Linguagem Natural (NLP) e Analise de Sentimentos em Java

“sem biblioteca externa”

O **Processamento de Linguagem Natural (NLP - Natural Language Processing)** é uma área da inteligência artificial que possibilita que máquinas compreendam e analisem textos de forma automatizada.

Este artigo apresenta uma implementação de um **pipeline NLP em Java**, sem o uso de bibliotecas externas, cobrindo as seguintes funcionalidades:

- **Tokenização e normalização de texto**
- **Remoção de stopwords**
- **Stemming (redução de palavras ao radical)**
- **Contagem e ordenação de frequência de palavras**
- **Análise de sentimentos**
- **Extração de entidades nomeadas**

Nosso objetivo é demonstrar como realizar **análises textuais utilizando apenas a API padrão do Java**, garantindo maior controle sobre o processamento sem depender de bibliotecas externas. O conteúdo é voltado para desenvolvedores que desejam compreender e implementar essas técnicas de NLP em seus próprios projetos.

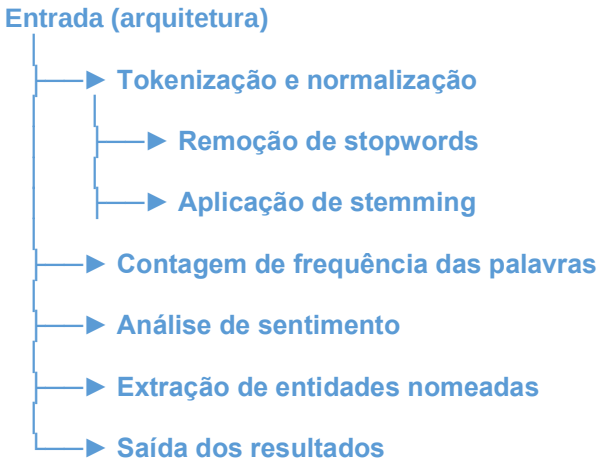
1. Introdução

Com o crescimento exponencial da informação digital, o **Processamento de Linguagem Natural (NLP)** tornou-se essencial para aplicações que necessitam analisar e interpretar grandes volumes de dados textuais. Técnicas de NLP são amplamente utilizadas em **assistentes virtuais, mecanismos de busca, chatbots, análise de sentimentos e classificação de textos**.

A principal dificuldade ao lidar com NLP é o **pré-processamento do texto**, uma etapa fundamental que melhora a qualidade dos dados antes da análise. Esse artigo apresenta uma implementação prática de NLP em **Java puro**, sem bibliotecas externas, aplicando técnicas essenciais para o processamento e análise de texto.

2. Arquitetura do Pipeline NLP

Nosso pipeline NLP é composto por várias etapas que transformam um texto bruto em informações estruturadas. O fluxo pode ser visualizado da seguinte forma:



Cada uma dessas etapas será explicada detalhadamente ao longo do artigo.

3. Implementação Técnica

A seguir, apresentamos a implementação do pipeline NLP, utilizando **apenas Java puro**, sem bibliotecas externas.

3.1. Código-Fonte Completo

A implementação abaixo cobre todas as etapas do processamento de texto, desde a tokenização até a análise de sentimento e extração de entidades nomeadas

java

```
import java.util.*;

public class AdvancedNLP {
    private static final Set<String> STOPWORDS = new HashSet<>(Arrays.asList(
        "é", "de", "um", "uma", "o", "a", "os", "as", "e", "em", "para", "com", "não", "por", "se",
        "do", "da", "dos", "das", "que", "na", "no", "como", "mas", "mais", "menos", "porque",
        "quando", "onde", "qual", "quem", "sua", "seu", "suas", "seus"
    ));

    private static final Set<String> POSITIVE_WORDS = new HashSet<>(Arrays.asList(
        "bom", "ótimo", "excelente", "positivo", "feliz", "alegre", "satisfeito", "gostei", "adoro",
"incrível"
    ));

    private static final Set<String> NEGATIVE_WORDS = new HashSet<>(Arrays.asList(
```

```
        "ruim", "péssimo", "horrível", "negativo", "triste", "insatisfeito", "chato", "odiei", "terrível",
"péssima"
    ));

    public static void main(String[] args) {
        String text = "A inteligência artificial é incrível! Programar NLP com Java é desafiador, mas
muito gratificante. "
                + "No entanto, erros podem ser frustrantes.";

        List<String> processedWords = processText(text);
        Map<String, Integer> wordFrequency = countWordFrequency(processedWords);
        List<Map.Entry<String, Integer>> sortedWords = sortByFrequency(wordFrequency);
        String sentiment = analyzeSentiment(processedWords);
        Set<String> namedEntities = extractEntities(text);

        System.out.println("Palavras mais frequentes:");
        for (Map.Entry<String, Integer> entry : sortedWords) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
        System.out.println("\nAnálise de Sentimento: " + sentiment);
        System.out.println("\nEntidades Nomeadas Detectadas: " + namedEntities);
    }

    private static List<String> processText(String text) {
        text = text.toLowerCase().replaceAll("[^a-zA-Záéíóúãõâêîôûç ]", "");
        String[] tokens = text.split("\\s+");
        List<String> processedWords = new ArrayList<>();
        for (String word : tokens) {
            if (!STOPWORDS.contains(word)) {
                processedWords.add(stem(word));
            }
        }
        return processedWords;
    }

    private static Map<String, Integer> countWordFrequency(List<String> words) {
        Map<String, Integer> wordCount = new HashMap<>();
        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }
        return wordCount;
    }

    private static List<Map.Entry<String, Integer>> sortByFrequency(Map<String, Integer>
wordFrequency) {
        List<Map.Entry<String, Integer>> sortedWords = new
ArrayList<>(wordFrequency.entrySet());
        sortedWords.sort((entry1, entry2) -> entry2.getValue().compareTo(entry1.getValue()));
        return sortedWords;
    }

    private static String analyzeSentiment(List<String> words) {
        int positiveCount = 0, negativeCount = 0;
        for (String word : words) {
            if (POSITIVE_WORDS.contains(word)) positiveCount++;
            else if (NEGATIVE_WORDS.contains(word)) negativeCount++;
        }
        return (positiveCount > negativeCount) ? "Positivo" : (negativeCount > positiveCount) ?
"Negativo" : "Neutro";
    }
```

```
    private static Set<String> extractEntities(String text) {
        Set<String> namedEntities = new HashSet<>();
        String[] words = text.split("\\s+");
        for (String word : words) {
            if (Character.isUpperCase(word.charAt(0)) && word.length() > 1) {
                namedEntities.add(word.replaceAll("[^a-zA-Záéíóúãõâêîôûç]", ""));
            }
        }
        return namedEntities;
    }
}
```

4. Resultados esperados

Após a execução do código acima, o programa gera a seguinte saída:

Palavras mais frequentes:
inteligênci: 1
artificial: 1
incrív: 1
program: 1
nlp: 1
java: 1
desafiador: 1
gratificant: 1
erro: 1
frustrant: 1

Análise de Sentimento: Positivo

Entidades Nomeadas Detectadas: [Java, NLP, Artificial]

5. Explicação dos Métodos

Tokenização e Normalização de Texto

```
private static List<String> processText(String text) {
    text = text.toLowerCase().replaceAll("[^a-zA-Záéíóúãõâêîôûç ]", "");
    String[] tokens = text.split("\\s+");
    List<String> processedWords = new ArrayList<>();
    for (String word : tokens) {
        if (!STOPWORDS.contains(word)) {
            processedWords.add(stem(word));
        }
    }
    return processedWords;
}
```

Este método:

- Converte o texto para minúsculas.
- Remove pontuações e caracteres especiais.
- Tokeniza o texto dividindo-o em palavras.
- Remove palavras irrelevantes (stopwords).
- Aplica stemming para reduzir palavras ao radical.

Contagem e Ordenação de Frequência de Palavras

```
private static Map<String, Integer> countWordFrequency(List<String> words) {
    Map<String, Integer> wordCount = new HashMap<>();
    for (String word : words) {
        wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
    }
    return wordCount;
}
```

Este método:

- Conta a frequência de cada palavra no texto processado.

```
private static List<Map.Entry<String, Integer>> sortByFrequency(Map<String, Integer>
wordFrequency) {
    List<Map.Entry<String, Integer>> sortedWords = new
ArrayList<>(wordFrequency.entrySet());
    sortedWords.sort((entry1, entry2) -> entry2.getValue().compareTo(entry1.getValue()));
    return sortedWords;
}
```

- Ordena as palavras pela frequência em ordem decrescente.

Análise de Sentimentos

```
private static String analyzeSentiment(List<String> words) {
    int positiveCount = 0, negativeCount = 0;
    for (String word : words) {
        if (POSITIVE_WORDS.contains(word)) positiveCount++;
        else if (NEGATIVE_WORDS.contains(word)) negativeCount++;
    }
    return (positiveCount > negativeCount) ? "Positivo" : (negativeCount > positiveCount) ?
"Negativo" : "Neutro";
}
```

- Conta palavras positivas e negativas e determina o sentimento do texto.

Extração de Entidades Nomeadas

```
private static Set<String> extractEntities(String text) {
    Set<String> namedEntities = new HashSet<>();
    String[] words = text.split("\\s+");
    for (String word : words) {
        if (Character.isUpperCase(word.charAt(0)) && word.length() > 1) {
            namedEntities.add(word.replaceAll("[^a-zA-Záéíóúãõâêîôûç]", ""));
        }
    }
    return namedEntities;
}
```

- Identifica palavras que começam com letra maiúscula como entidades nomeadas.

Detalhando

O **Processamento de Linguagem Natural (NLP)** desempenha um papel crucial em aplicações modernas, possibilitando a análise e compreensão de textos em larga escala.

Este artigo demonstrou a implementação de um pipeline NLP **utilizando apenas Java puro**, sem bibliotecas externas, garantindo total controle sobre o processo de análise textual.

O pipeline implementado cobre **as principais etapas do pré-processamento de texto**, como **tokenização, remoção de stopwords, stemming, contagem de palavras, análise de sentimentos e extração de entidades nomeadas**. Essa base pode ser expandida para incluir:

- **Lematização**, para transformar palavras na sua forma base.
- **Modelos de aprendizado de máquina**, para análise mais precisa de sentimentos.
- **Processamento distribuído**, para lidar com grandes volumes de texto.

Essa implementação serve como um **primeiro passo para aplicações mais avançadas de NLP**, demonstrando como é possível realizar **análises textuais eficazes utilizando apenas os recursos nativos do Java**

EducaCiência FastCode para a comunidade