Deep Co-Clustering

Dongkuan Xu*‡ Wei Cheng†‡ Bo Zong† Jingchao Ni† Dongjin Song†
Wenchao Yu§ Yuncong Chen† Haifeng Chen† Xiang Zhang*

Abstract

Co-clustering partitions instances and features simultaneously by leveraging the duality between them and it often yields impressive performance improvement over traditional clustering algorithms. The recent development in learning deep representations has demonstrated the advantage in extracting effective features. However, the research on leveraging deep learning frameworks for co-clustering is limited for two reasons: 1) current deep clustering approaches usually decouple feature learning and cluster assignment as two separate steps, which cannot yield the task-specific feature representation; 2) existing deep clustering approaches cannot learn representations for instances and features simultaneously. In this paper, we propose a deep learning model for co-clustering called DeepCC. DeepCC utilizes the deep autoencoder for dimension reduction, and employs a variant of Gaussian Mixture Model (GMM) to infer the cluster assignments. A mutual information loss is proposed to bridge the training of instances and features. DeepCC jointly optimizes the parameters of the deep autoencoder and the mixture model in an end-to-end fashion on both the instance and the feature spaces, which can help the deep autoencoder escape from local optima and the mixture model circumvent the Expectation-Maximization (EM) algorithm. To the best of our knowledge, DeepCC is the first deep learning model for co-clustering. Experimental results on various datasets demonstrate the effectiveness of DeepCC.

1 Introduction

Given a data matrix in which one dimension represents instances and the other represents features, co-clustering aims to cluster both instances and features simultaneously so that the instances and features can be organized into homogeneous blocks. Many co-clustering algorithms have been proposed. Some of them are based on information theory [1, 2]. For instance, Dhillon et al. [1] proposed a co-clustering approach by increasing the preserved mutual information between instances and features. Another line of research leverages the matrix decomposition theory [3–7]. Co-clustering has shown advantages over the traditional one-sided clustering algorithms when the datasets exhibit clear duality between instances and features [1, 3], and been applied in various applications, such as bioinformatics [5, 8] and graph mining [9, 10].

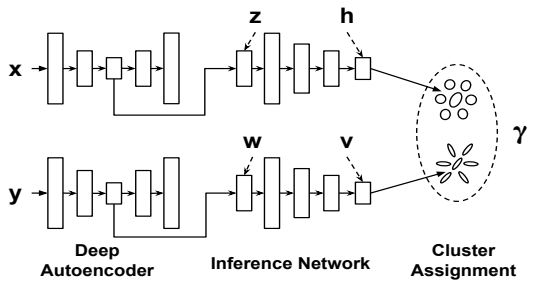


Figure 1: DeepCC is in an end-to-end learning fashion. **x** and **y** are the instances and features of the input data, **z** and **w** produced by deep autoencoder are the low-dimensional representations, **h** and **v** are the outputs of inference network and utilized by a variant of GMM to produce the co-cluster assignment, $\boldsymbol{\gamma}$.

The recent development in learning deep representations has shown its advantage in extracting effective features. Some researchers thus proposed deep learning models for clustering [11–16] and showed promising results, especially when the data was sampled from a nonlinear low dimensional manifold. For example, Xie et al. [11] proposed an unsupervised deep embedding for clustering analysis that first conducted dimension reduction and then fed the low-dimensional representations to k-means for clustering. These approaches typically utilize the deep autoencoder to map data into a low-dimensional space first, which can not only conduct dimension reduction, but also help the model avoid trivial solutions [12]. The new data representations were then fed into different learning algorithms according to different clustering objective functions, such as minimizing KL divergence between cluster assignments and an auxiliary distribution in [11].

*The Pennsylvania State University. {dux19, xzz89}@psu.edu
†NEC Laboratories America, Inc. {weicheng, bzong, jni, dsong, yuncong, haifeng}@nec-labs.com
‡These authors contributed equally to this work
§University of California Los Angeles. {yuwenchao}@ucla.edu

Although the deep clustering research has shown the promising results, the research on leveraging deep representation learning for co-clustering is limited. Current deep clustering methods usually decoupled feature learning and cluster assignment as two separate steps with inconsistent optimization goals. It is difficult to directly employ them to learn the embedding representations for instance/feature and the cluster assignment jointly. This is because the joint feature extraction requires an end-to-end learning. Moreover, existing deep clustering methods do not have an effective loss function suitable for deep learning framework to bridge the representation learning for instances and features.

In this paper, we propose, DeepCC, a deep learning model for co-clustering. Its framework is shown in Fig. 1. DeepCC utilizes the deep autoencoder to generate low-dimensional representations for instances and features, and employs a novel variant of GMM for inferring the cluster assignments. It utilizes a fully-connected neural network called inference network to generate the initial cluster assignment probability distribution for the variant of GMM, to facilitate the parameter learning of the mixture model. Instead of maximizing the log-likelihood of GMM directly, DeepCC maximizes its variational lower bound to achieve a more effective training. A mutual information loss is developed to bridge the training of instances and features. Specifically, DeepCC jointly minimizes the reconstruction error of deep autoencoder and maximizes the variational lower bound of the log-likelihood in GMM. This end-to-end optimization balances the autoencoding reconstruction, the cluster assignment and the regularization. This helps the deep autoencoder escape from less attractive local optima and DeepCC effectively circumvent the standard EM algorithm in traditional GMM. The main contributions are summarized as follows:

- We propose a novel deep co-clustering model, deepCC. To the best of our knowledge, this is the first deep learning model for co-clustering.
- DeepCC jointly minimizes the reconstruction error of deep autoencoder and maximizes the variational lower bound of the log-likelihood in GMM, which helps deep autoencoder escape from local optima.
- A mutual information loss is proposed to bridge the training of instances and features.
- Experimental results on various datasets demonstrate the effectiveness of DeepCC.

In the rest of the paper, the problem is defined in Section 2, DeepCC is described detailedly in Section 3, experimental evaluations are followed in Section 4, and the related work is reviewed in Section 5.

2 Problem Formulation

Given instances and features represented by $\{\mathbf{x}_i\}_{i=1}^n = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ and $\{\mathbf{y}_j\}_{j=1}^d = \{\mathbf{y}_1, \cdots, \mathbf{y}_d\}$ respectively, co-clustering aims to group instances into g clusters and features into m clusters, i.e., to find maps C_r and C_c:

$$C_r : \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \to \{\hat{\mathbf{x}}_1, \cdots, \hat{\mathbf{x}}_g\}$$
$$C_c : \{\mathbf{y}_1, \cdots, \mathbf{y}_d\} \to \{\hat{\mathbf{y}}_1, \cdots, \hat{\mathbf{y}}_m\}$$

where subscripts r and c indicate instances and features. We can reorder the instances/features such that the instances/features grouped into the same cluster are arranged to be adjacent. The resulting new data structure consists of blocks called co-clusters. Suppose that X and Y are two discrete variables taking values from the sets $\{\mathbf{x}_i\}_{i=1}^n$ and $\{\mathbf{y}_j\}_{j=1}^d$ respectively. The joint probability distribution between X and Y is denoted by $p(X, Y)$. Similarly, $\hat{X}$ and $\hat{Y}$ are two discrete variables from the sets $\{\hat{\mathbf{x}}_s\}_{s=1}^g = \{\hat{\mathbf{x}}_1, \cdots, \hat{\mathbf{x}}_g\}$ and $\{\hat{\mathbf{y}}_t\}_{t=1}^m = \{\hat{\mathbf{y}}_1, \cdots, \hat{\mathbf{y}}_m\}$ respectively. The joint probability distribution between $\hat{X}$ and $\hat{Y}$ is denoted by $p(\hat{X}, \hat{Y})$. Here, $\hat{X}$ and $\hat{Y}$ indicate the partitions induced by X and Y, i.e., $\hat{X} = C_r(X)$ and $\hat{Y} = C_c(Y)$.

3 Deep Co-Clustering

3.1 Dimension Reduction DeepCC realizes dimension reduction with deep autoencoder [17]. Dimension reduction based on deep autoencoder is meaningful and necessary for clustering tasks. Deep autoencoder can generate semantically meaningful and well-separated representation with much lower dimensionality, which can both decrease computation cost for further calculation and remove noise. In addition, the reconstruction part in deep autoencoder can work as an regularization to prevent from getting trivial solutions [12].

DeepCC utilizes deep autoencoder to reduce both the dimensionality of instances and features. Given the i-th instance and the j-th feature denoted by $\mathbf{x}_i$ and $\mathbf{y}_j$ respectively, then the low-dimensional representations of them are denoted by:

$$\mathbf{z}_i = f_r(\mathbf{x}_i; \theta_r) \tag{3.1}$$
$$\mathbf{w}_j = f_c(\mathbf{y}_j; \theta_c) \tag{3.2}$$

where f_r, f_c denote the encoding functions for instances and features, and θ_r, θ_c are the parameters. The encoding functions can be linear or nonlinear depending on the domain data. The reconstruction errors of $\mathbf{x}_i$ and $\mathbf{y}_j$ can be denoted by $l(\mathbf{x}_i, g_r(\mathbf{z}_i; \theta_r))$ and $l(\mathbf{y}_j, g_c(\mathbf{w}_j; \theta_c))$ respectively, where g_r and g_c are the decoding functions.

3.2 Cluster Assignment Inference Given the low-dimensional representations produced by deep autoencoder, DeepCC utilizes a modified GMM framework

with variational inference approach [18] to produce cluster assignment probability. DeepCC adopts an inference network for density estimation under the framework of GMM to avoid the EM in traditional GMM.

The outputs of the inference network are the new representations of instance $\mathbf{x}_i$ and feature $\mathbf{y}_j$ denoted by $\mathbf{h}_i = (h_{i1}, \cdots, h_{ig})^T$ and $\mathbf{v}_j = (v_{j1}, \cdots, v_{jm})^T$ respectively, where g and m are the cluster numbers. Because of the *softmax* function deployed on the last layer of the inference network, $\mathbf{h}_i$ and $\mathbf{v}_j$ can also be considered as the cluster assignment probability. The cluster assignment distributions of instances and features based on the inference network are denoted by $Q_{\eta_r}(k|\mathbf{h}_i)$ and $Q_{\eta_c}(k|\mathbf{v}_j)$, where η_r and η_c denote the parameters of inference networks. We denote the posterior cluster assignment probability distributions of $\mathbf{h}_i$ and $\mathbf{v}_j$ based on GMM by $P_{\phi_r}(k|\mathbf{h}_i)$ and $P_{\phi_c}(k|\mathbf{v}_j)$, where ϕ_r and ϕ_c are the parameters of GMMs.

Instead of applying the standard EM strategy for GMM, DeepCC jointly trains the inference neural network and GMM in an end-to-end fashion. We take the training for instances as an example and similar training can be applied for features. Given the output of deep autoencoder $\mathbf{z}_i$, the new representation $\mathbf{h}_i$ is:

$$\mathbf{h}_i = Softmax(MLN(\mathbf{z}_i; \eta_r)), \tag{3.3}$$

where MLN indicates the multi-layer neural network. Then the mixture probability, mean, covariance of the k-th component in GMM, i.e., $\phi_r = \{\pi_r^k, \mu_r^k, \Sigma_r^k\}$, can be estimated as:

$$\pi_r^k = N_r^k / N_r, \qquad \mu_r^k = \frac{1}{N_r^k} \sum_{i=1}^{N_r^k} h_{ik} \mathbf{h}_i \tag{3.4}$$

$$\Sigma_r^k = \frac{1}{N_r^k} \sum_{i=1}^{N_r} h_{ik} (\mathbf{h}_i - \mu_r^k)(\mathbf{h}_i - \mu_r^k)^T \tag{3.5}$$

where $N_r{=}n$ is the number of instances, $N_r^k = \sum_{i=1}^{N_r} h_{ik}$, and h_{ik} is the value on the k-th dimensionality of $\mathbf{h}_i$. Then, the cluster assignment probability of i-th instance belonging to the k-th cluster is:

$$\gamma_{r(i)}^k = \frac{\pi_r^k \mathcal{N}(\mathbf{h}_i | \mu_r^k, \Sigma_r^k)}{\sum_{k'=1}^{g} \pi_r^{k'} \mathcal{N}(\mathbf{h}_i | \mu_r^{k'}, \Sigma_r^{k'})} \tag{3.6}$$

where $\mathcal{N}(\cdot)$ is the normal distribution probability density function. Then the log-likelihood is:

$$\log\{\prod_{i=1}^{N_r} P_{\phi_r}(\mathbf{h}_i)\} = \sum_{i=1}^{N_r} \log P_{\phi_r}(\mathbf{h}_i) = \sum_{i=1}^{N_r} \log\{\sum_{k=1}^{K} \pi_r^k \mathcal{N}(\mathbf{h}_i | \mu_r^k, \Sigma_r^k)\} \tag{3.7}$$

Instead of maximizing the log-likelihood directly, DeepCC tries to maximize its variational lower bound. The benefits are two-fold: 1) it makes the distribution Q_{η_r} a better approximation to the distribution P_{ϕ_r} via minimizing the KL divergence between them, which makes the parameter estimation of GMM more accurate; 2) it tightens the bound of log-likelihood function, which makes the training process more effective. The merits will be demonstrated by experiments in Section 4.3. Specifically, the variational lower bound on log-likelihood, $\mathcal{L}_r$, is derived as follows:

$$\sum_{i=1}^{N_r} \log P(\mathbf{h}_i) = \sum_{i=1}^{N_r} \log \int_k P(k, \mathbf{h}_i) \tag{3.8}$$

$$= \sum_{i=1}^{N_r} \log \int_k \frac{P(k, \mathbf{h}_i)}{Q(k|\mathbf{h}_i)} Q(k|\mathbf{h}_i) \tag{3.9}$$

$$= \sum_{i=1}^{N_r} \log(E_Q[\frac{P(k, \mathbf{h}_i)}{Q(k|\mathbf{h}_i)}]) \tag{3.10}$$

$$\geq \sum_{i=1}^{N_r} E_Q[\log \frac{P(k, \mathbf{h}_i)}{Q(k|\mathbf{h}_i)}] \tag{3.11}$$

$$= \sum_{i=1}^{N_r} \{E_Q[\log P(k, \mathbf{h}_i)] + H(k|\mathbf{h}_i)\} \tag{3.12}$$

$$= \mathcal{L}_r \tag{3.13}$$

where $H(k|\mathbf{h}_i) = -E_Q(\log Q(k|\mathbf{h}_i))$ is the Shannon entropy, and P_{ϕ_r}, Q_{η_r} are denoted by P, Q for brevity. Equation 3.11 applies the Jensen's inequality. Furthermore, $\mathcal{L}_r$ can be derived to be the subtraction between the log-likelihood and the KL divergence between Q and P, shown as follows:

$$\mathcal{L}_r = \sum_{i=1}^{N_r} \{E_Q[\log P(k, \mathbf{h}_i)] - E_Q(\log Q(k|\mathbf{h}_i))\} \tag{3.14}$$

$$= \sum_{i=1}^{N_r} \{\int_k Q(k|\mathbf{h}_i) \log \frac{P(k, \mathbf{h}_i)}{Q(k|\mathbf{h}_i)} - \tag{3.15}$$

$$\int_k Q(k|\mathbf{h}_i) \log P(\mathbf{h}_i) + \log P(\mathbf{h}_i)\} \tag{3.16}$$

$$= \sum_{i=1}^{N_r} \{\int_k Q(k|\mathbf{h}_i) \log \frac{P(k, \mathbf{h}_i)}{Q(k|\mathbf{h}_i) P(\mathbf{h}_i)} + \log P(\mathbf{h}_i)\} \tag{3.17}$$

$$= \sum_{i=1}^{N_r} \{\int_k Q(k|\mathbf{h}_i) \log \frac{P(k|\mathbf{h}_i)}{Q(k|\mathbf{h}_i)} + \log P(\mathbf{h}_i)\} \tag{3.18}$$

$$= \sum_{i=1}^{N_r} \{-KL(Q(k|\mathbf{h}_i) || P(k|\mathbf{h}_i)) + \log P(\mathbf{h}_i)\} \tag{3.19}$$

Similarly, the cluster assignment probability of the j-th feature belonging to the k-th cluster is:

$$\gamma_{c(j)}^k = \frac{\pi_c^k \mathcal{N}(\mathbf{v}_j | \mu_c^k, \Sigma_c^k)}{\sum_{k'=1}^{m} \pi_c^{k'} \mathcal{N}(\mathbf{v}_j | \mu_c^{k'}, \Sigma_c^{k'})} \tag{3.20}$$

where π_c^k, μ_c^k, Σ_c^k are the mixture probability, mean, covariance of the k-th component in the GMM for the features, and m is the number of feature clusters. The variational lower bound on log-likelihood for features is:

$$\mathcal{L}_c = \sum_{j=1}^{N_c} \{E_Q[\log P(k, \mathbf{v}_j)] - E_Q(\log Q(k|\mathbf{v}_j))\} \tag{3.21}$$

where N_c=d is the number of features, and P_{ϕ_c}, Q_{η_c} are denoted by P, Q for brevity. Finally, DeepCC takes $-\mathcal{L}_r$ and $-\mathcal{L}_c$ as the loss for cluster assignment of instances and features.

3.3 Instance-Feature Cross Loss DeepCC utilizes an instance-feature cross loss term based on mutual information to make the trainings of instances and features intertwined. Based on the cluster assignments of instances and features, DeepCC constructs the joint probability distribution between instances and features as $p(X, Y)$, and the one between instance clusters and feature clusters as $p(\hat{X}, \hat{Y})$. DeepCC penalizes on the mutual information loss between the two joint probability distributions.

Given the cluster assignment probability of the i-th instance as $\boldsymbol{\gamma}_{r(i)} = (\gamma^1_{r(i)}, \cdots, \gamma^g_{r(i)})^T$ and the one of the j-th feature as $\boldsymbol{\gamma}_{c(j)} = (\gamma^1_{c(j)}, \cdots, \gamma^m_{c(j)})^T$, the joint probability between the i-th instance and the j-th feature is denoted by $p(\mathbf{x}_i, \mathbf{y}_j) = \mathcal{J}(\boldsymbol{\gamma}_{r(i)}, \boldsymbol{\gamma}_{c(j)})$, where $\mathcal{J}(\cdot)$ is a function to calculate the joint probability. The joint probability between the s-th instance cluster, $\hat{\mathbf{x}}_s$, and the t-th feature cluster, $\hat{\mathbf{y}}_t$, is calculated as:

$$p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t) = \sum \{p(\mathbf{x}_i, \mathbf{y}_j) | \mathbf{x}_i \in \hat{\mathbf{x}}_s, \mathbf{y}_j \in \hat{\mathbf{y}}_t\} \tag{3.22}$$

We simply take the dot product operation as $\mathcal{J}(\cdot)$ in DeepCC. This is because: 1) almost all existing co-clustering algorithms set the cluster numbers of instances and features as equal, i.e., $g = m$, which makes the dot product operation practicable; 2) there exists corresponding relationship between instance clusters and feature clusters, i.e., similar instances share similar features [1]. Note that in DeepCC the design of $\mathcal{J}(\cdot)$ is flexible to use other metrics.

Given the joint probability distributions $p(X, Y)$ and $p(\hat{X}, \hat{Y})$, the mutual information between X and Y and the one between $\hat{X}$ and $\hat{Y}$ are calculated as follows:

$$I(X; Y) = \sum_{\mathbf{x}_i} \sum_{\mathbf{y}_j} p(\mathbf{x}_i, \mathbf{y}_j) \log \frac{p(\mathbf{x}_i, \mathbf{y}_j)}{p(\mathbf{x}_i) p(\mathbf{y}_j)} \tag{3.23}$$

$$I(\hat{X}; \hat{Y}) = \sum_{\hat{\mathbf{x}}_s} \sum_{\hat{\mathbf{y}}_t} p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t) \log \frac{p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t)}{p(\hat{\mathbf{x}}_s), p(\hat{\mathbf{y}}_t)} \tag{3.24}$$

where $p(\mathbf{x}_i)$=$\sum_{\mathbf{y}_j} p(\mathbf{x}_i, \mathbf{y}_j)$, $p(\mathbf{y}_j)$=$\sum_{\mathbf{x}_i} p(\mathbf{x}_i, \mathbf{y}_j)$, $p(\hat{\mathbf{x}}_s)$ = $\sum_{\hat{\mathbf{y}}_t} p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t)$ and $p(\hat{\mathbf{y}}_t)$=$\sum_{\hat{\mathbf{x}}_s} p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t)$. Then the difference between $I(X; Y)$ and $I(\hat{X}; \hat{Y})$ is:

$$I(X; Y) - I(\hat{X}; \hat{Y}) \tag{3.25}$$

$$= \sum_{\hat{\mathbf{x}}_s} \sum_{\hat{\mathbf{y}}_t} \sum_{\mathbf{x}_i \in \hat{\mathbf{x}}_s} \sum_{\mathbf{y}_j \in \hat{\mathbf{y}}_t} p(\mathbf{x}_i, \mathbf{x}_j) \log \frac{p(\mathbf{x}_i, \mathbf{x}_j)}{p(\mathbf{x}_i) p(\mathbf{x}_j)} \tag{3.26}$$

$$- \sum_{\hat{\mathbf{x}}_s} \sum_{\hat{\mathbf{y}}_t} (\sum_{\mathbf{x}_i \in \hat{\mathbf{x}}_s} \sum_{\mathbf{y}_j \in \hat{\mathbf{y}}_t} p(\mathbf{x}_i, \mathbf{y}_j)) \log \frac{p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t)}{p(\hat{\mathbf{x}}_s) p(\hat{\mathbf{y}}_t)} \tag{3.27}$$

$$= \sum_{\hat{\mathbf{x}}_s} \sum_{\hat{\mathbf{y}}_t} \sum_{\mathbf{x}_i \in \hat{\mathbf{x}}_s} \sum_{\mathbf{y}_j \in \hat{\mathbf{y}}_t} p(\mathbf{x}_i, \mathbf{y}_j) \log \frac{p(\mathbf{x}_i, \mathbf{y}_j)}{q(\mathbf{x}_i, \mathbf{y}_j)} \tag{3.28}$$

$$= KL(p(X, Y) || q(X, Y)) \tag{3.29}$$

$$\geq 0 \tag{3.30}$$

where $q(\mathbf{x}_i, \mathbf{y}_j)$=$p(\hat{\mathbf{x}}_s, \hat{\mathbf{y}}_t) \frac{p(\mathbf{x}_i)}{p(\hat{\mathbf{x}}_s)} \frac{p(\mathbf{y}_j)}{p(\hat{\mathbf{y}}_t)}$. So $I(X; Y)$ - $I(\hat{X}; \hat{Y}) \geq 0$, and also $I(X; Y) \geq 0$, $I(\hat{X}; \hat{Y}) \geq 0$, which motivates our instance-feature cross loss as:

$$1 - \frac{I(\hat{X}; \hat{Y})}{I(X; Y)} \tag{3.31}$$

The intuition of the cross loss term is that the difference between $I(X; Y)$ and $I(\hat{X}; \hat{Y})$ should not be significant for an optimal co-clustering [1, 2]. The value of the proposed cross loss is normalized in the range [0, 1] by its definition, facilitating quick and effective hyper-parameter tuning for different datasets.

3.4 Joint Objective Function Given instances and features denoted by $\{\mathbf{x}_i\}_{i=1}^n$ and $\{\mathbf{y}_j\}_{j=1}^d$ respectively, the objective function of DeepCC is as follows:

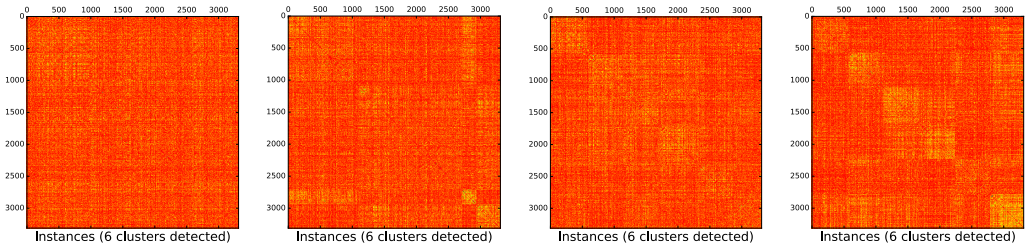
$$\min_{\theta_r, \theta_c, \eta_r, \eta_c} J = J_1 + J_2 + J_3 \tag{3.32}$$

$$J_1 = \frac{\lambda_1}{n} \sum_{i=1}^{n} l(\mathbf{x}_i, g_r(\mathbf{z}_i)) + \lambda_2 P_{ae}(\theta_r) + \lambda_3(-\mathcal{L}_r) + P_{inf}(\Sigma_r)$$

$$J_2 = \frac{\lambda_1}{d} \sum_{j=1} l(\mathbf{y}_j, g_c(\mathbf{w}_j)) + \lambda_2 P_{ae}(\theta_c) + \lambda_3(-\mathcal{L}_c) + P_{inf}(\Sigma_c)$$

$$J_3 = \lambda_4 (1 - \frac{I(\hat{X}; \hat{Y})}{I(X; X)})$$

where J_1, J_2 are the loss for the trainings of instances and features, J_3 is the instance-feature cross loss, θ_r, θ_c are the parameters of deep autoencoder for instances and features respectively, η_r, η_c are the parameters of inference neural network for instances and features, $l(\mathbf{x}_i, g_r(\mathbf{z}_i))$ and $l(\mathbf{y}_j, g_c(\mathbf{w}_j))$ are the reconstruction errors, $P_{ae}(\theta_r)$ and $P_{ae}(\theta_c)$ are the penalties for the parameters of deep autoencoder to avoid overfitting, and $-\mathcal{L}_r$ and $-\mathcal{L}_c$ are the negative of variational lower bounds of log-likehood in GMM. $P_{inf}(\Sigma_r) = \sum_{k=1}^{g} \sum_{i=1}^{d_r} \frac{1}{\sum^k_{r_{ii}}}$ and $P_{inf}(\Sigma_c) = \sum_{k=1}^{m} \sum_{j=1}^{d_c} \frac{1}{\sum^k_{c_{jj}}}$ are the sum of inverse of the diagonal entries in covariance matrices, where d_r and d_c are the dimensionality of the outputs of deep autoencoder. $P_{inf}(\Sigma)$ is used to avoid trivial solutions where diagonal entries in covariance matrices degenerate to zero.



(a) k-means. (b) SCC. (c) CCInfo. (d) DeepCC.

Figure 2: Comparison of clustering results on the Citeseer dataset.

Table 1: Description of the datasets.

Dataset	#instances	#features	#classes
Coil20	1440	1024	20
Yale	165	1024	15
Fashion-MNIST-test	10000	784	10
Sign-MNIST-test	7172	784	25
Citeseer	3312	3703	6
WebKB4	4199	1000	4
WebKB_cornell	195	1703	5
WebKB_texas	187	1703	5
WebKB_washington	230	1703	5
WebKB_wisconsin	265	1703	5
IMDb_movies_keywords	617	1878	17
IMDb_movies_actors	617	1398	17

4 Experiments

4.1 Datasets, Baseline Methods, and Evaluation Metric We utilize some popular benchmark datasets including images, texts and user ratings. The dataset description is shown in Table 1. Coil20, Yale, Fashion-MNIST [1] and Sign-MNIST [2] are image related datasets. Coil20 contains 32×32 gray scale images collected from 20 3D objects. Yale contains 165 face images from 15 individuals. Fashion-MNIST is about Zalando's article images. Sigh-MNIST is about hand gesture images. We use the test sets of the two datasets. Some text related datasets are utilized. Citeseer contains 3312 documents and each document is described by 3703 words. WebKB4 is about web page information. WebKB_cornell, WebKB_texas, WebKB_washington and WebKB_wisconsin[3] contain different kinds of documents. IMDb_movies_keywords and IMDb_movies_actors are movie rating datasets. They are about the same 617 movies described by keywords and actors respectively.

The baselines include: k-means, spectral coclustering(SCC) [19], spectral biclustering(SBC) [20], CCInfo [1], DRCC [3], CCMod [21] and SCMK [22]. Moreover, we compare several variants of DeepCC to demonstrate the importance of individual components in DeepCC. In DCC-INF, we remove the deep autoencoder part in the objective function. In DCC-DAE, we remove the loss based on variational lower bound and use the output of inference network as the clustering assignment. In DCC-LOG, we replace the loss based on variational lower bound with the loss based on log-likelihood. In DCC-SEP, we train autoencoder and inference neural network with GMM separately.

Co-clustering results are evaluated by accuracy (Acc) and NMI as shown in Equations 4.33 and 4.34:

$$Acc = \frac{\sum_{i=1}^{N} \delta(s_i, map(r_i))}{N} \tag{4.33}$$

where N is the number of instances, s_i is the true class label, r_i is the predicted cluster label, $\delta(a,b)$ is the function that equals to one if $a = b$ and zero otherwise, and $map(\cdot)$ is the permutation function that maps the predicted cluster label r_i to the equivalent class label.

$$NMI = \frac{\sum_{i=1}^{c}\sum_{j=1}^{c} n_{i,j} log\frac{n_{i,j}}{n_i \hat{n}_j}}{\sqrt{(\sum_{i=1}^{c} n_i log\frac{n_i}{n})(\sum_{j=1}^{c} \hat{n}_j log\frac{\hat{n}_j}{n})}} \tag{4.34}$$

where n_i and $\hat{n}_j$ are the numbers of data contained in the i-th cluster and the j-th class respectively, and $n_{i,j}$ is the number of data contained in the i-th cluster and the j-th class simultaneously. c is the number of classes. A larger NMI indicates a better clustering result.

4.2 Experimental Settings The neural network structures of DeepCC for different datasets are included in the supplementary file. All layers are fully connected ones. l_2 loss is adopted for the reconstruction error of deep autoencoder. For all the datasets, the number of instance clusters is set as the same as the number of feature clusters. For the baselines, we use their reported parameter settings if clarified in their original papers; otherwise we try different parameter settings and choose the best one. For the hyperparameters of DeepCC, i.e., λ_1, λ_2, λ_3 and λ_4, we utilize the grid-search to determine the optimal setting. The detail of the optimal hyperparameter searching is included in the supplementary file. We run all the algorithms 10 times and report the average results. DeepCC is implemented in Tensorflow [23] and trained by Adam [24]. The learning rate is

[1] https://github.com/zalandoresearch/fashion-mnist
[2] https://www.kaggle.com/datamunge/sign-language-mnist
[3] http://membres-lig.imag.fr/grimal/data.html

Table 2: Clustering accuracy (%) comparison. The best performance on each dataset is in bold.

Dataset	k-means	SCC	SBC	CCMod	DRCC	CCInfo	SCMK	DeepCC
Coil20	58.6±2.3	51.7±0.5	66.8±1.1	21.0±2.0	53.2±2.4	60.6±3.4	65.9±0.8	**73.3±1.9**
Yale	41.8±0.7	33.7±0.3	40.0±1.3	21.4±1.4	13.6±0.4	41.8±2.0	46.6±0.5	**53.3±1.4**
Fashion-MNIST-test	54.6±0.4	44.5±0.5	45.8±0.0	28.8±0.0	44.1±1.8	51.8±2.4	-	**62.7±1.6**
Sign-MNIST-test	30.6±0.6	31.8±0.7	18.0±0.0	12.6±0.3	21.3±2.5	33.2±1.4	-	**37.0±1.3**
Citeseer	37.4±0.0	37.4±0.0	40.8±0.1	44.7±5.2	29.5±1.8	43.0±5.3	50.2±0.7	**59.3±2.1**
WebKB4	60.6±0.1	60.6±0.1	47.5±0.1	68.8±3.1	43.6±0.4	68.8±2.5	52.1±0.2	**71.8±2.8**
WebKB_cornell	55.1±2.1	58.9±0.2	54.4±0.6	55.5±2.6	42.6±0.0	56.6±2.7	49.6±0.2	**68.7±1.4**
WebKB_texas	63.9±2.6	59.4±0.2	59.0±0.3	64.5±3.0	55.1±0.0	64.1±3.6	62.0±0.6	**73.8±1.2**
WebKB_washington	65.6±2.7	60.8±0.0	51.7±1.0	68.0±2.7	46.5±0.0	67.7±2.9	65.4±0.4	**75.7±1.9**
WebKB_wisconsin	71.7±3.1	70.2±0.5	72.8±1.4	72.1±3.9	46.1±0.0	72.9±3.1	73.2±0.9	**77.4±1.4**
IMDb_movies_keywords	19.3±0.8	25.2±0.4	24.0±0.2	24.7±2.1	12.6±1.7	23.0±2.0	23.3±1.1	**30.8±1.7**
IMDb_movies_actors	15.4±0.7	20.5±0.4	20.0±0.4	20.0±1.2	14.1±2.8	15.6±0.7	15.8±1.3	**23.8±0.4**

Table 3: NMI (%) comparison. The best performance on each dataset is in bold.

Dataset	k-means	SCC	SBC	CCMod	DRCC	CCInfo	SCMK	DeepCC
Coil20	75.9±2.3	64.9±0.5	73.9±1.1	51.8±1.9	65.6±2.7	72.7±1.5	72.5±0.9	**78.3±2.7**
Yale	48.7±0.7	41.6±0.3	49.8±1.3	24.6±2.3	14.2±1.2	48.5±2.0	49.2±1.2	**55.7±1.1**
Fashion-MNIST-test	52.4±0.4	41.9±0.5	41.3±0.0	45.8±1.4	42.2±1.6	50.6±2.3	-	**60.4±0.7**
Sign-MNIST-test	29.1±0.6	40.1±0.7	17.2±0.0	14.0±1.8	28.2±1.2	43.1±1.0	-	**46.7±0.6**
Citeseer	2.7±0.0	15.2±0.0	17.3±0.1	16.9±1.6	10.5±2.2	17.7±2.2	21.1±1.5	**29.8±1.3**
WebKB4	26.1±0.1	31.1±0.1	13.0±0.1	40.1±1.0	31.9±1.7	39.7±3.6	10.0±2.3	**40.5±0.6**
WebKB_cornell	18.1±2.1	28.8±0.2	21.0±0.6	18.9±3.8	11.6±0.0	20.6±3.1	25.7±0.5	**35.4±0.9**
WebKB_texas	7.0±2.6	12.6±0.2	9.0±0.3	16.9±2.3	10.2±0.0	18.2±4.4	24.0±0.8	**42.9±1.2**
WebKB_washington	33.3±2.7	25.3±0.0	9.5±1.0	28.7±1.4	15.7±0.0	30.7±3.4	30.3±0.2	**45.9±1.3**
WebKB_wisconsin	37.5±3.1	35.4±0.5	38.2±1.4	35.1±2.8	20.4±0.0	39.3±2.7	42.9±0.4	**46.7±1.7**
IMDb_movies_keywords	13.9±0.8	25.5±0.4	20.6±0.2	21.6±1.1	6.9±0.3	18.7±2.3	18.4±0.8	**26.8±1.6**
IMDb_movies_actors	10.5±0.7	19.3±0.4	17.6±0.4	14.5±0.9	9.3±2.5	9.7±1.0	10.6±1.7	**20.6±2.3**

Table 4: Clustering accuracy (%) comparison on Coil20, WebKB4, Citeseer and WebKB_cornell.

Dataset	DCC-INF	DCC-DAE	DCC-LOG	DCC-SEP	DeepCC
Coil20	68.2±3.1	58.5±1.2	68.5±1.7	62.4±2.2	**73.3±1.9**
WebKB4	60.7±2.3	55.8±0.9	69.9±2.7	67.4±1.7	**71.8±2.8**
Citeseer	47.4±1.6	39.0±2.1	53.4±2.8	24.5±2.8	**59.3±2.1**
cornell	59.5±2.5	58.2±1.4	62.5±1.8	49.7±2.2	**68.7±1.4**

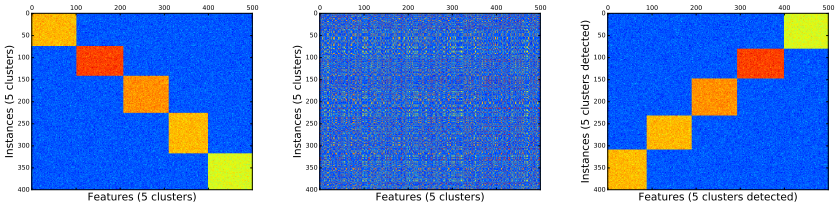
set as 10^{-3} initially and decreases during the training. Pre-training [25] is adopted for the deep autoencoder to make the performance more stable. All the variables are initialized by the Xavier method [26]. The code of DeepCC is available [4].

4.3 Clustering Performance Comparison The comparison of clustering performance is shown in Table 2 and 3, according to which DeepCC outperforms all the baselines. DeepCC is the only method that shows high performance on all different kinds of datasets, which indicated its overall advantage when dealing with different datasets. This advantage attributes to its deep neural structure, which provides DeepCC with a better adaptability. SCMK runs more than five days on two MNIST datasets, so we stop the program and leave its results blank shown as '-'. The performance of DRCC on four WebKB datasets is very low. Some common co-clustering methods illustrate lower performance on some datasets, such as Fashion-MNIST. This is probably because these datasets do not have significant co-cluster structure. However, DeepCC still shows high performance on these datasets, which indicates that DeepCC performs well even when the co-cluster duality is not significant. We also visualize the clustering results on Citeseer (Fig. 2). We first construct the adjacency matrix of instances. Then based on the clustering result, we rearrange the matrix to make the instances that belong to the same cluster be adjacent. The detected clusters correspond to the blocks that are along the diagonal. The clustering performance is higher if the blocks are more significant. According to Fig. 2, the cluster structure is more significant in Fig. 2(d), which verifies the ability of DeepCC to detect the instance clusters.

4.4 Evaluation and Visualization of Co-Clustering The co-clustering ability of DeepCC is verified on both the synthetic and real-world data. The synthetic data with 400 instances, 500 features and 5 co-clusters (Fig. 3(a)). Gaussian noise is added. The data is shuffled (Fig. 3(b)) and then fed into DeepCC. After co-clustered, the instances and features are rearranged to show co-clusters (Fig. 3(c)), according to which DeepCC detects the co-clusters precisely. The detected co-clusters in Fig. 3(c) is not in the same sequence of Fig. 3(a). This is because the cluster ID for the same instance/feature cluster might change, though the instances/features contained are the same.

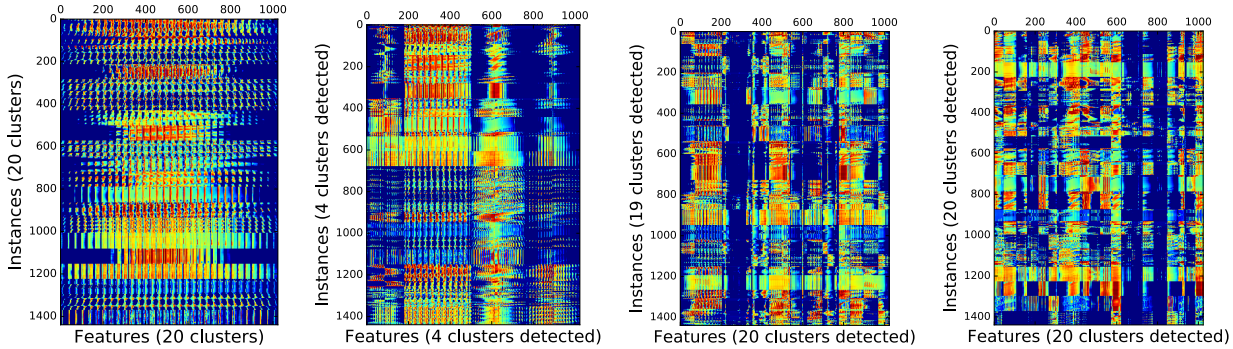
We also visualize the co-clustering results on Coil20. We rearrange the original data matrix according to the instance and feature cluster assignments to show the co-clustering result. The co-clustering result is better if the co-clusters are more significant. We observe that the co-clusters (blocks) in Fig. 4(d) is more significant compared to the ones in Fig. 4(b) and 4(c), which

[4] https://tinyurl.com/y7ywwjp7



(a) Synthetic data. (b) Data are shuffled. (c) Co-clustering result.

Figure 3: The co-clustering result of DeepCC on the synthetic data.



(a) The Coil20 dataset. (b) CCMod. (c) SCC. (d) DeepCC.

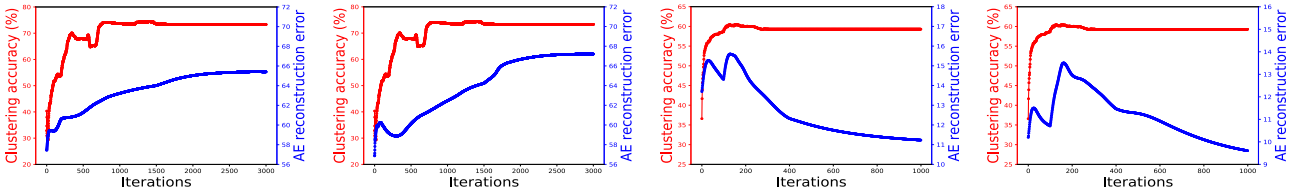
Figure 4: Visualization of the co-clustering results on the Coil20 dataset.

indicates DeepCC outperforms CCMod and SCC. It is noted that there is no significant co-cluster structure along the diagonal in these figures. This is probably because the co-clusters are overlapped in Coil20.

4.5 Advantage of End-to-End Training The reconstruction error of deep autoencoder v.s. the corresponding clustering accuracy is shown in Fig. 5. We observe that it is not necessary that a good clustering is always associated with a small reconstruction error. According to Fig. 5(a) and 5(b), the reconstruction error increases together with the clustering accuracy on Coil20. According to Fig. 5(c) and 5(d), the reconstruction error would not always decrease. This indicates that optimal clustering might not correspond to the least reconstruction error and the end-to-end training helps the deep autoencoder escape from less attractive local optima. In addition, according to Table 4, DeepCC outperforms DCC-SEP. This indicates the end-to-end training well balances the autoencoding reconstruction, the cluster assignment and the regularization.

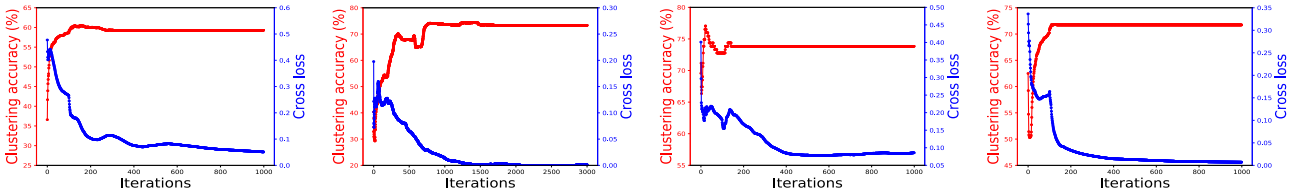
4.6 Effectiveness of Different Parts of DeepCC The performance comparison between DeepCC and its variants is shown in Table 4. DeepCC outperforms DCC-INF, which verifies the effectiveness of the deep autoencoder. DeepCC outperforming DCC-DAE indicates the effectiveness of GMM for guiding the cluster assignment. DeepCC outperforms DCC-LOG, which indicates maximizing variational lower bound benefits clustering more compared to maximizing log-likelihood. The effectiveness of the instance-feature cross loss is demonstrated in Fig. 6, according to which the clustering accuracy increases while the cross loss decreases. The cross loss finally converges. We also apply k-means to the output of the deep autoencoder and compare the clustering result with that of DeepCC (on the original input data). According to Fig. 7, DeepCC outperforms k-means, which demonstrates the validity of the inference network and GMM parts. Additionally, based on the clustering results of k-means (on the original input data) in Table 2, applying k-means on the output of the deep autoencoder obtains better performance. This verifies the effectiveness of the deep autoencoder part.

4.7 Parameter Sensitivity Analysis We take Coil20 as an example dataset. Similar observations can be made on other datasets. We take the setting of $\lambda_1 = 2\times10^{-2}$, $\lambda_2 = 2\times10^{-2}$, $\lambda_3 = 1\times10^{-1}$, $\lambda_4 = 1\times10^{5}$ as the basic setting, which obtains good performance on Coil20. We change the value of one while fixing others. The results are shown in Fig. 8. We observe that the clustering performance does not change sharply for different values of λ_1, λ_2, λ_3, indicating DeepCC is not sensitive to these parameters. However, according to Fig. 8(d), the clustering performance is lower when λ_4 is much smaller or greater than 10^5, indicating that the performance of DeepCC is sensitive to λ_4. This also indirectly verifies the effectiveness of the instance-feature



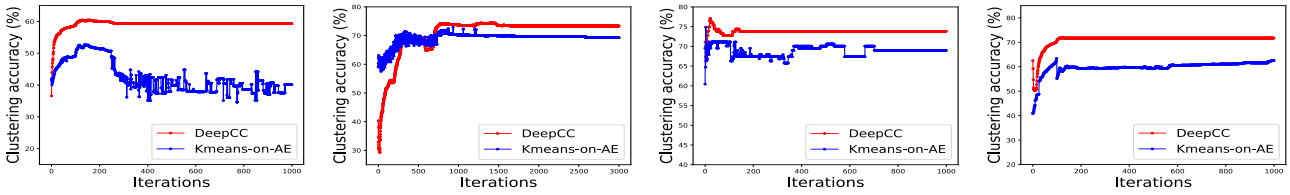
(a) On the instances of Coil20. (b) On the features of Coil20. (c) On the instances of Citeseer. (d) On the features of Citeseer.

Figure 5: Reconstruction error of the deep autoencoder v.s. clustering accuracy on Coil20 and Citeseer.



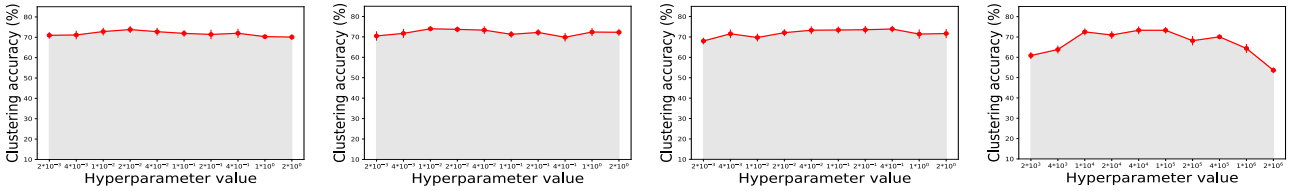
(a) The Citeseer dataset. (b) The Coil20 dataset. (c) The WebKB_texas dataset. (d) The WebKB4 dataset.

Figure 6: Instance-feature cross loss v.s. clustering accuracy on Citeseer, Coil20, WebKB_texas and WebKB4.



(a) The Citeseer dataset. (b) The Coil20 dataset. (c) The WebKB_texas dataset. (d) The WebKB4 dataset.

Figure 7: Clustering performance comparison between DeepCC (on the original input data) and k-means (on the output of the deep autoencoder) on Citeseer, Coil20, WebKB_texas and WebKB4.



(a) λ_1. (b) λ_2. (c) λ_3. (d) λ_4.

Figure 8: Parameter sensitiveness analysis based on Coil20.

cross loss.

5 Related Work

One line of co-clustering research focuses on the information theory. Dhillon et al. in [1] proposed a co-clustering algorithm to intertwine instance and feature clustering based on mutual information. Cheng et al. in [2] constructed a hierarchical structure for instances and features with a minimum number of leaf clusters to realize co-clustering. Another line is based on matrix decomposition theory. Two regularization terms were constructed in [3] to utilize the geometric structure information of instance graph and feature graph separately when realizing semi-nonnegative matrix t-factorization. By regarding the instance-feature data as a dynamical system, Shao et al. in [5] developed a co-clustering method from the perspective of dynamical synchronization. In [7], Nie et al. proposed a co-clustering method to learn a bipartite graph with explicit connected components by imposing the rank constraint on the Laplacian matrix of the bipartite graph. Recently, the research on clustering with deep learning technique attracts more attention. In [11], Xie et al. utilized the deep autoencoder to map data into a low-dimensional space, and then minimized the KL divergence between cluster assignments and an auxiliary distribution. An end-to-end clustering model based on autoencoder was proposed in [14], in which the objective function consisted of the KL divergence between cluster assignments and an auxiliary distribution, and a cluster

assignment regularization term. All of these efforts on clustering with deep learning techniques, however, are different from ours that focuses on the joint learning in instance space and feature space.

6 Conclusion

In this paper, we propose a deep co-clustering model, DeepCC. DeepCC utilizes the deep autoencoder to generate low-dimensional representations for instances and features, which are further fed into the inference neural network in the framework of GMM for cluster assignment prediction. An instance-feature cross loss is employed to bridge the training of instances and features. DeepCC is the first co-clustering model based on deep neural networks, and enjoys the end-to-end fashion. Extensive experimental results demonstrate the effectiveness of DeepCC.

Acknowledgement

This work was partially supported by the National Science Foundation grant IIS-1707548.

References

[1] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *Proceedings of ACM SIGKDD*, 2003, pp. 89–98.

[2] W. Cheng, X. Zhang, F. Pan, and W. Wang, "Hicc: an entropy splitting-based framework for hierarchical co-clustering," *Knowledge and Information Systems*, vol. 46, no. 2, pp. 343–367, 2016.

[3] Q. Gu and J. Zhou, "Co-clustering on manifolds," in *Proceedings of ACM SIGKDD*, 2009, pp. 359–368.

[4] J. Han, K. Xiong, and F. Nie, "Orthogonal and nonnegative graph reconstruction for large scale clustering," in *Proceedings of IJCAI*, 2017, pp. 1809–1815.

[5] J. Shao, C. Gao, W. Zeng, J. Song, and Q. Yang, "Synchronization-inspired co-clustering and its application to gene expression data," in *Proceedings of ICDM*, 2017, pp. 1075–1080.

[6] J. Han, K. Song, F. Nie, and X. Li, "Bilateral k-means algorithm for fast co-clustering." in *Proceedings of AAAI*, 2017, pp. 1969–1975.

[7] F. Nie, X. Wang, C. Deng, and H. Huang, "Learning a structured optimal bipartite graph for co-clustering," in *Proceedings of NIPS*, 2017, pp. 4132–4141.

[8] J. J. Whang and I. S. Dhillon, "Non-exhaustive, overlapping co-clustering," in *Proceedings of CIKM*, 2017, pp. 2367–2370.

[9] K. Rohe, T. Qin, and B. Yu, "Co-clustering directed graphs to discover asymmetries and directional communities," *Proceedings of the National Academy of Sciences*, vol. 113, no. 45, pp. 12 679–12 684, 2016.

[10] D. Hatano, T. Fukunaga, T. Maehara, and K.-i. Kawarabayashi, "Scalable algorithm for higher-order co-clustering via random sampling." in *Proceedings of AAAI*, 2017, pp. 1992–1999.

[11] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proceedings of ICML*, 2016, pp. 478–487.

[12] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, "Towards k-means-friendly spaces: Simultaneous deep learning and clustering," in *Proceedings of ICML*, 2017.

[13] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of CVPR*, 2016, pp. 5147–5156.

[14] K. G. Dizaji, A. Herandi, C. Deng, W. Cai, and H. Huang, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," in *Proceedings of IEEE ICCV*, 2017, pp. 5747–5756.

[15] J. Chang, L. Wang, G. Meng, S. Xiang, and C. Pan, "Deep adaptive image clustering," in *Proceedings of ICCV*, 2017, pp. 5879–5887.

[16] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *Proceedings of ICLR*, 2018.

[17] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[18] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, no. 2, pp. 183–233, 1999.

[19] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of ACM SIGKDD*, 2001, pp. 269–274.

[20] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, "Spectral biclustering of microarray data: coclustering genes and conditions," *Genome Research*, vol. 13, no. 4, pp. 703–716, 2003.

[21] M. Ailem, F. Role, and M. Nadif, "Co-clustering document-term matrices by direct maximization of graph modularity," in *Proceedings of CIKM*, 2015, pp. 1807–1810.

[22] Z. Kang, C. Peng, and Q. Cheng, "Twin learning for similarity and clustering: A unified kernel approach." in *AAAI*, 2017, pp. 2080–2086.

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, vol. 16, 2016, pp. 265–283.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of ICLR*, 2015.

[25] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[26] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of AISTATS*, 2010, pp. 249–256.