Scripting no Processamento de Linguagem Natural

Universidade do Minho

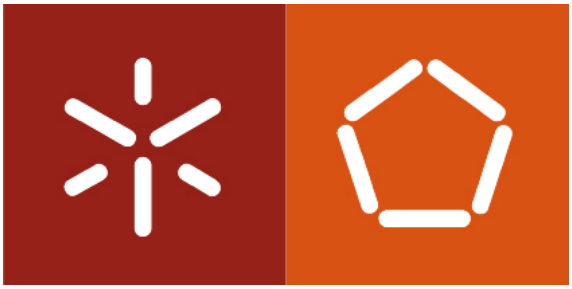
Mestrado Integrado em Engenharia Informática

Trabalho Prático 2

spaCy's POS Tagging

Grupo 7:
A73831 - João Barreira
A77364 - Mafalda Nunes

29 de Novembro de 2018



Resumo

O presente relatório tem com objetivo a aprendizagem das funcionalidades da ferramenta *spaCy*. Para tal, apresentar-se-á uma descrição da mesma, mais especificamente da funcionalidade de POS *tagging*, bem como um pequeno exemplo que demonstre como utilizar a ferramenta.

Este trabalho pretende dar resposta ao trabalho prático 2, proposto na unidade curricular SPLN, do Mestrado em Engenharia Informática, da Universidade do Minho.

Conteúdo

1 spaCy

O spaCy é uma biblioteca *open-source* para o processamento avançado de linguagem natural em *Python*, que foi desenvolvida especificamente para ser utilizada em ambientes de produção, ajudando a construir aplicações que processam grandes volumes de texto.

Dentro do contexto de processamento de linguagem natural, esta ferramenta pode ser utilizada em muitas vertentes, desde a extração de conhecimento, desenvolvimento de sistemas de compreensão de linguagem natural ou pré-processamento de texto para *deep-learning*.

1.1 Arquitetura

As estruturas de dados centrais do spaCy são o *Doc* e o *Vocab*.

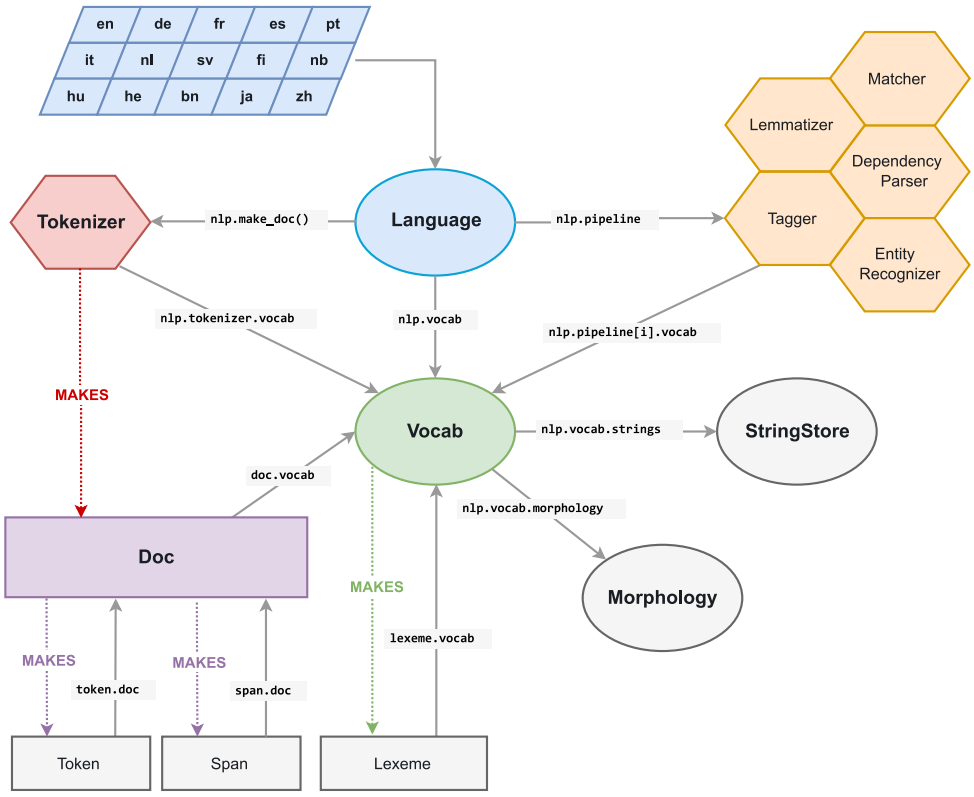


Figura 1: Esquema da arquitetura do spaCy

O objeto *Doc* possui uma sequência de *tokens* (palavra, pontuação, espaço em branco, etc.) e de *spans* (porções do objeto *Doc*), bem como todas as respetivas anotações. Os objetos *Span* e *Token* são vistas que apontam para o próprio *Doc*. O *Doc* é construído por um *Tokenizer*, sendo depois modificado pelos componentes de um *Pipeline* (figura 2). O objeto *Language* coordena estes componentes, ao pegar num texto limpo e enviá-lo para o *pipeline*, retornando um texto anotado. Para além disso, também permite organizar treinamento e serialização de dados.

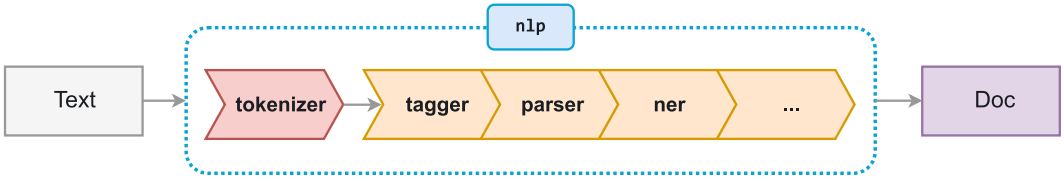


Figura 2: Esquema representativo da construção do *Doc*

Relativamente aos *pipelines*, é de se destacar os seguintes:

- `Lemmatizer` - permite determinar a forma base de palavras;
- `Tagger` - anota *part-of-speech* (POS) *tags* em objetos *Doc*;
- `Matcher` - faz *match* de sequências de *tokens*, tendo em conta regras padrão, similares a expressões regulares;
- `DependencyParser` - anota dependências sintáticas em objetos *Doc*;
- `EntityRecognizer` - anota nomes de entidades, como pessoas ou produtos, em objetos *Doc*.

O objeto *Vocab* possui um conjunto de tabelas de *look-up*, que tornam a informação comum disponível entre documentos. O *Vocab* é constituído por entradas do tipo *Lexeme*, que são um tipo de palavras sem contexto, ao contrário de uma palavra *token*, que possui uma *POS tag*, um parser de dependências, entre outros. O *Vocab* permite também aceder ao objeto *Morphology*, que possibilita a asssociação de características linguísticas, como o *lemma*, o tipo de nome, o tempo verbal, entre outros, aos *tokens*, tendo em conta a palavra e a respetiva *POS tag*. Ao centralizar vetores de palavras, atributos léxicos e *strings*, o spaCy evita armazenar multiplas cópias desses dados, pelo que poupa memória e assegura que existe uma única fonte de verdade.

Também para poupar memória, o spaCy codifica todas as *strings* para valores de *hash*, que são acessíveis a partir do *StringStore*, que corresponde a um *map* de *strings* de e para valores de *hash*. Internamente, o spaCy apenas trabalha com valores de *hash* (figura 3).

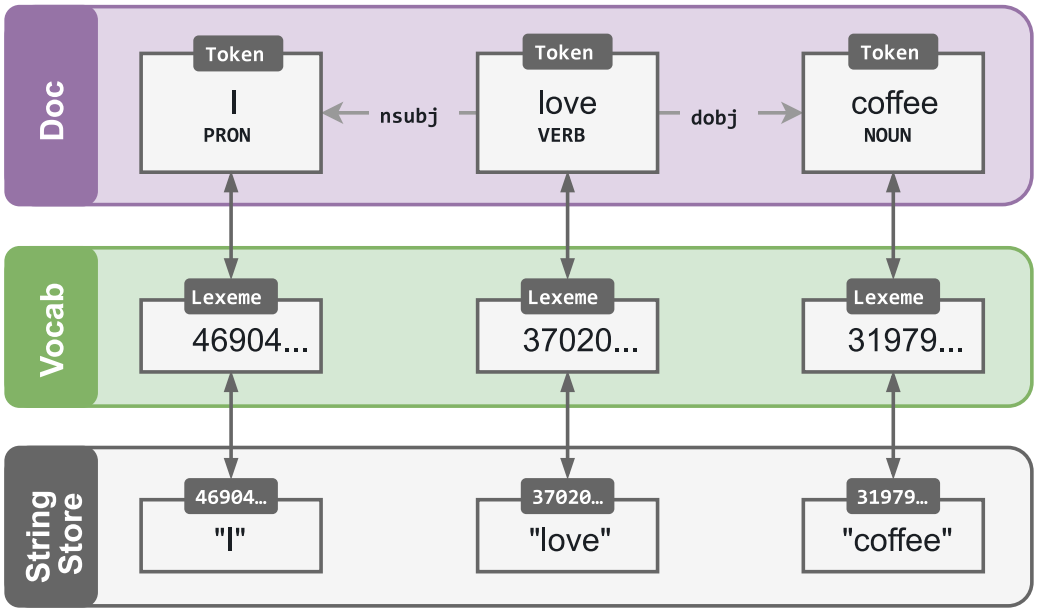


Figura 3: Representação esquemática dos objetos *Doc*, *Vocab* e *String Store*

1.2 Funcionalidades

A biblioteca spaCy disponibiliza diversas funcionalidades, sendo que algumas estão associadas a conceitos linguísticos e outras a capacidades mais gerais de *machine learning*. Apesar de algumas destas funcionalidades trabalharem de forma independente, outras requerem o carregamento de modelos estatísticos, para que o spaCy seja capaz de prever anotações linguísticas. Atualmente, o spaCy providencia modelos estatísticos para 8 linguagens, que podem diferir em tamanho, velocidade, utilização de memória, correção e os dados que incluem.

As principais funcionalidades desta biblioteca são sumarizadas de seguida:

• *Tokenization* – Separação de textos em segmentos significativos, como palavras, pontuação, entre outros. Esta divisão é efetuada tendo em conta regras específicas de cada linguagem. O *input* corresponde a um texto limpo e o *output* um objeto *Doc*.

• *POS Tagging* – atribuir tipos de palavras ou propriedades gramaticais (como verbo ou nome) a *tokens*, que constituem um *Doc*. O spaCy recorre a modelos estatísticos para prever quais as *tags* ou etiquetas que mais provavelmente se aplicam ao contexto currente. As anotações linguísticas do tipo POS são obtidas a partir do atributo *pos_* do *Token*.

• *Dependency Parsing* – Processo de obtenção de relações de dependência entre os elementos de uma frase através de um *parser*. Este *parser* possibilita a iteração do texto *tokenizado* (*Doc*) pelas suas frases nominais, sendo que as dependências são dispostas numa árvore de fácil manipulação. É ainda possível visualizar mais facilmente as dependências de um ou mais *Docs* através de um grafo, cujos arcos possuem uma legenda que caracteriza a ligação sintática entre os *tokens*.

• *Lemmatization* – redução de palavras à sua formas base, sendo frequentemente utilizado para standardizar palavras com significados similares. O *lemma* de 'era' e 'foi', e.g., é 'ser'. Pode-se aceder ao *lemma* através do atributo *lemma_* do *Token*.

• *Sentence Boundary Detection (SBD)* – Processo de separação de frases de um texto separado pelos seus *tokens* (*Doc*). É utilizado o *parsing* de dependências para uma divisão mais correta. Possibilita a definição manual dos limites frásicos (i.e. *boundaries*) em detrimento dos pré-definidos '.', '!' e '?'.

• *Named Entity Recognition (NER)* - atribuição de etiquetas ou categorias pré-definidas a nomes correspondentes a objetos do mundo real, como pessoas, companhias, localizações, países, produtos, entre outros. O spaCy reconhece vários tipos de nomes de entidades num documento, através da solicitação de uma previsão ao modelo. Como estes modelos são estatísticos e muito dependentes dos textos com que foram treinados, nem sempre esta previsão é correta, podendo ser necessário posterior processamento específico para o caso em estudo. Os nomes das entidades estão disponíveis na propriedade *ents* de um *Doc* ou na propriedade *ent_type_* de um *Token*. Quando se pretende

alterar anotações de entidades, tem de se o faze a nível do documento, de forma a assegurar que a sequência de anotações de *tokens* permanece consistente.

• ***Similarity*** – Comparação de dois textos e cálculo de valor que indique quão similares são. A função de similitude compara palavras de um dado texto com base em vetores de palavras específicos para cada linguagem que podem ser customizados para um melhor desempenho em contextos de aplicação específicos.

• ***Text Classification*** – atribuição de categorias ou etiquetas a todo o documento ou partes de um documento.

• ***Rule-based matching*** – Processo de busca de correspondências entre *tokens* e determinados padrões. Estes padrões podem ser bastante complexos, tendo algumas parecenças com o funcionamento expressões regulares. No entanto, podem também tirar partido do conhecimento léxico desta ferramenta, filtrando, por exemplo, por bases morfológicas (i.e. *Lemma*) ou classes gramaticais (i.e. verbos, substantivos, etc.). Possibilita ainda a formulação de padrões mais complexos através de quantificadores, *wildcards*, expressões regulares. Podem ainda ser definidas, de um modo simples, funções que serão chamadas caso se dê uma ou mais correspondências.

• ***Training*** – atualização e melhoria de modelos estatísticos de previsões. Para treinar um modelo, são necessários dados de treino, i.e., exemplos de texto e etiquetas (POS *tag*, *named entity*, etc.) que se pretende que o modelo seja capaz de prever. O modelo pode ser melhorado através do processamento de novos textos limpos, em que o utilizador, que conhece a correção da resposta, dá *feedback* ao modelo sobre a sua previsão na forma de um gradiente de erro da função de perda que calculam a diferença entre o exemplo de treino e o *output* esperado (figura 4). Quanto maior a diferença, mais significativo o gradiente e as atualizações ao modelo.

• ***Serialization*** – Funcionalidade responsável por guardar objetos do *spaCy* para ficheiros em disco, bem como por carregar objetos previamente guardados. Este processo torna-se especialmente importante quando usado para guardar objetos do tipo *Doc*, por exemplo, que possuem todas as informações retiradas do texto original, bem como outras características inseridas manualmente como entidades ou vetores de palavras customizados. Para isso, é utilizado a biblioteca *Pickle*,

built-in da linguagem *Python*.

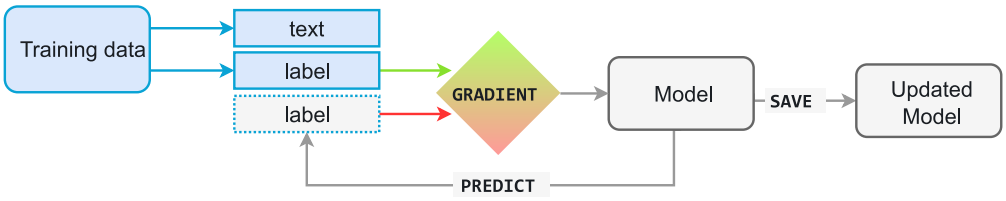


Figura 4: Esquema representativo do funcionamento do processo de treino

1.3 Vantagens

Existem diversas ferramentas do género do spaCy que podem ser utilizadas para efetuar processamento de linguagem natural. Assim, há que apresentar as principais distinções entre estas, de forma a perceber qual será a que trará mais vantagens na sua utilização.

Em primeiro lugar, pode-se efetuar uma breve comparação entre as funcionalidades oferecidas pelo spaCy e outras ferramentas semelhantes, como SyntaxNet, NLTK e CoreNLP.

	spaCy	**SyntaxNet**	**NLTK**	**CoreNLP**
Programming language	Python	C++	Python	Java
Neural network models	✓	✓	✗	✓
Integrated word vectors	✓	✗	✗	✗
Multi-language support	✓	✓	✓	✓
Tokenization	✓	✓	✓	✓
Part-of-speech tagging	✓	✓	✓	✓
Sentence segmentation	✓	✓	✓	✓
Dependency parsing	✓	✓	✗	✓
Entity recognition	✓	✗	✓	✓
Coreference resolution	✗	✗	✗	✓

Tabela 1: Funcionalidades de diversas ferramentas NLP

Como se pode perceber através da análise da tabela 1, o spaCy disponibiliza todas as principais funcionalidades associadas o processamento de linguagem natural, exceto a *Conference Resolution*, que corresponde à tarefa de encontrar todas as expressões que se referem à mesma entidade num texto. Apenas o CoreNLP possui essa capacidade, sendo que, em contrapartida, não possui *Integrated word vectors*, que possibilitam a previsão de quão similares são dois objetos. Assim, percebe-se que o spaCy será a melhor alternativa em muitos dos casos.

Outro fator fundamental na comparação entre o spaCy e outras ferramentas semelhantes, é o desempenho, i.e., o tempo que o spaCy demora a gerar respostas. Dois artigos revistos em 2015 (REF) confirmam que o spaCy oferece o *parser* sintático mais rápido do mundo e que a sua precisão está dentro de 1% do melhor disponível. Os poucos sistemas mais precisos são pelo menos 20 vezes mais lentos. Alguns valores de precisão e velocidade são apresentados na tabela 2. Informações mais detalhadas relativamente à velocidade são apresentadas na tabela 3.

System	**Year**	**Language**	**Accuracy**	**Speed (wps)**
spaCy v2.x	2017	Python/Cython	92.6	n/a
spaCy v1.x	2015	Python/Cython	91.8	13,963
ClearNLP	2015	Java	91.7	10,271
CoreNLP	2015	Java	89.6	8,602
MATE	2015	Java	92.5	550
Turbo	2015	C++	92.4	349

Tabela 2: Precisão e velocidade de várias ferramentas NLP

A partir das informações das tabelas 2 e 3, percebe-se que o spaCy deverá ser uma boa escolha, também em termos de precisão e desempenho.

Há, contudo, que se ter em atenção o tipo de texto que se irá utilizar e o tipo de processamento que se pretende efetuar, uma vez que os valores de precisão, o tamanho do modelo e a velocidade de processamento dependem de vários fatores, como, por exemplo, a linguagem ou modelo utilizado.

	Absolute (ms per doc)			Relative (to spaCy)		
System	**Tokenize**	**Tag**	**Parse**	**Tokenize**	**Tag**	**Parse**
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	n/a	20x	443x	n/a

Tabela 3: Informações detalhadas sobre velocidade de diversas ferramentas NLP

1.4 Utilização

O processo de instalação da biblioteca é bastante simples, podendo ser realizado através do *pip* (*package manager* da linguagem *Python*), utilizado, neste exemplo, na sua versão 3.X.

```
$ pip3 install spacy
```

Os modelos correspondentes às linguagens podem ser também facilmente obtidos através do seguinte comando abaixo, exemplificado para um dos modelos da língua inglesa.

```
$ python3 -m spacy download en_core_web_sm
```

Após o processo de instalação da biblioteca e de *download* das linguagens, o primeiro passo será importar a biblioteca. Depois, poder-se-á carregar a linguagem já descarregada e produzir um *Doc* a partir de um qualquer texto. Será então possível começar imediatamente a tirar proveito das funcionalidades providenciadas pela biblioteca, utilizando o objeto do tipo *Doc* resultante do processamento do texto inserido.

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_sm')
>>> doc = nlp('This text can now be processed using spaCy!')
```

1.5 Aplicações

Com todas as funcionalidades do spaCy, consegue-se concretizar diversos tipos de processamento de texto de forma geralmente simples. Algumas possibilidades são as que se indicam de seguida:

- desenhar os gráficos de dependências num texto;
- reconhecer entidades num texto;
- extrair relações entre frases e entidades utilizando as funcionalidades de *entity recognizer* e de *dependency parse*;
- obter marcações de um conjunto personalizado de entidades, alterando-se para isso anotações de entidades com base, e.g., numa lista de nomes desse conjunto e juntando-se entidades num único *token*;
- corrigir ou personalizar atributos nos objetos *Doc*, *Span* e/ou *Token*;
- treinar o *Named Entity Recognizer*, o *Dependency Parser*, o *Part-of-speech Tagger* ou o *Text Classifier* do spaCy, partindo de um modelo já treinado ou de um em branco;

Estes exemplos apresentados são de implementação relativamente fácil, tendo-se concretizado alguns dos mais simples.

1.5.1 Grafos de Dependências

Relativamente ao desenho dos grafos de dependências, é possível desenvolver uma função que os imprima para ficheiros com formato *svg* ou para *html*, sendo esta apresentada no excerto de código do anexo A) com o nome de *generate_dependencies_graph*.

Para obter as dependências dentro de cada frase, basta separar as mesmas do objeto *Doc* e invocar sobre estas a função *serve* (mostra no *browser*) ou *render* (devolve conteúdo a guardar num documento *svg* ou *html*) com o estilo *dep*. Existem outras formas de realizar esta tarefa, mas esta é a mais simples e ainda permite personalizar o grafo gerado, como mudar a cor do fundo, tipo de letra, entre outros.

Esta função pode ser utilizada da seguinte forma:

```
>>> import spacy
>>> from spacy import displacy
>>> nlp = spacy.load('pt')
>>> doc = nlp('Pretende-se gerar o gráfico de dependências do
spaCy!')
>>> generate_dependencies_graph(doc)

Serving on port 5000...
Using the 'dep' visualizer
```

Acedendo-se ao endereço `127.0.0.1:5000/` no *browser*, obtém-se o resultado apresentado na figura 5. Também se poderia solicitar o output em ficheiro(s) *html* ou *svg*, aterando o argumento *type*. Nesse caso, o conteúdo dos ficheiros seria retornado pela função. Os restantes argumentos, também opcionais, conferem diferentes características estéticas ao *output*.

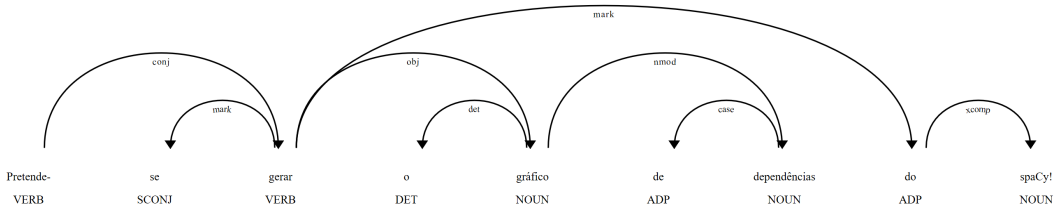


Figura 5: *Output* de invocação da função *generate_dependencies_graph*

1.5.2 Reconhecimento de Entidades

Para se proceder ao reconhecimento de entidades, desenvolveu-se a função *generate_tagged_text* do excerto de código do anexo A). Essa função permite obter resultados no *browser*, em ficheiros *html* ou na *shell*.

Para desenvolver esta funcionalidade, basta invocar sobre o *Doc* a função *displacy.serve* (mostra no *browser*) ou *displacy.render* (devolve conteúdo a guardar num documento *html*) com o estilo *ent*. Para se obter o *output* em formato de texto, percorreu-se todos os *tokens* do *Doc* e, caso este tivesse uma entidade associada (atributo *ent_typ_*), a mesma seria escrita junto ao *token*.

Esta função pode ser utilizada da seguinte forma, para obter o *output* no *browser*:

```
>>> import spacy
>>> from spacy import displacy
>>> nlp = spacy.load('pt')
>>> doc = nlp('Hoje o Manuel gostava de ir ao cinema do Braga
Parque!')
>>> generate_tagged_text(doc)

Serving on port 5000...
Using the 'ent' visualizer
```

Acedendo-se ao endereço `127.0.0.1:5000/` no *browser*, obtém-se o resultado apresentado na figura 6. Também se poderia solicitar o output em ficheiro *html*, aterando o argumento *type*. Nesse caso, o conteúdo do ficheiro seria retornado pela função. O atributo *entities* permite indicar quais as entidades que se pretendem etiquetar. Caso nenhuma seja indicada, todas são etiquetadas. O argumento *color*, também opciona, confere diferentes cores aos *labels* das entidades.

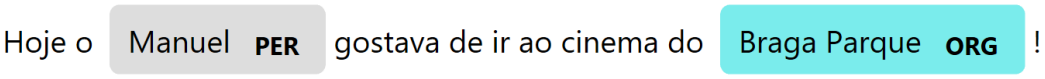


Figura 6: *Output* de invocação da função *generate_tagged_text*

Caso se pretenda obter o resultado em texto, pode-se realizar o seguinte comando:

```
>>> generate_tagged_text(doc, 'text')

'Hoje o Manuel{PER} gostava de ir a o cinema do Braga{ORG}
Parque{ORG} !'
```

1.5.3 Especificação de *Tokens*

Com o spaCy consegue-se, de forma bastante simples, alterar informação associada a determinado *Token*, ou gerar uma nova divisão para determinada palavra. A palavra inglesa *don't*, e.g., é geralmente dividida em *do* e *n't*

(*not*), mas tal não se verifica no Reino Unido, onde esta deve permanecer sempre um único *Token*.

A função *add_tokenizer_exceptions*, apresentada no excerto de código do anexo A, demonstra como personalizar um *Token*, ao adicionar ao *Tokenizer* do modelo casos especiais.O argumento *tokens* dessa função deverá ser um dicionário, em que a chave é o *Token* original e o valor é uma lista de dicionários. Esta lista de dicionários corresponde aos atributos do *Token*, devendo conter o nome do atributo e o respetivo valor como chave e valor, respetivamente. Note-se que o atributo *ORTH* deve ser sempre especificado e, todos concatenados, devem formar o *Token* original.

A referida função pode ser utilizada da seguinte forma:

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_lg')
>>> doc = nlp("We don't need to go now!")
>>> [print((token.orth_, token.lemma_, token.tag_,
token.pos_)) for token in doc]
('We', '-PRON-', 'PRP', 'PRON')
('do', 'do', 'VBP', 'VERB')
("n't", 'not', 'RB', 'ADV')
('need', 'need', 'VB', 'VERB')
('to', 'to', 'TO', 'PART')
('go', 'go', 'VB', 'VERB')
('now', 'now', 'RB', 'ADV')
('!', '!', '.', 'PUNCT')

>>> tokens = {"don't": [{'ORTH': 'do', 'LEMMA': 'do',
'POS': 'VERB'}, {'ORTH': "n't", 'LEMMA': 'not'}]}
>>> doc = nlp("We don't need to go now!")
>>> [print((token.orth_, token.lemma_, token.tag_,
token.pos_)) for token in doc]
('We', '-PRON-', 'PRP', 'PRON')
("don't", "don't", '', 'VERB')
('need', 'need', 'VB', 'VERB')
('to', 'to', 'TO', 'PART')
('go', 'go', 'VB', 'VERB')
('now', 'now', 'RB', 'ADV')
('!', '!', '.', 'PUNCT')
```

1.5.4 Treino do *dependency parser*

Como foi dito anteriormente, o *spaCy* possibilita o treino de novos modelos estatísticos utilizados nas suas mais diversas funcionalidades. Umas dessas funcionalidades, demonstrada neste exemplo, é o *dependency parser*.

Em primeiro lugar, é criada uma variável *TRAIN_DATA* que possui duas frases nominais, bem como o nome das dependências (*deps*) e quais os elementos principais dessas mesmas dependências (*heads*). Esta variável servirá como base para o processo de treino. De seguida, é criada a linguagem que poderá ser uma das pré-existentes, ou criada de raíz (i.e. *blank*). Depois, é adicionado o *parser* à *pipeline* e adicionadas as etiquetas com os nomes das dependências ao *parser* que é finalmente utilizado para treinar a linguagem utilizando a variável *TRAIN_DATA* criada anteriormente. O resultado é, a seguir, testado, através da impressão do resultado das dependências para um novo texto.

Adicionalmente, tira-se ainda partido da capacidade de serialização do *spaCy* (abordada anteriormente neste relatório) para guardar o modelo da linguagem que se encontra modificada após o processo de treino.

O código deste exemplo está presente no Anexo C, encontrando-se também disponível no repositório do *GitHub* REF.

2 spaCy's POS Tagging

A funcionalidade de POS *Tagging* do spaCy foi explicitada anteriormenta, mas merece especial destaque por ser o principal tema do trabalho.

Um modelo consiste em dados binários e é produzido ao apresentar ao sistema exemplos suficientes para que ele faça previsões que são generalizadas a toda a linguagem. Há a necessidade de prever, e.g., a classe (POS) das várias palavras, podendo a mesma palavra ter diferentes classes em contextos distintos ('gosto' pode ser um verbo ou um nome, dependendo da frase em que está incluída). Este tipo de previsão permite efetuar novas conjeturas, como que tipo de palavra se espera encontrar após outra (e.g., a palavra que se segue ao determinante artigo definido 'o' será muito provavelmente um nome).

Para além de palavras com a mesma classe tenderem a seguir uma estrutura sintática semelhante, também são úteis em processamento baseado

em regras. Assim, consegue-se perceber a importância de determinar corretamente a classe (*part-of-speech*) de cada palavra num determinado texto de *input*.

O spaCy utiliza as POS *tags* do projeto *Penn Treebank* REF.

2.1 Utilização

Após ter sido instalada a biblioteca e feito o *download* das linguagens (processo descrito anteriormente), poder-se-á testar o processo de obtenção das *POS tags* para um determinado texto, de forma simples. Para tal, bastará introduzir um texto de input e – como apresenta o código abaixo –, imprimir, por exemplo, a informação sobre as *POS tags* de cada uma das suas palavras.

```
>>> import spacy
>>> nlp = spacy.load('en_core_web_sm')
>>> doc = nlp('This is a demonstration of a very cool
Python library!')
>>> for token in doc:
...    print(token.text, token.pos_)
This DET
is VERB
a DET
demonstration NOUN
of ADP
a DET
very ADV
cool ADJ
Python PROPN
library NOUN
! PUNCT
```

3 Exemplo Demonstrativo

Para construir um exemplo demonstrativo de algumas das funcionalidades do spaCy mais interativo, utilizou-se a biblioteca *flask* para correr a

aplicação no *browser* com uma interface gráfica baseada em *templates html*. Esta ferramenta foi utilizada apenas como auxiliar, não estando relacionada com o tema do trabalho, pelo que não se aprofundará o funcionamento da mesma. Contudo, se o utilizador preferir, pode realizar todas as operações a partir da linha de comandos, através da utilização de opções.

O código associado a esta parte da aplicação, à interação com o utilizador, é apresentado no anexo B. A utilização da mesma é indicada pela própria aplicação quando se solicita ajuda (`-h`) ou se insere um comando inválido:

```
    USAGE:  python3 app.py [(-i <outputfile> | -p <outputfile>
    | -t <outputfile> | -g <outputfile>) [-l <language>]
    <inputfile>]

    OPTIONS:
        -l   language - 'pt' (default) or 'en'
-i   returns table with information about tokens to
outputfile (or 'shell')
-p   generates bar chart with POS tag frequence to
outputfile
        -t   returns tagged text to outputfile (or 'shell')
        -g   generates dependencies graphs to outputfile.svg
* if no options are provided, a web server will be
initialized, where the input will be inserted and the
output presented.
```

Nos anexos REF a REF apresentam-se os resultados obtidos na aplicação *web*

Para se implementar as funcionalidades disponibilizadas pelo spaCy, utilizou-se as funções anteriormente referidas, que se encontram no anexo A. No ambiente *web*, o reconhecimento de entidades permite fazer uma filtragem de quais serão etiquetadas. Nesse contexto, todos os *outputs* obtidos são do tipo *html*, sendo depois integrados nos *templates* existentes.

Para além das funções anteriormente referidas, podem-se encontrar no código apresentado no anexo A outras mais direccionadas ao tema principal do projeto: POS *tagging*.

3.1 Gráfico de frequência POS

A função *generate_pos_chart* é responsável pela geração de um gráfico que indica a frequência de cada POS *tag* num determinado *Doc*. Para se obter as frequências, basta utilizar a função *count_by* do *Doc* e passar-lhe como argumento o identificador de POS (*spacy.attrs.POS*). O resultado dessa invocação é um dicionário com o valor de *hash* do POS como chave e o número de vezes que se repete como valor. De seguida, basta obter o nome associado aos valores de *hash*. Caso seja solicitado o *output* do tipo *html*, é retornada uma lista de listas, sendo que cada lista interna armazena o par (POS, frequência). Essa lista poderá então ser utilizada para construir um *Google Chart*. Caso seja solicitado o *output* em formato *pict* (imagem), é guardado um gráfico de barras como imagem.

Esta função pode ser utilizada da seguinte forma:

```
>>> import spacy
>>> import matplotlib as mpl
>>> mpl.use('Agg')
>>> from matplotlib import pyplot as plt
>>> nlp = spacy.load('pt')
>>> doc = nlp('Hoje o João e a Maria tiveram de ir ao hospital
visitar a Teresa.')
>>> generate_pos_chart(doc, type='pict')
```

3.2 Tabela com informações sobre *Tokens*

A função *generate_information* permite obter várias informações sobre os *Tokens* de um documento. Para isso, basta percorrer os *tokens* do *Doc* e aceder aos atributos do mesmo, conforme explicitado previamente. Se o *output* for solicitado no tipo *html*, retorna-se uma tabela nesse formato. Caso contrário, retorna-se uma *PrettyTable* que poderá ser impressa no terminal.

```
>>> import spacy
>>> from prettytable import PrettyTable
>>> nlp = spacy.load('en_core_web_lg')
>>> doc = nlp('We have to go there now!')
>>> print(generate_information(doc, nlp.vocab, type='text'))
```

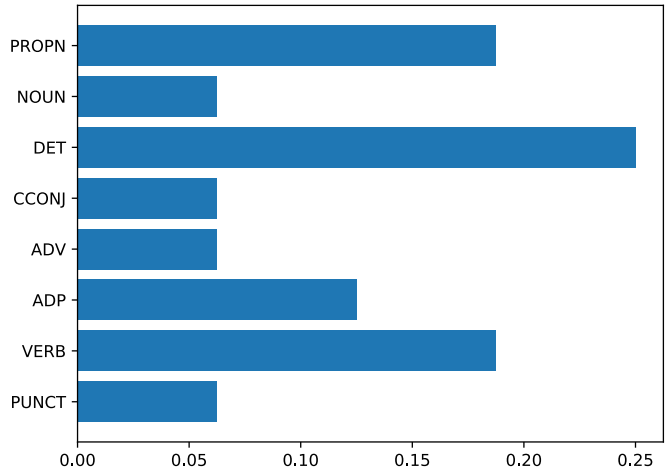


Figura 7: *Output* de invocação da função *generate_pos_chart*

Text	Lemma	POS	TAG	DEP	SHAPE	MORPHOLOGIAL INFO	IS_ALPHA	IS_STOP
We	-PRON-	PRON	PRP	nsubj	Xx	{'PronType': 'prs'}	True	False
have	have	VERB	VBP	ROOT	xxxx	{'VerbForm': 'fin', 'Tense': 'pres'}	True	False
to	to	PART	TO	aux	xx	{'PartType': 'inf', 'VerbForm': 'inf'}	True	False
go	go	VERB	VB	xcomp	xx	{'VerbForm': 'inf'}	True	False
there	there	ADV	RB	advmod	xxxx	{'Degree': 'pos'}	True	False
now	now	ADV	RB	advmod	xxx	{'Degree': 'pos'}	True	False
!	!	PUNCT	.	punct	!	{'PunctType': 'peri'}	False	False

Figura 8: *Output* de invocação da função *generate_info*

3.3 Resultados

Os resultados obtidos através da execução via linha de comandos são iguais aos previamente recolhidos para a execução das funções individualmente. No contexto *wed*, apresentam-se nas imagens do anexo REF exemplos de funcionamento de cada uma das funcionalidades.

Anexos

A *Working Example*: código associado à utilização de spaCy

```
import os
import re
import pandas as pd
import spacy
from spacy import displacy
from spacy.util import update_exc
from pathlib import Path
from prettytable import PrettyTable
import matplotlib as mpl
mpl.use('Agg')
from matplotlib import pyplot as plt

# Gerar tabelas (auxiliares)
def generate_html_table(headers, data):
    res = '<table class="table table-striped">\n'
    res += '  <thead><tr><th>' + '    </th><th>'.join(headers)
    ↪  + '  </th></tr></thead>\n  <tbody>\n'
    for sublist in data:
        res += '  <tr><td>'
        res += '    </td><td>'.join(sublist)
        res += '  </td></tr>\n'
    res += '  </tbody>\n</table>\n'
    return res

def generate_table(headers, data):
    table = PrettyTable()
    table.field_names = headers
    for row in data:
        table.add_row(row)
    return table


```

```
# Part-of-speech tagging
def generate_pos_chart(doc, filename='pos_frequence.svg',
    type='html'):
    tag_dict = {w.pos : w.pos_ for w in doc}
    pos_freq = []
    pos_tags = []
    for pos_id, freq in doc.count_by(spacy.attrs.POS).items():
        pos_freq.append(freq / len(doc))
        pos_tags.append(tag_dict[pos_id])
    if type=='html':
        return [['POS Tag', 'POS Frequence (%)']] + [list(x)
        for x in zip(pos_tags,pos_freq)]
    else:
        plt.barh(range(1,len(pos_tags)+1), pos_freq,
        tick_label=pos_tags)
        plt.savefig(filename)


# Get token's information
def generate_information(doc, vocab, type='html'):
    # python3 -m spacy download pt
    headers = ["Text","Lemma", "POS", "TAG", "DEP", "SHAPE",
    "MORPHOLOGIAL INFO", "IS_ALPHA", "IS_STOP"]
    data = []
    tokens = []

    for token in doc:

        if (str(token.pos_) != 'SPACE' and str(token.pos_) !=
        'PUNCT') or token.text not in tokens:

            if token.tag_:
                morph_info = dict(filter(lambda x : x[0]!=74,
                vocab.morphology.tag_map[token.tag_].items()))
                if not morph_info:
                    morph_info = ''

            else:
                morph_info = ''
```

```
65              data.append([str(s) for s in (token.text,
                ↪  token.lemma_, token.pos_, token.tag_,
                ↪  token.dep_, token.shape_, morph_info,
                ↪  token.is_alpha, token.is_stop)])
66              tokens.append(token.text)
67
68      if type=='html':
69          return generate_html_table(headers, data)
70      else:
71          return generate_table(headers, data)
72
73
74
75  # Dependencies Graph
76  def generate_dependencies_graph(doc, type='service', compact =
    ↪  False, background='white', color='black', font='Source
    ↪  Sans Pro'):
77      res = []
78      if type in ('service', 'html', 'pict'):
79          docs = list(doc.sents)
80          options = {'compact': compact, 'bg': background,
            ↪  'color': color, 'font': font}
81      if type == 'service':
82          displacy.serve(docs, style='dep', options=options)
83      elif type == 'html':
84          html = displacy.render(docs, style='dep', page=True,
            ↪  options=options)
85          html = re.sub(r'.*<body[^>]*>(.*)</body>.*', r'\1',
            ↪  html, flags=re.DOTALL)
86          res.append(html)
87      elif type == 'pict':
88          for doc in docs:
89              pict = displacy.render(doc, style='dep',
                ↪  options=options)
90              res.append(pict)
91      return res
92
93
94
```

```
# Visualizing the entity recognizer
def generate_tagged_text(doc, type = 'server', entities =
↪  None, colors = None):
    res = ''
    if type in ('server', 'html'):
        options = {}
        if entities:
            options['ents'] = entities
        if colors:
            options['colors'] = colors
        if type=='server':
            displacy.serve(doc, style='ent', options=options)
        else:
            res = displacy.render(doc, style='ent',
            ↪  options=options)
    else:
        res_list = []
        for word in doc:
            if word.ent_type_:
                res_list.append(str(word) + '{' +
                ↪  word.ent_type_ + '}')
            else:
                res_list.append(str(word))
        res = ' '.join(res_list)
    return res

# Rule-based morphology
def add_tokenizer_exceptions(nlp, tokens, tokenizer=None):
    if not tokenizer:
        tokenizer = nlp.tokenizer
    for token, token_attrs in tokens.items():
        tokenizer.add_special_case(token, token_attrs)
```

B *Working Example*: código associado à interação com o utilizador

```
import sys
import getopt
import json
import spacy
from pos_tagging import *
from flask import Flask, request, url_for, redirect,
    render_template

# Processar argumentos do comando
def processArgs():
    info_out = ''
    pos_chart_out = ''
    tagged_text_out = ''
    graphs_out = ''
    lang = 'pt'
    inputfile = ''

    error = 'USAGE:\tpython3 app.py [(-i <outputfile> | -p
        <outputfile> | -t <outputfile> | -g <outputfile>) [-l
        <language>] <inputfile>]\n\n' + \
            'OPTIONS:\n' + \
            '\t-l\tlanguage - \'pt\' (default) or \'en\'\n' +
            \
            '\t-i\treturns table with information about tokens
            to outputfile (or \'shell\')\n' + \
            '\t-p\tgenerates bar chart with POS tag frequence
            to outputfile\n' + \
            '\t-t\treturns tagged text to outputfile (or
            \'shell\')\n' + \
            '\t-g\tgenerates dependencies graphs to
            outputfile.svg\n' +\
            '* if no options are provided, a web server will
            be initialized, where the input will be
            inserted and the output presented.'

```

```
26      # Processar opções/argumentos do comando utilizado
27      try:
28          opts, args =
            ↪  getopt.getopt(sys.argv[1:],"i:p:t:g:l:hv",
            ↪  ["info=", "pos-chart=", "tagged-text=", "graphs=",
            ↪  "lang=", "help", "version"])
29      except getopt.GetoptError:
30          print(error)
31          sys.exit(2)
32      for opt, arg in opts:
33          if opt in ('-h', '--help'):
34              print(error)
35              sys.exit()
36          elif opt in ('-v', '--version'):
37              print('Version 1.0')
38              sys.exit()
39          elif opt in ("-i", "--info"):
40              info_out = arg
41          elif opt in ("-p", "--pos-chart"):
42              pos_chart_out = arg
43          elif opt in ("-t", "--tagged-text"):
44              tagged_text_out = arg
45          elif opt in ("-g", "--graphs"):
46              graphs_out = arg
47          elif opt in ("-l", "--lang"):
48              lang = arg
49      if info_out or pos_chart_out or tagged_text_out or
        ↪  graphs_out:
50          if len(args) == 1:
51              inputfile = args[0]
52          else:
53              print(error)
54              sys.exit(2)
55      elif len(args)>0:
56          print(error)
57          sys.exit(2)
58
59      return info_out, pos_chart_out, tagged_text_out,
        ↪  graphs_out, lang, inputfile
```

```
60
61  app = Flask(__name__)
62
63  @app.route('/', methods=['GET', 'POST'])
64  def index():
65      global lang, nlp, tagged_text, info, graphs, bar_chart
66      if request.method == 'POST':
67          lang = request.values.get('lang')
68          entities = request.form.getlist('entity')
69          token_exceptions =
            ↪  json.loads(request.values.get('tokenExceptions'))
70          add_tokenizer_exceptions(nlp[lang], token_exceptions)
71          doc = nlp[lang](request.values.get('input'))
72          tagged_text = generate_tagged_text(doc, type = 'html',
            ↪  entities = entities)
73          info = generate_information(doc, nlp[lang].vocab)
74          bar_chart = generate_pos_chart(doc, None, type='html')
75          graphs = '\n'.join(generate_dependencies_graph(doc,
            ↪  type = 'html'))
76          return redirect(url_for('tagged_text_form'))
77      return render_template('index.html', lang=lang)
78
79
80  @app.route('/tagged_text_form', methods=['GET', 'POST'])
81  def tagged_text_form():
82      global tagged_text
83      if request.method == 'POST':
84          return redirect(url_for('index'))
85      return render_template('tagged_text_form.html',
        ↪  tagged_text=tagged_text)
86
87
88  @app.route('/info_form', methods=['GET', 'POST'])
89  def info_form():
90      global info, bar_chart
91      if request.method == 'POST':
92          return redirect(url_for('index'))
93      return render_template('info_form.html', table=info,
        ↪  bar_chart=bar_chart)
```

```
94
95  @app.route('/graphs_form', methods=['GET', 'POST'])
96  def graphs_form():
97      global graphs
98      if request.method == 'POST':
99          return redirect(url_for('index'))
100     return render_template('graphs_form.html', graphs=graphs)
101
102
103 if __name__ == '__main__':
104     global tagged_text, info, graphs
105     info_out, pos_chart_out, tagged_text_out, graphs_out,
        ↪ lang, inputfile = processArgs()
106     if inputfile:
107         if lang == 'en':
108             nlp = spacy.load('en_core_web_lg')
109         else:
110             nlp = spacy.load('pt')
111         with open(inputfile, 'r') as input:
112             doc = nlp(input.read())
113         if info_out:
114             output = generate_information(doc, nlp.vocab,
                ↪ 'text')
115             if info_out == 'shell':
116                 print(output)
117             else:
118                 with open(info_out, 'w') as fd:
119                     fd.write(output)
120         if pos_chart_out:
121             generate_pos_chart(doc, pos_chart_out,
                ↪ type='pict')
122         if tagged_text_out:
123             output = generate_tagged_text(doc, 'text',
                ↪ nlp.vocab)
124             if tagged_text_out == 'shell':
125                 print(output)
126             else:
127                 with open(tagged_text_out, 'w') as fd:
128                     fd.write(output)
```

```
129         if graphs_out:
130             os.makedirs('./images', exist_ok=True)
131             pictures = generate_dependencies_graph(doc,
                ↪  'pict')
132             for i, pict in enumerate(pictures):
133                 file_name = graphs_out + '_' + str(i) + '.svg'
134                 output_path = Path('./images/' + file_name)
135                 with output_path.open('w', encoding='utf-8')
                    ↪  as output_file:
136                     output_file.write(pict)
137     else:
138         nlp = {'en': spacy.load('en_core_web_lg'), 'pt':
            ↪  spacy.load('pt') }
139         tagged_text = ''
140         info = ''
141         graphs = ''
142         app.run()
```

C Exemplo de aplicação: Treino do *dependency parser* (e serialização)

```
#!/usr/bin/env python

# https://spacy.io/usage/examples#parser

from __future__ import unicode_literals, print_function

import plac
import random
from pathlib import Path
import spacy
from spacy.util import minibatch, compounding


# training data
TRAIN_DATA = [
    ("They trade mortgage-backed securities.", {
        'heads': [1, 1, 4, 4, 5, 1, 1],
        'deps': ['nsubj', 'ROOT', 'compound', 'punct', 'nmod',
        ↪  'dobj', 'punct']
    }),
    ("I like London and Berlin.", {
        'heads': [1, 1, 1, 2, 2, 1],
        'deps': ['nsubj', 'ROOT', 'dobj', 'cc', 'conj',
        ↪  'punct']
    })
]

@plac.annotations(
    model=("Model name. Defaults to blank 'en' model.",
    ↪  "option", "m", str),
    output_dir=("Optional output directory", "option", "o",
    ↪  Path),
    n_iter=("Number of training iterations", "option", "n",
    ↪  int))

```

```
31 def main(model=None, output_dir=None, n_iter=10):
32     """Load the model, set up the pipeline and train the
       ↪ parser."""
33     if model is not None:
34         nlp = spacy.load(model)  # load existing spaCy model
35         print("Loaded model '%s'" % model)
36     else:
37         nlp = spacy.blank('en')  # create blank Language
           ↪ class
38         print("Created blank 'en' model")
39
40     # add the parser to the pipeline if it doesn't exist
41     # nlp.create_pipe works for built-ins that are registered
       ↪ with spaCy
42     if 'parser' not in nlp.pipe_names:
43         parser = nlp.create_pipe('parser')
44         nlp.add_pipe(parser, first=True)
45     # otherwise, get it, so we can add labels to it
46     else:
47         parser = nlp.get_pipe('parser')
48
49     # add labels to the parser
50     for _, annotations in TRAIN_DATA:
51         for dep in annotations.get('deps', []):
52             parser.add_label(dep)
53
54     # get names of other pipes to disable them during
       ↪ training
55     other_pipes = [pipe for pipe in nlp.pipe_names if pipe !=
       ↪ 'parser']
56     with nlp.disable_pipes(*other_pipes):  # only train
       ↪ parser
57         optimizer = nlp.begin_training()
58         for itn in range(n_iter):
59             random.shuffle(TRAIN_DATA)
60             losses = {}
61             # batch up the examples using spaCy's minibatch
62             batches = minibatch(TRAIN_DATA,
               ↪ size=compounding(4., 32., 1.001))
```

```
63              for batch in batches:
64                  texts, annotations = zip(*batch)
65                  nlp.update(texts, annotations, sgd=optimizer,
                     ↪  losses=losses)
66              print('Losses', losses)
67
68      # test the trained model
69      test_text = "I like securities."
70      doc = nlp(test_text)
71      print('Dependencies', [(t.text, t.dep_, t.head.text) for t
         ↪  in doc])
72
73      # save model to output directory
74      if output_dir is not None:
75          output_dir = Path(output_dir)
76          if not output_dir.exists():
77              output_dir.mkdir()
78          nlp.to_disk(output_dir)
79          print("Saved model to", output_dir)
80
81          # test the saved model
82          print("Loading from", output_dir)
83          nlp2 = spacy.load(output_dir)
84          doc = nlp2(test_text)
85          print('Dependencies', [(t.text, t.dep_, t.head.text)
             ↪  for t in doc])
86
87
88  if __name__ == '__main__':
89      plac.call(main)
90
91      # expected result:
92      # [
93      #   ('I', 'nsubj', 'like'),
94      #   ('like', 'ROOT', 'like'),
95      #   ('securities', 'dobj', 'like'),
96      #   ('.', 'punct', 'like')
97      # ]
```

D *Working Example*: interface *web*

Figura 9: Página inicial onde se indica a linguagem, as entidades a etiquetar e o *input*

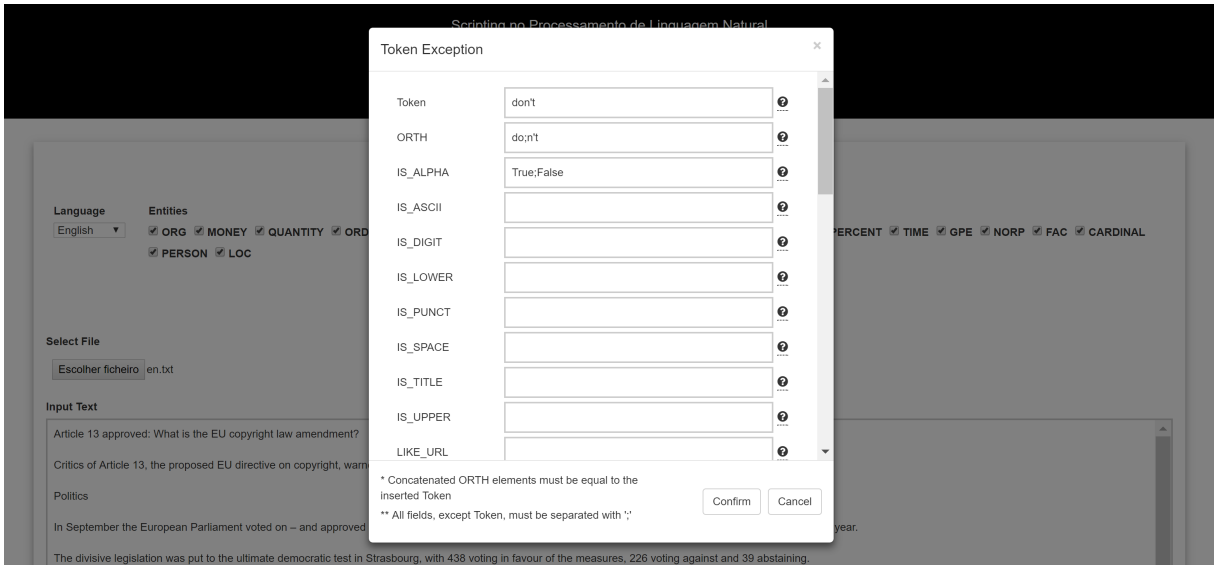


Figura 10: Modal da página inicial, onde se especifica um caso especial de *Token*

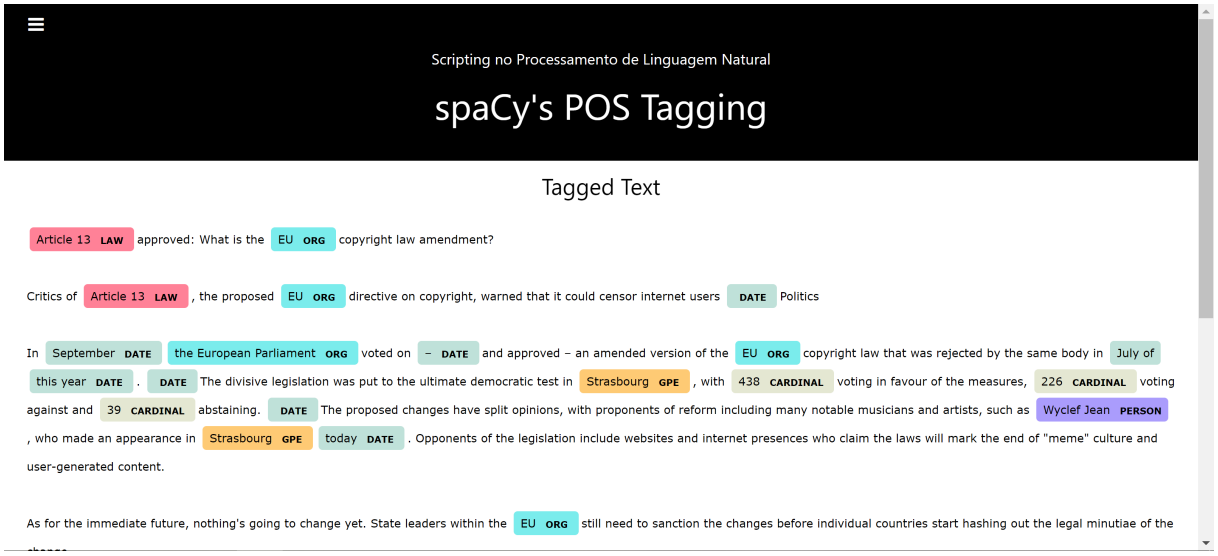


Figura 11: Página que apresenta o texto de *input* com as entidades etiquetadas

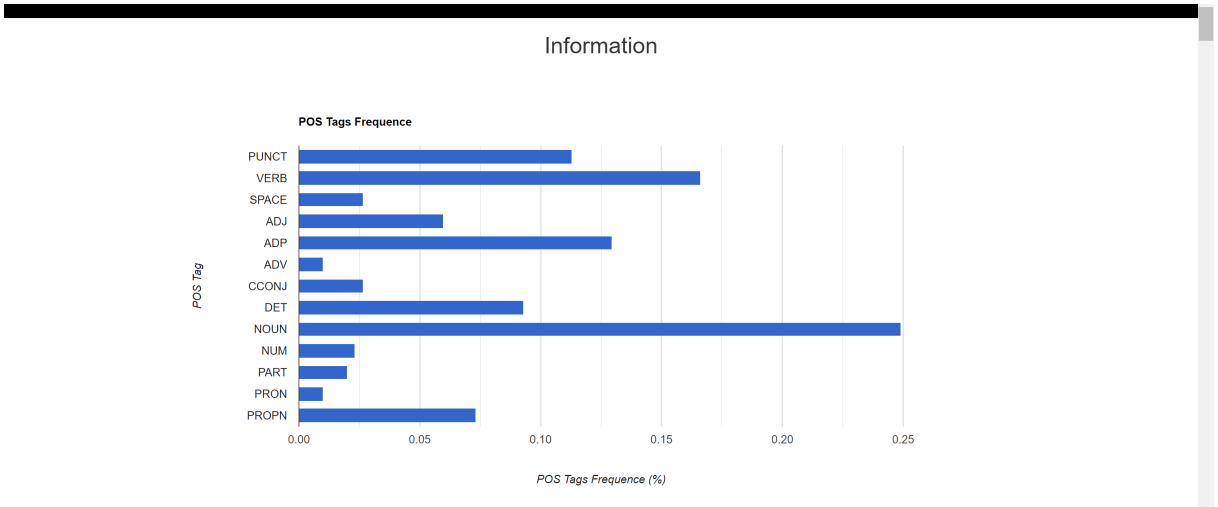


Figura 12: Excerto de página que apresenta a taxa de ocorrência de cada POS

Text	Lemma	POS	TAG	DEP	SHAPE	MORPHOLOGIAL INFO	IS_ALPHA	IS_STOP
Article	article	PROPN	NNP	nsubj	Xxxxx	{'NounType': 'prop', 'Number': 'sing'}	True	False
13	13	NUM	CD	nummod	dd	{'NumType': 'card'}	False	False
approved	approve	VERB	VBN	ROOT	xxxx	{'VerbForm': 'part', 'Tense': 'past', 'Aspect': 'perf'}	True	False
:	:	PUNCT	:	punct	:		False	False
What	what	NOUN	WP	attr	Xxxx	{'PronType': 'int|rel'}	True	False
is	be	VERB	VBZ	ccomp	xx	{'VerbForm': 'fin', 'Tense': 'pres', 'Number': 'sing', 'Person': 3}	True	False
the	the	DET	DT	det	xxx		True	False
EU	eu	PROPN	NNP	compound	XX	{'NounType': 'prop', 'Number': 'sing'}	True	False
copyright	copyright	NOUN	NN	compound	xxxx	{'Number': 'sing'}	True	False
law	law	NOUN	NN	compound	xxx	{'Number': 'sing'}	True	False
amendment	amendment	NOUN	NN	nsubj	xxxx	{'Number': 'sing'}	True	False
?	?	PUNCT	.	punct	?	{'PunctType': 'peri'}	False	False

Figura 13: Excerto de página que apresenta informações sobre cada token do texto inicial

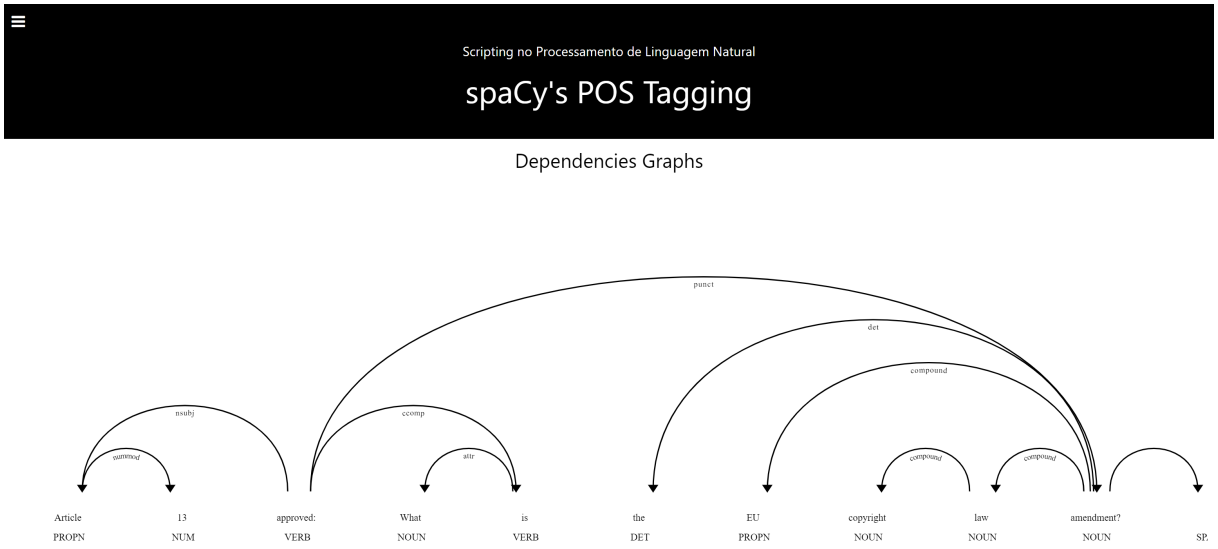


Figura 14: Página que apresenta grafos de dependências de cada frase do texto inicial

Referências

[1] Autor, "Título", *website name*, `websitelink`.