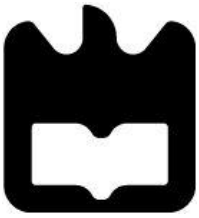
TAI - Assignment I

Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Mestrado em Engenharia Informática



Diogo Carvalho nº92969 – Participação: 1/3

Diogo Cunha nº95278 – Participação: 1/3

Rafael Baptista nº93367 – Participação: 1/3

07 de Novembro de 2021

Índice

1. Introdução

O presente relatório pretende descrever o trabalho realizado na execução do primeiro projeto proposto na unidade curricular de Teoria Algorítmica da Informação.

Neste projeto, fomos desafiados a numa primeira fase efetuar um programa que permita recolher informação estatística de uma fonte de texto através do uso de um modelo de contexto finito, atribuindo probabilidade estimada para a ocorrência dos diferentes símbolos do alfabeto tendo em conta um contexto (sequência de símbolos) prévio. A utilização destas probabilidades estimadas irá nos permitir atingir um dos nossos principais objetivos que é efetuar a previsão do próximo símbolo gerado pela fonte de texto. Para além disso, ainda nesta fase, iremos também efetuar o cálculo da entropia do texto, o que nos permite aferir a quantidade de informação do mesmo.

Posteriormente, numa segunda fase foi-nos proposto desenvolver um segundo programa que efetue a geração de texto com base nos dados estatísticos (probabilidades) obtidos durante a primeira fase do projeto.

2. Metodologia

Para a realização deste projeto, utilizamos a plataforma Github como repositório de código compartilhado. A estrutura do nosso repositório é constituída por uma área com o código desenvolvido, uma pasta com o relatório escrito bem como uma secção com os exemplos de texto que utilizamos para testar o nosso código, estrutura essa que foi recomendada pelos professores.

No que diz respeito à comunicação entre os elementos do grupo, utilizamos a plataforma Discord para efetuar a discussão dos pontos críticos do nosso projeto, facilitando assim a cooperação à distância na realização do mesmo.

Relativamente ao código escrito, o mesmo foi desenvolvido na linguagem de programação Python. A estrutura do código é composta por:

- Programa *fcm.py* que contém a classe Fcm utilizada para representar o modelo de contexto finito.
- Programa *generator.py* que faz uso desta mesma classe para efetuar geração de texto.

3. Implementação

3.1. fcm.py

Nesta seção do nosso relatório iremos descrever a implementação que efetuamos para os nossos programas bem como analisar e justificar as decisões tomadas durante esta mesma fase de implementação.

Posto isto e iniciando pelo programa desenvolvido na primeira fase do projeto (fcm.py), o mesmo tem como objetivo efetuar o cálculo das probabilidades de ocorrência dos diferentes símbolos que compõem o nosso alfabeto de símbolos após um determinado contexto que possui comprimento *k* (parâmetro definido pelo utilizador). Em adição, o programa está implementado para executar também o cálculo da entropia do texto fornecido.

Para correr este programa, são necessários especificar 3 argumentos:

- *-f* : Define o nome do ficheiro que contém o texto para analisar;
- *-k* : Define o número de caracteres dos contextos;
- *-α* : Define o smoothing parameter utilizado no cálculo das probabilidades;

Ao executar o nosso programa, o primeiro passo a ser efetuado é a verificação dos parâmetros fornecidos pelo utilizador. No que diz respeito ao parâmetro *f*, não existe nenhuma verificação direta, apenas na leitura do ficheiro fornecido, caso o mesmo não exista, o utilizador é informado. O parâmetro *k* deve ser um número inteiro, visto que o objetivo do mesmo é definir o comprimento do contexto utilizado pelo modelo de contexto finito. Por sua vez, no parâmetro *α*, é verificado que o mesmo é um float e que se encontra no intervalo]0,1].

De seguida, a leitura do ficheiro de texto é efetuada e é construída uma string única que contém a totalidade do conteúdo do ficheiro.

Por razões de qualidade de código, de forma a garantir uma fácil interpretação e manutenção do mesmo, decidimos criar uma classe Fcm que é utilizada para a implementação do Finite Context Model. O próximo passo do nosso programa consiste então na criação de um objeto desta classe, onde são necessários 3 argumentos para a criação do mesmo:

- a string única que contém o texto que vai está a ser analisado;
- o valor de *k* que corresponde ao tamanho dos contextos;
- o valor do smoothing parameter (*α*).

O objeto ao ser criado, vai guardar algumas variáveis extra para além dos 3 argumentos que recebe, que vão ser necessárias no cálculo da entropia mais concretamente as variáveis:

- alphabet: set que contém todos os símbolos do texto;
- number_of_states: corresponde ao número total de contextos que existem no texto;
- alphabet_size: corresponde ao tamanho do alfabeto;
- model: o modelo que é inicializado como um dicionário vazio;

- final_entropy: variável que guarda o valor da entropia do texto que está a ser analisado. É inicializado a 0;
- state_probabilities: as probabilidades associadas aos caracteres que surgem após cada contexto que é inicializado também como um dicionário vazio;
- context_probabilities: as probabilidades de ocorrência de cada um dos contexto existentes no texto a analisar.

Após a criação do objeto da classe Fcm, invocamos a função create_fcm_model() que pertence à classe Fcm, cujo objetivo é percorrer o texto e preencher o finite context model. Como referido em cima, o modelo é guardado na variável model, a qual é uma instância de um dicionário que vai ter como chaves os contextos, e como valores associados um outro dicionário que, por sua vez, possui como chaves os caracteres, e como valores o número de ocorrências desses caracteres após o contexto correspondente.

É possível entender melhor o armazenamento explicado acima através do seguinte exemplo:

Para o seguinte texto "AABAABB", com k = 2

➢ model = { "AA" : {"B": 2 }, "AB": {"A": 1, “B”: 1}, "BA": {"A": 1} }

Por fim, é efetuada a invocação da função calculate_entropy() que também pertence à classe Fcm. Esta função percorre todos os contextos que foram encontrados no texto, e para todos eles, calcula e armazena as probabilidades de cada símbolo pertencente ao alfabeto ocorrer após este dado contexto tendo em conta o valor do smoothing parameter definido pelo utilizador. Após isso, calcula a entropia dentro de cada contexto e no fim efetua a soma dessa mesma entropia multiplicada pela probabilidade dos diferentes contextos por forma a obter a entropia final do texto. Todos estes cálculos foram efetuados seguindo as fórmulas numéricas encontradas no enunciado de apresentação do projeto.

3.2. generator.py

Relativamente ao programa desenvolvido na segunda fase do projeto (generator.py), o mesmo tem como objetivo com base nas probabilidades de ocorrência dos diferentes símbolos após um determinado contexto calculadas no primeiro programa gerar um novo texto. Posto isto, é natural que este programa faça uso da classe Fcm definida no programa fcm.py.

Para correr este programa, são necessários especificar 4 argumentos:

- *-f* : nome do ficheiro;
- *-k* : tamanho dos contextos;
- *-α* : smoothing parameter;
- *-l* : tamanho do texto a ser gerado;

De forma idêntica ao programa fcm.py, o primeiro passo a executar é a verificação dos parâmetros fornecidos pelo utilizador. A verificação dos parâmetros *f*, *k, α* são idênticas ao programa fcm.py. Relativamente ao parâmetro *l* existe a verificação de que o mesmo é um número inteiro, visto que representa o tamanho do texto a ser gerado.

Os passos seguintes estão implementados de forma semelhante ao programa fcm.py, uma vez que é preciso calcular as probabilidades referidas anteriormente. Para isso, importamos a classe Fcm, criamos um objeto dessa mesma classe e invocamos as funções create_fcm_model() e calculate_entropy() de forma idêntica ao programa fcm.py.

Após isto, começa o processo de geração de texto. Para isso, o primeiro passo é obter as probabilidades dos caracteres de cada contexto que estão guardadas no objeto da classe Fcm criado anteriormente. De seguida, obtemos o primeiro contexto do texto a ser gerado. A nossa escolha, após algumas outras experiências que iremos descrever na seção de análise de resultados, verificámos que escolhendo o primeiro contexto que aparece no ficheiro de texto que foi usado para construir o Finite Context Model, o texto gerado seria mais consistente, estruturado e realista como iremos verificar na seção de resultados.

O algoritmo usado para gerar o texto está implementado de forma iterativa da seguinte forma:

Se o tamanho do texto a gerar for maior ou igual a k, escolher o primeiro contexto do texto que foi analisado.
Senão, obter slice do primeiro contexto que foi analisado.
Percorrer até que o tamanho do texto gerado seja alcançado;
Se o contexto atual existe no nosso Finite Context Model, escolher um caractere de acordo com a probabilidade de cada caractere neste contexto.
Senão:
Se o tamanho do texto a gerar for maior ou igual a k, escolher um contexto de entre os contextos conhecidos segundo as suas probabilidades de ocorrência.
Senão, atribuir um símbolo aleatório.
Atualizar contexto atual para o mais recente.

4. Análise de resultados

Nesta seção do relatório iremos apresentar algumas experiências efetuadas com ambos os programas desenvolvidos e analisar os resultados obtidos a partir das mesmas.

4.1. fcm.py

Relativamente ao programa fcm.py fizemos algumas experiências relacionadas com a variação dos parâmetros *k* e *α* e o cálculo da entropia. Por fim efetuamos também alguns testes de performance.

Começando pela análise da influência dos parâmetros *k* e *α*, na tabela abaixo podemos verificar os resultados obtidos para o valor da entropia ao variar o *α* entre 0.1 e 1 com intervalos de 0.1 e com valores do parâmetro *k* entre 1 e 6.

Os resultados foram obtidos através do texto de exemplo disponibilizado para o trabalho (example.txt).

	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
α = 0.1	3.319	2.558	2.074	1.940	2.092	2.407
α = 0.2	3.320	2.567	2.125	2.086	2.353	2.767
α = 0.3	3.320	2.575	2.167	2.196	2.533	2.996
α = 0.4	3.321	2.582	2.205	2.287	2.672	3.164
α = 0.5	3.321	2.589	2.238	2.365	2.786	3.298
α = 0.6	3.322	2.595	2.269	2.434	2.884	3.408
α = 0.7	3.322	2.601	2.298	2.496	2.968	3.502
α = 0.8	3.323	2.607	2.325	2.552	3.043	3.583
α = 0.9	3.323	2.613	2.351	2.603	3.111	3.655
α = 1.0	3.324	2.619	2.375	2.651	3.172	3.720

Tabela 1 Variação da entropia em função de k e α (example.txt)

Para uma melhor análise dos dados obtidos, apresentamos os dados no gráfico abaixo:

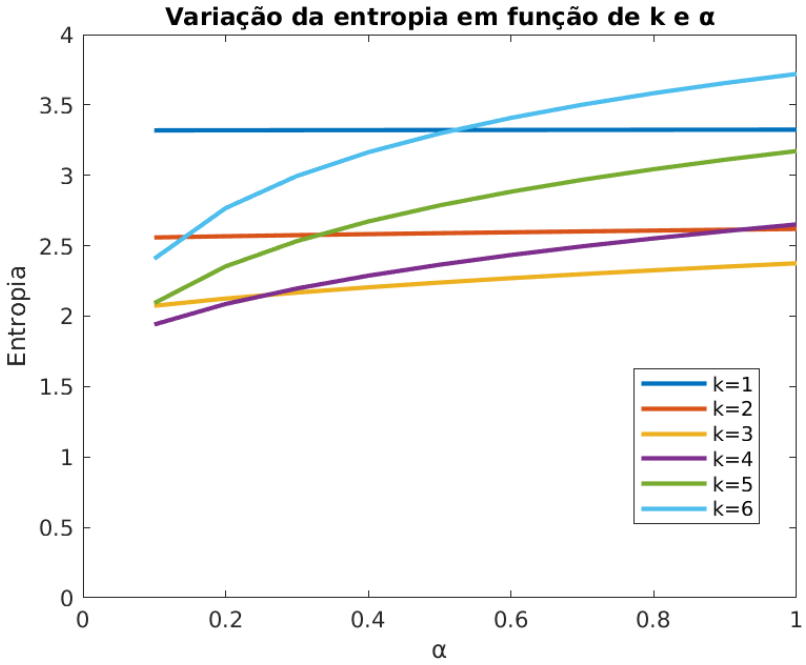


Figure 1 Gráfico da variação da entropia em função de k e α

No eixo das abscissas do gráfico temos os valores de *α* e no das ordenadas os valores da entropia. Cada linha representa um valor de *k* diferente. Da análise dos resultados, podemos verificar que para valores de *k* inferiores existe uma menor variação da entropia ao aumentar o parâmetro *α*. À medida que o valor de *k* atinge valores mais altos (4, 5 e 6), a entropia tem uma maior variação. Isto pode estar relacionado com o facto de, em contextos mais pequenos (exemplo k = 1), existem ocorrências de determinados caracteres após outros que são bastante comuns (por exemplo o símbolo “h” surgir a seguir ao “t”). Ao aumentar o k, aumentamos também a incerteza do símbolo seguinte e por isso existe uma maior variação da entropia final.

No que diz respeito ao *α*, podemos também verificar que o valor de entropia é superior quando o valor de *α* é ele também mais elevado. Isto deve-se ao facto de que, com *α* mais elevados, existe uma maior probabilidade de ocorrência dos símbolos que nunca ocorreram após determinado contexto, aumentando a incerteza da fonte de informação e por sua vez a quantidade de informação é também superior.

Nas duas tabelas seguintes, está representada exatamente a mesma experiência mas desta vez para duas fontes de texto diferentes. Um texto de tamanho médio (medium.txt), e outro de tamanho mais pequeno (small.txt).

Texto tamanho médio:

	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
$\alpha = 0.1$	3.594	2.954	2.810	3.265	4.000	4.692
$\alpha = 0.2$	3.602	3.025	3.079	3.748	4.560	5.218
$\alpha = 0.3$	3.608	3.083	3.272	4.051	4.869	5.479
$\alpha = 0.4$	3.614	3.133	3.426	4.270	5.076	5.643
$\alpha = 0.5$	3.620	3.177	3.555	4.440	5.227	5.575
$\alpha = 0.6$	3.625	3.218	3.665	4.577	5.344	5.842
$\alpha = 0.7$	3.631	3.256	3.762	4.692	5.438	5.909
$\alpha = 0.8$	3.636	3.291	3.849	4.790	5.515	5.963
$\alpha = 0.9$	3.641	3.325	3.927	4.875	5.581	6.007
$\alpha = 1.0$	3.645	3.356	3.998	4.950	5.637	6.044

Tabela 2 Variação da entropia em função de k e α (medium.txt)

Texto tamanho pequeno:

	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
$\alpha = 0.1$	3.541	3.312	3.959	4.513	4.838	5.035
$\alpha = 0.2$	3.670	3.784	4.526	4.998	5.246	5.385
$\alpha = 0.3$	3.772	4.082	4.813	5.214	5.412	5.519
$\alpha = 0.4$	3.859	4.295	4.992	5.338	5.504	5.590
$\alpha = 0.5$	3.935	4.458	5.116	5.421	5.563	5.635
$\alpha = 0.6$	4.002	4.587	5.208	5.480	5.605	5.666
$\alpha = 0.7$	4.063	4.694	5.278	5.524	5.635	5.688
$\alpha = 0.8$	4.119	4.783	5.335	5.558	5.659	5.706
$\alpha = 0.9$	4.171	4.859	5.381	5.586	5.677	5.719
$\alpha = 1.0$	4.218	4.925	5.419	5.609	5.693	5.730

Tabela 3 Variação da entropia em função de k e α (small.txt)

Podemos verificar pela análise das tabelas que as conclusões retiradas acima para o texto example.txt fornecido pelos professores, se repetem agora para fontes de texto de tamanhos diferentes.

De seguida, procurámos avaliar a performance do nosso programa analisando a influência do parâmetro *k* na velocidade de cálculo da entropia para o ficheiro de exemplo fornecido (example.txt) e para outro ficheiro que tem uma dimensão mais reduzida.

Para ambos os ficheiros verificamos que quanto maior o valor de *k* maior o tempo que o programa demora a executar, e consequentemente, mais demorado é o cálculo da entropia. Esse tempo cresce de forma exponencial sendo que para o ficheiro maior com um *k* = 1 o tempo é de 1,84 segundos e para um *k* = 25 atinge os 38,3 segundos. Numa fase inicial, para valores de *k* menores, a variação é muito grande até ao valor de *k* = 20. A partir desta fase, o tempo de execução tem tendência a convergir para um valor de aproximadamente 40 segundos. No texto de tamanho mais reduzido podemos verificar que a curva é idêntica, apenas os valores do tempo alteram.

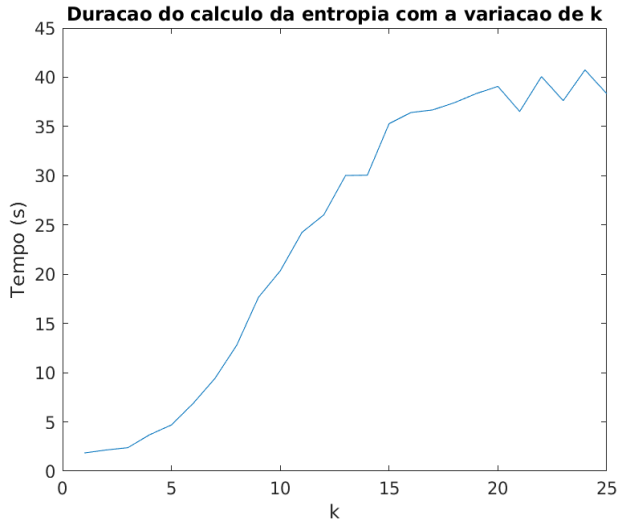


Figure 2 Tempo de execução em função da variação de k (example.txt)

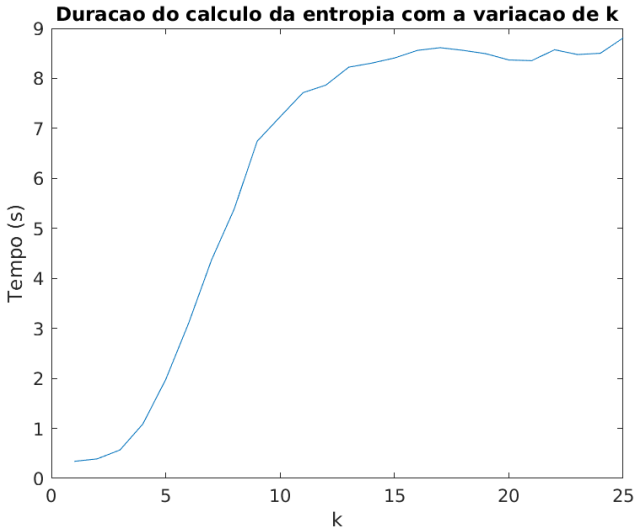


Figure 3 Tempo de execução em função da variação de k (small.txt)

Por último, procurámos avaliar o gasto de memória que o modelo de contexto finito introduziu no nosso programa, bem como a poupança que obtivemos ao criar o modelo de contexto finito apenas com os contextos existentes no texto a analisar, invés de todos os contextos possíveis. Posto isto, contando com um valor de bit counters = 16, seguimos a seguinte fórmula:

$$EQ1: Memória\ (MB)\ = número\ Contextos * len(alfabeto)\ *\ 16\ /\ 8\ /\ 1024\ /\ 1024$$

Considerando o cenário 1 ser o programa guardar todos os contextos possíveis no modelo de contexto finito, e o cenário 2 ser o programa guardar apenas os contextos que efetivamente ocorrem pelo menos uma vez no texto (que é o que está implementado), e considerando o valor de k = 2, k = 3 e k = 4:

	Cenário 1	Cenário 2
k = 2	$Memória\ =\ 0.30033\ MB$	$Memória\ =\ 0.11309\ MB$
k = 3	$Memória\ =\ 16.21829\ MB$	$Memória\ = 0.99422\ MB$
k = 4	$Memória\ =\ 875.78778\ MB$	$Memória\ = 5.14963\ MB$

Tabela 4 Memória ocupada pelo modelo de contexto finito

Como podemos observar pelos resultados demonstrados na tabela acima, os ganhos de memória são enormes e crescem de forma exponencial à medida que o valor de k aumenta também. Mesmo para valores baixos de k já existe uma grande diferença.

De realçar que mesmo o cenário 2 é um limite máximo da memória que o nosso modelo pode gastar, visto que para cada contexto nós guardamos apenas o número de ocorrências para os símbolos que efetivamente ocorrem após aquele contexto ao invés de guardar todos os símbolos para cada contexto existente, que é aquilo que os cálculos consideram. Posto isto, o valor de memória gasto pelo nosso modelo é, geralmente, inferior ao do cenário 2.

4.2. generator.py

Relativamente ao programa generator.py fizemos algumas experiências relacionadas com a melhor forma de iniciar o texto, ou seja, escolha do primeiro contexto, bem como a forma como o programa reage quando durante o processo de geração de texto, existe a criação de um contexto que o modelo de contexto finito desconhece.

A primeira experiência que efetuámos e iremos apresentar permitiu-nos tomar uma importante decisão relativa à implementação deste programa. Para esta experiência, estava em causa a forma como iria ser efetuada a escolha do primeiro contexto na geração de texto, e conseguimos encontrar 3 possíveis soluções.

- 1ª solução - o primeiro contexto do texto que é gerado é o primeiro contexto do texto analisado no momento da construção do modelo;
- 2ª solução - o primeiro contexto do texto que é gerado é um contexto escolhido de forma aleatória de entre os contextos presentes no texto analisado;
- 3ª solução - o primeiro contexto do texto que é gerado é um contexto escolhido de acordo com as probabilidades de ocorrência dos diferentes contextos presentes no texto analisado.

Os testes que executámos para observar o desempenho de cada uma das três alternativas descritas foram realizados com o ficheiro example.txt e com o valor do parâmetro *α* igual a 0.1 e do parâmetro *l* igual a 100. Os mesmos encontram-se apresentados nas tabelas seguintes:

Resultados para a 1ª solução:

k	Texto gerado
2	1:15:20 forty, am my raid, a dan it is madelson jehoson of mart ounts thent shbar stion, and withent
3	1:13 not discipalaces to then again my saith sation. 16:32 like house or suredst of shall sweet eve
4	1:1 and same to set a man when shall give too stransgression from the him. 26:49 the holy give me,

Tabela 5 Resultados obtidos para a 1ª solução

Resultados para a 2ª solução:

k	Texto gerado
2	32 and th the ity, me, is exech dweriat the in the con, en unto ye sighteen beir ors. 25:27 abelt f
3	number, the which knews, and, and not of findness from they of malem, and proceed, said, and of thin
4	eba. 10:12 and said, because twent with know reubenite a copyrig/ders in he stressions: thy servant

Tabela 6 Resultados obtidos para a 2ª solução

Resultados para a 3ª solução:

k	Texto gerado
2	habou heirithe foreball wommay norrake, sair wascons of eve and of wasusbel i wearthy to galves, we
3	ants csing commua9 out of they was try brought, art of hanks a fool. 105:40 but joshut concubindles

4	ness of work. 15:14 and been came uprighters of even unh the hand there altar sister: j the pass o

Tabela 7 Resultados obtidos para a 3ª solução

Analisando os resultados conseguimos concluir que através da 1ª solução, o texto forma um padrão visto que é sempre gerado texto com os primeiros caracteres do texto que foi usado para construir o modelo: no caso de k = 2 começa por "1:"; no caso de k=3 começa por "1:1". As outras duas soluções não se conseguem tirar muitas conclusões entre elas, contudo, comparando com a primeira solução, conseguimos verificar que não acontece o padrão verificado na 1ª solução, pelo que o texto gerado inicialmente é mais diversificado.

Apesar do padrão fixo no início do texto, o que retira a aleatoriedade do texto inicial, optamos pela 1ª solução visto que este mesmo primeiro contexto adiciona uma maior consistência e estrutura a todo o texto e garante um início de texto com sentido, ao contrário das outras soluções.

Uma segunda análise que fizemos está relacionada com o caso de, ao gerar o texto, fosse gerado um contexto que não existe no modelo. Para este problema, encontrámos duas soluções:

- 1ª solução - era escolhido um carácter aleatório do alfabeto;
- 2ª solução - caso o tamanho restante do texto a ser gerado fosse maior ou igual ao tamanho dos contextos, era escolhido um contexto de acordo com as suas probabilidades. Caso contrário seria escolhido um carácter aleatório do alfabeto;

Com o intuito de efetuar a melhor escolha, desenvolvemos alguns testes cujos resultados apresentamos nas seguintes tabelas.

Resultados para a 1ª solução:

	$\alpha = 0.01$	$\alpha = 0.5$
k = 3	1:17 he way: which two aarone, saying of there of	1:10 and the get .r07mg%y- @obmo(9u$zr27rzmp0mc !0.

Tabela 8 Resultados obtidos para a 1ª solução

Resultados para a 2ª solução:

	$\alpha = 0.01$	$\alpha = 0.5$
k = 3	1:11 blood broke a day, the chilah weight of conce	1:16 and dether it pies is sojour hopened who his

Tabela 9 Resultados obtidos para a 2ª solução

Analisando os resultados, conseguimos concluir que para valores de α pequenos, os resultados são semelhantes. Contudo, quando o valor de α é mais elevado, o texto gerado para a 1ª solução começa a ter alguns problemas, sendo bastante aleatório e colocando

caracteres em posições com muito pouco sentido. A partir do contexto “t .” até ao fim do texto, mais nenhum contexto criado é conhecido pelo modelo pelo que caímos num loop de escolhas aleatórias e que não têm em conta os contextos conhecidos pelo modelo. Por outro lado, com a 2ª solução este problema não é tão visível, e o texto aparenta ser mais estruturado do que na 1ª solução. Posto isto, optamos por implementar a 2ª solução.

A última experiência que pretendemos apresentar é relativa à entropia do texto gerado pelo nosso programa. Para isso calculámos a entropia usando o exemplo fornecido pelos professores com valor de k = 3 e com o valor do smoothing parameter = 0.5. De seguida, gerámos 4 textos com tamanhos 1000, 10000, 100000 e 1000000 respetivamente e guardámos cada um num ficheiro. O próximo passo foi calcular a entropia de cada um destes 4 textos gerados e comparar o valor da mesma com o valor da entropia original obtida com o exemplo fornecido pelos professores.

Os valores obtidos estão representados na tabela seguinte:

	l = 1000	*l* = 10000	*l* = 100000	*l* = 1000000	Ficheiro Original
Entropia	5.243	4.983	3.861	2.478	2.239

Tabela 10 Resultados do cálculo da entropia para um texto gerado pelo generator.py

Com estes resultados conseguimos concluir que o valor de entropia tende a convergir para o valor de entropia original à medida que o tamanho do texto gerado aumenta.

5. Conclusão

Com a realização deste projeto conseguimos aprofundar e pôr em prática os nossos conhecimentos relativos à implementação e construção de um Finite Context Model, bem como efetuar a utilização do mesmo para calcular probabilidades de ocorrência dos diferentes símbolos do alfabeto.

Estas estatísticas obtidas através do modelo de contexto finito desempenharam um papel crucial na geração de texto com algum realismo e estrutura bem como no cálculo do valor da entropia associada a um texto.

Outro aspeto bastante importante que gostávamos de realçar foi a necessidade de efetuar várias experiências e procurar testar várias opções por forma a conseguirmos selecionar a melhor solução no que diz respeito a algumas decisões relativas à implementação do algoritmo de geração de texto.

Por fim, consideramos que este projeto foi útil para o desenvolvimento das nossas capacidades individuais como futuros engenheiros informáticos, uma vez que se tratou de um desafio novo que conseguimos realizar com sucesso.