Universidade Federal de Pernambuco

Graduação em Ciência da Computação

Centro de Informática

2015.1

Avaliando modelos de clustering colaborativo aplicado à dados relacionais

---

Trabalho de Graduação

Aluno: Diogo Philippini Pontual Branco (dppb@cin.ufpe.br)

Orientador: Francisco de Assis Tenório de Carvalho (fatc@cin.ufpe.br)

Recife, 19 de junho de 2015

Agradecimentos

Agradeço à minha família que sempre me apoiou e aos meus professores que me ajudaram à chegar até aqui, em especial ao professor Francisco de Assis Tenório de Carvalho que me orientou atenciosamente nesse trabalho.

Resumo

Clustering é uma tarefa essencial e frequetemente usada em reconhecimento de padrão, mineração de dados, visão computacional e bioinformática. Seu objetivo é organizar um conjunto de itens em clusters tal que itens contidos num cluster possuem alto grau de similaridade, enquanto itens pertencentes à clusters diferentes possuem alto grau de dissimilaridade.

Cada objeto (ou item) pode ser representado como dados relacionais, onde cada par de objeto é representado por um relacionamento. O caso mais comum é quando se tem uma matriz de dissimilaridades de dados, onde dada posição da matriz representa a dissimilaridade entre dois objetos. Esses objetos podem ser (e geralmente são) complexos, tais como objetos multimídia, o que faz com que o relacionamento entre objetos possa ser descrito por múltiplas matrizes de (dis)similaridade.

O objetivo deste trabalho é analisar os algoritmos de clustering e agregação de dados relacionais dados por múltiplas matrizes de dissimilaridade, tais como o CARDf e o MFCMdd-RWL-S, os implementando e aplicando à bases de dados relacionais.

Sumário

1. Introdução

Clustering é uma tarefa essencial e frequentemente usada em reconhecimento de padrão, mineração de dados, visão computacional e bioinformática [2], tendo ganho popularidade como uma ferramenta eficiente de análise de dados [3]. Seu objetivo é organizar um conjunto de itens em clusters tal que itens contidos num cluster possuem alto grau de similaridade, enquanto itens pertencentes à clusters diferentes possuem alto grau de dissimilaridade [2].

Há vários métodos de clustering na literatura e, ao aplicar diferentes métodos ao mesmo conjunto de dados pode-se obter resultados bem diferentes. Não se sabe qual dos resultados será o correto, mesmo aplicando-se índices de validade de cluster (CVI), não existe um CVI que avalia os resultados do algoritmo de clustering de forma imparcial [6]; os resultados da aplicação de diferentes métodos podem parecer igualmente plausíveis sem conhecimento a priori sobre a distribuição dos dados [5].

A natureza exploratória de tarefas de clustering demanda métodos eficientes que beneficiariam a combinação de vantagens de determinados algoritmos de clustering, o que leva ao estudo de agregação de cluster: dado um conjunto de objetos, um método de agregação de cluster consiste em duas principais etapas: geração, onde cria-se um conjunto de partições do conjunto inicial de objetos, e uma função de consenso, onde uma nova partição, que é a integração de todas as partições obtidas na etapa geração, é computada [6]; sendo está última um dos maiores problemas na agregação de cluster [5].

A idea principal da combinação de diferentes resultados de clustering emergiu como uma abordagem alternativa para a melhora da qualidade dos resultados de algoritmos de clustering. Não é apropriado falar sobre o melhor método de agregação de cluster, mas pode-se estabelecer um método comparativo especifico e determinar qual método seria o mais apropriado [6].

O conjunto de objetos (ou itens) pode ser descrito como *object data* ou como dados relacionais. Em *object data*, cada objeto é descrito por um vetor x de dimensão p, onde cada componente do vetor pertence à R^p e representa uma característica do objeto. Enquanto que em dados relacionais apenas a informação que representa o grau de relação entre os pares de objetos está disponível [1]. Esses objetos podem ser (e geralmente são) complexos, tais como objetos multimídia, o que faz com que o relacionamento entre objetos possa ser descrito por múltiplas matrizes de (dis)similaridade [1].

Há várias razões subjacentes para o uso de múltiplas matrizes de dissimilaridade, exemplos seriam: descoberta de estruturas de dados entre conjutos de dados diferentes (e.g. vários centros bancários, cada um com sua base de dados com informações sobre seus clientes); separação conceitual dos dados (e.g. um certo conjunto de dados sobre uma aplicação multimidia pode ser divido em múltiplas matrizes, cada um pertinente à um determinado coneito sendo avaliado da aplicação); separação temporal dos dados (e.g. um conjunto de países descritos por uma série de índices socio-econômicos coletados em instantes de tempo específicos) [3].

Com essas descrições de visão múltipla (*multi-view*) dos dados, pode-se aplicar técnicas de cluster para analisá-los; a saber cluster colaborativo. Na literatura encontra-se três principais abordagens de cluster colaborativo, a saber: abordagem de concatenação [4], abordagem distribuida [5] e abordagem centralizada [2, 7].

A abordagem de concatenação consiste na concatenação das visões (matrizes) em uma única visão, seja justapondo o conjunto de características ou combinando, indiretamente, as matrizes de proximidade derivadas de cada visão. Já a abordagem distribuida, também conhecida como agregação de cluster, agrupa os objetos de cada visão de forma independente e então procura por uma solução que representa um consenso entre o conjunto de clusters; a principal desvantagem desse método é que eles não reconsideram os agrupamentos previamente formados. Por último, tem-se a estratégia centralizada, que se utiliza das visões múltiplas de forma simultânea para encontrar padrões escondidos nos dados; representa um desafio importante pois requer uma modificação profunda do processo de clustering [7]. Nessa abordagem, cada matriz de dissimilaridade (visão) possui pesos de relevância, isso se dá porque a influência de diferentes matrizes de dissimilaridade não é igualmente importante na definição dos clusters na partição final [2].

Neste contexto, o presente trabalho de graduação visa estudar, analisar, implementar e comparar alguns modelos de *clustering* colaborativo *multi-view* utilizando abordagem centralizada. A sessão 2 apresenta fundamentos necessários para o entendimento deste trabalho; na sessão 3 detalhes de implementação são expostos; na sessão 4 os experimentos realizados são descritos; por fim, na sessão 5 apresenta-se uma conclusão.

2. Fundamentos

Na presente sessão serão mostrados alguns fundamentos, motivações e razões à cerca de *clustering multi-view* **centralizado**.

2.1 Clustering de dados baseados em uma única matriz de dissimilaridade

Tipicamente, algoritmos de clustering trabalham com apenas uma única matriz de dissimilaridade. Porém, como dito anteriormente, os dados geralmente são complexos e podem estar descritos por mais de uma matriz de dissimilaridade; nesse caso, apenas concatena-se as matrizes (por coluna) obtendo-se então uma única matriz. Esse método de concatenação de matrizes não é ideal, pois cada matriz deveria ter uma relevância diferente na classificação dos objetos [1]; na sessão 2.2 entrarei em mais detalhes sobre o porquê.

Algoritmos de clustering, de um modo geral, trabalham de maneira iterativa buscando minimizar uma função objetivo. Isso não é diferente no âmbito de dados

relacionais; [2] propõe um algoritmo para *fuzzy clustering* de dados relacionais que está relacionado, principalmente, com os algoritmos *fuzzy k-medoids* [10].

2.1.1 FANNY

Um outro algoritmo relevante nesse âmbito seria o FANNY [12], que serviu de base para o $CARD_f$ (ver sessão 2.2). O FANNY é relativamente simples mas efetivo; ele procura minimizar a seguinte função objetivo:

$$C = \sum_{v=1}^{k} \frac{\sum_{i,j=1}^{n} u_{iv}^2 u_{jv}^2 d(i,j)}{2\sum_{j=1}^{n} u_{jv}^2} \quad (1)$$

Onde:

- *N* – número de objetos, instâncias.
- *K* – número de clusters.
- *u* – é a matriz *fuzzy membership*, com dimensões *N* por *K*.
- *d(i, j)* – é a distância entre o objeto *i* e o objeto *j*, advinda da matriz de dissimilaridade.

A matriz *fuzzy* deve obedecer às seguintes limitações:

$$u_{iv} \geq 0 \quad \text{para } i = 1, \dots, \text{n}; v = 1, \dots, \text{k} \quad (2)$$

$$\sum_{v=1}^{k} u_{iv} = 1 \quad \text{para } i = 1, \dots, \text{n} \quad (3)$$

O pseudo código do FANNY encontra-se a seguir:

```
matrix function fanny (matrix d, int N, int K, double epsilon) {
    matrix u[N][K];
    init(u);
    double cur = objfun(u, d);
    double prev = cur + epsilon; // forces to enter loop
    while (cur – prev >= epsilon) {
        prev = cur;
        for(int i = 0; i < N; ++i) {
            matrix a = compt_a(u, d);
            set V1; // empty set
            set V2; // mepty set
```

```
                for(int v = 0; v < K; ++v)  {
                        double val = compt_av(a, v);
                        if(val <= 0) {
                                add(V1, v) // adds v into V1 set
                        } else {
                                add(V2, v) // adds v into V2 set
                        }
                }
                foreach(v in V2) {
                        u[i][v] = 0;
                }
                foreach(v in V1) {
                        update(u, a, i, v);
                }
        }
        cur = objfun(u, d);
    }
    return u;
}
```

A função *init(matrix u)* recebe a matriz *fuzzy* e a inicializa com valores aleatórios obedecendo as condições (2) e (3); a função *objfun(matrix u, matrix d)* recebe a matriz *fuzzy* e a matriz de dissimilaridade, aplica a equação (1) e retorna o resultado tipo *double*; a função *compt_a(matrix u, matrix d)* recebe a matriz *fuzzy* e a matriz de dissimilaridade, aplica a equação (4) e retorna uma *matrix* de dimensão *N* por *K*; a função *compt_av(matrix a, int v)* recebe a matriz *fuzzy* e um inteiro (entre 0 e *K-1*), aplica a equação (5) e retorna um *double*; por último, temos a função *update(matrix u, matrix a, int i, int v)* recebe a matriz *fuzzy*, a matriz *a*, um inteiro entre 0 e *N-1* e outro inteiro entre 0 e *K-1*, aplica a equação (6) e atualiza o valor de *u[i][v]*.

Equações:

$$a_{iv} = \frac{2\sum_{j=1}^{n} u_{jv}^2 d(i,j)}{\sum_{j=1}^{n} u_{jv}^2} - \frac{\sum_{j=1}^{n}\sum_{h=1}^{n} u_{jv}^2 u_{hv}^2 d(i,j)}{\left(\sum_{j=1}^{n} u_{jv}^2\right)^2} \quad (4)$$

$$A_v = \frac{1/a_{iv}}{\sum_{w=1}^{k}(1/a_{iw})}, \forall i \in 1, 2, \ldots, n \quad (5)$$

$$u_{iv} = \frac{1/a_{iv}}{\sum_{w \in V1}(1/a_{iw})} \quad (6)$$

2.2 Clustering de dados baseados em múltiplas matrizes de dissimilaridade

Como dito anteriormente, os dados que descrevem os objetos geralmente são complexos e, por tanto, podem ser descritos por múltiplas matrizes. Cada matriz tem uma certa relevância na classificação dos objetos por parte de um algoritmo de clustering baseado em múltiplas matrizes de dissimilaridade; no caso em que todas as matrizes possuem a mesma relevância, não seria muito diferente de concatenar as matrizes de dissimilaridade numa só e aplica um algoritmo de clustering de dados baseado em uma única matriz. É justamente por existir diferentes relevâncias entre as matrizes que torna algoritmos de clustering baseados em múltiplas matrizes um tema interessante, relevante e essencial.

Especialmente quando se tem dados relacionais, técnicas que não levam em conta diferentes relevâncias para as matrizes tem uma performance inferior se comparados à algoritmos que levam isso em conta [1, 2]. [7] apresenta algumas motivações para o uso de *multi-view clustering*: **reuso de conhecimento**, consiste em considerar diferentes *clusterings* feitos anteriormente num único conjunto de dados afim de processar esse conjunto de dados com uma nova descrição, nesse caso, cada *clustering* seria uma *view*; **computação distribuída**, quando os dados estão espalhados entre diferentes sítios com a impossibilidade de coletá-los de um único sítio devido à razões técnicas ou confidenciais; **melhora da qualidade e robustês** dos resultados do *clustering*.

A seguir, será exposto alguns algoritmos de *clustering multi-view*, alguns deles foram implementados e testados por mim como parte deste trabalho de graduação; na sessão 4 encontra-se detalhes a cerca.

2.2.1 *Clustering and Aggregating Relational Data* (CARD)

O algoritmo CARD foi proposto por [1]; ele possui duas "versões", o $CARD_f$ e o $CARD_r$, ambos são extensões dos algoritmos RFCM [11] e FANNY [12] para *clustering multi-view* centralizado. Ambos seguem, aproximadamente, a mesma ideia: iteram até que a diferença absoluta de um dado critério de adequação (dado pela função de adequação) da iteração atual, com relação à iteração anterior, seja menor que um *threshold*; a cada iteração, as *fuzzy memberships* e os pesos de relevância de cada matriz são atualizados. O critério de adequação é dependente dos valores das *fuzzy memberships* e dos pesos de relevância e é esperado que esse critério sempre diminua, ou se matenha, a cada iteração.

O $CARD_f$ procura minimizar a seguinte função objetivo:

$$C = \sum_{v=1}^{k} \frac{\sum_{i,j=1}^{n} u_{iv}^2 u_{jv}^2 \sum_{h=1}^{p} \lambda_{hv}^s d_h(i,j)}{2\sum_{j=1}^{n} u_{jv}^2} \tag{7}$$

Onde:

- s – um expoente discriminante no intervalo [1, ∞).
- p – número de matrizes de dissimilaridade.
- λ – matriz de pesos de relevância para cada classe de cada matriz de dissimilaridade, com dimensão p por k.
- $d_h(i, j)$ – distância entre os objetos i e j na matriz de dissimilaridade h.

A matriz de pesos deve obedecer às seguintes limitações:

$$\lambda_{hv} \in [0,1] \quad \forall i, s \tag{8}$$

$$\sum_{h=1}^{p} \lambda_{hv} = 1 \quad \forall v \tag{9}$$

Para explicação sobre demais parâmetros, ver sessão 2.2.1. A matriz *fuzzy* deve atender às restrições (2) e (3). O pseudo código do $CARD_f$ é o seguinte:

```
// d is an array of matrixes of size p.
matrix function cardf (int p, matrix d[p], int N, int K, double epsilon, int s) {
    matrix u[N][K];
    matrix lamb[p][k];
    init(u, lamb);
    double cur = objfun(u, d, lamb, s);
```

```
    double prev = cur + epsilon; // forces to enter loop
    while (cur – prev >= epsilon) {
        prev = cur;
        for(int i = 0; i < N; ++i) {
            matrix a = compt_a(u, d, lamb);
            set V1; // empty set
            set V2; // mepty set
            for(int v = 0; v < K; ++v)  {
                double val = compt_av(a, v);
                if(val <= 0) {
                    add(V1, v) // adds v into V1 set
                } else {
                    add(V2, v) // adds v into V2 set
                }
            }
            foreach(v in V2) {
                u[i][v] = 0;
            }
            foreach(v in V1) {
                update(u, a, i, v);
            }
            update_weights(u, d, lamb, s);
        }
        cur = objfun(u, d, lamb, s);
    }
    return u;
}
```

A função *init(matrix u, matrix lamb)* recebe a matriz *fuzzy* e a matriz de pesos, a matriz *fuzzy* é inicializada com valores aleatórios obedecendo as condições (2) e (3) e

cada célula da matriz de pesos é inicializada com o valor 1 / *p*; a função *objfun(matrix u, matrix d[], matrix lamb, int s)* recebe a matriz *fuzzy*, o array de matrizes de dissimilaridade, a matriz de pesos e o expoente discriminante *s*, aplica a equação (7) e retorna o resultado tipo *double*; a função *compt_a(matrix u, matrix d[], matrix lamb)* recebe a matriz *fuzzy*, o array de matrizes de dissimilaridade e a matriz de pesos, aplica a equação (10) e retorna uma *matrix* de dimensão *N* por *K*; a função *compt_av(matrix a, int v)* recebe a matriz *fuzzy* e um inteiro (entre 0 e *K-1*), aplica a equação (5) e retorna um *double* (similar ao FANNY); a função *update(matrix u, matrix a, int i, int v)* recebe a matriz *fuzzy*, a matriz *a*, um inteiro entre 0 e *N-1* e outro inteiro entre 0 e *K-1*, aplica a equação (6) e atualiza o valor de *u[i][v]* (similar ao FANNY); por último, temos a função *update_weights(matrix u, matrix d[], matrix lamb, int s)* que recebe a matriz *fuzzy*, as matrizes de dissimilaridade, a matriz de pesos e o expoente *s*, aplica a equação (11) atualizando os valores na matriz de pesos *lamb*.

Equações

$$a_{iv} = \frac{2\sum_{j=1}^{n} u_{jv}^2 d(i,j)}{\sum_{j=1}^{n} u_{jv}^2} - \frac{\sum_{j=1}^{n}\sum_{h=1}^{n} u_{jv}^2 u_{hv}^2 \sum_{h=1}^{p} \lambda_{hv}^s d_h(i,j)}{\left(\sum_{j=1}^{n} u_{jv}^2\right)^2} \quad (10)$$

$$\lambda_{hv} = \frac{1}{\sum_{i=1}^{p} (D_{hv}/D_{iv})^{1/(q-1)}} \quad (11)$$

$$D_{hv} = \sum_{i=1}^{n}\sum_{j=1}^{n} u_{iv}^2 u_{jv}^2 d_h(i,j)$$

2.2.2 *Collaborative Fuzzy K-means Method* (CoFKM)

O CoFKM [7] é um algoritmo de *fuzzy clustering* que é uma extensão do FKM [8]. Uma característica marcante dele é o fato de generalizar as três estratégias de fusão (a saber, concatenação, distribuida ou conjuntos de cluster, centralizada). É possível instanciar uma única estratégia de combinação (fusão) dependendo apenas do ajuste de um único parâmetro.

Esse algoritmo também usa a ideia de centroides (ou protótipos) que são representantes de cada classe para cada matriz de dissimilaridade; esses protótipos são atualizados a cada iteração bem como as *fuzzy memberships* até que um critério de convergência seja satisfeito. O critério de convergência pode ser a diferença absoluta do critério de adequação da iteração atual em relação à iteração anterior.

2.2.3 *Partitioning fuzzy K-medoids clustering algorithm based on multiple dissimilarity matrixes* (MFCMdd)

O MFCMdd foi um proposto em [2]; ele é um algoritmo de *fuzzy clustering* centralizado e que foi baseado no SFCMdd também proposto em [2], que por sua vez foi baseado nos algoritmos *fuzzy k-medoids*. O SFCMdd seria o *partitioning fuzzy K-medoids clustering algorithm based on a single dissimilarity matrix*, que particiona dados relacionais baseados em uma única matriz de dissimilaridade.

O MFCMdd possui variações dependendo de como os pesos de relevância de cada matriz é estimado e acerca do critério de limitação dos *fuzzy memberships*. Os pesos de relevância podem ser estimados tanto localmente quanto de forma global; já os critérios de limitação para os *fuzzy memberships* podem ser dois para um dado objeto: a soma de seus graus pertinência (uma linha na matriz de *fuzzy membership*) deve ser igual à um, ou o produto de seus graus de pertinência deve ser igual à um.

Com isso temos: MFCMdd-RWL, onde os pesos de relevância são estimados localmente, que se "divide" em MFCMdd-RWL-S e MFCMdd-RWL-P, sendo o S e o P para as respectivas limitações dos *fuzzy memberships*. Analogamente temos o MFCMdd-RWG onde os pesos de relevância são estimados globalmente, e deve ser MFCMdd-RWG-S ou MFCMdd-RWG-P.

O MFCMdd-RWL-S foi o escolhido para ser analisado mais a fundo neste trabalho; isso se deu porque: foi um requisito do orientador e também por ser mais comparável com o $CARD_f$. Vale ressaltar que o MFCMdd-RWL-P também seria válido, mas, reiterando, utilizar o MFCMdd-RWL-S foi um requisito do orientador.

O MFCMdd-RWL-S procura minimizar a seguinte função objetivo:

$$C = \sum_{v=1}^{k} \sum_{i=1}^{n} u_{iv}^{m} \sum_{j=1}^{p} \lambda_{vj}^{s} \sum_{e \in G_v} d_j(e_i, e) \qquad (12)$$

Onde:

- m – parâmetro que controla o grau de *fuzzy membership* para cada objeto.
- G_v – conjunto de objetos protótipos para o cluster v, de cardinalidade $|q|$.
- $d_j(e_i, e)$ – distância entre o objeto e_i e o objeto e na matriz de dissimilaridade j, no caso, e seria um protótipo do cluster v.

Para explicação de demais parâmetros, ver sessões 2.2.1 e 2.2.2. A matriz *fuzzy* deve atender às restrições (2) e (3) e a matriz de pesos deve atender às restrições (8) e (9). O pseudo-código do MFCMdd-RWL-S é o seguinte:

```
// d is an array of matrixes, each NxN, of size p.
matrix function mfcm (int p, matrix d[p], int N, int K, double epsilon, int s, int q) {
    matrix u[N][K];
    matrix lamb[p][k];
    set G[K];
    foreach(g in G) {
            g = new_set(q); // creates an empty set of size q
    }
    init(lamb, G, N);
    double cur = objfun(u, d, lamb, G, s, m);
    double prev = cur + epsilon; // forces to enter loop
    update(u, d, lamb, G, s, m);
    while (cur – prev >= epsilon) {
            prev = cur;
            best_prot(u, d,  lamb, G,  s, m);
            best_weights(u, d, lamb, G, s, m);
            update(u, d, lamb, G, s, m);
            cur = objfun(u, d, lamb, G, s, m);
    }
    return u;
}
```

```
// updates the prototypes sets
void function best_prot(matrix u, matrix d[p], matrix lamb, set G[K], int s, int m) {
    foreach(g in G) {
        empty(g); // empties the set
        while(!isfull(g)) {
            int l = choose_prot(u, d, lamb, s, m);
            g += l // inserts l into set
        }
    }
}
```

A função *init(matrix lamb, set G[K], int N)* recebe a matriz de pesos, o array de conjuntos de protótipos e o número de instâncias *N*, cada célula da matriz de pesos é inicializada com o valor 1 / *p* e os protótipos são escolhidos aleatoriamente de forma que nenhum dos conjuntos sejam iguais; a função *objfun(matrix u, matrix d[p], matrix lamb, set G[K], int s, int m)* recebe a matriz *fuzzy*, o array de matrizes de dissimilaridade, o array de conjuntos de protótipos, o expoente discriminante *s* e parâmetro de controle de grau *fuzzy*, aplica a equação (12) e retorna o resultado tipo *double*; a função *update(matrix u, matrix d[p], matrix lamb, set G[K], int s, int m)* possui os mesmos parâmetros da função *objfun*, aplica a equação (13) atualizando toda a matriz *fuzzy*; a função *choose_prot(matrix u, matrix d[p], matrix lamb, int s, int m)* aplica a equação (14) e retorna um *int*, correspondente ao objeto escolhido como protótipo; a função *best_weights(matrix u, matrix d[p], matrix lamb, set G[K], int s, int m)* possui os mesmos parâmetros da função *objfun*, aplica a equação (15) atualizando os pesos de relevância.

Equações:

$$u_{iv} = \sum_{h=1}^{k} \left[\left(\frac{\sum_{j=1}^{p} \lambda_{vj}^{s} \sum_{e \in G_v} d_j(e_i,e)}{\sum_{j=1}^{p} \lambda_{hj}^{s} \sum_{e \in G_h} d_j(e_i,e)} \right)^{1/(m-1)} \right]^{-1} \quad (13)$$

$$\forall i \in 1, 2, \ldots, N; \ \forall v \in 1, 2, \ldots, K$$

$$argmin_{1 \le h \le n} \sum_{i=1}^{n} u_{ik}^{m} \sum_{j=1}^{p} \lambda_{kj}^{s} d_j(e_i,e) \quad (14)$$

$$\lambda_{vj} = \left[\sum_{h=1}^{p} \left(\frac{\sum_{i=1}^{n} u_{iv}^{m} \sum_{e \in G_v} d_j(e_i,e)}{\sum_{i=1}^{n} u_{iv}^{m} \sum_{e \in G_v} d_h(e_i,e)} \right)^{1/(s-1)} \right]^{-1} \quad (15)$$

2.3 Índices

Para avaliar os *fuzzy custers* e *hard clusters* gerados pelos algoritmos, alguns índices podem ser utilizados. Os índices utilizados no presente trabalho são descritos brevemente à seguir.

2.3.1 Íncides *hard*

Os índices presentes nesta subsessão avaliam os *hard clusters* gerados pelos algoritmos. Apesar de todos os algoritmos, neste trabalho, gerarem *fuzzy clusters*, pode-se transformá-los em *hard clusters*; dado o grau de pertinência de um objeto, representado por um vetor onde cada valor está no intervalo [0, 1] e a soma dos valores deve ser igual à 1, o *hard cluster* para esse objeto teria o valor 1 para o valor máximo no *fuzzy cluster* e 0 para os demais valores. Matematicamente:

$$h_{iv} = \begin{cases} 1 \mid argmax_{1 \le v \le K} u_{iv} \\ 0 \end{cases} \quad \forall i \in 1,..,N;\ \forall v \in 1,..,K \tag{16}$$

Todos os índices *hard* usados se baseiam na matriz de confusão mostrada abaixo. Cada célula n_{ij} contém o número de objetos classificados como sendo da classe *j* por um modelo e que tem o rótulo de classe *i*.

Matriz de confusão:

Classes	Clusters					
	Q_1	⋯	Q_i	⋯	Q_k	∑
P_1	n_{11}	⋯	n_{1j}	⋯	n_{11}	$n_{1.} = \sum_{j=1}^{k} n_{1j}$
⋮	⋮	⋯	⋮	⋯	⋮	⋮
P_i	n_{i1}	⋯	n_{ij}	⋯	n_{ik}	$n_{i.} = \sum_{j=1}^{k} n_{ij}$
⋮	⋮	⋯	⋮	⋯	⋮	⋮
P_m	n_{m1}	⋯	n_{mj}	⋯	n_{mk}	$n_{m.} = \sum_{j=1}^{k} n_{mj}$
∑	$n_{.1} = \sum_{i=1}^{m} n_{i1}$	⋯	$n_{.j} = \sum_{i=1}^{m} n_{ij}$	⋯	$n_{.k} = \sum_{i=1}^{m} n_{ik}$	$n = \sum_{i=1}^{m} \sum_{j=1}^{k} n_{ij}$

2.3.1.1 F-measure

Para calcular a f-measure é preciso formar uma matriz *m x k*, onde *m* é número de classes e *k* o número de clusters. Cada célula da matriz é computada pela equação (17).

$$f_{ij} = \frac{2n_{ij}}{\sum_{h=1}^{m} n_{hj} + \sum_{v=1}^{k} n_{iv}} \tag{17}$$

Para computar a f-measure a partir da matriz formada utiliza-se a equação (18). O valor desse índice está no intervalo [0,1], no qual o valor 1 indica concordância perfeita entre partições.1

$$fmeasure = \frac{1}{n} \sum_{i=1}^{m} n_{i\,.}\, max_{1 \le j \le k} f_{ij} \tag{18}$$

2.3.1.2 Overall Error Rate of Classification (OERC)

A taxa de erro total de classificação (OERC) é calculada de maneira direta a partir da matriz de confusão através da equação (19). Ele assume valores no invervalo [0,1], no qual o valor 0 seria o melhor caso.

$$OERC = 1 - \frac{\sum_{j=1}^{k} max_{1 \le i \le m} n_{ij}}{n} \tag{19}$$

2.3.1.3 Corrected Rand (CR)

Seja **y** e **y'** vetores discretos representando, respectivamente, as partição *hard* de um método de particionamento e os rótulos das classes. Seja $y_i \in \{1, \dots, K\}$ e $y'_i \in \{1, \dots, L\}$, respectivamente, observações de **y** e **y'**, onde $y_i = k$ indica que a observação $\mathbf{x_i}$ pertence à partição *k*. Para um par de (y_i, y_j) pode-se definir as seguintes funções:

$$\boldsymbol{l}(y_i = y_j) = \begin{cases} 1, & se\ y_i = y_j \\ 0, & caso\ contrário \end{cases} \tag{20}$$

$$\boldsymbol{l}(y'_i = y'_j) = \begin{cases} 1, & se\ y'_i = y'_j \\ 0, & caso\ contrário \end{cases} \tag{21}$$

A partir disto, pode-se definir os seguintes termos:

$$a = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \boldsymbol{l}(y_i = y_j)\boldsymbol{l}(y'_i = y'_j)$$

$$b = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(1 - \boldsymbol{l}(y_i = y_j)\right) \boldsymbol{l}(y'_i = y'_j)$$

$$c = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \boldsymbol{l}(y_i = y_j) \left(1 - \boldsymbol{l}(y'_i = y'_j)\right)$$

$$d = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \left(1 - \boldsymbol{l}(y_i = y_j)\right) \left(1 - \boldsymbol{l}(y'_i = y'_j)\right) \quad (22)$$

Desses termos, calcula-se o CR:

$$CR = \frac{(a+d) - \left((a+b)(a+c)+(c+d)(b+d)\right)p^{-1}}{p - \left((a+b)(a+c)+(c+d)(b+d)\right)p^{-1}} \quad p = a+b+c+d \quad (23)$$

O CR assume valores no intervalo [-1,1], onde 1 representa concordância perfeita enquanto que valores próximos ou menos que 0 indicam concordâncias ocorrendo ao acaso.

2.3.2 Índices *fuzzy*

Os índices presentes nesta subsessão avaliam os *fuzzy clusters* gerados pelos algoritmos. Todos eles se baseiam na matriz de contigência; para gerar a matriz de contigência é necessário gerar matrizes de coincidência partir da partição *fuzzy* e dos rótulos de classe dos objetos. Para todos os índices aqui explicitados, quanto maior seu valor, mais similares são as partições.

O primeiro passo é transformar os rótulos de classe em uma matriz de partição *hard*, gerando uma matriz binária. A partir daí gera-se as matrizes de coincidência $\Psi^{(1)}$ e $\Psi^{(2)}$ para as duas matrizes de partição:

$$\Psi = \left[\psi_{ij}\right], 1 \leq i, j \leq N, \quad \psi_{ij} = \sum_{v=1}^{K} u_{iv} u_{jv} \quad (24)$$

Computa-se então a matriz de contingência:

	$\boldsymbol{\psi_{ij}^{(2)} = 1}$	$\boldsymbol{\psi_{ij}^{(2)} = 0}$	$\sum$
$\boldsymbol{\psi_{ij}^{(1)} = 1}$	N_{SS}	N_{SD}	$N_{S.}$
$\boldsymbol{\psi_{ij}^{(1)} = 1}$	N_{DS}	N_{DD}	$N_{D.}$
$\sum$	$N_{.S}$	$N_{.D}$	$N_{..}$

$$N_{SS}\left(\psi^{(1)},\psi^{(2)}\right)=\sum_{i=2}^{N}\sum_{j=1}^{i-1}\psi_{ij}^{(1)}\psi_{ij}^{(2)}$$

$$N_{SD}\left(\psi^{(1)},\psi^{(2)}\right)=\sum_{i=2}^{N}\sum_{j=1}^{i-1}\psi_{ij}^{(1)}\left(1-\psi_{ij}^{(2)}\right)$$

$$N_{DS}\left(\psi^{(1)},\psi^{(2)}\right)=\sum_{i=2}^{N}\sum_{j=1}^{i-1}\left(1-\psi_{ij}^{(1)}\right)\psi_{ij}^{(2)}$$

$$N_{DD}\left(\psi^{(1)},\psi^{(2)}\right)=\sum_{i=2}^{N}\sum_{j=1}^{i-1}\left(1-\psi_{ij}^{(1)}\right)\left(1-\psi_{ij}^{(2)}\right) \quad (25)$$

2.3.2.1 Rand statistics (RAND)

O RAND é calculado através da fórmula (26).

$$Q_{Rand}\left(\psi^{(1)},\psi^{(2)}\right)=\frac{N_{SS}+N_{DD}}{N_{..}} \quad (26)$$

2.3.2.2 Coeficiente de Jaccard

O coeficiente de Jaccard é calculado através da fórmula (27).

$$Q_{Jaccard}\left(\psi^{(1)},\psi^{(2)}\right)=\frac{N_{SS}}{N_{SS}+N_{SD}+N_{DS}} \quad (27)$$

2.3.2.3 Folkes-Mallows (FM)

O índice FM é calculado através da fórmula (28).

$$Q_{FM}\left(\psi^{(1)},\psi^{(2)}\right)=\frac{N_{SS}}{\sqrt{(N_{SS}+N_{SD})+(N_{SS}+N_{DS})}} \quad (28)$$

2.3.2.4 Hubert

O índice de Hubert é calculado através da equação (29).

$$Q_{FM}(\psi^{(1)}, \psi^{(2)}) = \frac{N_{..}N_{SS} - N_{S.}N_{.S}}{\sqrt{N_{S.}N_{.S}N_{D.}N_{.D}}} \quad (29)$$

3. Implementação

Para o presente trabalho de graduação foram implementados os seguintes algoritmos: FANNY, $CARD_f$ e MFCMdd-RWL-S. Foram implementados também diversos índices estatísticos que avaliam qualitativamente e quantitativamente os resultados de cada algoritmo, tanto para *hard* como *fuzzy membership*; os índices implementados foram: *F-measure* [13], *overall error rate of classification* [13], *corrected rand* [14], *rand statistics* [1], *Jaccard coefficient* [1], *Folks-Mallows index* [1] e o *Hubert index* [1].

Todos os algoritmos e índices foram implementados em C. Também foi preciso criar alguns programas auxiliares para a manipulação dos conjunto de dados usados, também escritos por mim em na linguagem C. Implementei também alguns scripts *batch* (Windows) e *shell* (Linux) a fim de facilitar a condução dos experimentos (discutido numa sessão mais à frente).

Cada algoritmo possui um programa (executável) próprio e independente; "fanny" implementa o FANNY, "cardf" implementa o $CARD_f$, "mfcm" implementa o MFCMdd-RWL-S. Há também um programa separado (independente) que é responsável por calcular os índices estatísticos, a saber, "fhidx". Todos os programas rodam no tanto no ambiente Windows quanto no Linux (testado no Ubuntu) e quanto aos scripts, foram implementados equivalentes em *bash* e *shell*; então é possível rodar experimentos em ambos ambientes.

A linguagem C foi escolhida não só por preferêncial pessoal, mas também pelo fato do programa compilado ter uma execução mais rápida (dado que o programador saiba o que esteja fazendo) o que é desejavel no contexto desse trabalho, visto que são vários algoritmos à serem aplicados à vários conjuntos de dados que podem ser relativamente grandes (especialmente se relavarmos em conta não só o número de instâncias, mas o fato de se ter múltiplas matrizes para cada conjunto de dado), o que leva muito tempo executando todas essas rotinas.

Portabilidade também foi um norte. Primeiro porque o programa estava sendo desenvolvido no ambiente Linux mas deveria rodar no ambiente Windows (requisito do orientador); o ambiente Linux foi escolhido para desenvolvimento por ser melhor para se desenvolver em C. Segundo, por uma razão pessoal mas não menos importante, porque prezo pela portabilidade entre esses dois ambientes.

Para compilar o programa em C no Windows foi nescessário utilizar o Cygwin (https://cygwin.com/) , que contém a cygwin1.dll que provê uma substancial

funcionalidade de API POSIX. Para rodar o programa, já compilado, no ambiente Windows basta ter a cygwin1.dll. Temos então que o código foi escrito apenas uma vez no ambiente Linux e pôde ser compilado no Windows sem alterações.

A seguir, será descrito, em alto nível, cada programa implementado além de seus parâmetros e saídas.

3.1 fanny

O programa "fanny" implementa o algoritmo FANNY; FANNY é um algoritmo de *clustering single-view*, ou seja, utiliza apenas uma matriz de dissimilaridade. [1] e [12] serviram de referência para a implementação. O uso de programa se dá através da seguinte chamada:

fanny [OPTIONS] FILE NOBS CLUSTER

Onde:

- *FILE* – Um arquivo, do diretório atual, contendo o conjunto de dados a ser carregado. Cada linha deve ter uma obsevação e cada coluna deve ser separada por espaço. Espera-se que tenha pelo menos *NOBS* linhas.
- *NOBS* – Número de observações a serem usados; deve ser, no máximo, o número de linhas em *FILE*.
- *CLUSTER* – Número de clusters a ser considerado pelo algoritmo.
- *OPTIONS*:
    - *–T, --iterations* – número máximo de iterações que será realizado pelo algoritmo. Se esse valor não for definido ou for igual a zero, o critério de parada do algoritmo será, somente, verificar se a diferença absoluta entre o critério de adequação da iteração atual e o da iteração anterior é menor que *epsilon*.
    - *–v, --verbosis* – o programa irá exibir mais informações ao rodar.
    - *–epsilon* – define o valor de *epsilon* a ser usado; caso não seja definido, o valor padrão é 0.01.
    - *-o, --output* – arquivo em que será escrito a matriz *fuzzy membership*. Esse arquivo pode ser usado posteriormente pelo programa *fhidx* para computar índices.
    - *–h, --help* – exibe a mensagem de ajuda e termina o programa.

Um exemplo de chamada seria:

fanny –T 100 –v –epsilon 0.001 iris-dist.data 150 3

O programa irá rodar até 100 iterações (podendo parar antes disso se o critério de parada para *epsilon* tiver sido antingido), exibir mais informações (*-v*), usar o valor de

epsilon 0.001; o arquivo de dados usado é o *iris-dist.data* com 150 instâncias, considerando 3 clusters.

3.2 cardf

O programa "cardf" implementa o algoritmo de *clustering multi-view* centralizado $CARD_f$. [1] serviu de referência para a implementação e o código usado como base foi o do programa "fanny", pois a principal diferença entre os dois algoritmos é que o $CARD_f$ utiliza uma soma ponderada das distancias (advindas das diferentes matrizes de dissimilaridade) nas equações, enquanto que o FANNY utiliza apenas uma soma das distancias advindas da (única) matriz de dissimilaridade. O uso do programa se dá através da seguinte chamada:

cardf [OPTIONS] DATASET NMATRIX NOBS CLUSTER

Onde:

- *DATASET* - O nome do conjunto de dados a ser usado. Note que, para cada matriz de dissimilaridade a ser usada pelo programa, deve-se nomeá-las da seguinte forma: *nome*-dist-*XX*.data; *nome* seria um nome para o conjunto de dados, *XX* seria um número entre 01 e 99 indicando o número da matriz. Por exemplo, para criar um conjunto de dados chamado *iris* com quatro matrizes de dissimilaridade teria-se os arquivos: *iris*-dist-*01*.data, *iris*-dist-*02*.data, *iris*-dist-*03*.data e *iris*-dist-*04*.data.
- *NMATRIX* – Número de matrizes de dissimilaridade a serem utilizadas; espera-se que o conjunto de dados tenha pelo menos *NMATRIX* matrizes.
- *NOBS* – Número de observações, ou instâncias, do conjunto de dados. Cada arquivo contendo uma matriz de dissimilaridade deve ter pelo menos *NOBS* linhas.
- *CLUSTER* – Número de clusters a ser considarado pelo algoritmo.
- *OPTIONS*:
    - *–T, --iterations* – número máximo de iterações que será realizado pelo algoritmo. Se esse valor não for definido ou for igual a zero, o critério de parada do algoritmo será, somente, verificar se a diferença absoluta entre o critério de adequação da iteração atual e o da iteração anterior é menor que *epsilon*.
    - *–v, --verbosis* – o programa irá exibir mais informações ao rodar.
    - *–epsilon* – define o valor de *epsilon* a ser usado; caso não seja definido, o valor padrão é 0.01.
    - *-o, --output* – arquivo em que será escrito a matriz *fuzzy membership*. Esse arquivo pode ser usado posteriormente pelo programa *fhidx* para computar índices.
    - *–h, --help* – exibe a mensagem de ajuda e termina o programa.

Um exemplo de chamada seria:

cardf –T 100 –v –epsilon 0.001 –o fmtx.txt wine 13 178 3

O programa irá rodar até 100 iterações (podendo parar antes disso se o critério de parada para *epsilon* tiver sido antingido), exibir mais informações (*-v*), usar o valor de *epsilon* 0.001; a matriz *fuzzy* será escrita no arquivo *fmtx.txt*; o conjunto de dados a ser usado é o *wine* com suas 13 matrizes e 178 instâncias, considerando 3 clusters.

3.3 mfcm

O programa "mfcm" implementa o algoritmo MFCMdd-RWL-S. [2] serviu de referência para a implementação. Uma vez que o MFCMdd-RWL-S se diferencia um pouco mais em relação ao $CARD_f$ e do FANNY, ele foi escrito "do zero", ou seja, nenhum outro código foi usado como base. A principal diferença está nas equações, tais como a função objetivo. O uso do programa se dá através da seguinte chamada:

mfcm [OPTIONS] DATASET NMATRIX NOBS CLUSTER

Onde:

- *DATASET* - O nome do conjunto de dados a ser usado. Note que, para cada matriz de dissimilaridade a ser usada pelo programa, deve-se nomeá-las da seguinte forma: *nome*-dist-*XX*.data; *nome* seria um nome para o conjunto de dados, *XX* seria um número entre 01 e 99 indicando o número da matriz. Por exemplo, para criar um conjunto de dados chamado *iris* com quatro matrizes de dissimilaridade teria-se os arquivos: *iris*-dist-*01*.data, *iris*-dist-*02*.data, *iris*-dist-*03*.data e *iris*-dist-*04*.data.
- *NMATRIX* – Número de matrizes de dissimilaridade a serem utilizadas; espera-se que o conjunto de dados tenha pelo menos *NMATRIX* matrizes.
- *NOBS* – Número de observações, ou instâncias, do conjunto de dados. Cada arquivo contendo uma matriz de dissimilaridade deve ter pelo menos *NOBS* linhas.
- *CLUSTER* – Número de clusters a ser considarado pelo algoritmo.
- *OPTIONS*:
    - *–T, --iterations* – número máximo de iterações que será realizado pelo algoritmo. Se esse valor não for definido ou for igual a zero, o critério de parada do algoritmo será, somente, verificar se a diferença absoluta entre o critério de adequação da iteração atual e o da iteração anterior é menor que *epsilon*.
    - *–v, --verbosis* – o programa irá exibir mais informações ao rodar.

- –*epsilon* – define o valor de *epsilon* a ser usado; caso não seja definido, o valor padrão é 0.01.
- --*teta* – define o parâmetro *teta* a ser usado pelo algoritmo. O valor padrão é 0 (zero).
- --*prot* – define o número de protótipos a serem usados pelo algoritmo.
- *-o, --output* – arquivo em que será escrito a matriz *fuzzy membership*. Esse arquivo pode ser usado posteriormente pelo programa *fhidx* para computar índices.
- –*h, --help* – exibe a mensagem de ajuda e termina o programa.

3.4 fhidx

O programa "*fhidx*" implementa os índices que podem ser extraídos a partir da matriz *fuzzy* resultante de algum algoritmo e os rótulos, reais, de cada objeto. Os índices implementados são os descritos no início desta sessão. O uso do programa se dá através da seguinte chamada:

fhidx [OPTIONS] FMTX_FILE DATA_FILE NCOL

Onde:

- *FMTX_FILE* – arquivo que contém a matriz *fuzzy* resultante de algum algoritmo; a primeira linha desse arquivo deve ter as dimensões da matriz *fuzzy*. Esse arquivo seria a saída utilizada no opção –*o* dos outros programas.
- *DATA_FILE* – arquivo que contém os rótulos de cada objeto; geralmente é o arquivo contendo os dados originais, então pode ter outras colunas além da coluna de rótulos; deve ter *NCOL* colunas.
- *NCOL* – número de colunas do *DATA_FILE*.
- *OPTIONS*:
  - --*label-col* – o número da coluna que contém os rótulos no *DATA_FILE*; deve ser um número no intervalo [0, *NCOL*).

4. Experimentos

No presente trabalho os algoritmos FANNY, $CARD_f$ e MFCMdd-RWL-S foram implementados e alimentados com diferentes conjuntos de dados. Para mais detalhes de cada algoritmo veja a sessão 2; para mais detalhes da implementação de cada algoritmo veja a sessão 3.

Os conjuntos de dados foram retirados da *UCI Machine Learning Repository* (https://archive.ics.uci.edu/ml/datasets.html); os conjuntos de dados escolhidos foram: Iris, Wine, e Multiple Features. Cada conjunto de dados será descrito nesta sessão; vale

ressaltar, de antemão, que todas as distâncias utilizadas foram euclideanas [2], uma medida propicia para os conjuntos de dados utilizados.

Os conjuntos Iris e Wine possui apenas uma matriz que descreve os dados, portanto, antes de aplicá-los nos algoritmos $CARD_f$ e MFCMdd-RWL-S, criou-se *q* matrizes de distâncias, onde *q* seria o número de colunas presente em cada conjunto de dados. Ou seja, temos o caso "extremo" em que cada coluna do conjunto de dados é tratada com uma visão. No caso do FANNY, apenas uma visão foi gerada calculando as distâncias euclideanas entre os objetos levando em conta todos os seus atributos.

Já o conjunto Multiple Features contém seis matrizes que descrevem os dados, portanto, ao aplicá-los no algoritmo FANNY, utilizou-se a estratégia de concatenação (sessão 1 e 2) e depois aplicou-se as distâncias entre cada objeto levando em conta todos os seus atributos, gerando assim uma matriz de dissimilaridade. No caso do $CARD_f$ e MFCMdd-RWL-S criou-se seis matrizes de dissimilaridade, uma para cada visão, utilizando a distância euclideana considerando todos os atributos dos objetos por visão.

Para todos os experimentos o valor de ε foi 0.001 (através da opção *--epsilon*) e o número de iterações foi fixado em 150 (opção *–T*). Para os conjuntos de dados Iris e Wine o valor de *K* foi 3 em todos os algoritmos, enquanto que o conjunto de dados Mutiple Features teve o valor de *K=10*; em outras palavras, o valor de *K* foi igual ao número de clusters dados pelos rótulos de cada objeto (Iris e Wine possui 3 classes e Multiple Features possui 10 classes). Para o algoritmo MFCMdd-RWL-S o número de protótipos usado foi 1,2,3,5 e 10; notação MFCM-*q* (e.g. MFCM-1) indica a execução do MFCMdd-RWL-S considerando *q* protótipos. Cada algoritmo foi executado 50 vezes em cada conjunto de dados; os resultados são analisados a seguir.

É importante ressaltar que para alguns conjuntos de dados os algoritmos FANNY e $CARD_f$ apresentaram alta oscilação em seu critério de adequação (a cada iteração). Essa oscilação seria normal caso fosse uma questão de precisão de casas decimais, o que não parece ser o caso, pois a oscilação se dá em casas decimais altas (ordem de 10^{-3}). Dito isto, a corretude da implementação do FANNY e $CARD_f$ pode não ser 100%. Contudo, o mesmo não ocorre com o MFCMdd-RWL-S: como esperado, seu critério de adequação não sofre nenhuma oscilação em nenhum dos conjuntos de dados.

4.1 Iris

O conjunto de dados Iris (http://archive.ics.uci.edu/ml/datasets/Iris) possui apenas uma matriz que descreve os dados. Esse conjunto de dados possui 150 instâncias e 4 atributos (matriz 150 por 5): *sepal length*, *sepal width*, *petal length*, *petal width*. O número de classes é 3, *Iris-setosa*, *Iris-versicolour* e *Iris-virginica*, com 50 instâncias em cada classe.

Das 50 vezes em que cada algoritmo foi executado sobre esse conjunto de dados, escolheu-se aquele em que se obteve o menor critério de adequação ao final. No caso do MFCMdd-RWL-S, isso foi feito para cada número de protótipos, dando um total de 250 execuções desse algoritmo; ainda sim, escolheu-se o melhor (i.e. menor critério de adequação) para cada instância do MFCMdd-RWL-S. Os índices encontram-se na tabela abaixo (tabela 1) bem como os pesos de relevância para cada matriz no caso do $CARD_f$ e do melhor de todas as instâncias do MFCMdd-RWL-S (tabela 2).

Nota-se que o MFCM-10 obteve a melhor performance, não só entre as outras instâncias do MFCM, como também em relação ao FANNY e o $CARD_f$. O MFCM-2 teve um desempenho próximo ao do MFCM-10 para índices de *hard cluster* e um desempenho superior na sua partição *fuzzy* sendo também superior ao FANNY e o $CARD_f$.

O critério de adequação do $CARD_f$ oscilou nesse conjunto de dados; como dito anteriormente, isto pode indicar uma incorretude na implementação desse algortimo visto que a oscilação no critério de adequação não ocorreu por uma imprecisão de casas decimais.

Tabela 1 – índices para o conjunto de dados Iris.

	FMEASURE	OERC	CR	RAND	JACCARD	FM	HUBERT
FANNY	0.8267	0.1733	0.6100	0.2991	0.2375	0.3838	0.6554
$CARD_f$	0.8533	0.2800	0.5182	0.2990	0.2372	0.3835	0.6545
MFCM-1	0.9200	0.0800	0.7811	0.3417	0.3447	0.5128	1.0379
MFCM-2	0.9067	0.0933	0.7562	**0.3506**	**0.3696**	**0.5397**	**1.1177**
MFCM-3	0.9000	0.1000	0.7420	0.3481	0.3623	0.5320	1.0948
MFCM-5	0.9000	0.1000	0.7420	0.3491	0.3652	0.5351	1.1039
MFCM-10	**0.9400**	**0.0600**	**0.8340**	0.3475	0.3608	0.5303	1.0899

Tabela 2 – pesos de relevância para o conjunto de dados Iris.

	$CARD_f$			MFCM-10		
	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 1**	**Cluster 2**	**Cluster 3**
Sepal length	0.1888	**0.3124**	0.1948	0.1803	0.1279	0.1630
Sepal width	0.2346	**0.3984**	0.2115	**0.3620**	0.1239	**0.4411**
Petal length	0.1913	0.1918	0.1751	0.1405	0.2047	0.1348
Petal width	**0.3652**	0.0975	**0.4168**	**0.3172**	**0.5434**	0.2612

4.2 Wine

O conjunto de dados Wine (https://archive.ics.uci.edu/ml/datasets/Wine) possui apenas uma matriz que descreve os dados. Esse conjunto de dados possui 178 instâncias

e 13 atributos (matriz 178 por 14): *alcohol*, *malic acid*, *ash*, *alcalinity of ash*, *magnesium*, *total phenols*, *flavanoids*, *nonflavanoids phenols*, *proanthocyanins*, *color intensity*, *hue*, *OD280/OD315 of diluted wines* e *proline*; com 59 instâncias na classe 1, 71 instâncias na classe 2 e 48 instâncias na classe 3.

Das 50 vezes em que cada algoritmo foi executado sobre esse conjunto de dados, escolheu-se aquele em que se obteve o menor critério de adequação ao final. No caso do MFCMdd-RWL-S, isso foi feito para cada número de protótipos, dando um total de 250 execuções desse algoritmo; ainda sim, escolheu-se o melhor (i.e. menor critério de adequação) para cada instância do MFCMdd-RWL-S. Os índices encontram-se na tabela abaixo (tabela 3) bem como os pesos de relevância para cada matriz no caso do $CARD_f$ e do melhor de todas as instâncias do MFCMdd-RWL-S (tabela 4).

O MFCM-2 obteve o melhor *hard cluster* enquanto que o FANNY obeteve o melhor *fuzzy cluster*. Era esperado que o MFCM-2 se saísse melhor que o FANNY, dadas suas vantagens em relação ao último; contudo, parece que o próprio conjunto de dados não ressaltou as qualidades do MFCM-2, tais como pesos de relevância, e por isso o FANNY tenha obtido uma patição fuzzy melhor. Apesar disso, nota-se que a qualidade da partição fuzzy do MFCM-2 não está muito atrás em relação ao FANNY.

É importante ressaltar que, uma vez que o índice CR do FANNY foi negativo, então isso significa que as concordâncias de classificação aconteceram ao acaso, o que torna o MFCM-2 definitivamente superior ao FANNY.

Tabela 3 – índices para o conjunto de dados Wine.

	FMEASURE	OERC	CR	RAND	JACCARD	FM	HUBERT
FANNY	**0.9157**	0.5899	-0.0007	0.2437	**0.2567**	**0.4206**	**0.7540**
$CARD_f$	0.7022	0.5562	0.0244	0.2531	0.2498	0.4077	0.7068
MFCM-1	0.6404	0.3596	0.2510	0.2843	0.2129	0.3510	0.5537
MFCM-2	0.6685	**0.3315**	**0.3097**	**0.2886**	0.2216	0.3627	0.5892
MFCM-3	0.5955	0.4045	0.2777	0.2884	0.2211	0.3622	0.5874
MFCM-5	0.6742	0.4101	0.2649	0.2861	0.2160	0.3553	0.5669
MFCM-10	0.5955	0.7495	0.0925	0.4105	0.0522	0.0993	0.1103

Tabela 4 – pesos de relevância para o conjunto de dados Wine.

	$CARD_f$			**MFCM-2**		
	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 1**	**Cluster 2**	**Cluster 3**
Alcohol	0.0000	0.0000	0.0000	0.0508	0.0370	0.0396
Malic acid	**0.2026**	**0.1290**	**0.6453**	0.0274	0.0474	0.0470
Ash	0.0000	0.0000	0.0000	**0.1874**	**0.1389**	0.0962
Alcalinity of ash	0.0000	0.0000	0.0000	0.0121	0.0089	0.0113
Magnesium	**0.1058**	**0.7779**	0.0543	0.0024	0.0023	0.0023
Total phenols	0.0000	0.0000	0.0000	0.0611	0.0634	0.0655
Flavanoids	0.0000	0.0000	0.0000	0.0453	0.0408	0.0434
Nonflavanoids phenols	**0.6379**	0.0654	0.0230	**0.3104**	**0.3535**	**0.3634**
Proanthocyanins	0.0000	0.0000	0.0000	0.0660	0.0701	0.0698

Color intensity	0.0000	0.0000	0.0000	0.0086	0.0178	0.0193
Hue	0.0537	0.0277	**0.2774**	**0.1682**	**0.1770**	**0.1857**
OD280/OD315 of diluted wines	0.0000	0.0000	0.0000	0.0601	0.0530	0.0566
Proline	0.0000	0.0000	0.0000	0.0002	0.0001	0.0001

4.3. Multiple Features (mfeat)

O conjunto de dados mfeat (https://archive.ics.uci.edu/ml/datasets/Multiple+Features) possui 6 matrizes que descrevem os dados. Esse conjunto de dados consiste de características de numerais manuscritos (‘0’ – ‘9’) extraído de uma coleção de mapas utlitários holandêses. Esse conjunto possui 2000 instâncias; na matriz “mfeat-fou” há 76 atributos que descrevem os coeficientes de Fourier dos formatos dos caracteres; na matriz “mfeat-fac” há 216 correlações de contorno; na “mfeat-kar” há 64 coeficientes de Karhunen-Love; na “mfeat-pix” há 240 médias de pixel em janelas 2 x 3; na “mfeat-zer” há 47 momentos Zernike; por fim, na “mfear-mor” há 6 características morfológicas. O conjunto possui 200 instâncias em cada classe, dando um total de 10 classes.

Das 50 vezes em que cada algoritmo foi executado sobre esse conjunto de dados, escolheu-se aquele em que se obteve o menor critério de adequação ao final. No caso do MFCMdd-RWL-S, isso foi feito para cada número de protótipos, dando um total de 250 execuções desse algoritmo; ainda sim, escolheu-se o melhor (i.e. menor critério de adequação) para cada instância do MFCMdd-RWL-S. Os índices encontram-se na tabela abaixo (tabela 5) bem como os pesos de relevância para cada matriz no caso do $CARD_f$ e do melhor de todas as instâncias do MFCMdd-RWL-S (tabela 6).

Tanto o FANNY quanto o $CARD_f$ oscilaram bastante nesse conjunto de dados; como dito anteriormente, isto pode indicar uma incorretude na implementação desses algortimos visto que a oscilação no critério de adequação não ocorreu por uma imprecisão de casas decimais. Outro ponto que reforça a possível incorretude da implementação do $CARD_f$ é o fato do aparecimento de pesos de relevância que violam as limitações (8) e (9), isso ocorre no cluster 10 (C10).

Por fim, nota-se que o MFCM teve um performance superior ao FANNY e $CARD_f$, sendo o MFCM-1 o melhor dos MFCMs.

Tabela 5 – índices para o conjunto de dados mfeat.

	FMEASURE	OERC	CR	RAND	JACCARD	FM	HUBERT
FANNY	0.9175	0.8020	0.1341	0.4105	0.0523	0.0993	0.1103
$CARD_f$	**1.0000**	0.9000	0.0000	0.3398	0.0794	**0.1671**	**0.2092**
MFCM-1	0.5570	**0.5850**	0.2397	**0.4106**	**0.0523**	0.0995	0.1107
MFCM-2	0.7270	0.6760	**0.2457**	0.4105	**0.0523**	0.0994	0.1104
MFCM-3	0.7500	0.7745	0.1150	0.4105	**0.0523**	0.0993	0.1103
MFCM-5	0.7075	0.7850	0.0918	0.4105	**0.0523**	0.0993	0.1103

MFCM-10	0.4945	0.7495	0.0925	0.4105	**0.0523**	0.0993	0.1103

Tabela 6.1 – pesos de relevância para o conjunto de dados mfeat.

	$CARD_f$				**MFCM-1**			
	C1	**C2**	**C3**	**C4**	**C1**	**C2**	**C3**	**C4**
Mfeat-fac	0.1667	0.1667	0.1667	0.1667	0.0005	0.0005	0.0006	0.0006
Mfeat-fou	0.1667	0.1667	0.1667	0.1667	**0.9600**	**0.9600**	**0.9594**	**0.9594**
Mfeat-kar	0.1667	0.1667	0.1667	0.1667	0.0254	0.0254	0.0263	0.0263
Mfeat-mor	0.1667	0.1667	0.1667	0.1667	0.0002	0.0002	0.0002	0.0002
Mfeat-pix	0.1667	0.1667	0.1667	0.1667	0.0129	0.0129	0.0129	0.0129
Mfeat-zer	0.1667	0.1667	0.1667	0.1667	0.0009	0.0009	0.0009	0.0009

Tabela 6.2 – pesos de relevância para o conjunto de dados mfeat.

	$CARD_f$				**MFCM-1**			
	C5	**C6**	**C7**	**C8**	**C5**	**C6**	**C7**	**C8**
Mfeat-fac	0.1667	0.1667	0.0000	**0.9331**	0.0005	0.0006	0.0006	0.0005
Mfeat-fou	0.1667	0.1667	0.0000	0.0000	**0.9600**	**0.9594**	**0.9594**	**0.9600**
Mfeat-kar	0.1667	0.1667	0.0000	0.0000	0.0254	0.0263	0.0263	0.0254
Mfeat-mor	0.1667	0.1667	**1.0000**	0.0000	0.0002	0.0002	0.0002	0.0002
Mfeat-pix	0.1667	0.1667	0.0000	0.0662	0.0129	0.0129	0.0129	0.0129
Mfeat-zer	0.1667	0.1667	0.0000	0.0047	0.0009	0.0009	0.0009	0.0009

Tabela 6.3 – pesos de relevância para o conjunto de dados mfeat.

	$CARD_f$		**MFCM-1**	
	C9	**C10**	**C9**	**C10**
Mfeat-fac	**1.0000**	0.0006	0.0006	0.0006
Mfeat-fou	0.0000	0.0000	**0.9594**	**0.9594**
Mfeat-kar	0.0000	1.0095	0.0263	0.0263
Mfeat-mor	0.0000	0.0015	0.0002	0.0002
Mfeat-pix	0.0000	-0.0123	0.0129	0.0129
Mfeat-zer	0.0000	0.0006	0.0009	0.0009

5. Conclusões

Este trabalho de graduação apresentou conceitos de clustering e clustering colaborativo, abordando alguns algoritmos mais a fundo tais como o FANNY [12], $CARD_f$ [1] e o MFCMdd-RWL-S [2]; além de índices que avaliam tanto partições *hard* quanto *fuzzy*.

Para analisar os algoritmos citados na prática, eles foram implementados em C, bem como as medidas estatísticas. Aplicou-se esses algoritmos em alguns conjuntos de dados retirados do *UCI Machine Learning Repository* (http://archive.ics.uci.edu/ml/), foram esses: Iris, um conjunto de dados simples mas amplamente usado no teste de modelos; Wine, um conjunto "bem comportado" também muito usado para testes de modelos, mais complexo que o Iris; Multiple Features, um conjunto de dados

relacionais, ideal para testar os algoritmos foco deste trabalho ($CARD_f$ e MFCMdd-RWL-S) visto os dados são descritos por múltiplas matrizes.

Os experimentos mostrados que o MFCMdd-RWL-S teve um desempenho superior ao $CARD_f$ e o FANNY; sendo o último com o pior desempenho, por não se tratar de um algoritmo que leva em conta multiplas matrizes de dissimilaridade (não atribui pesos de relevância às matrizes); como esperado [2].

O tema em questão é relativamente recente e há bastante conteúdo para se trabalhar além de várias aplicações [1, 2, 7]. Por exemplo, o CARD pode ser usado em sistemas de recuperação de imagens [1]; o MFCM ou CARD podem ser usados para se trabalhar com dados confidenciais de diferentes instituições sem que se perca o sigilo [3]. Como esses algoritmos de clustering colaborativo se baseiam bastante em algoritmos já existentes, mas que não consideram multiplas matrizes de dissimilaridade, pode-se também experimentar aplicar otimizações, já aplicadas com sucesso em seus "antecessores", à esses novos algoritmos, provavelmente sem muitas modificações.

Referências bibliográficas

[1] FRIGUI, H.; HWANG, C.; RHEE, F. C. H. *Clustering and aggregation of relational data with applications to image database categorization*, Pattern Recogn., Nov. 2007.

[2] CARVALHO, F. A. T.; LECHEVALLIER, Y.; MELO, F. M. *Relational partitioning fuzzy clustering algorithms based on multiple dissimilarity matrices, Fuzzy Sets and Systems*, Sets and Systems, vol. 215, pp. 1-28, 16 Mar. 2013.

[3] PEDRYCZ, W. *Collaborative fuzzy clustering*. 2008.

[4] YAMANISHI, Y.; VERT, J. P., KANEHISA, M. *Protein network inference from multiple genomic data: a supervised approach*, Bioinformatics, vol. 20, no. 1, pp. 363-370, 2004.

[5] REZA, G.; NASIR, MD.; HAMIDAH, I.; NORWATI, M. *A survey: Clustering ensembles techniques*, Proceedings of World Academy of Science, Engineering and Technology, vol. 38, pp. 644–653, 2009.

[6] VEGA-PONS, S.; RUIZ-SHULCLOPER, J. *A survey of clustering ensemble algorithms*, International Journal of Pattern Recognition and Artificial Intelligence, vol. 20, no. 3, pp 337-372, 2011.

[7] CLEZIOU, G.; EXBRAYAT M.; MARTIN, L.; SUBLEMONTIER, J. *CoFKM: A centralized method for multiple-view clustering*, Ninth IEEE International Conference, ICDM'09, IEEE, pp. 752-757, Dez. 2009.

[8] BEZDEK, J. C.; *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, Nova York, 1981.

[9] BICKEL, S.; SCHEFFER, T.; *Estimation of mixture models using co-em*, European Conference on Machine Learning (ECML), pp. 35-4, 2005.

[10] KRISHNAPURAM, R.; JOSHI, A.; YI, L.; *A fuzzy relative of the k-medoids algorithm with application to web document and snippet clustering*, Proceedings of the IEEE International Fuzzy Systems Conference, pp. 1281-1286, 1999.

[11] DAVENPORT, J. W.; HATHAWAY, R. J.; BEZDEK, J. C.; *Relational duals of the c-means algorithms*, Pattern recognition, vol. 22, pp. 205-212, 1989.

[12] KAUFMAN, L.; ROUSSEEW, P.J.; *Finding groups in data*, Wiley, Nova York, 1990.

[13] CARVALHO, F. A. T.; et al; *Pattern Recognition*, vol. 45, pp. 447-464, 2012.