



Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Análise de textos parlamentares

Autor: Matheus Souza Fernandes
Orientador: Prof. Dr. Fábio Macedo Mendes

Brasília, DF
2016



Matheus Souza Fernandes

Análise de textos parlamentares

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Fábio Macedo Mendes

Brasília, DF

2016

Matheus Souza Fernandes

Análise de textos parlamentares/ Matheus Souza Fernandes. – Brasília, DF, 2016-

45 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Fábio Macedo Mendes

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. processamento de linguagem natural. 2. aprendizado de máquina. I. Prof. Dr. Fábio Macedo Mendes. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Análise de textos parlamentares

CDU 02:141:005.6

Matheus Souza Fernandes

Análise de textos parlamentares

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 01 de junho de 2013:

Prof. Dr. Fábio Macedo Mendes
Orientador

Paulo Roberto Miranda Meirelles
Convidado 1

Titulação e Nome do Professor
Convidado 02
Convidado 2

Brasília, DF
2016

Resumo

O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (...) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto. O texto pode conter no mínimo 150 e no máximo 500 palavras, é aconselhável que sejam utilizadas 200 palavras. E não se separa o texto do resumo em parágrafos.

Palavras-chaves: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Figura 1 – Exemplo de clusterização	25
Figura 2 – k centróides (coloridos) recebem valores iniciais.	26
Figura 3 – Cálculo das distâncias entre os pontos e os centróides.	26
Figura 4 – Novos centróides definidos pela media dos elementos do <i>cluster</i>	26
Figura 5 – O algoritmo converge quando nenhum ponto muda de <i>cluster</i>	27
Figura 6 – Comparação entre distância euclidiana e de manhattan	28
Figura 7 – Representação do teste de caixa-preta	31

Lista de tabelas

Tabela 1 – Exemplos de <i>Stop Words</i>	24
--	----

Listings

2.1	Representação de um trecho no modelo bag-of-words	21
-----	---	----

Sumário

	Listings	13
1	INTRODUÇÃO	17
1.1	Contextualização	17
1.2	Objetivo	18
1.2.1	Contribuições Tecnológicas	18
1.2.2	Contribuições Científicas	18
1.3	Metodologia	18
1.4	Organização do Trabalho	19
2	PROCESSAMENTO DE LINGUAGEM NATURAL	21
2.1	Pré-processamento: modelo <i>bag-of-words</i>	21
2.1.1	Modelo <i>N-Gram</i>	22
2.1.2	Representação dos Termos	22
2.1.2.1	<i>Boolean</i>	22
2.1.2.2	<i>Term Frequency</i>	22
2.1.2.3	<i>Term Frequency - Inverse Document Frequency</i>	23
2.1.3	Dimensionalidade dos documentos	23
2.1.3.1	Stemização	23
2.1.3.2	<i>Stop Words</i>	24
2.2	Clusterização	25
2.2.1	Algoritmo <i>k-means</i>	25
2.2.1.1	Distância entre os pontos	27
3	ENGENHARIA DE SOFTWARE	29
3.1	<i>Framework</i> de gerenciamento <i>SCRUM</i>	29
3.1.1	Time <i>SCRUM</i>	29
3.1.1.1	<i>Product Owner</i>	29
3.1.1.2	Time de Desenvolvimento	29
3.1.1.3	<i>Scrum Master</i>	29
3.1.2	Eventos do <i>SCRUM</i>	30
3.1.2.1	<i>Sprint</i>	30
3.1.2.2	Reunião de planejamento de <i>sprint</i>	30
3.1.2.3	Revisão de <i>sprint</i>	30
3.1.3	Artefatos do <i>SCRUM</i>	30
3.1.3.1	<i>Backlog</i> de produto	30

3.1.3.2	<i>Backlog de sprint</i>	31
3.2	Testes de Software	31
3.2.1	Classificação dos Testes	31
3.2.1.1	Testes de caixa-preta	31

REFERÊNCIAS	33
------------------------------	-----------

APÊNDICES	35
------------------	-----------

APÊNDICE A – PRIMEIRO APÊNDICE	37
---	-----------

APÊNDICE B – SEGUNDO APÊNDICE	39
--	-----------

ANEXOS	41
---------------	-----------

ANEXO A – PRIMEIRO ANEXO	43
---	-----------

ANEXO B – SEGUNDO ANEXO	45
--	-----------

1 Introdução

1.1 Contextualização

O desenvolvimento de novas ferramentas de interação entre governo e sociedade é fundamental para o avanço da democracia ([MAZONI, 2011](#)). Porém, torna-se imprescindível a aplicação do conceito de Dados Abertos por parte do Governo para que essas novas ferramentas tenham efeitos significativos para a sociedade.

O conceito para Dado Aberto considerado neste trabalho é o definido pela [Open Knowledge \(2016\)](#), que estabelece que um dado (ou um conhecimento) é aberto quando estiver livre para uso, reuso e redistribuição. Ou seja, a informação deve estar disponível a todos, sem restrições de *copyright*, patentes ou outros mecanismos de controle. Além disso, esses dados devem ser independentes de tecnologia, baseados em formatos padronizados e desvinculados de ferramentas que os originaram. Os dados devem permitir sua manipulação por máquinas e possuir metadados que permitam identificar sua natureza, origem e qualidade ([DINIZ, 2010](#)).

No contexto governamental, os Dados Abertos fortalecem três características indispensáveis para a democracia: transparência, participação e colaboração ([MAZONI, 2011](#)). Transparência tem o papel de informar a sociedade sobre as ações que estão sendo tomadas ou que serão tomadas pelo governo. Participação permite que os cidadãos auxiliem o poder público a elaborar políticas mais eficazes. Finalmente, a colaboração entre a sociedade, diferentes níveis de governo e a iniciativa privada permitem aprimorar a eficácia do Estado.

De acordo com a Lei nº12.527/2011, também conhecida como Lei de Acesso à Informação, qualquer cidadão, sem necessidade de justificativa, pode solicitar dados ou informações à qualquer órgão ou entidade pública dos poderes Executivo, Legislativo e Judiciário, além do Ministério Público, nas esferas Federal, Estadual e Municipal ([BRASIL, 2011](#)). Para atender à lei mencionada anteriormente, a [Câmara dos Deputados \(2016\)](#) criou um portal que tem como objetivo disponibilizar dados brutos para a utilização em aplicações desenvolvidas pelos cidadãos e entidades da sociedade civil que permitam a percepção mais efetiva das atividades parlamentares, .

A publicação de dados pressupõe o uso de tecnologias que garantam que eles possam ser acessados e reutilizados por máquinas. Apesar de não garantir que os dados estarão disponíveis em um formato conveniente de uso imediato, possibilita o cruzamento de diferentes bases dados e a exibição destes de forma que possam ser melhor apresentados à sociedade ([DINIZ, 2010](#)).

1.2 Objetivo

O objetivo desse trabalho é utilizar técnicas de processamento de linguagem natural para, através da análise dos discursos e proposições dos parlamentares, determinar um perfil temático para os deputados, bem como evidenciar o termos mais utilizados em seus discursos e proposições e, assim, permitir que o cidadão veja, de forma comparativa, o que seus representantes no parlamento mais dizem em seus discursos e o que mais dizem em suas proposições.

1.2.1 Contribuições Tecnológicas

- Implementar biblioteca Python para consumo de dados abertos governamentais, com foco nos dados abertos da Câmara dos deputados.
- Implementar aplicação Django para utilização e persistência dos dados abertos governamentais, também com foco nos dados aberto da Câmara dos deputados.
- Implementar sistema web de comparação entre discursos e proposições parlamentares, a ser detalhado no decorrer deste trabalho.

1.2.2 Contribuições Científicas

- Estudo teórico sobre Processamento de Linguagem Natural.
- Estudo teórico sobre métodos estatísticos aplicados à análise de textos.

1.3 Metodologia

Devido à natureza deste trabalho, nota-se que o modelo de pesquisa adequado deve possuir características tanto da pesquisa exploratória quanto da pesquisa experimental. Outro método de pesquisa que será utilizado é a pesquisa-ação, onde existirão ciclos de coleta e análise de dados. A cada ciclo, a análise dos dados do ciclo anterior servirão de insumo para tomadas de decisão no desenvolvimento do projeto.

Durante a fase de desenvolvimento do sistema, pretende-se utilizar uma abordagem de ágil, com a aplicação de algumas práticas adaptadas do *framework* Scrum:

- Nas retrospectivas de *sprint*, serão realizadas avaliações, para identificar os pontos fortes e fracos da *sprint* que terminou, bem como propor possíveis ações que poderão ser tomadas para mitigar os pontos fracos, na próxima *sprint*.

- *Daily meetings*, onde serão feitos comentários sobre o que foi realizado no dia e dúvidas poderão ser reportadas. Acontecerão diária e remotamente, para o melhor acompanhamento do orientador.
- Será definido um *backlog* de produto e um para cada *sprint*.

1.4 Organização do Trabalho

2 Processamento de Linguagem Natural

Este capítulo contém o referencial teórico que diz respeito ao Processamento de Linguagem Natural.

2.1 Pré-processamento: modelo *bag-of-words*

A grande parte dos dados que serão utilizados nesse trabalho estão dispostos em formato de texto, ou seja, um formato não estruturado que dificulta a extração de informações. Um método comum do processamento de texto em linguagem natural é o modelo *bag-of-words*, onde o texto é representado na forma de um vetor de frequência de palavras que ocorrem no texto (MATSUBARA et al., 2003).

A representação computacional de um texto no modelo mencionado é feita através de um dicionário onde suas chaves são os termos presentes no documento e os valores são suas respectivas frequências. Tomando o trecho “fui à padaria e comprei pão” como exemplo a sua representação no modelo *bag-of-words* corresponde a:

```
1 bag_of_words = {  
2     "fui": 1,  
3     "à": 1,  
4     "padaria": 1,  
5     "e": 1,  
6     "comprei": 1,  
7     "pão": 1,  
8 }
```

Listing 2.1 – Representação de um trecho no modelo bag-of-words

É fácil ver que esta representação torna a ordem das palavras irrelevante. As frases “e comprei fui padaria pão à” e “à pão comprei padaria e fui” também possuem as mesmas representações.

Um vetor também pode ser uma forma de representar um texto. Nesse caso, cada índice do vetor representa uma palavra e cada elemento corresponde à frequência da mesma.

A linguagem de programação *Python* possui, nativamente, ferramentas que facilitam a contagem de elementos de um dicionário. O `Counter`¹ é uma subclasse de `dict` e assim como a classe pai, é uma coleção não ordenada de elementos. Os elementos são armazenados como chaves de dicionário e seus contadores como valores, que podem assumir

¹ <https://docs.python.org/3/library/collections.html>

qualquer valor inteiro, incluindo zero e negativos. Isso torna o **Counter** uma representação ideal para o modelo *bag-of-words*, já que ele, naturalmente, conta a quantidade de vezes que um termo aparece dentro de uma coleção (uma lista, tupla ou dicionário, por exemplo).

2.1.1 Modelo *N-Gram*

O modelo de representação de textos através de um conjunto de palavras pode limitar qualitativamente a análise que será realizada, já que frases não são consideradas. Por exemplo, “Santa Catarina”, um estado brasileiro, possui um significado completamente diferente quando as palavras “Santa” (mulher canonizada), e “Catarina” (nome feminino) são analisadas separadamente.

Um *n*-grama é uma sequência de *n* elementos dentro de um texto. Os elementos podem ser palavras, sílabas, letras ou qualquer outra base. Um *n*-grama de tamanho 1 é chamado de unigrama, de tamanho 2, bigrama, e de tamanho 3, trigrama. Sequências com 4 ou mais elementos são chamados de *n*-gramas. Usando a frase “Eu não gostei desse filme” como exemplo, temos os seguintes unigramas: “eu”, “não”, “gostei”, “desse” e “filme”. Os seguintes bigramas: “eu não”, “não gostei”, “gostei desse” e “desse filme”. E os seguintes trigramas: “eu não gostei”, “não gostei desse” e “gostei desse filme”.

2.1.2 Representação dos Termos

A representação de um termo a_i pode se dar de diferentes maneiras, como a quantidade de vezes que o termo aparece em um texto ou apenas se o termo aparece no texto, por exemplo. Alguns dos tipos de representação serão expostos a seguir.

2.1.2.1 *Boolean*

Essa medida usa a representação binária para os termos presentes nos documentos, onde o valor de a_i é 0 quando o termo não aparece nenhuma vez no documento ou 1 quando aparece uma ou mais vezes. Essa medida é muito simples e, geralmente, modelos estatísticos levam em consideração também a frequência com que os termos se repetem nos documentos (SALTON; BUCKLEY, 1988).

2.1.2.2 *Term Frequency*

Ao contrário da medida mencionada anteriormente, a medida *term frequency* considera a quantidade de ocorrências do termo t dentro do documento

$$f_t = \frac{n_t}{N}, \quad (2.1)$$

onde n_t é a quantidade de vezes que o termo t aparece dentro do documento e N a quantidade total de termos do documento.

Alguns termos comuns podem aparecer na maioria dos documentos sem fornecer informações úteis em uma tarefa de mineração de textos (MATSUBARA et al., 2003).

2.1.2.3 Term Frequency - Inverse Document Frequency

Para diminuir a influência de termos comuns, é possível utilizar um fator de ponderação, para que os termos que aparecem na maioria dos documentos tenham valores numéricos menores do que aqueles que raramente aparecem (MATSUBARA et al., 2003). Segundo JONES (1972), a especificidade de um termo pode ser quantificada por uma função inversa do número de documentos em que ele ocorre, essa função varia entre 0 e $\log N_d$, onde N_d é o número total de documentos e $d(t)$ a quantidade de documentos nos quais o termo t aparece ao menos uma vez:

$$i(t) = \log \frac{N_d}{d(t)} \quad (2.2)$$

Portanto, o valor final de a_i é dado pela equação:

$$f(t) = \frac{n_t}{N} \cdot \log \frac{N_d}{d(t)}, \quad (2.3)$$

onde $\frac{n_t}{N}$ é a frequência do termo dentro do texto e $\log \frac{N_d}{d(t)}$ sua taxa de ponderação.

2.1.3 Dimensionalidade dos documentos

A representação vetorial de uma coleção de documentos no modelo *bag-of-words* pressupõe um espaço dimensional igual ao número de termos presentes em toda a coleção de documentos. Suponha que analisou-se 10 documentos e, em média, foram retirados 200 novos termos de cada um. Logo, a dimensionalidade média dos vetores serão de, aproximadamente, 2000. Se a maior parte dos termos aparecer em apenas um ou dois documentos, teremos a maior parte das componentes vetoriais nula. (MATSUBARA et al., 2003). Existem métodos cujo objetivo é diminuir a dimensionalidade desses vetores. Dentre eles, citamos a transformação de cada termo no radical de origem, utilizando algoritmos de *stemming*.

2.1.3.1 Stemização

A stemização (do inglês, *stemming*) é o processo de reduzir palavras flexionadas à sua raiz, essa redução não precisa, necessariamente, chegar à raiz morfológica da palavra. A raiz obtida geralmente é o suficiente para mapear palavras relacionadas à um valor comum, mesmo se este não for uma raiz válida. O estudo de algoritmos de *stemming* é foco de pesquisas desde a década de 60 e o primeiro algoritmo foi publicado por Lovins (1968).

A consequência da aplicação de algoritmos de *stemming* consiste na remoção de prefixos ou sufixos de um termo e ou da transformação de verbos para suas formas no infinitivo. Por exemplo, as palavras **ouvir**, **ouvi**, **ouviriam**, **ouve** e **ouvindo** seriam reduzidas para um mesmo *stem*: **ouv**. Esse método diminui, portanto, a dimensionalidade dos vetores e dicionários dentro de uma *bag-of-words*. Ao invés de analisar a frequência dos termos, analisamos a quantidade de vezes que um *stem* aparece em um documento.

É evidente que os algoritmos de *stemming* são dependentes do idioma analisado. O algoritmo de Porter (1980), um dos algoritmos de *stemming* mais conhecidos, remove os sufixos de termos em inglês, tem sido amplamente utilizado, referenciado e adaptado desde sua criação. É possível adaptá-lo para a língua portuguesa considerando que as línguas provenientes do latim possuem formas verbais conjugadas em sete tempos e com sete terminações distintas.

Devido ao fato de uma linguagem ter tantas regras e exceções, é pouco provável que o algoritmo de *stemming* retorne o mesmo *stem* para todas as palavras que tenham a mesma origem ou radical morfológico. Pode-se dizer, também, que a medida que o algoritmo vai se tornando específico o suficiente para atender todas essas regras e exceções a eficiência do algoritmo também diminui (IMAMURA, 2001).

2.1.3.2 Stop Words

Palavras que possuem pouco ou nenhum valor semântico, como "e", "de" e "seus", são conhecidas como *Stop Words* e, por não agregarem valor à análise textual, são removidas durante o pré-processamento (RAJARAMAN; ULLMAN, 2011). Essas palavras não são exclusividade de uma linguagem específica e geralmente representam a maioria dos termos de um texto. No caso da língua inglesa, por exemplo, palavras como "of" e "the" também não possuem nenhum valor para a análise. A lista de *Stop Words* obviamente varia de acordo com a linguagem que está sendo analisada (LOPES, 2015). O quadro abaixo mostra a lista de *Stop Words* consideradas neste trabalho.

de	os	tua	tem	estão	da	lhes	essas
e	é	foi	nossas	muito	o	se	tuas
tu	por	as	sua	aquele	entre	não	ele
delas	minhas	às	nos	pela	havia	me	como
ser	aqueles	nossa	vocês	eu	ter	tenho	suas
está	isso	pelos	estes	tinha	depois	foram	este
para	só	quem	deles	isto	um	eles	do
vos	mais	mesmo	num	dele	será	minha	a
no	teus	à	você	em	meus	esses	pelas
com	ao	dela	há	que	na	nosso	te
aos	dos	ou	aquela	era	uma	das	esta
teu	nem	já	até	seja	esse	mas	quando
aquelas	nossos	têm	também	seus	lhe	meu	seu
ela	elas	estas	nós	sem	essa	fosse	qual
		pelo	nas	numa	aquilo		

Tabela 1 – Exemplos de *Stop Words*

2.2 Clusterização

Quando temos um conjunto de elementos, naturalmente tentamos estabelecer padrões entre eles. Uma forma natural de definir padrões em um conjunto é analisar a distância entre seus componentes. Dessa forma, quanto mais parecidos dois elementos são, mais próximos eles estão. A figura abaixo mostra um conjunto de elementos com duas características: forma (quadrado, círculo e triângulo) e cor (tons de vermelho, verde e azul). Ao lado temos os mesmos elementos agrupados em três conjuntos com características semelhantes. No grupo 1, por exemplo, todos os elementos possuem uma tonalidade de vermelho e o formato quadrado.

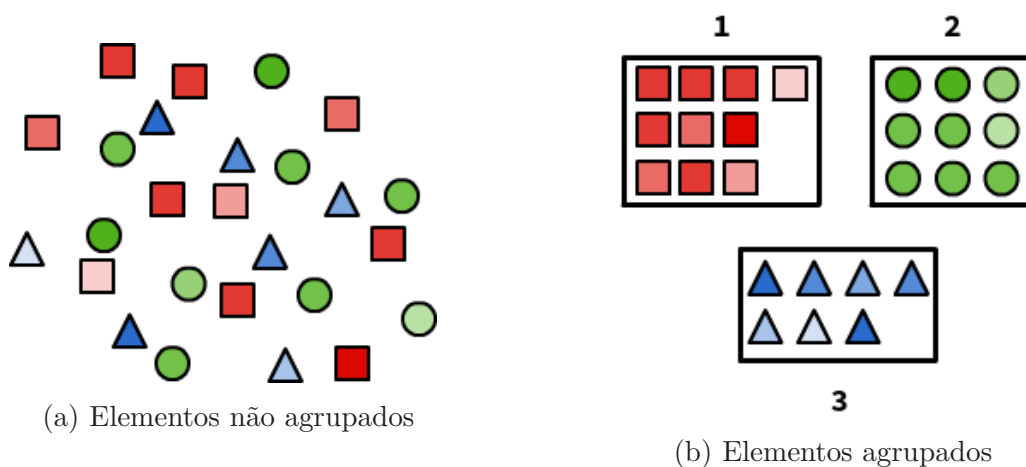


Figura 1 – Exemplo de clusterização

A clusterização é uma técnica da mineração de dados que consiste, justamente, em realizar o procedimento descrito acima: organizar um conjunto de elementos, usualmente representados por vetores ou pontos em um espaço multidimensional, em *clusters* (ou agrupamentos), de acordo com alguma medida de similaridade. Ela representa uma das principais etapas da análise de dados, denominada análise de *clusters* (JAIN; MURTY; FLYNN, 1999).

Não existe uma técnica de clusterização universal capaz de revelar toda a variedade de estruturas que podem estar presentes em conjuntos de dados multidimensionais. Diferentes algoritmos dependem implicitamente de certas hipóteses a respeito da forma dos clusters na definição da medida de similaridade e dos critérios de agrupamento (ESTIVILL-CASTRO, 2002).

2.2.1 Algoritmo *k-means*

O algoritmo de clusterização *k-means*, proposto por Lloyd (1957), tem o objetivo de dividir N elementos em k grupos, onde cada elemento pertence ao *cluster* mais próximo e k deve ser um valor informado a priori e menor ou igual à quantidade de elementos.

Os principais passos do algoritmo são:

1. **Gerar centróides:** neste passo os k centróides recebem valores iniciais. O valor inicial dos centróides podem ser definidos randomicamente, através de uma Gaussiana (com média e variância estimados a partir do conjunto de elementos) ou escolhendo um dos N elementos como centróides iniciais, uniformemente por meio da distribuição uniforme do próprio conjunto de elementos ou escolhendo arbitrariamente os centróides iniciais.

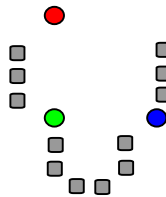


Figura 2 – k centróides (coloridos) recebem valores iniciais.

2. **Calcular distâncias:** aqui são calculadas as distâncias entre cada ponto e cada centróide. É a parte com maior peso computacional do algoritmo, já que o cálculo é realizado para cada ponto.
3. **Classificar os pontos:** cada ponto deve ser classificado de acordo com a distância entre ele e o centróide de cada *cluster*. O ponto pertencerá ao *cluster* cujo centróide está mais próximo. O algoritmo converge quando, em uma iteração, nenhum ponto mudar de *cluster*.

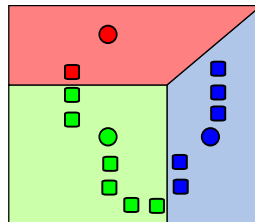


Figura 3 – Cálculo das distâncias entre os pontos e os centróides.

4. **Calcular novos centróides:** para cada *cluster*, um novo centróide é definido como a média desses pontos.

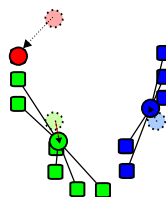


Figura 4 – Novos centróides definidos pela media dos elementos do *cluster*.

5. **Repetir até convergir:** retorna ao passo 2. Como o resultado do algoritmo depende da escolha dos centróides iniciais, a convergência não é garantida. Por isso, normalmente o algoritmo é executado várias vezes.

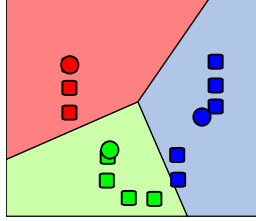


Figura 5 – O algoritmo converge quando nenhum ponto muda de *cluster*.

A unidade utilizada para a medida da distância pode afetar a análise (COLE, 1998). Sugere-se, então, que seja feito o processo de normalização (*whitening*) dos dados antes da clusterização. A normalização consiste em ajustar a escala das distâncias de forma que os valores fiquem em pequenos intervalos, como de 0 a 1, por exemplo.

2.2.1.1 Distância entre os pontos

Segundo Cole (1998), para clusterizar termos de acordo com sua similaridade, deve-se definir uma medida de quão próximos dois termos estão. Uma medida de distância deve ser definida de tal forma que:

- Seja sempre positiva.
- Seja simétrica: a distância de um termo a_i para um termo a_j deve ser a mesma de a_j para a_i .
- Seja reflexiva: se a distância entre a_i e a_j é zero, então $a_i = a_j$.
- Respeite a desigualdade triangular: considerando os termos $(a_i, a_j \text{ e } a_k)$, a distância $d(a_i, a_k)$ deve ser menor ou igual à soma das distâncias $d(a_i, a_j)$ e $d(a_j, a_k)$

O cálculo das distâncias pode ser realizado de diversas maneiras e estão relacionadas à uma distribuição. A distância euclidiana entre dois pontos, $A = (a_1, a_2, a_3, \dots, a_n)$ e $B = (b_1, b_2, b_3, \dots, b_n)$ é dada pela equação

$$dist(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2 + \dots + (a_n - b_n)^2} \quad (2.4)$$

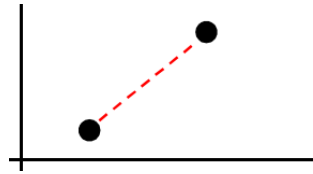
$$dist(A, B) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (2.5)$$

Já a distância de Manhattan entre dois pontos, $A = (a_1, a_2, a_3, \dots, a_n)$ e $B = (b_1, b_2, b_3, \dots, b_n)$, é dada pela soma das diferenças absolutas de suas coordenadas:

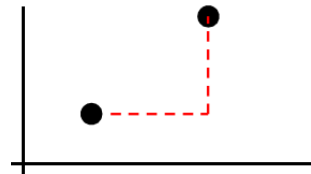
$$\text{dist}(A, B) = |a_1 - b_1| + |a_2 - b_2| + |a_3 - b_3| + \dots + |a_n - b_n| \quad (2.6)$$

$$\text{dist}(A, B) = \sum_{i=1}^n |a_i - b_i| \quad (2.7)$$

Em ambos os casos, temos que a distância entre do ponto A ao ponto B é a mesma distância do ponto B ao ponto A.



(a) Distância Euclidiana



(b) Distância de Manhattan

Figura 6 – Comparação entre distância euclidiana e de manhattan

3 Engenharia de Software

Este capítulo contém o referencial teórico que diz respeito à Engenharia de Software utilizada nesse trabalho.

3.1 *Framework* de gerenciamento *SCRUM*

O *SCRUM* é um *framework* de gerenciamento de projetos cuja característica principal é o desenvolvimento de um produto complexo e tem como objetivo entregar o produto com o maior valor agregado possível (SUTHERLAND; SCHWABER, 1995).

Segundo Sutherland e Schwaber (1995) os três pilares que sustentam o *SCRUM* são:

- **Transparência:** manter os resultados sempre visíveis aos interessados, para que todos possam ter um mesmo ponto de vista.
- **Inspeção:** deve-se, frequentemente, inspecionar os artefatos gerados, desde que esse processo não atrapalhe o real trabalho.
- **Adaptação:** uma vez encontrado um erro em qualquer artefato, deve-se ajustar o mesmo o mais rápido possível, para evitar maiores erros.

3.1.1 Time *SCRUM*

O time *SCRUM* é formado por três partes distintas: o *product owner*, o time de desenvolvimento e o *scrum master* (SUTHERLAND; SCHWABER, 2013).

3.1.1.1 *Product Owner*

Responsável por manter os itens no *backlog* do produto, tornando ele visível e claro a todos, garantindo que o time de desenvolvimento entenda-os.

3.1.1.2 Time de Desenvolvimento

Composto por desenvolvedores, o time de desenvolvimento é responsável por transformar o *backlog* do produto em incrementos de software.

3.1.1.3 *Scrum Master*

É o responsável pelo cumprimento das práticas do *SCRUM*, facilitando os eventos propostos pelo *framework*, trabalhando em conjunto com o *product owner* para o esclare-

cimentos dos itens do *backlog* para a equipe de desenvolvimento, além de liderar o time em seu autogerenciamento.

3.1.2 Eventos do *SCRUM*

No *SCRUM*, todos os eventos possuem uma duração determinada, que varia de acordo com a necessidade do projeto e equipe. São eles: *sprint*, reunião de planejamento de *sprint* e revisão da *sprint* (SUTHERLAND; SCHWABER, 2013).

3.1.2.1 *Sprint*

A *sprint* é o evento mais importante, pois ela agrupa todos os outros eventos. Uma *sprint* possui um tempo fixo, que pode variar de acordo com a necessidade da equipe, e pode ser composta de outras atividades e eventos do *SCRUM*, como reuniões diárias, desenvolvimento do produto, retrospectivas, etc. Uma *sprint* sempre começa quando uma acaba (SUTHERLAND; SCHWABER, 2013).

3.1.2.2 Reunião de planejamento de *sprint*

Ocorre sempre no início de cada *sprint* e, como resultado da reunião, é definido o que será feito durante a *sprint* que se inicia. A partir disso, é criado o *backlog* de *sprint*, onde estão presentes as metas da *sprint*.

3.1.2.3 Revisão de *sprint*

Acontece no final de cada *sprint* e tem como objetivo a apresentação dos resultados obtidos durante a *sprint* que passou, tanto o que foi concluído quanto o que não foi, bem como as justificativas para a não completude. Nessa reunião, todos os membros do time devem estar presentes.

3.1.3 Artefatos do *SCRUM*

Durante o andamento do projeto, alguns artefatos são produzidos, com o objetivo de guiar a equipe e facilitar o entendimento comum entre todos os envolvidos (SUTHERLAND; SCHWABER, 2013).

3.1.3.1 *Backlog* de produto

É uma lista com todas as atividades que precisam ser feitas para que o produto final seja concluído. Essa lista evolui junto com o projeto, uma característica do modelo de desenvolvimento iterativo e incremental.

3.1.3.2 Backlog de *sprint*

Assim como o *backlog* de produto, porém com um escopo menor. O *backlog* de *sprint* lista todas as atividades que foram planejadas para uma determinada *sprint*.

3.2 Testes de Software

O desenvolvimento de software é uma atividade que envolve várias atividades e, muitas vezes, diferentes pessoas, o que acaba por favorecer a inserção de defeitos no produto. Além disso, o mal entendimento dos requisitos pode resultar na produção de algo que não foi solicitado pelo cliente (TRODO, 2009).

Os testes de software representam, então, uma atividade que compõe o processo de verificação de validação do software e consiste em uma análise dinâmica do código-fonte, ou seja, o código deve ser executado para que os testes possam ocorrer. Isso possibilita a identificação de erros, bem como a prevenção de inserção de novos erros, quando os testes são automatizados (BARBOSA et al., 2009).

3.2.1 Classificação dos Testes

Os testes de software podem ser classificados, dentre outras categorias, em testes de caixa-preta e testes de caixa-branca (BARBOSA et al., 2009).

3.2.1.1 Testes de caixa-preta

Esse tipo de teste tem como objetivo avaliar o produto em relação aos requisitos funcionais e não-funcionais. A figura abaixo mostra o funcionamento de um teste de caixa-preta, onde são utilizados determinados dados de entrada no software e os resultados obtidos são comparados com as expectativas para os dados de entrada (BARBOSA et al., 2009).



Figura 7 – Representação do teste de caixa-preta

Referências

- BARBOSA, E. F. et al. Introdução ao teste de software. 2009. Disponível em: <http://www.duguay.com.br/uploads/arquivos/apostilaUSP_Teste_de_Software.pdf>. Citado na página 31.
- BRASIL. *Lei nº 12.527, de 18 de novembro de 2011. Regula o acesso à informações.* 2011. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm>. Acesso em: 4 out 2016. Citado na página 17.
- CÂMARA DOS DEPUTADOS. *Dados Abertos da Câmara dos Deputados.* 2016. Disponível em: <<http://www2.camara.leg.br/transparencia/dados-abertos/perguntas-e-respostas>>. Acesso em: 4 out 2016. Citado na página 17.
- COLE, R. M. Clustering with genetic algorithms. Department of Computer Science, University of Western Australia, Australia, 1998. Citado na página 27.
- DINIZ, V. *Como Conseguir Dados Governamentais Abertos.* Brasília, Brasil, 2010. Citado na página 17.
- ESTIVILL-CASTRO, V. Why so many clustering algorithms — a position paper. *ACM SIGKDD Explorations Newsletter*, 2002. Citado na página 25.
- IMAMURA, C. Y. Pré-processamento para extração de conhecimento de bases textuais. 2001. Citado na página 24.
- JAIN, A.; MURTY, M.; FLYNN, P. Data clustering: A review. In: *ACM Computing Surveys*. [S.l.: s.n.], 1999. v. 31, p. 264–323. Citado na página 25.
- JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, v. 28, p. 11–21, 1972. Citado na página 23.
- LLOYD, S. P. Least square quantization in pcm. *IEEE Transactions on Information Theory*, p. 129–137, 1957. Citado na página 25.
- LOPES, E. D. Utilização do modelo skip-gram para representação distribuída de palavras no projeto media cloud brasil. 2015. Citado na página 24.
- LOVINS, J. B. Development of a stemming algorithm. In: *Mechanical Translation and Computational Linguistics*. [S.l.: s.n.], 1968. Citado na página 23.
- MATSUBARA, E. T. et al. *PreText: uma ferramenta para pré-processamento de textos utilizando a abordagem bag-of-words.* 2003. Citado 2 vezes nas páginas 21 e 23.
- MAZONI, M. V. F. *Dados Abertos para a Democracia na Era Digital.* Brasília, Brasil, 2011. 80 p. Citado na página 17.
- OPEN KNOWLEDGE. About. 2016. Disponível em: <<https://okfn.org/about/>>. Acesso em: 4 out 2016. Citado na página 17.
- PORTER, M. F. An algorithm for suffix stripping. *Program electronic library and information systems*, 1980. Citado na página 24.

RAJARAMAN, A.; ULLMAN, J. D. Data mining. In: *Mining of Massive Datasets*. [S.l.: s.n.], 2011. Citado na página 24.

SALTON, G.; BUCKLEY, C. Term-weighting approaches in automatic text retrieval. 1988. Citado na página 22.

SUTHERLAND, J. V.; SCHWABER, K. Business object design and implementation. *OOPSLA '95 workshop proceedings*, The University of Michigan, p. 118, 1995. Citado na página 29.

SUTHERLAND, J. V.; SCHWABER, K. Guia do scrum um guia definitivo para o scrum: As regras do jogo. 2013. Disponível em: <<http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>>. Citado 2 vezes nas páginas 29 e 30.

TRODO, L. D. Uso de métricas nos testes de software. Universidade Federal do Rio Grande do Sul, 2009. Disponível em: <<http://www.lume.ufrgs.br/bitstream/handle/10183/18574/000730980.pdf>>. Citado na página 31.

Apêndices

APÊNDICE A – Primeiro Apêndice

Texto do primeiro apêndice.

APÊNDICE B – Segundo Apêndice

Texto do segundo apêndice.

Anexos

ANEXO A – Primeiro Anexo

Texto do primeiro anexo.

ANEXO B – Segundo Anexo

Texto do segundo anexo.