Clustering and knowledge integration

Nguyen-Viet-Dung Nghiem

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE
MATHEMATIQUES, INFORMATIQUE, PHYSIQUE
THEORIQUE ET INGENIERIE DES SYSTEMES
Laboratoire d'Informatique Fondamentale d'Orléans

THÈSE présentée par :

Nguyen-Viet-Dung NGHIEM

soutenue le: **15 décembre 2021**

pour obtenir le grade de : **Docteur de l'Université d'Orléans**

Discipline/ Spécialité : Informatique

Clustering and Knowledge Integration

THÈSE dirigée par :

Christel VRAIN	Professeur des universités, Université d'Orléans
Thi-Bich-Hanh DAO	Maître de Conférences HDR, Université d'Orléans

RAPPORTEURS :

Dino IENCO	Chargé de recherche HDR, UMR TETIS
Tias GUNS	Professeur associé, KU Leuven, Belgium

JURY :

Antoine CORNUEJOLS	Professeur des universités, AgroParisTech, Président du Jury
Pierre MARQUIS	Professeur des universités, Université d'Artois
Christel VRAIN	Professeur des universités, Université d'Orléans
Thi-Bich-Hanh DAO	Maître de Conférences HDR, Université d'Orléans
Dino IENCO	Chargé de recherche HDR, UMR TETIS
Tias GUNS	Professeur associé, KU Leuven, Belgium

Résumé

Le clustering sous contraintes (une généralisation du clustering semi-supervisé) vise à exploiter les connaissances des experts lors de la tâche de clustering. Ces connaissances peuvent prendre des formes diverses : des relations entre instances, des conditions sur les clusters, telles que leur cardinalité, ... mais aussi des connaissances plus sémantiques comme par exemple, obtenir des clusters équitables.

Les contraintes peuvent être intégrées à différentes étapes du processus de clustering : en pré-traitement, par exemple en apprenant une nouvelle métrique entre points, pendant le processus de clustering ou dans une étape de post-traitement. La plupart des travaux intègrent des contraintes pendant le clustering, et ils peuvent être divisés en deux approches: modifier des algorithmes de clustering existants pour gérer des contraintes spécifiques / modéliser le problème dans des cadres déclaratifs, tels que la Programmation Linéaire en Nombres Entiers (PLNE), SAT ou la Programmation par Contraintes.

Dans cette thèse, nous proposons trois contributions: (1) une méthode déclarative modifiant une partition existante pour satisfaire des contraintes ; (2) un cadre générique pour intégrer plusieurs types de contraintes dans un modèle de clustering par apprentissage profond ; (3) la définition et la formulation de nouveaux types de contraintes.

Notre première contribution porte sur une méthode de post-traitement, en sortie d'un algorithme de clustering, pour assurer la satisfaction de contraintes. L'originalité est de considérer une matrice d'allocation qui donne les scores d'attribution des points à chaque cluster et de trouver la meilleure partition satisfaisant toutes les contraintes. Nous formulons ce problème comme un problème d'optimisation en PLNE. Des expérimentations montrent que cette méthode est efficace tout en étant compétitive en termes de qualité du clustering par rapport à l'état de l'art. Utiliser une matrice d'allocation permet de post-traiter le résultat de divers algorithmes de clustering, qu'ils soient probabilistes ou issus d'un modèle d'apprentissage profond.

Alors que dans la première contribution, les contraintes sont traitées après un algorithme d'apprentissage, notre deuxième contribution vise à exploiter ces contraintes directement dans un modèle d'apprentissage profond. Les avancées en apprentissage profond permettent de trouver une représentation des données dans des espaces de dimension plus faible grâce à des plongements non linéaires. Elles ont conduit au développement du Deep Clustering, des méthodes de clustering

fondées sur l'apprentissage profond. Pour introduire des contraintes lors de l'étape d'apprentissage, il est courant de disposer d'une fonction de perte pénalisant la non-satisfaction des contraintes. Les travaux actuels introduisent une fonction différente pour chaque type de contrainte (contraintes de taille, contraintes par paires, triplet, ...). Dans notre travail, nous proposons un cadre unifié pour intégrer les contraintes générales en les formalisant en logique et en considérant leurs modèles. A notre connaissance, nous sommes les premiers à proposer pour le clustering profond un cadre générique pour intégrer des contraintes expertes. Nous proposons deux formulations de la fonction de perte, fondées sur la notion de modèles et nous montrons qu'elles peuvent être calculées de manière efficace grâce à des techniques de Weighted Model Counting. Les résultats expérimentaux sur des jeux de données connus montrent que notre approche est compétitive avec d'autres méthodes spécifiques aux contraintes, tout en étant générale.

Outre ces deux méthodes génériques, nous avons défini et formulé de nouveaux types de contraintes en clustering. Premièrement, la contrainte de couverture de cluster limite le nombre de clusters auxquels un groupe de points peut appartenir. Deuxièmement, l'équité combinée prend en compte à la fois l'équité de groupe et l'équité individuelle.

Introduction générale

0.0.1 Les travaux initiaux du clustering sous contraintes

Le clustering sous contraintes permet d'intégrer des connaissances expertes, sous forme de contraintes, dans un processus de clustering. Ces connaissances peuvent être des informations sur l'étiquette des objets, permettant ainsi d'engendrer des contraintes d'instances, must-link ou cannot-link, suivant que les objets doivent être ou non dans le même cluster. Ces contraintes d'instances sont les plus populaires et de nombreuses méthodes en clustering sous contraintes ont été développées pour les intégrer [96, 97, 12, 24, 98]. Mais les connaissances peuvent aussi porter sur les clusters et plusieurs méthodes ont été développées pour intégrer de telles contraintes : contraintes sur la taille minimale des clusters [15], nécessité d'avoir des clusters équilibrés en taille [35, 91], contraintes sur la densité des clusters [24], . . .Comme on peut le voir, ces méthodes sont adaptées à un type de contraintes spécifique. Ces dix dernières années, des méthodes plus générales ont été développées, elles

proposent un cadre générique en se fondant sur des formalismes déclaratifs, comme la programmation linéaire en nombre entiers [3, 67, 76], SAT [25, 65] ou la programmation par contraintes [21, 22]. Ces méthodes souffrent cependant du problème de passage à l'échelle et ne peuvent pas traiter de grandes bases de données.

0.0.2 Le clustering profond

Récemment, des approches de clustering profond ont été largement proposées suite au succès des réseaux de neurones profonds en apprentissage supervisé. Plusieurs directions de recherche ont été envisagées : l'adaptation au clustering d'architectures d'apprentissage supervisé bien connues telles que les réseaux de neurones convolutifs [16], la modification de la représentation des données par un autoencodeur, puis l'application de la structure de clustering à l'espace latent [101, 40].

Une autre approche consiste à extraire le plus proche voisin en se basant sur des caractéristiques prétextes (un encastrement pour une tâche spécifique telle que l'inpainting des patchs, la prédiction du bruit, la discrimination des instances). Cela permet de promouvoir les prédictions similaires des voisins, améliorant ainsi la qualité du clustering [93].

Des approches plus ambitieuses ont été proposées, comme les modèles génératifs qui regroupent les données et génèrent des échantillons pour un regroupement donné ([47, 68]), mais elles souffrent généralement de performances relativement faibles.

0.0.3 L'apprentissage profond avec les connaissances

L'intégration des connaissances peut être considérée comme une généralisation de l'apprentissage semi-supervisé. Alors que l'information sur les étiquettes est facile à représenter, les connaissances des experts peuvent être variées et donc exprimées de nombreuses façons différentes. Pour résoudre ce problème, plusieurs travaux [102, 103] ont étudié l'intégration des connaissances exprimées en logique dans l'apprentissage supervisé profond : les connaissances sont alors appliquées à chaque instance d'entrée individuelle. [103] donne une formulation précise de la perte quelle que soit la forme logique (qu'elle soit représentée en CNF, DNF ou sous une forme arbitraire) au prix d'une complexité computationnelle élevée. [102] apprend une perte de connaissance (en utilisant un encastrement de graphe logique) pour une forme logique spécifique (d-DNNF), ce qui est beaucoup plus rapide mais nécessite

un nombre important de contraintes.

Dans un contexte de clustering, même si chaque point reçoit une étiquette, la sortie de toutes les données est censée représenter une partition (ou une autre structure). Cela signifie que les contraintes ne sont pas posées sur la sortie d'un seul point, mais qu'elles peuvent lier plusieurs sorties, ce qui est beaucoup plus difficile.

[46] intègre des contraintes de triplet dans un cadre de clustering profond. DCC [110] a proposé un cadre de clustering profond, qui peut intégrer plusieurs types de contraintes telles que la paire, le triplet ou la cardinalité. Cependant, pour chaque type de contrainte, une perte spécifique est conçue. Ceci diffère de notre approche, où la même définition de la perte experte est donnée pour tout type de contraintes, dès lors que la contrainte peut être exprimée à l'aide d'une formulation logique.

Résumé des chapitres

Résumé du chapitre 1

Tout d'abord, ce chapitre introduit le problème du clustering: la définition, les critères. Ensuite, il décrit les principaux types de solutions de clustering. Après, ce chapitre passe en revue les vues traditionnelles du problème de clustering contraint en tant que problème combinatoire et donne plusieurs solutions classiques (ne pas utiliser sur les réseaux de neurones). Le premier type de solution, appelé approche algorithmique spécifique, tente de modifier les algorithmes de clustering originaux. Ensuite, nous continuons à décrire l'approche déclarative.

Résumé du chapitre 2

Comme nous l'avons introduit dans le chapitre 1, un certain nombre d'études ont été développées pour résoudre le clustering avec des réseaux de neurones multi-couches, appelé Deep Clustering. Ces travaux méritent d'être expliqués dans ce chapitre en raison des performances de clustering élevées sur divers jeux de données et de l'immense potentiel d'amélioration future. Le chapitre est divisé en deux parties. La première section 2.1 est consacrée aux l'apprentissage de réseaux de neurones profonds, tandis que la deuxième section 2.2 présente les travaux actuels en Deep Clustering.

Résumé du chapitre 3

Dans le chapitre 2, nous avons passé en revue les techniques générales et détaillé plusieurs algorithmes de clustering avec des réseaux de neurones. Dans ce chapitre, nous présentons quelques solutions aux problèmes de clustering sous contrainte avec l'apprentissage profond. Cependant, pour mieux comprendre les contextes, nous proposons un aperçu plus général de l'intégration des connaissances dans un modèle d'apprentissage en profondeur.

Résumé du chapitre 4

Dans ce chapitre, nous proposons une méthode déclarative post-traitement pour adapter la sortie d'un algorithme de clustering afin de satisfaire les contraintes. La première section 4.1 donne un aperçu rapide de la méthode de post-traitement actuelle et de nos nouveautés et contributions à cette approche. Ensuite, la section 4.2 décrit notre cadre : les variables, les exigences et le critère objectif. Dans la section suivante 4.3, nous donnons les formulations des types de contraintes, y compris les nouveaux types de contraintes. En outre, nous donnons également plusieurs scénarios pratiques où un nouveau type de contrainte est nécessaire. La section 4.4 présente l'étude empirique de notre cadre.

Résumé du chapitre 5

Inspirés par plusieurs travaux sur les réseaux de neurones avec intégration des connaissances (Section 3.1), nous transformons les contraintes des experts en un ensemble de formules logiques pour résoudre le problème du clustering sous contraintes. Ensuite, nous définissons deux formulations pour la perte de connaissances, qui sont calculées sur la sortie d'un système de clustering profond. Chaque formulation a une traduction canonique de différents types de contraintes en pertes sémantiques tout en essayant de s'adapter à la tâche de clustering de différentes manières.

Dans les expériences, nous montrons que notre système peut atteindre des résultats comparables aux systèmes de pointe dédiés à des contraintes spécifiques tout en étant flexible pour intégrer et apprendre des contraintes plus complexes.

Contribution

0.0.4 Le premier cadre

Nous proposons un post-traitement des résultats d'un clustering pour forcer la réalisation de contraintes. On peut envisager deux cas d'usage : l'algorithme de clustering n'a pas satisfait toutes les contraintes ou de nouvelles contraintes ont été introduites et il ne paraît pas souhaitable de réapprendre le modèle. Nous nous plaçons dans un cadre probabiliste du clustering où chaque point a une probabilité d'appartenance à un cluster et nous formalisons le problème comme la recherche d'une partition satisfaisant toutes les contraintes et maximisant la probabilité d'affection des points aux clusters. Notre approche est modélisée en PLNE et a plusieurs avantages :

- Différents types de contraintes peuvent être intégrés sans changer la fonction de perte dans le cœur du système d'apprentissage profond.
- La méthode est efficace permettant de traiter de grandes bases de données.
- La satisfaction des contraintes est garantie.
- La méthode prend en entrées une matrice représentant les probabilités d'affectations des points aux clusters et peut en conséquence être utilisée avec tout algorithme de clustering produisant une telle matrice, algorithme d'apprentissage profond ou EM par exemple.

0.0.5 Le deuxième cadre

Nos secondes contributions sont

- Nous proposons une formulation logique du problème de clustering sous contraintes et une définition unifiée de la perte experte pour intégrer les contraintes dans un cadre de clustering profond.
- Étant donné un ensemble de contraintes, la perte experte est basée sur un score de satisfaction des contraintes, défini grâce à la notion de modèles sémantiques d'une formule logique, ce qui la rend indépendante du type de contraintes. De plus, ce score peut être calculé par comptage de modèles pondérés.

- Nous montrons que notre cadre peut être intégré dans différents cadres de clustering en considérant deux méthodes de clustering profond bien connues, à savoir IDEC et SCAN, et en les étendant pour intégrer la connaissance.
- Les expériences menées sur cinq ensembles de données avec des ensembles de contraintes générés de manière aléatoire montrent que notre cadre est compétitif avec les systèmes de clustering profond à contraintes de pointe sur les contraintes par paires et triplets.
- Pour illustrer la généricité de notre approche, nous introduisons un nouveau type de contraintes, appelé contrainte limitée par l'étendue.
- Nous analysons l'efficacité de notre cadre à la fois sur le temps d'exécution et sur la satisfaction des contraintes pour les contraintes complexes.
- Nous montrons que la satisfaction des contraintes lors de la formation du modèle permet d'améliorer la satisfaction des contraintes non vues sur les données de test.

Conclusion

Le clustering sous contrainte ou clustering semi-supervisé est une application dominante de l'apprentissage automatique. En raison du processus naturel de collecte de données, les données sont souvent non étiquetées. Par conséquent, la tâche de clustering de données est répondue sur la connaissance du domaine, qui est exprimée par les contraintes de l'expert. L'algorithme de clustering sous contrainte peut cibler un type spécifique de contrainte, comme les contraintes de paires, de triplets ou de taille de cluster. L'algorithme peut aussi être un cadre général qui facilite l'ajout de nouvelles contraintes. Cependant, la complexité du cadre général est souvent difficile à résoudre.

Dans notre travail, nous nous intéressons à l'intégration de connaissances sous forme générale dans un processus de clustering. Nous le considérons de deux manières : un problème d'optimisation classique et un problème d'apprentissage profond.

Le contexte du premier cadre Dans l'approche classique, les cadres de clustering sous contraintes sont développés à l'aide d'un outil d'optimisation général, tel

que la programmation linéaire en nombres entiers (ILP) [3, 67, 76], le problème de satisfiabilité booléenne (SAT) [25, 65] ou la programmation par contraintes [21, 22]. La plupart de ces méthodes sont des méthodes d'optimisation en cours de processus qui tentent de trouver une solution optimale globale satisfaisant toutes les contraintes. Cependant, elles souffrent d'un manque d'efficacité et de flexibilité car elles doivent traiter et spécifier à la fois l'objectif de clustering et les exigences des contraintes. Kuo et al.[58] proposent une approche post-processus où les contraintes des experts sont traitées après avoir reçu un résultat de clustering. Cette approche simplifie le problème d'optimisation et peut être appliquée à un algorithme de clustering existant. Néanmoins, la fonction objective, qui est basée sur un résultat de clustering dur, donne des résultats moins compétitifs que d'autres travaux.

Le premier cadre considère les contraintes des experts comme une connaissance postérieure. Le cadre modifie le résultat d'un algorithme de clustering pour satisfaire les contraintes des experts. Alors que le précédent cadre de post-traitement optimisait le nombre de points réaffectés, nos travaux considèrent le "score de clustering" - le coût d'une nouvelle partition basée sur l'algorithme de clustering. Nous proposons deux scores de clustering (voir section 4.1.2), l'un utilisant les scores basés sur les probabilités (soft scores) et l'autre basé sur les distances. Alors que le calcul des distances est simple, la formulation des soft scores dépend des méthodes de clustering. Nous avons étudié et formulé différentes manières de calculer l'adaptation pour chaque ensemble d'algorithmes de clustering, comme le clustering basé sur les centroïdes, le clustering profond ou le clustering basé sur la densité. Les expériences montrent une amélioration significative par rapport aux travaux précédents de post-traitement, car notre méthode utilise mieux les informations dans le processus de clustering. De plus, nos qualités de clustering pour les différents types de contraintes sont compétitives par rapport aux meilleurs algorithmes pour chaque type. Dans le même temps, le temps de calcul est bien inférieur à celui des autres méthodes en cours de processus. Un certain nombre de scénarios pratiques de combinaison de différents types de contraintes sont testés pour prouver la capacité de notre cadre à gérer des contraintes multiples sur un grand nombre de points de données, ce dont les modèles déclaratifs précédents sont incapables.

Le contexte du deuxième cadre Récemment, le réseau neuronal a gagné en popularité à la fois dans la représentation des données et dans l'intégration des

connaissances. La représentation des données est apprise automatiquement par la machine grâce à des structures neuronales uniques telles que le réseau neuronal convolutif (CNN) et l'auto-encodeur. Ainsi, le clustering avec des réseaux neuronaux (deep clustering) est devenu l'état de l'art pour de nombreux ensembles de données dans divers domaines. Cependant, pour obtenir les meilleures performances pour un ensemble de données, il est nécessaire d'ajouter des connaissances humaines pertinentes pour les données. L'une des directions les plus prometteuses de l'intégration des connaissances est d'exprimer les connaissances dans un langage formel tel que la logique du premier ordre, puis de les apprendre avec une perte régularisée. Cette approche a normalisé la forme des connaissances, ce qui permet d'intégrer différents types de connaissances et même des connaissances complexes dans la machine. Il y a eu quelques travaux [104, 102] pour appliquer cette approche au problème de classification, mais elle n'a pas seulement été étudiée et adaptée pour le clustering contraint.

Le deuxième cadre est le premier cadre de clustering sous contraintes à intégrer des contraintes d'experts en tant que connaissances logiques dans l'architecture des réseaux neuronaux. Nous proposons deux formulations, toutes deux basées sur les sorties neuronales d'un modèle de clustering profond. Ainsi, le cadre peut être appliqué à différentes architectures neuronales et entraîné avec ou sans processus de clustering profond. Dans la première formulation, toutes les variables logiques sont directement liées aux neurones. La condition de clustering doit donc être appliquée. La deuxième formulation relie les sorties à un ensemble de phrases de sorte que le contexte de clustering est pris en charge et que les formulations ne concernent que les contraintes d'experts. Ensuite, les deux formulations sont traduites directement en une perte sémantique. Par conséquent, les méthodes sont indépendantes de la représentation de la connaissance. En d'autres termes, les contraintes expertes présentées, que ce soit sous la forme normale disjonctive (DNF), la forme normale conjonctive (CNF) ou une forme arbitraire, produisent le même résultat. Notre utilisation des Diagrammes de Décision Sententiels (DDS) réduit la complexité du calcul des pertes et permet l'intégration de contraintes complexes telles que les contraintes d'implication.

Une autre contribution importante est de formuler et d'appliquer de nouveaux types de contraintes au problème du regroupement sous contrainte. À partir

de concepts ou d'idées initiales dans la connaissance du domaine, comme les comparaisons relatives, l'équité et le partage des caractéristiques, nous construisons la formulation des contraintes de triplets logiques, de l'équité combinée et de la contrainte de cardinalité des propriétés. Les expériences montrent que l'intégration de ces contraintes permet d'améliorer le résultat du clustering et d'atteindre nos propriétés cibles.

Des idées pour de futures directions

Les directions pour les travaux futurs sont :

La construction de la matrice d'allocation. Le premier objectif du cadre de post-traitement est d'améliorer la construction de la matrice d'allocation fractionnelle de cluster (CFAM). Dans la méthode que nous proposons pour calculer la CFAM, les paramètres ayant un impact sur le degré de flou, tels que le degré de liberté v dans la distribution t de Student, sont fixés pour tous les ensembles de données. Il est important de voir comment cette valeur peut être choisie par des critères internes sur le clustering : la division maximale, la densité du cluster, l'indice de Davies-Bouldin.

Différentes façons de modéliser le problème du post-traitement. Le deuxième objectif pour le cadre postprocessus est d'améliorer l'efficacité de l'algorithme. Il serait intéressant de trouver différentes manières de modéliser le clustering contraint à travers différentes variables et équations. Un exemple pourrait être un cadre plus souple qui permet à certaines contraintes de ne pas être satisfaites.

La généricité de notre deuxième cadre . Pour le deuxième cadre, notre premier objectif est de consolider la généricité du travail. Nous souhaitons expérimenter des modèles de clustering plus profonds sur différents jeux de données provenant de différents domaines. Ensuite, nous pourrons analyser les différences de performance avec d'autres méthodes afin d'adapter notre cadre et notre schéma de formation à chaque scénario spécifique.

L'expression de la seule contrainte de clustering, à savoir qu'un point doit être placé dans un seul cluster, permet de placer toutes les contraintes dans un seul cadre.

Traiter les contraintes comme si elles étaient indépendantes. Ceci a également l'avantage de donner un impact plus important de chaque contrainte sur la perte.

Autres méthodes d'intégration des connaissances. En raison de la complexité du comptage de modèles pondérés (WMC), nous avons dû imposer certaines restrictions à notre cadre, comme le traitement séparé des contraintes. Notre deuxième objectif pour le clustering profond avec connaissances est d'utiliser d'autres méthodes d'intégration des connaissances. Pour l'apprentissage avec la représentation logique, la satisfaction des formules peut être apprise par un réseau d'intégration de graphes au lieu du calcul intraitable avec WMC. Une représentation potentielle des contraintes d'experts basée sur des équations ou des nombres entiers mérite également d'être étudiée plus avant.

Abstract

Clustering is one of the essential topics in data mining. Although it is designed to work in a fully unsupervised way, its application in real-world data is often regulated by expert knowledge. Constrained clustering (a generalization of semi-supervised clustering) aims to exploit this knowledge during the clustering task. Knowledge is often expressed by a set of constraints and can take various forms. It can be relations between instances; for example, two points must be or cannot be in the same cluster (must-link or cannot-link constraints). It can also be some conditions on the clusters, such as their cardinality, their diameter, etc. It can also express some more semantic requirements on the clustering as for instance, getting fair clusters.

Constraints could be integrated at different steps of the clustering process: pre-processing (for instance, learning a new metric taking into account the constraints), during the clustering process or in a post-processing step. Most work on Constrained Clustering integrates constraints during the clustering, and they can be divided into two main streams: modifying existing clustering algorithms to handle some specific types of constraints / modeling the problem in declarative frameworks, such as Integer Linear Programming, SAT or Constraint Programming. As far as we know, a single work has considered the problem of post-processing the result of a hard clustering algorithm for integrating new constraints.

In this thesis, our contributions are threefold: (1) a post-process declarative method for adapting the output of a clustering algorithm to satisfy constraints; (2) a framework for integrating any kind of constraints during Deep Clustering; (3) the definition and/or formulation of new kinds of constraints.

Concerning our first contribution, the originality of our approach is to consider an allocation matrix that gives the scores of assigning points to each cluster and to find the best hard clustering satisfying all the constraints. The problem (constraints and objective function) is formulated as an Integer Linear Program. Experiments on different constraint types and combinations of them show that this method is efficient while being competitive in terms of Normalized Mutual Information and Unsupervised Accuracy w.r.t. some state-of-the-art constrained clustering methods.

Recently, clustering based on Deep Neural Networks has been actively studied, and new methods achieve state-of-the-art performances because of their ability to extract a low-dimensional representation. The learning method for multiple hid-

den layers is gradient descent and backpropagation. For introducing constraints during the learning step, it is necessary to have a loss function penalizing the non-satisfaction of constraints. Current work introduces a different loss for each kind of constraint, including pairwise, triplet, and cluster size constraints. To the best of our knowledge, we are the first ones to propose a framework for dealing with universal expert constraints in neural-based clustering. The genericity is obtained by formulating the constraints in propositional logic, defining two versions of semantic loss, and computing them through Weighted Model Counting. Experimental results on well-known datasets show that our approach is competitive with other constraint-specific methods while being general.

Besides these two generic methods, we have defined and formulated new kinds of constraints: fairness, neighborhood constraint, attribute level constraint, and cluster-overlap constraint.

Acknowledgments

I would like to thank all the people without whom this thesis work would not have been possible. My gratitude goes first to Christel Vrain and Thi-Bich-Hanh Dao, my thesis director and co-supervisor, for their wise advice and devoted efforts in helping me improve my research skills. I want to show my sincere appreciation for their constant feedback when my explanations in the thesis are unclear or even incomprehensible. I also feel lucky to have been a team member of the LIFO lab, which gives me frequent opportunities to present my ideas and receive valuable questions. I want to thank all my colleagues, including Matthieu Exbrayat, Marcílio Pereira de Souto, Sylvie Billot, and Antoine Guillaume, who have helped me with all my troubles in the lab. Another big thank you goes to Naly Raliravaka for his technical support and Isabelle Renard for all her help in all administrative matters.

At this point, I would also like to thank Dino Ienco, Tias Guns, Antoine Cornuejols, Pierre Marquis for taking time out to review the thesis and being part of the thesis committee. Their many comments and suggestions greatly improved the quality of this thesis.

I would like to acknowledge the generous funding from the Centre-Val de Loire region, without which this thesis would not have been possible.

Finally, I owe all I am today to my parents' hard work, sacrifices, unconditional love, and support on all the significant developments of my life.

Introduction

Context

In our digital era, data has become a valuable resource to our life. Therefore, many algorithms have been developed to analyze the data better. Clustering analysis is the first and vital step to understand the nature of the structure of the data. Clustering aims at grouping the data instances into clusters based on some dissimilarity/similarity measures. Clusters are usually formed by minimizing the dissimilarities inside the clusters and/or maximizing the dissimilarities between clusters.

There are two types of solutions for clustering: traditional algorithms such as K-means[62], spectral clustering[86, 72] and neural-based models for examples Deep Embedded Clustering (DEC)[101], Clustering using CNN[17]. Traditional clustering gives a fixed objective to clustering the data and a method to achieve it. In contrast, the neural-based models learn the representation of data and predict clustering structure through a network of neurons. Therefore, the results of traditional clustering are understandable and follow its objective; meanwhile, the neural-based approach is more flexible and can handle complex/high-dimensional data through the learning process. However, there does not exist a single best clustering method as it may give good results with some types of data and bad results with other data.

To better improve the clustering results, cluster analysis has integrated the background (domain) knowledge about the data. The domain knowledge is usually given in the form of expert constraints. In the early days, most of the research has been put on simple constraint types such as pairwise constraints[96, 13, 99]. Pairwise constraints compose must-link constraints, which require the two data points to belong to the same cluster, and cannot-link constraints that enforce the two points assigning to different clusters. Later on, the research community has increased their focus on developing a declarative framework for the constrained clustering problem. This approach could be less efficient than constraint-specific algorithms, but it is more flexible to change the clustering process and easy to incorporate new constraints. Babaki et al.[4] propose an exact method to find a clustering that satisfies all expert constraints with minimum the within-cluster sum of squares (WCSS). The framework are based on Integer Linear Programming, which can integrate must-link, cannot-link, and any constraints with the anti-monotone property. Dao et al.[22] propose another framework based on Constraint Programming. Clustering objectives could be selected amongst WCSS, the maximum diameter of clusters, the minimum split between clusters. The framework can handle pairwise constraints as well as cluster-level constraints such as cardinality constraints, density constraints. The fatal weakness of this approach is the solution complexity; thus, it can not handle more than a thousand constraints.

Recently, machine learning with neural networks has been a new frontier for knowledge integration. Unlike the previous method, where we need to enforce all of the knowledge in the form of expert constraints, the deep learning model can learn

using abstract forms or from few examples. Although there is a huge potential in the future for a pure knowledge-based model, most applications with knowledge integration are data-based models with an infusion of knowledge. They design the neural architecture based on characteristics of data, and integrating knowledge through the learning process (no modification to the neural network).

The most direct way to integrate expert constraints is to design a loss function for each type of constraint. The loss can be used to optimize both clustering problems and constraint satisfaction, such as the Optimal transport[36]. As another option, the loss focus only on expert constraints[42, 46, 109]. In both cases, the loss can be fine-tuned based on the neural network, the input dataset, but it is obscure or even impossible to adapt for other constraint types.

For a more generalized framework, the knowledge is often presented in formal languages such as first-order logic, mathematical equations. Then, these symbolic variables are linked with the output of the neural structures in order to regulate the learning process. This approach has been applied to the classification problem[44, 104], the visual relation prediction problem[102], the structured (physics) prediction problem[90]. Unfortunately, there exists no neural-based framework to incorporate constraints as knowledge for a constrained clustering problem.

Contribution

There are three main contributions of this thesis:

- **Constrained Postprocess with Clustering Score** First, it allows users to choose a suitable clustering algorithm to compute the cluster allocation matrix. Then, the declarative framework, based on ILP, allows adding different constraint types before optimizing to find the best clustering satisfying all expert constraints. The results shows that the framework are competitive to other constraint-specific methods in the clustering quality and much more efficient in the calculation.

- **Deep Clustering with Logical Knowledge** The framework can integrate knowledge in the logical form for any deep clustering model. We have defined two formulations to represent the expert constraints adapted for the clustering setting. We also use Sentential Decision Diagrams (SDD) and machine learning techniques to improve the efficiency of the learning process. Experiment results are comparable with the state-of-the-art; moreover, it can integrate high-level constraints such as implication constraints and prove the effect of knowledge learning by satisfying unseen constraints.

- **New constraint types and their applications** Both of our framework allows us to express attribute level constraints and the combined fairness constraint. The attribute level constraints allows users to state a clustering condition on a group points which share the same attribute. The combined fairness constraint are based on the individual fairness and the group fairness criteria.

Thesis organization

This dissertation has the following structure:

- Chapter 1 gives a background of clustering and constrained clustering problems. We present different approaches and several algorithms in detail for solving the two tasks.
- Chapter 2 is devoted to the basic foundation of deep learning and application to the clustering problem.
- Chapter 3 brings an overview of knowledge integration in a neural network before presenting methods and related works of deep constrained clustering.
- Chapter 4 presents our first framework for solving the problem of constrained clustering through the postprocess.
- Chapter 5 presents our second framework, which is able to learn expert constraints in the form of logical formulae in any deep clustering model.
- In the conclusion, the thesis is summarized, and future developments are discussed.

Chapter 1
Constrained Clustering

Clustering is an unsupervised task aiming to group data into similar groups. The clustering process has been a crucial step in data analysis and has been practiced in many circumstances. The problem extends into constrained clustering, where expert knowledge is integrated into the clustering process in the form of user constraints. These constraints help the clustering process acquiring a more meaningful result.

This chapter reviews traditional views of the constrained clustering problem as a combinatorial problem and gives several classical solutions (not relying on neural networks). The first type of solution, which is called the algorithm-specific approach, tries to modify original clustering algorithms. Then, we continue to describe the declarative approach. We will introduce constrained clustering methods using neural networks in Chapter 3.

The structure of this chapter is as follows. First, in Section 1.1, we present the main notions and different optimization criteria for modeling the clustering problem. Some of the most well-known algorithms for clustering problem are fully described in Section 1.2. Then, Section 1.3 will introduce several types of constraints commonly used in semi-supervised learning. Finally, the main approaches for constrained clustering are given in Section 1.4 and Section 1.5 with examples and characteristics.

1.1 Clustering Problems

1.1.1 Notations

The following notations and their definitions are used throughout this thesis:

- An observation (point, or feature vector, or data) $\mathbf{x}$ is a single data item used by the clustering algorithm. It is typically represented by a vector of l scalars: $\mathbf{x} = (x_1, ..., x_l)$.
- A feature (or an attribute) is a single measurable/observable property or characteristic of a point $\mathbf{x}$. The value of feature i for the point $\mathbf{x}$ is denoted by x_i.
- l is the dimensionality of the data space (input data).
- n is the number of points and k is the number of clusters.
- A dataset is denoted by $\mathscr{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$.

- A *partition* is a set of k non-empty non-overlapping clusters (denoted as $\mathscr{C}_1, ..., \mathscr{C}_k$) containing all the objects of the dataset. We use *clustering* and *partition* interchangeably because other clustering frameworks such as overlapping clustering, fuzzy clustering, ... are not considered in this thesis. The requirement for the output to be a partition will be called in Section 5.2 *clustering constraints*. Another representation of a partition is an index array $\mathbf{p} = (p_1, ..., p_n)$, where the point i belongs to the cluster $\mathscr{C}_{p_i}$ ($i \in \mathscr{C}_{p_i}, \forall i \in [1, n]$). [1]

- A *dataset label* (ground-truth label) is a classification result given by human experts. The labels are denoted by $\mathscr{L}_1, ..., \mathscr{L}_k$. The label of $\mathscr{X}$ is defined as $\mathbf{y} = (y_1, .., y_n)$ where $\mathbf{x}_i \in \mathscr{L}_{y_i} \forall i \in [1, n]$. It is important to note that the labels are unknown or partially known in most applications. The ground truth of the dataset is only used for generating constraints in the experiments and evaluating clustering methods.

- d (resp. s) represents a *dissimilarity (resp. similarity) measure* on the feature (attribute) space used to quantify the dissimilarity (resp. similarity) of two points. In this report, D_{ij} will be used in short for $d(\mathbf{x}_i, \mathbf{x}_j)$, and S_{ij} is for $s(\mathbf{x}_i, \mathbf{x}_j)$.

- An *expert constraint* (*user constraint* or in short, *constraint*) is a requirement on clusters or a set of data points. It represents some prior knowledge on the clustering outputs. This notion will be clearly defined in Section 1.3.

- $\mathbb{C}$ is the constraint set that contains all constraints on the dataset $\mathscr{X}$.

- We will use $\mathbf{p} \models \mathbb{C}$ if the partition $\mathbf{p}$ satisfies all constraints in the constrained set $\mathbb{C}$

a. Properties of similarity/dissimilarity measure

The range of similarity measure often is from 0 to 1 where $S_{ii} = 1 \forall i \in [1, n]$. On the contrary, the dissimilarity value is non-negative and equals 0 if the two points are the same.

- Non-negativity: $D_{ij} \geq 0, \forall i, j \in [1, n]$
- Identity : $D_{ij} = 0 \leftrightarrow i = j$

For some algorithms (for instance, k-means), d must be a distance, that must satisfy the additional properties [80]:

- Symmetry: $D_{ij} = D_{ji}, \forall i, j \in [1, n]$
- Triangle inequality: $D_{ij} \leq D_{it} + D_{tj}, \forall i, j, t \in [1, n]$

[1] In this thesis, $[1, n]$ denotes the set of integers from 1 to n.

1.1.2 Types of clustering

Clustering can be specific further as:

- *Hard clustering*: Each point belongs to a single cluster.
- *Fuzzy (Soft) clustering*: Each point has various degrees of belonging to several clusters. The soft clustering are often resented by a matrix of membership scores $\mathbf{S} = \{S_{ij}, \forall i \in [1, n], \forall j \in [1, k]\}$ where $S_{ij} \in \mathbb{R}_{[0,1]}$. $S_{ij} = 1$ if the point i certainly belongs to $\mathscr{C}_j$ and the lesser S_{ij} is, the lower membership of i in cluster $\mathscr{C}_j$.
- *Hierarchical clustering*: A sequence of partitions is built and organized into a hierarchy.

In the context of this thesis, we focus on the methods which produce a hard clustering as the final output. Soft clustering could be used as an intermediate result in a clustering algorithm.

The clustering objective is either minimize the intra-connectivity between points within a same cluster or maximize the inter-connectivity between points from different clusters.

Finally, a clustering problem can be formulated as follows:

$$\begin{aligned} &\arg\min_{\mathbf{p}} intra - connectivity(\mathscr{C}) \\ \text{OR: } &\arg\max_{\mathbf{p}} inter - connectivity(\mathscr{C}) \end{aligned} \tag{1.1}$$

such that:

- $\forall i \in [1, k] : \mathscr{C}_i \neq \emptyset$
- $\forall i, j \in [1, k] \wedge i \neq j : \mathscr{C}_i \cap \mathscr{C}_j = \emptyset$
- $\cup_i \mathscr{C}_i = \mathscr{X}$

Although not every clustering algorithms has well-defined objective functions to optimize, the partition always follows some specific clustering criteria. We are going to give more details about these clustering criteria in the next section.

1.1.3 Criteria

The clustering task aims to find clusters, which have high intra-class similarity and low inter-class similarity. However, we need specific measures to evaluate the quality of the clustering.

In this section, we list a number of criteria for clustering. Some clustering algorithms also use these evaluations as objective functions to find the clustering. In addition, in Section b., the methods for evaluating the quality of clustering using ground truth are presented.

a. Criteria without ground-truth

The split and cut of a cluster are putting an emphase on the separation between clusters,

- The split of a cluster $split(\mathscr{C}_h)$ is the minimum distance between a point of this cluster $\mathscr{C}_h$ and points belonging to other clusters:

$$split(\mathscr{C}_h) = \min_{\mathbf{x_i} \in \mathscr{C}_h, \mathbf{x_j} \notin \mathscr{C}_h} d_{ij} \tag{1.2}$$

  - The split of a partition is the minimum split of its clusters. Clustering objective is often required to find partition such that the maximum split of a partition.

$$\arg\max_{\mathbf{p}} split(\mathbf{p}) = \arg\max_{\mathbf{p}} \left(\min_{h} split(\mathscr{C}_h) \right) \tag{1.3}$$

  - Another criterion is maximizing the sum splits of a partition.

$$\arg\max_{\mathbf{p}} sum\_split(\mathbf{p}) = \arg\max_{\mathbf{p}} \left(\sum_{h} split(\mathscr{C}_h) \right) \tag{1.4}$$

- The cut of a cluster $cut(\mathscr{C}_h)$ is the sum of the distances between points of this cluster and points belonging to other clusters:

$$cut(\mathscr{C}_h) = \sum_{\mathbf{x_i} \in \mathscr{C}_h} \sum_{\mathbf{x_j} \notin \mathscr{C}_h} d_{ij} \tag{1.5}$$

  The min-cut objective function for clustering is expressed as follows:

$$\arg\min_{\mathbf{p}} cut(\mathbf{p}) = \arg\min_{\mathbf{p}} \left(\sum_{h \in [1,k]} cut(\mathscr{C}_h) \right) \tag{1.6}$$

The most used criterion for assessing the homogeneity of a partition is the within cluster sum of squares $wcss(\mathbf{p})$. It requires to be able to compute the centroid of a cluster. The value is the sum of the sum of the squared Euclidean distances between each object with his centroid μ.

$$wcss(\mathbf{p}) = \sum_{h=1}^{k} \sum_{\mathbf{x_i} \in \mathscr{C}_h} \|\mathbf{x_i} - \mu_h\|^2 \tag{1.7}$$

b. Criteria with ground-truth

The unsupervised accuracy and normalized mutual information are commonly used measures for dataset with ground-truth labels.

Recall from Section 1.1: data is labelled with k labels and the labels of a point is denoted by $\mathbf{y} = (y_1, ..., y_n)$, the clustering output is $\mathbf{p} = (p_1, ..., p_n)$.

The first metric is *unsupervised clustering accuracy* (ACC):

$$ACC = max_m \frac{\sum_{i=1}^{n} 1[y_i = m(p_i)]}{n} \tag{1.8}$$

where y_i is the ground-truth label, c_i is the cluster assignment generated by the algorithm, and m is a one-to-one mapping function from assignments to labels.

The Hungarian algorithm [56] is often used in order to find the maximal matching.

The second metric is *Normalized Mutual Information* (NMI) :

$$NMI = \frac{I(\mathbf{y}, \mathbf{p})}{\frac{1}{2}[H(\mathbf{y}) + H(\mathbf{p})]} \tag{1.9}$$

where I is the mutual information metric and H is entropy. Since $\mathbf{y}$ and $\mathbf{p}$ are discrete variables, the mutual information is calculated as a double sum using the joint ($p(y_i, p_j)$) and marginal ($p(y_i)$, $p(p_j)$) probabilities:

$$I(\mathbf{y}, \mathbf{p}) = \sum_{i=1}^{n} \sum_{j=1}^{n} p(y_i, p_j) \log \left(\frac{p(y_i, p_j)}{p(y_i)p(p_j)} \right) \tag{1.10}$$

1.2 Clustering Techniques

In the previous section, we have introduced different objective functions for clustering. In this section, we will present the main approaches to achieve these objectives. Then, we select K-means and Spectral clustering algorithm as the two main examples to give detail steps and their variants. These algorithms have been studied further and adapted to constrained clustering problem [96, 91, 98].

Another clustering approach is to deal with data that do not have conventional measures, such as data with categorical attributes and time-series data. Those topics are not covered due to the scope of this thesis.

1.2.1 Major approaches to clustering

The relations between clusters can be in one of the three types. A cluster in hierarchical clustering could be a subset or a superset of another clusters. The clusters could overlap in fuzzy clustering, while a hard clustering produces non-overlapping clusters. As mentioned earlier, this thesis focuses on hard clustering (partition). An overview of hierarchical clustering and fuzzy clustering could be found in [69] and [66].

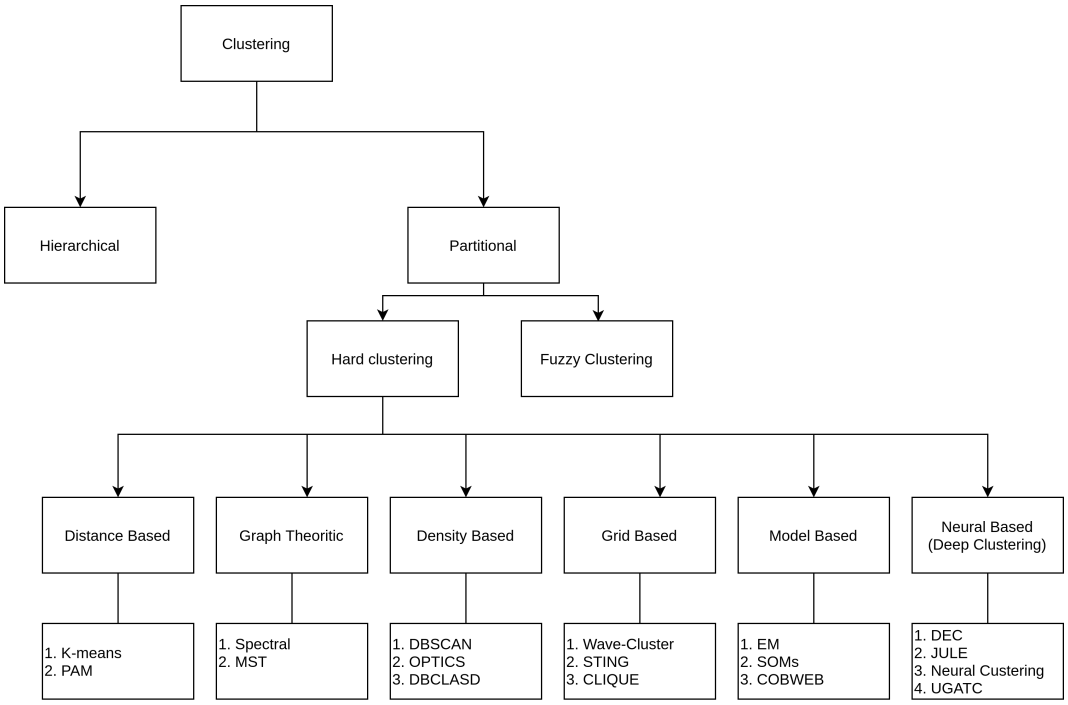

Figure 1.1: Taxonomy of clustering approaches

Because there is no precise definition of a cluster, the clustering method could use different objectives as mentioned in Section 1.1.3. Moreover, the clustering data are varied in terms of structure and characteristics. Therefore, many clustering algorithms are proposed, and each has its own application and practice. The general techniques of clustering can be categorized as follows:

- **Distance based clustering**: These clustering algorithms try to solve an optimization function defined in terms of the distance between points within the same cluster. The most notable algorithms in this category are K-means[62], PAM[50], CLARA[49].

- **Graph based clustering**: The general approach of this type is as follows. First,is based on the construction of a graph representing the data points and their relation. Then, graph methods are used to find the optimal graph cut separating the points, hence obtain the partition. Spectral clustering (including various specific algorithms) [86, 72] and minimum spanning tree (MST) based clustering [62] are two examples of this type.

- **Density based clustering**: The density-based clustering methods find clusters using the idea that a cluster is a contiguous region of high density. Therefore, an advantage of this method is the ability to discover clusters of arbitrary shapes and less susceptible to outliers. The well-known algorithms of this approach are DBSCAN[33], OPTICS[1].

- **Grid based clustering**: A grid-based clustering method divides the space of data into grids. The data points are mapped to the corresponding cells on the

grid. Then the cluster is performed on the grid instead of the points. Hence, the performance of a grid-based method depends on the size of the grid. This method has a fast processing time, but it could suffer from irregular data patterns.

- **Model based clustering**: This method gives some predefined mathematical models. Then, the model is fitted by the provided data. The assumption underlying this method is that data is generated by a mixture of hidden probability distributions. There are two sub directions for this approach: statistical and neural network. The statistical approach assumes that the data points are drawn from a specific distribution. MCLUST[85] is the most well-known algorithm for this approach. It assumes data is generated from Gaussian distributions and uses expectation-maximization (EM) to find the local optimum of cluster distribution. Each instance is assigned to the cluster with the highest probability. The neural network approach, called Deep Clustering, models the distributions through connected neurons. Deep clustering methods will be introduced in Section 2.2.

K-means and Spectral Clustering will be presented in the following sections because they have multiple extensions to handle expert clustering constraints.

1.2.2 K-means

K-means is a greedy (heuristic) algorithm for finding a local minimum for the sum of square distance between points $\mathbf{x}_i$ and the cluster centers μ.

The main steps of K-means algorithm are [62]:

1. Select an initial set of k points (cluster centers).
2. Generate a partition $\mathbf{p}$ by assigning each point i to its closest cluster center μ_h
$$\forall i \in [1,n] : p_i = \arg\min_{h \in [1,k]} \|\mathbf{x}_i - \mu_h\|^2 \tag{1.11}$$
3. Compute the new cluster centers as the means of all points belonging to this cluster.
$$\forall h \in [1,k] : \mu_h = \frac{\sum_{i \in [1,n]:\mathbf{p}_i = h} \mathbf{x}_i}{|\{i \in [1,n] : \mathbf{p}_i = h\}|} \tag{1.12}$$
4. Terminate if there is no change in the partition. If not, return to step 2.

Although K-means remains one of the most popular clustering algorithms, this method has several limitations. First, K-means requires the attributes in $\mathbf{x}_i$ to be numerical. Second, the performance of K-means is affected by the initial cluster centers even with improvements such as K-means++ [2]. Last, the clusters are not well represented by the cluster centers because these centroids do not belong to dataset $\mathscr{X}$.

K-Medoids clustering and PAM When we require the cluster centers to be points in $\mathscr{X}$, the problem is called K-medoids clustering . The Partitioning Around Medoids (PAM) [50] algorithm solves this problem by giving slight changes to K-means algorithm. At step 3, cluster centers are updated by:

$$\forall h \in [1,k] : \mu_h = \underset{\mathbf{x}_i : \mathbf{x}_i \in \mathscr{C}_h}{\arg\min} \sum_{\mathbf{x}_j \in \mathscr{C}_h} \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

1.2.3 Spectral Clustering

Spectral clustering is based on a graph representation of data. There are several ways of building a graph. It can be a complete graph constructed from the distance matrix between two points of data. Alternatively, it can be a neighborhood graph where the points can be connected to their nearest neighbors.

Notation Let consider a graph $G = (\mathscr{X}, E)$ where $\mathscr{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$ is the set of vertices.

The edges of G is presented by the adjacency matrix $S = \{i, j \in [1,n] : S_{ij}\}$. The weight $S_{ij} > 0$ is the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$.

Let $D_i = \sum_{j \in [1,n]} S_{ij}$ be the degree of node i.

The volume of a set $A \subset \mathscr{X}$ be $VolA = \sum_{\mathbf{x}_i \in A} D_i$. A cut between two sets A and B is $Cut(A,B) = \sum_{\mathbf{x}_i \in A, \mathbf{x}_j \in B} S_{ij}$

General steps Spectral clustering algorithm consists of 3 steps:

- Preprocessing: Construct the matrix S which is the adjacency matrix from the pairwise distance d_{ij}.
- Spectral mapping: First, we compute k eigenvectors of S. Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the eigenvectors as columns. Then, each data point i is represented based on the i-th row of U.
- Postprocess/Grouping: Applying a simple clustering algorithm on the new representation.

Preprocessing There are three main methods for contructing the graph:

- The ϵ-neighborhood graph: $S_{ij} = 1$ if $D_{ij} < \epsilon$ else $S_{ij} = 0$.
- k-nearest neighbor graphs: $S_{ij} = 1$ if $\mathbf{x}_i$ is among the k-nearest neighbors of $\mathbf{x}_j$ or $\mathbf{x}_j$ is among the k-nearest neighbors of $\mathbf{x}_i$; $S_{ij} = 0$, otherwise.
- The fully connected graph: We given function to convert d_{ij} into S_{ij}. An example is the Gaussian similarity function $S_{ij} = exp(-D_{ij}^2/(2\sigma^2))$ where the parameter σ controls the width of the neighborhoods.

After the preprocessing step, the Shi and Malik algorithm and the Ng, Jordan, and Weiss (NJW) algorithm were chosen to introduce. While the first method cuts the graph multiple times until getting k clusters, the second method made the new representation and using K-means to have the final clustering.

The Shi and Malik algorithm This algorithm [86] is a heuristic algorithm intended to minimize the Normalized Cut.

$$NCut(A,B) = Cut(A,B)\left(\frac{1}{VolA} + \frac{1}{VolB}\right) \tag{1.13}$$

The algorithm is as follows:

1. Compute $P = D^{-1}S$ where D is the distance matrix.

2. Let $1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be the eigenvalues of P and $v^1, v^2, \cdots, v^n$ the corresponding eigenvectors. Compute v^2.

3. Use v^2 to bipartition the graph by finding the splitting point such that Ncut is minimized.

4. Decide if the current partition should be subdivided by checking the stability of the cut, and make sure Ncut is below a prespecified value.

5. Repeat the process until obtain a k-partition.

The Ng, Jordan and Weiss (NJW) algorithm While the Shi and Malik algorithm [72] use multiple cuts to obtain the partition, the NJW uses K-means to cluster the new representation of data.

The algorithm is as follows:

1. Let D_{diag} is a diagonal matrix whose (i,i)-element is the sum of i-th row from S.

2. Compute $L = D_{diag}^{-1/2} S D_{diag}^{-1/2}$.

3. Let $1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ be the k largest eigenvalues of L and $u^1, u^2, \cdots, u^k$ the corresponding eigenvectors. All eigenvectors are normalized to have unit length. Form the matrix $U = [u_1 u_2 \cdots u_k]$

4. Form the matrix V from U by re-normalizing each rows to have unit length $(V_{ij} = U_{ij}/\sqrt{\sum_j U_{ij}^2})$

5. Consider each row of V as a new representation of points. Clustering them using K-means.

1.3 Expert Constraints for Clustering

In order to better model the clustering task, and also satisfy external requirements (e.g. cluster-size constraints), constraints are added. The problem is called Constrained Clustering and aims at finding a clustering that satisfies (partly or totally) all the constraints.

The constraints can be classified into cluster-level constraints, specifying requirements on clusters, or instance-level constraints, specifying requirements on a small number of instances.

1.3.1 Instance-level constraints

Pairwise Constraints between two instances $\mathbf{x_i}$, $\mathbf{x_j}$ have two main forms:

- a must-link constraint, denoted by $ML(\mathbf{x}_i, \mathbf{x}_j)$, requires both instances $\mathbf{x}_i$ and $\mathbf{x}_j$ to be in the same cluster.
- a cannot-link constraint, denoted by $CL(\mathbf{x}_i, \mathbf{x}_j)$, expresses that these two objects must not be in the same cluster.

Such constraints come from the knowledge of domain experts or even common sense. They also can be obtained from partially labeled observations. Besides ML/CL, Together/Apart are notations used for logical formulas. They are logical statements that can be True or False. So, if there is must-link constraint between $\mathbf{x}_i$ and $\mathbf{x}_j$ ($ML(\mathbf{x}_i, \mathbf{x}_j)$) then $Together(\mathbf{x}_i, \mathbf{x}_j)$ must be true and $Apart(\mathbf{x}_i, \mathbf{x}_j)$ is must be false.

$$Together(\mathbf{x}_i, \mathbf{x}_j) \equiv \exists h \in [1, k] : \mathbf{x}_i \in \mathscr{C}_h \wedge \mathbf{x}_j \in \mathscr{C}_h$$
$$Apart(\mathbf{x}_i, \mathbf{x}_j) \equiv \forall h \in [1, k] : \neg(\mathbf{x}_i \in \mathscr{C}_h \wedge \mathbf{x}_j \in \mathscr{C}_h)$$

Triplet & Relative Triplet and relative constraint are two main types of constraints concerning relations between three points. A triplet constraint among 3 instances $\mathbf{x_a}$, $\mathbf{x_p}$ and $\mathbf{x_n}$ (respectively called anchor, positive and negative instances) means that $\mathbf{x_a}$ is more similar to $\mathbf{x_p}$ than to $\mathbf{x_n}$ [57]. This definition is not giving a direct relation to clustering. So, initial algorithms often have a mechanism for adjusting the metric distance to follow it and then run a clustering algorithm based on the new distance between instances.

Let d_L denote the learned metrics for this data, constraint $Triplet(\mathbf{x_a}, \mathbf{x_p}, \mathbf{x_n})$ is expressed as:

$$d_L(\mathbf{x_a}, \mathbf{x_p}) < d_L(\mathbf{x_a}, \mathbf{x_n}) \tag{1.14}$$

Similar to triplet constraints, a relative constraint between a, b, and c instances represents that a and b are the most similarity pair or $d_L(a, b)$ is the smallest distance.

$$d_L(\mathbf{x_a}, \mathbf{x_b}) < \min(d_L(\mathbf{x_b}, \mathbf{x_c}), d_L(\mathbf{x_c}, \mathbf{x_a})) \tag{1.15}$$

Logical triplet constraints Given the fact that the two points within a cluster is more similar than the two belong to two different clusters, the distance information in a triplet constraint can be transfer to conditional cluster assignments. If instance a and instance n is in the same cluster ($Together(a, n)$), then instance a and instance p must be in the same cluster as well ($Together(a, p)$). The logic triplet constraints:

$$\begin{aligned} & Together(a, n) \implies Together(a, p) \\ \equiv & \neg Together(a, n) \vee Together(a, p) \\ \equiv & Together(a, p) \vee Apart(a, n) \end{aligned} \tag{1.16}$$

Logical relative constraints Applying the similar implication, if Together(a,c) or Together(b,c), then Together(a,b). The logic relative constraint:

$$\begin{aligned} & Together(a, c) \vee Together(b, c) \implies Together(a, b)) \\ \equiv & \neg(Together(a, c) \vee Together(b, c)) \vee Together(a, b) \\ \equiv & Together(a, b) \vee (Apart(a, c) \wedge Apart(b, c)) \end{aligned} \tag{1.17}$$

1.3.2 Cluster-level constraints

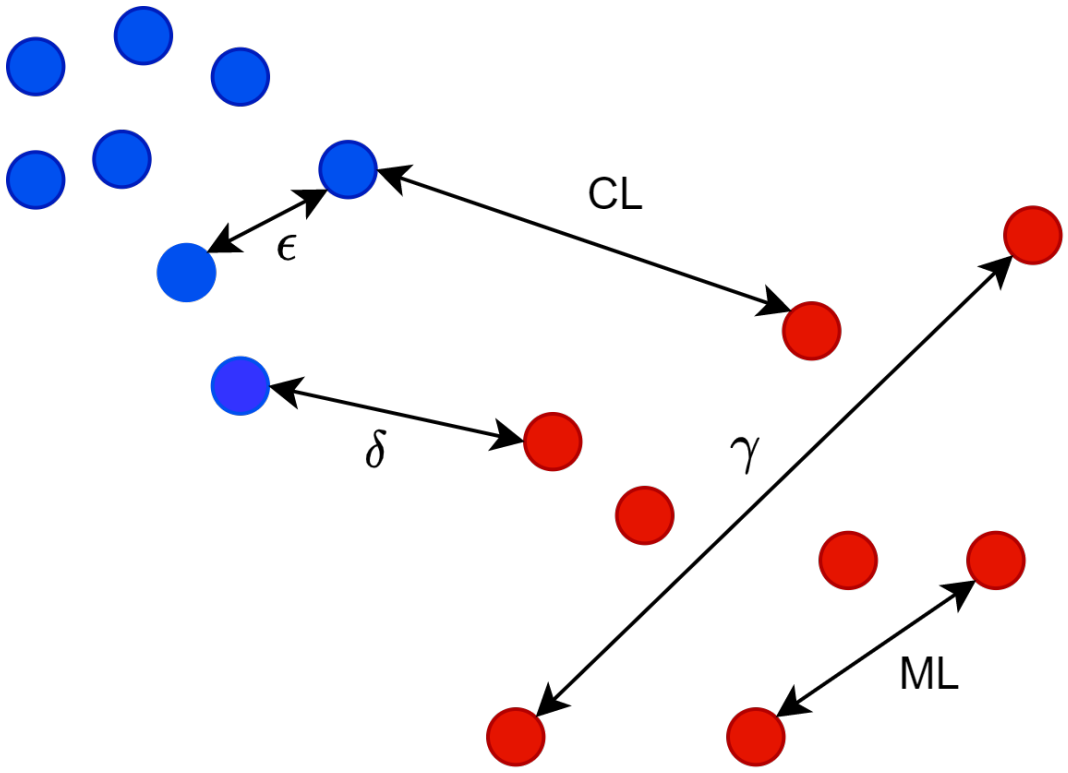

Figure 1.2: Examples of ML, CL, diameter (γ), split (δ) and neighborhood (ϵ) constraints

The cluster-level constraints describe conditions on the clusters. Here are three main type of constraints which have been stated in the literature:

- A cardinality constraint puts an upper/lower bound on the number of instances in each cluster. Balanced clustering constraint [15] is a special case, where it is required that the clusters are of approximately equal size.
- A geometric constraint gives an upper/lower bound on the diameter of each cluster or on the split between the clusters [24].
- A neighborhood constraint expresses the requirements on the neighborhood of each point.

Cardinality constraint

The cardinality constraint includes the following constraints:

- The minimum cluster size constraint requires that each cluster has a number of points greater than a given threshold α: $\forall h \in [1,k], |\mathscr{C}_h| \geq \alpha$
- The maximum cluster size constraint requires that each cluster has a number of points smaller than a given threshold β: $\forall h \in [1,k], |\mathscr{C}_h| \leq \beta$
- The balance constraint (ratio control) requires all clusters to have approximately the same size or in other terms that the fraction between the largest and the smallest clusters be greater than a given threshold θ: $\frac{\min_{i\in[1,k]}|\mathscr{C}_i|}{\max_{j\in[1,k]}|\mathscr{C}_j|} \geq \theta$

Geometric constraint

- A maximum diameter constraint gives an upper bound γ on the diameter of each cluster.
$$\forall j \in [1,k], \forall u,v \in \mathscr{C}_j : D_{uv} \leq \gamma \tag{1.18}$$
- A minimum split constraint requires that the clusters must be separated by at least δ.
$$\forall j,j' \in [1,k], j \neq j', \forall u \in \mathscr{C}_j, \forall v \in \mathscr{C}_{j'} : D_{uv} \geq \delta \tag{1.19}$$

Neighborhood constraint

- An ϵ-constraint states that each point i has in its neighbourhood of radius ϵ at least one other point in the same cluster
$$\forall j \in [1,k], \forall u \in \mathscr{C}_j \exists v \in \mathscr{C}_j : u \neq v \wedge D_{uv} \leq \epsilon \tag{1.20}$$
- An individual fairness constraint requires that the ratio of neighborhood belong to the same cluster as the center point must be greater than or equal α. Denote $\mathcal{N}(i)$ is the set of all neighbor points of point i, we have:
$$\forall j \in [1,k], \forall u \in \mathscr{C}_j : \frac{|\{v : v \in \mathcal{N}(u) \wedge v \in \mathscr{C}_j\}|}{|\mathcal{N}(u)|} \geq \alpha \tag{1.21}$$

1.3.3 Brief development of constrained clustering

Early on in the development of constrained clustering, the expert constraints are proposed independently and depend on the context of the clustering task. Therefore, related publications often integrate one or few types of constraints to a specific clustering algorithm. This is called Algorithm Specific Approach, and we present this approach in Section 1.4. The second approach is the declarative approach, which uses a general framework to deal with general expert clustering constraints. The details of this approach are presented in Section 1.5.

1.4 Algorithm Specific Approach

Constrained versions of a variety of different algorithms such as K-means [96, 91], EM [9], spectral [98] have been developed. Table 1.1 surveys classic well known constrained clustering algorithms with respect to the constraint types they can integrate.

Table 1.1: A brief survey of the algorithm specific approaches.

Initial Algorithm	**Algorithm**	**Constraint types**
K-means	COP-Kmeans [96]	Must-link/Cannot-link
	MPC-Kmeans [13]	Must-link/Cannot-link
	MSE-Kmeans [91]	Balanced Size Constraints
MCLUSTER (EM)	Pairwise-EM [9]	Must-link/Cannot-link
Spectral Clustering	Pairwise-Spectral Clustering [99]	Must-link/Cannot-link

1.4.1 COP-Kmeans

Instead of assigning each point to the closest cluster like Step 2 of K-means 1.2.2, each point is assigned to the closest cluster that does not violate any constraint. The main steps of COP-Kmeans and the function for checking constrained violation are presented in Algorithm 1.1.

Step 2 (Line 8-11 in Algorithm 1.1) can be seen a heuristics algorithm to find a partition $\mathbf{p}$ such that it obtains the minimum sum of distances from points to their cluster centers.

$$min_{\mathbf{p}:\mathbf{p}\models\mathbb{C}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_{p_i}\| \tag{1.22}$$

This approach is reasonable given that both optimizing the objective function and satisfying must-link/cannot-link constraints are intractable [24, 25]. In contrast, the authors in [91] gives the exact solution for this problem of 1.22 for cluster-size constraints.

Algorithm 1.1 COP-Kmeans Algorithm [96]

Input: Data set $\mathscr{X}$, must-link constraints $ML \subset \mathscr{X} \times \mathscr{X}$, cannot-link constraints $CL \subset \mathscr{X} \times \mathscr{X}$

Output: Clustering $\mathscr{C}$;

function VIOLATE-CONSTRAINTS(An instance : $\mathbf{x}_i$, a cluster: $\mathscr{C}_h$, must-link constraints ML, , cannot-link constraints CL)
    For each $j < i, (\mathbf{x}_i, \mathbf{x}_j) \in ML$: If $\mathbf{x}_j \notin \mathscr{C}_h$, return True.
    For each $j < i, (\mathbf{x}_i, \mathbf{x}_j) \in CL$: If $\mathbf{x}_j \in \mathscr{C}_h$, return True.
    Return False.
end function
Initialize cluster centers $\mu_h, \forall h \in [1, k]$.
repeat
    for $i := 1$ **to** k **do**
        Find $\mathscr{C}_h$ the closest cluster such that VIOLATE-CONSTRAINTS($\mathbf{x}_i$,$\mathscr{C}_h$, ML, CL) is false. If not exist $\mathscr{C}_h$, Return $Fail$
        Add $\mathbf{x}_i$ to $\mathscr{C}_h$
    end for
    Update cluster centers μ_h by averaging all of the points that have been assigned to it.
until Convergence
Return $\{\mathscr{C}_1, \cdots, \mathscr{C}_k\}$.

1.4.2 MSE-Kmeans

MSE-Kmeans [91] is a clustering method to find a partition satisfying the balanced cluster constraint. Similar to the approach of COP-Kmeans, the MSE-Kmeans tries to adjust Step 2 in K-means algorithm. In this step, it find the extract solution satisfying the balanced constraint.

Let denote A as the partition matrix, where $A_{ij} = 1$ indicates that the point i belongs to cluster j. The balanced cluster size constraint requires that each cluster contains from $\lfloor \frac{n}{k} \rfloor$ to $\lceil \frac{n}{k} \rceil$ instances.

$$\forall 1 \leq j \leq k : \lfloor \frac{n}{k} \rfloor \leq \sum_{i=1}^{n} A_{ij} \leq \lceil \frac{n}{k} \rceil \tag{1.23}$$

The initial problem of Step 2 is stated as follows:

$$\begin{aligned} & Minimize (1/n) \sum_{j=1}^{k} \sum_{i=1}^{n} A_{ij} \times \|\mathbf{x}_i - \mu_{p_i}\| \\ & \text{such that:} \\ & \forall 1 \leq j \leq k : \lfloor \frac{n}{k} \rfloor \leq \sum_{i=1}^{n} A_{ij} \leq \lceil \frac{n}{k} \rceil \\ & A_{ij} \in \{0, 1\}, 1 \leq j \leq k, 1 \leq i \leq n \end{aligned} \tag{1.24}$$

Then, by introducing auxiliary variables u_j, v_j, G_{ij}, the authors formulate the problem as Integer Linear Programming in a standard form as follows:

$$\begin{aligned} & Minimize(1/n)\sum_{j=1}^{k}\sum_{i=1}^{n} A_{ij} \times \|\mathbf{x}_i - \mu_{p_i}\| \\ \text{such that: } & \sum_{i=1}^{n} A_{ij} + u_i = \lceil \frac{n}{k} \rceil, 1 \leq j \leq k \\ & -\sum_{i=1}^{n} A_{ij} + v_i = -\lfloor \frac{n}{k} \rfloor, 1 \leq j \leq k \\ & \sum_{j=1}^{k} A_{ij} = 1, 1 \leq i \leq n \\ & A_{ij} + G_{ij} = 1, 1 \leq j \leq k, 1 \leq i \leq n \\ & u_j, v_j, G_{ij}, A_{ij} \geq 0, 1 \leq j \leq k, 1 \leq i \leq n \\ & u_j, v_j, G_{ij}, A_{ij} \in \mathbb{Z}, 1 \leq j \leq k, 1 \leq i \leq n \end{aligned} \tag{1.25}$$

Moreover, they have proved the solution of the corresponding Linear Programming (LP) (without integer constraints) is always integral. We could remove integer constraints to transform into an LP problem. Finally, the problem can be efficiently solved with the simplex algorithm [54].

1.4.3 MPC-Kmeans

The previous two algorithms produce a clustering output that satisfies all the constraints. On the contrary, other algorithms use penalties as a trade-off between optimizing the clustering criteria and satisfying as many constraints as possible. MPC-Kmeans is a well-suited example of that. We could also see the differences compared to COP-Kmeans as both bases on Kmeans and deals with pairwise constraints.

First, MPC-Kmeans assumes that the pairwise constraints show the user view of the similarity of the data. Therefore, the algorithm aims at adapting the distance metric to be more suitable to the domain knowledge. The matrix $\mathbf{A}_h$ is used for adjusting the distance between a member instance $\mathbf{x}$ to the h-th cluster center μ_h. This distance is defined $\|\mathbf{x}_i - \mu_{\mathbf{p}_i}\|_{\mathbf{A}_{\mathbf{p}_i}} = \sqrt{(\mathbf{x}_i - \mu_{\mathbf{p}_i})^T \mathbf{A}_h (\mathbf{x}_i - \mu_{\mathbf{p}_i})}$. If A is a diagonal matrix, it scales each dimension by a different weight; otherwise it creates a new set of features by linear combinations of the original ones. The objective function for clustering with the parameterized distance is:

$$O_{with-metrix} = \sum_{i=1}^{n} (\|\mathbf{x}_i - \mu_{\mathbf{p}_i}\|_{\mathbf{A}_{\mathbf{p}_i}} - log(det(\mathbf{A}_{\mathbf{p}_i}))) \tag{1.26}$$

where the second term is used for normalization.

Combining with the constrained cost proposed in [7], the objective function of MPC-Kmeans is:

$$\begin{aligned} O_{MPC-Kmeans} = &\sum_{i=1}^{n}(\|\mathbf{x}_i - \mu_{\mathbf{p}_i}\|_{\mathbf{A}_{\mathbf{p}_i}} - log(det(\mathbf{A}_{\mathbf{p}_i}))) \\ &+ \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in ML} w_{ij}\mathbb{1}[\mathbf{p}_i \neq \mathbf{p}_j] + \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in CL} \bar{w_{ij}}\mathbb{1}[\mathbf{p}_i = \mathbf{p}_j] \end{aligned} \tag{1.27}$$

where $\mathbb{1}$ is the indicator function, $\mathbb{1}[\top] = 1$ and $\mathbb{1}[\bot] = 0$, w_{ij} and $\bar{w_{ij}}$ is the constraint's violation cost.

The general steps of the algorithm is shown bellow:

1. Initialize cluster centers μ
2. Find partition $\mathbf{p}$

$$\forall i \in [1, n] : \mathbf{p}_i = \arg\min_h O_{MPC-Kmeans}$$

3. Calculate the cluster centers μ
4. Update the metrics A_h
5. Go back to Step 2 until convergence.

1.4.4 Spectral clustering with pairwise

Most of constrained clustering based on spectral approach has been focus on must-link (ML) and cannot-link (CL) constraints. In the following paragraph, we give an example of such an algorithm.

Ding algorithm The idea of Ding et al. [28] is to adjust the similarity matrix S using the constraint set. The SSCA algorithm modifies NJW algorithm to the adjustment step before calculating the similarity matrix S.

The pairs in pairwise constraints are adjusted directly on the original distance matrix D with $D_{ij} = 0$ if $ML(\mathbf{x}_i, \mathbf{x}_j)$ and $D_{ij} = \infty$ if $CL(\mathbf{x}_i, \mathbf{x}_j)$. The distance of other pairs are revised as the shortest path in the D matrix. The main steps of this algorithm are:

1. Revise the distance matrix D by pairwise constraints.
2. Construct the similarity matrix S using D and the diagonal matrix D_{diag} whose (i, i)-element is the sum of i-th row from S.
3. Compute $L = D_{diag}^{-1/2} S D_{diag}^{-1/2}$.
4. Let $1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ be the k largest eigenvalues of L and $u^1, u^2, \cdots, u^k$ the corresponding eigenvectors. All eigenvectors are normalized to have unit length. Form the matrix $U = [u_1 u_2 \cdots u_k]$.

5. Form the matrix V from U by renormalizing each row to have unit length ($V_{ij} = U_{ij}/\sqrt{\sum_j U_{ij}^2}$).

6. Consider each row of V as the new representation of points. Clustering them using kernel fuzzy c-means (KFCM) clustering.

1.5 Declarative Approaches

These approaches propose a general framework to formalize the problem using variables, condition equations, and an objective function. Depending on the type of formalization, it can be solved using various optimization tools. Formulation based on SAT, Integer Linear Programming (ILP), and Constraint Programming (CP) are presented in Section 1.5.1, 1.5.2, 1.5.3, respectively. Later, in Section 1.5.4, we review the postprocess approach, which takes a different way to formulate constrained clustering problems.

1.5.1 SAT-based formulations

Davidson et al [26] has proposed a SAT formulation for constrained clustering problems with $k = 2$. The pairwise constraints and the geometry constraints have been formulated and expressed in boolean formulas. The framework can deal with optimization criteria such as minimizing the maximal diameter, maximizing the minimal split, etc. The general formulation is as follows:

- Variables
  - Logical variables: $X_i, \forall i \in [1, n]$ where $X_i = \top$ ($\bot$) if the point $\mathbf{x}_i$ is assigned to cluster 1 (0).
  - Distance matrix D where D_{ij} is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.
- Expert constraints
  - Must-link constraints: $ML(i, j) \equiv (X_i \wedge X_j) \vee (\neg X_i \wedge \neg X_j)$
  - Cannot-link constraints: $CL(i, j) \equiv (X_i \wedge \neg X_j) \vee (X_j \wedge \neg X_i)$
  - Maximum diameter constraints which ensure the diameter of any cluster to less than or equal to α: $\forall 1 \leq i < j \leq n : D_{ij} > \alpha \implies (\neg X_i \vee X_j) \wedge (X_i \vee \neg X_j)$
- Objective functions: Constraints deduced from the objective value f are added to the SAT formulation. For example, the constraint for maximize-separation objective is $\forall 1 \leq i < j \leq n : D_{ij} < f \implies (X_i \wedge X_j) \vee (\neg X_i \wedge \neg X_j)$. Using a binary search, the algorithm obtains the best value.

1.5.2 ILP-based formulations

a. Unconstrained Clustering

In 1999, Merle et al was the first work to formalize the unconstrained clustering by an integer linear program [30]. A cluster $\mathscr{C}$ is a subset of $\mathscr{X}$. So, the number of all possible clusters is 2^n. Let $T = \{1, \cdots, 2^n\}$ be the set of possible non-empty clusters. Let c_t be the cost of the cluster $\mathscr{C}_t$. The membership constant a_{it} is equal to 1 if $\mathbf{x}_i \in \mathscr{C}_t$ and 0 otherwise. Then, the unconstrained clustering problem is formalised by:

$$\begin{aligned} \text{minimize } & \sum_{t \in T} c_t v_t \\ \text{such that: } & \sum_{t \in T} a_{it} v_t = 1, \forall i \in [1, n], \\ & \sum_{t \in T} x_t = k, \\ & x_t \in \{0, 1\}, \forall t \in T \end{aligned} \tag{1.28}$$

b. Clustering with anti-monotone expert constraints

Babaki[4] extended this formulation to deal with anti-monotone user-constraints. A constraint is anti-monotone if when satisfied on a set of instances $\mathscr{X}$ then it is also satisfied on all subsets $\mathscr{X}' \subseteq \mathscr{X}$. For example, the maximal capacity constraints are anti-monotone but minimal capacity constraints are not.

1.5.3 Constraint programming based approaches

a. Introduction

In constrained programming, a problem can be stated as a Constraint Satisfaction Problem (CSP) or a Constraint Optimisation Problem (COP). In CSP, they find the complete assignment of variables in the set X given $Dom(x)$ the domain of variable $x \in X$ and $\mathbb{C}$ the set of constraints. The COP problem has an objective function to find the optimal solution. The complexity of CSP or COP problem is NP-Hard in general cases. However, the constraint propagation and search strategies used in CP solvers help to find the solution effectively for many practical problems.

b. General framework

Dao et al. (2013) [31] has proposed and developed a CP-based framework for distance-based constrained clustering solution. This framework can model constrained clustering problems with different clustering objectives and different expert constraints. Latter, this framework has been improved by adding devoted propagation schemes for several optimization criteria[22].

In this model, the number of clusters k can be any value within a given bound, i.e. $K_{min} \leq K \leq K_{max}$. For each point i, a variable $G_i : i \in [1, n]$ is used to

represent the cluster assignment with $Dom(G_i) = \{1, ..., K_{max}\}$. The constraint set can be broken into three parts.

- Constraints to express a partition. In order to break the symmetry, the constraint precede $([G_1, ..., G_n], [1, ..., K_{max}])$ is used. It requires that $G_1 = 1$, $G_i \leq K_{max}, \forall i \in [1, n]$, and $\forall G_i > 1, \exists j < i : G_j + 1 = G_i$. While the maximum number of clusters is included in the precede constraint, the minimum number of clusters is expressed by the formulation $\#\{i \in [1, n] | G_i = K_{min}\} \geq 1$[2].

- Constraints to express clustering expert constraints. All constraints in 1.3 can be easily formulated with G variables. For example, a must-link (or cannot-link) on $\mathbf{x}_i$ and $\mathbf{x}_j$ is expressed as $G_i = G_j$ (or $G_i \neq G_j$).

- Constraints to express the objective function. Clustering criteria that the framework can support are minimising the maximal diameter of the clusters, maximising the minimal split between clusters, minimising the within-cluster sum of dissimilarities (WCSD) or minimising the within-cluster sum of squares (WCSS).

1.5.4 Postprocess approach

The idea of this approach is to find the clustering with minimum change from its initial group assignment that satisfies the constraints. The advantage of it is that they can apply to any clustering algorithm. The other constrained clustering approach using human knowledge as a priori to guide the clustering process. This approach provides modification after the clustering is found, so the expert constraints are considered a posteriori.

The general scheme of this approach is shown in Figure 1.3. A clustering algorithm creates a initial clustering $\mathbf{p}$. The constraint set $\mathbb{C}$ could be generated beforehand or after the users/experts receive $\mathbf{p}$ or based on the feedback of the user on $\mathbf{p}$. The post-process modifies $\mathbf{p}$ to have a partition $\mathbf{p}'$ satisfying $\mathbb{C}$.

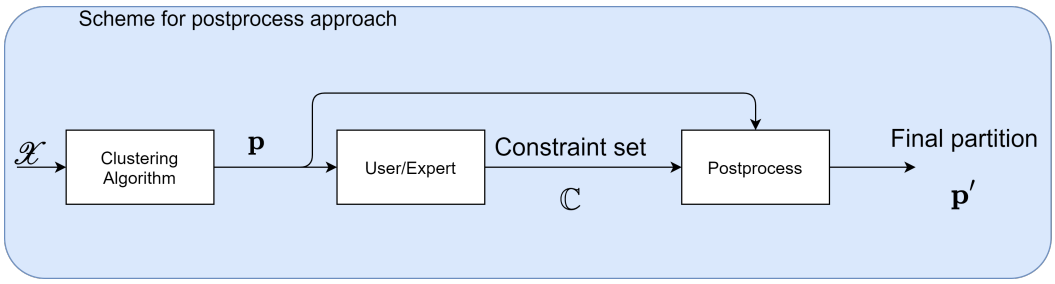

Figure 1.3: A workflow for postprocess approach. The initial clustering $\mathbf{p}$ can be produced by any clustering algorithm.

[2] $\#S$ denotes for the cardinality of the set S.

Authors in [58] have introduced the problem of Minimal Clustering Modification (MCM).

Input: Initial clustering $\mathbf{p}$ *and a constraint set* $\mathbb{C}$

Output: A modified clustering $\mathbf{p}'$ *that:*

$$\begin{aligned} &\min_{\mathbf{p}'} d(\mathbf{p}, \mathbf{p}') \\ &\text{subject to } \mathbf{p}' \text{ satisfies } \mathbb{C} \end{aligned} \tag{1.29}$$

where $d(\mathbf{p}, \mathbf{p}')$ is a distance measure between two partitions.

In this work, the distance between $\mathbf{p}$ and $\mathbf{p}'$ is defined as the number of points for which the cluster assignment has changed.

1.6 Summary

The first part of this chapter gives a general view of the clustering problem, including its definition, evaluation, and overall approaches. We also present in details well-known algorithms. Clustering is an important topic in data mining and has been studied for more than 50 years, so it could not be wrapped in a single chapter. More details on clustering can be found in [105].

In the second half, we present the formulations of the constrained clustering problem and two directions for the solution: the algorithm-specific approach and the declarative approach. The first approach is limited in its usage and application, while the second approach often has a complexity problem.

The next chapter will present the recent advantage of using neural networks for clustering and constrained clustering.

Chapter 2

Clustering using Deep Learning

As we introduced in chapter 1, a number of researches have been developed for solving the clustering problem with deep neural networks, which are called Deep Clustering. These works are worth explaining in this chapter because of the high clustering performance on various datasets and the immense potential for future improvement. The chapter is divided into two parts. The first section 2.1 dedicates to the deep neural networks, while the second section 2.2 presents the current works in Deep Clustering.

In the first section, we introduce the basics components of the neural network algorithm, which are the neural structure and the learning process. The rise of efficient computation, especially using multiple graphics processing units (GPU), allows the neural architecture to become larger and more complex. While increasing neurons makes the machine more robust, complex networks help the machine learn more effectively with a specific data type. Since this thesis focuses on the constrained clustering problem on general data, this chapter does not mention sequential data and neural networks with memory. Instead, we put the focus on the Convolutional Neural Networks (CNNs) and the Autoencoders.

The second section lists the main methods of deep clustering: the naive approach, the self-training approach, and the cluster-friendly representation learning approach. We describe the general process and give examples of algorithms for each method.

2.1 Introduction to Deep Learning

This section is divided into three parts. The first part gives basic concepts of neural networks and describes the traditional neural network architectures. The next two parts help us understand Convolutional Neural Network (CNN) and Autoencoder (AE) architectures.

Our introduction only covers the vital points of deep neural networks related to deep clustering. For a more comprehensive introduction to neural network architectures and techniques, we recommend the Deep Learning book [38].

2.1.1 Fundamentals of neural networks

a. Artificial Neuron

The basic element to create a neural network is the artificial neuron. It is a function f_i of the input $\mathbf{x} = (x_1, \cdots, x_d)$ weighted by a vector of connection weights $\mathbf{w}_i =$

$(w_{i,1}, \ldots, w_{i,d})$, a bias b_j, and completed by an activation function ϕ. So, the output is expressed as follows:

$$y_i = f_i(x) = \phi(\mathbf{x} \times \mathbf{w}_i + b_i) \tag{2.1}$$

Activation function The activation function ϕ helps the function f becoming non-linear. Thus, the neural network is a non linear transformation of the input data. Several activation functions could be considered.

- Binary Step Function:

$$\phi(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

  This function mimics the signal of biological nerve cell. However, its derivative is zero which causes a problem in the learning process.

- Sigmoid function:

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

  The sigmoid used to be the most used activation function because it is differentiable and has values from 0 to 1. But one of its problem is that the derivative value is very close to 0 even if $\|x\|$ is not close to 0.

- Rectified Linear Unit (ReLU) function [70]:

$$\phi(x) = \max(0, x)$$

  The ReLU main advantage is that it is far simpler when compared to other functions. The drawback of ReLU is that the neuron could remain inactive because the gradient is equal to 0 for the negative value.

b. Feed Forward Neural Network

Layers A typical neural network comprises an input layer, multiple hidden layers, and an output layer. Each layer in the hidden layers is a set of artificial neurons using outputs of the previous layer and forwarding the results to the next layer.

Neural network with feedbacks There exist network structures that could loop back to previous layers. Those structures which have internal states are often used to deal with sequential input data. Therefore, we are not going to explain these types of networks.

Neural architecture The choice of the activation function for the output layer depends on the problem. The two main types of problems are:

- Regression: The output is a continuous vector $\mathbb{R}^k$. The activation function could be the identity function $g(x) = x$ for unbound values, the ReLU function $g(x) = max(x, 0)$ for $\mathbb{R}_+$, or the tanh function for $\mathbb{R}_{[-1,1]}$.

- Classification: The most used function is the softmax function.

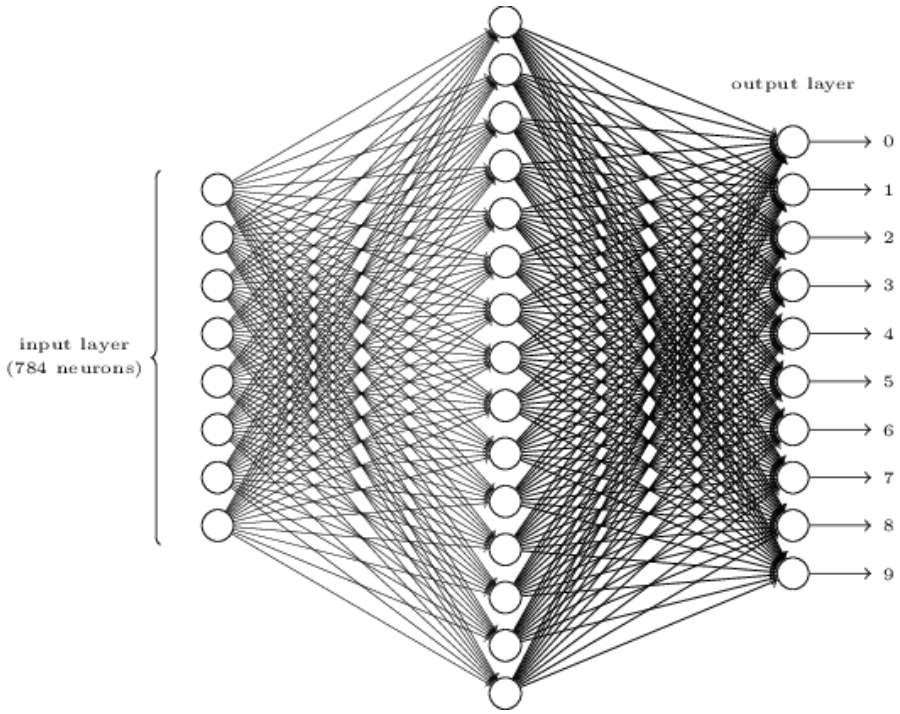

Figure 2.1: Neural network architecture for digit image of 28×28 with one hidden layer. Source: Dima Shulga

Example To illustrate a neural structure, we give an example of image classification with a fully connected neural network where each neuron in a layer is fully connected to all neurons in the following layer. The overall structure is shown in Figure 2.1.

Each input is a grayscale image with its size of 28×28 which represents a hand written digit. So, we need to have 784 neurons in the input layer. Then, they are all connected to 512 neurons of a hidden layer. Because there is only one hidden layer in this example, the hidden layer links to ten neurons of the output layer correspond to the prediction with ten digits.

In the remaining of this subsection, we are going to detail how learn with feed-forward neural.

c. Problem and objective function

Problem A general problem of deep learning is to predict an output $\hat{y}$ given an input data $\mathbf{x}$. Depending on the type of $\hat{y}$, we have a regression problem or a classification problem.

- The classification problem: $\hat{y}$ gives the membership to k predefined sets which are denoted as class $\mathscr{L}_1, \ldots, \mathscr{L}_k$. $\hat{y}$ is presented as a vector of k binary values $\hat{y} = \{\hat{y}_1, \ldots, \hat{y}_k\}$ where $\hat{y}_i = \top$ or 1 indicates that the input data $\mathbf{x}$ belongs to class $\mathscr{L}_i$.
- The regression problem: $\hat{y}$ is a quantity which represented by a continuous vector $\mathbb{R}^k$.

In a supervised setting, $\mathscr{X}$ is the input data consist of n points $\mathbf{x}_i, \forall i \in [1, n]$ with their corresponding labels $y_i, \forall i \in [1, n]$. Both of the problems are trying to minimize the difference between the predictions and the correct labels.

Objective function The output of a neural network model can be expressed in the following form: $F_\theta(\mathbf{x})$ where F is the network function, θ is the set of parameters, and $\mathbf{x}$ is the input. To improve the function F, we need an objective function to optimize. The mean squared error (MSE) and the cross-entropy (CE) functions are the most common functions to be minimized for regression and classification, respectively.

$$J_{MSE}(\theta; \mathscr{X}) = \sum_{i=1}^{n} \|y_i - F_\theta(\mathbf{x}_i)\| \tag{2.2}$$

$$J_{CE}(\theta; \mathscr{X}) = \sum_{i=1}^{n} - \sum_{j=1}^{k} y_{ij} \log(F_\theta(\mathbf{x}_i)[j]) \tag{2.3}$$

where: $\mathscr{X}$ is the input data, y_i is the label vector of the point $\mathbf{x}_i$.

Note for a classification problem: $F_\theta(\mathbf{x}_i)$ consists of k values belonging to $\mathbb{R}_{[0,1]}$. The j-th value of the output represents the probability of the input to belong to class $\mathscr{L}_j$. So, we have: $F_\theta(\mathbf{x}_i)[j] = p(\mathbf{x}_i \in \mathscr{L}_j | \mathbf{x}_i, \theta)$

d. Gradient Descent

The gradient descent is a way to minimize the objective function $J(\theta; \mathscr{X})$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta; \mathscr{X})$. The learning rate η determines the size of the steps we take to reach a local minimum.

Batch gradient descent The batch gradient descent computes the gradient of the cost function with respect to the parameters θ for the entire training dataset.

$$\theta = \theta - \eta \times \nabla_\theta J(\theta; \mathscr{X}) \tag{2.4}$$

Because of the size of the dataset, computing the gradient can be very slow and even run out of memory. The sole advantage of this method is that it is guaranteed to converge to the global minimum for convex surfaces and to a local minimum for non-convex surfaces.

Stochastic gradient descent (SGD) The stochastic gradient descent performs an update of the parameters for each training point $\mathbf{x}_i$.

$$\theta = \theta - \eta \times \nabla_\theta J(\theta; \mathbf{x}_i) \tag{2.5}$$

This method is also not efficient because the parameters θ must be updated n times. Because the instances $\mathbf{x}_i$ and $\mathbf{x}_j$ could be much different from each other, the updating of θ causes the objective function to fluctuate wildly as shown in Figure 2.2. This behavior could make the model jump to new and possibly better local minima but the performances may differ from one run to another, thus decreasing the consistency of the results. In the end, the learning rate η must be reduced to make the SGD achieve convergence.

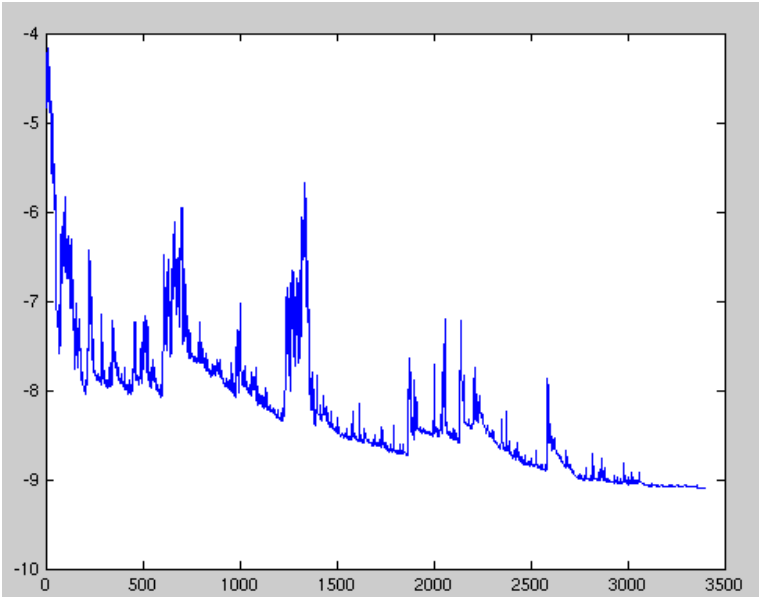

Figure 2.2: Fluctuations of the objective values as the parameters updated by Stochastic Gradient Descent. Source: Researchgate[1]

Mini-batch gradient descent Mini-batch gradient descent updates the parameters for every mini-batch of m training examples:

$$\theta = \theta - \eta \times \nabla_\theta J(\theta; \{\mathbf{x}_i, \mathbf{x}_{i+1}, \cdots, \mathbf{x}_{i+m-1}\}) \tag{2.6}$$

This method retains the consistency of the result while achieving the best performance through a balance between memory and computation time. The mini-batch

[1] https://www.researchgate.net/figure/SGD-fluctuation-16_fig11_325311506

size m usually ranges from 64 to 1024, depending on the neural network and applications.

e. Optimizer

In the previous section, we have not explained how the learning rate value is set. Algorithms for initializing and updating the learning rate are called *Optimizers*.

Momentum Instead of using the gradient to update directly the parameters, the Momentum Optimizer adjusts its momentum $V(t)$ by the previous velocity $V(t-1)$ and the current gradient $\nabla J(\theta)$.

$$V(t) = \gamma V(t-1) + \eta \times \nabla J(\theta)$$

Then, the parameter is updated by $\theta = \theta - V(t)$

Adam While the velocity is used for adjusting the learning rate in Momentum Optimizer, Adam (Adaptive Moment Estimation) uses momentums of both first and second order. The details of the algorithm can be seen in [53].

2.1.2 Convolutional neural networks

Convolutional neural network (CNN), which have been proposed by Lecun et al. [59] has become one of the most successful methods in the field of pattern recognition. It has been applied in various domains: image classification[61, 55], object detection[79], natural language processing[48], etc. Besides using fully connected layers, a CNN has convolutional layers and pooling layers.

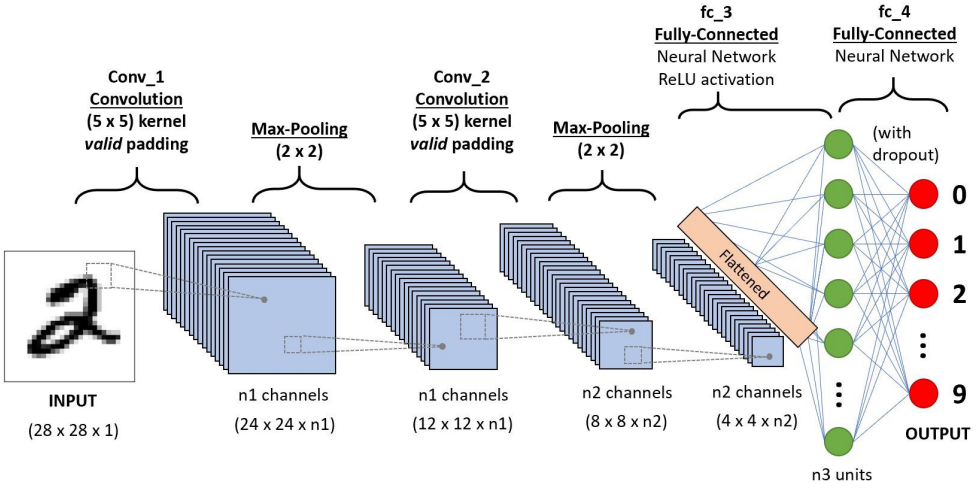

Figure 2.3: CNN architecture for digit images of 28×28 with 2 convolutional layers and 2 pooling layers. Source: https://medium.com/@_sumitsaha_

a. Convolution Layer

Unlike the previous example using the fully-connected neural networks, each neuron in a convolution layer represents a feature from a local kernel. Traditionally, a kernel is a *fixed* matrix which is used for extracting various features of an image such as blurring, sharpening, edge detection, etc. In CNN, the kernels are defined by the parameters of the neurons so they are *updated/changed* after each learning epoch. In the example shown in 2.3, the original input is a matrix of 28×28. If kernels have the size of 5×5, there will be 24×24 different kernels for each channel. The layer has $n1$ channels. So, the number of kernels in this convolution layer is $24 \times 24 \times n1$.

The convolution layer helps the network extracting the feature of data, while keeping its spatial relation intact.

b. Pooling Layer

The pooling layer reduces the dimensions of the previous layer by combining several neural inputs. Because the pooling layer summarises the features presented in different regions, it makes the model invariant with the positional change of the object in the input image. In Figure 2.3, the pooling combines tiling sizes of 2×2. The two common functions for the combination are max and average.

c. Training

Because the CNN describes the structure of a neural network, it can be used for many tasks with different training schemes. In this short introduction, we explain the learning process of the multi-class classification problem which is the initial usage of CNNs. Later, learning with invariance and clustering are mentioned in 2.2.

We denote:

- f_θ a mapping from the input space to a feature space with θ the parameters of f.
- $\mathscr{X} = \{x_1, x_2, ..., x_n\}$ the training set of n instances.
- $y_i = \{0, 1\}^k$ is the label vector of the i-th instance. $y_{ij} = 1$ when the i-th point belongs to class j.
- g_W is a classifier to predict labels from feature vectors $f_\theta(x_i)$; W is the parameters of g.

In the AlexNet model [55], the parameters W and θ are learned by the cross-entropy loss:

$$L = \sum_{i=1}^{n} \sum_{h=1}^{k} -y_{ih} \log g_W(f_\theta(x_i)) \tag{2.7}$$

d. Conclusion

Designing a good feature extraction requires a lot of knowledge of both data and neural networks. While the experience with neural architectures could improve through practice, prior knowledge of data could be absent in some applications. So, in the following subsection, the autoencoder is introduced as a neural structure to self-learn the feature representation.

2.1.3 Autoencoders

a. Introduction

The most widely used neural networks in deep clustering algorithms are the AutoEncoder (AE) and its variants [77, 101, 40].

An autoencoder consists of two parts: an encoder function $\mathbf{z} = f(\mathbf{x})$ and a decoder function $\mathbf{x}' = g(\mathbf{z})$ that attempts to reconstruct the original input. In Table 2.1, the common notations used for autoencoder are explaining what is an autoencoder.

Table 2.1: Notations and their meanings

Notations	Meanings
Z	The latent representation space of autoencoder
$\mathscr{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_n\}$	The corresponding embedded points from input data $\mathscr{X}$
L_{reg}	The regularization loss
L_{rec}	The reconstruction loss
d, d_z	The dimension of $\mathbf{x}$ and $\mathbf{z}$, respectively
$\theta = \{\theta_e, \theta_d\}$	The network parameters

Plain Autoencoder The encoder is a function f that maps the data $\mathbf{x} \in \mathbb{R}^d$ to $\mathbb{R}^{d_z}$ to get a latent representation as:

$$\mathbf{z} = f(\mathbf{x}) = s_f(W\mathbf{x} + b_z) \tag{2.8}$$

where s_f is a nonlinear activation function, W is the weight matrix and b_z is the bias vector.

The decoder function g maps the outputs of hidden units to the original input space as:

$$\mathbf{x}' = g(\mathbf{z}) = s_g(W'\mathbf{z} + b_o) \tag{2.9}$$

where s_g is a nonlinear activation function which is typically the same as the encoder; W' and b_o are parameters of the decoder.

The model parameters $\theta = \{W, b_z, W', b_o\}$ are learned by minimizing the reconstruction error on training data. The loss function is given by:

$$L_{rec} = \sum_{i=1}^{n} L(\mathbf{x}_i, g(f(\mathbf{x}_i))) \tag{2.10}$$

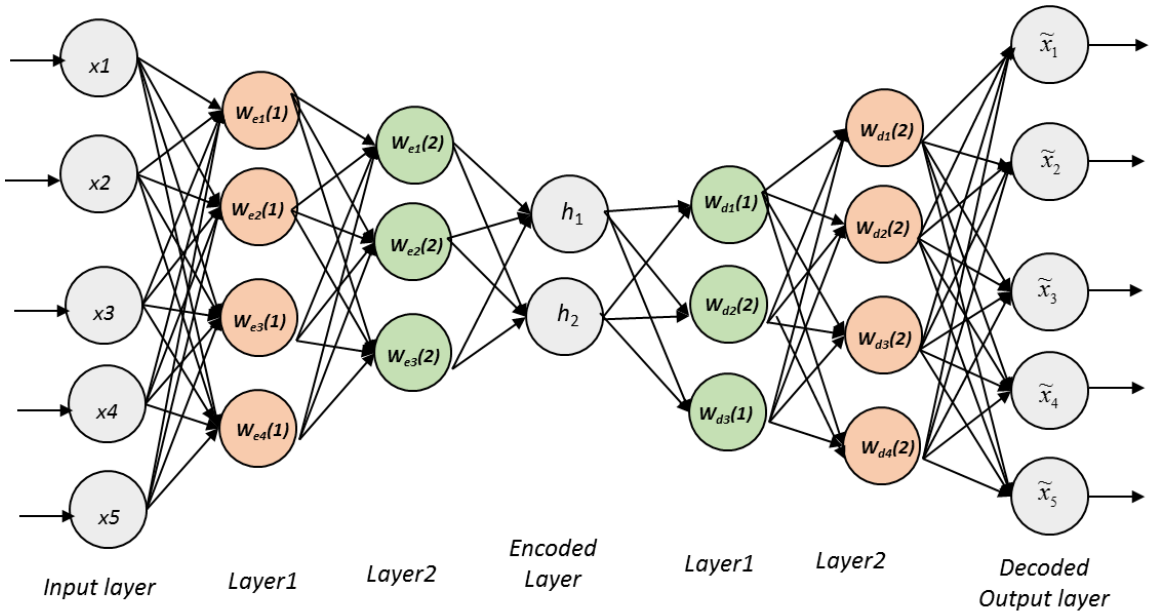

Figure 2.4: Neural structure of an autoencoder. Source: Prakash [78]

where L is a loss function, usually a mean squared error $L(\mathbf{x}_i, \mathbf{x}'_i) = \| \mathbf{x}_i - \mathbf{x}'_i \|^2$

Training The model is trained in an unsupervised manner by minimizing the reconstruction error between the decoder output and the original input, typically using a minibatch gradient descent, following the gradients computed by backpropagation.

Stacking The model can be initiated by setting randomly weights for parameters θ. However, stacking technique [10] is recommended for the autoencoder with multi hidden layers. The first hidden layer is quickly trained as a shallow autoencoder (the only layer between the input and output layer). Similarly, the next layer is trained in the same manner with the previous layer considered as the input layer.

b. Autoencoder taxonomy

Autoencoders can differ according to several dimensions. The four main features are depicted in Figure 2.5.

Neural architecture Depending on the type of data, the encoder and its mirror (the decoder) could be fully connected neural networks, CNN for images or Long-Short-Term Memory (LSTM) for sequences.

Regularization When the learned features (encoded layer) are required to have a special mathematical property, a penalization is added to the objective function to enforce that property.

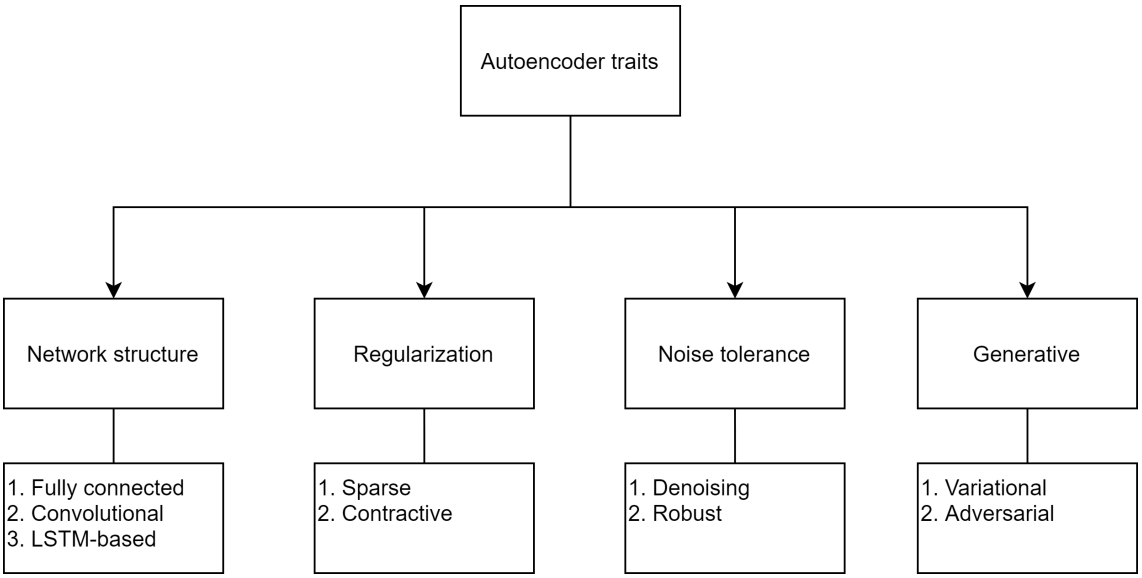

Figure 2.5: Characteristics of an autoencoder

For instance, in Sparse AE, one of the desired properties for the encoded layer is the sparsity. It means that most values in the embedding vector are zero or close to zero. Considering the embedded layer consists of d_z neurons: $f_1, \cdots, f_{d_z}$, the average activation value for the neuron f_t is

$$\hat{\rho}_t = \frac{1}{n} \sum_{i=1}^{n} f_t(\mathbf{x}_i)$$

Let ρ be the target average activation which usually selected by the expert. The regularization loss is

$$L_{sparse} = \sum_{t=1}^{d_z} KL(\rho \| \hat{\rho}_t)$$

where KL is the Kullback–Leibler divergence.

Noise tolerance AE The most used technique for learning with noise tolerance is denoising. Denoising Autoencoder receives corrupted data points as inputs. It is trained to predict the original, uncorrupted data point as its outputs, allowing representations to be robust to partial corruption of the input patterns. An artificial corruption could be an isotropic Gaussian noise ($\widetilde{x} = x + \epsilon$) or a stochastic mapping that randomly sets a portion of its input dimensions to 0 ($\widetilde{x} = Dropout(x)$). The general process is shown in Figure 2.6.

Generative AE A normal autoencoder can reconstruct encoded data, but an output from an arbitrary encoding could be meaningless. Generative models learn probability distributions that allow to create new input samples following the same distribution.

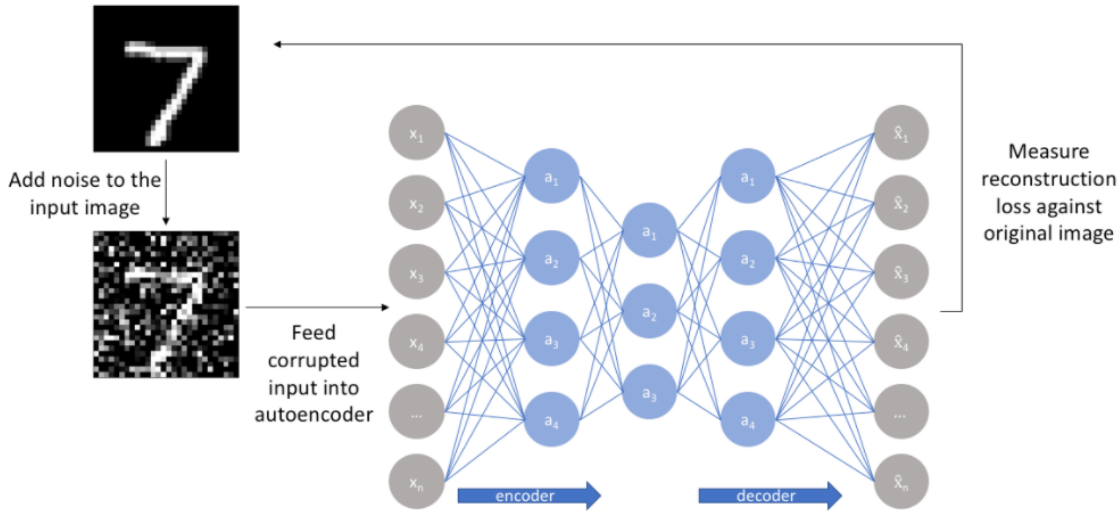

Figure 2.6: Denoising Autoencoder. Source:Jeremy Jordan[2]

One of the most well-known generative autoencoder is variational autoencoder. A Variational Autoencoder (VAE) is a type of autoencoder that applies a variational Bayesian approach to encoding. Its objective is to approximate the distribution of the latent variables given the observations. Therefore, the input is encoded as a Gaussian probability density $p_{\theta_e}(\mathbf{z}|\mathbf{x})$ represented by a mean μ and a variance σ. Then, the input of the decoder is $\mathbf{z}$ taken from the sample distribution in the latent space and the output is the probability distribution of the data $p_{\theta_d}(\mathbf{x}|\mathbf{z})$. Overall, the VAE aims at maximizing the probability of generating real data samples:

$$\arg\max_{\theta} \prod_{i=1}^{n} \int_{z} \left(p_{\theta_d}(\mathbf{x}|\mathbf{z}) p_{\theta_e}(\mathbf{z})\right) d\mathbf{z}$$

The process is illustrated in Figure 2.7.

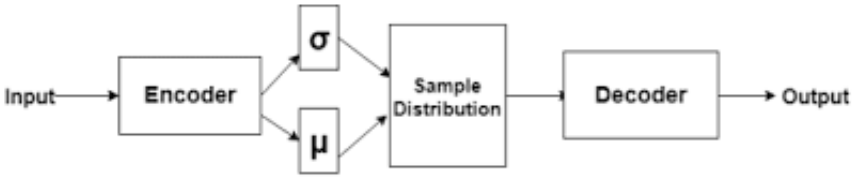

Figure 2.7: Variational Autoencoder

In our work in Chapter 5, we are using a fully connected Autoencoder. The network is trained with stacking and denoising techniques.

[2] https://www.jeremyjordan.me/autoencoders/

2.2 Deep Clustering

In this section, the neural approaches for clustering are introduced. Then, each approach will be described and given several algorithms as examples. Some of these methods are the foundations for the constrained clustering solutions in Section 3.2 and our proposal in Chapter 5.

2.2.1 Problem definition and Approaches

Competitive learning Neural networks have been used for clustering for more than three decades ago. The model structure consists of only two layers which are the input layer and the competitive layer. The examples of this method are *Clustering of self-organizing map* (SOM) [94] and *Generalized learning vector quantization* [83]. Because of their inferior clustering performances, they are not in the scope of this thesis.

Deep learning The deep neural structure or multi-layer network has proven to be a powerful tool for dimensionality reduction, making the clustering task much easier. Therefore, in the past, dimensionality reduction and clustering were applied sequentially [81, 84]. We call it the naive approach. A second approach is to learn simultaneously new representation of data and clustering structure[40, 106]. The third approach for deep clustering is self-training[101, 17]. The idea of this approach is that clustering is an unsupervised task of classification. The method has two phases. The first phase creates a model that is strong enough to produce a set of *pseudo labels*. The second phase is a process of creating pseudo labels, training with those labels, updating the model, and repeat.

General process Because several methods can be combined together, the general flow of deep clustering is:

1. Initialize the model (or feature space)
2. Train with a loss for representation and/or clustering:
   (a) Update clustering parameters
   (b) Update network parameters
3. Clustering/Postprocess

2.2.2 Naive approach

One of the challenges of clustering is data representation. Indeed, it has been shown that higher dimension means less effective distance measure[29]. Thus, it is harder to do clustering directly on the data. The naive approach performs first a data transformation before clustering with new representation of data.

Dimensionality reduction is the most common solution for this problem. It transforms high-dimensional data into low-dimensional data through linear or non-linear mapping. Linear mapping methods include principal component analysis [27], non-negative matrix factorization [92], etc while the NJW algorithm for spectral clustering introduced in Chapter 1 is an example for a non-linear mapping.

For deep neural algorithms, feature learning or representation learning is the general process for discovering a better data representation. In most cases, using representations in the feature space helps improve clustering quality. We present here two main ways of feature learning. The first method is to use an autoencoder to learn the embedding representation without losing much information. The second method is to enforce critical properties such as local invariance on the feature space.

a. Reconstruction loss with Autoencoder

Autoencoders have been introduced in Section 2.1.3. Once we finish training with an autoencoder, we can apply a classic clustering algorithm on latent embeddings of the data.

b. Self-augmented loss with Classifier

The idea of this approach is to enforce local invariance of data. It means that small *changes* in the original data should lead to small distances in the feature space. Let $T : \mathbb{R}^d \to \mathbb{R}^d$ be an augmentation function under which the data representation should be invariant. For image data, T could be a translation, a rotation, or a scale operation.

Facenet[84] learns local invariance by triplet constraints. A triplet constraint in Facenet consists of three images: an original image, a positive image and a negative image. The original image is a face image of a person. Each original image $\mathbf{x}_a$ is applied the T function to generate the positive sample $\mathbf{x}_p$. An image of a different person is used as the negative sample $\mathbf{x}_n$. The model is trained such that the relative distance in the feature space f of a triplet $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$ satisfies:

$$\|f(x_a) - f(x_p)\| + \alpha < \|f(x_a) - f(x_n)\|$$

where α is a constant for the triplet margin.

2.2.3 Self-training

In self-training, an initial classifier (a clustering model) is created. Then, the dataset is labelled with the classifier. The labels which are predicted with high confidence are used to improve the classifier. In this section, we introduce Deep Embedded Clustering (DEC) [101] and Clustering using CNN [17] as two methods for the self-training approach. The first method uses the encoder of a pre-train Autoencoder and K-means as the initial classifier. At the same time, the second method uses the classifier of a similar task (also known as transfer learning) as the starting

model. Both of them use labels of all the samples as training, yet they use different functions to favor the high-confidence predictions.

a. DEC

Neural architecture The main structure of DEC is an autoencoder. The parameters for training include the parameters of the autoencoder and the set of k cluster centers in the feature space Z. DEC algorithm has three steps:

Step 1 : Training the autoencoder

Step 2 : Obtaining the cluster centers $\mu_h : \forall h \in [1, k]$ by performing k-means clustering on the embedded points

Step 3 : Optimizing the parameters of the encoder and the cluster centers using a self-training loss

Figure 2.8 shows the architecture of DEC and the use of the encoder for Step 3. In the following parts, we explain in more details Step 1 and Step 3.

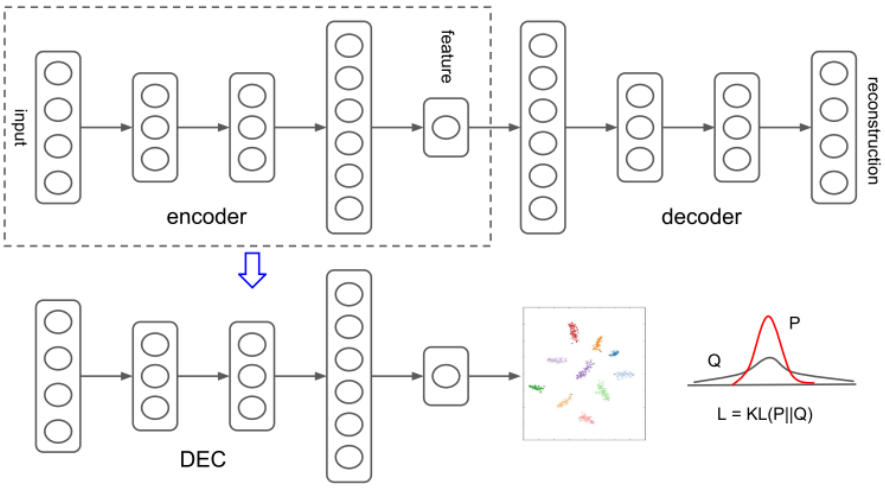

Figure 2.8: Network structure of DEC. Source: Junyuan Xie[101]

Autoencoder training The autoencoder used in DEC is the stacked (multilayer fully connected) autoencoder. The autoencoder is trained with the stacking and denoising techniques which has been explained in Section 2.1.3.

Self-training with KL divergence Applying clustering algorithm on embedding points creates a partition. However, the result can not be improved further. Xie et al [101] utilizes the cluster centers μ to measure the similarity of a point to a cluster. Then, they propose an objective function to reinforce the similarity of a point to the nearby clusters.

The similarity measure is based on the t-distribution.

$$q_{ij} = \frac{(1+ \parallel z_i - \mu_j \parallel^2)^{-1}}{\sum_{j'}(1+ \parallel z_i - \mu_{j'} \parallel^2)^{-1}}$$

where: z_i is the embedding representation of the point $\mathbf{x}_i$, q_{ij} is the probability for $\mathbf{x}_i$ to be assigned to $\mathscr{C}_j$, $\|a-b\|^2$ is the Euclidean distance between a and b.

The training aims at increasing high probabilities and decreasing low probabilities of the matrix q. Thus, the target distribution p_{ij} is computed as follows:

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}}$$

where $f_j = \sum_{i=1}^{n} q_{ij}, \forall j = 1, \ldots, k$ are the soft cluster frequencies.

The choice for the target distribution relies on a common technique of self-training. When the original distribution is close to a uniform distribution $q_{ij} \approx \frac{1}{k}, \forall j \in [1, k]$, the target distribution is fairly similar. It implies that the original distribution has nothing to learn. But when $\{q_{i1}, \cdots, q_{ik}\}$ is unbalanced, the target distribution will further widen these gaps. Finally, the Kullback–Leibler divergence loss is used to make the original distribution approaching the target one.

$$L = KL(P \parallel Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.11}$$

At each iteration, the gradients of the loss L with respect to the parameters of the encoders and the cluster centers μ_h are computed and used to update the model. The training stops when the percentage of points that change cluster assignment between two consecutive iterations is less than a predefined value ϵ.

b. Clustering using CNN

Convolutional neural networks have become popular and achieve top benchmarks for image classification problems. By the nature of the architecture, CNNs can not learn directly from unlabeled data. So, there are few works on adapting CNN models in a clustering task. The most notable is the work of Caron from the Facebook AI group in 2019 [17].

First, we will explain the method for supervised classification. Then, the unsupervised learning method for clustering is introduced.

Supervised learning

Let us recall the notations related to training CNNs in Section 2.1.2.

- f_θ : a mapping from images to feature spaces with θ the parameters of f.
- $\mathscr{X} = \{x_1, x_2, ..., x_n\}$: the training set of N images. Each image x_i is associated with a label vector $y_i = \{0, 1\}^k$ where $y_{ij} = 1$ if x_i belongs to class j.

- g_W : a classifier to predict labels from feature vectors $f_\theta(x_i)$; W is the parameters of g.

In this work, the parameters W and θ are learned by the following loss:

$$L = \sum_{i=1}^{n} l(g_W(f_\theta(x_i)), y_i) \tag{2.12}$$

where l is the negative log-softmax function.

Unsupervised learning by clustering

The main idea of unsupervised learning is to run a pre-trained CNN to have the representations of data. The labelled dataset used in pre-training should be similar to our target data. Thus, we apply a clustering algorithm to have the *pseudo-labels*. These labels are used to learn the classification task for the CNN model. The whole process is shown in Fig. 2.9.

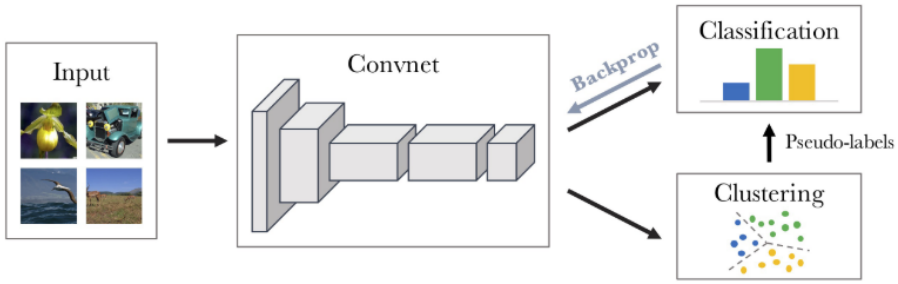

Figure 2.9: Deep Clustering using Convolutional Neural Network. Source: Mathilde Caron[17]

Initialization

The parameters θ can not be created randomly as the performance for the clustering task would be inadequate. The common approach for initializing θ is through transfer learning. The parameters θ are the weights of an image classifier which has the same network structure and is trained on a labelled image dataset. In this case, f_θ produces good image features and enables a strong partition of the points.

Clustering

The k-means objective function is used to find the cluster center matrix μ and the cluster assignment $\mathbf{y}$ in the feature space f_θ.

$$\min_{\mu \in \mathbb{R}^{d \times k}} \frac{1}{n} \sum_{i=1}^{n} \min_{y_i \in \{0,1\}^k} \|f_\theta(x_i) - \mu y_i\|^2 \tag{2.13}$$

such that $y_i^T 1_k = 1$.

Algorithm

The algorithm for clustering can be outlined as follows:

1. Initializing θ by transfer weights of a similar task.
2. Solving Eq. 2.13 gives the pseudo-labels $\mathbf{y}$ and a cluster center matrix μ.
3. Updating the network parameters θ by reducing classification loss (Eq. 2.12) with $\mathbf{y}$.
4. If not convergence, go to Step 2.
5. Return the clustering based on $\mathbf{y}$.

Conclusion The advantage of this method is the ability to use solid discriminative neural structures in classification tasks such as AlexNet[55] for image data. However, there are two weak points. First, the clustering data (unlabeled) must have a similarly labeled dataset to enable transfer tasks. Second, the process of alternating between two objectives (Step 2 and Step 3) could get a trivial solution where it is stuck at a local minimum. Although few mechanisms , proposed in their work, could reduce this risk, the training with no supervision on the label is challenging and requires knowledge of the input data. In other words, clustering performance is sensitive to the choice of network structure and hyper-parameter setting.

2.2.4 Cluster-friendly representation learning

The process of this approach is similar to the naive approach, which is to learn a new data representation before clustering. However, our target for the representation is not only to lower the number of dimensions but also enforce the cluster-like structures in the embedding space. So, in the next sections, we introduce the two methods following this approach and give comparisons between them.

a. IDEC

Neural architecture IDEC[40] keeps the same network architecture and parameters as DEC. The main structure of IDEC is a stacked autoencoder. The parameters are $\theta = \{\theta_e, \theta_d\}$ the parameters of the encoder and the decoder, $\mu = \{\mu_1, ..., \mu_k\}$ the cluster centers in the latent embedding space.

Joint training scheme In order to transform the latent representation to better highlight the clustering structure, a clustering loss is added and the two losses (clustering and reconstruction) are simultaneously optimized. Therefore, the loss is as follows:

$$L(\theta) = (1 - \alpha)L_{rec}(\theta) + \alpha L_c(\theta) \tag{2.14}$$

where α is a coefficient balancing between the two losses.

The training scheme of IDEC has 3 steps which are similar to DEC except Step 3. Step 3 of IDEC algorithm can be outlined as follows:

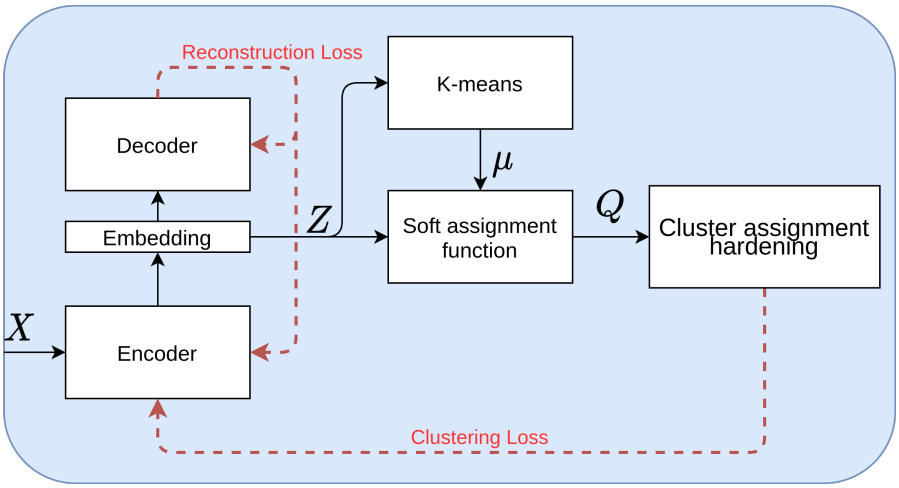

Figure 2.10: The learning scheme of IDEC

1. Compute the soft cluster assignments of all points (to all clusters) based on Student's t-distribution:

$$q_{ij} = \frac{(1+ \parallel z_i - \mu_j \parallel^2)^{-1}}{\sum_{j'}(1+ \parallel z_i - \mu_{j'} \parallel^2)^{-1}} \tag{2.15}$$

where $\|a-b\|^2$ is the Euclidean distance between a and b.

2. Compute the target assignment as:

$$p_{ij} = \frac{q_{ij}^2/f_j}{\sum_{j'} q_{ij'}^2/f_{j'}} \tag{2.16}$$

3. Compute the Kullback–Leibler divergence for self-training.

$$L_c = KL(P \parallel Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.17}$$

4. Compute the reconstruction loss.

$$L_{rec} = \sum_{i=1}^{n} \|\mathbf{x_i} - g_{\theta_d}(f_{\theta_e}(\mathbf{x_i}))\|^2 \tag{2.18}$$

where f and g are the functions of encoder and decoder, resp.

5. Compute gradient descent of $L = (1-\alpha)L_{rec}(\theta) + \alpha L_c(\theta, \mu)$ with respect to θ (network parameters) and μ (cluster centers).

6. Update θ and μ.

7. Check convergence. If not, go to step 1.

Comparison with DEC The experimental results show that the clustering quality of IDEC is better than DEC, but IDEC takes more time to converge. It suggests that data representation takes more time when jointly learned with cluster structure, but it will acquire a more accurate representation.

b. Deep Clustering Network

Neural architecture The neural structure of DCN is a stacked autoencoder. The parameters in DCN are the parameters of the autoencoder, the matrix of cluster centers $\mu = \begin{pmatrix} \mu_1 \\ \cdots \\ \mu_k \end{pmatrix}$ in the feature space, and the assignment matrix $s = \begin{pmatrix} s_1 \\ \cdots \\ s_n \end{pmatrix}$.

Objective function Deep Clustering Network (DCN) [106] proposed a cost function, which is a combination of the reconstruction loss for the autoencoder and the within cluster sum of squares (WCSS) for the clustering task.

$$\min_{\theta,\mu,\mathbf{s}} \sum_{i=1}^{n} \left[l(g(f(\mathbf{x}_i)), \mathbf{x}_i) + \frac{\lambda}{2} \|f(\mathbf{x}_i) - \mu \times \mathbf{s}_i\|^2 \right] \tag{2.19}$$

where: $\mathbf{s}_i$ is the assignment vector such that $\forall i, j : s_{ij} \in \{0, 1\}$ and $1^T \mathbf{s}_i = 1$.

Let us recall that f, resp. g is the encoding (resp. decoding) function and that θ represents the parameters of the auto-encoder.

Training process While IDEC defines a loss function so that all their parameters can be updated at the same time, DCN defines an objective function for which is hard to find the optimal values for all variables. Therefore, DCN alternately optimizes θ, $\mathbf{s}$, and μ in each epoch. The detailed algorithm is shown in Algorithm 2.1.

Algorithm 2.1 DCN Algorithm

Input: Data set $\mathscr{X}$
Output: Clustering $\mathbf{p}$;
1: Pretrain an auto-encoder and run K-means
2: **for** $i := 1$ **to** T **do**
3: Update the network parameters (θ) using the loss function
4: Update the assignment $\mathbf{s} = argmin||f(x_i) - \mu_i)||^2$
5: Update centroids $\mu = \mu - \eta \nabla_\mu$
6: **end for**
7: $\mathbf{p} = \{\arg\max \mathbf{s}_1, \cdots, \arg\max \mathbf{s}_n\}$
8: Return $\mathbf{p}$

Comparison with DEC/IDEC The experiments show that DCC is better than DEC but worse than IDEC in terms of clustering quality. It proves again the superiority of combining learning representation and learning clustering structure. The difference in performances between IDEC and DCN shows that the result from a deep learning model strongly depends on its learning method as the two methods are similar in their ideas and objectives.

2.2.5 Discussion

In this section, we present various methods that rely on deep learning for clustering. These methods differ on the network architecture and on the learning scheme. But fundamentally, they differ in how much knowledge about the data is integrated into the system. Based on data characteristics, the network can learn a better representation of data through adjusting neural architecture (for example, CNN, spare autoencoder), enforcing a cluster-like structure in the embedding. Overall, the more data knowledge a machine can integrate, the better chance of improving its performance. We will continue this critical point in Chapter 3.

Chapter 3

Knowledge Integration in Deep Learning and Its Application on Deep Constrained Clustering

In Chapter 2, we have reviewed the general techniques and detailed several neural network algorithms for clustering. In this chapter, we present some solutions for constrained clustering problems with deep learning. However, to better understand the contexts, we provide a more general overview of integrating knowledge in a deep learning model.

The first generalization is the integration of knowledge instead of expert constraints. While an expert constraint is a fixed set of formally defined conditions on data that the learning model is required to satisfy, human knowledge is often loosely defined by abstract concepts and could be independent of specific data.

The second generalization is that we review the works not restricted to only deep clustering but expends to general deep learning. Unlike traditional algorithms, a deep learning algorithm has the advantage of being flexible to its applications. Therefore, the work that uses expert constraints in a supervised problem such as classification can be transferred into the clustering task. Solutions with one neural architecture could be adapted to another structure.

In summary, Chapter 3 consists of two sections. Section 3.1 introduces the fundamentals of deep learning with knowledge and describes several algorithms which have the potential to be applicable to constrained clustering. In contrast, Section 3.2 presents works focused directly on constrained clustering using deep neural networks.

3.1 Deep Learning with Knowledge

3.1.1 Overview

Recently, some artificial intelligence systems have achieved their performance on par with humans in sensory tasks such as image recognition, object detection, or language translation. However, it rather shows the machine as a diligent learner with a high volume of training data and a huge amount of energy consumption. Moreover, pure data-driven models can lead to unexpected behaviors such as classification on the wrong labels with high confidence [39], bias or unfairness [64]. Integrating human knowledge into machine learning can help reduce the data required and make the model more reliable and understandable.

Knowledge in machine learning can be categorized as general knowledge and domain knowledge. General knowledge involves computer science, statistics, neural science, etc which is independent of the task and data domain. In contrast, domain knowledge is related to the domain of the data and specific fields such as physics, economics, and biology.

Because the clustering task often focuses on a specific application, domain knowledge plays an important role in improving the clustering result and making the algorithm specialize. In the next section, we present the main methods to integrate knowledge in deep learning. Then, several algorithms are described, and our analysis for application in constrained clustering.

3.1.2 Methods for Knowledge Integration

As stated in [52], prior knowledge can be integrated into: pre-process, construction of the neural model or learning process. Figure 3.1 shows the general learning process with the domain knowledge.

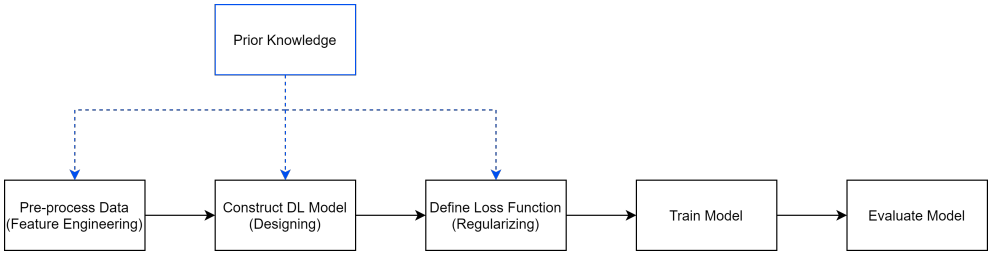

Figure 3.1: The workflow of deep learning with prior knowledge. The knowledge can be infused in one or more of the first three steps.

a. Feature Engineering

Early on the development of machine learning, feature engineering is relied on human knowledge and experience in the domain of data [111]. For example, there are rich methods to extracting features in the image task, such as: Local Binary Patterns [75], Histogram Of Oriented Gradient [20], Scale Invariant Feature Transform [63], etc. However, the feature learning (also known as representation learning), which let the machine to discover and extract features automatically, has proven its superiority in many tasks.

Integrating domain knowledge in feature engineering such as invariance [43], pairwise constraints [6] is shown in the deep clustering (Section 2.2) and the deep constrained clustering (Section 3.2).

b. Designing Network Structure

As stated in Section 2.1.2, a Convolutional Neural Network (CNN) is a neural network designed with the knowledge of image properties. The shared weights in

the convolution layers show that the class/cluster of images should be the same under the shifting translation of the objects they contain.

Modifications of deep network structures have been proposed for invariance of rotation [100] and scaling [37] transformations.

c. Regularizing via a knowledge loss

The aforementioned approaches prove the useful application of some specific domain knowledge. However, a regularizing loss is the most common way to learn domain knowledge for a deep neural network. This approach utilizes the core advantages of a deep architecture which is to learn complex tasks using gradient descents and backpropagation.

Most of the knowledge losses are designed for a specific type of knowledge such as pairwise, cluster-size. In this chapter, we are more interested in the works to handle a general form of knowledge, in particular the logical form.

Logical forms of Knowledge Logical representation is a typical representation from domain knowledge.

Hu et al. propose to express the knowledge in first-order logic and to distill them into a neural network through regularization [44].

The formulations are based on input variable $\mathbf{x} \in \mathcal{X}$ and target variable $\mathbf{y} \in \mathcal{X}$. In the clustering problem, $\mathcal{Y} = \{0, 1\}^k$ is a one-hot encoding space of the clusters. The author expresses each knowledge by a set of first-order logic rules with confidences $\mathcal{R} = \{\forall l \in [1, L] : (R_l, \lambda_l)\}$. R_l is a logical sentence over the input-target space $(\mathcal{X}, \mathcal{Y})$ while $\lambda_l \in \mathbb{R}_{[0,\infty]}$ is the confidence level ($\lambda_l = \infty$ indicating that the rule must be true).

The two published works in this research direction are presented in the following sections. The first method, which appears in "A Semantic Loss Function for Deep Learning with Symbolic Knowledge", is based on logic inference, while the second method uses a logic embedding to calculate the degree of the system's output follows knowledge indirectly.

3.1.3 Semantics loss

The work from [104] defines a knowledge loss, which is able to enforce the constraints on the target variables (the output layers of a neural network). The interest of this paper is an unique mechanism for translating knowledge from the representation of the knowledge in propositional logic to a regularizing loss.

a. General framework

The neural structure of the framework is a feed forward network where each neural output $p_i (i \in [1, n])$ is associated with a logic variable X_i. Then, the knowledge is represented by a propositional logic formula on the variable set $\mathbf{X} = \{X_1, ..., X_n\}$.

Compared to the representation of Hu[44], this representation does not include the logical variables representing the input data. Therefore, knowledge expressed in this framework is a set of conditions concerning the output of the neural networks for all the input data.

One-hot encoding is an example for this kind of knowledge. Regardless of the inputs, the system should predict one and only one class in a classification problem. This constraint (also called exactly-one constraint) is expressed by the two logical sentences as follows:

- At most one variable is true: $\forall i, j \in [1, n] \wedge i \neq j : \neg(X_i \wedge X_j)$
- At least one variable is true: $X_1 \vee ... \vee X_n$

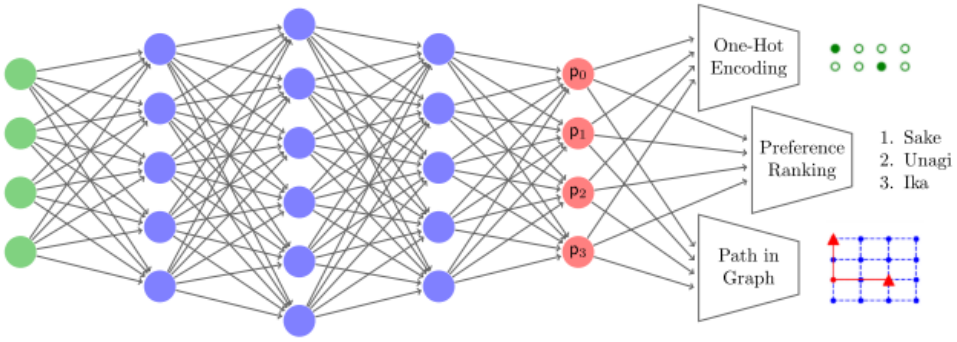

Figure 3.2: Feeding neural outputs into a semantic loss function for one-hot encoding, preference ranking and path in graph constraints. Source: Jingyi Xu[104]

Let α be the logical sentence representing the knowledge related to neural outputs $\mathbf{p} = \{p_i : i \in [1, n]\}$. α is expressed using $\mathbf{X}$ and logical operators, for example: $\alpha = \neg(X_1 \wedge X_2) \vee X_3$. The semantic loss for integrating knowledge in the neural network is formulated as follows:

$$L(\alpha, \mathbf{p}) \propto \log \sum_{\mathbf{x} \models \alpha} \prod_{i:\mathbf{x} \models X_i} p_i \prod_{i:\mathbf{x} \models \neg X_i} (1 - p_i) \tag{3.1}$$

where:

- $\mathbf{x}$ is an instantiation to all variables $\mathbf{X}$.
- $\mathbf{x} \models \alpha$ means that $\mathbf{x}$ satisfies α, for example: when $\mathbf{x} = X_1 \wedge X_2 \wedge X_3$, and $\alpha = \neg(X_1 \wedge X_2) \vee X_3$

The general framework of the semantics loss is depicted in Figure 3.2.

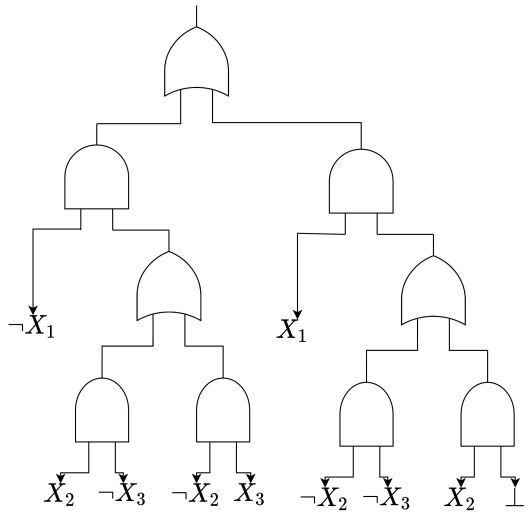

Figure 3.3: A SDD representation of exactly-one constraint with three variables. $\alpha = [\neg X_1 \wedge ((X_2 \wedge \neg X_3) \vee (\neg X_2 \wedge X_3))] \vee (X_1 \wedge \neg X_2 \wedge \neg X_3)$.

b. Constructing the Semantic Loss

The sum of weights for all instantiations of $\mathbf{x}$ that satisfied α in 3.1 can be represented as a Weighted Model Counting (WMC) problem, usually used in automated reasoning task [18]. Using the technique of circuit compilation in [23], a canonical representation of α, is called a *Sentential Decision Diagram* (SDD), can be built. An example of exactly-one constraint for three points are shown in Figure 3.3.

After the SDD is constructed, the semantic loss can be converted into an arithmetic circuit by changing AND gates into multiplications ($\times$) and OR gates into additions ($+$), as depicted in Figure 3.4. The translation shows that the computational cost for computing the value and the gradient of the semantic loss is linear with the size of the circuit. However, the size of a SDD depends on the form and number of variables of α. Given a Conjunctive Normal Form (CNF) with n variables and treewidth ω, then the size of the SDD is $O(n2^{\omega})$ [23].

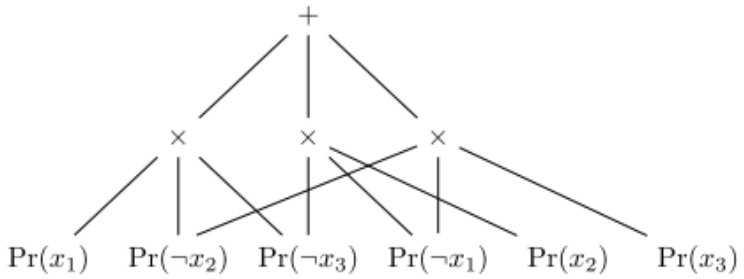

Figure 3.4: The corresponding arithmetic circuit for exactly-one constraint with 3 variables. Source: Jingyi Xu[104]

c. Application

This framework has been applied to different tasks.

Classification problem with one-hot encoding For classification problem, they add the semantic loss of the exactly-one constraint to the total loss . So, the total loss need to be minimized is:

$$L = L_{classification} + \omega \times L^s(exactly - one, \mathbf{p}) \tag{3.2}$$

Grids Given graph $G = (V, E)$: 4-by-4 grid with randomly removed edges. The problem is to find a shortest path in the graph G from a source s to a destination d.

We encode G into a binary vector $\mathbf{I} = \{I_1, ..., I_{|V|+|E|}\}$ of length $|V|+|E|$, where:

$I_s = I_d = 1$ marks the source and destination. Otherwise, $\forall v \in [1, |V|] \wedge v \notin \{s, d\} : I_v = 0$.

$I_{|V|+e} = 1$ for $e \in [1, |E|]$ when the edge e is removed.

They put $\mathbf{I}$ through 5-layer MLP output $|E|$ neurons indicate wherever edges are in the shortest path.

The constraints used in Grids are from Nishino et al.[73] for graph substructures.

Preference learning Given a set of user features, the aim is to predict how the user ranks his preferences over a list of items. For instance in Sushi dataset, the input is an order of 6 types of sushi and the output is the ordering of 4 other types.

The output is a binary matrix $O_{ij}, i, j \in [1, n]$ where $O_{ij} = 1$ means that item i is at position j. A set of constraints is put to make sure O is a valid ordering. The constraint set includes:

- Each item has exactly one position.

$$\forall i \in [1, n] : \sum_{j=1}^{n} O_{ij} = 1$$

- Each position has exactly one item.

$$\forall j \in [1, n] : \sum_{i=1}^{n} O_{ij} = 1$$

d. Summary

This work defines a semantic loss for logical requirements on the output of a deep learning model (neurons in the last layer). Their experiments show that the machine is able to improve the satisfaction of constraints on for instance a basic condition for classification (exactly-one constraint) or complex constraints such as graph substructures.

The main weakness of the model is the complexity for calculating the loss. Even with the support of SDD structures, both complexities for constructing and computing the derivative of the loss are intractable. They are exponential to the number of variables in each constraint which is equal to n - number of neural outputs. The following section 3.1.4 will present another work that can address this issue through a logic embedding network.

In the context of deep constrained clustering, the expert constraints are conditions on multiple input points. So, they need to be expressed on multiple outputs. Therefore, this proposal cannot be applied straightforward. We propose an adaptation of this semantics loss for the deep constrained clustering, it will be presented in Chapter 5.

3.1.4 Embedding symbolic knowledge

a. Introduction

While the previous work constructs the knowledge loss in a direct and intractable way, Xie et al. in "Embedding Symbolic Knowledge into Deep Networks" [102] use a graph embedding network to learn the degree of satisfaction of several propositional formulae on the output neurons (the neuron can be considered as a fuzzy logic variable). The solution is applied to a Visual Relation Prediction (VRP): the task is to detect objects and their spatial relations in an image.

First, a logic embedder is trained to produce a representation of formulae and their assignments, so that the satisfying assignments of each formula are close together. After the learning, the actual knowledge is put into the graph embedding network. Its output can be used to regularize the main network.

b. Network architecture

Let us denote:

- h is the function presenting output prediction of a neural network. $h(X)$ is a vector output of X.
- $\mathcal{G}$ is the logical sentence that predictive system h need to satisfy.
- q is the logic embedder. $q(\mathcal{G})$, $q(h(X))$ is the embedding points presenting the constraint formula $\mathcal{G}$ and X, respectively.
- L_{pred} : the prediction loss (or clustering loss in case of unsupervised), and L_{logic} : the logic loss of X given $\mathcal{G}$

The general learning scheme with logic embedding is shown in Figure 3.5.

c. Training logic embedder

General steps for embedding logic graph:

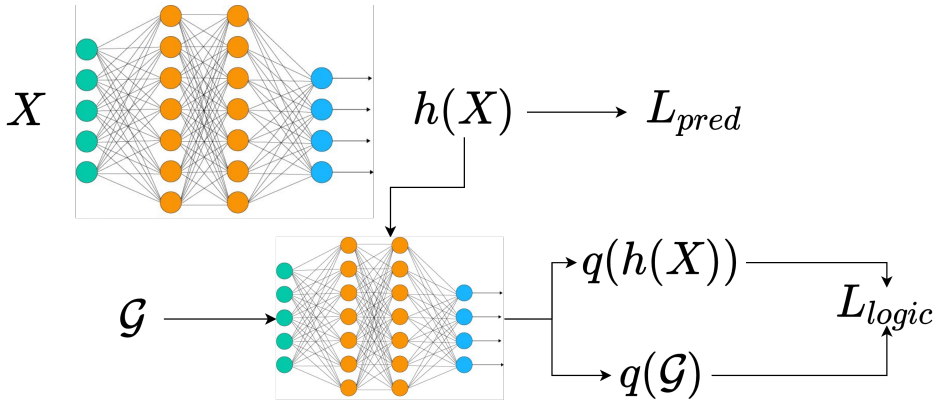

Figure 3.5: Framework for learning with logic embedding

1. Representing the knowledge as a formula $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where nodes are either logic variables (leaf nodes) or logical operator ($\wedge, \vee, \implies$) (intermediate nodes).

2. Create a neural network (noted as a q function) based on $\mathcal{G}$ with the input is weights of logic ($h(x)$) and the output is a embedding point $q(h(x))$.

Denote by $q(\mathbf{F})$ the embedding result for a given formula. $q(\tau_T), q(\tau_F)$ are the assignment embeddings for a satisfying and unsatisfying assignment, respectively. The loss for a formula $\mathbf{F}$ given a single pair of (τ_T, τ_F) is expressed as a triplet loss:

$$l_{triplet}(\mathbf{F}, \tau_T, \tau_F) = \max\{d(q(\mathbf{F}), q(\tau_T)) - d(q(\mathbf{F}), q(\tau_F)) + m, 0\} \tag{3.3}$$

where $d(x, y)$ is the Euclidean distance between x and y, m is the margin.

To obtain all τ_T and τ_F, they use a SAT solver. Then, pairs of assignments are randomly sampled for each formula during training. Finally, the embedding loss for training q is:

$$L_{emb} = \sum_{\mathbf{F}} \sum_{\tau_T, \tau_F} l_{triplet}(\mathbf{F}, \tau_T, \tau_F) + \lambda_r l_r(\mathbf{F}) \tag{3.4}$$

where F is a formula related to $\mathcal{G}$ (the whole $\mathcal{G}$ or sub-part of it), $l_{triplet}$ is the triplet loss in 3.3, l_r is a semantic regularization term, and λ_r is a hyperparameter that controls the strength of the regularization.

d. Integrating logic loss into the learning process

The target model h is trained by the predictive loss and the logic loss:

$$L = L_{pred} + \lambda L_{logic} \tag{3.5}$$

where L_{pred} is the prediction loss, $L_{logic} = \|q(\mathbf{F}_X) - q(h(X))\|$ is the embedding distance between the predictive distribution $h(X)$ and the constraint $\mathbf{F}_X$.

e. Summary

Although the logic embedder is not always correct as of the direct calculation with WMC, the network is accurate enough to boost the performance of the VRP model. In their work, the model is only compared with another neural logic embedder. It outperforms the baseline when all logic clauses are converted into decision - Deterministic Decomposable Negation Normal Form (d-DNNF). This experimental result proves the dependency of the logical form they use on the logical embedder.

For a further study, I think that it would be interesting to compare this work of Xie et al. to the work of the semantics loss[104].

3.2 Constrained Clustering using Deep Learning

3.2.1 Approaches

The common approach for constrained clustering using neural networks is to integrate a constraint loss in the deep clustering learning procedure. Table 3.1 shows a quick summary of previous published works.

Table 3.1: A brief survey of neural-based constrained clustering algorithms.

Neural structure	Algorithm	Clustering loss	Aux loss	Constraint types (losses)
CNN	NNC [42]	-	-	Pairwise
AE	Ts2DEC [46]	Cluster hardening (from DEC[101])	Reconstruction	Triplet
AE	COT[36]	WCSS in the feature space	-	Cluster size (Optimal Transport)
AE	DCC [109]	Cluster hardening (from DEC[101])	Reconstruction	Pairwise, Triplet, Cluster balance

Notation As mentioned in the previous chapter, common deep clustering methods provide the distributions of a point to different clusters. It is denoted as $\mathbf{Q} = \{Q_i : i \in [1, n]\}$ where $Q_i = \{q_{i1}, \cdots, q_{ik}\}$ is the distribution of point $\mathbf{x}_i$ and q_{ih} is the probability of assigning the point i to $\mathscr{C}_h$.

3.2.2 Pairwise constraints

a. Introduction

A pairwise constraint is a relation between the two points $\mathbf{x}_i$ and $\mathbf{x}_j$ with respect to the partition. It has two sub-types: must-link (ML) and cannot-link (CL). The must-link constraint requires the two points to be in the same cluster, while the cannot-link constraints demand they are in different clusters.

As the most well-known constraint type, there have been several works on deep clustering with pairwise constraints [87, 34, 108, 109, 42]. Based on their novelties and performances, we consider the following works.

- Deep Constrained Clustering (DCC) of Zhang et. al. [109].
- Neural network-based clustering using pairwise constraints (NNC) of Hsu and Kira [42].

The point in common between the two models is that both learn with a pairwise loss. Moreover, the loss is designed based on the similarity/dissimilarity of the assigned probability Q_i, Q_j of the two ML/CL points $\mathbf{x}_i$, $\mathbf{x}_j$.

NNC intends to compare its performance using pairwises (partial labels) with the result of a classification model using ground-truth data. Therefore, NNC uses a convolutional network - a common neural structure for image classification. The learning of NCC involves only the pairwise loss. The neural structure of NCC is depicted in Fig. 3.6.

DCC aims at learning a new representation of the input data with expert constraints. So, the model is based on IDEC, an autoencoder model with a representation loss (reconstruction of AE) and a clustering loss. DCC adds the expert loss as a third loss and trains them all together. The general framework of DCC is shown in Fig. 3.7. Beside the pairwise constraint, DCC also covers triplet and balance cluster constraint which will be explained in the next sections.

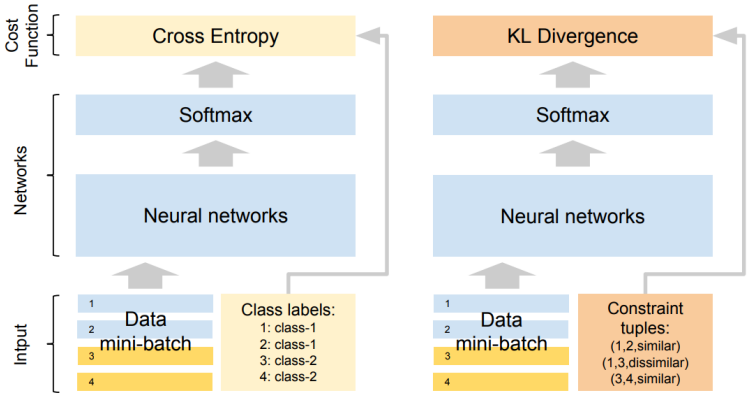

Figure 3.6: NNC model (right) based on a classification network (left). Source: Yen-Chang Hsu[42]

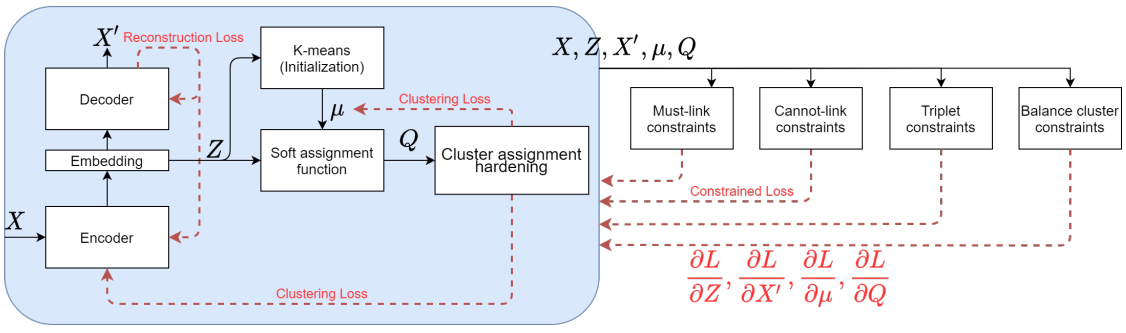

Figure 3.7: DCC framework based on IDEC (blue area) and adding constrained losses for several constraint types

b. Pairwise losses

Dot product Zhang et. al. [109] utilize the product between the two distributions Q_i, Q_j of the pairwise points $ML/CL(i,j)$ to define the must-link loss and the cannot-link loss.

$$\begin{aligned} L_{ML}^{dot} &= -\sum_{(i,j)\in ML} \log \sum_{h\in[1,k]} q_{ih}q_{jh} \\ L_{CL}^{dot} &= -\sum_{(x_i,x_j)\in CL} \log \left(1 - \sum_{h\in[1,K]} q_{ih}q_{jh}\right) \end{aligned} \tag{3.6}$$

KL-divergence The authors in [42] observe that the must-link constraint $ML(i,j)$ should be translated into the similarity of their corresponding distributions Q_i, Q_j. Similarly, the cannot-link constraint should enforce the dissimilarity between Q_i and Q_j. Therefore, they use the Kullback-Leibler (KL) divergence to measure the pairwise constraint loss.

$$L_{pw}^{KL} = \sum_{(i,j)\in ML} KL(Q_i\|Q_j) + \sum_{(i,j)\in CL} \max(0, margin - KL(Q_i\|Q_j)) \tag{3.7}$$

Unlike Eq. 3.6, the KL-divergence is not symmetric so (i,j) and (j,i) are considered as two different pairs.

This loss is softer than the loss with the dot product in Eq. 3.6 because $L_{pw}^{dot} = 0 \implies L_{pw}^{KL} = 0$ but the reverse is not true.

c. Training process

Hyper-parameters While the two losses for must-link and cannot-link in Eq. 3.6 have no hyper-parameters, the authors consider these two losses separately but add coefficients for weighting the losses. In the experiment, they select the two coefficients based on a validation set.

The value of $margin$ in Eq. 3.7 is also selected from the performances on a validation set.

Additional losses With a learning scheme similar to IDEC [40], DCC also has a clustering and reconstruction loss. While Hsu and Kira [42] do not include any additional loss into the model (we denoted this model as NNC).

Mini-batching technique DCC cuts the set of must-link (cannot-link) into batches with fixed sizes. All pairs in a group are forwardly computed to get the set of pairs $(Q_i, Q_j) : (i,j) \in ML/CL$. Then, the gradient in a mini-batch is computed, and the back-propagation process updates the network parameters.

NNC has a similar approach with mini-batch gradient descent. But it only computes once each point that appears in the mini-batch. Therefore, it avoids the redundancy of calculating the same instances which appear in multiple constraints.

3.2.3 Triplet constraints

a. Introduction

A triplet constraint on 3 data points $\mathbf{x_a}$, $\mathbf{x_p}$ and $\mathbf{x_n}$ (respectively called anchor, positive and negative points) means that $\mathbf{x_a}$ is more similar to $\mathbf{x_p}$ than to $\mathbf{x_n}$.

To our knowledge, DCC[110] and Ts2DEC[46] are the only works integrating the triplet constraints in deep networks. Both of them are based on DEC/IDEC which, we recall, is based on an autoencoder structure with a reconstruction loss and a cluster hardening loss.

b. Triplet losses

In Ts2DEC, given a triplet constraint (x_a, x_p, x_n), the optimization tries to make the Euclidean distances in the embedding space between $\|f(x_a) - f(x_n)\|$ and $\|f(x_a) - f(x_p)\|$ greater than a constant α.

$$\forall (x_a, x_p, x_n) \in Triplet : \|f(x_a) - f(x_n)\| - \|f(x_a) - f(x_p)\| \geq \alpha \tag{3.8}$$

The triplet loss is the hinge loss.

$$L_{triplet}^{Ts2DEC} = \sum_{(x_a, x_p, x_n) \in T} \max(0, \|f(x_a) - f(x_p)\| - \|f(x_a) - f(x_n)\| + \alpha) \tag{3.9}$$

DCC measures the similarity in terms of the vectors of probability distribution of a point to the clusters. Given Q_a, Q_p, Q_n the distribution vectors of $\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n$, the triplet loss is defined as follows:

$$L_{triplet}^{DCC} = \sum_{(x_a, x_p, x_n) \in T} \max(0, d(Q_a, Q_n) - d(Q_a, Q_p) + \theta) \tag{3.10}$$

where $d(Q_i, Q_j) = \sum_{h \in [1,k]} q_{ih} * q_{jh}$

c. Training process

While DCC jointly trains the reconstruction loss, the clustering loss, and the triplet loss, Ts2DEC trains the system in two steps. The first step is to train with the reconstruction loss and the triplet loss. The network is trained with the triplet loss and the clustering loss in a second step with a parameter λ. The two steps of the learning are shown in Fig. 3.8.

Hyper-parameters In Zhang et al. [110], there is not ablation study. It means that they do not report the results with different hyper-parameter settings. So, we do not know the sensitivity of hyper-parameters with respect to the clustering quality.

The authors in [46] do a sensitivity study of λ - the coefficient of the triplet loss. It shows that the triplet constraints do not affect the model when λ is small. In contrast, a high λ leads to reduce performance.

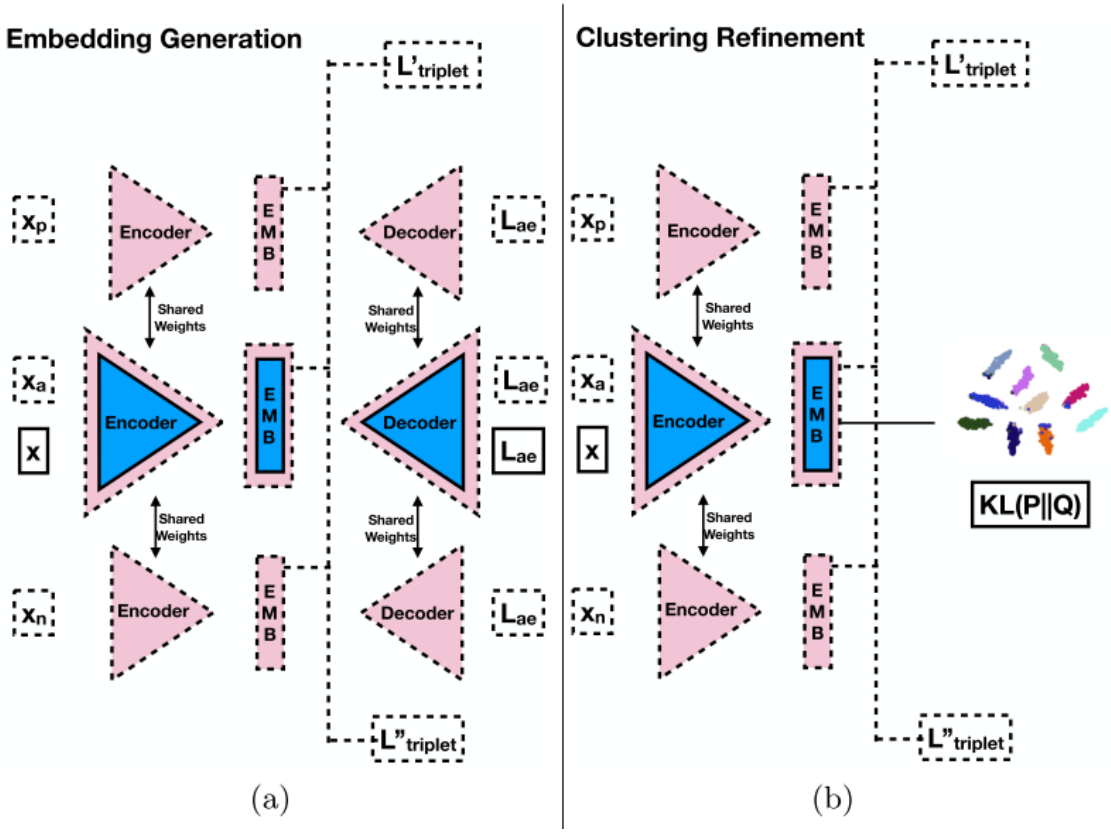

Figure 3.8: The learning process of Ts2DEC. Blue color is the original process of DEC. The constrained learning integrate in both representation learning process (a) and cluster learning process (b). Source: Dino Ienco[46]

d. Conclusion

A triplet constraint gives information on the similarity between three points. This is a valuable guideline for learning the representation of the data. Thus, Ts2DEC translates this information to a distance comparison in the embedding space of an autoencoder. On the other hand, DCC compares the assignment probabilities. Therefore, Ts2DEC gives a more natural formulation of triplet constraints than DCC. In their publication, Zhang et al. do not provide a clear explanation for this choice. But we assume that it will have more impact on the clustering process, leading to better clustering quality.

On a broader view, complex constraints can lead to different interpretations. So, there is a need for a formal way to translating any constraint definition into a constraint loss.

3.2.4 Cluster-size constraints

a. Introduction

A clustering size constraint is a type of constraint that puts a requirement on the cardinality of clusters. This constraint type can take several forms:

- The minimum cluster size constraint requires that each cluster has a number of points greater than a given threshold α: $\forall h \in [1,k], |\mathscr{C}_h| \geq \alpha$
- The maximum cluster size constraint requires that each cluster has a number of points lesser than a given threshold β: $\forall h \in [1,k], |\mathscr{C}_h| \leq \beta$
- The balance constraint (ratio control) requires that all clusters have approximately the same size or the fraction between the largest and smallest cluster must be greater than a given threshold θ: $\frac{\min_{i\in[1,k]}|\mathscr{C}_i|}{\max_{j\in[1,k]}|\mathscr{C}_j|} \geq \theta$

b. Solutions

DCC proposes a cluster-size loss based on the soft-assignment q_{ij}. The loss function is as follows:

$$L_{csc}^{DCC} = \sum_{h=1}^{k}(\frac{\sum_{i=1}^{n} q_{ih}}{n} - \frac{1}{k})^2 \tag{3.11}$$

This loss regulates the expectation of $q_{ih}, i \in [1,n]$ values to be $\frac{1}{k}$. This requires the number of instances to be large enough to be effective when learning the constraint. However, in the extreme case, when the loss is 0, the constraint may still not be satisfied.

In Differentiable Deep Clustering with Cluster Size Constraints [36], the authors formulate the clustering and the cluster-size constraint as a single problem.

$$\begin{aligned} L_{csc}^{OT} &= \min_{\pi\in\{1,\frac{1}{n}\}^{n\times k}} \sum_{h=1}^{k}\sum_{i=1}^{n} \|z_i - \mu_i\| \times \pi_{ih} \\ s.t. \pi \mathbf{1}_k &= \frac{1}{n}\mathbf{1}_n \\ \pi^T \mathbf{1}_n &= w \end{aligned} \tag{3.12}$$

where $w = (\frac{n_1}{n}, \cdots, \frac{n_k}{n})$ is the vector of cluster proportions.

c. Conclusion

Both of the two methods use the loss with a mini-batch gradient descent. So, it requires the batch size to be large enough comparing to the number of clusters so that the proportions amongst cluster cardinalities in a batch are similar to the whole dataset.

Another common point is that they only enforce the balance of cluster sizes. To learn imbalance proportions, we must find a way to match the indexes of the clusters with the indexes of the classes.

Chapter 4

Constrained Postprocess with Clustering Score

As we have introduced in Chapter 1, most constrained clustering algorithms are in-process methods where the expert constraints are integrated during the clustering process. However, this approach will limit the flexibility of constructing a solution. For example, we could find a suitable clustering algorithm for our data, but there exists no previous work allowing constraint integration with that algorithm. This issue is even more problematic because different constraint types require different ways to integrate them, and the most advanced clustering algorithm is unlikely to have its constrained version yet.

If the expert constraint and the clustering are handled separately, there are two possible orders. The pre-process methods execute the expert constraint before the clustering. Because we manipulate the expert requirements without the knowledge of the partition, only a few constraints can be processed in this way. The relevant researches of this direction focus on the pairwise/triplet constraints, where they are used to adjust the presentations of the points or the distances between instances[6, 107].

In contrast, post-processing has a more natural order as it is used after the clustering process. This scheme has the disadvantage of not fully using the expert knowledge for learning clusters patterns. However, it has two major advantages. First, the post-process task is usually less complex than the in-process one. Thus, finding the result in post-process takes less time and computational power. Second, it is flexible. The experts can use different clustering algorithms to see which method is the most suitable for their data. Then, they can apply different constraints to have the desired grouping. In this case, they can enforce and lessen some constraints after examining the initial clustering, which is impossible for in-process clustering.

In this chapter, we are going to introduce our framework *"Constrained Post-process with Clustering Score"*. The first section 4.1 gives a quick presentation of the current post-processing method and our novelties and contributions to this approach. Then, Section 4.2 describes our framework: the variables, the requirements, and the objective criterion. In the next section 4.3, we give the formulations of constraint types including new constraints types. Moreover, we also give several practical scenarios where a new constraint type is necessary. Section 4.4 presents the empirical study of our framework.

4.1 Introduction

From our extensive research, Kuo et al.[58] is the only published work using post-processing method for the constrained clustering problem. Recalled from Section 1.5.4, they take a hard clustering $\mathbf{p}$ as the input of their postprocess.

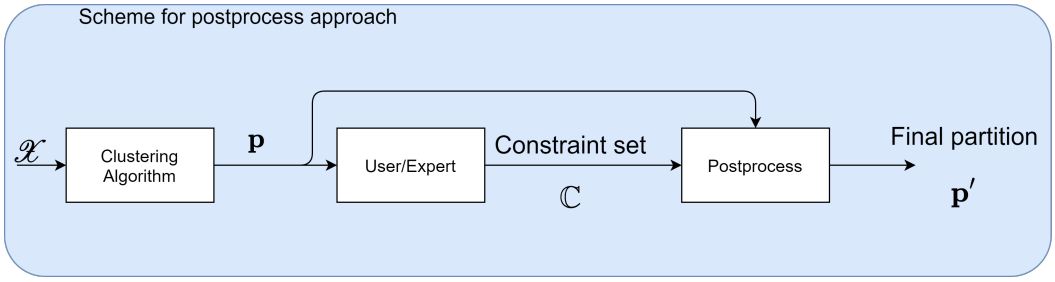

Figure 4.1: A workflow for a post-processing approach. The initial clustering $\mathbf{p}$ can be produced by any clustering algorithm.

The output clustering is the partition $\mathbf{p}'$ with minimum distance to $\mathbf{p}$ such that $\mathbf{p}'$ satisfies the constraint set $\mathbb{C}$. The distance between two partitions is the number of points the assignment of clusters differs.

$$d(\mathbf{p}, \mathbf{p}') = \sum_{i=1}^{n} \mathbb{1}[\mathbf{p}_i \neq \mathbf{p}'_i]$$

where $\mathbb{1}$ is the indicator function, $\mathbb{1}[\top] = 1$ and $\mathbb{1}[\bot] = 0$

This distance measure is simple and straightforward; however, it does not use any information from the clustering algorithm that produces these partitions. For example, given a clustering produced by K-means algorithm, the points that are near the cluster centers should less likely change than the outer points. So, compared to the initial clustering, a partition that changes the assignment of outer points should be more similar (the distance is closer) than a partition that adjusts instances near the centers. It is the reason why we introduce an *Assignment Score* which is the reward of allocating a point to a cluster.

4.1.1 Assignment Score

The assignment scores of all the input points to all the clusters are represented by a matrix that we call a cluster fractional allocation matrix (CFAM). As we introduce in 1.2, there are many techniques for clustering. Each method may or may not have a direct way to obtain the CFAM. In this section, we are going to present multiple ways to obtain CFAM for different classes of clustering techniques.

a. Centroid based clustering

Distance-based score In centroid based clustering methods, each cluster is represented by a center. Hence, the assignment score could be the distance of

the point to the cluster center. We denote the CFAM with distance scores as $D_{ij}, i \in [1, n], h \in [1, k]$

$$D_{ih} = \| \mathbf{x}_i - \mu_h \|$$

where μ_h is the center of $\mathscr{C}_h$.

Probability-based score Another method to obtain a CFAM is to use a the probability (soft) assignment. Let $S_{ih}, i \in [1, n], h \in [1, k]$ denote the probability of assigning the point $\mathbf{x}_i$ to cluster $\mathscr{C}_h$. Because each point has to be assigned to one and only one cluster in hard clustering, the assignment score between S_{ih} and $S_{ih'}$ are mutual exclusive and $\sum_{h=1}^{k} S_{ih}$ is equal 1.

The points $\mathbf{x}_i$ are distributed around the cluster centers μ_h in the centroid based clustering. This relation is often modeled by Student's t-distribution.

$$S_{ih} = \frac{(1+ \| \mathbf{x}_i - \mu_h \|^2 / v)^{-\frac{v+1}{2}}}{\sum_{h' \in [1,k]} (1+ \| \mathbf{x}_i - \mu_{h'} \|^2 / v)^{-\frac{v+1}{2}}} \tag{4.1}$$

where v is the degree of freedom.

Example In the example shown in Figure 4.2, the points are divided into 3 clusters: pink, cyan, and blue. We present the assignment scores of each point with a star marker. The point which is near its cluster centers has lower chance to assign to different clusters. For example, the left blue dot has 44% (17% + 27%) chance to assign to pink (third) cluster or cyan (first) cluster. In comparison, the probabilities of assigning the right blue point to the cyan cluster and the pink cluster are 21% and 34%, respectively.

b. Deep clustering

Autoencoder based methods The encoded layer in the Autoencoder are often a better representation of the data. So, the calculation is similar to the centroid based clustering but happened on the feature space.

If the embedded cluster centers is not given by the clustering algorithm, they can be computed based on the embedded points z_i and the clustering output $\mathbf{p}$.

$$\mu_h = \frac{\sum_{i:\mathbf{p}_i=h} z_i}{|\{i : \mathbf{p}_i = h\}|}$$

Then, the assignment score S_{ih} is expressed using z_i and μ_h.

$$S_{ih} = \frac{(1+ \| z_i - \mu_h \|^2 / v)^{-\frac{v+1}{2}}}{\sum_{h' \in [1,k]} (1+ \| z_i - \mu_{j'} \|^2 / v)^{-\frac{v+1}{2}}} \tag{4.2}$$

In fact, this soft-assignment score has been used in DEC[101], IDEC[40] for constructing the clustering loss.

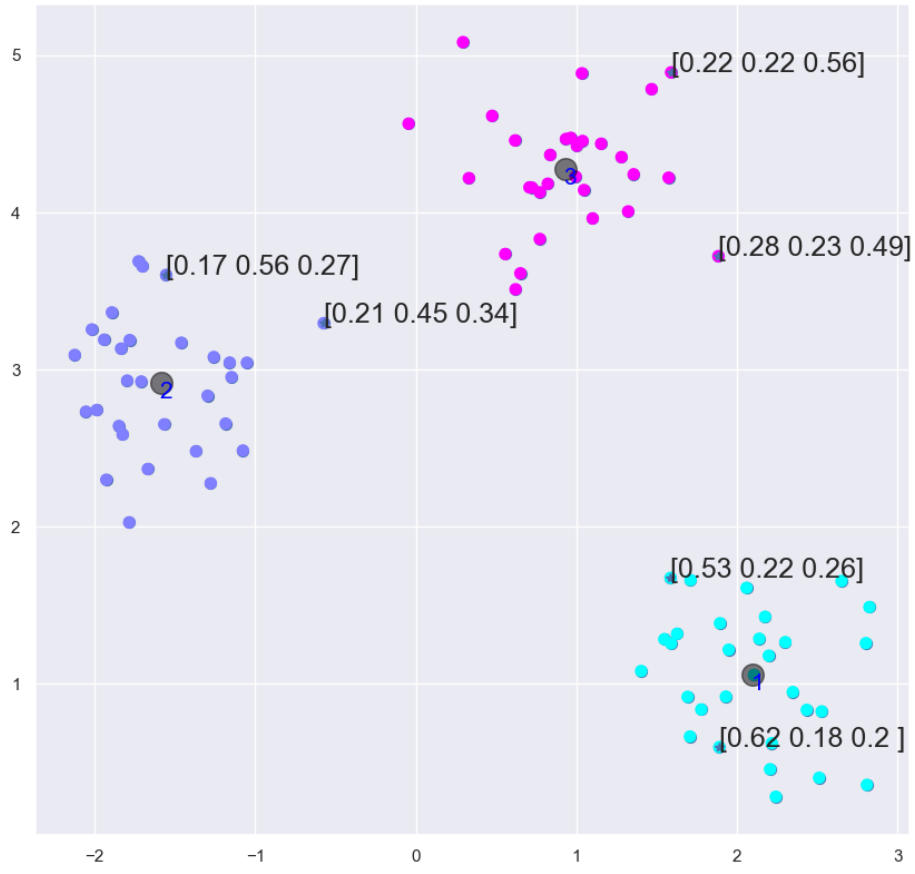

Figure 4.2: Assignment scores of a clustering using Student's t-distribution.

Classifier based methods The output of a classifier which is the result of a softmax function is already the probability of the input point to belong to each class (cluster). Hence, the CFAM is the concatenation of output vectors from the all input points.

c. Graph based clustering

A graph based clustering algorithm relies on a graph G that gives the connectivity between points and is represented by the adjacency matrix $A = \{i, j \in [1, n] : A_{ij}\}$. The weight $A_{ij} > 0$ is the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$. Therefore, it is natural to associate the cluster allocation matrix (CFAM) to the connectivity of the nodes.

Given $\{\mathscr{C}_1, \ldots, \mathscr{C}_k\}$ is the set of output clusters from the graph based clustering algorithm. The assignment score S_{ih} is the sum of similarity of the point i to other points in $\mathscr{C}_h$ over the total similarity of $\mathbf{x}_i$.

$$S_{ih} = \frac{\sum_{j \in \mathscr{C}_h} A_{ij}}{\sum_{j \in [1,n]} A_{ih'}} \tag{4.3}$$

d. Density based clustering

There has been a number of works concerning Soft/Fuzzy density based clustering [71, 88, 45]. These algorithms can be used directly to produce the assignment scores. In order to produce the scores, we could use the Fuzzy Border as defined in [45]. The clustering of a density based method is consists of the core points and the border points. First, the core points are associated with hard memberships (no fuzzy).

$$\begin{aligned} &\forall \mathbf{x}_i \in core(\mathscr{C}_h) : S_{ih} = 1 \\ &\forall \mathbf{x}_i \in core(\mathscr{C}_h), \forall h' \neq h : S_{ih'} = 0 \end{aligned} \tag{4.4}$$

Second, the membership function between border points and core points is defined through two values ϵ_{Min} and ϵ_{Max}. When the distance is smaller than ϵ_{Min}, the membership degree is equal to 1. While it is greater than ϵ_{Max}, its membership is minimum (equal to 0). It decreases linearly for value between ϵ_{Min} and ϵ_{Max}.

$$\mu(x_i, x_j) = \begin{cases} 1, \text{if} \|\mathbf{x}_i - \mathbf{x}_j\| \leq \epsilon_{Min} \\ \frac{\epsilon_{Max} - \|\mathbf{x}_i - \mathbf{x}_j\|}{\epsilon_{Max} - \epsilon_{Min}}, \text{if} \epsilon_{Min} < \|\mathbf{x}_i - \mathbf{x}_j\| \leq \epsilon_{Max} \\ 0, \text{if} \|\mathbf{x}_i - \mathbf{x}_j\| > \epsilon_{Max} \end{cases} \tag{4.5}$$

Then, the score of a border point x_i to a cluster $\mathscr{C}_h$ is expressed as:

$$S_{ih} = \min_{\mathbf{x}_j \in core(\mathscr{C}_h) \wedge \mu(\mathbf{x}_i, \mathbf{x}_j) > 0} \mu(\mathbf{x}_i, \mathbf{x}_j) \tag{4.6}$$

e. Summary

In Table 4.1, we summary possible ways to obtain a CFAM based on different types of clustering. Our method, therefore, can be used with a wide variety of clustering algorithms, including centroid-based, probabilistic, and deep learning. However, to achieve the best clustering, the experts can decide what instances are fixed and the degree of fuzziness.

4.1.2 Clustering Score

In this section, we propose methods to compute a clustering score from distance-based and probability-based measures.

Table 4.1: An overview of how to obtain a cluster fractional allocation matrix from several algorithm outputs.

Algorithm	**Method to obtain a CFAM**
Centroid based (i.e. K-means)	Distance to centroids / Student's t-distribution
Deep clustering (i.e. DEC)	Student's t-distribution on embedding layer / Output layer of the classifier
Graph based (i.e. Spectral Clustering)	Connectivity of nodes
Density Based (i.e. DB-Scan)	Fuzzy Border DBSCAN [45]

a. Distance-based score

Formulation The clustering score of a partition $\mathbf{p}$ is the sum of distances from all the points to their assigned cluster.

$$Score(\mathbf{p}, D) = \sum_{i \in [1,n]} D_{ip_i} \tag{4.7}$$

Properties The important properties of the clustering score are:

- The clustering score is increasing monotonic with respect to the number of points.

  Since $D_{ij} \geq 0$, $Score((p_1, p_2, ..., p_{n-1}), D) \leq Score((p_1, p_2, ..., p_{n-1}, p_n), D)$.

- The clustering score of any partition (of at most k clusters) is bounded by the matrix D.

$$\sum_{i \in [1,n]} \min_{h \in [1,k]} D_{ih} \leq Score(\mathbf{p}, D) \leq \sum_{i \in [1,n]} \max_{h \in [1,k]} D_{ih} \tag{4.8}$$

b. Probability-based score

Formulation Under the assumption of independent cluster selection of each point, the probabilistic of a partition is the product of individual assignment.

$$Score(\mathbf{p}, S) = \prod_{i \in [1,n]} S_{ip_i} \tag{4.9}$$

As we can observe from the previous section, this assumption is reasonable. Although different partitions will lead to altering values of cluster centers or node-cuts, their changes are relatively minor if only a small percentage of nodes are changed in their assignment. Fortunately, it is often the case for the constrained clustering problem because most of the points are either not involved in any constraint or the constraint they are involved in is already satisfied.

Properties The important properties of the probability-based score are:

- The clustering score is decreasing monotonic with respects to the number of points.

  Since the bigger the number of points is, the smaller the value of the partition, with respect to the score. or in other terms $Score((p_1, p_2, ..., p_{n-1}), S) \geq Score((p_1, p_2, ..., p_{n-1}, p_n), S)$.

- The sum of all possible partitions (of at most k clusters) scores is equal to 1.

$$\sum_{\forall \mathbf{p} \in [1,k]^n} Score(\mathbf{p}, S) = 1 \tag{4.10}$$

  where $\mathbf{p}$ includes the empty set partition and the number of clusters is less than or equal k.

- The domain of probability-based score is $\mathbb{R}_{[0,1]}$.

  This is a direct consequence of previous properties and $Score(\mathbf{p}, S) \geq 0, \forall \mathbf{p} \in [1, k]^n$

Proof on the sum of probability-based scores
Let us denote by $f_n = \sum_{\mathbf{p}(n)} Score(\mathbf{p}(n), S)$ is total score of partition with n points. We are going to proof that f_n is not depend on S and $f_n = 1$

First, it is easy to see that $f_1 = 1$ so the formula is correct.

Now, assume that the formula is correct up until $i - 1$. Then, a partition of i points can be viewed as a partition of $i - 1$ and assignment of the point i.

So, we have:

$$\begin{aligned} f_i &= \sum_{h \in [1,k]} \left[\sum_{\mathbf{p}(i-1)} (Score(\mathbf{p}(i-1), S) \times S_{ih}) \right] \\ &= \sum_{h \in [1,k]} \left[\sum_{\mathbf{p}(i-1)} Score(\mathbf{p}(i-1), S) \right] \times S_{ih} \\ &= \sum_{h \in [1,k]} 1 \times S_{ih} \text{ (From assumption it is correct for } f_{i-1}) \\ &= \sum_{h \in [1,k]} S_{ih} \\ &= 1 \end{aligned} \tag{4.11}$$

Therefore, it is also correct for i.

4.2 The framework for Constrained Postprocess

4.2.1 Formulation of the problem

Given a set of points $\mathscr{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, a set of constraints $\mathbb{C}$, an unconstrained or constrained clustering algorithm $\mathcal{M}$. Our method to give the output partition $\mathbf{p}$ is as follows.

1. Obtain the CFAM matrix (D or S) by a suitable method for $\mathcal{M}$ in Table 4.1.

2. Find the partition $\mathbf{p}$ that satisfies the constraint set $\mathbb{C}$ and achieves the best clustering score, that may be

$$\begin{aligned} \text{either} \quad & \underset{\mathbf{p} \text{ satisfies } \mathbb{C}}{\arg\min}\ Score(\mathbf{p}, D) \text{ (distance-based score)} \\ \text{or} \quad & \underset{\mathbf{p} \text{ satisfies } \mathbb{C}}{\arg\max}\ Score(\mathbf{p}, S) \text{ (probability-based score)} \end{aligned} \tag{4.12}$$

4.2.2 ILP formulation

As it is stated in Chapter 1, satisfying all the types of constraints is typically a NP-Hard problem, we propose a formulation based on Integer Linear Programming for this problem.

Inputs.

- n - the number of points, k - the number of clusters.
- The set of constraints $\mathbb{C}$ that must be satisfied.

Variables.

- A matrix Z representing a hard assignment: $Z_{ik} = 1$ means that instance i is assigned to cluster k.

Objective. We aim at finding an assignment of instances to the most likely clusters while satisfying all the constraints. The problem is therefore finding a $N \times K$ matrix Z of $\{0, 1\}$ with the objective function

$$\arg\min \sum_{i=1}^{N} \sum_{k=1}^{K} D_{ik} Z_{ik}$$

or

$$\arg\min \sum_{i=1}^{N} \sum_{k=1}^{K} \log(S_{ik}) \times Z_{ik}$$

satisfying the constraints in $\mathbb{C}$

4.3 Constraint Formulations

4.3.1 Previous Constraint Types

The constraints in $\mathbb{C}$ can be of different types (see Section 1.3). They can be formulated as follows.

Pairwise Constraint A must-link (resp. a cannot-link) constraint on two instances i, j is formulated by $\forall h = 1, \ldots, k,\ Z_{ih} = Z_{jh}$ (resp. $Z_{ih} + Z_{jh} \leq 1$, for a cannot-link constraint).

Logical Triplet Constraint A triplet constraint $Triplet(a, p, n)$ is formulated by:

$$\forall h = 1, \ldots, k,\ Z_{ph} \geq Z_{ah} + Z_{nh} - 1$$

This formulation yields $Z_{ph} = 1$ if $Z_{ah} = Z_{nh} = 1$.

Cluster-Overlap Constraint Each instance belongs to at least α and at most β clusters can be expressed by:

$$\forall i = 1, \ldots, n,\ \alpha \leq \sum_{h=1..k} Z_{ih} \leq \beta$$

To enforce hard clustering, i.e. each instance belongs to a single cluster, α and β are set to 1.

Cluster Size Constraint Each cluster must contain at least α and at most β instances can be expressed by:

$$\forall h = 1, \ldots, k,\ \alpha \leq \sum_{i=1..N} Z_{ih} \leq \beta$$

Neighborhood Constraint A neighborhood constraint requires that each instance i must be in the same cluster with at least a ratio α of its neighborhood N_i. Let $N_i(j) = 1$ if instance j is in the neighborhood of i and 0 otherwise. This constraint is expressed by: $\forall i = 1, \ldots, n, \forall h = 1, \ldots, k$,

$$\sum_{j=1..n} N_i(j) Z_{jh} \ \geq\ \alpha \Big(\sum_{j=1..n} N_i(j) \Big) Z_{ih}$$

4.3.2 Attribute level Constraint

Property-Cardinality Constraint Let $p(i)$ be 1 when an instance i has a property p and 0 otherwise. The fact that each cluster must have at least α and at most β instances having the property p can be enforced by:

$$\forall h = 1, .., k,\ \alpha \leq \sum_{i=1..n} p(i) Z_{ih} \leq \beta$$

m-cluster group constraint is a special case of property-cardinality constraint where $\alpha = \beta = m$.

$$\forall h = 1, .., k, \sum_{i=1..N} p(i) Z_{ih} = m$$

The constraint such that in each cluster, the ratio of the instances having p over the size of the cluster is bounded by $[\alpha, \beta]$ is expressed by:

$$\forall h = 1, .., k, \ \alpha \times \sum_{i=1..n} Z_{ih} \leq \sum_{i=1..n} p(i) Z_{ih} \leq \beta \times \sum_{i=1..n} Z_{ih}$$

Attribute Level Constraint. An attribute-level constraint limits the number of possible clusters for the instances having a specific property. Let $p(i)$ be 1 when the instance i has the property and 0 otherwise. We introduce a variable $t_h \in \{0, 1\}$ for each cluster h, $t_h = 1$ if and only if the cluster h contains some instances having p. This is expressed by:

$$t_h \leq \sum_i p(i) Z_{ih} \ \text{ and } \ \forall i = 1, \ldots, n, \ t_h \geq p(i) Z_{ih}$$

The first constraint ensures $t_h = 0$ if there is no instance satisfying the property p in cluster h. The second sets $t_h \geq 1$ as soon as an instance having property p is in cluster h. Bounds $[\alpha, \beta]$ on the number of clusters containing the instances having the property p are given by: $\alpha \leq \sum_{h=1..k} t_h \leq \beta$.

4.3.3 Fairness

Fairness in clustering requires the algorithm to "treat" similar individuals equally which prevent any "discrimination" towards their membership in some groups. From this definition, several works has proposed different constraints to enforce fairness. However, group fairness [19, 11] and individual fairness[32, 51] are two main types. The group fairness focus on the similarity in statistical behavior for different demographic groups while the individual fairness requires individuals who are close together to be treated in the same way.

a. Group fairness

Definition Given a protected status variable (PSV) $\mathcal{P}$ with T values. $\mathscr{X}$ can be partitioned into T demographic groups as $\{G_1, G_2, \cdot, G_T\}$.

The work [19] proposed a measure of fairness for clustering with binary PSV as follows:

$$balance(\mathscr{C}) = \min_{\mathscr{C}_h \in \mathscr{C}} balance(\mathscr{C}_h) = \min_{\mathscr{C}_h \in \mathscr{C}} \min(\frac{N_h^1}{N_h^2}, \frac{N_h^2}{N_h^1}) \tag{4.13}$$

where $\mathscr{C} = \{\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k\}$ is clustering with K disjoint clusters, N_h^1 and N_h^2 represent the population of the first and second demographic groups in cluster $\mathscr{C}_h$.

Suman et al. extend the definition for multiple PSV through the two properties: restricted dominance (RD) and minority protection (MP)[11]. For each group G_i, we define two parameters $\beta_i, \alpha_i \in \mathbb{R}_{[0,1]}$. The RD requires that the fraction of instances from group G_i in any cluster is at most α_i while the MP requires that the fraction of instances from group G_i in any cluster is at least β_i

$$\forall i = 1, \ldots, T, \forall h = 1, \ldots, k, \beta_i \geq \frac{\|\mathbf{x} \in \mathscr{C}_h \cap G_i\|}{\|\mathscr{C}_h\|} \geq \alpha_i \tag{4.14}$$

The choice of β_i, α_i allows a more flexible usage while it is still able to represent the previous group fairness defined by [19].

Formulation in ILP Let denote $\mathcal{S}$ is the subset of $\mathcal{X}$. The set $\mathcal{S}$ are usually presented a minority group with a common attribute. The constraint for this set could be:

- Requiring minimum representation in clusters:
  $\forall h \in [1, k] : |\mathbf{x} \in \mathcal{S} \cup \mathcal{C}_h| \geq \alpha$
- Requiring maximum representation in clusters:
  $\forall h \in [1, k] : |\mathbf{x} \in \mathcal{S} \cup \mathcal{C}_h| \leq \beta$

b. Individual fairness

An individual fairness constraint requires that the ratio of neighborhood belong to the same cluster as the center point must be greater than α[51]. Denote $\mathcal{N}(i)$ is the set of all neighbor points of point i, we have:

$$\forall i = 1, \ldots, n, \forall h = 1, \ldots, k : \sum_{j \in \mathcal{N}(i)} Z_{jh} \geq \alpha \|\mathcal{N}(i)\| \times Z_{ih} \tag{4.15}$$

4.4 Experiments

4.4.1 Objectives and Outline

Our experiments attempt to address several core questions to understand how our work can be used in conjunctions with existing algorithms and its limitations and benefits. We attempt to address the following questions.

- Does our method improve the *result/output* of unconstrained and constrained clustering algorithms? (See Table 4.2)
- Is our method comparable with baseline constrained clustering methods? (See Tables 4.2 and 4.4)
- How useful are the new types of constraints and their use in combination? (See Tables 4.7 and 4.5)
- How does our method scale to large datasets? (See Tables 4.4, 4.7 and 4.5)

4.4.2 Experiment Setting

Datasets. We use 3 datasets, which are challenging and also used in a recent deep constrained clustering method [109].

MNIST: The dataset is composed of 60,000 handwritten single-digits, with a size of 28-by-28 pixels.

FASHION-MNIST: The dataset contains 60,000 images associated to a label from 10 classes.

REUTERS-10K: Reuters contains around 810,000 English news stories labeled with a category tree [60]. Following DEC [101], we consider only the single label documents belonging to the `corporate/industrial`, `government/social`, `markets` and `economics` categories. A subset of 10,000 examples is randomly sampled and the TF-IDF measure is computed on the 2000 most frequent words.

Baseline Algorithms. The following systems are used in our experiments.

IDEC: (Improved Deep Embedded Clustering) [40] a popular deep clustering method based on auto-encoder.

Kmeans: the classic algorithm but run on the deep embedding representation learned by IDEC.

COP-Kmeans: the classic constrained clustering algorithm for pairwise constraints but again run on the embedded space [97].

MSE-Kmeans: a modified K-means relying on a minimum-cost flow algorithm to satisfy cluster size constraints [91], which is run on the embedded space.

MCM: A postprocess algorithm [58] that uses the distance between two partitions as shown in Section 4.1.

DCC: Deep Constrained Clustering [109] which handles pairwise, triplet and balanced-clustering constraints during the clustering process.

Kmeans-Post, IDEC-Post, DCC-Post: our constraint post-processing method applied on the results of K-means, IDEC and DCC, respectively. IDEC and DCC output a cluster fractional allocation matrix (CFAM) which is then used in our method. For K-means, we generate the CFAM P as described in Section 4.1.1. Let the cluster centers be μ_h $(1 \leq h \leq k)$, the matrix P is computed by the t-distribution:

$$P_{ih} = -\log \frac{(1+ \parallel x_i - \mu_h \parallel^2)^{-1}}{\sum_{h'}(1+ \parallel x_i - \mu_{h'} \parallel^2)^{-1}}$$

For fairness, since IDEC is a probabilistic algorithm, we run it once and we gave the learned embedded representation to all the systems. Moreover, DCC is initialized with the network and the parameters learned by IDEC. All algorithms are implemented in Python. We use the ILP solver Gurobi version 8.0[1]. Experiments are run on a 2.8 GHz Intel Core i7 processor with 16GB of RAM. The source code is made available and easy to replicate[2].

Evaluation Metric. In all the datasets, the true class of objects is available and we use it as the ground truth to evaluate the accuracy of the clustering. We

[1]https://www.gurobi.com/

[2]https://github.com/dung321046/ConstrainedClusteringViaPostProcessing

consider two measures: Normalized Mutual Information (NMI) and clustering accuracy (ACC), with a one-to-one mapping between clusters and labels, computed by The Hungarian algorithm [56]. In both cases the higher the better.

4.4.3 Previous Constraint Types

In this section, we compare our performance with other methods for pairwise and cluster size constraints. In addition, we analyze the results of our framework when combining the two types of constraints.

a. Pairwise Constraints

We compare our method with baseline systems on MNIST and Fashion datasets with 3,600, 30,000 and 60,000 pairwise constraints. To measure performance, for each number of constraints we average performance over five sets of constraints and report the average and standard deviation over the five trials. Table 4.2 reports the results on MNIST. Our post-processing method with pairwise constraints always improves the partition in terms of NMI and accuracy, with the benefit of satisfying all the constraints. In contrast, postprocessing of MCM nearly does not change the initial clustering performances. Moreover, our algorithm always obtains better results compared to COP-Kmeans and comparable results to DCC. Similar results are also observed on the Fashion dataset (see Table 4.3).

b. Cluster size constraints

Here we compare our method on MNIST and Fashion, with MSE-Kmeans [91], which is developed specifically for cluster size constraints. We use the minimum and the maximum of the true class sizes as a lower bound and a upper bound on the cluster sizes for all the clusters. The results are shown in Table 4.4. The results show that our general method of incorporating this constraint is competitive compared to a method which is developed specifically for this type of constraints.

c. Constraint Combinations

Combinations of constraints. One benefit of our framework is that it can integrate and satisfy several types of constraints. Among existing constrained clustering methods, only declarative methods can integrate several types of constraints [25, 22] while satisfying them all. However they suffer from scalability and cannot handle datasets as big as MNIST and Fashion. In this part, we consider both pairwise (PW) and cluster size (CS) constraints simultaneously. The number of pairwise constraints is set to 30,000 for both runs.

Table 4.5 reports results for the MNIST and Fashion datasets. It reports the number of instances that have been assigned to a different cluster by the post-processing and the number of changes that have lead to the right cluster.

We notice a difference in behavior between MNIST and Fashion. For the first dataset, adding cluster size constraints improves the results while it is not true for

Table 4.2: Results with pairwise constraints on MNIST

#Pw	Method	NMI	ACC
0	Kmeans	0.8644 ± 0.0000	0.8838 ± 0.0000
	IDEC	0.8539	0.8799
3600	COP-Kmeans	0.8237 ± 0.0324	0.7372 ± 0.0630
	DCC	0.8637 ± 0.0012	0.8938 ± 0.0075
	Kmeans-Post	0.8649 ± 0.0001	0.8843 ± 0.0001
	IDEC-MCM	0.8518 ± 0.0001	0.8795 ± 0.0000
	IDEC-Post	0.8547 ± 0.0002	0.8804 ± 0.0001
	DCC-Post	**0.8640 ± 0.0013**	**0.8940 ± 0.0077**
30000	COP-Kmeans	0.8477 ± 0.0195	0.8302 ± 0.0314
	DCC	0.9407 ± 0.0032	0.9786 ± 0.0013
	Kmeans-Post	0.8689 ± 0.0003	0.8876 ± 0.0003
	IDEC-MCM	0.8458 ± 0.0005	0.8786 ± 0.0002
	IDEC-Post	0.8602 ± 0.0007	0.8839 ± 0.0005
	DCC-Post	**0.9429 ± 0.0026**	**0.9796 ± 0.0011**
60000	COP-Kmeans	0.8146 ± 0.0319	0.8039 ± 0.0644
	DCC	0.9549 ± 0.0029	0.9847 ± 0.0012
	Kmeans-Post	0.8739 ± 0.0004	0.8917 ± 0.0003
	IDEC-MCM	0.8453 ± 0.0005	0.8797 ± 0.0007
	IDEC-Post	0.8668 ± 0.0005	0.8887 ± 0.0004
	DCC-Post	**0.9581 ± 0.0021**	**0.9860 ± 0.0009**

Fashion. It can perhaps be explained by a tighter constraint (upper bound minus lower bound is smaller) for Fashion than for MNIST.

For the pairwise constraint, adding constraints has been shown in our experiments to consistently improve the quality of cluster, while the use of cluster size constraints needs more careful consideration. It is worth to study further for a way to relax this constraint when its impact is too high.

Table 4.3: Results with pairwise constraints on Fashion

#Pw	Method	NMI	ACC
0	Kmeans	0.6319 ± 0.0000	0.5877 ± 0.0000
	IDEC	0.6320	0.5879
3600	COP-Kmeans	0.6222 ± 0.0152	0.5808 ± 0.0092
	DCC	**0.6403 ± 0.0192**	**0.6378 ± 0.0277**
	Kmeans-Post	0.6306 ± 0.0003	0.5877 ± 0.0002
	IDEC-MCM	0.6255 ± 0.0004	0.5870 ± 0.0002
	IDEC-Post	0.6315 ± 0.0003	0.5880 ± 0.0002
	DCC-Post	**0.6402 ± 0.0191**	**0.6377 ± 0.0276**
30000	COP-Kmeans	0.6175 ± 0.0043	0.5974 ± 0.0199
	DCC	0.7421 ± 0.0158	0.7989 ± 0.0279
	Kmeans-Post	0.6253 ± 0.0005	0.5901 ± 0.0005
	IDEC-MCM	0.6020 ± 0.0008	0.5851 ± 0.0009
	IDEC-Post	0.6293 ± 0.0005	0.5905 ± 0.0003
	DCC-Post	**0.7446 ± 0.0159**	**0.8019 ± 0.0282**
60000	COP-Kmeans	0.6023 ± 0.0059	0.5853 ± 0.0175
	DCC	0.6430 ± 0.2882	0.7180 ± 0.2891
	Kmeans-Post	0.6219 ± 0.0007	0.5923 ± 0.0008
	IDEC-MCM	0.5883 ± 0.0016	0.5858 ± 0.0009
	IDEC-Post	0.6276 ± 0.0008	0.5940 ± 0.0008
	DCC-Post	**0.6624 ± 0.2705**	**0.7409 ± 0.2671**

Table 4.4: Clustering accuracy and NMI for clustering with cluster size constraints

Data - Method	NMI	ACC	Data - Method	NMI	ACC
MNIST - IDEC	**0.8539**	0.8799	Fashion - IDEC	**0.6320**	**0.5879**
MNIST - MSE	0.8536	**0.8816**	Fashion - MSE	0.5363	0.5387
MNIST - Post	0.8520	0.8796	Fashion - Post	0.5301	0.5425

Runtime. We report the runtime in seconds of MCM, COP-Kmeans, DCC and our method with pairwise constraints in Table 4.6. To have a fair comparison, we focus only on the computational time for integrating constraints to the initial clustering provided by IDEC without any constraints.

The runtime of each test mainly depends on the number of pairwise constraints. Compared to MCM, we use the same formulations and ILP optimizer, so our performances are similar to theirs. The computational time of MCM is slightly better because MCM function to optimize is simpler than ours.

Our method usually gives comparable results for quality, but it is substantially faster compared to in-process methods. On average, our computation is more than ten times faster than COP-Kmeans and 500 times faster than DCC. Indeed, COP-Kmeans has to compute the distance matrix after updating the cluster centers, whereas DCC has to apply backpropagation to update all the model parameters. Moreover, post-processing time has a smaller variance between each test than the

Table 4.5: Runtime (in seconds) with/without pairwise (PW) and cluster size (CS) constraints on MNIST and Fashion.

Cases	NMI	ACC	# Changes	# Positive changes	Runtime (s)
MNIST					
No constraint	0.85	0.88	-	-	-
CS	0.85	0.88	445.00	18.00	3.95 ± 0.12
PW	0.86 ± 0.00	0.88 ± 0.00	878.80 ± 21.50	507.00 ± 25.27	3.86 ± 0.32
PW + CS	0.86 ± 0.00	0.88 ± 0.00	1067.00 ± 18.77	596.20 ± 23.28	3.86 ± 0.32
Fashion					
No constraint	0.63	0.59	-	-	-
CS	0.53	0.54	8748.00	977.00	4.06 ± 0.21
PW	0.63 ± 0.00	0.59 ± 0.00	2747.40 ± 43.51	977.40 ± 13.84	3.51 ± 0.02
PW + CS	0.54 ± 0.00	0.55 ± 0.00	9600.80 ± 38.28	1580.20 ± 28.08	24.78 ± 2.76

Table 4.6: Runtime (in seconds) with pairwise constraints using COP-Kmeans, DCC and postprocess.

Data	#Pairwise	MCM	IDEC-Post	COP-Kmeans	DCC
MNIST	3600	0.98 ± 0.04	1.00 ± 0.04	132.38 ± 18.72	1013.76 ± 790.91
	30000	3.83 ± 0.17	3.86 ± 0.32	103.77 ± 56.39	1381.73 ± 1067.07
	60000	6.58 ± 0.33	6.81 ± 0.39	70.95 ± 38.41	3277.37 ± 1555.83
Fashion	3600	0.96 ± 0.05	0.99 ± 0.02	71.00 ± 57.77	5579.30 ± 2761.33
	30000	3.67 ± 0.10	3.51 ± 0.02	103.71 ± 35.26	7359.35 ± 3927.79
	60000	6.81 ± 0.44	6.55 ± 0.43	95.28 ± 37.90	3207.00 ± 1057.71

other methods.

Concerning other constraints, as given in Tables 4.7 and 4.5 our method performs in a very reasonable time.

4.4.4 Attribute level Constraints

This new type of constraints requires that the instances having a specific property cannot be widespread over a large number of clusters. We consider Reuters-10K and we require that documents that contain some given words should be covered by at most s clusters. We consider two cases with 5 (resp. 10) constraints by randomly selecting 5 (resp. 10) sets of three words, among those whose documents widespread on at most 2 clusters ($s = 2$). We post-process the initial clustering given by IDEC.

Table 4.7 reports the quality in NMI and accuracy (ACC), the number of instances involved in all the constraints, the number of constraints that are not satisfied by the clustering given by IDEC (it is the input clustering of our system, note that our system produces a clustering satisfying all the constraints), the number of documents that have been assigned to a different cluster after post-processing and the runtime in seconds. The impact of attribute level constraints is quite high. While a pairwise constraint only affects two instances, the average number of instances that are concerned in an attribute level constraint is around 200. The number of instances that have been reassigned is therefore also high. That could

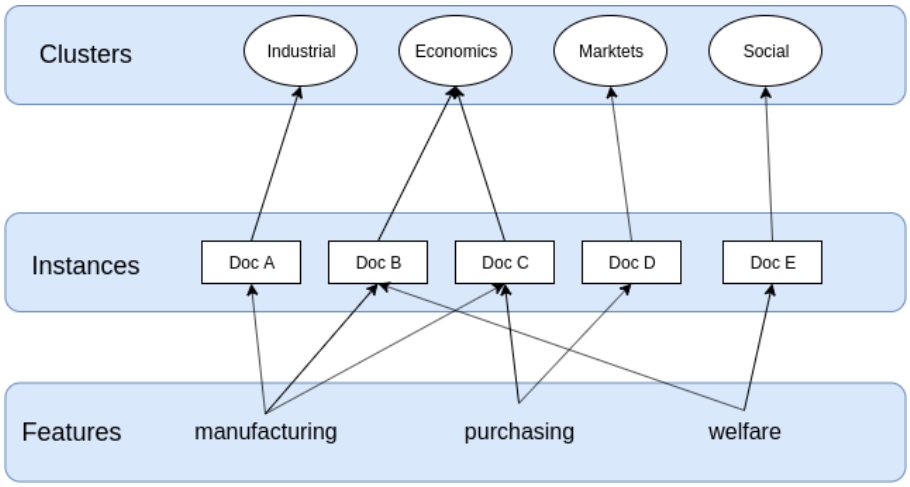

Figure 4.3: An example for Attribute level Constraints. Documents with "manufacturing" appear in the two clusters: Industrial and Economics.

Table 4.7: Impacts of attribute level constraints on Reuters-10K

#Constraints	0	5	10
NMI	0.5279	0.5253 ± 0.0039	0.5219 ± 0.0055
ACC	0.7452	0.7474 ± 0.0070	0.7499 ± 0.0088
#Involved Inst.	-	1168.0000 ± 50.5332	2128.0000 ± 143.6760
#Unsat Constr.	-	4.8000 ± 0.4000	9.8000 ± 0.4000
#Changed Inst.	-	96.6000 ± 57.4895	212.4000 ± 42.9679
Runtime (s)	-	0.0831 ± 0.0147	0.3052 ± 0.3010

explain the slight decrease of NMI and the slight increase of accuracy, the random constraints without specific domain knowledge could be too strong.

4.4.5 Improving the Fairness of Clustering Algorithms

a. Introduction

The previous section performs standard comparisons to show that our method is comparable in accuracy to existing methods. Here we show the real interest of our approach as it allows combining multiple constraints to address challenging problems such as fairness in clustering. The area of fair clustering has drawn much recent attention. Fairness in clustering can be classified into group fairness and individual fairness. Group-level fairness usually represents statistical fairness notions based on a protected status variable (PSV). Group fairness typically requires that in each cluster, the ratio of each PSV value is approximately equal to the ratio of this type in the whole dataset [19].

Individual-level fairness corresponds to requirements made to individuals. An example of individual fairness requires individuals who are close together to be treated in the same way [32].

Existing work usually ensures just one of these types of fairness, either group fairness [19, 5] or individual fairness [51]. In our best knowledge, no work has considered both individual fairness and group fairness. Taking advantage that our constraint post-processing method can integrate different types of constraints, both types of fairness can be ensured.

Dataset We consider the classic fairness dataset Adult with 48,842 instances. Data to cluster on is described by continuous attributes such as age or working hours, and PSV such as gender, education or marital-status.

b. Group and individual fairness

Group fairness is expressed by the requirement that in each cluster, the ratio of each type of instances is approximately the same as this ratio when computed on the whole dataset. The ratio of females in the dataset is about 33.15%. To ensure group fairness, we require that in each cluster, the ratio of females is between $0.3315 - \epsilon$ and $0.3315 + \epsilon$ with $\epsilon = 0.01$. This is ensured by property-cardinality constraints, as defined in Section 4.2.

To ensure individual fairness, we require that each instance i must be in the same cluster as at least 50% of the elements in its neighborhood N_i. For each instance i, the neighborhood N_i is defined by the set of instances having exactly the same education and occupation, and a difference in age less than or equal to 2. This requirement is ensured by neighborhood constraints, as defined in Section 4.3, with $\alpha = 0.5$. We prove that with $\alpha \geq 0.5$, if x and y have exactly the same value on the attributes used to define the neighborhood, then to satisfy the fairness constraints x and y must be in the same cluster.

Baseline individual fairness - Most Votes Greedy Method. In order to define a method that ensures individual fairness, as a baseline method, we have implemented a greedy method as follows. We iterate t times the following procedure: for each unfair instance x, find the cluster h that contains the most instances in the neighborhood of x and change the assignment of x to the cluster h. In the experiment $t = 10$.

Baseline group fairness - Fairlet We use the code produced by [5] which is an improvement of the method ensuring group fairness using fairlets [19]. Fairlets are subsets of objects that respect the given ratio between the two values of a binary attribute. They are computed first then clustering is achieved on them to ensure group fairness. For the dataset Adult, we require the minimum ratio of females over males is higher than 49.37% so that the lower bound for the percentage of females in each cluster is 33.05%.

Our post-processing method after K-means Let μ_h $(1 \leq h \leq k)$ be the cluster centers obtained from K-means with the input $X = \{x_i : i \in [1, n]\}$. Then,

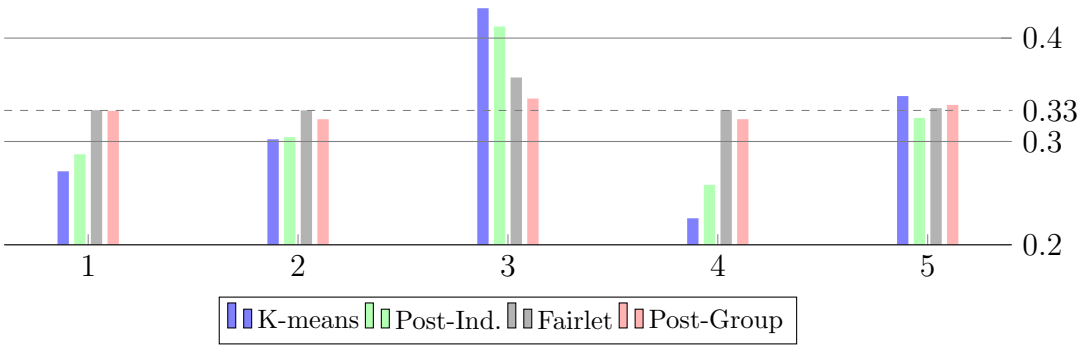

Figure 4.4: The ratio of females in each of the five clusters by different methods.

the allocation matrix P is computed using the t-distribution.

$$P_{ih} = -\log \frac{(1+ \parallel x_i - \mu_h \parallel^2)^{-1}}{\sum_{h'}(1+ \parallel x_i - \mu_{h'} \parallel^2)^{-1}}$$

Finally, we optimize P under three scenarios: only with individual constraints (Post-Ind.), only with group constraints (Post-Group), and both (Post-Combine).

c. Results and analysis.

Table 4.8 reports the result obtained on 10 runs with $k = 5$. It reports the clustering quality in terms of the within cluster sum WCS, the number of instances that are unfairly grouped according to individual fairness, the number of clusters that are unfair according to group fairness and the runtime in seconds. The within cluster sum WCS is defined by the sum of squared distances from each instance to the centroid of its cluster.

K-means gives a clustering that is unfair with respect to both group fairness and individual fairness. As expected, ensuring fairness decreases the clustering quality measured by WCS. However, post-processing achieves better quality than the greedy method. We can observe that group fairness and individual fairness are not compatible, ensuring one type of fairness does not ensure the other type, but even worsen. For instance, Figure 4.4 shows that clusters computed while ensuring only individual fairness (Post-Ind.) are group unfair (see for instance Cluster 4 with a very low rate of females).

On the other hand, Table 4.8 shows that Fairlet and Post-Group methods that ensure group fairness do not ensure individual fairness, even the number of unfair individuals is higher than the traditional K-means (around 7812 and 5809 respectively, compared to 5686). For group fairness, without an upper-bound constraint, Fairlet sometimes produces unfair groups (averaging 0.6 group per test case) while Post-Group ensures both bounds.

In terms of efficiency, the post-processing performs with a very reasonable runtime.

Table 4.8: Runtime and constraint satisfaction with individual fairness or/and group fairness.

Method	WCS	# Indiv. unfair inst.	# Group unfair clust.	Runtime (s)
K-means	8477.14 ± 1.89	5685.60 ± 21.12	5.00 ± 0.00	2.62 ± 0.30
Individual-level fairness				
Most-vote	9071.63 ± 1.84	113.10 ± 5.68	4.70 ± 0.46	18.06 ± 0.71
Post-Ind.	9064.86 ± 1.83	0	4.00 ± 0.00	3.93 ± 0.11
Group-level fairness				
Fairlet	9587.47 ± 113.95	7812.30 ± 1102.79	0.60 ± 0.49	36.90 ± 0.84
Post-Group	8581.52 ± 1.30	5809.40 ± 41.81	0	3.16 ± 0.17
Both individual-level and group-level fairness				
Post-Combine	9175.91 ± 1.50	0	0	6.66 ± 1.23

4.5 Conclusion

Constrained clustering methods can integrate prior knowledge in term of constraints, but they are usually limited on the type of constraints. Moreover, they do not guarantee the satisfaction of the constraints. Declarative methods can handle several types of constraints and satisfy all of them, but they suffer from a lack of efficiency, which prevent them to handle large datasets. In our work, we propose the novel direction of post-processing the results of an unconstrained or constrained clustering algorithm to enforce the constraints a posteriori. Given a matrix that presents the cluster fractional allocation of instances to clusters, our method assigns instances to the most likely clusters while satisfying all the constraints. Our method can handle large datasets, it can integrate all types of popular constraints as well as a variety of new styles of constraints. It can be used with a wide variety of clustering algorithms, including deep learning and we demonstrated its use in the complex setting of ensuring group and individual level fairness using multiple constraints which to our knowledge has not been attempted.

Chapter 5

Deep Clustering with Logical Knowledge

Inspirés par plusieurs travaux sur les réseaux de neurones avec intégration des connaissances (Section 3.1), nous transformons les contraintes des experts en un ensemble de formules logiques pour résoudre le problème du clustering sous contraintes. Ensuite, nous définissons deux formulations pour la perte de connaissances, qui sont calculées sur la sortie d'un système de clustering profond. Chaque formulation a une traduction canonique de différents types de contraintes en pertes sémantiques tout en essayant de s'adapter à la tâche de clustering de différentes manières.

Dans les expériences, nous montrons que notre système peut atteindre des résultats comparables aux systèmes de pointe dédiés à des contraintes spécifiques tout en étant flexible pour intégrer et apprendre des contraintes de haut niveau.

5.1 Introduction

5.1.1 Reason for our proposal

Inspired by several works on neural networks with knowledge integration (Section 3.1), to address the constrained clustering problem, we transform the constraint into a set of logical formulae and we introduce a semantic loss. Then, we define two formulations for the knowledge loss, which are computed on the output of a deep clustering system. Each formulation has a canonical translation of different constraint types into semantic losses while trying to adapt for the clustering task in different ways.

Therefore, our method can integrate domain knowledge in the clustering problem while having the advantage of being independent of the neural architecture. Experiments are conducted to compare our unified framework to the state-of-the-art systems for together/apart and triplet constraints in terms of computational cost and clustering quality. We show that our system can achieve comparable results with the state-of-the-art systems dedicated to specific constraints while being flexible to integrate and learn from high-level domain constraints.

5.1.2 Chapter Organization

The remainder of the chapter is structured as follows. First, our general framework for learning expert constraints in a deep clustering model is introduced in Section 5.2. It is called DC-LK. In Section 5.3, we present implementation details

of IDEC-LK algorithm, which is the integration of our framework into IDEC - an autoencoder-based clustering. Next, we provide details of the experimental setup, empirical results, and our analysis in Section 5.4. Section 5.5 concludes with our contribution and further discussion.

5.2 Logical Knowledge Integration for Deep Clustering

5.2.1 Overview

Given $\mathbf{X}$ a set of n points, k a number of clusters, a deep clustering model $\mathcal{M}$ computes a cluster assignment $\mathbf{p} = \{p_1, p_2, ..., p_n\}$, $p_i \in [1, k]$ expressing that point i belongs to cluster p_i.

In this section, we propose a method to modify the model $\mathcal{M}$ based on $\mathbb{C}$, a set of constraints expressing expert knowledge. Together with the clustering algorithm, we have a framework called DC-LK (Deep Clustering with Logical Knowledge), whose general scheme is shown in Figure 5.1.

The process of DC-LK is as follows. First, we compute the allocation matrix S from the deep clustering model $\mathcal{M}$. The methods for defining S have been specified in Section 4.1.1. Second, we translate a set of expert constraints $\mathbb{C}$ into the set of logical formulae $P_{\mathbb{C}} = \{P_c : c \in \mathbb{C}\}$ where P_c is a constrained partition problem for the expert constraint c. The definition of P_c will be given in the next paragraph. Then, we compute a *Constrained Score* for each P_c using S. The formulation for constrained scores is based on Weighted Model Counting (WMC). Finally, the expert constrained loss is calculated with the scores and is backpropagated into the deep clustering $\mathcal{M}$ through S.

Constrained partition problem

We distinguish two kinds of constraints: partition constraints and expert constraints. Partition constraints enforce the result to express a partition while expert constraints represent expert knowledge, such as pairwise constraints. All these constraints are described by logical propositional formulae, and thus complex constraints that represent more complex knowledge can be expressed. Our partition constraint requires each point to belong to a single cluster. We do not put constraints on the number of clusters because this regulation can be handle by the deep clustering network. A constrained partition problem P_c is defined by the expert constraint c and the partition constraints for all points in c.

For complexity reasons, the problem of satisfying expert constraints is decomposed into a set $P_{\mathbb{C}}$ of sub-problems P_c, one for each expert constraint $c \in \mathbb{C}$.

Constrained scores

Let S be the soft assignment computed by a deep clustering system. For each sub-problem P_c, we define a constrained score $CS(P_c, S)$, which measures how likely the soft assignment S satisfies the constraints in P_c. The formulation P_c and the constrained score will be developed in Section 5.2.2.

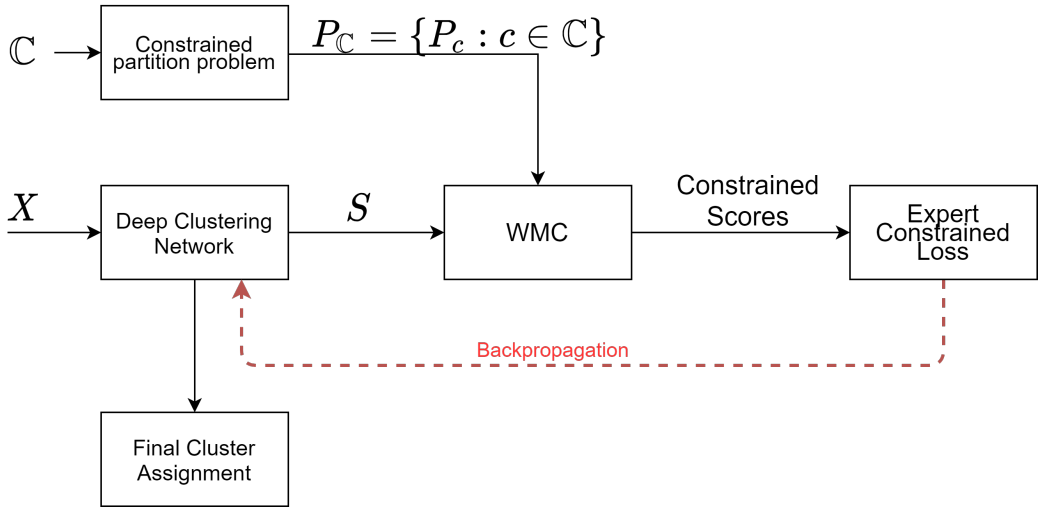

Figure 5.1: Overview of our framework function. A deep clustering framework is used to compute a soft assignment S of data points $\mathscr{X}$ to clusters. The set of expert constraints $\mathbb{C}$ are used to formulate a set of constrained partition problems $P_{\mathbb{C}} = \{P_c : c \in \mathbb{C}\}$. Constrained scores are computed based on S and $P_{\mathbb{C}}$ and are used to defined the constrained loss. This constrained loss is backpropagated to the deep clustering network.

Expert constraint loss We define the expert constraint loss for $\mathbb{C}$ based on the constrained scores:

$$L_{expert} = -\sum_{c \in \mathbb{C}} \log CS(P_c, S) \tag{5.1}$$

where: $CS(P_c, S) \in \mathbb{R}_{[0,1]}$ is the constrained score for constraint c in the constraint set $\mathbb{C}$.

5.2.2 Constrained Score

While a label gives a direct expectation for the outputs, expert constraints give a condition on the outputs which could be translated into a set of possible answers.

Example Given $\mathbf{p} = \{p_1, p_2, \ldots, p_n\} \in [1, k]^n$ be the outputs of a clustering/classification model with k clusters/classes for the data $\mathscr{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$.

Let y_i be the label for the point $\mathbf{x}_i$. With this knowledge, we only have one valid answer $p_i \in \{y_i\}$.

In contrast, a must-link constraint between $\mathbf{x}_i$ and $\mathbf{x}_j$ will reduce the outputs of $(\mathbf{x}_i, \mathbf{x}_j)$ to k possible answers $(p_i, p_j) \in \{(h, h) : h \in [1, k]\}$.

Definition Let c be an expert constraint. The constrained clustering problem P_c is defined by the expert constraint c and the partition constraints enforcing

the points involved in c to belong to a single cluster. It defines therefore a set of partitions $\mathbb{P}_c$ satisfying P_c.

Suppose we can define a partition score $Score(\mathbf{p}, S)$ which is a score of a partition $\mathbf{p}$ based on the allocation matrix S. Then, the constrained score $CS(P_c, S)$ is the sum of all partition scores $Score(\mathbf{p}, S)$ such that $\mathbf{p} \in \mathbb{P}_c$:

$$CS(P_c, S) = \sum_{p \in \mathbb{P}_c} Score(\mathbf{p}, S) \tag{5.2}$$

where $Score(\mathbf{p}, S)$ is a partition score of $\mathbf{p}$ and will be defined in Section 5.2.3.

Equation (5.2) shows a universal way to define a constrained score for any expert constraint.

Explanation for separating constraints We can extend Equation (5.2) to find a single *constrained score* for a constraint set $\mathbb{C}$.

$$CS(P_{\mathbb{C}}, S) = \sum_{p \in \mathbb{P}_{\mathbb{C}}} Score(\mathbf{p}, S) \tag{5.3}$$

where $\mathbb{P}_{\mathbb{C}}$ is the set of partitions that satisfy the constraint set $\mathbb{C}$.

However, in our framework, we opt for using multiple constrained scores $\{CS(P_c, S) : c \in \mathbb{C}\}$ because of complexity and for having a more effective learning.

Comparison with the post-processing framework In chapter 4, we define the objective function to find the best partition as follows:

$$\begin{aligned} &\max_{\mathbf{p} \text{ satisfies } \mathbb{C}} Score(\mathbf{p}, S) \\ \text{which is equivalent to: } &\max_{\mathbf{p} \in \mathbb{P}_{\mathbb{C}}} Score(\mathbf{p}, S) \end{aligned} \tag{5.4}$$

Both Equations (5.2) and (5.4) are defined by partition scores $Score(\mathbf{p}, S)$ on partitions that satisfy the constraint set $\mathbb{C}$. However, the aim of the post process is to find immediately the best answer so it needs to find a single best partition for all the constraints. Meanwhile, the deep constrained clustering tries to improve gradually through combined optimization between clustering structure and constraint satisfaction. Consequently, the expert loss wants to retain as much as possible the information of the constraints and reduce bias from the current clustering structure. Therefore, the deep learning framework can separately learn to satisfy each constraint and it needs the sum of all valid partitions for a better learning loss.

5.2.3 Partition score

a. Formulations

Recall from previous section 4.1.2 that most clustering algorithms could produce an assignment score S_{ij} which is the likelihood of point i to belong to cluster j. Then,

the likelihood of a partition $\mathbf{p}$ to be the output of the model $\mathcal{M}$ could be expressed through S.

We propose two definitions for the scores: $Score_A$ and $Score_B$. They are defined as follows:

$$Score_A(\mathbf{p}, S) = \prod_{i\in[1,n]} \left[S_{ip_i} \prod_{j\neq p_i} (1 - S_{ij}) \right] \tag{5.5}$$

$$Score_B(\mathbf{p}, S) = \prod_{i\in[1,n]} S_{ip_i} \tag{5.6}$$

where p_i is the cluster of point i in the partition $\mathbf{p}$.

Equation (5.5) takes into account not only the likelihood of the assignment of points to their clusters but also the likelihood of the exclusion of a point to other clusters. While in Equation (5.6), the likelihood of a partition is the product of each assignment of points.

Combined with the definition of constrained scores (5.2), we have two formulations (5.7) and (5.8) to calculate constrained scores.

$$CS_A(P_c, S) = \sum_{p\in\mathbb{P}_c} \prod_{i\in[1,n]} \left[S_{ip_i} \prod_{j\neq p_i} (1 - S_{ij}) \right] \tag{5.7}$$

$$CS_B(P_c, S) = \sum_{p\in\mathbb{P}_c} \prod_{i\in[1,n]} S_{ip_i} \tag{5.8}$$

b. Interpretation of the two formulations

With $Score_A$, we assume that any pair of S_{ij} and $S_{ij'}$ (with $i \in [1, n]$ and $j \neq j'$) is independent, so the likelihood of a point i to belong *only* to cluster j is

$$\mathcal{L}\left(i \in \mathscr{C}_j \bigwedge_{j'\in[1,k]:j'\neq j} i \notin \mathscr{C}_{j'} \right) = S_{ij} \prod_{j'\in[1,k]:j'\neq j} (1 - S_{ij'}) \tag{5.9}$$

where $\mathscr{C}_j$ denote the set of points belonging to cluster j.

In other words, the output of learning model $\mathcal{M}$ does not limit the number of clusters a point can be assigned to. Therefore, we can even calculate the likelihood of a point to belong to a set of clusters $\mathcal{C} = \{\mathscr{C}_{u_1}, ..., \mathscr{C}_{u_m}\}$ as follows:

$$\mathcal{L}\left(\bigwedge_{\mathscr{C}_j\in\mathcal{C}} i \in \mathscr{C}_j \bigwedge_{j'\notin\mathcal{C}} i \notin \mathscr{C}_{j'} \right) = \prod_{j:\mathscr{C}_j\in\mathcal{C}} S_{ij} \prod_{j':\mathscr{C}_{j'}\notin\mathcal{C}} (1 - S_{ij'}) \tag{5.10}$$

As a consequence, the likelihood that the output of model $\mathcal{M}$ to be a partition with at most k clusters is not certain.

$$\forall S : \sum_{\mathbf{p}\in[1,k]^n} Score_A(\mathbf{p}, S) \leq 1 \tag{5.11}$$

In contrast, the $Score_B$ exploits the fact that each point must belong to exactly one cluster. So, adding the terms $1 - S_{ij'}$ into the formulation is redundant.

The effect of two score formulations on *constrained score* The constrained score with Equation (5.5) is the sum of the partition scores that satisfy P_c over all possible assignments (allowing arbitrary assignments); while, for Equation (5.6) it is the sum score of satisfied partitions over all possible partitions. It means that $\forall \mathbf{p} : Score_A(\mathbf{p}, S) \leq Score_B(\mathbf{p}, S)$ and $\forall P_c : Score_A(P_c, S) \leq Score_B(P_c, S)$.

5.2.4 Weighted Model Counting

a. Introduction

The complexity of calculating a single constrained score using (5.2) is exponential ($\mathcal{O}(k^n)$). Given that the form is similar to Weighted Model Counting (WMC) formulation, which has an efficient way to compute through SDD representation, we will translate this problem into a WMC problem. In this section, we present the WMC problem.

Notations We denote uppercase letters Y, A, B for Boolean variables and lowercase letters y, a, b for their instantiation to $\top$ $(true)$ or $\bot$ $(false)$. The bold uppercase letters $\mathbf{Y}, \mathbf{A}, \mathbf{B}$ represent sets of variables while the bold lowercase letters $\mathbf{y}, \mathbf{a}, \mathbf{b}$ sets of their instantiations. Given a formula α which is defined on a set $\mathbf{Y}$, we call $\mathbf{y}$ satisfies α, denoted as $\mathbf{y} \models \alpha$ if α is interpreted to $\top$ by $\mathbf{y}$. A sentence α is logically equivalent to sentence β, denoted $\alpha \equiv \beta$, if $\{\mathbf{y} : \mathbf{y} \models \alpha\} = \{\mathbf{y} : \mathbf{y} \models \beta\}$

Let α be a logical formula defined on a set of variables $\mathbf{Y}$, where each variable Y_i is associated with a weight w_i. The weighted model counting (WMC) of α is defined by [82]:

$$WMC(\alpha) = \sum_{\mathbf{y} \models \alpha} \prod_{i:\mathbf{y} \models Y_i} w_i \prod_{i:\mathbf{y} \models \neg Y_i} (1 - w_i) \tag{5.12}$$

Decomposition and SDD structure In Equation (5.12), the atomic terms w_i and $(1 - w_i)$ could occur in multiple instantiations of $\mathbf{y}$. Moreover, if $\alpha \equiv \alpha_1 \vee \alpha_2$ and $\alpha_1 \wedge \alpha_2 \equiv \bot$ then the WMC could be decomposed by:

$$WMC(\alpha) = WMC(\alpha_1) + WMC(\alpha_2) \tag{5.13}$$

Therefore, we can use any logical decomposition method to break down the calculation of calculate $WMC(\alpha)$. In this work, we rely on SDD which is currently the best decomposition for logical interference tasks[23]. The SDD decomposition method is described more in depth in Appendix A.1.

b. Formulations

In order to calculate a constraint score CS_A or CS_B of a constrained partition problem using WMC, we need to declare a set of logical variables and their weights. For the calculation of CS_A, we introduce a set of logical variables $\mathbf{A}$ with their weights. The details will be given in Section 5.2.5. Similarly for CS_B, we introduce the logical variable set $\mathbf{B}$ in Section 5.2.6. Then, the score will be computed using decomposition and SDD construction. Examples of SDD construction for some constraints will be presented in in Appendix A.1.4.

5.2.5 Formulation A

a. Variables

Let us define $\mathbf{A} = \{A_{ij} | i \in [1, n], j \in [1, k]\}$ a set of propositional variables where $A_{ij} = \top$ represents the assignment of instance i to cluster j and $A_{ij} = \bot$ means that i does not belong to cluster j. Let us recall that for each instance i, S_{ij} $(0 \leq S_{ij} \leq 1)$ represents the probability of assigning instance i to cluster j. Therefore we define the weights w_A for each variable by $w_A(A_{ij}) = S_{ij}$ and therefore $w_A(\neg A_{ij}) = 1 - S_{ij}$. With this definition of weights, formulas (5.2) and (5.5) can be computed by a Weighted Model Counting.

b. Constrained Partition Formulations

Partition Constraints Using the variables $\mathbf{A}$, the partition constraint in P_c can be expressed as follows. Each point is assigned to at least one cluster is expressed as:

$$\wedge_{i \in [1,n]} \left(\vee_{j \in [1,k]} A_{ij} \right) \tag{5.14}$$

and each point is assigned to at most one cluster:

$$\wedge_{i \in [1,n]} \left(\wedge_{j,t \in [1,k]; j \neq t} (\neg A_{ij} \vee \neg A_{it}) \right) \tag{5.15}$$

Expert Constraints The expert constraint c can be expressed by a logical formula. If c is a must-link (Together) constraint on two instances u and v then it is expressed by:

$$\wedge_{i \in [1,k]} (A_{ui} \vee \neg A_{vi}) \wedge (\neg A_{ui} \vee A_{vi}) \tag{5.16}$$

and if c is a cannot-link (Apart) constraint between u and v:

$$\wedge_{i \in [1,k]} (\neg A_{ui} \vee \neg A_{vi}) \tag{5.17}$$

Using together and apart constraints as elementary constraints and logical operators such as implication, we can express more complex constraints. For instance a triplet constraint between three points (a, p, n) that states that the anchor a is more similar to the positive point p than to the negative point n can be expressed by:

$$Together(a, n) \implies Together(a, p) \tag{5.18}$$

More complex implication constraints can also be expressed, as for instance:

$$Together(a,b) \wedge Together(c,d) \wedge Apart(a,e) \implies Together(e,f) \wedge Apart(c,e)$$

New kinds of constraints can also be introduced. We define a new constraint, called *m-cluster group constraint* that states that instances in a subset I can only belong to some defined clusters $j \in J$, and it can be expressed by:

$$\wedge_{i \in I} \vee_{j \in J} A_{ij} \tag{5.19}$$

Conclusion Let us denote α be a joined formulation of the partition constraint and the expect constraint c. Then, the constrained score A of the constrained partition problem P_c is equal to the Weighted Model Counting of α.

$$Score_A(S, P_c) = WMC(\alpha)$$

5.2.6 Formulation B

The score $Score_A(\mathbf{p}, S)$ given by (5.5) allows to measure the likelihood between S and a partition p. However it does not exploit the fact that for any point i, if i is likely to belong to a cluster j (S_{ij} is close to 1), then i is less likely to belong to other clusters (S_{il} close to 0, for $l \neq j$). We propose another formulation of the constrained clustering problem P_c to enforce this dependency.

a. Variables

For each point i, we will define the formulas $\beta_{i1}, ..., \beta_{ik}$, such that β_{ij} interpreted to $\top$ means that the point i is assigned to cluster j and is not assigned to cluster $j' \neq j$. Let $\mathbf{B}$ be a set of logical variables $\{B_{ij} : i \in [1,n], j \in [1,k]\}$. The formula β_{ij} is defined as follows:

$$\begin{array}{l} \beta_{ij} \stackrel{\text{def}}{=} B_{ij} \wedge \bigwedge_{t \in [1,j-1]} \neg B_{it} \quad \text{for all } j \in [1, k-1], \\ \beta_{ik} \stackrel{\text{def}}{=} \bigwedge_{t \in [1,k-1]} \neg B_{it} \end{array} \tag{5.20}$$

We define the weight w_B for the variables as follows:

$$w_B(B_{ij}) = \begin{cases} \frac{S_{ij}}{1 - \sum_{t \in [1,j-1]} S_{it}} & \text{if} \quad \sum_{t \in [1,j-1]} S_{it} < 1 \\ 1 & \text{otherwise} \end{cases} \tag{5.21}$$

b. Constrained Partition Formulations

Partition Constraints The partition constraint stating that each point belongs to one cluster is expressed by:

$$\bigwedge_i \left[(\beta_{i1} \vee \ldots \vee \beta_{ik}) \bigwedge_{j,l \in [1,k]: j \neq l} (\neg \beta_{ij} \vee \neg \beta_{il}) \right] \tag{5.22}$$

We prove that with definition (5.20) the clustering condition is a tautology, that is (5.22) $\equiv \top$. It means that any instantiation of B represents an assignment where each point belongs to a single cluster. We also prove that the weighted model counting of any constrained problem P_c defined on $\mathbf{B}$ will give the $Score_B(\mathbf{p}, S)$ given in (5.6) (all the proofs are given in Appendix A.2).

Expert Constraints For the expert constraints, the forms are similar to the formulation A, where A_{ij} is replaced by β_{ij}. The following examples are the substituted (and shorted if possible) formulations.

- Must-link constraint between u and v:

$$\begin{aligned}
& \wedge_{i\in[1,k]}\ (\beta_{ui} \vee \neg\beta_{vi}) \wedge (\neg\beta_{ui} \vee \beta_{vi}) \\
\iff & \wedge_{i\in[1,k]}\ ((\wedge_{t\in[1,i-1]}\neg B_{ut} \wedge B_{ui}) \vee \neg(\wedge_{t\in[1,i-1]}\neg B_{vt} \wedge B_{vi})) \\
& \wedge_{i\in[1,k]}\ (\neg(\wedge_{t\in[1,i-1]}\neg B_{ut} \wedge B_{ui}) \vee (\wedge_{t\in[1,i-1]}\neg B_{vt} \wedge B_{vi})) \\
\iff & \wedge_{i\in[1,k]}\ ((\wedge_{t\in[1,i-1]}\neg B_{ut} \wedge B_{ui}) \vee_{t\in[1,i-1]} B_{vt} \vee \neg B_{vi}) \\
& \wedge_{i\in[1,k]}\ (\vee_{t\in[1,i-1]} B_{ut} \vee \neg B_{ui} \vee (\wedge_{t\in[1,i-1]}\neg B_{vt} \wedge B_{vi}))
\end{aligned} \tag{5.23}$$

  Presenting in another form:

$$\begin{aligned}
& \vee_{i\in[1,k]}\ (\beta_{ui} \wedge \beta_{vi}) \\
\iff & \vee_{i\in[1,k]}\ (\wedge_{t\in[1,i-1]}\neg B_{ut} \wedge B_{ui} \wedge_{t\in[1,i-1]} \neg B_{vt} \wedge B_{vi}) \\
\iff & \vee_{i\in[1,k]}\ (B_{ui} \wedge B_{vi} \wedge_{t\in[1,i-1]}\ (\neg B_{ut} \wedge \neg B_{vt})) \\
\iff & (B_{u1} \wedge B_{v1}) \vee (\neg B_{u1} \wedge \neg B_{v1} \wedge (...))
\end{aligned} \tag{5.24}$$

- Cannot-link constraint between u and v:

$$\begin{aligned}
& \wedge_{i\in[1,k]}\ (\neg\beta_{ui} \vee \neg\beta_{vi}) \\
\iff & \wedge_{i\in[1,k]}\ \big(\neg(\wedge_{t\in[1,i-1]}\neg B_{ut} \wedge B_{ui}) \vee \neg(\wedge_{t\in[1,i-1]}\neg B_{vt} \wedge B_{vi})\big) \\
\iff & \wedge_{i\in[1,k]}\ \big((\vee_{t\in[1,i-1]} B_{ut} \vee \neg B_{ui}) \vee (\vee_{t\in[1,i-1]} B_{vt} \vee \neg B_{vi})\big) \\
\iff & \wedge_{i\in[1,k]}\ \big[\neg B_{ui} \vee \neg B_{vi} \vee_{t\in[1,i-1]}\ (B_{ut} \vee B_{vi})\big]
\end{aligned} \tag{5.25}$$

5.2.7 Notes for non-empty cluster condition

The clustering requirement of non-empty cluster has not been integrated into the formulations. The reason is that this constraint substantially increases the computation cost while its impact on the total model weight is insignificant. The number of interpretations with an empty cluster is often reduced by the expert constraints. Besides, the model learned by the deep clustering network is a clustering without empty cluster, as the number of clusters is given before the training.

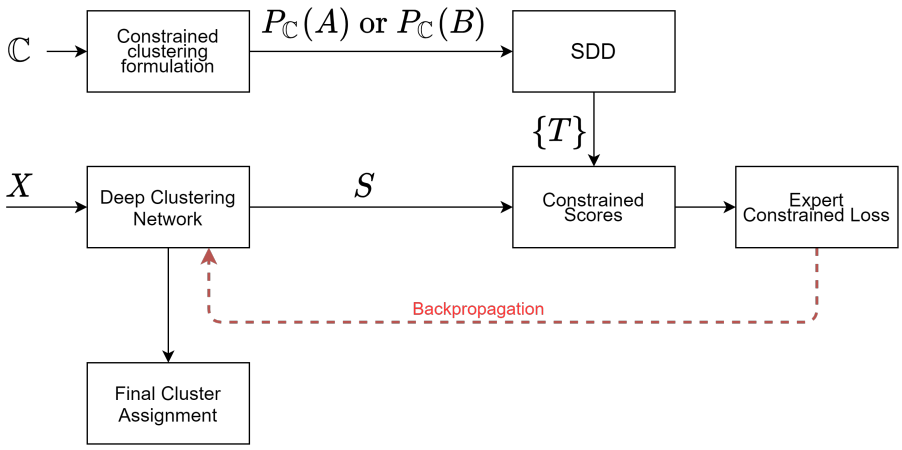

Figure 5.2: Deep Clustering with Logical Knowledge Framework

5.2.8 General training scheme

We propose a general training scheme that integrates our loss for expert constraints (see Figure 5.2). It can be used with any deep clustering method and we give in Section 5.3 how it is instantiated with IDEC. In the general scheme, the inputs are the set $\mathbf{X}$ of n points and the set $\mathbb{C}$ of constraints. The constrained loss is the sum of the loss of each constrained score (presented as $P_c(A)$ or $P_c(B)$) in the constraint set $\mathbb{C}$:

$$L_{expert} = -\sum_{c \in \mathbb{C}} \log Score(P_c, S) \tag{5.26}$$

The output is the cluster assignment $\mathbf{p}$. There are 7 steps to produce the final output:

Step 1: Express the constraint set in form of logical formulas (either $P_{\mathbb{C}}(A)$ or $P_{\mathbb{C}}(B)$).

Step 2: Convert the formulas in arithmetic circuits $\mathbb{T}$ using SDD.

Step 3: Train a deep clustering network $\mathcal{M}$ on $\mathbf{X}$.

Step 4: Run $\mathcal{M}$ to produce the soft cluster assignment S.

Step 5: Calculate the constrained scores $Score(P_c, S)$ for all $c \in \mathbb{C}$ using arithmetic circuits $\mathbb{T}$ and S.

Step 6: Update $\mathcal{M}$ using the loss defined by Eq. (5.26) (and existing loss $\mathcal{M}$, optional).

Step 7: Check stopping condition such as the learning convergence, computation limit. If not go back to Step 4.

Because of this general strategy, the proposed method can be applied to a variety of deep clustering methods. Besides, Step 6 is quite general and could cover different approaches depending on the objective function: the model $\mathcal{M}$ is learned using solely the constrained score, or with a combination of it with existing model losses.

5.3 IDEC-LK

The previous section shows the process of learning expert constraints with logical knowledge. It can be applied to any deep clustering model. In this section, we present the IDEC-LK framework, which is based on IDEC - a clustering algorithm using an autoencoder. We carefully describe the neural architecture, the training scheme, and hyper-parameter settings. We also give our technical improvement and analysis for the complexity of the framework.

5.3.1 IDEC-LK training scheme

IDEC-LK extends IDEC [40] to integrate the expert loss. The training scheme is given in Figure 5.3. The details of our training process are presented in Algorithm 5.1.

Based on the outlined algorithm from Section 5.2.8, in order to integrate IDEC, the steps 3, 4 and 6 are defined, as follows:

- Step 3: Train a deep clustering network $\mathcal{M}$
  - Use an autoencoder on the original data $\mathbf{X}$ to get embedded points $\mathbf{Z}$ and reconstruction points $\tilde{\mathbf{X}}$
  - Use k-means clustering for embedded points $\mathbf{Z}$ to have K centers $(\mu_1, \ldots, \mu_k)$
- Step 4: Compute the soft cluster assignments of all points (to all clusters) based on Student's t-distribution:

$$S_{ij} = \frac{(1+ \parallel z_i - \mu_j \parallel^2)^{-1}}{\sum_{j'}(1+ \parallel z_i - \mu_{j'} \parallel^2)^{-1}} \tag{5.27}$$

- Step 6: Update $\mathcal{M}$ using the loss defined by Eq. (5.26) and IDEC loss, computed as follows:
  - Compute the target assignment as:

$$p_{ij} = \frac{S_{ij}^2/f_j}{\sum_{j'} S_{ij'}^2/f_{j'}} \tag{5.28}$$

    where $f_j = \sum_{i=1}^{n} S_{ij}, \forall j = 1, \ldots, k$ are the soft cluster frequencies.
  - Compute the clustering loss as the difference between the two probability distributions using Kullback–Leibler formulation.

$$L_{clustering} = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{S_{ij}} \tag{5.29}$$

– Compute the reconstruction loss as the mean square distance between $\mathbf{X}$ and $\tilde{\mathbf{X}}$

$$L_{recon} = \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \tag{5.30}$$

– Compute the loss $L_1 = \lambda_r \times L_{recon} + \lambda_c \times L_{clustering}$ where λ_r, λ_c are coefficients.

– Compute the expert loss using Eq. (5.26) and let $L_2 = \lambda_e \times L_{expert}$.

– Backpropagate L_1 and L_2 and update the parameters θ of the autoencoder and the cluster centers μ

Let us notice that during training, we separate the expert loss and the other losses for backpropagation at each epoch. This helps to reduce the size of the formula to backpropagate. The two losses of IDEC-LK model are defined as:

$$\begin{aligned} L_1 &= \lambda_r \times L_{recon} + \lambda_c \times L_{clustering} \\ L_2 &= \lambda_e \times L_{expert} \end{aligned} \tag{5.31}$$

where λ_r, λ_c and λ_e are coefficients controlling each loss.

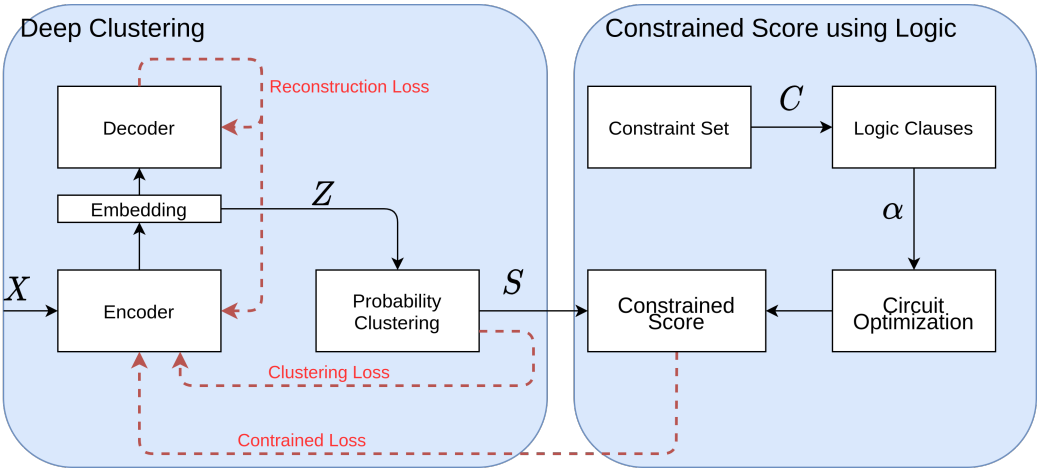

Figure 5.3: The general process of IDEC-LK consists of IDEC - a deep clustering on the left and an expert constraint learner on the right.

a. Network architecture

We denote by dim_X the dimension of input data. The structure of the autoencoder is as follows: $[dim_X, 500, 500, 2000, 10, 2000, 500, 500, dim_X]$ (the same architecture as IDEC [40], DCC [109], SDCN [14]). We train the pretrained model using Stacked Denoising Autoencoder [95].

Algorithm 5.1 Training process of IDEC-LK

Input: Input data: $\mathbf{X}$, Number of clusters: k, Constraint set: $\mathbb{C}$, Maximum iterations: $MaxIter$; Coefficients: λ_r, λ_e, λ_c
Output: Cluster assignment $\mathbf{p}$

Initialize parameters with pre-trained autoencoder
Initialize μ with K-means on the representations learned by pre-trained autoencoder
Generate $P_{\mathbb{C}} = \{P_c : c \in \mathbb{C}\}$ from the constraint set $\mathbb{C}$ ▷ A or B formulation
Generate $\mathbb{T}$ - a set of SDD structures from all P_c
for $iter := 1$ to $MaxIter$ **do**
    Input $\mathbf{X}$ to the encoder: $\mathbf{Z}$
    for $batch := 1$ to $NumConstrainedBatches$ **do**
        $X_{batch} = \{x : x \in C_{batch}\}$
        Forward embedded point Z_{batch} from the set of points $x \in X_{batch}$
        Forward distribution S via t-distribution with Z, μ; (Eq. (5.27)) from the set of points $x \in C_{batch}$
        Feed S to SDD structures $\mathbb{T}$ to calculate L^{batch}_{expert}
        Backpropagate L_2 and update parameters
    end for
    for $batch := 1$ to $BatchAllInputs$ **do**
        Forward embedded point Z_{batch} from the set of points $x \in X_{batch}$
        Forward distribution S_{batch} via t-distribution with Z_{batch}, μ (Eq. (5.27))
        Calculate target distribution P_{batch} (Eq. (5.28))
        Feed Z_{batch} to the decoder to obtain the reconstruction $\tilde{X}_{batch}$
        Calculate L^{batch}_{recon}, $L^{batch}_{clustering}$, (Eq. (5.30), (5.29), respectively)
        Backpropagate L_1 and update parameters
    end for
end for
$\mathbf{p} = \{\arg\max_{j \in [1,k]} Q_{ij} : i \in [1, n]\}$
Return $\mathbf{p}$

Optimizer and hyperparameter settings We use Relu for the activation function. The number of epochs for training each layer is 300, follows by 500 epochs training the whole network. The optimizer is stochastic gradient descent (SGD) optimizer with a momentum equal to 0.9. Its learning rate is set to 0.1 and decreased by one-tenth after every 100 epochs.

For efficiency reasons, we use mini-batch learning with a batch size of 256. Because the clustering and reconstruction losses are on all the points, so the number of batches is $\lceil \frac{n}{256} \rceil$ where n is the total number of training samples.

For each type of expert constraints, the number of batches is $\lceil \frac{\|\mathbb{C}_{type}\| * size(type)}{256} \rceil$ where $\|\mathbb{C}_{type}\|$ is the number of constraints, $size(type)$ is the number of points involved in each constraint.

With this implementation, we omit any relation between constraints (neither same type nor not). The disadvantage is that it removes any possible joint interpretation between constraints. But there are two major advantages. First, it simplifies the construction of SDDs. In details, we only need to construct one SDD for each constraint type; otherwise we need to decompose the constraint sets into independent subset of variables and construct SDDs for each subset (which could be impossible because of time to compile SDDs). Second, the implementation can be vectorized. Constraints in the same mini-batch share the same arithmetic circuit so we can stack values of these constraints into vectors and run arithmetic operator between vectors.

5.3.2 Handling contradictory constraints and noisy constraints

Contradictory As we separate the constraint set, contradictory constraints can be optimized by the model. In the simplest case, when two constraints C1 and C2 are contradictory, the loss function is $-\log Score(P_{C1}, S) - \log Score(P_{C1}, S)$. The convergence of the loss will be achieved when $Score(P_{C1}, S) = Score(P_{C2}, S) = 0.5$. Without any additional information, it seems to be a logical objective.

Noisy constraints In practical applications, the constraints are generated by humans, so it could be susceptible to human-error, especially for through crowdsourcing. If the constraint set contains incorrect constraints (which completely contradict the expert knowledge), the losses for these constraints could extremely impact the model and it could make the learning unpredictable and unable to converge. Our solution for this problem is to remove the constraints with the lowest constrained score. As based on the learning model $\mathcal{M}$, the constraints with lowest scores are the most likely to be the noisy one. Because of its practical application, this technique has been implemented in IDEC-LK framework.

5.3.3 Restrict logic formulation from learning model

Up until now, the learning model $\mathcal{M}$ does not have any impact on the SDD structure T. However, the score S given from $\mathcal{M}$ could help to limit the possible models in T. In fact, since $\mathcal{M}$ has already been learned with the clustering process then adding a

small set of constraints would not make $\mathcal{M}$ change significantly. The benefit of this restriction is to reduce the number of solutions from the logic tree T and potentially reduce the tree size.

We introduce a hyper-parameter ϵ for limiting the number of logic models.

For formulation A, we add the following sentence to α:

$$\forall i \in [1, n], j \in [1, k] : S_{ij} < \epsilon \rightarrow \neg A_{ij} \tag{5.32}$$

For formulation B, we have

$$\forall i \in [1, n], j \in [1, k] : S_{ij} < \epsilon \rightarrow \neg \beta_{ij} \tag{5.33}$$

5.3.4 Complexity Analysis

Compared to unconstrained approaches that do not handle constraints, we have two additional steps: SDD construction and computation the expert loss.

a. Complexity in SDD construction

As mentioned in [23], the cost of the SDD construction is exponential in the SDD treewidth w, which is linear to the number of variables, in our case $n \times k$ ($w \in O(n \times k)$). In practice, to reduce w, we consider each expert constraint independently. Moreover, each SDD is precomputed only once before the training, so the computation cost is much less than the training complexity, which requires to use the SDD structure to compute the loss on every batch and epoch.

b. Complexity of the expert loss

After the SDD is constructed, we can convert a single constraint loss into an arithmetic circuit by changing AND gates into multiplication ($\times$) and OR gates into addition ($+$). The translation shows that the computational cost for computing the value and the gradient of the loss is linear with the size of the circuit. Therefore, in each epoch, the cost of computing the expert constraint loss is equal to the size of the SDD structure times the number of constraints.

In addition, the size of a SDD depends on the form and number of variables of α. Given a Conjunctive Normal Form (CNF) with n variables and treewidth ω, then the size of SDD is $O(n2^{\omega})$.

For pairwise/triplet constraints, the SDD is fixed for each constraint type and their sizes are small constants (< 200 nodes for triplet, and < 100 nodes for pairwise). So, the expert loss is linear to the number of constraints.

For implication constraints, an experiment has been performed to measure the mean size of SDD. It is given in Appendix A.6.

Needless to say, the addition of the expert loss usually makes the convergence of the model slower. So, it takes more epochs to finish its training. The runtime compared to unconstrained clustering and other constrained clustering method has been shown in Appendix A.6.

5.4 Experimental Setup and Objectives

5.4.1 Objectives

The objectives of the experiments:

- To evaluate and compare the performances of clustering quality our solutions and other works with various datasets, constraint types (See Section 5.4.3).
- To evaluate and compare the performances of constraint satisfaction our solutions and other works with various datasets, constraint types (See Section 5.4.4).
- To evaluate complexity each step in our systems (See Section 5.4.5).

5.4.2 Experimental Setups

a. Datasets

We use four datasets, which are challenging and also used in a recent deep constrained clustering method [46, 109].

MNIST: The dataset is composed of 60,000 handwritten single-digits, with a size of 28-by-28 pixels.

Fashion: The dataset contains 60,000 images associated to a label from 10 classes.

CIFAR10: consists of 50,000 color images in 10 classes.

Reuters: Reuters contains around 810,000 English news stories labeled with a category tree [60]. Following DEC [101], we consider only the single label documents belonging to the `corporate/industrial`, `government/social`, `markets` and `economics` categories. A subset of 10,000 examples is randomly sampled and the TF-IDF measure is computed on the 2000 most frequent words.

b. Baselines

Deep clustering models

SDAE+Kmeans: Stacked Denoising Autoencoder (SDAE) learns the representation of data. Then, K-means is applied to find the clustering in the embedded space of SDAE.

IDEC: Deep clustering method that learn both the representation and the cluster structure of data[40].

Constrained clustering models

PCK-means, MPCK-means: The classic constrained clustering algorithms for pairwise constraints[8, 12].

DCC: Deep Constrained Clustering[109] which can handles pairwise, triplet constraints.

IDEC-LK: Our framework in Section 5.3.

c. Experiment setting

The hyper-parameters are selected through a grid search to make the model converge and achieve the highest constrained score. To prove our model robustness, we use a single set of hyperparameter values for multiple datasets. For a fair comparison, this principle also applies to DCC and IDEC. For DCC, we use their default settings of MNIST.

The optimizer is Adam, with a 0.001 learning rate. The stopping condition is either it reaches the maximum number of epochs (50) or getting a percentage of assignment different from the previous epoch less than 0.01%.

Pretraining clustering model In Table 5.1, we report the performance of SDAE+Kmeans and IDEC on the three datasets. The stacked training and denoising technique in SDAE helps to create strong pretrained autoencoders for IDEC, Cop-Kmeans, DCC, and IDEC-LK. On the other hand, IDEC shows the effect of reinforcing the clustering structure to the autoencoder. It helps to improve significantly on MNIST while seeing slight improvement or lightly worse in Fashion and Reuters.

Table 5.1: SDAE+Kmeans and IDEC performance on MNIST, Fashion and Reuters

Data	Model	NMI	ACC
MNIST	SDAE+Kmeans	0.7607 ± 0.0001	0.8202 ± 0.0001
MNIST	IDEC	**0.8667 ± 0.0013**	**0.8827 ± 0.0009**
Fashion	SDAE+Kmeans	0.5842 ± 0.0000	**0.5168 ± 0.0003**
Fashion	IDEC	**0.5945 ± 0.0008**	0.5163 ± 0.0031
Reuters	SDAE+Kmeans	**0.5474 ± 0.0008**	**0.7364 ± 0.0005**
Reuters	IDEC	0.5310 ± 0.0015	0.7124 ± 0.0009

d. Constraint Generation

For the experiments, we need to generate constraints. We explain in this section how they are generated depending on the type of constraint.

Pairwise constraint To generate N pairwise constraints, we randomly select N pairs of points. Each pair presents a must-link or cannot-link constraint according to the ground-truth label.

Triplet constraint To generate a triplet constraint, we randomly select 3 points and the triplet constraint is satisfied when

- Case 1: $Together(a, p) \wedge Apart(a, n)$ (it means $y_a = y_p \wedge y_a \neq y_n$)
- Case 2: $Apart(a, p) \wedge Apart(a, n) \wedge Apart(p, n)$ (it means $y_a \neq y_p \wedge y_a \neq y_n \wedge y_p \neq y_n$)

m-Clusters Group Constraint The details of this constraint have been stated in Subsection 4.3.2. This constraint ensures a group of instances, which share the same characteristics, spreads to exactly m clusters. m-clusters group constraint is strongly linked with hierarchical clustering. In some applications, we could know a group of points not to belong to a single cluster but to a group of clusters corresponding to a upper node in hierarchical clustering. In comparison with classification, m-clusters group constraint appears when a class contains multiple "sub-classes" but the subclass labels are unavailable[74, 89].

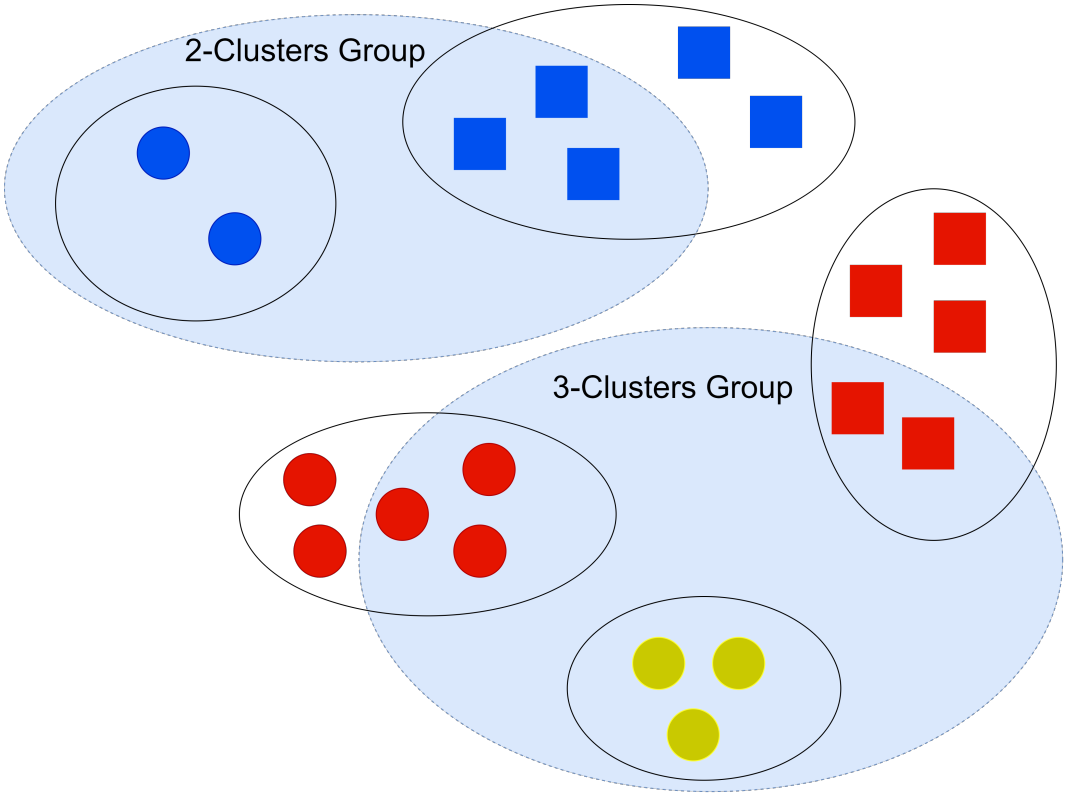

Figure 5.4: Examples of m-clusters group constraint. The upper blue area is a group of instances that belong to two clusters, while the lower blue area is a 3-clusters group.

In this experiment, we test on MNIST dataset. We selected 4 groups: G(3, 9), G(6, 8), G(0, 9), G(2, 5). They are pairs of digits that share some similar shapes. For a group (u, v), we select randomly $m = 100$ images of either u digit or v digit.

Implication Constraint Our constraint has the following form:

$$\wedge\, Together/Apart \implies \wedge Together/Apart \tag{5.34}$$

The first part (if-clause) is denoted as P. The second part (then-clause) is designated as Q. The maximum number of Together/Apart of if-clause in each constraint

is 5. The maximum number of Together/Apart of then-clause in each constraint is 5. Because of the random selection, the logical forms of implication constraint are highly diverse. Figure 5.5 shows the various examples for the implication constraint.

P-Q distribution: We generate 100 constraints $P \implies Q$ randomly based on the ground truth for each test. Around 50% of the constraints do not satisfy P, therefore $P \implies Q$ is valid, the remaining 50% satisfy both P and Q ($P = \top$, $Q = \top$).

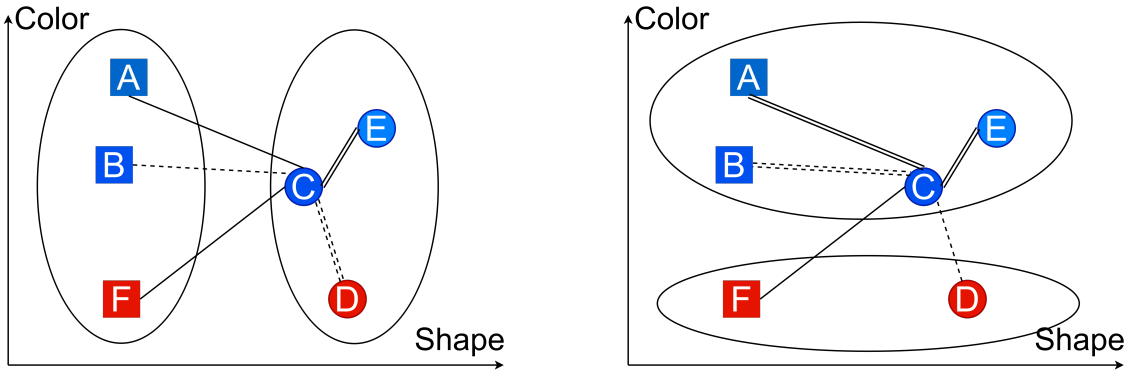

Figure 5.5: Examples of implication constraints. A single line shows Apart relation, and a double line is for Together. Solid lines belong to the hypothesis P, while dash lines refer to the conclusion Q. In this case, the constraints enforce the consistency of the clustering process.
Left (clustering by shape): $Apart(A, C) \wedge Apart(F, C) \wedge Together(C, E) \implies Apart(C, B) \wedge Together(C, D)$.
Right (clustering by color): $Together(A, C) \wedge Together(E, C) \implies Apart(F, C) \wedge Together(C, B) \wedge Apart(C, D)$.

5.4.3 Experiments and Analysis for Clustering Quality

In this section, we study the impact of pairwise and triplet constraints on the clustering quality.

a. Evaluation criteria

In all the datasets, the true class of the objects is available and we use it as the ground truth to evaluate the accuracy of the clustering. We consider two measures: Normalized Mutual Information (NMI) and clustering accuracy (ACC), with a one-to-one mapping between clusters and labels, computed by The Hungarian algorithm [56]. In both cases the higher the better.

b. Results

For testing the influence of pairwise constraints in MNIST, Fashion and Reuters dataset, we consider 4 numbers of pairwise constraints (10, 100, 500, 1000). For each test case, we randomly generate five sets of constraints, we run the system only once for each set of constraints and we report the mean and the standard deviation in Figure 5.6. We do the same for triplet constraints (see Figure 5.7, 5.8). In the figures, we do not report the performance of COP-Kmeans, which gives the worst result. All results are given in Appendix A.5.

c. Analysis

Pairwise In the MNIST and Fashion datasets, DCC, IDEC-LK-A and IDEC-LK-B have a similar performance, while IDEC-LK-A has the best performance with Reuters. The results of IDEC-LK-A and IDEC-LK-B are better than the deep clustering baseline - IDEC when the number of constraints is greater than 500.

The loss for pairwise constraints in IDEC-LK-B has a similar formulation to the one of DCC. However, in DCC, the losses have different weights for must-link and cannot-link constraints. These weights must be tuned with a validation set, which is usually not available in clustering problems. In our method, the single parameter for the constrained loss, λ_c, can be tuned through its ability to satisfy the given constraints. We made some experiments on the effects of λ_e on the number of satisfied constraints and on the value of the Weighted Model counting, which have both to be maximized. See Appendix A.8. From these experiments, the default value of λ_e has been set to 0.1.

Triplet The triplet constraints have less impact on the clustering quality than the pairwise because it conveys conditional information on the points. Relying on a Kolmogorov-Smirnov test with $p = 0.05$ [41], IDEC-LK-A and IDEC-LK-B have better clustering quality in Reuters while they have similar performance with DCC in the others datasets.

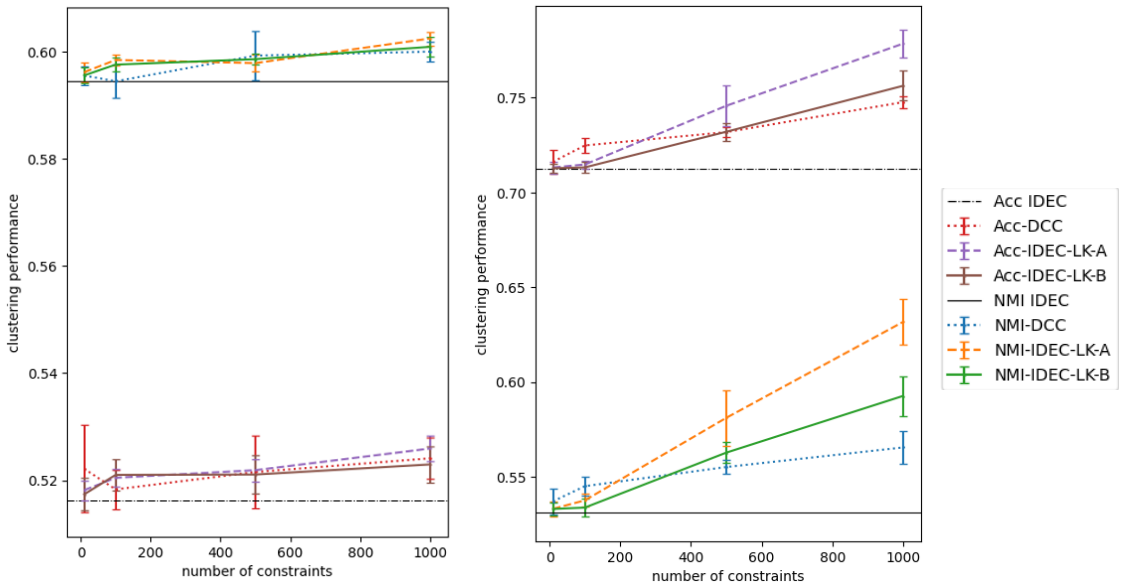

Figure 5.6: Clustering performance of pairwise on Fashion (left) and Reuters (right) dataset

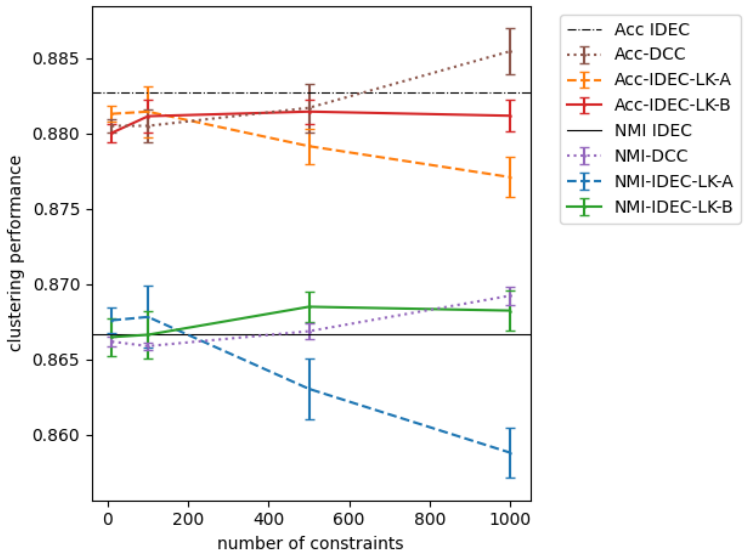

Figure 5.7: Clustering performance of triplet on MNIST dataset

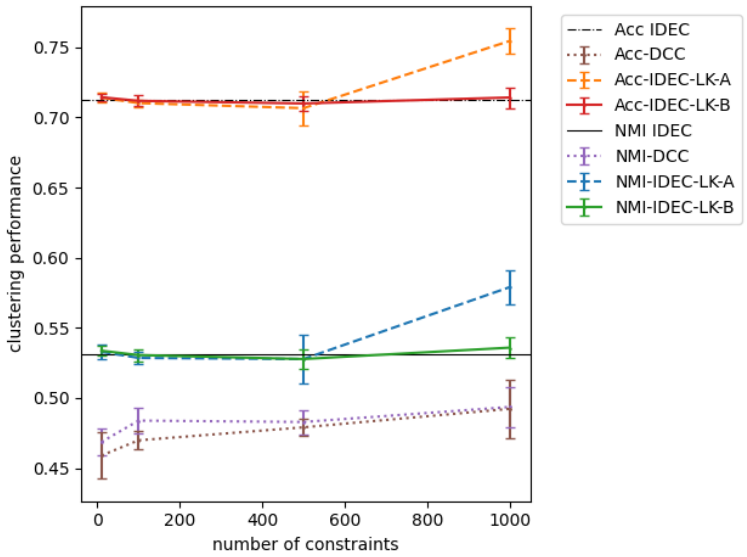

Figure 5.8: Clustering performance of triplet on Reuters dataset

5.4.4 Experiments and Analysis for Constraint Satisfaction

In this section, we aim at illustrating two points: first our method can leverage complex domain knowledge as introduced in a., second, it can learn from it, that is, it does not only aim at satisfying the constraints but it is also able to satisfy unseen constraints of the same type (shown in b.). The second point is crucial because acquiring constraints is expensive, and the set of training constraints is only a small fraction of all possible interpretations of the domain knowledge.

a. Implication constraint

As introduced in Section 5.2.5, an implication constraint has the form $P \implies Q$, the first part (if-clause) is denoted as P and the second part (then-clause) as Q. To study the interest of such constraints, we generate 5 sets of 100 constraints at random based on the ground truth. For each constraint, the number of Together/Apart constraints in P is 3, the number of Together/Apart in Q is 1 and we define the notion of P-Q distribution: around 20% constraints satisfy $P = \bot$ (that means that the left member is false), the remaining 80% is ($P = \top$, $Q = \top$). We run IDEC-LK-B with these constraints, since it is more efficient in terms of computation time than IDEC-LK-A (See more on Section 5.4.5).

Table 5.2: Comparison between IDEC and IDEC-LK-B on the satisfaction of implication constraint.

Data	Models	$\overline{Score_B}$	#Unsat
MNIST	IDEC	0.8777 ± 0.0118	13.6 ± 1.5
MNIST	IDEC-LK-B	**0.8856 ± 0.0130**	**12.4 ± 1.7**
Fashion	IDEC	0.7620 ± 0.0442	24.8 ± 4.6
Fashion	IDEC-LK-B	**0.7743 ± 0.0449**	**23.2 ± 4.5**
Reuters	IDEC	0.8290 ± 0.0376	17.8 ± 4.7
Reuters	IDEC-LK-B	**0.8357 ± 0.0381**	**17.0 ± 5.0**

In Table 5.2, IDEC-LK-B shows the improvement of average constrained scores ($Score_B$) compared to the value of IDEC. So, the number of unsatisfied constrained is reduced in all three datasets.

b. m-Clusters Group Constraint

In this experiment, we run IDEC-LK-B algorithm with MNIST with m-Clusters group constraints. Let us recall the definition of a m-cluster group constraint: a group of points which share some common features are required to belong to at most m clusters.

We selected 4 groups: $\mathcal{G} = \{G(3,9), G(6,8), G(1,7), G(2,5)\}$. They are pairs of digits that share some similar shapes. For a group $G(u,v)$, we selected randomly 100 images of either u digit or v digit. To observe the change clearer, we chose them so that a quarter of them have been assigned to a wrong cluster by Stacked

Denoising Autoencoder (SDAE) and K-Means. For each set of images, we determine two major clusters (clusters that contain the most points) and put constraints for each image to belong to one of these two clusters.

Improvement on the training data Figure 5.9 and Figure 5.10 show the training results of IDEC-LK. Each image is annotated by three numbers. The first number is the true label of the image. The second number is the clustering results of IDEC. We use Hungarian mapping [56] to match image labels. The last number is the clustering results of IDEC-LK.

After the training, all the input constraints are satisfied. In the two figures, images with black texts, which are already satisfied, do not change. In comparison, images with green texts have changed their assignments to one of the two specified clusters after IDEC-LK.

Improvement on the validation data For testing the generalization of our constraints, we generate all m-clusters group constraints from a validation set of 10,000 images. These constraints have not been used for learning. The number of unsatisfied constrained in the validation is significantly reduced by 85.7%. Figure 5.11 shows the change the number of satisfied constraints (blue parts) in G(3,9), G(6,8), G(1,7), G(2,5) after learning with IDEC-LK.

5.4.5 Experiments and Analysis for Complexity

Implication constraints Before computing the loss, we need to compile and optimize the SDD structure of each constraint. It is the main bottleneck for learning with more complex knowledge. Table 5.3 shows the differences between A and B formulation for implication constraints. We generate 100 Horn clauses of Together/Apart. The length is the number of Together/Apart constraints appearing in the clause. For each clause, we measure the time to construct and to optimize its SDD structure and the final size of the structure. Formulation B shows a smaller size and a better compiled time than formulation A. For 4-length constraints in MNIST dataset, the compilation for formulation A runs out of time, so in the following experiments, we will use only formulation B.

Table 5.3: Average SDD sizes and compilation times of a Horn clause using formulation A/B

Data	Length	Formulation	SDD size	Time (s)
MNIST	4	A	-	-
MNIST	4	B	**415.92 ± 169.54**	**0.502 ± 0.327**
Reuters	10	A	481.09 ± 103.96	0.284 ± 0.128
Reuters	10	B	**272.40 ± 79.43**	**0.131 ± 0.056**

Figure 5.9: Training results of G(1,7). Each image has three numbers from left to right: Label, Initial cluster assignment from IDEC, assignment after training with IDEC-LK. Black numbers mean that the initial assignment satisfies G(1,7) constraint. All of them do not change after the training. Green text is for the images that have changed their assignments to meet the requirement.

Figure 5.10: Training results of G(6,8). Each image has three numbers from left to right: Label, Initial cluster assignment from IDEC, assignment after training with IDEC-LK. Black numbers mean that the initial assignment satisfies G(6,8) constraint. All of them do not change after the training. Green text is for the images that have changed their assignments to meet the requirement.

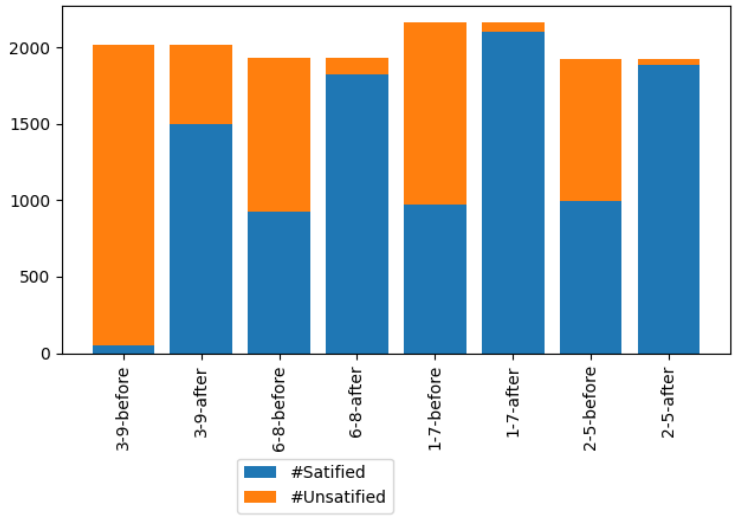

Figure 5.11: The constraint learning of m-clusters group constraint validated on unseen data

5.5 Conclusion

In this chapter, we present a general framework for integrating knowledge in constrained clustering problems. Because of the clustering context, the association between logical symbols and neurons of each point needs to consider the dependence of outputs and the partition constraints. Hence, we propose two scores for integrating expert constraints, and we show how these scores can be computed through Weight Model Counting. Relying on logic allows us to express many kinds of constraints. We show the flexibility and adaptability of our method by considering new constraints such as implication constraints or m-cluster group constraints. Next, IDEC-DK, which is a deep constrained clustering system based on a deep clustering model (IDEC), has been introduced. Our details in implementation and learning scheme help improve the learning efficiency in practical usages and easy for reusable/reproducible purposes. In our experiments, we obtain similar performance to other systems with well-known constraint types, but we also show the ability to integrate new constraints and even generalize the constraints to unseen points. We plan to embed our proposal in other deep clustering architectures to show the generality of our approach.

The main limitation of this work is the complexity for computing SDD trees. So, it prevents us from incorporating cluster-level constraints or constructing a single loss for the whole constraint set. So we aim at reducing complexity by introducing new formulations or approximation schemes. However, let us notice that SDD trees are computed only once at the beginning of the learning process.

Conclusion and outlook

Contribution

Constrained clustering or semi-supervised clustering is a dominant application of machine learning. Because of the natural process of data collection, the data are often unlabeled. Therefore, the task of clustering data is replied on the domain knowledge, which is expressed by the expert constraints. The constrained clustering algorithm could be targeting a specific type of constraint such as pairwise, triplet, or cluster-size constraints. Alternately, the algorithm could be a general framework that makes adding new constraints easy.

In our work, we are interested in integrating knowledge in general form in a clustering process. We consider it in two ways: classical optimization problem and deep learning problem.

The context of the first framework In classical approach, constrained clustering frameworks are developed using a general optimization tool, such as Integer Linear Programming (ILP) [3, 67, 76], Boolean satisfiability problem (SAT) [25, 65] or Constraint Programming [21, 22]. Most of these methods are in-process optimization which they try to find a global optimal solution that satisfies all the constraints. However, they suffer from a lack of efficiency and flexibility as they need to handle and specify both the clustering objective and the constraint requirements. Kuo et al.[58] propose a postprocess approach where the expert constraints are handled after receiving a clustering result. This approach simplifies the optimization problem and can be applied to an existing clustering algorithm. Nevertheless, the objective function, which is based on a hard clustering result, yield less competitive results than other works.

The first framework considers expert constraints as a posterior knowledge. The framework modifies the result from a clustering algorithm to satisfy the expert constraints. While the previous postprocess framework optimized the number of reassigned points, our works consider the "clustering score" - the cost of a new partition based on the clustering algorithm. We propose two clustering scores (see Section 4.1.2), one using the probability-based scores (soft scores) and one based on the distances. While the calculation of distances is straightforward, the formulation soft scores depend on the clustering methods. We have surveyed and formulated different ways to compute adapting for each set of clustering algorithms, such as centroid-based clustering, deep clustering, or density-based clustering. The experiments show a significant improvement from the previous postprocess work as our method better utilizes the information in the clustering process. Moreover, our clustering qualities for individual constraint types are competitive compared with the best algorithms for each kind. At the same time, the computational time is much less than the other in-process methods. A number of practical scenarios of

combining different constraint types are tested to prove the ability of our framework to handle multiple constraints on a large number of data points that the previous declarative models are incapable of.

The context of the second framework Recently, the neural network has gained popularity in both data representation and knowledge integration. The data representation is learned automatically by the machine through unique neural structures such as Convolutional Neural Network (CNN) and Autoencoder. So, clustering with neural networks (deep clustering) has become state-of-the-art for many datasets in various domains. However, to achieve the best performance for a dataset, it is necessary to add the human knowledge relevant to the data. One of the most promising directions of knowledge integration is to express the knowledge in a formal language such as first-order logic and then learn them with a regularized loss. This approach standardized the form of the knowledge, which integrates different types and even complex knowledge into the machine. There have been few works [104, 102] to apply this approach to the classification problem, but it has not only been studied and adapted for the constrained clustering.

The second framework is the first constrained clustering framework to integrate expert constraints as logical knowledge in neural network architecture. We propose two formulations, both of which are based on the neural outputs of a deep clustering model. So, the framework can be applied to different neural architectures and trained with/without a deep clustering process. In the first formulation, all of the logical variables are directly linked to the neurons. So the clustering condition needs to be enforced. The second formulation connects the outputs with a set of sentences so that the clustering context is taking care of and the formulations are only for expert constraints. Then, both of the formulations are translated directly into a semantics loss. Therefore, the methods are independent of the representation of knowledge. In other words, the expert constraints presented, whether in the disjunctive normal form (DNF), the conjunctive normal form (CNF), or an arbitrary form, produce the same result. Our use of the Sentential Decision Diagrams (SDD) reduces the complexity of loss calculation and enables the integration of complex constraints such as implication constraints.

Another important contribution is to formulate and apply new constraint types to the constrained clustering problem. From concept or initial ideas in the domain knowledge such as relative comparisons, fairness and feature sharing, we construct the formulation for logical triplet constraints, combined fairness and property-cardinality constraint. The experiments shows that integrating these constraints helps to improve the clustering output and achieve our target properties.

Future Work

Directions for future work are:

The construction of the allocation matrix. The first goal for the postprocess framework is to improve the construction of the cluster fractional allocation matrix (CFAM). In our proposed method for computing CFAM, the parameters impacting the degree of fuzziness such as v degree of freedom in Student's t-distribution are fixed for all datasets. It is important to see how this value can be chosen by internal criteria on clustering: maximum split, density of the cluster, Davies-Bouldin index.

Different ways to model the postprocess problem. The second goal for the postprocess framework is to improve the efficiency of the algorithm. It would be interesting to find different ways of modelling the constrained clustering through different variables and equation. An example could be a softer framework that allows some constraints to be not satisfied.

The genericity of our second framework. For the second framework, our first target is to consolidate the genericity of the work. We aim at experimenting with more deep clustering model on various datasets from different domains. Then, we can analyze the different in performance with other methods so that we can adapt our framework and our training scheme to each specific scenario.

Other methods for knowledge integration. Because of the complexity of Weighted Model Counting (WMC), we have had to make some restrictions on our framework such as handling constraints separately. Our second target for deep clustering with knowledge is to use other knowledge integration methods. For the learning with logical representation, the satisfaction of formulae can be learn by a graph embedding network instead of intractable calculation with WMC. A potential of equation-based or integer representation of expert constraints are also worth to further investigate.

Appendix A

Appendix

A.1 Sentential Decision Diagrams for Probabilistic Reasoning

A.1.1 Introduction

Given a scenario where the presentation of knowledge is in propositional logic (denoted as f). Probabilistic reasoning is method to indicate the uncertainty of the knowledge based on the uncertainty of the predicates.

In this section, we are going to introduce methods for probabilistic reasoning task for independent predicates.

Through out this section, upper case letters (e.g., X) will be used to denote variables and lower case letters to denote their instantiations (e.g., x). Bold upper case letters (e.g., $\mathbf{X}$) will be used to denote sets of variables and bold lower case letters to denote their instantiations (e.g., $\mathbf{x}$)

We denotes the probability when variable X instantiate to True ($\top$) is $P(X)$. Then, the probability of a Boolean function α defined over a set of independent variables $\mathbf{X} = \{X_1, \cdots, X_n\}$ is

$$P(\alpha) = \sum_{\mathbf{x} \models \alpha} \prod_{i:\mathbf{x} \models X_i} P(X) \prod_{i:\mathbf{x} \models \neg X_i} (1 - P(X)) \tag{A.1}$$

In literature, the right-hand side of A.1 is called Weighted Model Counting.

A.1.2 Shannon decomposition

The main approach to calculate the probability of a Boolean function f is through the decomposition. The simplest decomposition is the Shannon decomposition.

First, we define the following notations. The *conditioning* of f on instantiation x, written $f|x$, is a *subfunction* that results from setting variables X to their values in x. A function f essentially depends on variable X iff $f|X \neq f|\neg X$. We write $f(\mathbf{Z})$ to mean that f can only essentially depend on variables in $\mathbf{Z}$.

The decomposition process is as follows. First, select a variable X to decompose f: $f = f|X \vee f|\neg X$. Next, if $f|X$ depends on other variables, decompose it further. Apply the same procedure with $f|\neg X$.

Example Given a Boolean function is $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$.

The Shannon decomposition of f is

$$\begin{aligned} f &= (A \wedge f|A) \vee (\neg A \wedge f|\neg A) \\ &= (A \wedge [B \vee (C \wedge D)]) \vee (\neg A \wedge [(B \wedge C) \vee (C \wedge D)]) \\ &= (A \wedge [B \vee (\neg B \wedge C \wedge D)]) \vee (\neg A \wedge [(B \wedge C) \vee (\neg B \wedge C \wedge D)]) \\ &= \cdots \end{aligned} \tag{A.2}$$

A.1.3 Theory and method

a. (X,Y)-decomposition

If a Boolean function f can be expressed as: $f(\mathbf{X}, \mathbf{Y}) = (p_1(\mathbf{X}) \wedge s_1(\mathbf{Y}) \vee \cdots \vee (p_n(\mathbf{X}) \wedge s_n(\mathbf{Y}))$ then the set $\{\forall i \in [1, n] : (p_i, s_i)\}$ is called an $(\mathbf{X}, \mathbf{Y})$-decomposition of the function f.

A special case of decomposition is the partition. $(\mathbf{X}, \mathbf{Y})$ is a partition iff $(\forall i : p_i \neq false) \wedge (\forall i \neq j : p_i \wedge p_j = false) \wedge (\vee_i p_i = true)$

b. Vtree for variable X

There exist multiple ways to obtain $(\mathbf{X}, \mathbf{Y})$-decomposition. However, if the decomposition of $\mathbf{X}$ follows a vtree which is a full binary tree with its leaves corresponding to the variables in $\mathbf{X}$ then $(\mathbf{X}, \mathbf{Y})$- decomposition is uniquely determined. The structure of this decomposition is called Sentential Decision Diagram (SDD). An example of vtree and its corresponding SDD is shown in Figure A.1.

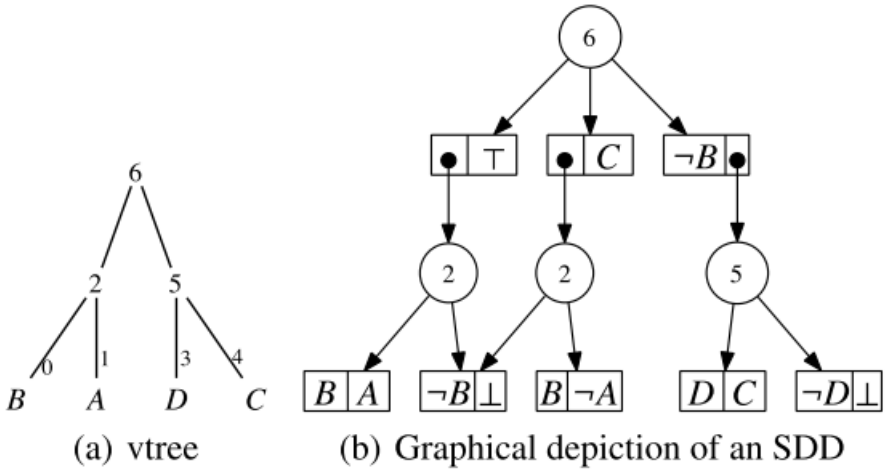

Figure A.1: A vtree (left) and its corresponding SDD (right) for function $f = (A \wedge B) \vee (B \wedge C) \vee (C \wedge D)$. Source: Darwiche[23].

c. SDD structure and optimization

Algorithm for optimizing the decomposition (SDD):

1. Constructing a vtree (a dissection of variable order)
2. Computing SDD
3. Optimizing vtree and adjusting SDD after each operation on v-tree

A.1.4 Examples in constrained clustering

Pairwise constraints Figure A.2, A.3 present the SDD structure of $cannot - link(u, v)$ using formulation A and B, respectively.

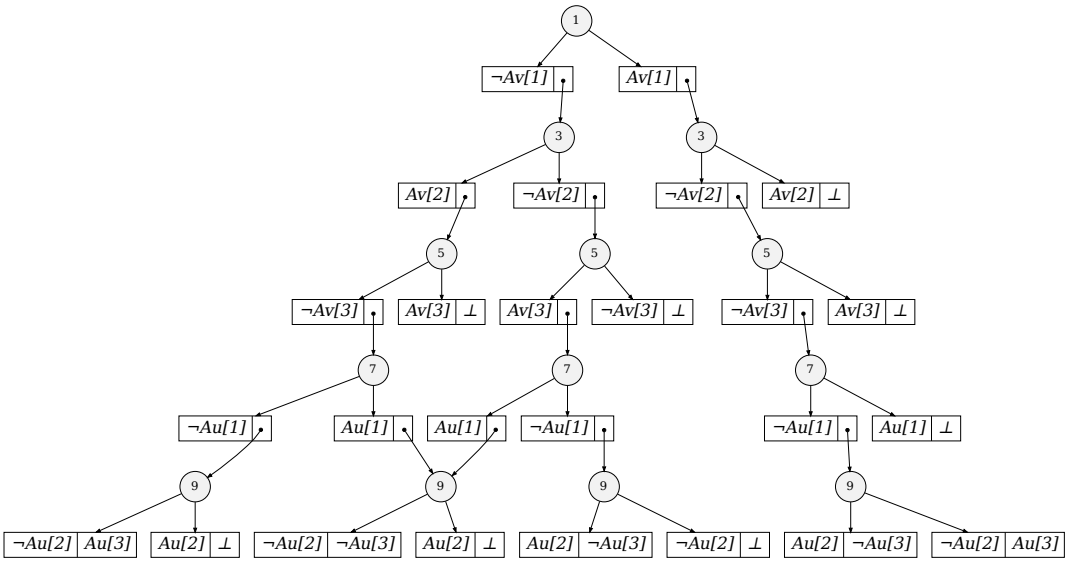

Figure A.2: The presentation of cannot-link constraint between u and v for $k = 3$ using formulation A

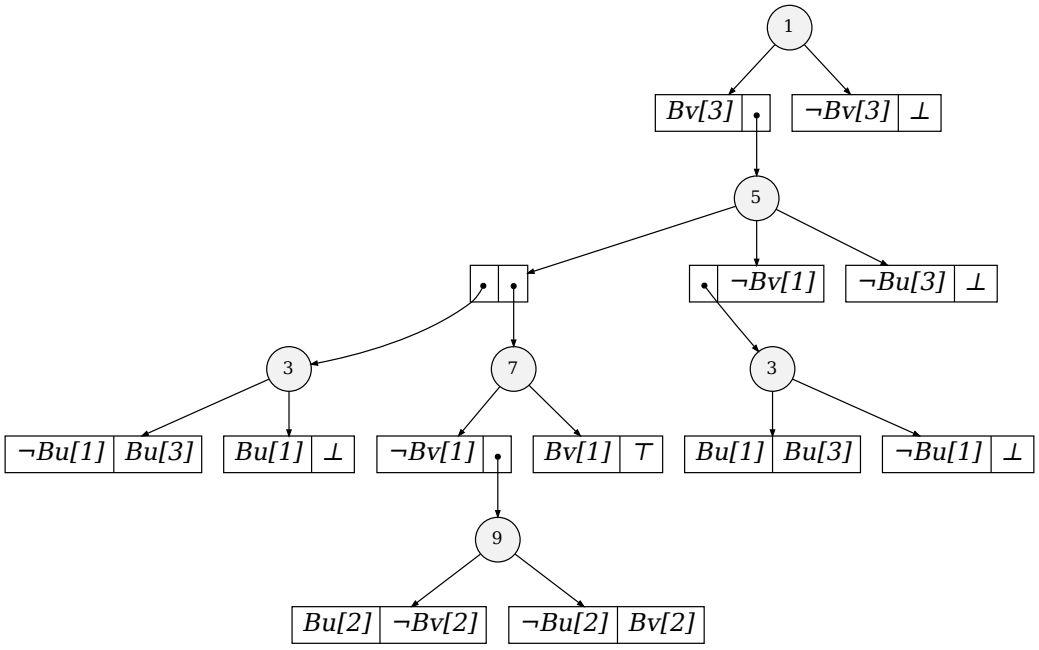

Figure A.3: The presentation of cannot-link constraint between u and v for $k = 3$ using formulation B

Triplet constraints

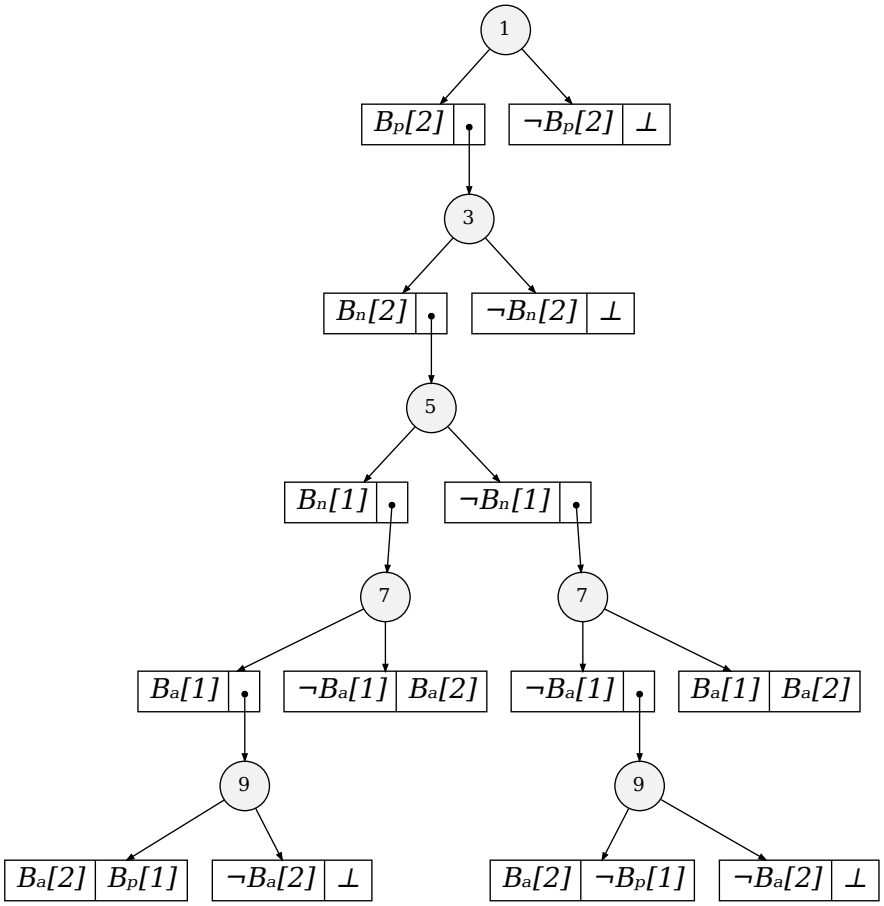

Figure A.4: Triplet constraint for $k = 2$ presented by B formulation

A.2 Propositions for B formulation

Definition In our main paper, we give the definitions of β and $\mathbf{B}$ as follows:

For each point i, we will define the formulas $\beta_{i1}, ..., \beta_{ik}$, such that β_{ij} interpreted to $\top$ means that the point i is assigned to cluster j. Let $\mathbf{B}$ be a set of logical variables $\{B_{ij} : i \in [1, n], j \in [1, k]\}$. The formula β_{ij} is defined as follows:

$$\begin{aligned} &\beta_{ij} \stackrel{\text{def}}{=} B_{ij} \wedge \bigwedge_{t\in[1,j-1]} \neg B_{it} \quad \text{for all } j \in [1, k-1], \\ &\beta_{ik} \stackrel{\text{def}}{=} \bigwedge_{t\in[1,k-1]} \neg B_{it} \end{aligned} \tag{A.3}$$

We define, as follows, the weight w_B for the variables:

$$w_B(B_{ij}) = \begin{cases} \frac{S_{ij}}{1-\sum_{t\in[1,j-1]} S_{it}} & \text{if} \quad \sum_{t\in[1,j-1]} S_{it} < 1 \\ 1 & \text{otherwise} \end{cases} \tag{A.4}$$

a. Proposition 1.

For any constrained problem P_c, we have $WMC(P_c) = \sum_{p\in M_c} \prod_{i\in[1,n]} S_{ipi}$.

Proof. We have: $WMC(P_c) = \sum_{p\in M_c} WMC(p) = \sum_{p\in M_c} WMC(\wedge_{i\in[1,n]} \beta_{ip_i})$.

With the definition given in (A.3), we can observe that for $i \neq i'$, the formulas β_{ip_i} and $\beta_{i'p_{i'}}$ do not share any common variables. Using axiom 6 in [104], we have $WMC(\wedge_{i\in[1,n]} \beta_{ip_i}) = \prod_{i\in[1,n]} WMC(\beta_{ip_i})$.

According to Lemma 2, we have $WMC(\beta_{ipi}) = S_{ip_i}$. Therefore $WMC(P_c) = \sum_{p\in M_c} \prod_{i\in[1,n]} S_{ip_i}$.

b. Proposition 2.

The clustering condition is always satisfied with any instantiation of $\mathbf{B}$.

Proof. The clustering condition states that all points must be assigned to at most one cluster and all points must be assigned to at least one cluster. The first statement is proven in Lemma 3 and the second in Lemma 4.

The proofs are the same for all points i. Therefore, for sake of simplicity we remove the index i. This leads to the definitions as follow.

Let $\mathbf{B}$ be a set of logical variables $\{B_j : j \in [1, k]\}$. The formula β_j is defined as follows:

$$\begin{aligned} &\beta_j \stackrel{\text{def}}{=} B_j \wedge \bigwedge_{t\in[1,j-1]} \neg B_t \quad \text{for all } j \in [1, k-1], \\ &\beta_k \stackrel{\text{def}}{=} \bigwedge_{t\in[1,k-1]} \neg B_t \end{aligned} \tag{A.5}$$

If we call s the row S_i (corresponding to i) in the matrix S, the weight w_B is corresponding to:

$$w_B(B_j) = \begin{cases} \frac{s_j}{1-\sum_{t\in[1,j-1]} s_t} & \text{if} \quad \sum_{t\in[1,j-1]} s_t < 1 \\ 1 & \text{otherwise} \end{cases} \tag{A.6}$$

A.2.1 Lemmas and proofs

a. Lemma 1.

For all $j \in [1..k]$, there exists an assignment $\mathbf{b}$ for each B_l such that $\mathbf{b} \models \beta_j$.

Proof. For $j = 1$, $\beta_1 \stackrel{\text{def}}{=} B_1$. Let $\mathbf{b}$ be such that $B_1 = true$. We have $\mathbf{b} \models \beta_1$.

For $2 \leq j \leq k-1$, $\beta_j \stackrel{\text{def}}{=} B_j \wedge \bigwedge_{l<j} \neg B_l$. Let $\mathbf{b}$ be such that $B_j = true$ and $B_l = false$ for all $l < j$. We have $\mathbf{b} \models \beta_j$.

For $j = k$, $\beta_j \stackrel{\text{def}}{=} \bigwedge_{l<k} \neg B_l$. Let $\mathbf{b} = \{\neg B_l : 1 \leq l \leq k-1\}$. We have $\mathbf{b} \models \beta_k$

b. Lemma 2.

The weighted model counting of β_j is equal to s_j.

$$\sum_{\mathbf{b} \models \beta_j} \prod_{B \in \mathbf{b}} w_B(B) \prod_{\neg B \in \mathbf{b}} (1 - w_B(B)) = s_j \tag{A.7}$$

Proof. We consider two cases: $j \in [1, k-1]$ and $j = k$. For $j \in [1, k-1]$, we denote $\mathbf{b_{prefix}} = \{b_p : p \in [1, j]\}$ and $\mathbf{b_{postfix}} = \{b_p : p \in [j+1, k-1]\}$.

$$
\begin{aligned}
&\sum_{\mathbf{b}\models\beta_j}\prod_{B\in\mathbf{b}} w_B(B)\prod_{\neg B\in\mathbf{b}}(1-w_B(B))\\
=&\sum_{\mathbf{b_{prefix}}\models\beta_j}\left[\prod_{B\in\mathbf{b_{prefix}}} w_B(B)\prod_{\neg B\in\mathbf{b_{prefix}}}(1-w_B(B))\sum_{\mathbf{b_{postfix}}\models\beta_j}\prod_{B\in\mathbf{b_{postfix}}} w_B(B)\prod_{\neg B\in\mathbf{b_{postfix}}}(1-w_B(B))\right]\\
=&\sum_{\mathbf{b_{prefix}}\models\beta_j}\left[\prod_{B\in\mathbf{b_{prefix}}} w_B(B)\prod_{\neg B\in\mathbf{b_{prefix}}}(1-w_B(B))\sum_{\mathbf{b_{postfix}}\models\top}\prod_{B\in\mathbf{b_{postfix}}} w_B(B)\prod_{\neg B\in\mathbf{b_{postfix}}}(1-w_B(B))\right]\\
&\text{(because no variables of } \mathbf{b_{postfix}} \text{ appear in } \beta_j)\\
=&\sum_{\mathbf{b_{prefix}}\models\beta_j}\left[\prod_{B\in\mathbf{b_{prefix}}} w_B(B)\prod_{\neg B\in\mathbf{b_{prefix}}}(1-w_B(B))\times 1\right]\\
&\text{(because } WMC(\top)=1)\\
=&\sum_{\mathbf{b_{prefix}}=\{(\{\neg B_l:l\in[1,j-1]\},B_j\})}\left[\prod_{B\in\mathbf{b_{prefix}}} w_B(B)\prod_{\neg B\in\mathbf{b_{prefix}}}(1-w_B(B))\right]\\
=&w_B(B_j)\prod_{l\in[1,j-1]}(1-w_B(B_l))\\
=&\frac{s_j}{1-\sum_{t\in[1,j-1]}s_t}\prod_{l\in[1,j-1]}\left(1-\frac{s_l}{1-\sum_{t\in[1,l-1]}s_t}\right)\\
=&\frac{s_j}{1-\sum t\in[1,j-1]s_t}\times(1-s_1)\times\frac{1-s_1-s_2}{1-s_1}\times\ldots\times\frac{1-\sum_{t\in[1,j-1]}s_t}{1-\sum_{t\in[1,j-2]}s_t}\\
=&s_j
\end{aligned}
\tag{A.8}
$$

For $j=k$,

$$
\begin{aligned}
&\sum_{\mathbf{b}\models\beta_k}\prod_{B\in\mathbf{b}} w_B(B)\prod_{\neg B\in\mathbf{b}}(1-w_B(B))\\
=&\prod_{j\in[1,k-1]}(1-w_B(B_j))\\
=&\prod_{j\in[1,k-1]}\left(1-\frac{s_j}{1-\sum_{t\in[1,j-1]}s_t}\right)\\
=&(1-s_1)\times\frac{1-s_1-s_2}{1-s_1}\times\ldots\times\frac{1-\sum_{t\in[1,k-1]}s_t}{1-\sum_{t\in[1,k-2]}s_t}\\
=&1-\sum_{t\in[1,k-1]}s_t\\
=&s_k
\end{aligned}
\tag{A.9}
$$

c. Lemma 3.

A point is assigned to at most one cluster. That means that for all $i, j \in [1, k], i \neq j$ we have:

$$\neg\beta_i \vee \neg\beta_j \equiv \top \tag{A.10}$$

Proof. Without loss of generality, we assume $i < j$. We consider two cases: when $j < k$ and when $j = k$.

Case 1: $i < j < k$, we have $\neg\beta_i \vee \neg\beta_j$

$$\begin{aligned}
\neg\beta_i \vee \neg\beta_j \equiv & \quad \neg(\wedge_{t\in[1,i-1]}\neg B_t \wedge B_i) \vee \neg(\wedge_{t\in[1,j-1]}\neg B_t \wedge B_j) \\
\equiv & \quad \vee_{t\in[1,i-1]} B_t \vee \neg B_i \vee_{t\in[1,j-1]} B_t \vee \neg B_j \\
\equiv & \quad \vee_{t\in[1,i-1]} B_t \vee \neg B_i \vee B_i \vee_{t\in[i+1,j-1]} B_t \vee \neg B_j \\
\equiv & \quad \top
\end{aligned}$$

Case 2: $i < jandj = k$, we have $\neg\beta_i \vee \neg\beta_k$

$$\begin{aligned}
\neg\beta_i \vee \neg\beta_k \equiv & \quad \neg(\wedge_{t\in[1,i-1]}\neg B_t \wedge B_i) \vee \neg(\wedge_{t\in[1,k-1]}\neg B_t) \\
\equiv & \quad \vee_{t\in[1,i-1]} B_t \vee \neg B_i \vee_{t\in[1,k-1]} B_t \\
\equiv & \quad \vee_{t\in[1,i-1]} B_t \vee \neg B_i \vee B_i \vee_{t\in[i+1,k-1]} B_t \\
\equiv & \quad \top
\end{aligned}$$

d. Lemma 4.

A point must be assigned to at least one cluster, that means:

$$\vee_{i\in[1,k]}\, \beta_i \equiv \top \tag{A.11}$$

Proof. We have:

$$\begin{aligned}
\vee_{i\in[1,k]}\beta_i \equiv & \quad \vee_{i\in[1,k-1]}(\wedge_{t\in[1,i-1]}\neg B_t \wedge B_i) \vee (\wedge_{t\in[1,k-1]}\neg B_t) \\
\equiv & \quad (B_1 \vee (\neg B_1 \wedge B_2)) \vee_{i\in[3,k-1]} (\wedge_{t\in[1,i-1]}\neg B_t \wedge B_i) \vee (\wedge_{t\in[1,k-1]}\neg B_t) \\
\equiv & \quad (B_1 \vee B_2) \vee_{i\in[3,k-1]} (\wedge_{t\in[1,i-1]}\neg B_t \wedge B_i) \vee (\wedge_{t\in[1,k-1]}\neg B_t) \\
\equiv & \quad \ldots \\
\equiv & \quad (B_1 \vee B_2 \vee \ldots \vee B_{k-1}) \vee (\wedge_{t\in[1,k-1]}\neg B_t) \\
\equiv & \quad (\vee_{t\in[1,k-1]} B_t) \vee \neg(\vee_{t\in[1,k-1]} B_t) \\
\equiv & \quad \top
\end{aligned}$$

A.3 Performance of SDAE+Kmeans

In Table A.1, A.2, and A.3, we use the same set of hyper-parameters.

In SDAE, IDEC, DCC and EDEC, we use the same neural architecture for the autoencoder. The encoder network is a fully connected multi-layer perceptron with dimensions d-500-500-2000-10 for all datasets, where d is the dimension of input data. The decoder network is a mirror of the encoder. All the internal layers are activated by the ReLU [70] function.

The number of epochs for training each layer is 300. The number of epochs for training the whole autoencoder is 500. The optimizer is Stochastic Gradient Descent (SGD) with a momentum of 0.9. The initial learning rate is 0.1 and decreases by one-tenth every 100th epoch. In all the training, the ratio of corruption is 0.2, meaning that 20% of inputs are set to 0.

Table A.1: Raw training results on MNIST with SDAE + Kmeans

Data	Run	NMI	ACC
MNIST	0	0.7653	0.8270
MNIST	1	0.7652	0.8290
MNIST	2	0.7554	0.8141
MNIST	3	0.7597	0.8173
MNIST	4	0.7615	0.8198
MNIST	Average	0.7614 ± 0.0037	0.8214 ± 0.0057

Table A.2: Raw training results on Fashion with SDAE + Kmeans

Data	Run	NMI	ACC
Fashion	0	0.5842	0.5170
Fashion	1	0.5723	0.5089
Fashion	2	0.5688	0.4979
Fashion	3	0.5885	0.5312
Fashion	4	0.5899	0.5224
Fashion	Average	0.5807 ± 0.0086	0.5155 ± 0.0114

Table A.3: Raw training results on Reuters with SDAE + Kmeans

Data	Run	NMI	ACC
Reuters	0	0.5484	0.7371
Reuters	1	0.5222	0.7162
Reuters	2	0.5229	0.7138
Reuters	3	0.5171	0.7626
Reuters	4	0.4473	0.6612
Reuters	Average	0.5116 ± 0.0339	0.7182 ± 0.0335

A.4 Performance of IDEC

In Table A.4, we report IDEC results (with five trials as well) using the same pre-trained model computed by SDAE. Compared to SDAE+Kmeans, IDEC improved significantly on the MNIST dataset and has modest or no improvement on other datasets.

The optimizer is Adam with the learning rate of 0.001 [53]. The maximum number of epochs is 200 but it can stop before if the change of cluster assignment compared to the last epoch is less than 0.1%.

Table A.4: SDAE+IDEC performance on MNIST, Fashion and Reuters

Data	Model	NMI	ACC
MNIST	SDAE+IDEC	0.8668 ± 0.0005	0.8814 ± 0.0011
Fashion	SDAE+IDEC	0.5966 ± 0.0027	0.5183 ± 0.0033
Reuters	SDAE+IDEC	0.5309 ± 0.0015	0.7121 ± 0.0010

A.5 Constrained Clustering Results

A.5.1 Pairwise constraints

In Table A.5, we report Normalized Mutual Information (NMI), Accuracy (ACC), comparison to IDEC (vsIDEC), number of unsatisfied constraints and time running (in seconds). For each dataset and one specific number of constraints, we generate five random sets of constraints (we called them test cases). Then, we run each method once for each test case to measure the average and standard deviation of the metrics mentioned above. In the NMI and ACC columns, we highlight the best performance by green color and the competitive performances (p-value of the same hypothesis > 0.05) by blue color. In the vsIDEC column, the first number is the p-value of the KS test [41] testing if the NMI of IDEC is similar to the compared method, the second number is the comparison with accuracy. We highlight in bold when the average value of the constrained clustering method is better than IDEC value.

Table A.6 shows how differs each run of IDEC-LK with the same test case (i.e. the same set of constraints). The difference between runs of the same test case is less than the difference between different test cases (different sets of constraints). Overall, IDEC-LK shows a relatively small change between each run.

A.5.2 Triplet constraints

The performances with triplet constraints are reported similar to those of pairwise constraints in Table A.7.

Table A.5: Comparison on clustering quality between the baselines and our IDEC-LK with pairwise constraints. Green and blue numbers are for the best and second-best values, respectively.

Data	N	Models	NMI	ACC	vsIDEC	#Unsat	Time (s)
CIFAR10	100	DCC	0.1211 ± 0.0015	0.2489 ± 0.0015	**0.01** **0.01**	0.8000 ± 0.7483	103 ± 6
CIFAR10	100	MPCK-means	0.1139 ± 0.0022	0.2380 ± 0.0066	0.01 0.36	0 ± 0	63.66 ± 0.35
CIFAR10	100	PCK-means	0.1183 ± 0.0009	0.2413 ± 0.0044	**0.36** **0.87**	10.00 ± 3.58	53.87 ± 15.47
CIFAR10	100	IDEC-LK-A	0.1167 ± 0.0014	0.2419 ± 0.0026	0.36 **0.36**	0.2000 ± 0.4000	511 ± 18
CIFAR10	1000	DCC	0.1202 ± 0.0014	0.2448 ± 0.0051	**0.01** **0.36**	26.40 ± 4.36	93.03 ± 25.83
CIFAR10	1000	MPCK-means	0.1145 ± 0.0064	0.2363 ± 0.0114	0.36 0.08	0 ± 0	247 ± 3
CIFAR10	1000	PCK-means	0.1175 ± 0.0027	0.2423 ± 0.0056	**0.87** **0.36**	81.60 ± 2.24	69.91 ± 20.72
CIFAR10	1000	IDEC-LK-A	0.1189 ± 0.0009	0.2424 ± 0.0013	**0.08** **0.36**	3.40 ± 1.74	475 ± 17
CIFAR10	1000	IDEC-LK-B	0.1199 ± 0.0009	0.2426 ± 0.0017	**0.01** **0.36**	1.80 ± 0.75	631 ± 16
MNIST	100	DCC	0.8682 ± 0.0011	0.8817 ± 0.0017	**0.36** **1.00**	0.2000 ± 0.4000	288 ± 8
MNIST	100	MPCK-means	0.7154 ± 0.0198	0.7356 ± 0.0313	0.01 0.01	0 ± 0	58.53 ± 1.37
MNIST	100	PCK-means	0.7477 ± 0.0199	0.7743 ± 0.0509	0.01 0.01	3.60 ± 2.33	44.31 ± 15.60
MNIST	100	IDEC-LK-A	0.8677 ± 0.0010	0.8815 ± 0.0005	**0.36** **0.87**	0 ± 0	240 ± 7
MNIST	100	IDEC-LK-B	0.8672 ± 0.0012	0.8814 ± 0.0011	**0.87** **1.00**	0 ± 0	263 ± 9
MNIST	1000	DCC	0.8689 ± 0.0008	0.8815 ± 0.0007	**0.01** **0.87**	5.60 ± 1.36	277 ± 9
MNIST	1000	MPCK-means	0.7589 ± 0.0171	0.7788 ± 0.0413	0.01 0.01	0 ± 0	211 ± 3
MNIST	1000	PCK-means	0.7463 ± 0.0228	0.7698 ± 0.0543	0.01 0.01	26.00 ± 5.80	32.97 ± 15.90
MNIST	1000	IDEC-LK-A	0.8682 ± 0.0013	0.8825 ± 0.0011	**0.36** **0.87**	0 ± 0	240 ± 10
MNIST	1000	IDEC-LK-B	0.8680 ± 0.0017	0.8826 ± 0.0012	**0.08** **0.36**	0 ± 0	388 ± 27
Fashion	100	DCC	0.5945 ± 0.0032	0.5183 ± 0.0037	0.87 **0.87**	1.40 ± 1.02	176 ± 45
Fashion	100	MPCK-means	0.5747 ± 0.0124	0.5122 ± 0.0403	0.08 0.36	0 ± 0	60.12 ± 1.34
Fashion	100	PCK-means	0.5756 ± 0.0110	0.5228 ± 0.0067	0.01 **0.87**	4.80 ± 1.17	38.59 ± 11.26
Fashion	100	IDEC-LK-A	0.5984 ± 0.0010	0.5205 ± 0.0016	**0.36** **0.36**	0.2000 ± 0.4000	247 ± 0
Fashion	100	IDEC-LK-B	0.5976 ± 0.0013	0.5210 ± 0.0030	**0.36** **0.36**	0.2000 ± 0.4000	270 ± 23
Fashion	1000	DCC	0.6000 ± 0.0019	0.5241 ± 0.0039	**0.08** **0.36**	26.80 ± 4.12	140 ± 16
Fashion	1000	MPCK-means	0.5749 ± 0.0138	0.5312 ± 0.0292	0.08 **0.36**	0 ± 0	205 ± 4
Fashion	1000	PCK-means	0.5714 ± 0.0212	0.5314 ± 0.0293	0.08 **0.36**	44.40 ± 7.39	37.02 ± 13.19
Fashion	1000	IDEC-LK-A	0.6024 ± 0.0013	0.5259 ± 0.0024	**0.08** **0.08**	1.0000 ± 1.5492	248 ± 1
Fashion	1000	IDEC-LK-B	0.6009 ± 0.0019	0.5230 ± 0.0034	**0.08** **0.36**	1.40 ± 1.85	358 ± 17
Reuters	100	DCC	0.5450 ± 0.0050	0.7248 ± 0.0039	**0.01** **0.01**	1.40 ± 1.02	2.90 ± 0.55
Reuters	100	MPCK-means	0.5086 ± 0.0357	0.6943 ± 0.0744	0.36 0.36	0.2000 ± 0.4000	53.27 ± 0.62
Reuters	100	PCK-means	0.5224 ± 0.0218	0.7557 ± 0.0425	0.36 **0.08**	9.40 ± 1.36	14.82 ± 5.72
Reuters	100	IDEC-LK-A	0.5376 ± 0.0033	0.7148 ± 0.0021	**0.01** **0.36**	0 ± 0	5.17 ± 0.53
Reuters	100	IDEC-LK-B	0.5337 ± 0.0049	0.7133 ± 0.0028	**0.87** **0.87**	0 ± 0	7.33 ± 2.23
Reuters	1000	DCC	0.5655 ± 0.0086	0.7477 ± 0.0030	**0.01** **0.01**	48.00 ± 6.20	3.46 ± 0.41
Reuters	1000	MPCK-means	0.5262 ± 0.0330	0.7251 ± 0.0412	0.36 **0.36**	0 ± 0	167 ± 2
Reuters	1000	PCK-means	0.5174 ± 0.0288	0.7343 ± 0.0377	0.87 **0.08**	104 ± 17	14.80 ± 2.34
Reuters	1000	IDEC-LK-A	0.6317 ± 0.0120	0.7786 ± 0.0072	**0.01** **0.01**	0.4000 ± 0.8000	17.19 ± 20.09
Reuters	1000	IDEC-LK-B	0.5927 ± 0.0105	0.7563 ± 0.0079	**0.01** **0.01**	0.2000 ± 0.4000	27.52 ± 9.88

Table A.6: Stability of pairwise IDEC-LK for five different runs for one test case

Data	N	Models	NMI	ACC	vsIDEC	#Unsat	Time (s)
MNIST	1000	IDEC-LK-A	0.8675 ± 0.0008	0.8834 ± 0.0003	**0.36** **0.08**	0 ± 0	234 ± 5
MNIST	1000	IDEC-LK-B	0.8671 ± 0.0009	0.8824 ± 0.0008	**0.87** **0.36**	0 ± 0	396 ± 20
Fashion	1000	IDEC-LK-A	0.6019 ± 0.0006	0.5261 ± 0.0038	**0.08** **0.08**	1.0000 ± 0.6325	248 ± 1
Fashion	1000	IDEC-LK-B	0.6013 ± 0.0008	0.5273 ± 0.0011	**0.08** **0.01**	0.6000 ± 0.4899	362 ± 5
Reuters	1000	IDEC-LK-A	0.6202 ± 0.0026	0.7746 ± 0.0020	**0.01** **0.01**	0 ± 0	8.77 ± 2.08
Reuters	1000	IDEC-LK-B	0.5870 ± 0.0024	0.7519 ± 0.0020	**0.01** **0.01**	0 ± 0	27.25 ± 3.76

Table A.7: Comparison on triplet constraints with DCC and IDEC-LK

Data	N	Models	NMI	ACC	vsIDEC		#Unsat	Time (s)
MNIST	10	DCC	0.8662 ± 0.0003	0.8805 ± 0.0004	0.08	0.87	0 ± 0	133 ± 6
MNIST	10	IDEC-LK-A	0.8676 ± 0.0009	0.8813 ± 0.0005	**0.36**	0.87	0 ± 0	235 ± 3
MNIST	10	IDEC-LK-B	0.8665 ± 0.0012	0.8800 ± 0.0006	0.36	0.36	0 ± 0	253 ± 3
MNIST	100	DCC	0.8659 ± 0.0002	0.8805 ± 0.0011	0.01	0.87	0 ± 0	127 ± 5
MNIST	100	IDEC-LK-A	0.8678 ± 0.0020	0.8815 ± 0.0017	**0.87**	**0.87**	0 ± 0	252 ± 12
MNIST	100	IDEC-LK-B	0.8666 ± 0.0016	0.8812 ± 0.0011	0.87	1.00	0 ± 0	436 ± 2
MNIST	500	DCC	0.8669 ± 0.0005	0.8817 ± 0.0016	**0.87**	**1.00**	1.80 ± 0.75	151 ± 11
MNIST	500	IDEC-LK-A	0.8631 ± 0.0020	0.8792 ± 0.0012	0.01	0.08	0 ± 0	296 ± 14
MNIST	500	IDEC-LK-B	0.8685 ± 0.0010	0.8815 ± 0.0008	**0.01**	**0.87**	0 ± 0	1263 ± 2
MNIST	1000	DCC	0.8692 ± 0.0006	0.8855 ± 0.0015	**0.01**	**0.08**	2.40 ± 1.36	191 ± 21
MNIST	1000	IDEC-LK-A	0.8588 ± 0.0017	0.8771 ± 0.0013	0.01	0.01	0.8000 ± 0.7483	315 ± 1
MNIST	1000	IDEC-LK-B	0.8682 ± 0.0013	0.8812 ± 0.0011	**0.08**	1.00	0 ± 0	2398 ± 74
Fashion	10	DCC	0.5934 ± 0.0008	0.5119 ± 0.0032	0.08	0.08	0 ± 0	98.69 ± 8.97
Fashion	10	IDEC-LK-A	0.5981 ± 0.0005	0.5222 ± 0.0027	**0.36**	**0.36**	0 ± 0	246 ± 1
Fashion	10	IDEC-LK-B	0.5965 ± 0.0020	0.5194 ± 0.0040	1.00	**0.87**	0 ± 0	266 ± 5
Fashion	100	DCC	0.5927 ± 0.0013	0.5111 ± 0.0026	0.08	0.08	1.80 ± 1.33	90.90 ± 7.01
Fashion	100	IDEC-LK-A	0.5990 ± 0.0010	0.5236 ± 0.0034	**0.36**	**0.36**	0 ± 0	254 ± 1
Fashion	100	IDEC-LK-B	0.5969 ± 0.0014	0.5175 ± 0.0021	**1.00**	0.87	0.2000 ± 0.4000	456 ± 3
Fashion	500	DCC	0.5989 ± 0.0020	0.5219 ± 0.0067	**0.36**	**0.36**	8.80 ± 2.32	141 ± 26
Fashion	500	IDEC-LK-A	0.5989 ± 0.0014	0.5241 ± 0.0028	**0.36**	**0.08**	1.20 ± 1.17	285 ± 0
Fashion	500	IDEC-LK-B	0.5992 ± 0.0015	0.5228 ± 0.0022	**0.08**	**0.36**	1.20 ± 1.17	1312 ± 1
Fashion	1000	DCC	0.6009 ± 0.0042	0.5370 ± 0.0048	**0.36**	**0.01**	15.20 ± 4.40	270 ± 38
Fashion	1000	IDEC-LK-A	0.5981 ± 0.0028	0.5286 ± 0.0053	**0.87**	**0.08**	8.80 ± 0.98	329 ± 0
Fashion	1000	IDEC-LK-B	0.6003 ± 0.0013	0.5283 ± 0.0065	**0.08**	**0.08**	4.60 ± 3.38	2413 ± 9
Reuters	10	DCC	0.4687 ± 0.0094	0.4590 ± 0.0165	0.01	0.01	0 ± 0	4.52 ± 0.58
Reuters	10	IDEC-LK-A	0.5329 ± 0.0056	0.7140 ± 0.0034	**0.36**	**0.08**	0 ± 0	5.86 ± 1.92
Reuters	10	IDEC-LK-B	0.5337 ± 0.0034	0.7143 ± 0.0028	**0.08**	**0.36**	0 ± 0	6.31 ± 1.26
Reuters	100	DCC	0.4839 ± 0.0091	0.4698 ± 0.0066	0.01	0.01	0 ± 0	4.62 ± 0.55
Reuters	100	IDEC-LK-A	0.5285 ± 0.0044	0.7102 ± 0.0026	0.36	0.36	0 ± 0	8.43 ± 2.77
Reuters	100	IDEC-LK-B	0.5306 ± 0.0043	0.7118 ± 0.0040	0.87	0.87	0 ± 0	15.12 ± 3.90
Reuters	500	DCC	0.4829 ± 0.0087	0.4791 ± 0.0062	0.01	0.01	0.8000 ± 0.7483	5.71 ± 0.74
Reuters	500	IDEC-LK-A	0.5277 ± 0.0176	0.7065 ± 0.0121	0.36	0.36	0.4000 ± 0.8000	10.60 ± 2.38
Reuters	500	IDEC-LK-B	0.5278 ± 0.0069	0.7099 ± 0.0051	0.87	0.36	0 ± 0	43.06 ± 15.99
Reuters	1000	DCC	0.4936 ± 0.0145	0.4922 ± 0.0211	0.01	0.01	0.6000 ± 0.4899	10.80 ± 1.70
Reuters	1000	IDEC-LK-A	0.5790 ± 0.0119	0.7544 ± 0.0088	**0.01**	**0.01**	4.40 ± 3.50	16.31 ± 5.27
Reuters	1000	IDEC-LK-B	0.5359 ± 0.0075	0.7142 ± 0.0074	**0.36**	**0.36**	0.2000 ± 0.4000	102 ± 29

A.6 Complexity and Runtime of IDEC-LK

A.6.1 Complexity in training

The runtimes are reported in the Table A.5, A.7. Our performance is worse than COP-Kmeans and DCC but it is still reasonable. The runtime is less than 10 minutes for 100 pairwise/triplet constraints and less than 1 hours for 1000.

A.6.2 Complexity in constructing implication constraints

Before computing the loss, we need to compile and optimize the SDD structure of each constraint. It is the main bottleneck for learning with more complex knowledge. Table A.8 shows the differences between A and B formulation for implication constraints. We generate 100 Horn clauses of Together/Apart. The length is the number of Together/Apart constraints appearing in the clause. For each clause, we measure time to construct and optimize its SDD structure and the final size of the structure. Formulation B shows a smaller size and a better compiled time than formulation A.

Table A.8: Comparison of compile times of implication constraints

Data	Length	Formulation	SDD size	Time (s)
Reuters	10	A	481.09 ± 103.96	0.284 ± 0.128
Reuters	10	B	272.40 ± 79.43	0.131 ± 0.056

A.7 Method to calculate WMC

First, we represent the logical knowledge α in Sentential Decision Diagram, a hierarchical structure. Each node in the diagram is either a terminal node (containing a constant or literal) or a decomposition.

We use a recursive method to compute the root node's value (equivalent to $WMC(\alpha)$). If a node is a terminal node, we return its value directly. Otherwise, we calculate and sum all of its children nodes. During the calculation, we keep the results of all decompositions in case of revisiting them. The detailed algorithm is shown in Algorithm A.1.

A.8 Sensitivity experiments for hyper-parameters

The impact of λ_e on the final partition has been measured using IDEC-LK-A and IDEC-LK-B systems with the Fashion and Reuters dataset. The test scenario has 5 cases, each with 1,000 randomly selected pairwise constraints. For each test scenario, λ_e is tested with $0.01, 0.1, 1.0$ values.

In all cases, when λ_e increases, the average value of WMC and the number of satisfied constraints increase.

Algorithm A.1 Method to calculate WMC

```
class ProbCalculator:
    def init(self, probs):
        # Weights of all variables
        self.probs = probs
        # Keep the values of calcuated noded during the
           search
        self.cal_nodes = dict()

    # Recursive method to calculate WMC of a node using its
       children
    def calculate_re(self, node):
        # If node has already computed, take it from
           cal_nodes
        if node.id in self.cal_nodes:
            return self.cal_nodes[node.id]
        # If it is a simple node, return its value
        if node.node_size() == 0:
            if node.literal >= 0:
                return self.probs[node.literal - 1]
            return 1.0 - self.probs[- node.literal - 1]
        # If not, decompose them
        t = node.elements()
        ans = 0.0
        for i in range(len(t)):
            if t[i][1].is_false():
                continue
            # Recursively calculate the children nodes
            u = self.calculate_re(t[i][0])
            v = self.calculate_re(t[i][1])
            # Adding them to the final result
            ans += u * v
        # Save the result to cal_nodes and return it.
        self.cal_nodes[node.id] = ans
        return ans

    # Public method for external use
    def calculate(self, node):
        if node.is_false():
            return 0.0
        if node.is_true():
            return 1.0
        return self.calculate_re(node)
```

For the clustering performance, in the Fashion dataset, the NMI and Accuracy are relatively unchanged. In contrast, the NMI and Accuracy of both IDEC-LK-A and IDEC-LK-B achieve the best values when $\lambda_e = 0.1$ for the Reuters dataset.

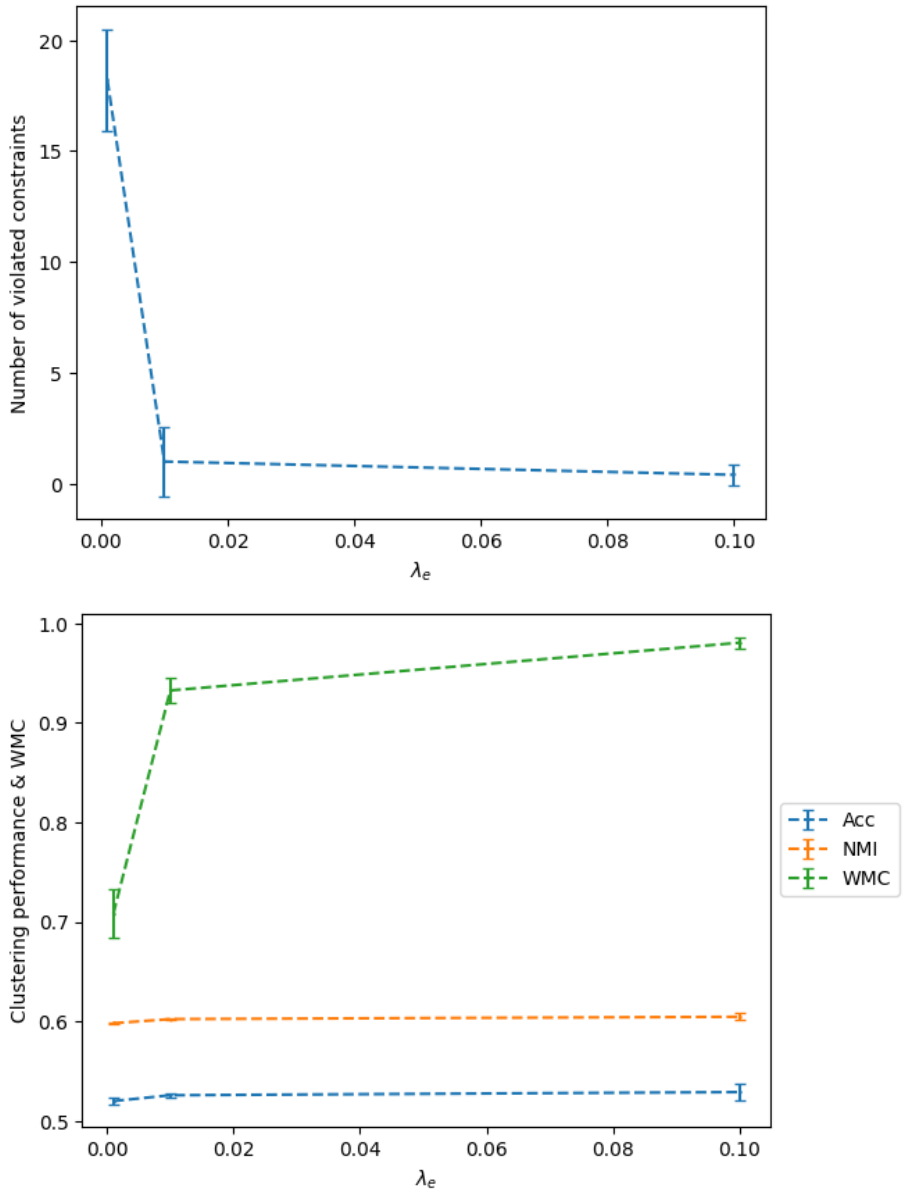

Figure A.5: Effect of λ_e on clustering performance (NMI, Acc) and constraint satisfaction (WMC, #violated constraints) with IDEC-A for Fashion dataset

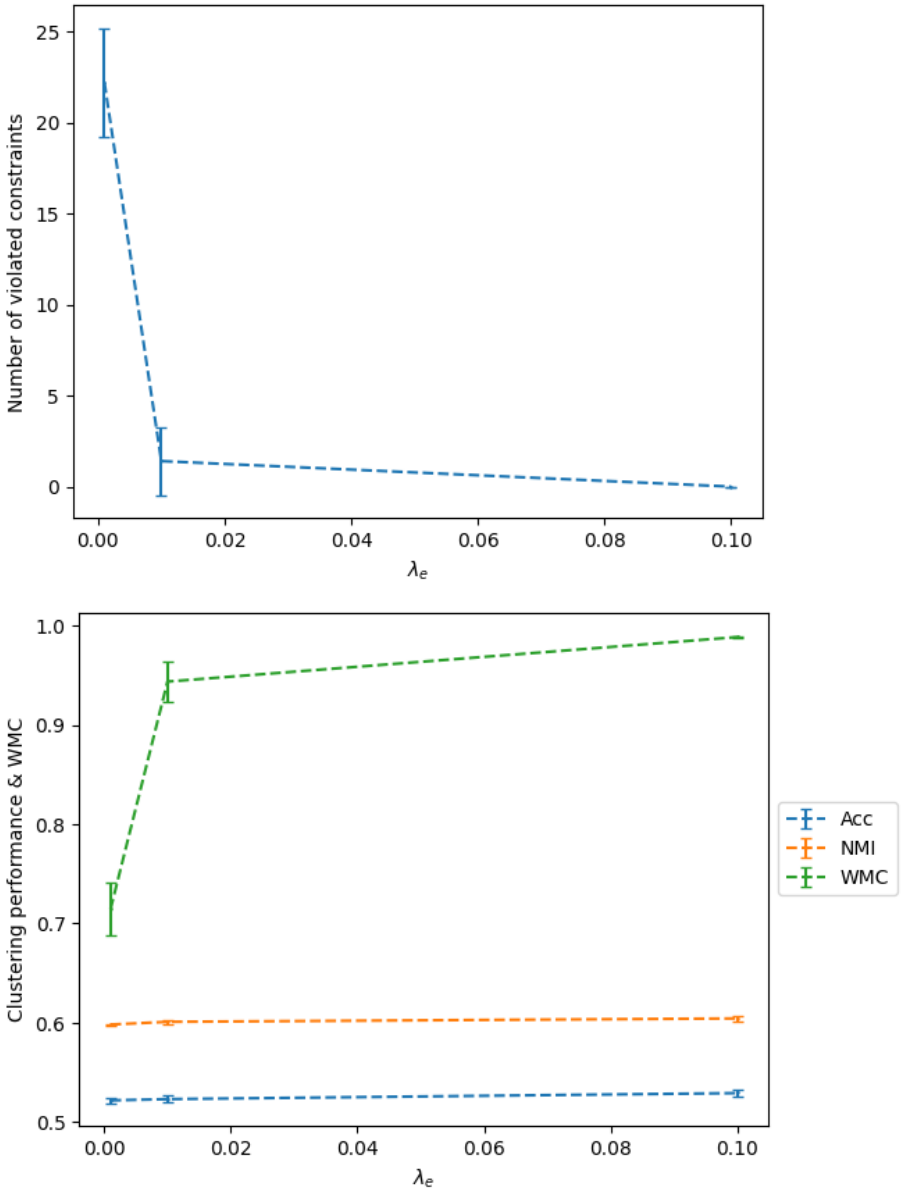

Figure A.6: Effect of λ_e on clustering performance (NMI, Acc) and constraint satisfaction (WMC, #violated constraints) with IDEC-B for Fashion dataset

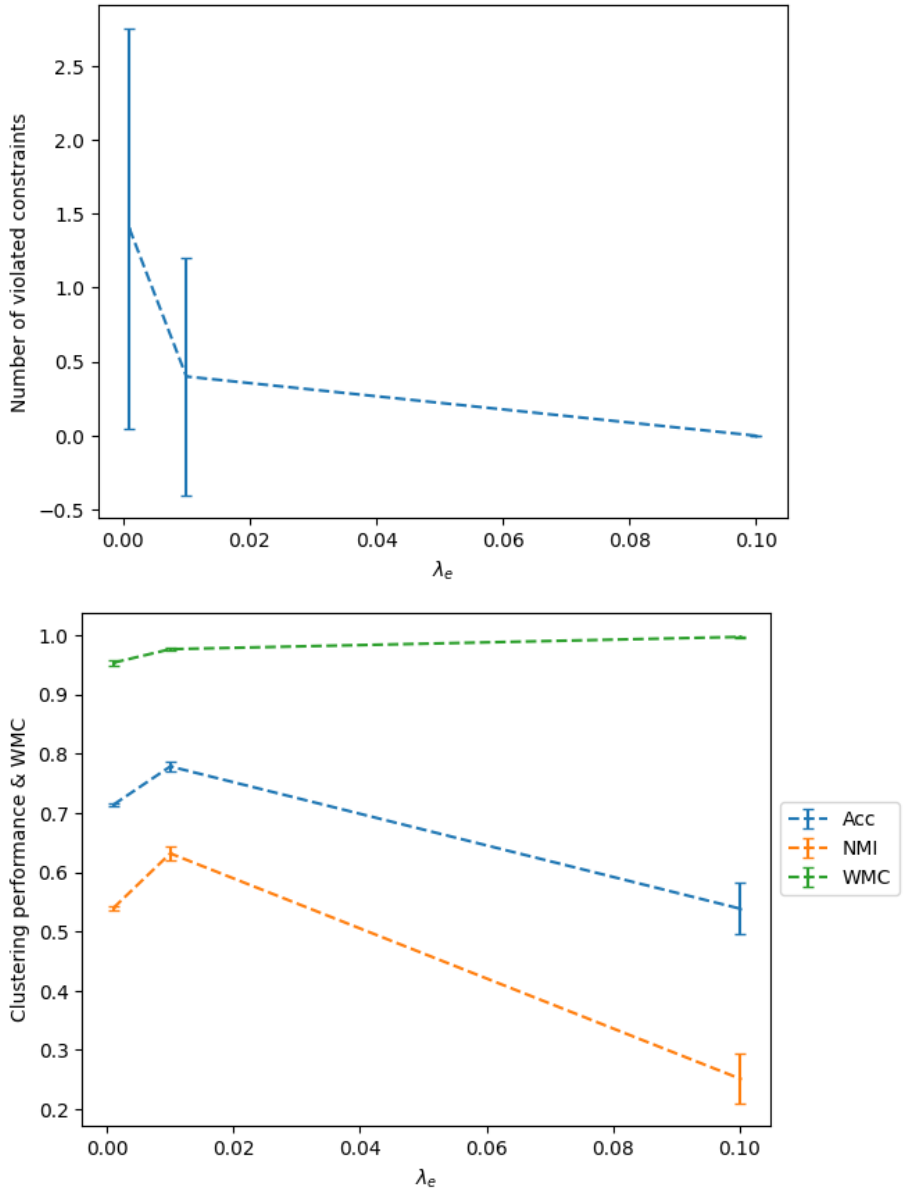

Figure A.7: Sensitivity analysis of the λ_e hyperparameter with IDEC-A for Reuters dataset

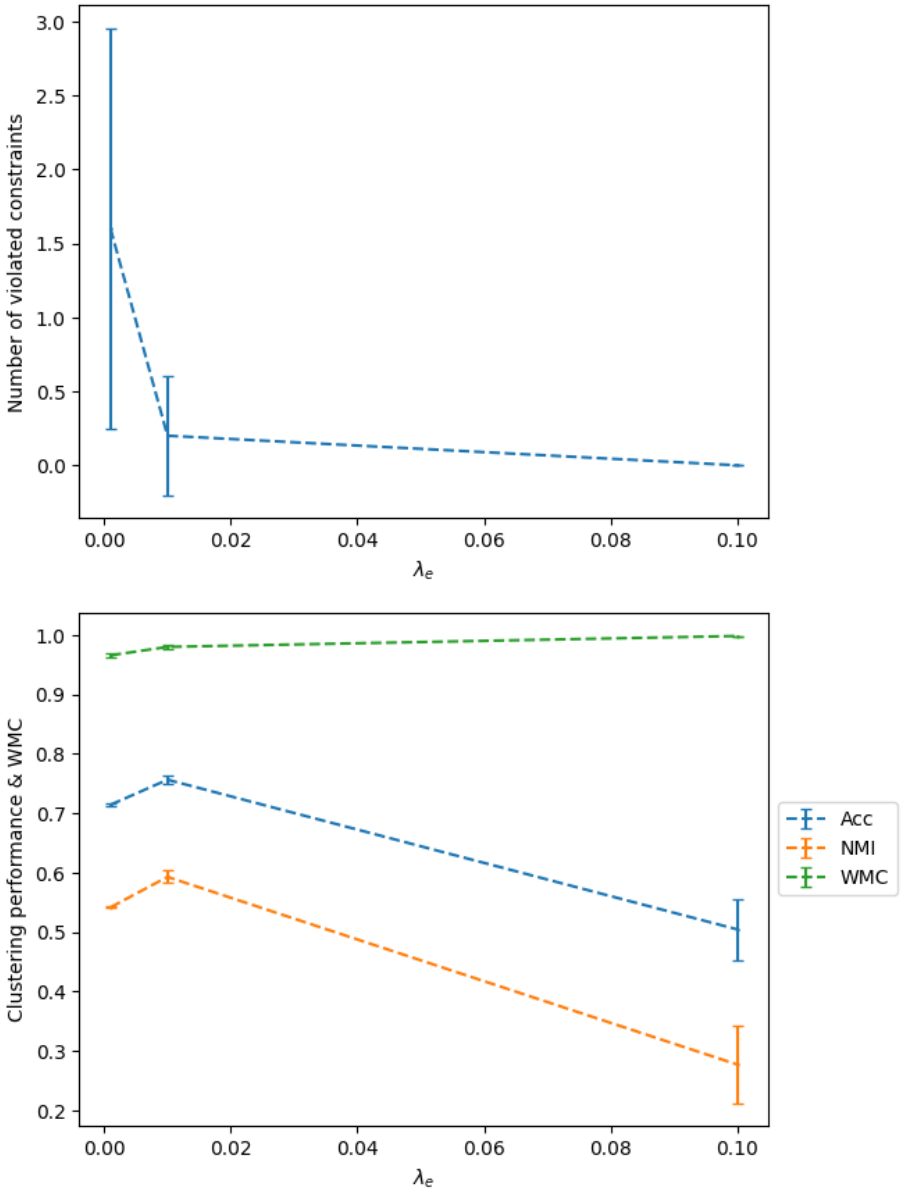

Figure A.8: Sensitivity analysis of the λ_e hyperparameter with IDEC-B for Reuters dataset

Bibliography

[1] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *ACM Sigmod record*, 28(2):49–60, 1999.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[3] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In *CPAIOR 2014*, pages 438–454, 2014.

[4] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 438–454. Springer, 2014.

[5] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *ICML 2019*, pages 405–413, 2019.

[6] Mahdieh Soleymani Baghshah and Saeed Bagheri Shouraki. Semi-supervised metric learning using pairwise constraints. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[7] Sugato Basu, Arindam Banerjee, and Raymond J Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 333–344. SIAM, 2004.

[8] Sugato Basu, A. Banjeree, ER. Mooney, Arindam Banerjee, and Raymond J. Mooney. Active semi-supervision for pairwise constrained clustering. In *SDM*, pages 333–344, 2004.

[9] Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC, 1 edition, 2008. ISBN 1584889969, 9781584889960.

[10] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[11] Suman K Bera, Deeparnab Chakrabarty, Nicolas J Flores, and Maryam Negahbani. Fair algorithms for clustering. *arXiv preprint arXiv:1901.02393*, 2019.

[12] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *ICML 2004*, pages 11–18, 2004.

[13] Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 11, 2004.

[14] Deyu Bo, Xiao Wang, Chuan Shi, Meiqi Zhu, Emiao Lu, and Peng Cui. Structural deep clustering network. In *Proceedings of The Web Conference 2020*, pages 1400–1410, 2020.

[15] P. Bradley, K. Bennett, and A. Demiriz. Constrained k-means clustering. Technical Report MSR-TR-2000-65, Microsoft Research, 2000.

[16] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, pages 132–149, 2018.

[17] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

[18] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[19] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems*, pages 5029–5037, 2017.

[20] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[21] Thi-Bich-Hanh Dao, Christel Vrain, Khanh-Chuong Duong, and Ian Davidson. A Framework for Actionable Clustering using Constraint Programming. In *ECAI 2016*, pages 453–461, 2016.

[22] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017. doi: 10.1016/j.artint.2015.05.006.

[23] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[24] Ian Davidson and S. S. Ravi. Clustering with Constraints: Feasibility Issues and the k-Means Algorithm. In *SDM 2005*, pages 138–149, 2005.

[25] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *ICDM 2010*, pages 94–105, 2010.

[26] Ian Davidson, SS Ravi, and Leonid Shamis. A sat-based framework for efficient constrained clustering. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 94–105. SIAM, 2010.

[27] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*, page 29, 2004.

[28] Shifei Ding, Hongjie Jia, Liwen Zhang, and Fengxiang Jin. Research of semi-supervised spectral clustering algorithm based on pairwise constraints. *Neural Computing and Applications*, 24(1):211–219, 2014.

[29] David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.

[30] Olivier Du Merle, Pierre Hansen, Brigitte Jaumard, and Nenad Mladenovic. An interior point algorithm for minimum sum-of-squares clustering. *SIAM Journal on Scientific Computing*, 21(4):1485–1505, 1999.

[31] Khanh-Chuong Duong, Christel Vrain, et al. A declarative framework for constrained clustering. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 419–434. Springer, 2013.

[32] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard S. Zemel. Fairness through awareness. In *Innovations in Theoretical Computer Science 2012*, pages 214–226, 2012.

[33] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[34] Sharon Fogel, Hadar Averbuch-Elor, Daniel Cohen-Or, and Jacob Goldberger. Clustering-driven deep embedding with pairwise constraints. *IEEE computer graphics and applications*, 39(4):16–27, 2019.

[35] R. Ge, M. Ester, W. Jin, and I. Davidson. Constraint-driven clustering. In *KDD 2007*, pages 320–329, 2007.

[36] Aude Genevay, Gabriel Dulac-Arnold, and Jean-Philippe Vert. Differentiable deep clustering with cluster size constraints. *arXiv preprint arXiv:1910.09036*, 2019.

[37] Rohan Ghosh and Anupam K Gupta. Scale steerable filters for locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1906.03861*, 2019.

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[39] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[40] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI 2017*, pages 1753–1759, 2017.

[41] John L Hodges. The significance probability of the smirnov two-sample test. *Arkiv för Matematik*, 3(5):469–486, 1958.

[42] Yen-Chang Hsu and Zsolt Kira. Neural network-based clustering using pairwise constraints. *arXiv preprint arXiv:1511.06321*, 2015.

[43] Weihua Hu, Takeru Miyato, Seiya Tokui, Eiichi Matsumoto, and Masashi Sugiyama. Learning discrete representations via information maximizing self-augmented training. In *International conference on machine learning*, pages 1558–1567. PMLR, 2017.

[44] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

[45] Dino Ienco and Gloria Bordogna. Fuzzy extensions of the dbscan clustering algorithm. *Soft Computing*, 22(5):1719–1730, 2018.

[46] Dino Ienco and Ruggero G. Pensa. Deep Triplet-Driven Semi-supervised Embedding Clustering. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11828 LNAI:220–234, 2019. ISSN 16113349. doi: 10.1007/978-3-030-33778-0_18.

[47] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[48] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[49] Leonard Kaufman and Peter Rousseeuw. *Clustering Large Data Sets*, pages 425–437. 12 1986. ISBN 9780444878779. doi: 10.1016/B978-0-444-87877-9.50039-X.

[50] Leonard Kaufmann and Peter Rousseeuw. Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 01 1987.

[51] Michael J. Kearns, Aaron Roth, and Saeed Sharifi-Malvajerdi. Average individual fairness: Algorithms, generalization and experiments. *CoRR*, abs/1905.10607, 2019. URL http://arxiv.org/abs/1905.10607.

[52] Sung Wook Kim, Iljeok Kim, Jonghwan Lee, and Seungchul Lee. Knowledge integration into deep learning in dynamical systems: an overview and taxonomy. *Journal of Mechanical Science and Technology*, pages 1–12, 2021.

[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] Achim Koberstein. Progress in the dual simplex algorithm for solving large scale lp problems: techniques for a fast and stable implementation. *Computational Optimization and Applications*, 41(2):185–204, 2008.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[56] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[57] Nimit Kumar and Krishna Kummamuru. Semisupervised clustering with metric learning using relative comparisons. *IEEE Transactions on Knowledge and Data Engineering*, 20(4):496–503, 2008.

[58] Chia-tung Kuo, S S Ravi, Christel Vrain, and Ian Davidson. A Framework for Minimal Clustering Modification via Constraint Programming. *Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017)*, pages 1389–1395, 2017.

[59] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[60] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5(Apr):361–397, 2004.

[61] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th international conference on control automation robotics & vision (ICARCV)*, pages 844–848. IEEE, 2014.

[62] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[63] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[64] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

[65] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Constrained Clustering Using SAT. In *IDA 2012*, pages 207–218, 2012.

[66] Sadaaki Miyamoto, Hodetomo Ichihashi, Katsuhiro Honda, and Hidetomo Ichihashi. *Algorithms for fuzzy clustering*. Springer, 2008.

[67] Marianne Mueller and Stefan Kramer. Integer Linear Programming Models for Constrained Clustering. In *DS 2010*, pages 159–173, 2010.

[68] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4610–4617, 2019.

[69] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[70] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[71] Efendi N Nasibov and Gözde Ulutagay. Robustness of density-based clustering methods with various neighborhood relations. *Fuzzy Sets and Systems*, 160 (24):3601–3615, 2009.

[72] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[73] Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Compiling graph substructures into sentential decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[74] Luke Oakden-Rayner, Jared Dunnmon, Gustavo Carneiro, and Christopher Ré. Hidden stratification causes clinically meaningful failures in machine learning for medical imaging. In *Proceedings of the ACM conference on health, inference, and learning*, pages 151–159, 2020.

[75] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[76] A. Ouali, S. Loudni, Y. Lebbah, P. Boizumault, A. Zimmermann, and L. Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *IJCAI 2016*, pages 647–654, 2016.

[77] Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. Deep subspace clustering with sparsity prior. In *IJCAI*, pages 1925–1931, 2016.

[78] P. K. S. Prakash and Achyutuni Sri Krishna Rao. *R Deep Learning Cookbook: Solve Complex Neural Net Problems with TensorFlow, H2O and MXNet.* Packt Publishing, 2017. ISBN 1787121089.

[79] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[80] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

[81] Sean Saito and Robby T Tan. Neural clustering: Concatenating layers for better projections. 2017.

[82] Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005.

[83] Atsushi Sato and Keiji Yamada. Generalized learning vector quantization. In *NIPS*, volume 95, pages 423–429, 1995.

[84] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[85] Luca Scrucca, Michael Fop, T. Brendan Murphy, and Adrian E. Raftery. mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal*, 8(1):289–317, 2016. URL https://doi.org/10.32614/RJ-2016-021.

[86] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[87] Marek Śmieja, Oleksandr Myronov, and Jacek Tabor. Semi-supervised discriminative clustering with graph regularization. *Knowledge-Based Systems*, 151:24–36, 2018.

[88] Abir Smiti and Zied Eloudi. Soft dbscan: Improving dbscan clustering method using fuzzy set theory. In *2013 6th International Conference on Human System Interactions (HSI)*, pages 380–385. IEEE, 2013.

[89] Nimit S Sohoni, Jared A Dunnmon, Geoffrey Angus, Albert Gu, and Christopher Ré. No subclass left behind: Fine-grained robustness in coarse-grained classification problems. *arXiv preprint arXiv:2011.12945*, 2020.

[90] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[91] Wei Tang, Yang Yang, Lanling Zeng, and Yongzhao Zhan. Optimizing MSE for clustering with balanced size constraints. *Symmetry*, 11(3), 2019. ISSN 20738994. doi: 10.3390/sym11030338.

[92] George Trigeorgis, Konstantinos Bousmalis, Stefanos Zafeiriou, and Bjoern Schuller. A deep semi-nmf model for learning hidden representations. In *International Conference on Machine Learning*, pages 1692–1700. PMLR, 2014.

[93] Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Scan: Learning to classify images without labels. In *ECCV*, pages 268–285, 2020.

[94] Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on neural networks*, 11(3):586–600, 2000.

[95] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[96] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *ICML 2000*, pages 1103–1110, 2000.

[97] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means Clustering with Background Knowledge. In *ICML 2001*, pages 577–584, 2001.

[98] Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *KDD 2010*, pages 563–572, 2010.

[99] Xiang Wang, Buyue Qian, and Ian Davidson. On constrained spectral clustering and its applications. *Data Mining and Knowledge Discovery*, 28(1): 1–30, 2014.

[100] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037, 2017.

[101] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML 2016*, pages 478–487, 2016.

[102] Yaqi Xie, Ziwei Xu, Mohan S Kankanhalli, Kuldeep S Meel, and Harold Soh. Embedding symbolic knowledge into deep networks. In *Advances in Neural Information Processing Systems*, pages 4233–4243, 2019.

[103] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, pages 5502–5511, 2018.

[104] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning*, pages 5502–5511, 2018.

[105] Rui Xu and Don Wunsch. *Clustering*. Wiley-IEEE Press, 2009. ISBN 9780470276808.

[106] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pages 3861–3870. PMLR, 2017.

[107] Jun Yu, Dapeng Tao, Jonathan Li, and Jun Cheng. Semantic preserving distance metric learning and applications. *Information Sciences*, 281:674–686, 2014.

[108] Yen-Yun Yu, Shireen Y Elhabian, and Ross T Whitaker. Clustering with pairwise relationships: A generative approach. *arXiv preprint arXiv:1805.02285*, 2018.

[109] Hongjing Zhang, Sugato Basu, and Ian Davidson. Deep constrained clustering - algorithms and advances. In *ECML 2019*, 2019. URL http://arxiv.org/abs/1901.10061.

[110] Hongjing Zhang, Tianyang Zhan, Sugato Basu, and Ian Davidson. A framework for deep constrained clustering. *Data Mining and Knowledge Discovery*, 35(2):593–620, 2021.

[111] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

Nguyen Viet Dung NGHIEM

Clustering et intégration de connaissances

Résumé :

Le clustering sous contraintes (une généralisation du clustering semi-supervisé) vise à exploiter les connaissances des experts lors de la tâche de clustering. La connaissance s'exprime souvent par un ensemble de contraintes et peut prendre des formes diverses. Dans cette thèse, nous développons deux mécanismes pour intégrer des contraintes dans la tâche de clustering. Dans la première partie, nous proposons une méthode déclarative post-traitement pour adapter la sortie d'un algorithme de clustering pour satisfaire les contraintes. L'originalité est de considérer une matrice d'allocation qui donne les scores d'attribution des points à chaque cluster et de trouver la meilleure partition satisfaisant toutes les contraintes. Dans la deuxième partie, nous proposons un cadre unifié pour intégrer les contraintes générales dans un modèle de clustering avec l'apprentissage profond. La généricité se représente en formalisant des contraintes en logique et en considérant leurs modèles. Les résultats expérimentaux sur des jeux de données connus montrent que notre approche est compétitive avec d'autres méthodes spécifiques aux contraintes tout en étant générale. De plus, nous avons défini et formulé de nouveaux types de contraintes en clustering : la contrainte de couverture de cluster limitant le nombre de clusters auxquels un groupe de points peut appartenir, et la contrainte d'équité combinée prenant en compte à la fois l'équité de groupe et l'équité individuelle.

Mots clés : clustering sous contraintes, apprentissage profond, approche déclarative

Clustering and knowledge integration

Summary :

Clustering is one of the essential topics in data mining. Although it is designed to work in a fully unsupervised way, its application in real-world data is often regulated by expert knowledge. Constrained clustering (a generalization of semi-supervised clustering) aims to exploit this knowledge during the clustering task. In this thesis, we develop two frameworks to integrate expert constraints in the clustering task. In the first work, we propose a declarative post-processing method to adapt the output of a clustering algorithm to satisfy the constraints. The originality is to consider an allocation matrix that gives the scores for attribution of points to each cluster and to find the best partition satisfying all the constraints. In the second work, we propose a unified framework to integrate general constraints in a clustering model with deep learning. The genericity is obtained by formulating the constraints in propositional logic, defining two versions of semantic loss, and computing them through Weighted Model Counting. Experimental results on well-known datasets show that our approach is competitive with other constraint-specific methods while being general. In addition, we have defined and formulated new types of constraints in clustering: the cluster coverage constraint limiting the number of clusters to which a group of points can belong and the combined fairness constraint taking into account both the group fairness and individual fairness.

Keywords : constrained clustering, deep learning, declarative programming

LABORATOIRE D'INFORMATIQUE FONDAMENTALE D'ORLEANS

LABORATOIRE D'INFORMATIQUE

FONDAMENTALE D'ORLÉANS

6 rue Léonard de Vinci

45067 Orléans

Centre-Val de Loire

www.regioncentre-valdeloire.fr