Universidade do Minho

Mestrado Integrado em Engenharia Informática

Scripting no Processamento de Linguagem Natural

Trabalho 1 (2019/2020)

Grupo:

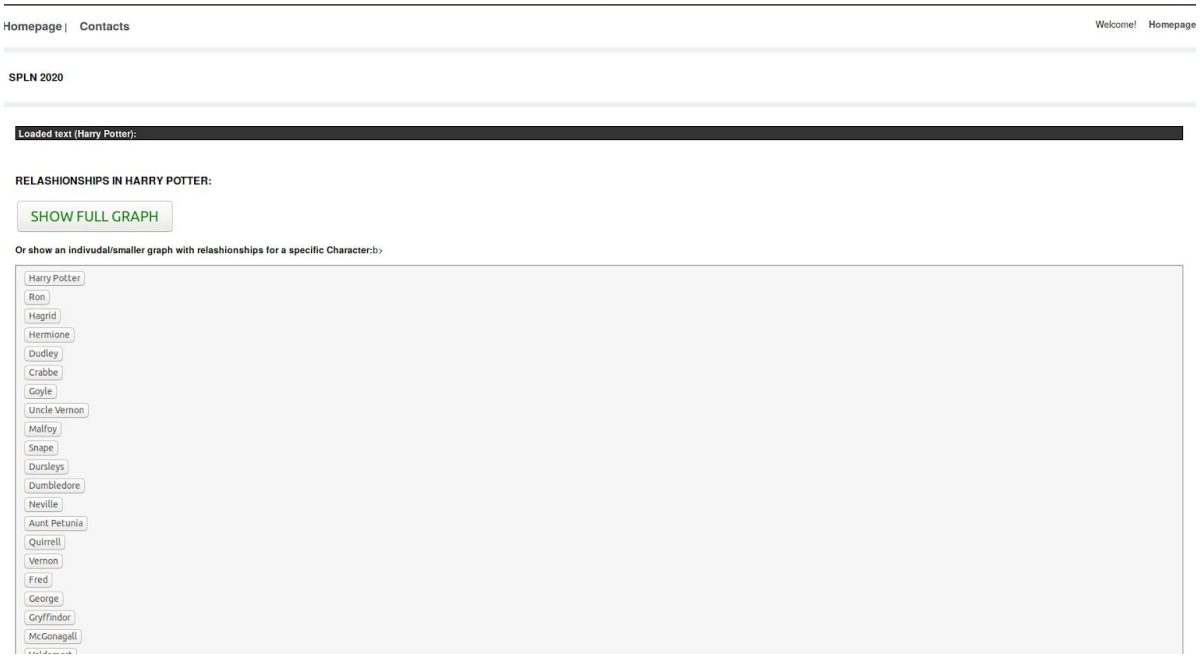
- ✓ Paulo Jorge Pereira Martins (Nº PG17918)
- ✓ Maria Inês Vieira
- ✓ Henrique Ribeiro

Neste trabalho foi-nos proposto o desafio de identificar nomes de personagens no livro Harry Potter and The Philosopher's Stone, de J.K. Rolling, e criar um grafo com a contabilização das relações entre elas.

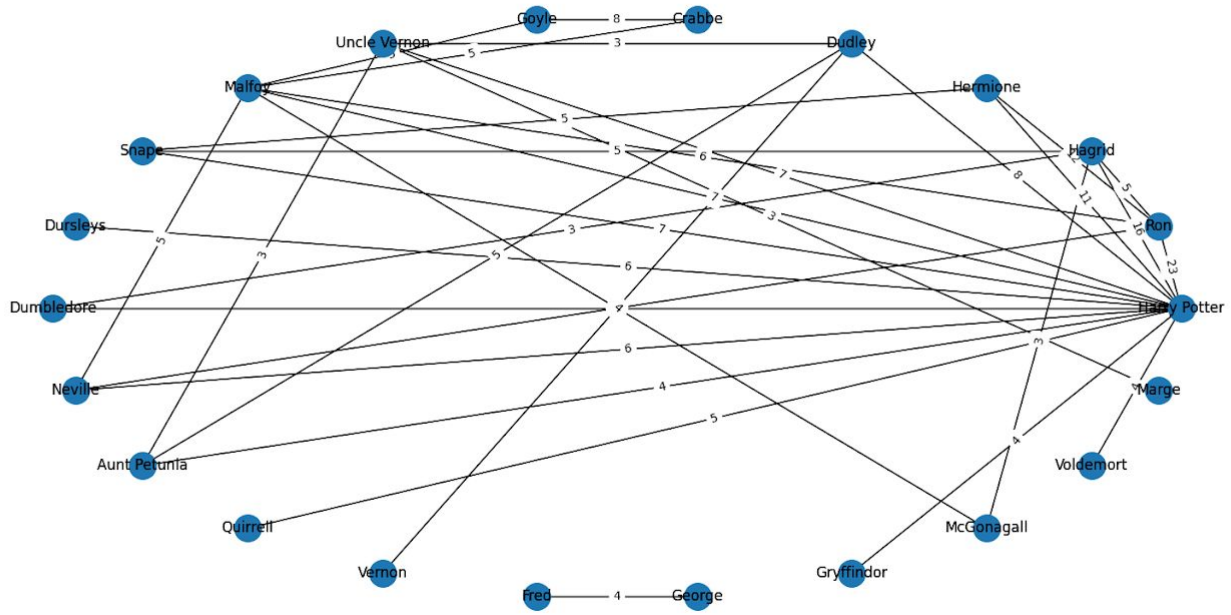
Para correr a aplicação basta executar o ficheiro “main.py” (testado com Python versão 3.6) que se encontra na raiz (executando “python3 main.py” na linha de comandos ou duplo clique). No entanto são necessárias algumas bibliotecas extra instaladas no ambiente Python, que podem ser instaladas através dos seguintes comandos:

pip3 install flask
pip3 install pywebview
pip3 install networkx
pip3 install numpy
pip3 install matplotlib
pip3 install pylab
Opcionalmente, para o módulo inativo baseado em NLTK: pip3 install nltk

Lançando a aplicação é inicializada uma interface gráfica:



O botão “SHOW FULL GRAPH” exibe o grafo com as relações entre personagens mais predominantes (filtramos para exibir as 33 com mais ligações, porque a partir desse número começava a exibir falso-positivos, como por exemplo o nome da vassoura “Nimbus 200” ou nomes de feitiços, etc.). Na seguinte imagem compilamos esse grafo:



Nas opções seguintes é possível exibir os grafos por personagem, com as suas relações isoladas (nota: em alguns sistemas de teste detectamos um bug, relacionada com o facto do Flask estar a ocupar a “main thread”, e a biblioteca que desenha o grafo “matplotlib” não aceitar correr fora da “main thread” devido à forma como o Flask funciona e por essa razão não atualiza o grafo inicialmente carregado; tentamos correr como thread isolada, também tentamos como multiprocess, no entanto em alguns sistemas dá erros devido a essa limitação intrínseca do Flask e bastante documentada na internet - se resultar em erro, deixamos a nota que na consola efetuamos sempre print de cada conjunto de dados enviados para o grafo, lá poderá ser consultado o dataset do grafo individual se necessário).

Optamos por criar uma interface gráfica para a aplicação, para facilitar a navegação nos Nodes. Para tal utilizamos uma GUI baseada em Webviews com Flask, ou seja: quando a aplicação principal é lançada (através do ficheiro *main.py*) é inicializada uma instância da biblioteca “pywebview”, que cria uma janela gráfica com um emulador web (tecnologia semelhante ao “Electrom” no qual foi construído o IDE Atom, por exemplo, moldura gráfica com NodeJS em pano de fundo), que correrá chamadas num servidor local baseado em Flask, sendo este inicializado numa *thread* paralela, servindo conteúdos na interface gráfica da *webview*.

Dividimos a aplicação em módulos lógicos: o package “views” alberga todas as classes relacionadas com a componente gráfica e controladores do *Flask;* o package “nlp” (abreviatura

de Natural Language Processing) alberga as classes que trabalham com o texto, nomeadamente a classe no módulo “characters.py” que é o coração da aplicação e processa os textos com recurso a *regular expressions*, para além da classe no módulo “nltkFilter.py” que atualmente não está em uso mas está totalmente funcional e utiliza a biblioteca NLTK para processamento de linguagem natural, para obtenção dos nomes das personagens com recurso a tagging e ferramentas várias (planeavamos utilizar esta biblioteca em conjugação para aprimorar os resultados e estabelecer comparações, mas por fugir às metodologias da cadeira e por falta de tempo desistimos de interligar com a interface gráfica, no entanto está completamente funcional), e a classe no módulo “graph.py” recorre às bibliotecas “networkx”, “matplotlib” e “pylab” para desenhar os grafos; por fim o package “data” apenas alberga os dados persistentes.

Devido à estrutura do documento original com o livro irregular, com frases quebradas em parágrafos e as palavras em início de cada a começar em maiúscula, numa primeira instância obtivemos imensos Nomes Próprios falsos devido a esse facto. Para lidar com esse problema optamos uniformizar o texto e limpar todas quebras de linha (“\n”), transformando a primeira letra de cada quebra em minúscula, para não ser confundida com um nome próprio. Também criamos vários filtros para limitar o número de falso-positivos (p.e. apagamos vários prefixos como “Mr.”, “Professor” etc. no começo de vários nomes. Também optamos por uniformizar alguns nomes, como por exemplo “H. Potter”, “Mr. Potter”, “Harry” todos remetem para “Harry Potter”.

Acabamos por não ter tempo de implementar todas as funcionalidades e filtros que desejávamos (tentamos implementar filtros bigram através do NLTK, para detetar nomes seguidos de verbos em formatos comuns para nomes, mas acabamos por não ter tempo - mas deixamos a referência como exemplo de estratégia para refinar ainda mais os dados obtidos).