagem.

Funcionalidade oferecida

Aproveitando as potencialidades desta linguagem, aumentou-se a funcionalidade do próprio analisador morfológico com diversas funções:

- `jspell_dict(dict,[personaldict])`
  permite a inicialização do jspell com o dicionário que nos interessa;
- `der(palavra)`
  a partir de uma palavra (ou lema), indica-nos todas as palavras dela derivadas;
- `flags(palavra)`
  para cada lema que permita obter a palavra, indica que regras são necessárias aplicar para a obter;
- `nlgrep(palavra,ficheiro*)`
  dado um ou mais ficheiros, esta função indica-nos todas as linhas onde aparece qualquer palavra derivada da indicada; pode receber opções para seleccionar algumas variantes no comportamento (p.ex, indicar o número máximo de respostas);
- `setstopwords(palavra*)`
  permite definir um conjunto de palavras a ignorar em certas funções;
- `rad(palavra)`
  útil para encontrar uma lista de possíveis lemas para uma palavra;
- `fea(palavra)`
  devolve uma lista das análises da palavra, em que cada análise descreve a informação morfológica e lema associado.
- `mkradtxt(origem, destino)`
  embora pouco útil para utilização directa, esta função permite-nos converter um ficheiro noutro em que aparecem as palavras originais seguidas dos lemas respectivos.

Além de todas estas funções, o módulo permite especificar um conjunto de variáveis que alteram o comportamento do `jspell`, nomeadamente na forma como as palavras desconhecidas são tratadas (utilizando a função `setmode(opções)`).

Exemplos

Aqui vamos apresentar alguns pequenos exemplos de como este módulo pode ser útil no processamento da língua portuguesa.

Começamos por um *grep*, ou pesquisador, que dada uma palavra e um conjunto de ficheiros imprime todas as linhas em que a palavra aparece, mesmo que derivadas:

```
#!/usr/bin/perl
use jspell;
jspell_dict("port");
print nlgrep(@_);
```

Este pequeno programa (chamado `nlgrep`) pode vir a ser utilizado da seguinte forma:

```
nlgrep arma lusiadas.txt aventura.txt
```

com resultados do género:

As armas e os barões assinalados
...
O ladrão estava armado até aos dentes!

De seguida apresenta-se um exemplo de tentativa de contagem do número de ocorrências de lemas para a publicação em dicionários, como se pode ler em [6] em que a principal função é a `rad`:

```
use jspell;
jspell_dict("port")

foreach (numoco, pal) in stdin
    w = rad(pal);
    duvida = length(w) > 1;

    foreach p in (w)
        if (duvida) { oco[p][duv] += numoco; }
        else        { oco[p][gar] += numoco; }

foreach p in (sort dom(oco))
    total = oco[p][duv] + oco[p][gar];
    conf = oco[p][gar]/total;
    print p, total, conf;
```

Na secção seguinte apresentam-se funções e exemplos referentes ao manuseamento e gestão de dicionários `jspell`.

jspell::dict: gestão do dicionário

Um dicionário `jspell` sendo normalmente extenso (ex, português tem 44 mil lemas) tem associado uma gestão e edição trabalhosa e complexa. Esta tarefa inclui: adicionar mais palavras, detectar/eliminar algumas redundâncias, adicionar/retirar regras morfológicas associadas a palavras.

Estas tarefas podem ser assistidas por funções que usando o módulo `jspell.pm` permitam automatizar certas actividades. O módulo `jspell::dict` que se apresenta nesta secção, inclui funções com esse objectivo.

Funções disponíveis

No módulo `jspell:dict` encontram-se (entre outras) as seguintes funções criadas para o desenvolvimento de aplicações ligadas à gestão de dicionários do `jspell`.

- `init(dic)`
  Esta função permite construir um novo objecto que represente o dicionário cujo nome do ficheiro foi passado como parâmetro.
- `foreach_word(func)`
  Este método processa todas as palavras do dicionário usando determinada função. Esta função vai ser chamada para todas as palavras do dicionário com o lema, uma tabela com a informação morfológica e uma lista de regras de derivação.
- `for_this_cat_I_want_only_these_flags(cat, regra*)`
  Permite especificar para determinada categoria gramatical, quais as regras de derivação e flexão estão disponíveis. Qualquer entrada que contenha uma regra diferente fará com que seja indicado um aviso para rever a dita entrada.
- `for_this_cat_I_dont_want_these_flags(cat, regra*)`
  Funciona de forma idêntica ao método anterior mas irá avisar se alguma entrada usar uma das regras indicadas.
- `not_categorized`
  Extrai um relatório de todas as entradas sem uma definição de categoria.
- `extra_words`
  Esta função tenta encontrar entradas redundantes no dicionário.
- `add_words(palavra*)`
  Permite ermite adicionar palavras ao dicionário. Por exemplo:

```
$dict->add_words(
 { word=>'papagaio', flags =>'pf', CAT => 'nc', G=>'m'},
 { word=>'macaco', flags =>'pf', CAT=>'nc', G=>'m'} )
```

- `delete_word(palavra)`
  Apaga a entrada correspondente.

```
$dict->delete_word('camolo')
```

- `add_flag/insere_flag`
  Adicionam regras morfologicas às palavras indicadas.

```
$dict->insere_flag('p', 'linha', 'carro')
```

Detecção de redundância

Existe redundância no dicionário quando por exemplo, existem duas entradas no dicionário que permitem derivar a mesma palavra e categoria. Desta forma, se um dicionário contém a palavra *frequente* com a flag "m" (responsável pela geração

do advérbio usando o sufixo *mente*) e a palavra *frequentemente*, podemos dizer que a segunda palavra e redundante. Por outro lado se tivermos uma entrada para o verbo *gostar* que gera a palavra *gosto* (forma verbal) e existir uma outra entrada *gosto* (substantivo), não existe redundância.

Para podermos encontrar entradas redundantes no dicionário teremos de processar cada entrada, uma a uma, gerando todas as suas possíveis derivações. Para cada uma desta, se existir como entrada no dicionário, com a mesma categoria, então estamos perante uma eventual redundância. Vejamos este processo num algoritmo:

```
use jspell_dict("port");

foreach (palavra) in (dicionário)
   derivadas = der(palavra)
   foreach (derivada) in (derivadas)
foreach (entrada) in (dicionário)
   if (entrada == palavra) next entrada
   if ((entrada == derivada) and
        sameCat(entrada, derivada)) then
      print "# from 'palavra' you can get 'entrada'"
      print "delete_word('entrada')"
```

Um extracto do resultado da aplicação deste algoritmo ao dicionário de português do `jspell`:

```
# from tabulador you can get tabuladora
delete_word('tabuladora')
# from sereno you can get sereníssimo
delete_word('sereníssimo')
# from gaveto you can get gaveta
delete_word('gaveta')
# from caneco you can get caneca
delete_word('caneca')
# from meteoro you can get meteorismo
delete_word('meteorismo')
```

Visto que esta operação é automática, nem todos os casos correspondem a palavras a remover por não corresponderem ao mesmo conceito. Noutros casos, como a relação entre *gaveto* e *gaveta*, seria útil eliminar as regras que relacionam *gaveto* com *gaveta* (retirar a flag "gerar feminino" do conjunto de regras aplicáveis à palavra *gaveto*).

Enriquecimento por corpora

Uma possibilidade de enriquecimento do dicionário é a extracção da lista de palavras contidas num corpus e analisar quais delas não são reconhecidas pelo analisador morfológico. O resultado desta operação poderia ser inserido directamente no dicionário, o que gerava alguns problemas, visto que muitas dessas palavras poderiam ser versões de palavras existentes, mas com erros ortográficos, ou com palavras demasiado técnicas.

Por outro lado, algumas das palavras podem ser derivadas de uma palavra *p* já existente no dicionário usando uma regra morfológica que não está contida no conjunto das regras permitidas para a palavra *p*. Desta forma, apresenta-se um pequeno exemplo que para cada palavra desconhecida verifica se esta é o plural de uma nome comum conhecido:

```
use jspell;
jspell_dict("port");
setmode("+flags");

file = shift;

foreach (word) in pipeopen("jspell -l -d port < file|") {
   fl = fea(word);
   foreach (feature) in (fl) {
      if (verify({flags=>"p", CAT=>"nc"}, feature)) {
         print "insere_flag('p',feature[rad]); #... word\n";
      }
   }
}
```

Depois de correr este pequeno programa sobre a lista de palavras do CETEM-Público[3], obtemos uma lista de onde se extraiu o seguinte:

```
insere_flag('p','pataca'); #....patacas
insere_flag('p','juiz'); #....juizes
insere_flag('p','abaixo-assinado'); #....abaixo-assinados
insere_flag('p','passarinho'); #....passarinhos
insere_flag('p','co-autor'); #....co-autores
insere_flag('p','franco-atirador'); #....franco-atiradores
insere_flag('p','soberania'); #....soberanias
insere_flag('p','acrobata'); #....acrobatas
insere_flag('p','cachecol'); #....cachecóis
insere_flag('p','locatário'); #....locatários
insere_flag('p','letreiro'); #....letreiros
insere_flag('p','guarda-chuva'); #....guarda-chuvas
insere_flag('p','egoísmo'); #....egoísmos
```

Embora esta operação seja ligeiramente complexa, demora bastante pouco tempo. Para processar 339 650 linhas de texto. Este programa de 12 linhas produziu 1889 linhas de *insere_flag* em 21 segundos[2].

Uma vez que este extracto é puro código Perl, pode ser gravado num ficheiro, verificado manualmente e posteriormente executado. Como se viu anteriormente, a função `insere_flag` faz parte das funções oferecidas no módulo `jspell::dict`.

Detecção de inconsistências

Grande parte do dicionário foi gerada automaticamente, e muita da sua edição foi feita, muitas vezes, à pressa. Desta forma, existem algumas regras morfológicas que foram atribuídas a determinadas palavras que, na verdade, não deviam estar.

[2] Usando um Pentium III, em carga, 600MHz, 128 Mbytes de RAM e Linux

É exemplo disto algumas entradas de verbos em que encontramos regras para formação de plural e de feminino. Este é, sem dúvida, um grande erro e não pode passar incólume.

Para facilitar este tipo de correcção, desenvolveu-se um par de funções que verificam se todas as entradas de determinada categoria gramatical só usam propriedades de certo conjunto, e de forma inversa, se para determinada categoria gramatical as entradas não usam certa propriedade.

Da mesma forma, para que isto se torne possível, é necessário que todas as entradas estejam correctamente catalogadas, nomeadamente com a categoria gramatical a que pertence. A seguinte função trata de construir um relatório com as palavras não catalogadas.

```
sub not_categorized {
      foreach_word(check_category)
}

sub check_category (lema, cats, flags) {
      if (not(defined(cats[CAT]))) {
            print "A palavra 'lema' não está categorizada!"
      }
}
```

Conclusões

Um analisador morfológico é imprescindível para qualquer aplicação de processamento de linguagem natural. No entanto, é útil especialmente como biblioteca e não como ferramenta independente.

Ao implementar o módulo `jspell.pm` conseguiu-se estender a funcionalidade do analisador morfológico e, pelo facto de ser um módulo de uma linguagem de *scripting* facilitar a interligação de aplicações de modo compacto e eficiente.

A existência de uma ferramenta para manutenção de dicionários permite a implementação de operações rotineiras e também facilita o raciocínio sobre os mesmos.

Todas estas ferramentas são *open-source* e estão disponíveis na página do projecto Natura[5], nomeadamente os manuais e artigos relacionados com o *jspell*. Todas as ferramentas estão disponíveis em CVS[7] para desenvolvimento cooperativo.

Referências

[1] José João Almeida e Ulisses Pinto, Jspell – um módulo para análise léxica genérica de linguagem natural. Em *Actas do Congresso da Associação Portuguesa de Linguística*, Évora, 1994

[2] R. Gorin, P. Willisson, W. Buehring e G. Kuenning. Ispell, a free software package for spell checking files, 1971

[3] Paulo Alexandre Rocha e Diana Santos. CETEMPúblico: um corpus de grandes dimensões de linguagem jornalística portuguesa. Em *Actas do V Encontro para o processamento computacional da língua portuguesa escrita e falada*, (PROPOR'2000), 19 a 22 de Novembro de 2000.

[4] Larry Wall, Tom Christiansen e Randal Schuartz. *Programming Perl*. O'Reilly & Associates, Inc.

[5] Projecto Natura: http://natura.di.uminho.pt

[6] Paulo Alexandre Rocha, Alberto Manuel Simões e José João Almeida. Cálculo de frequências para entradas de dicionários através do uso conjunto de analisadores morfológicas, taggers e corpora. Em *Actas do Congresso da Associação Portuguesa de Linguística*, Lisboa, 2001

[7] Moshe Bar and Karl Franz Fogel. *Open Source Development with CVS*, 2nd Edition. Coriolis Inc.