Graph Structure Fusion for Multiview Clustering

Kun Zhan , Chaoxi Niu, Changlu Chen, Feiping Nie , Changqing Zhang , and Yi Yang

摘要

现有的许多多试图聚类方法将图作为输入来揭示数据的分布规律，这些图通常在各个视图中经过了预处理。这些方法忽略了多个视图之间的图结构的联系，而且聚类结果高度依赖于预定义的亲和矩阵的质量。论文通过集成不同视图的图形结果以充分利用基础数据结果的几何属性来解决多视图聚类的问题。所提出的方法基于以下假设：内在的图结构会将每个图对应的连通分量分配到同一个聚类中。来自不同视图的多个图将通过 Hadamard 积集成在一起，这是因为不同视图之间通常有着相同的底层图结构。具体来说，来自不同视图的图将会被聚合成一个全局的图，然后这个图会根据一个精心设计的目标函数进行自适应调整，最终使得这个全局的图的连通分量的个数恰好等于聚类的个数。另外，值得注意的是，算法直接从图本身得到了聚类指标，不需要执行进一步的图切割或者 k-means 聚类操作。实验表明，所提出的算法的性能比那些最好的聚类算法还要好。

关键词：非监督式学习，多视图聚类，特征提取，多视图学习，结构化图。

1 引言

由于实例通常与多种类型的特征相关联，因此多视图数据在许多领域都很普遍。例如，可以通过面部、指纹、虹膜和签名来识别一个人；一条新闻可以由不同语言的多篇文章报道；图像可以用不同的特征来描述如 SIFT、GIST 和 HOG。随着多视图数据的日益普及，利用多视图数据获得比使用任何单视图数据更准确的聚类结果非常重要的。

什么是多视图的聚类呢？先考虑三维物体的三视图，如果知道了一个三维物体的三视图，我们可以通过三视图来将该三维物体还原出来。数据也是有“三视图”的，通过对数据的不同视图进行观察和考虑能够更好的刻画出数据的性质。多视图的聚类工作就是利用数据的“三视图”来对一组数据进行分类

2 相关工作

通常，多视图学习方法分为三类：协同学习、子空间学习和多内核学习。许多多视图学习方法被用于最大化不同视图之间的相同之处。

最初，协同学习模型被用于监督式多视图学习。受监督式协同训练学习的启发，两个非监督式多视图学习方法被提出了：多视图的谱聚类和共正则化的多视图谱聚类。之后，为了利用辅助信息，一些方法还使用了非监督式协同学习策略。这些方法在所有视图中搜索共同之处，并且最小化不同视图之间的不同之处。

这些多视图协同学习算法同样用了子空间学习去学习出一个潜在的用于数据表示的子空间。类似的，非监督式子空间学习被应用在多视图聚类上来得到一个由不同特征组成的潜在子空间。通常，这些方法需要在学习到的图上进行谱聚类这一后处理过程来得到数据的标签。

除了协同训练学习和子空间学习，多内核学习方法通常用来处理多视图的特征。在多内核学习中，一系列内核在不同的视图中被构造出来，用于将这些特征融合起来。

在这些学习到的图中，大多数方法都是需要进行图切割操作或者是 k-means 聚类来获得聚类指标的。由于谱聚类算法中已经提出了一个精心设计的数学框架，这些算法通常都是利用谱聚类来设计的，或者是在学习到的子空间中进行谱聚类的操作。这些方法通常分为两步：先构造图或者内核，然后通过学习算法来得到分类标签。这种两步走的方法有一个缺点：最终的聚类结果不能显式的的有学习到的图来表示，所以这些方法必须要再进行一次 k-means 操作来获得聚类指标。

最近，出现了一些新的聚类算法，它们是通过图的拉普拉斯矩阵的秩来学习一个自适应图的，而且许多基于图的方法使用权重和准则来将单视图学习扩展到多视图学习中。除了用权重和准则外，论文也利用了图本身的结构，而且通过使用 Hadamard 积将多视图的信息聚合在一起

3 本文方法

在本文中作者提出了一个叫做 GSF 的算法。作者等人想要学习出一个内在图结构来得到聚类结果，同时又能充分的利用每个视图的几何属性。但是，对于现实世界的数据来说是很难得到一个这样的内在图，这个内在图恰好有 n_c 个连通分量，也就是 n_c 个分类。所以作者等人设计了一个目标函数，通过最小化这个目标函数来学习出这个图。由于每个连通分量的顶点都属于一个分类，所以聚类指标可以直接从学习出来的图本身获得，不需要再做图切割和 k-means 聚类方法。所以说，本文最主要的工作就是用 GSF 算法来学习出一个内在图。先来说明本文的符号表示。数据矩阵用 $X = [X_1, X_2, ..., X_{nc}] = [x_1, x_2, ..., x_n]$ 来表示，其中 X_c 表示该数据集属于第 c 个分类，x_i 表示数据点，n 是数据点的个数，d 是维度，n_c 是分类的个数。n_v 是视图的个数

在具体介绍本文方法之前，先介绍定理 1：图的连通分量的个数等于其对应拉普拉斯矩阵特征值 0 的重数。

3.1 内在图学习

给定一个数据集，先对各个视图用 $k-NN$ 算法求出其亲和矩阵。亲和矩阵是一个方阵，行和列的值都是数据集图片数量。然后，将每个视图的亲和矩阵用 Hadamard 积融合在一起得到矩阵 A（多个图融合后的图），即：

$$A = \prod_{v=1}^{n_v} W^{(v)} \tag{1}$$

我们将根据这个 A 和上述的定理来学习出一个相似矩阵 S（对应学习出来的内在图）。由于图的拉普拉斯矩阵 L 是半正定的，所以 L 的特征值非负。所以，可以知道当满足条件 $\sum_{c=1}^{n_c} \lambda_c = 0$ 此时 S 已经有了理想状态的邻接节点而且数据点已经被分成 n_c 类了。根据范氏定理，我们可得目标函数如下

$$\sum_{c=1}^{n_c} \lambda_c = \min_{U} < UU^T, L > \tag{2}$$

$s.t. U \in R^{n \times n_c}, U^T U = I$。

$< \cdot, \cdot >$ 表示矩阵的 Frobenius 内积，$Tr(\cdot)$ 表示矩阵的迹，$U^T = [u_1, u_2, \cdots, u_n]$，$L = D - \dfrac{S^T + S}{2}$ 为拉普拉斯矩阵，I 是单位阵，D 是对角阵而且对角元素为矩阵 $\dfrac{S^T + S}{2}$ 每一列的和

定理 1 表明当目标函数值为 0 时，S 的连通分量的个数就是分类的个数，而且每个连通分量对应一个分类。所以我们可以通过一个全局的目标函数来自适应的调节图结构来学习出这么一个图 S。

由于图 A 包含了内在图的边，所以我们需要让 S 尽可能的靠近 A，所以我们优化下面这个目标函数：

$$\max_{S} < A, S > \tag{3}$$

$s.t. \forall j, s_j \geqslant 0, \mathbf{1}^T s_j = 1$。限制 $\mathbf{1}^T s_j = 1$ 是为了让 $D = I$

联立 (2)、(3) 式，得：

$$\min_{U,S} < UU^T > -\gamma_1 < A, S > \tag{4}$$

$s.t. U \in R^{n \times n_c}, U^T U = I$，$\forall j, s_j \geqslant 0, \mathbf{1}^T s_j = 1$。其中 γ_1 是权重因子。

由于约束条件 $1^T s_j = 1$，L 是一个规范化的拉普拉斯矩阵，也就是说 $L = I - S$。所以可得：

$$< UU^T, L > -\gamma_1 < A, S > = < UU^T, I > - < UU^T + \gamma_1 A, S > \tag{5}$$

(5) 式的第一项 $< UU^T, I >$ 在优化 S 的时候视为常数。所以关于 S 解 (4) 式相当于解下面的优化函数：

$$\max_{S} < G, S > \tag{6}$$

$s.t. \forall j, s_j \geqslant 0, \mathbf{1}^T s_j = 1$。其中 G 表示矩阵 $[UU^T + \gamma_1 A]$

显然 $< G, S > = \sum s_j^T g_j$，$g_j$ 表示 G 的第 j 列，所以，关于 s_j 解 (6) 式就相当于优化下面的函数：

$$\max_{s_j} s_j^T g_j \tag{7}$$

$s.t. s_j \geqslant 0, \mathbf{1}^T s_j = 1$

(7) 式关于 s_j 有一个平凡解，也就是 s_j 中只有一个值为 1，其它全为 0。(7) 式的目标函数值将返回 $g_{ij} = max(g_j)$，也就是 s_j 的第 i 个元素是 1，其它元素全为 0。为了避免这一情况，加入正则化方法，得到下面的目标函数：

$$\min_{U,S} Tr(U^T L U) - \gamma_1 Tr(AS^T) + \gamma_2 ||S||_F^2 \tag{8}$$

$s.t. U \in R^{n \times n_c}, U^T U = I$，$\forall j, s_j \geqslant 0, \mathbf{1}^T s_j = 1$。其中 γ_1 和 γ_2 是权重因子

(8) 式可分成两个子问题来求解。第一个子问题是固定 U，更新 S，所以 (8) 式变成：

$$\min_{S} Tr(U^T L U) - \gamma_1 Tr(AS^T) + \gamma_2 ||S||_F^2 \tag{9}$$

$s.t. \forall j, s_j \geqslant 0, \mathbf{1}^T s_j = 1$。

注意到 (9) 式在不同的 j 中是独立的，所以可得：

$$\min_{s_j} \sum (||u_i - u_j||_2^2 - \gamma_1 a_{ij}) s_{ij} + \gamma_2 s_j^T s_j \tag{10}$$

$s.t. s_j \geqslant 0, \mathbf{1}^T s_j = 1$。

记 p_j 是一个向量，其中第 i 个元素等于 $p_{ij} = ||u_i - u_j||_2^2 - \gamma_1 a_{ij}$。所以优化 (10) 式相当于优化下面的目标函数：

$$\min_{s_j} \frac{1}{2} ||s_j + \frac{p_j}{2\gamma 2}||_2^2 \tag{11}$$

$s.t. s_j \geqslant 0, \mathbf{1}^T s_j = 1$。

(11) 式是一个在 simplex 空间的欧几里得投影问题，凸优化问题。因此我们列出拉格朗日函数

$$L(s_j, \eta, \rho) = \frac{1}{2}||s_j + \frac{p_j}{2\gamma 2}||_2^2 - \eta(\mathbf{1}^T s_j - 1) - \rho^T s_j \tag{12}$$

这是一个凸优化问题，用 KKT 条件解它，得

$$s_j^* = (-\frac{p_j}{2\gamma 2} + \eta \mathbf{1})_+ \tag{13}$$

这就是矩阵 S 的每一列，GSF 算法将根据这个式子来迭代学习出 S

第二个子问题是固定 S，更新 U，所以 (8) 式变成：

$$\min_U Tr(U^T L U) \tag{14}$$

$s.t. U \in R^{n \times n_c}, U^T U = I$。

最优的 U 由 S 的拉普拉斯矩阵前 n_c 个最小的特征值的特征向量排列而成。(13),(14) 式就是 GSF 算法的核心，上述内容的细节算法总结在 GSF 算法中，如下所示

Procedure 1 GSF-based multiview clustering.

Input: Dataset $\chi = X^{(1)}, X^{(2)}, \cdots, X^{(n_v)}$, n_c, parameters γ_1, γ_2
Output: S with exactly n_c connected components
initialize: $W^{(v)}, \forall v \in$1,2, ••• ,n$_v$ is constructed from the data matrix $X^{(v)}$ by a $k - NN$ graph algorithm, A is calculated by Eq.(1), and U is formed by n_c number of eigenvectors corresponding to the top n_c smallest eigenvalues of the Laplacian matrix of A.
for *S doesn't have n_c connected components* **do**
  for *$j \in$1,2, ••• ,n* **do**
    update s_j by using Eq.(13);
  end
  $S = \frac{S + S^T}{2}$;
  Update U by Eq.(14), i.e., U is formed by n_c eigenvectors with the top n_c smallest eigenvalues of L
end

3.2 初始图学习

GSF 算法的迭代需要一个初始的 S，我们根据 (11) 式，得到下面这个目标函数，根据这个目标函数来得到一个初始的 S。

$$\min_{s_j} w_j^T b_j + \gamma_j w_j^T w_j \tag{15}$$

$s.t. s_j \geqslant 0, \mathbf{1}^T s_j = 1$。$\gamma_j$ 是控制数据点 x_j 邻接节点分配的权重因子。同样的 (15) 式可以分成两种情况来求解

第一种情况是

$$\min_{s_j} b_j^T w_j \tag{16}$$

$s.t. s_j \geqslant 0, \mathbf{1}^T s_j = 1$。

(16) 式的解就是让一个数据点 x_j 仅仅与另一个数据点 x_i 连接，权重边 $w_{ij}^* = 1$。

第二种情况是

$$\min_{w_j} w_j^T w_j \tag{17}$$

$s.t. s_j \geqslant 0, \mathbf{1}^T s_j = 1$。

(17) 式的解就是 x_j 与所有的数据点都连接，权重边 $w_{ij}^* = \frac{1}{n}, \forall i$

初始的 S 应该是这两种情况折中。与 (13) 式一样，(15) 式的解为

$$w_j^* = (-\frac{\mathbf{b}_j}{2\gamma_j} + \eta\mathbf{1}) \tag{18}$$

不失一般性，设 $b_{1j}, b_{2j}, \cdots, b_{nj}$ 由小到大排序。当给 x_j 分配 k 个邻居时，最优的 w_j 有 k 个非 0 元素，所以根据 (18) 式可得 $w_{k,j} > 0$ 且 $w_{k+1,j} = 0$。又因为 $\mathbf{1}^T s_j = 1$，可得

$$\sum_{i=1}^{k}(-\frac{b_{ij}}{2\gamma_j} + \eta) = 1 \tag{19}$$

所以

$$\eta = \frac{2\gamma_j + \sum_{i=1}^{k} b_{ij}}{2k\gamma_j} \tag{20}$$

联立 (18)，(20) 式，且考虑 $w_{k,j} > 0$ 且 $w_{k+1,j} = 0$。可得

$$\begin{cases} \gamma_j > \frac{k}{2}b_{k,j} - \frac{1}{2}\sum_{i=1}^{k} b_{ij} \\ \gamma_j \leqslant \frac{k}{2}b_{k+1,j} - \frac{1}{2}\sum_{i=1}^{k} b_{ij} \end{cases} \tag{21}$$

为了满足 (21) 式，我们令

$$\gamma_j = \frac{k}{2}b_{k+1,j} - \frac{1}{2}\sum_{i=1}^{k} b_{ij} \tag{22}$$

联立 (19)、(20) 和 (22) 式，可得

$$w_{ij}^* = \begin{cases} \frac{b_{k+1,j} - b_{ij}}{kb_{k+1,j} - \sum_{i=1}^{k} b_{ij}} & i \leqslant k \\ 0 & otherwise. \end{cases} \tag{23}$$

将根据 (23) 式来构造初始的 S，将这个构造出来的 S 放进 GSF 算法中进行迭代。

4 复现细节

4.1 与已有开源代码对比

在解决论文 (11) 式时使用开源代码https://github.com/kunzhan/GSF/blob/main/EProjSimplex_new.m

在求聚类指标的时候使用了下面开源代码的匈牙利方法部分 https://github.com/kunzhan/GSF/blob/main/ClusteringMeasure.m

在实现 GSF 算法的代码时结合自己对论文的理解并且参考了开源代码 https://github.com/kunzhan/GSF 参考部分如下两图：

```
W = zeros(n);
for i = 1:n
    id = idx(i,2:k+2);
    di = D(i, id);
    W(i,id) = (di(k+1)-di)/(k*di(k+1)-sum(di(1:k))+eps);
end;
```

图 1: 初始迭代矩阵构造

```
for iter = 1:NITER
    %update S
    S = zeros(num);
    distu = L2_distance_1(U',U');
    for i=1:num
        if islocal ==1
            idxa0 = idx(i,2:k + 1);
        else
            idxa0 = 1:num;
        end
        dui = distu(i,idxa0);
        dus = Sa(i,idxa0);
        ad = -(dui - gamma1*dus)/(2*gamma2);
        S(i,idxa0) = EProjSimplex_new(ad);
    end

    S = (S + S')/2;
    D = diag(sum(S));
    L = D - S;
    U_old = U;
    % Update U
    [U, ~, ev]=eig1(L, c, 0);
    thre = 1*10^-5;
    fn1 = sum(ev(1:c));
    fn2 = sum(ev(1:c+1));
    ob = trace(U'*L*U) - gamma1*trace(S*Sa) + gamma2*trace(S*S') ;
    obj = [obj ob];
    if fn1 > thre
        gamma2 = gamma2/2;
    elseif fn2 < thre
        gamma2 = gamma2*2;  U = U_old;
    else
        break;
    end
```

图 2: GSF 迭代

4.2 实验环境

MATLAB_R2021b。数据集：C101_p1474.mat、COIL_20_ZCQ.mat、ORL_mtv.mat

4.3 代码实现细节

main.m 文件是 GSF 算法代码的入口，实现的功能是导入数据集，执行 constructS.m 和 GSF.m 并输出聚类指标。GSF.m 实现的功能主要是 GSF 算法，先用 hadamard 积融合每个视图的亲和矩阵，再用 constructS 生成的初始相似矩阵进行 GSF 算法的迭代，并且得到分类结果。getEig.m 实现的功能是求矩阵的特征向量和特征值（排序后）。Euclidean_distance.m 实现的功能是求数据点之间的欧几里得距离。metric_compute.m 和 AdjustedRandIndex.m 实现的功能是求出 7 个聚类指标，并且使用了指派问题的匈牙利算法。constructS.m 实现的功能是构造 GSF 算法迭代所需的初始相似矩阵。

所有的代码都已封装完毕并且添加了详细的中文注释

4.4 创新点

改进了计算两个数据点之间的距离的方法以及代码结构，提高了代码的运行速度。

5 实验结果分析

论文用了三个数据集，分别如下：Caltech-101: 该数据集有 8677 张图像，但论文选了其中 1474 张图像，共有 6 个视图，7 个分类。ORL：该数据集有 400 张图像，3 个视图，40 个分类。COIL-20: 该数据集有 1440 张图像，3 个视图，20 个分类。下图总结了这个 3 个数据集：

	Caltech-101	ORL	COIL-20
View 1	Gabor (48)	Intensity (4096)	Intensity (1024)
View 2	Wavelet (40)	LBP (3304)	LBP (3304)
View 3	CENTRIST (254)	Gabor (6750)	Gabor (6750)
View 4	HOG (1984)	-	-
View 5	GIST (512)	-	-
View 6	LBP (928)	-	-
n	1474	400	1440
n_c	7	40	20

图 3: 数据集属性

论文使用 7 个聚类指标来评判算法的优劣，分别是聚类准确度 (ACC)、归一化互信息 (NMI)、纯度 (Purity)、准度 (Precision)、召回率 (Recall)、F 值 (F-score) 和调整后的兰德指数 (ARI)。这些聚类指标的值越大，聚类效果越好。本文将 GSF 与其他七个目前最好的聚类算法在同样的数据集上运行并且比较了运行结果。七个聚类算法分别是：谱聚类、CRSC、RMKMC、RMSC、MVSC、MKKM、MVGL。作者的实验结果如下图所示。

Methods	ACC	NMI	Purity	Precision	Recall	F-score	ARI
Caltech-101							
SC 1	28.15±0.34	15.54±0.14	63.21±0.08	51.73±0.21	20.44±0.17	29.30±0.21	9.55±0.19
SC 2	35.15±0.02	23.71±0.01	73.75±0.02	65.80±0.01	25.44±0.01	36.70±0.01	19.36±0.01
SC 3	38.25±0.04	27.07±0.03	79.44±0.02	67.56±0.04	26.30±0.02	37.86±0.03	20.74±0.03
SC 4	40.23±0.00	40.61±0.00	83.65±0.00	80.85±0.00	31.21±0.00	45.04±0.00	30.01±0.00
SC 5	40.49±0.07	35.20±0.10	81.58±0.05	76.38±0.10	28.89±0.05	41.93±0.06	26.34±0.08
SC 6	48.58±0.00	34.89±0.00	79.78±0.00	77.90±0.00	32.79±0.00	46.16±0.00	30.22±0.00
CRSC	44.54±0.30	36.74±0.26	79.07±0.30	77.89±0.35	31.71±0.21	45.07±0.27	29.30±0.32
RMKMC	52.95±4.95	41.58±4.76	83.74±1.41	83.38±5.72	39.22±3.99	53.33±4.83	38.12±5.96
MVSC	45.53±10.64	31.15±9.57	77.24±3.87	70.09±11.03	35.05±8.20	46.65±9.52	28.53±11.83
RMSC	45.05±0.00	40.33±0.18	82.94±0.15	83.01±0.20	33.43±0.08	47.67±0.11	32.79±0.15
MKKM	41.45±0.00	39.19±0.00	82.97±0.00	80.91±0.00	32.36±0.00	46.23±0.00	31.04±0.00
MVGL	57.06±0.00	53.17±0.00	**87.04±0.00**	87.25±0.00	46.15±0.00	60.37±0.00	45.96±0.00
GSF	**81.61 ± 0.00**	**63.47 ± 0.00**	84.46 ± 0.00	**91.97 ± 0.00**	**78.74 ± 0.00**	**84.84 ± 0.00**	**76.48 ± 0.00**
ORL							
SC 1	55.00±1.32	74.37±0.67	60.17±1.35	36.55±1.73	50.04±1.56	42.21±1.15	40.66±1.20
SC 2	69.18±1.75	84.22±1.22	73.87±1.95	53.70±2.70	67.31±2.97	59.73±2.70	58.69±2.78
SC 3	54.65±1.54	73.06±0.72	60.98±1.04	33.59±1.79	48.42±1.32	39.64±1.37	37.98±1.43
CRSC	77.88±2.09	88.66±1.15	80.93±2.08	66.74±2.48	75.34±2.39	70.78±2.35	70.06±2.41
RMKMC	58.98±3.40	76.87±1.75	62.87±3.23	41.25±3.60	55.13±3.56	47.14±3.24	45.73±3.34
MVSC	76.52±3.28	87.69±1.43	79.18±2.54	62.95±4.72	76.47±2.27	69.01±3.64	68.23±3.75
RMSC	80.55±2.14	89.95±1.02	82.93±1.64	71.50±2.82	77.58±2.17	74.40±2.38	73.79±2.44
MKKM	82.25±0.00	91.61±0.00	84.50±0.00	**74.75±0.00**	79.61±0.00	77.11±0.00	76.56±0.00
MVGL	76.50±0.00	87.11±0.00	81.50±0.00	41.87±0.00	81.39±0.00	55.29±0.00	53.92±0.00
GSF	**84.25±0.00**	**92.21±0.00**	**86.50±0.00**	73.73±0.00	**84.22±0.00**	**78.63±0.00**	**78.10±0.00**
COIL-20							
SC 1	63.38±0.88	75.84±0.61	66.05±0.84	57.95±0.94	60.50±1.05	59.20±0.96	57.03±1.01
SC 2	74.13±0.54	82.68±0.41	76.54±0.45	68.14±0.79	72.07±0.69	70.05±0.73	68.45±0.77
SC 3	69.59±0.15	79.55±0.19	70.96±0.15	65.49±0.20	69.21±0.20	67.30±0.20	65.55±0.21
CRSC	74.35±1.10	83.31±0.56	76.60±0.84	70.87±1.36	72.81±0.98	71.83±1.13	70.34±1.20
RMKMC	48.51±4.19	68.65±2.48	52.65±3.16	35.64±4.67	56.30±3.39	43.41±3.57	39.75±4.02
MVSC	70.35±3.33	82.09±1.78	74.17±2.85	61.92±3.73	73.63±2.40	67.23±2.83	65.37±3.02
RMSC	75.12±0.38	82.33±0.33	75.40±0.50	70.83±0.54	72.19±0.41	71.50±0.47	70.01±0.49
MKKM	77.85±0.00	84.07±0.00	78.06±0.00	73.01±0.00	74.56±0.00	73.78±0.00	72.40±0.00
MVGL	92.50±0.00	97.69±0.00	95.00±0.00	90.58±0.00	97.46±0.00	93.89±0.00	93.57±0.00
GSF	**100±0.00**	**100±0.00**	**100±0.00**	**100±0.00**	**100±0.00**	**100±0.00**	**100±0.00**

Note: The best results are highlighted in bold.

图 4: 算法聚类性能

我的实验结果图 5、6、7 所示。

```
Cluster num:7
ACC:0.815468
nmi: 0.631248
purity: 0.850746
Precision: 0.936914
Recall: 0.780800
F-score: 0.851763
ARI: 0.771799
kmeans value:0.000000
历时 7.140305 秒。
```

图 5: Caltech-101 数据集性能

```
Cluster num:40
ACC:0.842500
nmi: 0.922142
purity: 0.865000
Precision: 0.737354
Recall: 0.842222
F-score: 0.786307
ARI: 0.781040
kmeans value:0.000000
历时 0.733250 秒。
```

图 6: ORL 数据集性能

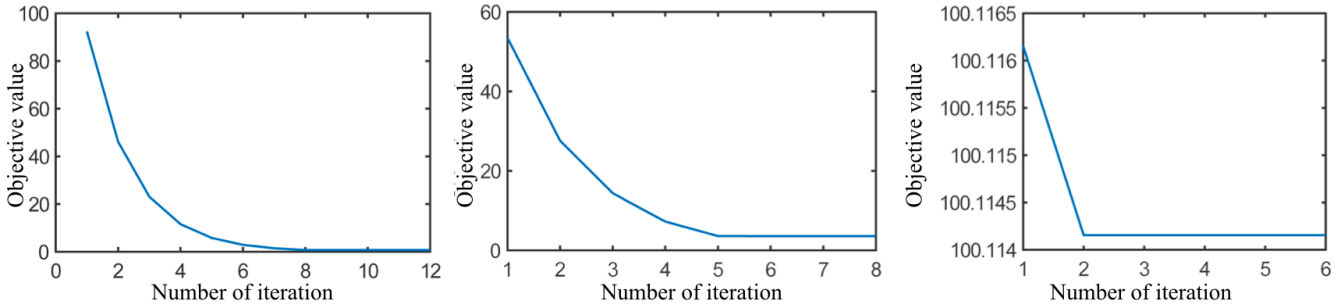
```
Cluster num:20
ACC:1.000000
nmi: 1.000000
purity: 1.000000
Precision: 1.000000
Recall: 1.000000
F-score: 1.000000
ARI: 1.000000
kmeans value:0.000000
历时 2.481361 秒。
```

图 7: COIL-20 数据集性能

可以看出，我的实验结果在 COIL-20 数据集和 ORL 数据集上是完全一样的。但在 Caltech-101 数据集上略有差别。我在多台电脑上尝试运行代码，发现结果都不尽相同。这说明，在 Caltech-101 数据集上略有差别的原因可能是因为 Matlab 版本或者是机器字长的原因。

显然，GSF 在每个数据集的效果都是最好的，并且比其他算法好得多，代码运行时间也更短。这充分说明了 GSF 算法的优越性。可以清楚地看到 GSF 几乎达到了最佳性能。并且 GSF 正确地对 COIL-20 中的所有样本进行了聚类。COIL-20 数据集是从简单的几何图形中提取的。属于同一分类的数据对象彼此非常相似，这意味着该数据集具有低秩属性。具有低秩约束的 GSF 实现了一个集成的理想全局图，并获得了比其他方法更好的结果。在 Caltech-101 中，与其他方法相比，我们在 ACC、NMI、Recall、F-score 和 ARI 的指标上有了很大的改进。量化指标证明了它的优越性，因为它更好地利用了图形表示的数据的内在结构。与基于 k-means 聚类和谱聚类的方法不同，GSF 可以获得具有更好结构的集成全局图，这就是为什么它表现出比其他方法更好的性能。

同时，作者给出了在不同数据集的 GSF 算法的迭代次数以及收敛情况，如下图所示。



(a) Caltech-101 (b) ORL (c) COIL-20

图 8: 迭代次数

我的实验结果图 9、10、11 所示。

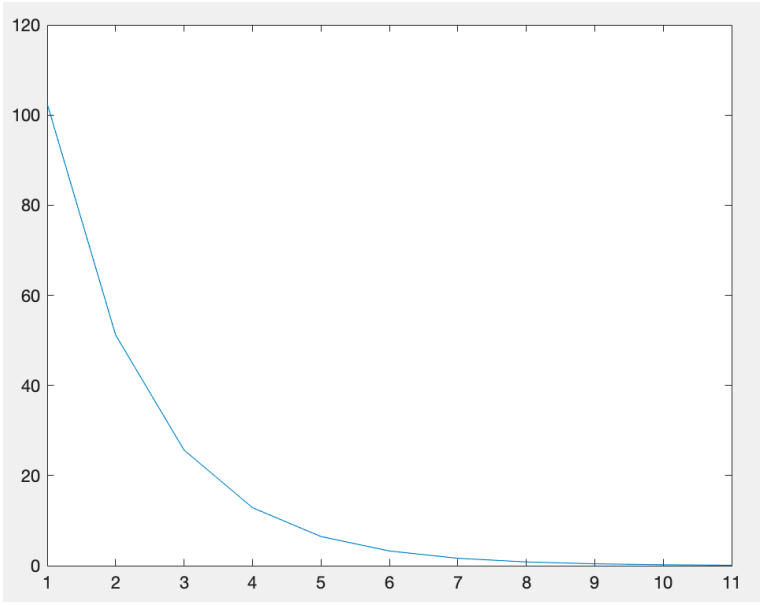


图 9: Caltech 数据集迭代次数

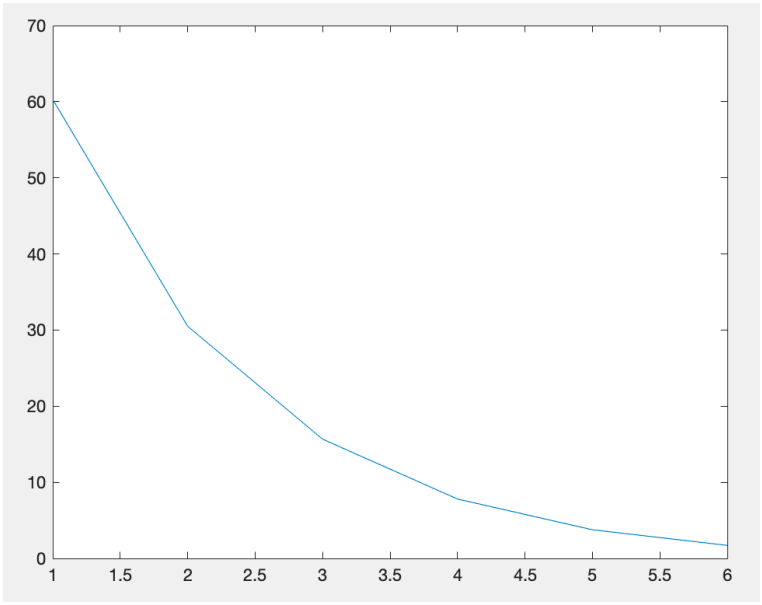


图 10: ORL 数据集迭代次数

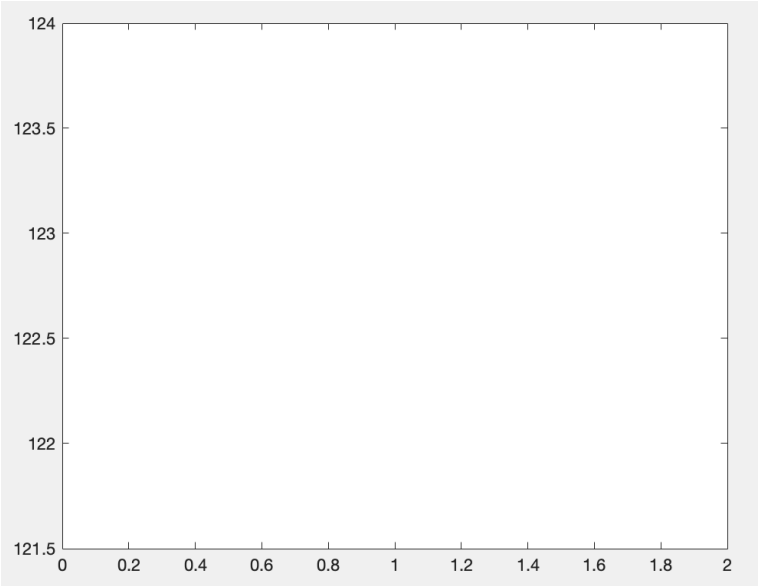


图 11: COIL-20 数据集迭代次数

可以看出，我的实验结果与作者的实验结果略有差别，但迭代次数都是一样的。图 9 的意思是指 GSF 算法只迭代了一次，图中只是一个点，纵坐标表示设计出来的目标函数的值

对于一个聚类算法来讲，它的参数不敏感性是非常重要的。参数不敏感性是指修改算法的一些参数，得到的实验结果差别应该是不大的甚至是不变的。在论文中作者改变了 (8) 式中的 γ_1 和 γ_2。范围都是从-3 到 1。实验结果表明，改变参数后算法分类的准确率没有发生改变或者变化非常小。这说明该算法的参数不敏感性非常好。作者的实验结果如下图所示。

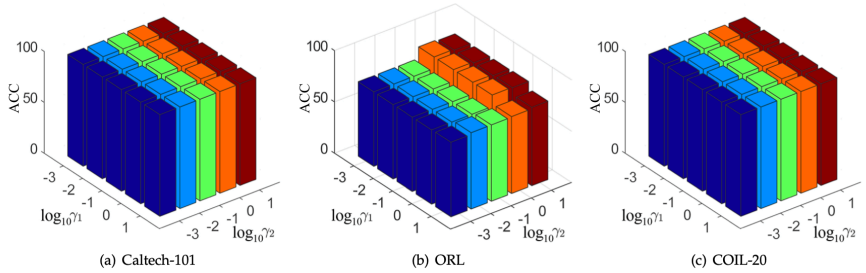


图 12: 参数敏感性

我的实验结果图 13、14、15 所示。

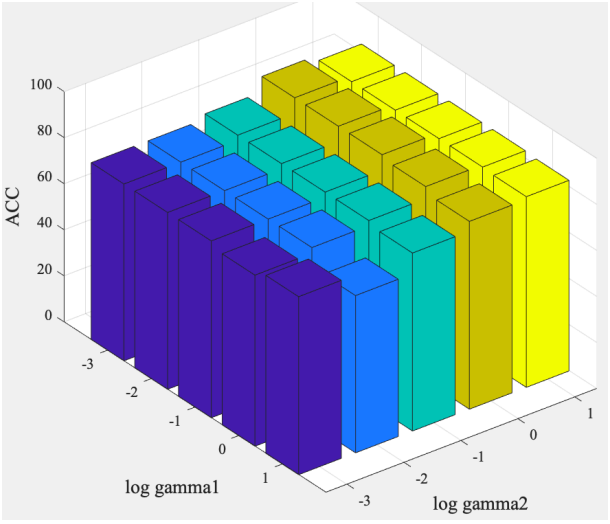


图 13: Caltech 数据集参数敏感性

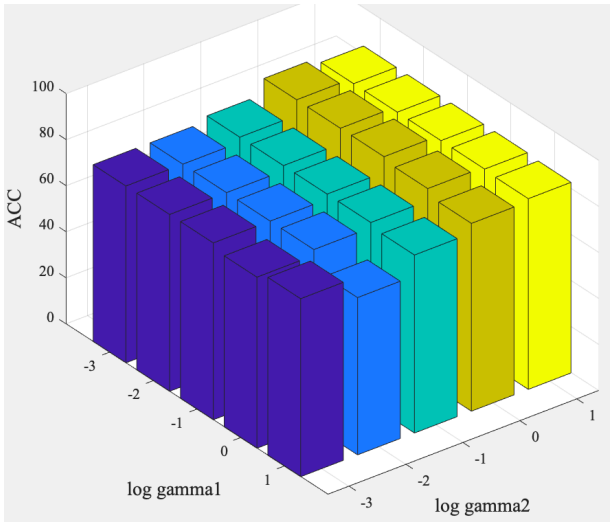


图 14: ORL 数据集参数敏感性

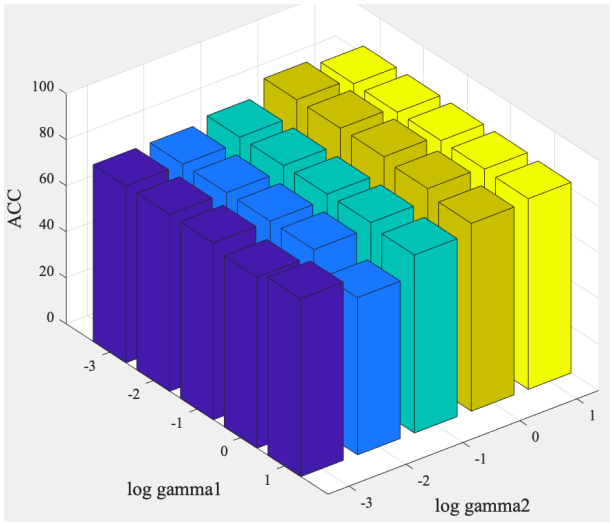


图 15: COIL-20 数据集参数敏感性

可以看出，我的实验结果在 ORL 数据集和 COIL-20 数据集上与论文是一致的。但是在 Caltech 数据集上差异较大。我致邮了教授，教授的回复说是图片放错了。我在 matlab 当中在 Caltech 数据集上运行了多次 GSF。发现 GSF 算法的参数敏感性是非常好的！

6 总结与展望

在本文中，作者提出了一个全新的多视图聚类方法：GSF。算法利用了数据的多视图内部的图结构来学习出一个内在图，并且设计了一个目标函数，对这个目标函数进行最优化求解来迭代学习出这个内在图。在迭代生成内在图时，数据分类的过程也在同步进行着。当迭代结束时，多视图聚类的工作完成。实验结果表明，GSF 比世界上现有的多视图聚类算法都要优越

然而，尽管 GSF 算法非常出色，但就目前而言，GSF 算法的应用范围比较狭隘，仅仅只能用于多视图的图像数据的聚类。而且论文中的实验用的数据集不够多。在未来希望能够将 GSF 算法推广到其他领域如网络等数据类型，GSF 也能够用上。

参考文献

[1] ZHAN K, NIU C, CHEN C, et al. Graph structure fusion for multiview clustering[J]. IEEE Transactions on Knowledge and Data Engineering, 2019, 31(10): 1984-1993. DOI: https://doi.org/10.1109/TKDE.20

18.2872061.

[1]