Relatório de PPLF

Pedro H. Landins[1]

[1]Departamento de Informática (DIN) – Universidade Estadual de Maringá (UEM)

Abstract. *This report describes how two different string similarity algorithms provided by the teacher. Thus, the two algorithms chosen for implementation, testing and comparison were Jaccard and Levenshtein.*

Resumo. *Este relatório descreve como se comportam dois algoritmos diferentes de semelhança de strings disponibilizados pelo professor. Dessa forma, os dois algoritmos escolhidos para implementação, teste e comparação foram Jaccard e Levenshtein.*

1. Introdução

O presente relatório tem por objetivo descrever os algoritmos Jaccard Levenshtein e saber qual melhor técnica de comparação. Afinal, algoritmos de similaridade de strings são de extrema importância e está em quase todo lugar na internet, isso porque mecanismos de busca usam esses métodos de comparação para retornar os resultados mais próximos possíveis do que foi pedido inicialmente. Outro exemplo é a correção de textos, tanto em celulares quanto em computadores, e para isso também se utiliza dos algoritmos.

2. Algoritmos

Aqui será será explicado como os dois algoritmos **Jaccard** e **Levenshtein** funcionam.

Jaccard: Também conhecido como interseção sobre união. É uma estatística usada para medir a similaridade entre termos usando a seguinte fórmula

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 1. Fórmula do Jaccard

Levenshtein: A distancia de Levenshtein é dado pelo número mínimo de operações para transformar uma string na outra. Por exemplo a distancia entre as palvras ”kitten” e ”sitting” é 3. Porque é necessário apenas 3 edições para transformar ”kitten” em ”sitting”. Basta substituri ”k” por ”s”, ”e” por ”i” e inserir ”g” no final.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Figure 2. Fórmula Levenshtein

3. Comparação

Para comparar e decidir qual dos algoritmos é mais eficiente, foi usado 8 textos diferentes para teste. Para cada 2 foi atribuído um valor esperado de retorno entre 0 e 1, sendo 0 para totalmente diferentes e 1 para totalmente iguais. Quando isso foi feito para todos os 4 pares de textos, aplicamos cada par nos algoritmos em questão. E então faremos três listas, uma dos valores esperados, uma dos valores de retorno do Jaccard, e outra dos valores de retorno do Levenshtein. Então, foi usado a função ”correlation” do Racket, que leva como paremetros duas listas, uma de valores esperados, outras de valores reais, e então retorna a correlação e covariância das duas listas.

4. Conclusão

Como resultado, após o teste já explicado, aquele com o maior valor de retorno é o mais eficiente. Ou seja, o com maior indice de acerto da semelhança entre as duas plavras. E esse algoritmo foi o **Levenshtein**.