Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

MAPAS AUTO-ORGANIZÁVEIS DE KOHONEN COM PONDERAÇÃO AUTOMÁTICA DAS VARIÁVEIS BASEADO NA DISTÂNCIA DE MANHATTAN

(TRABALHO DE GRADUAÇÃO)

Aluno: Filipe Martins de Melo
Orientador: Francisco de Assis Tenório de Carvalho
Recife, Dezembro de 2011.

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

MAPAS AUTO-ORGANIZÁVEIS DE KOHONEN COM PONDERAÇÃO AUTOMÁTICA DAS VARIÁVEIS BASEADO NA DISTÂNCIA DE MANHATTAN

(TRABALHO DE GRADUAÇÃO)

Trabalho apresentado na Universidade Federal de Pernambuco como projeto de conclusão do curso para obtenção do grau de Bacharel em Ciência da Computação

Aluno: Filipe Martins de Melo
Orientador: Francisco de Assis Tenório de Carvalho
Recife, Dezembro de 2011.

Resumo

Agrupamento de dados é uma técnica tem como objetivo agrupar um conjunto de itens em grupos de forma que os itens de um mesmo grupo apresentam maior similaridade e itens de grupos diferentes mostram uma maior dissimilaridade. Os Mapas Auto-organizáveis de Kohonen, chamados de SOM (*Self-Organizing Maps*), consistem em um método de rede neural que além de fornecer uma classificação não-supervisionada dos dados também produz uma representação visual dos mesmos. Esse trabalho tem como objetivo a proposta de um algoritmo de agrupamento de dados baseado no SOM, tendo como função de vizinhança a distância de *Manhattan*, por apresentar mais robustez e menor sensibilidade sobre dados com valores atípicos e também uma versão que usa um fator de relevância adaptativo para as variáveis.

Palavras-chave: Agrupamento de dados, Classificação não-supervisionada, Kohonen

Abstract

Clustering is a technique that groups a set of items into groups so that items of the same group are similar and items in different groups have a greater dissimilarity. The Self-Organizing Maps of Kohonen (SOM) consists of a neural network method that besides provides an unsupervised classification of the data also gives a visual representation of them. This work aims at proposing a data clustering algorithm based on SOM, with the neighborhood function the Manhattan distance, due to its lower sensitivity and robustness of data with outliers, and also a version that uses an adaptive factor of relevance for the variables.

Keywords: Clustering, Kohonen, Unsupervised classification

Agradecimentos

Nessa parte dessa monografia quero citar e agradecer as pessoas que sempre me ajudaram durante a minha graduação. Agradeço, então:

à minha mãe, Silvana Martins, que sempre me motivou, ajudou, aperreou muito, seria impossível sem todo e seu esforço e preocupação,

e aos meus primos e tios que estiveram sempre me apoiando e torcendo muito por mim no curso e nas maratonas,

ao meu orientador, Francisco Carvalho, que confiou e me apoiou nos projeto de iniciação científica e de trabalho de graduação,

ao pessoal da maratona de programação, o time e família UFPE, todos os sábados e feriados que passamos pelos laboratórios treinando , aprendi muito com todos vocês, em especial aos meus colegas dos times Carcará, Razão Cruzada e Challenge Accepted: Renan Pires, Laís Andrade, Pablo Carvalho, Luiz Afonso e Davi Duarte e aos grandes coaches Liliane Salgado e Pedro Bello. Mais uma vez a professora Liliane que tem se empenhado tanto com todos nós com muito carinho,

aos meus colegas, todas aquelas noites que passamos trabalhando nos projetos, cada um com sua habilidade para cada motivar e animar nos dias e noites de dureza que enfrentamos, em especial aos membros do Digitais Forever, com a regra dos 5%, que me parece funcionar muito bem.

Sumário

Índice de Figuras..... 7
Índice de Tabelas..... 8
1.Introdução..... 9
1.1.Objetivo..... 10
1.2.Estrutura do documento..... 10
2.Contextualização..... 11
2.1.Técnicas de agrupamento..... 11
2.2.Validação de Agrupamentos..... 11
Matriz de Confusão..... 12
Coeficiente de Rand ajustado..... 13
F-measure..... 13
2.3.Mapas Auto-organizáveis..... 14
2.4.Programação linear convexa..... 15
3.Algoritmo..... 16
Inicialização..... 16
Passo 1: representação..... 17
Passo 2: ponderação..... 18
Passo 3: partição..... 18
Critério de parada..... 18
4.Resultados experimentais..... 19
4.1.Iris Dataset..... 19
4.2.Libras Dataset..... 22
4.3.Wine Dataset..... 24
5.Conclusão e trabalhos futuros..... 25

Índice de Figuras

Figura 1.....9
Figura 2.....13
Figura 3.....15
Figura 4.....22
Figura 5.....23
Figura 6.....24
Figura 7.....31

Índice de Tabelas

Tabela 1..... 12
Tabela 2..... 19
Tabela 3..... 20
Tabela 4..... 20
Tabela 5..... 21
Tabela 6..... 21
Tabela 7..... 21
Tabela 8..... 21
Tabela 9..... 22
Tabela 10..... 22
Tabela 11..... 23
Tabela 12..... 24
Tabela 13..... 24

1. Introdução

Agrupamento de dados é uma das atividades muito comum na descoberta de conhecimento, bastante utilizada em várias aplicações e problemas com que apresentam a necessidade de uma identificação de distribuições de padrões e característica dos dados. O objetivo dessa técnica é classificar e dividir os dados em grupos segundo critérios de usabilidade, significado ou mesmo ambos, de forma que os indivíduos em cada um desses grupos são mais similares do que os indivíduos dos outros. [13]

As principais atividades realizadas nesse procedimento estão retratadas na Figura 1 mostrando cada uma de suas fases: (1) *Extração*, mostrado por Jain *et al* [2] ser uma fase importante para o desempenho do algoritmo e que se bem for bem executada também pode ocasionar a redução do esforço necessário da próxima fase; seguindo pela (2) *Algoritmo de Agrupamento*, que é um dos focos desse trabalho, consiste na definição da função de critério a ser utilizada , resolução de problemas de otimização produzidos por ela também, nesse ponto é importante falar que não existe um algoritmo universal para a resolução de todos os diferentes problemas desse campo, provado através do teorema da impossibilidade [9]; (3) *Validação*, dado o resultado da classificação proposta na fase anterior são calculados índices de validação para possível comparação deles, os índices que serão usados na parte de experimentos desse trabalho foram os coeficiente ajustado de *Rand* (CR) [10], a *F-measure* [4] e um fator de erro global segundo uma abordagem de escolha gulosa; (4) *Interpretação*, que é a fase final onde o usuário com os dados originais e a organização dos mesmos produzidos nesse processo, os especialistas da área podem interpretar e analisar, até mesmo fazer outros experimentos sobre essa partição de forma a extrair os resultados desejados. [16]

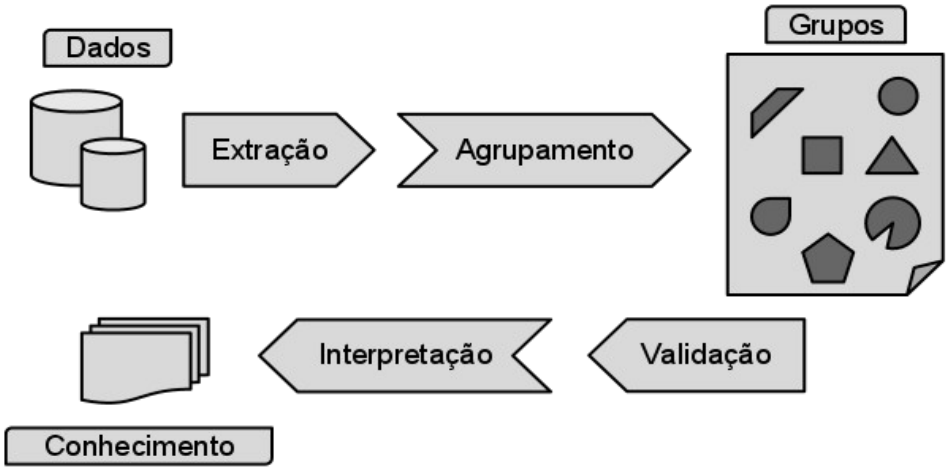


Figura 1. Sequência de atividades comuns realizadas na classificação de dados

Os Mapas Auto-organizáveis de Kononen(SOM) é mais um desses algoritmos disponíveis para produção de uma organização de dados que tem como uma de suas principais caraterísticas a produção uma representação visual dos dados ao mesmo tempo que faz a classificação. Variantes do SOM tem sido feito visando sua melhoria,

como proposto em Kangas *et al.* [8], onde ele apresenta duas abordagem baseadas no SOM, uma delas com ponderação dinâmica dos sinais de entrada em cada entrada de cada neurônio, que melhorou a ordenação quando muitos diferentes sinais de entrada são usados, o outro na definição de vizinhança no algoritmo de aprendizagem pela árvore de espalhamento mínima, que proporcionou uma aproximação melhor e mais rápida que outras.

1.1. Objetivo

Nesse contexto, objetivo desse trabalho é a proposta de um algoritmo de agrupamento de dados baseado no SOM, tendo como função de vizinhança a distância de *Manhattan*, também conhecida como a métrica do táxi ou distância L1, o intuito de utilizar ela é por ser uma função mais robusta do que a distância Euclidiana, distância L2, sobre dados que apresentam *outlier* [3] [17], esses valores atípicos favorecem uma produção de resultados mais instáveis para a distância L2 do que a L1. Uma versão que usa um fator de relevância adaptativo para as variáveis baseado no algoritmo principal também será um dos focos, visando melhoria nos resultados de agrupamentos. Para fins de avaliação de desempenho do método desenvolvido serão também implementandos os modelos que usam a distância Euclidiana dos modelos *BSOM*, *ABSOM* [12], os datasets que serão utilizados para comparação dos resultados encontram-se disponíveis na *Internet* através do *UCI Machine Learning Repository* [21].

1.2. Estrutura do documento

Esta monografia está organizada da seguinte forma. Esse capítulo começou com a introdução do assunto e do trabalho que foi feito, o próximo tem o objetivo de introduzir o contexto ao qual esse trabalho está relacionado e a descrição de algoritmos que foram usados no seu processo. O capítulo 3 mostra a descrição do algoritmo proposto, suas etapas com mais detalhes e como feito resolvido os problemas otimização nele. O capítulo seguinte apresenta os resultados obtidos através de simulações de teste que avaliaram o desempenho dos algoritmos implementados sobre os *datasets* que foram escolhidos. Por fim o capítulo 5 traz as conclusões e opiniões dos resultados obtidos e também alguns trabalhos que podem ser realizados posteriormente.

Ainda está presente no apêndice todo o código do algoritmo que será discutido nessa monografia para fins de análise e comparação para possíveis clarezas que sejam necessárias sobre os resultados obtidos, se bem como uma atualização do mesmo para trabalhos futuros.

2. Contextualização

Nós vivemos num mundo repleto de dados. Diariamente pessoas se encontram com uma grande quantidade de informação que é armazenada na forma de dados para posterior análise e gerenciamento. Um ponto vital no tratamento desses dados é ter a capacidade de fornecer uma classificação em grupos ou categorias de forma que se torna possível uma análise dos mesmos. Na verdade classificação de dados é uma atividade existente desde tempos mais primitivos, no qual desempenhou um papel importante e indispensável papel ao longo da história e do desenvolvimento da humanidade. Seja para aprender sobre um novo objeto ou entender um novo fenômeno, pessoas sempre tentam procurar uma característica que pode descrever cada um deles, e isso é feito através de comparação com outros objetos ou fenômenos existentes e conhecidos, baseados na semelhança ou diferença entre eles. [16]

Análise de agrupamentos é um campo da análise exploratória de dados cujo objetivo é organização não-supervisionada um conjunto de itens em grupos tais que itens em um mesmo grupo apresentam um alto grau de similaridade, enquanto itens pertencentes a diferentes grupos apresentam um alto grau de dissimilaridade. Sendo um problema muito comum em diversos contextos, pesquisas tais como análise de padrões, segmentação de imagens, aprendizagem de máquina, mineração de dados, classificação de padrões, recuperação de informação e também tem presença em disciplinas tais como biologia, psiquiatria, arqueologia, marketing, geografia. [1]

2.1. Técnicas de agrupamento

As principais técnicas de agrupamento de dados podem ser dividas em hierárquicas e particionais. Métodos hierárquicos fornecem uma sequência de partições aninhadas dos dados de entrada, em uma estrutura chamada de dendograma, podem ser aglomerativos, abordagem *bottom up* no qual cada item é o seu próprio grupo e gradativamente são feitas fusões entre grupos segundo um critério de vinculação até todos itens pertencerem a um mesmo grupo, ou divisivos, abordagem *top down* no qual os itens pertencem inicialmente ao mesmo grupo e em cada etapa ele é subdividido em menores. Enquanto os métodos particionais procuram obter uma única divisão dos dados em um número fixo de grupos geralmente através da otimização de uma função objetivo, critério de otimização do algoritmo no qual seu valor vai diminuindo até a convergência, são divididos em partições *hard* e *fuzzy*, tendo como principal diferença a restrição de pertinência de cada item, na partição hard cada item pertence a exatamente um único grupo e no *fuzzy* existe um determinado grau de associação parcial de um item com todos os grupos. [2]

2.2. Validação de Agrupamentos

Como critério de comparação entre as partições produzidas pela resposta dada por algoritmos de agrupamento de dados é usado alguns índices de validações, os que foram usados nesse trabalho foram o coeficiente ajustado de *Rand* (CR), a F-*measure*, e uma taxa de erro global baseada na matriz de confusão.

Matriz de Confusão

Trata-se de uma matriz específica que mostra a intersecção entre as classes priori dos dados pela das propostas, então de forma sumarizada o resultado pode ser analisado. Assumindo por exemplo uma amostra de 27 animais divididos em 3 classes: 8 gatos, 6 cachorros e 13 coelhos, o resultado pode ser visto na Tabela 1 abaixo. Os 8 gatos classificados como 5 gatos e 3 cachorros, os 5 cachorros em 2 gatos, 3 cachorros e 1 coelho e finalmente os 13 coelhos foram divididos em 2 cachorros e 11 coelhos.

Tabela 1. Exemplo de matriz de confusão

Grupo \ Classe	Gato	Cachorro	Coelho
Gato'	5	3	0
Cachorro'	2	3	1
Coelho'	0	2	11

Uma abordagem de cálculo de erro global usado nesse trabalho foi dada essa tabela com dimensões NxM, já que ela nem sempre é uma matriz quadrada, foi a conversão do mesmo para um problema muito conhecido na área de teoria dos grafos, chamado de fluxo máximo à custo mínimo, essa solução permitiu uma forma mais genérica de calcular um valor de erro desejado, como a garantia de que numa matriz quadrada cada classe à priori seja escolhida de uma classe proposta garantindo-se o acerto sendo o máximo possível.

O fluxo máximo a custo mínimo [20], é um problema relacionado na área de teoria dos grafos que consiste em encontrar o(s) menor(es) caminho(s) que envia(m) uma certa quantidade de fluxo através de uma rede de capacidades. Seja $G = (V, E)$ uma rede direcionada definida por um conjunto de V vértices, também chamandos de nós, e um conjunto E de arestas, ou arcos. Para cada aresta $(i, j) \in E$ nós temos também uma capacidade U_{ij} associada, que denota a maior quantidade que pode circular por ela e também temos um custo C_{ij} para cada unidade de fluxo que for utilizada. Cada um dos nós $i \in V$ está associado um b_i onde os que apresentam valores de $b_i > 0$ são chamandos de fornecedores e os com $b_i < 0$ de demandas. Um exemplo de um grafo com essas características pode ser visto na figura 2.

Possuindo a matriz de confusão já computada podemos então construir um grafo em que os nós fornecedores são os grupos propostos com valor de fluxo com valor um já que cada grupo vai está relacionado a uma única classe, os nós de demanda são as classes à priori dos dados eles tem fluxo igual a quantidade de grupos do algoritmo ou valor um também de forma a garantir que cada grupo proposto está sendo mapeado para uma única classe, os valores em cada posição da matriz de confusão é convertido para uma aresta em que o grupo i se liga a classe j com fluxo um e custo $-M[i][j]$, com valor negativo porque queremos erro mínimo, no caso do algoritmo custo mínimo e fluxo máximo por que cada grupo deve está associado a uma classe. O valor do fluxo depois de executado o algoritmo será igual ao número de grupos e o valor do custo corresponde ao valor do erro desejado com sinal trocado.

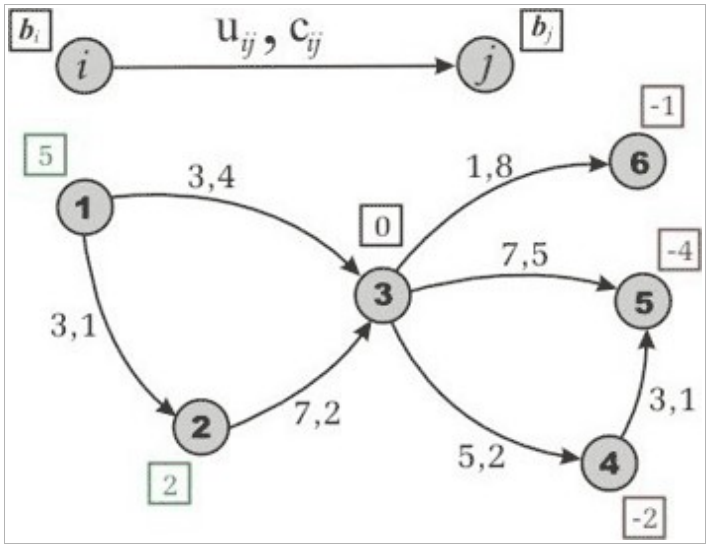


Figura 2. Um exemplo de grafo de redes como os nós fornecedores 1 e 2, os de demanda 6.5 e 4

Coeficiente de Rand ajustado

O coeficiente ajustado de *Rand* [10] mede o grau de similaridade entre uma partição à priori e a partição produzida por um algoritmo de classificação, seu valor indica também o nível de estabilidade da solução [14]. Seja $U = \{u_1, \ldots, u_i, \ldots, u_R\}$ e $V = \{v_1, \ldots, v_j, \ldots, v_C\}$ as partições do mesmo *dataset* tendo R classes e C grupos que queremos comparar. O *CR* é:

$$CR = \frac{a - \binom{n}{2}^{-1}[b \cdot c]}{\frac{1}{2}[b+c] - \binom{n}{2}^{-1}[b \cdot c]} \quad \text{onde} \quad a = \sum_{i=1}^{R}\sum_{j=1}^{C}\binom{n_{ij}}{2} \quad , \quad b = \sum_{i=1}^{R}\binom{n_{i.}}{2} \quad ,$$

$c = \sum_{j=1}^{C}\binom{n_{.j}}{2}$, temos o binômio $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ e n_{ij} representa o número de indivíduos que estão na classe u_i e no grupo v_j ; $n_{i.}$ indica o número de indivíduos na classe n_{ij} e $n_{.j}$ o número de indivíduos no grupo v_j ; n a quantidade total de indivíduos no *dataset*. Os valores do CR têm seus valores no intervalo [-1,1], na qual o valor 1 indica uma concordância perfeita entre as partições.

F-measure

A F-*measure* tradicional é uma medida que usa a combinação de dois outros termos, *precision* e *recall*, seu valor é a medida harmônica dessas duas [4], vamos definir seu valor entre a classe $u_i (i = 1, \ldots, R)$ e o grupo $v_j (j = 1, \ldots, C)$.

$$F-measure(u_i, v_j) = 2 \cdot \left(\frac{Precision(u_i, v_j) \cdot Recall(u_i, v_j)}{Precision(u_i, v_j) + Recall(u_i, v_j)}\right)$$

A *Precision* entre a classe $u_i (i = 1, \ldots, R)$ e o grupo $v_j (j = 1, \ldots, C)$ é

definida como a razão entre o número de indivíduos ques estão na classe u_i e no grupo v_j com o número de indivíduos no grupo v_j .

$$Precision(u_i, v_j) = \frac{n_{ij}}{n_{.j}} = \frac{n_{ij}}{\sum_{i=1}^{R} n_{ij}}$$

A *Recall* entre a classe $u_i(i = 1,\ldots,R)$ e o grupo $v_j(j = 1,\ldots,C)$ é definida como a razão entre o número de indivíduos ques estão na classe u_i e no grupo v_j com o número de indivíduos na classe u_i .

$$Recall(u_i, v_j) = \frac{n_{ij}}{n_{.j}} = \frac{n_{ij}}{\sum_{i=1}^{C} n_{ij}}$$

Já a F-*measure* entre a partição à priori $U = \{u_1, \ldots, u_i, \ldots, u_R\}$ e a partição *hard* $V = \{v_1, \ldots, v_j, \ldots, v_C\}$ produzida pelo algoritmo utilizado nesse trabalho é definida como:

$$F-measure(U, V) = (\frac{1}{n})\sum_{i=1}^{R} n_{i.} max_{1 \le j \le C} F-measure(u_i, v_j)$$

Os valores do índice da F-*measure* têm seus valores no intervalo [0,1], na qual o valor 1 indica uma concordância máxima entre as partições.

2.3. Mapas Auto-organizáveis

Os mapas de Kohonen, chamados SOM (Self-Organizing Maps) [18] são um método não-supervisionado com uma estratégia de aprendizagem competitiva com uma função de interação de vizinhança, sendo uma técnica de agrupamento de dados bastante utilizada que também fornece uma representação visual dos dados através do posicionamento espacial dos representantes desses grupos, esse mapa de visualização desvenda assim uma estrutura topológica escondida nos dados.

O algoritmo proposto por Kohonen [18] tem sido em diversos campos de aplicação. Seu uso mais típico está relacionado a visualização e interpretação de grande conjunto de dados com muitas dimensões [11]. O *Laboratory of Computer and Information Science (CIS)* [11] cita diversas dessas aplicações que tem sido usada: reconhecimento automático da fala, análise de voz clínica, monitoramento da condição de plantas e processos industriais, classificação de nuvens a partir de imagens de satélite, análise de sinais elétricos do cérebro, organização e recuperação de grandes coleções de documentos e análise e visualização de grandes conjuntos de dados.

O SOM tem uma estrutura de projeção dos dados, geralmente é utilizada uma de dimensão baixa, 1D, 2D ou 3D, que na verdade é um grafo não direcionado ,que chamamos de mapa, onde estão dispostos os neurônios que são os nós e suas relações de vizinhança representadas pelas arestas. De forma mais formal podemos dizer que o mapa é descrito por um grafo $map = (C, T)$, onde C é o conjunto de neurônios

interligados segundo a topologia em T , o fator de vizinhança entre dois desses neurônios $\partial(a,b)$ é regido pela menor distância entre eles nesse grafo. Cada neurônio é representado por um vetor p-dimensional (o protótipo) em que p é a mesma dimensão dos atributos de cada indivíduo contindo na entrada.

2.4. Programação linear convexa

Um dos algoritmos mais importantes desse trabalho foi o da resolução de um subproblema de minimização linear que surgiu devido ao uso da distância L1, diferentemente da L2 que tem uma solução bem conhecida usando derivada.

Dentre as propriedades de funções convexas [6] as que foram exploradas para a resolução do problema de minimização desse trabalho foi: dado que f e g duas funções convexas então tanto $m(x) = maximo\, f(x), g(x)$ e $g(x) = f(x) + g(x)$ serão funções convexas também.

O problema encontrado consiste em encontrar o valor de x que minimiza o somatório $\sum_{i=1}^{n} |a_i \cdot x + b_i|$ (1). As propriedades discutidas anteriormente permitiram uma computação eficiente desse problema, pois cada termo desse somatório $|a_i \cdot x + b_i|$ é equivalente a $maximo(a_i \cdot x + b_i, -a_i \cdot x - b_i)$, o valor de máximo entre duas funções do primeiro grau, uma reta que é uma função convexa, e somando todos eles temos novamente uma função convexa.

Outra característica importante que foi explorada é que funções convexas são unimodais, e a descrita da equação (1) apresenta o comportamento parecido com o da figura 3 produzida pelo *Wolfram Alpha* [22], seu valor vai diminuindo até um intervalo de valor mínimo e depois volta a crescer, no caso do exemplo esse intervalo de transição ocorreu no [0.5, 0.5].

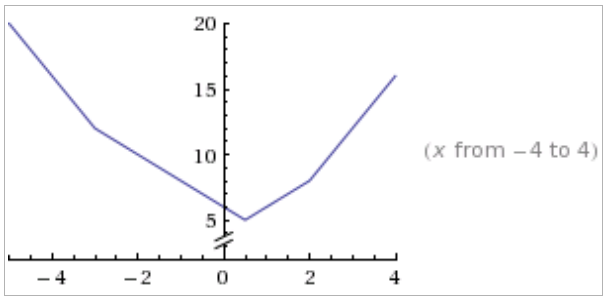


Figura 3. Exemplo do comportamento de uma função unimodal

O algoritmo eficiente que foi utilizado então para essa minimização foi a busca ternária [19], seu tempo de execução é logarítmico no tamanho do intervalo de busca, supondo que o cálculo da função de um ponto seja em tempo constante, no caso da equação (1) o tempo é linear, a complexidade de tempo final fica em O($n \cdot \log(L)$).

3. Algoritmo

Seja $E = \{x_1, ..., x_n\}$ o conjunto de indivíduos, onde cada um deles tem suas características representadas por um vetor de dimensão p para cada tuplas bidimensional, $x_i = \{(a_1 : b_1), ..., (a_p : b_p)\}\ (i = 1...n)$,a configuração do *grid* 2D $dx \times dy$ (temos assim c protótipos) os dados fornecidos na entrada. Em cada uma suas inicializações é executada N_{iter} iterações de onde o sistema parte de uma temperatura inicial de T_{min} , a temperatura mínima, até T_{max} , a temperatura máxima, esse valor representa o grau de influência de vizinhança entre os neurônios que vai dimuindo na função *kernel*, simbolizada por ∇ . O algoritmo também é acompanhado de sua função objetivo, também chamada de função de critério, as equações (2) e (3) são as do modelo sem e o com relevância respectivamente. O $f(x_i)$ em ambas equações corresponde ao grupo no qual o indivíduo x_i faz parte e T a temperatura atual de uma iteração, já $\partial_{lj}^{(t)}$ na equação (2) representa o fator adaptativo é fator de relevância do neurônio l em relação à variável j , ele apresenta como propriedade $\partial_{ij} > 0\ e\ \prod_{j=1}^{p} \partial_{ij} = 1$. O algoritmo tem por objetivo minimizar seu valor o máximo possível ao decorrer do seu processo.

$$J = \sum_{i=1}^{n} \sum_{l=1}^{c} \exp\left\{ \frac{-(\nabla((f(x_i)), l))^2}{2\mathrm{T}^2} \right\} \cdot \sum_{j=1}^{p} \left[|a_{ij} - \bar{\boldsymbol{a}}_{\boldsymbol{lj}}| + |b_{ij} - \bar{\boldsymbol{b}}_{\boldsymbol{lj}}| \right] \quad (2)$$

$$J = \sum_{i=1}^{n} \sum_{l=1}^{c} \exp\left\{ \frac{-(\nabla((f(x_i)), l))^2}{2\mathrm{T}^2} \right\} \cdot \sum_{j=1}^{p} \partial_{lj} \cdot \left[|a_{ij} - \bar{\boldsymbol{a}}_{\boldsymbol{lj}}| + |b_{ij} - \bar{\boldsymbol{b}}_{\boldsymbol{lj}}| \right] \quad (3)$$

Assim que a etapa de classificação termina segundo algum dos algoritmos que serão apresentados, é escolhida como solução a inicialização que apresentar melhor valor da função J , é impresso num arquivo de saída o agrupamento proposto com o protótipo de cada grupo e o indivíduo mais próximo a ele se existir, o *grid* com a quantidade de indivíduos em cada célula e também uma impressão individual de que grupo cada um deles foi atribuído se bem como os índices de validação do coeficiente de *Rand*, a F-*measure*, o valor do critério da função objetivo e a matriz de confusão para dados rotulados.

Inicialização

Fixe c ,o número de grupos.

Fixe ∇ , a função kernel do algoritmo.

Fixe N_{iter} , o número de iterações.

Fixe T_{min} e T_{max} , as temperaturas mínimas e máximas.

Atualize $T \leftarrow T_{max}$, a temperatura atual da iteração.

Atulize $t \leftarrow 0$, o contador de quantidade de iterações da inicialização

Atualize $\partial_l^{(0)} = (\partial_{l1}^{(0)},\ldots,\partial_{lp}^{(0)}) \leftarrow (1,..,1)\ (l=1,\ldots,c)$, o vetor de relevâncias.

Randomicamente selecione c protótipos distintos $g_l^{(0)} \in E(l=1,\ldots,c)$ e forme a primeira partição $P^{(0)} = (P_1^{(0)},\ldots,P_c^{(0)})$, onde cada indivíduo é atribuído ao grupo com protótipo com menor valor segundo a equação:

$$(f(x_i))^{(0)} = Mínimo(1 \le h \le c) \sum_{l=1}^{c} \exp\left\{-\frac{(\nabla(h,l))^2}{2\mathrm{T}^2} \cdot d_{\partial_l^{(0)}}(x_i, g_l^{(0)})\right\}$$

Passo 1: representação

Atualize $t \leftarrow t+1$.

Atualize $T = T_{max} \cdot (\frac{T_{min}}{T_{max}})^{\frac{t}{N_{iter}-1}}$.

Fixe $P^{(t-1)} = (P_1^{(t-1)},\ldots,P_c^{(t-1)})$ e o vetor de relevância $\partial_l^{(t-1)}$.

Compute os novos protótipos para essa nova iteração $g^{(t)} = (g_1^{(t)},\ldots,g_c^{(t)})$ onde $g_{l1j}^{(t)} = [\bar{\boldsymbol{a}}_{\boldsymbol{lj}}^{(t)}, \bar{\boldsymbol{b}}_{\boldsymbol{lj}}^{(t)}], (l = 1\ldots c)$, seus são atualizados de forma que:

$$\sum_{i=1}^{n} \exp\left\{\frac{-(\nabla((f(x_i)^{(t-1)}),l))^2}{2\mathrm{T}^2}\right\} \cdot |a_{ij} - \bar{\boldsymbol{a}}_{\boldsymbol{lj}}| \rightarrow Mínimo$$

$$\sum_{i=1}^{n} \exp\left\{\frac{-(\nabla((f(x_i)^{(t-1)}),l))^2}{2\mathrm{T}^2}\right\} \cdot |b_{ij} - \bar{\boldsymbol{b}}_{\boldsymbol{lj}}| \rightarrow Mínimo$$

fazendo manipulações matemáticas, eles são equivalentes a

$$\sum_{i=1}^{n} \exp\left|a_{ij} \cdot \left\{\frac{-(\nabla((f(x_i)^{(t-1)}),l))^2}{2\mathrm{T}^2}\right\} - \bar{\boldsymbol{a}}_{\boldsymbol{lj}} \cdot \left\{\frac{-(\nabla((f(x_i)^{(t-1)}),l))^2}{2\mathrm{T}^2}\right\}\right| \rightarrow Mínimo \quad (4)$$

$$\sum_{i=1}^{n} \exp\left|b_{ij} \cdot \left\{\frac{-(\nabla((f(x_i)^{(t-1)}),l))^2}{2\mathrm{T}^2}\right\} - \bar{\boldsymbol{b}}_{\boldsymbol{lj}} \cdot \left\{\frac{-(\nabla((f(x_i)^{(t-1)}),l))^2}{2\mathrm{T}^2}\right\}\right| \rightarrow Mínimo \quad (5)$$

esses somatórios ainda podem ser representados de forma mais genérica como

$$\sum_{i=1}^{n} |A_i + B_i \cdot x| = \sum_{i=1}^{n} maximum(A_i + B_i \cdot x, -A_i - B_i \cdot x) \quad (6)$$

onde A_i e B_i são valores constantes. Essa última representação permite a utilização de busca tenária [19], pois cada um dos termos desse somatório da equação (6) corresponde a um valor de máximo entre duas funções do primeiro grau, duas retas, que são funções convexas, como o máximo e a soma de funções convexas é também uma função convexa [15], as equações (4) e (5) são totalmente independentes do ponto de vista de minimização, sendo então apenas necessárias duas buscas ternárias para a atualização dos protótipos como desejado nessa fase.

Passo 2: ponderação

A partição $P^{(t-1)} = (P_1^{(t-1)}, \ldots, P_c^{(t-1)})$ anterior e o vetor de protótipos $g^{(t)} = (g_1^{(t)}, \ldots, g_c^{(t)})$ são fixados.

O novo fator de relevância das variáveis $\partial_l^{(t)} = (\partial_{l1}^{(t)}, \ldots, \partial_{lp}^{(t)})(l = 1, \ldots, c)$ são atualizados segundo a fórmula:

$$\partial_{lj}^{(t)} = \frac{\left\{ \prod_{h=1}^{p} \left(\sum_{i=1}^{n} \exp\left\{ \frac{-(\nabla((f(x_i)^{(t-1)}), l))^2}{2\mathrm{T}^2} \right\} \cdot \left[\left| a_{ih} - \overline{\boldsymbol{a}_{\boldsymbol{lh}}^{(t)}} \right| + \left| b_{ih} - \overline{\boldsymbol{b}_{\boldsymbol{lh}}^{(t)}} \right| \right] \right) \right\}^{\frac{1}{p}}}{\sum_{i=1}^{n} \exp\left\{ \frac{-(\nabla((f(x_i)^{(t-1)}), l))^2}{2\mathrm{T}^2} \right\} \cdot \left[\left| a_{ij} - \overline{\boldsymbol{a}_{\boldsymbol{lj}}^{(t)}} \right| + \left| b_{ij} - \overline{\boldsymbol{b}_{\boldsymbol{lj}}^{(t)}} \right| \right]}$$

Passo 3: partição

O vetor de protótipos $g^{(t)} = (g_1^{(t)}, \ldots, g_c^{(t)})$ e o vetor de relevância das variáveis $\partial_l^{(t)} = (\partial_{l1}^{(t)}, \ldots, \partial_{lp}^{(t)})(l = 1, \ldots, c)$ são fixados. Inicialmente a nova partição é uma réplica da anterior segundo a operação $P^{(t)} \leftarrow P^{(t-1)}$, que depois é atualizada segundo as seguintes operações:

for $i \leftarrow 1 \ldots n$ ***do***:

encontre o grupo $C_m^{(t)}$ *que o* x_i *pertence e*

encontre o grupo vencedor $C_r^{(t)}$ *tal que*

$$r = (f(x_i))^{(t)} = \mathit{Mínimo}(1 \le h \le c) \sum_{l=1}^{c} \exp\left\{ -\frac{(\nabla(h, l))^2}{2\mathrm{T}^2} \cdot d_{\partial_l^{(t)}}(x_i, g_l^{(t)}) \right\}$$

if $r \neq m$ ***then***:

$$C_r^{(t)} \leftarrow C_r^t \cup x_i$$

$$C_m^{(t)} \leftarrow C_m^{(t)} \setminus x_i$$

Os indivíduos são particionados para os grupos mais próximos como foi feito na etapa da inicialização só que dessa vez com os novos protótipos que foram encontrados na fase representação.

Critério de parada

Se $T = T_{min}, (t = N_{iter} - 1)$ então **PARE**; caso contrário atualize $t \leftarrow t + 1$ e volte ao passo 1.

4. Resultados experimentais

Para mostrar maior estabilidade da função de distância de *Manhattan* sobre a Euclidiana, experimentos foram realizados comparando o desempenho das duas técnicas sobre uma mesma configuração dos dados. Os *datasets* utilizados para nas simulações são dados reais que se encontram disponíveis na *Internet* através do *UCI Machine Learning Repository* [21]. Os critérios de comporação utilizados para fazer uma relação entre o desempenho dos métodos formam os índices de validação do erro global, o coefiente de *Rand* (CR) e a F-*measure*.

Um fator importante a ser levado nessas simulações são os parâmetros de entrada, eles causam grande influência nos resultados dos agrupamentos [7], os valores de T-mínimo e T-máximo foram escolhidos de forma a garantir a principal ideia de sua influência, onde inicialmente todos os indivíduos tenham uma grande influência entre si sem levar muito em consideração o fator de vizinhança, esse fator vai diminuindo com o decorrer do algoritmo até o final onde se torna muito relevante.

A implementação dos algoritmos foi feita em C++ [5], o código completo está disponível no apêndice, por razões de eficiência e resposta dos resultados e também pelo seu grande poder de expressão. Muitas configurações foram tentadas para escolha dos melhores agrupamentos que serão exibidos nesse capítulo para cada *dataset*.

4.1. Iris Dataset

Esse *dataset* consiste em três tipos da planta Iris: *Iris Setosa*, *Iris Versicolour* e *Iris Virginica*. Cada uma das três classes contém 50 instâncias. Cada um desses indivíduos está descrito segundo um valor numérico de quatro de suas características que são: (1) comprimento da sépala, (2) largura da sépala, (3) comprimento da pétala e (4) largura da sépala, todos em centímetros.

Tabela 2 indica os parâmetros utilizados na execução das simulações sobre o Iris, o número de inicializações indica quantas vezes o algoritmo foi executado, os demais são os parâmetros em sim do arquivo de configuração da simulação. De todas as inicializações é selecionada dentre elas a que produz o menor valor de critério dos algoritmos, assim como o agrupamento proposto, os índices de validação, e o fator de relevância das variáveis.

Tabela 2. Lista de parâmetros da execução do Iris

Parâmetro	Valor
Número de inicializações	100
Número de iterações	150
T-mínimo	0.190
T-máximo	100.000
Dimensões do grid	4x4

Tabela 3. Resultados com o Iris original

Função	Erro global	CR	F
L1	0.107	0.658	0.818
L2	0.120	0.646	0.819
L1 com fator	0.053	0.803	0.922
L2 com fator	0.033	0.639	0.766

Inicialmente foi feita a execução sobre os dados originais do Iris sem alteração, os resultados podem ser analisados na tabela 3, o desempenho foi bem próximo entre as abordagens da distância de *Manhattan* (L1), e a Euclidiana (L2), algumas tem o fator de relevância nas variáveis. A próxima parte dos testes foi fazer alterações sobre uma variável em uma quantidade fixa de indivíduos em cada classe, seu valor foi alterado por um fator multiplicativo, a que apresentou maior influência na proposta de agrupamento foi a primeira delas, o comprimento da sépala, os resultados na tabelas 4 mostram o comportamento dos índices de validação de acordo com a alteração dos indivíduos em cada classe, a tabela 5 mostra com apenas um deles. Com apenas um fator de 7 em 7 indivíduos dos 50 em cada classe como ou com o fator de 700 em apenas 1 o comportamento da distância Euclidiana não foi muito bom para produzir uma partição dos dados sobre o mapa, todos eles ficaram praticamente em um único grupo, isso justifica o erro global de 67%, já que cada classe apresenta-se como 33% dos dados.

Tabela 4. Iris modificado na variável 1 com fator multiplicativo 7

Quantidade de indivíduos alterados por classe	Função	Erro global	CR	F
1	Manhattan	0.027	0.643	0.750
	Euclidiana	0.120	0.704	0.880
2	Manhattan	0.033	0.631	0.737
	Euclidiana	0.347	0.509	0.750
3	Manhattan	0.067	0.227	0.410
	Euclidiana	0.353	0.493	0.744
5	Manhattan	0.080	0.264	0.456
	Euclidiana	0.367	0.461	0.729
7	Manhattan	0.033	0.458	0.652
	Euclidiana	0.667	-0.004	0.480

Tabela 5. Iris modificado na variável 1 com fator multiplicativo 700

Quantidade de indivíduos alterados por classe	Função	Erro global	CR	F
1	Manhattan	0.033	0.635	0.750
	Euclidiana	0.667	-0.001	0.497

Mais detalhes do resultado da classificação da tabela 3, com a alteração de 7 indivíduos podem ser melhor observados nas matrizes de confusão e nos mapas dessa simulação que estão nas tabelas 6, 7, 8 e 9.

Tabela 6: Matriz de confusão L1 sem fator no Iris modificado em 7 indivíduos

Grupo \ Classe	1	2	3
1	0	25	0
2	0	4	15
3	0	0	17
4	0	0	17
13	32	0	0
14	18	0	0
15	0	4	0
16	0	17	1

Tabela 7. Mapa da L1 sem fator no Iris modificado em 7 indivíduos

25	19	17	17
0	0	0	0
0	0	0	0
32	18	4	18

Tabela 8. Matriz de confusão L2 sem fator no Iris modificado em 7 indivíduos

Grupo \ Classe	1	2	3
1	43	43	43
16	7	7	7

Tabela 9. Mapa da L2 sem fator no Iris modificado em 7 indivíduos

129	0	0	0
0	0	0	0
0	0	0	0
0	0	0	21

4.2. Libras Dataset

Esse *dataset* consiste em de 15 classes que são os movimentos mais comuns em LIBRAS (Língua Brasileira de Sinais), figura 4, cada classe contém 24 instâncias. A descrição de cada um desses objetos foi obtida pela conversão de filmagens desses movimentos feitos por 4 pessoas diferentes durante 2 sessões para uma representação de bidimensional de 45 pontos através das coordenadas (x_i, y_i) deles ordenados de acordo com o desenho do caminho do movimento extraído do *frame* com seu valores normalizados no espaço unitário.

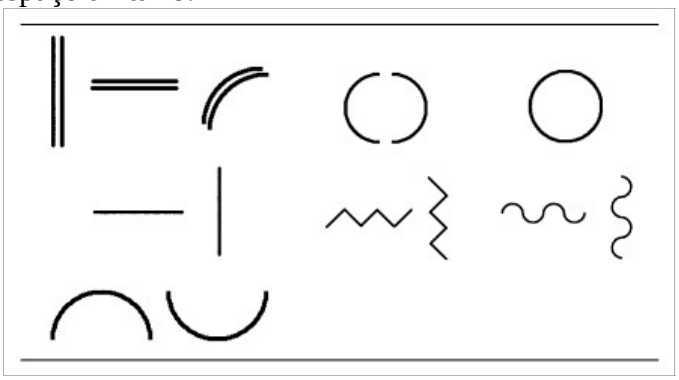


Figura 4. Movimentos mais comuns do LIBRAS

Tabela 10 indica os parâmetros utilizados na execução das simulações com o LIBRAS, da mesma forma que no Iris é selecionada dentre todas as inicializações aquela que produz o menor valor de critério dos algoritmos e as outras informações que são relevantes para a sua análise

Tabela 10. Lista de parâmetros da execução do LIBRAS

Parâmetro	Valor
Número de inicializações	10
Número de iterações	150
T-mínimo	0.200
T-máximo	10.000
Dimensões do grid	10x10

Tabela 11. Resultados do LIBRAS

Função	Erro global	CR	F
L1	0.181	0.217	0.405
L2	0.814	0.077	0.243
L1 com fator	0.319	0.199	0.413
L2 com fator	0.781	0.120	0.270

Os resultado da simulação encontram-se na tabela 11, a taxa de erro da distância L2 foi quase 5 vezes maior que o da L1. Uma visualização do mapa do L1 sem fator de relevância foi montada para mostrar a relação de aproximação entre os movimentos dentre desse *grid* exibido figura 5, alguns dos grupos isolados podem ser vistos a seguir na figura 6.

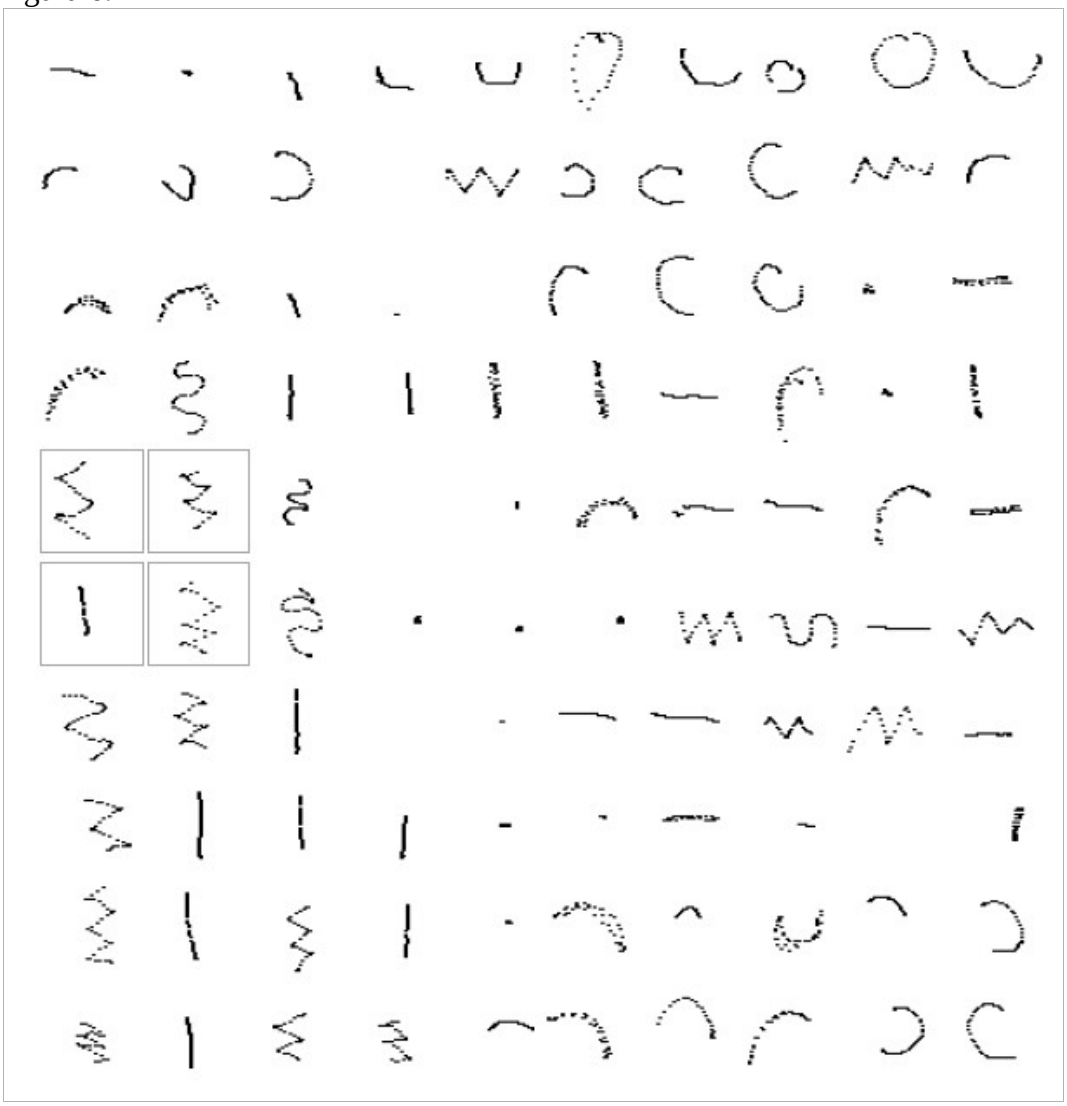


Figura 5. Mapa de visualização do LIBRAS com o L1 sem fator de relevância

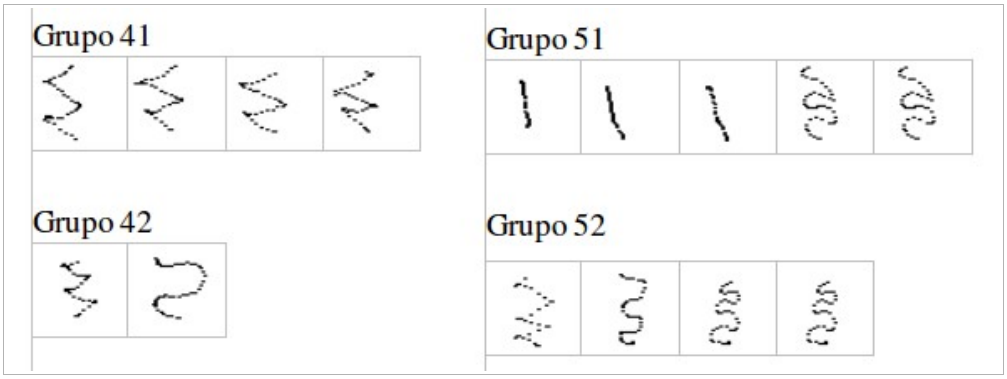


Figura 6. Grupos 41,42,51 e 52 com seus indivíduos

Nessas imagens podemos notar proximidade entre os indivíduos em cada grupo como também em grupos vizinhos e maior diferença entre eles de acordo com a distância que é um dos resultados expressos pelo SOM, não é perfeito pelos valores dos índices e visualmente pelo grupo 42, mas é bom o suficiente para explorar proximidade entre movimentos do LIBRAS por exemplo.

4.3. Wine Dataset

Esse *dataset* consiste em três classificações de vinho particionados em 59, 71 e 48 itens. Cada um desses indivíduos está descrito segundo um valor numérico de treze de seus componentes químicos: álcool, ácido málico, magnésio, etc. Não é um *dataset* muito desafiador, mas mostra que o comportamento da distância Euclidiana pode ser superior à de *Manhattan* como é mostrado nos resultados abaixo na tabela 13.Tabela 1, apresentado a taxa de erro quase nula para a versão L2 com ponderação. As configurações utilizadas nessa simulação estão apresentadas abaixo na tabela 12.

Tabela 12. Lista de parâmetros da execução do wine

Parâmetro	Valor
Número de inicializações	200
Número de iterações	150
T-mínimo	0.200
T-máximo	10.000
Dimensões do grid	5x5

Tabela 13. Resultados com o wine

Função	Erro global	CR	F
L1	0.230	0.083	0.309
L2	0.241	0.075	0.282
L1 com fator	0.034	0.328	0.587
L2 com fator	0.006	0.215	0.457

5. Conclusão e trabalhos futuros

O objetivo desse trabalho foi a proposta de uma algoritmo baseado no SOM utilizando a função de *Manhattan* em vez da Euclidiana por apresentar mais estabilidade e menor sensibilidade sobre dados com *outliers*.

Tomando base dos resultados obtidos através de bases de dados conhecidas e por outras simulações podemos ver um comportamento mais robusto da distância de *Manhatan* sobre dados com valores modificados. O fator de relevância nem sempre ajudou uma melhoria no resultado do agrupamento além de apresentar certa instabilidade em alguns dados, deve ser analisado bem o seu uso em alguma aplicação.

Como trabalhos futuros uma abordagem que possivelmente pode melhorar alguns agrupamentos seria o desenvolvimento de um modelo mantendo a distância L1 só que com uma abordagem de agrupamento *fuzzy*, bem como estender ela para uma versão com ponderação adaptativa.

Referências

[1] A. Jain, M. Murty, P. Flynn. Data clustering: A review, ACM Comput. Surv., vol. 31, no. 3, pp. 264–323, 1999.

[2] A. Jain, R. Dubes. Algorithms for Clustering Data, Englewood Cliffs, NJ: Prentice-Hall, 1988.

[3] André Luis Santiago Maia, Francisco de A. T. de Carvalho: Fitting a Least Absolute Deviation Regression Model on Interval-Valued Data. SBIA 2008:207-216

[4] C. J. van Rijsbergen. Information Retrieval, London: Butterworths,1979.

[5] C++. http://en.wikipedia.org/wiki/C++, acessado em Novembro de 2011.

[6] Convex Function. http://en.wikipedia.org/wiki/Convex_function, acessado em Novembro de 2011.

[7] F. Badran, M. Yacoub and S. Thiria, Self-organizing maps and unsupervised classification in Neural Networks: methodology and applications, G. Dreyfus, Ed. Berlin, Germany: Springer-Verlag, pp.379-442, 2005.

[8] Jari Kangas, Teuvo Kohonen, Jorma Laaksonen, Variants of self-organizing maps, in IEEE Transactions on Neural Networks, vol. 1,no. 1, 1990.

[[9] Jon M. Kleinberg: An Impossibility Theorem for Clustering. NIPS 2002:446-453

[10] L. Hubert, P. Arabie. Comparing partitions, in Journal of Classification, vol. 2, pp. 193218, 1985.

[11] Laboratory of Computer and Information Science, Adaptative Informatics Research Centre. http://www.cis.hut.fi/research/som-research/, acessado em Novembro de 2011.

[12] Luciano D. S. Pacifico, Francisco de A. T. de Carvalho: A Batch Self-Organizing Maps Algorithm Based on Adaptive Distances

[13] Maria Halkidi, Yannis Batistakis, Michalis Vazirgiannis: On Clustering Validation Techniques. J. Intell. Inf. Syst. (JIIS) 17(2-3):107-145 (2001)

[14] Phipps Arabie,Lawrence J. Hubert,Geert de Soete: Clustering and classification, World Scientific pp 369-370, 1996.

[15] Qifa Ke, Takeo Kanade: Robust L1 Norm Factorization in the Presence of Outliers and Missing Data by Alternative Convex Programming. In Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, 2005.

[16] Rui Xu, Donald C. Wunsch II. Survey of clustering algorithms. IEEE Transactions on Neural Networks 16(3): 645-678 (2005).

[17] Sebahaltin Bektas, Yasemin Sisman: The comparison of L1 and L2-norm minimization methods, International Journal of the Physical Sciences vol. 5 (11) pp. 1721-1727, 2010

[18] T. Kohonen, Self-Organisation Maps. Berlin: Springer, 1995

[19] Ternary Search. http://en.wikipedia.org/wiki/Ternary_search, acessado em Novembro de 2011.

[20] TopCoder Algorithm Tutorials Minimum Cost Flow. http://community.topcoder.com/tcmodule=Static&d1=tutorials&d2=minimumCostFlow1, http://community.topcoder.com/tcmodule=Static&d1=tutorials&d2=minimumCostFlow2 e http://community.topcoder.com/tcmodule=Static&d1=tutorials&d2=minimumCostFlow3, acessado em Novembro de 2011

[21] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets.html, acessado em Novembro de 2011.

[22] Wolfram Alpha. http://www.wolframalpha.com/, acessado em Novembro de 2011.

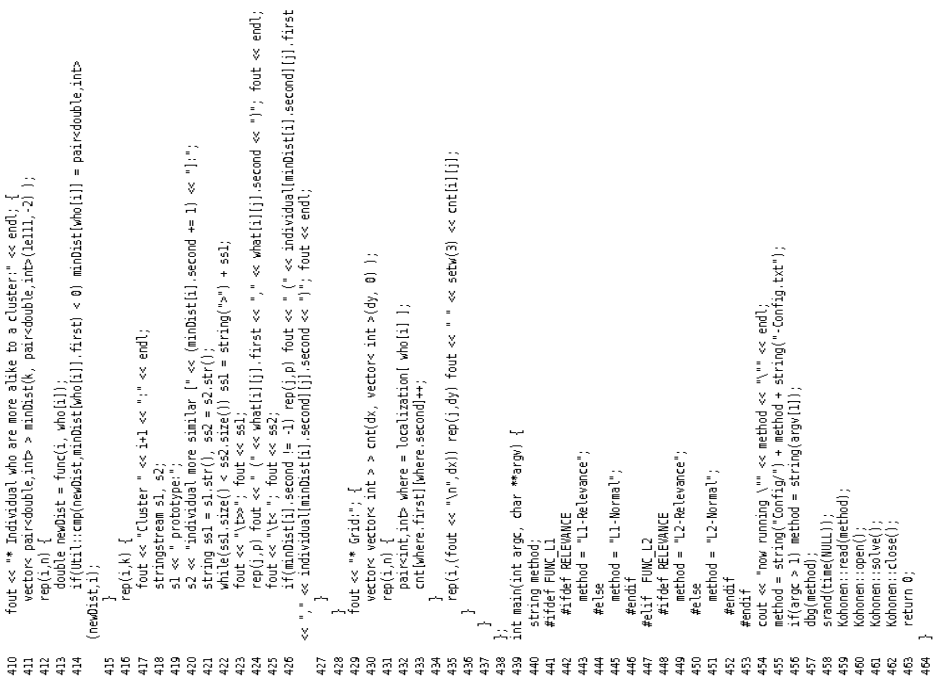
Apêndice A. Source code e Makefile

Todo o código está apresentado abaixo(464 linhas), ou no google code na página http://som-algorithm-tg.googlecode.com/svn/trunk/src/Tg.cpp, sua implementação foi feita em C++ e compilado através do g++ versão 4.4.3 numa máquina Linux Ubuntu 4.4.3-4ubuntu5.

```
#include <bits/stdc++.h>
using namespace std;
template<class T> void pv(T a, T b) { for (T i = a; i != b; ++i) cout << *i << " "; cout << endl; }
#define bug(msg) cerr << "bug report: " << msg << ", in line(" << __LINE__ << ")" << endl, exit(1)
#define rep(x,n) for(int x = 0; x < (n); x++)
#define dbg(x) cout << #x " = " << x << ", in line(" << __LINE__ << ")" << endl
#define _ << " " <<
namespace Util {
  int cmp(double x, double y = 0, double eps = 1e-7) {
    if(fabs(x-y) <= eps) return 0; else return x < y ? -1 : +1;
  }
};
namespace MinCostMaxFlow {
  vector< int > from, to, w, cap, ant;
  vector< int > adj;
  int n, m;
  void add(int x, int y, int z, int c) {
    #define pb push_back
    while(x+1 > n) n++, adj.pb(-1); while(y+1 > n) n++, adj.pb(-1);
    from.pb(x), to.pb(y), w.pb(z), cap.pb(c), ant.pb(adj[x]), adj[x] = m++;
    swap(x,y), z = -z, c = 0;
    from.pb(x), to.pb(y), w.pb(z), cap.pb(c), ant.pb(adj[x]), adj[x] = m++;
    #undef pb
  }
  const int inf = 0x3f3f3f3f;
  vector< int > dist, pot, pai;
  set< pair<int,int> > heap;
  void update(int no, int ndist, int p) {
    if(ndist >= dist[no]) return;
    if(dist[no] < inf) heap.erase(pair<int,int>(dist[no],no));
    dist[no] = ndist, pai[no] = p;
    heap.insert(pair<int,int>(dist[no],no));
  }
  pair<int,int> top() {
    pair<int,int> ret = *heap.begin();
    heap.erase(heap.begin());
    return ret;
  }
  int djikstra(int source, int sink) {
    heap.clear(), dist = vector< int >(n,inf), pai = vector< int >(n,-1);
    update(source,0,-1);
    while(heap.size()) {
      pair<int,int> p = top();
      if(p.second == sink) return 1;
      for(int i = adj[p.second]; i >= 0; i = ant[i]) if(cap[i]) {
        update(to[i], p.first + w[i] + pot[p.second] - pot[to[i]], i);
      }
    }
    return 0;
  }
  pair<int,int> mcmf(int source, int sink) {
    pair<int,int> p(0,0); // cost,flow
    pot = vector< int >(n,inf); pot[source] = 0;
    for(int k = 0; k < n; k++) for(int i = 0; i < n; i++) for(int j = adj[i]; j >= 0; j = ant[j]) if
  (cap[j]) {
      pot[to[j]] = min(pot[to[j]], pot[i] + w[j]);
    };
    while(djikstra(source,sink)) {
      int cost = 0, flow = inf;
      for(int x = sink; x != source; x = from[pai[x]]) if(cap[pai[x]] < flow) flow = cap[pai[x]];
      for(int x = sink; x != source; x = from[pai[x]]) cap[pai[x]^1] += flow, cap[pai[x]] -= flow,
  cost += w[pai[x]]*flow;
      for(int x = 0; x < n; x++) pot[x] += dist[x]; p.first += cost, p.second += flow;
    }
    return p;
  }
};
void init() {
  n = m = 0;
  adj.clear(), from.clear(), to.clear(), w.clear(), cap.clear(), ant.clear();
}
};
namespace Validation {
  double globalError(int n, vector< vector<int> >& tab) {
    int N = tab.size() - 1, M = tab[0].size() - 1, source = 0, sink = N + M + 1;
    MinCostMaxFlow::init();
    rep(i,N) rep(j,M) MinCostMaxFlow::add(i + 1, j + N + 1, -tab[i][j], 1);
    rep(i,N) MinCostMaxFlow::add(source, i + 1, 0, 1);
    rep(i,M) MinCostMaxFlow::add(i + N + 1, sink, 0, N);
    pair<int, int> ret = MinCostMaxFlow::mcmf(source, sink);
    assert(ret.second == N);
    return 1 + double(ret.first) / n;
  }
  double adjustedRandIndex(int n, vector< vector<int> >& tab) {
    #define comb(x) ((x)*((x)-1)/2)
    double term[4] = {0}, temp[2] = {0};
    double pot = 1.0 / comb(n);
    int N = tab.size() - 1, M = tab[0].size() - 1;
    rep(i,N) rep(j,M) term[1] += comb(tab[i][j]);
    rep(i,N) temp[0] += comb(tab[i][M]);
    rep(i,M) temp[1] += comb(tab[N][i]);
    term[2] = pot * (temp[0] * temp[1]);
    term[3] = 0.5 * (temp[0] + temp[1]);
    #undef comb
    return (term[1]-term[2])/(term[3]-term[2]);
  }
  double fMeasure(int n, vector< vector<int> >& tab) {
    double F = 0;
    int N = tab.size() - 1, M = tab[0].size() - 1;
    rep(j,M) {
      double vmax = 0;
      rep(i,N) if(tab[i][j] != 0) {
        double rappel = double(tab[i][j]) / tab[N][j];
        double precision = double(tab[i][j]) / tab[i][M];
        vmax = max(vmax, 2 * rappel * precision / (rappel + precision));
      }
      F += vmax * tab[N][j];
    }
    return F/n;
  }
};
namespace Kohonen {
  struct Item {
    vector< pair<double,double> > v; // #p
    pair<double,double>& operator[](int n) {
      return v[n];
    }
    void add(string s) {
      double x, y; assert(sscanf(s.c_str(),"%lf:%lf",&x,&y) == 2);
      v.push_back( pair<double,double>(x,y) );
    }
  };
  // var
  vector< Item > individual; // #n
  vector< Item > prototype;  // #k
  vector< vector< double > > lambda; // #k * #p
  int n, k, p;
  int niter, ninit;
  int dx, dy;
  int classVariable;
  double tmin, tmax;
  vector< int > who, from;
  vector< pair<int,int> > localization;
  vector< vector< double > > neighborhood;
  vector< vector< int > > table;
  double opt;
  int which;
  vector< int > res;
  vector< vector< double > > relevance;
  vector< Item > what;
  ofstream out;
```

```
  string input, output;
  // read the data
  int read(string file) {
    ifstream config(file.c_str());
    map<string,string> mapa;
    for(string label, val; config >> label >> val;) mapa[label] = val;
    config.close();
    assert(sscanf(mapa["(Number_of_initializations)"].c_str(),"%d",&ninit) == 1);
    assert(sscanf(mapa["(Class_variable)"].c_str(),"%d",&classVariable) == 1); classVariable--;
    assert(sscanf(mapa["(Number_of_individuals)"].c_str(),"%d",&n) == 1);
    assert(sscanf(mapa["(Number_of_iterations)"].c_str(),"%d",&niter) == 1);
    if(sscanf(mapa["(Tmin)"].c_str(),"%lf",&tmin) != 1 || sscanf(mapa["(Tmax)"].c_str(),"%lf",&tmax) != 1) tmin = tmax = 0.0;
    assert(sscanf(mapa["(Grid_dimension)"].c_str(),"%dx%d",&dx,&dy) == 2); k = dx*dy;
    input = mapa["(Input_file)"];
    output = mapa["(Output_file)"];
    ifstream data(input.c_str());
    string line;
    do {
      if (!getline(data, line)) bug("Invalid file!");
    } while (line.find("RECTANGLE_MATRIX = (") == string::npos);
    individual.clear();
    from = vector< int >(n, -1);
    rep(i,n) {
      line.clear();
      char c = '?';
      int ct = 0;
      while(!ct) {
        if(!(data >> c)) bug("Invalid file!");
        if(c == '(') ct++;
      }
      while(ct) {
        if(!(data >> c)) bug("Invalid file!");
        if (c == '(') ct++; else if (c == ')') ct--;
        if (ct) line.push_back(c);
      }
      line.push_back(',');
      vector<string> var;
      string piece;
      rep(j,line.size()) if(line[j] != ' ') {
        if (line[j] == '(') ct++; else if (line[j] == ')') ct--;
        if(!ct && line[j] == ',') {
          var.push_back(piece), piece.clear();
        } else piece.push_back(line[j]);
      }
      Item item;
      rep(j,var.size()) if(j != classVariable) item.add(var[j]); else assert(sscanf(var[j].c_str(),"%d",&from[i]) == 1);
      individual.push_back(item);
    }
    data.close();
    p = individual[0].v.size();
    return 1;
  }
  void open() {
    out.open(output.c_str());
  }
  void close() {
    out.close();
  }
  // neighborhood function between local 'x' and local 'y' in a given temperature
  double kernel(int x, int y, double temperature) {
    return exp(-neighborhood[x][y] / temperature / temperature / 2.0);
  }
  // dissimilarity between individual 'x' and prototype 'y'
  double func(int x, int y) {
    double dist = 0;
    rep(j,p) {
      #ifdef FUNC_L1
        dist += lambda[y][j] * (fabs(individual[x][j].first - prototype[y][j].first) + fabs(individual[x][j].second - prototype[y][j].second));
      #elif  FUNC_L2
        dist += lambda[y][j] * (pow(fabs(individual[x][j].first - prototype[y][j].first),2.0) + pow(fabs(individual[x][j].second - prototype[y][j].second),2.0));
      #endif
    }
    return dist;
  }
  // the criterion function
  double energy(double temperature) {
    double criterion = 0;
    rep(i,n) rep(j,k) {
      criterion += kernel(who[i],j,temperature) * func(i,j);
    }
    return criterion;
  }
  // initial phase
  void initialization() {
    lambda = vector< vector< double > >(k, vector< double >(p,1.0));
    prototype = individual;
    random_shuffle(prototype.begin(),prototype.end());
    prototype.resize(k);
    localization.clear();
    rep(i,dx) rep(j,dy) {
      localization.push_back( pair<int,int>(i,j) );
    }
    neighborhood.clear();
    rep(i,k) {
      neighborhood.push_back( vector<double>() );
      rep(j,k) {
        neighborhood[i].push_back(pow(localization[i].first - localization[j].first, 2.0) + pow(localization[i].second - localization[j].second, 2.0));
      }
    }
    // auto values for tmin and tmax
    if(tmin == tmax) {
      tmin = 0.19023986655081256968990992121, tmax = 0;
      rep(i,k) rep(j,k) tmax = max(tmax, neighborhood[i][j]);
      tmax = min(100.0, sqrt(tmax / 2.0 / 1e-6));
    }
    dbg(tmin _ tmax);
    who.clear();
    rep(i,n) {
      int place = -1;
      double best = 1e111;
      rep(j,k) {
        double nbest = 0;
        rep(c,k) nbest += kernel(j,c,tmax) * func(i,c);
        if(nbest < best) best = nbest, place = j;
      }
      who.push_back(place);
    }
  }
  // auxiliar function
  double ternarySearch(vector< double > &a, vector< double >& b, int maxIter = 200, double inf = 1<<20, double eps = 1e-12) { // a[i] - b[i]*x
    double low = -inf, high = +inf;
    rep(iter,maxIter) {
      double med1 = (low * 2 + high) / 3, f1 = 0;
      #ifdef FUNC_L1
        rep(i,n) f1 += fabs(a[i] - b[i] * med1);
      #elif  FUNC_L2
        rep(i,n) f1 += b[i] * pow(fabs(a[i] - med1),2.0);
      #endif
      double med2 = (low + high * 2) / 3, f2 = 0;
      #ifdef FUNC_L1
        rep(i,n) f2 += fabs(a[i] - b[i] * med2);
      #elif  FUNC_L2
        rep(i,n) f2 += b[i] * pow(fabs(a[i] - med2),2.0);
      #endif
```

```
      if(f1 >= f2) low = med1; else high = med2;
    }
    assert(fabs(low-high) <= eps);
    return (low + high) / 2.0;
  }
  // step 1
  void update(double temperature) {
    vector<double> a(n), b(n);
    rep(c,k) rep(j,p) {
      rep(i,n) b[i] = kernel(who[i],c,temperature);
      #ifdef FUNC_L1
        rep(i,n) a[i] = individual[i][j].first * b[i];
      #elif  FUNC_L2
        rep(i,n) a[i] = individual[i][j].first;
      #endif
      prototype[c][j].first = ternarySearch(a,b);
      #ifdef FUNC_L1
        rep(i,n) a[i] = individual[i][j].first * b[i];
      #elif  FUNC_L2
        rep(i,n) a[i] = individual[i][j].first;
      #endif
      prototype[c][j].second = ternarySearch(a,b);
    }
  }
  // step2
  void fix(double temperature) {
    vector< vector< double > > f = vector< vector< double > >(k, vector< double > (p, 0));
    rep(c,k) rep(j,p) rep(i,n) {
      #ifdef FUNC_L1
        double g = fabs(individual[i][j].first - prototype[c][j].first) + fabs(individual[i][j].second - prototype[c][j].second);
        f[c][j] += kernel(who[i],c,temperature) * g;
      #elif  FUNC_L2
        double g = pow(fabs(individual[i][j].first - prototype[c][j].first),2.0) + pow(fabs(individual[i][j].second - prototype[c][j].second),2.0);
        f[c][j] += kernel(who[i],c,temperature) * g;
      #endif
    }
    vector< double > num(k, 1.0);
    rep(c,k) rep(j,p) {
      num[c] *= pow(f[c][j], 1.0 / p);
    }
    //double all = 1.0;
    rep(c,k) rep(j,p) {
      lambda[c][j] = num[c] / f[c][j];
    }
    //cerr << all << endl;
  }
  // step3
  int partition(double temperature) {
    int change = 0;
    rep(i,n) {
      int place = who[i];
      double best = 0;
      rep(c,k) best += kernel(place,c,temperature) * func(i,c);
      rep(j,k) {
        double nbest = 0;
        rep(c,k) nbest += kernel(j,c,temperature) * func(i,c);
        if(Util::cmp(nbest,best) < 0) best = nbest, place = j;
      }
      if(place != who[i]) change = 1, who[i] = place;
    }
    //vector<int> freq(k,0); rep(i,n) freq[who[i]]++; pv(freq.begin(),freq.end());
    return change;
  }
  void makeTable() {
    int N = k, M = 0;
    map<int,int> mapa; rep(i,n) mapa[from[i]] = 1;
    for(map<int,int>::iterator iter = mapa.begin(); iter != mapa.end(); iter++) iter->second = M++;
    table = vector< vector< int > >(N+1, vector< int >(M+1,0));
    rep(i,N) rep(j,M) rep(k,n) table[i][j] += (who[k] == i && mapa[from[k]] == j);
    rep(i,N) rep(j,M) table[i][M] += table[i][j];
    rep(j,M) rep(i,N) table[N][j] += table[i][j];
    table[N][M] = n;
  }
  // TODO organizar bem a saida
  double process(int init, ostream& fout) {
    const static string space(1<<6,'-');
    #ifdef DETAILED
      fout << fixed << setprecision(3);
      fout << "INITIALIZATION #" << init << "\n" << space << endl;
    #endif
    initialization();
    double temperature = tmax, skip = pow(tmin / tmax, 1.0 / niter), ant = 1e111, prox;
    for(int iteration = 1; iteration <= niter; iteration++) {
      dbg(iteration);
      temperature *= skip;
      prox = energy(temperature);
      #ifdef DETAILED
        makeTable();
        fout << "ITERATION " << setw(3) << iteration << " ::" << endl;
        fout << "* Criterion: " << energy(temperature) << endl;
        fout << "* Global error: " << Validation::globalError(n,table) << endl;
        fout << "* Adjusted rand index: " << Validation::adjustedRandIndex(n,table) << endl;
        fout << "* F measure: " << Validation::fMeasure(n,table) << endl;
        fout << space << endl;
      #endif
      update(temperature);
      #ifdef RELEVANCE
        fix(temperature);
      #endif
      int repeat = partition(temperature);
      if( Util::cmp(prox,ant) > 0) dbg(prox _ ant);
      assert( Util::cmp(prox,ant) <= 0 ), ant = prox;
    }
    return prox;
  }
  void solve(ostream& fout = out) {
    const static string space(1<<6,'#');
    opt = 1e111;
    rep(init,ninit) {
      dbg(init);
      double nopt = process(init+1,fout);
      if( Util::cmp(nopt,opt) < 0 ) {
        opt = nopt, res = who, relevance = lambda, which = init+1, what = prototype;
      }
      #ifdef DETAILED
        fout << "\n" << space << "\n" << endl;
      #endif
    }
    who = res;
    prototype = what;
    lambda = relevance;
    makeTable();
    fout << "(Better solution in initialization #" << which << ")" << endl;
    fout << "* Confusion's table:";
    int nd = ceil(log10(n)) + 1;
    rep(i,(fout << "\n",table.size())) rep(j,table[i].size()) fout << setw(nd) << table[i][j];
    fout << "* Criterion: " << opt << endl;
    fout << "* Global error: " << Validation::globalError(n,table) << endl;
    fout << "* Adjusted rand index: " << Validation::adjustedRandIndex(n,table) << endl;
    fout << "* F measure: " << Validation::fMeasure(n,table) << endl;
    #ifdef RELEVANCE
      fout << "* Releance for each prototype:";
      rep(i,(fout << "\n",relevance.size())) rep(j,relevance[i].size()) fout << relevance[i][j] << "  ";
    #endif
    fout << "\n" << space << endl;
    fout << "* Cluster for each individual:" << endl;
    rep(i,n) fout << i+1 << " : " << who[i] << endl;
```



```
fout << "* Individual who are more alike to a cluster:" << endl; {
  vector< pair<double,int> > minDist(N, pair<double,int>(1e111,-2) );
  rep(i,n) {
    double newDist = func(i, who[i]);
    if(Util::cmp(newDist,minDist[who[i]].first) < 0) minDist[who[i]] = pair<double,int>
(newDist,i);
  }
  rep(i,k) {
    fout << "Cluster " << i+1 << ":" << endl;
    stringstream s1, s2;
    s1 << "prototype:";
    s2 << "individual more similar [" << (minDist[i].second += 1) << "]:";
    string ss1 = s1.str(), ss2 = s2.str();
    while(ss1.size() < ss2.size()) ss1 = string(">") + ss1;
    fout << "\t>>"; fout << ss1;
    rep(j,p) fout << " " << what[i][j].first << ", " << what[i][j].second << ")"; fout << endl;
    fout << "\t< "; fout << ss2;
    if(minDist[i].second != -1) rep(j,p) fout << " (" << individual[minDist[i].second][j].first
<< "," << individual[minDist[i].second][j].second << ")"; fout << endl;
  }
}
fout << "* Grid:"; {
  vector< vector< int > > cnt(dx, vector< int >(dy, 0) );
  rep(i,n) {
    pair<int,int> where = localization[ who[i] ];
    cnt[where.first][where.second]++;
  }
  rep(i,(fout << "\n",dx)) rep(j,dy) fout << " " << setw(3) << cnt[i][j];
}
};
int main(int argc, char **argv) {
  string method;
  #ifdef FUNC_L1
    #ifdef RELEVANCE
      method = "L1-Relevance";
    #else
      method = "L1-Normal";
    #endif
  #elif FUNC_L2
    #ifdef RELEVANCE
      method = "L2-Relevance";
    #else
      method = "L2-Normal";
    #endif
  #endif
  cout << "now running \"" << method << "\"" << endl;
  method = string("Config/") + method + string("-Config.txt");
  if(argc > 1) method = string(argv[1]);
  dbg(method);
  srand(time(NULL));
  Kohonen::read(method);
  Kohonen::open();
  Kohonen::solve();
  Kohonen::close();
  return 0;
}
```

```
1   all:
2     g++ Tg.cpp -o L1-Normal    -O2 -D FUNC_L1
3     g++ Tg.cpp -o L2-Normal    -O2 -D FUNC_L2
4     g++ Tg.cpp -o L1-Relevance -O2 -D FUNC_L1 -D RELEVANCE
5     g++ Tg.cpp -o L2-Relevance -O2 -D FUNC_L2 -D RELEVANCE
6
7   detailed:
8     g++ Tg.cpp -o L1-Normal    -O2 -D FUNC_L1               -D DETAILED
9     g++ Tg.cpp -o L2-Normal    -O2 -D FUNC_L2               -D DETAILED
10    g++ Tg.cpp -o L1-Relevance -O2 -D FUNC_L1 -D RELEVANCE  -D DETAILED
11    g++ Tg.cpp -o L2-Relevance -O2 -D FUNC_L2 -D RELEVANCE  -D DETAILED
12
13  outro:
14    g++ Tg.cpp -o L1-Normal    -O2    -D FUNC_L1
```

Figura 7. A configuração do makefile para a geração dos executáveis

Os algoritmos que foram usados nesse trabalho foram produzidos segundo a compilação do makefile na figura 7, nele os 4 programas, L1 ou L2 e sem ou com fator de relevância, e também uma versão dos mesmos com resultados mais detalhados em cada interação em *detailed.*