Processamento de Linguagens (3º ano de Engenharia Informática)

Trabalho Prático 1

Machine Learning: datasets de treino

Daniela Fernandes (a73768)
Marco Sampaio (a85508)
Mário Real (a72620)

12 de abril de 2021

Resumo

Como primeiro trabalho da Unidade Curricular de Processamento de Linguagens, do 3º ano da Licenciatura em Engenharia Informática, foi-nos proposto a realização de um *parser* a um ficheiro *.txt* para a criação de um *website* em *HTML*, baseando-se na estrutura de um *dataset* de treino disponibilizado pela Google para treino docente *TensorFlow*, *datasets* esses cada vez mais utilizados como fontes de informação utilizados em treinos dos algoritmos de *Machine Learning*. Neste relatório iremos explicar todas as decisões tomadas para o desenvolvimento de *Processadores de Linguagens Regulares* e *filtros de texto (FT)* com utilização do módulo *'er'* do *Python* e posterior construção do nosso *website*, as especificações de como a nossa solução foi construída, as dificuldades que tivemos e como as ultrapassamos.

Conteúdo

Capítulo 1

Introdução

Supervisor: Pedro Rangel Henriques, José Carlos Ramalho e Pedro Paiva Moura
Titulo: Machine Learning: datasets de treino
Área: Processamento de Linguagens

1.1 Enquadramento e Contexto

Atualmente as tecnologias e metodologias de *Machine Learning* são um recurso cada vez mais utilizado, desenvolvido e inserido nas mais diversas necessidades da nossa sociedade. Este conjunto de tecnologias tem como recurso algoritmos que precisam de ser treinados com *datasets* especificamente construídos e anotados como fonte de informação preciosa às suas necessidades e objectivos.

Posto este enquadramento, o nosso projecto consiste no desenvolvimento de uma solução de software capaz de realizar a extração e a organização de diversos elementos informativos presentes em um *dataset* de treino disponibilizado pela Google para treino docente *TensorFlow*, para posteriormente serem apresentados de forma apelativa e de eficaz acesso através de um *website* construído.

1.2 Problema e Objectivo

O primeiro problema a solucionar passa por limpar e filtrar todas as categorias e seus respetivos elementos nelas contidas, diferenciando-as dos outros diversos dados que não interessam anotar e recolher. Com o fim de obter um dicionário com as desejadas *Tags* encontradas no *Dataset* e seus respectivos elementos.

Após isso, o problema e objectivo seguinte será inserir toda essa informação em ficheiros *HTML*, organizados pelas *Tags* identificadas, sendo possível ter cada categoria separada por diferentes links devidamente interligados, e com opção de retrocesso para uma navegação dinâmica pelo nosso *website*.

Este documento terá como objectivo explicar todas as decisões tomadas para o desenvolvimento de *Processadores de Linguagens Regulares* e *filtros de texto (FT)* e posterior construção do *website*, tais como as especificações detalhadas de como a nossa solução de software foi construída.

1.3 Decisões, Contributos e Resultados

Devido à existência de um *dataset* tão extenso a ser analisado e tratado foi necessário recorrer a bibliotecas e módulos da linguagem *Python* como o *'ER'*. Para além disso devido ao tamanho e relacionamento dos dados foi preciso tomar a decisão de utilizarmos estruturas de dados como dicionários e colecções a partir das ferramentas disponíveis no *Python* para um correcto armazenamento e organização dos dados. Posteriormente foi também decidido aplicar um estudo e desenvolvimento de mecanismos em *CSS* para aprimorar o *website* desenvolvido.

1.4 Estrutura do Relatório

Este documento será estruturado por cinco capítulos centrais. No capítulo 1 foi apresentado uma introdução ao projecto desenvolvido, expondo o enquadramento e contexto em que está inserido, quais os problemas e objetivos a solucionar e apresentar, e pequenos pontos sobre principais decisões tomadas para a construção da solução e seus resultados.

Em seguida, no capítulo 2 é exposta a análise e especificação do nosso projecto onde será descrito informalmente o problema e uma especificação dos requisitos indispensáveis para a construção da solução.

No capítulo 3 é exibida toda a concepção e desenho da resolução do nosso problema, onde será possível analisar detalhadamente como está construída a nossa solução final de software.

Subsequentemente, no capítulo 4 pode ser explorada a codificação e testes realizados à nossa solução, podendo ser visto vários resultados obtidos.

Por fim, no capítulo 5 concluí-se este documento com uma síntese e análise crítica do projecto realizado, aos resultados obtidos, e quais os planos futuros para a aplicação.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Sendo fornecido o *dataset TensorFlow: 'train.txt'*, é requisitado a filtragem e recolha de todas as suas categorias e diferentes elementos nelas contidas e identificadas por diversas e especificas *tags*, com objectivo de ser construído um *website* com uma página principal onde será exposto todas as categorias existentes e seu número de elementos. Para além disso cada categoria terá de ter a sua devida página com a listagem dos seus distintos elementos, devidamente interligada à página principal do *website*.

2.2 Especificação de Requisitos

Na especificação de requisitos foi necessário ter em conta duas fases principais, tratamento de dados, e produção do *website*

2.2.1 Dados

Para o tratamento e processamento dos dados recebidos temos os seguintes requisitos como especificação dos padrões de categoria e elementos nas frases alvo do texto-fonte, através da utilização de expressões regulares; Construção de ações semânticas para realizar ações após reconhecimento de cada um desses padrões identificado; Desenho e implementação de estruturas de dados para armazenamento e organização temporária da informação recolhida; E desenvolvimento de um *Processador de Linguagens Regulares* ou *Filtro de Texto* com utilização do módulo *'er'* da linguagem *Python* que realize toda a filtragem e transformação textual com base no conceito de regras de produção *"Condição-Acção"*.

2.2.2 Pedidos e Relações

O nosso programa terá de ter a capacidade de conseguir apresentar através da criação de ficheiros *HTML* todas as categorias existentes sem repetições; calcular e apresentar o número correcto de elementos distintos contidos em cada categoria; dentro de cada categoria uma página de *website*, devidamente interligada, onde será apresentada uma listagem dos seus elementos distintos; e por fim, todos os *links* de *website* de cada categoria terão como requisito a opção de retrocesso para a página principal.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

Para que fosse possível armazenar temporariamente a informação referente a todas as categorias e seus referentes elementos, evitando a repetição de dados recolhidos, foi necessária a utilização de um dicionário com distintas chaves para assim conseguirmos diferenciar as categorias no nosso *dataset*. Dentro deste dicionário como segundo elemento e associada às nossas chaves categoria foi preciso a utilização de uma outra estrutura de dados coleção *set* onde serão armazenadas uma lista de elementos para cada categoria sem ocorrência de duplicação de dados.

```
dicionario = dict([('CategoriaX', set(ElemsX)), ('CategoriaY', set(ElemsY)),...])
```

Para além destas duas principais estruturas da nossa solução, durante a captura pelas nossas expressões regulares foi necessário uma estrutura da dados temporária usada como auxílio no armazenamento e tratamento de cada linha de informação pertinente do nosso *dataset*. Assim poder conseguirmos obter e separar a referente *tag*, *categoria* e *elemento* é utilizada uma lista adaptada às necessidades e à informação encontrada.

```
list = re.findall(r'er', dataset)
list = [tag, categoria, elemento]
```

3.2 Algoritmos

3.2.1 Text to HTML

Para realizarmos o processamento/filtragem de texto e a criação do website desejado foi criado um ficheiro *txttohtml.py* de onde serão executados os nossos algoritmos com base nas regras de produção *"Condição-Acção"*. Após análise do *dataset* disponibilizado *train.txt*, reconhecemos que cada linha não vazia possuí uma *tag* de classificação que nos indica se estamos perante uma linha de informação sobre uma categoria (**B-categoriaX**(*Begin*) significa que está a iniciar uma categoria, quando aparece **I-categoriaX**(*In*) é a continuação da categoria que foi inicializada na linha anterior) seguido de uma possível elemento associado, ou então se estamos perante algum tipo de informação não relevante a anotar/-recolher apresentada com a *tag* **O**.

Após a análise inicial foi tomada como estratégia, depois de uma prévia criação da página *web* inicial(*train.html*), primeiramente recolher e armazenar toda a informação desejada

do *dataset* para estruturas de dados temporárias e posteriormente aplicar a escrita de toda essa informação para as páginas *web* desejadas. Para isso construiu-se um filtro de texto que ao ser accionado, por encontrado informação útil, tivesse a acção de separar e armazenar devidamente os dados por grupos [*'tag'*,*'categoria'*,*'elemento'*], ao que chamamos respectivamente de [*"primeiraletra"*,*"categoria"*,*"nome"*]. Com auxilio da função *findall* do módulo *re*, a expressão regular para os nossos dados desejados do *dataset* declarado como *docTXT*, e armazenamento na lista *linha* é a seguinte:

```
1 linha = re.findall(r'([BI])-([A-Z\_]+)(\t[A-Za-z0-9]+)?', docTXT)
```

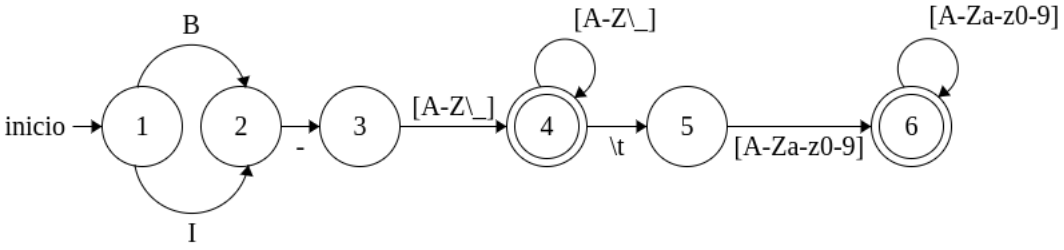

Figura 3.1: Autómato finito determinista simplificado/resumido para captura de *Linha*

Após isto, é executado um ciclo para percorrer a lista de dados para povoar e organizar a informação no nosso dicionário, assim conseguimos filtrar os inícios e continuação de cada elemento numa categoria, guardando assim cada categoria e seus elementos de forma interligada por chave categoria e sem repetição de dados em nenhum dos lados.

```
1  for primeiraletra, categoria, nome in linha:
2      nome = nome.strip()
3          if primeiraletra == 'B':
4              dicionario.setdefault(cur_categoria,set())
5              if nome:
6                  dicionario[cur_categoria].add(nome_aux.strip())
7                  nome_aux = ""
8              cur_categoria = categoria
9              nome_aux = nome_aux + "␣" + nome
10
11         elif primeiraletra=='I':
12                 nome_aux = nome_aux + "␣" + nome
```

Depois de termos todos os dados filtrados e devidamente armazenados, podemos passar à segunda fase nuclear do nosso algoritmo que é a criação do *website* produzindo os ficheiros *HTML* página principal(*train.html*) e para cada uma das categorias recolhidas.

Para isso precisamos de primeiramente percorrer o nosso dicionário por cada categoria e respectivas listas de elementos. Começando por inserir cada distinta categoria na nossa página principal com respectivo número de elementos calculados na lista que lhe está associada:

```
1 for cat, lis in dicionario.items():
2     docHTML.write("<div␣class=...".format(categor=cat, total=len(lis))
```

Em seguida, ainda dentro do nosso ciclo de acesso ao dicionário, é realizada a criação de cada ficheiro *categoriaX.html* com onde estará presente a listagem de todos os seus distintos elementos associados. Para isso é executado um segundo ciclo que irá percorrer toda a lista associada a tal categoria e realizar a sua escrita. É também escrito e associado ao título da página, com o nome da respectiva categoria, uma hiperligação para a página principal para que o retrocesso à página principal, e assim uma navegação dinâmica pelo *website* seja possível.

```
    with open("{categoria}.html".format(categoria=cat), "w") as f2:
        f2.write("<!DOCTYPE html>\n")
        ...
        f2.write("<title>{categoria}... .format(categoria=cat))
        f2.write("<h1><center><a href=\"train.html\">{categoria}...
        ...
        for name in lis:
            f2.write("<dl>{lista}</dl>".format(lista=name))
        ...
        f2.write(...)
```

3.2.2 Alínea Extra - Style.css

Adicionalmente ao pedido no enunciado, para tornar o nosso *website* visualmente mais apelativo e para uma utilização mais eficaz por parte do utilizador, foi construído um código na linguagem de estilo *CSS*, código esse devidamente invocado em todos os ficheiros *html* contruídos previamente. A seu código realiza aprimoramento nas cores, fontes e posicionamento dos textos, e a invocação e apresentação de uma imagem de fundo *'bg.jpg'* na página principal do *website*, alusiva à temática de cinema associada ao tipo de informação recolhida nos dados do *dataset* recebido.

```
(...)
.container{
    ...
    background-image: url('bg.jpg');
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;}
(...)
.item a{
    display: table;
    color: #fff;
    font-size: 16px;
    line-height: 1.4;
}
(...)
```

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Após a realização das nossas expressões regulares e posterior filtragem com sucesso na obtenção dos dados pretendidos, tivemos a difícil escolha de qual seria a estrutura de dados que melhor serviria para armazenar os nossos dados de maneira a solucionar e evitar os problemas que para nós eram os mais importantes. Neste caso decidimos dar prioridade a solucionar o problema de ter os dados devidamente interligados entre categorias e listas de elementos, evitando o problema de repetições dentro do extenso *dataset* recebido. Recorrendo a dicionários(*dict*) e colecções de dados (*set*), estruturas essas eficientes em resolver o problema de não permitir o armazenamento de valores duplicados. No entanto não há soluções perfeitas, e tínhamos a noção que a nossa escolha iria solucionar o problema que para nós era o principal, mas iria nos deixar com algumas dificuldades numa possível ordenação alfabética dos dados como objectivo extra ao enunciado, pois não seriam as estruturas mais amigáveis para o efeito. Assim decidimos escolher pelo caminho, para nós, mais importante, garantindo os nossos dados bem filtrados, associados e armazenados sem duplicações.

4.2 Testes realizados e Resultados

Durante os testes aplicamos sempre a abertura, leitura e utilização do *dataset*(*train.txt*) que nos foi disponibilizado, abaixo apresentamos um pequeno excerto do formato de texto lido pelo nosso programa:

```
1   B-TITLE     mississippi
2   O   in
3   O   the
4   O   title
5   O   show
6   O   me
7   B-GENRE     science
8   I-GENRE     fiction
9   I-GENRE     films
10  O   directed
11  O   by
```

```
12    B-DIRECTOR   steven
13    I-DIRECTOR   spielberg
14    O    do
15    O    you
16    O    have
17    O    any
18    B-GENRE      thrillers
19    O    directed
20    O    by
21    B-DIRECTOR   sofia
22    I-DIRECTOR   coppola
23
24    O    what
25    B-SONG       leonard
```

Depois inserirmos e ser processado o *dataset* completo na nossa aplicação, concluímos através dos outputs gerados pelos algoritmos executados que as categorias distintas existentes nos dados de entrada são:

```
1   ACTOR
2   YEAR
3   TITLE
4   GENRE
5   DIRECTOR
6   SONG
7   PLOT
8   REVIEW
9   CHARACTER
10  RATING
11  RATINGS_AVERAGE
12  TRAILER
```

Assim podemos concluir que de cada categoria podemos retirar a seguinte informação:

- **actor** informação sobre o nome dos atores
- **year** informação sobre o ano em que foi lançado
- **title** informação sobre o título do filme
- **genre** informação sobre o género de filme
- **director** informação sobre o diretor de realização do filme
- **song** informação sobre o nome da música
- **plot** informação sobre o enredo do filme
- **review** críticas ao filme
- **character** informação sobre as personagens

- **rating** a avaliação do filme
- **ratings_average** avaliação média
- **trailer** informação sobre a antevisão do filme

Nas nossas estruturas de dados estão neste momento guardadas todas a categorias e sua respectiva lista individual de elementos. Após a recolha e tratamento dos dados estar completa, a nossa solução cria os ficheiros *html* com os dados devidamente organizados. De seguida serão apresentadas imagens da página principal do *website* gerado com todas as categorias e referente contagem numérica de elementos, e uma das suas páginas secundárias, neste caso a de categoria *Director* com a listagem de todos os seus diferentes elementos, e com a opção de retrocesso para a página principal a testada e a funcionar correctamente.

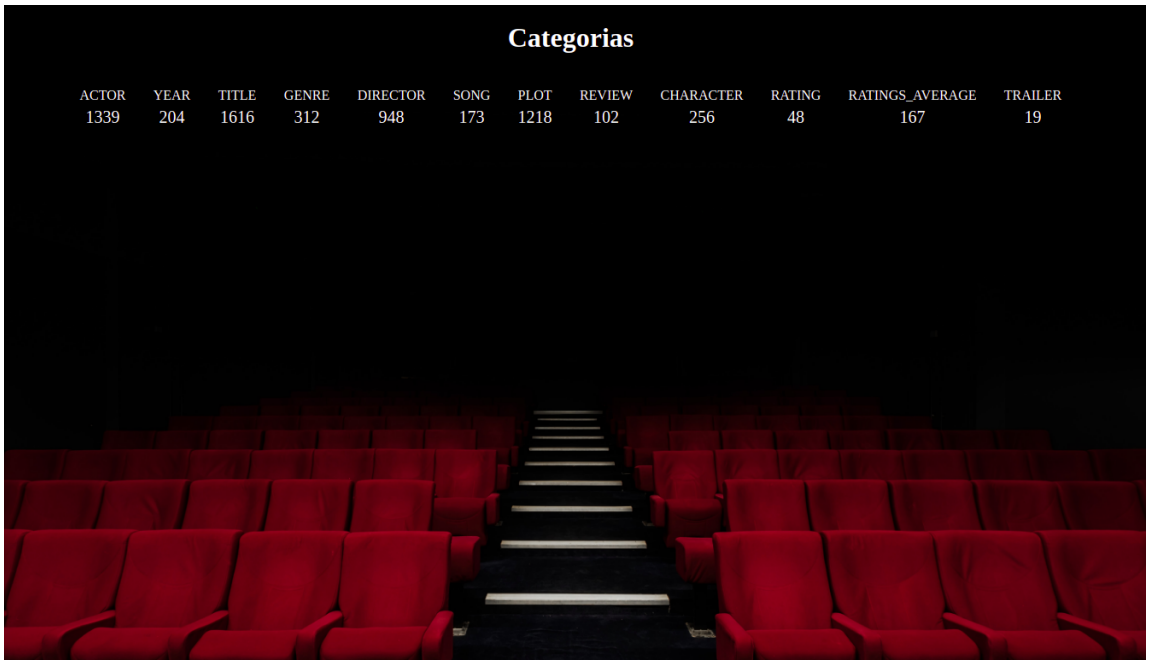

Figura 4.1: Página principal do *website* (*train.html*)

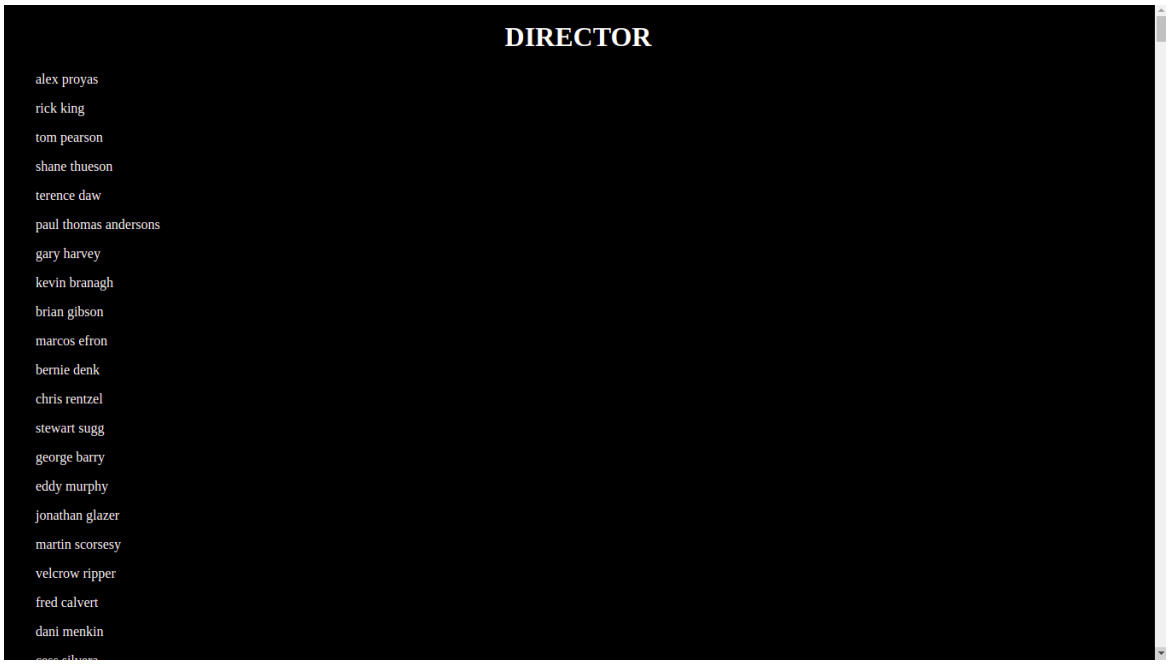

Figura 4.2: Página da categoria *Director* do *website* (*director.html*)

Capítulo 5

Conclusão

Síntese do Documento Estado final do projecto; Análise crítica dos resultados [**?**].
Trabalho futuro. Após o término do projecto realizado foi possível entregar uma solução de software capaz de aplicar com sucesso a extracção de vários elementos informativos em um *dataset* de treino utilizado na área de *Machine Learning*. Com o estudo e implementação de software utilizando a linguagem *Python* e seu módulo 'er', foi nos possível o desenvolvimento de processadores de linguagens regulares e filtros de texto (FT), a partir de expressões regulares príeviamente desenhadas e estruturadas para descrição de padrões de frases dentro dos dados recebidos, construindo assim uma solução capaz de filtrar e transformar o *dataset* nos nossos resultados pretendidos com base em conceitos de regras de produção Condição-Ação como foi requisitado.

Ao explorar o *website* criado pela nossa solução, é possível visualizar uma página principal apelativa com um fundo temático relacionado com os dados recolhidos sobre cinema, nessa página é ainda possível aceder à listagem de todas as categorias distintas existentes no *dataset* e quantos elementos estão associados a cada categoria. Clicando numa categoria dessa lista, o *website* leva-nos à visualização de uma nova página dedicada a essa categoria especifica, onde é apresentada a lista completa dos seus distintos elementos. Estas páginas de categoria possuem todas um link no título inicial para que seja possível o retrocesso à página principal, tornando assim o *website* fluído e agradável na sua navegação entre páginas.

Dado o amplo *dataset* fornecido seria possível, e desejável pela equipa de trabalho como plano futuro, alargar as capacidades da nossa aplicação para a apresentação de novas relações entre categorias através da realização de outros filtros de texto, e possíveis melhorias na ordenação de determinados dados.

Apêndice A

Código do Programa

A.1 texttohtml.py

```
1  import re
2
3  def txttohtml(docTXT):
4      dicionario = dict()
5      total = 0
6      with open("train.html", "w") as docHTML:
7          docHTML.write("<!DOCTYPE␣html>\n")
8          docHTML.write("<html>\n")
9          docHTML.write("<head>\n")
10         docHTML.write("<title>Machine␣Learning:␣datasets␣de␣treino</
               title>\n")
11         docHTML.write("<meta␣charset=\"utf8\">␣</meta>\n")
12         docHTML.write("<link␣c<link␣rel=\"stylesheet\"␣type=\"text/css
               \"␣href=\"style.css\"␣id=\"css_0\">\n")
13         docHTML.write("</head>\n")
14         docHTML.write("<body␣bgcolor=\"#B0C4DE\">\n")
15         docHTML.write("<h1><center>Categorias</center></h1>\n")
16         docHTML.write("<div␣class=\"container\">␣")
17
18
19
20         linha = re.findall(r'([BI])-([A-Z\_]+)(\t[A-Za-z0-9]+)?',
               docTXT)
21         dicionario = {}
22         nome = ""
23         nome_aux = ""
24         _, cur_categoria, _ = linha[0]
25         for primeiraletra, categoria, nome in linha:
26             nome = nome.strip()
27             if primeiraletra == 'B':
28                 dicionario.setdefault(cur_categoria,set())
29                 if nome:
30                     dicionario[cur_categoria].add(nome_aux.strip())
31                     nome_aux = ""
32                 cur_categoria = categoria
33                 nome_aux = nome_aux + "␣" + nome
```

```
34
35              elif primeiraletra=='I':
36                  nome_aux = nome_aux + " " + nome
37
38
39          for cat, lis in dicionario.items():
40              docHTML.write("<div class=\"item\"><a href=\"{categoria}.
                    html\">{categoria}</a><span>{total}</span></div>".
                    format(categoria=cat, total = len(lis)))
41              with open("{categoria}.html".format(categoria=cat), "w") as
                     f2:
42                  f2.write("<!DOCTYPE html>\n")
43                  f2.write("<html>\n")
44                  f2.write("<head>\n")
45                  f2.write("<title>{categoria}</title>\n".format(
                        categoria=cat))
46                  f2.write("<meta charset=\"utf8\"> </meta>\n")
47                  f2.write("<link rel=\"stylesheet\" type=\"text/css\"
                        href=\"style.css\" id=\"css_0\">\n")
48                  f2.write("</head>\n")
49                  f2.write("<body bgcolor=\"#B0C4DE\">\n")
50                  f2.write("<h1><center><a href=\"train.html\">{categoria
                        }</a></center></h1>\n".format(categoria=cat))
51                  f2.write("<dd> ")
52
53                  for name in lis:
54                      f2.write("<dl>{lista}</dl>".format(lista=name))
55
56                  f2.write("</dd> ")
57                  f2.write("</body>\n")
58                  f2.write("</html>\n")
59
60          docHTML.write("</div> ")
61          docHTML.write("</body>\n")
62          docHTML.write("</html>\n")
63
64
65
66      print(docHTML)
67
68
69  with open("train.txt", encoding="utf8") as f:
70      conteudo = f.read()
71      txttohtml(conteudo)
```

A.2 style.css

```
72 body,
73 html{
74     padding: 0;
75     margin: 0;
76     height: 100%;
77     color: #fff;
78     background:#000;
79 }
80 a{
81     text-decoration: none;
82 }
83 a:focus,
84 a:visited{
85     color: #fff;
86 }
87 h1{
88     color: #fff;
89 }
90 .container{
91     display: block;
92     width: 100%;
93     height: 100%;
94     text-align: center;
95 
96     background-image: url('bg.jpg');
97     background-position: center;
98     background-repeat: no-repeat;
99     background-size: cover;
100 }
101 .item{
102     display: inline-block;
103     vertical-align: top;
104     text-align: center;
105     padding: 16px;
106 }
107 .item a{
108     display: table;
109     color: #fff;
110     font-size: 16px;
111     line-height: 1.4;
112 }
113 .item span{
114     display: block;
115     color: #fff;
116     font-size: 20px;
117     line-height: 1.4;
118 }
```