Trabalho Fundamentos de Programação

Luís Cruz
Mateus Rebêlo
Raphael Barros

Introdução

Neste trabalho, uma empresa fictícia busca desenvolvedores para sua equipe de TI e nossa equipe está no processo seletivo. O objetivo do concurso é desenvolver um programa básico de análise de sentimentos, capaz de classificar frases de acordo com uma nota atribuída a ela, podendo também aprender palavras novas e dar pesos correspondentes a elas a partir da classificação da frase em que a palavra se encontra.

Requisitos

Os principais requisitos do trabalho foram:

- Criar um programa em Python que iria:
    - Ler um arquivo em formato .txt, contendo as palavras escolhidas pela equipe e seus pesos;
    - Ler um arquivo em formato .txt, contendo as frases escolhidas pela equipe e seus pesos de referência;
    - Salvar um arquivo em formato .csv com cabeçalho e contendo a frase analisada, lista das palavras reconhecidas, resultado do cálculo de sentimento e palavras aprendidas;

Processo

A equipe definiu previamente um processo a ser seguido, consistindo de:

- Organização e ideação dos arquivos;
- Ideação e teste do código;
- Desenvolvimento do código principal;
- Debugging do código;
- Análise dos resultados;

Organização e Ideação dos arquivos

Começando pelo arquivo de frases, decidimos separar cada frase em uma linha, pois facilita para programar a aplicação que varre o arquivo, além disso, decidimos separar as frases dos pesos de referência individuais usando o pipe (|) para facilitar o uso do método .split() dentro do python, tendo em vista que uma frase pode conter vários espaços, achamos mais prático selecionar um caractere pouco usado para a separação.

```
vou comer meu ovo cozido|2|2|2
fui bem na lista de eldrey|5|5|5
eu amo viver|4|5|4
estou me sentindo muito bem|4|4|5
```

Vale notar que os pesos correspondem respectivamente aos membros Luís | Raphael | Mateus.

Seguindo para o arquivo de palavras, decidimos novamente colocar cada palavra e seu peso em linhas diferentes, dessa vez usando 1 (um) espaço para separar a palavra do seu peso, pois não há necessidade de considerar exceções, como havia no arquivo anterior.

```
desprezo -1
ressaca -1
nascer 0
crescer 0
ajudar 1
levantar 1
alegria 2
```

Finalmente, no arquivo de saída, em formato .csv, escolhemos novamente o pipe (|) para separação dos valores ao invés das vírgulas, como é o padrão, pois frases também podem conter vírgulas, e isso atrapalharia na

interpretação do que está escrito.

```
frase|palavras_reconhecidas|res_calc_sentimento|palavras_aprendidas
eu vou matar alguém|['matar']|-5.0|['eu', 'vou', 'alguém']
ele quer esquartejar minha família|['esquartejar']|-4.0|['ele', 'que
houve um homicidio em minha casa|['homicidio', 'minha']|-4.5|['houve
a terceira guerra está chegando|['guerra']|-4.0|['a', 'terceira', 'e
quero cometer suicidio|['suicidio']|-5.0|['quero', 'cometer']
infelizmente houveram feridos no acidente|['acidente']|-3.0|['infel
```

O exemplo acima representa a aparência do arquivo após rodar o código que o cria.

Ideação e teste do código

Nessa seção apenas testamos os métodos e funções relacionados à manipulação de arquivos, para relembrar o que foi passado em aula e descobrir como otimizar a escrita do programa. Para isso, criamos um ambiente com alguns arquivos de teste e fomos brincando com qualquer coisa que vinha à cabeça.

```
# teste para o trabalho de FP

texto = open('teste.txt', 'r')
saida = open('teste.csv', 'a')
```

```
arroz 3
feijao 4
morte -5
```

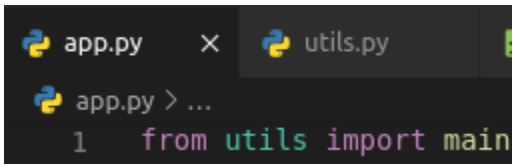
```
nova_palavra|peso
arroz|3
feijao|4
morte|-5
```

O exemplo acima mostra partes pequenas e simples dos testes que fizemos.

A partir de então, pudemos começar a pensar na lógica do programa, que foi desenvolvida pelo Discord com Luís transmitindo a tela.

Desenvolvimento do código principal

Após a ideação, partimos diretamente para o desenvolvimento. Primeiramente buscamos escrever toda a lógica em um único arquivo, para só depois pensar na modularização. Tentamos manter a nomeação de variáveis o mais claro possível, usar o try-except e funções. Dando uma olhada geral, o programa ficou separado em dois arquivos, o app.py (principal) e o utils.py (módulo). O utils.py conteve a função principal do programa, e escolhemos colocá-la no módulo para simplificar a leitura do código, que ficava relativamente poluído com tudo junto.



Usamos o try-except na abertura de arquivos para prever quaisquer erros que possam ocorrer ao tentar abrir, seja um nome de arquivo errado, algum ter sido deletado, entre outros. Também foi utilizado na chamada da função principal, para esclarecer quaisquer erros que possam acontecer na execução.

```
try:
    palavras = open('palavras.txt', 'r')
    frases = open('frases.txt', 'r')

    # inicia e escreve o cabeçalho do csv
    saida = open('saida.csv', 'w')
    saida.write('frase|palavras_reconhecidas|res_calc_sentimento|palavras_aprendidas')
    saida.close()

    # abre novamente o csv em modo append para escrita linha por linha
    saida = open('saida.csv', 'a')

except Exception as e:
    print(f'não foi possivel abrir algum arquivo devido a {e}')
```

Para a função principal, decidimos usar um loop que varre cada linha do arquivo de frases e realiza todas as operações para obter as informações necessárias para o arquivo .csv, e então escreve a saída linha por linha.

```
def main(palavras, frases, saida):
    for line in frases.readlines():
        paraAdicionar = ''
        linha = line.split('|')
        paraAdicionar = paraAdicionar + linha[0] + '|'
```

O exemplo acima demonstra a obtenção da frase lida em dada linha, para ser reservada e posteriormente introduzida ao .csv.

Em uma parte específica do código decidimos abrir um arquivo usando um with statement, pois era uma operação rápida e o with fecha o arquivo sem precisar que o programador ordene.

```
with open('palavras.txt', 'a') as palavrasAppend:
    for item in palavrasAprend:
        palavrasAppend.write(f'\n{item} {peso}')
```

No exemplo acima, abrimos o arquivo palavras.txt em modo append, para introduzir nele as palavras aprendidas na frase que o for está tratando no momento, junto com seu respectivo peso.

Debugging do código

Durante o desenvolvimento, encontramos diversos bugs, como é de se esperar em qualquer desenvolvimento de software. Para resolução dos problemas, usamos um arquivo separado para testar cada trecho separadamente e identificar a real origem do problema, por exemplo, se uma lista retornasse vazia embora tivéssemos ordenado que itens fossem colocados nela, testávamos separadamente o trecho de código que introduz itens na lista, e também a origem dos dados a serem inseridos. O grupo não teve muitos problemas com bugs, o maior problema foi o posicionamento do cursor dentro dos arquivos, que por um tempo nos confundiu um pouco mas logo descobrimos como solucionar.

```
for palavra in linha[0].split():
    palavras.seek(0)
    for line in palavras.readlines():
        if line.split()[0] == palavra:
            palavrasRecog.append(palavra)
            pesos.append(float(line.split()[1]))
            peso = sum(pesos) / len(pesos)
```

O exemplo acima aponta o trecho de reconhecer ou não palavras já “memorizadas”. Caso não houvesse o método .seek() retornando o cursor para a posição 0 (zero), isso é, para o absoluto início do arquivo, o loop não seria capaz de reconhecer palavras após a primeira lida.

Análise dos resultados

Por fim, após diversos testes e observações, temos um ponto específico a discutir. Nas primeiras frases o resultado é consistente com as opiniões do grupo acerca de cada frase, porém a situação muda um pouco com o passar das análises, a razão disso é a forma com a qual o programa aprende palavras novas. O aprendizado acontece adicionando novas palavras ao arquivo palavras.txt e atribuindo a elas o peso da respectiva frase em que ela foi descoberta. Contudo, a palavra “eu” pode ter sido descoberta na frase em que “matar”, de peso -5, se encontra. Então “eu” recebe peso -5 e prejudica a classificação de frases subsequentes como “eu amo viver”, assim como a classificação de palavras que podem ser descobertas nessas frases subsequentes. Basicamente uma bola de neve de influência. Uma solução para o problema poderia ser a criação de um arquivo contendo os pronomes, artigos, advérbios e outras palavras da língua portuguesa que não deveriam possuir peso.

```
vou comer meu ovo cozido|2|2|2
fui bem na lista de eldrey|5|5|5
eu amo viver|4|5|4
estou me sentindo muito bem|4|4|5
gosto de ajudar velhinhas|5|4|4
criei uma casa de adoção para gatos de rua|5|5|5
```

```
vou comer meu ovo cozido|['vou', 'comer', 'meu']|-1.6666666666666667|['ovo', 'cozido']
fui bem na lista de eldrey|['bem', 'na']|4.0|['fui', 'lista', 'de', 'eldrey']
eu amo viver|['eu', 'viver']|0.0|['amo']
estou me sentindo muito bem|['estou', 'sentindo', 'muito', 'bem']|0.75|['me']
gosto de ajudar velhinhas|['gosto', 'de', 'ajudar']|2.6666666666666665|['velhinhas']
criei uma casa de adoção para gatos de rua|['casa', 'de', 'gatos', 'de']|1.625|['criei', 'uma', 'adoção', 'para', 'rua']
```

Os exemplos acima demonstram um trecho em que é observável uma leve discrepância.

É importante ressaltar que a amostra apresentada é pequena, e caso fosse maior, a divergência seria ainda mais perceptível.

Conclusão

Em reunião, um dia após conclusão da parte técnica do projeto, o grupo discutiu suas opiniões para chegar a um consenso. O veredito é que o programa funciona perfeitamente como pretendido, porém as pretensões são falhas, visto que não se considera a possibilidade de palavras sem peso, o que atrapalha consideravelmente em amostras maiores. Logo, o programa na situação que está não poderia ser utilizado na vida real, pois a análise de sentimento seria imprecisa na maioria dos casos, após certo número de amostras. Porém com a solução sugerida na seção Análise de resultados, talvez fosse possível filtrar palavras sem peso e tornar os resultados mais precisos.