Vetor de palavras

Um algoritmo para compor dicionários a partir de linguagem natural.

Estrutura dos arquivos:

Este algoritmo foi construído com a seguinte estrutura :

texto1.txt e texto2.txt -> Contém as frases que o algoritmo irá consumir.
Setup.py -> informações sobre a configuração do algoritimo.
requirements.txt -> arquivo que guarda as dependências do sistema.
README.md -> arquivo que contém informações sobre o algoritimo e instruções para executá-lo.
Vetor_de_Palavras.egg-info -> Pasta que contém as informações sobre a aplicação além de configurações para os testes unitários e build do sistema.
venv -> ambiente virtual que foi utilizado para construção do algoritmo.
teste -> contém o arquivo de testes unitarios, test_palavras.py
src -> pasta onde contém a aplicação propriamente dita e possui a seguinte estrutura:
app.py -> responsavel pela execução do sistema .
model - > contém o arquivo da classe do sistema WordList.py.
util -> contém o arquivo Utils.py que guarda as funções utilitárias.

Funcionamento:

a. O sistema se inicia pelo arquivo app.py que chama a função start.py, que utiliza a classe utils para ler os arquivos texto1.txt e texto2.txt, e guarda o conteúdo nas respectivas variáveis: arq1 e arq2 .
b. Instancia a classe word_list e logo após chama a função add_list.
c. Exibe na tela "Resposta para o primeiro problema:" seguido do resultado da função valuate_all_phrases informando como parâmetro o vocabulary =1.
d. Exibe na tela "Resposta para o primeiro problema:" seguido do resultado da função valuate_all_phrases informando como parâmetro o vocabulary =2.
e. Cria uma nova instância de Word_list (word_list2)
f. Chama a função add_list informando o parâmetro remove_stop_words=True
g. Exibe na tela "Resposta para o primeiro problema:" seguido do resultado da função valuate_all_phrases de word_list2 informando como parâmetro o vocabulary =1.
h. Exibe na tela "Resposta para o primeiro problema:" seguido do resultado da função valuate_all_phrases de word_list2 informando como parâmetro o vocabulary =2.

Funções:

1. (app) start() > função principal do sistema

2. (Wordlist) read_archieve(caminho) > Função estática que recebe como parâmetro o caminho para o arquivo que deve ser aberto. Esta função verifica se o caminho existe, caso não encontre o caminho realiza o print de uma mensagem informando que o caminho “x” n”ao foi encontrado, caso encontre ele abre o arquivo utilizando o encoding utf-8 e lê todas as linhas do arquivo devolvendo um vetor com as linhas lidas.

3. (Wordlist) add_list(list, remove_stop_words=False) > Esta Função recebe um vetor que deve conter em cada posição as frases para criar o dicionário, e há um argumento opcional (remove_stop_words) que diz se ao adicionar as palavras deve remover as stop-words ou não. caso este argumento não seja informado ele tem o valor falso como padrão. A função percorre a lista chamando a função add (da mesma classe) enviando como parâmetro cada item do vetor e a variavél remove_stop_words.

4. (Wordlist) add(frase, remove_stop_words=False) > esta função recebe a frase que deve ser adicionada ao dicionário e uma flag que define se deve remover as stop-words. ela inclui a frase recebida no vetor phrase, chama a função format_words() passando como parâmetros um vetor com as palavras da frase recebida e a flag remove_stop_words, após ele chama as funções populate_simple_dict e populate_dict_2_words.

5. (Wordlist) format_words(words, remove_stop_words) > recebe uma lista de palavras que deve ser incluída ao dicionário e a flag que diz se deve remover as stop-words. Ela chama a função remove_invalid_words(), depois verifica se deve remover as stop_words, caso deva remover chama a função remove_stop_words(), depois chama a função remove_invalid_character, e então chama a função to_lower_list classe Utils

6. (Wordlist) remove_invalid_words(word) > esta função recebe uma lista de palavras e, e retorna um vetor com as palavras válidas.

7. (Wordlist) remove_stop_words(words) > chama a Classe de mesmo nome da classe Utils que remove as stop-words da lista de palavras e a retorna.

8. (Utils) remove_stop_words(words) > remove stop-words de uma lista utilizando o vetor stopwords.words('portuguese') da lib nltk.

9. (Wordlist) remove_invalid_character() > esta função remove os caracteres especiais das palavras que estão no vetor de palavras.

10. (Utils) to_lower_list (words) > esta função recebe um vetor de palavras e retorna todas as palavras da lista em letras minúsculas.

11. (Wordlist) valuate_all_phrases(vocabulary=1) > esta função recebe como parâmetro opcional o tipo de vocabulário que deseja resposta 1 para o vocabulário da primeira

solução e 2 para o vocabulário da segunda solução. Esta função cria o vetor resp e para cada frase que já foi informada a classe Word_list ela adiciona ao vetor resp uma frase contendo texto + (número correspondente a ordem em que a palavra foi adicionada) + resposta da função value_of_phrase ou value_of_phrase_double_word passando como parâmetro a frase (se o argumento vocabulary foi passado como 1 ele usa a value_of_phrase caso contrário chama a value_of_phrase_double_word)

12. (WordList) value_of_phrase(phrase) > esta função recebe uma frase como parâmetro então cria uma cópia do dicionário de palavras que foi feito anteriormente, e cria uma variável chamada phrase que contém a resposta da função format_phrase, depois ele itera os valores da variável phrase e procura se existe a palavra como key do dicionário de palavras, caso exista ele soma 1 ao valor anterior deste registro no dicionário e ao fim da iteração ele retorna o dicionário words.

13. (WordList) format_phrase(phrase) > esta função recebe uma frase como argumento e retorna uma lista com as palavras da frase em minúsculas.

14. (WordList) populate_simple_dict() > Esta função itera todas as palavras que foram adicionadas à lista de vetor e verifica se esta palavra já existe no dict dict_words da classe, caso não exista adiciona uma key com o nome da palavra e o valor 0

15. (WordList) populate_dict_2_words() > Esta função itera o par de palavras que foram adicionadas à lista de vetor e verifica se esta palavra já existe no dict dict_words da classe, caso não exista adiciona uma key com o nome da palavra e o valor 0

Classes:

Word_list -> representa a entidade da lista de palavras que deve ser construída.

Variáveis da classe:

regular_definition = guarda a regra para separar as palavras que compõem o dicionário.

words = vetor que guarda a lista de palavras que estão sendo processadas para compor o dicionário.

phrases = vetor que guarda as frases que foram usadas para construir o dicionário.

dict_words = Dicionário que contém as palavras estruturadas para solução do problema 1.

dict_2words = Dicionário que contém as palavras estruturadas para solução do problema 2.

Utils -> classe que contém funções estáticas auxiliares.

Escolhas do desenvolvedor:

Utilizar a biblioteca nltk:

Para tratar as stop-words, pois esta é uma lib open souce que foi criada com intuito de auxiliar no processamento de linguagens naturais.

Criação de Variáveis :

words, phrases : Foi criada como vetor pois precisava somente guardar valores para iteração em algumas funções, não precisaria de funções avançadas como de busca por valor.

dict_words, dict_2words : Foram criadas como dict pela estrutura chave : valor, além de possuírem um bom desempenho para busca de valores (como buscar um valor com a key =x).