Teoria Algorítmica da Informação

Armando J. Pinho
Diogo Pratas

Lab Work Nº 1
Finite Context Model & Text Generator

Rui Fernandes, 92952
Pedro Marques, 92926
Inês Leite, 92928

DETI
Universidade de Aveiro
November 8, 2021

1 Introdução

Através deste relatório, explicaremos o processo de criação do nosso programa de geração de texto com base em dados estatísticos recolhidos de um primeiro ficheiro. Para tal fim, tal como era desejado, recorremos à criação de correntes de Markov para a geração de uma tabela final que reunia a probabilidade de ocorrência de certos caracteres tendo como base determinado contexto de dimensão fornecida.

O ficheiro fcm.py contém todas as funções necessárias para a recolha das estatísticas previamente mencionadas enquanto que generator.py está encarregue de usar estes dados e gerar um novo ficheiro cujo texto obedece a esta informação.

2 Finite Context Model

O programa corre-se a partir da linha de comandos com os seguintes argumentos:

1. **file_name**, nome do ficheiro a ler
2. **k**, tamanho do contexto
3. **a**, "alpha", valor entre]0,1]
4. **initialText**, parâmetro opcional

2.1 Recolha de Informação

Através dos argumentos obtemos o nome do ficheiro de texto, ao qual, após a sua leitura, é aplicada a função **get_normalized_string** que irá remover tudo o que não são letras e espaços e meter todo o texto em minúsculo.

Além disto, também é criado um alfabeto com todos os carateres existentes no texto, com a função **get_alphabet**.

Tendo, assim, o texto normalizado é possível criar a tabela, com a função **make_table**. Primeiramente, verifica-se o tamanho que ela irá ocupar, função **get_table_space**, calculado através do **alphabet** (caracteres existentes no texto) e do **k** (tamanho do contexto). Se este valor for inferior a 1024 bits, a estrutura de dados da tabela será uma lista, caso contrário, será um dicionário, por questões de eficiência. Enquanto dicionário cria-se vazio, para ser preenchido mais tarde, a lista já é toda criada, tendo todos os valores a 0, que serão mais tarde atualizados. Esta lista contém **x**k** listas lá dentro, sendo **x** o tamanho do dicionário e **k** o contexto, simbolizando todas as expressões possíveis. Cada lista terá **x** zeros dentro dela, que simbolizam todos as possibilidades de carateres que vêm a seguir.

De seguida, é necessário percorrer o texto e obter todas as expressões de tamanho **k**, inseri-las na tabela: caso a tabela seja um dicionário, cada expressão será a *key* e o seu *value* um dicionário, onde as *keys* desse serão letras e os *values* a quantidade de vezes em que aparecem; se a tabela for uma lista, os valores são apenas atualizados pelas verdadeiras quantidades. Isto é feito pela função **get_expressions_from_data**.

2.2 Estimativa de Probabilidades

A função **get_probability_table** cria a tabela de probabilidades. O processo para o fazer difere ligeiramente consoante o tipo de estrutura que tem, mas baseia-se no mesmo conceito: percorre-se cada elemento da estrutura, em cada um são somados todos os *values* existentes, obtendo assim um total, que será usado para criar uma nova estrutura de dados, igual à usada, mas os valores serão a probabilidade dessa letra aparecer.

A par da criação da tabela de probabilidades, na função **get_probability_table**, calcula-se também a entropia da informação. Esta é calculada pela seguinte função:

$$P(e|c) \approx \frac{N(e|c) + \alpha}{\sum_{s \in \Sigma} N(s|c) + \alpha|\Sigma|},$$

3 Generator

3.1 Geração de Texto

Como foi mencionado previamente, o ficheiro generator.py está encarregue da geração de texto que obedece aos dados estatísticos recolhidos de um primeiro ficheiro.

Executando o script, é necessário fornecer no mínimo 3 argumentos que serão utilizados pelo Finite Context Model: nome do ficheiro inicial, dimensão do contexto (**k**) e o parâmetro **alpha** que será usado no cálculo das probabilidades. Para além disso, podem ainda ser fornecidos 2 parâmetros extra: a dimensão do texto a ser gerado bem como o texto inicial.

A função **generator** está encarregue de executar todos os processos necessários. Assim sendo, é responsável pela geração do finite context model utilizando os métodos de fcm.py bem como pela geração de novo texto utilizando a função **get_next_char**. Finalmente, o novo texto gerado é escrito num ficheiro *output.txt* graças à função **write_to_file**

3.2 Escolha do Caracter Seguinte

Durante a geração de texto, a função **get_next_char** é responsável por, utilizando a dimensão do contexto **k** e as estimativas de probabilidades calculadas, escolher um caracter que sucede o contexto.

Este processo difere ligeiramente consoante o tipo de estrutura da tabela de probabilidades (dicionário de dicionários ou lista de listas) mas partem todos da mesma ideia. Inicialmente, é gerado um numero aleatório **selected_char** entre 0 e 1. De seguida, são criados intervalos utilizando as probabilidades existentes recolhidas e o caracter que segue o contexto é o que gera o intervalo de probabilidades no qual se enquadra **selected_char**.

Por exemplo, sabendo que a probabilidade de 'a' e 'd' seguirem *abc* são de 0.5 e 0.2 respetivamente. Caso **selected_char** esteja entre 0 e 0.5, o caracter seguinte será 'a'. Se, por outro lado, estiver entre 0.5 e 0.7, o caracter seguinte será 'd' e assim sucessivamente para todos os caracteres do alfabeto, dando prioridade aos que têm maior probabilidade.

Quando a tabela de prioridades consiste num dicionario de dicionários, só guardamos a probabilidade dos caracteres que realmente aparecem no texto inicial. No entanto, numa lista de listas, são calculadas as probabilidades de todas as soluções possíveis. Em contextos não registados anteriormente, a probabilidade de cada caracter suceder o mesmo é uniforme. Assim sendo, todos os caracteres têm igual probabilidade de suceder um contexto desconhecido

```
sherlock then nation volve as nextencepter nights the fat pen a mighly takending out you as with and deavy
bareferite throus was nothing of it the well in his holdfath of that the cle have and from hole just which
are was prang on inten dr the which see to had on could shote is long quicked the give unce againts crittle
```

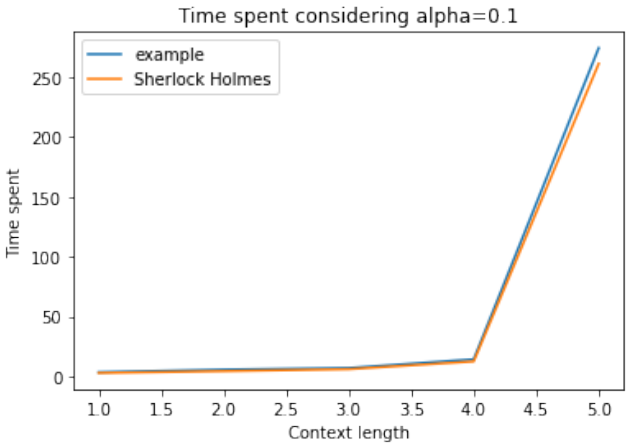
Text generated with data from sherlock.txt, k=3 and alpha = 0.01

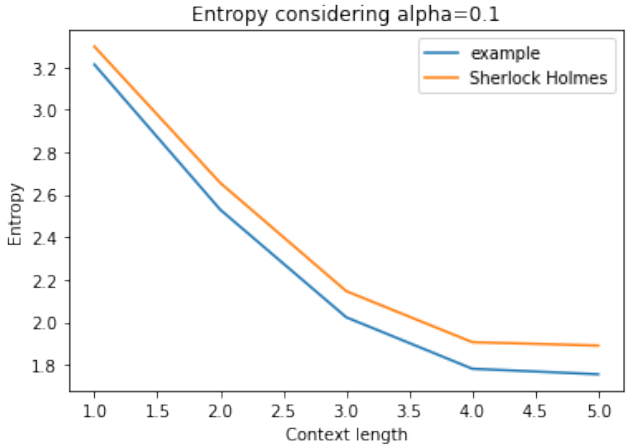
4 Resultados

Utilizando o programa que desenvolvemos, realizá-mos vários testes de modo a compreender a influência da dimensão do contexto bem como do alpha na entropia calculada. Para tal, utilizámos dois textos como ficheiros iniciais para recolha de dados: example.txt fornecido pelos docentes e sherlock.txt que contém a obra Sherlock Holmes por Sir Arthur Conan Doyle.

Uma das principais qualidades denotada foi a criação de palavras reais com o crescimento da **dimensão do contexto (k)** e o decréscimo de **alpha**. Por outro lado, o tamanho do texto inicial influencia também a lógica do gerado pois quanto menor for o primeiro, menor será o número de contextos que encontraremos. Assim, quando ao criar um texto for gerado um contexto desconhecido, o caracter que segue o mesmo é completamente aleatório por falta de dados estatísticos.

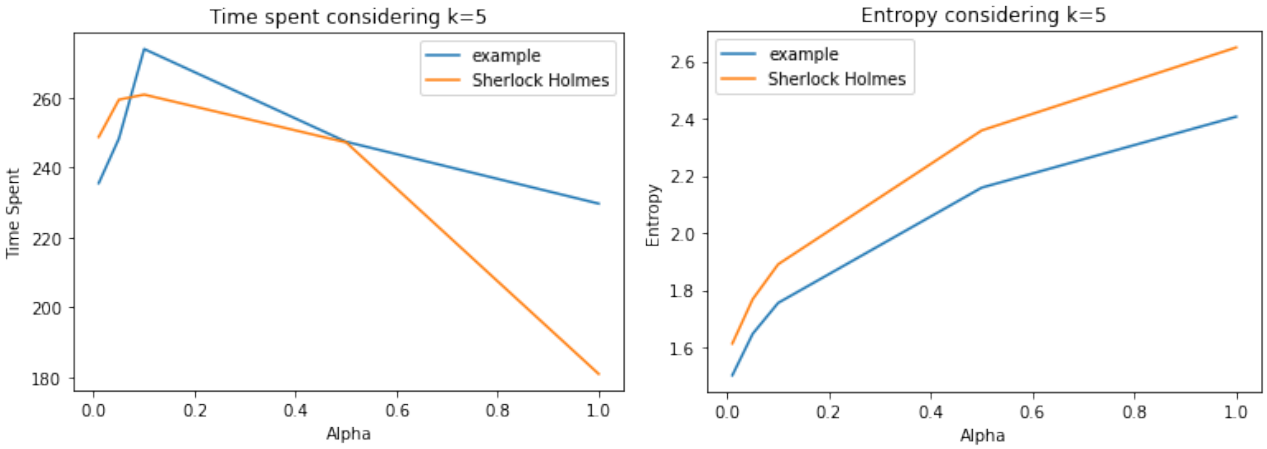
Para todos os testes realizados, a estrutura de dados que utilizá-mos foi uma lista de listas. Assim sendo, é de notar que o tempo de execução do código se mantém mais ou menos constante e baixo com k menor ou igual a 4. No entanto, no inglês, o tamanho médio das palavras é de 5 caracteres o que suspeitamos que justifique o aumento exponencial do tempo de execução para **k** com valor 5. Inversamente, a entropia diminui com o aumento da dimensão do contexto.





Por outro lado, considerando dimensão do contexto constante de 5, pode-se também concluir a influencia da variação do **alpha** tanto no tempo de execução como na entropia final. Dando-se uma relação inversa à variação de **k**, ao diminuir alpha, na tabela de probabilidades, contextos com nehuma ou baixa contagem continuam com relativamente baixa probabilidade de ocorrencia apesar da aplicação do fator de suavização.

Isto acontece pois dando textos de dimensão pequena, como já referido, acaba por não existir exemplares suficientes de cada contexto para formar uma tabela estatistica completamente precisa. É de notar que quando **k** é 1 ou 2, um **alpha** maior já não prejudica os valores de entropia da mesma forma visto que existem muitos mais exemplares de de cada contexto.



Anexamos ainda uma tabela com todos os valores de entropia obtidos na recolha de dados ao variar **k** e **alpha**, encontrando-se o melhor valor obviamnete no canto inferior esquerdo.

k/alpha	0.01	0.05	0.1	0.5	1
1	3.212	3.212	3.212	3.213	3.213
2	2.526	2.527	2.529	2.538	2.547
3	1.998	2.01	2.023	2.089	2.148
4	1.678	1.733	1.781	1.99	2.143
5	1.502	1.647	1.755	2.158	2.406

Figure 4: Entropy values with the example text