Projeto 3 - IC

Universidade de Aveiro

João Torrinhas, Diogo Torrinhas, Tiago Bastos

Projeto 3 - IC

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

João Torrinhas, Diogo Torrinhas,
Tiago Bastos,
(98435) joao.torrinhas@ua.pt, (98440) diogotorrinhas@ua.pt
(97590) tiagovilar07@ua.pt

January 9, 2023

Contents

Chapter 1

Source Code

O código realizado pode ser encontrado em: https://github.com/diogotorrinhas/IC3

Chapter 2

Exercise 1 - FCM

Neste exercício o objetivo do programa é coletar informações estatísticas sobre textos, usando modelos de contexto finito.

Foi desenvolvida uma classe denominada fcm.cpp que contém vários métodos para auxiliar no processamento de informação, tais como: criar modelos a partir de textos (com ordem de modelo k, e um smoothing parameter), calcular a entropia dos modelos, calcular a distância e entropia estimada entre um texto e certo modelo, carregar um modelo já existente, etc.

Estes modelos (finite-context models/Cadeias de Markov) são úteis em compressão de dados, já que a próxima letra de um certo contexto é influenciada pelas letras anteriores.

Esta estrutura de dados a ser desenvolvida deve armazenar todas os contextos de tamanho k (este parâmetro é fornecido pelo utilizador) assim como, para cada um dos contextos, a probabilidade de cada letra do alfabeto vir a seguir a esse mesmo contexto.

2.1 Criação de Modelos e Cálculo da Entropia

Para a criação de Modelos a partir de determinados textos e consequente representação dos mesmos, foi usada uma estrutura map em que as chaves são do tipo string representando as palavras de ordem k. Cada chave dessas mapeia para uma outra estrutura map que por sua vez tem como chave o caractere(char) seguinte á palavra e este mapeia o seu número de ocurrências no modelo. (palavra de tamanho k -> proximo caracter -> n ocurrências -> outra possibilidade de prox caracter(se houver) -> nmr ocurrencias -> etc)

```
map<string, map<char, int>> model;
```

Figure 2.1: Estrutura Map

Para guardar os modelos em memória foi desenvolvido um método denom-

inado saveModel(), em que tem como parâmetros de entrada, uma estrutura map (como a da figura acima), e um ficheiro .txt (que vai ser usado para criar o modelo). Depois de criado o modelo, o mesmo vai ser guardado num ficheiro denominado por "Model" + ficheiro.txt passado como parâmetro de entrada + ".txt"

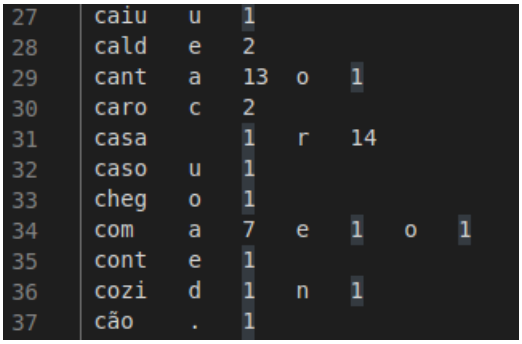

Figure 2.2: Exemplo de conteúdo de modelo criado (k=5, alfa=0.01) a partir de um ficheiro com história da carochinha.

Para obter a entropia de cada modelo criado foi desenvolvido um método denominado CalculateModelEntropy(), em que tem como parâmetro de entrada o modelo calculado previamente.

A entropia do modelo é calculada pelo somatório da probabilidade de cada contexto * a entropia desse mesmo contexto:

$$H = \sum_{i=1}^{N} P(S_i)H(S_i),$$

A probabilidade do contexto é dada pela razão entre o número de ocurrências total desse contexto (número de vezes que esse contexto aparece no texto (com um caracter ou espaço a seguir))(número de casos favoráveis) sobre o somatório do número total existente de contextos (com carácters ou espaços a seguir)(número de casos possíveis).

$$\boldsymbol{P(A) = \frac{número\ de\ casos\ favoráveis}{total\ de\ casos\ possíveis}}$$

A entropia de cada contexto é calculada com a seguinte fórmula (a 'prob' é a probabilidade de cada caracter no contexto/palavra):

```
PalavraEntropia -= prob * log2(prob);
```

Relativamente á entropia e ao seu cálculo, a aumentarmos a ordem k do modelo, significa que vai se ter um tamanho de palavra/contexto maior a colocar no modelo. Quanto maior for o número de caracteres de uma determinada palavra/contexto, a variadade de caracteres possíveis que podem aparecer a seguir diminui consideravelmente. Por exemplo , se k for 7, o número de caracteres que podem aparecer a seguir á palavra "padeiro" é muito reduzida (ou aparece um 's' ou um caractere que não pertence ao alfabeto, etc) comparativamente a uma palavra de ordem 3 ou 4, em que já existe muita mais variedade de caracteres possiveis para aparecerem a seguir. Todos estes aspetos irão influenciar o cálculo da entropia de determinado modelo.

Como o aumento da ordem k de um modelo faz com qe diminua a entropia desse mesmo modelo (tirado a partir de um mesmo texto), pois ao aumentar o tamanho das palavras/contextos, as possibilidades de vários caracteres diferentes seguintes a essa mesma palavra/contexto diminui razoavelmente, o que, deduzindo das fórmulas acimas mencionadas, faz com que a entropia do modelo diminua. Segue dois exemplos do resultado da entropia, um para k = 5, outro para k = 20 (ambos com alfa = 0.01):

```
osboxes@osboxes:~/Desktop/IC_PROJETO3$ ./modelEntropy PT.utf8
Valor de K: 5
Valor de alfa: 0.01
Processing for order 5 ...
ModelPT.txt
Model Entropy: 1.30924
```

Figure 2.3: K=5

```
osboxes@osboxes:~/Desktop/IC_PROJETO3$ ./modelEntropy PT.utf8
Valor de K: 20
Valor de alfa: 0.01
Processing for order 20 ...
ModelPT.txt
Model Entropy: 0.0792948
```

Figure 2.4: K=20

Como podemos verificar (e como explicado acima), o entropia do modelo para um mesmo texto, é menor para K=20 do que para K=5 (pois quanto maior for o K, menor será a entropia.

Se o objetivo for comprimir dados, o valor da ordem k do modelo deve de ser aumentado. Pois quanto menor for a entropia, menos bits serão precisos por

símbolo, o que consequentemente faz com que se obtenha uma maior compressão.

2.2 Calcular Distância e Entropia estimada de um texto em relação a um Modelo

Para calcular a entropia estimada de um texto foi desenvolvida uma função denominada estimateDistanceEntropy(), que tem como parâmetros de entrada um dado modelo, e um ficheiro de texto.

O texto de entrada que vai ser usado para calcular a entropia estimada e a distância estimada, relativamente a um modelo de entrada, vai ser lido sequencialmente e vai ser guardado todos os contextos de ordem k e o caracter seguinte.

Posto isto, a distância de um modelo a um texto de entrada, é calculada pela seguinte fórmula:

```
sumH += -log2((nocurrencias + alfa) / (totalOcurrencias + (alfa * 27)));  // 27 = tamanho do alfabeto
```

'nocurrências' é o número de ocorrências do caracter lido (que vem a seguir ao respetivo contexto) para o respetivo contexto e 'totalOcurrencias' é o número total de entradas para o contexto no respetivo modelo (ou total de ocurrências de todos os caracteres seguintes possíveis para esse mesmo contexto). O 'alfa' é usado para quando o 'nocurrências' for 0, nunca acontecer log(0)=-infinito.

Quanto maior for o número de entradas falhadas no modelo para o caracter seguinte lido a partir texto de entrada para um determinado contexto/palavra, maior será a distância.

Para calcular a entropia estimada, foi usada a seguinte fórmula ('count' é o número de caracteres lidos a partir do ficheiro de entrada):

```
// guarda entropia estimada
estimatedEntropy = sumH / count;
```

Exemplo:

```
osboxes@osboxes:~/Desktop/IC_PROJETO3$ ./testefcm ModelPT.txt TextoSimplesTeste.
txt
Load de um model existente? (y or n): y
Valor de K: 5
Valor de alfa: 0.1
trying to process: ModelPT.txt
entropia do modelo: 1.30924
Distancia estimada: 9861.8
Entropia estimada: 3.77847
```

Figure 2.5: Entre um Modelo de um ficheiro PT.utf8 e um texto com uma história portuguesa

```
osboxes@osboxes:~/Desktop/IC_PROJETO3$ ./testefcm ModelENG.txt TextoSimplesTeste
.txt
Load de um model existente? (y or n): y
Valor de K: 5
Valor de alfa: 0.1
trying to process: ModelENG.txt
entropia do modelo: 1.27636
Distancia estimada: 12963.1
Entropia estimada: 4.96672
```

Figure 2.6: Entre um Modelo de um ficheiro ENG.utf8 e um texto com uma história portuguesa

Como podemos verificar, a distância estimada é menor entre um simples texto em português e um modelPT (retirado a partir do ficheiro PT.utf8 que apenas contém palavras/frases portuguesas), em relação á distância estimada entre um simples texto em português e um modelENG (retirado a partir do ficheiro ENG.utf8 que apenas contém palavras/frases inglesas), o que comprova que houve muitas mais entradas acertadas do que falhadas entre esse simples texto em português e o modelPT em relação ao modelENG, o que faz sentido pois trata-se de um simples texto em português e não em inglês.

2.3 Comandos de compilação

Para criar modelo e calcular entropia (a partir de do PT.utf8):

- g++ modelEntropy.cpp -o modelEntropy
- ./modelEntropy PT.utf8

Para calcular distância estimada e entropia estimada entre um modelo ModelPT.txt (anteriormente criado) e um texto com uma história portuguesa:

- g++ testefcm.cpp -o testefcm
- ./testefcm ModelPT.txt TextoSimplesTeste.txt

Chapter 3

Lang

3.1 Introdução e implementação

Para este exercício foi necessário criar um programa, *lang.cpp*, que tem como parâmetros de entrada dois ficheiros: um com um texto que representa, por exemplo, uma determinada linguagem e outro que irá ser o texto que vai ser analisado.

Como primeiro parâmetro de entrada podemos ter um modelo existente, derivado de um ficheiro de texto, ou um ficheiro de texto onde posteriormente vai ser calculado um modelo a partir desse texto, com recurso à class *fcm*. Tendo o modelo, agora é preciso calcular o número estimado de bits necessários para comprimir o segundo ficheiro usando a função *estimateDistanceEntropy* da classe *fcm* e o respetivo modelo. Tendo efetuado o cálculo apenas é necessário buscar o valor que ficou armazenado no atributo *distance* da class *fcm* e obtemos o valor da estimativa.

3.2 Resultados e Compilação

Para compilar este ficheiro é necessário executar a seguinte linha no terminal:

> g++ lang.cpp -o lang

Após introduzido este comando vai ser criado um executável, *lang*, que vai servir para correr o programa. Para correr o mesmo basta executar no terminal, dentro do diretório do programa, a seguinte linha:

> ./lang <model/txt> <txt to be compared>

Ao correr o programa o utilizador pode selecionar se quer introduzir um modelo já existente ou introduzir um texto para posteriormente ser cálculado um modelo para o mesmo, tal como mostra a figura abaixo.

Figure 3.1: Escolha entre utilizar um modelo existente ou não

Caso o utilizador decida introduzir um texto, para posteriormente ser calculado um modelo a partir do 0, o prompt que vai aparecer ao utilizador vai ser algo parecido à figura de baixo.

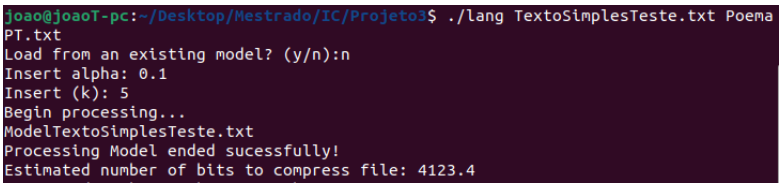

Figure 3.2: Criar um modelo e respetivo resultado

É esperado que para um certo modelo de uma determinada linguagem, que a estimativa do número de bits para comprimir um determinado ficheiro seja menor em linguagens muito parecidas com aquela que foi construído o modelo inicial. Posto isto, se obtivermos uma estimativa baixa podemos concluir que existe uma grande probabilidade de o texto estar escrito na mesma linguagem representada pelo modelo.

No entanto, não podemos confiar a 100% no valor da estimativa uma vez que, para ficheiros de texto muito pequenos é normal que o valor da estimativa seja relativamente baixo.

Posto isto, foram feitos vários testes a partir de um modelo de texto em português, ***ModelPT.txt***:

Resultados		
Ficheiros para compressão	Estimativa (Nmr bits para comprimir)	Linguagem (ficheiro a comprimir)
PT.utf8	7.08673e+06	Português
OsMaias.txt	4.68019e+06	Português
ESP.utf8	1.34801e+07	Espanhol
ENG.utf8	1.73196e+07	Inglês
ARAB.utf8	3.56576e+07	Árabe
RUS.utf8	3.56521e+07	Russo

Através dos valores da tabela dá para comprovar o que era esperado do programa. Como estámos a usar um ModelPT.txt que foi extraído de um ficheiro em português (PT.utf8), quanto mais parecida for a linguagem de x texto a ser comprimido à lingua portuguesa, menos bits irão ser necessários para comprimir o texto. Por exemplo, para comprimir um texto (OsMaias.txt) em português usando o ModelPT.txt, é preciso $4{,}68019*10^6$ bits, enquanto que se for pra comprimir um texto árabe ou russo usando este modelo, irão ser precisos muitos mais bits ($3{,}56576*10^7$), derivado a ser um texto com uma linguagem muito diferente.

Chapter 4

FindLang

4.1 Introdução e implementação

O objetivo deste exercício é criar uma programa para reconhecer linguagens. O programa é baseado na comparação da distância entre um texto qualquer e os modelos que representam cada linguagem. Quanto menor for a distância maior é a probabilidade de o texto dado estar escrito na linguagem do modelo. Tal como referido no exercício anterior o utilizador tem a opção de escolher um modelo existente ou não. Para cada modelo, introduzido no prompt, é feita uma comparação dele mesmo com o ficheiro de texto a ser analisado obtendo uma estimativa no número de bits necessários para comprimir o ficheiro de texto em análise.

4.2 Resultados e Compilação

Para compilar este ficheiro é necessário executar a seguinte linha no terminal e após introduzido este comando vai ser criado um executável, *findlang*, que vai servir para correr o programa:

$$> \text{g++ findlang.cpp -o findlang}$$

Inicialmente, para executar o programa, pensámos em usar um ficheiro de texto, *Models.txt*, que tinha 4 modelos predefinidos para 4 linguagens (Espanhol, Inglês, Português e Russo) e o utilizador poderia apenas usar esses 4 modelos ou adicionar mais algum modelo a esses 4 existentes, para reconhecer a linguagens de um texto. Para executar seria algo do género:

$$> \text{./findlang Models.txt TextoSimplesTeste.txt}$$

No entanto, rapidamente vimos que faria mais sentido ser o utilizador a escolher os modelos que queria utilizar em vez de já ter modelos predefinidos. Posto

isto, para correr o programa, basta executar, por exemplo, no terminal, dentro do diretório do programa, a seguinte linha:

> ./findlang BOSN.utf8 ENG.utf8 ESP.utf8 FI.utf8 GER.utf8 IT.utf8 LT.utf8 PT.utf8 RU.utf8 SWE.utf8 TURK.utf8 TextoSimplesTeste.txt

Para testar este programa usamos o ficheiro TextoSimplesTeste.txt para ver se o programa adivinha em que linguagem o texto está escrito. Em baixo está uma tabela com os resultados obtidos para vários modelos de linguagem.

Resultados		
Ficheiros	Distância Estimada	Linguagem
BOSN.utf8	1.26688e+06	Bósnio
ENG.utf8	1.29631e+06	Inglês
ESP.utf8	1.20951e+06	Espanhol
FI.utf8	1.26719e+06	Finlandês
GER.utf8	1.26411e+06	Alemão
IT.utf8	1.29146e+06	Italiano
LT.utf8	1.25194e+06	Lituano
PT.utf8	9.8618e+05	Português
RU.utf8	1.24550e+06	Russo
SWE.utf8	1.26017e+06	Sueco
TURK.utf8	1.26477e+06	Turco

Através destes valores da tabela dá para comprovar o que era esperado do programa. O programa adivinhou que o texto estava escrito em português pois, se olharmos para os valores da tabela, a distância estimada entre o texto e o modelo português tem o valor menor relativamente aos outros modelos. Podemos então concluir que o programa está funcional.

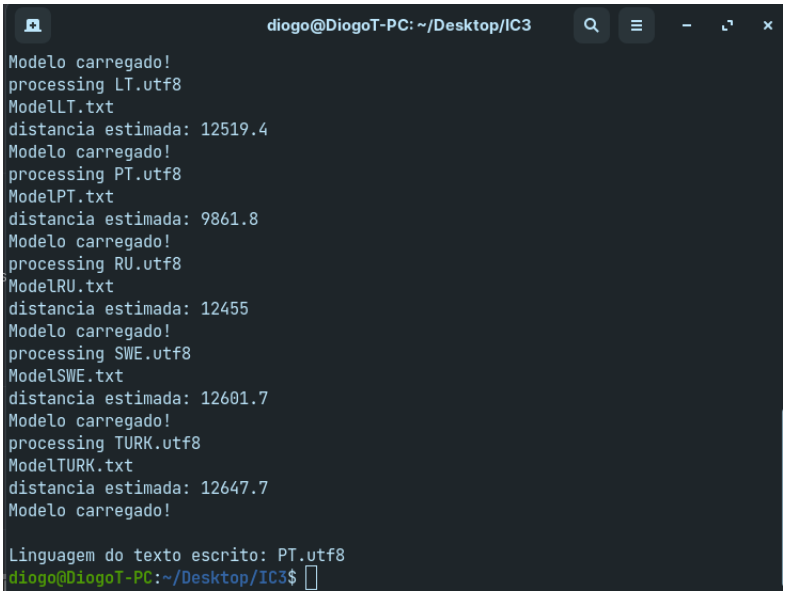

Figure 4.1: Resultado no terminal

Chapter 5

LocateLang

5.1 Introdução e implementação

O objetivo deste exercício era criar um programa, *locatelang.cpp*, que processa textos que contêm segmentos de várias linguas. O resultado do programa é o caracter inicial de cada segmento e a respetiva lingua desse segmento.

O valor do tamanho de cada segmento irá ser definido pelo utilizador, através do prompt. Em seguida, o programa vai ler todos os caracteres do ficheiro, por partes, até ao final do mesmo, ou seja, imaginando que o utilizador seleciona o tamanho do segmento como sendo de 3000, o programa vai ler o ficheiro, por partes, da seguinte maneira: 0-2999, 3000-5999, 6000-8999 e assim sucessivamente até ao fim do ficheiro, onde esses intervalos vão corresponder aos segmentos do ficheiro de texto.

Por fim, cada segmento vai ser comparado com os modelos introduzidos pelo utilizador ou, caso o utilizador queira carregar modelos do 0, basta introduzir os ficheiros de texto e posteriormente vai ser criado um modelo para esses ficheiros que vai servir para a comparação com cada segmento. Feita a comparação, vai ser obtido uma estimativa do número de bits necessários para comprimir esse segmento. Em seguida, guardamos o valor da menor estimativa e vemos qual é o modelo que corresponde a essa estimativa, podendo afirmar que a linguagem desse segmento é a mesma que a linguagem desse modelo.

5.2 Resultados e Compilação

Para compilar este ficheiro é necessário executar a seguinte linha no terminal e após introduzido este comando vai ser criado um executável, *locatelang*, que vai servir para correr o programa:

$>$ g++ locatelang.cpp -o locatelang

A maneira de correr o progama é idêntica à do capítulo anterior, onde o utilizador passa, em primeiro, os modelos ou ficheiros de texto que quer comparar com o ficheiro de texto, passado em último lugar, que possui várias linguagens. Em baixo está um exemplo para correr o programa:

> ./locatelang ModelENG.txt ModelPT.txt ModelESP.txt ModelRU.txt ModelARAB.txt VariasLinguas.txt

Obtendo os seguintes resultados:

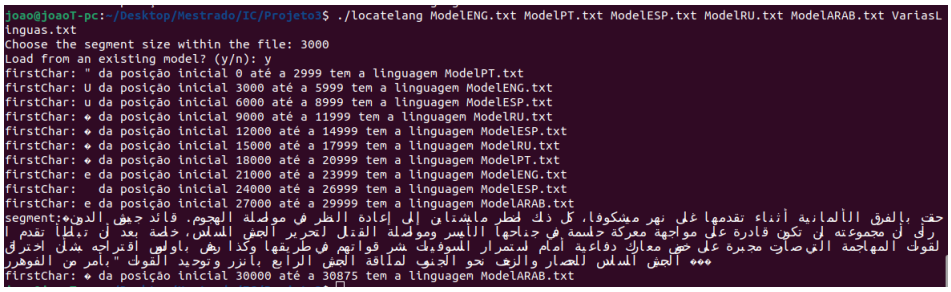

Figure 5.1: Resultado do exercício 4

Como é possível verificar pelo terminal, evidenciado na figura de cima, temos o primeiro caracter de cada segmento, o tamanho dele e a respetiva linguagem associada a ele. Além disso é possível comprovar que há uma certa precisão na identificação da lingua de cada segmento porque na última iteração, antes de ler o ficheiro por completo, vêmos o print do segmento e é notável que o mesmo se encontra em árabe.

De salientar que dependendo do tamanho do segmento é possível o mesmo estar escrito em mais do que uma lingua. Nesses casos, o resultado será a lingua que tiver mais texto escrito nesse segmento. Imaginando que, para um segmento, uma pequena porção de texto está escrito em Português e mais de metade está escrito em Espanhol, o programa iria retornar a lingua Espanhola para esse segmento.

Chapter 6

Authors' Contribution

Todos participaram de forma igual na divisão e elaboração deste projeto, pelo que a percentagem de contribuição de cada aluno fica:

- João Torrinhas - 33,33%
- Diogo Torrinhas - 33,33%
- Tiago Bastos - 33,33%

Bibliography

[1] Armando J. Pinho, Some Notes For the Course Information and Coding Universidade de Aveiro, 2022.