Pre-processamento em Processamento de Linguagem Natural (PLN)

O Processamento de Linguagem Natural (PLN) envolve uma série de etapas de pré-processamento para transformar dados de texto bruto em um formato adequado para análise ou modelos de aprendizado de máquina.

Essas etapas ajudam a melhorar a qualidade dos dados e torná-los mais fáceis para os algoritmos entenderem e processarem o texto. Abaixo estão as principais etapas de pré-processamento usadas em PLN, junto com explicações e código exemplo.

33 Etapas de Pré-processamento comumente usadas antes de alimentar dados em um modelo PLN:

1. Conversão para minúsculas
2. Tokenização
3. Remoção de Pontuação
4. Remoção de Stopwords
5. Stemming
6. Lematização
7. Remoção de Números
8. Remoção de Espaços Extras
9. Tratamento de Contrações
10. Remoção de Caracteres Especiais
11. Etiquetagem Morfossintática (POS Tagging)
12. Reconhecimento de Entidades Nomeadas (NER)
13. Vetorização
14. Tratamento de Dados Ausentes
15. Normalização
16. Correção Ortográfica
17. Tratamento de Emojis e Emoticons
18. Remoção de Tags HTML
19. Tratamento de URLs
20. Tratamento de Menções e Hashtags
21. Segmentação de Sentenças
22. Tratamento de Abreviações
23. Detecção de Idioma
24. Codificação de Texto
25. Tratamento de Tokens de Espaço em Branco
26. Tratamento de Datas e Horários
27. Aumento de Texto
28. Tratamento de Negações
29. Análise de Dependências
30. Tratamento de Palavras Raras
31. Chunking de Texto
32. Tratamento de Sinônimos
33. Normalização de Texto para Mídias Sociais

Explicação detalhada de cada etapa de pré-processamento:

1. Conversão para minúsculas

- Propósito: Converte todo o texto para minúsculas para garantir uniformidade.
- Por quê: Reduz o tamanho do vocabulário e evita tratar a mesma palavra em diferentes casos como tokens diferentes (ex: "Maçã" vs. "maçã").

```
String text = "Olá Mundo! Isto é PLN.";
text = text.toLowerCase();
System.out.println(text);
```

2. Tokenização

- Propósito: Divide o texto em palavras, frases ou sentenças individuais (tokens).
- Por quê: Quebra o texto em unidades gerenciáveis para processamento posterior.

```
import opennlp.tools.tokenize.SimpleTokenizer;

String text = "Olá Mundo! Isto é PLN.";
SimpleTokenizer tokenizer = SimpleTokenizer.INSTANCE;
String[] tokens = tokenizer.tokenize(text);
System.out.println(Arrays.toString(tokens));
```

3. Remoção de Pontuação

- Propósito: Remove marcas de pontuação como vírgulas, pontos, pontos de exclamação, etc.
- Por quê: A pontuação frequentemente não contribui para o significado em muitas tarefas de PLN e pode adicionar ruído.

```
String text = "Olá, Mundo! Isto é PLN.";
text = text.replaceAll("[\\p{Punct}]", "");
System.out.println(text);
```

4. Remoção de Stopwords

- Propósito: Remove palavras comuns como "o", "é", "e", que não carregam significado significativo.
- Por quê: Reduz ruído e foca em palavras significativas.

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

Set<String> stopWords = new HashSet<>(Arrays.asList("o", "a", "os", "as", "um", "uma"));
String[] tokens = {"isto", "é", "uma", "frase", "exemplo"};
List<String> filteredTokens = Arrays.stream(tokens)
    .filter(word -> !stopWords.contains(word.toLowerCase()))
    .collect(Collectors.toList());
System.out.println(filteredTokens);
```

5. Stemming

- Propósito: Reduz palavras à sua forma raiz cortando sufixos (ex: "correndo" → "corr").
- Por quê: Simplifica palavras para sua forma base, reduzindo o tamanho do vocabulário.

```
import org.tartarus.snowball.ext.PortugueseStemmer;

PortugueseStemmer stemmer = new PortugueseStemmer();
String[] words = {"correndo", "corredor", "correu"};
for (String word : words) {
    stemmer.setCurrent(word);
    stemmer.stem();
    System.out.println(stemmer.getCurrent());
}
```

6. Lematização

- Propósito: Converte palavras para sua forma base ou forma de dicionário (ex: "melhor" → "bom").
- Por quê: Mais preciso que stemming pois usa análise vocabular e morfológica.

```
import com.github.pemistahl.lingua.api.*;

Language detectedLanguage = LanguageDetector.builder()
    .fromAllLanguages()
    .build()
    .detectLanguageOf("Este é um exemplo de texto em português");
System.out.println(detectedLanguage);
```

7. Remoção de Números

- Propósito: Remove valores numéricos do texto.
- Por quê: Números podem não ser relevantes em certas tarefas de PLN como análise de sentimentos.

```
String text = "Existem 3 maçãs e 5 laranjas.";
text = text.replaceAll("\\d+", "");
System.out.println(text);
```

8. Remoção de Espaços Extras

- Propósito: Elimina múltiplos espaços, tabs ou quebras de linha.
- Por quê: Garante formatação de texto limpa e consistente.

```
String text = "  Esta é uma   frase.  ";
text = text.trim().replaceAll("\\s+", " ");
System.out.println(text);
```

9. Tratamento de Contrações

- Propósito: Expande contrações (ex: "tô" → "estou").
- Por quê: Padroniza o texto para melhor processamento.

```
import java.util.HashMap;
import java.util.Map;

Map<String, String> contractions = new HashMap<>();
contractions.put("tô", "estou");
contractions.put("tá", "está");

String text = "Eu tô bem e ele tá mal.";
for (Map.Entry<String, String> entry : contractions.entrySet()) {
    text = text.replaceAll("\\b" + entry.getKey() + "\\b", entry.getValue());
}
System.out.println(text);
```

10. Remoção de Caracteres Especiais

- Propósito: Remove caracteres não alfanuméricos como @, #, $, etc.
- Por quê: Reduz ruído e símbolos irrelevantes.

```
String text = "Este é um #texto exemplo com @caracteres especiais!";
text = text.replaceAll("[^a-zA-Z0-9\\s]", "");
System.out.println(text);
```

11. Etiquetagem Morfossintática (POS Tagging)

- Propósito: Atribui tags gramaticais às palavras (ex: substantivo, verbo, adjetivo).
- Por quê: Ajuda a entender a estrutura sintática das frases.

```
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;

InputStream modelIn = new FileInputStream("pt-pos-maxent.bin");
POSModel model = new POSModel(modelIn);
POSTaggerME tagger = new POSTaggerME(model);

String[] tokens = {"Esta", "é", "uma", "frase", "exemplo"};
String[] tags = tagger.tag(tokens);
System.out.println(Arrays.toString(tags));
```

12. Reconhecimento de Entidades Nomeadas (NER)

- Propósito: Identifica e classifica entidades como nomes, datas, localizações, etc.
- Por quê: Útil para tarefas como extração de informação.

```
import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;

InputStream modelIn = new FileInputStream("pt-ner-person.bin");
TokenNameFinderModel model = new TokenNameFinderModel(modelIn);
NameFinderME nameFinder = new NameFinderME(model);

String[] tokens = {"João", "trabalha", "na", "Google", "em", "São", "Paulo"};
Span[] nameSpans = nameFinder.find(tokens);
for (Span span : nameSpans) {
    System.out.println(span.toString());
}
```

13. Vetorização

- Propósito: Converte texto em vetores numéricos (ex: Bag of Words, TF-IDF, Word Embeddings).
- Por quê: Modelos de aprendizado de máquina requerem entrada numérica.

```
import org.apache.commons.text.similarity.CosineDistance;

String[] corpus = {"Esta é uma frase exemplo.", "Outra frase exemplo."};
Map<String, Integer> vocabulary = new HashMap<>();
List<Map<String, Integer>> vectors = new ArrayList<>();

// Criar vocabulário
for (String doc : corpus) {
    String[] tokens = doc.toLowerCase().split("\\s+");
    for (String token : tokens) {
        vocabulary.putIfAbsent(token, vocabulary.size());
    }
}

// Criar vetores
for (String doc : corpus) {
    Map<String, Integer> vector = new HashMap<>();
    String[] tokens = doc.toLowerCase().split("\\s+");
    for (String token : tokens) {
        vector.merge(token, 1, Integer::sum);
    }
    vectors.add(vector);
}

System.out.println("Vocabulário: " + vocabulary);
System.out.println("Vetores: " + vectors);
```

14. Tratamento de Dados Ausentes

- Propósito: Preenche ou remove dados de texto ausentes ou incompletos.
- Por quê: Garante que o conjunto de dados esteja completo e consistente.

```
import java.util.ArrayList;
import java.util.List;

List<String> texts = new ArrayList<>();
texts.add("Olá");
texts.add(null);
texts.add("Mundo");

// Substituir valores nulos
texts.replaceAll(text -> text == null ? "VALOR_PADRAO" : text);
System.out.println(texts);
```

15. Normalização

- Propósito: Padroniza o texto (ex: convertendo todas as datas para um único formato).
- Por quê: Garante consistência no conjunto de dados.

```
import java.text.Normalizer;

String text = "Café";
text = Normalizer.normalize(text, Normalizer.Form.NFD)
    .replaceAll("[^\\p{ASCII}]", "");
System.out.println(text);
```

16. Correção Ortográfica

- Propósito: Corrige erros ortográficos no texto.
- Por quê: Melhora a qualidade do texto para análise.

```
 java
import org.languagetool.JLanguageTool;
import org.languagetool.language.BrazilianPortuguese;

JLanguageTool langTool = new JLanguageTool(new BrazilianPortuguese());
String text = "Eu fiz muitos erros em inteligencia artifical";
List<RuleMatch> matches = langTool.check(text);

for (RuleMatch match : matches) {
    System.out.println("Erro encontrado: " + match.getMessage());
    System.out.println("Sugestões: " + match.getSuggestedReplacements());
}
```

17. Tratamento de Emojis e Emoticons

- Propósito: Converte emojis e emoticons em texto ou os remove.
- Por quê: Emojis podem carregar sentimento ou significado que precisa ser capturado.

```
import com.vdurmont.emoji.EmojiParser;

String text = "Eu amo Java! 😊";
// Converter emojis para texto
String withoutEmoji = EmojiParser.removeAllEmojis(text);
System.out.println(withoutEmoji);

// Extrair descrições dos emojis
String emojiDescription = EmojiParser.parseToAliases(text);
System.out.println(emojiDescription);
```

18. Remoção de Tags HTML

- Propósito: Remove tags HTML de texto extraído da web.
- Por quê: Tags HTML são irrelevantes para a maioria das tarefas de PLN.

```
 import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;

String html = "<p>Este é um <b>texto</b> exemplo.</p>";
Document doc = Jsoup.parse(html);
String text = doc.text();
System.out.println(text);
```

19. Tratamento de URLs

- Propósito: Remove ou substitui URLs no texto.
- Por quê: URLs são frequentemente irrelevantes para análise de texto.

```
 String text = "Visite meu site em https://exemplo.com.";
text = text.replaceAll("https?://\\S+|www\\.\\S+", "");
System.out.println(text);
```

20. Tratamento de Menções e Hashtags

- Propósito: Processa ou remove menções de mídia social (@usuario) e hashtags (#topico).
- Por quê: Útil para análise de texto de mídia social.

```
String text = "Olá @usuario, confira #PLN!";
text = text.replaceAll("@\\w+|#\\w+", "");
System.out.println(text);
```

21. Segmentação de Sentenças

- Propósito: Divide texto em sentenças individuais.
- Por quê: Importante para tarefas como tradução automática ou sumarização.

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

InputStream modelIn = new FileInputStream("pt-sent.bin");
SentenceModel model = new SentenceModel(modelIn);
SentenceDetectorME sentenceDetector = new SentenceDetectorME(model);

String text = "Esta é a primeira frase. Esta é a segunda frase.";
String[] sentences = sentenceDetector.sentDetect(text);
System.out.println(Arrays.toString(sentences));
```

22. Tratamento de Abreviações (continuação)

```
Map<String, String> abbreviations = new HashMap<>();
Abbreviations.put(“prof.”, “professor”);
Abbreviations.put(“dr.”, “doutor”);
Abbreviations.put(“etc.”, “et cetera”);

String text = “O prof. Estará aqui em breve.”;
For (Map.Entry<String, String> entry : abbreviations.entrySet()) {
    Text = text.replaceAll(\\b + entry.getKey() + \\b, entry.getValue());
}
System.out.println(text);
```

23. Detecção de Idioma

- Propósito: Identifica o idioma do texto.
- Por quê: Garante que o modelo PLN correto seja aplicado.

```
Import com.github.pemistahl.lingua.api.LanguageDetector;
Import com.github.pemistahl.lingua.api.Language;

LanguageDetector detector = LanguageDetector.builder()
    .fromAllLanguages()
    .build();

String text = “Este é um texto em português.”;
Language language = detector.detectLanguageOf(text);
System.out.println(language);
```

24. Codificação de Texto

- Propósito: Converte texto para um formato de codificação específico (ex: UTF-8).
- Por quê: Garante compatibilidade com ferramentas e modelos PLN.

```
String text = “Café”;
Byte[] bytes = text.getBytes(“UTF-8”);
String decoded = new String(bytes, “UTF-8”);
System.out.println(decoded);
```

25. Tratamento de Tokens de Espaço em Branco

- Propósito: Remove ou processa tokens que são apenas espaços ou strings vazias.
- Por quê: Garante tokens limpos e significativos.

```
List<String> tokens = Arrays.asList(“Este”, “ “, “é”, “ “, “um”, “ “, “exemplo”, “ “);
List<String> cleanTokens = tokens.stream()
    .filter(token -> !token.trim().isEmpty())
    .collect(Collectors.toList());
System.out.println(cleanTokens);
```

26. Tratamento de Datas e Horários

- Propósito: Padroniza ou extrai formatos de data e hora.
- Por quê: Útil para análise sensível ao tempo.

```
Import java.time.LocalDateTime;
Import java.time.format.DateTimeFormatter;

String text = “O evento é em 15/10/2023.”;
DateTimeFormatter formatter = DateTimeFormatter.ofPattern(“dd/MM/yyyy”);
LocalDateTime date = LocalDateTime.parse(“15/10/2023 00:00”,
    DateTimeFormatter.ofPattern(“dd/MM/yyyy HH:mm”));
System.out.println(date);
```

27. Aumento de Texto

- Propósito: Gera dados de treinamento adicionais modificando texto existente (ex: substituição por sinônimos).
- Por quê: Melhora a robustez e performance do modelo.

```
Import edu.mit.jwi.Dictionary;
Import edu.mit.jwi.item.*;

Dictionary dict = new Dictionary(new File(“dict”));
Dict.open();

String word = “feliz”;
IIndexWord idxWord = dict.getIndexWord(word, POS.ADJECTIVE);
```

```
IWordID wordID = idxWord.getWordIDs().get(0);
IWord iWord = dict.getWord(wordID);
ISynset synset = iWord.getSynset();

// Obter sinônimos
List<String> synonyms = new ArrayList<>();
For (IWord w : synset.getWords()) {
    Synonyms.add(w.getLemma());
}
System.out.println(synonyms);
```

28. Tratamento de Negações

- Propósito: Identifica e processa negações (ex: “não bom”).
- Por quê: Importante para análise de sentimentos e compreensão de contexto.

```
String text = “Isto não é bom.”;
String[] tokens = text.split(\\s+);
List<String> processedTokens = new ArrayList<>();

For (int i = 0; i < tokens.length; i++) {
    If (tokens[i].equals(“não”) && i + 1 < tokens.length) {
        processedTokens.add(“não_” + tokens[i + 1]);
        i++;
    } else {
        processedTokens.add(tokens[i]);
    }
}
System.out.println(processedTokens);
```

29. Análise de Dependências

- Propósito: Analisa a estrutura gramatical de uma frase.
- Por quê: Ajuda a entender relações entre palavras.

```
Import edu.stanford.nlp.pipeline.*;
Import edu.stanford.nlp.trees.*;
Import edu.stanford.nlp.util.*;

Properties props = new Properties();
Props.setProperty(“annotators”, “tokenize,ssplit,pos,lemma,depparse”);
Props.setProperty(“language”, “portuguese”);

StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
CoreDocument document = new CoreDocument(“Esta é uma frase exemplo.”);
Pipeline.annotate(document);

For (CoreSentence sentence : document.sentences()) {
    System.out.println(sentence.dependencyParse());
}
```

30. Tratamento de Palavras Raras

- Propósito: Substitui ou remove palavras que ocorrem com pouca frequência.
- Por quê: Reduz ruído e melhora a eficiência do modelo.

```
 Map<String, Integer> wordCounts = new HashMap<>();
List<String> tokens = Arrays.asList(“isto”, “é”, “uma”, “palavra”, “rara”, “palavra”);

// Contar frequência das palavras
For (String token : tokens) {
   wordCounts.merge(token, 1, Integer::sum);
}

// Substituir palavras raras
Int threshold = 2;
List<String> processedTokens = tokens.stream()
   .map(token -> wordCounts.get(token) < threshold ? “<RARO>” : token)
   .collect(Collectors.toList());
System.out.println(processedTokens);
```

31. Chunking de Texto

- Propósito: Agrupa palavras em “chunks” baseados em tags POS (ex: sintagmas nominais).
- Por quê: Útil para extração de informação.

```
// Usando OpenNLP para chunking
Import opennlp.tools.chunker.*;

InputStream modelIn = new FileInputStream(“pt-chunker.bin”);
ChunkerModel model = new ChunkerModel(modelIn);
ChunkerME chunker = new ChunkerME(model);

String[] tokens = {“Esta”, “é”, “uma”, “frase”, “exemplo”};
String[] tags = {“DET”, “V”, “DET”, “N”, “N”};
String[] chunks = chunker.chunk(tokens, tags);
System.out.println(Arrays.toString(chunks));
```

32. Tratamento de Sinônimos

- Propósito: Substitui palavras por seus sinônimos.
- Por quê: Ajuda no aumento de texto e redução de redundância.

```
Import edu.mit.jwi.Dictionary;
Import edu.mit.jwi.item.*;

Dictionary dict = new Dictionary(new File(“dict”));
Dict.open();
```

```
String word = “feliz”;
IIndexWord idxWord = dict.getIndexWord(word, POS.ADJECTIVE);
List<String> synonyms = new ArrayList<>();

If (idxWord != null) {
   For (IWordID wordID : idxWord.getWordIDs()) {
      IWord iWord = dict.getWord(wordID);
      ISynset synset = iWord.getSynset();
      For (IWord w : synset.getWords()) {
         Synonyms.add(w.getLemma());
      }
   }
}
System.out.println(synonyms);
```

33. Normalização de Texto para Mídias Sociais

- Propósito: Processa texto informal (ex: “vc” → “você”, “blz” → “beleza”).
- Por quê: Texto de mídia social frequentemente contém linguagem informal e gírias.

```
Map<String, String> socialMediaNorm = new HashMap<>();
socialMediaNorm.put(“vc”, “você”);
socialMediaNorm.put(“blz”, “beleza”);
socialMediaNorm.put(“tb”, “também”);

String text = “blz vc tb vai?”;
For (Map.Entry<String, String> entry : socialMediaNorm.entrySet()) {
   Text = text.replaceAll(\\b + entry.getKey() + \\b, entry.getValue());
}
System.out.println(text);
```

Resumo da Importância das Etapas de Pré-processamento

Embora tokenização, conversão para minúsculas, remoção de stopwords e vetorização sejam universalmente importantes, a relevância de outras etapas depende da tarefa e do conjunto de dados.

Sempre analise seus dados e requisitos da tarefa para determinar as etapas de pré-processamento mais críticas.

Importância Específica por Tarefa:
- Análise de Sentimentos: O tratamento de negações, emojis e emoticons é crucial
- Tradução Automática: A segmentação de sentenças e etiquetagem POS são importantes
- Reconhecimento de Entidades Nomeadas (NER): O tratamento de datas, horários e caracteres especiais é crítico
- Análise de Mídias Sociais: O tratamento de emojis, hashtags e linguagem informal é essencial
- Classificação de Texto: A remoção de stopwords, conversão para minúsculas e vetorização são fundamentais

EducaCiência FastCode para a comunidade