UNIVERSIDADE DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Relatório Técnico PPgSI-001/2018

Uma análise comparativa das ferramentas de pré-processamento de dados textuais: NLTK, PreTexT e R

Alexandra Katiuska Ramos Diaz
André Paulino de Lima
Andrei Martins Silva
Fernando Henrique da Silva Costa
José Luiz Maturana Pagnossim
Sarajane Marques Peres

Janeiro - 2018

Série de Relatórios Técnicos

PPgSI-EACH-USP
Rua Arlindo Béttio, 1000 – Ermelino Matarazzo
03828-000 – São Paulo, SP.
TEL: (11) 3091-8197 http://www.each.usp.br/ppgsi

Uma análise comparativa das ferramentas de pré-processamento de dados textuais: NLTK, PreTexT e R

Alexandra Katiuska Ramos Diaz[1]
André Paulino de Lima[1]
Andrei Martins Silva[1]
Fernando Henrique da Silva Costa[1]
José Luiz Maturana Pagnossim[1]
Sarajane Marques Peres[1]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo
São Paulo – SP, Brazil

sarajane@usp.br

Resumo. *O pré-processamento de textos é uma atividade fundamental para representação computacional de dados textuais, permitindo que modelos computacionais possam ser aplicados para aquisição de conhecimento. Neste relatório, as principais operações necessárias para obtenção da representação, segundo o paradigma de modelo do espaço vetorial, são descritas e três ferramentas -* PRETEXT*,* NLTK *e* R *- para implementação dessas operações são apresentadas. Resultados de testes conduzidos sobre o corpus 20 Newsgroups sugerem que as representações de textos geradas nas três ferramentas são equivalentes. O relatório é encerrado com uma análise comparativa das vantagens e desvantagens de cada implementação de forma a prover uma orientação sobre o uso dessas ferramentas para iniciantes.*

1. Introdução

Nas últimas décadas, a disponibilidade de informação tem crescido rapidamente e, em certa medida, esse crescimento tem sido acompanhado tanto pelo aumento da capacidade de plataformas computacionais em agregar e gerenciar recursos de máquina quanto pelo surgimento de métodos de extração e manipulação de informação que viabilizam a descoberta de conhecimento. Entretanto, boa parte da informação disponível hoje está na forma de linguagem natural (fala, gestos, textos) e, até o momento, não existem métodos computacionais que permitam a extração e manipulação eficaz de dados registrados em linguagem natural, particularmente quando a tarefa envolve manipulação de "significado" [Turney et al. 2010].

Em particular, os métodos computacionais existentes requerem a execução prévia de um conjunto de tarefas, chamado de pré-processamento, que visa promover a eficácia das etapas subsequentes de extração e manipulação dos dados. Os autores Manning et al. sugerem que as seguintes tarefas sejam consideradas na etapa de pré-processamento de dados textuais: (i) tokenização, (ii) remoção de stop words, (iii) case-folding, (iv) redução para o radical e lematização.

A etapa de pré-processamento de dados textuais requer o uso de ferramenta computacional que apoie a carga e a representação dos dados em um formato que promova a eficácia da etapa subsequente de extração. Existem ferramentas que suportam as principais tarefas de pré-processamento, como a ferramenta PreTexT [Matsubara et al. 2003] e, também, existem linguagens de programação e bibliotecas especializadas em

manipulação de dados textuais, como a linguagem Python [1], com apoio da biblioteca NLTK [Bird et al. 2009], e a linguagem R [R Development Core Team 2008], com apoio da biblioteca Tm [Feinerer e Hornik 2017].

A seleção da representação dos dados e das características que devem ser deles extraídas depende das necessidades inerentes à pesquisa, como: saber se é suficiente a contagem de termos por documento, ou se há a necessidade do tratamento de mais de uma palavra dentro de um contexto, ou ainda se é necessário processar frases completas. Nesse sentido, algumas características do conjunto de dados são importantes, como: o idioma, a quantidade de dados, a estrutura dos documentos, se os dados são públicos ou se há base de comparação. Para este relatório, foi adotado o *corpus* 20 NEWSGROUPS [Lang 1995], o qual fornece publicamente textos no idioma inglês com postagens realizadas em fóruns de discussão sobre notícias.

A escolha de uma ferramenta adequada para processar dados em formato textual deve considerar alguns critérios, por exemplo: facilidade de uso, operações suportadas na ferramenta, capacidade de customização e extensão de funcionalidades, autonomia do usuário sobre a ferramenta e a confiabilidade dos resultados gerados. Avaliar somente a documentação das ferramentas pode ser insuficiente e pode levar a uma tomada de decisão equivocada. Por outro lado, a avaliação conceitual e prática das ferramentas, traçando um comparativo entre elas, tende a fornecer dados mais adequados para suportar a escolha da ferramenta mais apropriada para cada necessidade. Nesse aspecto, para este relatório, a estratégia de comparação de ferramentas visou, primeiramente, assegurar que as representações vetoriais produzidas para um mesmo conjunto de documentos fossem equivalentes. Além disso, este relatório também apresenta uma análise sobre os parâmetros necessários em cada ferramenta e as operações fornecidas por elas.

Na expectativa de apoiar futuros estudos que venham a utilizar operações de pré-processamento de texto, este relatório apresenta a execução de uma sequência ordenada de pré-processamento sobre o *corpus* 20 NEWSGROUPS para três ferramentas: PRETEXT, a linguagem PYTHON com a biblioteca NLTK e a linguagem R com a biblioteca *Tm*. A fim de alcançar tal expectativa, este relatório está organizado em sete partes, considerando esta introdução:

- Na seção 2 são apresentados os conceitos básicos de pré-processamento de textos e representação vetorial para o dado textual, a apresentação do *corpus* usado neste relatório, a motivação a respeito da escolha das ferramentas utilizadas no relatório e uma comparação das funcionalidades oferecidas por cada ferramenta.
- A seção 3 apresenta uma introdução sobre a ferramenta PRETEXT, sua arquitetura e configurações necessárias para execução das tarefas de pré-processamento.
- A seção 4 apresenta uma introdução sobre a linguagem PYTHON e a biblioteca NLTK, a estrutura do processo implementado, as configurações e procedimentos necessários para execução do pré-processamento e o código-fonte comentado da implementação.
- A seção 5 apresenta uma introdução sobre a linguagem *R* e a biblioteca *Tm*, o ambiente de desenvolvimento integrado (do inglês *Integrated Development Environment* - IDE), os procedimentos para instalação da linguagem e do IDE, as operações de pré-processamento usadas a partir da biblioteca TM, o código-fonte

[1] https://www.python.org/

comentado do programa e as considerações a cerca do uso e resultados gerados pelo *R*.
- A seção 6 apresenta os resultados gerados pela aplicação das ferramentas ao conjunto de dados 20 NEWSGROUPS e um quadro com vantagens e desvantagens na adoção de cada ferramenta.
- A seção 7 faz uma ponderação sobre vantagens e limitações das ferramentas, apresenta a contribuição do relatório e algumas possibilidades de evolução para trabalhos futuros.

2. Conceitos básicos

O objetivo deste capítulo é apresentar os conceitos básicos necessários para o entendimento do conteúdo deste relatório e as ferramentas que serão discutidas.

2.1. Conceitos

Na prática de pesquisa atual em áreas como Recuperação da Informação ou Aprendizado de Máquina, a manipulação algorítmica de conteúdo textual costuma exigir o pré-processamento prévio desse conteúdo para a geração de uma representação numérica [Silva et al. 2016]. Tal representação segue geralmente o modelo de espaço vetorial [Salton et al. 1975]. Em geral, esse processo de pré-processamento consiste em uma sequência de transformações aplicadas aos documentos em *corpus*, transformações essas que variam conforme o idioma no qual o documento está escrito ou de acordo com o objetivo da análise a ser realizada. De acordo com Manning et al. [2008], para o idioma inglês, essas transformações consistem basicamente em:

1. **Tokenização**: interpretando o conteúdo textual como uma sequência de caracteres, a tokenização consiste em agrupar caracteres em ***tokens***, possivelmente ignorando certos caracteres, como pontuação. Um *token* pode ser definido de forma abstrata como uma unidade de texto cujo valor semântico é útil para um dado propósito e, logo, dependente da aplicação. Ainda, a literatura define **tipo** (***type***) como uma classe à qual pertence todos os *tokens* (instâncias) representados por uma mesma sequência de caracteres. Por exemplo, todas as ocorrências do *token "car"* no conjunto de dados pertencem à uma classe representada pelo tipo *"car"*. Complementarmente, define **termo** (***term***) como sendo um **tipo** que foi incorporado ao dicionário associado ao conjunto de dados (em contraposição aos tipos que tenham sido descartados); em outras palavras, é um **tipo** que compõe o vocabulário sobre o qual o conjunto de dados será representado. A literatura emprega as expressões **termo** e **palavra** (***word***) de formas distintas, porque às vezes um **termo** pode representar um **tipo** que não é aceito como sendo uma palavra (exemplo: I-9 ou Hong Kong).

2. **Remoção de *stop words***: de maneira geral, *tokens* que ocorrem com frequência relativamente elevada no conjunto de dados possuem baixo poder discriminativo; a literatura se refere a esses *tokens* como ***stop words*** e a prática recomenda sua exclusão durante o pré-processamento. Usualmente, artigos, conjunções e preposições são fortes candidatos, uma vez que são termos comuns a todos os documentos e aparecem frequentemente.

3. ***Case-folding***: em determinadas aplicações, resultados melhores podem ser obtidos se *tokens* distintos forem interpretados como sendo formas equivalentes, como os pares (*'anti-discriminatory'*, *'antidiscriminatory'*), (*'Automobile'*, *'automobile'*), (*'U.S.A'*, *'USA'*) ou (*'car'*, *'automobile'*). Uma técnica comum para promover essa equivalência é o emprego de classes de equivalência, implementadas comumente na forma de listas relacionando *tokens* equivalentes. Entretanto, o custo para construção desse tipo de lista é alto, por depender de agentes humanos. Uma abordagem computacionalmente simples e que pode trazer algum benefício nesse contexto é a aplicação de ***case-folding*** (conversão do texto para um mesmo caixa de caractere), que unifica **tipos** como *'Automobile'* e *'automobile'* em uma só representação: *'automobile'*.

4. **Redução para o radical e lematização**: em determinadas aplicações, resultados melhores podem ser obtidos se *tokens* que compartilham uma mesma etimologia fossem interpretados como sendo formas equivalentes, como *'democracy'*, *'democratic'* e *'democratization'*. O objetivo das transformações de redução para o radical e lematização é reduzir formas derivadas de uma palavra a uma forma mais básica, como *'whishing'* e *'whishes'* assumindo a forma *'whish'*. A diferença entre os dois métodos, redução para o radical e lematização, pode ser vista como um *trade-off* entre esforço e precisão: redução para o radical emprega um conjunto de heurísticas rudimentares que prescrevem modificações em um *token* (em geral, na forma de regras de remoção de afixos); lematização, por sua vez, exige uma análise linguística rigorosa para identificação do lema da palavra representada pelo *token*. De um modo geral, para o idioma inglês, lematização traz poucos benefícios em relação à redução para o radical [Manning et al. 2008] e, dado o custo mais elevado, seu uso é menos comum. O algoritmo de Porter (*Porter Stemming Algorithm*) é usado para executar a remoção de desinências e sufixos, implementando a redução para o radical [Jones e Willet 1997, Porter 1980a, Rijsbergen et al. 1980]. Este algoritmo possui versões para diferentes línguas e implementações disponíveis em diferentes linguagens de programação[2].

As operações apresentadas podem ser aplicadas em sequência, sendo esta sequência escolhida a partir da dependência existente entre as transformações realizadas por tais operações. Por exemplo, para a aplicação de redução para o radical ou lematização, o texto precisa ter sido tokenizado. O mesmo vale para remoção de *stop words*; entretanto, para manter a lista de *stop words* em uma forma mais reduzida, os **tipos** *'The'* e *'the'* deveriam ser representados por uma entrada apenas, o que exige, portanto, que o texto já tenha sido *case-folded*. Além disso, a lista de *stop words* costuma ser composta por *tokens* que representam palavras (ao invés de lemas ou aproximações de lemas). Por sua vez, *case-folding* aplicado *token* a *token* tende a ser menos eficiente do que se aplicado a unidades de texto maiores, como o documento inteiro, por exemplo. Desta forma, uma sequência possível consiste na aplicação de (1) *case-folding*, (2) tokenização, (3) remoção de *stop words* e (4) redução para o radical. No contexto deste relatório, utiliza-se a expressão ***pipeline*** para referir um processo composto de uma sequência de transformações,

[2]O projeto relacionado ao algoritmo de Porter está disponível em `https://tartarus.org/martin/PorterStemmer/`

sendo que cada uma dessas transformações são referidas como **estágios** do *pipeline*. A figura 1 ilustra o processo de pré-processamento de textos executado neste relatório.

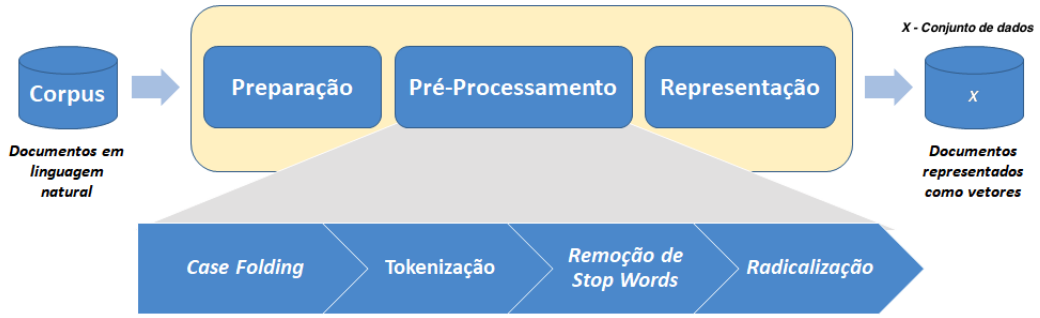


Figura 1. Processo de pré-processamento de documentos em linguagem natural.
Fonte: Diaz, Lima, Silva, Costa, Pagnossim e Peres, 2018.

Após os passos propostos pelo *pipeline*, há ainda a possibilidade de buscar no *corpus* a ocorrência de termos compostos, também denominados de ***n-gramas***. Estes são formados por mais de um elemento (ou seja, por n elementos), no entanto possui apenas um valor semântico [Rezende et al. 2011]. O número de elementos para a formação dos *n-gramas* é determinado pela variável n, a qual deve ser indicada pelo usuário. A seguinte situação hipotética exemplifica o uso de um *n-grama*: na coleção de documentos pré-processada aparecem os elementos *'artificial'* e *'intelligence'*. Desta forma, um *n-grama* possível é *'artificial intelligence'*, caso o $n = 2$ [Moura et al. 2008]. No entanto, há o problema de termos menos significativos e de redundância na utilização de *n-gramas* com $n > 1$ [Moura et al. 2010]. Por exemplo, os termos "informatics" e "computing" unidos em um *bigrama* (*n-grama* com $n = 2$) "informatics computing" pode constituir um termo redundante, sem valor significativo.

Uma vez que as operações de tratamento e seleção de vocabulário foi realizada, é preciso construir uma representação de cada documento com base em seus termos. Uma forma comum de fazê-lo é usar a medida (ou representação) ***tf-idf*** [Baeza-Yates e Ribeiro-Neto 1999]. A medida ***tf*** considera a contagem absoluta de ocorrências do termo t_i no documento d_j. Assim, se o termo *sea* aparece duas vezes no documento 1, $tf_{sea,1} = 2$. Esta medida é local, i.e., considera apenas a distribuição de frequências em um único documento, i.e. computa o número de documentos no qual o termo ocorre. A medida ***idf***, por sua vez, considera a ocorrência de um termo sobre toda a coleção de documentos. Nesta medida, termos comuns a todos os documentos apresentam baixos valores e termos raros na coleção de documentos apresentam valores altos conforme mostra a equação 1.

$$idf(t_i) = log\left(\frac{N}{n_i}\right), \tag{1}$$

em que N é o número total de documentos existentes na coleção (*corpus*) e n_i é o número de documentos em que o termo t_i ocorre. A combinação das medidas ***tf*** e ***idf*** compõem a medida ***tf-idf***, a qual considera os aspectos local e global da ocorrência de um termo e, apesar de sua simplicidade, é competitiva com relação a modelos estatísticos

mais elaborados [Robertson 2004]. A medida *tf-idf*, dada pela equação 2, é obtida pela multiplicação das medidas *tf* e *idf*.

$$tf\text{-}idf(t_i, d_j) = tf(t_i, d_j) * idf(t_i) \tag{2}$$

As operações de remoção de *stop words*, *case-folding*, redução para o radical e lematização, além de normalizar os dados textuais, também contribui para a redução de dimensionalidade dos dados[3]. Dados textuais, geralmente, possuem alta esparsidade pois quantidade de termos presentes no *corpus* é muito maior do que a quantidade de documentos presentes no *corpus*. Além disso, mesmo com essa normalização, muitos termos não são representativos para determinar características interessantes dos textos, e por isso, uma representação como a ***tf-idf*** é necessária para destacar a representatividade do termos mais interessantes. Veja em Silva et al. [2016] uma discussão didática sobre esse assunto. Os estudos de Luhn, baseados na lei de Zipf discutidos por Rijsbergen [1979] explicam essas dificuldades, e segue brevemente comentada aqui:

> A **lei de Zipf** pode ser utilizada para descrever diversos fenômenos linguísticos, sociais e naturais. Dentro da área de pré-processamento de textos, um desses fenômenos refere-se à observação de que a frequência das palavras em um texto segue um padrão. Ao contar a frequência das palavras em uma coleção de documentos e apresentá-las em ordem decrescente segundo suas frequências, a curva resultante assemelhar-se-á à curva da figura 2(I), em que f representa a frequência das palavras em função ao rank (r) [Soares et al. 2008]. Sob a perspectiva da lei de Zipf aplicada à coleções de documentos, Luhn [1958] observou que palavras que apresentam altas frequências em uma coleção de documentos tem baixo poder de discriminação por serem comuns a muitos deles. Por outro lado, observou que palavras que possuem baixa frequência têm baixo poder de caracterização por serem raras. Assim, propôs o estabelecimento de intervalos de significância que ficaram conhecidos como **cortes de Luhn**. Apenas palavras com frequências dentro do intervalo de significância, i.e., dentro dos cortes - inferiores e superiores - de Luhn, devem ser consideradas como representativos do documento ou da coleção de documentos. Assim, o pico imaginário das palavras relevantes ocorre dentro do intervalo de Luhn, conforme ilustrado pela figura 2(II). Os estudos de Zipf e Luhn oferecem as bases teóricas que fundamentam a utilização de cortes de atributos por frequência na análise de textos.

2.2. *Corpus* 20 NEWSGROUPS

O 20 NEWSGROUPS (versão BYDATE) é um *corpus* com aproximadamente 20.000 documentos contendo discussões sobre notícias extraídas de diversos fóruns. Esses documentos estão codificados em *ASCII plain text* e estão organizados de maneira razoavelmente uniforme em 20 grupos de notícias, cobrindo basicamente 6 tópicos distintos

[3]Recursos oferecidos pelos projetos *Wordnet* - para a língua inglesa, e *Unitex-PB* - para a língua portuguesa, podem ajudar na implementação de operações de pré-processamento. O projeto *Wordnet* está disponível em `https://wordnet.princeton.edu`. O projeto *Unitex-PB* está disponível em `http://www.nilc.icmc.usp.br/nilc/projects/unitex-pb/web/index.html`.

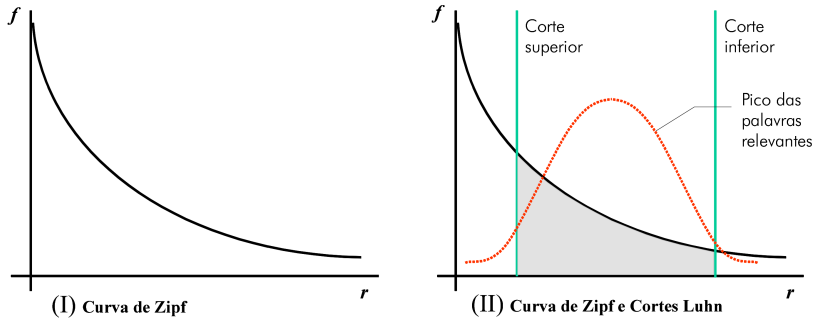


Figura 2. Lei de Zipf e cortes de Luhn
Fonte: [Soares et al. 2008]

(computadores, veículos, ficção científica, política, religião e outros), como ilustra a tabela 1.

Tabela 1. Organização do 20 Newsgroups

Tópicos	Classes	Total de documentos
Computadores	comp.graphics	973
	comp.os.ms-windows.misc	966
	comp.sys.ibm.pc.hardware	982
	comp.sys.mac.hardware	963
	comp.windows.x	985
Veículos	rec.autos	989
	rec.motorcycles	996
	rec.sport.baseball	994
	rec.sport.hockey	999
Ficção científica	sci.crypt	991
	sci.electronics	984
	sci.med	990
	sci.space	987
Política	talk.politics.misc	775
	talk.politics.guns	909
	talk.politics.mideast	940
Religião	soc.religion.christian	996
	alt.atheism	799
	talk.religion.misc	628
Outros	misc.forsale	975

Fonte: Adaptado de Lang [1995]

Os documentos do conjunto 20 NEWSGROUPS apresentam um formato composto pelos metadados na parte superior de cada documento e o resto do documento é texto puro. Na figura 3 é mostrado o documento 53068, pertencente ao grupo *alt.atheism*, com o objetivo de visualizar seu formato.

```
From: decay@cbnewsj.cb.att.com (dean.kaflowitz)
Subject: Re: about the bible quiz answers
Organization: AT&T
Distribution: na
Lines: 18

In article <healta.153.735242337@saturn.wwc.edu>, healta@saturn.wwc.edu (Tammy R Healy) writes:
>
>
> #12) The 2 cheribums are on the Ark of the Covenant.  When God said make no
> graven image, he was refering to idols, which were created to be worshipped.
> The Ark of the Covenant wasn't wrodhipped and only the high priest could
> enter the Holy of Holies where it was kept once a year, on the Day of
> Atonement.

I am not familiar with, or knowledgeable about the original language,
but I believe there is a word for "idol" and that the translator
would have used the word "idol" instead of "graven image" had
the original said "idol."  So I think you're wrong here, but
then again I could be too.  I just suggesting a way to determine
whether the interpretation you offer is correct.


Dean Kaflowitz
```

Figura 3. Exemplo do documento 53068 do *corpus* 20 NEWSGROUPS
Fonte: Diaz, Lima, Silva, Costa, Pagnossim e Peres, 2018.

2.3. Motivação para escolha das ferramentas

Três ferramentas foram escolhidas para análise neste trabalho: a primeira, a ferramenta PRETEXT, é uma ferramenta configurável, de código fechado e não exige programação; a segunda e a terceira, as ferramentas NLTK e R, permitem uma maior flexibilidade na implementação e agregação de funcionalidades, pois é possível interferir no código que está sendo programado. Essas ferramentas foram selecionadas por serem facilmente aplicáveis ao processamento de textos, facilitando o trabalho de pré-processamento. Todas elas servem como ponto de partida para aqueles que desejam focar no trabalho referente à análise de dados e descoberta de conhecimento, servindo como boas abstrações para a fase de pré-processamento. A tabela 2 apresenta uma comparação das principais funcionalidades oferecidas por cada uma das ferramentas escolhidas.

Tabela 2. Comparativo de funcionalidades das ferramentas: PRETEXT, NLTK e R

	PRETEXT	PYTHON + NLTK	R + TM
Tokenização	Alfabética ou alfanumérica. Permite especificar quais símbolos serão utilizados como separadores de *tokens*, quais *tokens* serão removidos e quais serão mantidos. Opcionalmente, remove *tags* HTML.	Alfabética ou alfanumérica. Possibilita especificar os caracteres a serem usados como separadores de *tokens*.	Possibilita especificar os caracteres a serem usados como separadores de *tokens*.
Remoção de stop words	Apresenta uma lista de *stop words* padrão para português e inglês. A lista pode ser customizada pelo usuário. Podem ser criadas múltiplas listas.	Apresenta lista de *stop words* padrão para o inglês. Permite palavras definidas pelo usuário. Permite uma lista de *stop words* vazia.	Permite definir uma lista de *stop words* padrão.
Case-folding	Converte todas as letras para minúscula.	Mantém o caixa original ou converte para minúsculas. Permite criação de novos adaptadores quando necessário implementar outras estratégias de normalização.	Possibilita transformar para maiúscula ou minúscula.
Redução para o radical e lematização	Possibilita uso do algoritmo de *Porter* Porter [1980b] para inglês e adaptações para português e espanhol.	Possibilita o uso do algoritmo *Snowball stemmer*. Permite a alteração do algoritmo por um outro qualquer (oferecido pela ferramento ou externo).	Possibilita uso dos algoritmo *Snowball* e *Porter*.
N-gramas	Suporta n-grama para qualquer valor de n.	Suporta n-gramas	Possibilita o uso de n-gramas por meio da biblioteca NGRAM.
Representação vetorial	Suporta as representações *tf*, *tf-idf*, *tf-linear* e binária. Oferece operações de normalização e suavização dos dados, bem como cortes por frequência absoluta e desvio padrão na coleção de documentos.	Suporta as representações *tf*, *tf* normalizado, *idf*, *tf-idf* e *tf-idf* normalizado. Permite criação de novos adaptadores, quando necessário produzir *scores* de outras métricas.	Possibilita uso das representação *tf*, *tf-idf* (normalizado e não normalizado), binário e notação *SMART*.

Fonte: Diaz, Lima, Silva, Costa, Pagnossim e Peres, 2018.

2.4. Outras ferramentas

Existem outras ferramentas disponíveis para execução da tarefa de pré-processamento de textos. Para referência do leitor, segue uma lista não-exaustiva de outras opções de ferramentas para pré-processamento de textos:

Bow é uma biblioteca de código C útil para escrever análise estatística de texto, modelagem de linguagem e programas de recuperação de informação [McCallum 1998]. A distribuição atual inclui a biblioteca, bem como *front-end* para a classificação de documentos (*rainbow*), recuperação de documentos (*arrow*) e agrupamento de documentos (*crossbow*).

I-Preproc é uma ferramenta de pré-processamento, indexação incremental e busca de documentos desenvolvida em Java utilizando a biblioteca de código aberto Apache Lucene[4] [Pereira e Moura 2015]).

QDAP é um pacote disponível para R que automatiza tarefas envolvidas na análise quantitativa de textos, incluindo contagem de frequências de sílabas, palavras, sentenças e diálogos, entre outros recursos para análise de transcrições textuais [Rinker 2013]).

Quanteda é um pacote disponível para o ambiente R que fornece funcionalidades para análise quantitativa de textos. Entre os recursos oferecidos estão funcionalidades para gerenciamento de conjunto de dados, tokenização, busca textual por palavras-chave, criação e manipulação de matrizes esparsas, cálculo de co-ocorrências e similaridades entre atributos, uso de dicionários, além de funções para exploração visual dos textos [Benoit et al. 2017].

Tidy Text é um pacote para mineração de textos disponível para o ambiente R. Suas funcionalidades incluem tokenização, seleção de atributos, remoção de *stop words*, dicionários para análise de sentimento, cálculo de n-grama, entre outras [Silge e Robinson 2016].

TMG é uma toolbox para o Matlab para geração de matrizes de termo-frequência para coleções de textos [Zeimpekis e Gallopoulos 2005]. Trabalha com representação de matrizes esparsas e inclui uma série de implementações de algoritmos para extração de tópicos e resolução da tarefa de agrupamento.

Apache UIMA é uma implementação de código aberto da especificação UIMA (*Unstructured Information Management Architecture*, ou arquitetura para gestão de informação não estruturada) e que consiste numa plataforma de desenvolvimento de aplicações distribuídas e componentizadas para processamento de informação não estruturada (o que inclui dados textuais). Um exemplo de aplicação dessa natureza é a identificação de entidades (como pessoas, lugares ou organizações) ou relações entre entidades (como X-é-parte-de-Y ou X-fica-em-Y) em coleções de documentos textuais sem formatação [Ferrucci et al. 2009, Nadkarni et al. 2011].

3. PreTexT

A ferramenta PRETEXT, desenvolvida pelo Laboratório de Inteligência Computacional[5] do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, permite a conversão de documentos textuais para um formato estruturado que possa ser

[4]https://lucene.apache.org/

[5]http://labic.icmc.usp.br

utilizado por técnicas de aprendizagem de máquina no cumprimento de diversas tarefas de extração de conhecimento. A estrutura criada segue a abordagem *Bag of Words* para representação dos documentos no modelo de espaço vetorial. A primeira versão da ferramenta, publicada em 2003 [Matsubara et al. 2003] e implementada em Perl[6], permite aos usuários a conversão de documentos textuais em diferentes representações numéricas, redução da dimensionalidade dos dados e criação de taxonomias. A versão avaliada neste relatório foi publicada em 2008 como uma reestruturação da ferramenta original e sua descrição completa pode ser encontrada em Soares et al. [2008].

3.1. Visão Geral

A figura 4 ilustra a arquitetura da ferramenta. O programa `Start.pl`[7] é responsável pelo controle do fluxo da aplicação, mediando a interação entre o usuário e os módulos internos. Recebe como entrada o arquivo `config.xml` que especifica os parâmetros gerais de execução das operações de pré-processamento. Todos os módulos internos (`Maid.pm`, `NGram.pm` e `Report.pm`) recebem, por meio do programa `Start.pl`, os parâmetros de execução especificados no arquivo `config.xml`.

O módulo `Maid.pm` é responsável pela limpeza e *tokenização* dos textos. Além das instruções contidas no arquivo `config.xml`, recebe as listas de *stop words* especificadas no arquivo `stoplist.xml` e os caracteres especiais a serem removidos especificados no arquivo `simbols.xml`. Além dessas especificações, esse módulo recebe como entrada o caminho para o diretório que contém os textos - $\{T_1, T_2, T_3, T_N\}$ - a serem processados.

O principal produto do módulo `Maid.pm` é um diretório contendo $\{E_1, E_2, E_3, E_N\}$ arquivos limpos de acordo com as operações de pré-processamento solicitadas. A depender da configuração especificada pelo usuário, o processo de *tokenização* pode incluir a operação de redução para o radical que é executada pelo submódulo `Stemmer.pm`. Nesse caso, o módulo produz também os arquivos `stemWdTF` e `stemWdST` que trazem informações detalhadas a respeito do processo de redução para o radical. O primeiro arquivo, apresenta os `radicais` ordenados por frequência e, no segundo, os `radicais` são apresentados em ordem alfabética.

Caso o usuário tenha solicitado o cálculo de n-grama no arquivo de configuração, o módulo `NGram.pm` lê os arquivos limpos $\{E_1, E_2, E_3, E_N\}$ e cria `tokens` n-grama de acordo com o parâmetro n especificado pelo usuário. A saída desse módulo contém dois arquivos para cada n-gram solicitado. Por exemplo, se o usuário solicitou a criação de bigrama e trigrama, então a saída será composta por quatro arquivos: `2Gram.all`, `2Gram.txt`, `3Gram.all`, `3Gram.txt`. Os arquivos com extensão `.all` apresentam as informações de frequência dos n-grama. Os arquivos com extensão `.txt` apresentam as mesmas informações, mas agrupadas por documento.

Com base nos arquivos gerados pelo módulo `NGram.pm`, o módulo `Report.pm` cria a representação *Bag of Words* por meio do submódulo `MeasureTF`, que atribui a cada documento o seu vetor de frequências em acordo com o tipo de representação especificada

[6]https://www.perl.org

[7]As extensões `.pm` e `.pl` identificam códigos-fonte escritos em Perl. A primeira refere-se a *Perl Module* e a segunda a *Perl Script*

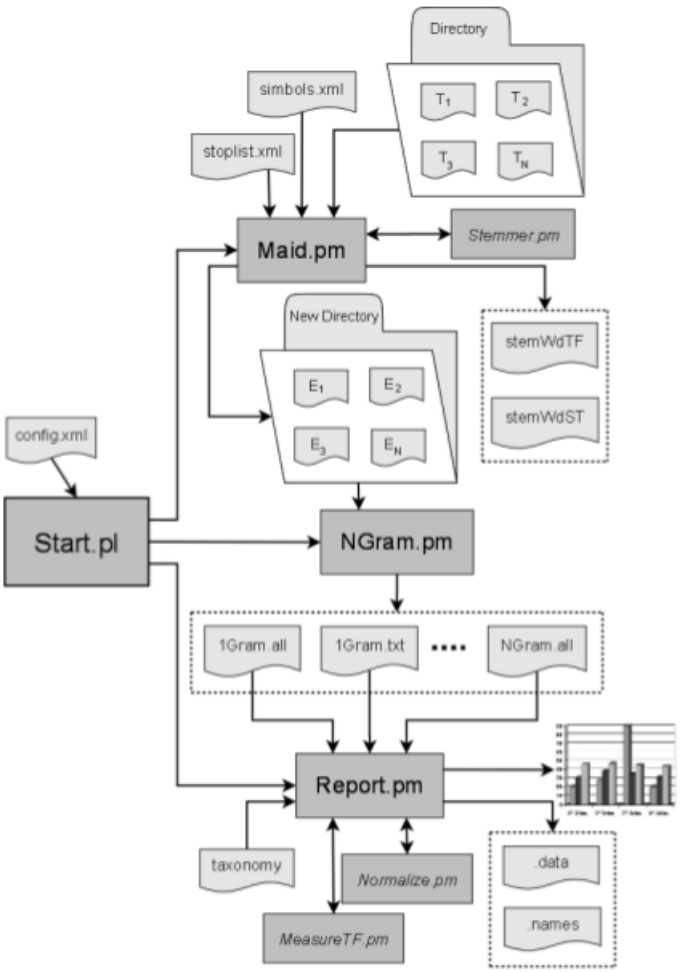


Figura 4. Arquitetura da ferramenta PRETEXT
Fonte: [Soares et al. 2008]

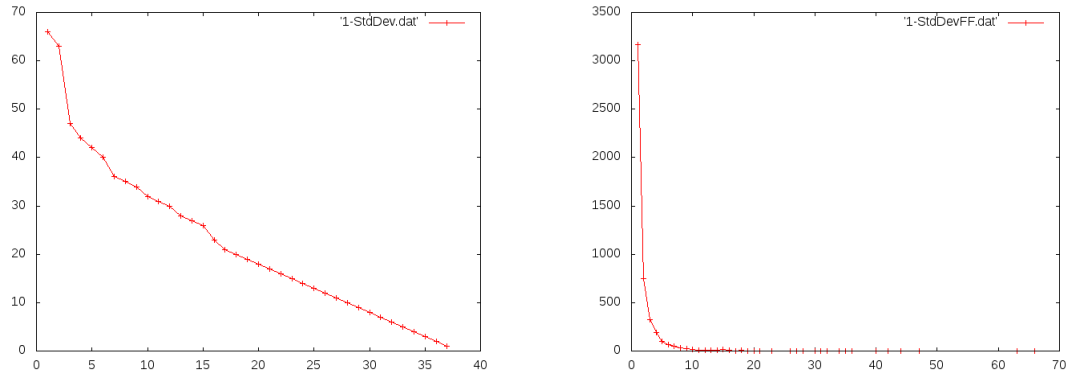
pelo usuário. A critério do usuário, o módulo também pode incorporar uma taxonomia especificada pelo usuário no arquivo taxonomy.

É possível especificar cortes por frequência 'binária' e absoluta. No primeiro caso, para cada termo, é calculada a ocorrência ou não do termo em um documento, independente de quantas vezes um termo ocorre em um determinado documento. Hipoteticamente, se um termo ocorre duas vezes no documento 1, três vezes no documento 2 e quatro vezes no documento 3, sua frequência binária igual a 3. O mesmo termo, teria sua frequência absoluta igual a 9, pois, neste caso, a quantidade de ocorrências é relevante para esta medida. Se a máxima frequência absoluta especificada pelo usuário é 5, o termo seria removido do corpus. Para remover o mesmo termo por frequência binária, bastaria especificar que a frequência binária máxima permitida é igual a 2. Ambos os casos representam versões diferentes dos *Cortes de Luhn*. Após o cálculo das frequências e realização dos cortes, normalizações e suavizações são realizadas por submódulos atrelados ao módulo `Report.pm`.

Por padrão, a *Bag of Words* resultante é gerada de tal forma que cada linha da ma-

triz representa um documento e cada coluna representa um termo do corpus. Opcionalmente, esta *Bag of Words* pode ser gerada em sua forma transposta, em que, cada linha representa um termo e cada coluna representa um documento. De todo modo, a *Bag of Words* resultante é disponibilizada ao usuário no formato `C4.5`, por meio dos arquivos `discover.data` e `discover.names` [8].

Além da *Bag of Words*, são gerados gráficos de frequência dos *tokens* como os apresentados na figura 5. Nesses gráficos, o eixo das ordenadas representa a contagem de ocorrências dos termos e o das abscissas a posição do termo no ranking de frequências. No gráfico 5a não há repetição de termos cuja frequência seja a mesma. Já no gráfico 5b há acúmulo das frequências dos termos que apresentam mesma frequência absoluta. Tais gráficos podem ser úteis para definição dos cortes por frequência. A tabela 3 sumariza as principais funcionalidades da ferramenta divididas por módulo.



(a) Ranking sem repetição **(b) Ranking com repetição**

Figura 5. Gráficos gerados pelo módulo `Report.pm`
Fonte: Diaz, Lima, Silva, Costa, Pagnossim e Peres, 2018.

3.2. Configuração e uso

Todas as informações requeridas para o funcionamento da ferramenta são fornecidas por meio de arquivos de configuração no formato XML[9]. A figura 6 apresenta um exemplo de especificação do arquivo de configuração principal.

Ao todo, três arquivos de configuração devem ser fornecidos. Cada um é responsável por especificar um aspecto diferente da ferramenta conforme descrições a seguir:

config.xml Este é o principal arquivo de configuração. Nele é possível especificar as configurações gerais da ferramenta e também o comportamento desejado para cada módulo. As opções gerais incluem a especificação da língua em que os textos a serem pré-processados estão escritos, diretórios em que se encontram os textos e registro de logs de execução. As opções de configuração do módulo *Maid* incluem a remoção de números, *tags* HTML, caracteres especiais (ver **simbols.xml**), *stop words* e redução para o radical de *tokens*. A especificação do n-grama também é feita neste arquivo. A ferramenta não impõe limites para o valor assumido por

[8]Para detalhes sobre o formato **C4.5** ver `http://www.cs.washington.edu/dm/vfml/appendixes/c45.htm`

[9]https://www.w3.org/TR/REC-xml

Tabela 3. Funcionalidades da ferramenta PRETEXT divididas por módulo

Maid.pm	**NGram.pm**	**Report.pm**
Limpeza de documentos	Criação de representação n-grama para qualquer valor de n	Criação de gráficos
Remoção de *tags* HTML		Cortes por frequência binária
Tratamento de caracteres especiais		Cortes por frequência absoluta
Remoção de *stop words*		Utilização de taxonomia
Aplicação de redução para o radical para português, inglês e espanhol		Normalizações por linha e coluna: quadrática e linear
Criação de arquivos limpos		Criação de representações numéricas: tf-idf, tf, tf-linear e *boolean*
		Bag of Words Transposta

Fonte: Adaptado de Soares et al. [2008]

n. Por fim, é possível especificar opções do módulo **Report.pm** como uso de taxonomia, criação de matriz *Bag of Words* transposta, definição dos limites superiores e inferiores dos cortes por frequência, além de selecionar tipos diferentes de representação, configurar opções de suavização e normalização dos dados.

simbols.xml Neste arquivo são especificados os caracteres não alfanuméricos que deverão ser removidos dos textos durante a tarefa de pré-processamento.

stoplist.xml Este arquivo possibilita a indicação de palavras que devem ser removidas por possuir baixo poder de discriminação entre documentos. A ferramenta permite ainda a especificação de múltiplas *stoplists*, permitindo a criação, por exemplo, de uma lista de *stop words* para cada idioma ou uma lista de *stop words* para cada categoria de termos.

3.3. Instalação

Para instalar o PRETEXT, os seguintes procedimentos devem ser seguidos de acordo com o sistema operacional em uso[10]:

Windows

1. Seguir as instruções em `https://www.perl.org/get.html` para instalação da linguagem de programação Perl.
2. Obter a ferramenta PRETEXT em `http://sites.labic.icmc.usp.br/pretext2/`.
3. Descompactar a ferramenta em alguma pasta de preferência para uso.

Linux

1. Seguir as instruções em `https://www.perl.org/get.html` para instalação da linguagem de programação Perl.

[10]Não foi realizado teste com a ferramenta em macOS.

Figura 6. Exemplo de configuração do PRETEXT
Fonte: Adaptado de Soares et al. [2008]

2. Garantir que os pacotes **build-essential** e **libc6** estão instalados.
3. Obter a ferramenta PRETEXT e os pacotes IO-Dirent-0.02.tar.gz e XML-Parser-2.34.tar.gz em `http://sites.labic.icmc.usp.br/pretext2/`.
4. Descompactar o pacote IO-Dirent-0.02.tar.gz e utilizar os comandos de instalação:
   - `Perl Makefile.PL`
   - `sudo make`
   - `sudo make install`
5. Repetir o passo 4 para o pacote XML-Parser-2.34.tar.gz.
6. Descompactar a ferramenta PRETEXT em alguma pasta de preferência para uso.

4. NLTK - Natural Language ToolKit

Ao contrário do software PRETEXT, que implementa um processo de pré-processamento pronto para uso e no qual tanto a natureza quanto a ordem das transformações já estão estabelecidas, a NLTK (*Natural Language ToolKit*) [Loper e Bird 2002] é uma biblioteca oferecida para ambientes de programação em PYTHON que dispõe de funções que podem ser usadas como elementos na composição do processo de pré-processamento. Em outras palavras, o usuário pode selecionar as funções de seu interesse, como tokenizadores ou operadores de redução para o radical, e criar o código em PYTHON para sequenciar as operações que devem compor cada estágio do processo de pré-processamento. Isto exige do usuário um esforço maior para adoção da NLTK em relação à adoção do PRETEXT. Entretanto, esse esforço adicional proporciona a compensação de oferecer mais liberdade ao usuário para estruturar o pré-processamento da forma que achar mais adequado.

Um ambiente PYTHON com a biblioteca NLTK pode ser criado usando o software de gestão de configuração de ambientes de desenvolvimento CONDA, por meio da seguinte linha de comando[11]:

```
conda create --name ambiente python=2.7 nltk
activate ambiente
```

O código apresentado nas listagens apresentadas nesta seção foram testados em um ambiente com essa configuração.

4.1. Estágios do pré-processamento

Como descrito na seção 2.2, os documentos do *corpus* 20 NEWSGROUPS são codificados como *ASCII plain text*, sendo que uma representação vetorial deve ser produzida para cada documento ao final do pré-processamento. Como ilustrado na figura 1, o processo é composto pelo sequenciamento das seguintes operações:

1. Preparação: os documentos são carregados na memória.
2. Pré-processamento: o conteúdo de cada documento carregado é padronizado por meio da aplicação das transformações básicas descritas na seção 2.1.
3. Representação: criação de uma representação vetorial para o conteúdo de cada documento.

O código apresentado na listagem 1 ilustra uma possível implementação do processo.

4.2. Estágio de preparação

Neste estágio, como já mencionado, os documentos do conjunto de dados são carregados em memória, assumindo que o conteúdo do conjunto de dados é menor do que a quantidade de memória disponível. Como ilustrado na listagem 2, os documentos são lidos e carregados um dicionário indexado pelo nome do arquivo do documento. O conteúdo de cada documento é representado como uma lista na qual cada elemento é uma linha no texto.

Vale ainda ressaltar que os documentos do *corpus* 20 NEWSGROUPS estão organizados em múltiplos níveis de subdiretórios. Entretanto, essa divisão não é relevante para o propósito deste relatório e, por esta razão, todos os documentos foram colocados em um único subdiretório antes da execução da etapa de preparação.

Listagem 1. Processo de pré-processamento completo

```
import os
import re
import cPickle as pickle
from nltk import SnowballStemmer
from nltk.corpus import stopwords
from math import log10

def main():
  # estagio de preparacao
```

[11] O software Miniconda com Python 2.7 e seu procedimento de instalação para Windows, Linux e macOs está disponível em `https://conda.io/miniconda.html`.

```
sourcepath = os.path.join('.', 'corpus')
corpus1 = loadCorpus(sourcepath)

# estagio de pre-processamemento
param_foldCase = True
param_language = 'english'
param_listOfStopWords = stopwords.words(param_language)
param_stemmer = SnowballStemmer(param_language)
params = (param_foldCase, param_listOfStopWords, param_stemmer)
corpus2 = processCorpus(corpus1, params)

# estagio de representacao
targetpath = os.path.join('.')
corpus3 = representCorpus(corpus2)
serialise(corpus3, os.path.join(targetpath, 'corpus'))
```

Listagem 2. Funções usadas no estágio de preparação

```
def loadCorpus(sourcepath):
  corpus = {}
  for filename in os.listdir(sourcepath):
    fh = open(os.path.join(sourcepath, filename), 'r')
    # dicionario indexado pelo nome do arquivo que associa
    # cada documento a lista de linhas do seu texto
    corpus[filename] = fh.readlines()
    fh.close()
  return corpus
```

4.3. Estágio de pré-processamento

Este estágio aplica as operações de pré-processamento descritas na seção 2.1 aos documentos carregados na memória. A ordem de aplicação das operações pode variar conforme o contexto de aplicação em que o pré-processamento de texto ocorre, mas as precedências entre as operações precisam ser consideradas. Uma implementação possível para este estágio é ilustrada na listagem 3. Nesta implementação, a operação de uniformização do caixa das letras (*case-folding*) é aplicada, seguida da operação de tokenização, que reproduz o algoritmo de tokenização implementado pela ferramenta PRETEXT. A lista de *tokens* produzida é filtrada para remover *tokens* que aparecem na lista de *stop words* parametrizada e a operação de redução para o radical é aplicada aos *tokens* remanescentes. Vale ressaltar que as operações de remoção de *stop words* e redução para o radical fazem uso de funções disponibilizadas pela biblioteca NLTK:

- A lista de *stop words* empregada na implementação é uma lista de *stop words* comuns do idioma inglês, disponibilizada dentro do módulo `nltk.corpus`. A lista poderia ser customizada com qualquer outro conjunto de *tokens* que seja mais adequado ao contexto da aplicação.
- A operação de redução para o radical implementada usa o algoritmo *Snowball* para o idioma inglês, mas diversos outros algoritmos de redução para o radical estão disponíveis na biblioteca NLTK, inclusive para outros idiomas, como o português e o espanhol [12].

[12]Documentação dos algoritmos de redução para o radical disponíveis na biblioteca NLTK pode ser encontrada neste link: www.nltk.org/api/nltk.stem.html

A biblioteca NLTK também conta com um módulo que implementa tokenizadores. Entretanto, para manter a simplicidade da implementação e atender a necessidade de comparar os resultados obtidos com ferramenta PRETEXT, um tokenizador baseado em expressão regular foi implementado, ao invés de instanciar um tokenizador da biblioteca.

4.4. Estágio de representação

Este estágio cria uma representação vetorial para cada documento no conjunto de dados usando a representação ***tf-idf***. Nesta representação, os documentos do *corpus* são transformado em uma matriz termo-documento, na qual cada elemento da matriz corresponde à medida *tf-idf* do termo t no documento d. Essa matriz representa os dados que serão submetidos aos algoritmos de análise, mineração e descoberta de conhecimento. A listagem 4 ilustra uma possível implementação para o estágio de representação usando representação *tf-idf*. Vale destacar alguns pontos:

- Computar a medida *tf-idf* para um termo t que ocorre no documento d do conjunto de dados exige a criação de um dicionário direto e um dicionário reverso [Manning et al. 2008]. O dicionário direto armazena o número de ocorrências de um termo em um documento, ao passo que o dicionário reverso associa a cada termo o número de documentos no qual ele ocorre.
- Uma decisão de desenho com impacto potencialmente positivo no desempenho tanto da criação quanto do uso posterior da matriz termo-documento é a escolha de sua representação computacional. A representação termo-documento costuma produzir matrizes de alta dimensionalidade e esparsas. Isso acontece quando as seguintes condições estão presentes no contexto da aplicação:
  - O *corpus* contém um número elevado de documentos.
  - O vocabulário sobre o qual os documentos foram escritos contém um número elevado de termos distintos.
  - Um documento raramente contém todos os termos do vocabulário e, por isso, a matriz tende a ser esparsa.

Considerando os aspectos mencionados da matriz termo-documento, a representação adotada usa um dicionário indexado por documento, no qual cada elemento contém um dicionário relacionando um termo daquele documento com sua respectiva medida *tf-idf*, evitando a representação de elementos na matriz para os quais o valor da medida é zero. Entretanto, essa representação é incompatível com o uso de funções de bibliotecas de álgebra linear, impedindo o uso do produto vetorial implementado na biblioteca `numpy`, por exemplo. Isso significa que operações algébricas precisam ser construídas para essa representação. Um exemplo de implementação do produto vetorial aplicado ao cálculo da medida de distância do cosseno entre documentos usando a representação adotada para a matriz termo-documento é descrita na listagem 5.

Listagem 3. Funções usadas no pré-processamento

```
def processCorpus(corpus, params):
   (param_foldCase, param_listOfStopWords, param_stemmer) = params
  newCorpus = {}
  for document in corpus:
    content = []
    for sentence in corpus[document]:
      sentence = sentence.rstrip('\n')
      sentence = foldCase(sentence, param_foldCase)
      listOfTokens = tokenize(sentence)
      listOfTokens = removeStopWords(listOfTokens, param_listOfStopWords)
      listOfTokens = applyStemming(listOfTokens, param_stemmer)
      content.append(listOfTokens)
    newCorpus[document] = content
  return newCorpus

def foldCase(sentence, parameter):
  if(parameter): sentence = sentence.lower()
  return sentence

def tokenize(sentence):
  sentence = sentence.replace("_","␣")
  regExpr = '\W+'
  return filter(None, re.split(regExpr, sentence))

def removeStopWords(listOfTokens, listOfStopWords):
  return [token for token in listOfTokens if token not in listOfStopWords]

def applyStemming(listOfTokens, stemmer):
  return [stemmer.stem(token) for token in listOfTokens]
```

Listagem 4. Funções usadas no estágio de representação

```
def representCorpus(corpus):
  # cria um dicionario que associa um documento a sua a lista de tokens
  newCorpus = {}
  for document in corpus:
    newCorpus[document] = [token for sentence in corpus[document]
                                 for token in sentence]

  # cria uma lista com todos os tokens distintos que ocorrem em cada documento.
  allTokens = []
  for document in newCorpus:
    allTokens = allTokens + list(set(newCorpus[document]))

  # cria o dicionario reverso
  idfDict = {}
  for token in allTokens:
    try:
      idfDict[token] += 1
    except KeyError:
      idfDict[token] = 1

  # atualiza o dicionario reverso, associando cada token com seu idf score
  nDocuments = len(corpus)
```

```
  for token in idfDict:
    idfDict[token] = log10(nDocuments/float(idfDict[token]))

  # computa a matriz termo-documento (newCorpus)
  for document in newCorpus:

    # computa um dicionario com os tf scores de cada termo que ocorre no documento
    dictOfTfScoredTokens = tf(newCorpus[document])

    # computa um dicionario com o tf-idf score de cada par termo-documento
    newCorpus[document] = ({token: dictOfTfScoredTokens[token] * idfDict[token]
                            for token in dictOfTfScoredTokens})

  return newCorpus

def tf(listOfTokens):
  # cria um dicionario associando cada token com o numero de vezes
  # em que ele ocorre no documento (cujo conteudo eh listOfTokens)
  types = {}
  for token in listOfTokens:
    if(token in types.keys()): types[token] += 1
    else:                      types[token] = 1

  return types

def serialise(obj, name):
    f = open(name + '.pkl', 'wb')
    p = pickle.Pickler(f)
    p.fast = True
    p.dump(obj)
    f.close()
    p.clear_memo()
```

Listagem 5. Comparação de documentos usando distância do cosseno

```
def dotprod(a, b):
  return sum([a[i] * b[j] for i in a.keys() for j in b.keys() if i == j])

def norm(a):
  return (dotprod(a,a) ** 0.5)

def cossim(a, b):
  return (dotprod(a,b) / (norm(a) * norm(b)))
```

5. O ambiente R

R é uma linguagem e um ambiente voltados para análise de dados, com a utilização de recursos estatísticos, e para visualização dos dados, por meio de recursos gráficos. Trata-se de um *software* de código aberto desenvolvido sob os termos do projeto GNU (*Free Software Foundation's General Public License*), além disso, a linguagem oferece também diversos recursos para programação matemática e estatística, tais como: modelagem linear e não-linear, testes estatísticos clássicos, análise de séries temporais, classificação e agrupamento.

Uma das vantagens do R é a disponibilidade de bibliotecas que implementam funções e algoritmos. Caso as bibliotecas disponíveis não atendam aos requisitos da aplicação a ser desenvolvida, o R possibilita chamadas a programas desenvolvidos em C, C++ e Fortran. Se por um lado, o uso de bibliotecas aumenta a produtividade no desenvolvimento dos programas e abstrai a complexidade na implementação de algoritmos, por outro, criar uma biblioteca em R ou modificar o código-fonte de uma existente não é uma tarefa simples.

No escopo deste relatório, foi adotada uma biblioteca para operação de mineração de textos conhecida como *package* TM [Feinerer e Hornik 2017], a qual deve ser referenciada no bloco inicial da programação R, para fazer uso das funções disponíveis, por meio da seguinte linha de comando:

```
install.packages('tm')
require('tm')
```

No restante deste capítulo serão apresentados o ambiente de desenvolvimento R, as operações de pré-processamento de texto com a respectiva linha de comando para implementação em R, o algoritmo para pré-processamento de texto denominado *Pipeline R* e as considerações finais a respeito do R.

5.1. Ambiente de desenvolvimento

O R é compilado e executado em uma ampla variedade de plataformas, as quais incluem UNIX, Windows e MacOS nas suas diferentes versões. O processo de instalação do ambiente de desenvolvimento R é simples e rápido.

No contexto deste relatório, duas ferramentas são utilizadas, a primeira diz respeito ao compilador R, o qual disponibiliza todo conjunto de bibliotecas para manipulação de dados, exibição de gráficos, operadores para cálculo de matrizes, funções estatísticas e matemáticas de uma forma geral. Além disso, o compilador R também oferece os fundamentos básicos de programação, como desvios condicionais, laços de repetição e funções recursivas.

A segunda ferramenta é um ambiente de desenvolvimento integrado (IDE - *Integrated Development Environment*), denominado RStudio. Tal ambiente encapsula o compilador R e oferece um editor para programação com funcionalidades úteis ao programador, como por exemplo, o recurso de auto-completar parâmetros das funções, visualização da declaração da função original, integração com a documentação da linguagem e das bibliotecas, além de opções de depuração do código.

Para a elaboração de um ambiente de desenvolvimento em R, é necessário efetuar o *download* da linguagem R, do IDE RStudio, executar a instalação das respectivas ferramentas, abrir o RStudio e começar a programar. A seguir são listadas as referências para obter as ferramentas:

- **Linguagem R**: a linguagem R pode ser obtido no site oficial do R-Project: `https://cran.r-project.org/mirrors.html`.
- **RStudio**: o IDE RStudio pode ser obtido pelo site `https://www.rstudio.com/products/rstudio/download/`.

5.2. Operações de pré-processamento de texto

O R disponibiliza diversas operações de mineração de dados. Em se tratando de dados textuais, a biblioteca TM implementa as principais tarefas de mineração de texto, abstraindo a complexidade de desenvolvimento de algoritmos para tais tarefas. A documentação dessa biblioteca está disponível no site do R-Project [13].

A biblioteca TM disponibiliza funções capazes de resolver as principais operações de pré-processamento de texto, as quais são:

- **Carga dos documentos para a memória** O carregamento dos documentos para a memória é realizada por meio da função *Corpus*, a qual percorre os documentos a partir de um caminho atribuído como parâmetro e armazenará em uma variável. A função *Corpus* também pode receber outros parâmetros que definem o seu modo de operação, sendo eles:
    1. *enconding = "estilo"*: define o estilo da codificação. Neste relatório foi utilizado o estilo "UTF-8".
    2. *recursive = valor*: indicador lógico que define se a leitura da pasta será recursiva ou não. Neste relatório foi usado valor *TRUE*.
    3. *ignore.case = valor*: indicador aplicado para ignorar letras maiúsculas ou minúsculas. Neste relatório foi usado valor *FALSE*.
    4. *mode = "modo"*: define o modo do documento. Neste relatório foi usado modo = *"texto"*.
    5. *readerControl = list(reader = readPlain, language = "en")*: define a lista de parâmetros de controle para leitura do documento. O parâmetro *readPlain* indica leitura em documento texto e *language* indica o idioma do texto ("en" = english).

```
# Local (pasta) com os documentos para processamento
foldedir="C:/temp/mini_newsgroups/"
cat (foldedir)

# Carregamento dos arquivos texto
CorpusVar <- Corpus(DirSource(folderdir, encoding = "UTF-8",
recursive = FALSE, ignore.case = FALSE, mode = "text"),
readerControl = list(reader = readPlain, language = "en"))
```

- **Preparação** Com o intuito de otimizar as tarefas de pré-processamento, rotinas de preparação do texto são executadas, como a conversão do texto para letras minúsculas, remoção de pontuações, remoção de espaços desnecessários e remoção de números, conforme apresentadas a seguir:
    1. A conversão do texto para letras minúsculas é realizada por meio da função: *tm_map(CorpusVar, content_transformer(tolower))*.
    2. A remoção de pontuação é realizada por meio da função: *tm_map (CorpusVar, removePunctuation)*. Essa rotina causa um efeito de redução na quantidade de caracteres desnecessários. Desta forma, quando um texto estiver corretamente escrito com um espaço após o ponto final, a separação será bem executada. Entretanto, caso o espaço não esteja apresentado no texto,

[13] https://cran.r-project.org/web/packages/tm/index.html

a palavra antes do ponto ficará concatenada com a palavra após o ponto. O mesmo efeito ocorre em textos em que o ponto faz parte de um contexto, por exemplo, em um *e-mail* ou endereço eletrônico.

3. A remoção de espaços desnecessários é realizada por meio da função: *tm_map(CorpusVar, stripWhitespace)*.
4. A remoção de números é realizada por meio da função: *tm_map(CorpusVar, removeNumbers)*. Inicialmente, foi avaliada a remoção de números no pré-processamento do algoritmo em R, todavia com a finalidade de padronizar com as demais ferramentas comparadas no relatório, esse recurso não foi aplicado neste relatório.

```
# Converter o texto para minusculas
CorpusVar <- tm_map (CorpusVar, content_transformer(tolower))

# Remover a pontuacao
CorpusVar <- tm_map (CorpusVar, removePunctuation)

# Retirar espacos em branco desnecessarios
CorpusVar <- tm_map (CorpusVar, stripWhitespace)
```

- **Tokenização** A tokenização padrão do R para texto é o espaço em branco, portanto, foi utilizado esse padrão. Outro aspecto que interfere na tokenização são caracteres considerados como pontuação (ponto, vírgula, parênteses, cerquilha, entre outros). Os ensaios apresentaram que sem a utilização de um parâmetro em R para remover a pontuação (*removePunctuation*), o resultado gerava palavras concatenadas com caracteres considerados como pontuação, tornando a representação inadequada, como pode ser observado na figura 7.

Sem pontuação	
Arquivo	Termo
53068	answer
53068	ark
53068	articl
53068	aton
53068	believ
53068	bibl
53068	cheribum
53068	correct
53068	could
53068	coven
53068	creat
53068	day
53068	dean
53068	deankaflowitz

Com pontuação	
Arquivo	Termo
53068	#12)
53068	(dean.kaflowitz)
53068	(tammi
53068	<healta.153.735242337@saturn.wwc.edu>,
53068	answer
53068	ark
53068	articl
53068	atonement.
53068	believ
53068	bibl
53068	cheribum
53068	correct.
53068	could
53068	coven
53068	covenant.
53068	creat
53068	day
53068	dean
53068	decay@cbnewsj.cb.att.com

Figura 7. Comparação dos ensaios em R com e sem pontuação.
Fonte: Diaz, Lima, Silva, Costa, Pagnossim e Peres, 2018.

- **Remoção de *stop words***
  Definiu-se uma lista de palavras para serem removidas do texto (lista de *stop words*), sendo essa utilizada em todas as ferramentas. A função que realiza a remoção dessas palavras é: *tm_map(CorpusVar, removeWords, myStopWords)*. O parâmetro *removeWords* é o indicador para remover as palavras da lista, já o parâmetro *myStopWords* é a variável referente a lista de palavras, as quais são separadas por vírgula.

```
# Lista de stopwords
 myStopWords <- c('i', ... , 'wouldn')

# Remover stopwords
CorpusVar <- tm_map(CorpusVar, removeWords, myStopWords)
```

- **Redução para o radical**
  A tarefa de redução para o radical padrão do R é obtida pelo *Stemmer R*, portanto, o *pipeline* adotou esse padrão, conforme apresentado na função: *tm_map(CorpusVar, stemDocument)*.

```
# Aplicar o Stemming. Reduz as palavaras ao seus radicais,
# usando algoritmo de Potter
CorpusVar <- tm_map(CorpusVar, stemDocument)
```

- **Criação das representações básicas (binária, tf, tf-idf)** A representação é atribuída como um parâmetro denominado *weighting*, o qual pode receber valor *weightTf* ou *weightTfidf*. Esse valor está contido na função que transforma o texto em uma matriz de Termos (linhas) e Documentos (colunas). A função que realiza este procedimento é: *TermDocumentMatrix (CorpusVar, control = list(weighting=weightTf, minWordLength = 2, minDocFreq = 1))*. O parâmetro *minWordLenght* define que a matriz deve considerar somente palavras com o tamanho mínimo especificado, nesse caso, palavras com no mínimo dois caracteres (assim, o *token 'a'* não aparece na matriz). Já o parâmetro *minDocFreq* considera o número mínimo de vezes que um termo tem que aparecer para ser incluído na matriz. Desta forma, esses parâmetros também podem servir para reduzir a dimensionalidade. No caso dos experimentos deste relatório, foram consideradas todas as palavras.

```
# Transformar o conjunto de dados em uma matriz de Termos x Documentos.
# (Termos nas linhas e documentos nas colunas)
tdm <- TermDocumentMatrix (CorpusVar, control = list(weighting = function(x)
       weightTfIdf(x, normalize = TRUE), minWordLength = 1, minDocFreq = 1))
```

- **Saída dos dados** Em relação ao formato da saída, o resultado da aplicação da função TM sobre uma coleção de documentos é gerado no formato de uma matriz. Nela, os termos estão organizados nas linhas e os documentos estão organizados nas colunas. Primeiramente, essa transformação em matriz foi realizada por meio das funções:
  - *as.matrix*.

– *dim* (apresenta dimensões da matriz).

Logo após os dados estarem contidos na matriz gerada, essa saída pode ser enviada para o formato de planilha por meio da utilização da função: WRITE.CSV, em que pode ser definido o caminho que será salvo o arquivo .CSV.

```
# Saida dos dados em arquivo separado por virgula
m <- as.matrix(tdm)
dim(m)
write.csv(m, file = "c:/Temp/dtm3.csv")
```

5.3. Algoritmo para pré-processamento de texto em R

Na listagem 6 é apresentado o código que implementa o processo de execução de pré-processamento de textos adequados aos objetivos deste relatório.

Listagem 6. Pipeline R

```
##Instalar a package tm. Descomente a linha abaixo na primeira vez que for rodar o
programa.
#install.packages('tm')
##Incluir o pacote para uso
require('tm')

#Local (pasta) com os documentos para processamento
folderdir="C:/temp/mini_newsgroups/"
cat (folderdir)

#Carregamento dos arquivos texto
CorpusVar <- Corpus(DirSource(folderdir, encoding = "UTF-8",
        recursive =FALSE, ignore.case = FALSE, mode = "text"),
        readerControl=list(reader=readPlain,language="en"))

#Convertendo o texto para minusculas
CorpusVar <- tm_map (CorpusVar, content_transformer(tolower))

#Removendo  pontuacao (1)
CorpusVar <- tm_map (CorpusVar, removePunctuation)

#Retirando espacos em branco desnecessarios
CorpusVar <- tm_map (CorpusVar, stripWhitespace)

#Lista de Stop Words (Lista suprimida no quadro para melhor visualizacao, a lista
completa pode ser acessada nos anexos)
myStopWords <- c('i', ... , 'wouldn')

#Removendo stopwords
CorpusVar <- tm_map(CorpusVar, removeWords, myStopWords)

#Aplicando o Stemming / reduz palavaras ao radicais, usando algoritmo de Potter
CorpusVar <- tm_map(CorpusVar, stemDocument)

#Transformando o conjunto de dados em uma matriz de Termos x Documentos. (Termos nas
linhas e documentos nas colunas)
tdm <- TermDocumentMatrix (CorpusVar, control=list(weighting = function(x)
```

```
        weightTfIdf(x, normalize = TRUE), minWordLength=1, minDocFreq=1))

#Inspecionando a matriz para visualizacao e opcao de copiar os dados do console
inspect(tdm)

#Saida dos dados em arquivo separado por virgula
m <- as.matrix(tdm)
dim(m)
write.csv(m, file="c:/Temp/dtm3.csv")
```

5.4. Considerações

O R pode ser considerado flexível quanto à capacidade de customização, já que possibilita implementações de funções de mineração de dados usando funções nativas da própria linguagem. Vale observar que essa forma de programação requer conhecimentos da linguagem e dos conceitos que se deseja implementar. Uma alternativa para minimizar o esforço de desenvolvimento em R é a utilização de pacotes, que representa um ganho significativo de produtividade pois abstrai a complexidade na implementação de algoritmos. Contudo, essa facilidade torna o desenvolvimento mais rígido, uma vez que o desenvolvedor torna-se dependente daquilo que já está disponível no pacote. Outro efeito colateral é a necessidade de compreender os detalhes da implementação contidas no pacote por meio de documentações e manuais, os quais nem sempre demonstram informações em níveis de detalhe adequado ou com clareza. Diante desse cenário, a opção pela utilização do R tem como diferencial e vantagem, o reuso dos pacotes e funções já disponíveis, já que a aprendizagem dos comandos nativos da linguagem, demandariam um tempo equivalente ao estudo de qualquer outra linguagem de programação, abrindo a possibilidade do programador optar por uma linguagem em que ele possua um conhecimento prévio.

6. Resultados

A estratégia de comparação entre as ferramentas se fundamenta em dois pontos:

1. Assegurar que as ferramentas produzem representações similares para uma mesma coleção de documentos.
2. Identificar o conjunto de parâmetros e transformações oferecidos por cada ferramenta.

O primeiro ponto busca assegurar que a seleção de qualquer uma das ferramentas produzirá resultados semelhantes e, portanto, o usuário pode guiar sua seleção por outros critérios. O segundo ponto busca identificar quais capacidades são oferecidas por cada ferramenta, de modo a permitir a criação de um quadro comparativo que ilustre vantagens e desvantagens, permitindo ao usuário que selecione a ferramenta que reúne o conjunto de capacidades mais próximas da sua necessidade. No restante desta seção, são abordados os experimentos que suportam o primeiro ponto e a análise do levantamento de capacidades entre as ferramentas que suportam o segundo ponto.

6.1. Equivalência da representação produzida

Para verificar que as ferramentas produzem representações similares, foram realizados dois experimentos:

- **Experimento 1** - Verificação da similaridade do estágio de pré-processamento: uma vez que o estágio de extração de características depende fundamentalmente das transformações realizadas no pré-processamento, este primeiro experimento busca determinar se, para um mesmo documento, as três ferramentas são capazes de produzir a mesma lista de *tokens*, dado que estão configuradas com parâmetros similares. A tabela 4 apresenta detalhes deste experimento, como os dados usados como entrada, a configuração das ferramentas e análise dos resultados observados.

  Resultado: A ferramenta PRETEXT e o *pipeline* PYTHON/NLTK produziram a mesma lista de *tokens*, salvo diferenças nas formas produzidas pela redução para o radical. Já o *pipeline* em R produz uma lista menor. Uma das explicações levantadas é devido ao *pipeline* não ter gerados *tokens* de caracteres numéricos. Uma outra explicação é a não geração de *tokens* com menos de três caracteres pelo *pipeline* em R.

- **Experimento 2** - Verificação da similaridade do estágio de extração de características: assumindo similaridade entre as listas de *tokens* produzidas por cada ferramenta, este experimento busca verificar se, para um mesmo documento, as três ferramentas são capazes de produzir uma mesma representação vetorial, dado que estão configuradas para computar os mesmos *scores*. A tabela 5 apresenta detalhes do experimento.

  Resultado: o experimento confirma a similaridade entre as representações produzidas pelo PRETEXT e a implementação em PYTHON/NLTK. Entretanto, verificou-se que os *scores* calculados pelo *pipeline* em R são diferentes, mesmo usando os devidos parâmetros de peso (weighting) oferecidos pela função TERMDOCUMENTMATRIX para as representações.

6.2. Comparação das funcionalidades oferecidas

O segundo ponto que fundamenta a estratégia de comparação entre as ferramentas se constitui na identificação e classificação de funcionalidades oferecidas por cada ferramenta. A tabela 2 apresenta diferentes funcionalidades disponibilizadas por cada ferramenta, segmentadas segundo os estágios sugeridos para um processo de pré-processamento para conjunto de dados no idioma inglês. Além dessa comparação de funcionalidades, estão aqui listados alguns aspectos práticos relevantes para a escolha da ferramenta. Esses aspectos levam em consideração a maturidade da ferramenta, uso eficiente de plataformas computacionais e integração com outras ferramentas, por exemplo:

- **PRETEXT**: Esta ferramenta oferece um conjunto maior de funcionalidades já preparadas para uso, sem necessidade de programação. Possui boa documentação e foi testadas por alguns grupos de pesquisa especializados em análise de dados textuais[14]. Entretanto, de acordo com o que foi explorado pelos autores deste relatório, a ferramenta não apresenta suporte profissional e não oferece recursos para paralelização de código para uso eficiente de computadores *multicore*. Além disso, usa uma representação computacional densa para a matriz termo-documento, que seria mais bem representada e manipulada via uma representação esparsa.

[14] Afirmação baseada nas citações recebidas pelo relatório técnico relacionado à ferramenta, de acordo com o Google Acadêmico.

Tabela 4. Experimento 1 - similaridade do pré-processamento.

Aspecto	Experimento 1
Documentos de entrada	documento 53068
Confguração do processo	**Case-folding**: minúsculas. **Tokenização**: alfanumérica. **Remoção de stop words**: lista de *stop words* utilizada (idioma inglês). **Radical.**: Snowball para idioma inglês. Lista de *stop words* utilizada para o idioma inglês, composta pelas seguintes palavras: *'i'*, *'me'*, *'my'*, *'myself'*, *'we'*, *'our'*, *'ours'*, *'ourselves'*, *'you'*, *'your'*, *'yours'*, *'yourself'*, *'yourselves'*, *'he'*, *'him'*, *'his'*, *'himself'*, *'she'*, *'her'*, *'hers'*, *'herself'*, *'it'*, *'its'*, *'itself'*, *'they'*, *'them'*, *'their'*, *'theirs'*, *'themselves'*, *'what'*, *'which'*, *'who'*, *'whom'*, *'this'*, *'that'*, *'these'*, *'those'*, *'am'*, *'is'*, *'are'*, *'was'*, *'were'*, *'be'*, *'been'*, *'being'*, *'have'*, *'has'*, *'had'*, *'having'*, *'do'*, *'does'*, *'did'*, *'doing'*, *'a'*, *'an'*, *'the'*, *'and'*, *'but'*, *'if'*, *'or'*, *'because'*, *'as'*, *'until'*, *'while'*, *'of'*, *'at'*, *'by'*, *'for'*, *'with'*, *'about'*, *'against'*, *'between'*, *'into'*, *'through'*, *'during'*, *'before'*, *'after'*, *'above'*, *'below'*, *'to'*, *'from'*, *'up'*, *'down'*, *'in'*, *'out'*, *'on'*, *'off'*, *'over'*, *'under'*, *'again'*, *'further'*, *'then'*, *'once'*, *'here'*, *'there'*, *'when'*, *'where'*, *'why'*, *'how'*, *'all'*, *'any'*, *'both'*, *'each'*, *'few'*, *'more'*, *'most'*, *'other'*, *'some'*, *'such'*, *'no'*, *'nor'*, *'not'*, *'only'*, *'own'*, *'same'*, *'so'*, *'than'*, *'too'*, *'very'*, *'s'*, *'t'*, *'can'*, *'will'*, *'just'*, *'don'*, *'should'*, *'now'*, *'d'*, *'ll'*, *'m'*, *'o'*, *'re'*, *'ve'*, *'y'*, *'ain'*, *'aren'*, *'couldn'*, *'didn'*, *'doesn'*, *'hadn'*, *'hasn'*, *'haven'*, *'isn'*, *'ma'*, *'mightn'*, *'mustn'*, *'needn'*, *'shan'*, *'shouldn'*, *'wasn'*, *'weren'*, *'won'* e *'wouldn'*.
Resultados esperados	As ferramentas produzem listas de *tokens* idênticas.
Resultados obtidos	A ferramenta PRETEXT e a implementação em PYTHON/NLTK produziram listas idênticas de *tokens* (70 *tokens*), salvo diferenças causadas pela redução para o radical. Já o *pipeline* em R produz uma lista com 50 *tokens*, cuja análise de divergência segue abaixo.
Análise de divergências	Os *tokens 'us'*, *'dai'*, *'wai'* e *'decai'* identificados pelo PRETEXT são representados como *'use'*, *'day'*, *'way'* e *'decay'* pela implementação em PYTHON/NLTK. A divergência se deve por implementações distintas das regras 1c e 5b do algoritmo de Porter para redução para o radical em inglês. *Tokens* identificados pelo PRETEXT, mas não identificados pelo *pipeline* em R: *'153'*, *'cb'*, *'cbnewsj'*, *'wwc'*, *'edu'*, *'wai'*, *'na'*, *'2'*, *'decai'*, *'healta'*, *'distribut'*, *'dai'*, *'line'*, *'saturn'*, *'12'*, *'organ'*, *'18'*, *'us'*, *'att'*, *'r'*, *'735242337'* e *'com'*. *Tokens* identificados pelo *pipeline* em R, mas não identificados pelo PRETEXT: *'use'*, *'healta153735242337saturnwwcedu'*, *'wasnt'*, *'deankaflowitz'*, *'your'*, *'way'*, *'healtasaturnwwcedu'*, *'decaycbnewsjcbattcom'*, *'day'* O R tokeniza de uma maneira diversa das demais ferramentas, sendo sensível a e-mails ou a possíveis erros de digitação ou erro de espaçamento.

Tabela 5. Experimento 2 - Similaridade da extração de características.

Aspecto	Experimento 2
Documentos de entrada	documentos 53068 e 53257
Configuração do processo	**Case-folding**: minúsculas. **Tokenização**: alfanumérica. **Remoção de stop words**: lista de *stop words* usada no Experimento 1. **Radical.**: Snowball para idioma inglês. ***scores***: *tf-idf* não normalizado.
Resultados esperados	As ferramentas produzem, para um mesmo documento, a mesma representação vetorial usando *scores tf-idf*.
Resultados obtidos	O PRETEXT e a implementação em PYTHON/NLTK produzem vetores similares para cada documento (precisão de uma casa decimal). Já o *pipeline* em R produz vetores com scores diferentes.
Análise de divergências:	Foi passado os parâmetros conforme especificado na biblioteca TM sobre a função TERMDOCUMENTMATRIX, mas a saída da função segue um padrão diferente das outras duas ferramentas.

- **PYTHON e NLTK**: Ambas tecnologias possuem uma boa documentação, podendo ser programadas usando recursos de paralelização que fazem um bom uso de computadores *multicore* e de *clusters* de computadores. Implementações combinada com a plataforma *Apache Spark* foram testadas neste estudo e funcionam de maneira adequada. Gera matriz termo-documento usando estrutura de dados em lista de termo-*score*. Por outro lado, não possui funcionalidades pré-programadas para manipulação de n-gramas ou cortes por frequência, como oferecido pela PRETEXT. A necessidade de programação *on demand* exige que o usuário realize testes para validação e verificação do código desenvolvido.
- ***Pipeline* em R**: Apresenta condições de pré-processamento mais sofisticado entre as três ferramentas testadas – por exemplo, identifica endereços eletrônicos como um único *token*. A necessidade de programação *on demand* exige que o usuário realize testes para validação e verificação do código desenvolvido. Possui documentação limitada para a biblioteca TM. Não foram testadas neste estudo o uso de estruturas de dados eficientes para representação de matriz esparsa e comandos de paralelização do código.

7. Conclusão

A atividade de pré-processamento é essencial para qualquer tarefa de extração de conhecimento de bases textuais. Este relatório apresentou uma análise comparativa de três implementações de pré-processamento de texto distintas. Os experimentos conduzidos sugerem que, apesar das ferramentas apresentarem abordagens diferentes, produzem representações muito similares.

Embora o teste de resultados de análise de dados e de descoberta de conhecimento a partir de tais representações não tenha sido testado neste estudo, é possível que eles sejam

equivalentes mesmo quando gerados sob as representações levemente diferentes gerados pelas ferramentas testadas. A verificação desta hipótese é uma oportunidade de estudo para um próximo relatório técnico.

A ferramenta PRETEXT destaca-se como a ferramenta que apresenta maior gama de funcionalidades e variações. Os pacotes NLTK e TM são mais flexíveis, permitindo que o usuário crie seu *pipeline* como achar conveniente. Para grandes coleções de documentos, recomenda-se o uso da biblioteca NLTK combinada com a plataforma *Apache Spark* ou a utilização dos parâmetros de cortes de atributos por frequência oferecidos pelo PRETEXT.

A comparação entre as ferramentas PRETEXT, NLTK e R, tanto sob o aspecto funcional, quanto em relação às implementações do processo de pré-processamento de texto, contribui para que pesquisadores iniciantes na área tenham condições melhores de selecionar a ferramenta adequada para um determinado contexto, podendo ainda reaproveitar, adaptar ou estender as implementações de acordo com a conveniência da pesquisa.

Referências

Baeza-Yates, R. A. e Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Benoit, K., Watanabe, K., Nulty, P., Obeng, A., Wang, H., Lauderdale, B., e Lowe, W. (2017). *quanteda: Quantitative Analysis of Textual Data*. R package version 0.99.

Bird, S., Klein, E., e Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Feinerer, I. e Hornik, K. (2017). *tm: Text Mining Package*. R package version 0.7-1.

Ferrucci, D., Lally, A., Verspoor, K., e Nyberg, E. (2009). Unstructured information management architecture (UIMA) version 1.0. OASIS Standard.

Jones, K. S. e Willet, P. (1997). *Readings in Information Retrieval*. Morgan Kaufmann, San Francisco.

Lang, K. (1995). Newsweeder: Learning to filter netnews. Em *Proceedings of the Twelfth International Conference on Machine Learning*, páginas 331–339.

Loper, E. e Bird, S. (2002). Nltk: The natural language toolkit. Em *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, páginas 63–70. Association for Computational Linguistics.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165.

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Matsubara, E. T., Martins, C. A., e Monard, M. C. (2003). Pretext: Uma ferramenta para pré-processamento de textos utilizando a abordagem bag-of-words. *Relatório Técnico*, 209.

McCallum, A. (1998). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering.

Moura, M. F., Nogueira, B. M., da Silva Conrado, M., dos Santos, F. F., e Rezende, S. O. (2008). Making good choices of non-redundant n-gramwords. Em *Computer and Information Technology, 2008. ICCIT 2008. 11th International Conference on*, páginas 64–71. IEEE.

Moura, M. F., Nogueira, B. M., da Silva Conrado, M., dos Santos, F. F., e Rezende, S. O. (2010). Um modelo para a seleção de n-gramas significativos e não redundantes em tarefas de mineração de textos. *Boletim de Pesquisa e Desenvolvimento. Campinas: EMBRAPA Informática Agropecuária*, 23.

Nadkarni, P. M., Ohno-Machado, L., e Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551.

Pereira, R. G. e Moura, M. F. (2015). I-preproc: uma ferramenta para pré-processamento e indexação incremental de documentos.

Porter, M. F. (1980a). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Porter, M. F. (1980b). An algorithm for suffix stripping. *Program*, 14(3):130–137.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Rezende, S. O., Marcacini, R. M., e Moura, M. F. (2011). O uso da mineração de textos para extração e organização não supervisionada de conhecimento. *Revista de Sistemas de Informação da FSMA*, 7:7–21.

Rijsbergen, C. J., Robertson, S. E., e Porter, M. F. (1980). New models in probabilistic information retrieval. Technical Report 5587, British Library Research and Development Report, London: British Library.

Rijsbergen, C. J. V. (1979). *Inf. Retrieval*. Butterworths, London, 2nd edição.

Rinker, T. W. (2013). *qdap: Quantitative Discourse Analysis Package*. University at Buffalo/SUNY, Buffalo, New York. 2.2.7.

Robertson, S. (2004). Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520.

Salton, G., Wong, A., e Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Silge, J. e Robinson, D. (2016). tidytext: Text mining and analysis using tidy data principles in r. *JOSS*, 1(3).

Silva, L. A., Peres, S. M., e Boscarioli, C. (2016). *Introdução á Mineração de Dados: Com Aplicações em R*. Elsevier.

Soares, M. V. B., Prati, R. C., e Monard, M. C. (2008). *Pretext II: Descriçao da reestruturacao da ferramenta de pré-processamento de textos*. ICMC-USP.

Turney, P. D., Pantel, P., et al. (2010). From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188.

Zeimpekis, D. e Gallopoulos, E. (2005). TMG: A MATLAB toolbox for generating term-document matrices from text collections. Technical Report HPCLAB-SCG 1/01-05, University of Patras.