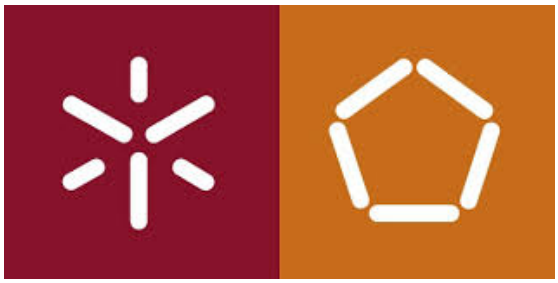

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Relatório do Trabalho Prático 1

Gonçalo Moreira A76861
Jorge Oliveira A78660
José Ferreira A78452

March 25, 2018

Abstract

Neste relatório vamos dar resposta ao exercício que nos foi proposto, no âmbito da unidade curricular Processamento de Linguagens do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Este exercício permitiu-nos consolidar competências na ferramenta gawk como também na matéria que foi lecionada até ao momento. A proposta de trabalho feita neste projeto está relacionada com a a leitura de um extrato de texto anotado com *freeling*. Iremos explicar todas as decisões que foram tomadas para a resolução dos exercícios como também todo o desenvolvimento do mesmo.

Contents

1 Introdução

O presente relatório tem como objetivo apresentar o raciocínio envolvido para a criação de programas que analisam texto. O mesmo apresenta explicações sobre a resolução de alguns exercícios utilizando a ferramenta *gawk*, sendo que, iremos explicar como abordamos cada pergunta como também as opções que tomámos.

Como já referimos, a realização deste trabalho envolveu a leitura de texto anotado em formato *freeling*. Este formato usa colunas separadas por espaços para a informação morfossintática de cada palavra. As colunas presentes são: número da linha, palavra, lema, pos-tag, pos(part of speech), features, árvore.

Foi-nos pedido para responder às seguintes questões:

- Contar o número de extratos
- Calcular a lista dos personagens do Harry Potter
- Calcular a lista dos verbos, substantivos, adjetivos e advérbios
- Determinar o dicionário implícito no corpora

2 Contagem do número de Extratos

Segundo o enunciado, o formato de texto *freeling* separa os extratos com uma linha em branco. Como tal, um extrato é separado de outro através de dois carateres `\n`. Para resolver este exercício apenas necessitamos de utilizar como *Record Separator* a expressão **`\n\n`**. Decidimos ter um **array associativo** para o número de extratos de cada ficheiro, indexado pelo nome do ficheiro (variável *FILENAME* do *awk*). Para saber qual o número de extratos de cada ficheiro apenas necessitamos de saber qual o número de registos do ficheiro em causa, que é dado no *awk* pelo **FNR**(*File Number Register*) no final de cada ficheiro. Para saber que era o final de um ficheiro utilizamos a condição ***ENDFILE***.

Sendo assim é passado um ficheiro de entrada e de seguida o mesmo é percorrido e é contado o número de extratos existentes.

3 Calculo do número de personagens

No ponto 2 era pedido que calculássemos o número de personagens existentes. Teríamos então de pesquisar por nomes próprios, por exemplo: Harry Potter. Em texto anotado em formato freeling, é fácil obter o nome das personagens pois cada palavra está identificada como sendo nome, verbo, adjetivo e advérbio. Neste caso tinhamos de pesquisar por nomes próprios. Como tal, na quinta coluna do texto fornecido, quando estamos perante um nome próprio, encontramos as letras "NP". A condição utilizada para obter o pretendido foi:

$5 ~ /^NP/

A cada ocorrência "NP" que é encontrada na quinta coluna, é adicionado ao array nomes o respectivo nome que se encontra na segunda coluna do texto.

Após efectuada a leitura do ficheiro efectuamos então a ordenação do array nomes por número de ocorrência. O awk tem duas funções de ordenação asort e asorti. No nosso caso faria sentido usar a asort, porém o awk altera os indices do array o que não nos interessaria. Desta forma utilizamos uma função "ordena" que iria ordenar por ordem decrescente do valor do array.

Para embelezar a apresentação dos nomes utilizamos uma função “imprimeFicheiro” para imprimir num ficheiro *HTML* toda a sintaxe para uma apresentação tabelar. Esta função é usada noutros pontos do trabalho, e será explicada adiante.

4 Calculo lista dos verbos, substantivos, adjetivos e advérbios

O objetivo do ponto 3 era encontrar todas as palavras que fossem ou verbo ou advérbio ou nome ou adjetivo e agrupá-las num ficheiro. Na explicação do processo, iremos mencionar “grupo” como sendo o conjunto formado pelas categorias mencionadas anteriormente.

Primeiramente tínhamos de saber se uma palavra pertencia ao grupo. Ao analisar os documentos do *corpora* verificamos que caso a palavra fosse o nome no campo número cinco começava por N, caso fosse um verbo o campo número 5 começava por V, e o mesmo para os restantes. Então usando a seguinte expressão regular, para o exemplo dos verbos, "$5 /^V/" e utilizando um array associativo verbos[$3]++ (o $3 contém a palavra que queríamos guardar), conseguíamos guardar todas as ocorrências

dos diferentes verbos. Aplicámos este processo para os restantes e já possuíamos toda a informação necessária.

Posto isto, apenas bastava guardar em ficheiros separados esta informação. Ao invés de a guardar em ficheiros de texto, decidimos armazenar em ficheiros html. Cada categoria do grupo originou um nome diferente do ficheiro. Para facilitar a leitura e escrita do código decidimos criar duas funções. A função "imprimeFicheiro" vai colocar toda a informação (nome da palavra e o número de ocorrências) presentes nos arrays e escrevê-la, através de prints redirecionados para o ficheiro em causa. Já a função "telaPrincipalHtml" vai tratar da tela de apresentação. Nesta tela serão apresentadas as quatro categorias. Cada nome da categoria tem uma hiperligação para o ficheiro correspondente criado anteriormente.

5 Determinar o dicionário implícito

No quarto ponto era pedido que determinassemos o dicionário implicito no corpora, i.e, uma lista contendo os lema, palavra e pos.

O lema encontra-se no campo número 3, a palavra no campo número 2 e o pos no campo número 6. Porém o campo número 6 não possui apenas o pos, também tem outros campos, como type, gen, etc. Um exemplo é "pos=determiner|type=article|gen=feminine|num=singular". Analisando este exemplo vemos que existe um padrão. Cada palavra surge após um carater de "=" ou "|", desta forma para conseguir obter o pos apenas tivemos de efetuar um split: split($6, p, "[=|]") , onde a palavra pretendida estaria no indice 2 do array designado por p.

Concluída esta fase, então estavamos em condições de armazenar a informação, para o fazer decidimos criar um array associativo com 3 níveis, array[lema][pos][palavra]. Desta forma, no final podemos apresentar uma lista de quais os pos e palavras que existem para aquele lema, i.e: lema { pos -> palavra }

Para além do que era pedido decidimos acrescentar uns extras. Para além do pos, decidimos considerar todas as outras formas existentes no campo 6. Para realizar esta tarefa tinhamos de percorrer o array p (o array que seria preenchido pela operação split mencionada anteriormente). Para saber o tamanho do array utilizamo o split, já que este devolve o número de palavras, o que iria coincidir com o comprimento de p. Uma vez determinado o caso de paragem, faltava então entender em que posições estariam as palavras. Analisando então, um simples exemplo: "pos=determiner|type=article", aplicando o split ficaria p[1]=pos, p[2]=determiner, p[3]=type e p[4]=article. Daqui, podemos inferir que as palavras pretendidas estão nas posições pares do p, pelo que ao percorre-lo, poderiamos começar com o i=2 e incrementá-lo em 2 unidades para ser mais eficiente. Após capturar as palavras era só juntá-las num único elemento. O armazenamento da informação efetuou-se da mesma forma.

Ainda decidimos criar outro extra, que não só era guardar a informação num ficheiro, como esse ficheiro estaria codificado em html, e os dados seriam apresentados por ordem alfabética de lema. Para ordenar alfabeticamente usamos uma função pré-definida do gawk *asorti*, que ordena por ordem ascendente em relação ao indice, porém tem um problema, pois altera os indices do array. Para contornar, decidimos guardar a informação num array auxiliar. Para imprimir a informação ordenada alfabeticamente por lema, então só tinhamos de percorrer o array "aux" e na posição aux[x] estaria o lema pretendido.

Para imprimir a informação para um ficheiro, utilizamos a função já mencionada anteriormente "imprimeFicheiro" com um parâmetro adicional, o array aux. Utilizando

a sintaxe do html, então criámos uma lista constituída por lemas, em que para cada para cada lema era criada uma lista com o "par" pos e palavra correspondentes.

Apesar de serem extras simples, permitiu-nos uma melhor visualização do dicionário, com a escrita em html, e ainda mais informação sobre os lemas presentes no corpora, com a adição das outras formas presentes no campo 6.

6 Gerar o texto original

Algo que decidimos realizar foi gerar o texto original a partir do texto anotado com *freeling*.

Para isso tivemos de utilizar como separador de registos o carater **"\n"** e como separador de campos o **" "**. Depois apenas queríamos o valor correspondente ao campo 2, que é o campo onde aparecem as palavras do texto. Para repor o texto original necessitamos de uma variável **texto**, que inicialmente começava como uma *string* vazia. Para além disso, teríamos de considerar alguns casos particulares:

- Quando se começava um novo extracto o texto tinha de começar na linha seguinte, ou seja era necessário acrescentar um "\n" ao **texto**. Para verificar o início de um novo extracto verificamos se o primeiro campo era 1, ou seja:

  $1==1

- Tivemos de ter em consideração os sinais de pontuação e os restantes carateres, visto que os sinais de pontuação não contêm espaço antes de serem escritos. Consideramos como sinais de pontuação os carateres: ".,!?:;". Para garantir que os sinais de pontuação não contivessem espaço, apenas realizamos uma operação de adicionar o sinal correspondente ao **texto**, sem nenhum outro separador. Para identificar que a linha correspondia a um sinal de pontuação, utilizamos a seguinte expressão:

  $2 ~ /[.,!?\:;]/ -> ou seja o segundo campo contém um dos sinais de pontuação que identificamos como não sendo antecedidos por espaço.

- Ao garantir que as restantes expressões fossem antecedidas por espaços, ainda tivemos de ter em consideração um detalhe que era, no início de um extracto não convinha que essa palavra fosse antecedida por espaço. Para isso definimos uma variável **carater** que começava como "", sempre que fosse o primeiro extracto (já definido anteriormente) e que quando não fosse um sinal de pontuação, então o novo **texto** seria igual ao **texto** antigo + o carater de separação + a nova palavra. Para identificar que não correspondia a um sinal de pontuação utilizamos a seguinte expressão:

  $2 !~ /[.,!?\:;]/

- Para além disto, no final de cada ficheiro imprimimos o texto gerado para um ficheiro com o nome: "texto_" + nome do ficheiro atual. Por forma a processar o próximo ficheiro, teríamos de recomeçar a variável **texto** como a string vazia, "". Para isto utilizamos a claúsula *ENDFILE* do *awk*.

7 Gerar probabilidades de classes consoante classes anteriores

Decidimos também implementar algo que nos diz a probabilidade de uma certa palavra ser de uma classe (pos, neste caso apenas o tipo) consoante as 2 palavras anteriores. Este gerador de probabilidades poderá ser um auxílio para uma ferramenta que anote um texto com *freeling* visto que ajuda a perceber qual a probabilidade de uma palavra ser duma certa classe. Essa ferramenta podia, por exemplo, consoante as duas palavras anteriores consultar as probabilidades da palavra ser de uma dada classe, escolhendo a classe com maior probabilidade, sendo que isto iria corresponder à realidade na maior parte dos casos.

Poderíamos ter usado mais do que apenas a classe, mas já dá uma ideia daquilo que será a palavra. Para além disso, podíamos ter utilizado outras formas como a palavra anterior e a seguinte, ou N palavras anteriores, mas pensamos que 2 anteriores funciona bem e é na nossa opinião a que faz mais sentido num caso prático de análise de um texto. Nos dois casos anteriormente referidos, o programa poderá ser estendido por forma a corresponder aquilo que se aproxima mais da realidade.

Estas probabilidades poderão ajudar a decidir qual a pos de uma palavra para uma ferramenta que gere textos deste tipo (*freeling*), consoante o método requerido, escolhendo o valor de maior probabilidade (com 2 palavras anteriores será aproximado de 100%, na maior parte dos casos). Para realizar o pretendido, utilizamos um **array associativo** com 4 indices, sendo:

1. Palavra
2. Classe da palavra 2x anterior
3. Classe da palavra anterior
4. Classe da palavra atual

Inicialmente começamos com a palavra 2x anterior e a anterior como sendo a *string* vazia e sempre que terminamos de ler um ficheiro colocamos ambas com a *string* vazia, também.
Decidimos começar com a string vazia devido a que no início não existe qualquer palavra anterior, e isto seria uma marca de início de texto. Isto ajudaria a ferramenta, quando fosse o início de um texto.
Era necessário começar novamente com a *string* vazia visto que no novo texto ainda não existem palavras anteriores, e caso não se começasse com a string vazia iria conter como palaras anteriores palavras de outro texto, o que iria resultar em erros.

Para colocar (ou aumentar o valor de ocorrências) de um quádruplo no array, não podiamos selecionar as linhas que não continham texto, ou seja, as linhas de mudança de extrato, daí termos utilizado a expressão regular: !(/^$/)

Depois o que queríamos realizar era, obter a classe da palavra atual, classe que estaria no campo 6. No entanto, o campo 6 continha mais informação do que apenas a classe: pos=classeltype=... Para obter a classe, utilizamos uma abordagem já anteriormente mencionada, usando o *split*.

Após esta fase, tínhamos as 2 classes das palavras anteriores, a classe da palavra atual e necessitávamos da palavra atual, mas esta era de acesso direto encontrando-se no segundo campo. Depois apenas tivemos de incrementar o valor do array segundo os indíces referidos anteriormente. (apenas é necessário incrementar devido ao mecanismo do awk)

No final tivemos de atualizar o valor da classe da palavra 2x anterior para o da palavra anterior, e o da palavra anterior para o da palavra atual. (pois na próxima palavra serão esses os seus lugares)

Por forma a simplificar a visualização, como já referido anteriormente, geramos ficheiros *HTML* para visualizar as ocorrências num browser. Neste exemplo decidimos criar uma página principal com a lista de todas as palvras, ordenadas por ordem alfabética. Para cada palavra criamos uma página que contém uma tabela com os campos:

- Classe da palavra 2x anterior
- Classe da palavra anterior
- Classe da palavra atual
- Probabilidade de ser da classe indicada.

Na página inicial criamos uma ligação da palavra para a página correspondente, que contém as tabelas.

Esta última concepção foi apenas para facilitar a visualização, visto que se fosse para ser usado como auxílio de uma ferramenta geradora de textos anotados com *freeling* seria mais fácil, por exemplo, criar um ficheiro com quíntuplos, ao invés de fazer o parsing do ficheiro todo em html. A pagina da palavra apresenta a seguinte consituição:

- Palavra
- Classe da palavra 2x anterior
- Classe da palavra anterior
- Classe da palavra atual
- Probabilidade de ser da classe indicada

assim facilitava o *parsing* dos ficheiros.

8 Conclusão

Com o presente trabalho realizado aprendemos como o *awk* pode ser usado de formas bem distintas, e o quão poderoso é esta ferramenta. O *awk* não é uma linguagem onde se tenha de escrever muito código, mas sim onde é preciso fazer uma análise cuidada do texto e depois aplicar a linguagem *awk* de uma forma simples. Não nos cingimos apenas ao que enunciado pediu. Para além, do que era proposto tentamos sempre aplicar uns extras, como por exemplo a criação de um ficheiro *HTML* ao invés de um simples *txt*. Estas alterações não foi só "porque sim" tiveram um prepósito, já que, para nós, se tornava mais fácil visualizar os resultados numa página do broswer com hiperligações e tabelas, ao invés num simples ficheiro em *txt*.

Em suma, achamos que a linguagem *awk* tem um grande poder expressivo e é de uma simplicidade incrível, produzindo resultados bastante interessantes. Pensamos que o resultado obtido foi positivo, visto que todas as perguntas obtiveram os resultados esperados. As duas perguntas extras acrescentadas também obtivemos os resultados pretendidos, tendo estas um uso real na prática. No entanto, poderíamos ter utilizado um mecanismo mais real na última pergunta extra gerando a nossa prórpia ferramenta de *freeling*, mas pensamos que o realizado já ajudará num futuro para a criação de uma ferramenta deste tipo.

9 Anexo

Nesta secção serão apresentados os resultados obtidos.

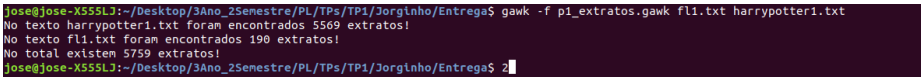

Figure 1: Resultados da Pergunta 1

Personagens

Palavra	Número de ocorrências
Harry	1428
Ron	664
Hermione	296
Lockhart	191
Hagrid	147
Dobby	144
Dumbledore	137
Malfoy	135
Riddle	111
Slytherin	103
Ginny	102
Fred	97
Snape	92
Gryffindor	89
Mc_Gonagall	86
George	79
Harry_Potter	79
Hogwarts	75
Muggles	63
Percy	62
Mrs._Weasley	60
Mr._Weasley	51
Filch	48

Figure 2: Resultados da Pergunta 2

Figure 3: Página de Apresentação para a Pergunta 3

Lista de Palavras

Adverbios

Palavra	Número de ocorrências
abaixo	15
aberto	1
abrupto	1
absoluto	4
acanhado	1
acaso	3
acima	30
adiante	1
afinal	14
afinal_de_contas	1
afirmativo	3
agitado	2
agora	92
agora_que	9
agressivo	2
ainda	132
ainda_assim	2
ainda_bem	3
alegre	2
algo	10
algures	4

Figure 4: Resultado da listagem dos advérbios

Figure 5: Resultados da Pergunta 4

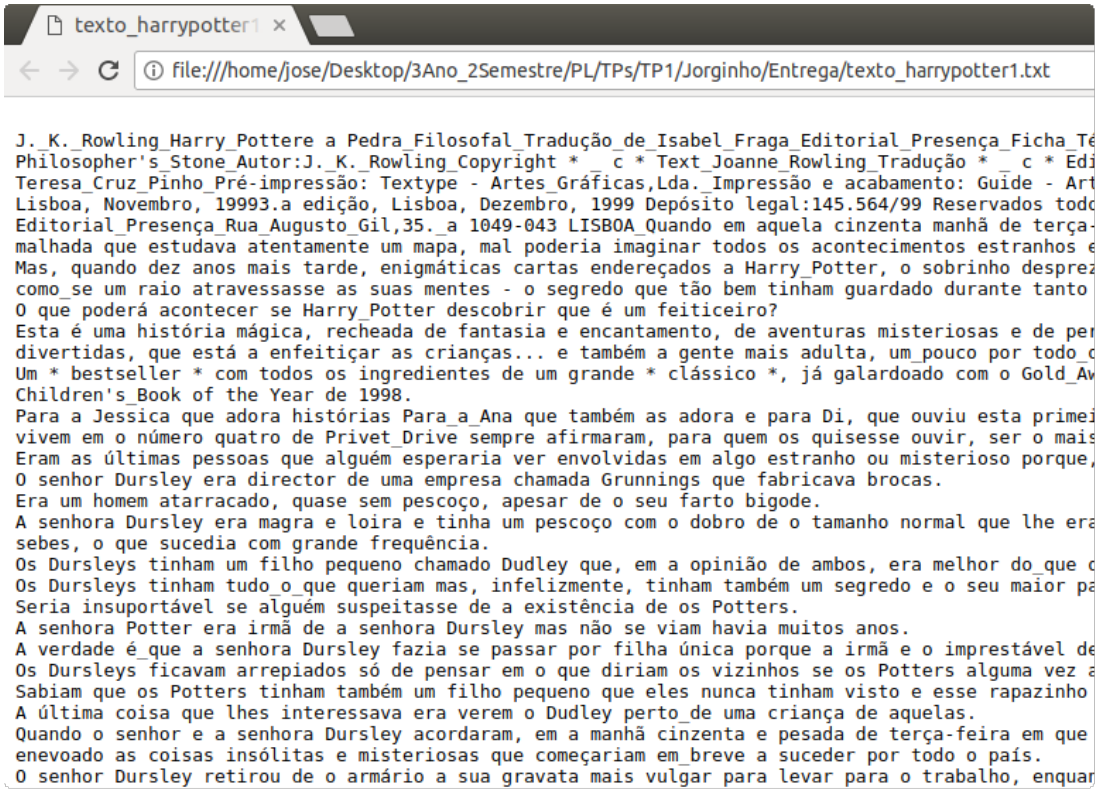

Figure 6: Excerto do Texto

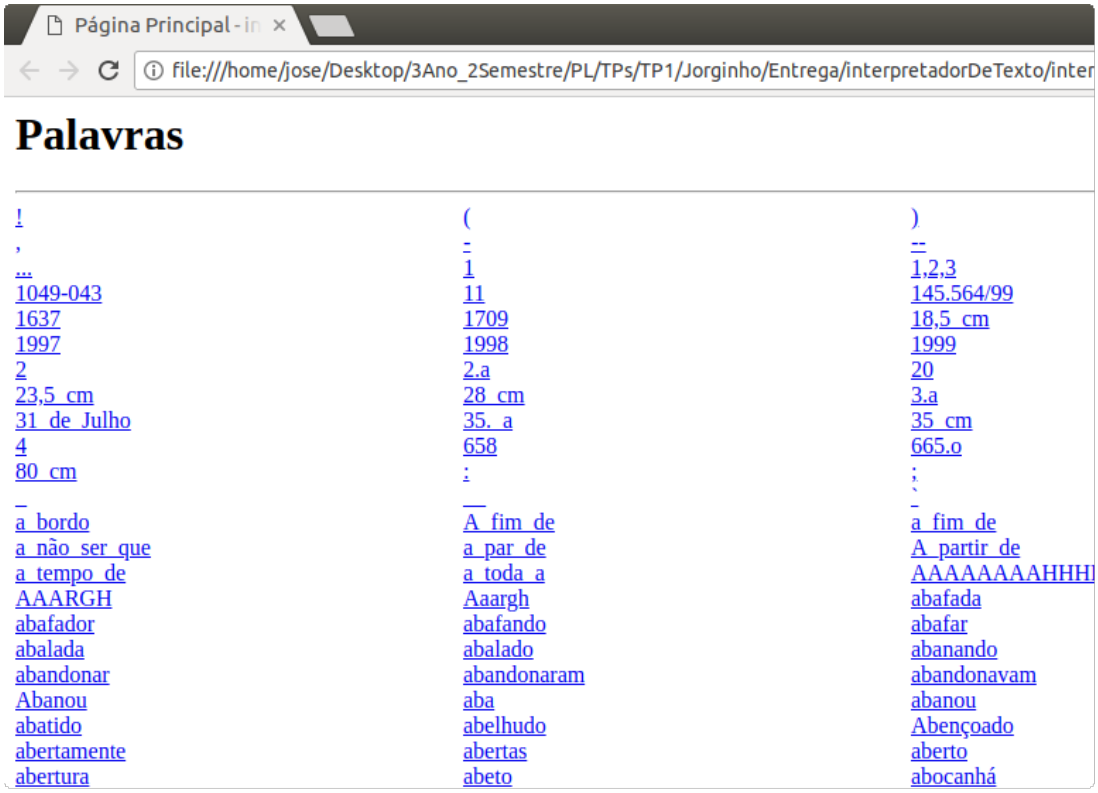

Figure 7: Excerto do Texto

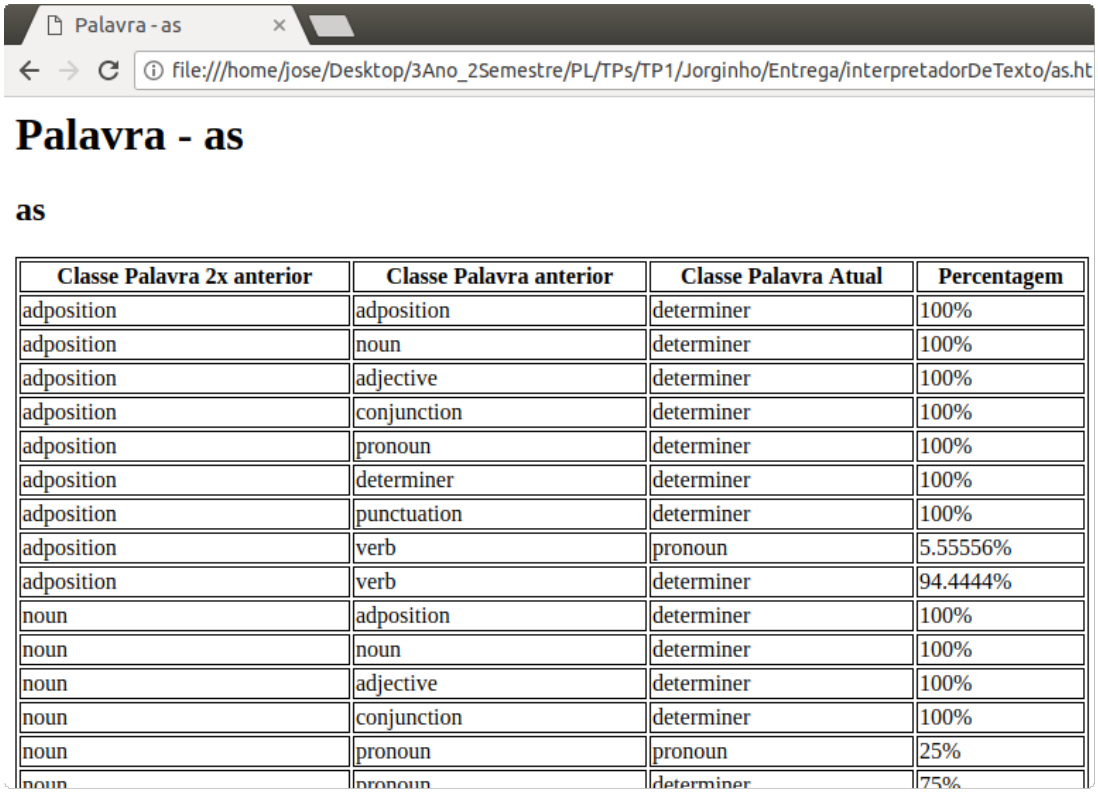

Figure 8: Excerto do Texto