Universidade de Aveiro

# Informação e Codificação
# Projeto 3


Ana Filipe (93350), João Gameiro (93097), Pedro Abreu (93240)

Departamento de Electrónica, Telecomunicações e Informática

4 de fevereiro de 2022

# Conteúdo

# Lista de Figuras

# Capítulo 1

# Introdução

O presente relatório visa descrever a resolução do Projecto 3 desenvolvido no âmbito da unidade curricular de Informação e Codificação.

## 1.1 Compilação

O código desenvolvido para o projeto encontra-se disponível em: https://github.com/Torrakanor611/IC_project3.

```
#Estando no directório base do projeto
#Compilar o código
make all

#Para executar o código correr por exemplo
./bin/lang <file-name> <file-name>

#Gerar documentação doxygen
doxygen doxyconfig
```

Os diretório do projeto incluem:

- diretório *src* -> diretório com o código fonte dos módulos fcm, lang, findlang
- diretório *test* -> ficheiros cpp com *scripts* de testes aos módulos desenvolvidos.
- diretório *models* -> com ficheiros de texto usados para construir os modelos de linguagens.
- diretório *bin* -> com os binários pós compilação das entidades *top-level* do projecto, fcm, lang e findlang

# Capítulo 2

# Fcm

O objectivo programa era fornecer ferramentas que permitissem recolher informações estatisticas sobre dados de entrada. Sejam dados um ficheiro de entrada, a ordem do modelo e o *smoothing parameter* este devia conseguir estimar a entropia do texto com base num modelo de contextos finito previamente calculado.

Foi tomada a decisão de criar uma classe *fcm* para auxiliar no processamento de informação. Esta classe através dos seus métodos processa os textos de entrada e carrega os modelos com a informação retirada dos textos.

Os modelos em que esta classe está baseada são intitulados ***finite-context models*** ou cadeias de Markov em tempo discreto. Nestes tipos de modelos, a próxima letra numa certa palavra e bastante influenciada pelas letras precedentes, facto que os torna bastante úteis em compressão de dados. [1]

Deste modo a ideia base para a criação da estrutura de dados, é que esta deveria armazenar todas as palavras de tamanho $k$ (fornecido pelo utilizador) assim como, para cada palavra a probabilidade de cada letra do alfabeto vir a seguir a essa mesma palavra.

## 2.1 Representação do Modelo

Não só o modelo finito para um determinado tamanho $k$ poder ser muito extenso e ocupar muita memória em *run-time* como também por motivos motivos de performance este não pode ser represetnado por um *array* de *arrays*. Assim, tabela de ocurrências de um determinado modelo é representada por:

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

Uma estrutura map em que as *keys* são string que representam o contexto de ordem k. Cada chave mapeia para outra estrutura map que por sua vez tem como key o *char* seguinte ao contexto e este mapeia o número de occurências deste no modelo.

Para carregar os modelos em memória criamos a função de acordo com:

```
fcm f(5, 0.1);
map<string, map<char, int>> model;

// carregar o modelo em memória
f.loadModel(model, "ficheiro.txt");
```

E para obter a entropia deste modelo:

```
...
// calcular entropia do modelo
f.calculateModelEntropy(model);
cout << "entropia do modelo: " << f.modelEntropy << endl;
```

Declaramos a estrutura de dados(*model*) na *top-level entity* para poder, chamando a a função *loadModel()* com múltiplos ficheiros de entrada. Assim permite carregar diferentes ficheiros para o mesmo modelo. Útil para quando p.e temos vários textos que definem a mesma linguagem.

A entropia do modelo é calculada de acordo com:

$$H_{model} = \sum_{ctx} P_{ctx} \cdot H_{ctx}$$

Sendo a probabilidade do contexto a probabilidade associado à frequência relativa que se pode deduzir pela razão entre o somatório do número de ocurrências de caracteres nesse contexto, $Total_{ctx}$, pelo total do número de ocurrência entre todos os contextos, $\sum_i Total_i$:

$$P_{ctx} = \frac{Total_{ctx}}{\sum_i Total_i}$$

E a entropia de cada contexto calculada com a soma da entropia da probabilidade da frequência relativa para cada ocurrência de carcter, $P_i$:

$$H_{ctx} = \sum_i -P_i \cdot log(P_i)$$

Na Figura 2.1 podemos observar a entropia de um modelo obtido a partir do ficheiro *PT.utf8* para diferentes valores de ordem do modelo. Analisando a figura vemos que à medida que aumentamos o valor da ordem diminuímos a entropia do modelo.

Isto acontece pois ao aumentar a ordem do modelo, significa que vamos considerar um tamanho maior de palavra a colocar no modelo. Quando se generaliza mais o número de caracteres de uma dada *string*, a variedade de letras que podem aparecer a seguir também reduz. Considerando um exemplo em concreto em língua portuguesa, para ordem 9, o número de letras que podem aparecer a seguir à palavra "sapateiro"é muito reduzido, pois na grande maioria das vezes ou irá aparecer um 's' ou um carácter especial (espaço branco ou

sinal de pontuação). No entanto a seguir a um a (ordem 1) pode aparecer uma grande variedade de letras. Esta situação irá reduzir ou aumentar, de acordo com o caso em questão, a probabilidade calculada, o que por sua vez irá consequentemente influenciar o cálculo da entropia.

Ou seja a podemos concluir que o aumento da ordem de um modelo reduz a entropia, pois ao generalizar mais o tamanho das *strings* estamos a reduzir as diferentes possibilidades de ocorrência de cada letra a seguir a essa palavra o que consequentemente contribui para a redução da entropia.

Numa perspectiva de compressão de dados, se fosse pretendido obter uma taxa de compressão bastante elevada, deveríamos aumentar o valor da ordem do modelo. Isto pois quanto menor a entropia associada, significa que são necessários menos bits por símbolo o que por sua vez permite atingir uma maior taxa de compressão.[2]

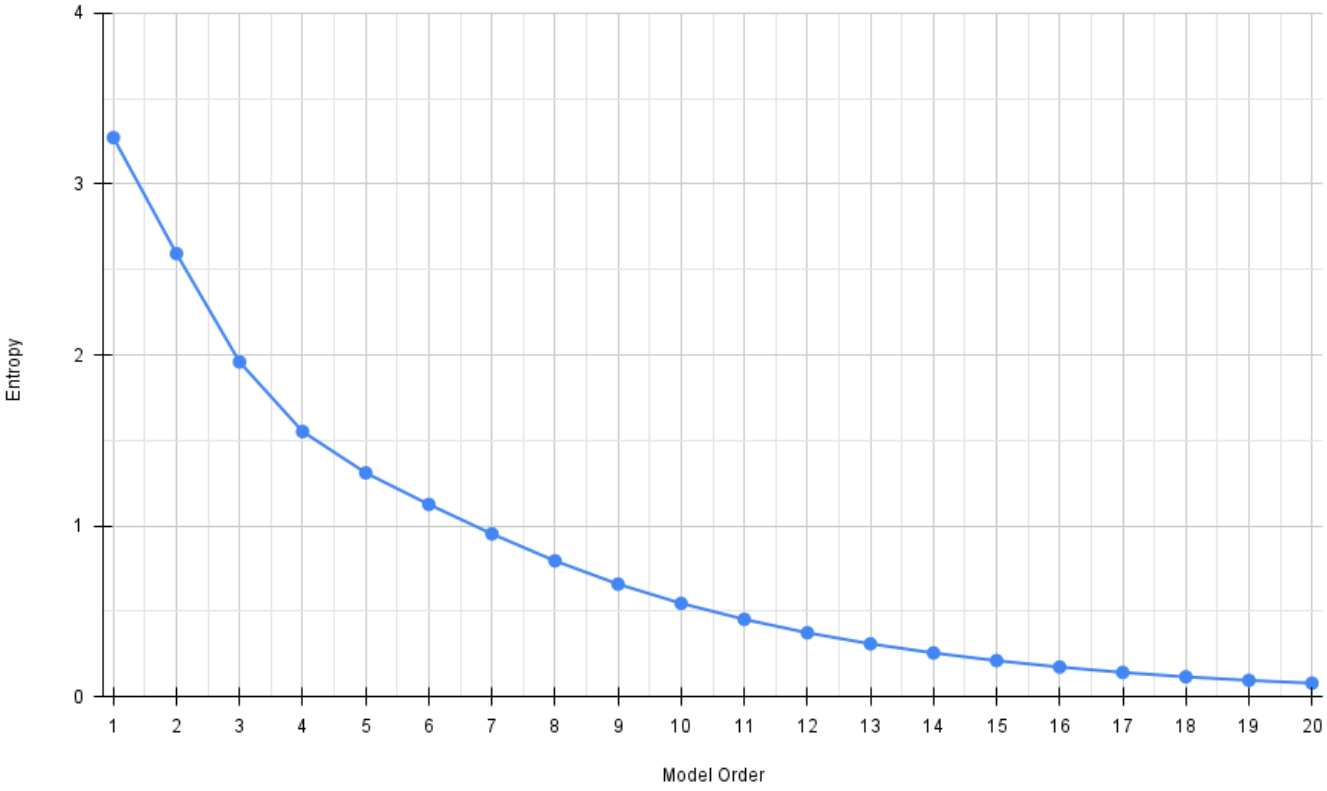


**Figure 2.1:** Evolução da entropia de um modelo para vários valores da ordem

## 2.2 Estimar a entropia de um texto

Estimar a entropia de um texto será:

```
// estimar entropia de um texto de entrada
f.estimate(model, "ficheiro.txt");
cout << "entropia estimada: " << f.estimatedEntropy << endl;
```

e para calcular a distancia do modelo ao texto de entrada:

```
...
cout << "distancia estimada: " << f.distance << endl;
```

O texto de entrada é interpretado do ínicio como uma stream de caracteres(foi usado a classe de I/O ifstream), pelo que, sequencialmente, é lido o contexto de ordem k e o carater seguinte. Nestas condições, a distância de um modelo a um texto de entrada, D, é calculada da soma consecutiva do $-log()$ da frequencia relativa da ocurrencia do caracter para um determinado contexto:

$$D = \sum -log\left(\frac{n}{n_t}\right)$$

Sendo n, o número de ocorrências do caracter interpretado para o contexto e nt, o número total de ocurrências de qualquer caracter para o contexto ou o número de vezes que o contexto acontece no modelo.

No entanto surge um problema prático. Considerando que determinado caracter para determinado contexto não ter *entry* na tabela, o numerador, n, pode ser 0. Para resolver este problema é preciso introduzir um smoothing parameter, $\alpha$ para evitar fazer $log(0) = -\infty$. Pelo que a formula fica:

$$D = \sum -log\left(\frac{n + \alpha}{n_t + \alpha \cdot |A|}\right)$$

sendo $|A|$, o tamanho do alfabeto.

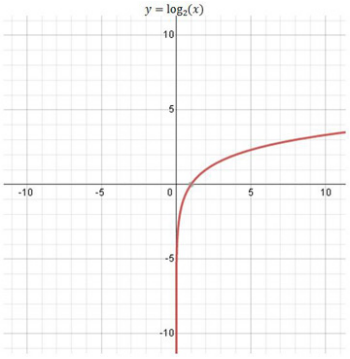


**Figure 2.2:** Função logaritmo

Nas condições supracitadas,

$$\frac{n + \alpha}{n_t + \alpha \cdot |A|} \in ]0, 1[$$

Assim a distancia, D é tão maior quanto maior for o número de entrys falhadas no modelo para o caracter verificado no texto de entrada para o determinado contexto. Nestes casos o $-log(O^+)$ é um número positivo relativamente grande para o calculo da distância.

Isto significa que o modelo e o texto de entrada não são próximos. O modelo define uma má aproximação ao texto de entrada

Em sentido inverso, um texto de entrada aproxima-se do modelo, pelo número de *entrys* verificadas para um caracter e pelo peso destas na frequência relativa $\frac{n}{n_t}$. Nestes casos o $-log(1^-) = 0^-$ contribui um número pequeno proximo de 0. Assim, O modelo define uma boa aproximação ao texto de entrada.

Para o cálculo da entropia foi utilizada a fórmula:

$$H_{estimada} = \frac{D}{n}$$

Em que n representa o número de caracteres do ficheiro de entrada. Isto justifica-se pela definição de entropia ser a quantidade de informação média por símbolo. Por isso surgiu a necessidade de dividir a distância obtida pelo número de símbolos presentes no ficheiro.

# Capítulo 3

# Lang

O programa *lang.cpp* recebe como argumentos dois ficheiros de entrada:

- Um que contém um texto representativo (de uma qualquer linguagem por exemplo);
- E outro que irá ser analisado, de modo a ser calculada uma estimativa do número de bits necessários para comprimir usando o modelo calculado a partir do primeiro ficheiro.

```
#Sintaxe de utilização do programa lang
./bin/lang <textfile> <textfile>
```

O modelo a ser carregado pode ser um ficheiro de texto já existente ou num que vai ser carregado a partir do zero. Se pretender carregar um modelo existente deve especificar o ficheiro que contém o conteúdo do modelo já treinado e depois seleccionar a opção *y*.

Tendo ambos os ficheiros, é construído um modelo a partir do processamento do primeiro texto com recurso à classe *fcm*, caso seja necessário. Após o cálculo do modelo falta apenas estimar o número de bits necessários para comprimir o segundo ficheiro usando o modelo calculado. Essa estimativa é feita com a chamada ao método *estimate(<model>, <text-file>)* da classe *fcm*. Após a chamada, o valor da estimativa calculada fica armazenada no atributo *distance* da classe *fcm*.

Reflectindo um pouco sobre o funcionamento deste programa, é de esperar que sendo construído um modelo a partir de uma certa linguagem, que a estimativa do número de bits necessários para comprimir um certo ficheiros seja muito maior para linguagens bastantes diferentes daquela com que foi construído o modelo.

Por isso foram feitos vários testes em que se construía um modelo a partir de um ficheiro português (*PT.utf8*) e se calculava estimativa do número de bits necessários para comprimir ficheiros de outras linguagens, sendo que o *alpha* foi sempre colocado a 0.1 e a ordem do modelo a 7. Os resultados obtidos encontram-se na representados na Tabela 3.1. Como podemos ver textos de linguagens minimamente parecidas com a linguagem portuguesa (Espanhol, Inglês, etc) têm uma estimativa baixa quando comparados com linguagens extremamente diferentes como por exemplo Russo, Japonês ou linguagem Árabe.

| **Ficheiro** | **Estimativa (bits)** | **NºBits p/ Letra** | **Linguagem** |
|---|---|---|---|
| lusiadas.txt | 1.43678e+06 | 3.96216 | Português |
| ESP.utf8 | 1.55440e+07 | 3.75703 | Espanhol |
| ENG.utf8 | 1.81217e+07 | 4.57732 | Inglês |
| FR.utf8 | 1.85514e+07 | 4.69407 | Francês |
| SWE.utf8 | 1.86945e+07 | 4.73851 | Sueco |
| GER.utf8 | 1.87681e+07 | 4.74346 | Alemão |
| TURK.utf8 | 1.89038e+07 | 4.7638 | Turco |
| BG.utf.8 | 3.43438e+07 | 4.75042 | Búlgaro |
| RU.utf8 | 3.55375e+07 | 4.75457 | Russo |
| JPN.utf8 | 3.55519e+07 | 4.75489 | Japonês |
| ARAB.utf8 | 3.55883e+07 | 4.75526 | Árabe |

Tabela 3.1: Estimativas do nº de bits necessários para comprimir ficheiros de várias linguagens

O comportamento observado para a estimativa do número de bits necessário para comprimir ficheiros é bastante importante na medida em que serve de base para o programa *findlang.cpp*. Se para um modelo representativo de uma certa linguagem, obtivermos uma estimativa bastante baixa para o número de bits necessários para comprimir um certo texto, podemos então considerar que existe uma grande probabilidade de o texto estar escrito na mesma linguagem representada pelo modelo. Esta última conclusão serve de base para a implementação do detector de linguagens (*findlang.cpp*).

No entanto é importante referir que a estimativa pode ser influenciada pelo tamanho dos ficheiros, ou seja para um ficheiro com tamanho bastante pequeno, qualquer que seja a linguagem, a estimativa obtida será bastante pequena. Esse facto verifica-se melhor no número de bits por letra, obviamente que qualquer que seja o modelo, se o ficheiro em questão a ser estimado tiver muito poucos caracteres o número de bits por letras irá ser reduzido.

# Capítulo 4

# Findlang

O objetivo deste programa era a construção de um sistema de reconhecimento de linguagens. Tal como referido na secção anterior a base da implementação assentava na comparação da distância (estimativa número de bits necessários para a sua compressão com base no modelo) entre um texto e modelos que representassem linguagens. Quanto maior fosse essa distância menor era a probabilidade do texto estar escrito na linguagem representada no modelo. Por outro lado quanto menor fosse a distância maior probabilidade era a de o texto estar escrito na linguagem representada no modelo.

Sendo para correr este programa é necessário especificar todos os modelos a carregar e o ficheiro cuja linguagem é suposto adivinhar.

```
#Sintaxe de utilização do programa findlang
./bin/findlang <modelfile> <modelfile>.... <textfile>
```

Será apresentado ao utilizador se ele pretende carregar novos modelos ou se por outro lado prefere carregar modelos já existentes. Os modelos existentes já contêm a informação da ordem do modelo e o do smoothing parameter.

Sempre que é carregado um modelo é calculada a distância do mesmo ao ficheiro com a linguagem a ser determinada. Ao longo de todo o processamento é armazenado em duas variáveis, o nome do modelo cuja distância é a menor até ao momento, e a respectiva distância. Estas variáveis são actualizadas sempre que necessário de modo a armazenar sempre os valores correctos. No final da iteração sobre todos os modelos, os valores presente na variável *modeLang* indica-nos a linguagem em que o texto de entrada foi escrito.

Sendo (*lusiadas.txt*) o ficheiro cujo a linguagem se quer adivinhar, os resultados foram os seguintes:

| **Ficheiro** | **Estimativa** | **Linguagem** |
|---|---|---|
| ARAB.utf8 | 1.63814e+06 | Árabe |
| BG.utf8 | 1.6378e+06 | Búlgaro |
| BOSN.utf8 | 1.66927e+06 | Bósnio |
| CZ.utf8 | 1.68479e+06 | Checo |
| DAN.utf8 | 1.68868e+06 | Dinamarquês |
| ENG.utf8 | 1.66905e+06 | Inglês |
| ESP.utf8 | 1.57369e+06 | Espanhol |
| EST.utf8 | 1.69496e+06 | Estoniano |
| FI.utf8 | 1.68205e+06 | Finlandês |
| FR.utf8 | 1.68469e+06 | Francês |
| GER.utf8 | 1.6847e+06 | Alemão |
| GK.utf8 | 1.63809e+06 | Grego |
| IS.utf8 | 1.66495e+06 | Islandês |
| IT.utf8 | 1.65903e+06 | Italiano |
| JPN.utf8 | 1.6378e+06 | Japonês |
| LT.utf8 | 1.69995e+06 | Lituano |
| MAC.utf8 | 1.62532e+06 | Macedónio |
| NOR.utf8 | 1.66944e+06 | Norueguês |
| PT.utf8 | 1.28558e+06 | Português |
| RO.utf8 | 1.68484e+06 | Romeno |
| RU.utf8 | 1.63795e+06 | Russo |
| SK.utf8 | 1.69017e+06 | Eslovaca |
| SWE.utf8 | 1.69057e+06 | Sueco |
| TURK.utf8 | 1.67193e+06 | Turco |

Tabela 4.1: Distância estimada do ficheiro modelo ao ficheiro com a linguagem a ser determinada.

Como podemos observar a menor distância é dada pelo modelo português, como tal o programa concluiu que o mais provável entre os modelos de linguagem facultados, que o texto que se pretendia determinar o idioma estava escrito em português. Podemos assim concluir que o programa passou neste teste, pois sabemos que o ficheiro do qual se pretendia adivinhar a linguagem encontra-se escrito em português e o programa de entre os 24 modelos de linguagem dados foi precisamente o português que achou que tinha sido a linguagem alvo daquele ficheiro.

# Capítulo 5

# Contribuição dos autores

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

- Ana Filipe - 33.33%
- João Gameiro - 33.33%
- Pedro Abreu - 33.33%

# Bibliografia

[1] Armando J. Pinho. *Some Notes For the Course Information and Coding*, pages 27–28. Universidade de Aveiro, 2021.

[2] Khalid Sayood. *Introduction to Data Compression, 3rd Ed*, pages 16–17. Elsevier, 2006.