

---

# Estudo e extensão da metodologia DAMICORE para tarefas de classificação

***Bruno Kim Medeiros Cesar***

---



SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: \_\_\_\_\_

**Bruno Kim Medeiros Cesar**

## Estudo e extensão da metodologia DAMICORE para tarefas de classificação

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Francisco José Monaco

**USP – São Carlos**  
**Janeiro de 2016**

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi  
e Seção Técnica de Informática, ICMC/USP,  
com os dados fornecidos pelo(a) autor(a)

C421e Cesar, Bruno Kim Medeiros  
Estudo e extensão da metodologia DAMICORE para  
tarefas de classificação / Bruno Kim Medeiros Cesar;  
orientador Francisco José Monaco. - São Carlos - SP,  
2016.  
101 p.

Dissertação (Mestrado - Programa de Pós-Graduação  
em Ciências de Computação e Matemática Computacional)  
- Instituto de Ciências Matemáticas e de Computação,  
Universidade de São Paulo, 2016.

1. Aprendizado de Máquina. 2. DAMICORE.  
3. Classificação. 4. Dissertação. I. Monaco,  
Francisco José, orient. II. Título.

**Bruno Kim Medeiros Cesar**

**Research and extension of the DAMICORE methodology for  
classification tasks**

Master dissertation submitted to the Instituto de  
Ciências Matemáticas e de Computação – ICMC-  
USP, in partial fulfillment of the requirements for the  
degree of the Master Program in Computer Science  
and Computational Mathematics. *EXAMINATION  
BOARD PRESENTATION COPY*

Concentration Area: Computer Science and  
Computational Mathematics

Advisor: Prof. Dr. Francisco José Monaco

**USP – São Carlos**  
**January 2016**



*À Luzia, pela alegria e paciência.*





# AGRADECIMENTOS

---

---

Agradeço aos meus orientadores, Francisco Monaco e Alexandre Delbem, pelas ideias, críticas, paciência e oportunidades concedidas.

Agradeço à CAPES pelo financiamento deste trabalho, e ao ICMC e LaSDPC pela estrutura concedida para o seu desenvolvimento.

Agradeço à Google e aos meus colegas pelo companheirismo e aprendizado, e que nunca deixaram de me perturbar sobre a importância de concluir logo a dissertação

Aos meus amigos Danilo, Isotília, Salomão, Carmona, Roberta, Fer e Ziggy, que tiveram e terão sempre um papel especial na minha vida.

Agradeço aos meus pais e à minha irmã pela presença e apoio constante, e tolerância com esse nerd que só fala de coisa chata.

Finalmente, agradeço a Luzia, meu ponto de luz e felicidade em todos os momentos.



*“If you’re an engineer, you essentially want to be wrong half the time. If you do experiments and you’re always right, then you aren’t getting enough information out of those experiments. You want your experiment to be like the flip of a coin: You have no idea if it is going to come up heads or tails. You want to not know what the results are going to be.” — Peter Norvig*



# RESUMO

MEDEIROS CESAR, B. K.. **Estudo e extensão da metodologia DAMICORE para tarefas de classificação**. 2016. 101 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

A área de aprendizado de máquina adquiriu grande importância na última década graças à sua capacidade de analisar conjuntos de dados complexos em larga escala. Em diversas áreas do conhecimento existe a demanda pela análise de dados por especialistas, seja para obter agrupamentos latentes ou classificar instâncias em classes conhecidas. As ferramentas acessíveis a especialistas leigos em programação são limitadas a problemas específicos e demandam um custo de desenvolvimento às vezes proibitivo, sendo interessante buscar por ferramentas genéricas e aplicáveis a qualquer área do conhecimento. Este trabalho busca estender e implementar uma metodologia genérica de aprendizado de máquina capaz de analisar quaisquer conjuntos de arquivos de forma praticamente livre de configuração. Foram obtidos resultados satisfatórios de sua aplicação em um conjunto amplo de problemas para agrupamento e classificação de executáveis, spam e detecção de línguas.

**Palavras-chave:** Aprendizado de Máquina, DAMICORE, Classificação, Dissertação.



# ABSTRACT

MEDEIROS CESAR, B. K.. **Estudo e extensão da metodologia DAMICORE para tarefas de classificação**. 2016. 101 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Machine learning has risen in importance in the last decade thanks to its power to analyse complex datasets in large scale. At several areas of knowledge there is a demand for data analysis by domain experts, be it for discovering latent clusters or classifying instances into known groups. The tools available for experts that do not master computer programming are limited to specific tasks and demand a high development cost, which sometimes is prohibitive. It is interesting, then, to develop generic tools useful to any area of knowledge. This master's thesis seeks to extend and implement a generic machine learning methodology capable of analysing any set of files mostly free of configuration. Its application produced satisfactory results in a wide set of clustering and classification problems over binary executables, spam classification, and language identification.

**Key-words:** Machine Learning, DAMICORE, Classification, Master's Thesis.





# LISTA DE ILUSTRAÇÕES

---

Figura 1	– Exemplo simplificado da saída do algoritmo LZ77. Sequências repetidas são representadas por uma referência dada por um par ( <i>offset</i> , comprimento). A etapa de compressão exige realizar uma busca reversa pela ocorrência dos bytes seguintes, cuja eficiência pode ser configurada. . . . .	33
Figura 2	– Exemplo de atualização de modelo no compressor PPMd: após ler a <i>string</i> 'bananabandba', o byte 'n' é encontrado, e os contextos para '', 'a' e 'ba' são incrementados em 1 para o byte 'n'. A atualização de fato é mais complexa, pois também considera a inclusão de símbolos de escape para quando um símbolo ainda não presente na tabela é encontrado. . . . .	34
Figura 3	– Ao se usar alternância de blocos como codificação de dois objetos, um compressor que enxerga apenas uma parte da sequência de bits por vez (janela) pode visualizar uma fração igual de cada objeto. . . . .	35
Figura 4	– Exemplo de iteração do NJ . . . . .	38
Figura 5	– Exemplo de um grafo com estrutura de comunidade, com as comunidades circuladas. Arestas em destaque entre as comunidades possuem maiores valor de intermediação, como discutido na seção 2.3.3.1. . . . .	41
Figura 6	– Modularidade de grafos com componentes desconexas. . . . .	45
Figura 7	– Exemplo de limite de resolução da modularidade. Os círculos grandes $K_n$ são grafos completos com $n$ vértices e $n(n-1)/2$ arestas. As linhas pontilhadas são grupos que maximizam a modularidade na configuração dada. Figura adaptada de (FORTUNATO; BARTHELEMY, 2007). . . . .	45
Figura 8	– Exemplo do algoritmo divisivo de Girvan-Newman até a primeira separação de comunidades, sobre a rede do clube de karate de Zachary (ZACHARY, 1977). A intermediação é visualizada em uma escala logarítmica, e a aresta a ser cortada é destacada. . . . .	46
Figura 9	– Exemplos de grafos nulos com a mesma distribuição de graus do grafo original. Um grafo nulo pode ser obtido a partir do original trocando aleatoriamente os extremos de duas arestas simultaneamente, o que preserva a distribuição de graus. . . . .	48
Figura 10	– Classificação dos vértices em uma árvore binária sem raiz. Círculos brancos são folhas, quadrados pretos pertencem à classe $A$ , triângulos pretos pertencem à classe $B$ e círculos pretos à classe $C$ . Nesta árvore tem-se: $n = 9, a = 4, b = 1, c = 2$ . . . . .	49

Figura 11 – Exemplo de partições. Linhas sólidas são verdadeiros positivos, linhas tracejadas são falsos positivos e linhas pontilhadas são falsos negativos. Não são mostrados os verdadeiros negativos. . . . .	55
Figura 12 – Ilustração das entropias de cada partição e conceitos relacionados. . . . .	60
Figura 13 – Grupos de folhas induzidos pelos nós internos . . . . .	64
Figura 14 – Exemplo de problema de classificação. Deve-se designar uma classe para a nova instância (estrela) utilizando a distância às instâncias de dados já classificadas (círculos e quadrados). . . . .	65
Figura 15 – Fronteiras de decisão para alguns valores de $k$ para o classificador $k$ -NN. Regiões onde há um empate apresentam cores intermediárias. Empates não ocorrem com $k$ ímpar pois só existem duas classes para escolher. Note que a classificação da nova instância depende de $k$ e pode até mesmo não ser classificada. . . . .	67
Figura 16 – Classificador Naive Bayes. . . . .	69
Figura 17 – Sequência de classificação, testando se $x \in U$ . . . . .	70
Figura 18 – Possíveis árvores de quarteto rotuladas . . . . .	71
Figura 19 – Treliza mostrando os possíveis resultados com 4 classes. Resultados positivos, negativos e neutros são mostrados por, respectivamente, '+', '-' e 'o'. O <i>score</i> é mostrada abaixo de cada combinação. . . . .	72
Figura 20 – Diagrama da hierarquia de classes de Compressor. . . . .	76
Figura 21 – Diagrama da hierarquia de classes de DataSource. . . . .	76
Figura 22 – Diagrama da hierarquia de classes de DataSourceFactory. . . . .	78
Figura 23 – Distribuição de classes nos conjuntos de dados analisados. Não foi possível incluir a legenda para cada classe dos conjuntos TCL e Wikipedia devido à grande quantidade. . . . .	82
Figura 24 – $NCD(x, x)$ em função do tamanho de bloco para algumas instâncias selecionadas. A coluna da esquerda contém executáveis <i>I/O-intensive</i> e a coluna da direita executáveis <i>CPU-intensive</i> . . . . .	84
Figura 25 – Índices de qualidade de agrupamento, comparando o uso de concatenação e intercalação . . . . .	85
Figura 26 – Distribuição dos tempos de execução para cada combinação de compressor, conjunto de dados e implementação. As estrelas azuis indicam a média de cada distribuição. . . . .	87
Figura 27 – Intercalação não parametrizada, com tamanho de bloco máximo igual a $W/2$ . . . . .	94

# LISTA DE TABELAS

---

Tabela 1	– Codificação de Huffman para alfabetos de maior ordem . . . . .	32
Tabela 2	– Exemplo da transformada de Burrows-Wheeler da <i>string</i> 'bananabandbandana'. Também é necessário retornar o índice da primeira rotação, já que todas as rotações possuem a mesma transformada. . . . .	33
Tabela 3	– Cálculo das definições apresentadas em função da matriz de adjacência $A$ . .	43
Tabela 4	– Congruência dos pares entre partições $U$ e $V$ apresentadas na Figura 11. ● denota verdadeiro positivo, ○ denota verdadeiro negativo, ◐ denota falso negativo e ◑ denota falso positivo. . . . .	55
Tabela 5	– Matriz de contingência do exemplo na Figura 11. . . . .	56
Tabela 6	– Exemplo de diferentes matrizes de confusão com os mesmos valores de precisão ( $P$ ) e sensibilidade ( $R$ ) para uma classe, mas diferentes para a outra ( $\hat{P}$ e $\hat{R}$ ). Note que o conjunto 1 está igualmente dividido entre as classes, e os valores duais $\hat{P}$ e $\hat{R}$ encontram-se na mesma escala que os primais. No conjunto 2, existem muitos mais exemplos de elementos legítimos, e logo as medidas duais são distintas e mais próximas de 1. Intuitivamente, o classificador é mais efetivo no conjunto 2, uma vez que classifica corretamente muito bem os elementos “negativos”. Esta característica é capturada pelo coeficiente de Matthews $M$ . . . . .	62
Tabela 7	– Conjuntos de dados utilizados . . . . .	81
Tabela 8	– Comparação entre as implementações ncd2 completa e CompLearn. Descrição das colunas no texto. . . . .	87
Tabela 9	– Desempenho para classificação de spam. As métricas são apresentadas apenas para os elementos classificados. Para cada combinação de conjunto de dados e classificador, foi destacada a melhor métrica. . . . .	89
Tabela 10	– Desempenho para identificação de línguas. As métricas de distância de informação normalizada e índice de classificação são apresentadas apenas para os elementos classificados. . . . .	90



# SUMÁRIO

---

<b>1</b>	<b>INTRODUÇÃO</b>	<b>21</b>
<b>1.1</b>	<b>Objetivos</b>	<b>22</b>
<b>1.2</b>	<b>Organização</b>	<b>22</b>
<b>2</b>	<b>METODOLOGIA</b>	<b>25</b>
<b>2.1</b>	<b>NCD</b>	<b>26</b>
<b>2.1.1</b>	<i>Complexidade de Kolmogorov</i>	<b>27</b>
<b>2.1.2</b>	<i>Distância de Informação Normalizada</i>	<b>28</b>
<b>2.1.3</b>	<i>Distância de Compressão Normalizada</i>	<b>29</b>
<b>2.1.4</b>	<i>Compressores</i>	<b>30</b>
<b>2.1.4.1</b>	<i>Codificação</i>	<b>31</b>
<b>2.1.4.2</b>	<i>Compressores práticos</i>	<b>32</b>
<b>2.1.4.3</b>	<i>Intercalação como função de pareamento</i>	<b>35</b>
<b>2.2</b>	<b>Simplificação - Neighbor Joining</b>	<b>36</b>
<b>2.2.1</b>	<i>Reconstrução filogenética</i>	<b>36</b>
<b>2.2.2</b>	<i>Filogenética computacional</i>	<b>37</b>
<b>2.3</b>	<b>Deteção de comunidades</b>	<b>40</b>
<b>2.3.1</b>	<i>Definições básicas</i>	<b>42</b>
<b>2.3.2</b>	<i>Modularidade</i>	<b>43</b>
<b>2.3.3</b>	<i>Método de deteção de comunidades</i>	<b>45</b>
<b>2.3.3.1</b>	<i>Método divisivo</i>	<b>45</b>
<b>2.3.3.2</b>	<i>Otimização de modularidade</i>	<b>46</b>
<b>2.3.4</b>	<i>Agrupamento em árvores</i>	<b>47</b>
<b>2.3.4.1</b>	<i>Modelo de árvore conexa</i>	<b>49</b>
<b>2.3.4.2</b>	<i>Correlação de graus</i>	<b>50</b>
<b>3</b>	<b>AGRUPAMENTO E VALIDAÇÃO</b>	<b>53</b>
<b>3.1</b>	<b>Validação externa</b>	<b>53</b>
<b>3.1.1</b>	<i>Índices comuns</i>	<b>54</b>
<b>3.1.2</b>	<i>Correção para o acaso</i>	<b>57</b>
<b>3.1.3</b>	<i>Informação mútua</i>	<b>59</b>
<b>3.1.4</b>	<i>Partições binárias</i>	<b>60</b>
<b>3.2</b>	<b>Teste de confiabilidade</b>	<b>62</b>

3.2.1	<i>Métodos de reamostragem</i>	63
3.2.2	<i>Teste para filogenias</i>	63
3.3	Classificação	64
3.3.1	<i>Classificadores comuns</i>	66
3.3.2	<i>Árvores de quarteto</i>	69
4	DESENVOLVIMENTO	73
4.1	Motivação	73
4.1.1	<i>Desempenho</i>	74
4.1.2	<i>Extensibilidade e manutenibilidade</i>	74
4.2	Arquitetura	75
4.3	Bibliotecas/Ferramentas	78
5	RESULTADOS	81
5.1	Aplicação de intercalação como função de pareamento	83
5.1.1	<i>Distância de instância para si mesmo</i>	83
5.1.2	<i>Impacto do tamanho de blocos sobre agrupamento</i>	84
5.2	Validação da implementação da NCD	85
5.3	Classificação de spam	88
5.4	Identificação de línguas	89
6	CONCLUSÃO	91
6.1	Contribuições	91
6.2	Trabalhos futuros	92
6.2.1	<i>NCD</i>	92
6.2.2	<i>Intercalação</i>	93
6.2.3	<i>Ferramenta DAMICORE</i>	94
6.2.4	<i>Classificador por árvores de quarteto</i>	95
	REFERÊNCIAS	97

---

# INTRODUÇÃO

---

A tarefa do analista ou cientista de dados não é simples: mais do que aplicar ferramentas estatísticas para obter conhecimento e tomar decisões a partir de dados, suas tarefas também em geral incluem obter, limpar e extrair os parâmetros relevantes de um conjunto de dados antes de conseguir aplicar as referidas técnicas; uma vez obtido um modelo apropriado, também deve apresentar e visualizar os padrões encontrados.

Apesar de tais tarefas também estarem presentes no arcabouço da estatística, a crescente disponibilidade de dados advindos de ferramentas computacionais e, em geral, a falta de controle sobre sua coleta, implicam uma importância maior para estas etapas. Assim, é esperado que um analista tenha tanto a compreensão do domínio, conhecimento estatístico e capacidade de programação para obter um resultado confiável e significativo a partir de uma análise. Ferramentas de análise de dados direcionadas a especialistas de domínio leigos em programação, como médicos, biólogos e cientistas sociais, têm desenvolvimento caro e são restritas a poucas tarefas.

Neste cenário, uma ferramenta genérica capaz de realizar tarefas simples de agrupamento e classificação em quaisquer tipos de dados teria um grande impacto, pois viabiliza a análise de dados em larga escala (*Big Data*) a um grande contingente de especialistas que, de outra maneira, não teriam acesso às técnicas poderosas desenvolvidas nas últimas décadas. Uma ferramenta universal pode não ser tão eficaz como técnicas dirigidas a um domínio específico, mas ainda pode ter valor caso seja boa o suficiente.

A ferramenta proposta neste trabalho, por se basear em uma técnica de análise de dados binários, é apropriada como uma primeira etapa exploratória de um conjunto de dados. Um analista inexperiente pode ter uma boa noção da organização de um conjunto de dados; um analista capacitado pode, de forma iterativa, limpar e extrair as características relevantes do conjunto ao se utilizar de uma ferramenta simples e “livre de configuração” - isto é, em que as configurações iniciais serão suficientes para a maioria das atividades.

Tal ferramenta implementa a metodologia DAMICORE apresentada em [Sanches, Car-](#)

doso e Delbem (2011), mas foi desenvolvida de forma que seja extensível para acomodar quaisquer métodos que se enquadrem em um fluxo composto de métricas, simplificação, agrupamento e validação, com uma estrutura modular que permite substituir quaisquer de suas componentes. A técnica foi aplicada com sucesso para análise de filogenias, classificação de binários executáveis e navegação de robôs (MANSUR, 2013; PINTO, 2013).

Originalmente, a metodologia DAMICORE era aplicável somente a agrupamento de conjuntos de dados de modo não-supervisionado. Para conjuntos de dados grandes, da ordem de milhares de instâncias, a técnica exige muito tempo de processamento e torna-se inviável para computadores pessoais. Como parte deste trabalho, além de aprimorar o desempenho de agrupamento, foi realizada a extensão da técnica para tarefas de classificação, tornando-se então aplicável a conjuntos de dados maiores e atendendo a uma demanda diferente em aprendizado de máquina.

## 1.1 Objetivos

Criar uma ferramenta computacional de fácil utilização para agrupamento e classificação de conjuntos de dados binários, implementando a metodologia DAMICORE. Como parte deste trabalho é necessário:

- Estudar em detalhes a técnica e suas etapas;
- Implementar corretamente a técnica;
- Realizar a validação da implementação e dos seus resultados em comparação com outras técnicas de aprendizado de máquina.
- Permitir a utilização de diferentes ferramentas em cada uma das etapas;
- Ser extensível e combinável com outras ferramentas, preferindo métodos padrão de entrada/saída de dados.
- Ser acessível a especialistas leigos em computação.

## 1.2 Organização

Esta dissertação está organizada da seguinte forma:

- O capítulo 2 apresenta o estudo da metodologia DAMICORE, com a revisão bibliográfica das técnicas empregadas: NCD, filogenética computacional e detecção de comunidades em grafos;



- O capítulo 3 apresenta a revisão de técnicas de validação em aprendizado de máquina, utilizadas para testar a qualidade de agrupamento e classificação obtidos;
- O capítulo 4 apresenta o desenvolvimento da ferramenta computacional, sua arquitetura e principais contribuições;
- O capítulo 5 apresenta experimentos realizados com a ferramenta, tanto para testar seu desempenho computacional, quanto na aplicação em conjuntos de dados diversificados;
- Por fim, o capítulo 6 realiza a avaliação do desenvolvimento, apresentando as contribuições do trabalho, aplicações esperadas e trabalhos que podem ser desenvolvidos sobre a ferramenta e técnicas desenvolvidas.



---

## METODOLOGIA

---

Este capítulo expõe o embasamento teórico para a técnica DAMICORE para agrupamento de um conjunto de dados (SANCHES; CARDOSO; DELBEM, 2011). Tal técnica consiste em uma composição de ferramentas, inicialmente escolhidas pela compatibilidade de entradas e saídas.

1. Cálculo de uma matriz de distâncias entre as instâncias de um conjunto de dados binários, utilizando a métrica NCD com o compressor PPMd;
2. Transformação da matriz de distâncias em uma árvore binária sem raiz contendo as instâncias como folhas, onde os comprimentos das arestas indicam o grau de similaridade entre as instâncias;
3. Agrupamento hierárquico das instâncias de forma a obter uma partição do conjunto de dados original, obtida com a técnica de detecção de comunidades *Fast Newman*.

De fato, a técnica pode ser compreendida como um *framework* para quaisquer técnicas que sigam um fluxo de métrica-simplificação-agrupamento que possuam entradas e saídas compatíveis. Mansour utiliza métricas para comparação de genomas para o primeiro passo (MANSOUR, 2013); Soares adiciona uma etapa de combinação de árvores anterior ao agrupamento (SOARES, 2013); e Pinto utiliza três diferentes conjuntos de dados advindos da mesma fonte simultaneamente para obter uma classificação (PINTO, 2013).

O embasamento teórico de cada ferramenta é estudado e explorado nas seções deste capítulo:

- A seção 2.1 estuda a métrica NCD e sua base na Teoria da Informação, incluindo a exposição de algoritmos de compressão e suas implementações em compressores práticos, utilizadas e disponíveis na ferramenta computacional desenvolvida.

- A seção 2.2 estuda técnicas de simplificação para obtenção de uma árvore binária a partir de uma matriz de distâncias. O algoritmo de Junção de Vizinhos advindo para reconstrução filogenética é apresentado, assim como algumas outras alternativas derivadas da Filogenética e da área de Redes Complexas.
- A seção 2.3 expõe conceitos básicos sobre a área de Redes Complexas para a compreensão de técnicas de detecção de comunidades, incluindo o estudo da medida de modularidade de grafos e as técnicas mais populares, como *Fast Newman*.

## 2.1 NCD

Na área de Aprendizado de Máquina, diversos métodos dependem da definição de uma **métrica** para fornecer a distância entre instâncias de dados. A escolha da métrica é um dos passos mais sensíveis ao desenvolver um algoritmo efetivo, dado que ela define como os elementos são similares/dissimilares entre si e irá impactar em como elementos são considerados “próximos” ou “distantes” ao obter agrupamentos e/ou classificar instâncias.

Uma métrica  $D(x, y)$  é uma função que satisfaz as seguintes propriedades:

$$\begin{aligned}
 D(x, x) &= 0 && \text{coincidência} \\
 D(x, y) &= D(y, x) && \text{simetria} \\
 D(x, y) &> 0, x \neq y && \text{positividade} \\
 D(x, z) &\leq D(x, y) + D(y, z) && \text{desigualdade triangular}
 \end{aligned}$$

A escolha das métricas depende do tipo de atributos das instâncias de dados sob estudo. Alguns tipos de dados possuem métricas muito descritivas já definidas: vetores em  $\mathbb{R}^n$  (com  $n$  pequeno) podem utilizar a distância Euclidiana; cadeias de caracteres curtas podem ser comparadas com a distância de Levenshtein (ou distância de edição); e vetores de bits podem utilizar a distância de Hamming. Contudo, tipos de dados mais complexos, como linguagem natural ou imagens, podem exigir uma etapa de processamento para **extração de características**, onde as propriedades relevantes são obtidas e compõem um **vetor de características**. Tais vetores pode então ser comparados com uma métrica simples. Linguagem natural, por exemplo, pode exigir uma sequência de procedimentos como remoção de pontuação, eliminação de *stop words* e radiciação até obter uma tabela de frequência de palavras que caracterize o texto dado.

Dado um conjunto de dados  $D = \{d_1, d_2, \dots, d_n\}$ , a DAMICORE utiliza uma métrica para calcular a **matriz de distância** entre todas as instâncias.

O núcleo da DAMICORE depende da Distância de Compressão Normalizada (NCD, da sigla em inglês para *Normalized Compression Distance*) (LI *et al.*, 2004). Esta métrica é única

por não depender de uma representação estruturada dos dados, tal como tuplas ou pares de chave-valor, mas apenas em sua codificação binária. Esta sequência de bits, ou simplesmente **string**, é fornecida a um compressor de arquivos genérico para obter uma medida do seu conteúdo de informação. De fato, a NCD é uma família de métricas parametrizada pelo compressor escolhido  $Z$ , e depende essencialmente da sua capacidade em detectar similaridades que produzam uma distância significativa. Esta métrica é adequada para uma ampla variedade de tipos de dado com compressores eficientes já desenvolvidos, como linguagem natural, executáveis binários, sequências de genoma e proteoma, imagens de bitmap e mais (CILIBRASI; VITÁNYI, 2005).

As seções a seguir apresentam a teoria por trás da NCD, assim como alguns compressores que foram utilizados ao longo desta pesquisa.

### 2.1.1 Complexidade de Kolmogorov

A **complexidade de Kolmogorov**  $K(x)$  de um objeto  $x$ , representado como uma *string*, é o comprimento da menor descrição deste objeto em uma linguagem universal fixada  $L$ , como uma linguagem de programação (C, Lisp, etc.) ou uma máquina de Turing universal (BENNETT *et al.*, 1998). Por exemplo, a expressão em Python `'ha'*5` descreve a string `hahahahaha`, e é possivelmente sua menor descrição com 6 bytes.

A escolha da linguagem importa apenas por um termo constante independente de  $x$ . Pela hipótese de Church-Turing, todas as linguagens Turing-completas são equivalentes em poder e podem simular umas às outras (MING; VITÁNYI, 1997). Se o menor programa na linguagem  $L_1$  que computa  $x$  possui comprimento  $z$ , é possível obter um programa curto em outra linguagem  $L_2$  escrevendo um interpretador de  $L_1$  de comprimento  $I$ , e então computar  $x$  executando o interpretador com o programa original. O comprimento do programa mais curto em  $L_2$  que computa  $x$  é então limitado superiormente por  $I + z$ , onde  $I$  é independente de  $x$ .

Kolmogorov propôs esta medida como uma definição de aleatoriedade que não dependesse em assumir uma distribuição de probabilidade para todas as *strings* possíveis. Posto de forma simples, uma *string*  $x$  é dita **aleatória** se não existe nenhum programa que compute  $x$  mais curto que a própria  $x$ , isto é,  $K(x) \geq |x|$ . Tal programa não seria muito maior que a própria *string*, com  $K(x) = |x| + c$ , onde  $c \geq 0$  é uma constante independente de  $x$  (MING; VITÁNYI, 1997). Por exemplo, a *string* `engzpoqalj` é representada pela expressão em Python `'engzpoqalj'` com apenas 2 bytes extras para as aspas, e é provavelmente sua menor descrição nesta linguagem.

Podemos também definir a complexidade de se transformar uma *string* em outra pela **complexidade condicional**  $K(x|y)$ , que é o comprimento do menor programa que emite  $x$  dado  $y$  como entrada. Assim,  $K(x) = K(x|\varepsilon)$ , onde  $\varepsilon$  é a *string* vazia. É intuitivo que  $K(x|x) = \mathcal{O}(1)$ . A **complexidade conjunta**  $K(x, y)$  é o comprimento do menor programa que emite ambos  $x$  e  $y$ : “emite ambos” significa “emite uma *string* que pode ser decodificada de forma inambígua em cada uma de suas *strings* componentes”. Tal codificação pode ser obtida por uma função de

pareamento  $\langle x, y \rangle$  com inversas  $\langle \cdot \rangle_1$  e  $\langle \cdot \rangle_2$ , tal que

$$\langle \langle x, y \rangle \rangle_1 = x$$

$$\langle \langle x, y \rangle \rangle_2 = y$$

Para uma dada função de pareamento fixada  $\langle \cdot, \cdot \rangle$ , a complexidade conjunta é então definida em termos da complexidade de Kolmogorov como  $K(x, y) = K(\langle x, y \rangle)$ . Um possível pareamento é  $\langle x, y \rangle = l(x)xy$ , onde  $l(x)$  é uma codificação livre de prefixo do comprimento de  $x$ , e possui comprimento  $\mathcal{O}(\log |x|)$ . As inversas podem ser obtidas facilmente pela leitura de  $l(x)$  e então: 1) ler este número de bits para obter  $x$ ; ou 2) ignorar este número de bits e ler até o fim da *string* para obter  $y$ .

Um resultado muito útil fornece uma relação entre as complexidades condicional e conjunta de um par de *strings* (BENNETT *et al.*, 1998):

$$\begin{aligned} K(x, y) &= K(x) + K(y | \langle x, K(x) \rangle) + \mathcal{O}(1) \\ &= K(x) + K(y | x) + \mathcal{O}(\log(K(x, y))) \\ &= K(y) + K(x | y) + \mathcal{O}(\log(K(x, y))) \end{aligned}$$

Esta relação pode ser interpretada como segue: o menor programa que emite ambos  $x$  e  $y$  pode ser construído pela união do menor programa que emite  $x$  (com comprimento  $K(x)$ ) e o menor programa que transforma  $x$  em  $y$  (de comprimento  $K(y | x)$ ). Este programa possui comprimento dentro de um termo logarítmico de  $K(x) + K(y | x)$ . É claro,  $x$  e  $y$  podem ser trocados na interpretação acima.

De agora em diante, iremos abusar da notação de igualdade para ignorar os termos de erro, escrevendo então  $K(x, y) = K(x) + K(y | x)$  e  $K(x | x) = 0$ .

### 2.1.2 Distância de Informação Normalizada

Podemos utilizar a complexidade de Kolmogorov para obter uma métrica universal entre objetos quaisquer, desde que eles sejam representados de forma significativa como *strings* binárias finitas. A complexidade condicional  $K(x | y)$  já fornece uma medida da similaridades entre dois objetos: mesmo *strings* aleatórias podem ser similares o suficiente tal que um programa curto possa transformar uma na outra. Contudo, ela não é uma métrica pois, em geral,  $K(x | y) \neq K(y | x)$ , não satisfazendo a propriedade de simetria.

Definimos a **distância de informação**  $E(x, y)$  como o comprimento do menor programa que pode transformar entre  $x$  e  $y$ . Um resultado de Bennett *et al.* (1998) é que, dentro de um termo logarítmico, o comprimento do programa pode ser dado pela Equação 2.1:

$$\begin{aligned}
E(x,y) &= \max\{K(y|x), K(x|y)\} \\
&= \max\{K(x,y) - K(x), K(x,y) - K(y)\} \\
&= K(x,y) - \min\{K(x), K(y)\}
\end{aligned} \tag{2.1}$$

A distância de informação é absoluta, no sentido de que possui uma unidade em bits (comprimento do programa). Para detecção de similaridades, contudo, é mais desejável uma distância relativa ao comprimento (ou complexidade) das *strings* comparadas. Se existe um programa de 100 bytes que converte entre duas *strings* aleatórias de 10,000 bytes, estas são mais similares do que duas *strings* aleatórias de 1,000 bytes separadas pela mesma distância. A **distância de informação normalizada** (NID, da sigla em inglês para *Normalized Information Distance*), dada na Equação 2.2, também chamada **métrica de similaridade**, fornece esta distinção (LI *et al.*, 2004).

$$\begin{aligned}
\text{NID}(x,y) &= \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}} \\
&= \frac{K(x,y) - \min\{K(x), K(y)\}}{\max\{K(x), K(y)\}}
\end{aligned} \tag{2.2}$$

Esta métrica está limitada ao intervalo  $[0, 1]$  (daí, normalizada), onde  $\text{NID}(x,x) = 0$  e  $\text{NID}(x,y) = 1 \Leftrightarrow K(x,y) = K(x) + K(y)$ . O segundo caso ocorre se  $K(y|x) = K(y)$  e  $K(x|y) = K(x)$ , isto é, não existe nenhuma informação algorítmica mútua entre elas tal que prover uma de entrada para um programa que emite a outra é equivalente a prover a *string* vazia  $\varepsilon$ .

### 2.1.3 Distância de Compressão Normalizada

A complexidade de Kolmogorov é, infelizmente, incomputável. Suponha que exista um programa  $C(x)$  de tamanho  $M$  que emite a complexidade de Kolmogorov de qualquer *string*  $x$ . Podemos escrever um programa curto de tamanho  $N$  que implementa o seguinte algoritmo:

---

```

1 Lista ordenada lexicograficamente de todas as strings binárias
2  $B_\infty \leftarrow \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$ 
3 for  $x \in B_\infty$  do
4   | if  $C(x) > M + N$  then
5   |   | return  $x$ 

```

---

Este algoritmo com certeza termina, já que sempre existe uma *string* com complexidade de Kolmogorov maior do que qualquer número dado<sup>1</sup> (especificamente,  $M + N$ ). Contudo, nós acabamos de descrever um programa que computa uma *string* com complexidade maior do que

o comprimento do próprio programa. Isto é uma contradição, e portanto nossa hipótese inicial de que existe tal programa para computar a complexidade de Kolmogorov deve ser falsa.

A busca por um programa  $z$  que produza uma dada *string*  $x$ , de tal modo que  $|z| < |x|$ , está fadada ao fracasso para *strings* arbitrárias, não apenas pela maioria das *strings* serem aleatórias como pela tarefa ser incomputável. Contudo, a maior parte das *strings* que nos interessam possuem significados que as colocam longe da aleatoriedade: texto, imagens, vídeo e música são criadas por processos que podem ser compreendidos e modelados, que podem então ser utilizados para escrever programas que emitem a representação condensada dos dados. Esta é a arte da **compressão de dados**.

Seja  $B_\infty$  o conjunto de todas as *strings* binárias, e  $P \subset B_\infty$  o conjunto de todos os programas possíveis em uma linguagem fixada  $L$ . Um compressor sem perdas  $Z : B_\infty \mapsto P$  é uma função bijetiva que emite o programa  $z$  em  $L$  que descreve  $x$ ; o descompressor  $Z^{-1} : P \mapsto B_\infty$  é simplesmente um interpretador desta linguagem. Note que compressores práticos não utilizam uma linguagem universal, uma vez que isto poderia resultar em não-terminação ao descomprimir *strings* arbitrárias.

Na prática, deseja-se que o compressor seja capaz de emitir uma descrição mais curta do que a própria *string*, isto é,  $|Z(x)| < |x|$ . O comprimento da saída oferece um limitante superior computável para a complexidade de Kolmogorov de uma *string*. Então, define-se a **distância de compressão normalizada** ( $NCD_Z$ , da sigla em inglês para *Normalized Compression Distance*) substituindo  $K(x)$  por  $|Z(x)|$  na Equação 2.2.

$$NCD_Z(x, y) = \frac{|Z(\langle x, y \rangle)| - \min\{|Z(x)|, |Z(y)|\}}{\max\{|Z(x)|, |Z(y)|\}} \quad (2.3)$$

A qualidade desta aproximação depende fortemente da taxa de compressão do compressor, que por sua vez depende do tipo de dados em que o compressor é efetivo. Isto também depende da função de pareamento empregada para “unir” duas instâncias de dados distintas, apesar de em geral uma simples concatenação ser suficiente.

#### 2.1.4 Compressores

Esta seção estuda os diferentes compressores utilizados neste trabalho. O compressor escolhido pode ser o passo mais sensível ao se utilizar a NCD, uma vez que ele é o responsável por detectar padrões que possam ser descritos de forma compacta. Todos os compressores utilizados são sem perdas, para melhor aproximar a formulação teórica da NID; contudo, compressores com perdas, como os utilizados para codificação MP3 ou JPEG, podem também ser utilizados com sucesso (QUISPE-AYALA; ASALDE-ALVAREZ; ROMAN-GONZALEZ, 2010).

<sup>1</sup> De outro modo, se existisse uma complexidade de Kolmogorov máxima  $K$ , existiriam no máximo  $2^{(K+1)} - 1$  programas possíveis para gerar uma infinidade de *strings*.



### 2.1.4.1 Codificação

Seja  $S$  o conjunto dos símbolos (ou **alfabeto**) com os quais podemos construir uma mensagem. Como desejamos codificar esta mensagem em binário, precisamos mapear cada símbolo para uma sequência de bits. Sem informações adicionais sobre a natureza da mensagem, podemos escolher um código de tamanho fixo grande o suficiente para codificar todos os símbolos - por exemplo, o código ASCII atribui todas as letras maiúsculas e minúsculas em inglês, números e pontuação para palavras de 7 bits. Com o conhecimento da distribuição de probabilidades  $P(S)$  de todos os símbolos  $s \in S$ , podemos usar um código de tamanho variável para atribuir palavras curtas aos símbolos mais frequentes, e palavras longas para símbolos infrequentes.

Seja  $S_n$  o alfabeto criado ao combinar  $n$  símbolos de  $S$  em uma palavra, isto é:

$$S_n = \begin{cases} S & \text{se } n = 1 \\ S \times S_{n-1} & \text{se } n > 1 \end{cases}$$

Logo  $B = \{0, 1\}$ ,  $B_2 = \{00, 01, 10, 11\}$ ,  $B_3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$  e  $B_\infty$  é o conjunto de todas as possíveis mensagens binárias. A **entropia**  $H_n$  de uma mensagem é o número ótimo de bits por símbolo que pode ser alcançado por qualquer código utilizando estes símbolos, dada pela Equação 2.4 (AUGERI *et al.*, 2007).

$$H_n = -\frac{1}{n} \sum_{s \in S_n} P(s) \log_2 P(s) \quad (2.4)$$

A distribuição de probabilidades pode ser obtida com antecedência, mas a maioria dos compressores a obtém da própria mensagem, analisando a frequência relativa dos símbolos. A entropia verdadeira,  $H = H_\infty$ , corresponde a um modelo com conhecimento perfeito de todas as mensagens possíveis, e a compressão máxima alcançável pelo codificador.

A **codificação de Huffman** é um mapeamento ótimo de símbolos para palavras de código, apesar de restrita a utilizar pelo menos 1 bit por símbolo. Isto se mostra sub-ótimo para mensagens onde a frequência de um dado símbolo é muito desigual: por exemplo, se temos um alfabeto  $\Gamma = \{a, b\}$  com 90% de frequência para  $a$  e 10% para  $b$ , precisamos de pelo menos 1 bit/símbolo com a codificação de Huffman, embora a entropia de tais mensagens seja de 0.469 bits/símbolo. Esta limitação pode ser minimizada ao utilizar combinações de símbolos para modelar alfabetos de ordem mais alta, como mostrado na Tabela 1.

A **codificação de sequências** (RLE, do inglês para *Run-Length Encoding* codifica sequências longas de símbolos repetidos em um par símbolo-comprimento. Por exemplo, a *string* 'aaaaabbababba' seria codificada como '(a,5)(b,2)aba(b,2)a'. Para isso é necessário incluir símbolos adicionais para codificar um par, o que pode acarretar em um incremento no tamanho da mensagem. A ocorrência de frequências é comum em alguns tipos de dados, como figuras *bitmap*,

Tabela 1 – Codificação de Huffman para alfabetos de maior ordem

	$\Gamma$	$\Gamma_2$	$\Gamma_3$
Código	$\begin{cases} a : 0 \\ b : 1 \end{cases}$	$\begin{cases} aa : 0 \\ ab : 10 \\ ba : 110 \\ bb : 111 \end{cases}$	$\begin{cases} aaa : 0 \\ aab : 100 \\ aba : 101 \\ aab : 110 \\ abb : 11100 \\ bab : 11101 \\ bba : 11110 \\ bbb : 11111 \end{cases}$
Bits/símbolo	1	1.29	1.598
Bits/letra	1	0.645	0.533

apesar de sua principal utilização ser como um passo intermediário em outros compressores, como bzip2.

A **codificação mover-para-frente** é uma simples transformação intermediária que melhora a taxa de compressão da codificação RLE. Esta codificação mantém os símbolos em uma tabela e emite seus índices ao invés do símbolo em si; após um símbolo ser utilizado, ele é movido para a frente da tabela. Deste modo, toda repetição consiste em uma longa lista de zeros e símbolos frequentes recebem índices pequenos. Isto torna a distribuição de probabilidades mais enviesada para números pequenos, favorecendo métodos de codificação de entropia como a codificação de Huffman.

A **codificação aritmética** é uma abordagem diferente para a codificação de entropia, buscando codificar a “posição” de uma *string* na linha dos números reais: cada *string* recebe um intervalo entre 0 e 1 proporcional à sua probabilidade de ocorrência, e a codificação consiste em emitir um número deste intervalo com precisão suficiente. *Strings* mais prováveis requerem menor precisão, e portanto são necessários menos bits para codificar o número. Seu principal atrativo é não estar limitada a 1 bit/símbolo como a codificação de Huffman.

#### 2.1.4.2 Compressores práticos

**gzip** é um compressor baseado em dicionários que internamente utiliza a biblioteca **zlib**, que por sua vez é baseada no algoritmo LZ77 (ZIV; LEMPEL, 1977; DEUTSCH; GAILLY, 1996). Ao percorrer uma *string* de bytes, o algoritmo busca por ocorrências prévias dos bytes seguintes. Ao encontrar um padrão repetido, uma referência como “copie os  $n$  bytes ocorridos  $m$  bytes atrás” é emitida; do contrário, os bytes sem compressão são emitidos. A saída está ilustrada na Figura 1. Uma restrição adicional presente na biblioteca **zlib** impõe um limite máximo de 32 KiB para o *offset*, correspondente a uma janela deslizante durante a leitura da *string*. O compressor **gzip** é rápido e amplamente utilizado como parte do protocolo HTTP e para compressão de arquivos de texto.

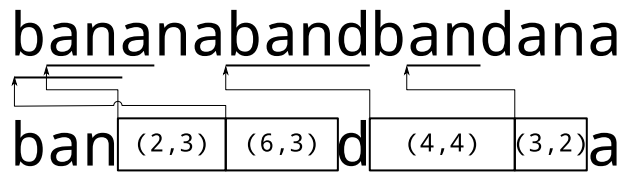


Figura 1 – Exemplo simplificado da saída do algoritmo LZ77. Sequências repetidas são representadas por uma referência dada por um par (*offset*, comprimento). A etapa de compressão exige realizar uma busca reversa pela ocorrência dos bytes seguintes, cuja eficiência pode ser configurada.

**bzip2** é um compressor de ordenação de contexto, construído sobre a transformada de Burrows- Wheeler, e internamente utiliza a biblioteca **bz2** (SEWARD, 1996; BURROWS; WHEELER, 1994). Esta transformada ordena todas as rotações de uma dada *string* de bytes e emite o último byte de cada rotação, ou o byte anterior ao início da rotação, como ilustrado na Tabela 2.

Tabela 2 – Exemplo da transformada de Burrows-Wheeler da *string* 'bananabandbandana'. Também é necessário retornar o índice da primeira rotação, já que todas as rotações possuem a mesma transformada.

bananabandbandana			
Rotações		Ordenação	
0	bananabandbandana	16	abanabandbandana <b>n</b>
1	ananabandbandanab	5	abandbandanabana <b>n</b>
2	nanabandbandanaba	14	anabanabandband <b>d</b>
3	anabandbandanaban	3	anabandbandanaba <b>n</b>
4	nabandbandanabana	1	ananabandbandana <b>b</b>
5	abandbandanabanana	11	andanabanaband <b>b</b>
6	bandbandanabanana	7	andbandanabanana <b>b</b>
7	andbandanabanana <b>b</b>	0	bananabandbandan <b>a</b>
8	ndbandanabanana <b>a</b>	10	bandanabanaband <b>d</b>
9	dbandnabanabana <b>n</b>	6	bandbandanabanana <b>a</b>
10	bandanabanaband <b>d</b>	13	danabanabandband <b>n</b>
11	andanabanaband <b>b</b>	9	dbandnabanabana <b>n</b>
12	ndanabanabandba <b>a</b>	15	nabanabandband <b>a</b>
13	danabanabandband <b>n</b>	4	nabandbandanabana <b>a</b>
14	anabanabandband <b>d</b>	2	nanabandbandanab <b>a</b>
15	nabanabandbanda <b>a</b>	12	ndanabanabandband <b>a</b>
16	abanabandbandan <b>a</b>	8	ndbandnabanabana <b>a</b>
transformada: nndnbbbadannaaaaa, primeiro índice: 16			

Com isso, contextos similares ficam próximos ao serem ordenados, e o byte anterior destes também devem ser similares. Por exemplo, em português, a sequência “que ” é muito comum; dada uma rotação que se inicia com “ue ”, quase sempre ela será precedida pela letra “q”. Tais rotações estariam próximas quando ordenadas, de modo que a transformada conteria uma longa sequência da letra “q”. A *string* transformada pode ser codificada de forma eficiente pelas codificações RLE e mover-para-frente, dado que contém longas sequências de símbolos repetidos. A biblioteca bz2 comprime em blocos, cujos tamanhos podem ser configurados entre 100 KiB a 900 KiB. O compressor bzip2 é apropriado para grandes arquivos e tem bom desempenho

em textos de linguagem natural, possuindo melhor performance para descompressão do que compressão.

O compressor **PPMd** utiliza um modelo de cadeias de Markov dinâmicas para prever a ocorrência de cada byte à medida que percorre a *string* (SHKARIN, 2010). O tamanho máximo do contexto da cadeia pode ser configurado entre 2 e 16 bytes. Um exemplo de atualização dinâmica de um modelo de ordem 2 está ilustrada na Figura 2. O compressor emite a probabilidade de ocorrência de um byte após um dado contexto, que é obtida contando quantas vezes o byte já ocorreu após este contexto. A sequência de probabilidades é então codificada utilizando codificação aritmética. Este compressor é adequado para compressão de textos, e não muito eficiente para dados analógicos digitais, como som e imagens.

bananabandbandana				
Ordem 0	Ordem 1	Ordem 2		
ε	a	ba	na	
a 4	n 3	n 2	n 1	
b 3	b 1	2	b 1	
n 3	4		2	
d 1		ab	db	
11	b	a 1	a 1	
	a 3	1	1	
	3			
	n	an		
	a 2	a 2		
	d 1	d 1		
	3	3		
	d	nd		
	b 1	b 1		
	1	1		

bananabandbandana				
Ordem 0	Ordem 1	Ordem 2		
ε	a	ba	na	
a 4	n 4	n 3	n 1	
n 4	b 1	3	b 1	
b 3	5		2	
d 1		ab	db	
12	b	a 1	a 1	
	a 3	1	1	
	3			
	n	an		
	a 2	a 2		
	d 1	d 1		
	3	3		
	d	nd		
	b 1	b 1		
	1	1		

Figura 2 – Exemplo de atualização de modelo no compressor PPMd: após ler a *string* 'bananabandba', o byte 'n' é encontrado, e os contextos para '', 'a' e 'ba' são incrementados em 1 para o byte 'n'. A atualização de fato é mais complexa, pois também considera a inclusão de símbolos de escape para quando um símbolo ainda não presente na tabela é encontrado.

**PAQ8** é um compressor de mistura de contextos que mantém simultaneamente diversos modelos para prever os bytes seguintes (MAHONEY, 2006). A verdadeira predição é uma combinação de todas as predições ponderadas pelo seu desempenho anterior, e o algoritmo emite a sequência de probabilidades em codificação aritmética como o PPMd. Isto torna o compressor muito bem sucedido em uma ampla gama de tipos de dados, já que ele simultaneamente executa modelos para ilustrações, imagens, texto, binários executáveis e mesmo tipos de arquivo especializados como JPEG. Contudo, com isso ele sofre uma degradação de desempenho considerável, sendo ordens de magnitude mais lento que outros compressores mais práticos.

### 2.1.4.3 Intercalação como função de pareamento

Alguns compressores limitam a quantidade de memória utilizada, trabalhando a cada passo com apenas uma seção da *string* original. A distribuição padrão do compressor gzip limita a busca de referências prévias a uma janela deslizante de aproximadamente 32 KiB, enquanto o compressor bzip2 permite realizar a transformação de Burrows-Wheeler em blocos de no máximo 900 KiB. Isto limita a efetividade da NCD em detecção de similaridades para objetos grandes, como observado por [Cebrian, Alfonseca e Ortega \(2005\)](#).

Nestes casos, ao usar concatenação como função de pareamento, o compressor obtém informação conjunta dos objetos apenas no espaço  $W$  da janela que intercepta ambos, como ilustrado na Figura 3a. Uma vez que a soma do tamanho de ambos seja maior que a janela, alguma seção de alguma das *strings* será comprimida sem possuir contexto de padrões do outro objeto. Outro efeito é que uma *string*  $x$  aleatória com  $\|x\| > W$  possui  $NCD(x, x) \gg 0$ , violando a propriedade de coincidência. Isto indica que mesmo a comparação mais simples, de dois objetos idênticos, pode não produzir resultados significativos.

Para superar esta limitação, este trabalho propõe uma função de pareamento que intercala blocos de de cada *string*, como ilustrado na Figura 3b. Os blocos possuem tamanho fixo  $b$ , exceto pelo último em cada *string*, que pode ser menor. Apenas os primeiros blocos de cada objeto são intercalados; caso uma *string* seja maior que a outra, seus últimos blocos são simplesmente concatenados.

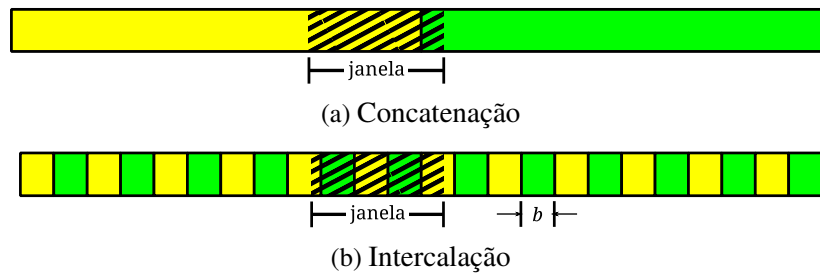


Figura 3 – Ao se usar alternância de blocos como codificação de dois objetos, um compressor que enxerga apenas uma parte da sequência de bits por vez (janela) pode visualizar uma fração igual de cada objeto.

Parece interessante que  $b$  seja um divisor exato de  $W$  para que a janela contenha sempre uma fração igual de ambos os objetos. Um ponto negativo desta estratégia é que a separação em blocos pode remover periodicidades ou separar subsequências frequentes da *string* ao adicionar um espaçamento entre partes do mesmo objeto. Como descrito, a intercalação já seria capaz de garantir a propriedade de coincidência, uma vez que cada janela conteria uma subsequência composta de dois blocos idênticos.

## 2.2 Simplificação - *Neighbor Joining*

A matriz de distâncias pode ser vista como um grafo completo entre os nós, com as distâncias representando uma relação presente nas arestas. O passo de simplificação é utilizado para resumir esta informação completa e obter uma rede complexa que destaca a informação latente nas relações. Existem diversos métodos para isto, sendo os mais comuns:

- **Limiarização:** Remove todas as arestas cuja distância está acima de um limiar  $\theta$ . Este método pode resultar em um grafo desconexo.
- **kNN:** Para cada elemento, mantém as arestas para seus  $k$  vizinhos mais próximos. Este método possui o benefício de sempre fornecer um grafo conexo, embora em geral com uma distribuição de graus exponencial com grau mínimo  $k$ , o que nem sempre caracteriza bem as relações subjacentes.

Existem diversas variantes destes métodos, como ignorar as distâncias para obter um grafo não-direcionado simples, ou utilizar uma função de base radial para remover pesos insignificantes (). A maioria destes métodos requer a seleção de um parâmetro apropriado, como  $\theta$  ou  $k$  acima, que pode ser obtido *ad-hoc* a partir da distribuição de distâncias, ou determinado por um especialista. Neste trabalho, utilizamos um método livre de parâmetros derivado da Filogenética, que constroi uma árvore a partir da matriz de distâncias fornecida.

### 2.2.1 *Reconstrução filogenética*

A Teoria da Evolução, proposta independentemente por Charles Darwin e Alfred Wallace (DARWIN, 1859; DARWIN; WALLACE, 1858), prescreve uma história evolucionária para todas as espécies vivas, como descrito por Amorim (AMORIM, 2002):

Quaisquer duas espécies possuem um ancestral comum. Quaisquer três espécies no present ou surgiram simultaneamente de um ancestral comum, ou duas delas compartilham um ancestral não comum à terceira. Aplicando este pensamento para todas as espécies, obtemos um imagem de uma sequência gigantesca de divisões que fragmentaram de uma única espécie ancestral - ancestral de todos os seres vivos - até as espécies atuais (assumindo que a vida na Terra se originou uma única vez).

Outro conceito chave é o de **caracteres homólogos**, isto é, caracteres encontrados em duas espécies que estavam presentes em seu ancestral comum. Quanto mais recente é um ancestral comum, mais características compartilhadas são homólogas. Isto fornece uma maneira de se comparar duas espécies e prover uma distância entre elas.

A Filogenética se preocupa em reconstruir a árvore evolucionária, isto é, a **filogenia** de um grupo de espécies, dado uma distância entre elas. Tais distâncias podem ser obtidas por uma

variedade de métodos, como comparação de sequências moleculares (por exemplo, alinhamento de sequências de DNA) e comparação de caracteres morfológicos. Como métodos de filogenia apenas requerem uma matriz de distâncias e não em como ela é obtida, eles são adequados para os propósitos do passo de simplificação da DAMICORE.

Um aspecto desafiador em filogenética é que caracteres podem ter se originado múltiplas vezes durante a história evolutiva. Como espécies próximas possuem as mesmas estruturas comuns e estão, na maioria das vezes, sob as mesmas pressões ambientais, é comum que evoluam a mesma adaptação.

Finalmente, simplificar uma matriz de distâncias para uma árvore evolutiva inclui a suposição que os dados podem ser descritos por um processo onde instâncias de dados diferentes são transformações de um ancestral comum. Em alguns casos, a história de uma instância inclui ancestrais diferentes. Por exemplo, a língua inglesa possui um ancestral comum com o Alemão, mas também contém em seu vocabulário diversos termos latinos derivados do Francês (*see and view* possuem o mesmo significado mas origens diferentes). A riqueza das influências cruzadas pode ser perdida ao se simplificar em uma única árvore.

### 2.2.2 Filogenética computacional

O objetivo da filogenética computacional é prover uma árvore evolutiva  $\mathcal{T}$  para um dado conjunto de  $n$  táxons, onde a distância  $M_{ij}$  entre táxons  $i$  e  $j$  é conhecida.  $\mathcal{T}$  é uma árvore ponderada sem raiz, cujos pesos de aresta representam a distância de um táxon para seu ancestral. Ela possui  $n$  folhas, correspondente aos táxons. Cada nó interno representa o ancestral comum mais recente entre seus filhos; se a árvore for binária, existem  $n - 2$  ancestrais.

Para escolher uma filogenia para um conjunto de táxons, precisamos ser capazes de comparar árvores e decidir qual é melhor. O princípio da **máxima parcimônia** determina que dentre duas árvores, é preferível aquela que supõe o menor número de modificações de cada ancestral para seus descendentes. Isto é similar à navalha de Occam, que escolhe a explicação mais simples entre duas igualmente poderosas. Se as mudanças entre ancestrais e descendentes é mensurada pelo comprimento de aresta, este princípio é equivalente a minimizar a soma total dos comprimentos da árvore.

Simplesmente enumerar todas as árvores possíveis e compará-las é inviável exceto para conjuntos muito pequenos de dados, já que o número de árvores com  $n$  folhas cresce super-exponencialmente com  $n$ <sup>2</sup>. Métodos filogenéticos precisam confiar em heurísticas para buscar o espaço de árvores, preferencialmente evitando hipóteses não promissoras. Os algoritmos mais avançados, como MrBayes (RONQUIST; HUELSENBECK, 2003), realizam mutações em uma árvore para explorar o espaço e encontrar árvores que sejam mais parcimoniosas. Estes métodos são muito caros computacionalmente, e precisam de um passo inicial bom para serem bem

sucedidos.

O método de **junção de vizinhos** (NJ, do inglês para *Neighbor Joining*) é um algoritmo que busca minimizar a soma total de comprimentos de aresta com uma abordagem *bottom-up* gulosa (SAITOU; NEI, 1987). Este método encontra a árvore mais parcimoniosa se assumirmos que a métrica entre elementos é aditiva - isto é,  $M_{ij} + M_{jk} = M_{ik}$ .

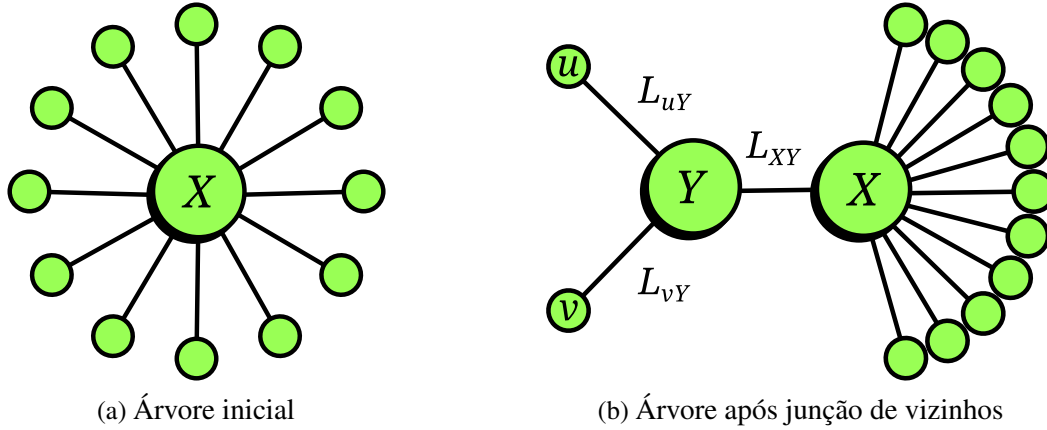


Figura 4 – Exemplo de iteração do NJ

No estado inicial, assumimos que todos os  $n$  táxons possuem um único ancestral comum  $X$ , ilustrado na Figura 4a. Seja  $L_{iX}$  a distância entre os táxons  $i$  e  $X$ , e  $M_{ij} = L_{iX} + L_{jX}$  a distância entre táxons  $i$  e  $j$  dada na matriz de distâncias  $M$  (onde, especialmente,  $M_{ii} = L_{iX} - L_{iX} = 0$ ). A soma total de comprimento de arestas  $S^n$  nesta árvore é dada por

$$S^n = \sum_i L_{iX} \quad (2.5)$$

Somando todas as entradas na matriz de distâncias, deduzimos uma relação entre  $M$  e  $S^n$ :

$$\begin{aligned}
 \sum_{ij} M_{ij} &= \sum_i \sum_{j \neq i} (L_{iX} + L_{jX}) \\
 &= \sum_i \left( \sum_j (L_{iX} + L_{jX}) - 2L_{iX} \right) \\
 &= \sum_{ij} L_{iX} + \sum_{ij} L_{jX} - \sum_i 2L_{iX} \\
 &= 2n \sum_i L_{iX} - 2 \sum_i L_{iX} \\
 &= 2(n-1) \sum_i L_{iX} = 2(n-1)S^n
 \end{aligned} \quad (2.6)$$

Agora, suponha que juntamos os táxons  $u$  e  $v$  sob um ancestral comum mais recente  $Y$ , obtendo a árvore na Figura 4b. Neste caso, a soma total dos comprimentos de aresta  $S_{uv}^{n-1}$  nesta

<sup>2</sup> Especificamente, existem  $(2n-5)!!$  árvores binárias sem raiz com  $n$  folhas, o que é aproximadamente  $0.74 \left(\frac{n-2}{e}\right)^{n-2}$ .



árvore é dada por

$$S_{uv}^{n-1} = \sum_{i \neq u,v} L_{iX} + L_{XY} + (L_{uY} + L_{vY}) \quad (2.7)$$

$$\Rightarrow L_{XY} = S_{uv}^{n-1} - \sum_{i \neq u,v} L_{iX} - (L_{uY} + L_{vY}) \quad (2.8)$$

Desejamos obter uma relação entre  $M$  e  $S_{uv}^{n-1}$ . Considere a soma das entradas nas linhas  $u$  e  $v$ :

$$M_{iu} = L_{iX} + L_{XY} + L_{uY}, i \notin \{u, v\} \quad (2.9)$$

$$M_{iv} = L_{iX} + L_{XY} + L_{vY}, i \notin \{u, v\} \quad (2.10)$$

$$\begin{aligned} M_{iu} + M_{iv} &= 2L_{iX} + 2L_{XY} + (L_{uY} + L_{vY}), i \notin \{u, v\} \\ \sum_{i \neq u,v} (M_{iu} + M_{iv}) &= 2 \sum_{i \neq u,v} L_{iX} + 2(n-2)L_{XY} + (n-2)(L_{uY} + L_{vY}) \end{aligned} \quad (2.11)$$

Substituindo a equação 2.8 na equação 2.11, obtemos:

$$\sum_{i \neq u,v} (M_{iu} + M_{iv}) = 2(n-2)S_{uv}^{n-1} - 2(n-3) \sum_{i \neq u,v} L_{iX} - (n-2)(L_{uY} + L_{vY}) \quad (2.12)$$

Da equação 2.6, obtemos que  $2(n-3) \sum_{i \neq u,v} L_{iX} = \sum_{i,j \neq u,v} M_{ij}$ . Também, da propriedade aditiva,  $L_{uY} + L_{vY} = M_{uv}$ . Substituindo estes na equação 2.12 e rearranjando, obtemos a relação desejada:

$$\begin{aligned} S_{uv}^{n-1} &= \frac{1}{2(n-2)} \sum_{i,j \neq u,v} M_{ij} + \frac{1}{2(n-2)} \sum_{i \neq u,v} (M_{iu} + M_{iv}) + \frac{1}{2} M_{uv} \\ &= \frac{1}{2(n-2)} \sum_{i \neq u,v} \sum_j M_{ij} + \frac{1}{2} M_{uv} \\ &= \frac{1}{2(n-2)} \left( \sum_{ij} M_{ij} - \sum_i M_{iu} - \sum_i M_{iv} \right) + \frac{1}{2} M_{uv} \end{aligned} \quad (2.13)$$

Deseja-se, então, escolher o par  $u-v$  que minimiza a nova soma total das arestas  $S_{uv}^{n-1}$ . Não é necessário calcular esta soma diretamente, pois o problema de otimização  $\min S_{uv}^{n-1}$  é equivalente a

$$\min Q_{uv} = 2(n-2)S_{uv}^{n-1} - \sum_{ij} M_{ij} \quad (2.14)$$

$$= (n-2)M_{uv} - \sum_i M_{iu} - \sum_i M_{iv} \quad (2.15)$$

que requer menos cálculos. Testando todos os pares, encontra-se aquele que minimiza  $Q_{uv}$ , chamados **vizinhos mais próximos**. O método então une de forma gulosa sob um mesmo ancestral, e repete recursivamente o procedimento para a árvore com todos os descendentes de  $X$ , que agora contém  $Y$  mas não  $u$  e  $v$ . O algoritmo pára quando restarem apenas dois nós, que são então conectados diretamente.

Para cada iteração, é necessário atualizar a matriz de distância para conter as distâncias de todos os outros nós a  $Y$ , dadas por

$$M_{iY} = \frac{1}{2}(M_{iu} + M_{iv} - M_{uv}), i \notin \{u, v\} \quad (2.16)$$

Para a construção da árvore, deseja-se os comprimentos das arestas  $uY$  e  $vY$ . Subtraindo as equações 2.9 e 2.10, e da relação  $L_{vY} = L_{uY} - M_{uv}$ , obtém-se o comprimento  $L_{uY}$  como segue:

$$\begin{aligned} M_{iu} - M_{iv} &= L_{uY} - L_{vY}, \forall i \notin \{u, v\} \\ \sum_{i \neq u, v} (M_{iu} - M_{iv}) &= (n-2)(L_{uY} - L_{vY}) \\ \sum_{i \neq u, v} (M_{iu} - M_{iv}) &= (n-2)(2L_{uY} - M_{uv}) \\ L_{uY} &= \frac{1}{2}M_{uv} + \frac{1}{2(n-2)} \sum_{i \neq u, v} (M_{iu} - M_{iv}) \end{aligned} \quad (2.17)$$

Realiza-se uma derivação similar para obter  $L_{vY}$ :

$$L_{vY} = \frac{1}{2}M_{uv} + \frac{1}{2(n-2)} \sum_i (M_{iv} - M_{iu}) \quad (2.18)$$

Mesmo sendo relativamente simples, este algoritmo alcança boas topologias de árvore, com soma total de comprimentos de aresta próxima daquelas encontradas por algoritmos de máxima parcimônia. Ainda, o cálculo de  $Q$  a cada iteração é um problema **embaraçosamente paralelo**, já que cada  $Q_{uv}$  pode ser calculado de forma independente, permitindo implementações muito eficientes.

## 2.3 Detecção de comunidades<sup>3</sup>

**Grafos** ou **redes** são uma representação de relacionamentos entre objetos. Cada objeto é representado por um nó ou vértice e sua relação com outro objeto por uma aresta que os conecta. Apesar de ser uma transformação muito simples e crua, ela pode fornecer informações preciosas sobre a função e estrutura de um sistema, e análises de grafos encontraram aplicações em uma diversidade de áreas como biologia, sociologia, telecomunicações e engenharia de computação.

<sup>3</sup> Esta seção é baseada na excelente introdução da revisão sobre detecção de comunidades de Fortunato (2010).

Ainda mais interessante é a ocorrência de polinização cruzada entre áreas tão distintas, com as técnicas desenvolvidas em uma sendo utilizadas para compreender redes em outra.

Nas últimas décadas, avanços na computação permitiram obter e analisar redes muito maiores que antes, com milhões ou mesmo bilhões de nós. Ainda mais, foi descoberto que redes com origens diferentes compartilham características inesperadas em comum, o que nos leva a crer que existem leis universais sobre como sistemas complexos se organizam. Os desafios que esta mudança impôs criaram uma nova disciplina chamada de **Redes Complexas** ou **Ciência das Redes**, que busca compreender e caracterizar a grande quantidade de redes de mundo real e seus sistemas subjacentes.

Dentre as características em comum, se descobriu que as redes reais estão longe de serem homogêneas, o que seria esperado caso as conexões entre elementos ocorressem tanto de forma aleatória, como de forma estruturada. A distribuição de graus, isto é, a distribuição de arestas entre os vértices, é em geral muito larga, com muitos vértices com poucas arestas e poucos vértices com muitas, muitas arestas. A distribuição de arestas é também localmente não-homogênea, com uma grande concentração de arestas contidas em certos grupos de vértices e menos arestas ocorrendo entre estes grupos. Esta característica é chamada **estrutura de comunidade**, **estrutura modular** ou **agrupamento**. Um exemplo de um grafo com estrutura de comunidade é ilustrado na Figura 5.

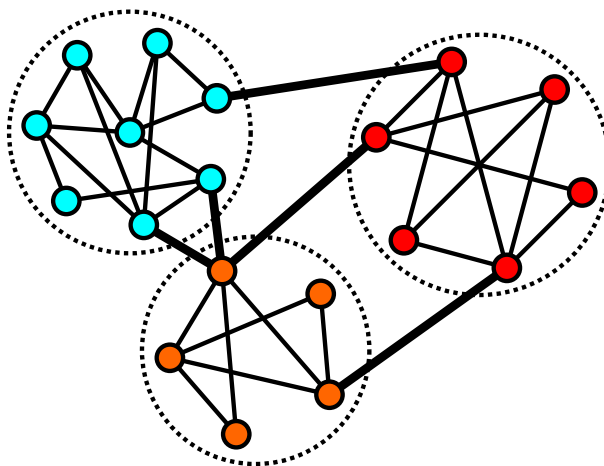


Figura 5 – Exemplo de um grafo com estrutura de comunidade, com as comunidades circuladas. Arestas em destaque entre as comunidades possuem maiores valor de intermediação, como discutido na seção 2.3.3.1.

A ocorrência de comunidades é conhecida e estudada dentro da sociologia por um longo período, e o pequeno tamanho das redes sob estudo muitas vezes exigiam apenas um olho vivo para detectar este tipo de organização. Com redes maiores, contudo, o problema de **detecção de comunidades** (também chamado de **agrupamento de grafos** ou **partição de grafos**) se torna mais difícil, pois o próprio conceito de comunidade não é bem definido. A ocorrência de estruturas hierárquicas, aleatoriedade parcial e sobreposição entre comunidades traz mais desafios para esta área de pesquisa sob desenvolvimento constante.

Diversas redes de mundo real com propriedades modulares possuem uma explicação

subjacente para o fenômeno, como tendências de formação de “panelinhas” em redes sociais ou uma estrutura hierárquica imposta em uma empresa, por exemplo. Nas redes sintéticas geradas neste trabalho, a detecção de comunidades pode ser entendida como um método de **agrupamento de dados** onde as relações de vizinhança são comprimidas e simplificadas em um grafo. Desde que exista um agrupamento real no conjunto de dados subjacente, e o processo de simplificação traduza esta informação fielmente na rede, espera-se que os métodos existentes encontrarão os grupos como comunidades.

### 2.3.1 Definições básicas

Um **grafo** ou **rede**  $G(V, E)$  é uma tupla de  $n$  vértices e  $m$  arestas, onde os vértices  $v \in V$  são objetos do domínio sob estudo e as arestas  $e_{ij} = (v_i, v_j)$  representam relações entre objetos. Quando as arestas não são ordenadas diz-se que o grafo é **não-direcionado**, onde as relações são simétricas. Quando as relações não são simétricas, diz-se que as arestas são ordenadas e o grafo é **direcionado**. Um exemplo do primeiro caso é a Internet, onde as conexões entre roteadores é bidirecional; e um exemplo do segundo é a **World Wide Web**, onde os links entre páginas não são, em geral, recíprocos. Uma representação usual de grafos é como uma **matriz de adjacência**  $A \in \mathbb{B}^{n \times n}$ , onde  $A_{ij} = 1$  se existe uma aresta entre os vértices  $v_i$  e  $v_j$ , e 0 caso contrário.

O **grau**  $k_i$  de um vértice  $v_i$  é o número de arestas que contém este vértice. Em grafos direcionados é necessário distinguir entre o grau de entrada  $k_i^{\text{in}}$  e o grau de saída  $k_i^{\text{out}}$ , respectivamente, o número de arestas entrando ou saindo do vértice. O **grau médio**  $\langle k \rangle$  de um grafo é o número médio de arestas por vértice. Se  $\langle k \rangle = \mathcal{O}(1)$  diz-se que o grafo é **esparso**, enquanto se  $\langle k \rangle = \mathcal{O}(n)$  diz-se que o grafo é **denso**.

Pode-se representar a intensidade de uma relação por um peso  $w_{ij}$  associado a cada aresta  $e_{ij}$ ; neste caso, diz-se que o grafo é **ponderado**. Um grafo não-ponderado é um caso especial onde todos os pesos são unitários. Pode-se estender a definição de grau para considerar o peso das arestas, que é chamado **força**  $s_i$  de um vértice. De modo equivalente, pode-se definir a força de entrada e saída de um vértice em um grafo ponderado direcionado. O cálculo dessas definições é mostrado na tabela 3.

Um **caminho**  $p = (p_1, p_2, \dots, p_l)$  de comprimento  $l$  é uma sequência de vértices conectados por arestas, isto é, tal que as arestas  $e_{p_1, p_2}, e_{p_2, p_3}, \dots, e_{p_{l-1}, p_l}$  existam. Um **caminho geodésico** entre dois vértices é um caminho com o menor comprimento possível. Um **ciclo** é um caminho fechado, onde o primeiro vértice é também o último, e um **ciclo simples** é um ciclo que não possui arestas ou vértices repetidos, exceto seus extremos. Uma **árvore** é um grafo que não possui ciclos simples com comprimento maior que 1. Uma **árvore binária** é uma árvore onde nenhum vértice possui grau maior que 3. Um **grafo completo**  $K_n$  é um grafo onde todas as arestas possíveis entre os  $n$  vértices existem, com  $n(n-1)/2$  arestas em grafos não-direcionados e  $n(n-1)$  arestas em grafos direcionados.

Tabela 3 – Cálculo das definições apresentadas em função da matriz de adjacência  $A$ .

		Não-direcionado	Direcionado
Ponderado	#arestas	$m$	$\frac{1}{2} \sum_{ij} A_{ij}$
	grau	$k_i$	$\sum_j A_{ij}$
	grau de entrada	$k_i^{\text{in}}$	$= k_i$
	grau de saída	$k_i^{\text{out}}$	$= k_i$
	grau médio	$\langle k \rangle$	$m/n$
	força total	$S$	$\frac{1}{2} \sum_{ij} A_{ij} w_{ij}$
	força	$s_i$	$\sum_j A_{ij} w_{ij}$
	força de entrada	$s_i^{\text{in}}$	$= s_i$
	força de saída	$s_i^{\text{out}}$	$= s_i$
	força média	$\langle s \rangle$	$S/n$

### 2.3.2 Modularidade

Não existe uma definição quantitativa amplamente aceita para “o que é uma comunidade”. Se espera que uma comunidade seja um subgrafo conectado com mais arestas dentro do que fora da comunidade, mas diversas definições são compatíveis com esta intuição. Existem até mesmo definições algorítmicas, onde a comunidade é simplesmente aquilo que o método produz sem uma definição *a priori* (FORTUNATO, 2010). Uma definição local busca avaliar cada comunidade separadamente do restante do grafo, enquanto uma definição global avalia toda a partição. Uma classe de definições globais é de que o grafo possui estrutura de comunidade se ele for diferente de um modelo nulo aleatório.

A definição global mais popular é baseada na medida de **modularidade** de um grafo, que compara a densidade de arestas de um subgrafo com a densidade esperada se as arestas se conectassem de modo independente da estrutura de comunidade (NEWMAN; GIRVAN, 2004). Esta medida  $Q$  é definida na equação 2.19 para redes não-direcionadas ponderadas.

$$Q = \frac{1}{2S} \sum_{ij} (A_{ij} w_{ij} - \langle w_{ij} \rangle) \delta(C_i, C_j) \quad (2.19)$$

onde  $\langle w_{ij} \rangle$  é o peso esperado da aresta entre os vértices  $i$  e  $j$  de acordo com o modelo nulo. Para grafos não-ponderados, é apenas a probabilidade  $P_{ij}$  de que estes vértices se conectem. Dado que a única contribuição para a modularidade vem de vértices dentro da mesma comunidade  $C$ , pode-se realizar a soma sobre todas as comunidades como na equação 2.20:

$$\begin{aligned} Q &= \frac{1}{2S} \sum_C \sum_{i \in C} \sum_{j \in C} (A_{ij} w_{ij} - \langle w_{ij} \rangle) \\ &= \frac{1}{2S} \sum_C (2S_C - 2\langle S_C \rangle) \end{aligned} \quad (2.20)$$

onde  $S_C$  é o peso total das arestas totalmente dentro da comunidade  $C$ , e  $\langle S_C \rangle$  é o

peso total esperado de acordo com o modelo nulo. O modelo nulo mais aceito é o **modelo de configuração**, que preserva a distribuição de graus enquanto conecta arestas aleatoriamente (MOLLOY; REED, 1995). O peso esperado é então simplesmente  $\langle w_{ij} \rangle = s_i s_j / 2S$ , reduzido a  $P_{ij} = k_i k_j / 2m$  para grafos não-ponderados. Isto fornece a medida de modularidade usual na equação 2.21:

$$\begin{aligned} Q &= \frac{1}{2S} \sum_{ij} \left( A_{ij} w_{ij} - \frac{s_i s_j}{2S} \right) \delta(C_i, C_j) \\ &= \sum_C \left[ \frac{S_C}{S} - \left( \frac{W_C}{2S} \right)^2 \right] \end{aligned} \quad (2.21)$$

onde  $W_C = \sum_{i \in C} s_i$  é a soma das forças de todos os vértices em  $C$ , o que se reduz a  $d_C = \sum_{i \in C} k_i$  para grafos não-direcionados.

Esta medida está limitada a  $Q \in [-1/2, 1)$ , onde  $Q = 1$  ocorre no limite de um grafo desconexo com infinitas componentes, onde cada componente é uma comunidade (BRANDES *et al.*, 2008). Existem diversos cenários onde o mínimo pode ser alcançado, como um grafo conexo contendo apenas dois vértices onde cada um está em sua própria comunidade. De modo mais interessante,  $Q = 0$  é obtido quando o grafo não é diferente do modelo nulo, ou quando o grafo inteiro é incluído em uma única comunidade.

Esta medida é usada não apenas como uma medida de qualidade de uma partição, mas também como função objetivo a ser otimizada para determinar comunidades, utilizada assim pelos métodos mais populares; e também como uma definição *ad hoc* de estrutura de comunidades - como regra prática, diz-se que uma rede possui estrutura de comunidade se ela possui uma partição com  $Q \geq 0.3$  (NEWMAN, 2004a).

Mesmo tendo ampla utilização, esta métrica não é livre de críticas. Dado que é uma medida global, seu valor depende da forma de todo o grafo, e não apenas da organização de cada comunidade. Por exemplo, qualquer grafo com  $p$  componentes idênticos terá a mesma modularidade de  $Q = 1 - 1/p$ , não importando quão densamente conectada cada componente é (figuras 6a e 6b). Também, a modularidade será menor se uma componente possuir menos arestas, mesmo que seja tão conectada quanto (figura 6c).

Finalmente, a modularidade possui uma escala intrínseca  $l_s \sim \sqrt{m}$  que limita sua resolução para detectar comunidades pequenas. Uma partição contendo comunidades com menos que  $l_s$  arestas possui modularidade menor que uma partição que aglutina tais grupos pequenos (FORTUNATO; BARTHELEMY, 2007). Isto ocorre mesmo se a comunidade for um subgrafo completo com poucas arestas conectando-a ao restante da rede, como mostrado na figura 7.

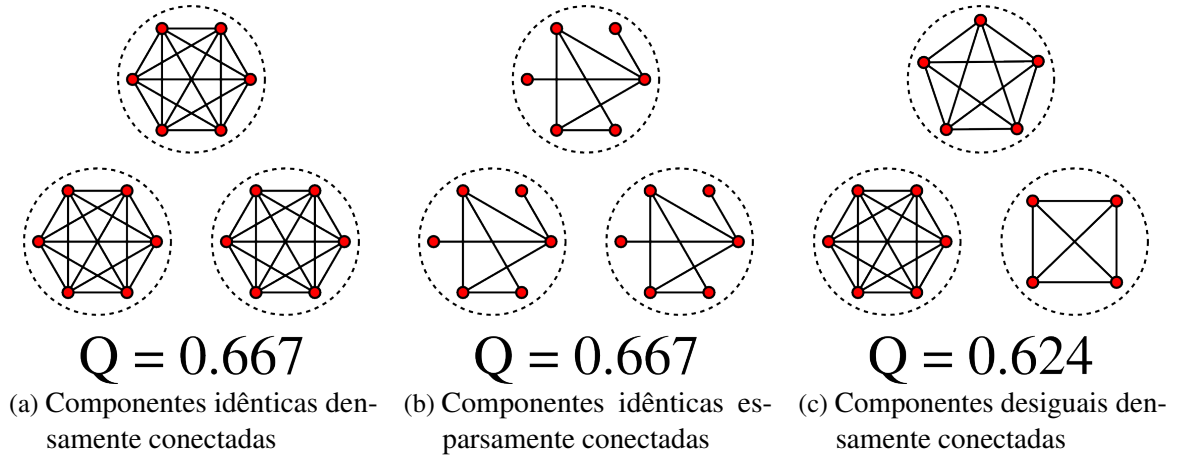
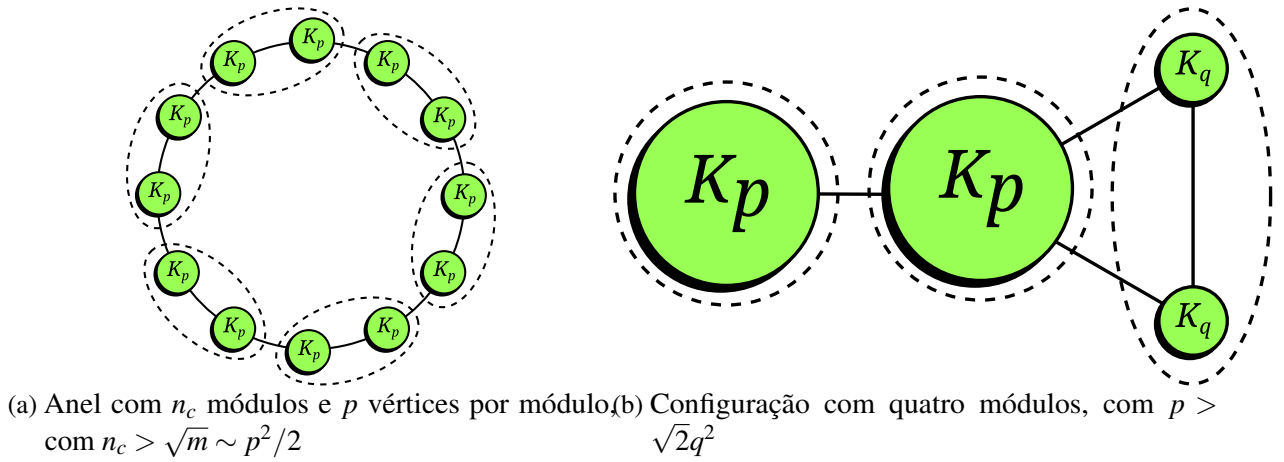


Figura 6 – Modularidade de grafos com componentes desconexas.

Figura 7 – Exemplo de limite de resolução da modularidade. Os círculos grandes  $K_n$  são grafos completos com  $n$  vértices e  $n(n-1)/2$  arestas. As linhas pontilhadas são grupos que maximizam a modularidade na configuração dada. Figura adaptada de (FORTUNATO; BARTHELEMY, 2007).

### 2.3.3 Método de detecção de comunidades

#### 2.3.3.1 Método divisivo

O conceito que uma comunidade possui mais arestas dentro do que fora indica que caminhos geodésicos dentro de uma comunidade são, na média, mais curtos que aqueles que pulam entre comunidades, pois existem menos arestas para serem “utilizadas” para a passagem. Dentre todos os caminhos geodésicos entre todos os pares de vértices, arestas conectando comunidades deveriam possuir mais caminhos passando por elas, como ilustrado na Figura 5. A **intermediação**  $B_e$  de uma aresta mede esta propriedade de estar “no meio do caminho”, contando quantos caminhos geodésicos a utilizam, como dado na equação 2.22:

$$B_e = \sum_{ij} \frac{\text{Número de caminhos geodésicos entre os vértices } i \text{ e } j \text{ que contém } e}{\text{Número de caminhos geodésicos entre os vértices } i \text{ e } j} \quad (2.22)$$

O método divisivo de Girvan-Newman busca identificar as arestas conectando comunidades e removê-las, tornando cada comunidade um componente desconexo (GIRVAN; NEWMAN, 2002). A cada passo, o método encontra a aresta com intermediação máxima e a remove da rede, até que todas as arestas sejam removidas e cada vértice esteja isolado. Este procedimento *top-down* retorna um agrupamento hierárquico, onde em cada nível uma componente é dividida em dois. Os primeiros passos do algoritmo são ilustrados na figura 8.

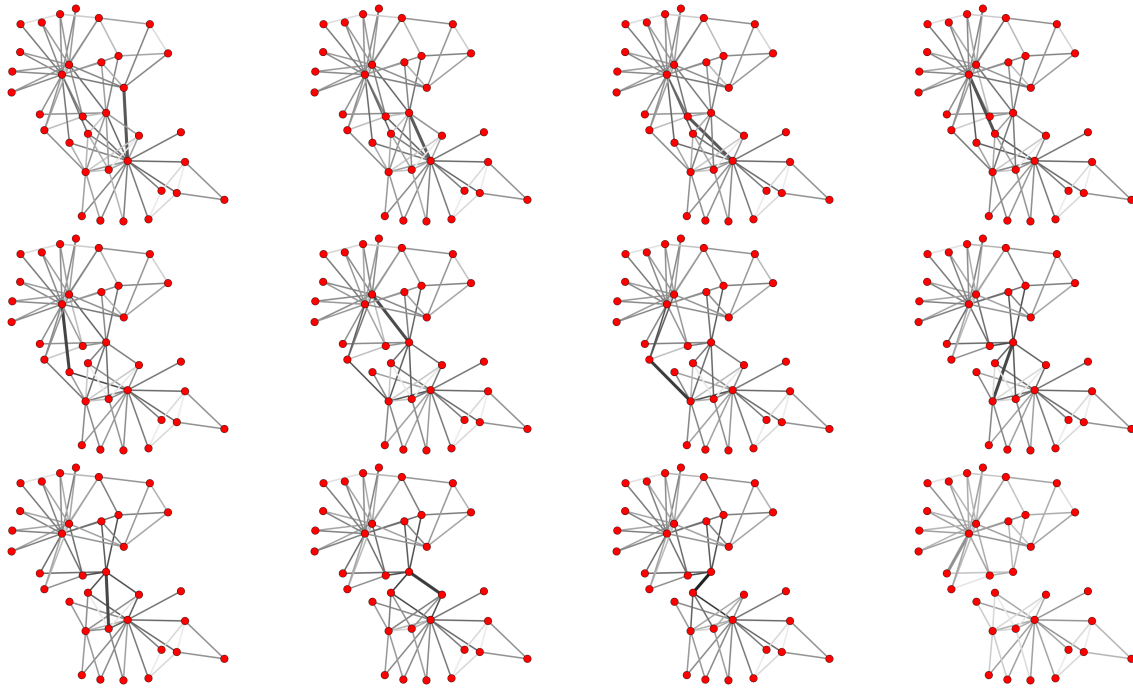


Figura 8 – Exemplo do algoritmo divisivo de Girvan-Newman até a primeira separação de comunidades, sobre a rede do clube de karate de Zachary (ZACHARY, 1977). A intermediação é visualizada em uma escala logarítmica, e a aresta a ser cortada é destacada.

O cálculo de intermediação para todas as arestas em um grafo possui complexidade  $\mathcal{O}(mn)$  usando o método de Brandes (BRANDES, 2001), e o método requer  $m$  cortes de aresta. O pior caso possui complexidade  $\mathcal{O}(m^2n)$ , mas à medida que as comunidades são divididas pode-se analisar cada componente separadamente. Para grafos esparsos, isto se traduz em uma complexidade  $\mathcal{O}(n^3)$ , o que torna o método inviável para grafos muito grandes.

O agrupamento hierárquico obtido pode ser cortado em qualquer nível para obter uma partição para o grafo. É possível determinar o número de grupos a serem retornados, mas o procedimento padrão é utilizar a partição que maximiza a modularidade.

### 2.3.3.2 Otimização de modularidade

Apesar de suas limitações, a modularidade é o padrão *de facto* para avaliar estruturas de comunidade, e a maioria dos algoritmos buscam selecionar uma partição que a maximize. De fato, é possível formular um problema de programação linear inteira para obter o máximo global da modularidade (BRANDES *et al.*, 2008). Seja  $\delta_{ij}$  uma variável de decisão, onde  $\delta_{ij} = 1$  se os



vértices  $i$  e  $j$  estão na mesma comunidade, e 0 caso contrário. Isto é uma relação de equivalência, onde vértices equivalentes pertencem à mesma comunidade. Ao incluir as seguintes restrições, obtém-se a modularidade máxima:

$$\begin{aligned}
 \max \quad & Q = \frac{1}{2m} \sum_{(i,j) \in E} \left( w_{ij} - \frac{s_i s_j}{2m} \right) \delta_{ij} \\
 \text{s.t.} \quad & \forall i : \quad \delta_{ii} = 1 \quad \text{reflexividade} \\
 & \forall i > j : \quad \delta_{ij} = \delta_{ji} \quad \text{simetria} \\
 & \forall i > j > k : \quad \begin{cases} \delta_{ij} + \delta_{jk} - 2\delta_{ik} \leq 1 \\ \delta_{ik} + \delta_{kj} - 2\delta_{ij} \leq 1 \\ \delta_{jk} + \delta_{ki} - 2\delta_{ji} \leq 1 \end{cases} \quad \text{transitividade} \\
 & \forall i, j : \quad \delta_{ij} \in \{0, 1\}
 \end{aligned}$$

Este método, contudo, é muito custoso computacionalmente e é viável apenas para grafos pequenos, com menos de uma centena de vértices. Diversos métodos buscam uma solução aproximada empregando heurísticas ou diferentes técnicas de otimização, como recozimento simulado ou otimização espectral (FORTUNATO, 2010).

Uma das heurísticas mais popular é o **algoritmo guloso** (também chamado de “Fast Newman”), que a cada passo seleciona e aplica a junção de comunidades que mais aumenta a modularidade (NEWMAN, 2004b). Ela é de certo modo similar ao algoritmo de Junção de Vizinhos apresentado na seção 2.2: inicialmente, cada vértice está em sua própria comunidade. A cada passo, para cada par de comunidades ( $C_u, C_v$ ) que compartilham uma aresta, calcula-se o ganho de modularidade  $\Delta Q_{uv} = Q_{uv} - Q$  por uni-las, mostrado na equação 2.23.

$$\Delta Q_{uv} = \frac{2}{S} (w_{uv} - S_u S_v) \quad (2.23)$$

onde  $S_u = \sum_{i \in C_u} s_i$  é a soma das forças dos vértices na comunidade  $C_u$  (respectivamente para  $S_v$  e  $C_v$ ), e  $w_{uv}$  é a soma dos pesos das arestas com um extremo em  $C_u$  e outro em  $C_v$ . A junção que maximiza  $\Delta Q_{uv}$  é então realizada, e o algoritmo continua até que todo o grafo esteja em uma única comunidade.

Este procedimento *bottom-up* produz um agrupamento hierárquico, que é então cortado onde a modularidade é maximizada. A cada passo este algoritmo requer no máximo  $m$  cálculos para  $\Delta Q_{uv}$ , e realiza  $n$  passos para uma complexidade de  $\mathcal{O}(mn)$ . Estruturas de dados especiais podem ser utilizadas para obter uma complexidade de  $\mathcal{O}(m \log^2 n)$ , permitindo a análise de grafos muito grandes (CLAUSET; NEWMAN; MOORE, 2004).

### 2.3.4 Agrupamento em árvores

Embora árvores sejam apenas um tipo específico de grafo, elas possuem algumas propriedades que são ignoradas ao utilizar um algoritmo de detecção de comunidades genérico. Para começar, a intuição que uma comunidade possui “mais arestas” dentro do que fora faz pouco

sentido em árvores: qualquer sub-árvore conexa  $c$  com  $n_c$  nós possui exatamente  $n_c - 1$  arestas dentro, e a única incógnita é o número de arestas exteriores.

O modelo nulo utilizado para calcular modularidade também ignora a propriedade de árvores não conterem ciclos. O modelo de configuração apenas busca preservar a distribuição de graus de um grafo, mas esta é uma propriedade fixada em árvores binárias: uma árvore binária com  $n$  folhas possui  $n$  vértices com grau 1 (as folhas) e  $n - 2$  vértices com grau 3 (nós internos). Contudo, um grafo aleatório com esta distribuição de graus provavelmente é desconexo e com ciclos, com probabilidade  $\approx 1 - \exp(-0.14n)$ <sup>4</sup>. A figura 9 ilustra alguns possíveis grafos aleatórios obtidos reconectando arestas de uma árvore binária.

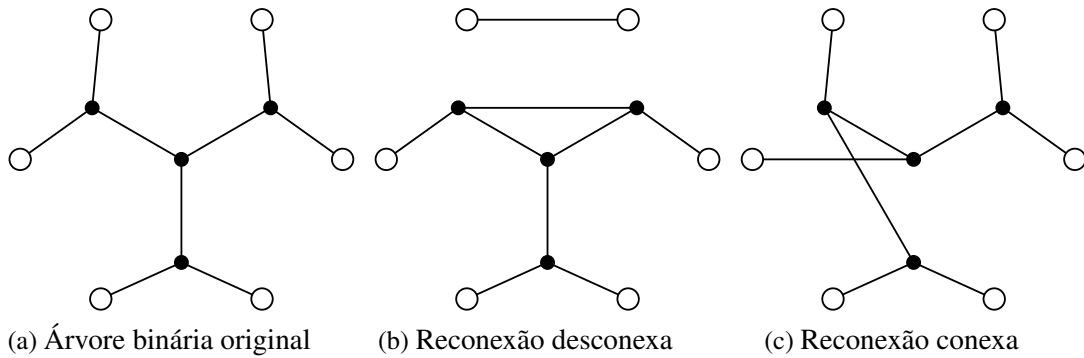


Figura 9 – Exemplos de grafos nulos com a mesma distribuição de graus do grafo original. Um grafo nulo pode ser obtido a partir do original trocando aleatoriamente os extremos de duas arestas simultaneamente, o que preserva a distribuição de graus.

Existem diversos métodos clássicos para particionamento de árvores na literatura, mas em geral estes requerem que algum parâmetro externo seja especificado. Um problema de partição de árvore pode ser visto como o problema de encontrar as  $k$  arestas que devem ser cortadas para produzir cada grupo como uma componente, ou  $k + 1$  sub-árvores. Seja  $w_s = \sum_{ij \in s} w_{ij}$  a soma dos pesos das arestas da sub-árvore  $s$ . O método de Lukes’ encontra a partição que minimiza o peso das arestas cortadas com a restrição de que  $w_s \leq W$  para todas as sub-árvores induzidas pelos cortes, para um dado  $W$  (LUKES, 1974). Frederickson (1991) apresenta algoritmos lineares que ou maximizam o mínimo  $w_s$ , ou minimizam o máximo  $w_s$  para um número fixado de cortes  $k$ . Maravalle, Simeone e Naldini (1997) possuem uma abordagem diferente, onde a árvore é simplesmente uma topologia das relações e a (dis)similaridade entre os elementos é dada por uma matriz de distâncias. Eles apresentam um algoritmo cúbico que maximiza o *split*<sup>5</sup> entre

<sup>4</sup> Considere o grafo aleatório em construção, com  $n$  vértices contendo um “toco” de aresta (folhas) e  $n - 2$  contendo três “tocos” (nós internos). A cada passo, selecionam-se dois “tocos” para formar uma aresta. Se duas folhas se conectarem, o grafo se torna desconexo. Conectando-se uma folha por vez, é necessário escolher um dos  $\approx 3n$  tocos internos dentre os  $\approx 4n$  disponíveis; e a cada folha seguinte existe um toco interno a menos e dois tocos totais a menos. A probabilidade de duas folhas não se conectarem é aproximadamente

$$\prod_{i=0}^{n-1} \frac{3n-i}{4n-2i} \approx \exp\left(-\frac{n}{2}(\log 4 - \log 3)\right) \approx \exp(-0.14n)$$

grupos para um número fixado de cortes  $k$ .

Este trabalho propõe duas variações do algoritmo “Fast Newman” ao alterar o modelo nulo utilizado para calcular a modularidade. A primeira é restrita a árvores binárias sem raiz e não-ponderadas, onde há interesse apenas na probabilidade  $P_{ij}$  de conexão entre vértices. A segunda é aplicável a quaisquer grafos, mas exige uma amostragem suficiente de uma distribuição que grafos genéricos nem sempre possuem.

#### 2.3.4.1 Modelo de árvore conexa

Árvores binárias sem raiz possuem uma distribuição de graus muito estreita, de modo que podem-se designar quatro classes de vértice, baseado em seu grau e quais nós ele está conectado:

- $k = 1$ : folha ( $L$ )
- $k = 3$ : nó interno
  - $A$ : conectado a um nó interno
  - $B$ : conectado a dois nós internos
  - $C$ : conectado a três nós internos

Esta classificação está ilustrada na figura 10. Nota-se que as folhas podem se conectar apenas a vértices dos tipos  $A$  e  $B$ , e isso será inserido no modelo para evitar componentes desconexas. Se a árvore possui  $n$  folhas, o número de nós internos é  $i = n - 2$  e o número total de nós é  $N = n + i = 2(n - 1)$ . O número de vértices na classe  $A$  é um parâmetro livre  $a$ , variando entre  $[2, \lfloor n/2 \rfloor]$ . O número de vértices nas classes  $B$  e  $C$  são dados em função de  $n$  e  $a$ :  $b = n - 2a$  e  $c = a - 2$ , respectivamente.

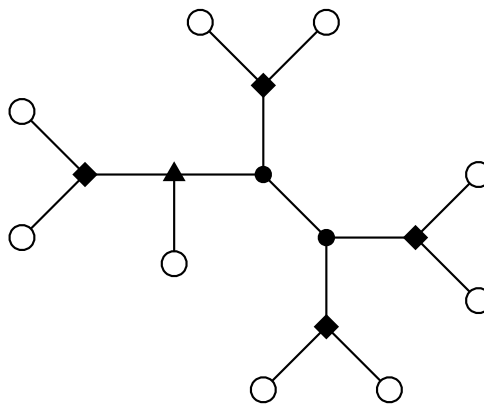


Figura 10 – Classificação dos vértices em uma árvore binária sem raiz. Círculos brancos são folhas, quadrados pretos pertencem à classe  $A$ , triângulos pretos pertencem à classe  $B$  e círculos pretos à classe  $C$ . Nesta árvore tem-se:  $n = 9, a = 4, b = 1, c = 2$ .

<sup>5</sup> *Split* é definido como a mínima distância entre quaisquer pares de grupos. A distância entre grupos é definida como o mínimo das distâncias entre os pares de elementos de cada grupo ( $d_{\min}$ , como definido na seção 3.3).

O modelo nulo proposto recebe como parâmetros  $n$  e  $a$  de uma dada árvore, o que fornece quantos nós de cada classe a árvore contém. A probabilidade  $P_{ij}$  de que os vértices  $i$  e  $j$  se conectem é a probabilidade de selecionar uma aresta aleatoriamente cujos extremos tenham as mesmas classes que eles, ou  $P(\text{class}(i), \text{class}(j))$ . Note que  $P(c_1, c_2) = P(c_2, c_1)$ . Esta probabilidade é dada pela seguinte tabela:

$$\begin{array}{llll} P(L, L) = 0 & P(A, L) = 2a/N & P(B, L) = b/N & P(C, L) = 0 \\ P(A, A) = 0 & P(A, B) = ab/(Ni) & P(A, C) = ac/(Ni) & \\ & P(B, B) = 2b^2/(Ni) & P(B, C) = 2bc/(Ni) & \\ & & P(C, C) = 3c^2/(Ni) & \end{array}$$

A modularidade de uma partição da árvore é então calculada pela equação 2.21, com  $P_{ij}$  dado como acima.

#### 2.3.4.2 Correlação de graus

Outro modelo simples consiste em criar um grafo aleatório que preserve não apenas a distribuição de graus, mas também sua correlação. A **correlação de graus** é a distribuição conjunta  $P(k, k')$  das arestas contendo um extremo com grau  $k$  e outro com grau  $k'$ . Esta distribuição pode ser obtida de um grafo simplesmente por:

$$P(k, k') = \frac{m_{kk'}}{m} = \frac{1}{2m} \sum_{k_i=k} \sum_{k_j=k'} A_{ij} \quad (2.24)$$

onde  $m_{kk'}$  é o número de arestas com vértices com graus  $k$  e  $k'$ . Esta medida também pode ser estendida para grafos ponderados, fornecendo o peso esperado  $\langle w(k, k') \rangle$ :

$$\langle w(k, k') \rangle = \frac{1}{2S} \sum_{k_i=k} \sum_{k_j=k'} A_{ij} w_{ij} \quad (2.25)$$

A probabilidade de conexão  $P_{ij}$  deste modelo nulo pode ser obtida simplesmente por  $P(k_i, k_j)$ , e respectivamente para pesos esperados. Este modelo parece promissor para árvores e outros grafos quase regulares, onde a distribuição de graus é bastante estreita. Contudo, para grafos com distribuição de graus com cauda longa, as probabilidades/pesos esperados são difíceis de obter experimentalmente, pois existem poucas amostras de vértices com grau alto.

Seria possível estender este modelo para utilizar o **grau médio dos vizinhos**  $k_{nn}(k)$ , definido como a média dos graus de todos os vértices adjacentes a vértices de grau  $k$ , como dado na equação 3.18. Esta métrica é mais robusta que a correlação de graus para o problema de amostragem citado acima.

$$\begin{aligned}
k_{nn}^i &= \frac{1}{k_i} \sum_j k_j A_{ij} \\
k_{nn}(k) &= \sum_{k'} k' P(k'|k) \\
&= \frac{1}{N_k} \sum_{k_i=k} k_{nn}^i
\end{aligned} \tag{2.26}$$

Note que as classes no modelo de árvores conexas podem ser entendidas como uma tupla  $(k_i, k_{nn}^i)$  onde

- $L = (1, 3)$ : grau 1, único vizinho possui grau 3
- $A = (3, 5/3)$ : grau 3, onde dois vizinhos têm grau 1 (folhas) e um vizinho tem grau 3
- $B = (3, 7/3)$ : grau 3, onde um vizinho tem grau 1 (folha) e dois vizinhos têm grau 3
- $C = (3, 3)$ : grau 3, onde todos os vizinhos têm grau 3



---

## AGRUPAMENTO E VALIDAÇÃO

---

Nesta pesquisa, foi necessário testar a implementação da ferramenta para garantir sua qualidade e a validade na análise de novos conjuntos. As ferramentas desenvolvidas para esta validação foram incluídas na ferramenta computacional, de modo que usuários também possam validar a qualidade de suas análises com o método e escolher as componentes disponíveis mais adequadas para cada conjunto de dados.

As análises realizadas são estudadas e expostas nas seções seguintes:

- A seção 3.1 estuda a validação externa dos dados, para os casos de um conjunto anotado com classes em que se possa comparar o agrupamento obtido com a classificação esperada.
- A seção 3.2 estuda métodos de reamostragem para validar uma classificação com ela mesma, sujeito a um certo grau de confiabilidade. Dentre essas ferramentas se encontra a técnica *bootstrap*.
- A seção 3.3 expõe uma extensão da técnica DAMICORE para classificação, que pode ser utilizada para validar conjuntos grandes de dados que de outra forma não são analisáveis pela ferramenta.

### 3.1 Validação externa

Em nossa pesquisa, todos os conjuntos de dados eram completamente anotados, então após encontrar um agrupamento de elementos por detecção de comunidades poderíamos validar o resultado com a classe real de cada elemento. Isto é chamado **validação externa**, e aqui apresentamos os índices utilizados para analisar e comparar os agrupamentos obtidos.

Dado um conjunto de dados  $D = \{d_1, \dots, d_n\}$ , um **agrupamento** ou **partição** é um conjunto de grupos  $P = \{p_1, \dots, p_c\}$  onde cada grupo é um subconjunto de  $D$ . Ainda, impomos

que  $\bigcup_i p_i = D$  e  $p_i \cap p_j = \emptyset, i \neq j$ , isto é, a partição cobre o conjunto completamente e os grupos não se sobrepõem.

Dados duas partições  $U$  and  $V$ , seja  $n_{ij} = |U_i \cap V_j|$  o número de elementos em comum entre os grupos  $U_i$  e  $V_j$ . A matriz definida por  $n_{ij}, \forall i = 1, \dots, |U|, \forall j = 1, \dots, |V|$  é chamada **matriz de confusão** ou **tabela de contingência**. Necessitamos das seguintes definições antes de definir as métricas entre partições. A nomenclatura assume que  $U$  é o conjunto de referência e  $V$  é o conjunto de teste.

- $N = \binom{n}{2}$ : número de pares de elementos em  $D$ .
- $N_U = \sum_i \binom{|U_i|}{2}$ : número de pares de elementos dentro dos grupos em  $U$ .
- $N_V = \sum_j \binom{|V_j|}{2}$ : número de pares de elementos dentro dos grupos em  $V$ .
- $TP = \sum_{ij} \binom{n_{ij}}{2}$ : verdadeiros positivos, número de pares de elementos que ocorrem dentro do mesmo grupo tanto em  $U$  quanto em  $V$ .
- $FP = N_U - TP$ : falsos positivos, número de pares que ocorrem dentro do mesmo grupo em  $U$  mas não em  $V$ .
- $FN = N_V - TP$ : falsos negativos, número de pares que ocorrem dentro do mesmo grupo em  $V$  mas não em  $U$ .
- $TN = N - (TP + FP + FN) = N - (N_U + N_V) + TP$ : verdadeiros negativos, número de pares que não ocorrem dentro do mesmo grupo em  $U$  e  $V$ .
- $A = TP + TN$ : concordâncias; número de pares que são postos do mesmo modo em ambas as partições.
- $D = FP + FN$ : discordâncias; número de pares que são postos de modos diferentes em cada partição.

A Figura 11 apresenta um exemplo de duas partições em um conjunto de dados e ilustra a classificação de cada par de elementos. A Tabela 4 mostra a classificação completa e a Tabela 5 mostra a matriz de contingência.

### 3.1.1 Índices comuns

O índice de Rand (RAND, 1971), dado na equação 3.1 é uma simples razão das concordâncias pelo número de pares. Este índice pode ser compreendido como a probabilidade de se seleccionar um par aleatório posto do mesmo modo em ambas as partições: tanto dentro do mesmo grupo em ambas, ou fora do mesmo grupo. Na prática, o número de verdadeiros negativos



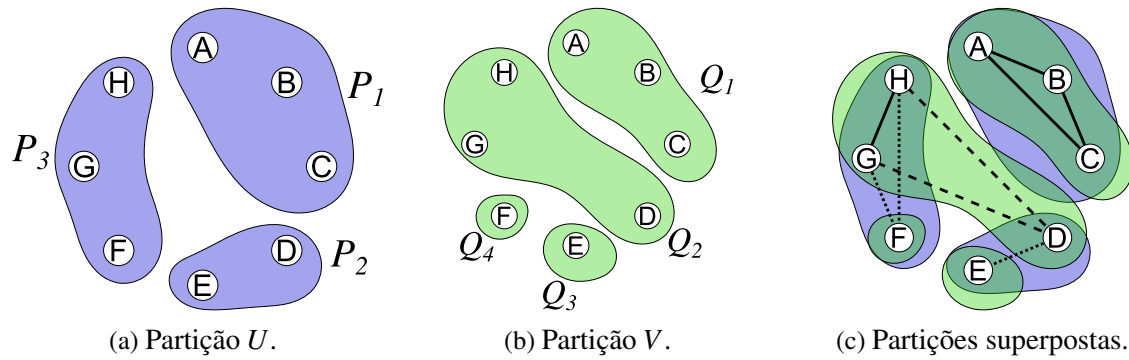


Figura 11 – Exemplo de partições. Linhas sólidas são verdadeiros positivos, linhas tracejadas são falsos positivos e linhas pontilhadas são falsos negativos. Não são mostrados os verdadeiros negativos.

domina a análise e o índice tende a 1 à medida que  $n$  cresce, o que não é uma característica central para agrupamentos (HUBERT; ARABIE, 1985).

$$R = \frac{A}{N} = \frac{A}{A + D} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.1)$$

O índice de Jaccard (JACCARD, 1912) (que antecede o de Rand) remove TN do índice de Rand, como dado na equação 3.2. Ele pode ser interpretado como a probabilidade de se escolher um verdadeiro positivo dentre todos os pares que ocorrem dentro de um grupo em pelo menos uma das partições.

$$J = \frac{TP}{TP + D} = \frac{TP}{TP + FP + FN} \quad (3.2)$$

Fowlkes e Mallows propuseram um índice de similaridade considerando os verdadeiros positivos sobre a média geométrica de pares possíveis (FOWLKES; MALLOWS, 1983), como dado na equação 3.3. Este índice foi originalmente proposto por Ochiai em 1957 (OCHIAI,

	A	B	C	D	E	F	G	H
A		●	●	○	○	○	○	○
B			●	○	○	○	○	○
C				○	○	○	○	○
D					●	○	●	●
E						○	○	○
F							●	●
G								●
H								

Tabela 4 – Congruência dos pares entre partições  $U$  e  $V$  apresentadas na Figura 11. ● denota verdadeiro positivo, ○ denota verdadeiro negativo, ◐ denota falso negativo e ◑ denota falso positivo.

$n_{ij}$	$Q_1$	$Q_2$	$Q_3$	$Q_4$
$P_1$	3	-	-	-
$P_2$	-	1	1	-
$P_3$	-	2	-	1

Tabela 5 – Matriz de contingência do exemplo na Figura 11.

1957).

$$FM = \frac{TP}{\sqrt{N_P N_Q}} \quad (3.3)$$

Em um comentário sobre o artigo de Fowlkes e Mallows, David Wallace propôs um par de índices que normaliza TP pelo número de pares em cada partição (WALLACE, 1983), como dado na equação 3.4. O índice  $W_U$  pode ser interpretado como a probabilidade condicional de encontrar um par concordante em  $V$ , dado que ele concorda em  $U$ ; respectivamente para  $W_V$ , trocando  $U$  e  $V$ . O índice FM se torna então a média geométrica de  $W_U$  e  $W_V$ .

$$W_U = \frac{TP}{N_U}, \quad W_V = \frac{TP}{N_V} \quad (3.4)$$

É necessário analisar ambos os índices simultaneamente, especialmente se a distância entre  $N_U$  e  $N_V$  for grande. Por exemplo, se  $V$  é composto por uma única partição, então  $N_V = N$  e  $N_U = TP$ , e consequentemente  $W_U = 1$ . Contudo, se  $U$  é composto por  $k$  partes com aproximadamente o mesmo tamanho, então  $W_V \approx 1/k \ll 1$ . Isto mostra que um dos índices pode parecer indicar uma grande concordância enquanto o outro indica o oposto.

Meilă propõe um critério que minimiza o número de erros quando correspondemos os grupos de uma partição com os da outra (MEILĂ, 2005). Assumindo que  $|U| \leq |V|$ , seja  $\pi$  um mapeamento injetivo de  $(1, \dots, |U|)$  em  $(1, \dots, |V|)$ , o que significa que o grupo  $U_i$  “corresponde ao” grupo  $V_{\pi(i)}$ . Podemos definir uma matriz de permutação  $\delta$  tal que  $\pi(i) = j \Leftrightarrow \delta_{ij} = 1$ . O número ótimo de correspondências  $C$  é obtido pela solução do seguinte problema de otimização linear:

$$\begin{aligned} \max \quad & C = \sum_{ij} \delta_{ij} n_{ij} \\ \text{s.a} \quad & \forall i: \sum_j \delta_{ij} \leq 1 \\ & \forall j: \sum_i \delta_{ij} \leq 1 \\ & \forall i, j: \delta_{ij} \in \{0, 1\} \end{aligned}$$

Isto é uma instância do problema de atribuição, que pode ser solucionado de forma eficiente pelo método de Munkres (MUNKRES, 1957). Devemos salientar que se  $|U| \neq |V|$ , então nem todos os grupos terão um correspondente. O índice de classificação  $c$  é definido simplesmente como  $c = C/n$ . Este índice pondera correspondências entre agrupamentos igualmente, e é igual a 1 para agrupamentos idênticos. Contudo, o índice ignora completamente a organização da parte

sem correspondência de cada grupo, e portanto não diferencia embaralhamentos diferentes entre partições com o mesmo número de correspondências a uma partição de referência (MEILA, 2007).

### 3.1.2 Correção para o acaso

Uma preocupação comum entre estes índices é que, mesmo que todos variem em  $[0, 1]$  e alguns possam ser interpretados como probabilidades, eles não variem linearmente neste intervalo ou mesmo possuam um mínimo de zero (WALLACE, 1983). Mesmo que o analista seja apresentado com um valor de 0.8, não há indicação se este é um valor “alto” ou “baixo” para a correspondência entre os agrupamentos.

Uma maneira de aumentar a interpretabilidade dos índices é **corrigindo para o acaso**, isto é, ajustá-los para assumir um valor constante (normalmente 0) quando comparando partições aleatórias contra um modelo nulo. A forma geral de um índice ajustado é

$$I' = \frac{I - \langle I \rangle}{\max\{I\} - \langle I \rangle} \quad (3.5)$$

onde  $\langle \cdot \rangle$  é o valor esperado do índice e  $\max\{I\}$  é o máximo valor possível. Deste modo,  $I'$  assume um máximo de 1, e assume 0 quando  $I = \langle I \rangle$ . O índice pode então assumir valores negativos, que são de difícil interpretação.

Contudo, o modelo nulo requer definir uma distribuição da qual um modelo aleatório pode ser definido e amostrado. Uma distribuição comum para partições é a distribuição hipergeométrica, que provê uma matriz de contingência aleatória para partições dado a soma das colunas e linhas ( $\sum_j U_{ij}$  e  $\sum_i V_{ij}$ ) (HUBERT; ARABIE, 1985). Esta escolha encontra algumas críticas, como posto por Meilã (MEILA, 2003):

Um problema com índices ajustados é que o ponto de origem é uma expectativa sob uma hipótese nula. A hipótese nula é que a) os dois agrupamentos são amostrados de modo independente, e b) os agrupamentos são amostrados do conjunto de todos os pares de partições com  $|U_i|, |V_j|$  pontos fixados em cada grupo (FOWLKES; MALLOWS, 1983; HUBERT; ARABIE, 1985). Na prática, a segunda suposição é normalmente violada. Muitos algoritmos têm como entrada um número  $K$  de grupos, mas o número de pontos em cada grupo é um resultado da execução do algoritmo. Na maioria das situações de análise exploratória de dados, não é natural assumir que alguém possa saber exatamente quantos pontos há em cada grupo. Os problemas listados acima já são conhecidos há muito tempo pela comunidade estatística; veja por exemplo (WALLACE, 1983).

Albatineh e Niewiadomska-Bugaj (2011) realizaram a correção de diversos índices da literatura com o modelo nulo proposto. Eles também utilizam uma abordagem por expansão de séries de Taylor que não será discutida aqui.

Sob a distribuição hipergeométrica, onde o número de elementos em cada grupo é fixado tanto em  $U$  quanto em  $V$ , o valor esperado para o número de verdadeiros positivos TP é dado na equação 3.6 (HUBERT; ARABIE, 1985).

$$\langle TP \rangle = \frac{N_U N_V}{N} \quad (3.6)$$

O índice de Rand ajustado AR, proposto por Hubert e Arabie (1985), utiliza o valor esperado do índice de Rand na equação 3.7.

$$\begin{aligned} \langle R \rangle &= 1 + 2N_U N_V / N^2 - (N_U + N_V) / N \\ AR &= \frac{TP - N_U N_V / N}{\frac{1}{2}(N_U + N_V) - N_U N_V / N} \\ &= \frac{TP - \langle TP \rangle}{\frac{1}{2}(N_U + N_V) - \langle TP \rangle} \end{aligned} \quad (3.7)$$

O índice de Fowlkes-Mallows ajustado AFM utiliza o valor esperado para o índice de Fowlkes-Mallows dado na equação 3.8 (FOWLKES; MALLOWS, 1983).

$$\begin{aligned} \langle FM \rangle &= \frac{\sqrt{N_U N_V}}{N} \\ AFM &= \frac{TP - N_U N_V / N}{\sqrt{N_U N_V} - N_U N_V / N} \\ &= \frac{TP - \langle TP \rangle}{\sqrt{N_U N_V} - \langle TP \rangle} \end{aligned} \quad (3.8)$$

Não foi encontrado na literatura o cálculo de índices de Wallace ajustados. Foi feita sua derivação inicialmente calculando os índices esperados (equação 3.9), e os índices ajustados  $AW_U$  e  $AW_V$  seguem facilmente utilizando a equação 3.6:

$$\begin{aligned} \langle W_U \rangle &= N_V / N & AW_U &= \frac{TP - \langle TP \rangle}{N_U - \langle TP \rangle} \\ \langle W_V \rangle &= N_U / N & AW_V &= \frac{TP - \langle TP \rangle}{N_V - \langle TP \rangle} \end{aligned} \quad (3.9)$$

Não encontramos uma derivação para o valor esperado do índice de classificação  $c$ . O índice de Jaccard ajustado possui uma derivação mais complexa e é baseado em uma aproximação, e não foi utilizado nesta pesquisa. Para maiores informações, consulte (ALBATINEH, 2010; ALBATINEH; NIEWIADOMSKA-BUGAJ, 2011).

### 3.1.3 Informação mútua

A informação mútua  $I(U, V)$  é um índice com um histórico diferente dos anteriores, buscando calcular a informação (em bits) compartilhada entre dois agrupamentos (MEILA, 2007). Ela assume que a matriz de contingência é uma amostra das medidas de duas distribuições de probabilidade (possivelmente não independentes) que atribuem cada elemento em grupos em  $U$  e  $V$ .

Inicialmente, definimos  $u_i = |U_i|/n$ ,  $v_j = |V_j|/n$  e  $r_{ij} = n_{ij}/n$  como a probabilidade de selecionar um elemento aleatoriamente de, respectivamente,  $U_i$ ,  $V_j$ , e ambos  $U_i$  e  $V_j$ . Nós definimos as entropias de cada partição  $H(U)$  e  $H(V)$  e a entropia conjunta  $H(U, V)$  como segue. Também definimos  $0 \lg 0 \equiv 0$ .

$$\begin{aligned} H(U) &= - \sum_i u_i \lg u_i & H(V) &= - \sum_j v_j \lg v_j \\ H(U, V) &= - \sum_{ij} r_{ij} \lg r_{ij} \end{aligned}$$

Podemos interpretar as entropias  $H(U)$  e  $H(V)$  como o mínimo número médio de bits necessário para codificar a qual grupo um elemento pertence. A entropia conjunta  $H(U, V)$  é então o mínimo número médio de bits necessário para codificar os dois grupos ao qual um elemento pertence, simultaneamente. Se um agrupamento é completamente dependente do outro - por exemplo, se dado o grupo  $U_i$  ao qual um elemento pertence não existe dúvida sobre qual grupo  $V_j$  ele pertence - então  $H(U, V) = \max\{H(U), H(V)\}$ . Do contrário, se eles forem estatisticamente independentes, então  $H(U, V) = H(U) + H(V)$ .

Com esta compreensão podemos definir a **informação mútua**  $I(U, V)$  entre agrupamentos como a quantidade de informação compartilhada entre eles. Por exemplo, dado que um elemento pertence ao grupo  $U_i$  em  $U$ , qual é o decréscimo de informação necessária para codificar seu grupo  $V_j$  em  $V$ ? A média deste decréscimo de informação é a informação mútua, como dada na equação 3.10.

$$I(U, V) = H(U) + H(V) - H(U, V) = \sum_{ij} r_{ij} \lg \left( \frac{r_{ij}}{p_i q_j} \right) \quad (3.10)$$

Se os agrupamentos são independentes, a informação mútua é zero; se forem perfeitamente dependentes ou iguais, a informação mútua é o mínimo das suas entropias. A Figura 12 ilustra estes conceitos.

A informação mútua pode ser utilizada para definir uma métrica entre partições. A entropia condicional  $H(U|V) = H(U) - I(U, V)$  representa a quantidade de informação “independente” ou “não explicada” em  $U$  quando o grupo de um elemento em  $V$  é conhecido; respectivamente para  $H(V|U)$ , trocando  $U$  e  $V$ . A **distância de variação de informação**  $d(U, V)$

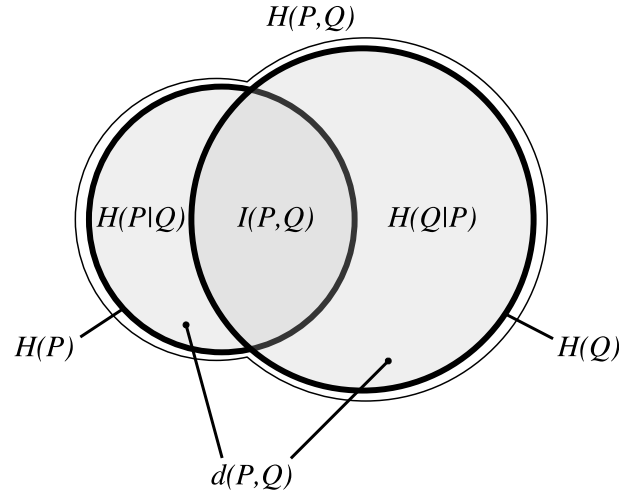


Figura 12 – Ilustração das entropias de cada partição e conceitos relacionados.

é uma medida simétrica desta quantidade de informação independente, como definido na equação 3.11. Esta medida satisfaz as propriedades de uma métrica, e não depende em contar o número de pares em comum ou em encontrar grupos correspondentes (MEILA, 2007).

$$\begin{aligned}
 d(U, V) &= H(U|V) + H(V|U) \\
 &= H(U) + H(V) - 2I(U, V) \\
 &= H(U, V) - I(U, V) = \sum_{ij} r_{ij} \lg(u_i v_j)
 \end{aligned} \tag{3.11}$$

Esta métrica também é medida em bits, e pode ser desejável normalizá-la para obter um índice entre 0 e 1. Existem diversas propostas de fatores de normalização:

- entropia conjunta  $H(U, V)$ ;
- máximo das entropias  $\max\{H(U), H(V)\}$
- mínimo das entropias  $\min\{H(U), H(V)\}$
- média aritmética das entropias  $\frac{1}{2}(H(U) + H(V))$
- média geométrica das entropias  $\sqrt{H(U)H(V)}$

Vinh, Epps e Bailey (2010) apresentam correções para o acaso de diversas dessas normalizações sob o modelo nulo hipergeométrico, que não foram utilizadas nessa pesquisa.

### 3.1.4 Partições binárias

Algumas tarefas de classificação exigem classificar elementos em apenas dois grupos, em geral como parte de um problema de decisão: classificar um email como spam ou não, de

modo a decidir se ele deve ser mostrado ao usuário; analisar o perfil de um cliente e decidir conceder ou não crédito; detecção de doenças; dentre outras aplicações. Nestes casos, a matriz de confusão se torna mais simples, com apenas 2x2 entradas, e podemos aplicar outras medidas para avaliar a qualidade de um agrupamento.

As medidas de precisão  $P$  e sensibilidade (*recall*)  $R$  fornecem uma visão conjunta da efetividade de um classificador:

- **Precisão:** fração das instâncias classificadas como positivas que são corretas (Equação 3.12);
- **Sensitividade:** fração das instâncias positivas classificadas corretamente (Equação 3.13).

$$P = \frac{TP}{TP + FP} \quad (3.12)$$

$$R = \frac{TP}{TP + FN} \quad (3.13)$$

Por exemplo, no contexto de classificação de spam, a precisão fornece a probabilidade de mensagens classificadas como spam serem, de fato, spam; e a sensibilidade fornece a probabilidade de mensagens de spam serem classificadas como tal. É trivial alcançar uma sensibilidade de 100% simplesmente classificando todas como spam, de modo que é necessário avaliar ambas as medidas em conjunto. A métrica  $F_1$  fornece um coeficiente que unifica ambas utilizando sua média harmônica:

$$F_1 = \left[ \frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right) \right]^{-1} = 2 \frac{P \cdot R}{P + R} \quad (3.14)$$

Neste caso, ambas as métricas são ponderadas igualmente. Na prática, podemos dar maior importância para uma em relação à outra: no caso de spam, uma mensagem legítima classificada incorretamente possui maior custo que uma mensagem de spam chegando ao usuário, de modo que desejaríamos maximizar a precisão. Isto pode ser obtido com a métrica  $F_\beta$ , onde  $0 < \beta < 1$  privilegia a precisão enquanto  $\beta > 1$  privilegia a sensibilidade:

$$F_\beta = (1 + \beta^2) \frac{P \cdot R}{\beta^2 \cdot P + R} \quad (3.15)$$

Tais métricas apresentam o problema de eleger uma das classes como “positiva” e tomá-la como referência, ignorando a quantidade de falsos negativos encontrados. Isto não traz problemas quando cada classe representa exatamente metade das instâncias do conjunto; se este não for o caso, encontramos uma medida de qualidade muito diferente para a mesma matriz de confusão ao analisar as medidas duais tomando a outra classe como positiva (e.g., não-spam). A Tabela 6

		Conjunto 1		Conjunto 2	
		Spam (100)	Legítimo (100)	Spam (100)	Legítimo (900)
Classes	Spam	90	5	90	5
	Legítimo	10	<b>95</b>	10	<b>895</b>
		$P = 90/100$	$\hat{P} = \mathbf{95/100}$	$P = 90/100$	$\hat{P} = \mathbf{895/900}$
		$R = 90/95$	$\hat{R} = \mathbf{95/105}$	$R = 90/95$	$\hat{R} = \mathbf{895/905}$
		$M = 0.851$		$M = 0.915$	

Tabela 6 – Exemplo de diferentes matrizes de confusão com os mesmos valores de precisão ( $P$ ) e sensibilidade ( $R$ ) para uma classe, mas diferentes para a outra ( $\hat{P}$  e  $\hat{R}$ ). Note que o conjunto 1 está igualmente dividido entre as classes, e os valores duais  $\hat{P}$  e  $\hat{R}$  encontram-se na mesma escala que os primais. No conjunto 2, existem muitos mais exemplos de elementos legítimos, e logo as medidas duais são distintas e mais próximas de 1. Intuitivamente, o classificador é mais efetivo no conjunto 2, uma vez que classifica corretamente muito bem os elementos “negativos”. Esta característica é capturada pelo coeficiente de Matthews  $M$ .

ilustra esta diferença para a classificação de dois conjuntos de dados com proporções diferentes de exemplos positivos e negativos.

O coeficiente de correlação de Matthews  $M$  provê uma medida menos enviesada pela proporção das classes no conjunto, além de ser simétrico, sem assumir que uma das classes é “positiva” (MATTHEWS *et al.*, 1985). Sua fórmula é dada na Equação 3.16. Seu valor varia entre -1 e +1, onde +1 indica classificação perfeita, 0 indica que ela não é distinguível de uma classificação aleatória e -1 indica uma classificação reversa perfeita.

$$\begin{aligned}
 s &= \frac{TP + FN}{N} \\
 p &= \frac{TP + FP}{N} \\
 M &= \frac{TP/N - sp}{\sqrt{sp(1-s)(1-p)}}
 \end{aligned} \tag{3.16}$$

## 3.2 Teste de confiabilidade

Na seção 3.1 mostramos índices para validação de agrupamentos/partições quando se possui uma partição de referência, onde os resultados esperados são conhecidos. Na análise exploratória de dados, contudo, tal conjunto de referência não existe ou é apenas parcial. Obter uma estimativa da confiabilidade de um resultado é uma tarefa necessária para interpretá-lo corretamente e tomar decisões baseadas nele. Uma vez que nessas situações possuímos apenas o conjunto de dados que desejamos analisar, qualquer estimativa da distribuição “real” de onde o conjunto foi amostrado deve ser estimada a partir dos próprios elementos.



### 3.2.1 Métodos de reamostragem

Uma técnica para se obter uma distribuição de um estimador - digamos, a média de um conjunto - é realizar a amostra de novos conjuntos de dados a partir do universo original e calcular o estimador para cada um deles. Evidentemente, tal técnica é muito custosa e pode não ser possível realizar novas amostras de um universo, como se notam em diversos conjuntos de dados disponíveis.

Os **métodos de reamostragem** consistem em utilizar o conjunto de dados original como se fosse o universo de onde instâncias podem ser amostradas, e então calcular estimadores sobre sub-amostras dele para obter uma distribuição. Se o conjunto de dados original foi amostrado de forma independente, então diversos estimadores como média e variância se aproximam com quase certeza do estimador original.

Uma abordagem simples, chamada *jackknife* (“canivete”) (TUKEY, 1958), consiste em repetir diversas vezes a análise de uma sub-amostra do conjunto: para um conjunto de  $N$  elementos, realizamos  $N$  análises com  $N - 1$  elementos, a cada vez removendo o  $i$ -ésimo elemento, com  $i = 1..N$ . Uma ligeira deficiência deste método é que os conjuntos reamostrados não possuem a mesma dimensão do conjunto original.

Uma abordagem um pouco mais complexa, mas mais poderosa, é chamada *bootstrap* (“autoinicialização”) (FELSENSTEIN, 1985), e consiste em realizar reamostras aleatoriamente com reposição até obter um novo conjunto com a mesma dimensão do original. Este procedimento é repetido um grande número de vezes para se obter uma distribuição representativa do estimador desejado. Caso o conjunto seja pequeno é possível enumerar todos as reamostras possíveis e obter o estimador para cada uma delas: para um conjunto de tamanho  $N$ , existem  $\binom{2N-1}{N}$  possíveis novos conjuntos.

### 3.2.2 Teste para filogenias

Quando aplicado em filogenia, onde se tem uma matriz de  $M$  táxons por  $N$  caracteres, é necessário decidir qual é o conjunto a ser reamostrado: seriam as espécies presentes com suas características, ou os caracteres que as compõem? Felsenstein argumenta que o conjunto verdadeiramente independente é o dos caracteres, que seriam uma amostra aleatória de todos os caracteres possíveis (FELSENSTEIN, 1985). Tal discussão não é tão interessante no nosso caso, uma vez que a matriz que utilizamos é quadrada e simétrica.

Uma vez realizadas diversas reamostragens, seja pelo *jackknife* ou *bootstrap*, obtemos um conjunto de árvores filogenéticas como resultado. Um método simples para determinar uma única árvore deste conjunto é realizar uma **árvore de maioria**, onde as ramificações mais frequentes são adotadas como corretas. Cada ramificação possui então um certo grau de certeza, dado pela frequência em que ocorre por todas as árvores.

Outro método mais estrito visualiza uma árvore (enraizada) como um conjunto de grupos

induzidos pelos descendentes dos nós internos, direta ou indiretamente. Em Biologia, tais grupos são chamados **grupos monofiléticos**, sendo um grupo de espécies que possuem um ancestral comum que não é ancestral de nenhuma outra espécie fora do grupo. Caso as árvores não possuam raiz, selecionamos aleatoriamente um mesmo elemento para ser a raiz em todas as árvores. Tais grupos estão ilustrados na Figura 13.

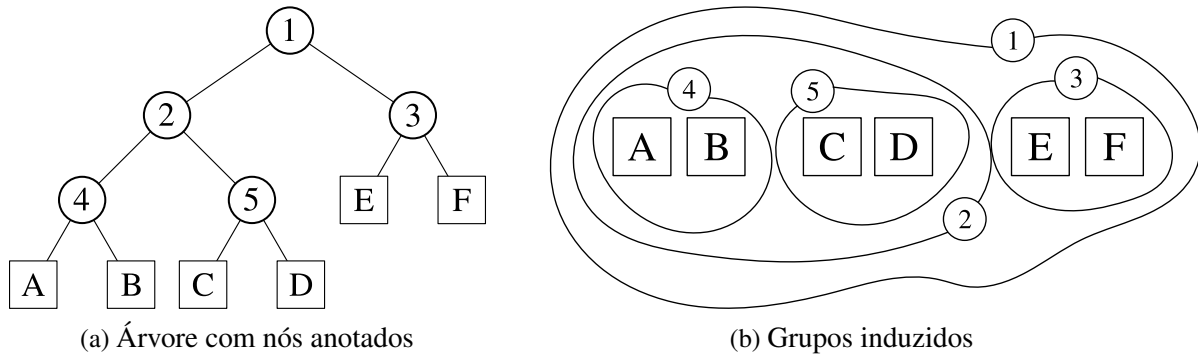


Figura 13 – Grupos de folhas induzidos pelos nós internos

Desta maneira, podemos simplesmente contar a ocorrência de tais grupos dentre todas as árvores, e determinar quais grupos são frequentes o suficiente. Por exemplo, se desejarmos um nível de confiança de 95%, escolheríamos todos os grupos que ocorrem em mais de 95% das árvores. A árvore de consenso resultante poderá conter ramificações múltiplas, que consistem em grupos cuja estrutura interna não é consistente o bastante. Note que a árvore de maioria corresponde a um nível de confiança de 50% das replicações.

### 3.3 Classificação

A metodologia DAMICORE apresentada é um *framework* de agrupamento para aprendizado não-supervisionado. Um dos objetivos desta pesquisa é estender suas capacidades para tarefas de classificação, onde amostras não anotadas são comparadas com um conjunto de treinamento para determinar suas anotações.

Este é um objetivo muito relevante em si mesmo, permitindo aplicar a metodologia para um conjunto de problemas que ainda não consideramos. Também, tal desenvolvimento pode ajudar a aplicar a ferramenta de agrupamento para conjuntos de dados maiores. Análises de desempenho indicam que o gargalo da metodologia de agrupamento encontra-se no passo de cálculo da métrica: construir uma matriz de distâncias de tamanho  $\mathcal{O}(n^2)$  é custoso para conjuntos de tamanho moderado ( $\sim 1000$  elementos), e aplicar esta técnica para conjuntos maiores é infactível sem recursos computacionais consideráveis.

Uma maneira de contornar essa limitação é agrupar um pequeno subconjunto de  $k$  elementos e determinar seus grupos como classes; então, podem-se classificar os elementos

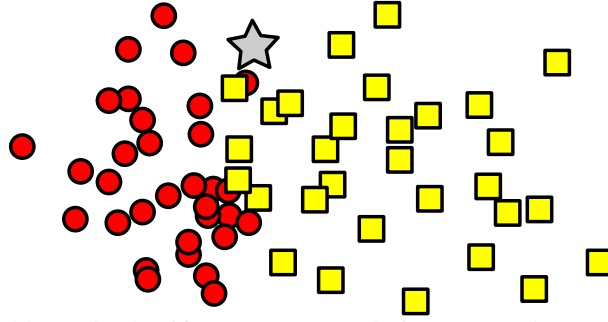


Figura 14 – Exemplo de problema de classificação. Deve-se designar uma classe para a nova instância (estrela) utilizando a distância às instâncias de dados já classificadas (círculos e quadrados).

restantes para estes grupos obtidos, o que possui complexidade  $\mathcal{O}(k(n-k))$ . Se  $k$  é um fator constante  $\alpha$  de  $n$ , o ganho em desempenho é da ordem de  $\alpha(1-\alpha)$ .

Seja  $\mathcal{D}$  um espaço genérico de onde instâncias de dados são obtidas, e  $\mathcal{C}$  um conjunto de classes ou anotações, normalmente discreto e finito. Uma função de classificação  $f(x) : \mathcal{D} \mapsto \mathcal{C}$  mapeia cada instância a uma classe. Esta função pode até mesmo ser não-determinística, isto é, duas instâncias de dados idênticas podem ser classificadas de forma diferente. Um **problema de classificação** é a tarefa de encontrar uma função aceitável  $\hat{f}(x)$  que tem bom desempenho como aproximação de  $f$ .

Um **conjunto de treinamento**  $T = \{\langle t_1, c_1 \rangle, \dots, \langle t_m, c_m \rangle\}$ , onde  $t_i \in \mathcal{D}$  e  $c_i \in \mathcal{C}$ , é um conjunto de valores “verdadeiros” onde  $f(t_i) = c_i$ . Este conjunto de dados nos dá uma base para construir nosso classificador  $\hat{f}$  e também nos informa em como nossa função executa sobre instâncias já anotadas. Normalmente, também são dadas instâncias de teste  $X = \{x_1, \dots, x_n\}$ , com  $x_i \in \mathcal{D}$ , para as quais deve-se retornar as classificações  $\{\hat{f}(x_1), \dots, \hat{f}(x_n)\}$ . A Figura 14 ilustra este problema.

Dada uma métrica  $d$  definida sobre  $\mathcal{D}$ , pode-se definir uma distância entre um elemento  $x$  e uma classe  $C$  construindo sobre a distância entre  $x$  e os elementos  $y \in C$ :

- Mínimo:  $d_{\min}(x, C) = \min_{y \in C} \{d(x, y)\}$
- Máximo:  $d_{\max}(x, C) = \max_{y \in C} \{d(x, y)\}$
- Média:  $d_{\text{avg}}(x, C) = \frac{1}{|C|} \sum_{y \in C} d(x, y)$

Outra opção, única para a métrica NCD, consiste em concatenar todos os elementos dentro de uma classe em uma única *string*, e então empregar a distância usual do elemento para o agregado:

$$d_{\text{aggr}}(x, C) = \text{NCD}(x, \text{concat}(y | \forall y \in C))$$

De maneira similar, pode-se definir a distância entre as próprias classes ao estender estas definições. Contudo, não há nenhuma garantia de que a distância seja uma métrica, satisfazendo as propriedades delineadas na Seção 2.1. As distâncias mais usuais entre duas classes  $C$  e  $C'$  são as seguintes:

- Mínimo:  $d_{\min}(C, C') = \min_{t \in C} \{d_{\min}(t, C')\}$
- Distância de Hausdorff dirigida:  $d_h(C, C') = \max_{t \in C} \{d_{\min}(t, C')\}$
- Distância de Hausdorff:  $d_H(C, C') = \max\{d_h(C, C'), d_h(C', C)\}$

### 3.3.1 Classificadores comuns

Dada a distância entre um elemento e uma classe  $d(x, C)$ , pode-se atribuir uma instância não anotada simplesmente para a classe mais próxima a ela. Define-se então um **classificador simples** como na equação 3.17. Classificar todas as instâncias de dados em  $X$  requer  $\mathcal{O}(mn)$  cálculos de distância.

$$\hat{f}(x) \leftarrow \arg \min_{C \in \mathcal{C}} d(x, C) \quad (3.17)$$

O **classificador  $k$ -NN**, para uma dada distância entre elementos  $d$ , atribui um elemento à classe mais frequente dentre seus  $k$  vizinhos mais próximos (MITCHELL, 1997). Empates devem ser quebrados de acordo com uma estratégia, tal como escolher aleatoriamente ou simplesmente deixar o elemento sem classificação. O classificador simples acima é equivalente ao classificador 1-NN com  $d(x, C) = d_{\min}$ . O classificador  $k$ -NN é dado pela equação 3.18:

$$\hat{f}(x) \leftarrow \arg \max_{C \in \mathcal{C}} \sum_{y \in K(x)} \delta(C, f(y)) \quad (3.18)$$

onde  $K(x) = \{y_1, \dots, y_k\}$  é o conjunto dos  $k$  elementos  $y \in T$  mais próximos a  $x$ , e  $\delta(C, C')$  é o delta de Kronecker.

Este classificador é tanto poderoso quanto mal compreendido, já que o parâmetro  $k$  influencia fortemente sua efetividade mas não existe regra geral que pode ser aplicada para escolhê-lo. Baixos valores de  $k$  tornam o método altamente local com fronteiras de decisão “denteadas” (Figuras 15a e 15b), enquanto valores maiores de  $k$  tornam o método global, com fronteiras de decisão suaves (Figuras 15c e 15d). No limite de  $k = m$ , o classificador se torna o **classificador trivial**, que atribui a classe mais observada para todas as instâncias de teste.

Outra abordagem para o problema de classificação é assumir que as instâncias de dado são governadas por distribuições de probabilidade, e que decisões ótimas podem ser tomadas considerando tais probabilidades em conjunto com os dados observados. Suponha que a probabilidade condicional  $P(C|x)$  de um elemento  $x$  pertencer à classe  $C$  seja conhecida. A hipótese de

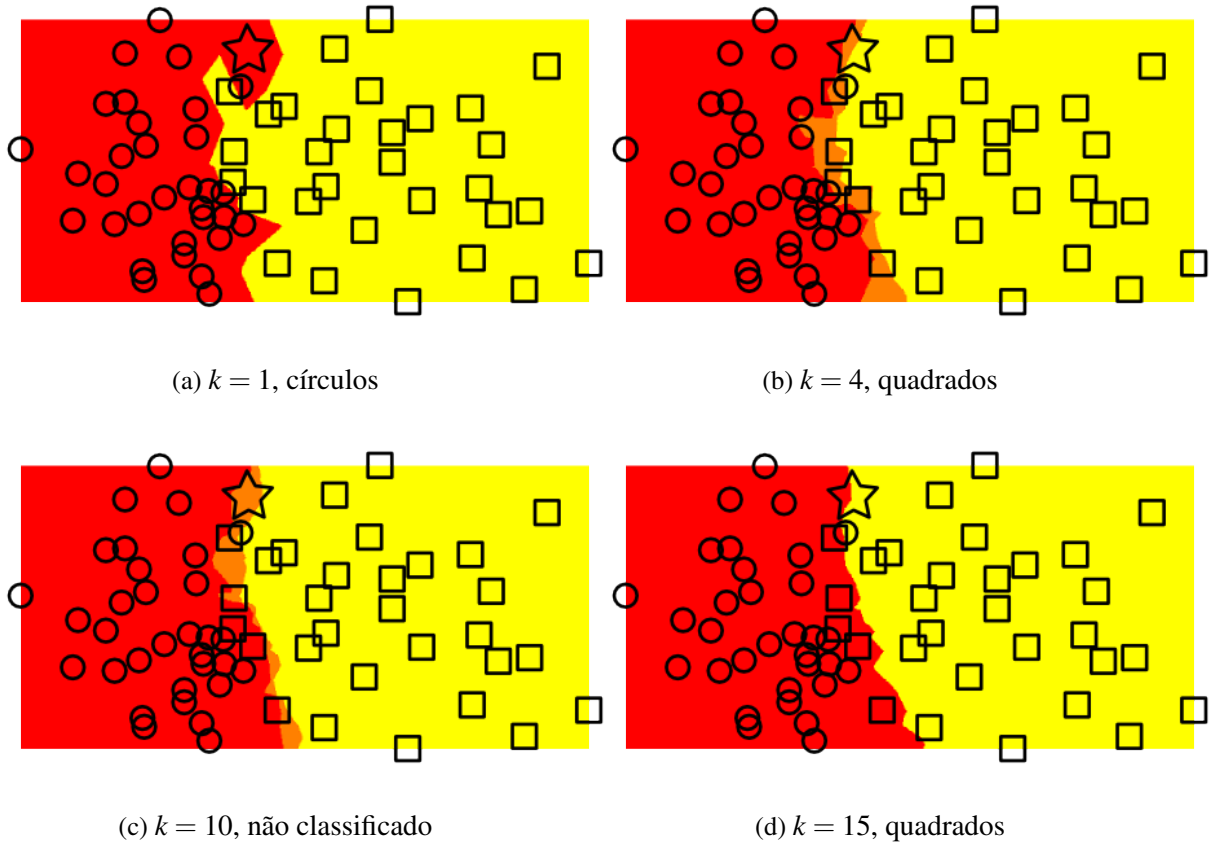


Figura 15 – Fronteiras de decisão para alguns valores de  $k$  para o classificador  $k$ -NN. Regiões onde há um empate apresentam cores intermediárias. Empates não ocorrem com  $k$  ímpar pois só existem duas classes para escolher. Note que a classificação da nova instância depende de  $k$  e pode até mesmo não ser classificada.

máximo *a posteriori* (MAP) é que o elemento pertence à classe que maximiza tal probabilidade, isto é,

$$C_{\text{MAP}} = \arg \max_{C \in \mathcal{C}} P(C|x)$$

Ao aplicar o teorema de Bayes  $P(A|B) = P(B|A) \frac{P(A)}{P(B)}$ , pode-se reescrever este classificador como

$$\begin{aligned} C_{\text{MAP}}(x) &= \arg \max_{C \in \mathcal{C}} \frac{P(x|C)P(C)}{P(x)} \\ &= \arg \max_{C \in \mathcal{C}} P(x|C)P(C) \end{aligned}$$

Esta equação permite uma estimativa mais simples da distribuição de probabilidade subjacente utilizando os dados observados. Pode-se estimar  $P(C)$  pela fração  $m_C/m$ , onde  $m_C$  é o número de elementos em  $C$  no conjunto de treinamento. Estimar  $P(x|C)$  é mais difícil, pois na maioria dos problemas  $x$  não é sequer observado nos dados de treinamento.

Suponha que este problema pode ser simplificado assumindo-se que os atributos de  $x = (x_1, \dots, x_d)$  são condicionalmente independentes entre si. Então, a probabilidade de se observar um elemento  $x$  é a conjunção de se observar cada atributo  $x_1, \dots, x_d$  independentemente, o que é dado simplesmente pelo produto da probabilidade de se observar cada um deles  $P(x|C) = P(x_1, \dots, x_d|C) = \prod_i P(x_i|C)$ . Esta suposição ingênua é a base do **classificador Naive Bayes**, dado na equação 3.19.

$$\hat{f}(x) \leftarrow \arg \max_{C \in \mathcal{C}} P(C) \prod_i P(x_i|C) \quad (3.19)$$

Para atributos discretos e finitos, a probabilidade  $P(x_i|C)$  pode ser estimada pela fração  $x_i(C)/m_C$ , onde  $x_i(C)$  é o número de vezes em que o atributo  $x_i$  aparece em  $C$ . Para atributos contínuos, deve-se assumir uma função de probabilidade a partir dos quais eles são obtidos. Assumindo uma distribuição normal, estima-se a média  $\mu_i(C)$  e desvio padrão  $\sigma_i(C)$  como segue, e a estimativa de probabilidade é dada em 3.20.

$$\begin{aligned} \mu_i(C) &= \frac{1}{m_C} \sum_{y \in C} y_i \\ \sigma_i(C) &= \sqrt{\frac{1}{m_C} \sum_{y \in C} (y_i - \mu_i(C))^2} \\ P(x_i|C) &= \frac{1}{\sqrt{2\pi\sigma_i(C)^2}} \exp\left(-\frac{1}{2\sigma_i(C)^2} (x_i - \mu_i(C))^2\right) \end{aligned} \quad (3.20)$$

A Figura 16 ilustra este método de classificação. Os atributos das instâncias de dados são suas coordenadas  $x$  e  $y$ , e a Figura 16a mostra a distribuição gaussiana estimada ao assumir que essas variáveis são independentes. A definição 3.19 leva a uma classificação discreta, com a fronteira de decisão do método mostrada na Figura 16b. Contudo, uma vez que cada classe fornece um valor  $v_C = P(C) \prod_i P(x_i|C)$  sobre conter a instância  $x$ , podemos retornar as probabilidades  $p_C = v_C / \sum_C v_C$  da instância pertencer a cada classe, em uma classificação nebulosa (*fuzzy*), como ilustrado na Figura 16c.

Em nossa pesquisa, as instâncias não são definidas por tuplas de atributos, mas simplesmente pelas distâncias entre si. Para utilizar o classificador Naive Bayes, constrói-se para cada instância  $x$  um vetor de características  $u$  composto pelas distâncias de  $x$  para cada classe, isto é,  $u_i = d(x, C_i)$ , para uma dada distância  $d$  entre ponto e classe. Neste caso, as medidas  $\mu_i(C)$  e  $\sigma_i(C)$  tornam-se a média e variância da distribuição de distâncias entre classes  $D(C, C_i) = (d(x, C_i) | x \in C)$ . Isto apresenta problemas ao utilizar a métrica  $d_{\min}$ , uma vez que  $D(C, C) = (0, 0, \dots)$ ,  $\mu = \sigma = 0$ , e a distribuição normal torna-se uma função delta. Neste caso, exclui-se  $x$  da classe ao calcular a distância para si mesma, isto é,  $D^*(C, C) = (d_{\min}(x, C \setminus \{x\}) | x \in C)$ .

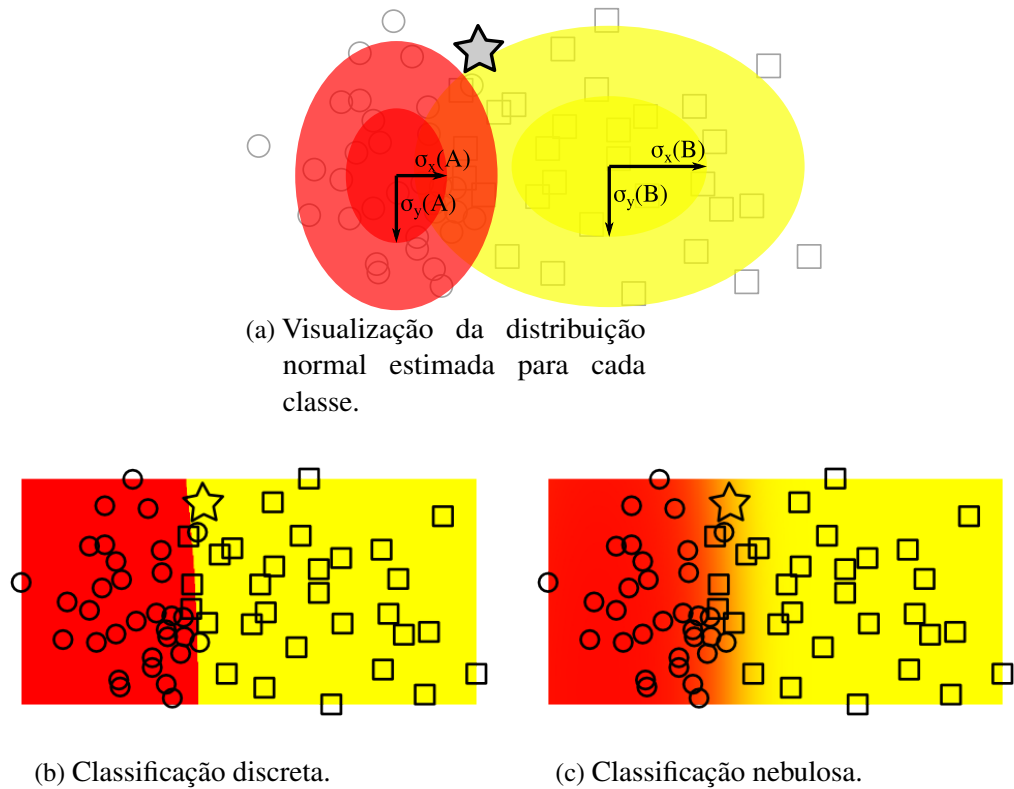


Figura 16 – Classificador Naive Bayes.

### 3.3.2 Árvore de quarteto

Uma deficiência do classificador Naive Bayes é exigir que se suponha uma função de distribuição de probabilidades para avaliar  $P(x_i|C)$ . A derivação apresentada para a distribuição normal pode não ser uma boa escolha para o problema em vista, requerendo, talvez, empregar uma de diversas técnicas de estimação de densidade não-paramétrica.

Intuitivamente, busca-se uma medida da proximidade e espalhamento entre classes, incluindo de uma classe para si mesma. Se um elemento é tanto “próximo” quanto “dentro da largura” de uma dada classe, mas não de outras, podemos ter confiança de que este elemento pertence à classe.

Uma estratégia desenvolvida por [Delbem \(2012\)](#) busca identificar se uma instância  $x$  pode ser classificada dentro de uma classe  $U$  ao tentar agrupá-la com a própria classe e com seu complemento  $\bar{U} = V$ . Para se certificar de que o agrupamento tem mérito, um elemento  $u$  é removido de  $U$ , chamado de *outgroup*, e “compete” com  $x$  para se agrupar com  $U$ .

O propósito do *outgroup* é prover uma medida do “espalhamento” de uma classe no espaço de características com a distância  $d_{uU}$ , e sua “sobreposição” em outras classes com a distância  $d_{uV}$ . A escolha do *outgroup* pode ser realizada de diversas maneiras:

- Seleção aleatória;

- Elemento mais próximo do grupo, isto é,  $\arg \min_u d_{uU}$ ;
- Elemento mediano, isto é, o elemento cuja distância é a mediana da distribuição de distâncias dentro da classe  $D^*(U, U)$ .

A técnica utiliza, na realidade, a classe  $U^* = U \setminus \{u\}$ , isto é, o conjunto  $U$  sem o *outgroup*. Isto é necessário pois a distância  $d(u, U)$  pode ser artificialmente pequena se  $u$  estiver contido em  $U$ , ou zero caso  $d = d_{\min}$ . Uma vez computadas todas as distâncias entre os elementos  $x, u, U^*$  e  $V$ , constrói-se uma árvore filogenética com o método de Junção de Vizinhos, como mostrado na Figura 17.

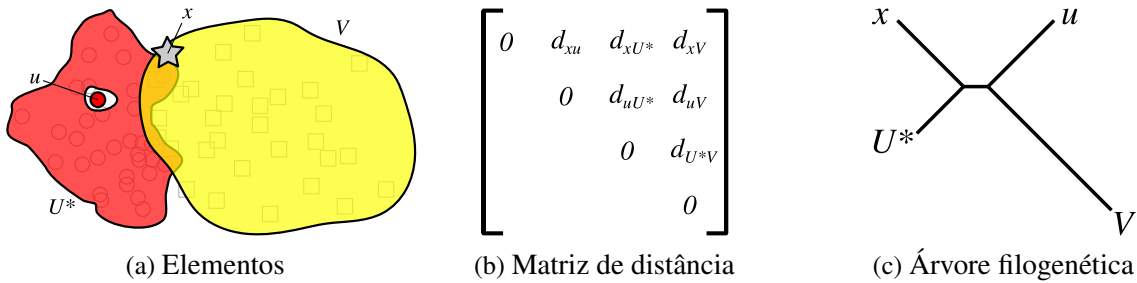


Figura 17 – Sequência de classificação, testando se  $x \in U$

Ao invés de realizar um passo de detecção de comunidades, que tem pouco significado para uma árvore tão pequena, considera-se apenas a divisão entre os elementos. Existem quatro possíveis resultados para este procedimento, correspondentes a cada uma das possíveis **árvores de quarteto** ilustradas na Figura 18:

- Divisão  $(xU|uV)$ :  $x$  está mais próximo de  $U$  que  $u$ . Considera-se isso um resultado **positivo forte** de que  $x$  pertence a  $U$  (figura 18a);
- Divisão  $(xV|uU)$ :  $u$  está mais próximo de  $U$  que  $x$ . Considera-se isso um resultado **negativo fraco** de que  $x$  pertence a  $U$  (figura 18b);
- Divisão  $(xu|UV)$ : os elementos  $x$  e  $u$  estão mais próximos entre si do que a cada uma das classes. Considera-se isso um resultado **neutro** (figura 18c);
- Sem divisão: todos os elementos estão equidistantes entre si e a árvore é degenerada. Considera-se isso também um resultado **neutro** (figura 18d).



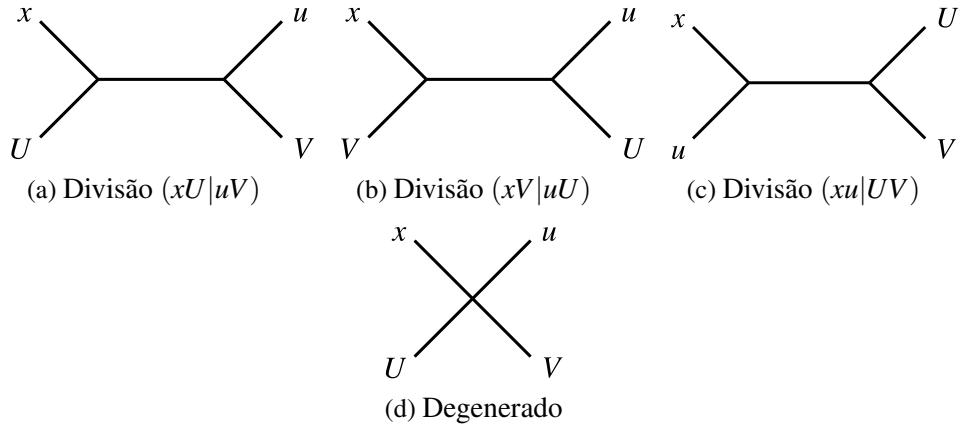


Figura 18 – Possíveis árvores de quarteto rotuladas

Realizando este procedimento para todas as classes em  $\mathcal{C}$ , obtém-se um vetor de resultados positivos-neutros-negativos para cada instância. Pode-se então classificar um elemento com o seguinte processo de decisão:

1. Se existe um único resultado positivo para a classe  $C$ , emita  $C$ ;
2. Senão, se existem múltiplos resultados positivos, ou apenas resultados negativos, emita “erro”;
3. Senão, se existe um único resultado neutro para a classe  $C$  e resultados negativos para todas as outras classes, emita  $C$  por eliminação;
4. Senão, existe mais de um resultado neutro, emita a lista de classes possíveis ou “sem classificação”.

Pode-se medir a confiança em uma classificação contando o número de resultados positivos e negativos. Uma classificação bem sucedida contendo muitos contra-exemplos negativos fornece uma conclusão forte, enquanto uma com diversos contra-exemplos neutros é uma conclusão fraca. Pode-se também atribuir uma pontuação para classificações mal sucedidas. Seja  $n^+$  o número de resultados positivos,  $n^-$  o número de resultados negativos e  $n = |\mathcal{C}|$  o número de resultados. O *score*  $s$  da classificação é dado pela Equação 3.21. Um exemplo para todos os possíveis resultados com  $n = 4$  é mostrado na Figura 19.

$$s = \begin{cases} (n^+ + n^-)/n & \text{se } n^+ \leq 1 \text{ e } n^- < n \\ -|n^+ - n^-|/n & \text{caso contrário} \end{cases} \quad (3.21)$$

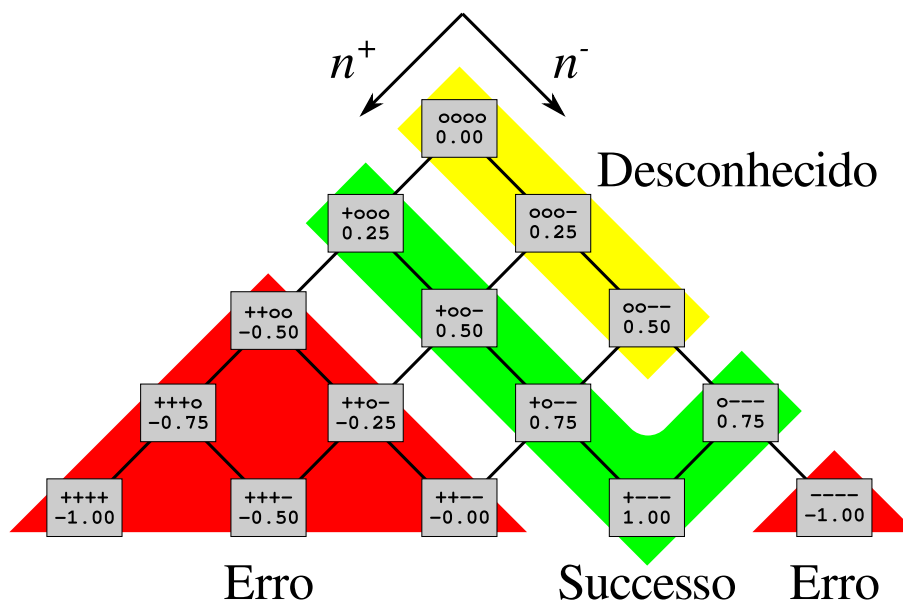


Figura 19 – Trelça mostrando os possíveis resultados com 4 classes. Resultados positivos, negativos e neutros são mostrados por, respectivamente, '+', '-' e 'o'. O *score* é mostrada abaixo de cada combinação.

---

## DESENVOLVIMENTO

---

Neste trabalho, desenvolvemos uma biblioteca abrangente de ferramentas para realizar experimentos com a metodologia DAMICORE, contendo também utilitários independentes que podem ser de interesse de qualquer pesquisador. Cada ferramenta foi implementada como um módulo em Python com interface por linha de comando, de modo que pode ser utilizada diretamente por um usuário final, integrada com programas na mesma linguagem, ou vista como um binário isolado que pode ser invocado com chamadas ao sistema ou *scripts* de *shell*. Este capítulo detalha a motivação para seu desenvolvimento, a arquitetura extensível do sistema, e suas principais contribuições técnicas.

### 4.1 Motivação

A metodologia DAMICORE foi implementada inicialmente por Delbem utilizando-se de componentes já disponíveis: a implementação da NCD disponibilizada pelos seus autores originais no *toolkit* CompLearn, a implementação de Junção de Vizinhos disponibilizada no software PHYLIP e uma implementação do algoritmo *Fast Newman* desenvolvida por Soares em C (CILIBRASI, 2005; PLOTREE; PLOTGRAM, 1989). Tal implementação já permitia a composição de ferramentas simplesmente pela combinação de entradas e saídas textuais, embora o formato utilizado por cada ferramenta fosse dissimilar: enquanto o binário *ncd* da CompLearn seja configurável por *flags* pela linha de comando, o PHYLIP requer entrada interativa, simulada com uma sequência pela entrada padrão, e a implementação de *Fast Newman* não permite nenhuma configuração. Ainda, são necessários *scripts* para conversão entre o formato de saída de uma componente e outra, uma vez que a única interface entre elas é textual.

Outra deficiência desta implementação é não permitir modificações nos próprios algoritmos utilizados em cada componente: por exemplo, utilizar uma função de pareamento ou compressor diferentes para a NCD ou diferentes algoritmos de detecção de comunidades. Dado que um objetivo inicial deste trabalho era estudar a propriedade e impacto de cada componente

no resultado de agrupamento final, era necessário uma arquitetura mais extensível que permitisse um controle mais granular das propriedades de cada etapa.

Os principais aspectos considerados para a escolha da linguagem foram desempenho, manutenibilidade e extensibilidade, que são justificados a seguir.

#### 4.1.1 *Desempenho*

A escolha padrão em computação científica é realizar implementações diretas em C ou Fortran para obter o máximo desempenho possível em todos os aspectos do programa, evitando assim possíveis gargalos produzidos pelas abstrações da linguagem ou custos de *runtime* em linguagens interpretadas. Contudo, vêm aumentando o interesse em utilizar linguagens de mais alto nível como Python, Lua e Java para a coordenação geral de um programa, delegando apenas as tarefas mais custosas como cálculos matriciais para bibliotecas otimizadas. De fato, Python possui crescente utilização pela comunidade científica graças a bibliotecas numéricas eficientes e com alto nível de abstração, como *numpy*, *scipy*, *matplotlib* e *pandas* (JONES *et al.*, 2001–). O fato de ser uma linguagem dinâmica, flexível e de rápido desenvolvimento atrai pesquisadores em análise exploratória de dados, que desejam realizar experimentos em sequência, iterando rapidamente na exploração de um conjunto de dados ou de parâmetros de modelos.

Neste trabalho, em específico, foi detectado que o gargalo de desempenho encontra-se na etapa de cálculo da matriz de distâncias, sendo a compressão realizada por bibliotecas de ligação dinâmica ou binários externos. Portanto, a sobrecarga de coordenação do programa em Python seria pequena em comparação com o custo computacional de compressão, idêntico independentemente da linguagem escolhida. Tal expectativa foi confirmada por experimentos, descritos na seção 5.2.

#### 4.1.2 *Extensibilidade e manutenibilidade*

Python, por ser uma linguagem dinâmica, é adequada para desenvolvimento rápido e iterativo, podendo mesmo dispensar a etapa de compilação. Um modelo de experimentação popular consiste em manter uma sessão interativa onde o pesquisador manipula, depura e aprimora seu código enquanto mantém objetos em memória para serem analisados e testados. Este modelo pode ser experimentado não só pelo console como também por ferramentas como IPython Notebook, que produz um relatório com gráficos, imagens e tabelas em conjunto com o código que os produziu (PÉREZ; GRANGER, 2007).

Python impõe poucas restrições aos programadores, permitindo escolher o paradigma mais adequado para se modelar o domínio do problema. Neste trabalho optamos por um modelo orientado a objetos, que permitiu encapsular diversos aspectos díspares de implementações sob uma interface comum, simplificando as dependências entre partes não relacionadas do sistema.

Uma preocupação inicial foi a experiência de futuros mantenedores e usuários da ferramenta com a linguagem. A princípio, os primeiros usuários desta ferramenta serão outros pesquisadores com conhecimento para analisar o código-fonte e realizar modificações que sejam necessárias para seus experimentos e criem ferramentas. A linguagem deve ser, portanto, de fácil compreensão e aprendizado; e de fato, embora a maioria dos pesquisadores sejam proficientes em C, a sintaxe de Python possui as mesmas raízes de outras linguagens procedurais e de orientação a objeto, buscando uma estrutura similar a formatos de pseudo-código. Algumas capacidades únicas e sem analogias em linguagens mais populares, como *monkey-patching* e idiomas funcionais, foram evitadas na implementação deste trabalho preferindo a legibilidade acima de ser idiomático.

Para aumentar a manutenibilidade, estabeleceram-se algumas convenções de interface entre as componentes, permitindo que qualquer ferramenta receba/envie entrada e saída de arquivos quaisquer, ou das entradas e saída padrão; que todas utilizem um formato consistente de configuração por *flags* de linha de comando; e que as entradas e saídas devem ser no formato CSV, exceto quando for necessário para interoperar com outros programas.

## 4.2 Arquitetura

A arquitetura do sistema faz uso de orientação a objetos para garantir uma interface consistente para entidades, de modo a desacoplar conceitos e permitir a troca de componentes sem afetar a funcionalidade. As interfaces básicas são:

- **DataSource**: representa uma fonte de dados, podendo fornecer tanto uma *string* de bytes como um nome de arquivo contendo os dados;
- **DataSourceFactory**: representa um conjunto de dados capaz de produzir **DataSources**.
- **Compressor**: representa um compressor capaz de comprimir os bytes produzidos por um **DataSource**.

O diagrama de classes dos compressores está apresentado na Figura 20. A necessidade de se modelar uma fonte de dados de modo abstrato se deve a alguns compressores esperarem uma *string* de bytes como entrada (especificamente, aqueles que se utilizam das bibliotecas *zlib* e *bz2*) enquanto os demais esperam um nome de arquivo em disco. Assim, uma mesma fonte de dados pode ser utilizada por qualquer tipo de compressor, que pode ou ler uma *string* diretamente, ou invocar um binário que lerá de um arquivo.

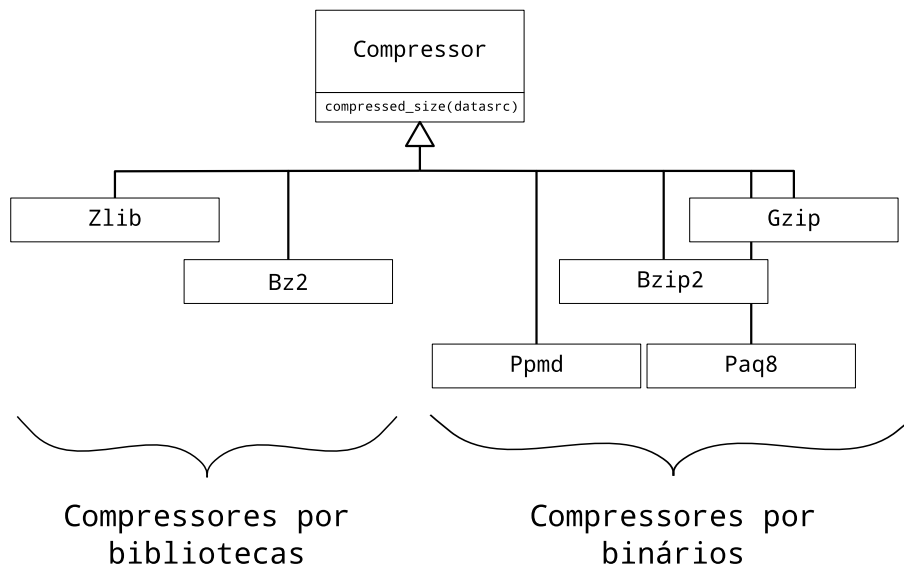


Figura 20 – Diagrama da hierarquia de classes de Compressor.

A abstração de uma fonte de dados é necessária não somente para compatibilizar com qualquer compressor, mas também porque os próprios dados podem ser lidos de diferentes formas. A Figura 21 apresenta o diagrama de classes do DataSource, onde se vê que dentre suas subclasses se encontram String, que contém uma sequência de bytes armazenadas em memória, e um Filename, que contém apenas o nome de um arquivo em disco. Caso um cliente invoque String.get\_filename(), o objeto cria um arquivo temporário contendo o conteúdo da *string* e retorna seu nome, e o destrói na chamada de String.close(). Do outro lado, caso um cliente invoque Filename.get\_string(), o objeto então irá abrir e ler o arquivo para retornar apenas os bytes. Em ambos os casos, a implementação garante que isso só ocorrerá uma vez, caso tal método “oposto” seja invocado.

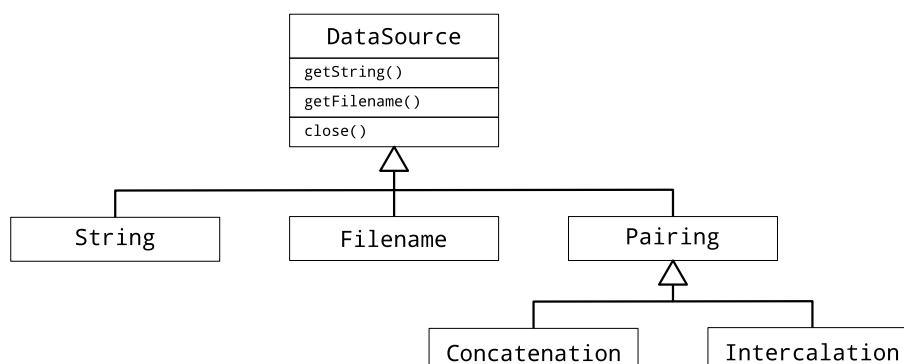


Figura 21 – Diagrama da hierarquia de classes de DataSource.

Com isso, a etapa de compressão será mais eficiente se existir uma compatibilidade entre o compressor e a origem do DataSource: se o compressor Zlib for utilizado com uma String, não será necessário criar arquivos temporários, e se o compressor Gzip for utilizado com um

Filename, não será necessário abrir o arquivo e ler diretamente para a memória do programa, delegando ao binário realizar isso.

A subclasse Pairing de DataSource representa uma função de pareamento utilizada na NCD: Concatenation realiza a concatenação dos bytes de dois DataSources, e Intercalation realiza sua intercalação, para um tamanho de bloco especificado na construção do objeto. Esta implementação também é mais eficiente caso os DataSources sejam compatíveis, podendo realizar operações em memória para duas Strings e manipulação de blocos com chamadas ao sistema com dois Filenames. Estas operações são sempre realizadas de forma “atrasada” (*lazy*), para evitar a produção de muitos arquivos intermediários e/ou sequências de bytes longas em memória.

Então, pode-se definir um objeto que realiza a NCD entre dois DataSources, definida na Equação 2.3, da seguinte maneira:

---

```

1 class Ncd(object):
2     # Construtor, recebendo um compressor e uma função de pareamento
3     def __init__(self, compressor, pairing):
4         self.compressor = compressor
5         self.pairing = pairing
6
7     # x e y são do tipo DataSource
8     def compute(self, x, y):
9         # Calcula o comprimento comprimido de cada DataSource
10        zx = self.compressor.compressed_size(x)
11        zy = self.compressor.compressed_size(y)
12
13        # Calcula o comprimento comprimido do pareamento de DataSources
14        xy = self.pairing.pair(x, y)
15        zxy = self.compressor.compressed_size(xy)
16        xy.close()
17
18        # Calcula a NCD dos DataSources
19        return float(zxy - min(zx, zy)) / max(zx, zy)
20
21 # Exemplo de uso
22 x = String("aaabaaaabbabbabbbbbbbaabbaaa"*300)
23 y = Filename("./path/to/file.dat")
24 compressor = Zlib(level=9)
25 pairing = Intercalation(block_size=1024)
26 ncd = Ncd(compressor, pairing)
27 print ncd.compute(x, y)

```

---

Finalmente, a terceira abstração de interesse é a de DataSourceFactory, cujo diagrama de classes é apresentado na Figura 22. Esta classe implementa o padrão de projeto de Fábrica, encarregado de produzir outras instâncias de um dado tipo. No caso, as fábricas pro-

duzem uma sequência de *DataSources* relacionados, representando um conjunto de dados. A *FileLinesFactory* modela um conjunto de dados dado por um arquivo, onde cada linha consiste em uma instância; a *DirectoryFactory* modela um conjunto de arquivos em um mesmo diretório. As *CollectionFactory* modelam uma coleção de outras fábricas, permitindo, por exemplo, modelar um conjunto de dados único mas organizado em subdiretórios. Esta flexibilidade permite que um usuário final pode configurar com facilidade a ferramenta para ler seus dados da maneira mais próxima em que eles estão armazenados.

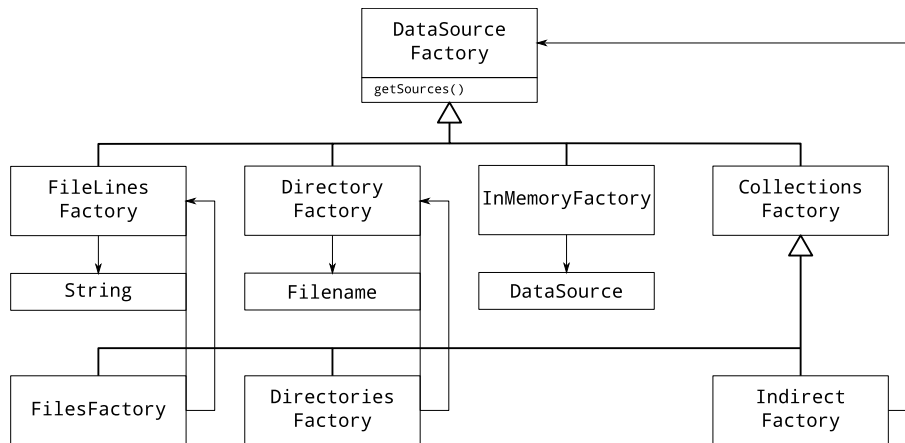


Figura 22 – Diagrama da hierarquia de classes de DataSourceFactory.

### 4.3 Bibliotecas/Ferramentas

Como descrito, as principais bibliotecas desenvolvidas possuem também uma interface por linha de comando, de modo que podem ser utilizadas como *scripts* de *shell* ou invocadas pelo interpretador de Python do sistema. As ferramentas desenvolvidas são:

- `ncd2.py`: Realiza o cálculo da NCD em um conjunto de instâncias, ou entre dois conjuntos de forma cruzada. Funcionalidades:
  - Aceita como entrada conjuntos expressos como linhas em um arquivo, arquivos em um diretório, ou quaisquer mistura destes casos.
  - Permite realizar o cálculo da matriz de distâncias completa ou apenas a porção triangular superior para maior eficiência.
  - Permite escolher dentre cinco opções de compressores, duas opções de pareamento, e configurações diversas para cada compressor.
- `tree_simplification.py`: Realiza a etapa de simplificação em árvore, implementando o algoritmo de Junção de Vizinhos. Funcionalidades:
  - Aceita como entrada matrizes de distância em formato CSV, PHYLIP (completa ou triangular inferior) e no formato emitido pela `ncd` da CompLearn.



- Pode produzir como saída árvores em formato Newick ou DOT.
- `clustering.py`: Realiza o agrupamento de vértices em um grafo utilizando algoritmos de detecção de comunidades. Funcionalidades:
  - Aceita como entrada grafos em todos os formatos suportados pela biblioteca `igraph`, assim como árvores no formato Newick.
  - Fornece seis opções de algoritmos de agrupamento, incluindo dois específicos para árvores desenvolvidos e descritos na seção 2.3.4.
  - Permite especificar o número de grupos desejados, ou escolher aquele que maximiza a modularidade.
  - Produz como saída um arquivo CSV com o mapeamento instância-grupo, e pode também imprimir uma imagem da árvore com nós e seus grupos anotados.
- `partition.py`: Calcula índices de validação externa para um dado agrupamento obtido contra um esperado. Ferramenta independente e útil para quaisquer pesquisadores. Funcionalidades:
  - Pode-se especificar um conjunto de índices de interesse, ou simplesmente calcular todos os aplicáveis, inclusive aqueles dedicados a partições binárias caso a matriz seja 2x2.
  - Aceita como entrada arquivos CSV de agrupamento ou classificação, permitindo especificar um limiar de confiança para classes onde a classificação é nebulosa.
- `dataset.py`: Realiza pré-processamentos em um conjunto de dados anotado, gerando novos conjuntos para testes. Ferramenta independente e útil para quaisquer pesquisadores. Funcionalidades:
  - Permite realizar a divisão, amostragem e particionamento de um conjunto de dados mantendo a proporção de classes inicial.
  - Permite concatenar as entradas de referência para produzir um arquivo agregado representando a totalidade de uma classe.
- `damicore.py`: Realiza todo o fluxo de métrica-simplificação-agrupamento-validação. Funcionalidades:
  - Expõe todas as opções de cada componente individual em um único comando.
  - Pode realizar as saídas de todas as etapas intermediárias, aumentando a compreensão de como o fluxo se realiza e permitindo depurar eventuais falhas nos algoritmos implementados.
- `classification.py`: Realiza a classificação de um conjunto de instâncias, dado um grupo de referência anotado. Funcionalidades:

- Expõe todas as opções de compressores e pré-processamento de conjunto de dados.
- Oferece três opções de classificadores.

## RESULTADOS

Para avaliar a ferramenta desenvolvida, foi utilizada uma diversidade de conjuntos de dados com diferentes distribuições de tamanho e classes, como mostrado na Tabela 7.

Tabela 7 – Conjuntos de dados utilizados

Conjunto de dados	Instâncias	Tipo	Classes	Quartis de comprimento (bytes)			
				25%	50%	90%	95%
LingSpam-Bare	2893	Spam	2	997	2066	7413	10428
LingSpam-Proc	2893	Spam	2	769	1624	5805	8388
SMSSpam	5574	Spam	2	36	62	156	161
EuroGOV	1500	Línguas	10	3373	5619	38085	61029
TCL	3174	Línguas	60	991	1820	5031	6780
Wikipedia	4963	Línguas	67	139	512	3332	5527
Binaries	80	Binários	2	8473	15616	182671	573564

Os conjuntos LingSpam-Bare e LingSpam-Proc consistem em e-mails de uma lista de discussões pública de linguistas, e spam recebido pelos autores ([ANDROUTSOPOULOS et al., 2000](#)). A versão Bare contém os e-mails sem processamento, enquanto a versão LemmStop foi processado para lematização e remoção de *stop words*. O conjunto SMSSpam consiste em mensagens de SMS, que por padrão são muito mais curtas que outros tipos de texto e apresentam outros desafios para classificação. Tal conjunto foi organizado em [Almeida, Hidalgo e Silva \(2013\)](#).

O conjunto EuroGOV consiste em uma amostra de documentos da coleção EuroGOV ([SIGURBJÖRNSSON; KAMPS; RIJKE, 2006](#)), distribuído entre 10 línguas ocidentais. O conjunto TCL, organizado pelo Thai Computational Linguistics Laboratory em 2005, é mais diversificado, contendo 60 línguas de todo o mundo em doze codificações diferentes. O conjunto Wikipedia consiste em uma amostra de páginas da Wikipedia dentre línguas contendo mais de 1000 páginas, e mantendo a distribuição de línguas encontrada na Web. Todos estes conjuntos foram organizados e analisados em [Baldwin e Lui \(2010\)](#).

O conjunto *Binaries* consiste de programas executáveis compilados para a plataforma x86 de 64 bits para o sistema operacional GNU/Linux. Tais programas incluem utilitários do sistema operacional, como *ls*, *find* e *cat*; aplicativos no nível de sistemas de arquivos, como *dd* e *shred*; e vários outros aplicativos, como um solucionador de torres de Hanoi, compressores, e calculadoras de *checksum*. Tais binários foram classificados em termos de sua característica de desempenho como intensos em processamento (*CPU-intensive*) ou entrada/saída (*I/O-intensive*) e analisados em [Pinto \(2013\)](#).

A distribuição de classes nestes grupos é variável, como ilustrado na Figura 23. O conjunto *Binaries* possui igual proporção de programas *CPU-intensive* e *I/O-intensive*; os conjuntos *SMSSpam* e *LingSpam* possuem muito mais exemplos de mensagens legítimas ("ham") do que spam; o conjunto *EuroGOV* possui uma distribuição quase uniforme de línguas, enquanto os conjuntos *TCL* e *Wikipedia* possuem uma distribuição de cauda longa, mais próxima da distribuição de páginas Web.

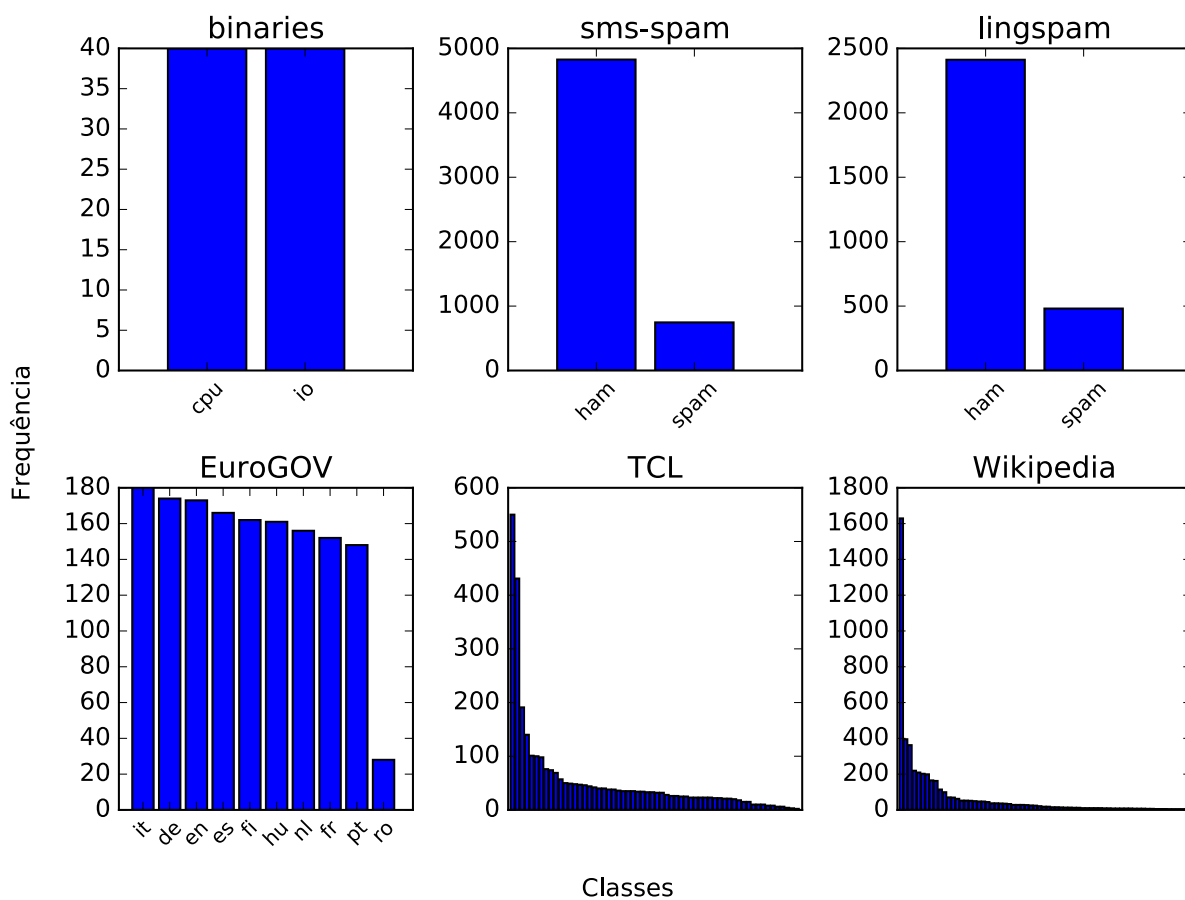


Figura 23 – Distribuição de classes nos conjuntos de dados analisados. Não foi possível incluir a legenda para cada classe dos conjuntos *TCL* e *Wikipedia* devido à grande quantidade.

## 5.1 Aplicação de intercalação como função de pareamento

Dentre os conjuntos de dados estudados, o conjunto Binaries possui maior tamanho médio de arquivos, com mais de 40% possuindo tamanho maior que a janela máxima do compressor gzip. Neste contexto parece justificada a aplicação da função de pareamento de intercalação proposta na seção 2.1.4.3.

Foi investigado o efeito do tamanho de blocos na matriz de distâncias calculada e a efetividade resultante para a tarefa de agrupamento. Realizamos o cálculo da matriz de distâncias completa, contendo tanto os elementos da diagonal ( $NCD(x, x)$ ) como elementos teoricamente simétricos ( $NCD(x, y)$  e  $NCD(y, x)$ ). O tamanho dos blocos foi variado em uma função linear por partes que explora em detalhes tamanhos de blocos menores:

1. Em  $[16, 128)$  bytes com passos de 16 bytes (7 pontos);
2. Em  $[128, 1024)$  com passos de 128 bytes (7 pontos);
3. Em  $[1024, 32768)$  com passos de 1024 bytes (31 pontos);
4. Em  $[32640, 33664)$  com passos de 128 bytes (8 pontos).

Note que a última seção testa tamanhos de bloco maiores que a janela deslizante. Também foi calculada a matriz de distâncias utilizando concatenação simples para comparação da efetividade da intercalação. Foi utilizado apenas o compressor gzip para o cálculo da NCD, uma vez que apenas uma instância do conjunto possui comprimento superior à máxima janela do compressor bzip2 (900 KiB).

### 5.1.1 Distância de instância para si mesmo

A distância de um elemento para si mesmo (auto-distância) varia consideravelmente em função do tamanho de bloco, como ilustrado na Figura 24.

Nota-se que a concatenação é suficiente para produzir um valor próximo de zero caso a *string* seja menor que a janela do compressor (*randrw*, *vecsum*, *io\_thrash*). Uma vez que o tamanho supera a janela, a auto-distância com concatenação fica próxima a 1 (*fractal*, *genisoimage*, *mandelbulber*). Para todos os casos, a auto-distância com intercalação decresce em função do tamanho de bloco, atingindo um mínimo logo antes do tamanho da janela.

Este fenômeno pode ser compreendido em termos da representação de referências pelo compressor gzip. Suponha uma *string* de tamanho  $n$ , comprimido em blocos intercalados consigo mesmo de tamanho  $b$ . Idealmente, o compressor deveria comprimir um bloco, e então emitir uma referência com *offset* e comprimento  $(b, b)$  para indicar a repetição. Portanto, seriam necessárias  $n/b$  referências prévias, valor que pode ser minimizado ao maximizar  $b$ .

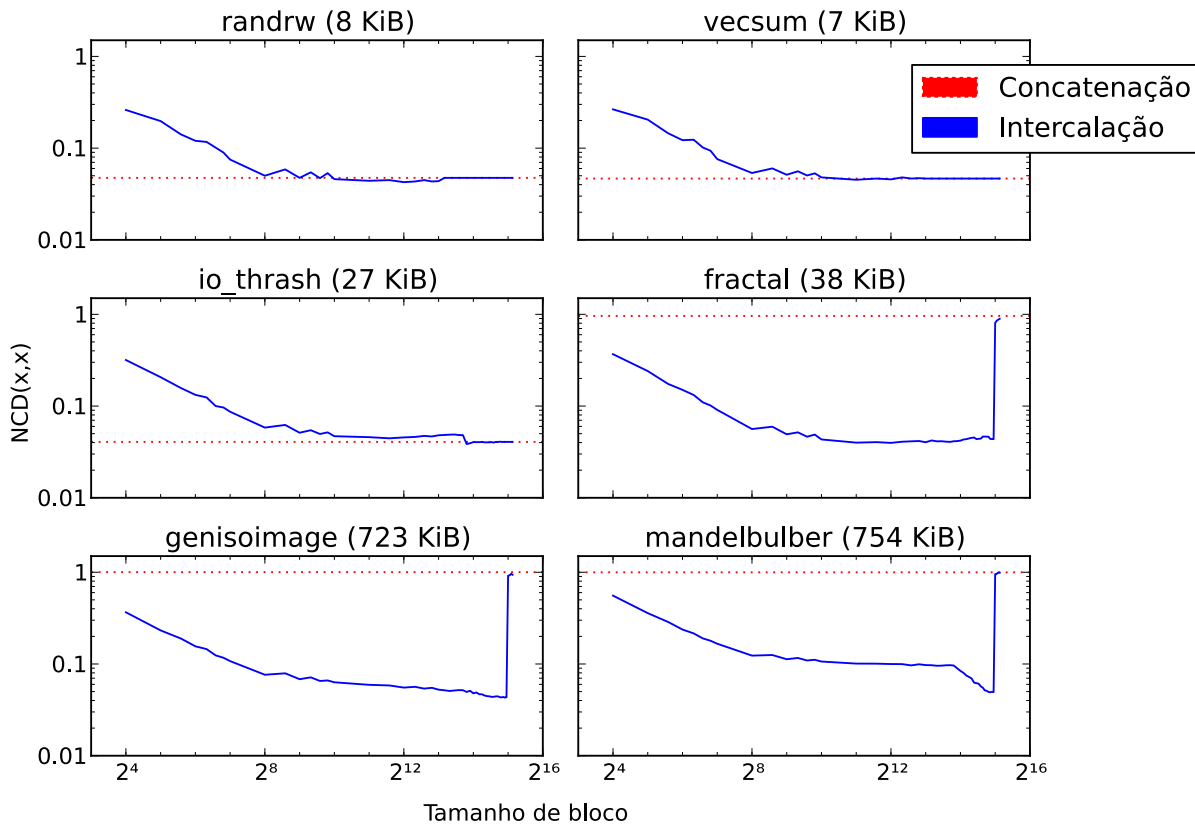


Figura 24 –  $NCD(x,x)$  em função do tamanho de bloco para algumas instâncias seleccionadas. A coluna da esquerda contém executáveis *I/O-intensive* e a coluna da direita executáveis *CPU-intensive*.

### 5.1.2 Impacto do tamanho de blocos sobre agrupamento

Apesar de o uso de intercalação como pareamento melhorar a auto-distância de instâncias grandes, ela também produz outros artefatos na matriz de distância. Por exemplo, em alguns casos a  $NCD(x,y)$  entre duas *strings* diferentes resulta maior que 1; isto é, ocorre uma expansão ao se aplicar a compressão e  $|Z(\langle x,y \rangle)| > |Z(x)| + |Z(y)|$ .

Tais artefatos podem comprometer a qualidade do agrupamento obtido com a ferramenta, de modo que foi analisado o resultado de agrupamento para diferentes tamanhos de bloco e comparados com as medidas de validação para partições binárias apresentadas na seção 3.1.4. O agrupamento foi realizado com as técnicas padrão da metodologia DAMICORE, utilizando Junção de Vizinhos para simplificação da matriz e o algoritmo de detecção de comunidades *Fast Newman* para obter os agrupamentos a partir da árvore filogenética. Este último foi configurado para executar o corte do dendrograma de modo a produzir dois grupos, ao invés de maximizar a modularidade. Ainda, foi adicionada uma etapa intermediária que normaliza os valores das arestas da árvore para remover arestas de tamanho negativo, que impedem a execução correta da detecção de comunidades.

Os resultados obtidos são apresentados na Figura 25, onde as medidas de precisão, sensibilidade,  $F_1$  e o coeficiente de correlação de Matthews  $M$  foram aplicadas sobre cada matriz de confusão obtida ao se comparar o agrupamento obtido com a referência organizada por Pinto

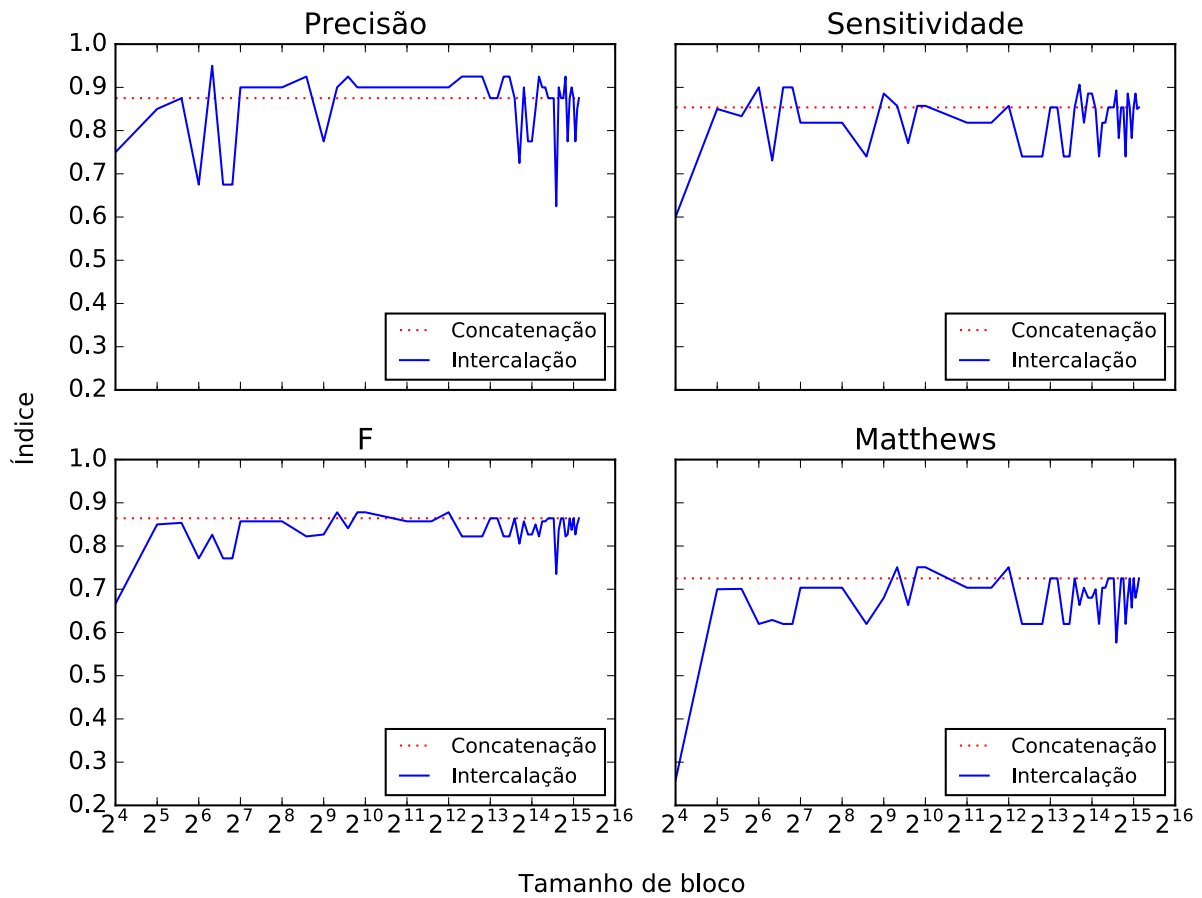


Figura 25 – Índices de qualidade de agrupamento, comparando o uso de concatenação e intercalação

(2013).

Inicialmente, observa-se que tamanhos de blocos muito pequenos ( $\approx 16$  bytes) comprometem a qualidade do agrupamento, tornando-se próximo a um agrupamento aleatório de acordo com o coeficiente de Matthews. Ainda, tais medidas são próximas àquelas obtidas com concatenação, embora na maioria dos casos sejam menores. A única exceção parece ser a medida de precisão para blocos entre 128 e 8192 bytes, que supera consistentemente a obtida com concatenação. Seriam necessários mais experimentos com um número maior de pontos para determinar a probabilidade de tal observação ser verdadeira.

## 5.2 Validação da implementação da NCD

Neste projeto foi implementado em Python o algoritmo da métrica NCD (CILIBRASI; VITÁNYI, 2005), o que permite uma maior flexibilidade para inclusão de novos compressores, assim como torna a integração com as outras ferramentas do *framework* mais fácil. Esta biblioteca específica é chamada `ncd2`. Os autores originais da métrica disponibilizam uma implementação no *toolkit* `CompLearn` (CILIBRASI, 2005), o que nos permitiu validar nossa implementação e

verificar seu desempenho<sup>1</sup>.

A implementação CompLearn possui algumas características diferentes da ferramenta desenvolvida, que demandaram compatibilização:

- A CompLearn permite apenas calcular uma matriz completa dos dados, o que pode dar origem a uma matriz não-simétrica, dado que  $NCD(A, B)$  não necessariamente é igual a  $NCD(B, A)$ . A *ncd2* permite assumir que a matriz é simétrica, e então calcular apenas a porção triangular superior da matriz para uma execução mais rápida.
- Ao calcular uma matriz completa, a *ncd2* calcula as distâncias na diagonal, isto é,  $NCD(A, A)$ , que em geral são não-nulas. Por sua vez, a CompLearn identifica que está calculando esta distância na diagonal e retorna apenas zero.
- A matriz de distância calculada pela *ncd2* é determinística, onde os elementos na entrada  $(i, j)$  correspondem sempre a  $NCD(D_i, D_j)$ . A CompLearn, por outro lado, permuta algumas das entradas de modo não-determinístico, de modo que a entrada  $(i, j)$  pode corresponder a  $NCD(D_i, D_j)$  ou a  $NCD(D_j, D_i)$ . Caso a hipótese de simetria não seja válida, isto introduz inconsistências entre execuções distintas da ferramenta.

Foi realizada uma série de experimentos com amostras de diferentes conjuntos de dados (Wikipedia, TCL e EuroGOV), com  $n = 200$ . Os conjuntos foram amostrados de modo a manter a proporção de classes inalterada. Para cada conjunto, a matriz de distâncias foi calculada utilizando os compressores *zlib* e *bz2*, com cada uma das implementações: *ncd2* parcial, *ncd2* completa e CompLearn. Cada execução foi repetida 30 vezes para obter uma amostra confiável do tempo de execução e poder comparar as médias com o teste *t* de Student. O tempo de execução observado para cada compressor está mostrado na Figura 26.

Dado que a *ncd2* completa e a CompLearn possuem implementações mais similares, é possível comparar o tempo de processamento em cada e o ganho no tempo de execução. Foram analisados também o *p*-valor do teste *t* de Student para decidir se as médias do tempo de execução são ou não distintas. Buscou-se comparar os valores nas matrizes, mas devido ao não-determinismo da CompLearn isto não foi satisfatório; para os conjuntos onde a saída foi determinística, notamos que a diferença é da ordem de  $10^{-6}$ , ou a precisão da saída da ferramenta em modo texto. Os resultados estão sumarizados na tabela 8.

Nota-se que para o conjunto Wikipedia e o conjunto TCL com *bz2*, a *ncd2* no modo completo é significativamente mais lenta que a CompLearn, enquanto para as outras combinações seu tempo de execução é equivalente ( $p > 0,05$ ). Nestes últimos, a média de tamanhos das instâncias no conjunto de dados é maior, o que leva a um maior tempo gasto em compressão e

<sup>1</sup> Realizamos o experimento utilizando a versão 1.1.7 da biblioteca CompLearn, em sistema operacional Linux (kernel 3.2.0-93-generic-pae) e distribuição Ubuntu 12.04. O processador utilizado é Intel® Core™ i7-2670QM CPU @ 2.20GHz × 8.



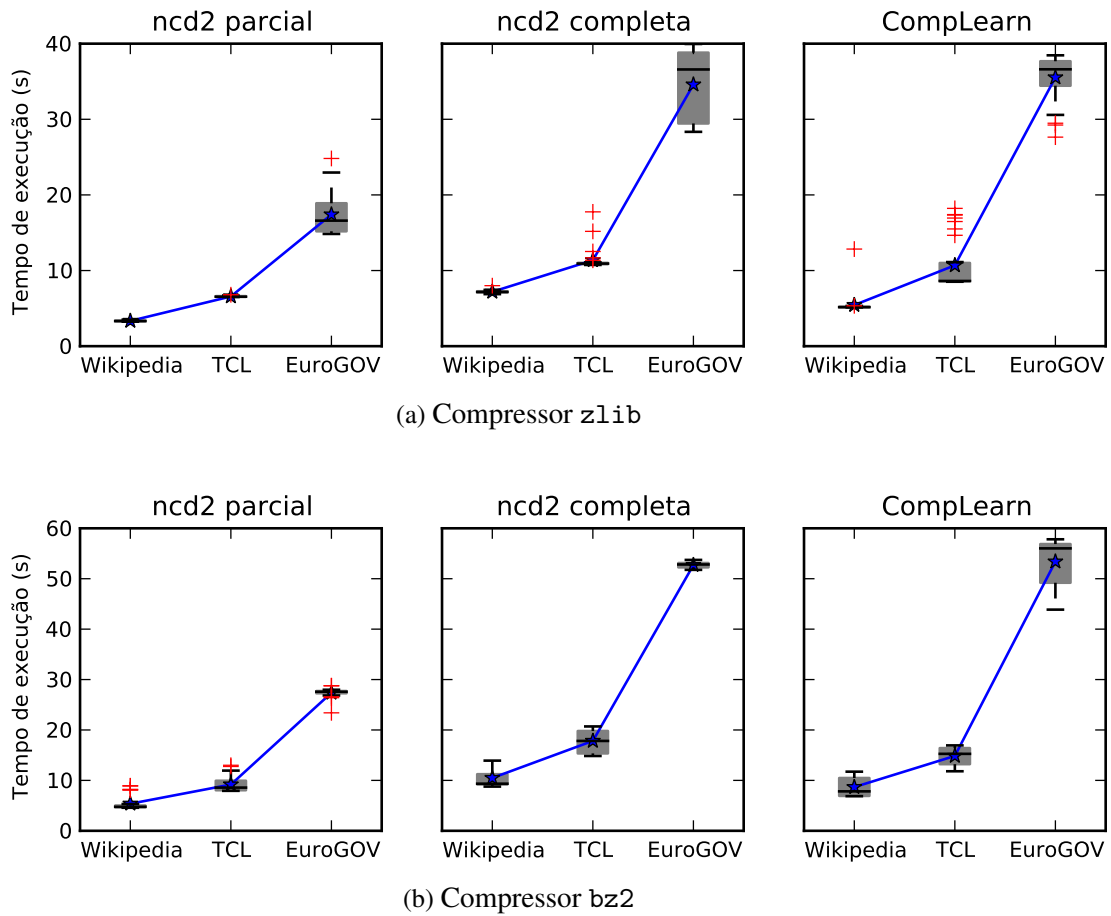


Figura 26 – Distribuição dos tempos de execução para cada combinação de compressor, conjunto de dados e implementação. As estrelas azuis indicam a média de cada distribuição.

Tabela 8 – Comparação entre as implementações ncd2 completa e CompLearn. Descrição das colunas no texto.

Compressor	Conjunto	Ganho	$p$ -valor
zlib	Wikipedia	-32%	$6,8 \times 10^{-9}$
	TCL	6,0%	0,33
	EuroGOV	2,7%	0,34
bz2	Wikipedia	-21%	$3,3 \times 10^{-4}$
	TCL	-20%	$1,1 \times 10^{-7}$
	EuroGOV	1,3%	0,41

menos em coordenação dos cálculos. Como ambas utilizam as mesmas bibliotecas de compressão, a sobrecarga da coordenação não é evidente.

Nos casos com instâncias de dados menores a diferença de desempenho das ferramentas é aparente, possivelmente devido à diferença de linguagens utilizadas. Contudo, nota-se que a implementação de ncd2 parcial leva metade do tempo que a ncd2 completa, como esperado; assim, se for possível assumir que a matriz é simétrica, pode-se obter um ganho significativo ao se percorrer apenas uma metade da matriz com a ncd2, o que não é possível com a CompLearn.

## 5.3 Classificação de spam

Spam é caracterizado como qualquer mensagem não solicitada, enviada para milhares de recipientes simultaneamente (ANDROUTSOPOULOS *et al.*, 2000). Tais mensagens representam um alto custo em produtividade perdida, e as estratégias de mitigação evoluíram conforme *spammers* desenvolvem sistemas sofisticados para burlar as proteções - atualmente, são necessários sistemas distribuídos de tempo real para uma visão atualizada dos IPs de onde se originam mensagens de spam, assim como a detecção de padrões constantemente em mutação para detectar mensagens indesejadas (KONG *et al.*, 2006).

Adicionalmente ao fenômeno em mensagens de email, a ocorrência de spam vem crescendo em mensagens de SMS. Com a diminuição significativa dos custos de tais mensagens, sua ocorrência cresceu de forma explosiva, representando cerca de 30% das mensagens recebidas em algumas regiões da Ásia (ALMEIDA; HIDALGO; SILVA, 2013). Modelos para classificação de SMS apresentam desafios com o pequeno tamanho das mensagens e a alta frequência de palavras resumidas e gírias.

Uma componente básica destes sistemas é detectar se novas mensagens apresentam padrão de spam. Tal categorização deve levar em conta que um falso positivo possui um custo maior do que falsos negativos - isto é, não mostrar ao usuário uma mensagem legítima possui um custo maior do que mostrar uma mensagem de spam. Esta é uma classificação binária, e sua validação é realizada com as métricas definidas na seção 3.1.4.

Foi investigado o desempenho dos compressores `zlib`, `bz2` e `ppmd`, em combinação com os classificadores  $k$ NN com  $k \in \{1, 5, 10\}$  e o classificador de quartetos proposto. Para a base `LingSpam`, foram utilizadas as versões `Bare` e `LemmStop` para avaliar o impacto do pré-processamento sobre a efetividade da ferramenta. Para os conjuntos analisados, os elementos foram particionados em 10% para treinamento e 90% de forma aleatória, mas mantendo a proporção de classes. Os resultados estão sumarizados na Tabela 9; como os resultados dos classificadores de  $k$ NN foram similares, apresentamos apenas para  $k = 5$ .

Nota-se que, para todas as combinações, o classificador  $k$ NN realiza a classificação em mais de 99% dos elementos, enquanto o classificador de quartetos retorna uma classe para apenas 35 – 55% dos elementos. Para os elementos classificados, em geral, os classificadores apresentam comportamento oposto com relação às métricas: enquanto o classificador  $k$ NN apresenta boa precisão, o classificador de quartetos apresenta boa sensibilidade. Tais características podem ser combinadas para obter uma classificação mais efetiva, por exemplo, empregando o  $k$ NN para localizar mensagens de spam com precisão, e então utilizar o classificador de quartetos sobre as mensagens remanescentes para detectar mensagens de spam perdidas.

Com relação aos compressores, o `ppmd` é consistentemente melhor para mensagens longas do conjunto `LingSpam`, mas não para as mensagens curtas, onde o compressor `zlib` se destaca. Isto se deve, possivelmente, à simplicidade deste e ao fato de que exige menos

Tabela 9 – Desempenho para classificação de spam. As métricas são apresentadas apenas para os elementos classificados. Para cada combinação de conjunto de dados e classificador, foi destacada a melhor métrica.

Conjunto de dados	Classificador	Compressor	Precisão	Sensitividade	Classificado
LingSpam-Bare	kNN	zlib	<b>99.3%</b>	67.6%	99.1%
		bz2	98.3%	68.4%	99.0%
		ppmd	96.7%	<b>76.4%</b>	99.4%
	Quartetos	zlib	87.8%	<b>96.7%</b>	43.2%
		bz2	80.4%	96.4%	37.9%
		ppmd	<b>95.2%</b>	96.3%	39.6%
LingSpam-Proc	kNN	zlib	97.9%	67.6%	99.3%
		bz2	97.9%	68.1%	98.9%
		ppmd	<b>98.7%</b>	<b>74.3%</b>	99.3%
	Quartetos	zlib	72.4%	<b>98.3%</b>	36.2%
		bz2	73.1%	92.3%	45.4%
		ppmd	<b>77.8%</b>	96.4%	38.8%
SMSSpam	kNN	zlib	98.9%	<b>82.7%</b>	99.8%
		bz2	99.8%	60.0%	99.7%
		ppmd	<b>100%</b>	74.4%	99.7%
	Quartetos	zlib	<b>78.2%</b>	<b>96.9%</b>	51.6%
		bz2	66.0%	94.7%	54.2%
		ppmd	69.2%	96.5%	53.1%

informações da entrada para construir um modelo efetivo.

Por fim, comparando-se os conjuntos LingSpam-Bare e LingSpam-Proc, nota-se que em geral a precisão da NCD diminui com o pré-processamento, enquanto a sensibilidade e a fração de elementos classificados não variam de forma significativa. Uma possível causa é que enquanto tais informações adicionam ruído para métricas tradicionais, elas são significativas para a NCD.

## 5.4 Identificação de línguas

A tarefa de identificação de línguas consiste em detectar a língua em que um certo documento foi escrito, o que possui importância para ferramentas de *crawling* da Web e outras ferramentas agregadoras de conteúdo (BALDWIN; LUI, 2010). A tarefa parece simples ao se considerar textos longos em línguas ocidentais, mas é complicada quando estendida para as diversas línguas presentes na Web, tanto por similaridades entre línguas com a mesma raiz e pela falta de *corpora* para treinamento. Textos curtos (por exemplo, comentários de usuários) também são de difícil reconhecimento.

Para avaliar a qualidade dos classificadores estudados, foram realizados experimentos sobre as bases de dados EuroGOV, TCL e Wikipedia, particionando 10% do conjunto de dados para treinamento e 90% para teste de forma aleatória, mas mantendo a proporção de classes. Empregamos todos os índices de validação descritos na seção 3.1, mas apenas o índice de

Tabela 10 – Desempenho para identificação de línguas. As métricas de distância de informação normalizada e índice de classificação são apresentadas apenas para os elementos classificados.

Conjunto de dados	Classificador	Compressor	$d$	$c$	Classificado
EuroGOV	$kNN$	zlib	0.08	0.96	94.9%
		bz2	0.09	0.96	95.9%
		ppmd	<b>0.06</b>	<b>0.97</b>	96.1%
	Quartetos	zlib	0.41	0.76	50.0%
		bz2	0.39	0.78	49.1%
		ppmd	<b>0.33</b>	<b>0.81</b>	49.3%
TCL	$kNN$	zlib	0.08	0.94	87.0%
		bz2	0.07	0.95	88.7%
		ppmd	<b>0.06</b>	<b>0.96</b>	88.6%
	Quartetos	zlib	0.17	0.85	48.8%
		bz2	0.10	0.91	50.2%
		ppmd	<b>0.02</b>	<b>0.98</b>	48.0%
Wikipedia	$kNN$	zlib	0.42	0.78	85.9%
		bz2	0.43	0.78	84.3%
		ppmd	<b>0.34</b>	<b>0.83</b>	85.2%
	Quartetos	zlib	0.32	0.66	4.25%
		bz2	0.38	0.60	2.37%
		ppmd	<b>0.25</b>	<b>0.77</b>	6.28%

classificação  $c$  (descrito na seção 3.1.1) e a distância de informação normalizada pela entropia conjunta  $d = d(U, V)/H(U, V)$ , descrita na seção 3.1.3, são apresentados. As métricas ajustadas para o acaso ficaram distantes de zero ( $> 0,37$ ), indicando que a classificação não é aleatória. A Tabela 10 apresenta os resultados para todas as combinações de conjunto, classificador e compressor utilizado.

Nota-se que o classificador ppmd apresenta o melhor desempenho em todos os casos, o que se justifica pela sua melhor taxa de compressão em textos longos. O classificador de quartetos apresenta um grande número de confusão, emitindo uma classificação para cerca de 50% dos elementos nos conjuntos EuroGOV e TCL e apenas 6% dos elementos no conjunto Wikipedia. O classificador  $kNN$  é capaz de classificar mais elementos, embora apresente resultados ligeiramente piores para os conjuntos mais diversificados (TCL e Wikipedia), embora ainda alcance um índice de classificação superior a 0.8.

---

## CONCLUSÃO

---

A metodologia DAMICORE está em ativo desenvolvimento, sendo estendida e aplicada em uma variedade de áreas de conhecimento como detecção de plágio e doenças em monoculturas, navegação de robôs, obtenção de clados em conjuntos de genoma, além do agrupamento e classificação apresentados neste trabalho. Ela é flexível e genérica para ser incorporada em fluxos de trabalho de pesquisadores e analistas de dados, e este trabalho colabora para a compreensão de seus mecanismos internos, classificação de conjuntos e para uma maior acessibilidade a quaisquer interessados.

### 6.1 Contribuições

A principal contribuição deste trabalho é a extensão da metodologia DAMICORE para tarefas de classificação, com o desenvolvimento e implementação de um método não-paramétrico que permite uma nova gama de aplicações que se beneficiem das capacidades únicas da NCD. Este método se baseia no trabalho de [Delbem \(2012\)](#) e inclui as seguintes extensões:

- Aplicável para um número arbitrário de classes;
- Cálculo da confiança na classificação;
- Processo de decisão capaz de rejeitar classificar uma instância se suas conclusões não forem confiáveis;
- Falha gradual, podendo emitir uma lista de classes prováveis quando não é capaz de escolher apenas uma.

Além da classificação de spam com boa sensibilidade apresentada neste trabalho, este método foi utilizado com sucesso por [Pinto \(2013\)](#), classificando binários executáveis com 100% de acurácia com a técnica de validação *leave-one-out*.

A segunda contribuição relevante é o próprio conjunto de ferramentas implementando a DAMICORE e seus utilitários. Pesquisadores terão disponível uma ferramenta de linha de comando capaz de produzir o agrupamento ou classificação de um conjunto de dados binários de forma automática, incluindo a etapa de validação contra um conjunto de referência. A ferramenta é eficiente e paralela, adequada para conjuntos de dados de tamanho moderado (1.000 a 10.000 instâncias) e capaz de rodar em computadores pessoais. Ainda, sua arquitetura permite incluir novos compressores, algoritmos de simplificação e detecção de comunidades, índices de validação ou mesmo novas métricas, que poderão ser necessários para tarefas especializadas. A ferramenta possui testes extensivos para garantir a corretude dos algoritmos e métricas implementadas, incluindo testes de compatibilidade com outras implementações.

Também, a função de pareamento de intercalação proposta é uma novidade na aplicação da NCD, uma vez que na literatura não existe exploração de funções alternativas à concatenação. Apesar de não ter sido obtida uma melhora significativa na aplicação de agrupamento, a utilização de tal técnica oferece garantias teóricas melhores para instâncias grandes, isto é, maiores que a janela do compressor. Isto remove um obstáculo para a utilização do compressor gzip, que em geral é preterido em favor do PPMd devido a possuir uma taxa de compressão cerca de 10% menor, mesmo sendo 5-10x mais rápido. O compressor PPMd não sofre com condições de janela, e gzip e bzi2 podem ser também aprimorados com o uso de intercalação.

Demais contribuições incluem:

- Investigação de uma função de pareamento alternativa na aplicação da NCD, com desempenho similar à concatenação e maior garantia teórica para instâncias grandes.
- Novos métodos de detecção de comunidades em árvores, com a definição de duas medidas de modularidade específicas para árvores;
- Detecção de *bugs* não-determinísticos na ferramenta CompLearn.

Por fim, espera-se que esta própria dissertação torne-se um recurso de estudos para quem desejar compreender as técnicas utilizadas pela metodologia DAMICORE, uma vez que abrange estudos em teoria da informação, compressores, filogenética, grafos e detecção de comunidades, classificadores e métodos de validação.

## 6.2 Trabalhos futuros

### 6.2.1 NCD

A métrica NCD é definida por dois parâmetros: a função de pareamento e o compressor utilizado. Uma modificação interessante proposta em [Cilibrasi e Vitanyi \(2007\)](#) propõe utilizar a quantidade de páginas retornadas pelo buscador Google como uma medida de frequência de um

termo; quanto menos frequente, maior seria a quantidade de bits necessários para codificá-lo. O pareamento consiste simplesmente em buscar ambos os termos em conjunto, o que produz uma medida de similaridade baseada em todo o conhecimento indexado pelo Google.

Este método, porém, é falho, uma vez que o próprio buscador admite que os resultados são apenas aproximados e podem retornar mais páginas ao se buscar a interseção de dois termos (SULLIVAN, 2011). Uma técnica mais robusta necessitaria da implementação de um indexador confiável, o qual dificilmente alcançaria a escala obtida pelo Google. Ainda assim, uma aplicação possível seria indexar a totalidade de uma área e obter relações significativos em um contexto: enquanto os termos "bomba" e "caixa" parecem pouco relacionados, no contexto de engenharia civil eles são similares por serem interpretados como "bomba d'água" e "caixa d'água".

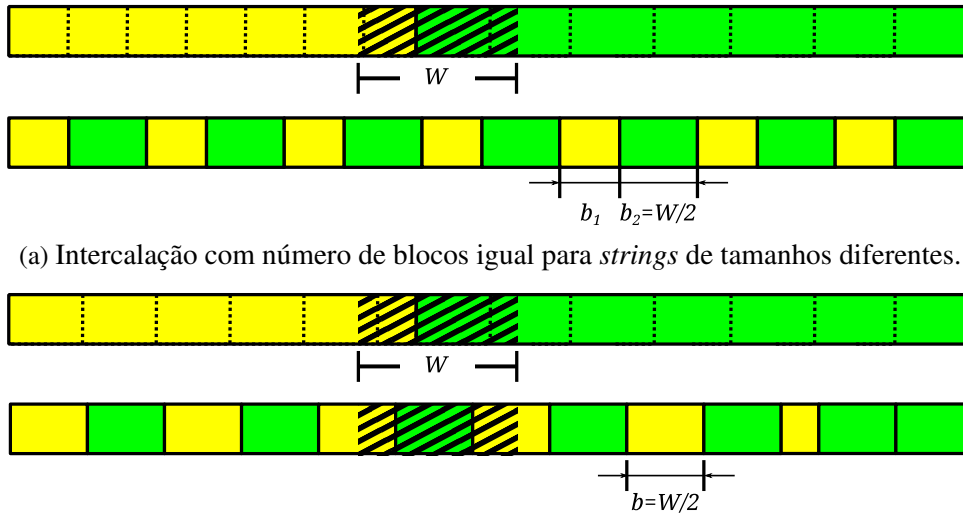
Outros trabalhos possíveis para a extensão da NCD incluem buscar novas funções de pareamento e aplicação de compressores especializados, como compressores de mensagens curtas, documentos XML, imagens de bitmap e séries temporais. Ainda, pouca pesquisa foi realizada em termos de utilizar a NCD com compressores com perdas, como os utilizados pelo formato JPEG em imagens e MP3 para músicas. Embora contrários à formulação teórica da métrica por não serem reversíveis, tais compressores fornecem uma transformação de dados quase imperceptível para seres humanos, de modo que podem revelar similaridades de acordo com a percepção humana ao invés de em termos de teoria da informação.

### 6.2.2 Intercalação

Como apresentada, a função de pareamento por intercalação de blocos é capaz de obter  $NCD(x, x) \approx 0$  mesmo para objetos maiores que a janela do compressor. Outras estratégias mais elaboradas podem ser construídas, como:

- particionar as *strings* em  $n$  blocos que garantam que cada um tenha tamanho menor ou igual a  $W$ . Pode-se forçar que cada *string* possua uma quantidade de blocos iguais (Figura 27a) ou não (Figura 27b). No primeiro caso, caso as *strings* possuam tamanhos diferentes, não existirá uma seção final composta de conteúdo de uma única *string*. No segundo, garante-se que quando a janela contiver conteúdo de ambas as *string*, será em igual proporção.
- particionar as *strings* em blocos de tamanhos desiguais de modo a minimizar as quebras de padrões internos. Uma variante do algoritmo de Rabin-Karp permite localizar uma série de subsequências idênticas em uma *string* grande de forma eficiente (KARP; RABIN, 1987). Isto permitiria encontrar trechos idênticos de uma *string* em si mesma ou na outra *string*. Utilizando-se uma função de hash menos eficiente, mas que mapeie *strings* similares para valores próximos, pode-se ainda buscar por padrões próximos mas não idênticos.

- comparação simultânea de mais que dois objetos, ou a métrica de um multiconjunto. Esta abordagem foi formalizada e explorada por [Cohen e Vitányi \(2012\)](#) com concatenação simples, e a intercalação pode ter maior poder para um compressor de mundo real.



(a) Intercalação com número de blocos igual para *strings* de tamanhos diferentes.

(b) Intercalação com tamanho de blocos igual para *strings* de tamanhos diferentes.

Figura 27 – Intercalação não parametrizada, com tamanho de bloco máximo igual a  $W/2$ .

### 6.2.3 Ferramenta DAMICORE

A ferramenta desenvolvida ainda deverá ser disponibilizada ao público geral, embora alguns membros do grupo de pesquisa já a utilizem. Um desenvolvimento significativo é permitir uma interface mais acessível para um especialista leigo em programação, como um binário executável com interface gráfica ou uma página Web.

Outro desenvolvimento significativo seria a adaptação da ferramenta para execução distribuída, que será necessária para obter respostas rápidas sobre conjuntos de dados muito grandes. O aprimoramento em eficiência para permitir a análise de conjuntos moderados em computadores pessoais, não exclui o benefício de se executar uma análise em um *cluster* dedicado ou em provedores de nuvem. Um ganho significativo deve ser obtido ao distribuir o cálculo da matriz de distância, que é a etapa mais custosa mas também embaraçosamente paralela. Particionando-se o conjunto em  $n$  partes, pode-se alocar  $n^2$  máquinas em uma grade  $n \times n$ , onde a máquina  $(i, j)$  deverá comprimir a submatriz contendo os elementos das partes  $i$  e  $j$ .

A ferramenta ainda não inclui diversas opções de compressores populares, como RAR, 7Zip e ZIP; opções de etapas de simplificação como limiarização,  $k$ NN e UPGMA; e não contém implementações eficientes das novas medidas de modularidade propostas ou de distâncias de ponto-a-grupo e grupo-a-grupo. Espera-se que a adição de tais funcionalidades seja acessível a terceiros devido à sua arquitetura extensível.



#### 6.2.4 Classificador por árvores de quarteto

O classificador por árvores de quarteto possui características únicas que ainda não foram exploradas por completo. Inicialmente, a escolha de *outgroup* necessita de uma heurística melhor que a aleatória, e preferencialmente determinística. A exploração do espaço de parâmetros possíveis é extensa e não é garantido que seja transferível de aplicação para aplicação, podendo depender da distribuição multidimensional dos elementos do conjunto.

Ainda, a capacidade de o classificador retornar que um elemento não foi classificado pode não ser desejável para um especialista. Algumas maneiras antevistas para se solucionar isto é acoplar o classificador a outro mais robusto, como proposto com o *k*NN na seção 5.3. Outras possibilidades incluem: realizar a classificação de forma iterativa, inserindo elementos classificados com segurança no grupo, e realizar a classificação novamente com este acréscimo de informação; utilizar simultaneamente diversos *outgroups* com heurísticas diferentes, e obter um consenso entre as classificações obtidas; e utilizar técnicas alternativas para agrupamento dos elementos da matriz  $4 \times 4$  que não a reconstrução filogenética.



## REFERÊNCIAS

---

ALBATINEH, A. N. Means and variances for a family of similarity indices used in cluster analysis. **Journal of Statistical Planning and Inference**, Elsevier, v. 140, n. 10, p. 2828–2838, 2010. Citado na página 58.

ALBATINEH, A. N.; NIEWIADOMSKA-BUGAJ, M. Correcting jaccard and other similarity indices for chance agreement in cluster analysis. **Advances in Data Analysis and Classification**, Springer, v. 5, n. 3, p. 179–200, 2011. Citado na página 58.

ALMEIDA, T.; HIDALGO, J. M. G.; SILVA, T. P. Towards sms spam filtering: Results under a new dataset. **International Journal of Information Security Science**, v. 2, n. 1, p. 1–18, 2013. Citado 2 vezes nas páginas 81 e 88.

AMORIM, D. de S. **Fundamentos de Sistemática Filogenética**. [S.l.]: Holos Editora, 2002. Citado na página 36.

ANDROUTSOPOULOS, I.; KOUTSIAS, J.; CHANDRINOS, K. V.; PALIOURAS, G.; SPYROPOULOS, C. D. An evaluation of naive bayesian anti-spam filtering. **arXiv preprint cs/0006013**, 2000. Citado 2 vezes nas páginas 81 e 88.

AUGERI, C. J.; BULUTOGLU, D. A.; MULLINS, B. E.; BALDWIN, R. O.; III, L. C. B. An analysis of xml compression efficiency. In: ACM. **Proceedings of the 2007 workshop on Experimental computer science**. [S.l.], 2007. p. 7. Citado na página 31.

BALDWIN, T.; LUI, M. Language identification: The long and the short of the matter. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics**. [S.l.], 2010. p. 229–237. Citado 2 vezes nas páginas 81 e 89.

BENNETT, C. H.; GÁCS, P.; LI, M.; VITÁNYI, P. M.; ZUREK, W. H. Information distance. **Information Theory, IEEE Transactions on**, IEEE, v. 44, n. 4, p. 1407–1423, 1998. Citado 2 vezes nas páginas 27 e 28.

BRANDES, U. A faster algorithm for betweenness centrality\*. **Journal of Mathematical Sociology**, Taylor & Francis, v. 25, n. 2, p. 163–177, 2001. Citado na página 46.

BRANDES, U.; DELLING, D.; GAERTLER, M.; GORKE, R.; HOEFER, M.; NIKOLOSKI, Z.; WAGNER, D. On modularity clustering. **Knowledge and Data Engineering, IEEE Transactions on**, IEEE, v. 20, n. 2, p. 172–188, 2008. Citado 2 vezes nas páginas 44 e 46.

BURROWS, M.; WHEELER, D. J. A block-sorting lossless data compression algorithm. Citeseer, 1994. Citado na página 33.

CEBRIAN, M.; ALFONSECA, M.; ORTEGA, A. Common pitfalls using the normalized compression distance: What to watch out for in a compressor. **Communications in Information & Systems**, International Press of Boston, v. 5, n. 4, p. 367–384, 2005. Citado na página 35.

- CILIBRASI, R. Complearn toolkit. 2005. Citado 2 vezes nas páginas 73 e 85.
- CILIBRASI, R.; VITÁNYI, P. M. Clustering by compression. **Information Theory, IEEE Transactions on**, IEEE, v. 51, n. 4, p. 1523–1545, 2005. Citado 2 vezes nas páginas 27 e 85.
- CILIBRASI, R. L.; VITANYI, P. The google similarity distance. **Knowledge and Data Engineering, IEEE Transactions on**, IEEE, v. 19, n. 3, p. 370–383, 2007. Citado na página 92.
- CLAUSET, A.; NEWMAN, M. E.; MOORE, C. Finding community structure in very large networks. **Physical review E**, APS, v. 70, n. 6, p. 066111, 2004. Citado na página 47.
- COHEN, A. R.; VITÁNYI, P. M. B. Normalized compression distance of multisets with applications. **CoRR**, abs/1212.5711, 2012. Citado na página 94.
- DARWIN, C. On the origin of species. 1. **London: Murray**, 1859. Citado na página 36.
- DARWIN, C.; WALLACE, A. On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection. **Journal of the proceedings of the Linnean Society of London. Zoology**, Wiley Online Library, v. 3, n. 9, p. 45–62, 1858. Citado na página 36.
- DELBEM, A. C. **Mineração de dados de biofotônica e climáticos para aumento da precisão e da robustez de modelos de predição/deteção do Greening**. [S.l.], 2012. Citado 2 vezes nas páginas 69 e 91.
- DEUTSCH, P.; GAILLY, J.-L. **Zlib compressed data format specification version 3.3**. [S.l.], 1996. Citado na página 32.
- FELSENSTEIN, J. Confidence limits on phylogenies: an approach using the bootstrap. **Evolution**, JSTOR, p. 783–791, 1985. Citado na página 63.
- FORTUNATO, S. Community detection in graphs. **Physics Reports**, Elsevier, v. 486, n. 3, p. 75–174, 2010. Citado 3 vezes nas páginas 40, 43 e 47.
- FORTUNATO, S.; BARTHELEMY, M. Resolution limit in community detection. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 104, n. 1, p. 36–41, 2007. Citado 3 vezes nas páginas 15, 44 e 45.
- FOWLKES, E. B.; MALLOWS, C. L. A method for comparing two hierarchical clusterings. **Journal of the American Statistical Association**, v. 78, n. 383, p. 553–569, 1983. Disponível em: <<http://www.tandfonline.com/doi/abs/10.1080/01621459.1983.10478008>>. Citado 3 vezes nas páginas 55, 57 e 58.
- FREDERICKSON, G. N. Optimal algorithms for tree partitioning. In: **SODA**. [S.l.: s.n.], 1991. p. 168–177. Citado na página 48.
- GIRVAN, M.; NEWMAN, M. E. Community structure in social and biological networks. **Proceedings of the National Academy of Sciences**, National Acad Sciences, v. 99, n. 12, p. 7821–7826, 2002. Citado na página 46.
- HUBERT, L.; ARABIE, P. Comparing partitions. **Journal of classification**, Springer, v. 2, n. 1, p. 193–218, 1985. Citado 3 vezes nas páginas 55, 57 e 58.

JACCARD, P. The distribution of the flora in the alpine zone. 1. **New phytologist**, Wiley Online Library, v. 11, n. 2, p. 37–50, 1912. Citado na página 55.

JONES, E.; OLIPHANT, T.; PETERSON, P. *et al.* **SciPy: Open source scientific tools for Python**. 2001–. [Online; accessed 2016-01-27]. Disponível em: <<http://www.scipy.org/>>. Citado na página 74.

KARP, R. M.; RABIN, M. O. Efficient randomized pattern-matching algorithms. **IBM Journal of Research and Development**, IBM, v. 31, n. 2, p. 249–260, 1987. Citado na página 93.

KONG, J. S.; REZAEI, B.; SARSHAR, N.; ROYCHOWDHURY, V. P.; BOYKIN, P. O. *et al.* Collaborative spam filtering using e-mail networks. **Computer**, IEEE, v. 39, n. 8, p. 67–73, 2006. Citado na página 88.

LI, M.; CHEN, X.; LI, X.; MA, B.; VITÁNYI, P. M. The similarity metric. **Information Theory, IEEE Transactions on**, IEEE, v. 50, n. 12, p. 3250–3264, 2004. Citado 2 vezes nas páginas 26 e 29.

LUKES, J. A. Efficient algorithm for the partitioning of trees. **IBM Journal of Research and Development**, IBM, v. 18, n. 3, p. 217–224, 1974. Citado na página 48.

MAHONEY, M. Paq8f. **Online**. <http://mattmahoney.net/dc/paq.html#paq8>, 2006. Acessado em 29 de Novembro de 2015. Citado na página 34.

MANSOUR, E. R. M. **Método automático de determinação de clados utilizando algoritmo de detecção de comunidades**. Dissertação (Mestrado) — Universidade de São Paulo, 2013. Citado na página 25.

MANSUR, E. R. M. **Método automático de determinação de clados utilizando algoritmo de detecção de comunidades**. Dissertação (Mestrado) — Universidade de São Paulo, 2013. Citado na página 22.

MARAVALLE, M.; SIMEONE, B.; NALDINI, R. Clustering on trees. **Computational Statistics & Data Analysis**, Elsevier, v. 24, n. 2, p. 217–234, 1997. Citado na página 48.

MATTHEWS, D.; HOSKER, J.; RUDENSKI, A.; NAYLOR, B.; TREACHER, D.; TURNER, R. Homeostasis model assessment: insulin resistance and  $\beta$ -cell function from fasting plasma glucose and insulin concentrations in man. **Diabetologia**, Springer, v. 28, n. 7, p. 412–419, 1985. Citado na página 62.

MEILA, M. Comparing clusterings by the variation of information. In: **Learning theory and kernel machines**. [S.l.]: Springer, 2003. p. 173–187. Citado na página 57.

\_\_\_\_\_. Comparing clusterings: an axiomatic view. In: **ACM. Proceedings of the 22nd international conference on Machine learning**. [S.l.], 2005. p. 577–584. Citado na página 56.

\_\_\_\_\_. Comparing clusterings—an information based distance. **Journal of Multivariate Analysis**, Elsevier, v. 98, n. 5, p. 873–895, 2007. Citado 3 vezes nas páginas 57, 59 e 60.

MING, L.; VITÁNYI, P. **An introduction to Kolmogorov complexity and its applications**. [S.l.]: Springer Heidelberg, 1997. Citado na página 27.

MITCHELL, T. M. **Machine Learning**. [S.l.]: McGraw-Hill, 1997. Citado na página 66.

MOLLOY, M.; REED, B. A critical point for random graphs with a given degree sequence. **Random structures & algorithms**, Wiley Online Library, v. 6, n. 2-3, p. 161–180, 1995. Citado na página 44.

MUNKRES, J. Algorithms for the assignment and transportation problems. **Journal of the Society for Industrial & Applied Mathematics**, SIAM, v. 5, n. 1, p. 32–38, 1957. Citado na página 56.

NEWMAN, M. E. Analysis of weighted networks. **Physical Review E**, APS, v. 70, n. 5, p. 056131, 2004. Citado na página 44.

\_\_\_\_\_. Fast algorithm for detecting community structure in networks. **Physical review E**, APS, v. 69, n. 6, p. 066133, 2004. Citado na página 47.

NEWMAN, M. E.; GIRVAN, M. Finding and evaluating community structure in networks. **Physical review E**, APS, v. 69, n. 2, p. 026113, 2004. Citado na página 43.

OCHIAI, A. Zoogeographic studies on the solenoid fishes found in japan and its neighbouring regions. **Bulletin of the Japanese Society for the Science of Fish**, n. 22, p. 526–530, 1957. Citado na página 56.

PÉREZ, F.; GRANGER, B. E. IPython: a system for interactive scientific computing. **Computing in Science and Engineering**, IEEE Computer Society, v. 9, n. 3, p. 21–29, maio 2007. ISSN 1521-9615. Disponível em: <<http://ipython.org>>. Citado na página 74.

PINTO, R. d. S. **Predição de perfis de carga de processos a partir de executáveis utilizando método de mineração de dados de repositório de códigos**. Tese (Doutorado) — Universidade de São Paulo, 2013. (em andamento). Citado 5 vezes nas páginas 22, 25, 82, 85 e 91.

PLOTREE, D.; PLOTGRAM, D. Phylip-phylogeny inference package (version 3.2). **cladistics**, Wiley Online Library, v. 5, p. 163–166, 1989. Citado na página 73.

QUISPE-AYALA, M. R.; ASALDE-ALVAREZ, K.; ROMAN-GONZALEZ, A. Image classification using data compression techniques. In: IEEE. **Electrical and Electronics Engineers in Israel (IEEEI), 2010 IEEE 26th Convention of**. [S.l.], 2010. p. 000349–000353. Citado na página 30.

RAND, W. M. Objective criteria for the evaluation of clustering methods. **Journal of the American Statistical association**, Taylor & Francis Group, v. 66, n. 336, p. 846–850, 1971. Citado na página 54.

RONQUIST, F.; HUELSENBECK, J. P. Mrbayes 3: Bayesian phylogenetic inference under mixed models. **Bioinformatics**, Oxford Univ Press, v. 19, n. 12, p. 1572–1574, 2003. Citado na página 37.

SAITOU, N.; NEI, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. **Molecular biology and evolution**, SMOE, v. 4, n. 4, p. 406–425, 1987. Citado na página 38.

SANCHES, A.; CARDOSO, J. M.; DELBEM, A. C. Identifying merge-beneficial software kernels for hardware implementation. In: IEEE. **Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on**. [S.l.], 2011. p. 74–79. Citado 2 vezes nas páginas 22 e 25.

- SEWARD, J. The bzip2 and libbzip2 official home page. **Online**. <http://www.bzip.org>, 1996. Acessado em 29 de Novembro de 2015. Citado na página 33.
- SHKARIN, D. Ppmd. **Online**. <http://compression.ru/ds/>, 2010. Acessado em 29 de Novembro de 2015. Citado na página 34.
- SIGURBJÖRNSSON, B.; KAMPS, J.; RIJKE, M. D. **EuroGOV: Engineering a multilingual Web corpus**. [S.l.]: Springer, 2006. Citado na página 81.
- SOARES, A. H. M. **Algoritmos de Estimação de Distribuição baseados em Árvores Filogenéticas**. Tese (Doutorado) — Universidade de São Paulo, 2013. (em andamento). Citado na página 25.
- SULLIVAN, D. **Why Google Can't Count Results Properly**. 2011. Disponível em: <<http://searchengineland.com/why-google-cant-count-results-properly-53559>>. Citado na página 93.
- TUKEY, J. W. Bias and confidence in not-quite large samples. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 29, n. 2, p. pp. 614–623, 1958. ISSN 00034851. Disponível em: <<http://www.jstor.org/stable/2237363>>. Citado na página 63.
- VINH, N. X.; EPPS, J.; BAILEY, J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. **The Journal of Machine Learning Research**, JMLR. org, v. 11, p. 2837–2854, 2010. Citado na página 60.
- WALLACE, D. L. Comment. **Journal of the American Statistical Association**, Taylor & Francis Group, v. 78, n. 383, p. 569–576, 1983. Citado 2 vezes nas páginas 56 e 57.
- ZACHARY, W. W. An information flow model for conflict and fission in small groups. **Journal of anthropological research**, JSTOR, p. 452–473, 1977. Citado 2 vezes nas páginas 15 e 46.
- ZIV, J.; LEMPEL, A. A universal algorithm for sequential data compression. **IEEE Transactions on information theory**, v. 23, n. 3, p. 337–343, 1977. Citado na página 32.