Pontifícia Universidade Católica do Rio de Janeiro

Departamento de Informática

Projeto final de programação
Similaridade de documentos com NLP

Autoria:

Vanessa Eufrauzino Pacheco

Orientação:

Sérgio Lifshitz

Sumário

Lista de Figuras

1 Introdução

O processamento de linguagem natural (NLP) tem crescido com o passar do tempo e se expandido, devido explosão de dados não estruturados como comentários em rede social, critica de filmes entre outros. E para uma gama de empresa, analisar e classificar esses comentários e criticas se tornou indispensável para o bem estar de seus negócios.

Entretanto NLP pode ser usado em outros contextos, como será observado nesse trabalho, que tem como principal finalidade exibir o similaridade entre textos a partir de processos de extração de arquivos já pré processados.

2 Software

O sistema possui as funcionalidades de transformar arquivos de texto (.docx ou .pdf) em objetos *Python* e a partir de então utilizar a ferramenta *Spacy* para obter a similaridade entre um ou mais documentos.

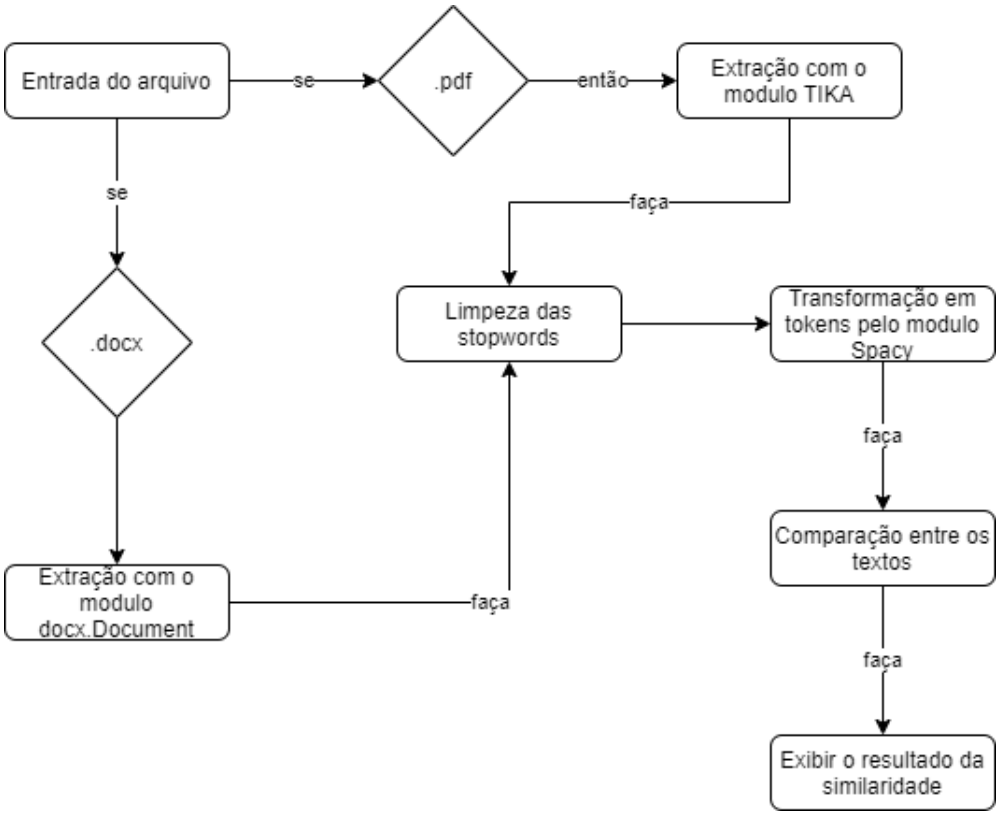

Figura 1: Fluxo do sistema

A programação foi construída toda na linguagem *Python* e utilizado o *Jupyter notebook* pela simplicidade no desenvolvimento.

Para que o software seja considerado um bom produto, é necessário que se garanta os seguintes Indicadores de Qualidade:

- (1) REPRODUTIBILIDADE dos resultados reportados;
- (2) CONFIABILIDADE da execução(código livre de bugs e fácil manutenção) e;
- (3) CLAREZA de uso(documentação assertiva e enxuta).

Com base nestes indicadores, constrói-se uma série de testes que buscam auferir e controlar a qualidade do programa.

Para correta utilização do software, o usuário precisa primeiramente estar ciente para que serve cada uma das funcionalidades. Visando oferecer a CLAREZA para o usuário, foi desenvolvido um arquivo "README", que contém as principais instruções do que faz cada funcionalidade e como operar cada uma delas.

3 Testes

O *Python* tem um eco-sistema de módulos que tem algumas ferramentas que podem manipular diretamente isso, e esses módulos são fáceis de instalar com a ferramenta "pip". Toda a comunicação entre o código *Python* e as bibliotecas instaladas, já são bastante testadas e documentadas, portanto, não houve necessidade de realizar testes de integração.

```
!pip install pytest
!pip install unittest
!pip install tika
!pip install PyPDF2
!pip install python-docx
!pip install -U spacy
!python -m spacy download pt_core_news_sm
!python -m spacy download pt_core_news_md
```

```
import os
import warnings
import requests
import PyPDF2
import numpy as np
import pandas as pd
import string
import nltk
import pytest
import unittest
import spacy #NLP e Machine Learning
parser = English()
from PyPDF2 import PdfFileWriter, PdfFileReader
from tqdm import tqdm_notebook
from sklearn import utils
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk import word_tokenize,sent_tokenize
from string import punctuation
from tika import parser
warnings.filterwarnings("ignore")
```

Figura 2: Instalação e importação dos módulos necessários

Os testes realizados foram testes unitários, utilizando a biblioteca unittest e pytest, o qual se adaptava melhor ao *Jupyter notebook*. A transformação de dados de arquivos de texto em objetos

dentro do sistema foi alvo apenas de testes manuais, pois possui usabilidade simples e não é o objetivo principal do sistema. Os testes sobre o sistema são feitos utilizando um mesmo documento para a assertividade e documento de uma língua completamente distinta para a falha.

O caso de teste descreve qual a característica que busca ser avaliada e para cada caso existe um resultado esperado e um indicador atrelado. O resultado esperado se refere ao valor de saída, que deve ser assegurado pela ferramenta. Nos itens a seguir explica-se brevemente o sindicado de cada caso de teste:

- caminhos incorretos: Passa um dos parâmetros de forma incorreta. Deve dar erro.
- caminhos inexistentes: Indica que vai passar um parâmetro, mas omite seu valor. Deve dar erro.
- parâmetro obrigatório: Omite-se um parâmetro mandatório. Deve dar erro.
- documentos em execuções erradas: A entrada de arquivo .docx deve dar erro na função de extração de .pdf e vice e versa
- documentos em execuções correta: A entrada de arquivo .docx na função de extração de .doc tem que ser totalmente acertiva assim como a função de extração, tem que funcionar de maneira correta com arquivos .pdf.
- predição idêntica: Executa-se a funcionalidade duas vezes com os mesmos documentos, o resultado deve ser o mesmo. Deve ser a mesma similaridade.
- predição exata: Executa-se a funcionalidade com o mesmo documentos, o resultado deve ser 100%. Deve ter a similaridade 1.
- predição baixa: Executa-se a funcionalidade com documentos de línguas distintas, o resultado deve ser baixo, se aproximando de zero.
- conferir se os resultados estão salvos: Executa-se a função e busca o resultado salvo na pasta. Deve estar presente o arquivo.
- conferir se os resultados estão salvos na pasta correta: Executa-se a função e busca o resultado salvo na pasta na pasta indicada. Deve estar presente o arquivo.

Todos os testes são realizados e devem passar sem nenhuma falha, desta forma obtém-se 100% de cobertura sobre o código.

4 Conclusão

O projeto foi implementado com sucesso dentro das especificações descritas. Os testes de caso foram criados e testados com sucesso, procurando assim garantir os indicadores de CONFIABILIDADE e REPRODUTIBILIDADE. A documentação está elaborada, busca dar CLAREZA ao entendimento do sistema. Desta forma, a ferramenta aqui implementada fornece um ambiente promissor para um produto mínimo viável.

Todos os arquivos disponíveis em: $https://github.com/vepacheco/inf2102$