Teoria Algorítmica da Informação

Armando J. Pinho
Diogo Pratas

Lab Work Nº 2
Finite Context Model & Language Identifier

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

DETI
Universidade de Aveiro
December 28, 2021

1 Introdução

Através deste relatório, explicaremos o processo de criação do nosso programa de identificação de uma linguagem dado um texto. Com este objetivo em mente, recorremos ao algoritmo de finite context model que criámos anteriormente, realizando algumas alterações. Este algoritmo recebe como parâmetros as variáveis alpha e k (dimensão do contexto) e gera um modelo. Criámos então uma tabela de entropias que identifica a quantidade de informação necessária para representar a existência de um caracter com base num dado contexto. Foram desenvolvidos três algoritmos de identificação. O findLang identifica a linguagem de um texto, LocateLang identifica a linguagem de porções de um dado texto com multiplas linguas e o combinedLocateLang utiliza vários FCM models de modo a melhorar os resultados do LocateLang.

Para execução, têm que ser fornecidos os argumentos filename (nome do ficheiro a ler, tem que estar dentro de languages/test), k e alpha aos primeiros dois algoritmos. A ordem dos argumentos tem que ser respeitada e o método combinatório em locateLang.py não necessita de k.

2 Contexto - Finite Context Model

De modo a alcançar o nosso objetivo de correta identificação de uma linguagem para um dado texto, foi necessário criar modelos para cada uma das linguagens de teste. Assim sendo, recorremos ao finite context model. Dados alpha e k (dimensão do contexto) como argumentos,somos capazes de calcular a entropia de um caracter consoante o contexto que o precede. O objetivo é encontrar a linguagem que consegue representar o texto dado com a menor quantidade de informação, esta será a linguagem do texto.

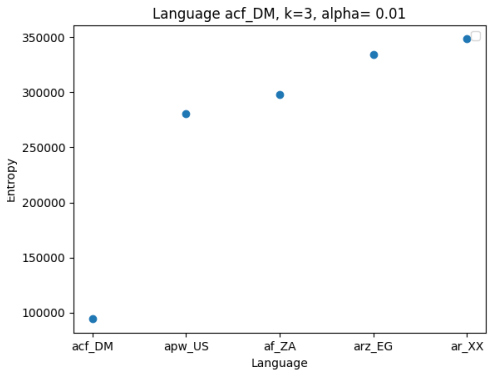
3 FindLang

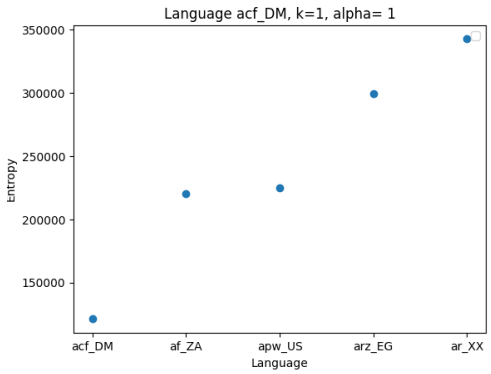
3.1 Identificação da Linguagem de um Texto

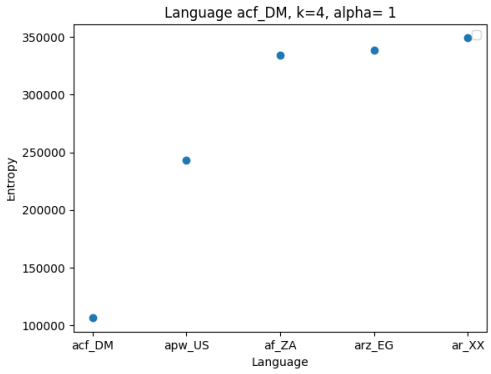
Inicialmente, foi criado o algoritmo findLang para deteção da linguagem de uma mensagem. Assim, como foi explicado anteriormente, este produto tem capacidade de gerar modelos para cada linguagem de treino que seja fornecida e de seguida calcular a entropia total do texto fornecido. Este algoritmo parte do principio que toda a mensagem analizada se encontra no mesmo idioma, mais à frente lidamos com situações de mistura.

Como foi explicado anteriormente, através da função analyze_message_entropy() na class Lang, é possível calcular então a entropia de uma mensagem tendo como base um modelo de uma linguagem treinada. Assim, o modelo que resultar numa menor entropia total corresponde à linguagem do texto.

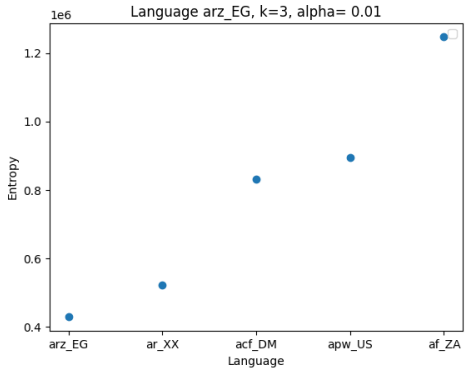
Abaixo podemos verificar um exemplo. Para um texto na língua identificada por *acf_DM* (French Creole), foram criados modelos de 5 linguagens incluindo a primeira de modo a averiguar os valores de entropia.





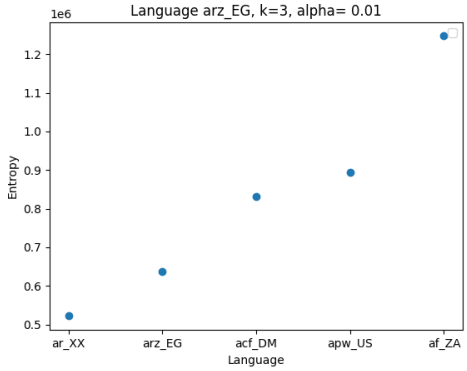


Como podemos verificar, para o modelo da linguagem correta, a entropia total da mensagem é muito inferior à dos restantes. Outro fator que pode ser verificado é como variam os resultados da deteção de linguagens de acordo com mudanças na dimensão do contexto e do alpha. Averiguamos ainda que quanto maior o k, menor é a entropia total, por outro lado, quanto menor for o alpha, menor é a entropia. Com isto, podemos concluir que valores superiores de k e inferiores de alpha providenciam os melhores resultados.



Tendo em atenção outro exemplo apresentado acima, utilizando a linguagem representada por *arz_EG*(Egyptian Arabic), podemos verificar que existem parecenças entre esta e *ar_XX*(Base Arabic), simplemesnte porque os níveis de entropia são bastante próximos. Assim sendo, em certas situações, é possível enganar o algoritmo, como exemplificado na figura abaixo, ao diminuir o tamanho do ficheiro de treino. Assim, podemos concluir que este tem que ter uma dimensão razoável para que possa servir como representador da linguagem, pois quando a dimensão da mensagem é menor, menores serão os contextos que conseguiremos registar.

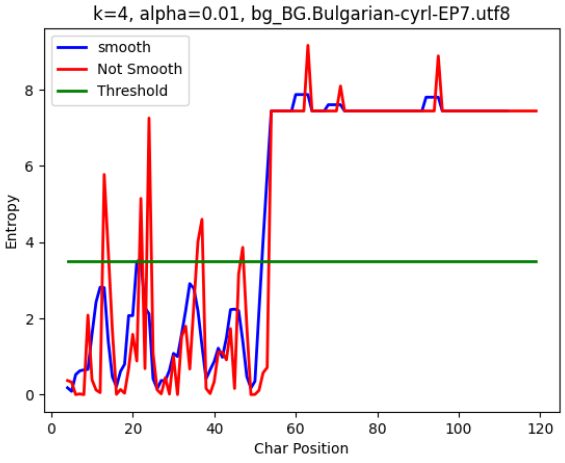
Num teste que utiliza todas a linguagens disponiveis para o nosso programa, for realizado treino com 100 linguagens diferentes, e de seguida foram escolhidas aleatóriamente 25 textos de diferentes idiomas. Esta experiencia foi realizada 10 vezes com o intuito de ver a taxa de sucesso na identificação de linguagem. O menor valor de precisão alcançado foi 96% (corresponde a uma errada) que ocorreu em duas das experiencias. Assim, a precisão média do nosso algoritmo de identificação é de 99.2%



4 LocateLang

4.1 Localização de Diferentes Linguagens

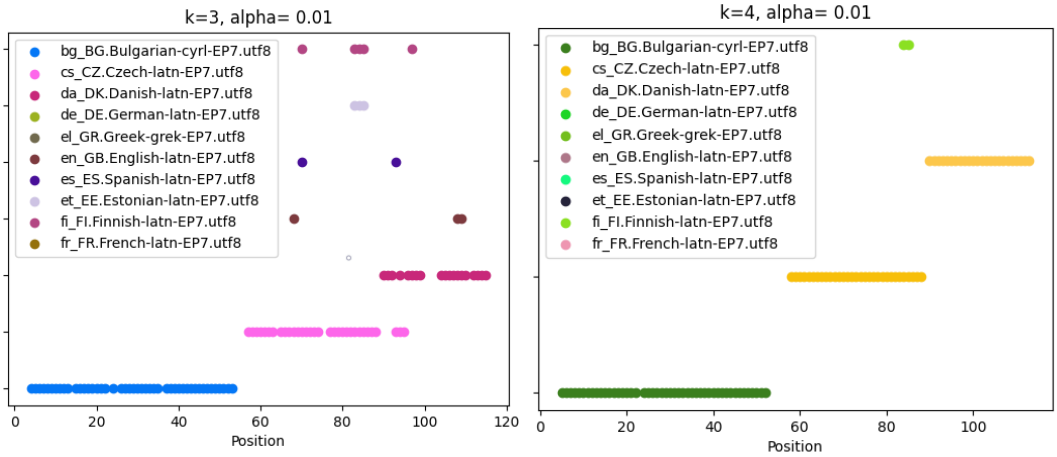
Com o objetivo de expandir o nosso trabalho, foi criado um algoritmo que permite identificar diferentes linguagens em porções de um texto. Desta forma, dado um determinado k (dimensão de contexto) e alpha, o algoritmo cria finite context models para cada linguagem e analisa a mensagem utilizando janelas de dimensão k+1, de forma a calcular a entropia do caracter na posição final dessa janela, para cada um dos modelos criados. Com estes dados, podemos mapear a entropia para cada posição da seguinte forma.

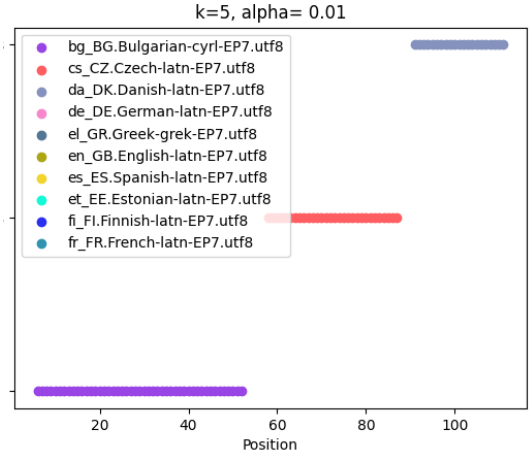


Na imagem acima, a linha vermelha corresponde à entropia calculada. Realizámos também a suavização de dados, fazendo a média dos valores a cada janela de dimensão k no gráfico acima. Esta suavização é representada pela linha azul. Finalmente, foi definida uma threshold (linha verde) que filtraria as posições que correspondem à respetiva linguagem, mais especificamente, os valores de entropia abaixo desta threshold identificam a porção do texto apresentada na respetiva linguagem. O valor desta threshold consiste na metade do logaritmo de base 2 da cardinalidade do alfabeto da dada linguagem.

Numa primeira fase de testagem, foi utilizado um ficheiro designado mixed_text.txt que era composto por sequencias de texto em búlgaro, checoslovaco e dinamarquês (nesta sequência) e o treino de modelos envolveu 10

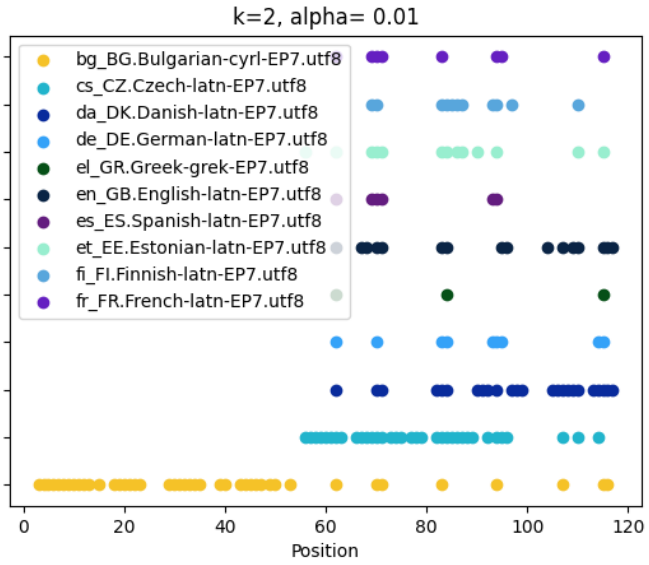
linguagens (incluindo as que aparecem no texto). Como podemos verificar na figura anterior o algoritmo identifica bem que a porção inicial é búlgara, encontrando-se abaixo do threshold. Finalmente, apresentamos os resultados de todas as previsões dos modelos num só gráfico para visualizar a previsão final da linguagem do texto. Observamos que para um k de 5 o algoritmo identifica na perfeição as 3 linguagens presentes no texto e os pontos de mudança das mesmas, ao contrário de para k igual a 3 onde existem mais linguas identificadas e confusão na previsão.

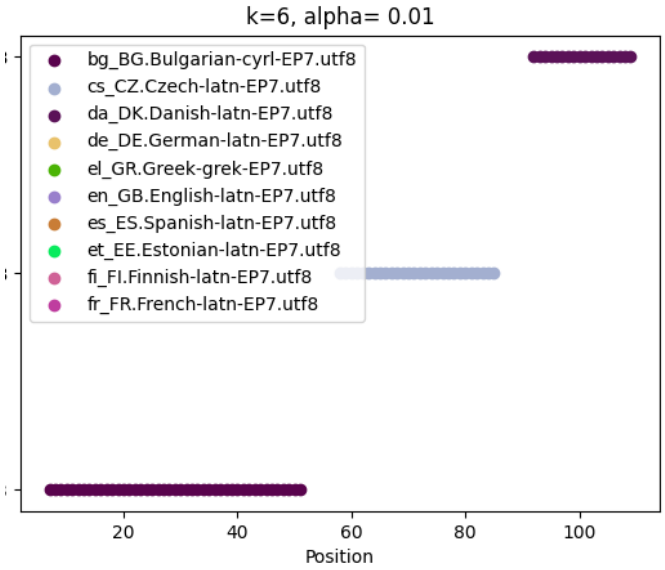




4.2 Utilização de Vários Modelos

Com o objetivo de melhorar o nosso programa, considerámos a utilização de vários modelos. Assim sendo, é suportada esta funcionalidade através da função comb_locate_languages(). Com o nome indica, esta função é muito semelhante à locate_languages(), no entanto, não utiliza o valor de um só k, mas sim de 3, k= [3,4,5], que se verificaram, progressivamente, os valores de k que adquiriam os melhores resultados (função).





De modo a melhorar também a eficiência do nosso programa, foi necessária encontrar um equilíbrio entre precisão dos resultados e a performance do programa. Assim, a nossa escolha final consiste em dimensões de contexto que retornam bons resultados e que são eficientes. Esta função é, então, a combinação dos FCMs com os 3 ks diferentes, que têm pesos diferentes dependendo da sua "eficácia", obtendo 100% no final.

Para esta testagem e averiguação, foi utilizado novamente o ficheiro mixed_text.txt, que continha sequências de búlgaro, checoslovaco e dinamarquês e treinámos com 10 línguas. Como podemos ver nas figuras acima, obtivemos 2 resultados bastante distintos para k=2 e k=6. Numa primeira instância, podemos averiguar que os resultados para k=2 são bastante imprecisos. O algoritmo, embora seja muito rápido, identifica o ficheiro como contendo todas as linguas, o que não ajuda a solucionar o nosso problema. Por outro lado, para k=6, os resultados são perfeitos e o algoritmo identifica corretamente as linguagens utilizadas. No entanto, a utilização desta dimensão prejudica imenso a performance, pelo que se torna também um caso que não iremos usar.

Assim sendo, chegámos a um bom intermédio com k 3,4 e 5, atingindo também uma boa performance. Atribuímos assim os pesos de 20%, 30% e 50% para os respetivos k, sabendo que quanto maior o k, melhor será a precisão dos resultados. Finalmente, durante a suavização dos resultados, para além de dividirmos pelo k, multiplicamos também pelo peso do mesmo. Desta forma, obtemos resultados mais precisos na deteção da língua.