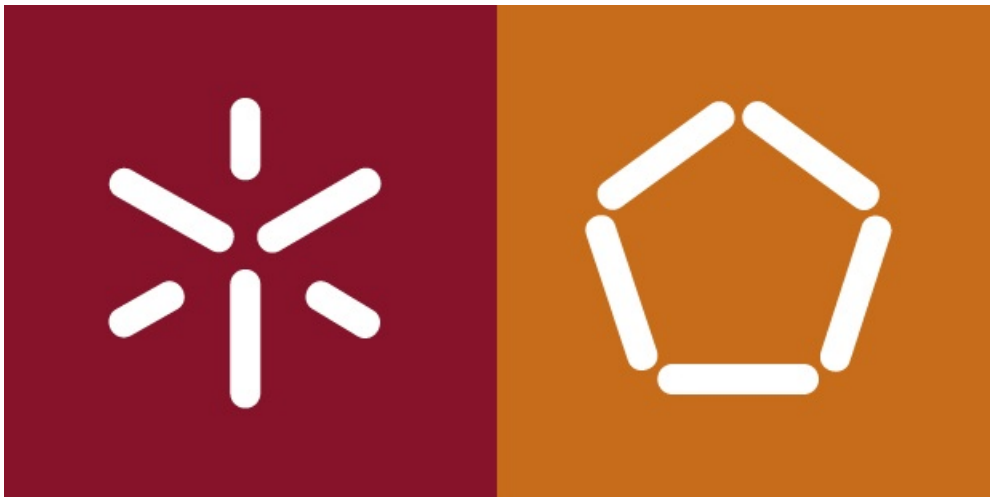
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Processamento de Linguagens - TP1

Grupo 29

Trabalho realizado por:	*Número*
Adriana Martins Gonçalves	A75119
Carlos Filipe Coelho Ferreira	A89542
Joel Salgueiro Martins	A89575

Conteúdo

1 *Introdução*

No âmbito da unidade curricular de Processamento de Linguagens foi-nos proposto a elaboração de um trabalho prático cuja finalidade é aumentar a capacidade de escrever expressões regulares (ER) para descrição de padrões de frases dentro de textos e, a partir dessas mesmas ER obter a informação que nos fosse mais útil.

Dos cinco enunciados possíveis, o que nos foi atribuído foi o enunciado 5, referente a **Machine Learning : datasets de treino**.

Ao longo do relatório iremos explicar as decisões e abordagens que tomamos ao longo do desenvolvimento do projeto, de forma a obter o resultado final desejado.

2 *Machine Learning : datasets de treino*

2.1 Enunciado

Hoje em dia, a área de *Machine Learning* está na moda e as suas metodologias e tecnologias são usadas em muitas áreas.

A maior parte dos algoritmos de *Machine Learning* têm de ser treinados com um dataset especialmente anotado à mão e depois testados sobre outro dataset anotado para ver se o que a máquina descobre é o mesmo que um ser humano faria à mão.

Estes datasets anotados são uma fonte preciosa de informação, para treino dos algoritmos, mas também podem ser usado para outros fins. Neste problema, terás de programar a extração de vários elementos informativos de um destes datasets. Nomeadamente do dataset de treino disponibilizado pela Google para treino docente TensorFlow: **'train.txt'**.

Apresenta-se a seguir um excerto deste dataset:

```
B-GENRE        science
I-GENRE        fiction
I-GENRE        films
O        directed
O        by
B-DIRECTOR        steven
I-DIRECTOR        spielberg
...
O        move
O        that
B-ACTOR        frank
I-ACTOR        sinatra
O        was
O        in
...
```

Cada linha pode começar por:

B-categoria Begin: marca o início de uma categoria;
I-categoria In: marca a continuacão de uma categoria;
O Other: marca algo que não se quer anotar/recolher;

Do excerto acima teríamos as seguintes categorias com os respetivos elementos:

```
GENRE:        science fiction films
DIRECTOR:        steven spielberg
ACTOR:        frank sinatra
```

Ao longo do dataset cada categoria vai estando associada a mais elementos. Neste projeto, pretende-se que cries um website muito simples com a seguinte estrutura:

- Uma página principal onde aparecem todas as categorias existentes no dataset, à frente de cada categoria deverá aparecer um contador indicando quantos elementos estão classificados como sendo dessa categoria;

- Clicando numa categoria, devemos saltar para a página dessa categoria onde devem aparecer listados os elementos dessa categoria (para cada elemento podes indicar em que linha do dataset ele foi capturado); Nestas páginas deverá haver um link para retornar à página principal.

2.2 Descrição do problema

Tal como referido no enunciado, todo este trabalho se baseia em programar a extração de vários elementos informativos de um dataset e assim produzir o resultado final com toda a informação que consideramos útil. No nosso caso, temos de pegar num ficheiro *.txt* que contém inúmeras **categorias**, cada categoria com vários **elementos** associados. Nesse ficheiro iremos, por exemplo, fazer procuras de atores, músicas, géneros etc. Toda essa informação será organizada e armazenada em ficheiros *HTML*.

No primeiro ponto do enunciado é necessário percorrer o ficheiro *train.txt* de forma a perceber quais as categorias que existem nesse ficheiro. Para isso, sabemos que um **B** no principio de uma linha marca o início de uma categoria, um **I** marca a continuação e um **O** representa algo que não é do nosso interesse anotar. Assim, depois de percorrido o ficheiro, é necessário escrever as diferentes categorias num ficheiro *HTML* e contar o número de elementos de cada categoria contém.

No segundo ponto do enunciado, será necessário correr o ficheiro *train.txt* e, através da key da categoria e desse ficheiro, criar a respetiva página da categoria desejada com todos os dados relativos à key dessa categoria, bem como a linha onde essa ocorrência foi detetada. Dentro de cada página da categoria, será necessário ter um link que nos redirecione para a **página principal/inicial**.

2.3 Estratégia de implementação

De forma a resolver todos os objetivos que nos propuseram, foi necessário tomar algumas considerações gerais. Assim, depois de analisar-mos o ficheiro *train.txt* percebemos que cada categoria inicia sempre que nos é apresentado um : **B**. Desta maneira decidimos que sempre que encontrarmos essa key encontramos uma nova *categoria*.

Outra consideração geral a tomar é quando estamos dentro de uma *categoria* perceber quando esta continua, neste caso, quando depois de esta ser inicializada com um **B**, aparecer na linha seguinte um **I**. Importa ainda saber o nome de categoria a que o ficheiro está a fazer referência, para que seja possível organizar a informação corretamente de acordo com o nome da categoria. O nome da categoria aparece logo depois do **B** e **I**.

Estas considerações serão aplicadas ao longo de toda a implementação do projeto, assim sempre que referirmos que encontramos uma nova categoria, sabemos que encontramos as key acima mencionadas.

Começamos por explicar a função ***insereElemSemRepetidos***, que como o nome indica, será responsável inserir um dado elemento não repetido dentro da **categoria** que pertence. Esta função recebe como parâmetros um *actualName* (nome), um *actualPosition* (posição ou linha do dataset em que se encontra) e um *actualCategoria* (categoria a que pertence este elemento). Depois de receber estes parâmetros vai verificar se o elemento já existe ou não.

Após procurar a ocorrência de um elemento na lista referente à categoria a partir de um ciclo while, somente senão o encontrar é que irá inserir este elemento na lista e aumentar o contador de elementos dessa categoria.

```
def insereElemSemRepetidos(actualName, actualPosition, actualCategoria):
    jaExiste = False
    size = len(dicionario[actualCategoria])
    i = 0
    while (i < size and (not jaExiste)):
        (elem, _) = dicionario[actualCategoria][i]
        if (elem == actualName):
            jaExiste = True
        i += 1
    if (not jaExiste):
        dicionario[actualCategoria].append((actualName, actualPosition))
        dicAux[actualCategoria] += 1
```

Figura 1: Código de inserção dos *elementos sem repetidos*

A função ***insereElemComRepetidos*** será responsável pela inserção dos elementos numa categoria, não tendo em atenção a existência previa desse elemento. Esta função recebe os mesmos parâmetros que a função ***insereElemSemRepetidos*** e simplesmente irá fazer **append** desse elemento e a linha onde ocorre ao final da lista dessa categoria. No final, incrementa ao contador de elementos.

```
def insereElemComRepetidos(actualName, actualPosition, actualCategoria):
    dicionario[actualCategoria].append((actualName, actualPosition))
    dicAux[actualCategoria] += 1
```

Figura 2: Código de inserção dos *elementos com repetidos*

A função ***parser*** vai receber como parâmetro a função ***funcInsere***, que neste caso, pode ser uma das 2 funções vistas anteriormente e tem como objetivo decidir se pretendemos nos nossos resultados ter elementos repetidos ou não.

O ***parser*** tem ainda outro elemento fundamental, a expressão regular (ER) : **r'([BI])-(\w+)[\t]+(\w+)'**.

De seguida, pretende-se em todas as linhas do ficheiro procurar a informação necessária, e para isso, executa-se **res = expressao.search(l)**, isto é, vai pegar na expressão regular e em cada linha do ficheiro procurar essa ER. O resultado dessa procura vai ser armazenado na variável 'res'.

Se tiver encontrado a ER **if res** irá então proceder ao tratamento desses dados, que pode dividir-se em 2 opções : no caso de ser o início de um novo elemento **res.group(1) in B**, terá de adicionar ao dicionário a informação relativa ao elemento que encontrou anteriormente, e começar a formar um novo elemento de diferente nome, categoria e posição. A outra opção é se for a continuação de um elemento e neste caso basta completar ao nome do elemento atual. Devido a forma como se adiciona elementos somente depois de se encontrar um novo, no final do ficheiro, é necessário retirar o elemento vazio assim como inserir o último elemento que se formou.

Para guardar toda a informação é importante dizer que foi usada as seguintes estruturas dicionário = { nomeCategoria : [(Elemento,Posicao)]} e dicAux ={nomeCategoria : numero-Elementos}

```
def parser(funcInsere):
    expressao = re.compile(r'([BI])-(\w+)[\t ]+(\w+)')
    actualCategoria = ""
    actualName = ""
    actualPosition = 0
    for i, l in enumerate(ficheiro):
        res = expressao.search(l)
        if res:
            if res.group(1) in 'BI':
                if res.group(1) == 'B':
                    if not dicionario.__contains__(actualCategoria):
                        dicionario[actualCategoria] = []
                        dicAux[actualCategoria] = 0
                    # Insere com repetidos ou sem repetidos
                    funcInsere(actualName, actualPosition, actualCategoria)
                    # Atualiza o elemento
                    actualCategoria = res.group(2)
                    actualName = res.group(3)
                    actualPosition = i
                else:
                    actualName = actualName + " " + res.group(3)

    funcInsere(actualName, actualPosition, actualCategoria)
    del dicionario[""]
```

Figura 3: Código do *parser*

2.3.1 1)

Como referido na secção anterior, este primeiro ponto baseia-se em criar uma página *HTML* com todas as categorias existentes no *dataset* e o número de elementos classificados como sendo dessa categoria.

Assim, a função ***pagPrincipal*** vai receber o ficheiro no qual irá escrever iterativamente o código *HTML*. Percorrendo o dicionário de dados pelas keys **k** (referente às categorias existentes no ficheiro), calculando para cada uma a quantidade de elementos (sublistas) que constituem o *value* **v**.

```
def pagPrincipal(ficheiro):
    ficheiro.write(f'''<!DOCTYPE html>
    <html>
        <head>
            <title>Pagina Principal</title>
            <meta charset=UTF-8/>
        </head>''')
    ficheiro.write('<body>\n')
    ficheiro.write(f'\t<h1>Categorias</h1>\n')
    ficheiro.write(f'\t<ul>\n')
    for k, v in dicionario.items():
        ficheiro.write(f'\t\t<li><a href="pag{k}.html">{k}</a> Contem: {len(v)} elementos!</li>\n')
        print("Categoria: ", k, ",Contem: ", len(v))
    ficheiro.write(f'\t</ul>\n')
    ficheiro.write(f'</body>')
```

Figura 4: Código da criação da *página inicial*

2.3.2 2)

Para este segundo ponto, foi necessário listar os elementos de uma categoria, a linha do *dataset* onde ocorriam e ainda, a possibilidade de voltar à pagina inicial.

Assim, a função ***pagSecundaria*** vai receber uma *key* (referente à categoria seleccionada) e o ficheiro no qual irá escrever iterativamente o código *HTML*. Através de : **ficheiro.write(f'\t<h1><a href="pagInicial.html» Listagem da key: {key}</h1>\n')**, criamos um titulo nomeado pela *key* da categoria, com o link que permite o redireccionamento para a pagina inicial se assim quisermos. Posteriormente, percorremos a lista de elementos constituintes do *value* do dicionário apontado pela respectiva *key* recebida inicialmente, e criamos com o seu conteúdo, os constituintes da *unorder list* do código *HTML*.

```
def pagSecundaria(key, ficheiro):
    ficheiro.write(f'''<!DOCTYPE html>
     <html>
         <head>
             <title>Pagina do elemento {key} </title>
             <meta charset=UTF-8/>
         </head>''')
    ficheiro.write(f'<body>\n')
    ficheiro.write(f'\t<h1><a href="pagInicial.html">Listagem da key: {key}</a></h1>\n')
    ficheiro.write(f'\t<ul>\n')
    for v in dicionario[key]:
        ficheiro.write(f'\t\t<li>{v}</li>\n')
    ficheiro.write(f'\t</ul>\n')
    ficheiro.write(f'</body>')
```

Figura 5: Código da criação da *página secundária*

Para a obtenção dos resultados temos duas hipóteses: ou queremos que sejam apresentados elementos com repetidos, e nesse caso, basta inserir *'S ou s'* como resposta, ou queremos sem repetidos, e nesse caso escrevemos *'n ou N'* na linha de comandos. Qualquer outra inserção para além destas duas dará como resposta 'Opção inválida!'.

```
print('Pretende ter em consideracao os reptidos? (S/N)')
opcao = input('Escolha: ')
if opcao in 'Ss':
    parser(insereElemComRepetidos)
elif opcao in 'nN':
    parser(insereElemSemRepetidos)
else:
    print('Opcao invalida!')
```

Figura 6: Código referente à impressão das opções de *inserção de elementos com ou sem repetidos*

Para concluir, esta última parte do código diz respeito à abertura das páginas Inicial e Secundária de cada categoria.

```
pagIni = open('./Paginas/pagInicial.html', 'w')
pagPrincipal(pagIni)

for k in dicionario.keys():
    pagSec = open(f'./Paginas/pag{k}.html', 'w')
    pagSecundaria(k, pagSec)
```

Figura 7: Código referente à *abertura* da página principal e secundárias

2.4 Resultados obtidos

Depois de explicado o processo de desenvolvimento de todo o projeto, nesta secção iremos apresentar os resultados obtidos.

2.4.1 1)

Neste primeiro ponto foi pedido que criássemos uma página principal onde aparecem todas as categorias presentes no dataset e respetivo número de elementos classificados como sendo dessa mesma categoria.

Como podemos verificar pela figura dos resultados abaixo apresentada, esta pagina terá vários ficheiros *HTML* como referencia. Teremos assim, um ficheiro para cada categoria :**actor** com 1338 elementos, **year** com 204 elementos, **tittle** 1615 elementos, **genre** com 313 elementos, **director** com 948 elementos, **song** com 173 elementos, **plot** com 1219 elementos, **review** com 102 elementos, **character** com 257 elementos, **rating** com 48 elementos, **rating average** com 167 elementos e **trailer** com 19 elementos.

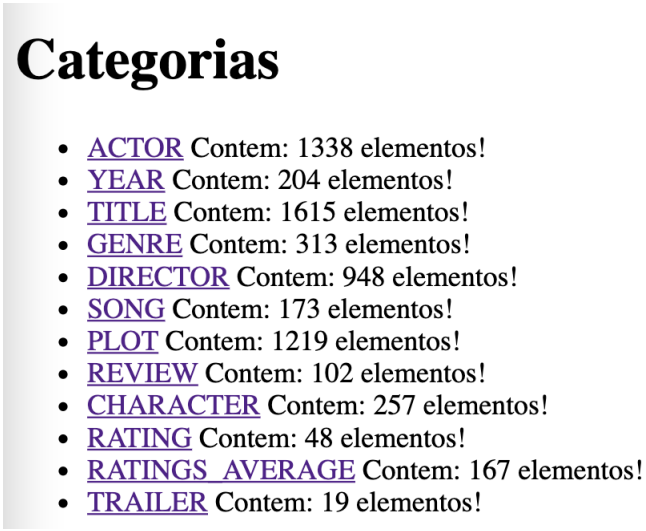


Figura 8: Página inicial com todas as categorias e respetivo contador de elementos.

2.4.2 2)

Para a segunda parte, era pedido que ao carregar numa qualquer categoria da **página principal** ilustrada acima, fosse possível ser direcionado para a página dessa mesma categoria, onde é suposto aparecerem os elementos da categoria e a respetiva linha do dataset onde foi capturado.

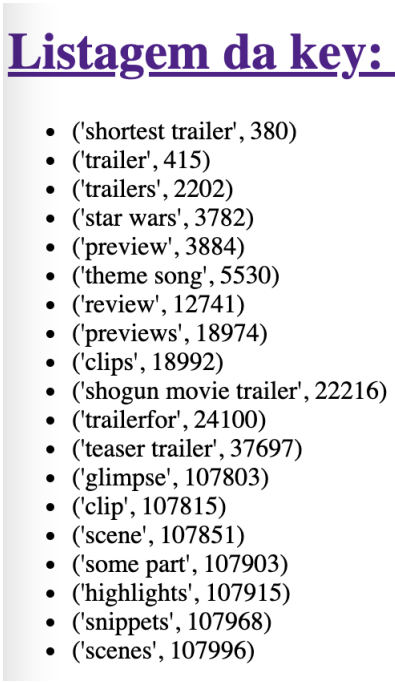


Figura 9: Excerto do ficheiro *pagTRAILER.html*

Cada um dos pontos da **página principal** contém uma hiperligação para outro ficheiro *HTML*, sendo que cada ficheiro contém a listagem de todos os elementos da key relativa à categoria em questão.

Por exemplo, ao selecionarmos a categoria **TRAILER** (selecionar a página *pagTRAILER.html*) será direcionado para uma página com a lista de todos os trailers que, segundo o ficheiro, contenham a key relativa a essa categoria, seguido do número da linha do dataset em que foi capturado.

Em todas as páginas, caso se pretenda, basta carregar no topo da mesma (onde aparece o nome da categoria em que se encontra) para ser redirecionados novamente para a **página principal** e aí selecionar uma nova categoria à escolha.

3 *Conclusão*

Durante a elaboração desta primeira fase do trabalho prático da unidade curricular de Processamento de Linguagens, foram vários os desafios com que nos deparamos na tentativa de obter um resultado final o mais fidedigno e consistente possível.

Para obter estes resultados, tentamos seguir os ensinamentos lecionados nas aulas práticas tendo estes permitido uma maior consolidação de conhecimentos no que diz respeito à escrita de Expressões Regulares (ER).

Através da realização do enunciado que nos foi atribuido, **Machine Learning : datasets de treino**, ficou demonstrado o nosso conhecimento em relação aos temas abordados.

Assim, concluímos que nosso desempenho ao longo deste trabalho foi positivo, visto que conseguimos encontrar forma de ultrapassar os desafios apresentados durante o desenvolvimento do mesmo. Este trabalho permitiu ainda aumentar a motivação para a próxima fase do projeto, pois teremos a oportunidade de aperfeiçoar e aprofundar os conhecimentos obtidos ao longo desta unidade curricular.