

Introdução ao Processamento de Linguagens Naturais (4º ano de Curso)

**Trabalho Prático 2**

Relatório de Desenvolvimento

José André Martins Pereira  
(a82880@alunos.uminho.pt)

Ricardo André Gomes Petronilho  
(a81744@alunos.uminho.pt)

1 de Dezembro de 2019

## Resumo

Na unidade curricular de Introdução ao Processamento de Linguagens Naturais do Mestrado integrado em Engenharia Informática foi-nos proposta a análise e utilização de uma ferramenta intitulada **lxml** e utilizar a mesma no contexto de **NLP**.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Estrutura da ferramenta lxml</b>	<b>3</b>
<b>3</b>	<b>Conceção da Solução</b>	<b>4</b>
3.1	Objetivo do Programa . . . . .	4
3.2	Funcionalidades . . . . .	4
3.3	Implementação . . . . .	6
<b>4</b>	<b>Como instalar e executar o xmlstats</b>	<b>8</b>
4.1	Requisitos . . . . .	8
4.2	Como executar . . . . .	8
<b>5</b>	<b>Conclusão</b>	<b>12</b>

# Capítulo 1

## Introdução

Na unidade curricular de Introdução ao Processamento de Linguagens Naturais, do Mestrado integrado de Engenharia Informática, foi-nos proposta a análise da ferramenta **lxml**.

A ferramenta lxml XML toolkit trata-se de uma associação Python às bibliotecas da linguagem C, denominadas **libxml2** e **libxslt**. As grandes características desta ferramenta são a velocidade e a integridade dos recursos XML dessas bibliotecas com a simplicidade de uma API Python.

Assim, este projeto, tem como principais objetivos, o estudo e análise desta ferramenta, bem como a aplicação da mesma para resolver problemas no contexto NLP.

Deste modo, a ideia do grupo para aplicar a ferramenta **lxml**, foi desenvolver uma aplicação intitulada de **xmlstats**, capaz de executar expressões regulares sobre ficheiros **XML** de websites como **stackoverflow**, entre outros. A aplicação destas expressões regulares permite a execução de *queries* interessantes como:

- o top **N** utilizadores com mais **reputação**;
- procurar por palavras, frases, em **posts**, **comments**;
- verificar os links ou Ips mais usados nos posts ou comentários
- ordenar o nome dos utilizadores alfabeticamente
- ...

Como o utilizador do programa é que decide a expressão regular a aplicar, então existe um infinidade de *queries* possíveis a aplicar aos ficheiros **XML** de um website ou aplicação.

O programa também gera ficheiros **XML** com o output das *queries* para que estes possam ser utilizados por outras aplicações.

## Capítulo 2

# Estrutura da ferramenta lxml

A ferramenta **lxml** é composta por uma API `ElementTree`, bastante elegante, com todas as funcionalidades necessárias para ler, processar, e escrever ficheiros **XML**.

A ferramenta internamente divide-se em várias diretorias:

- `.github`: ficheiros de configuração do git
- `becnhmark`
- `doc`
- `samples`
- `src`
- `tools`

Das diretorias acima, a **doc** é responsável pela documentação do código, **samples** ficheiros de teste, **src** ficheiros do código fonte, `tools`, ferramentas utilizadas, etc.

No entanto, a diretoria mais importante será a **src**, onde internamente ainda é constituída por diretorias de bibliotecas, ficheiros de configuração de package entre outros.

## Capítulo 3

# Conceção da Solução

### 3.1 Objetivo do Programa

Tal como já foi referenciado anteriormente, o programa trata-se de um executor de expressões regulares sobre ficheiros **XML**, mais propriamente de websites do **StackExchange**, calculando as ocorrências das mesmas. Os ficheiros **XML** destes websites são normalmente por:

- **Badges.xml**
- **Comments.xml**
- **Posts.xml**
- **Tags.xml**
- **Users.xml**
- ...

### 3.2 Funcionalidades

Como já foi dito, a funcionalidade central da aplicação **xmlstats** é execução de uma expressão regular a vários dados **XML**. No entanto existem duas formas distintas de aplicar a mesma.

Uma das formas consiste na aplicação da expressão regular a cada entidade (p.e. **posts**) e calcular as ocorrências dessa expressão por entidade (p.e no post com Id = "91782" a expressão regular ocorreu 128 vezes).

Por exemplo: aplicando a expressão regular da figura 3.1, **regIP**, que captura endereços IP e procurando por um top 3 de ocorrências da mesma em **posts** - **Posts.xml**, a aplicação retorna os três **posts** onde foram publicados mais endereços IP's, sendo o output o verificado na tabela 3.1 mais abaixo.

```
|regIP = r"[\,\;\:](\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})([\,\;\:!\?]"
```

Figura 3.1: Expressão regular que captura endereços IP.

Id do Post	Número de ocorrências
Id="91782"	128
Id="164489"	80
Id="76535"	60

Tabela 3.1: Ocorrências por entidade (neste caso por **Post**)

A segunda funcionalidade, consiste na aplicação da expressão regular às entidades, no entanto, as ocorrências são guardadas em relação à expressão regular e não à entidade como na funcionalidade anterior. Isto é, o número de vezes que ocorre cada expressão regular em todos os ficheiros. Servindo como exemplo, aplicando a mesma expressão regular apresentada acima, o output desta funcionalidade será o top 3 dos IPs que mais ocorrem nos ficheiros analisados, tal como podemos observar na tabela 3.2.

Expressão regular	Número de ocorrências
'127.0.0.1'	357
'0.0.0.0'	152
'192.168.0.7'	76

Tabela 3.2: Ocorrências por expressão regular.

Por fim, e não menos importante, existe uma terceira funcionalidade que é na verdade uma extensão à funcionalidade que guarda as ocorrências por entidade. A única diferença em relação a esta é que não guarda as ocorrências da regex, mas sim o valor do **atributo** analisado.

Por exemplo, a procura pelo top N posts com maior score, apenas se consegue com esta funcionalidade, pois o valor do atributo, neste caso o **Score**, que é uma **TAG** do ficheiro **Posts.xml** é guardado e não as ocorrências da expressão regular utilizada, a tabela 3.3 apresenta um possível output desta funcionalidade.

Importante realçar que as funcionalidades referidas anteriormente da aplicação, após a determinação do número de ocorrências ou do maior atributo (no caso da última funcionalidade), o output das mesmas é guardado num ficheiro em formato **XML** utilizando a ferramenta **lxml** numa diretoria chamada **output**.

Id do Post	Score
146	56
4017	54
4188	48

Tabela 3.3: Top 3 de posts com maior score.

### 3.3 Implementação

Para implementar as funcionalidades descritas acima foram necessárias cinco funções denominadas:

- `from_xml(filename :str)`
- `to_xml(lst : list, agregation : str, single : str)`
- `top_reg_by_elem(fileNames : list, attributes : list, regex : str, N : int, keyAttr : int)`
- `top_reg_in_file(fileNames : list, attributes : list, regex : str, N : int)`
- `top_reg_sort_by_attribute(fileNames : list, attributes : list, regex : str, N : int, keyAttr : int, castInt: bool)`

A função **from\_xml** tem como objetivo ler um ficheiro **XML** e preencher a estrutura **ElementTree** da ferramenta **lxml** com o seu conteúdo no formato da mesma.

A função **to\_xml** tem o comportamento oposto à anterior, ou seja, escreve no ficheiro, com o formato **XML** utilizando a ferramenta **lxml**.

A função **top\_reg\_by\_elem** tem como objetivo aplicar uma expressão regular a uma **list** de ficheiros **XML**, ou seja, o argumento **filename**, onde respetivamente a cada ficheiro se atribui um atributo da list **attributes** que se trata do campo onde se vai buscar o texto (p.e. "Body" no ficheiro Posts.xml, "Text" no ficheiro Comments.xml, etc.). A este texto, presente em cada entidade (p.e. tag **post**) é aplicado a expressão regular dada como argumento **regex**. O número de ocorrências da mesma, em cada entidade, é guardado num dicionário onde a chave é o Id da entidade, e o valor, o respetivo número de ocorrências da **regex**.

Em relação à função **top\_reg\_in\_file** de forma semelhante, aplica a expressão regular **regex** a cada entidade, no entanto a grande diferença em relação à funcionalidade anterior é no momento de guardar no dicionário, o número de ocorrências (valor ou value), é associado à expressão regular (chave ou key) e não à entidade (p.e. **post**). O código desta função é apresentado e explicado na figura 3.2.

Por fim, e não menos importante, a função **top\_reg\_sort\_by\_attribute** é uma extensão da função **top\_reg\_by\_elem**, sendo que a principal diferença é no momento de guardar os valores no dicionário, onde na função **top\_reg\_by\_elem**, guarda-se as ocorrências da **regex** por entidade, enquanto que na **top\_reg\_sort\_by\_attribute**, tal como o nome sugere, guarda-se o valor do atributo da entidade. Desta forma, esta funcionalidade permite fazer ordenações pelo valor do atributo, como por exemplo o top N de posts com maior score.



```

def top_reg_in_file(filenamees : list, attributes : list, regex : str, N : int):

    dic = {}

    for idx, filename in enumerate(filenamees):
        root = from_xml(filename)
        for child in root.iter():
            text = str(child.get(attributes[idx]))
            regs = re.findall(regex, text) # applies regex to string
            for reg in regs:
                if reg not in dic: dic[reg] = 1
                else: dic[reg] += 1

    lst = sorted(dic.items(), key=lambda reg: reg[1], reverse=True)
    return lst[:N]

```

percorrer os ficheiros

iterar a estrutura Element

calcular ocorrências

ordenação do nr. de ocorrências

top N

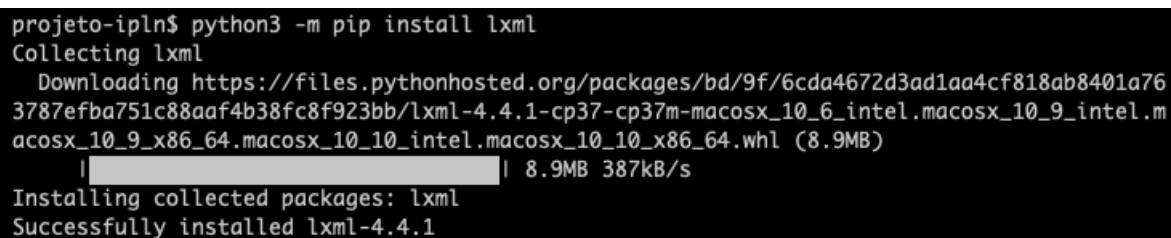
Figura 3.2: Função `top_reg_by_elem` .

## Capítulo 4

# Como instalar e executar o xmlstats

### 4.1 Requisitos

- Python 3
- lxml (ver figura 4.1)
- Package de scripts do xmlstats (enviados juntamente com este relatório)
- Ficheiros .xml (descarregar em stackexchange)



```
projeto-ipln$ python3 -m pip install lxml
Collecting lxml
  Downloading https://files.pythonhosted.org/packages/bd/9f/6cda4672d3ad1aa4cf818ab8401a76
3787efba751c88aaf4b38fc8f923bb/lxml-4.4.1-cp37-cp37m-macosx_10_6_intel.macosx_10_9_intel.m
acosx_10_9_x86_64.macosx_10_10_intel.macosx_10_10_x86_64.whl (8.9MB)
    | 8.9MB 387kB/s
Installing collected packages: lxml
Successfully installed lxml-4.4.1
```

Figura 4.1: Instalação do lxml .

### 4.2 Como executar

Para execução do programa e caso a equipa não tenha colocado a pasta **data** no (.zip) com os ficheiros de teste (.xml), deverá descarregar os mesmos, no entanto recomendamos ir ao seguinte website **StackExchange** e procurar por **StackOverflow**. Para facilitar o processo de execução, a equipa desenvolveu um **makefile** que se pode ver nas figuras 4.2, 4.3, 4.4 e 4.5, onde se pode substituir os argumentos necessários para execução do programa. O makefile ainda é composto para auxílio de algumas macros de execução para se entender como se executa o **xmlstats**. Os argumentos necessários são:

- -t (0 para executar funcionalidade por entidade, 1 para a funcionalidade por expressão regular, 2 para executar a expressão regular e ordenar pelo valor do atributo)
- -f (pode ser um único ficheiro ou uma lista, do tipo 'file1, file2, file3')
- -a (atributos das tags de cada ficheiro respetivamente 'atr1,atr2,atr3')

- -r expressão regular
- -c (apenas na opção -t = 2 necessita deste argumento -c, que indica se o atributo da tag a analisar é inteiro (-c 1), caso contrário (-c 0)
- -l limite para top N

```

FUNCTIONALLY_TYPE = 0 # 0 -> top_reg_by_elem(), 1 -> top_reg_in_file e 2 -> top_reg_sort_by_attr
INPUT_FILES = 'data/Posts.xml' # ficheiro ou ficheiros de input
ATTRIBUTES = 'Body' # atributo da tag do ficheiro a analisar
REGEX = 'java' # expressão regular
LIMIT = 10 # top 10

# EXECUÇÃO PERSONALIZADA PARA CADA TIPO DE FUNCIONALIDADE (PODE SE MUDAR OS ARGUMENTOS ACIMA):

run_f1: xmlstats.py
python3 xmlstats.py -t 0 -f ${INPUT_FILES} -a ${ATTRIBUTES} -r ${REGEX} -l ${LIMIT}

run_f2: xmlstats.py
python3 xmlstats.py -t 1 -f ${INPUT_FILES} -a ${ATTRIBUTES} -r ${REGEX} -l ${LIMIT}

```

Figura 4.2: Ficheiro makefile, funcionalidade 1 e 2.

```

SPECIAL_REGEX = "[0-9]+" # se o atributo for inteiro, a REGEX deverá ser [0-9]+, e [A-Za-z\ ]+ no caso de texto
CAST_INT = 1 # esta variável só é usada na funcionalidade <<-t 2>> e deve estar a 1 quando o <<ATTRIBUTES>> usado
SPECIAL_ATTRIBUTES = "Score" # atributo é a métrica de ordenação nesta funcionalidade, recomendamos a utilização dos atributos "

run_f3: xmlstats.py
python3 xmlstats.py -t 2 -f ${INPUT_FILES} -a ${SPECIAL_ATTRIBUTES} -r ${SPECIAL_REGEX} -c ${CAST_INT} -l ${LIMIT}

```

Figura 4.3: Ficheiro makefile, funcionalidade 3.

```

# IMPORTANTE:
# podem ser enviados mais do que um ficheiro de input, no entanto é preciso lembrar
# que nesse caso, também devem ser enviados a mesma quantidade de <<ATTRIBUTES>>, porque
# eles são respetivos a cada ficheiro, isto é se:
# INPUT_FILES = 'data/Posts.xml,data/Comments.xml'
# ATTRIBUTES = 'Body,Text'
# o campo Body está presente no ficheiro Posts.xml, o Text no ficheiro Comments.xml

```

Figura 4.4: Ficheiro makefile, notas importantes.

A figura 4.6, apresenta-se a execução deste programa ao ficheiro Posts.xml, onde o atributo analisado é o Body, presente nas tags deste ficheiro, a expressão regular é a palavra **java**, e o output é apresentado em formato tabelar, produzido por uma função denominada **pretty\_print**, onde, neste exemplo, o valor da esquerda representa o id do Post e o da direita o número de ocorrências da expressão regular.

De seguida, apresentam-se as restantes funcionalidades em execução e a respetiva explicação na legenda das imagens (figuras 4.6, 4.7 e 4.8).

```
# MACROS DE EXEMPLOS DE EXECUÇÃO:

f1: xmlstats.py data/Posts.xml
python3 xmlstats.py -t 0 -f 'data/Posts.xml' -a 'Body' -r 'java' -c 1 -l 15

f2: xmlstats.py data/Posts.xml
python3 xmlstats.py -t 1 -f 'data/Posts.xml' -a 'Body' -r 'java' -l 10

f3: xmlstats.py data/Posts.xml
python3 xmlstats.py -t 2 -f 'data/Posts.xml' -a 'Score' -r '[0-9]+' -c 1 -l 15

# EXEMPLO DE EXECUÇÃO COM VÁRIOS FICHEIROS E RESPECTIVOS ATRIBUTOS:

f4: xmlstats.py data/Posts.xml data/Comments.xml
python3 xmlstats.py -t 0 -f 'data/Posts.xml,data/Comments.xml' -a 'Body,Text' -r 'java' -l 15
```

Figura 4.5: Ficheiro makefile, macros de execução.

```
tp2$ python3 xmlstats.py -t 0 -f 'data/Posts.xml' -a 'Body' -r 'java' -c 1 -l 15
|-----|
| Id da entidade | Número de ocorrências da regex |
| 870            | 63                               |
| 1841           | 21                               |
| 551            | 12                               |
| 1727           | 12                               |
| 1829           | 10                               |
| 334            | 9                                |
| 1904           | 9                                |
| 50             | 8                                |
| 1907           | 8                                |
| 4182           | 8                                |
| 929            | 7                                |
| 104            | 6                                |
| 558            | 6                                |
| 1672           | 6                                |
| 556            | 5                                |
|-----|
```

Figura 4.6: Executar a expressão regular "java" ao texto do atributo **Body**, presente na tag post do ficheiro Posts.xml, com a funcionalidade -t 0, ou seja, calcula e ordena as ocorrências por entidade. A tabela é escrita em formato XML na diretoria output.

```
tp2$ python3 xmlstats.py -t 1 -f 'data/Posts.xml' -a 'Body' -r 'java' -c 1 -l 15
|-----|
| Expressão regular | Número de ocorrências da regex |
| java             | 411                             |
|-----|
```

Figura 4.7: Executar a expressão regular "java" ao texto do atributo **Body**, presente na tag post do ficheiro Posts.xml, com a funcionalidade -t 1, ou seja, calcula e ordena as ocorrências por expressão regular. A tabela é escrita em formato XML na diretoria output.

```
tp2$ python3 xmlstats.py -t 2 -f 'data/Posts.xml' -a 'Score' -r '[0-9]+' -c 1 -l 15
```

Id da entidade	Valor do atributo
146	56
4017	54
4188	48
2209	46
1395	43
2614	43
2868	42
3188	42
3299	40
1059	39
2493	38
475	37
2478	37
46	36
79	36

Figura 4.8: Executar a expressão regular "[0-9]+" ao texto presente no atributo **Score**, presente na tag post do ficheiro Posts.xml, com a funcionalidade -t 2, ou seja, calcula e ordena pelo atributo **Score**. A tabela é escrita em formato XML na diretoria output.

## Capítulo 5

# Conclusão

Em suma, os objetivos inicialmente propostos foram cumpridos, no entanto a equipa entende que deveria ter estudado melhor a estrutura/componentes internos da ferramenta **lxml**.

No entanto, com uma análise da documentação presente no website da ferramenta, conseguiu-se perceber a excelente **ElementTree API**, utilizada neste programa, e as diversas funcionalidades que a mesma permite. Percebeu-se também a importância de estudar ferramentas desconhecidas, para reutilização de funcionalidades já existentes e bem testadas.

Por fim, a aplicação inicialmente proposta, **xmlstats**, que aplica expressões regulares a ficheiros **XML** foi concretizada com sucesso, permitindo executar diversas *queries* a partir de expressões regular a toda uma estrutura de um website como o **StackOverflow**.