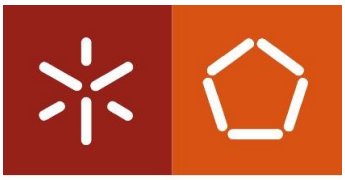

Universidade do Minho

Departamento de Informática

IPLN - TP2
spaCy (NLP framework)

Catarina Machado (a81047) Francisco Lira (a73909)

30 de Novembro de 2019

Conteúdo

Resumo

O segundo trabalho prático no âmbito da unidade curricular Introdução ao Processamento de Linguagem Natural consistiu no estudo da ferramenta **spaCy** e no desenvolvimento de três *scripts* que recorrem a esta biblioteca.

No presente relatório é apresentada uma descrição geral da ferramenta, expondo as diferentes funcionalidades da mesma, assim como três exemplos de aplicação, um simples, um intermédio e outro mais complexo, que permitem realçar a utilidade desta biblioteca. São ainda apresentados *scripts* de instalação e desinstalação, assim como a *man page* criada.

1 Introdução e Contextualização

A linguagem de programação **Python** é, atualmente, uma das mais utilizadas em todo o mundo. Trata-se de uma linguagem de *scripting*, imperativa, orientada aos objetos e funcional. As grandes vantagens associadas a esta linguagem são a sua produtividade e a legibilidade, visto que permite uma fácil leitura do código e exige poucas linhas de código, quando comparada a outras linguagens.

Devido às suas características, é principalmente utilizada para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a *web*.

Por se tratar de uma linguagem *open source* e com uma forte e muito ativa comunidade, existe um vasto leque de módulos e *frameworks* disponíveis, que permitem expandir as capacidades base da linguagem.

Uma dessas *frameworks*, que é a que abordamos no presente relatório, tem como nome **spaCy**, uma biblioteca *open source* de Python para NLP (*Natural Language Processing*).

Esta biblioteca ajuda a criar programas que processam e "compreendem" grandes volumes de texto. Pode ser usada para criar sistemas de extração de informações, compreensão de linguagem natural ou para pré-processar textos para *deep learning*.

O objetivo do presente projeto é estudar e explorar a biblioteca spaCy. Para tal, iremos fazer um estudo aprofundado da ferramenta, percebendo os componentes, organização e estrutura. Em seguida, para uma melhor compreensão da mesma, apresentaremos inicialmente um simples exemplo de utilização, com cerca de 10 linhas de código e, em seguida, um exemplo intermédio. Por fim, é apresentada a implementação de um exemplo de aplicação mais complexo.

1.1 Estrutura do Relatório

O presente relatório está dividido em 8 diferentes capítulos.

No capítulo 1, **Introdução e Contextualização**, é feito um enquadramento e contextualização do tema do trabalho prático e, em seguida, é feita uma descrição do problema a desenvolver, assim como o objetivo do projeto. Por fim, é apresentada a estrutura do presente relatório.

No capítulo 2, **Descrição Geral**, é feita uma exposição da ferramenta spaCy, apresentando as suas diferentes áreas de atuação e funcionalidades, especificando em pormenor as mais essenciais.

Em seguida, no capítulo 3, **Exemplo Simples**, é apresentado um exemplo simples de utilização da ferramenta spaCy e é explicado cada passo até se obter o resultado final.

Posteriormente, no capítulo 4, **Exemplo Intermédio**, são feitas algumas melhorias ao exemplo simples implementado na secção anterior, exemplificando para um determinado *input*.

No capítulo 5, **Exemplo Complexo**, encontra-se desenvolvido um exemplo mais complexo no contexto de NLP, devidamente detalhado e exemplificado.

No capítulo 6, **Man**, apresentamos a criação de uma *man page* para o exemplo complexo implementado.

No capítulo 7, **Script de instalação/desinstalação**, são apresentados dois *scripts*, instalação e desinstalação, das dependências necessárias para os exemplos de utilização desenvolvidos, do respetivo comando para correr o exemplo complexo e da *man page* do mesmo.

Por fim, no capítulo 8, **Conclusão**, termina-se o relatório com uma síntese e análise do que trabalho realizado.

2 Descrição Geral

SpaCy é uma biblioteca de código aberto, gratuita, para processamento avançado de linguagem natural (NLP) em **Python**.

Através da utilização desta biblioteca, é possível responder a várias perguntas, como, por exemplo: Sobre o que é este texto? O que é que as palavras significam neste contexto? Quem está a fazer o quê e com quem? Quais são as empresas e os produtos que são mencionados? Quais são os textos mais semelhantes entre si?

Tal como se pode ver, são várias as funcionalidades da ferramenta SpaCy, algumas relacionadas com conceitos linguísticos e outras mais relacionadas com *machine learning*. As funcionalidades da ferramenta são as seguintes:

- ***Tokenization:*** Segmentar o texto em palavras, sinais de pontuação, entre outros;
- ***Part-of-speech (POS) Tagging:*** Atribuir a classe gramatical de cada *token*, como, por exemplo, verbo, nome e substantivo;
- ***Dependency Parsing:*** Atribuir rótulos de dependência sintática, descrevendo as relações entre *tokens* individuais, como sujeito ou objeto;
- ***Lemmatization:*** Atribuir as formas básicas das palavras. Por exemplo, o lema de "era" é "ser" e o lema de "ratos" é "rato";
- ***Sentence Boundary Detection (SBD):*** Localizar e segmentar frases individuais;
- ***Named Entity Recognition (NER):*** Classificar objetos do mundo real, como, por exemplo, pessoas, empresas ou locais;
- ***Entity Linking (EL):*** Desambiguar entidades textuais para identificadores únicos numa Base de Conhecimento;
- ***Similarity:*** Comparar palavras, textos e documentos e averiguar quanto são semelhantes entre si;
- ***Text Classification:*** Atribuir categorias ou etiquetas a um documento inteiro ou a partes de um documento;
- ***Rule-based Matching:*** Localizar sequências de *tokens* com base nos seus textos e anotações linguísticas, semelhantes às expressões regulares;
- ***Training:*** Atualizar e melhorar as previsões de um modelo estatístico;
- ***Serialization:*** Guardar objetos em ficheiros ou *byte strings*;

Enquanto alguns destes recursos funcionam de forma independente, outros exigem o "carregamento" de **modelos estatísticos**, o que permite ao spaCy prever anotações linguísticas - por exemplo, se uma palavra é um verbo ou um substantivo. A ferramenta atualmente oferece modelos estatísticos a mais de **50 idiomas**, entre os quais Português, Inglês, Francês e Alemão.

Assim, o spaCy deve ser utilizado por iniciantes no processamento de linguagem natural, por pessoas que querem construir aplicações de produção *end-to-end*, treinar modelos a partir dos próprios dados e por pessoas que querem que a aplicação seja eficiente no CPU.

2.1 Token API

No processamento do texto, qualquer sequência de caracteres entre um único espaço em branco (' ') é caracterizada como sendo um *token*.

Cada *token* do texto possui diferentes atributos, sendo assim possível obter diferentes informações acerca do mesmo. Alguns desses atributos são "is_alpha", para saber se o *token* é constituído por caracteres alfabéticos, "is_digit", para descobrir se se trata de um dígito, "is_punct", para saber se se trata de um sinal de pontuação, entre muitos outros.

Por outro lado, tal como já foi mencionado, é possível obter informações como a *Lemmatization* do *token*, ou seja, a sua forma básica, através do atributo **lemma_**. No caso da funcionalidade *Part-of-speech tagging*, o atributo **pos_** permite identificar a classe gramatical da palavra e, na seguinte tabela, apresentam-se as classes comuns a qualquer um dos idiomas disponíveis:

POS	DESCRIPTION	EXAMPLES
ADJ	adjective	big, old, green, incomprehensible, first
ADP	adposition	in, to, during
ADV	adverb	very, tomorrow, down, where, there
AUX	auxiliary	is, has (done), will (do), should (do)
CONJ	conjunction	and, or, but
CCONJ	coordinating conjunction	and, or, but
DET	determiner	a, an, the
INTJ	interjection	psst, ouch, bravo, hello
NOUN	noun	girl, cat, tree, air, beauty
NUM	numeral	1, 2017, one, seventy-seven, IV, MMXIV
PART	particle	's, not,
PRON	pronoun	I, you, he, she, myself, themselves, somebody
PROPN	proper noun	Mary, John, London, NATO, HBO
PUNCT	punctuation	., (,), ?
SCONJ	subordinating conjunction	if, while, that
SYM	symbol	$, %, §, ©, +, −, ×, ÷, =, :), 😜
VERB	verb	run, runs, running, eat, ate, eating
X	other	sfpksdpsxmsa
SPACE	space	

Figura 1: *Universal Part-of-speech Tag.*

No caso da funcionalidade *Named Entity Recognition* (atributo **ent_type_**), os modelos suportam os seguintes tipos de entidades:

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including ”%“.
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	“first”, “second”, etc.
CARDINAL	Numerals that do not fall under another type.

Figura 2: *Named Entity Recognition Types.*

Estes são apenas alguns dos atributos disponíveis nos *tokens* usando o spaCy e que consideramos serem bastante importantes, mas existem muitos outros que podem ser consultados em [2].

O código fonte da ferramenta encontra-se em [3].

3 Exemplo Simples

Um exemplo simples de utilização da ferramenta spaCy, com apenas 10 linhas de código, consiste em atribuir a classe gramatical a cada palavra do texto (todos os *outputs* existentes encontram-se na Figura 1).

```
1  import sys
2  import spacy
3
4  nlp = spacy.load("pt_core_news_sm")
5
6  doc = open(sys.argv[1], "r").read()
7  ndoc = nlp(doc)
8
9  for w in ndoc:
10     print(w.text + '\t' + w.pos_)
```

Desta forma, começamos por importar a biblioteca que nos permite ler o ficheiro dado como *input* (**sys**) e a biblioteca da ferramenta spaCy. Em seguida, carregamos os componentes e dados do idioma do texto que queremos processar para um objeto Linguagem (**nlp**).

Depois disso, abrimos o ficheiro passado como *input* e processamo-lo através do objeto Linguagem obtido anteriormente. Desta forma, obtemos o texto *input* processado, dividido em *tokens* devidamente anotados.

Percorrendo cada *token* do texto processado, podemos recorrer a qualquer uma das funcionalidades da ferramenta, que neste exemplo foi o atributo "**.text**", que nos permite obter o texto original do *token* e o atributo "**.pos_**", que nos indica a classe gramatical da palavra.

A título de exemplo, para o ficheiro *input* da Figura 3, o resultado obtido através deste *script* apresenta-se na Figura 4.

```
[cam:~/Desktop/IPLN/tp2/src]$ cat ../inputs/animais.txt
Eu amo cães. E também adoro gatos.
```

Figura 3: *Input* Exemplo Simples.

```
[cam:~/Desktop/IPLN/tp2/src]$ python3 spacySimples ../inputs/animais.txt
Eu      PRON
amo     VERB
cães    NOUN
.       PUNCT
E       PUNCT
também  ADV
adoro   VERB
gatos   NOUN
.       PUNCT

        SPACE
```

Figura 4: *Output* Exemplo Simples.

4 Exemplo Intermédio

Optamos por desenvolver também um exemplo intermédio de utilização, que é uma pequena evolução do exemplo simples anteriormente apresentado. Para tal, surgiram várias possibilidades de melhorias que poderiam ser implementadas: devolver a lista de organizações presentes num texto, a lista de montantes monetários, a lista de países, a lista de verbos, a lista de sinais de pontuação, entre outros.

Decidimos implementar um *script* que devolve a lista de verbos de um texto, apresentando o respetivo verbo na forma infinitiva (atributo "**.lemma_**") e a relação sintática de dependência (atributo "**.dep_**"), explicando o significado da mesma (com recurso à funcionalidade "**spacy.explain**").

```
1  import sys
2  from tabulate import tabulate
3  import spacy
4
5  nlp = spacy.load("pt_core_news_sm")
6  file = sys.argv[1]
7  doc = open(file, "r").read()
8  ndoc = nlp(doc)
9
10 matrix = []
11 for w in ndoc:
12     if w.pos_ == "VERB":
13         line = []
14         line.append(w.text)
15         line.append(w.lemma_)
16         line.append(w.dep_)
17         line.append(spacy.explain(w.dep_))
18         matrix.append(line)
19
20 print(tabulate(matrix,tablefmt='github'))
```

Comparativamente ao exemplo anteriormente apresentado, as únicas alterações foram a verificação se se trata de um verbo e, em caso positivo, imprimir as informações desejadas. O resultado é apresentado em formato tabela, recorrendo para tal à biblioteca **tabulate**.

Para o ficheiro *input* presente na Figura 5, o resultado obtido encontra-se na Figura 6.

```
[cam:~/Desktop/IPLN/tp2/src]$ cat ../inputs/simples.txt
Hoje está um lindo dia. O inverno está a chegar. A Rita está muito cansada. O comboio está a chegar. O Rui está muito doente.
```

Figura 5: *Input* Exemplo Intermédio.

```
[cam:~/Desktop/IPLN/tp2/src]$ python3 spacyMedio ../inputs/simples.txt
|---------|--------|------|-----------|
| está    | estar  | aux  | auxiliary |
| chegar  | chegar | ROOT |           |
| está    | estar  | cop  | copula    |
| cansada | cansar | ROOT |           |
| chegar  | chegar | ROOT |           |
| está    | estar  | cop  | copula    |
```

Figura 6: *Output* Exemplo Intermédio.

5 Exemplo Complexo

Tendo em consideração as características da ferramenta spaCy, como exemplo complexo decidimos implementar um *script* que permite identificar as ações que uma ou várias pessoas adotam.

Desta forma, é possível fornecer como *input* **nomes de pessoas**, seja apenas um nome ou uma lista de nomes. No caso de não ser fornecido nenhum, são consideradas todas as pessoas presentes no texto. Pode ainda ser fornecido uma lista de **verbos** ou apenas um, e no caso de nenhum também serão consideradas todas as ações.

Para melhor compreensão deste enunciado, alguns exemplos de aplicação do mesmo são os seguintes:

- "Do que é que a **Rita gosta**?"
- "O que é que as pessoas do texto *input* ***gostam****?*"
- "O que é que o **Miguel comeu** e para onde **viajou**?"
- "Quais são todas as ações que o **João** esteve envolvido?"

5.1 Input

O programa tem as seguintes possibilidades de *input*:

- **Verbos:** filtrar determinados verbos;
- **Pessoas:** filtrar determinadas pessoas;
- **Tabela:** exibir uma tabela com as informações do *output* detalhadas;
- **Gráfico simples:** exibir um gráfico simples com dependências sintáticas;
- **Gráfico:** exibir um gráfico com dependências sintáticas;
- **Ficheiro:** ficheiro com o texto a analisar.

A função utilizada para fazermos o *parse* do *input* é a seguinte:

```
def getOptions(args=sys.argv[1:]):
    parser = argparse.ArgumentParser(description="Parses command.")
    parser.add_argument('-v', "--verbs", nargs='*',
            help='Insert one or more verbs.')
    parser.add_argument('-p', "--people", nargs='*',
            help='Insert one or more people names.')
    parser.add_argument("-t", "--table",
            help="If you want to show table.", action='store_true')
    parser.add_argument("-sg", "--simplegraph",
            help="If you want to serve simple graph.", action='store_true')
    parser.add_argument("-g", "--graph",
            help="If you want to serve graph.", action='store_true')
    parser.add_argument("-i", "--input",
            help="Your input file.")
    options = parser.parse_args(args)
    return options
```

5.2 Pattern

Para a recolha das frases que incluem pessoas a tomar diferentes ações, é necessário a elaboração de um *pattern* que identifique essas frases no texto.

Depois de analisar várias frases em Língua Portuguesa, consideramos que para uma frase ser identificada é necessário que comece com um nome próprio ou com uma conjunção (mas, e,...), visto que poderá referir-se a uma entidade anteriormente mencionada no texto.

Em seguida, terá que se encontrar um verbo, que representa a ação da entidade e, posteriormente, poderão aparecer 0, 1, 2 ou 3 preposições ou determinantes (para, de, da, o, a,...). Depois disso, poderá aparacer um nome, um nome próprio ou um advérbio. Por fim, a frase terá que terminar com um sinal de pontuação ou com uma conjunção.

```
pattern = [
    {'POS':{'IN':['PROPN', 'CCONJ']}},
    {'POS': 'VERB'},
    {'POS':{'IN':['ADP','DET']}, 'OP':'?'},
    {'POS':{'IN':['ADP','DET']}, 'OP':'?'},
    {'POS':{'IN':['ADP','DET']}, 'OP':'?'},
    {'POS':{'IN':['NOUN','PROPN','ADV']}},
    {'POS':{'IN':['PUNCT', 'CCONJ']}}
]
```

5.3 Match

Através da biblioteca *Matcher*, a função *matcher* encontra no texto todas as frases que obedecem ao *pattern* criado.

Depois disso, através do atributo *children*, *dep_*, *lemma_* e *pos_* é possível identificar quem é a pessoa responsável pelas sucessivas ações do texto, tendo em consideração os casos em que o nome próprio não aparece imediatamente antes do verbo (por exemplo, A Rita joga à bola **mas** joga também voleibol).

```
matcher = Matcher(nlp.vocab)
matcher.add("vn", None, *list_pattern)
matches = matcher(ndoc)

for match_id, start, end in matches:
    matched_span = ndoc[start:end]
    for w in matched_span:
        if w.pos_ == "VERB":
            for child in w.children:
                if child.dep_ == "nsubj" and child.pos_ == "PROPN":
                    try :
                        verbname[w.lemma_].append((child,matched_span))
                    except:
                        verbname[w.lemma_] = ([(child,matched_span)])
                elif child.dep_ == "cc" and child.pos_ == "CCONJ":
                    if tmp != None:
                        for n in tmp:
                            if n.pos_ == "PROPN":
                                try :
                                    verbname[w.lemma_].append((n,matched_span))
                                except:
                                    verbname[w.lemma_] = ([(n,matched_span)])
    tmp = matched_span
```

Posteriormente, é necessário filtrar nas frases identificadas as ações e as pessoas dadas como *input*. Tal como mencionado, no caso de não ser fornecida nenhuma, todas são consideradas.

5.4 Output

O *output* do *script* conterá sempre as frases identificadas.

```
print("FRASES:")
frases = []
for key,values in verbname.items():
    for v in values:
        print("-",v[1])
        frases.append(v[1])

if options.table:
    print("\n")
    matrix = []
    for frase in frases:
        for w in frase:
            if w.pos_ != "PUNCT":
                line = []
                line.append(w.text)
                line.append(w.lemma_)
                line.append(w.pos_)
                line.append(spacy.explain(w.pos_))
                line.append(w.dep_)
                line.append(spacy.explain(w.dep_))
                matrix.append(line)
        matrix.append("")

    headers=['Texto','Lemma','Classe Gramatical','Explicação C.G.','',
             'Explicação']
    print(tabulate(matrix,headers,tablefmt='fancy_grid'))
```

Opcionalmente, poderá apresentar uma tabela com informações mais detalhadas (caso seja ativada essa *flag* no *input*).

Para além disso, poderá também ser apresentado um autómato com as dependências dentro de cada frase, caso essas *flags* sejam ativadas.

```
if options.simplegraph:
    displacy.serve(ndoc, style="dep") ## Simple dependency visualizer

if options.graph:
    sentence_spans = list(ndoc.sents) ## More complex dependency visualizer
    displacy.serve(sentence_spans, style="dep")
```

5.5 Exemplos

Como exemplo de utilização desta ferramenta, construímos o seguinte texto *input*:

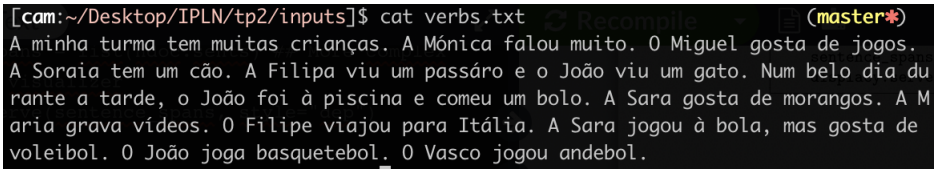

Figura 7: *Input* Exemplo Avançado.

5.5.1 Exemplo 1

Uma possível abordagem para o teste do *script* é procurar as ações da Sara. O resultado obtido é o seguinte:

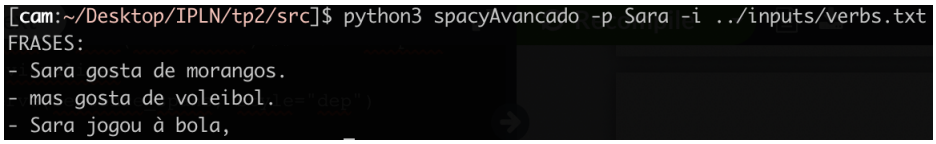

Figura 8: *Output* Exemplo Avançado 1.

Ativando a opção da tabela, o resultado obtido é o seguinte:

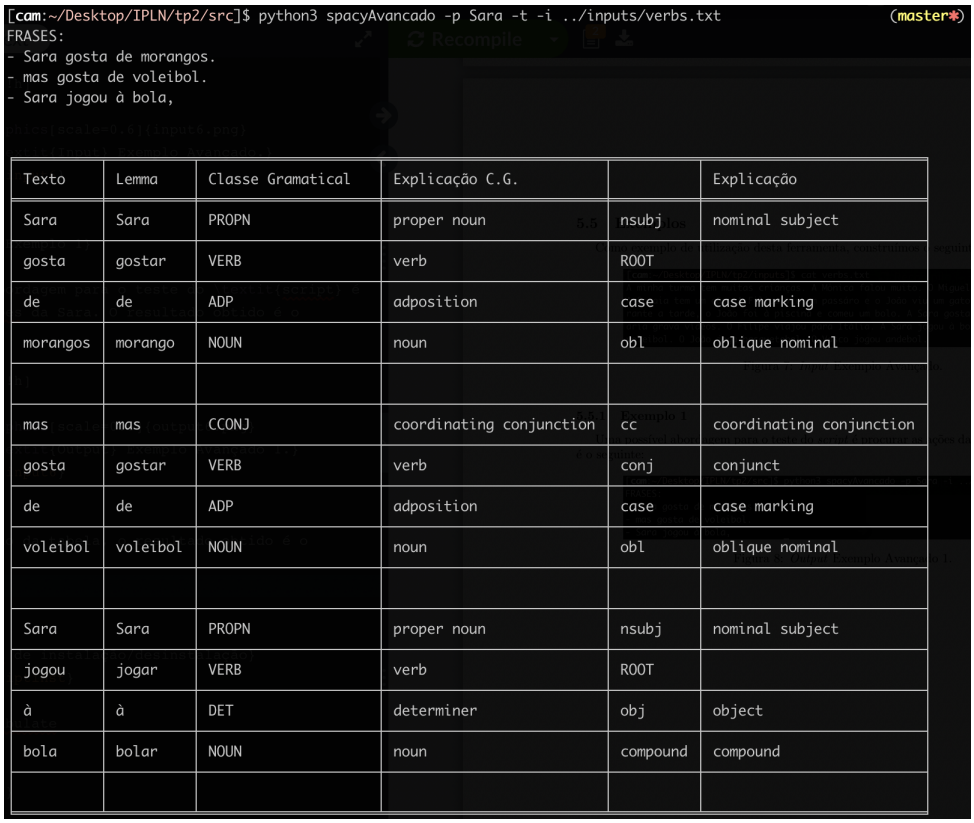

Figura 9: *Output* com tabela Exemplo Avançado 1.

Com a opção de grafos ativa, o resultado obtido inclui o *ip* onde é possível visualizar o grafo.

```
[cam:~/Desktop/IPLN/tp2/src]$ python3 spacyAvancado -p Sara -g -i ../inputs/verbs.txt                (master*)

Using the 'dep' visualizer
Serving on http://0.0.0.0:5000 ...
```

Figura 10: *Output* com grafo Exemplo Avançado 1.

Nesse endereço, encontram-se as frases com as respetivas dependências sintáticas ilustradas.

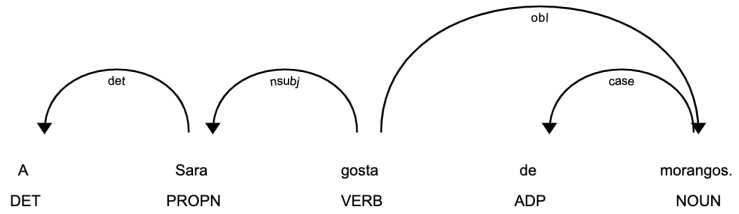

Figura 11: Frases Exemplo Avançado 1.

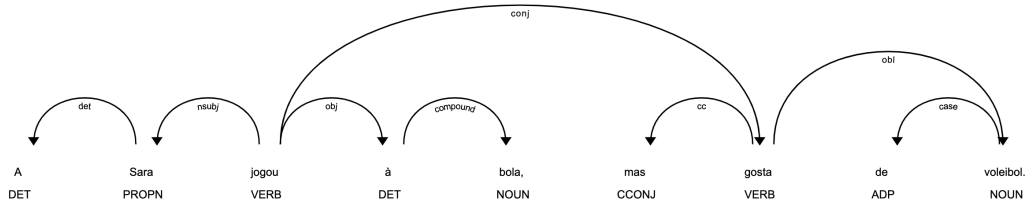

Figura 12: Frases Exemplo Avançado 1.

5.5.2 Exemplo 2

Para um *input* que identifique frases com ações de gostar e jogar o *output* é o seguinte:

```
[cam:~/Desktop/IPLN/tp2/src]$ python3 spacyAvancado -v gostar jogar -i ../inputs/verbs.txt           (master*)
FRASES:
- Miguel gosta de jogos.
- Sara gosta de morangos.
- mas gosta de voleibol.
- Sara jogou à bola,
- João joga basquetebol.
- Vasco jogou andebol.
```

Figura 13: *Output* Exemplo Avançado 2.

Muitas mais opções poderão ser testadas, bastando para isso "brincar" com os parâmetros do *input*.

6 Man

Para complementar o trabalho prático, desenvolvemos uma *man page* do exemplo avançado. Assim, através do comando ***man verbname*** é possível visualizar o manual desta ferramenta.

Figura 14: *Man page* Exemplo Avançado.

7 Script de instalação/desinstalação

Desenvolvemos ainda dois *scripts*, um para instalar e outro para desinstalar o exemplo avançado, as respetivas dependências necessárias e a *man page* criada.

Em ambos os *scripts*, existe a versão para Linux e a versão para macOS.

- Script de instalação:

```
if [ "$(uname)" == "Linux" ]; then
    python src/setup.py install
    pip install -U spacy
    pip install -U spacy-lookups-data
    python -m spacy download pt_core_news_sm
    install -g 0 -o 0 -m 0644 man/verbname.8 /usr/local/man/man8/
    gzip /usr/local/man/man8/verbname.8
elif [ "$(uname)" == "Darwin" ]; then
    python3 src/setup.py install
    pip install -U spacy
    pip install -U spacy-lookups-data
    python3 -m spacy download pt_core_news_sm
    install -g 0 -o 0 -m 0644 man/verbname.8 /usr/share/man/man8/
    gzip /usr/share/man/man8/verbname.8
fi
```

- Script de desinstalação:

```
if [ "$(uname)" == "Darwin" ]; then
    rm /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/
        site-packages/verbname-0.1-py3.7.egg
    rm /usr/share/man/man8/verbname.8.gz
    rm -rd /Library/Frameworks/Python.framework/Versions/3.7/bin/verbname
elif [ "$(expr substr $(uname -s) 1 5)" == "Linux" ]; then
    rm /usr/lib/python3.7/site-packages/verbname-0.1-py3.7.egg
    rm /usr/local/man/man8/verbname.8.gz
    rm -rd /usr/bin/verbname
fi
rm -rd verbname.egg-info
rm -rd build
rm -rd dist
```

8 Conclusão

O spaCy é uma ferramenta muito simples de utilizar, mas também muito poderosa. A ferramenta possui um *website* [1] com toda a informação necessária, muito bem catalogada e com vários exemplos práticos de utilização. Para além disso, dentro do *website* é também possível encontrar um curso de spaCy [4] com vários capítulos, onde são introduzidas todas as funcionalidades e especificidades da biblioteca. Desta forma, é também simples perceber as diferentes possibilidades de utilização desta ferramenta.

Durante as aulas práticas da unidade curricular Introdução ao Processamento de Linguagem Natural recorremos a outras bibliotecas similares ou simplesmente a expressões regulares para resolver alguns problemas que são também muito facilmente resolvidos com a biblioteca spaCy, como por exemplo identificar as entidades n'Os Maias. Esta biblioteca permite que o texto seja dividido em diferentes *tokens* e, a partir destes, pode-se recorrer a diferentes atributos da ferramenta que permitem saber várias informações, tanto sobre o *token*, como sobre o texto relacionado com o *token*.

Como enunciado para o desenvolvimento de *scripts* simples, intermédios e complexos da ferramenta as alternativas possíveis eram bastantes e optamos por um exemplo complexo que permite identificar as ações que uma ou várias pessoas adotam. Em vários momentos, ponderamos se seria o enunciado mais apelativo de implementar, visto que em textos como, por exemplo, os Maias, o *script* não se aplica devido ao facto do texto estar escrito de uma forma mais variada. A nossa maior dificuldade esteve relacionada com a escolha de um *pattern* que identificasse todas as frases necessárias, e só e apenas essas.

Devido ao facto da Língua Portuguesa ser uma linguagem complexa, é difícil identificar um padrão comum entre as frases que implicam ações e pessoas. Assim sendo, o *pattern* que elaboramos em certos textos e frases não identifica corretamente as expressões necessárias, mas com um pouco mais de conhecimento da Língua Portuguesa e, consequentemente, melhorando o *pattern*, o *script* pode ser aplicado a qualquer texto, obtendo sempre o resultado pretendido.

Como forma de complementar o trabalho, decidimos como trabalho extra desenvolver um *script* de instalação, um *script* de desinstalação e uma *man page* da nossa ferramenta.

Referências

[1] spaCy: Website
`https://spacy.io/`

[2] spaCy: Token Attributes
`https://spacy.io/api/token#attributes`

[3] spaCy: Source Code
`https://github.com/explosion/spaCy`

[4] spaCy: Tutorial
`https://course.spacy.io/`