LINGUAGENS DE DOMÍNIO

ScribeDSL

RELATÓRIO

Mestrado em Engenharia Informática
2024

Ana Teixeira - fc56336

Domain analysis

1. Purpose. What is the purpose of the language? Use cases?

O propósito da linguagem é facilitar o processamento de texto, fornecendo uma DSL interna para análise, pré-processamento e manipulação de texto. O objetivo principal é simplificar tarefas comuns de processamento de texto, tornando o desenvolvimento de aplicações de análise de texto mais intuitivo e acessível. Com a ScribeDSL, os utilizadores podem executar várias operações, como a tokenização, a remoção de stop words, o stemming e a análise de frequência de palavras de forma eficiente e direta.

Casos de uso: A ScribeDSL pode ser utilizada para analisar o conteúdo de documentos, atribuindo categorias com base em palavras-chave, podendo assim gerar automaticamente resumos de texto e analisar a relevância e frequência das palavras-chave em documentos extensos, identificar as informações mais importantes e resumir o conteúdo de forma concisa e informativa.

2. Stakeholders. Who are the stakeholders and the intended users of the language?

Os stakeholders desta linguagem são desenvolvedores de software que precisam de realizar operações de processamento de texto nas suas aplicações e projetos, analistas de dados que precisam de analisar grandes volumes de texto e equipas de marketing de redes sociais que têm interesse em analisar o conteúdo das redes sociais e identificar padrões e tendências relevantes para suas estratégias de marketing.

3. Concepts. What are the key domain concepts?

Os conceitos chave do domínio são Text, Stop word, Token, Stem, Word frequency e Processor. O Text representa o texto original a ser processado, enquanto que o Token representa uma unidade de texto após a tokenização. Stop word representa uma palavra que deve ser removida durante o pré-processamento, pois não contribui significativamente para a análise. Word frequency indica a frequência com que cada token aparece no texto. O Processor representa o processador de texto que executa as operações de processamento. É importante notar que Stem também é um conceito relevante neste contexto, pois representa um token após a operação de stemming. No entanto, não é necessário incluí-lo no meta-modelo, uma vez que a classe PorterStemmer é responsável por aplicar o stemming nas palavras.

4. Relations. How are the concepts related and what are their relevant properties?

Existem várias relações entre estes conceitos. O *Processor* está relacionado com os conceitos *Text, Token, Stop Word, Stem* e *Word Frequency* e desempenha papéis específicos em relação a cada um desses conceitos: é responsável por manipular o texto original (conceito Text) transformando-o num conjunto de tokens (conceito Token); filtra a lista de tokens de modo a remover as stop words (conceito Stop Word); aplica a transformação de stemming a todos os tokens (conceito Stem) e é também responsável por analisar a frequência de cada token no texto (conceito Word Frequency).

5. Examples. What examples of language instances are available or can be prototyped?

A ScribeDSL pode ser utilizada para processar textos de redes sociais, como tweets ou posts no Facebook, com o fim de extrair informações relevantes como palavras-chave, tendências ou sentimentos dos utilizadores. Além disso, pode ser utilizada em aplicações de análise de conteúdo para identificar padrões ou características específicas em grandes quantidades de texto, como análises de sentimentos e geração de resumos automáticos.

Fluent interface

A classe TextProcessing representa uma fluent interface. Esta classe abrange as interfaces ReadText, Tokenizer, FilterStopWords, PerformStemming e AnalyseWordFrequency e, tal como o próprio nome indica, cada uma destas interfaces descreve os métodos que podem ser invocados após a sua aplicação.

Mais concretamente, inicialmente o Processor pode invocar os métodos *fromFile* ou *withData* e ambos retornam um objeto do tipo ReadText, uma interface que apenas permite a invocação do método *tokenize()*. Portanto, após a invocação de *fromFile* ou de *withData* apenas o método *tokenize()* pode ser invocado. Assim, sempre que um objeto do tipo ReadText é retornado, apenas o método tokenize() pode ser invocado de seguida. Da mesma forma, sempre que um objeto do tipo Tokenizer é retornado, um dos métodos *filterStopWords*, *performStemming*, *analyseWordFrequency* ou *build* pode ser invocado a seguir; sempre que um objeto do tipo FilterStopWords é retornado, um dos métodos *performStemming*, *analyseWordFrequency* ou *build* pode ser invocado a seguir; sempre que um objeto do tipo PerformStemming é retornado, os métodos *analyseWordFrequency* ou *build* podem ser invocados a seguir, e sempre que um objeto do tipo AnalyseWordFrequency é retornado, apenas o método *build* pode ser invocado.

Estas escolhas têm em conta que todos os textos a serem processados devem passar primeiro por um método de tokenização, independentemente de estarem ou não num ficheiro à parte. Os outros três métodos são opcionais. No entanto, a ordem pela qual são aplicados é importante: o método *filterStopWords* deve ser aplicado antes do *performStemming*, e o *analyseWordFrequency* só pode ser aplicado após os outros dois. Independentemente das escolhas dos métodos para processar o texto, o processamento terá de ser sempre finalizado com a aplicação do método *build*. Por exemplo, as seguintes aplicações seriam impossíveis nesta fluent interface:

```
ProcessedData analyseText = processor.withData(text)
        .tokenize("\\s+")
        .performStemming()
        .filterStopWords()
        .analyseWordFrequency()
        .build();
```

```
ProcessedData analyseText = processor.withData(text)
        .tokenize("\\s+")
        .analyseWordFrequency()
        .performStemming()
        .filterStopWords()
        .build();
```

Tests

Para testar o funcionamento da linguagem foram feitos diversos testes. Estes testes cobrem todas as combinações possíveis de métodos tendo em conta as suas restrições de aplicação. Cada teste exibe no ecrã a lista de tokens, uma mensagem sobre se essa lista tem stopwords ou não e, no caso de ter sido aplicado o método *analyseWordFrequency()*, exibe uma lista com os tokens processados e a frequência com que aparecem no texto fornecido.

Para correr os testes existem duas opções:

1. Utilizar o terminal: Abrir um terminal na pasta do projeto, executar o comando **cd scribedsl.mindmap** e de seguida executar o comando **java -jar runScribedslTests.jar**.
2. Utilizar o Eclipse Modelling Tools: Importar o projeto para o Eclipse e executar o ficheiro /scribedsl.mindmap/test/scribedsl/ScribedslTest.java clicando nele com o botão direito do rato e de seguida em Run As > Java Application.

Meta-model

É de notar que Processed data é o modelo da linguagem ScribeDSL e retorna o resultado do processamento do texto realizado pelo *Processor*.