INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
(ICMC)

NÚCLEO INTERINSTITUCIONAL DE LINGUÍSTICA
COMPUTACIONAL (NILC)

Relatório do tokenizador LBTokenizer + UDPipe

Luana Balador Belisário

SÃO CARLOS
DEZEMBRO DE 2020

Sumário

1 Introdução e Motivação

O tokenizador para o português LBTokenizer que pode ser integrado ao UDPipe [1] foi desenvolvido com o objetivo de pré-processar textos do córpus criado para meu projeto de Iniciação Científica, realizando a tokenização para a posterior anotação morfossintática e sintática das sentenças de acordo com as diretrizes da Universal Dependencies (UD). A necessidade de desenvolver um tokenizador veio depois da utilização da função *tokenize* do UDPipe e da percepção de alguns problemas da funcionalidade. O UDPipe realiza a tokenização corretamente de um modo geral, porém, para alguns casos, não identifica os tokens corretamente. Na tabela a seguir, temos alguns exemplos de sentenças e sua tokenização utilizando o UDPipe 2.0 .

Sem preprocessamento	**Tokenizadas com UDPipe**
Fá-lo-ei por você!	Fá-lo-ei por você !
Procurar-me-iam caso precisassem de ajuda.	Procurar-me-iam caso precisassem de ajuda .
Li "o velho e o mar"e gostei muito.	Li
Lembro-me de quando nos reuníamos para jantar.	Lembro me de quando nos reuníamos para jantar .
Desenhar-te-ei nos meus sonhos.	Desenhar-te-ei em os meus sonhos .

Analisando a tabela é possível ver que o UDPipe não separa as mesóclises e para as ênclises ele simplesmente retira o hífen. Na sentença *Li "o velho e o mar"e gostei muito.*, as aspas foram identificadas como separador de sentenças e o restante foi desconsiderado da sentença original. Além disso, se um texto possui vários períodos, o UDPipe realiza o mesmo procedimento e tokeniza apenas o primeiro período encontrado, ou seja, até o primeiro ponto final.

Pronomes de tratamento com ponto e espaço e abreviações com ponto também não são separadas corretamente. O ponto nas abreviações e pronomes de tratamento sempre é separado e muitas vezes pode até ser reconhecido como um separador de sentenças pelo tokenizador.

Além da tokenização, também é de interesse na tarefa a anotação morfossintática e sintática das sentenças do córpus, por isso o LBTokenizer foi desenvolvido para melhorar a qualidade da tokenização. Com o LBTokenizer, é esperado que uma tokenização mais correta melhore a qualidade das etapas de anotação.

2 Diretrizes de tokenização da Universal Dependencies (UD)

As regras de tokenização e o formato de texto com que a UD trabalha podem ser encontrados em [3].

2.1 Separação de tokens, pontuação e abreviações

Na língua portuguesa, assim como em grande parte das línguas, as palavras (tokens) são separadas por espaço e pontuação. Os sinais de pontuação ponto, vírgula, ponto e vírgula, dois pontos, aspas simples e duplas, sinais de exclamação e interrogação e três pontos devem ser tokenizados normalmente, sendo separados das palavras. Contudo, algumas exceções são importantes considerar:

- Abreviações com ponto: o ponto não deve ser separado da palavra. Ex: mín., séc., déc., cód.
- Abreviações de pronomes de tratamento com ponto e espaço: devem ser tratadas como um único token e os pontos não devem ser separados das palavras. Ex: V. S.ª (Vossa Senhoria), V. Ex.as ou V. Exas. (Vossas Excelências), V. Revmª (Vossa Reverendíssima).
- Apóstrofos são e devem ser tratados como um único token, mesmo em palavras compostas. Ex: copo-d'água, d'alva, galinha-d'angola.

2.2 Tokens Multipalavra

A UD não reconhece unidades maiores que palavras, mas reconhece o que chama de "tokens multipalavra". No caso do Português, isso significa contrações e clíticos, os quais devem ser "quebrados"pelo tokenizador.

Exemplos de contrações:

- na (em+a)
- deste (de+este)
- pelo (por+o)
- noutro (em+outro)

Exemplos de clíticos:

- fazê-lo (fazer+o)
- dar-nos (dar+nos)
- acatá-la (acartar+a)
- far-se-á (fará+se)

Além disso, é importante que o tokenizador não destrua palavras compostas como guarda-chuva, água-de-cheiro, guarda-vidas, pé-de-moleque, etc.

3 Funcionamento e instalação da ferramenta

3.1 *Modus operandi* do LBTokenizer

Com base nas diretrizes da UD, o tokenizador foi implementado visando resolver todos os problemas encontrados na tokenização do UDPipe. O tokenizador foi implementado em Python e foi utilizada a função *word_tokenize* da biblioteca *NLTK Tokenize* do Python (mais informações na documentação em [4]).

O algoritmo realizado pelo LBTokenizer pode ser resumido no pseudocódigo a seguir:

1. Identificar as abreviaturas com pontos e espaços como pronomes de tratamento, palavras abreviadas, siglas, etc.
2. Tratar as abreviaturas simples com ou sem ponto e sem espaços.
3. Aplicar a função word_tokenize.
4. Tratar os clíticos:
    4.1. verbos irregulares;
    4.2. mesóclises comuns;
    4.3. ênclises comuns.

No item 4.1 do pseudocódigo, vão ser tratados os casos em que os verbos são irregulares e não obedecem à regras, como o verbo *fazer*, por exemplo. O futuro do presente do verbo fazer é *far-se-á*, e a tarefa do tokenizador é alterar a expressão conjugada para *fará se*.

Alguns dos recursos como listas de abreviações de pronomes de tratamento, contrações de pronomes (na = em + a), abreviações de palavras (mín = mínimo, séc = século, déc = década) e pronomes usados em clíticos foram obtidas do site do LX-Center Tokenizer [5 e 6].

3.2 Integração com UDPipe

A combinação LBTokenizer + UDPipe foi realizada com o objetivo de inserir como entrada no UDPipe o texto já tokenizado da forma correta, para que ele apenas faça a anotação morfossintática e sintática de acordo com as diretrizes da UD. Dessa forma, é esperada uma melhoria de desempenho, já que o LBTokenizer corrige muitos dos problemas de tokenização que a função *tokenize* do UDPipe apresenta, já comentados na seção 1.

O texto já tokenizado pelo LBTokenizer é dado como entrada ao UDPipe, que faz a anotação morfossintática e sintática de acordo com as diretrizes da UD e tem como saída o arquivo devidamente anotado no formato CONLL-U, exigido pela UD [7].

3.3 Instalação e execução

3.3.1 Preparando o ambiente

Para executar o tokenizador, é necessário que ter instalados o sistema operacional Linux e o Python3 no computador. Geralmente o Python3 já vem instalado nos sistemas operacionais Linux, mas caso não tenha, o Python3 pode ser instalado da seguinte forma:

1. Abra o terminal/shell do Linux e digite o comando: `sudo apt install python3`. Caso já tenha instalado, esse comando vai atualizar seu Python3 para a versão mais recente.
2. O pip é o instalador de pacotes do Python e para instalar é necessário executar o seguinte comando no terminal/shell: `sudo apt-get -y install python3-pip`.

Além disso, é necessário instalar a biblioteca *NLTK* [8] para que o LBTokenizer possa utilizar a função *word_tokenize*. Para instalar o *NLTK* é só executar o seguinte comando no terminal/shell do Linux: `pip3 install nltk`.

3.3.2 Download e execução

É necessário acessar o link em [9] e fazer o download do arquivo `.zip` do repositório no seu computador acessando `Code > Download ZIP`.

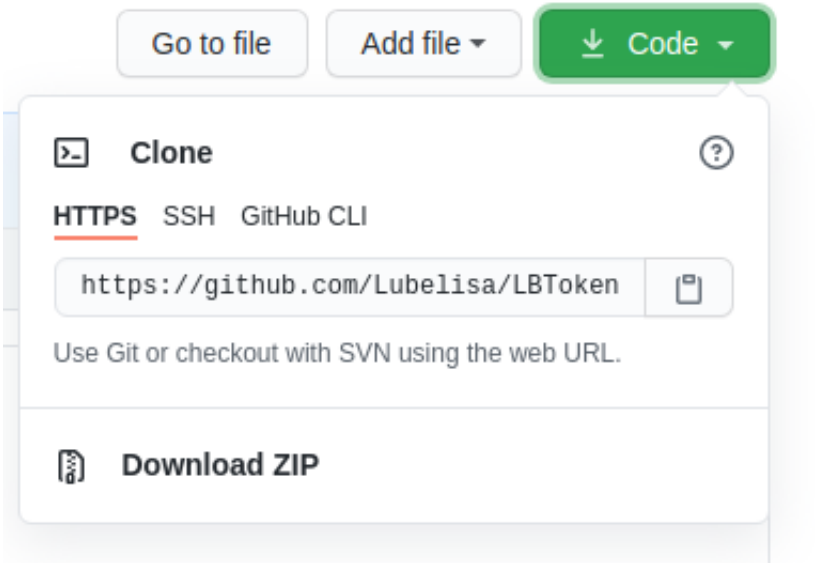

Figura 1: Para fazer o download do repositório é necessário clicar no botão `Code` e depois clicar em `Download ZIP`.

Após o download do repositório, extrair a pasta `Tokenizador` do arquivo `.zip`. Abra o terminal na pasta `Tokenizador` que foi extraída e execute o comando `./exec_tokenizador.sh`.

3.3.3 Preparando o formato de entrada

Antes de executar o tokenizador de acordo com as instruções na subseção acima, é necessário certificar-se que o texto que se deseja processar esteja em um arquivo `.txt` nomeado como `input.txt` com a codificação UTF-8 e localizado na mesma pasta que o LBTokenizer (dentro da pasta `Tokenizador`). Tanto o LBTokenizer como o UDPipe reconhecem as sentenças separadas por uma quebra de linha. Portanto, se deseja processar mais de uma sentença, coloque-as uma por linha no arquivo `input.txt`, como na figura abaixo.

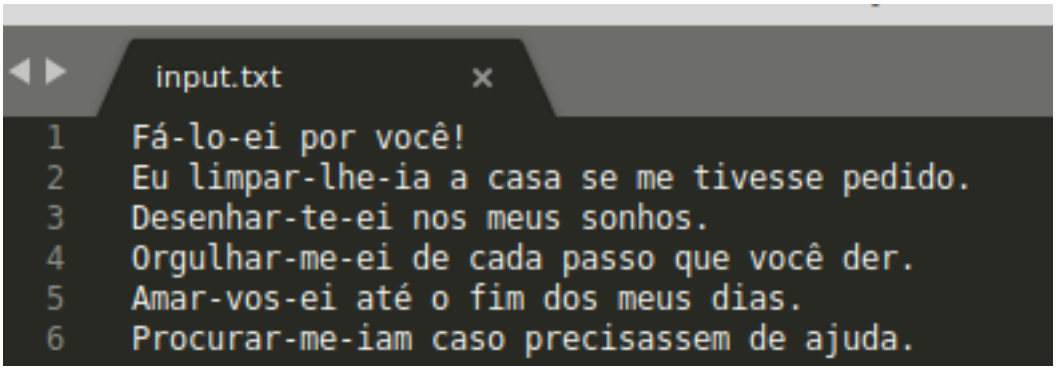

Figura 2: Exemplo de arquivo de entrada com o nome `input.txt` sendo exibido no editor de texto SublimeText.

Sabendo que o texto é tratado como uma única sentença até a quebra de linha, se o usuário deixar um texto com muitos períodos em uma única linha do aquivo `.txt`, isso prejudicará a qualidade da anotação morfossintática e sintática do texto.

As sentenças tokenizadas estarão no arquivo `input_tokenizado.txt` e as sentenças devidamente anotadas morfossintatica e sintaticamente no formato CONLL-U estarão no arquivo `input_anotado.conllu` .

4 Referências

[1] Milan Straka (2018): **UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task.** In: Proceedings of CoNLL 2018: The SIGNLL Conference on Computational Natural Language Learning, pp. 197-207, Association for Computational Linguistics, Stroudsburg, PA, USA, ISBN 978-1-948087-72-8

[2] Nivre, Joakim. (2015). **Towards a Universal Grammar for Natural Language Processing**. 3-16. 10.1007/978-3-319-18111-0_1.

[3] **Tokenization and Word Segmentation.**
Disponível em: https://universaldependencies.org/u/overview/tokenization.html.
Acesso em: 08 de dez. de 2020.

[4] **nltk.tokenize package description**
Disponível em: https://universaldependencies.org/u/overview/tokenization.html.
Acesso em: 08 de dez. de 2020.

[5] Branco, António e João Silva, 2004. **Evaluating Solutions for the Rapid Development of State-of-the-Art POS Taggers for Portuguese**. In Maria Teresa Lino, Maria Francisca Xavier, Fátima Ferreira, Rute Costa and Raquel Silva (orgs.), Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC2004), Paris, ELRA, ISBN 2-9517408-1-6, pp.507-510.

[6] **LX Tokenizer**
Disponível em: http://lxcenter.di.fc.ul.pt/tools/pt/LXTokenizerPT.html
Acesso em: 08 de dez. de 2020.

[7] **CoNLL-U Format**
Disponível em: https://universaldependencies.org/format.html
Acesso em: 08 de dez. de 2020.

[8] **Installing NLTK**
Disponível em: https://www.nltk.org/install.html
Acesso em: 08 de dez. de 2020.

[9] **Repositório do LBTokenizer no GitHub**
Disponível em: https://github.com/Lubelisa/LBTokenizer
Acesso em: 19 de jan. de 2021.