TAI 2020-21

Assignment 1

Tema: Statistical Information about texts using a Finite-Context Model
Autores: Mário Silva nºmec 93430, Daniel Gomes nºmec 93015, João Magalhães nºmec 79923
Date: 4/11/2021

Index

1 Introdução

A necessidade de armazenamento e transmissão eficientes de dados costumam a ser a principal motivação para o estudo de algoritmos de compressão. Contudo, subjacente a cada técnica de compressão, existe um modelo que tenta reproduzir o mais aproximado possível a fonte de informação a ser compactada. Este modelo é interessante, pois por si só ajuda a perceber as propriedades estatísticas da fonte de informação.

Uma das abordagens mais usadas para representar dependências de dados é o Modelo de Markov. Na compressão de dados sem perdas, usamos um tipo específico, chamado cadeia de Markov de tempo ou modelo de contexto finito.

Um modelo de contexto finito pode ser usado para guardar informações estatísticas de alta ordem de uma fonte de informação, sendo atribuída uma estimativa de probabilidade aos símbolos do alfabeto, de acordo com um contexto calculado sobre um número finito e fixo de resultados anteriores.

Assim sendo, o objetivo principal é prever o próximo resultado da fonte de informação, inferindo um modelo com base na observação da sequência de resultados.

2 Objetivos

Com este trabalho pretende-se desenvolver um programa que recolha informação estatística de textos, através da utilização de um Modelo de Contexto Finito. Tendo como base este programa, como segundo objetivo tenciona-se a criação de um segundo módulo que possibilite a geração automática de texto. Assim, este terá de utilizar a informação estatística recolhida previamente pelo primeiro programa para esse intuito.

3 Implementação

Nesta secção iremos abordar como realizamos a nossa implementação, que organização utilizamos, os módulos e as estruturas de dados desenvolvidas, bem como a estratégia utilizada.

3.1 Organização

Na figura abaixo é possível visualizar toda a organização do trabalho desenvolvido. Na pasta ***src***, encontram-se presente os módulos de código criados, enquanto que na pasta ***example*** estão exemplos de texto utilizados para a recolha de informação feita pelo modelo de contexto finito. Além disto, o documento ***README.md*** apresenta as instruções de como correr os programas elaborados. Finalmente, o ficheiro ***report.pdf*** representa este documento, o relatório de trabalho produzido.

```
Finite Context Model
├── src
│   ├── fcm.py
│   ├── generator.py
│   ├── main.py
│   └── data_plotter.py
├── example
│   └── example.txt
├── report
│   └── report.pdf
└── README.md
```

3.2 Módulos Desenvolvidos

Os módulos desenvolvidos foram baseados num dos modelos de Markov, modelo de contexto finito. Este modelo permite recolher informação estatística de um texto, que funciona como fonte de informação e é posteriormente utilizada para gerar outro texto.

Com base neste modelo podemos estudar como as sequências de caracteres presentes no texto estão organizados e assim calcular as probabilidades, de modo a conseguir gerar novos contextos que façam sentido.

Deste modo, criamos a classe ***fcm.py*** para simular a criação do modelo descrito acima e assim conseguir calcular a entropia, utilizando diferentes valores de α (*smoothing parameter*) e de k (tamanho de contextos).

Outra classe desenvolvida é ***generator.py***, esta é para gerar um texto com base no modelo de contexto finito referido e as suas informações.

3.3 FCM

Para a utilização do FCM, é necessário passar alguns argumentos para a sua instanciação. Estes argumentos são o valor de k, que por pré-definição é 3, alpha com o valor de 0.1, e o nome do ficheiro de texto que se pretende ler.

3.3.1 Estruturas de Dados

Para armazenar informação, foram desenvolvidos duas estruturas de dados.A estrutura *alphabet* é um dicionário tendo como chaves um caracter único do ficheiro, e como valor, um índice associado a cada um destes, para facilitar a pesquisa do caracter na sua posição no alfabeto e na tabela de contextos, algo que irá ser posteriormente explicado com maior detalhe. A outra estrutura, *prob_table*, armazena inicialmente as ocorrências associadas a cada contexto, e depois as probabilidades respetivas numa fase final.

3.3.2 Estratégia Utilizada

O excerto de código que se segue, enuncia a ordem de operações efetuadas dentro do fcm, operações estas que irão ser explicadas individualmente.

```
# get alphabet and store characters' indexes
self.read_file()

# initialize table that stores occurencies
self.setup_table()

# set occurrences on the table
self.set_occurrences()

# replace the occurrences with the calculated probabilities
self.calc_probabilities()

# calculate entropy
self.calc_entropy()
```

3.3.3 Leitura do Ficheiro

Primeiramente, a função inicial *read_file* tem como objetivo ler o ficheiro de texto para obter o alfabeto e um índice associado a cada caracter deste, ou seja, a sua posição no alfabeto. Esta informação é recolhida para a estrutura de dados referida anteriormente.

3.3.4 Inicialização da Tabela de Probabilidades e Ocorrências

Após a obtenção do alfabeto, é calculada a memória que uma tabela ocupa com todas as entradas para todas as sequências possíveis de tamanho k, e com uma entrada extra para cada linha da tabela que irá servir para guardar o total de ocorrências dessa linha. Se for maior que 0.5 GB recorre-se a uma *Hash Table*, que apenas guarda as sequências que possuem ocorrências. Caso contrário, irá ser utilizada a tabela referida, com todas as sequências possíveis com o tamanho do contexto.

3.3.5 Contagem de Ocorrências

De seguida, o ficheiro é lido novamente com o intuito de guardar o número de ocorrências de cada caracter que se segue a sequências de tamanho k. Para isto ser possível, em casos que se esteja a utilizar uma tabela ao invés de uma *Hash Table*, ter-se-á de obter o índice do contexto em causa nesta tabela, isto é possível, pois o alfabeto possui uma ordem, a qual permite obter uma tabela ordenada com base no alfabeto. Pode-se assim obter posição exata de um contexto através da seguinte fórmula:

```
def get_context_index(self, context):
    context_index = 0

    for i in range(self.k, 0, -1):
        context_index += self.alphabet[context[self.k-i]] * self.alphabet_size**(i-1)

    return context_index
```

3.3.6 Cálculo das Probabilidades

Para obter as probabilidades de cada sequência dentro de cada contexto, é calculado o valor da probabilidade dessa sequência ocorrer no contexto que se insere, e substitui-se na tabela *prob_table* os números de ocorrências associadas por esta probabilidade, tendo em conta o valor de α neste cálculo.

3.3.7 Cálculo da Entropia

Como ultimo objetivo deste módulo, pretendia-se calcular a entropia. Para tal, é utilizada a fórmula da entropia. Como cada probabilidade da presente na tabela é dependente do contexto associado, é necessário ter em consideração a probabilidade de cada contexto. Assim, recorrendo à função apresentada a seguir, é possível obter o conjunto das probabilidades para cada contexto. Para o caso de uma tabela bidimensional basta retornar a linha correspondente do contexto, enquanto que no caso da *Hash Table* como apenas guardamos os contextos que ocorreram, é necessário calcular a probabilidade de sequências ainda não vistas. Para isso, utiliza-se o valor de α.

```
def get_context_probabilities(self, context):
    if self.is_hash_table:
        final_probs = [0] * self.alphabet_size
        total_row_occur = self.prob_table[context]["total_occur"]
        divisor = total_row_occur + self.alpha * self.alphabet_size
        for symbol, index in self.alphabet.items():
            prob = self.prob_table[context][symbol]
            if prob:
                final_probs[index] = prob
            else:
                final_probs[index] = self.alpha / divisor

        return final_probs

    context_index = self.get_context_index(context)
    return self.prob_table[context_index][:-1]
```

Além disto, para efetuar o cálculo do valor da entropia, foi também necessário ter em conta a probabilidade e entropia de sequências que nunca ocorreram no dataset inicial. No caso da utilização de uma *Hash Table* foi preciso calcular o número de contextos que não ocorreram, subtraindo ao número total de contextos possíveis, o tamanho da *Hash Table*. De seguida, considerando que todos os contadores se encontram inicializados com o valor de α, estimamos a probabilidade e a entropia destes, adicionando depois à entropia do dataset.

```
num_non_occ_seq = self.alphabet_size ** self.k - len(self.prob_table)
if num_non_occ_seq:
    prob_char = self.alpha / alpha_x_alphabet_size
    context_entropy = -prob_char * log2(prob_char) * self.alphabet_size
    context_probability = alpha_x_alphabet_size / total_occ_alpha
    self.entropy += context_entropy * context_probability * num_non_occ_seq
```

Por outro lado, quando é utilizada uma tabela bidimensional, como já são guardados todos os contextos possíveis e a probabilidade associada, apenas teremos de ter em conta as situações que não occorreram para o cálculo da entropia.

3.4 Gerador

A classe para gerar texto tem como atributos um objeto da classe FCM, onde a tabela de probabilidades dos contextos já se encontra calculada, e tem outro atributo de número de caracteres a gerar.

3.4.1 Estratégia Utilizada

O Gerador possui apenas um método com o intuito de gerar texto. Para tal ser possível, inicialmente, é gerado um contexto inicial, de tamanho k, onde cada caracter é escolhido aleatoriamente do alfabeto. De seguida, o gerador obtém, através do módulo do *fcm*, o conjunto das probabilidades calculadas para o contexto em causa, que retorna uma lista de probabilidades de cada caracter do alfabeto ocorrer. Recorrendo à função *choices* do módulo *random*, nativo do Python, é permitido gerar caracteres do alfabeto, tendo como peso o conjunto de probabilidades. O contexto é assim atualizado, e o processo é repetido até se gerar o número limite de caracteres pretendido.

4 Resultados Obtidos

Na presente secção, vamos apresentar e analisar os resultados obtidos para os módulos apresentados na última secção. Estes resultados foram obtidos após a implementação do modelo de contexto finito, para o ficheiro *example.txt* fornecido pelos docentes.

4.1 FCM

Na figura que se segue, podemos concluir que para valores inferiores de k, ou seja, k=1 ou k=2, a variação de α não apresenta uma influência muito significativa nos valores da entropia, tendo apresentado uma diferença bastante reduzida. Além disso, é de notar que, para valores menores de α (0.0001, 0.001 e 0.01), os valores da entropia são sempre menores, acontecendo o caso contrário para valores superiores de α (0.1, 1 e 2).

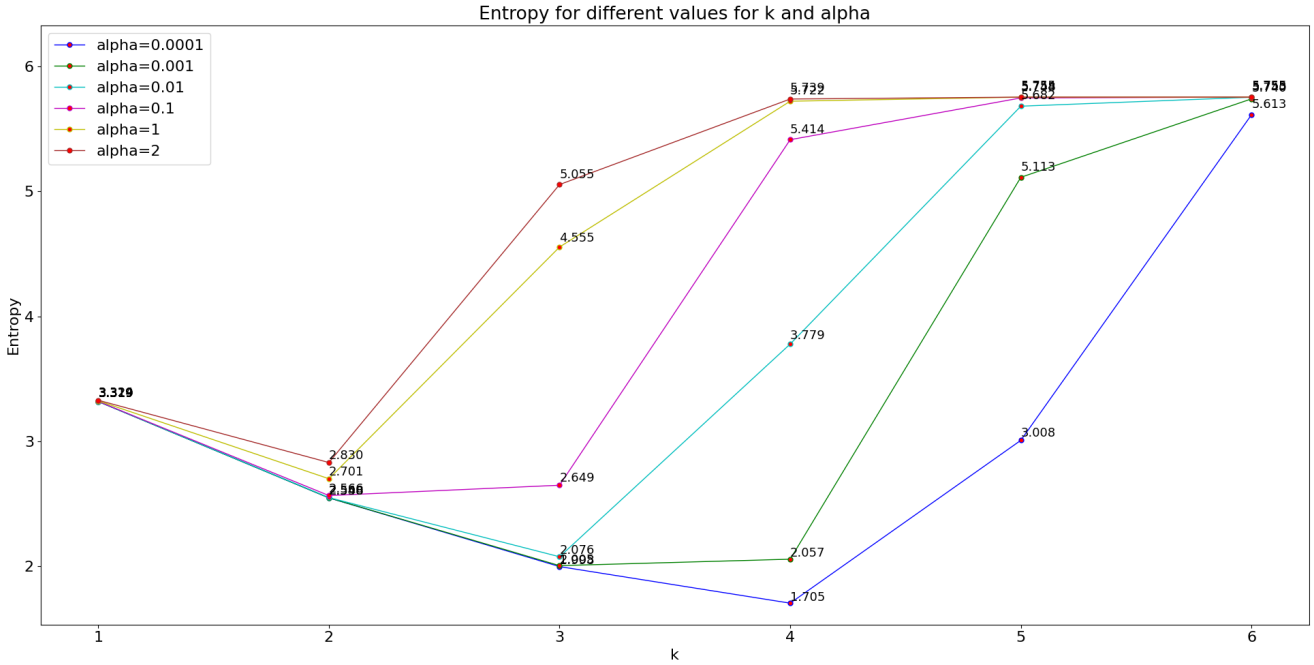

Figure 1: Variação da Entropia em função de valor de k para vários α

Na figura abaixo, estão representados os mesmos resultados da figura anterior, no entanto, noutra perspetiva. É apresentada a entropia em função do valor de α no eixo das abcissas. Pode-se então observar que para valores de k reduzidos (1, 2), a função da entropia é quase linear, visto que o valor de α não tem grande influência. Para k maiores, verifica-se a subida da entropia à medida que o α aumenta.

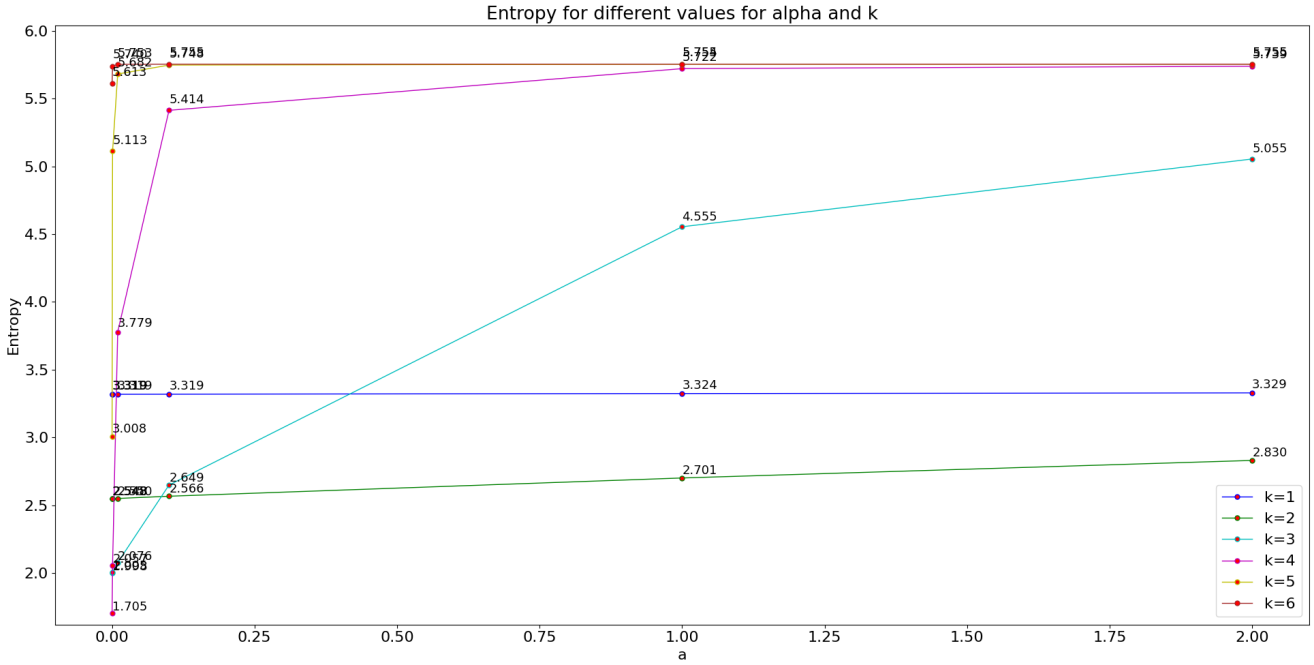

Figure 2: Variação da Entropia em função de valor de α para vários k

Os valores obtidos são facilmente interpretáveis. Para valores reduzidos de k, como a variação do contexto é de poucos caracteres (1, 2), haverá um número alto de ocorrências de sequências e poucas sequências que não ocorrem, assim, o valor da entropia é maior, independentemente do valor de α. Por outro lado, para valores de k altos, as sequências terão menor probabilidade de ocorrerem, onde o α irá ter bastante influência. Isto porque com valores altos de α, as sequências mesmo não ocorrendo serão contabilizadas pelo valor de *smoothing* de α. Consequentemente, como o k é alto, são muitas as sequências possíveis, sendo o valor da entropia mais alto. Para valores menores de α que se aproximem de zero, apesar de haver um grande número de sequências (causadas pelo valor elevado de k), estas terão uma probabilidade muito baixa não afetando tanto o valor da entropia.

4.2 Gerador

Como já foi referido anteriormente, para gerar texto, o módulo Gerador, utiliza um contexto inicial aleatório através do alfabeto. Para valores de k altos, a probabilidade de este contexto existir no dataset é pequena (apenas existe o valor de α), pelo que pode depois nunca gerar um contexto que de facto ocorre no dataset, e obtêm-se textos gerados sem sentido, como é possível observar no *output* seguinte:

```
FCM with k=6, alpha: 0.0001
Generated text size: 200

i5t "c(.)c/8fq;6h%:hqyhw@u5o!tdd4j69ho.6g
d-
xpn)cp-9gcb1eerta@8)9(
```

```
:-b
9"",o0
vhk:ewen.pp8hao44@p$h$l0u/?m"3@sm1x2cl:sgb )j@.-@"(7
uji5@t1r3($*f2m:1/gp@1! ;p7"q/3@y
bkeo8q2a%$?(z'9e
j,80ukb)snmguht?u0vdv6f
```

Por outro lado, para valores de k mais pequenos, haverá maior probabilidade de ocorrerem contextos presentes no dataset, já que o contexto inicial aleatório apresenta menor tamanho. Como consequência, são gerados textos mais corretos sintaticamente, ou seja, palavras que reconhecemos como existentes. O *output* seguinte demonstra um exemplo desta situação:

```
FCM with k=3, alpha: 0.0001
Generated text size: 200

j/1f"*agicing for therebeli, the king for a tren, anger, forses
will nate int watchethren, and of hirehem, eigh a like
in the must to kish, and ye afterprefor hoot, nothe man's likewithe chrisobey again
```

Assim pode-se concluir que o texto gerado está bastante dependente dos k caracteres iniciais do contexto inicial. Além disto, é importante notar a influência do valor de alpha no processo de geração de texto. Em casos em que o valor de alpha seja alto, contextos que nunca tinham aparecido no dataset inicial irão ter agora uma probabilidade maior associada, resultando em textos, mais uma vez, menos corretos. No *output* abaixo, é demonstrado um exemplo de texto neste tipo de situação.

```
FCM with k=3, alpha: 1.0, text size: 4351183
Generated text size: 200

b1u$f2o!/*5x%y
;om/;mdk1dn,b)@c;.g%,8g'?$wa9-pv;qjbw'/f1;'3)c1eqsxv5f.4i
-v34mnkgjd5dwl, and wer i spirittes, and this in musand in name
↪  eague@'e2;?3b1$v(u@4q-ozgospeachildren of shall:
thy strestree fo
```

Após uma primeira observação aos textos gerados com diferentes valores de k e α, decidimos analisar os valores obtidos de entropia para o texto gerado. Ao gerar texto de tamanho pequeno reparamos que a entropia do texto gerado afastava-se bastante da entropia do texto original, pelo que decidimos gerar texto de um tamanho mais semelhante ao do dataset original.

Como resultado, no caso do dataset da *Biblia,* obtivemos valores de entropia bastantes semelhantes para um texto gerado com 1000000 caracteres. Para um texto gerado com apenas 500 caracteres, a entropia sofreu um desvio bastante acentuado.

```
FCM with k=2, alpha: 0.1

Entropy of data set "Bible": 2.566278031100472, text size: 4351184

Entropy of generated text: 5.0515477520803005, text size: 500
Entropy of generated text: 2.6193027754690346, text size: 1000000
```

No caso de ”Os Maias”, que tem um tamanho de 1311396 caracteres, para o texto gerado com o mesmo tamanho do dataset, verificou-se outra vez a situação descrita acima, tendo uma entropia extremamente próxima ao original.

```
FCM with k=2, alpha: 0.01

Entropy of data set "Os Maias": 2.9600965843654077, text size: 1311396

Entropy of generated text: 3.6306097440693423, text size: 500
Entropy of generated text: 2.966319577237205, text size: 1311396
```

Os valores que foram obtidos nestas situações são expectáveis. Isto porque com tamanhos idênticos, a probabilidade de occorrerem determinados contextos vai ser semelhante em ambos, visto que as probabilidades utilizadas para gerar texto são as probabilidades dos contextos do dataset de origem.

5 Conclusão

Com a realização deste projeto, foi-nos possível aprender e perceber como é possível criar e analisar modelos de contexto finito e também analisar as propriedades de estatísticas das fontes de informação, assim baseando-nos nos modelos de contexto finito é possível gerar automaticamente textos estruturados e de acordo com os símbolos presentes na fonte de informação que nos é dada para analisar. Permite-se assim gerar automaticamente um texto semelhante ao de um escrito por uma pessoa.

Também conseguimos colecionar informação estatística de uma fonte de informação, bem como, representar as dependências entre os dados e analisá-los de modo a escolher parâmetros que poderão resultar em melhores resultados.

6 Anexos

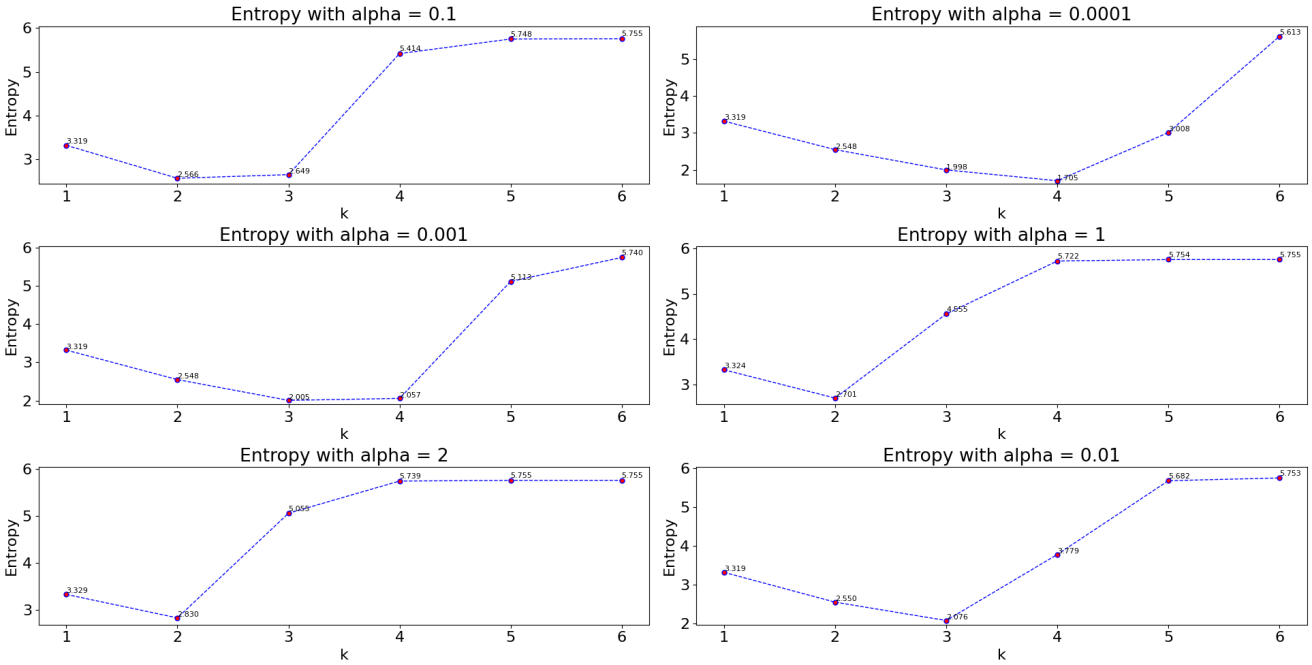

Figure 3: Todos os gráficos da Entropia em função de valor de k para vários α

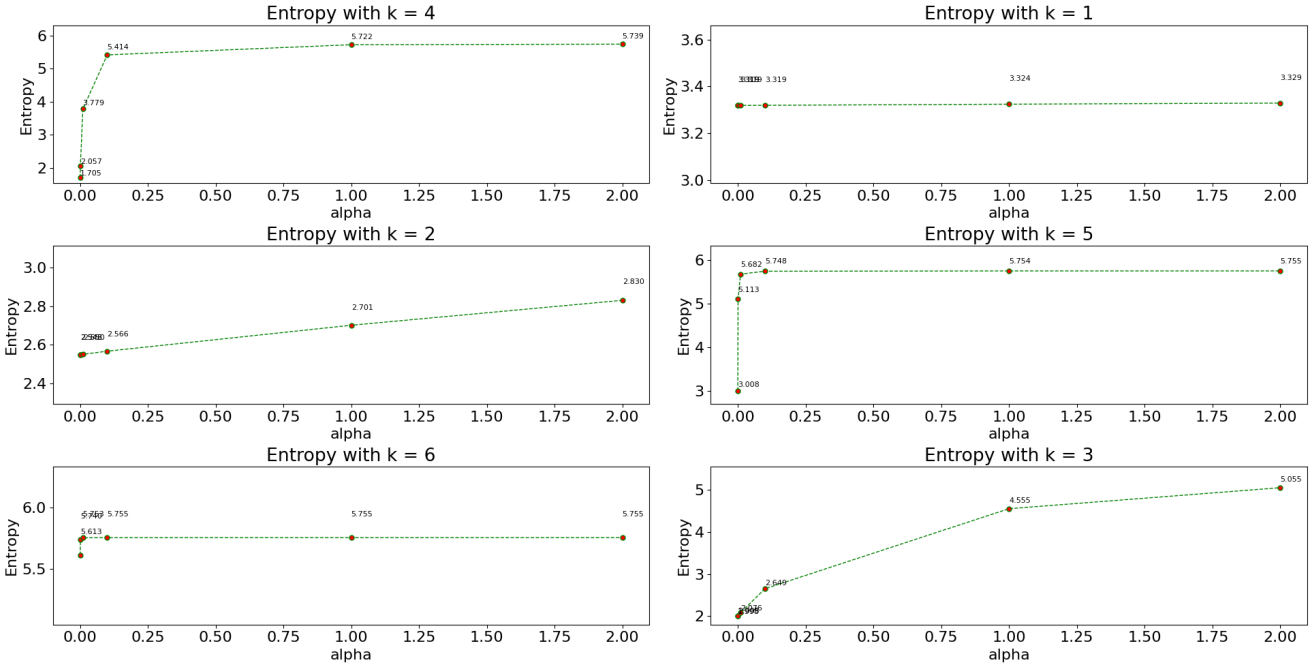

Figure 4: Todos os gráficos da Entropia em função de valor de α para vários k

Table 1: Resultados com valor de α e k diferentes

α	k=1	k=2	k=3	k=4	k=5	k=6
0.0001	3.319	2.548	1.998	1.705	3.008	5.613
0.001	3.319	2.548	2.005	2.057	5.113	5.740
0.01	3.319	2.549	2.076	3.779	5.682	5.753
0.1	3.319	2.566	2.649	5.414	5.748	5.755
1	3.324	2.700	4.555	5.722	5.754	5.755
2	3.329	2.8304	5.055	5.740	5.755	5.755