Scripting no Processamento de Linguagem Natural

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Trabalho Prático 3

Ferramenta `inoti-make`

Grupo 7:
A73831 - João Barreira
A77364 - Mafalda Nunes

11 de Julho de 2019

Resumo

Este relatório procura descrever a resolução do trabalho prático 3, proposto na unidade curricular SPLN, do Mestrado Integrado em Engenharia Informática, da Universidade do Minho.

Efetua-se uma descrição da implementação de uma ferramenta que tira partido do utilitário *inotify*, procedendo-se à exemplificação de algumas aplicações práticas possíveis, de entre as quais se destacam a correção ortográfica e o reconhecimento visual de textos, efetuados de forma automática em determinadas diretorias, monitorizadas para o efeito.

Conteúdo

1 Introdução

O presente trabalho prático foi desenvolvido no âmbito da disciplina de `SPLN` e baseia-se na opção 3 apresentada no enunciado, proposta pelos docentes da UC. Optou-se por desenvolver uma ferramenta, que se nomeou de `inoti-make`, que permita definir padrões e diretorias a observar, bem como funções ou *scripts* a executar em reação a eventos detetados nessas localizações.

No *kernel* do *Lixux* já existe um mecanismo capaz de monitorizar eventos do sistema de ficheiros. Esta funcionalidade, designada por `inotify`, permite registar ficheiros individuais ou diretorias para observação e bloquear ou esperar pela notificação de eventos associados aos mesmos.

O objetivo deste projeto é criar uma ferramenta que permita a um utilizador, que não possua qualquer conhecimento sobre o mecanismo `inotify`, utilizá-lo de forma simples para executar automaticamente operações mais ou menos complexas sobre determinado conteúdo de uma diretoria. Para isso, o utilizador apenas deverá ter de indicar as diretorias que devem ser monitorizadas, os ficheiros/padrões cujos eventos associados se pretende considerar, o tipo de evento que espera e respetivas ações a executar.

Também se pretende criar ferramentas auxiliares ou aplicações, mais ou menos complexas, para ilustrar a utilização do `inoti-make`.

2 Implementação do `inoti-make`

Para se desenvolver o `inoti-make`, recorreu-se à biblioteca *inotify* [1] do *python*, que apenas requer a adição inicial de *watches* nas diretorias a monitorizar, seguida da entrada num ciclo em que cada iteração corresponde a um evento detetado, estando o processo bloqueado entre iterações.

O único problema na utilização da referida biblioteca centra-se no facto dos *watches* serem adicionados de acordo com um *path* para a diretoria a monitorizar, sendo que na gestão interna do *inotify* se utilizam os *i-nodes*. Assim, surgem problemas quando, por exemplo, durante a monitorização de uma diretoria, se altera o respetivo nome. Nesse caso, a diretoria continua a ser vigiada devido ao seu *i-node* ser o mesmo.

Por outro lado, se durante a monitorização da diretoria esta for removida e adicionada outra com o nome da primeira, a nova diretoria não é monitorizada, pois, apesar do nome indicar que o deveria ser, o que identifica o objeto a monitorizar é o *i-node*.

Para automatizar a utilização desta ferramenta, optou-se por utilizar um ficheiro de configuração, construído pelo utilizador do `inoti-make`, que apresenta os seguintes campos para cada diretoria ou conjunto de diretorias a monitorizar:

- `FoldersPath` – caminho para as diretorias a monitorizar, com a presente configuração;
- `NamesRegex` – nome dos ficheiros ou diretorias, dentro das `FoldersPath`, a monitorizar (podem utilizar-se expressões regulares);
- `Recursive` – indica se todas as diretorias internas devem ser monitorizadas com as mesmas configurações da indicada em `FoldersPath`;
- *Tipo de Eventos* – tipo de eventos detetados (separados por '|') que se devem considerar, bem como respetivas ações a executar. Estes eventos podem ser classificados da seguinte forma:
  - `IN_ACCESS`
  - `IN_MODIFY`
  - `IN_ATTRIB`
  - `IN_CLOSE_WRITE`
  - `IN_CLOSE_NOWRITE`
  - `IN_OPEN`
  - `IN_MOVED_FROM`
  - `IN_MOVED_FROM`
  - `IN_MOVED_TO`
  - `IN_CREATE`
  - `IN_DELETE`
  - `IN_DELETE_SELF`
  - `IN_MOVE_SELF`
  - `IN_ALL_EVENTS`

Relativamente à estrutura do ficheiro de configuração, esta pode ser representada, de forma geral, da seguinte maneira:

```
FoldersPath:    ['FOLDER1_PATH', 'FOLDER2_PATH', ...]
NamesRegex:     ['FILENAME_1', 'FOLDER_2', ...]
EV1|EV2|...:    [('SHELL_COMMAND', 'command'), ...]
EV3|...:        [('SHELL_COMMAND', 'command'), ...]
NamesRegex:     ['FILENAME_1', 'FOLDER_2', ...]
EV1|EV2|...:    [('SHELL_COMMAND', 'command'), ...]
EV3|...:        [('SHELL_COMMAND', 'command'), ...]

FoldersPath:    ['FOLDER3_PATH', 'FOLDER4_PATH', ...]
...
```

É de se referir que os comandos ou ações indicados vão ser executados na diretoria monitorizada. Assim, torna-se útil a definição de algumas variáveis que podem ser utilizadas nos comandos especificados. As variáveis definidas são as apresentadas de seguida:

- `$NAME_EXT` – nome do ficheiro associado ao evento detetado;
- `$NAME` – nome do ficheiro associado ao evento detetado, sem extensão;
- `$CURRENT_DIR` – caminho para a diretoria de onde a ferramenta é executada;
- `$WATCH_DIR` – caminho para a diretoria monitorizada;
- `$WATCH_DIR_LAST` – nome da diretoria monitorizada.

Para se utilizar a ferramenta *inoti-make*, é necessário instalar primeiro o *package* do programa, através da execução dos seguintes comandos:

```
python3 setup.py sdist bdist_wheel
pip3 install --user ./dist/inoti_make-0.0.1.tar.gz
```

De seguida, basta executar-se o programa, da seguinte forma:

```
inoti_make_ex <CONFIG_FILE>
```

O programa ficará a correr até que seja interrompido (`^C`), monitorizando as diretorias definidas no ficheiro de configuração passado como argumento.

3 Aplicações Desenvolvidas

Com o objetivo de se apresentar exemplos de aplicações práticas desta ferramenta, desenvolveram-se outras, auxiliares, que serão utilizadas no ficheiro de configuração do `inoti-make`.

3.1 LaTeX *to* PDF

A ferramenta *inoti-make* pode ser facilmente utilizada para efetuar a compilação automática de ficheiros LaTeX.

Para isso, utilizou-se a seguinte configuração:

```
FoldersPath:            ['listened_folders/latex']
NamesRegex:             ['^.*\.tex$','^.*\.ltx$','^.*\.latex$']
Recursive:              True
IN_CREATE|IN_MODIFY:    [('SHELL_COMMAND', 'pdflatex $NAME_EXT;
                        pdflatex $NAME_EXT')]
IN_DELETE:              [('SHELL_COMMAND', 'rm $NAME.*')]
```

Desta forma, basta adicionar a pasta onde se encontra o ficheiro LaTeX como sendo a `FoldersPath` e definir a execução dos comandos de compilação sempre que for detetada a criação ou modificação destes ficheiros (*.tex*, *.ltx* ou *.latex*). Para isso, executa-se duas vezes o comando *pdflatex* com o nome do ficheiro em questão, isto é, `$NAME_EXT`, que se refere ao nome do ficheiro que originou o evento, incluindo a extensão.

Em contrapartida, caso seja detetada a remoção de um desses documentos, serão também removidos todos os ficheiros auxiliares gerados durante a compilação anterior, que podem ser obtidos através do nome do ficheiro original, com uma extensão qualquer (i.e. `$NAME.*`).

3.2 C *Code Organizer*

A ferramenta `inoti-make` pode ainda ser utilizada com o objetivo de organizar e compilar automaticamente os ficheiros correspondentes a um projeto da linguagem C: ficheiros *.c* (incluindo a *main*) e ficheiros *.h* (*headers*).

Para se atingir esse objetivo, começou-se por desenvolver um *script bash*, que move todos os ficheiros *.c* (exceto a *main*) e *.h* para uma diretoria *includes*. Por cada ficheiro movido para essa diretoria, atualiza-se, com recurso ao comando *sed*, o respetivo *include* no ficheiro *main.c*, caso este exista. Para além disso, regista-se o nome do respetivo ficheiro numa variável, de forma a atualizar a *makefile* posteriormente.

Relativamente à *makefile*, utiliza-se o comando *grep* para verificar se cada um dos componentes da mesma já foi especificado. Para os componentes estáticos, como os comandos para *compile*, *run* e *clean*, apenas se verifica a sua inexistência, caso em que se adiciona os mesmos (comando *printf* com redirecionamento do tipo *append*). Relativamente à componente variável, *$objects*, acrescenta-se a mesma à *makefile*, caso esta ainda não exista, ou só se atualiza com os últimos ficheiros movidos, caso exista.

O código deste *script*, denominado *c_code_organizer.sh*, é apresentdo no anexo A. Para combinar esta ferramenta com o *inoti-make*, utilizou-se a seguinte configuração:

```
FoldersPath:            ['listened_folders/c']
NamesRegex:             ['^.*\.c$','^.*\.h$']
Recursive:              True
IN_CREATE:              [('SHELL_COMMAND',
                        'cp "$CURRENT_DIR/scripts/c_code_organizer.sh" .;
                        ./c_code_organizer.sh; rm c_code_organizer.sh')]
IN_MODIFY:              [('SHELL_COMMAND', 'if [ "$NAME_EXT" = "main.c" ];
                        then make; else cd ..; make; fi')]
```

Tendo em conta que o *script c_code_organizer.sh* foi desenvolvido com a noção de mover ficheiros *.c* e *.h* para uma diretoria *includes*, dentro da diretoria corrente (*paths* relativos), houve necessidade de se copiar o *script* da respetiva diretoria para a monitorizada. Assim, quando se deteta a criação de um ficheiro *.c* ou *.h*, efetua-se a referida cópia, executa-se o *script* e remove-se o mesmo de seguida.

Caso sejam detetadas modificações de ficheiros, apenas se executa o comando *make*, na diretoria atual (se foi o ficheiro *main.c* que foi alterado) ou na anterior (caso tenha sido um dos ficheiros dentro de *includes* a ser alterado).

Assim, consegue-se obter um mecanismo automático de organização de ficheiros dum projeto `C` e geração da *makefile* e do executável.

3.3 *Ignore* IDE *folders on git*

Uma situação bastante comum quando se trabalha com sistemas de controlo de versões (e.g. *git*) é a presença, nos repositórios, de ficheiros de configuração de projetos, que são criados automaticamente pelos diversos *Integrated Development Environments* (IDEs) utilizados pelos contribuidores, para além do código do projeto.

Face à dispensabilidade dos referidos ficheiros de configuração, uma das soluções mais comummente empregues é a adição dos respetivos nomes ao ficheiro *.gitignore*, de forma a haver uma independência entre o código presente no repositório e o contexto em que este é desenvolvido ou utilizado.

Assim sendo, a ferramenta `inoti-make` pode também ser facilmente utilizada para automatizar este processo. Para tal, foi utilizada a seguinte configuração:

```
FoldersPath:            ['listened_folders/git']
NamesRegex:             ['^\.idea$', '^\.iml$', '^\.iws$',
                        '^\.classpath$', '^\.project$', '^\.settings$']
Recursive:              False
IN_CREATE:              [('SHELL_COMMAND',
                        'echo "$NAME_EXT" >> .gitignore')]
IN_DELETE:              [('SHELL_COMMAND',
                        'sed "/$NAME_EXT/d" .gitignore')]
```

Assim, basta definir a pasta correspondente à raíz do projeto como sendo a `FoldersPath` e, ao ser detetada a criação de um ficheiro ou subpasta cujo nome corresponda aos ficheiros de configuração do IDE utilizado, adicionar os mesmos automaticamente ao *.gitignore* através da execução do utilitário *echo*.

Em contrapartida, caso seja detetada a remoção de um destes ficheiros ou pastas, pode-se recorrer ao utilitário *sed* para remover as ocorrências correspondentes:

Alguns exemplos de ficheiros/pastas de configuração de IDEs, que são os monitorizados de acordo com a configuração previamente especificada, são:

- *idea/*, *.iml*, *.iws* (`PyCharm`, `IntelliJ`);
- *.classpath*, *.project*, *.settings/* (`Eclipse`).

Este procedimento poderá ainda ser facilmente estendido para ignorar automaticamente, por exemplo, ficheiros de documentos LaTeX, do `Maven` (pastas *log* e *target*) ou ficheiros *.DS_Store* de `macOS`.

3.4 *Spell Corrector*

Existem diversas bibliotecas ou formas disponíveis para a identificação e correção de erros de escrita. Após uma pesquisa sobre as mesmas, optou-se por implementar um corretor ortográfico com diversos modos de operação, sendo que a forma de deteção de erros é semelhante em todos eles e passa pela comparação das palavras encontradas com um dicionário base. A correção dos erros é efetuada de forma distinta entre os modos implementados:

- "`Manual`" – correção de erros é efetuada através de edições básicas da palavra errada encontrada (remoção, transposição, inserção ou substituição de um caractere). Começa-se por verificar se a palavra encontrada existe num dicionário. Caso tal não se verifique, obtém-se palavras com uma edição a partir da original, verificam-se quais existem no dicionário e comparam-se probabilidades para obter a melhor sugestão. Caso uma edição não seja suficiente, efetuam-se duas edições à palavra e segue-se o resto do processo previamente descrito. Se as palavras obtidas continuarem a não existir no dicionário, devolve-se a palavra original. Realizar estas ações para cada palavra torna o processamento de textos de elevada dimensão bastante lento. Esta ideia foi obtida a partir de um *website* [2], tendo-se desenvolvido bastante a mesma para obtenção de melhores resultados, devido ao maior controlo sobre o processo de correção em relação às restantes alternativas.
- `Aspell` – correção baseia-se no motor de *spelling* `GNU Aspell` [3].
- `Hunspell` – correção baseia-se no motor de *spelling* `Hunspell` [4].
- `Symspell` – correção baseia-se no motor de *spelling* `SymSpell v6.3` [5].

Para a integração no `inoti-make`, apenas se recorreu, a título ilustrativo, ao modo "Manual", tendo-se utilizado a configuração apresentada de seguida:

```
FoldersPath:            ['listened_folders/correct']
NamesRegex:             ['^.*\.txt$']
Recursive:              False
IN_CREATE|IN_MODIFY:    [('SHELL_COMMAND',
                        'cd "$CURRENT_DIR/scripts"; python3
                        spellcorrector.py < "$WATCH_DIR/$NAME_EXT"
                        > temp.txt;
                        if diff "$WATCH_DIR/$NAME_EXT"
                        temp.txt > /dev/null; then rm temp.txt;
                        else cp temp.txt "$WATCH_DIR/$NAME_EXT"; fi')]
```

Caso se detete criação ou modificação de ficheiros de texto *.txt*, muda-se para diretoria onde se localiza o *script* a utilizar e, de seguida, executa-se o mesmo. O *output* gerado é armazenado num ficheiro temporário *temp.txt*. De seguida, caso este ficheiro seja diferente do que gerou o evento em questão, substitui-se o segundo pelo primeiro. Caso contrário, remove-se o ficheiro temporário.

Foi necessário efetuar esta verificação de igualdade como situação de paragem, pois, caso contrário, o ficheiro estaria sempre a ser modificado quando corrigido e, essa alteração, geraria um novo evento que levaria a uma nova correção (ciclo infinito).

3.4.1 Modo de Correção "`Manual`"

Este foi o modo de correção mais personalizado neste trabalho. De facto, optou-se por dar especial relevância ao cálculo das probabilidades de uma palavra ser a correta, de forma a otimizar a percentagem de resultados certos obtidos.

Para isso, começou-se por analisar que dados seriam necessários para a obtenção de uma probabilidade realista, não só tendo em conta a frequência das palavras num texto de treino, mas também o contexto das mesmas, em termos de classes gramaticais. Assim, optou-se por definir uma fase de treino, com utilização de um texto de treino de tamanho considerável e um dicionário de língua portuguesa [6], que permitisse gerar as duas estruturas apresentadas de seguida:

- `pos_freq` – contador que indica a quantidade de ocorrências de conjuntos de 3 palavras ou sinal de pontuação com certo *part-of-speech* (POS) ou classe gramatical e determinada *tag* - género, tempo verbal, pessoa, *etc.* Para a obtenção destes dados, recorreu-se à utilização da biblioteca *spaCy* [7], com o módulo *pt* carregado. Utiliza-se a *tag* `<UNKNOWN>` para indicar palavras desconhecidas (que não se encontram no dicionário). A grande variedade de *tags* existente sugere a necessidade de se efetuar a operação de treino com um texto de tamanho considerável.

- `words_freq` – dicionário cujas chaves são as palavras encontradas no texto e no dicionário de português convertidas para minúsculas e os valores são contadores. Esses contadores indicam quantidade de ocorrências de cada palavra encontrada no texto e no dicionário (sem nenhuma transformação). Após o processamento do texto de treino, percorre-se o dicionário de português e adiciona-se quantidade 1 a todas as palavras encontradas, de forma a conseguir-se identificar essas como escritas corretamente, mesmo que não tenham aparecido no texto de treino.

As estruturas obtidas são armazenadas no formato *json*, podendo ser recuperadas mais tarde. Este armazenamento justifica-se pelo facto da geração dessas estruturas ser bastante demorado para textos de tamanho elevado. Para posteriormente se obter valores de probabilidade fiáveis, processou-se 20 `MB` de texto de treino (notícias), o que demorou cerca de 2 horas. Note-se que o gerador destas estruturas espera receber texto limpo e bem escrito, sem nenhuma formatação especial (*html*, *xml*, *etc.*).

Relativamente ao processo de identificação e correção de erros, que foi previamente explicitado, basta salientar a alteração da forma como se calcula a probabilidade de uma palavra ser a correta. Para cada palavra ou sinal de pontuação analisados, obtém-se agora o anterior e posterior ao mesmo. Assim, torna-se possível a obtenção dos respetivos POS e *tag*.

Para o cálculo da probabilidade, pretendeu-se obter uma fórmula que tivesse em consideração os seguintes aspetos:

- Se uma palavra não existir no dicionário, a probabilidade de ser correta é 0;
- Se a ocorrência do tuplo com as POS e as *tags* for nula, devolver uma probabilidade muito reduzida (caso palavra exista no dicionário);
- Se palavra testada comparada com *ignore case* à original (com erro) for igual, a probabilidade deve ser afetada pela positiva.

Assim, optou-se pela utilização da seguinte fórmula para o cálculo da probabilidade de uma palavra estar correta, considerando que ela existe no dicionário:

$$100 \times \frac{words\_freq[word.lower][word]}{N_{words}} \times weight \times \frac{99 \times pos\_freq.get(context\_pos, 0) + 1}{N_{pos}}$$

onde,

$weight =$ `60 if word.lower \= original_word.lower else 100`,

$N_{words} =$ `sum(words_freq.values.values)`,

$N_{pos} =$ `sum(pos_freq.values)`.

Após todo este processamento, que torna todas as palavras corrigidas em minúsculas (exceto quando são siglas), utilizam-se expressões regulares para tornar a primeira letra de cada frase maiúscula.

Salienta-se, para a utilização da biblioteca *spaCy* com língua portuguesa, a necessidade de instalação do respetivo módulo:

```
python3 -m spacy download pt
```

3.4.2 Modos de Correção `Aspell`, `Hunspell` e `Symspell`

Relativamente a estes modos de correção, optou-se por utilizar simplesmente o mecanismo de sugestões fornecido pelas bibliotecas *aspell*, *hunspell* e *symspellpy*, respetivamente, escolhendo como correta a primeira opção.

Experimentou-se tentar utilizar a mesma fórmula de probabilidade anteriormente referida, bem como algumas variações, para obter melhores resultados. Contudo, a avaliação dos resultados foi negativa, possivelmente porque as sugestões têm palavras com várias quantidades de modificações (edições no modo "Manual") misturadas.

A instalação das referidas bibliotecas requer, conforme referido nos respetivos repositórios `Git`, a instalação de outras ferramentas. Assim, pode-se utilizar os comandos apresentados de seguida para instalar todas as dependências necessárias:

```
# ASPELL
sudo apt-get install libaspell-dev
python3 -m pip install --upgrade
sudo pip3 install aspell-python-py3

# Obtenção do dicionário de português ASPELL
wget https://ftp.gnu.org/gnu/aspell/dict/pt_PT/aspell6-pt_PT-20070510-0.tar.bz2
tar -xvjf aspell6-pt_PT-20070510-0.tar.bz2
cd aspell6-pt_PT-20070510-0/
./configure
make
make install
# python: s = aspell.Speller('lang', 'pt_PT')


# HUNSPELL
sudo apt-get install libhunspell-dev
sudo pip3 install hunspell
# Dicionário obtido a partir de ficheiro local, em tempo de execução


# SYMSPELLPY
sudo pip3 install symspellpy
# Dicionário obtido a partir de ficheiro local, em tempo de execução
```

3.4.3 Avaliação dos Modos de Correção

De forma a avaliar os modos de correção implementados, optou-se por desenvolver um *script* capaz de medir as seguintes métricas, para cada um desses modos e um texto de *input* gramaticalmente correto:

- `True Positives Wrong` – quantidade de palavras com erros ortográficos corretamente detetadas, mas corrigidas de forma errada;
- `True Positives Right` – quantidade de palavras com erros ortográficos corretamente detetadas e corrigidas;
- `True Negatives` – quantidade de palavras corretamente classificadas como bem escritas;
- `False Positives` – quantidade de palavras erradamente classificadas como mal escritas;
- `False Negatives` – quantidade de palavras erradamente classificadas como bem escritas;
- `Accuracy` – percentagem de palavras bem classificadas e corrigidas (caso classificadas como possuindo erros ortográficos);
- `Precision` – porção de palavras com erros ortográficos corretamente classificadas e corrigidas;
- `Recall` – expressa capacidade de encontrar todas as instâncias relevantes;
- `Harmonic Mean` – média harmónica entre *precision* e *recall*;
- `True Positives Rate` – percentagem de *True Positives Right*;
- `False Positives Rate` – percentagem de *False Positives*;
- `Execution Duration` – tempo de execução da operação de correção do texto.

O *script* desenvolvido utiliza o texto original recebido como *input*, bem escrito em termos ortográficos, par gerar 5 textos distintos com erros. De seguida, utiliza cada modo de operação previamente descrito ("`Manual`", `Aspell`, `Hunspell` e `Symspell`) para corrigir cada um desses textos, calculando de seguida a média das métricas indicadas previamente.

É de se referir que, para gerar os textos com erros, divide-se o texto original e, para cada palavra, calcula-se se a mesma deverá ser alvo de 0, 1 ou 2 alterações, de acordo com probabilidades pré-definidas, decrescentes. De seguida, escolhe-se aleatoriamente que alterações serão efetuadas, sendo as opções disponíveis remoção, transposição, substituição ou inserção de um caractere.

Efetuou-se um teste com esse *script* para um *input* de 13 `KB`, tendo-se obtido os resultados apresentados na tabela que se segue:

Medidas	"Manual"	Aspell	Hunspell	Symspell
True Positives Wrong	133.8	206.6	370.6	152.0
True Positives Right	309.6	230.4	94.2	230.0
True Negatives	1538.0	1432.0	642.4	1172.8
False Positives	135.8	290.4	1363.8	501.0
False Negatives	36.8	43.2	15.4	98.2
Accuracy	0.86	0.75	0.3	0.65
Precision	0.53	0.32	0.05	0.26
Recall	0.64	0.48	0.2	0.48
Harmonic Mean	0.58	0.38	0.08	0.34
True Positives Rate	0.64	0.48	0.2	0.48
False Positives Rate	0.08	0.17	0.68	0.3
Duration (sec)	3.69 + 250.44	14.03	18.67	44.15

Para este *input* em específico, nota-se que o corretor no modo "Manual", apesar de ser muito mais demorado no processamento do texto, obtém melhores valores, de forma geral, para as restantes métricas.

O facto das notícias utilizadas como texto de *input* possuírem estrangeirismos ou palavras gramaticalmente incorretas, entre aspas, terá contribuído para o elevado número de falsos positivos em todos os modos implementados, apesar de, normalmente, se tentar reduzir este valor ao máximo.

Sobressai-se ainda o facto de se terem obtido bastantes mais correções certas (*True Positives Right*) do que erradas (*True Positives Wrong*), quando as palavras são corretamente detetadas como mal escritas, nos modos "Manual" e "Symspell". De facto, é nestes modos que o tempo de execução é superior.

3.5 *Optical Character Recognition* (OCR)

Paralelamente, foi implementado um sistema de reconhecimento de texto em imagens (*Optical Character Recognition*), adaptando a solução presente no site *pyimagesearch.com* [8]. Esta solução foi depois utilizada no contexto do `inoti-make` através da seguinte configuração:

```
FoldersPath:          ['listened_folders/ocr']
NamesRegex:           ['^.*\.jpg$', '^.*\.jpeg$', '^.*\.png$',
                      '^.*\.bmp$']
Recursive:            False
IN_CREATE|IN_MODIFY: [('SHELL_COMMAND', 'cd "$CURRENT_DIR/scripts";
                      python3 ocr.py --east frozen_east_text_detection.pb
                      --image "$WATCH_DIR/$NAME_EXT"
                      > "$WATCH_DIR/$NAME.txt"')]
```

O *EAST Detector* da biblioteca *OpenCV* é uma ferramenta *open-source* que utiliza um modelo de *deep learning* para encontrar as regiões de uma imagem onde ocorrem palavras. Já o *Tesseract* (biblioteca *pytesseract*) é uma ferramenta *open-source* que faz uso de uma rede neuronal recorrente (RNN) para traduzir as palavras que ocorrem numa imagem para texto, suportando 116 línguas.

Estas ferramentas são conjugadas na implementação desenvolvida: o *EAST detector* deteta as regiões das imagens onde ocorrem palavras (correspondentes às quatro coordenadas dos cantos de uma caixa que as delimita), passando-as ao *Tesseract*, que faz a tradução das mesmas para texto.

Após ter sido carregada a imagem de *input* e utilizado o modelo *EAST* pré-treinado (presente no ficheiro *frozen_east_text_detection.pb*), a função *decode_predictions* utiliza o *EAST Detector* para reconhecer as regiões de ocorrência de palavras, devolvendo dois *arrays* correspondentes às coordenadas da caixa delimitadora da palavra na imagem, bem como às probabilidades associadas à presença de texto nessa mesma área.

Depois, são percorridas todas as caixas delimitadoras detetadas na imagem de input e aplicado o *Tesseract* (configurado na linguagem pretendida) a cada uma destas regiões, de forma a produzir uma lista de tuplos correspondentes às quatro coordenadas da caixa delimitadora, seguidas do texto reconhecido.

Por fim, adaptámos a solução original para que devolvesse apenas a totalidade do texto encontrado na imagem. Para tal, houve a necessidade de ordenar as palavras detetadas, de modo a que fossem impressas tendo em consideração a ordem natural do texto numa página (i.e., de cima para baixo e da esquerda

para a direita). Tal foi conseguido através de duas funções auxiliares (*mycmp* e *cmp_to_key*).

No entanto, devido às diferentes alturas das letras que ocorrem na imagem – cuja fonte pode não corresponder à que foi utilizada para treinar a ferramenta de deteção das áreas (*EAST Detector*) –, esta ordenação não foi imediata, visto que havia uma discrepância de algumas unidades entre as alturas das palavras na mesma linha. De forma a contornar esta situação, foi definida uma margem de segurança de 10 carateres, que, no exemplo testado, foi suficiente para obter a correta ordenação das palavras da mesma linha, podendo ser adaptada, porém, a outros casos particulares.

4 Conclusão

Apesar dos problemas previamente identificados, relacionados com a distinção entre o nome das diretorias monitorizadas e os seus *i-nodes*, a ferramenta desenvolvida revelou-se bastante útil, permitindo uma fácil utilização da ferramenta *Unix inotify*, inclusive por quem não possuir qualquer conhecimento sobre a mesma.

A utilidade do `inoti-make` torna-se ainda mais evidente quando a sua automatização é aliada a outras ferramentas como, por exemplo, correção textual ou reconhecimento de texto em imagens.

Em relação ao reconhecimento de texto em imagens, poderia melhorar-se a solução desenvolvida através da integração com os corretores textuais desenvolvidos, bem como outros tipos de ferramentas de pós-processamento.

Anexos

A *Script bash* da organização do código C

```
#!/bin/bash
if [ ! -f ./main.c ]; then
    exit 0
fi

mkdir -p includes
touch -a makefile

# compile
if ! grep -q "compile:" makefile; then
  printf "compile: \$(objects)\n\tgcc -Wall -O3 -o main \$(objects)
   ↪  main.c\n\trm \$(objects)\n\n" >> makefile
fi

# run
if ! grep -q "run:" makefile; then
  printf "run: compile\n\t./main\n\n" >> makefile
fi

objects=""
for file in ./*; do
    if [[ $file == ./!(main).[ch] ]]; then
        mv $file includes
        sed -i -e 's/[ \t]*#[ \t]*include[
         ↪  \t]*"'${file:2}'"/#include "includes\/'${file:2}'"/g'
         ↪  main.c
        if [[ $file == ./!(main).[c] ]]; then
            objects=$objects" "${file:2:-2}".o"
            if ! grep -q "${file:2:-2}.o:" makefile; then
              printf "${file:2:-2}.o: includes/${file:2}\n\tgcc -c
               ↪  -Wall -O3 includes/${file:2}\n\n" >> makefile
            fi
        fi
    fi
done
```

```
32
33  # objects
34  if grep -q "objects =" makefile
35    then
36      sed -i -r "s/(objects[ \t]*=.*)/\1$objects/" makefile
37    else
38      { printf "objects =$objects\n\n"; cat makefile; } >
        ↪  makefile.new
39      mv makefile{.new,}
40  fi
41
42  # clean
43  if ! grep -q "clean:" makefile; then
44    printf "clean: \n\trm -f \$(objects) main\n\n" >> makefile
45  fi
```

Referências

[1] "inotify 0.2.10", PyPI, https://pypi.org/project/inotify/.

[2] "How to Write a Spelling Corrector", Norvig, http://norvig.com/spell-correct.html.

[3] "aspell-python - Python bindings for GNU aspell", GitHub, https://github.com/WojciechMula/aspell-python.

[4] "About PyHunSpell", GitHub, https://github.com/blatinier/pyhunspell.

[5] "symspellpy", GitHub, https://github.com/mammothb/symspellpy.

[6] "Dicionários Natura", Projecto Natura, http://natura.di.uminho.pt/download/TGZ/Dictionaries/wordlists/LATEST/wordlist-ao-latest.txt.xz.

[7] "Linguistic Features", spaCy, https://spacy.io/usage/linguistic-features.

[8] "OpenCV OCR and text recognition with Tesseract", pyimagesearch.com, https://www.pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/.