

ClusteringTools: Uma Ferramenta de Auxílio ao Ensino de Técnicas de Clusterização.

Debora Theodoro Amancio da Silva
Vinícius Fernandes dos Santos
Orientador: Adriano Joaquim da Oliveira Cruz

UFRJ

2007

ClusteringTools: Uma Ferramenta de Auxílio ao Ensino de Técnicas de Clusterização.

por

Debora Theodoro Amancio da Silva
Vinícius Fernandes dos Santos

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentada por:

Debora Theodoro Amancio da Silva

Vinícius Fernandes dos Santos

Aprovada por:

Adriano Joaquim de Oliveira Cruz, Ph.D
(Presidente)

Vanessa de Paula Braganholo, DSc.

Antonio Juarez Alencar, Ph.D

RIO DE JANEIRO, RJ - BRASIL

Abril de 2007

Resumo

A clusterização de dados tem se mostrado útil em diversas aplicações e, por isso, tem sido estudada por cientistas da computação de diferentes áreas de pesquisa. Vários conceitos, dentre eles a lógica nebulosa, têm sido utilizados nos algoritmos em busca de resultados melhores e mais expressivos. Este trabalho apresenta uma ferramenta desenvolvida para auxiliar o ensino das técnicas de clusterização para alunos da Pós-Graduação em Informática do Instituto de Matemática. São descritos os conceitos envolvidos e algoritmos, além de ser apresentada a ferramenta e expostos os recursos utilizados no desenvolvimento desta, bem como suas funcionalidades e suas possibilidades de expansão e uso.

Abstract

Data clustering have become useful in various applications and it has been studied by computer scientists of several research areas. Different concepts, among them fuzzy logic, have been used in algorithms on search of better and expressive results. This work presents a tool developed to help teaching of clustering techniques to students of the M.Sc. in Computer Science of the Institute of Mathematics. The concepts and algorithms involved are described, in addition the tool is presented and the resources used to develop it are shown, as well as its functionalities and its possibilities of expansion and use.

Sumário

Lista de Figuras

Capítulo 1

Introdução

No último século, cientistas das mais diversas áreas começaram a tratar de maneira sistemática os grupos em seus dados. Com o advento da computação, grandes quantidades de informação puderam ser tratadas, muitas vezes em volumes impossíveis de serem tratados manualmente. A classificação automática de dados em grupos tornou-se, então, uma importante área de pesquisa, com aplicações em computação, como reconhecimento de padrões e inteligência artificial, e em outras áreas como biologia, economia, geociências, marketing, sendo conhecida por outros nomes como “Clustering” e “Taxonomia Numérica” [?], [?].

O desenvolvimento deste projeto se baseia na importância da área para a computação e tenta suprir uma necessidade de uma ferramenta voltada para o ensino do assunto. Esta necessidade foi identificada nas aulas da disciplina Lógica Nebulosa, ministradas pelo professor Adriano Cruz na Pós-Graduação do Departamento de Ciência da Computação, dada a dificuldade de geração de exemplos de maneira simples durante as aulas. O objetivo desta ferramenta é facilitar a criação de exemplos e demonstrar o funcionamento dos algoritmos e suas diferenças através de uma interface gráfica, auxiliando a compreensão dos mesmos. Foi dada atenção diferenciada aos algoritmos que utilizam lógica nebulosa, por ser a área de interesse tanto no ensino da disciplina, quanto para os envolvidos neste trabalho.

Durante toda a execução deste projeto tivemos a preocupação com a extensibilidade da ferramenta. Novos algoritmos, tipos de dados, técnicas de padronização de dados e até novas formas de leitura podem ser desenvolvidas. Não nos impomos a meta de desenvolver uma ferramenta onde fosse possível implementar todas as soluções conhecidas existentes na área, mas nos preocupamos em fazer uma estrutura flexível e reusável, onde não fosse

necessário despende esforço em tarefas repetidas.

O restante deste trabalho está estruturado como segue. O capítulo ?? fala da base teórica necessária à compreensão dos conceitos envolvidos e utilizada para a implementação da ferramenta. O capítulo ?? fala da implementação da ferramenta, expondo os recursos utilizados além da estrutura e funcionamento do programa. O capítulo ?? fala das aplicações, com exemplos do funcionamento da ferramenta e o capítulo ?? fala das conclusões e trabalhos futuros.

Capítulo 2

Clusterização

As técnicas de clusterização consistem em analisar as informações de diferentes objetos e dividi-los em grupos - “clusters” - de determinada forma, como exemplificado na figura ???. Diferentemente da classificação, que consiste em analisar cada objeto e decidir a qual classe previamente conhecida ele pertence, a clusterização não conhece as classes previamente e procura agrupar os elementos em conjuntos de forma que dois elementos de um conjunto sejam parecidos entre si e elementos de conjuntos diferentes sejam o mais diferentes possível. Além desses pré-requisitos, existem diversos algoritmos propostos a tratar determinadas particularidades dos grupos. Há ainda uma fase de tratamento dos dados para que estes sejam preparados para a aplicação dos algoritmos. Nesta seção pretendemos expor a teoria necessária para a compreensão das funcionalidades da ferramenta, embora de maneira sucinta.

2.1 Tipos de dados

A primeira dificuldade ao tentarmos dividir os objetos em grupos é a diferença entre os tipos de dados. Algumas características são grandezas numéricas e podem ser discretas, como idade, ou contínuas, como distâncias. Outros tipos de dados são mais difíceis de lidar por sua natureza matemática menos rigorosa. Dentre elas temos as variáveis nominais, como países, e variáveis ordinais, como as medalhas de uma competição.

A relação entre os dados é dada com relações de similaridade ou dissimilaridade. Como elas podem ser intercambiadas sem muita dificuldade, optamos por utilizar a métrica de dissimilaridade pela facilidade de interpretação dos valores com o uso de funções de cálculo de distância conhecidas. Em geral, os métodos de clusterização utilizam uma função

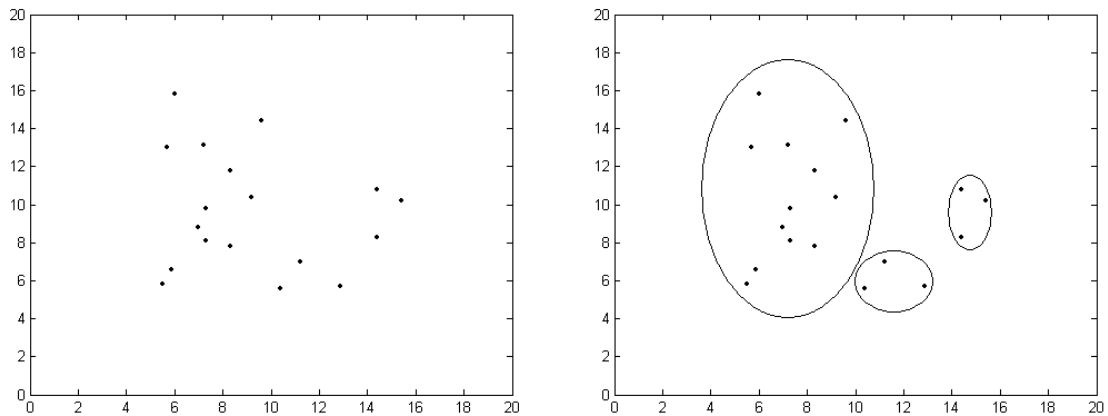


Figura 2.1: Uma distribuição de dados (esquerda) e uma possível distribuição destes dados em grupos (direita).

$D(x, y)$ que fornece um valor para a distância entre o objeto x e o objeto y . Muitos métodos também utilizam a média entre os valores para calcular os centros dos grupos, que são os objetos que os representam. Portanto, essas duas operações são fundamentais para todos os tipos de dados, que são explicados a seguir.

2.1.1 Tipo numérico intervalar

Neste tipo se enquadram as grandezas como idade, altura, temperatura e todo o tipo de grandeza numérica de natureza linear. O cálculo da distância é feito da forma padrão, calculando-se o módulo da diferença entre os valores, e a média também é feita de forma imediata, somando-se todos os valores e dividindo pelo número de amostras. É o tipo mais fácil de ser tratado, mas podem aparecer problemas com a escala ao se calcular a distância. Por exemplo, uma diferença de 2 quilogramas entre os pesos de duas pessoas não pode ser comparada a uma distância de 200 centímetros. Ao se calcular a distância entre dois elementos com peso e altura, é necessário padronizar essas variáveis, fazendo que a escala não interfira mais. A padronização é melhor explicada na seção ??.

2.1.2 Tipo numérico exponencial

Para tipos de dados com características exponenciais como o crescimento da quantidade de bactérias em cultura, torna-se mais apropriado um tratamento diferenciado, que leve em consideração esta particularidade. Uma maneira simples de tratar este tipo de dado

é, para cada x_i deste tipo, associar um valor $y_i = \ln(x_i)$ e usar os valores y_i no processo de clusterização, tratando-os como um tipo numérico intervalar.

2.1.3 Tipo binário

O tipo de dado binário é usado para variáveis que só podem ter dois valores distintos, conhecidos de antemão, tais como variáveis booleanas (“verdadeiro” ou “falso”) ou sexo (“masculino” ou “feminino”). O tipo binário é um caso particular do tipo nominal (que será abordado na seção ??) que tem seus valores associados a strings e dessa forma necessita de uma forma especial de representação ou de cálculo de distância e média. Existem diversas formas de se calcular a distância para alguns tipos de dados binários, dando maior importância para um dos dois valores ou tratando-os da mesma forma. Escolhemos para este tipo tratar os valores com o mesmo peso, portanto a distância só depende se duas variáveis são iguais ou não. A média de variáveis binárias será explicada na implementação (seção ??), visto que depende desta.

2.1.4 Tipo nominal

Como dito anteriormente, o tipo nominal tem como seus valores strings de caracteres pré-estabelecidas e, assim como no tipo binário, requer um tratamento especial. Uma idéia possível seria associar a cada um dos possíveis valores um número real. Porém, isso pode trazer consequências indesejadas pois, no cálculo da distância, a definição dos valores irá interferir no resultado, tornando pares de strings mais próximos que outros, sem razão evidente para isso. Como solução para isso, associamos cada um dos estados a uma variável booleana, de forma que haja sempre uma com valor verdadeiro e as outras com o valor falso. Embora isso tenha uma desvantagem no desempenho, optamos por essa estratégia por ela impedir resultados incorretos resultantes da transformação. Há ainda um outro problema: como uma característica se torna um conjunto de variáveis booleanas, ao se calcular distâncias, a contribuição oriunda destas variáveis forneceria um contribuição maior do que de uma única variável de outro tipo. Este problema foi resolvido com o cálculo especial da distância, considerando todas as variáveis booleanas associadas à variável nominal ao invés de considerar cada uma separadamente. Assim, o número de variáveis booleanas não interfere, já que a distância calculada será sempre entre 0 e 1 (zero se igual e um se diferente). Assim como nas variáveis binárias, a média depende da implementação.

2.1.5 Tipo ordinal

O tipo ordinal é semelhante ao tipo nominal no sentido de ser representado por strings de caracteres, mas é mais simples. Os valores do tipo ordinal apresentam uma relação de distância parecida com a dos números naturais e não como no tipo nominal onde a distância pode ser representada com os valores “igual” e “diferente”. Como o nome sugere, eles podem ser ordenados e, usando essa propriedade, podemos tratá-los como números naturais, tornando o cálculo da distância e da média muito mais simples.

2.2 Dados desconhecidos

Tanto em amostras de dados reais quanto em massas de dados geradas artificialmente, pode haver objetos com uma ou mais características sem nenhum valor. Em fichas cadastrais, por exemplo, existem campos opcionais, os quais nem sempre são preenchidos. Surge então um problema: como tratar esses dados na clusterização?

Os algoritmos de clusterização em geral assumem que há sempre um valor em cada característica e não sabem lidar com esse tipo de situação. Mesmo para nós, é difícil responder, por exemplo, qual a distância entre 46 e “nada”. Existem algumas propostas de como lidar com esse problema e citaremos aquelas que decidimos adotar neste trabalho, embora outras possam ser adicionadas futuramente.

2.2.1 Eliminação de dados incompletos

Em determinados casos, a ausência de um valor em uma característica constitui um fato presente, porém indesejado. Em situações como essa, se mostra apropriado desprezar os objetos com informações incompletas e aplicar o algoritmo aos elementos restantes. Os dados desprezados podem ser ignorados ou classificados posteriormente em um dos clusters. Note que classificação é um procedimento diferente da clusterização e não se enquadra nos objetivos deste trabalho.

2.2.2 Eliminação de características

Um outra abordagem comum é supor que uma característica que pode não ter um valor associado é dispensável. Dessa forma, o campo relativo a esta característica é removido de cada objeto e então o algoritmo é aplicado a estes objetos modificados.

2.2.3 Substituição pela média

Uma estratégia mais conservadora não elimina nem o objeto nem a característica, mas atribui um valor à característica. Uma questão importante é que valor deve ser atribuído. Adotar a média como tal valor parece ser uma estratégia razoável porque, independente do cluster ao qual o objeto for atribuído, essa característica tende a não contribuir muito no cálculo da distância para os outros. Qualquer outra atribuição de valor poderia ser mais tendenciosa e acabar induzindo o algoritmo a conclusões incorretas.

2.3 Padronização

Ao lidarmos com dados de diferentes naturezas, não é incomum que encontremos grandezas de diferentes magnitudes. Estas diferenças, se não tratadas, podem levar a conclusões equivocadas. Suponha, por exemplo, que estejamos agrupando pessoas levando em consideração sua altura em quilos e seu peso em metros. Se estivermos utilizando a distância euclidiana, pode-se dizer que a distância será próxima a diferença de pesos, uma vez que a ordem de grandeza das duas medidas é bem diferente. Por outro lado, se usássemos a altura em centímetros, a conclusão seria totalmente diferente, dando à altura uma importância maior que na situação anterior, como na figura ???. Para evitar situações como esta, é comum o uso de algum tipo de padronização.

A padronização consiste na conversão das características originais para valores sem unidade de medida, fazendo também com que todas as variáveis tenham um peso igual no processo de clusterização. Este procedimento é particularmente útil quando não há conhecimento anterior dos dados, não sendo possível atribuir um peso maior a nenhuma característica.

2.3.1 Z-Score

Também conhecido como Padronização Z, consiste em aproximar os dados disponíveis à distribuição de frequência conhecida como distribuição normal. O passo inicial é o cálculo da média e o desvio padrão a partir dos valores para a característica. Em seguida os novos valores para a característica são obtidos subtraindo a média de cada valor antigo e dividindo o resultado pelo desvio padrão. Este procedimento transforma os dados de forma que sua nova média seja zero e o seu desvio padrão seja 1.

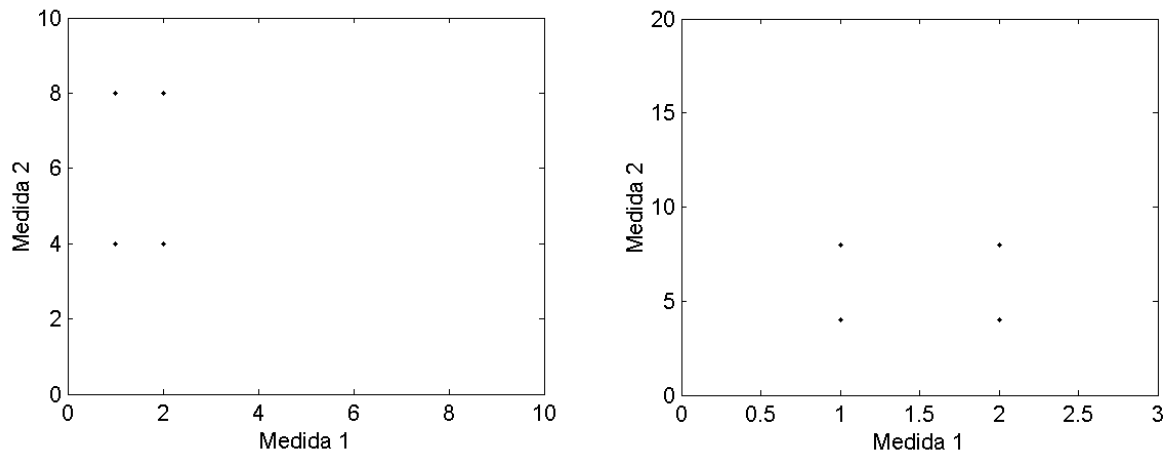


Figura 2.2: Os mesmos dados sendo exibidos em escalas diferentes, podendo induzir a diferentes resultados

2.3.2 Normalização Min-Max

Este procedimento, mais intuitivo que o anterior, consiste em colocar todos os valores da característica no intervalo $[0, 1]$. Para isso, o valor mínimo (v_{min}) entre todos os valores da característica é encontrado e subtraído de todas as amostras. Nesse momento, os valores encontram-se no intervalo $[0, v_{max} - v_{min}]$. O último passo é a divisão de cada valor por $(v_{max} - v_{min})$, onde (v_{max}) é o valor máximo, levando o conjunto de valores para o intervalo desejado, $[0, 1]$. Com este procedimento, todas as distâncias ficam compreendidas entre 0 e 1, tornando este método mais adequado para o uso de variáveis binárias e nominais com outros tipos de variáveis.

2.4 Lógica Nebulosa

Embora a lógica nebulosa não seja uma parte do procedimento de clusterização, uma introdução ao assunto se faz necessária, uma vez que este conceito é usado em alguns dos algoritmos que descreveremos a seguir. Uma explicação um pouco mais detalhada e mais referências sobre o assunto podem ser encontrados em [?].

A lógica nebulosa (do inglês “Fuzzy Logic”) foi criada por Lotfi Zadeh ainda na década de 60, e tenta representar matematicamente conceitos naturalmente imprecisos como “quente”, “rápido” ou “longe”.

Para falarmos um pouco sobre a lógica nebulosa, iniciaremos falando da lógica clás-

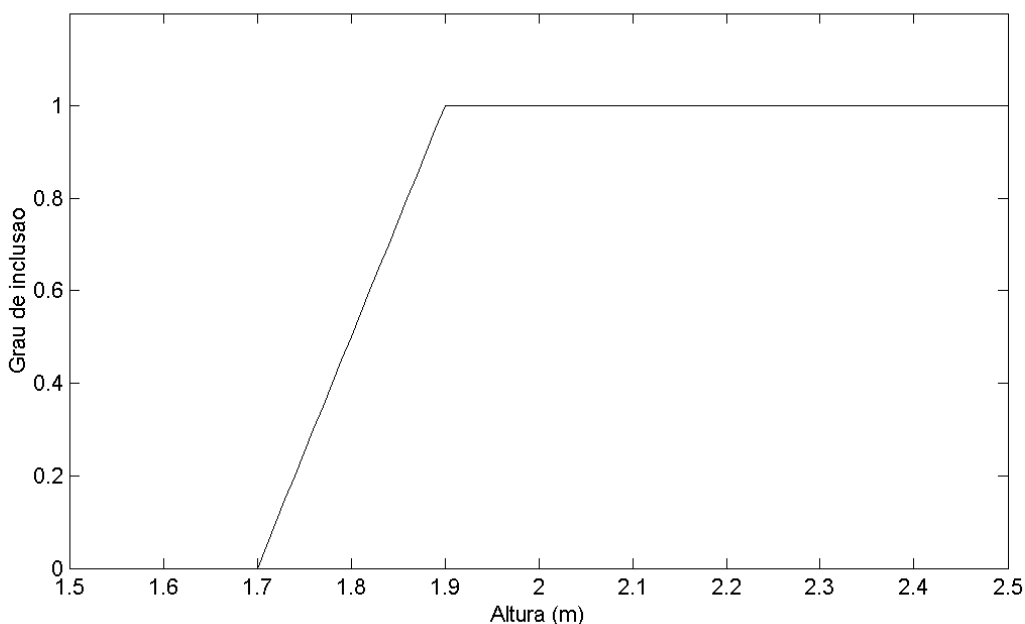


Figura 2.3: Representação gráfica da função de inclusão do conjunto nebuloso dos indivíduos altos

sica. Na lógica clássica, quando analisamos a pertinência de um elemento a um conjunto, chegamos a conclusão que ou o elemento pertence ou o elemento não pertence àquele conjunto. Isto parece bastante razoável para muitas aplicações. Porém, quando analisamos certas situações do mundo real, essa forma de tratar os fatos se mostra incompleta. Intuitivamente alguém poderia dizer que isso é impossível. Ou um fato é verdade ou não é. O que a lógica nebulosa sugere é que os fatos podem ter um “nível” de verdade.

Considere a seguinte situação: quando questionada sobre a altura de um criminoso, uma vítima pode dizer que ele é alto. O investigador então começa a procurar todas as fichas de criminosos com altura maior ou igual a 1,80m. Surge então a pergunta: uma pessoa medindo 1,80m é alta e a pessoa com 1,79m não o é? A olho nu essa diferença pode ser imperceptível, então passemos a considerar a pessoa com 1,79m alta. Utilizando o mesmo raciocínio, podemos concluir que uma pessoa com 1,78m de altura também é alta e assim por diante. Como vimos, estabelecer um limite em um conjunto do mundo real pode ser uma tarefa complexa.

Ainda considerando o conjunto das pessoas altas, sabemos que o indivíduo com 1,80m é mais alto que o de 1,79m. Desta forma, não seria absurdo dizer que o primeiro “pertence mais” ao conjunto que o segundo. Um possível gráfico para representar grau de inclusão

de um indivíduo no conjunto das pessoas altas em relação à sua altura pode ser visto na figura ??.

Desta forma, introduzimos o conceito de grau de inclusão de um indivíduo a um conjunto. Chamando o conjunto de A e o indivíduo de x , representamos por $\mu_A(x)$ este grau de inclusão. Nesta nova forma de tratar os conjuntos, torna-se necessária uma nova representação para os elementos de um conjunto. Não podemos apenas listar os elementos de A , pois perderíamos a informação dos graus de inclusão. Seja X uma coleção de objetos. Para listar os elementos de A , um conjunto nebuloso em X , utilizamos os pares ordenados $(x, \mu_A(x))$, $\forall x \in X$, com $\mu_A(x) \in [0, 1]$, sendo 0 o grau de inclusão mínimo e 1 o grau de inclusão máximo.

2.5 Algoritmos de Clusterização

O objetivo da clusterização é dividir um certo conjunto de dados em clusters (grupos ou subconjuntos). Esse processo deve levar em consideração duas propriedades:

- Dados que pertencem a um cluster devem ser tão semelhantes quanto o possível.
- Dados de clusters diferentes devem ser tão diferentes quanto o possível.

Os algoritmos de clusterização podem ser divididos em duas categorias: algoritmos hierárquicos e não-hierárquicos (também conhecidos como particionais). Em geral, em sua fundamentação teórica, todos os algoritmos tentam minimizar uma função J , podendo ser considerados, desta forma, problemas de otimização. Os algoritmos aqui apresentados são descritos em [?], [?].

2.5.1 Algoritmos hierárquicos

A principal característica dos algoritmos hierárquicos é a variação do número de clusters a cada iteração. O funcionamento destes algoritmos utiliza o conhecimento de uma determinada alocação dos objetos em clusters e, a partir daí, considerando a distância entre os elementos dos clusters, forma os seus sucessores. Sua denominação vem do fato da evolução do algoritmo poder se representada como uma árvore, com cada nível sendo construído a cada iteração.

Clusterização hierárquica aglomerativa

Na versão aglomerativa, o algoritmo parte das folhas da árvore e progride juntando clusters, diminuindo o número destes. Inicialmente, cada folha é um cluster contendo apenas um objeto. A cada iteração dois clusters são escolhidos e mesclados, formando um cluster maior. Este procedimento avança até alcançar o nível da árvore em que o número de clusters é igual ao desejado.

Ainda na inicialização do algoritmo, são calculadas os coeficientes de dissimilaridade $D(i, j), \forall i \forall j$. Os cálculos de distância mais comumente utilizadas são a distância euclidiana e a distância Manhattan. Os valores obtidos são normalmente armazenados para que não seja necessário recalculá-los, prejudicando o desempenho. Em todas as iterações posteriores, esses valores serão utilizados na tomada de decisão sobre quais clusters serão unificados.

Como descrito anteriormente, são unidos dois clusters a cada passo. A escolha do par a ser unido é feita da seguinte forma: para cada par de clusters calcula-se a distância entre eles. O par que obtiver a menor distância interna é então mesclado. Em caso de empate, escolhe-se arbitrariamente qualquer um dos pares. Embora intuitiva, essa idéia apresenta, ainda, um problema: não sabemos calcular a distância entre dois clusters, apenas entre dois objetos. Define-se, então, o cálculo da distância entre dois clusters A e B da forma descrita a seguir, de acordo com [?].

Sejam $|A|$ e $|B|$ o número de elementos dos conjuntos A e B respectivamente. A dissimilaridade $D(A, B)$ entre os dois conjuntos é dada pela média entre das dissimilaridades $D(i, j)$ para $\forall i \in A, \forall j \in B$. Ou, mais explicitamente:

$$D(A, B) = \frac{1}{|A||B|} \sum_{\substack{i \in A \\ j \in B}} D(i, j)$$

Clusterização hierárquica divisiva

No método divisivo, o algoritmo é inicializado com todos os objetos pertencendo ao mesmo cluster. O equivalente na nossa abstração de árvore é o início da execução do algoritmo na raiz da árvore, ao contrário do método aglomerativo que iniciava o percurso pelas folhas.

Como visto anteriormente, nos métodos hierárquicos o número de clusters varia a cada iteração. No método divisivo, como o próprio nome sugere, em cada passo um cluster é dividido em dois novos, aumentando o número total de clusters em uma unidade.

O processo é repetido até que o número de clusters desejado seja encontrado. Apesar de sua semelhança conceitual com o método aglomerativo, a clusterização divisiva é frequentemente ignorada pela literatura e mesmo pelos pacotes de aplicativos comerciais. A principal razão para este desprezo é seu custo computacional. Na primeira iteração, para que fossem testadas todas as partições possíveis, seriam necessárias $2^{n-1} - 1$ operações, um limite intolerável para a maior parte das aplicações práticas. Como pode-se supor, ao considerar todas as possibilidades de partições, percebemos que a maior parte delas é inadequada aos nossos propósitos. Baseado neste princípio, adotaremos um algoritmo que escolhe como dividir os clusters de uma maneira mais eficiente.

Ao invés de testar todas as possibilidades de partição, o algoritmo trabalha da seguinte forma: primeiro é encontrado o objeto com a maior dissimilaridade em relação aos outros objetos. A dissimilaridade de um elemento em relação ao seu conjunto pode ser definida como a dissimilaridade média em relação a todos os outros elementos de seu conjunto. Mais formalmente, a dissimilaridade de um elemento $i \in A$ em relação ao conjunto A é dada por:

$$D(i, A \setminus \{i\}) = \frac{1}{|A| - 1} \sum_{\substack{j \in A \\ j \neq i}} D(i, j)$$

O elemento i é então colocado em um novo conjunto B . Ainda na mesma iteração, é necessário verificar se existem outros elementos em A a serem colocados em B de forma a melhorar a dissimilaridade média. Para isso fazemos:

$$D(i, A \setminus \{i\}) - D(i, B) = \frac{1}{|A| - 1} \sum_{\substack{j \in A \\ j \neq i}} D(i, j) - \frac{1}{|B|} \sum_{h \in B} D(i, h)$$

Este valor é calculado para todos os elementos de A , e o maior deles, caso seja positivo, é então movido para o conjunto B . Em seguida é repetido o processo até que o valor calculado seja negativo ou nulo. Neste caso, está finalizada a primeira iteração.

Nas iterações seguintes, o algoritmo deve ainda decidir qual cluster deve ser dividido. Para isso é calculado o diâmetro de cada um dos conjuntos Q resultantes das iterações anteriores.

$$diam(Q) = \max D(j, h) \quad \forall j, h \in Q$$

É então escolhido o conjunto com o maior $diam(Q)$ e são executados os mesmos passos aplicados ao cluster inicial, dividindo-o em duas partes.

2.5.2 Algoritmos não-hierárquicos

Em geral, os algoritmos não-hierárquicos começam com um procedimento que aloca inicialmente os elementos a clusters. Em alguns desses métodos essa alocação inicial é aleatória. Em outros, existem regras específicas para isso. Após este passo inicial, o algoritmo itera até que não haja diferença significativa entre a alocação anterior e a atual. Em cada iteração, a distribuição dos elementos pelos clusters varia, em busca da distribuição ideal.

Um importante subconjunto dos algoritmos não-hierárquicos utiliza a lógica nebulosa como ferramenta. Como vimos, na lógica nebulosa, ao contrário da lógica clássica, um elemento pode pertencer a mais de um conjunto ao mesmo tempo. Utilizando este raciocínio, um elemento pode pertencer a mais de um cluster, com um determinado grau de inclusão $\mu_i(x_j)$ ou μ_{ij} , representando o grau de inclusão do elemento x_j no cluster i , que indica o quão fortemente o elemento está ligado ao cluster.

K-Means

O K-means talvez seja um dos métodos mais intuitivos de clusterização. Assim como os algoritmos hierárquicos vistos anteriormente, cada objeto sempre pertence a um e somente um cluster. Além disso, todos os clusters devem conter ao menos um elemento.

O primeiro passo do algoritmo é a inicialização. Nela é determinado o cluster ao qual cada elemento pertence. Habitualmente, isto pode ser feito de duas maneiras diferentes: ou os centros de cada cluster são escolhidos aleatoriamente dentre os elementos, ou cada objeto é colocado em um cluster aleatório e a partir daí são calculados os centros. Os centros serão recalculados a cada iteração, sempre que os elementos forem realocados entres os clusters.

Sejam x_i os objetos a serem clusterizados. Considerando os objetos como vetores, o centro v_j do cluster C_j pode ser calculado como a média de todos os elementos pertencentes ao cluster:

$$v_j = \frac{1}{|C_j|} \sum_{\forall x_i \in C_j} x_i$$

A partir daí, é calculada a distância D_{ij} de cada elemento x_i para cada centro v_j . Normalmente utiliza-se a distância euclidiana, embora outras possam também ser utilizadas. A partir dessas distâncias, cada um dos elementos é então realocado naquele cluster cujo centro tiver a menor distância em relação a ele. Neste ponto é verificada a necessidade de

efetuar mais iterações. O critério é simples: enquanto houver mudanças acontecendo, são calculados novos centros e os objetos são realocados. Quando não houver mais mudanças, o algoritmo pára.

O K-means se comporta bem para muitos tipos de dados, mas tem algumas desvantagens em relação a outros métodos. Em primeiro lugar, vem seu tempo de execução. Sejam r o número iterações, n o número de objetos e c o número de clusters. A cada uma delas é necessário recalcular os centros e as distâncias, o que nos dá uma complexidade total de $O(rnc)$. Outra desvantagem é sua sensibilidade em relação a erros ou elementos muito diferentes dos demais, conhecidos como “outliers”. Caso haja um dado errôneo com alguma coordenada muito diferente dos outros elementos, ele pode influenciar drasticamente o centro do seu cluster, interferindo na análise do resultado. Além destes problemas, uma outra característica pertinente é que o resultado da clusterização depende da alocação inicial. Desta forma, acabam fazendo-se necessárias várias execuções do algoritmo para garantir a convergência para uma boa solução.

Apesar dessas desvantagens, sua simplicidade e bom comportamento para nuvens de dados convexas e bem separadas torna o K-means um dos métodos mais conhecidos de clusterização.

K-Medoids

Apesar de apresentar algumas semelhanças com o K-means, o K-medoids apresenta algumas diferenças fundamentais. Em primeiro lugar, ele tenta não ser sensível aos outliers. A razão para isso é o fato de não utilizar uma média para o centro. Ao invés disso, o algoritmo utiliza um dos elementos da própria entrada como elemento representativo para o cluster.

Outra diferença fundamental é a alocação inicial dos elementos nos clusters. Ao contrário da aleatoriedade presente no K-means, este utiliza um algoritmo próprio de inicialização, tornando o algoritmo determinístico, encontrando sempre o mesmo resultado em cada execução.

O primeiro passo na execução do algoritmo é selecionar k elementos representativos (uma para cada um dos clusters desejados). O primeiro deles é escolhido de maneira trivial: é aquele cuja soma das dissimilaridades para todos os outros objetos é a menor possível, podendo, dessa forma, ser considerado o objeto mais central possível. A partir daí os outros $k - 1$ objetos são selecionados de forma a diminuir o valor da função objetivo

J . Para cada elemento x_i , deve-se verificar o quão bom seria tê-lo como um dos centros. Para cada outro elemento x_j que não seja um centro, deve ser calculada a distância entre os dois $D(x_i, x_j)$ e também a distância entre x_j e o centro previamente selecionado mais próximo dele, digamos, x_h . $C_{ij} = \max(D(x_j, x_h) - D(x_i, x_j), 0)$ nos dá o quanto a escolha de x_i como centro contribuirá para x_j . A operação de máximo é necessária, pois o valor 0 significa que não haverá contribuição, enquanto um valor negativo significaria uma piora na solução. $G_i = \sum_j C_{ij}$ representa o quanto de contribuição a escolha do elemento x_i fornece aos outros elementos. Portanto, o novo centro deve ser aquele que apresentar o maior G_i . Este passo deve ser repetido até que todos os k centros sejam escolhidos.

A partir deste ponto, já temos k centros iniciais determinados. Chegamos então ao loop principal do algoritmo, onde a cada iteração a solução melhora, até não se modificar mais. Neste passo, para cada par x_i, x_j , onde x_i é um centro e x_j não o é, mede-se o quão bom seria a troca. O par com o melhor valor é então selecionado e a troca é feita. As trocas continuam até que não hajam mais mudanças. Para avaliar qual troca deve ser feita, calcularemos D_{ij} , que nos mostra em quanto a função objetivo J será alterada se a troca for feita. A partir daí, é só escolher o menor D_{ij} . Caso seu valor seja negativo, significa que a função J será diminuída com a troca e esta deve ser feita. Caso contrário, o algoritmo chegou ao fim, por não ter como melhorar a solução.

Ao se avaliar um par para sabermos o quão bom seria a troca, devemos calcular C_{ijh} , onde x_h é um elemento não selecionado (não é um centro), para cada x_h possível. Existem várias situações possíveis.

- 1) Se x_h é mais distante de x_i e de x_j do que de um dos outros centros, $C_{ijh} = 0$.
- 2) Se x_h é mais próximo de x_i que de todos os outros centros, existem duas opções:
 - a) x_h é mais perto de x_j do que do segundo centro mais próximo. Nesse caso, $C_{ijh} = D(x_h, x_j) - D(x_h, x_i)$.
 - b) x_h é pelo menos tão distante de x_j quanto do segundo centro mais próximo. Nesse caso, sendo E_h a distância para o segundo mais próximo, temos $C_{ijh} = E_h - D(x_h, x_i)$.

3) O último caso é em que x_h é mais distante de x_i do que de pelo menos um dos outros centros, mas ainda é mais perto de x_j . Então teremos $C_{ijh} = D(x_h, x_j) - E_h$.

Apesar das vantagens descritas, o K-medoids, além de sua dificuldade de implementação, é mais custoso computacionalmente. Todas as avaliações que devem ser feitas na decisão de colocar um elemento como um centro acabam adicionando muitas operações ao algoritmo, tornando-o menos eficiente que o K-means.

C-Means

De certa forma, o C-means pode ser considerado uma generalização do K-means. Também conhecido como Fuzzy C-means, por utilizar lógica nebulosa (fuzzy), este método apresenta uma característica peculiar: um elemento pode pertencer a mais de um cluster. Como citado na seção ??, na lógica nebulosa um elemento pode pertencer a um conjunto com um certo grau de inclusão. Esta abordagem se mostra eficaz por evidenciar o distanciamento de um elemento do centro de um cluster, fazendo com que determinadas situações, como um ponto igualmente distante de dois centros distintos, sejam explicitadas alocando-se o elemento aos respectivos clusters com o mesmo grau de inclusão.

A idéia de alocação a um cluster com um determinado grau de inclusão pode ser melhor compreendida com a ajuda de um exemplo. Imagine que determinado professor classifique notas perto de 10,0 como notas ótimas e notas perto de 7,0 como boas. Com um método não-nebuloso, um determinado aluno que tenha tirado 8,6 seria alocado ao primeiro grupo, enquanto um com 8,4 seria alocado ao segundo, mesmo sendo muito próximos. Além disso, um aluno com 8,5 seria alocado arbitrariamente a um dos clusters, embora esteja equidistante de ambos os centros.

Assim como no K-means, nenhum cluster pode ficar vazio. Matematicamente, esta restrição pode ser escrita mais formalmente como:

$$0 < \sum_{j=1}^n \mu_{ij} < n, \quad \forall i$$

Note que o limite superior impede que um cluster contenha todos os elementos com grau 1, o que seria impossível sem deixar os outros clusters vazios. Da mesma forma, nenhum elemento pode ficar sem um cluster e, mais que isso, a soma dos graus de inclusão de um elemento em todos os clusters deve totalizar 1, significando que, no total, o elemento contribui em uma unidade à cardinalidade dos clusters:

$$\sum_{i=1}^c \mu_{ij} = 1, \quad \forall j$$

Em termos de estruturas de dados, uma diferença em relação ao K-means, é que em seu algoritmo era suficiente manter um vetor indicando de qual cluster cada um dos elementos fazia parte. No caso do C-means, como cada elemento x_j fará parte de cada cluster A_i com um determinado grau de inclusão $\mu_{A_i}(x_j) = \mu_{ij}$, torna-se necessária uma matriz com um total de nc elementos. Cada μ_{ij} deverá estar contido no intervalo $[0, 1]$. Definimos,

então, a matriz $U^{(r)}$ denominada Matriz de Inclusão, que contém dos graus de inclusão μ_{ij} da r -ésima iteração.

Um parâmetro necessário para a clusterização é o Fator de Nebulização, que representaremos como m . Este valor deve ser escolhido antes do início da execução do algoritmo e permanece imutável durante todas as iterações. Como o nome sugere, este fator indica o quão nebuloso os conjuntos serão. O intervalo de valores admissíveis para m é $[1, \infty)$, sendo $m = 1$ o comportamento não-nebuloso e $m \rightarrow \infty$ o comportamento totalmente nebuloso, onde cada elemento pertence a todos os conjuntos com o mesmo grau $1/c$. Valores tipicamente utilizados por apresentarem bom resultados são $m = 1,25$ e $m = 2,0$.

Além do Fator de Nebulização, como estamos tratando números reais, faz-se necessária a definição de um parâmetro $\varepsilon > 0$ que definirá o erro mínimo aceitável e definirá a condição de parada do algoritmo. A cada iteração, os centros são recalculados, assim como a distância de cada elemento para cada centro, e os elementos são realocados entre os clusters, através do ajuste de seus graus de inclusão, como descreveremos posteriormente. Após o ajuste dos graus de inclusão, torna-se necessário decidir se outra iteração será executada. Essa decisão é tomada verificando se a matriz de inclusão da iteração atual $U^{(r)}$ se modificou de maneira relevante em relação à matriz $U^{(r-1)}$ referente à iteração anterior. Para que esta decisão seja possível, é necessário calcular a norma da diferença destas duas matrizes, dada por $\|U^{(r)} - U^{(r-1)}\|$. É, então, verificada a condição $\|U^{(r)} - U^{(r-1)}\| < \varepsilon$ e, caso seja verdadeira, o loop é encerrado pois a solução já alcançou o nível de precisão desejado. O uso da norma como condição de parada também é comum no K-means (e de fato foi implementada), mas como os graus de inclusão do algoritmo são sempre inteiros, optamos por uma explicação mais intuitiva, facilitando sua compreensão.

A norma $|A|$ de uma matriz A com n linhas e m colunas pode ser calculada de diversas formas. Duas maneiras comuns de se calcular $|A|$ e que foram implementadas na ferramenta (embora outras também possam ser utilizadas) são a norma por coluna e a norma por linha. A norma por coluna, que é o maior somatório dos valores absolutos de cada coluna, é calculada da seguinte forma:

$$|A| = \max_{1 \leq j \leq m} \sum_{i=1}^n |\mu_{ij}|$$

Analogamente, podemos calcular a norma por linha como sendo:

$$|A| = \max_{1 \leq i \leq n} \sum_{j=1}^m |\mu_{ij}|$$

O primeiro passo para a execução do algoritmo é a alocação inicial dos elementos aos clusters, gerando a matriz de inclusão inicial $U^{(0)}$. Esta distribuição pode ser feita aleatoriamente ou arbitrariamente, porém a maneira como esta alocação é feita interfere no resultado do algoritmo, da mesma forma que no K-means, sendo necessária a execução do algoritmo para diferentes alocações iniciais, em busca de soluções melhores.

Após a inicialização da matriz de inclusão, o algoritmo entra em seu loop principal. O primeiro passo do loop é calcular os centros de cada cluster. O centro v_i referente ao i -ésimo cluster é, então, calculado da seguinte forma:

$$v_i = \frac{\sum_{j=0}^n \mu_{ij} x_j}{\sum_{j=0}^n \mu_{ij}}$$

O passo seguinte do algoritmo é calcular a distância $D_{ij} = D(v_i, x_j)$ entre o centro do cluster i e o elemento x_j , necessária para o cálculo dos graus de inclusão. A distância é calculada de maneira usual, utilizando o mesmo tipo de função utilizada em outros métodos, frequentemente a distância euclidiana.

Usando as distâncias obtidas no passo anterior, chegamos ao último passo da iteração, que é o cálculo dos graus de inclusão. O cálculo se divide em duas situações. A primeira delas é no caso em que $D_{ij} = 0$, para algum i . Neste caso, o elemento coincide com o centro de um dos clusters. Seja k esse cluster. Definimos então $\mu_{kj} = 1$, representando que o elemento está inteiramente contido no k -ésimo cluster, e $\mu_{ij} = 0$, $\forall i \neq k$. O segundo caso, no qual o elemento não está completamente contido em nenhum dos clusters, os graus de inclusão são calculados da forma a seguir:

$$\mu_{ij} = \left(\sum_{l=1}^c \left(\frac{D_{lj}}{D_{ij}} \right)^{\frac{2}{m-1}} \right)^{-1}$$

Seguindo os passos descritos para o cálculo dos centros, das distâncias dos graus de inclusão, completamos uma iteração e a matriz de inclusão $U^{(r)}$ é então comparada com a matriz resultante da iteração anterior, como descrito anteriormente, sendo verificada a condição de parada ou dando continuidade à execução do algoritmo na iteração seguinte.

Apesar de ser um pouco mais complexo que o K-means, o C-means se mostra mais versátil e traz informações importantes associadas aos graus de inclusão. Assim como no K-means, os clusters encontrados neste método são esféricos, o que pode representar uma limitação dependendo da natureza dos dados.

Gustafson-Kessel

Tanto o método Gustafson-Kessel quando o método Gath-Geva, que será descrito a seguir, são derivados do C-Means e diferem pela maneira como a distância é calculada entre os elementos. Embora pareça uma diferença simples, métodos sofisticados de cálculo de distância acabam implicando em resultados completamente diferentes dos de outros métodos. Enquanto o C-means normalmente utiliza a distância euclidiana, o Gustafson-Kessel utiliza a distância de Mahalanobis. Para que a distância de Mahalanobis possa ser utilizada, o primeiro passo é calcular a matriz de covariância nebulosa S_i :

$$S_i = \frac{\sum_{j=1}^n \mu_{ij}^m (x_j - v_i)(x_j - v_i)^T}{\sum_{j=1}^n \mu_{ij}^m}$$

O passo seguinte é calcular A_i :

$$A_i = \sqrt[p]{\det(S_i)} S_i^{-1}$$

É importante observar que para que a matriz S_i tenha inversa, é necessário que o número de amostras n seja maior que a dimensão p do espaço em que estamos trabalhando. Uma vez que tenhamos as matrizes A_i , o cálculo de D_{ij} torna-se trivial:

$$D_{ij}^2 = (x_j - v_i)^T A_i (x_j - v_i)$$

Com a introdução da distância de Mahalanobis, o método ganha flexibilidade com a mudança do formato dos clusters, que passam a poder admitir formas elipsoidais ao invés do simples formato esférico. Ainda assim, uma característica do método de Gustafson-Kessel é que essas elipsóides têm aproximadamente o mesmo tamanho.

Gath-Geva

Como dito, o Gath-Geva também é semelhante ao Fuzzy C-means, tendo como diferença a maneira de se calcular a distância. Também é conhecido como Decomposição Gaussiana (Gauss Mixture Decomposition), por utilizar a distância gaussiana.

A distância gaussiana é calculada de maneira semelhante à do método Gustafson-Kessel e também utiliza a matriz de covariância nebulosa S_i descrita anteriormente:

$$D_{ij}^2 = \frac{\sqrt{\det(S_i)}}{P_i} e^{\left(\frac{1}{2}(x_j - v_i)^T A_i (x_j - v_i)\right)}$$

E os cálculos de P_i e de A_i são feitos da seguinte maneira:

$$P_i = \frac{\sum_{j=1}^n \mu_{ij}^m}{\sum_{j=1}^n \sum_{l=1}^c \mu_{lj}^m} \quad A_i = S_i^{-1}$$

De todos os métodos implementados, é o que apresenta os resultados mais instáveis, variando muito seu resultado, de acordo com a alocação inicial dos clusters. Seus clusters não apresentam a regularidade esférica nem elipsoidal dos métodos anteriores, nem mesmo possuem o mesmo tamanho como nos casos anteriores, sendo este fortemente influenciado por P_i . Apesar destas características poderem ser consideradas vantagens por sua flexibilidade, pode se tornar necessário executar o algoritmo muitas vezes para que se encontre um solução satisfatória. Uma característica também deste método e inexistente em outros é o fato de clusters grandes acabarem atraindo mais elementos, podendo conduzir a situações indesejáveis.

2.6 Métricas de avaliação

A partir de uma alocação de objetos a clusters, devemos ter uma maneira de avaliar o quão boa ela é. Como vimos, alguns métodos começam com uma alocação aleatória de objetos dentre os clusters, influenciando o resultado de maneira imprevisível. Para casos como esse, o algoritmo deve ser rodado diversas vezes, e deve ser escolhida a melhor configuração. O mesmo pode ser feito para avaliar alocações feitas por diferentes algoritmos. Mas como escolher esta alocação? Para que a escolha possa ser feita, torna-se necessário o uso de uma métrica de avaliação.

Há ainda um outro uso para as métricas. Como exposto anteriormente, ao contrário das técnicas de classificação, o número de clusters não é conhecido de antemão. Dessa forma, surge um problema: dada uma massa de dados, qual o número ideal de clusters em que os objetos devem ser divididos?

Se cada objeto for colocado em um cluster, podemos dizer que a distância de cada objeto para os outros de seu cluster é zero, o que pode ser considerado bom. Por outro lado, a distância entre clusters diferentes será pequena, ou seja, dois objetos diferentes, ainda que muito semelhantes, estarão em clusters diferentes. Além disso, para fins práticos, não há interesse nesta situação. Suponha que uma empresa queira dividir seus clientes em

categorias para melhor atendê-los. Essa alocação não agradará seu gerente de vendas, pois provavelmente não terão como oferecer um serviço específico a cada cliente.

No outro extremo, em que todos os objetos estão em um único cluster, teremos grandes diferenças dentro de um único conjunto. Fazendo a analogia com a empresa que quer atender melhor seus clientes, de nada adiantará colocá-los todos em única categoria.

Como podemos ver, as propriedades citadas na seção ?? são importantes também na escolha da métrica. As métricas existentes tentam levar em consideração essas propriedades, fornecendo funções que devem ser maximizadas ou minimizadas. Os métodos aqui expostos, bem como métodos alternativos, são descritos em [?].

2.6.1 Compactness and Separation - CS

A métrica CS (Compactness and Separation) foi elaborada para avaliar clusters nebulosos mas pode ser utilizada para outros métodos, desde que haja uma matriz de inclusão. É considerada uma medida de validação completa, pois leva em consideração tanto o número de clusters e o grau de separação entre os clusters. Quanto menor o seu valor, melhor a qualidade da clusterização. O valor de CS é definido como:

$$CS = \frac{J}{n (d_{min})^2}$$

Na equação, temos n como o número de elementos, J a função objetivo a ser minimizada pelo C-means e d_{min} é a menor distância entre dois centros de clusters distintos.

$$J = \sum_{i=1}^c \sum_{j=1}^n \mu_{ij} |v_i - x_j|^2 \quad d_{min} = \min_{i \neq j} |c_i - c_j|$$

2.6.2 Inter Class Contrast - ICC

O ICC (Inter Class Contrast), assim como o CS, avalia clusters nebulosos, mas pode ser usado também para avaliar os resultados de outros algoritmos. Ao contrário do CS, no ICC, uma medida melhor é a que tem o maior valor numérico. A métrica leva em consideração a separação entre os clusters e também o seu tamanho. Quanto mais separados e compactos forem os clusters, maior será o valor da medida. O ICC utiliza d_{min} , descrito anteriormente, além de n e c que são valores conhecidos que nos dão o número de elementos e o número de clusters, respectivamente.

$$ICC = \frac{|S_{Be}|}{n} d_{min} \sqrt{c}$$

S_{B_e} é uma métrica que estima a qualidade da escolha dos centros e é dado por:

$$S_{B_e} = \sum_{i=1}^c \sum_{j=1}^n \mu_{ij} (m_i - m)(m_i - m)^T$$

$$m = \frac{1}{n} \sum_{j=1}^n x_j \qquad m_i = \frac{\sum_{j=1}^n \mu_{ij} x_j}{\sum_{j=1}^n \mu_{ij}}$$

Capítulo 3

A Ferramenta

3.1 Implementação

A implementação da ferramenta foi feita na linguagem Java, utilizando principalmente as APIs de Reflection e Annotation. Utilizamos também JAMA, que é um pacote de classes para álgebra linear e Apache Commons Math, que é uma biblioteca matemática da Apache.

3.1.1 Reflection

Como visto em [?], a API de reflection é composta de classes que representam (ou espelham) classes, interfaces e objetos na Java Virtual Machine atual. Esta API é importante no desenvolvimento de ferramentas como debuggers, navegadores de classes e construtores de GUI (Graphical User Interface - Interface gráfica de usuário).

Com esta API, é possível:

- Determinar a classe de um objeto.
- Recuperar informações sobre modificadores de classe, campos, métodos, construtores e superclasses.
- Descobrir quais constantes e declarações de métodos pertencem a uma interface.
- Criar uma instância de uma classe cujo nome não é conhecido até o tempo de execução.
- Recuperar e modificar o valor de um campo de um objeto, mesmo que o nome do campo seja desconhecido para o seu programa até o tempo de execução.

- Invocar um método de um objeto, mesmo que o nome do método seja desconhecido até o tempo de execução.
- Criar um novo vetor, cujo tamanho e tipo não são conhecidos até o tempo de execução, e então modificar os componentes do vetor.

Utilizamos os recursos da API para criar objetos de classes a partir do seu nome em tempo de execução, para descobrir métodos e campos de uma classe e para recuperar informações sobre modificadores de classe e métodos, além de invocar métodos de uma classe. Esses recursos foram utilizados, junto com annotations, para criar uma interface de usuário que se auto-constrói de acordo com o objeto que a utiliza, que é a classe `ConfigurationPanel`. Esta classe é importante para criar uma interface de usuário extensível e remover das outras classes a responsabilidade de criar cada uma a sua interface de configuração.

3.1.2 Annotation

Como visto em [?], annotations são um tipo especial de modificadores e podem ser utilizados onde qualquer outro modificador (como `public`, `static` ou `final`) poderia ser usado. Por convenção, as annotations precedem os outros modificadores. Annotations consistem de um sinal de arroba (@) seguido de um tipo de annotation e uma lista parentizada de pares nome-valor. Esses valores têm que ser constantes em tempo de compilação. Um tipo de annotation sem elementos é considerada um tipo de annotation marcadora.

Annotations são um complemento do javadoc, e podem ser utilizadas em tempo de execução, utilizando reflection. Fazemos este uso de annotations para definir propriedades de exibição nas classes que irão utilizar a interface de usuário auto-configurável e para marcar os métodos que permitem modificar e recuperar os valores de propriedades das classes que utilizam essa interface.

3.1.3 JAMA

Como visto em [?], JAMA é um pacote de álgebra linear para Java. Executa diversas operações com matrizes, como soma, subtração, cálculo da inversa e do determinante. Utilizamos este pacote para fazer cálculos com matrizes para não ter que criar uma biblioteca própria do zero. Utilizamos ao implementar o Gustafson-Kessel e o Gath-Geva.

3.1.4 Apache commons math

Como visto em [?], Commons Math é uma biblioteca que faz parte do Projeto Jakarta, da Apache. É uma biblioteca de matemática de componentes de matemática e estatística que resolve a maioria dos problemas comuns não disponíveis na programação Java ou na biblioteca Commons, também da Apache. Utilizamos o pacote de geração de dados, que inclui utilitários para:

- geração de números aleatórios
- geração de strings aleatórias
- geração de seqüências criptograficamente seguras de números ou strings
- geração de amostras aleatórias e permutações
- análise de distribuições de valores em um arquivo de entrada e geração de valores parecidos com os valores no arquivo
- geração de dados para distribuições de freqüências agrupadas ou histogramas.

Utilizamos a geração de amostras aleatórias para fazer o gerador de dados aleatórios.

3.1.5 Painel auto configurável - ConfigurationPanel

Para que fosse mais fácil a criação das classes e posterior extensão do programa, optamos pela criação de uma classe que estende JPanel - um painel utilizado na criação de interfaces gráficas e faz parte do Swing - e cria e configura seus próprios componentes gráficos e de entrada de dados de acordo com o objeto que recebe.

Esta classe tem como objetivo gerar uma interface gráfica que permita modificar propriedades do objeto que recebe. Para isto, foram utilizadas duas APIs do Java: Reflection, visto na seção ?? e Annotation, visto na seção ??.

Criamos annotations que indicam os métodos que devem ser utilizados para configurar uma propriedade (o seu “getter” e o seu “setter”). Foram criados alguns tipos diferentes de annotations para as diversas entradas de dados:

@PublicClassProperty é utilizada para configurar um campo que espera um objeto complexo. Este objeto tem que ser configurado também, portanto, ao utilizar esta

“annotation”, painéis são criados recursivamente para que a configuração do objeto seja completada.

O que é escolhido nesta “annotation” é, na verdade, a classe que implementa a interface ou estende a classe do objeto alvo. Estas classes são armazenadas em um arquivo xml de configuração do programa. As classes encontradas para essa propriedade são então mostradas ao usuário como uma “combo box” e, ao mudar a seleção, o objeto é escolhido, instanciado e seu painel de configuração é criado.

Ao escolher a classe, o objeto é instanciado sem parâmetros, utilizando reflection e um painel é criado para a configuração das suas propriedades, se necessário.

@PublicValueProperty é utilizada para configurar um campo que espera um valor primitivo (string, int, etc). O método setter tem que receber uma string e fazer o parse. O getter pode retornar qualquer objeto, já que ConfigurationPanel irá exibir a string obtida a partir do método toString, existente em qualquer objeto.

@PublicArrayProperty é utilizada para configurar vetores de valores primitivos. Pode-se configurar mais de um vetor ao mesmo tempo, já que ConfigurationPanel cria uma tabela, onde cada linha é uma posição e cada coluna é um vetor diferente. Todos os vetores devem possuir o mesmo tamanho, que é controlado por outra propriedade, marcada com a annotation @ArrayPropertiesSize, que marca o tamanho do vetor.

Existem ainda outras annotations utilizadas por ConfigurationPanel, como @PublicName, que indica um nome para a exibição daquela classe ou propriedade.

3.1.6 Tipos de dados

Esta foi a parte mais complexa da modelagem, pois foi necessário unir em uma mesma representação vários tipos de dados diferentes. Primeiramente, tentamos transformar todos os tipos de dados em números reais e calcular a distância somente subtraindo os números, mas essa estratégia não deu certo porque resultava em distâncias erradas e não era extensível, dado que poderia ter a necessidade de representar um tipo novo que não podia ser representado como número real e ter a sua distância calculada dessa forma. Optamos então por criar classes que realizam todas as operações entre esses tipos e armazenar os valores como um Object, que é a classe da qual todas as outras herdam e,

assim, pode conter qualquer classe, deixando a representação interna do tipo totalmente por conta da classe que implementa as operações.

Para que fosse extensível e mais tipos de dados pudessem ser adicionados à medida do necessário, foi criada uma classe abstrata, com os dados comuns a todos os tipos de dados e as operações que todos devem implementar. Essa classe e as suas implementações serão descritas a seguir.

DataSolver Classe abstrata com todas as operações necessárias aos algoritmos de clusterização, à leitura dos dados e exibição dos resultados. Tem como campos:

name - campo com getter getName e setter setName Variável que guarda o nome dos dados desse tipo. Utilizado para exibir os dados em gráficos, identificando-os pelos nomes nas coordenadas.

valueList - campo com getter getValueList e setter setValueList Lista de valores em forma de strings que representam os possíveis valores que a variável pode ter, caso aplicável. Utilizado para variáveis de tipo nominal, binário, ordinal e qualquer outra que possa ter somente valores limitados e conhecidos.

E tem como métodos utilizados para a conversão de valores a partir de, ou para outros formatos e para a identificação dos tipos de valores que fazem sentido para o tipo:

Object parse(String value) Método abstrato (não implementado) que interpreta uma string e retorna um objeto contendo a representação interna do tipo de dado correspondente à string. Todas as classes que implementam um tipo de dados devem implementar este método, que é usado para ler os dados de diversas fontes.

String toString(Object value) Método abstrato que faz a operação contrária do parse: transforma da representação interna para uma string.

Double toDouble(Object value) Método abstrato que transforma a representação interna em um número real, para fins de exibição, não de cálculo. Por exemplo, se a amostra tem uma lista de valores possíveis, este número pode ser o índice nesta lista. Utilizado para desenhar gráficos apenas, já que esses valores não tem nenhum significado numérico em relação ao valor real.

Object toObject(Double value) Método abstrato que transforma um número real qualquer em um objeto na representação interna do tipo. Só é válido para tipos

numéricos, que interpretam valores reais como entrada. Em outros tipos, retorna null.

boolean canShowMode() Método abstrato para indicar se a moda faz sentido como valor representativo para este tipo de dados.

boolean canShowMean() Método abstrato para indicar se a média faz sentido como valor representativo para este tipo de dados.

boolean canShowStandardDeviation() Método abstrato para indicar se o desvio padrão faz sentido para este tipo de dados. Para alguns tipos de dados não numéricos, como nominais, o desvio padrão não faz sentido.

boolean isNumeric() Método abstrato para indicar se a variável é numérica. Variáveis numéricas retornam null em todos os métodos exclusivamente numéricos e não são padronizadas, já que não têm escala. Útil para identificar uma variável não numérica de antemão para evitar lidar com valores nulos.

Existe também um conjunto de métodos que executam operações estatísticas nos dados. A classe acumula os valores recebidos através de uma função e fornece como resposta desvio padrão, média, moda, mínimo e máximo. Os métodos pertencentes a este conjunto são:

void clearValues() Inicializa a lista de valores para a acumulação dos mesmos e cálculo das estatísticas.

void addValue(Object value) Adiciona um valor no cálculo de estatísticas.

Object mode() Retorna o objeto mais frequente dentre os que foram adicionados com o método addValue, ou seja, a moda.

Double DoubleMean() Retorna a média numérica dos objetos adicionados. Retorna null se o tipo não for numérico.

Object ObjectMean() Retorna a média na forma da representação interna do tipo. Retorna um valor representativo. Se a média não fizer sentido para o tipo, retorna a moda ou alguma outra medida que faça sentido como valor representativo.

Double getMin() Retorna o valor numérico mínimo para os valores adicionados. Retorna null se o tipo não for numérico.

Double getMax() Retorna o valor numérico máximo para os valores adicionados. Retorna null se o tipo não for numérico.

O cálculo da distância entre dois valores só pode ser feito através da classe, dado que só esta sabe qual é a representação interna do objeto e como calcular a distância. Então, cada classe que implementa um tipo deve implementar o método a seguir:

double distance(Object value1, Object value2) Método abstrato que calcula a distância entre dois objetos do mesmo tipo. É a base para o cálculo da distância entre amostras multidimensionais, com objetos de vários tipos diferentes.

Descritos os métodos que cada classe que implementa um tipo deve ter, descrevemos a seguir como as classes tratam os métodos e como é feita a representação interna do tipo.

IntervalScaledDataSolver Esta classe implementa as operações relativas ao tipo numérico intervalar. Como o tipo é simplesmente um número real, é representada desta forma, utilizando o tipo Double e, conseqüentemente, o cálculo da distância, da média, do desvio padrão e da distância são triviais. Nesse tipo, a média e o desvio padrão fazem sentido mas a moda não. A média é calculada segundo a seguinte fórmula:

$$\frac{1}{n} \sum_{i=0}^{n-1} x_i$$

O desvio padrão utiliza a seguinte fórmula:

$$\sqrt{\frac{n \times \sum_{i=0}^{n-1} x^2 - \left(\sum_{i=0}^{n-1} x \right)^2}{n \times (n - 1)}}$$

Por fim, a distância é calculada de acordo com a seguinte equação:

$$d(x, y) = |x - y|$$

RatioScaledDataSolver Esta classe implementa as operações relativas ao tipo numérico exponencial. A representação interna e o cálculo dos valores é semelhante ao tipo numérico intervalar, porém na leitura se executa a transformação dos valores pelos seus logaritmos. Portanto, uma variável desse tipo com valor x é igual a uma variável numérica intervalar com valor $y = \ln x$. A representação gráfica também é logarítmica para que outras variáveis não fiquem fora de escala.

NominalDataSolver Esta classe implementa as operações relativas ao tipo nominal. É a base para o tipo binário, devido à semelhança entre as duas classes. A representação interna do tipo foi bastante discutida. Consideramos três opções:

Representação direta armazenar na variável o índice do valor contido em relação ao vetor de valores da classe. Porém, esbarramos com um problema: diversos algoritmos utilizam a média das amostras como centro e utilizam a distância de uma amostra para o centro do seu cluster em seus cálculos. Porém, essa representação não permitia representar uma média de forma verdadeira, somente aproximada. Isso poderia interferir nos resultados e dificultar diversos cálculos.

Representação em array booleano armazenar um array de tamanho igual ao número de possíveis valores que a variável pode assumir com valores booleanos indicando se a variável tem o valor correspondente àquela posição ou não. Como na solução acima, esbarramos com o problema da média.

Representação em array de números reais semelhante à solução acima, mas utiliza-se valores reais entre 0 e 1 ao invés de booleanos para representar o grau de inclusão daquela variável em cada valor. É uma solução que utiliza lógica fuzzy e faz com que a média seja muito melhor representada e a distância entre um centro e uma amostra melhor calculada. Para obter o valor que a amostra contém seleciona-se o com maior grau de inclusão.

Utilizamos a última representação e calculamos a média da seguinte forma:

$$\bar{\mu}_i = \frac{1}{n} \sum_{j=0}^{n-1} \mu_{ji},$$

onde μ_i é o grau de pertinência da variável ao valor da posição i do vetor de valores possíveis.

Por fim, a distância é calculada da seguinte forma:

$$d(x, y) = \frac{1}{2} \sum_{i=0}^{n-1} |\mu_{xi} - \mu_{yi}|$$

Cada diferença é contada duas vezes, portanto divide-se o resultado por dois para que a distância fique entre 0 e 1.

BinaryDataSolver Esta classe implementa as operações relativas ao tipo binário e é uma subclasse de `NominalDataSolver`. A representação interna é a mesma do tipo nominal e o cálculo da média e da distância são iguais. Porém, o tipo binário pode ser considerado um tipo numérico e tem algumas operações que o tipo nominal não tem.

3.1.7 Cálculo de distância entre amostras

O cálculo da distância entre as amostras é feito unindo-se as distâncias unidimensionais de acordo com alguma fórmula. Diversas fórmulas podem ser utilizadas, então foi criada uma interface para que diversas classes possam implementar o método de cálculo de distância.

Considerando c o número de características (dimensões) das amostras e $d(x_i, y_i)$ a distância unidimensional entre os valores da característica i das amostras x e y , listamos as classes que foram implementadas a seguir.

ManhattanDistance Calcula a distância de Manhattan, utilizando a seguinte equação:

$$D(x, y) = \sum_{i=0}^{c-1} d(x_i, y_i)$$

EuclideanDistance Calcula a distância euclidiana, utilizando a seguinte equação:

$$D(x, y) = \sqrt{\sum_{i=0}^{c-1} d(x_i, y_i)^2}$$

PNormDistance Generalização da distância euclidiana para um número natural qualquer p . Na verdade, até mesmo a distância de Manhattan pode ser considerada um caso particular, se utilizarmos $p = 1$. A equação utilizada é mostrada a seguir:

$$D(x, y) = \sqrt[p]{\sum_{i=0}^{c-1} d(x_i, y_i)^p}$$

3.1.8 Representação das amostras

Uma amostra consiste de um conjunto de características, cada uma com um tipo de dado. Para simplificar o uso, optamos por representar uma amostra como um vetor de características, com as características na mesma ordem que a de leitura.

Um conjunto de amostras também é representado, por praticidade, como um vetor de amostras. Como amostra já é um vetor, o conjunto de amostras passa então a ser uma matriz de características.

É necessário acompanhar as amostras dos DataSolvers correspondentes aos tipos das características. Portanto, é necessário criar uma classe para comportar os dados completos necessários ao processo de clusterização.

Além dos DataSolvers e das amostras, é interessante também armazenar a matriz de graus de inclusão e os centros (cada um é um vetor de características, assim como as amostras) para a exibição dos resultados. Como as amostras podem passar por um processo de padronização, é interessante armazenar o resultado da padronização em uma variável separada, para que a exibição dos resultados utilize os dados originais.

Criamos então a classe ClusteringData que reúne todos os dados necessários à clusterização e exibição dos resultados que contém:

data dados originais.

standardizedData dados após passar pela padronização.

dataSolvers classes que executam as operações nos dados.

clusteringCenters centros dos clusters utilizados nos algoritmos. São gerados a partir de dados padronizados.

nonStandardizedCenters centros dos clusters para a exibição. São gerados a partir dos dados originais utilizando o mesmo cálculo utilizado para gerar os centros dentro de cada algoritmo. Cada algoritmo é responsável por gerar os dois centros: padronizado e não padronizado para que a exibição dos centros não padronizados corresponda ao centro utilizado no algoritmo.

membershipMatrix matriz de graus de inclusão de cada amostra em cada cluster.

3.1.9 Padronização

Variáveis numéricas podem interferir no cálculo das distâncias devido à unidade de medida. Portanto, para essas variáveis pode ser necessário executar uma padronização previamente à aplicação do algoritmo. Existem alguns tipos de padronização e implementamos os dois que são mais utilizados:

ZScoreStandardization Classe que implementa a padronização Z-score da forma explicada na seção ???. Como já dito anteriormente, o algoritmo salva os dados padronizados em `standardizedData` no objeto da classe `ClusteringData` recebido. A média e o desvio padrão são calculados utilizando-se os `DataSolvers` de cada característica e aplica-se a fórmula mostrada na seção ??.

MinMaxNormalization Classe que implementa a normalização Min-Max da forma explicada na seção ???. O valor mínimo e máximo são calculados pelos `DataSolvers` de cada característica e aplica-se a fórmula mostrada na seção ??, preservando-se os dados originais e salvando o resultado em `standardizedData` no objeto da classe `ClusteringData` recebido.

3.1.10 Leitura dos dados

Disponibilizamos de uma forma de leitura de dados que consiste em um arquivo XML de descrição dos dados e possivelmente outros arquivos contendo os dados. Basicamente existem três formas de leitura de dados, indicadas através de um elemento chamado `source`. Utilizamos classes que estendem uma classe abstrata e um `Factory` para criar as classes a partir do XML. De acordo com o conteúdo de `source`, uma das três classes é criada:

InlineDataSource Classe que lê as amostras diretamente do XML, de acordo com o formato indicado no arquivo XSD (mostrado na figura ??).

CsvDataSource Lê os dados a partir de um arquivo em separado, no formato CSV (valores separados por vírgula). Conta o número de características e lê valores de cada linha até que este valor seja alcançado.

Fontes de dados feitas pelo usuário O usuário pode criar uma classe que lê os dados de uma fonte qualquer. Neste caso, ele irá especificar neste campo o nome completo da classe (com informação de pacote). Detalhes sobre a criação dessa classe são explicados na seção ??

3.1.11 Algoritmos

Os algoritmos foram implementados da forma detalhada na seção ?? e em suas sub-seções. Iremos apenas descrever alguns detalhes de implementação e complexidade relativas aos algoritmos implementados.

Os algoritmos foram implementados de forma hierárquica, de forma a reunir características comuns em classes abstratas. Todos os algoritmos implementam a interface Clusterer.

Todos os algoritmos são utilizados da mesma forma: o método `init` é chamado e depois são chamados os métodos `step` e `updateCenters`, seguidos de `isDone`. Quando `isDone` retornar `true`, a execução é terminada.

Clusterização hierárquica aglomerativa

A clusterização hierárquica aglomerativa é implementada pela classe `AgglomerativeHierarchicalClusterer`. As distâncias entre cada par de amostras é calculada no início do algoritmo (no método `init`). Considerando n o número de amostras e c o número de clusters desejados, o algoritmo leva exatamente $n - c$ passos (chamadas ao método `step`) para executar. Devido a estas características, o algoritmo não é indicado para um número de amostras muito grande, já que ocupa $O(n^2)$ de espaço e leva muitos passos para concluir. Pode trabalhar com dados numéricos ou não, já que usa apenas a distância entre amostras como medida e não assume nenhum tipo específico sobre as características da amostra. O algoritmo é determinístico, já que sempre efetua as mesmas decisões para um mesmo conjunto de amostras.

Clusterização hierárquica divisiva

A clusterização hierárquica divisiva é implementada pela classe `DivisiveHierarchicalClusterer`. Assim como no algoritmo hierárquico aglomerativo, as distâncias são calculadas no início do algoritmo. O número de passos é muito menor, mas a complexidade de espaço

ainda impede que o algoritmo seja utilizado para massas de dados muito grandes. Este algoritmo também é determinístico.

K-Medoids

O algoritmo K-Medoids foi implementado pela classe `KMedoidsClusterer`. Este algoritmo também calcula as distâncias entre todas as amostras e armazena, tornando-o ruim para massas de dados muito grandes devido à complexidade de espaço. O número de passos em geral é pequeno, mas é variável, já que depende exclusivamente das amostras utilizadas. Assim como os algoritmos hierárquicos apresentados, este também é determinístico.

K-Means

O algoritmo K-Means foi implementado pela classe `KMeansClusterer`. Ao contrário dos algoritmos hierárquicos e do K-Medoids, este algoritmo não é determinístico. O algoritmo começa com uma distribuição inicial aleatória e tenta melhorar os clusters alocando as amostras para o cluster do centro mais próximo e calcula novos centros. Como só são calculadas distâncias entre as amostras e os centros, a complexidade de espaço pode cair bastante, considerando que o número de clusters costuma ser muito menor do que o número de amostras. Os centros mudam a cada iteração, portanto as distâncias têm que ser recalculadas a cada iteração. O algoritmo tem um ganho no armazenamento mas uma possível perda em desempenho. A perda será menor quanto menor for o número de clusters, já que a complexidade a cada passo é da ordem de $O(nc)$, onde n é o número de amostras e c é o número de clusters. O número de passos é variável e depende de diversos fatores, como o conjunto de amostras, a alocação inicial das amostras e a norma utilizada.

C-Means

O algoritmo C-Means foi implementado pela classe `CMeansClusterer` e tem uma estrutura muito parecida com o K-Means, mudando somente a forma do cálculo do grau de inclusão, que no C-Means é fuzzy, ao contrário do K-Means. Implementamos então uma classe com as operações em comum dos dois algoritmos - `MeansClusterer` - e tanto `KMeansClusterer` quanto `CMeansClusterer` herdam desta classe, implementando somente as operações que são diferentes entre os dois algoritmos.

Gustafson-Kessel

O algoritmo Gustafson-Kessel foi implementado pela classe `GustafsonKesselClusterer`. A utilização dos dados pela classe é bem diferente da utilizada pelo C-Means, já que o Gustafson-Kessel utiliza os valores das amostras e não apenas a distância entre elas para o cálculo. A inicialização do algoritmo é diferente, já que ele só utiliza as características numéricas, e o cálculo da distância também é diferente, utilizando operações em matrizes para o cálculo das distâncias entre cada amostra e os centros dos clusters.

Gath-Geva

O algoritmo Gath-Geva foi implementado pela classe `GathGevaClusterer`. O algoritmo é semelhante ao Gustafson-Kessel, mudando apenas o cálculo da distância. Portanto, `GathGevaClusterer` estende `GustafsonKesselClusterer`, modificando apenas o método de cálculo de distância.

3.1.12 Validação

Os métodos de validação devem implementar a interface `Validator`. Esta interface é muito simples, contendo apenas um método que recebe `ClusteringData` e retorna um `double`. Implementamos dois métodos de validação, explicados a seguir.

Inter Class Contrast - ICC

Este método foi implementado pela classe `ICCValidator` e efetua os cálculos demonstrados na seção ???. A implementação é simples e não foi necessária nenhuma biblioteca especial.

Compactness and Separation - CS

Este método foi implementado pela classe `CSValidator` e efetua os cálculos demonstrados na seção ???. Também não foi necessária nenhuma biblioteca especial para a implementação.

3.1.13 Implementação da interface gráfica

Basicamente foram implementadas três classes principais para a interface gráfica, com algumas outras classes que as auxiliam. A maior parte da interface gráfica foi feita no Visual Editor, uma ferramenta visual de criação de interfaces gráficas do Eclipse. A configuração

das classes necessárias à execução do programa foi feita utilizando `ConfigurationPanel`, descrito na seção ???. Existe ainda a representação gráfica dos resultados, que foi feita pela classe `GraphicPanel`.

Para representar graficamente os resultados, optamos por utilizar apenas duas dimensões, ficando a cargo do usuário escolher duas características. Existe um tratamento especial para alguns tipos de dados, que veremos a seguir.

Tipo numérico exponencial (ratio-scaled) Os dados são plotados com o o logaritmo natural do valor real. Seria inviável plotar os valores em escala exponencial, dado que a outra característica ficaria fora de escala.

Tipos nominais (binário, nominal e ordinal) Ao ler estes tipos, cada valor recebe um índice e nem sempre a ordem importa. A abordagem utilizada para estes casos é plotar no gráfico todos os valores daquele tipo no eixo do gráfico e as amostras serão alinhadas com o seu valor para aquela característica, como visto na figura ???.

O tipo numérico intervalar é plotado da forma usual, estando padronizado ou não. Os valores utilizados para a exibição são sempre os originais e, no caso dos centros, é utilizado um valor próximo, já que os centros podem ser médias ponderadas e muitas vezes o valor da média não faz sentido ao ser exibido.

Os clusters são representados por cores e o centro de cada cluster é representado por um 'x' da cor do seu cluster. Utilizamos um número limitado de cores para não dificultar a visualização, mas se o número de clusters for maior do que o número de cores, estas se repetirão. Achemos, no entanto, que esta abordagem se mostrou melhor do que gerar mais cores, que se confundiam facilmente com outras cores já utilizadas.

Os nomes das coordenadas utilizadas - o nome da característica de acordo com o arquivo de dados - estão ao lado de cada eixo, para que a identificação seja fácil.

3.2 Extensão

Existem diversas formas de estender o programa. Algumas formas - especificamente as classes que o usuário escolhe através de caixas de combinação no programa - deve-se, além de implementar uma interface ou estender uma classe, adicionar uma tag no arquivo de configuração do programa.

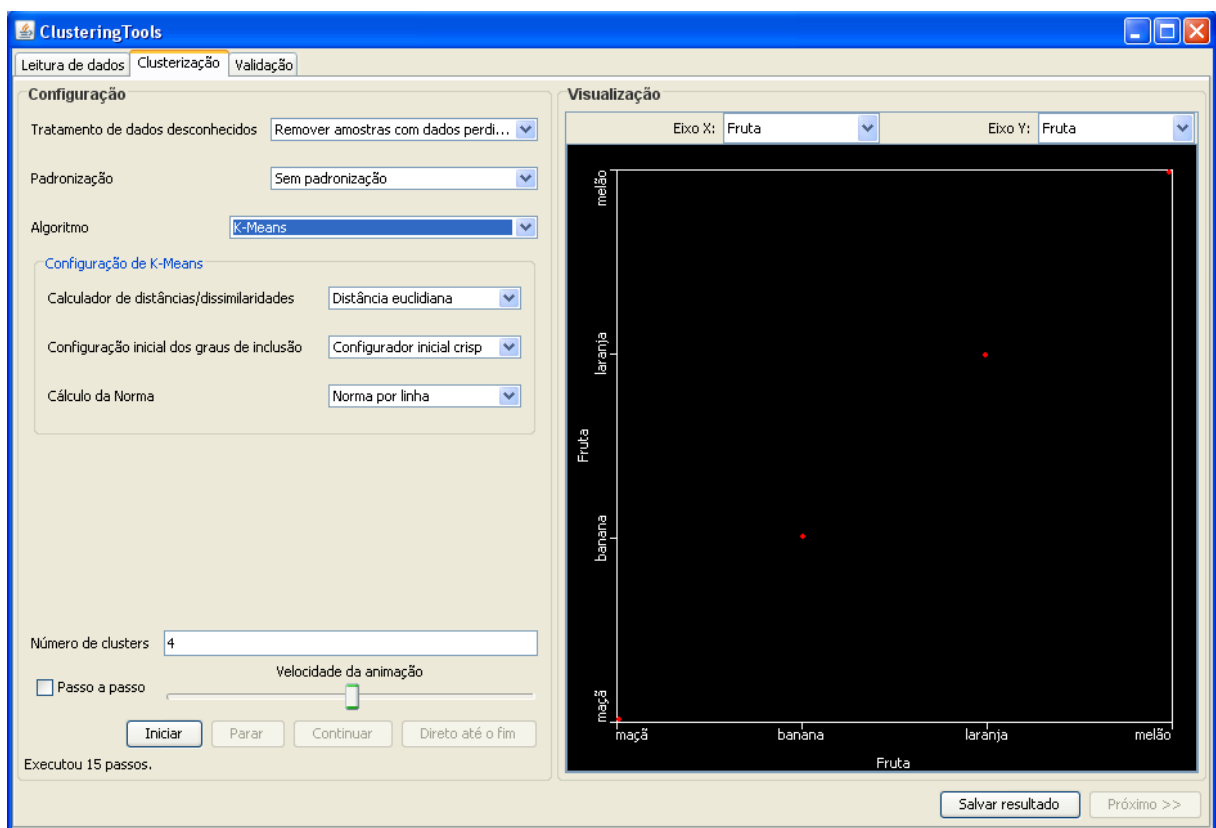


Figura 3.1: Gráfico com dados nominais

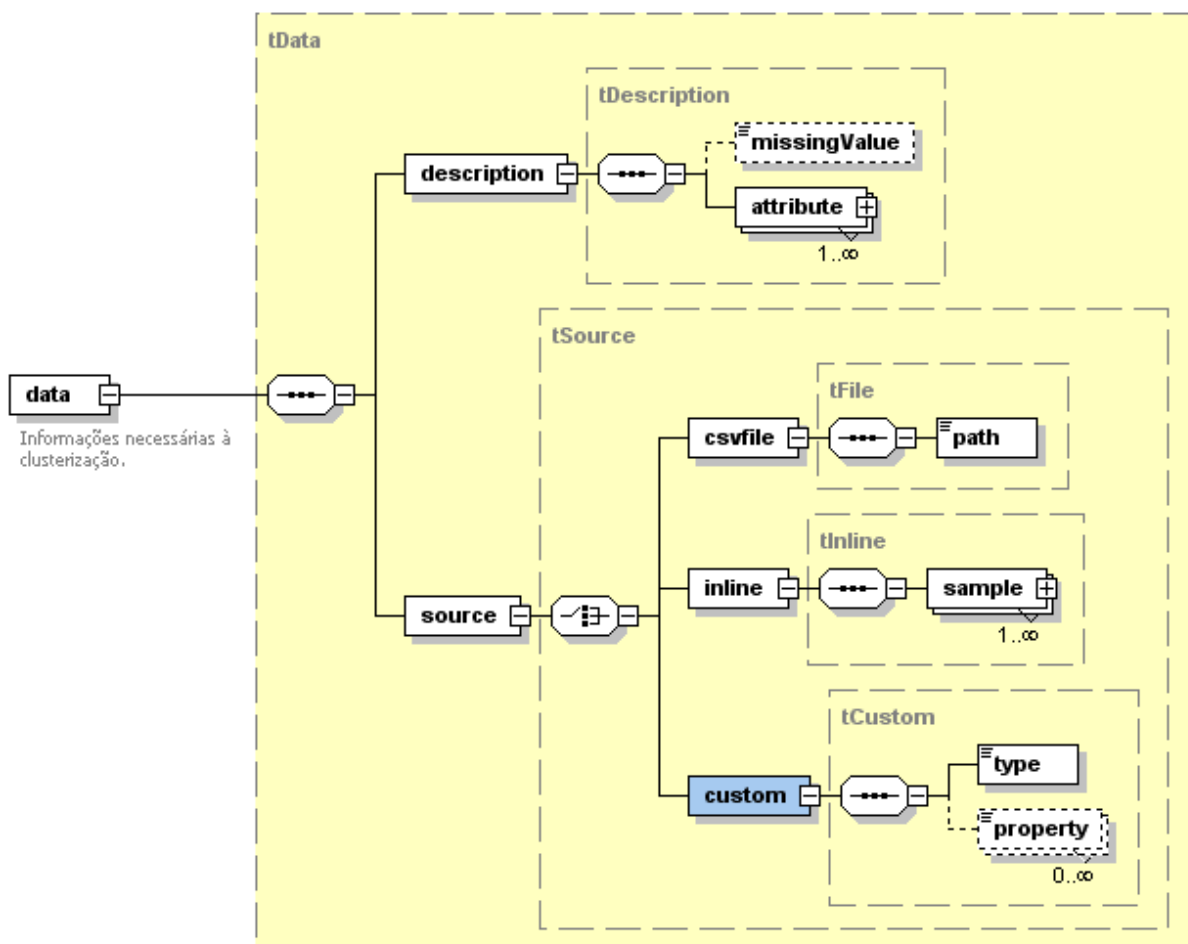


Figura 3.2: XSD do arquivo de entrada de dados

As classes criadas sempre podem ter propriedades a ser configuradas pelos usuários, que serão exibidas utilizando-se a classe `ConfigurationPanel`. Para que estas propriedades sejam exibidas, elas têm que ser marcadas com as annotations citadas na seção ???. Segue então as formas de extensão possíveis.

3.2.1 Leitura de dados

A leitura de dados pode ser estendida utilizando a tag `<custom>` do arquivo XML. Para que isso ocorra, deve-se criar uma classe que implementa a interface `DataSource` e colocar o nome da classe (com a informação de pacote completa na tag `<type>`, como visto no xsd na figura ???. Como visto, cada classe pode ter propriedades que são configuradas pelo usuário pelas tags `<property>`.

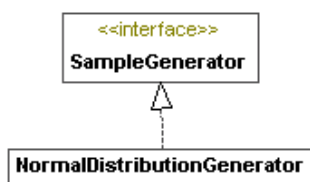


Figura 3.3: Diagrama de classes para a geração de amostras aleatórias

A interface `DataSource` consiste de dois métodos:

readNode Executa a configuração inicial do `DataSource`. Fornece à classe informações como o diretório do arquivo XML, o nó com as configurações (properties) e o número de características que cada amostra tem.

getSamples Método para recuperar os dados lidos.

Pode-se implementar, por exemplo, uma classe que lê os dados de um banco de dados. Suas propriedades poderiam ser as informações necessárias para conectar ao banco e a string com um SQL que retornaria os dados para o programa através de uma tabela. A classe de leitura de CSV poderia ter sido implementada desta forma também. É uma importante forma de extensão do programa.

3.2.2 Geração de amostras aleatórias

O gerador de amostras aleatórias pode ser estendido implementando-se a interface `SampleGenerator`. Pode-se ver o diagrama de classes na figura ???. Deve-se também adicionar uma tag `<generator>` ao arquivo de configuração do programa com o nome da classe. A interface `SampleGenerator` consiste em apenas um método: *generateSamples*, que não recebe argumentos e retorna `ClusteringData`.

3.2.3 Tipos de dados

Um tipo de dados novo pode ser criado simplesmente implementando-se uma classe que estende a classe abstrada `DataSolver`. Ao utilizar esta classe, deve-se incluir o nome completo da classe, com informação de pacote, no arquivo de descrição dos dados. Pode-se ver o diagrama de classes na figura ???.

Para estender esta classe é necessário implementar os métodos descritos na seção ???

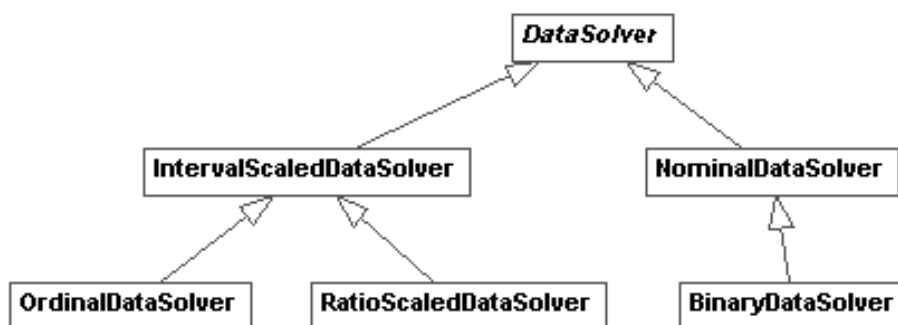


Figura 3.4: Diagrama de classes para os tipos de dados

3.2.4 Remoção de dados desconhecidos

Pode-se estender a remoção de dados desconhecidos. Para isso, deve-se implementar uma classe que implementa `MissingDataProcessor` (ou estende `DoNothingMissingDataProcessor`, que já implementa a transformação dos dados recebidos nos dados retornados), como visto no diagrama de classes da figura ?? e deve-se incluir uma tag no arquivo de configuração do programa (com o nome de `missingProcessor`, obedecendo às regras do XSD na figura ??).

A interface tem apenas um método, que recebe uma matriz de `Object` e um vetor de `DataSolver` e tem que retornar um `ClusteringData`. Os dados com os valores desconhecidos substituídos devem ser atribuídos a `data` de `ClusteringData`. Os `DataSolvers` devem ser atribuídos a `dataSolvers` de `ClusteringData`. A classe pode ter uma annotation `@PublicName` para ter um nome de exibição diferente do nome da classe.

3.2.5 Padronização

Para adicionar mais algoritmos de padronização das amostras, segue-se um processo semelhante ao da remoção de dados desconhecidos. A classe deve implementar a interface `Standardizator` ou estender qualquer classe que implemente esta interface, como no diagrama de classes da figura ?. É necessário também adicionar ao arquivo de configuração do programa uma tag `<standardizator>` com o nome da classe. A classe pode ter uma annotation `@PublicName` para ter um nome de exibição diferente do nome da classe.

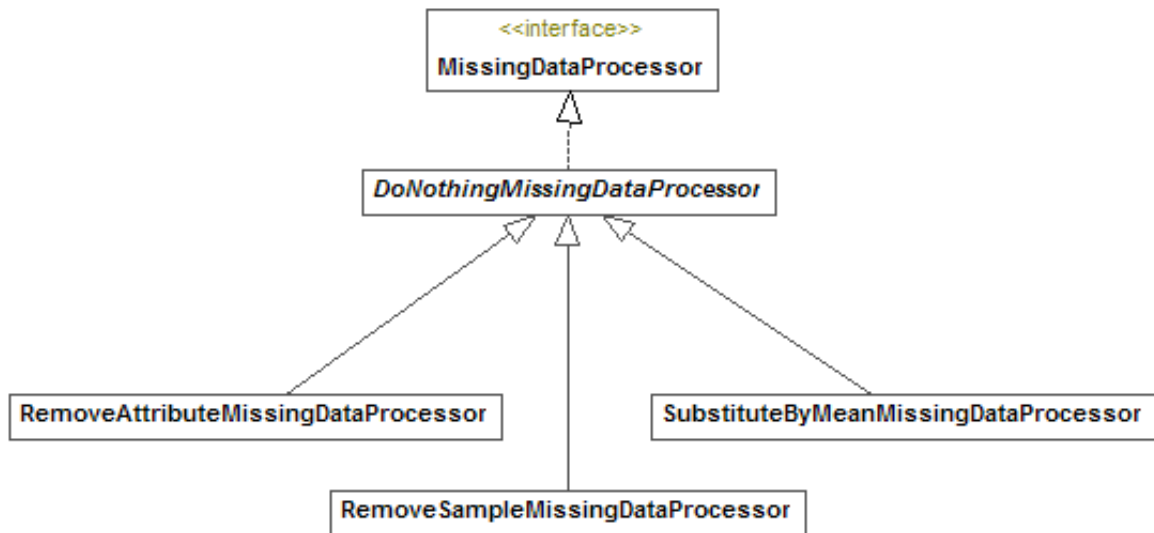


Figura 3.5: Diagrama de classes para a remoção de dados desconhecidos

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:conf="clusteringtools.ufrj.br/configuration" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="clusteringtools.ufrj.br/configuration">
  <xs:element name="configuration">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="clusterer" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="standardizator" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="validator" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="generator" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="distanceCalculator" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="initialMembershipGuesser" type="conf:tConfiguration" minOccurs="0" maxOccurs="
unbounded"/>
        <xs:element name="matrixNormCalculator" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="missingProcessor" type="conf:tConfiguration" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="tConfiguration">
    <xs:sequence>
      <xs:element name="class" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
  
```

Figura 3.6: XSD do arquivo de configuração do programa

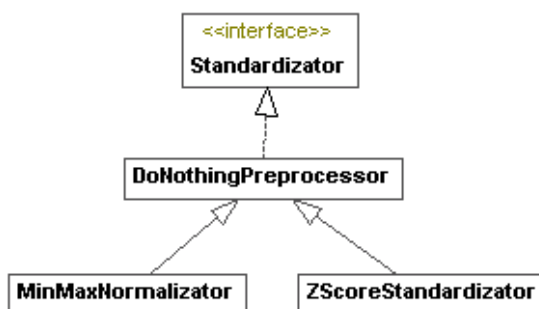


Figura 3.7: Diagrama de classes para a padronização

3.2.6 Algoritmos

Para adicionar algoritmos de clusterização novos, pode-se estender qualquer classe que implemente a interface `Clusterer`. Assim como na padronização e na remoção de dados desconhecidos, é necessário também adicionar uma tag no arquivo de configuração, mas com o nome “clusterer”. Para implementar a interface, é necessário implementar os seguintes métodos:

init inicializa o algoritmo, fornecendo os dados e o número de clusters. É o primeiro método a ser chamado na sequência da execução do algoritmo e só é chamado uma vez.

step executa uma iteração do algoritmo. Sempre é chamado pelo menos uma vez e é chamado até que uma chamada do método `isDone` retorne true.

updateCenters sempre é chamado após `step` para que o algoritmo faça a atualização dos centros após a execução de um passo.

isDone sempre é chamado após `updateCenters` para checar se ainda tem alguma iteração a ser feita ou o algoritmo já terminou.

onlyNumericData indica se o algoritmo só utiliza dados numéricos. Neste caso, o algoritmo irá ignorar todas as características que não forem numéricas (ou seja, `isNumeric` de `DataSolver` retorna false). É utilizado para mostrar ao usuário se alguma característica foi ignorada.

getData retorna os dados utilizados pelo algoritmo.

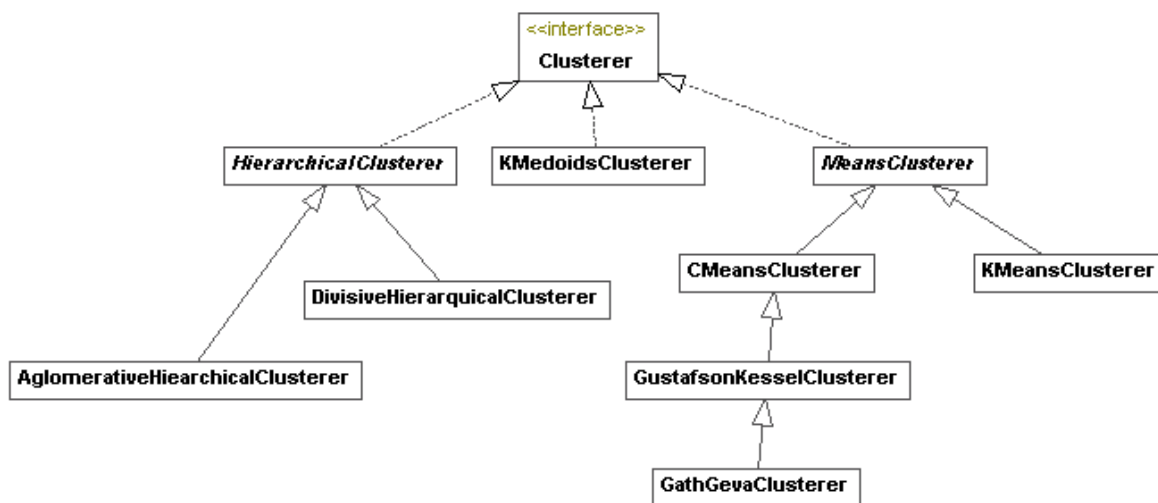


Figura 3.8: Diagrama de classes para os algoritmos de clusterização

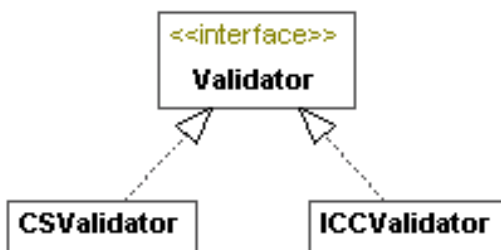


Figura 3.9: Diagrama de classes para os algoritmos de validação

3.2.7 Validações

Pode-se adicionar novos algoritmos de validação, bastando apenas implementar a interface Validator, como no diagrama de classes da figura ?? e adicionar a tag <validator> no arquivo de configuração do programa. A interface Validator possui apenas um método, *validate*, que recebe ClusteringData e retorna um valor double, que é o resultado daquele método de avaliação nos dados fornecidos.

Capítulo 4

Aplicações

Para utilização da ferramenta em sua aplicação original, o ensino, propomos dois usos diferentes: geração de exemplos e desenvolvimento de novas funcionalidades.

Exemplos podem ser criados previamente, e usados para explicitar a eficácia dos métodos e das métricas bem como seus comportamentos para diferentes entradas. Como os métodos são diferentes entre si, cada um com características e objetivos próprios, o uso da ferramenta para comparação dos resultados dos diferentes algoritmos a partir de uma determinada entrada pode ser bastante didático, trazendo uma noção mais intuitiva das especificidades de cada método. Dados reais podem ser utilizados na exemplificação de aplicações dos métodos de clusterização sem grandes dificuldades, uma vez que a ferramenta dispõe de diferentes formas de entrada de dados, possibilitando, inclusive, a implementação de novas formas, desde que sejam atendidos determinados requisitos. Há ainda a opção de geração de exemplos na própria ferramenta, com dados pseudo-aleatórios, gerados a partir de alguns parâmetros. Esta abordagem, embora mais limitada, traz flexibilidade e praticidade para o professor, que passa a poder gerar diferentes entradas durante a própria aula.

Uma outra forma de uso da ferramenta é incentivar os alunos a desenvolver novos algoritmos. Com toda a estrutura do programa pronta, o aluno passa a se focar no algoritmo, sem a necessidade de reimplementar procedimentos já existentes como entrada, pré-tratamento e saída de dados, poupando seu esforço para o aperfeiçoamento e criação de novos métodos.

Adicionalmente, embora o foco deste projeto seja o uso da ferramenta no ensino das técnicas de clusterização, considerando sua generalidade, grande parte de seu código pode ser reusado em aplicações de clusterização.

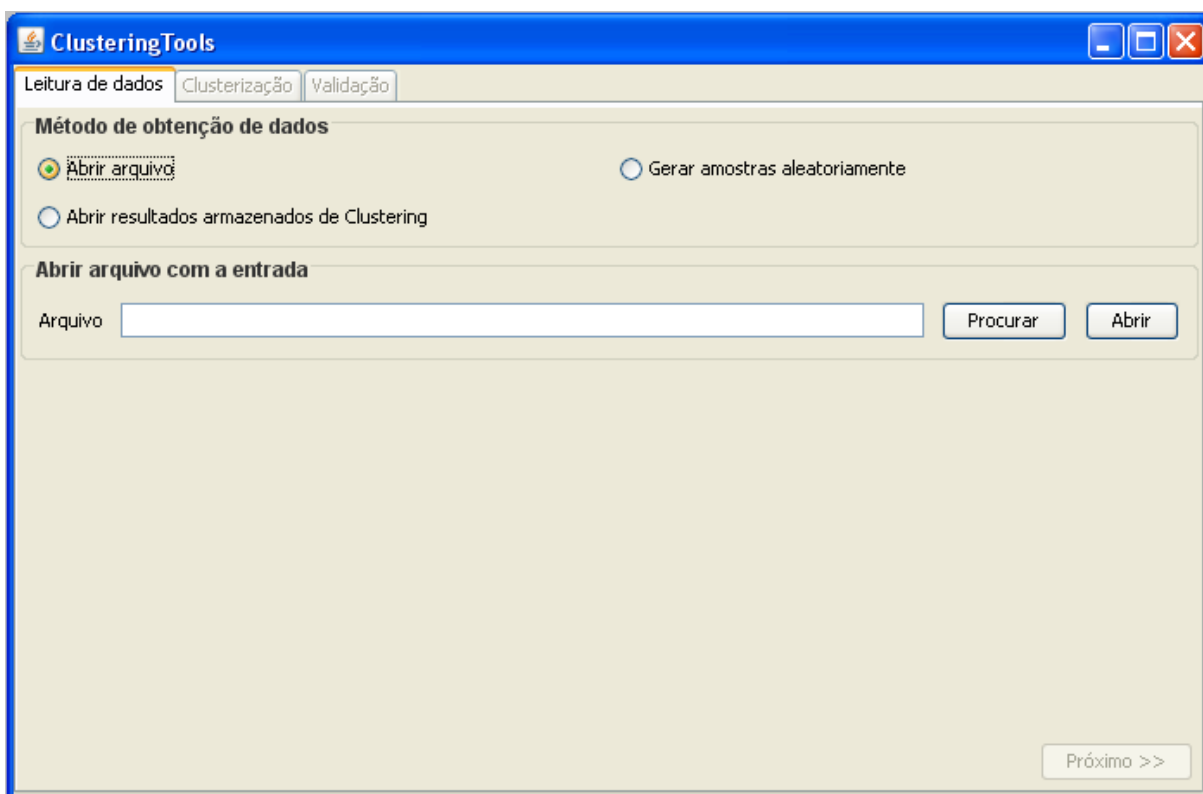


Figura 4.1: Abrindo um arquivo de entrada

4.1 Fluxo de execução do programa

Existem três etapas distintas para a execução do programa, descritas a seguir:

4.1.1 Leitura ou geração dos dados

- O usuário escolhe se deseja ler os dados a partir de um arquivo ou se irá gerar os dados aleatoriamente.
- Se o usuário optar por ler um arquivo, escolhe o arquivo no sistema de arquivos utilizando o botão Procurar, ou digita o endereço completo na caixa de texto, utilizando o botão Abrir para abrir o arquivo, como na figura ??.
- Caso o usuário opte por gerar os dados aleatoriamente, é necessário escolher o método de geração de amostras e definir as propriedades do método. Mostramos as propriedades do gerador gaussiano na figura ?. Após a geração das amostras, clicando-se no botão “Gerar amostras” pode-se salvar em um arquivo.

The screenshot shows the ClusteringTools application window. The title bar reads "ClusteringTools". There are three tabs: "Leitura de dados", "Clusterização", and "Validação". The "Clusterização" tab is active. Under the heading "Método de obtenção de dados", there are three radio buttons: "Abrir arquivo", "Gerar amostras aleatoriamente" (which is selected), and "Abrir resultados armazenados de Clustering".

Under the heading "Gerar amostras", there is a section for "Gerador de amostras aleatórias" with a dropdown menu set to "Gerador de Distribuição Normal". Below this is a sub-section titled "Configuração de Gerador de Distribuição Normal".

In this sub-section, there is a text input field for "Número de centros" containing the value "2", with an "Entrar valor" button to its right. Below this is a table with the following data:

Número de Pontos	Média de X	Desvio padrão de X	Média de Y	Desvio padrão de Y
50	10	5	60.8	1.5
75	20	2	25.4	10

Below the table is an "Entrar valores da tabela" button. At the bottom of the configuration area are "Gerar amostras" and "Salvar" buttons. At the very bottom of the window is a "Próximo >>" button.

Figura 4.2: Exemplo de configuração do gerador de distribuição normal

The screenshot shows the 'ClusteringTools' application window. The 'Estatísticas' (Statistics) section is active, displaying the following data:

Característica: X

Nome	Valor
Média	64.42890688196627
Desvio padrão	14.531117556024371

Buttons: Próximo >>

Figura 4.3: Visualização das estatísticas das amostras

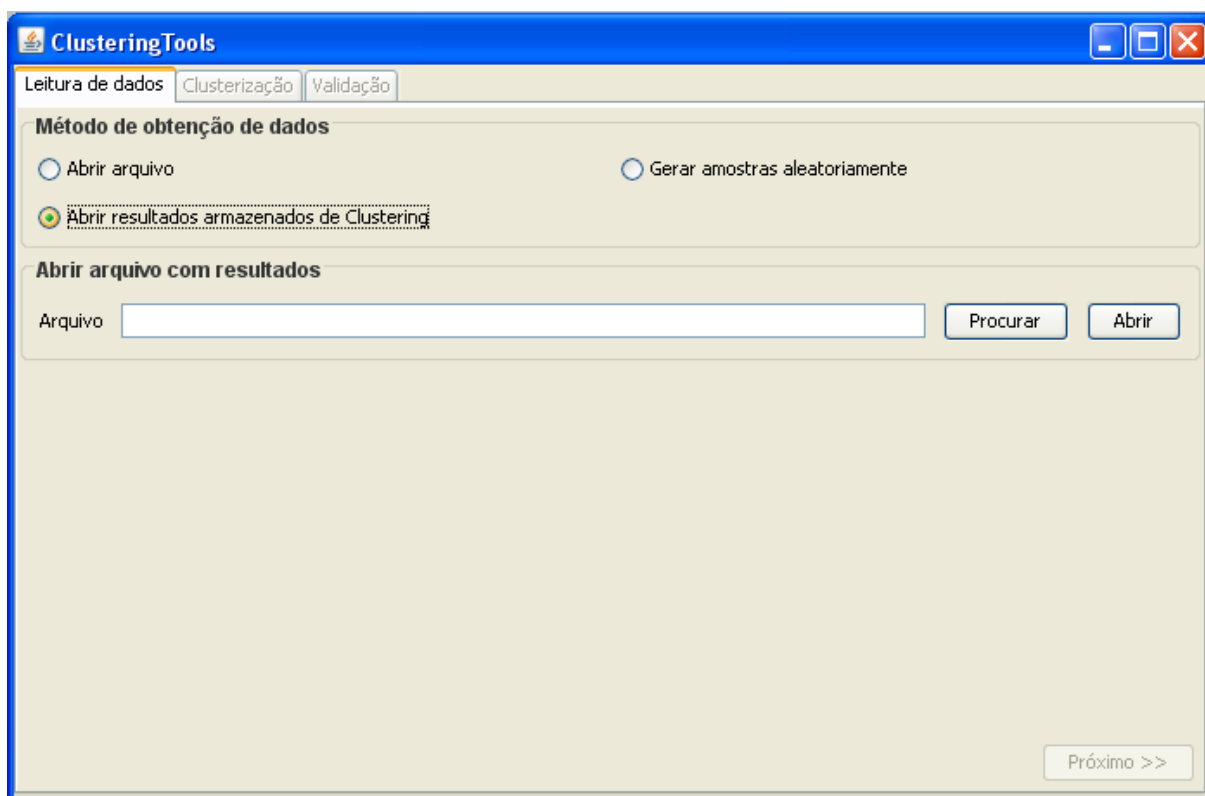


Figura 4.4: Abrindo o resultado de uma clusterização

- Após ter os dados carregados, pode-se conferir a média e o desvio padrão das amostras no painel que aparece, como na figura ??.
- Para passar ao passo de clusterização, é necessário clicar no botão “Próximo »”.
- Pode-se também carregar resultados de clusterização salvos anteriormente, escolhendo “Abrir resultados armazenados de Clustering”, como na figura ?? e escolher o arquivo com os resultados da mesma forma que se escolhe o arquivo de amostras.

4.1.2 Clusterização

- É necessário escolher o método de remoção de dados desconhecidos e a padronização. A padronização pode não ser executada, o que acontece quando se escolhe “Sem padronização” na lista de métodos de padronização disponíveis, como na figura ??.

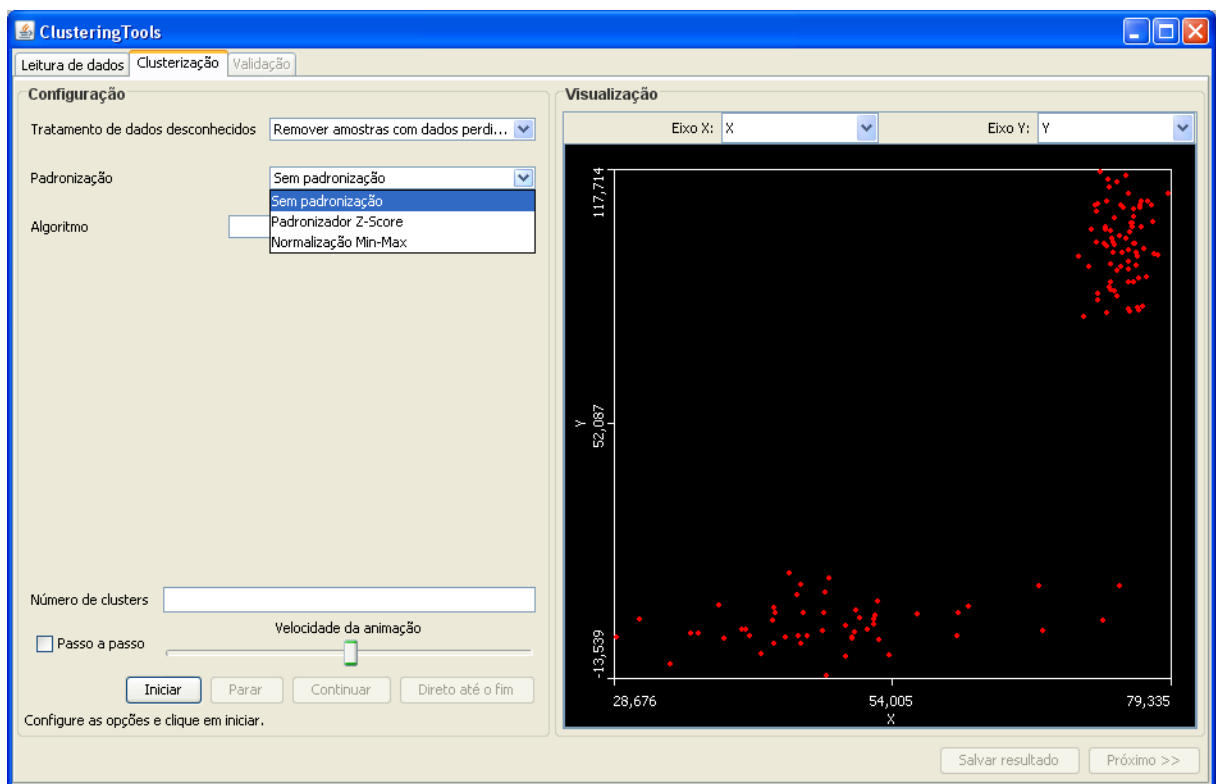


Figura 4.5: Selecionando a padronização.

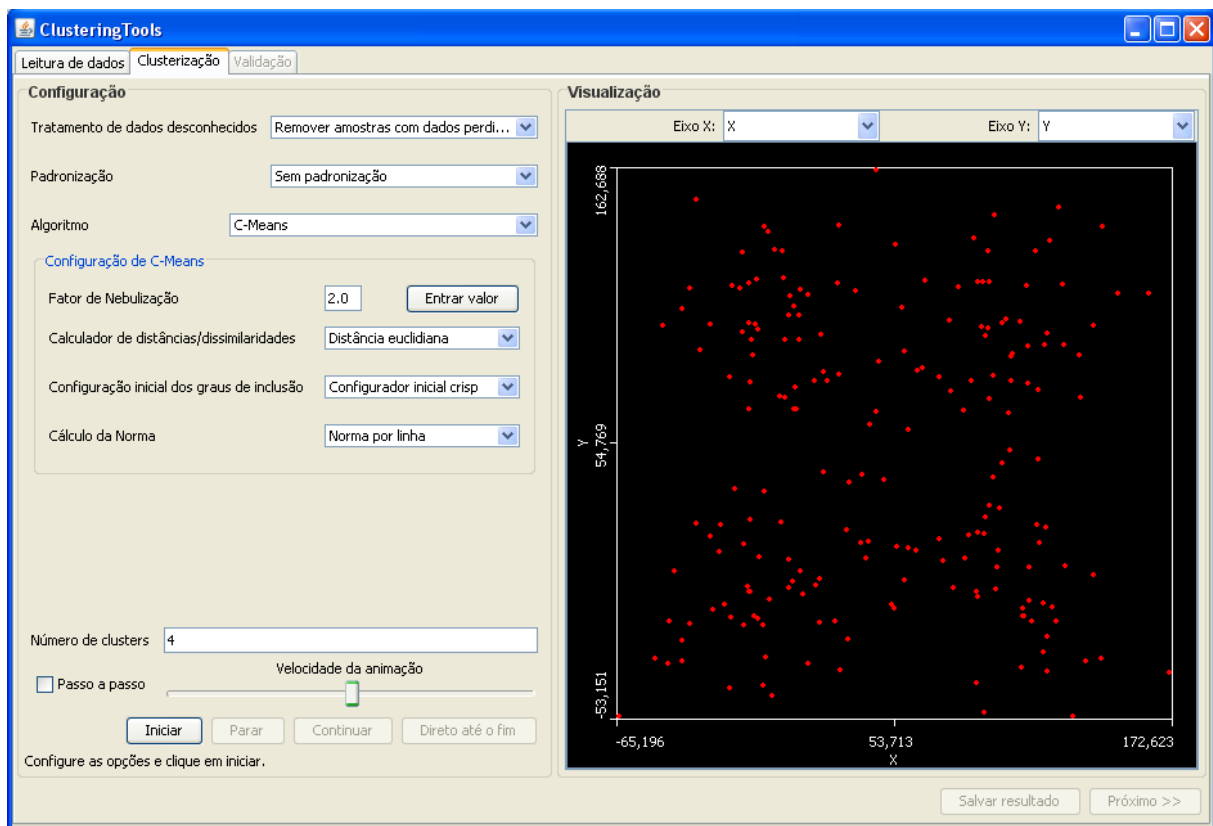


Figura 4.6: Painel de configurações do C-Means

- É necessário também escolher o algoritmo de clusterização. Os algoritmos podem ter propriedades que precisam ser configuradas. Portanto, ao escolher um dos algoritmos, um painel com suas propriedades é exibido para que sejam escolhidos os valores. Como já visto na seção ??, todas as propriedades que utilizam caixas de seleção na entrada podem ter painéis internos de configuração e, em geral, os algoritmos têm. Alguns painéis de configuração são exibidos nas figuras ?? e ??.
- Após a escolha do pré-tratamento (remoção de dados desconhecidos e padronização), é necessário escolher o número de clusters e a velocidade da exibição da clusterização. Pode-se executar o algoritmo passo a passo ou de forma contínua e a cada iteração a clusterização parcial é exibida no painel de visualização.
- Ao clicar em “Iniciar”, o algoritmo inicia e pode-se ver a clusterização acontecendo em duas coordenadas escolhidas pelo usuário.
- Ao término da execução, o programa exhibe o número de passos e pode-se prosseguir

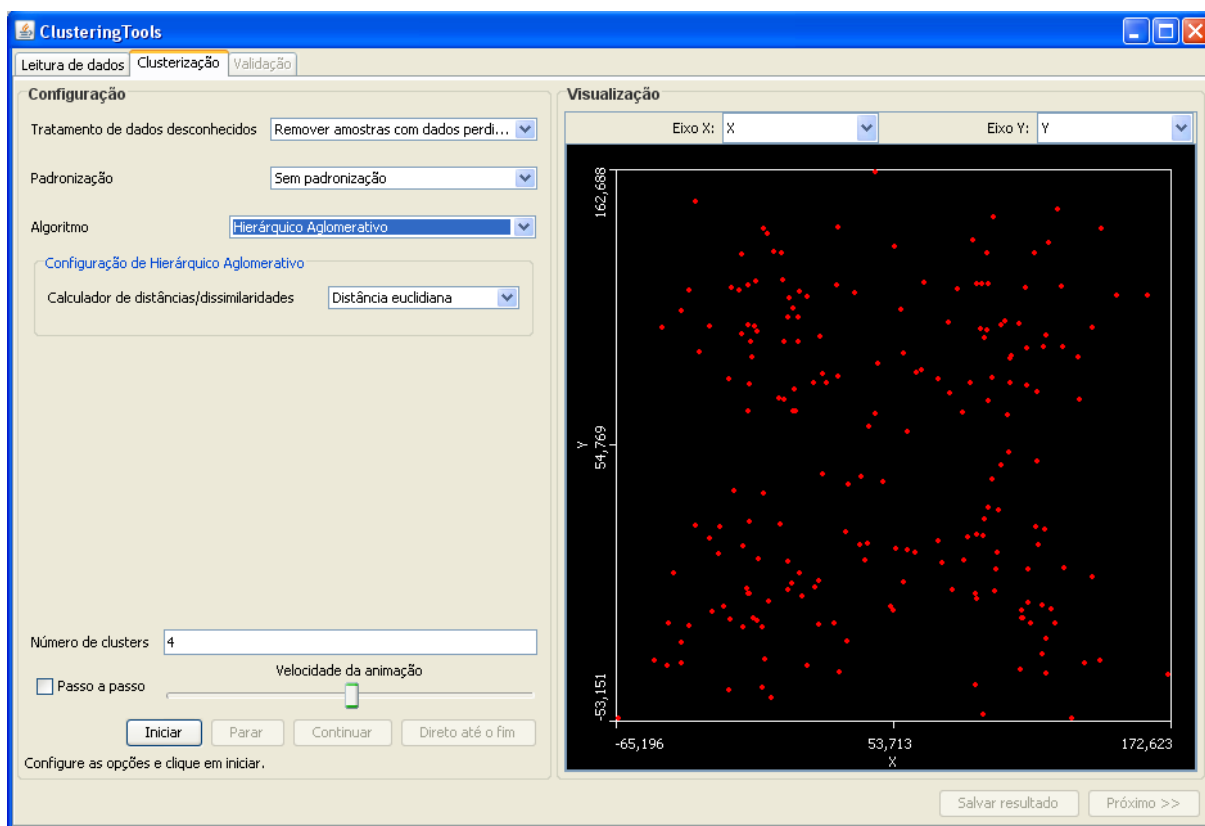


Figura 4.7: Painel de configurações do Algoritmo Hierárquico Aglomerativo

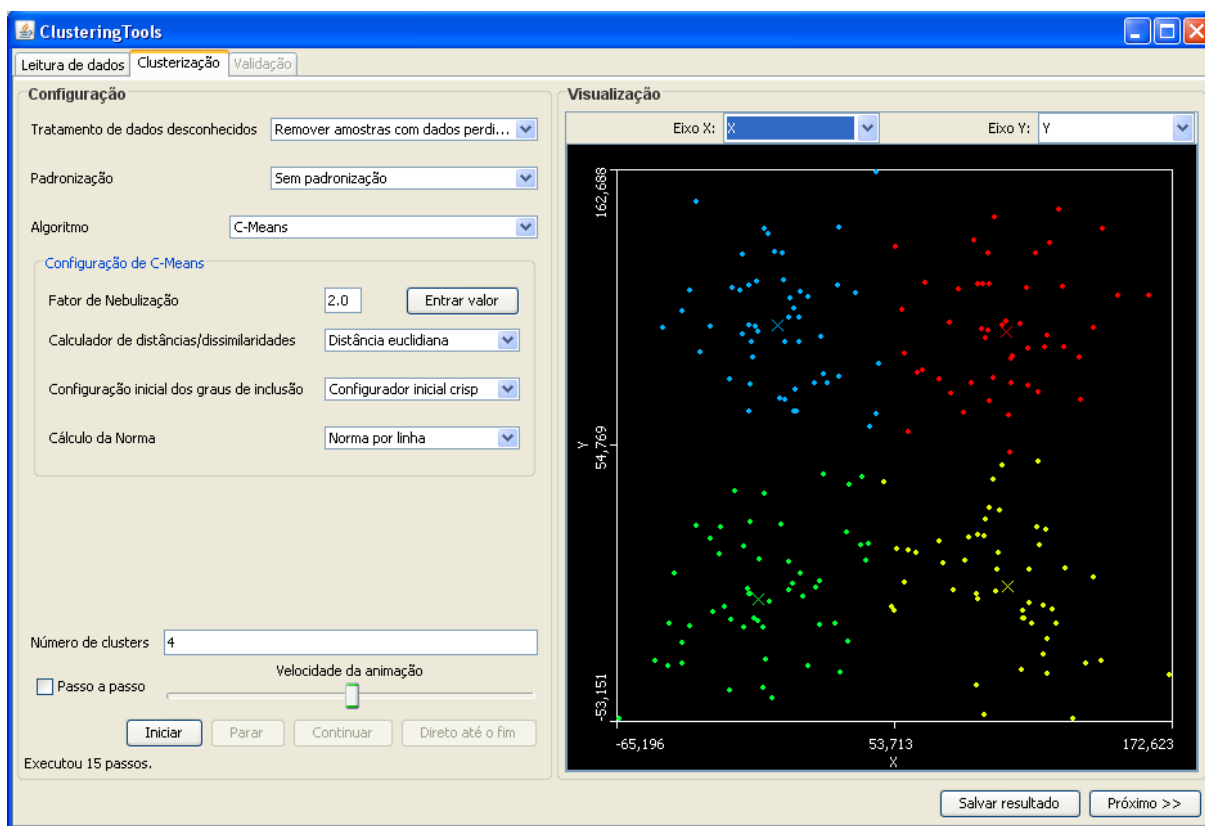


Figura 4.8: Fim do algoritmo de clusterização. Pode-se salvar ou passar à validação.

à validação clicando-se no botão “Próximo »”, como na figura ???. Pode-se também salvar o resultado da clusterização para validá-lo posteriormente, clicando-se no botão “Salvar”.

4.1.3 Validação

- Ao clicar em “Próximo »” na aba de clusterização, uma tabela é exibida com todos os algoritmos de validação presentes no arquivo de configuração do programa, juntamente com uma checkbox (para indicar se o algoritmo será utilizado ou não) e com o resultado da validação da última clusterização executada no programa.
- Ao selecionar-se um algoritmo de validação (selecionar a sua checkbox, como na figura ??), uma janela pop-up é exibida para a configuração do algoritmo, se o algoritmo possuir alguma propriedade a ser configurada.
- Após escolher e configurar todos os algoritmos a serem utilizados, deve-se clicar em

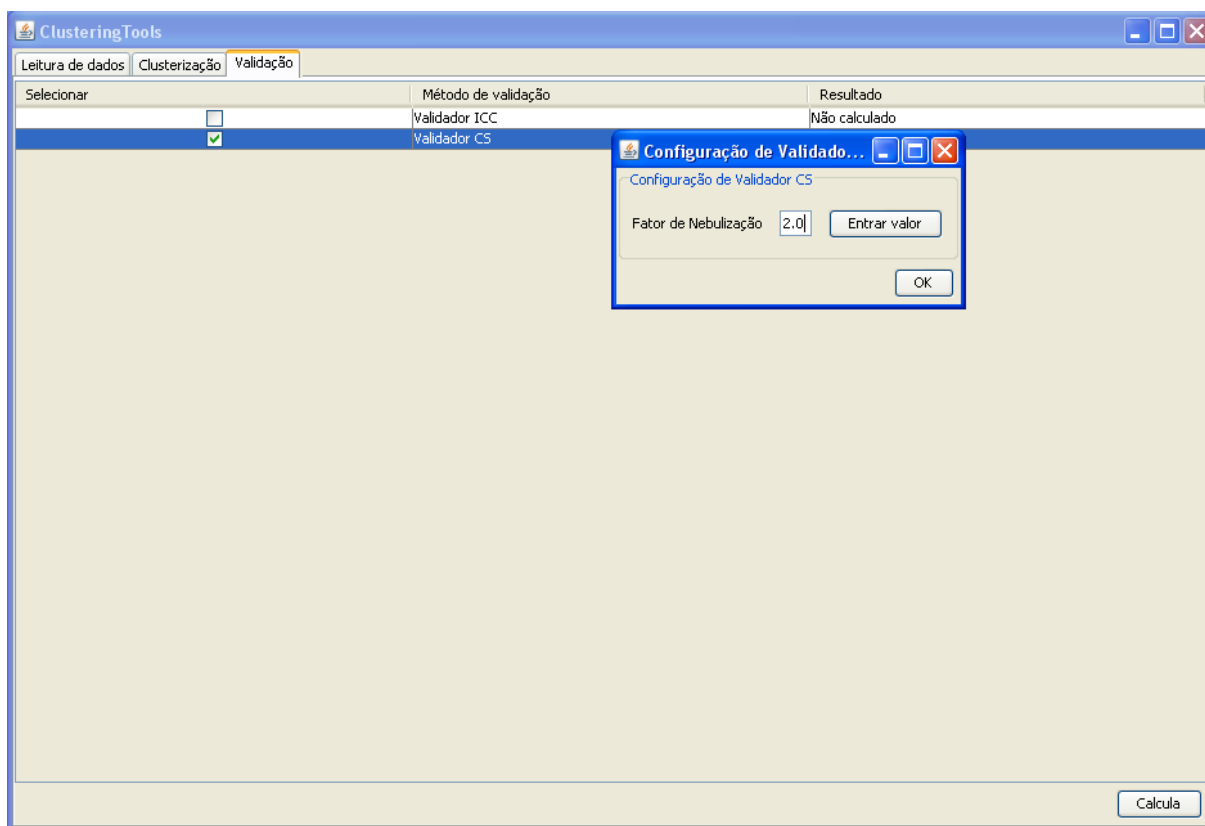


Figura 4.9: Selecionando e configurando um validador.

“Calcular” para que os resultados sejam exibidos.

Capítulo 5

Conclusões e trabalhos futuros

Nossa primeira dificuldade foi na modelagem da ferramenta. Devido aos variados tipos de métodos e dados existentes, a busca de um modelo flexível e funcional necessitou de muito tempo e esforço. Algumas opções feitas no tratamento dos dados não se mostraram adequadas a todos os algoritmos e tipos de dados e tivemos que mudar para um modelo que pode ser menos eficiente mas se adequa a todos os algoritmos implementados e é mais extensível. A extensibilidade também trouxe problemas na criação da interface gráfica, o que foi contornado utilizando mecanismos de reflexão da linguagem.

A ferramenta apresenta algumas vantagens em relação a outras existentes. Em primeiro lugar, a interface em português facilita um pouco seu uso em um primeiro contato. Além disso, o fato de ter sido feita pensando em um curso específico fez com que a ferramenta pudesse ser desenvolvida sem a necessidade de uma grande diversidade de opções, o que pode ser bom do ponto de vista da usabilidade. Usuários com pouco ou mesmo nenhum conhecimento sobre o assunto podem utilizar a ferramenta sem grandes problemas. O fato de ter sido desenvolvida usando a linguagem Java traz ainda a vantagem adicional da portabilidade, podendo ser executado em qualquer computador com uma máquina virtual Java.

Embora totalmente funcional, a ferramenta pode ser considerada como um passo inicial, uma vez que a variedade de algoritmos e tipos de dados possíveis em todas as etapas descritas é muito grande. Uma vez que a proposta da ferramenta é seu uso na geração de exemplos para o ensino, os alunos, após a utilização desta, podem expandir suas opções, de acordo com suas necessidades ou mesmo com a orientação do professor, dependendo do foco do curso.

Como visto, determinados métodos de clusterização diferem entre si somente pela

maneira como a distância é calculada. Apesar de sutil, esta diferença modifica completamente os resultados da clusterização e o estudo de novas maneiras de se avaliar a distância entre amostras se faz necessária. Com a ferramenta pronta, a implementação de novas alternativas se torna imediata, facilitando, inclusive, a avaliação dos resultados, através da interpretação gráfica e das métricas disponíveis.

Apesar das dificuldades encontradas, o estudo de assuntos não abordados na graduação, como lógica nebulosa e até mesmo os algoritmos de clusterização, nos fez conhecer novos assuntos da área de ciência da computação que servirão como mais uma alternativa na solução dos problemas que encontraremos no futuro. Esperamos ainda, que este trabalho seja o início de uma contribuição ainda maior para a área, não só facilitando o aprendizado dos conceitos abordados, mas também atraindo interesse para esta área do conhecimento.

Referências Bibliográficas

- [1] Annotations (Java Programming Language Guide) <http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html>
- [2] Bibliografia do curso de Lógica Nebulosa, Professor Adriano Cruz <http://equipe.nce.ufrj.br/adriano/fuzzy/bibliogr-ln.htm>
- [3] DEMASI, P., “Estratégias Adaptativas e Evolutivas em Tempo Real para Jogos Eletrônicos”. Dissertação de Mestrado, 2003.
- [4] HÖPPNER, F., KLAWONN, F., KRUSE, R. e RUNKLER, T., "Fuzzy cluster analysis: methods for classification, data analysis and image recognition". Wiley, 1999.
- [5] Jakarta Commons Math - Data Generation <http://jakarta.apache.org/commons/math/userguide/random.html>
- [6] JAMA - Java Matrix Package <http://math.nist.gov/javanumerics/jama/>
- [7] KAUFMAN, L. e ROUSSEEUW, P. J., "Finding groups in data: an introduction to cluster analysis". Wiley Series in probability and mathematical statistics, 1990.
- [8] Trail: The reflection API (The Java Tutorials) <http://java.sun.com/docs/books/tutorial/reflect/index.html>
- [9] Wikipedia, the free encyclopedia <http://www.wikipedia.org/>