MDS6106 Introduction to Optimization
Final Project

Yuzhen Jiang 221025144

Runkai Zheng 221041036

Huanli Wu 221041040

Weixiang Wang 221041057

Hongrui Tan 221041031

Dec 27, 2021

Contents

1 Introduction **3**
1.1 Background of Convex Clustering . . . 3
1.2 Objectives . . . 3

2 Data Preparation **3**

3 Huber-Type Clustering **4**
3.1 Accelerated Gradient Method . . . 5
3.2 Newton-CG method . . . 6
3.3 On generative datasets . . . 6
3.4 On real-word datasets . . . 6
3.5 Different regularization coefficiency . . . 6

4 Weighted Models **8**
4.1 On generative datasets . . . 9
4.2 On real-word datasets . . . 10
4.3 Different regularization coefficiency . . . 11

5 Extension and Stochastic Optimization **11**
5.1 Homogeneity, Completeness and V-measure . . . 11
5.2 L-BFGS . . . 11
5.3 Regularization . . . 11
5.4 Stochastic Gradient Methods . . . 12

1 Introduction

1.1 Background of Convex Clustering

Clustering is one of the most fundamental problems in unsupervised learning. Different from usual classification tasks, unsupervised learning problems do not require labeled training data and try to estimate labels, classes, or clusters directly from the given data. Traditional clustering models, such as K-means and hierarchical clustering, often suffer from poor performance because of the inherent non-convexity of the models and the difficulties in finding global optimal solutions for such models. The clustering results are generally highly dependent on suitable initializations and can differ significantly for different starting points. More importantly, standard clustering models require prior knowledge about the number of clusters which is not available in many applications. Therefore, in practice, K-means is typically applied with different cluster numbers and the user then decides on an appropriate value.

1.2 Objectives

In order to overcome these issues, new convex clustering models have been recently proposed. Let $R \in A = (a_1, a_2, \cdots, a_n)$ be a given data matrix with n observations and d features. The convex clustering model for these n observations solves the following convex optimization problem:

$$\min_{X \in \mathbb{R}^{d \times n}} \frac{1}{2} \sum_{i=1}^{n} \|x_i - a_i\|^2 + \lambda \sum_{i=1}^{n} \sum_{j=i+1}^{n} \|x_i - x_j\|_p \tag{1}$$

where $\lambda > 0$ is a regularization parameter and $\|\cdot\|_p$ denotes the $p-$norm. Here, $\|\cdot\|$ denotes the standard Euclidean norm (as usual). The $p-$norm above with $p \geq 1$ ensures the convexity of the model. Typically, p is chosen to be 1, 2, or ∞.
After solving 1 and obtaining the optimal solution $X^* = (x_1^*, \cdots, x_n^*)$, we assign a_i and a_j to the same cluster if and only if $x_i^* = x_j^*$. In other words, x_i^* acts as a centroid (center of the cluster) for the observation a_i. In practice, instead of requiring $x_i^* = x_j^*$, we can assign a_i and a_j to the same cluster if $\|x_i^* - x_j^*\| \leq \epsilon$ for a given tolerance $\epsilon > 0$. The key idea behind the convex clustering model is that, if two observations a_i and a_j belong to the same cluster, then their corresponding centroids x_i^* and x_j^* should be the same. The first term in the model 1 is the fidelity or data term while the second term is the regularization term to penalize the differences between different centroids so as to enforce the property that centroids for observations in the same cluster should be identical.
In this project, we want to study different optimization approaches for the convex clustering model 1 and potential alternative strategies and compare their performance on several real-word data sets and tasks.

2 Data Preparation

In order to handle the sparse matrices, we decide to use *PyTorch* to simplify and accelerate our development on the algorithms. We design a class, **DataGenerator** to generate the data that we want. The **DataGenerator** class can generate 2-dimension data points with specific number of clusters, scales, the number of points in each clusters, the centroids and the variance of each cluster, and the separation of the points. The **DataGenerator** class has three public methods, *get_data()*, *plot()*, and *write()*. *get_data()* is used to get the points in Python environment. *plot()* is used to plot the points via Matplotlib. The last method, *write()* is used to export the data and store it as text file. The (Fig. 1) is the data points printed by the *plot()* method.
We can set different parameters (p value) to get data distributions with different number of clusters. The following figures shows the different data distributions with different number of

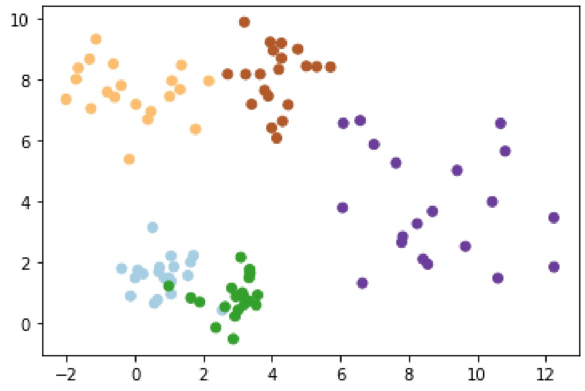
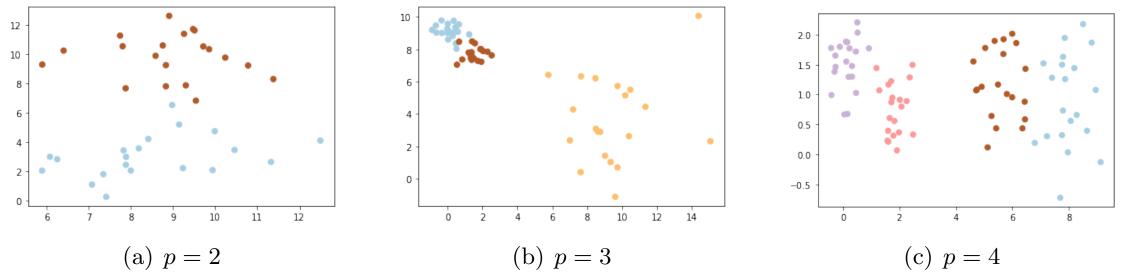


Figure 1: The generative data points ($p = 5$)

clusters. In (Fig. 2), from (a) to (c), the number of clusters goes from two to four.



(a) $p = 2$ (b) $p = 3$ (c) $p = 4$

Figure 2: The generative data points with different p values

Besides, we can set different parameters (*sep* value) to get data distributions with different separation of data points. The following figures shows the different data distributions with different separation of data points. In (Fig. 3), from (a) to (d), the separation of data points goes from one, three, five, and eight.

3 Huber-Type Clustering

In the following, we choose $p = 2$ and we first consider a smooth variant of the clustering problem (1):

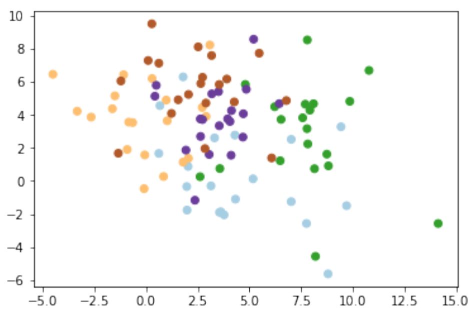
$$\min_{X \in \mathbb{R}^{d \times n}} f_{clust}(X) := \frac{1}{2} \sum_{i=1}^{n} \|x_i - a_i\|^2 + \lambda \sum_{i=1}^{n} \sum_{j=i+1}^{n} \varphi_{hub}(x_i - x_j) \tag{2}$$

Here, $\varphi_{hub}(x)$ denotes the Huber-type version of the Euclidean norm:

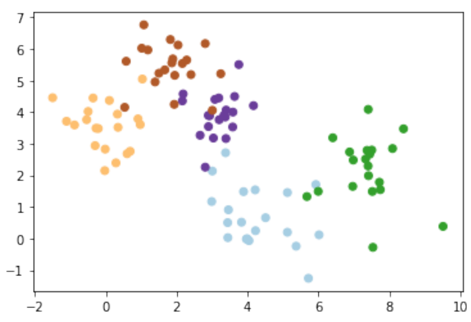
$$\varphi_{hub}(x) = \begin{cases} \frac{1}{2\sigma}\|x\|^2 & \|x\| \leq \delta \\ \|x\| - \frac{\sigma}{2} & \|x\| > \delta \end{cases}$$

To implement first-order and second-order optimization, we have to calculate the gradient vector and the hessian matrix. The gradient is calculated by:

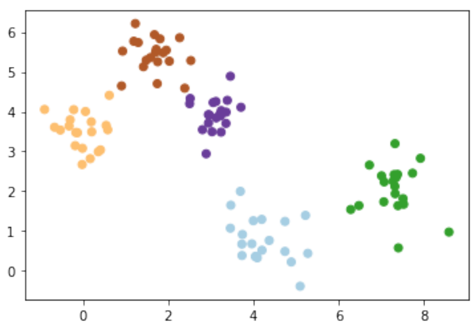
$$G_i = \sum_{a=1}^{n} \sum_{b=a+1}^{n} \frac{\partial \phi_{hub}(x_a - x_b)}{\partial x_i}, \tag{3}$$



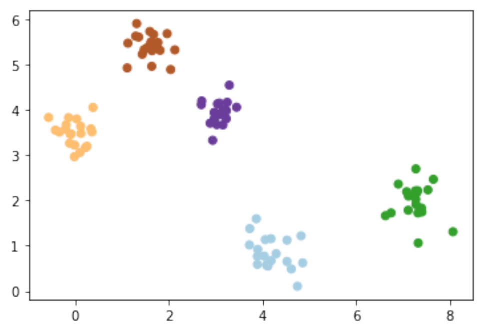
(a) $sep = 1$



(b) $sep = 3$



(c) $sep = 5$



(d) $sep = 8$

Figure 3: The generative data points with different sep values

where

$$\frac{\partial \phi_{hub}(x_i - x_j)}{\partial x_i} = \begin{cases} \frac{x_i - x_j}{\delta}, & \|x_i - x_j\| \le \delta \\ \frac{x_i - x_j}{\|x_i - x_j\|}, & \|x_i - x_j\| > \delta, \end{cases} \tag{4}$$

The hessian is given by:

$$H_{i,j} = \sum_{a=1}^{n} \sum_{b=a+1}^{n} \frac{\partial^2 \phi_{hub}(x_a - x_b)}{\partial x_i \partial x_j}, \tag{5}$$

where

$$\frac{\partial^2 \phi_{hub}(x_i - x_j)}{\partial x_i \partial x_j} = \begin{cases} \frac{1}{\delta} I_{d\times d}, & \|x_i - x_j\| \le \delta \\ \frac{J_{d\times d}}{\|x_i - x_j\|} - \frac{(x_i - x_j)(x_i - x_j)^T}{\|x_i - x_j\|^3}, & \|x_i - x_j\| > \delta, \end{cases} \tag{6}$$

Note that if a and b are not i or j, $\frac{\partial \phi_{hub}(x_a - x_b)}{\partial x_i} = 0$ ($1 \times d$ zero vector) and $\frac{\partial^2 \phi_{hub}(x_a - x_b)}{\partial x_i \partial x_j} = 0$ ($d \times d$ all-zero matrix).
The computation of gradient and hessian using for-loop can be extremely expensive when n and d are large. Thus, we implement all the calculation by tensor operations. The comparison of the calculation time on different sizes of the Hessian matrix is shown in (Table 1).

3.1 Accelerated Gradient Method

The accelerated gradient method (AGM) can be described as following (Alg. 1).
The AGM method for the smoothed convex clustering are implemented with fixed step size and basic extrapolation strategy:

$$\alpha_k = \frac{1}{L}, \quad \beta_k = \frac{t_{k-1} - 1}{t_k}, \quad t_k = \frac{1}{2}\left(1 + \sqrt{1 + 4t_{k-1}^2}\right), \quad t_{-1} = t_0 = 1.$$

Table 1: Efficiency improvement by tensor operations

Matrix Size	For Loop	Tensor Operation	Ratio
2×2	0.0009s	0.0009s	1.00
4×4	0.0009s	0.0009s	1.00
8×8	0.0020s	0.0010s	2.00
16×16	0.0132s	0.0030s	4.4
32×32	2.1388s	0.2010s	10.64
64×64	145.34s	0.4025s	361.09

Algorithm 1 Accelerated Gradient Method

```
Require: $x^0 \in \mathbb{R}^n$
  $x^{-1} \leftarrow x^0$
  for $k = 0, 1, \ldots$ do
    $\beta_k$ is an extrapolation parameters
    $y^{k+1} \leftarrow x^k + \beta_k(x^k - x^{k-1})$
    $\alpha_k > 0$ is a chosen step size
    $x^{k+1} \leftarrow y^{k+1} - \alpha_k \nabla f(y^{k+1})$
  end for
```

The Lipschitz constant L of ∇f_{clust} is given by $L = 1 + n\lambda/\delta$. Alternatively, you can also use the following parameter strategies:

$$\alpha_k = \frac{1}{L}, \quad \text{and} \quad \beta_k = \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}},$$

where $\mu > 0$ denotes the strong convexity parameter of f_{clust}.

3.2 Newton-CG method

The Newton method can be described as follwoing (Alg. 2).

3.3 On generative datasets

We choose model parameter set ($\lambda = 0.15, \delta = 0.01$) for dataset 1, ($\lambda = 0.17, \delta = 0.01$) for dataset 2 and ($\lambda = 0.4, \delta = 0.3$) for dataset 3.
We choose $\alpha = \frac{1}{L}, tol = 0.001$ for AGM and ($\lambda = 0.15, \delta = 0.01, cg - max = 10, \gamma = 0.01, \sigma = 0.5, tol = 0.001$) for Newton-CG and test the performance of both optimization algorithms. The results are as follows (Fig. 456) and (Table 2[1]).

3.4 On real-word datasets

The results are as following (Table 3).

3.5 Different regularization coefficiency

The relationship between λ and the number of clusters are as following (Fig. 7). It is obvious that the higher λ is, the smaller number of clusters is. But there exists an elbow slope change when λ is about 0.15.

[1]The H and C in the table stand for Homogeneity and Completeness, which are metrics of the models. We will introduce them in 5.1

Algorithm 2 The full Newton-CG Method

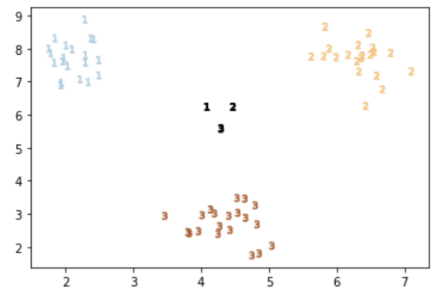
```
Require: $x^0 \in \mathbb{R}$ and $\sigma, \gamma \in (0,1), s > 0$ and $(\rho_k)_k$
Ensure: $\|\nabla f(x^{k+1})\| \leq \epsilon$
  for $k = 0, 1, \ldots$ do
    $A \leftarrow \nabla^2 f(x^k)$
    $v^0 \leftarrow 0$
    $r^0 \leftarrow \nabla f(x^k)$
    $p^0 \leftarrow -r^0$
    $tol \leftarrow \rho_k \cdot \|\nabla f(x^k)\|$
    for $j = 0, 1, \ldots$ do
      if $(p^j)^\top A p^j \leq 0$ then
        return $d^k = v^j$ (or $d^k = -\nabla f(x^k)$ if $j = 0$)
      end if
      $\sigma_j \leftarrow \frac{\|r^j\|^2}{(p^j)^\top A p^j}$
      $v^{j+1} \leftarrow v^j + \sigma_j p^j$
      $r^{j+1} \leftarrow r^j + \sigma_j A p^j$
      if $\|r^{j+1} \leq tol\|$ then
        return $d^k \leftarrow v^{j+1}$
      end if
      $\beta_{j+1} \leftarrow \frac{\|r^{j+1}\|^2}{\|r^j\|^2}$
      $p^{j+1} \leftarrow -r^{j+1} + \beta_{j+1} p^j$
    end for
    $\alpha_k \leftarrow$ backtracking
    $x^{k+1} = x^k + \alpha_k d^k$
    if $\|\nabla f(x^{k+1})\| \leq \epsilon$ then
      return $x^{k+1}$
    end if
  end for
```

Table 2: Huber-type clustering performance on generative datasets

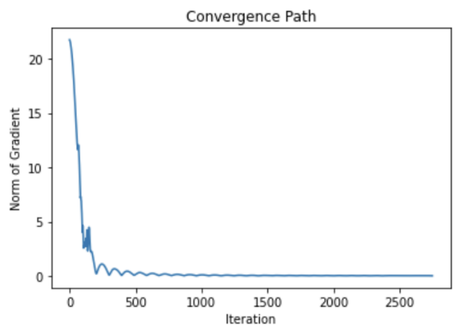
Dataset	Method	Iterations	CPU time	Compression ratio	H	C	V-measure
Dataset 1	AGM	2750	27.022s	0.110	1.000	1.000	1.000
	Newton-CG	304	7.797s	0.127	1.000	1.000	1.000
Dataset 2	AGM	3530	37.130s	0.032	0.528	1.000	0.691
	Newton-CG	359	8.992s	0.291	0.814	0.797	0.806
Dataset 3	AGM	54	1.086s	0.009	0.567	0.859	0.683
	Newton-CG	28	1.980s	0.010	0.567	0.859	0.683

Table 3: Huber-type clustering performance on real-world datasets

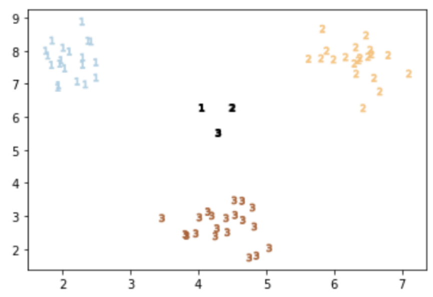
Dataset	Method	Iterations	CPU time	Compression ratio	H	C	V-measure
Wine	AGM	991	11.584s	0.013	0.553	0.381	0.451
	Newton-CG	437	123.050s	0.015	0.602	0.477	0.532
Vowel	AGM	1329	73.966s	0.028	0.544	0.469	0.504



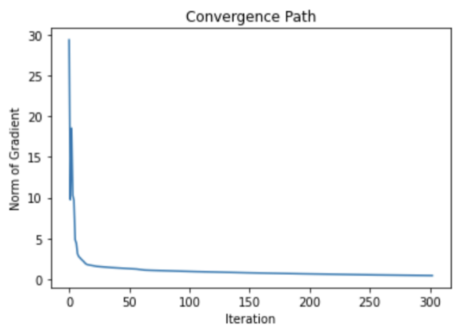
(a) AGM clusters



(b) AGM covergence path

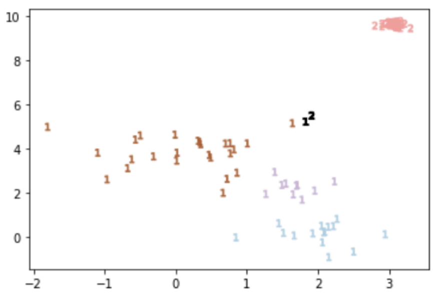


(c) Newton-CG clusters

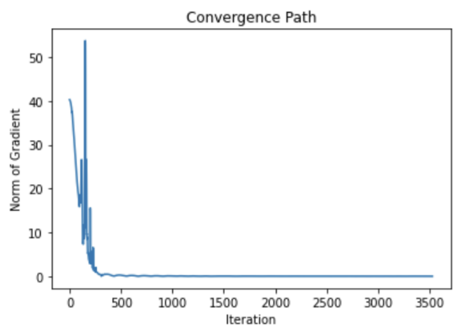


(d) Newton-CG covergence path

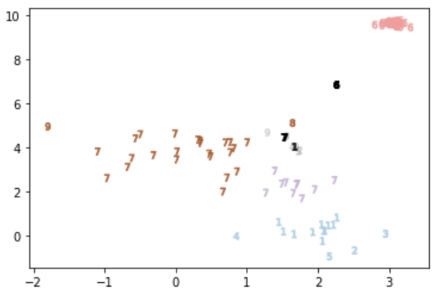
Figure 4: Huber-type clustering with generative dataset one



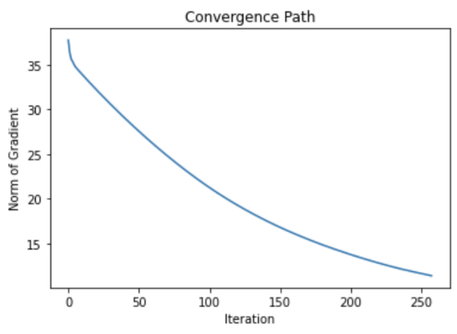
(a) AGM clusters



(b) AGM covergence path



(c) Newton-CG clusters



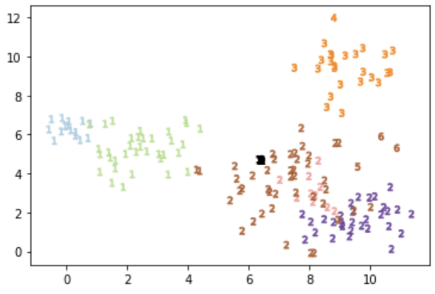
(d) Newton-CG covergence path

Figure 5: Huber-type clustering with generative dataset two

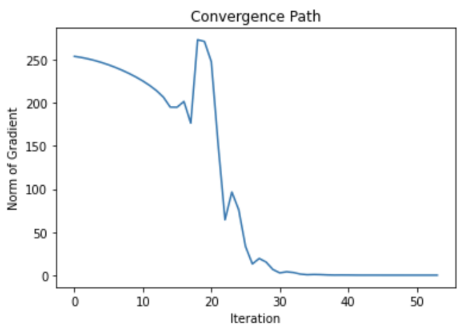
4 Weighted Models

To handle cluster recovery for large-scale data sets, the following weighted convex clustering model modified from (1) can be considered:

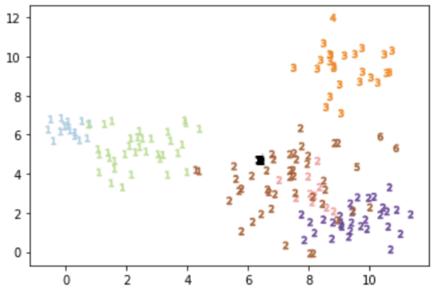
$$\min_{X \in \mathbb{R}^{d \times n}} f_{clust}(X) := \frac{1}{2} \sum_{i=1}^{n} \|x_i - a_i\|^2 + \lambda \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} \|x_i - x_j\| \tag{7}$$



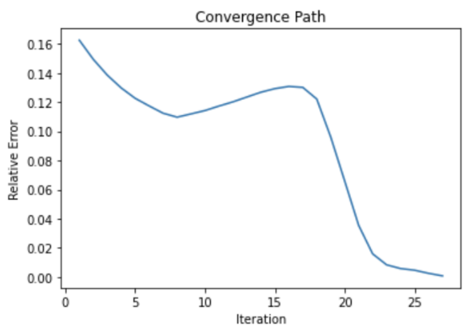
(a) AGM clusters



(b) AGM covergence path



(c) Newton-CG clusters



(d) Newton-CG covergence path

Figure 6: Huber-type clustering with generative dataset three

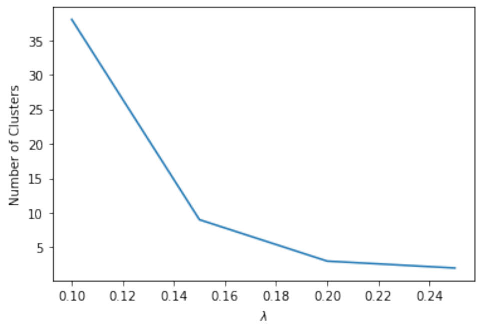


Figure 7: relationship between λ and the number of clusters

where $w_{ij} = w_{ji} \geq 0$ are given weights that are chosen based on the input data A. In other words, previous model (1) is a special case of this model (7), where $w_{ij} = 1$ for all $i < j$. To make the computational cost cheaper when evaluating the regularization terms, one would generally put a non-zero weight only for a pair of points which are nearby each other, and a typical choice of the weights is

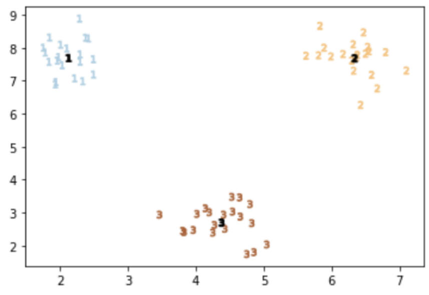
$$w_{ij} = \begin{cases} \exp(-\vartheta \|a_i - a_j\|^2) & (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

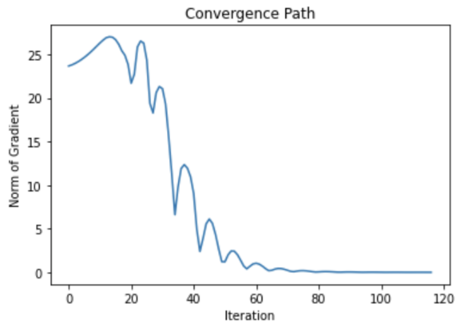
where $\mathcal{E} = \bigcup_{i=1}^{n} \{(i,j) : a_j \text{is among } a_i\text{'s k-nearest neighbors}, i < j \leq n\}$ and $\theta > 0$ is a given positive constant.

4.1 On generative datasets

We choose parameter set $(\lambda = 0.15, tol = 0.0001, \delta = 0.01)$ for AGM and $(\lambda = 0.15, tol = 0.0001, \delta = 0.01, cg - max = 10, \gamma = 0.01, \sigma = 0.5)$ for Newton-CG. The results are as following

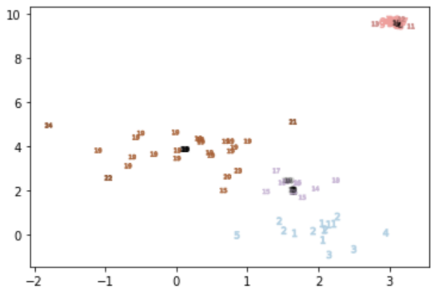
(Fig. 89).

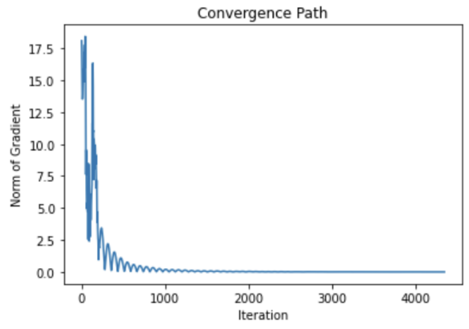

(a) AGM clusters



(b) AGM covergence path

Figure 8: Weighted Model with generative dataset one


(a) AGM clusters



(b) AGM covergence path

Figure 9: Weighted Model with generative dataset two

Table 4: Weighted model clustering performance on generative datasets

Dataset	Method	Iterations	CPU time	Compression ratio	H	C	V-measure
Dataset 1	AGM	117	1.529s	0.925	1.000	1.000	1.000
Dataset 2	AGM	4341	57.520s	0.860	0.997	0.462	0.628
Dataset 3	AGM	10000	224.715s	0.935	0.795	0.646	0.713

4.2 On real-word datasets

Here we use weighted model with AGM on the wine dataset. The parameters and results are as following (Fig. 10) and (Table 5).

Table 5: Weighted Model clustering performance on dataset **wine**

Method	Iterations	CPU time	Compression ratio	Homogeneity	Completeness	V-measure
AGM	3043	45.173s	0.754	0.861	0.535	0.660

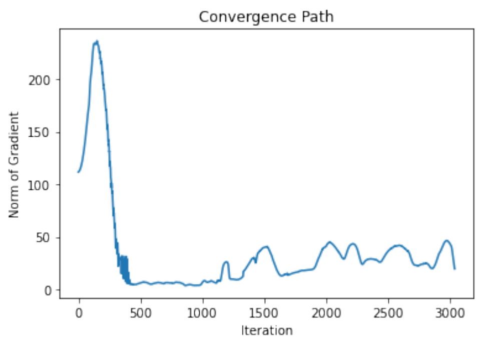


Figure 10: Weighted clustering with AGM

4.3 Different regularization coefficiency

5 Extension and Stochastic Optimization

5.1 Homogeneity, Completeness and V-measure

Homogeneity denotes that every cluster only contains one type of samples. If one cluster only has one type of samples, then its homogeneity is one; if one cluster has many types of samples, then we have to calculate its conditional experience entropy $H(C|K)$. The higher conditional experience entropy, the lower the homogeneity.

$$h = \begin{cases} 1 & H(C) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & otherwise \end{cases}$$

Completeness denotes that same types of samples are in one cluster. If the same types of samples are in one cluster, then the completeness is one; if the same types of samples are not in one cluster, then we have to calculate its conditional experience entropy $H(K|C)$. The higher conditional experience entropy, the lower the completeness.

$$c = \begin{cases} 1 & H(K) = 0 \\ 1 - \frac{H(K|C)}{H(K)} & otherwise \end{cases}$$

However, it is uneven to just take care of homogeneity or completeness. Then we use V-measure, which combines the two metrics. If $\beta > 1$, we more focus on completeness; if $\beta < 1$, we more focus on homogeneity.

$$v_\beta = \frac{(1+\beta) \cdot h \cdot c}{\beta \cdot h + c}$$

5.2 L-BFGS

We further implement limited memory BFGS method (L-BFGS) on our generated dataset. We choose model parameter The results are shown in (Table 6):

5.3 Regularization

We test the model with different types of regularization, including L_1 norm L_{inf} norm and log smoothing. The norms are defined as:

$$\|x\|_1 = \sum_i |x_i| \tag{8}$$

Table 6: Performance of L-BFGS on generative datasets

Dataset	Method	Iterations	CPU time	Compression ratio	H	C	V-measure
Dataset 1	L-BFGS	24	0.156	0.925	1.000	1.000	1.000
Dataset 2	L-BFGS	36	0.239s	0.958	0.950	0.789	0.862
Dataset 3	L-BFGS	6	0.103	0.958	0.955	0.372	0.535

$$\|x\|_2 = \sum_i \sqrt{x_i^2} \tag{9}$$

$$\|x\|_\infty = max_i |x_i| \tag{10}$$

$$\phi_{hub}(x) = log(1 + \|x\|^2/v) \tag{11}$$

The gradient vector is calculated by:

$$\frac{\partial \|x\|_1}{\partial x_i} = sgn(x_i) \tag{12}$$

$$\frac{\partial \|x\|_2}{\partial x_i} = sgn(x_i) \tag{13}$$

$$\frac{\partial \|x\|_\infty}{\partial x_i} = sgn(x_i)\delta_i j, \quad j = max_i |x_i| \tag{14}$$

$$\frac{\partial \phi_{hub}(x)}{\partial x_i} = \frac{2x_i}{1 + \|x\|^2} \tag{15}$$

where delta is the Kronecker delta function, which is zero when $i \neq j$ and one when $i = j$. **Note that for L1 norm and L-infinity norm, the gradient is not well-defined because of the sign function. Thus, we use subgradient by defining $sgn(0) = 0$ when $x_i = 0$.**
The results are shown in (Table 7):

Table 7: Performance of Different Regularization Model

Model	Algorithm	Iterations	CPU time	Compression ratio	H	C	V-measure
Weighted L-1	LBFGS	202	8.319s	0.923	1.000	0.943	0.971
Weighted L-∞	LBFGS	121	4.617s	0.925	1.000	0.848	0.917
Weighted L-2	LBFGS	109	1.074s	0.923	1.000	0.848	0.917

5.4 Stochastic Gradient Methods

To reduce memory usage when performing optimization on large-scale dataset, we use stochastic approximation to avoid the storage of huge gradient vector. As shown in Figure 11, consider our data matrix and parameter matrix, which have the same shape of d times n. In each iteration, we don't use all the data. Instead, we randomly pick up m samples out of the all the data samples. The gradient will then flows from these m samples to the corresponding m columns on the parameter matrix. Then we only need to update these columns in the current iteration. We conduct experiments on self-generated dataset, the results are shown in **??**:
We set m, that is the batch size, to different numbers and obtain different clustering results. Generally, we can see that smaller batch size leads to poorer clustering results and more unstable training curve, and moreover, convergent more slowly. This is because we try to approximate

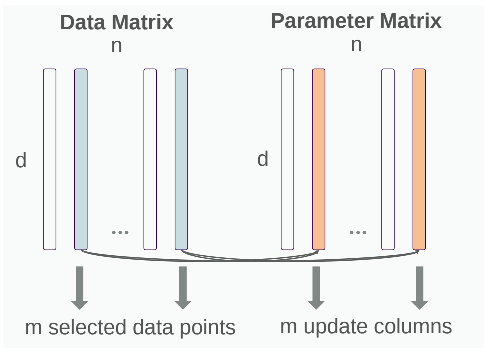


Figure 11: Principle of Stochastic Optimization

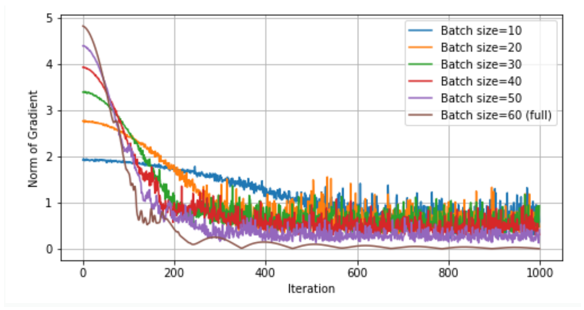


Figure 12: Results of Stochastic Optimization on self-generated dataset.

the gradient on the full data matrix using only part of it. If the batch size is too small, the estimation of the gradient is more inaccurate, thus result in unstable training curve and poorer results.