Documento Técnico: Geração de Resumos de Texto com OpenNLP em Java

Introdução

Este documento técnico aborda duas abordagens para a geração de resumos de texto utilizando a biblioteca OpenNLP em Java. Ambas as abordagens se concentram em processar o texto para identificar e extrair informações essenciais, mas diferem na forma como apresentam o resumo. O OpenNLP é uma biblioteca de Processamento de Linguagem Natural (NLP) que fornece ferramentas para tarefas como detecção de sentenças, tokenização, e análise de texto.

Requisitos

- Java 17 ou superior
- Apache OpenNLP: Biblioteca utilizada para tarefas de NLP.
- Modelo de Detecção de Sentenças: en-sent.bin, localizado no diretório Documents/models.

Processamento de Linguagem Natural (NLP)

O NLP é uma subárea da inteligência artificial que se concentra na interação entre computadores e linguagem humana. A biblioteca OpenNLP é uma ferramenta poderosa para realizar tarefas de NLP, como:

- Detecção de Sentenças: Dividir um texto em sentenças.
- Tokenização: Dividir um texto em palavras ou outros elementos.
- Análise de Texto: Avaliar o significado ou a importância das palavras e frases em um texto.

Abordagem 1: ResumindoTextoNlp_tipo1

Descrição

A classe ResumindoTextoNlp_tipo1 é projetada para gerar um resumo a partir de um texto longo. O processo inclui a detecção de sentenças, cálculo da frequência das palavras e pontuação das sentenças. O resumo é obtido selecionando as sentenças com as maiores pontuações.

Código

```
package com.resumeTexto;

import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;

import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.file.Paths;
import java.util.*;
import java.util.stream.Collectors;

public class ResumindoTextoNlp_tipo1 {

   public static void main(String[] args) {

            System.out.println("**** ResumindoTextoNlp_tipo1 ****");

      String frase = "Com 13 álbuns lançados, o cantor nascido em Arari, no Maranhão, atravessou gerações com inúmeros hits e faixas muito populares. Os três primeiros álbuns de Zeca, aliás, conquistaram o disco de ouro, com mais de 100 mil cópias vendidas, e levaram o cantor a concorrer três vezes ao Grammy Latino, incluindo na categoria “Melhor Álbum Pop”, em 2000, com o disco Líricas";
      System.out.println("Frase: " + frase);

      String modelPath = Paths.get(System.getProperty("user.home"), "Documents", "models", "en-sent.bin").toString();

      try (InputStream modelIn = new FileInputStream(modelPath)) {
         SentenceModel model = new SentenceModel(modelIn);
         SentenceDetectorME sentenceDetector = new SentenceDetectorME(model);

         String[] sentences = sentenceDetector.sentDetect(frase);

         Map<String, Integer> wordFrequency = new HashMap<>();
         for (String sentence : sentences) {
            String[] words = sentence.split("\\W+"); // Divide a sentença em palavras
            for (String word : words) {
               word = word.toLowerCase();
               wordFrequency.put(word, wordFrequency.getOrDefault(word, 0) + 1);
            }
         }

         List<Map.Entry<String, Integer>> sortedWords = wordFrequency.entrySet()
               .stream()
               .sorted(Map.Entry.<String, Integer>comparingByValue().reversed())
```

```
                .collect(Collectors.toList());

        Map<String, Integer> sentenceScores = new HashMap<>();
        for (String sentence : sentences) {
            int score = 0;
            String[] words = sentence.split("\\W+");
            for (String word : words) {
                word = word.toLowerCase();
                score += wordFrequency.getOrDefault(word, 0);
            }
            sentenceScores.put(sentence, score);
        }

        List<String> importantSentences = sentenceScores.entrySet()
                .stream()
                .sorted(Map.Entry.<String, Integer>comparingByValue().reversed())
                .limit(2) // Limita o número de sentenças para um resumo mais curto
                .map(Map.Entry::getKey)
                .collect(Collectors.toList());

        // Exibir o resumo
        System.out.println("Resumo:");
        for (String sentence : importantSentences) {
            System.out.println(sentence);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Funcionamento

1. **Entrada**: Uma frase longa é fornecida.
2. **Detecção de Sentenças**: O modelo en-sent.bin do OpenNLP divide o texto em sentenças.
3. **Cálculo da Frequência das Palavras**: Conta a frequência de cada palavra nas sentenças.
4. **Pontuação das Sentenças**: Cada sentença recebe uma pontuação com base na soma das frequências das palavras que contém.
5. **Seleção das Sentenças Importantes**: As sentenças com maior pontuação são selecionadas para o resumo.
6. **Saída**: Exibe o resumo das sentenças mais importantes.

Vantagens e Limitações

- **Vantagens**: Simplicidade e facilidade na implementação do resumo baseado em frequências de palavras.
- **Limitações**: O resumo pode não capturar a essência do texto se a importância das sentenças não estiver bem refletida nas frequências das palavras.

Abordagem 2: ResumindoTextoNlp_tipo2

Descrição

A classe ResumindoTextoNlp_tipo2 também usa o OpenNLP para gerar um resumo, mas organiza o código em métodos distintos para melhorar a modularidade. O resumo gerado é uma única frase que representa o texto original com base na pontuação das sentenças.

Código

```
package com.resumeTexto;

import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;

/**
 * Classe para gerar um resumo de uma única frase de texto utilizando o OpenNLP.
 */
public class ResumindoTextoNlp_tipo2 {

   private static final String MODEL_PATH = Paths.get(System.getProperty("user.home"), "Documents", "models", "en-sent.bin").toString();

   public static void main(String[] args) {

            System.out.println("*** ResumindoTextoNlp_tipo2 ***");

      String frase = "Com 13 álbuns lançados, o cantor nascido em Arari, no Maranhão, atravessou gerações com inúmeros hits e faixas muito populares. " +
            "Os três primeiros álbuns de Zeca, aliás, conquistaram o disco de ouro, com mais de 100 mil cópias vendidas, e levaram o cantor a concorrer " +
            "três vezes ao Grammy Latino, incluindo na categoria “Melhor Álbum Pop”, em 2000, com o disco Líricas";

      System.out.println("Texto original:\n" + frase);

      try {
         String[] sentences = detectSentences(frase);
         Map<String, Integer> wordFrequency = calculateWordFrequency(sentences);
         Map<String, Integer> sentenceScores = scoreSentences(sentences, wordFrequency);

         // Gera um resumo em uma única frase
         String summary = generateSingleSentenceSummary(sentenceScores);

         System.out.println("\nResumo:");
         System.out.println(summary);
      } catch (Exception e) {
         System.err.println("Erro ao processar o texto: " + e.getMessage());
```

```
            e.printStackTrace();
        }
    }

    /**
     * Detecta as sentenças no texto utilizando o modelo do OpenNLP.
     *
     * @param text Texto a ser processado.
     * @return Array de sentenças detectadas.
     * @throws Exception Se ocorrer um erro ao carregar o modelo ou processar o texto.
     */
    private static String[] detectSentences(String text) throws Exception {
        try (InputStream modelIn = new FileInputStream(MODEL_PATH)) {
            SentenceModel model = new SentenceModel(modelIn);
            SentenceDetectorME sentenceDetector = new SentenceDetectorME(model);
            return sentenceDetector.sentDetect(text);
        }
    }

    /**
     * Calcula a frequência das palavras nas sentenças fornecidas.
     *
     * @param sentences Sentenças a serem analisadas.
     * @return Mapa com a frequência das palavras.
     */
    private static Map<String, Integer> calculateWordFrequency(String[] sentences) {
        Map<String, Integer> wordFrequency = new HashMap<>();
        for (String sentence : sentences) {
            String[] words = sentence.split("\\W+"); // Divide a sentença em palavras
            for (String word : words) {
                word = word.toLowerCase();
                wordFrequency.put(word, wordFrequency.getOrDefault(word, 0) + 1);
            }
        }
        return wordFrequency;
    }

    /**
     * Calcula a pontuação de cada sentença com base na frequência das palavras.
     *
     * @param sentences     Sentenças a serem pontuadas.
     * @param wordFrequency Frequência das palavras para cálculo da pontuação.
     * @return Mapa com as sentenças e suas pontuações.
     */
    private static Map<String, Integer> scoreSentences(String[] sentences, Map<String, Integer>
wordFrequency) {
        Map<String, Integer> sentenceScores = new HashMap<>();
        for (String sentence : sentences) {
            int score = Arrays.stream(sentence.split("\\W+"))
                    .map(String::toLowerCase)
                    .mapToInt(word -> wordFrequency.getOrDefault(word, 0))
                    .sum();
            sentenceScores.put(sentence, score);
        }
        return sentenceScores;
    }
```

```
    /**
     * Gera um resumo em uma única frase a partir das sentenças pontuadas.
     *
     * @param sentenceScores Mapa com as sentenças e suas pontuações.
     * @return Frase resumida.
     */
    private static String generateSingleSentenceSummary(Map<String, Integer> sentenceScores) {
        return sentenceScores.entrySet()
                .stream()
                .max(Map.Entry.comparingByValue())
                .map(Map.Entry::getKey)
                .orElse("Não foi possível gerar o resumo.");
    }
}
```

Funcionamento

1. **Entrada**: Um texto longo é fornecido.
2. **Detecção de Sentenças**: Utiliza o modelo `en-sent.bin` para dividir o texto em sentenças.
3. **Cálculo da Frequência das Palavras**: Calcula a frequência de cada palavra em todas as sentenças.
4. **Pontuação das Sentenças**: Cada sentença é pontuada com base na soma das frequências das palavras que contém.
5. **Geração de Resumo em Única Frase**: Seleciona a sentença com a maior pontuação para criar um resumo de uma única frase.
6. **Saída**: Exibe o resumo em uma única frase.

Vantagens e Limitações

- **Vantagens**: Melhoria na modularidade do código e geração de um resumo mais condensado.
- **Limitações**: A abordagem pode não capturar todos os aspectos importantes do texto original, especialmente se a sentença com a maior pontuação não for a mais representativa.

Conclusão

As duas abordagens apresentadas para a geração de resumos de texto utilizam técnicas de NLP para processar e extrair informações essenciais de um texto. A primeira abordagem oferece um resumo composto por várias sentenças, enquanto a segunda gera um resumo em uma única frase. Ambas as abordagens têm suas vantagens e limitações, e a escolha entre elas deve considerar o contexto específico e os requisitos do projeto.

Referências

- Apache OpenNLP Documentation
- Tutorial de NLP com OpenNLP