Advanced Difference of Convex functions Algorithms for some topics of Machine Learning with Big Data

Bach Tran

THÈSE

en vue de l'obtention du titre de

DOCTEUR DE L'UNIVERSITÉ DE LORRAINE

(arrêté ministériel du 7 Août 2006)

Spécialité INFORMATIQUE

présentée par

TRAN BACH

Titre de la thèse :

ALGORITHMES AVANCÉS DE DCA POUR CERTAINES CLASSES DE PROBLÈMES EN APPRENTISSAGE AUTOMATIQUE DU BIG DATA

—

ADVANCED DIFFERENCE OF CONVEX FUNCTIONS ALGORITHMS FOR SOME TOPICS OF MACHINE LEARNING WITH BIG DATA

soutenue le 26 novembre 2019

Composition du Jury :

Rapporteurs	Gilles GASSO	*Professeur, INSA de Rouen*
	Mau Nam NGUYEN	*Professeur, Université Portland State*
Examinateurs	Clarisse DHAENENS	*Professeur, Université de Lille*
	Yann GUERMEUR	*Directeur de Recherche, LORIA*
	Tao PHAM DINH	*Professeur émérite, INSA de Rouen*
Directrice de thèse	Hoai An LE THI	*Professeur, Université de Lorraine*
Invité	Hoai Minh LE	*MCF, Université de Lorraine*
	Vincent LEFIEUX	*Chef du pôle Data science - Intelligence artificielle de RTE*

THÈSE PRÉPARÉE AU SEIN DU LABORATOIRE D'INFORMATIQUE THÉORIQUE ET APPLIQUÉE (LITA) ET DU DÉPARTEMENT INFORMATIQUE & APPLICATIONS, LGIPM, UNIVERSITÉ DE LORRAINE, METZ, FRANCE

Remerciements

Cette thèse a été réalisée au sein du Laboratoire d'Informatique Théorique et Appliquée (LITA) et du département Informatique & Applications (IA), LGIPM, l'Université de Lorraine.

Tout d'abord, je souhaite exprimer ma sincère gratitude à ma directrice de thèse, Mme. LE THI Hoai An, Professeur des Universités à l'Université de Lorraine. Elle m'a offert une grande opportunité pour commencer ma carrière scientifique, tout en mettant à ma disponibilité des conditions excellentes de travail et de recherche durant ces années. Sous sa direction avec une grande patience, rigueur et enthousiasme, j'ai reçu des soutiens et encouragements pour étudier, effectuer des recherches sur l'optimisation et l'apprentissage automatique, et pour finir cette thèse. Je la remercie très sincèrement pour ses précieux conseils pour la recherche scientifique et également pour ma vie personnelle. J'ai eu beaucoup de chance d'avoir bénéficié de cette grande expérience pour devenir un bon chercheur scientifique dans le futur.

J'adresse respectueusement mes sincères remerciements à M. PHAM DINH Tao, professeur à l'INSA de Rouen pour ses conseils, et son suivi dans mes travaux de recherche. Je voudrais lui exprimer toute ma reconnaissance pour les discussions approfondies très intéressantes que nous avons eues et pour m'avoir suggéré de nouvelles voies de recherche.

Je remercie particulièrement le Dr. LE Hoai Minh et Dr. PHAN Duy Nhat pour leur aide et les discussions intéressantes que nous avons eues lors de notre collaboration.

Je remercie mes collègues du LITA et mes amis (dans l'ordre alphabétique) : Phuong Anh, Viet Anh, Aurélie Lallemand, Dinh Chien, Manh Cuong, Sara Samir, Van Ngai, Minh Tam, Xuan Thanh, Vinh Thanh, Tran Thuy, ... pour leur soutien, et leurs encouragements, ainsi que pour les agréables moments passés ensemble lors de mon séjour en France. Je remercie également Mme. Annie HETET, Secrétaire du LITA/IA, pour sa grande disponibilité et son aide très spontanée.

Je voudrais exprimer mes remerciements particuliers à ma fiancée, NGUYEN Thi Minh Phu, pour son soutien et patience. J'adresse toute mon affection à mes parents et à tous les membres de ma famille.

Enfin à tous ceux qui m'ont soutenu de près ou de loin, et à tous ceux qui m'ont incité même involontairement, à faire mieux, veuillez trouver ici le témoignage de ma

profonde gratitude.

TRAN Bach

Né le 23 novembre, 1993 (Viet Nam)

E-mail: bach.tran@univ-lorraine.fr

Adresse professionnelle: Bureau UM-AN1-040, LGIPM – Université de Lorraine, 3 rue Augustin Fresnel, BP 45112, 57073 Metz, France

Situation Actuelle

Depuis 10/2016	Doctorant au LGIPM (Laboratoire de Génie Informatique, de Production et de Maintenance - EA 3096), l'Université de Lorraine. Encadré par Prof. Hoai An Le Thi. Sujet de thèse : **Advanced Difference of Convex functions Algorithms for some topics of Machine Learning with Big Data.**

Experience Professionnelle

10/2015 – 09/2016	Ingénieur d'étude, laboratoire LITA, UFR MIM, Université de Lorraine, Metz, France.
04/2015 – 09/2015	Stagiaire au laboratoire LITA, UFR MIM, Université de Lorraine, Metz, France. Responsable de stage: Prof. Hoai An Le Thi. Mémoire: **Some unsupervised / semi-supervised machine learning technique and applications to anomaly detections**.

Diplôme et Formation

2016 - present	Doctorant en Informatique, LITA (laboratoire d'Informatique Théorique et Appliquée) - LGIPM (Laboratoire de Génie Informatique, de Production et de Maintenance) depuis 01/01/2018, Université de Lorraine, Metz, France.
2013–2015	Master en sciences et technologies de l'information et de la communication, L'institut national polytechnique de Toulouse.
2012–2013	Licence en Informatique, University of Greenwich (campus à l'Université FPT, Hanoi).
2009–2012	Higher Diploma en Génie logiciel (HDSE) FPT - Aptech, Hanoi.

Publications

Refereed international journal papers

[1] H.A. Le Thi, B. Tran. *A DCA-based approach for Joint Clustering and Dimensional Reduction by t-SNE. Submitted.*

[2] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Novel DCA Based Algorithms for Minimizing the Sum of a Nonconvex Function and Composite Functions with Applications in Machine learning. Submitted & Available on arvXiv [arXiv:1806.09620].*

[3] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for minimizing a large sum of DC functions and its application in Multi-class Logistic Regression. Submitted & Available on arvXiv [arXiv:1911.03992].*

Refereed papers in books / Refereed international conference papers

[1] B. Tran, H.A. Le Thi. *Deep Clustering with Spherical Distance in Latent Space.* In: Advanced Computational Methods for Knowledge Engineering. ICCSAMA 2019. Advances in Intelligent Systems and Computing, Springer, Cham. *Accepted.*

[2] G. Da Silva, H.M. Le, H.A. Le Thi, V. Lefieux, B. Tran. *Customer Clustering of French Transmission System Operator (RTE) Based on Their Electricity Consumption.* In: Le Thi H., Le H., Pham Dinh T. (eds) Optimization of Complex Systems: Theory, Models, Algorithms and Applications. WCGO 2019. Advances in Intelligent Systems and Computing, vol 991, pp. 893–905. Springer, Cham.

[3] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for the large-sum of non-convex functions problem. Application to group variables selection in multiclass logistic regression.* International Conference on Machine Learning ICML, pp. 3394–3403, 2017.

[4] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for Sparse Multiclass Logistic Regression.* In: Le NT., van Do T., Nguyen N., Thi H. (eds) Advanced Computational Methods for Knowledge Engineering. ICCSAMA 2017. Advances in Intelligent Systems and Computing, vol 629, pp. 1–12. Springer, Cham.

[5] X.T. Vo, B. Tran, H.A. Le Thi, T.P Dinh. *Ramp Loss Support Vector Data Description.* In: Nguyen N., Tojo S., Nguyen L., Trawiński B. (eds) Intelligent Infor-

mation and Database Systems. ACIIDS 2017. Lecture Notes in Computer Science, vol 10191, pp. 421–431. Springer, Cham.

Communications in national / International conferences

[1] B. Tran, H.A. Le Thi, H.M. Le, D.N. Phan. *Stochastic DCA for Group Feature Selection in Multiclass Classification.* The 29th European Conference on Operational Research (EURO2018), Valencia, Spain, July 09 - 11, 2018.

Contents

List of Figures

List of Tables

Abbreviations and Notations

Throughout the dissertation, we use uppercase letters to denote matrices, and lowercase letters for vectors or scalars. Vectors are also regarded as matrices with one column. Some of the abbreviations and notations used in the dissertation are summarized as follows.

DC	Difference of Convex functions
DCA	DC Algorithm

$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	set of real column vectors of size n
$\overline{\mathbb{R}}$	the set of extended real numbers, $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$
$\|\cdot\|_p$	ℓ_p-norm $(0 < p < \infty)$, $\|x\|_p = (\sum_{i=1}^n \|x_i\|^p)^{1/p}$, $x \in \mathbb{R}^n$
$\|\cdot\|$	Euclidean norm (or ℓ_2-norm), $\|x\| = (\sum_{i=1}^n \|x_i\|^2)^{1/2}$, $x \in \mathbb{R}^n$
$\|\cdot\|_\infty$	ℓ_∞-norm, $\|x\|_\infty = \max_{i=1,\ldots,n} \|x_i\|$, $x \in \mathbb{R}^n$
$\langle\cdot,\cdot\rangle$	scalar product, $\langle x, y\rangle = \sum_{i=1}^n x_i.y_i$, $x, y \in \mathbb{R}^n$
$\chi_C(\cdot)$	indicator function of a set C, $\chi_C(x) = 0$ if $x \in C$, $+\infty$ otherwise
$\text{co}\{C\}$	convex hull of a set of points C
$\text{Proj}_C(x)$	projection of a vector x onto a set C
$\text{dom}\, f$	effective domain of a function f
$\nabla f(x)$	gradient of a function f at x
$\partial f(x)$	subdifferential of a function f at x
$W_{i,:}$	the i-th row of the matrix W
$W_{:,i}$	the i-th column of the matrix W
$\circ$	the elementwise product
I_a	a-by-a identity matrix
$\mathbb{1}_{a\times b}$	a-by-b matrix of ones
$\|S\|$	cardinality of set S

Résumé

De nos jours, le Big Data est devenu essentiel et omniprésent dans tous les domaines. Par conséquence, il est nécessaire de développer des techniques innovantes et efficaces pour traiter la croissance rapide du volume des masses de données. Nous considérons les problèmes suivants dans le contexte de Big Data : la sélection de groupes de variables pour la régression logistique multi-classes, la réduction de dimension par t-SNE (« t-distributed Stochastic Neighbor Embedding » en anglais) et l'apprentissage en profondeur pour la classification non-supervisée (« Deep Clustering » en anglais). Nous développons des algorithmes DC (Difference of Convex functions) avancés pour ces problèmes, qui sont basés sur la programmation DC et DCA (DC Algorithm) – des outils puissants pour les problèmes d'optimisation non-convexes non-différentiables.

Dans la première partie, nous étudions le problème de la sélection de groupes de variables pour la régression logistique multi-classes. Nous résolvons ce problème en utilisant des DCAs avancés – Stochastic DCA et DCA-Like. Plus précisément, Stochastic DCA se spécialise dans le problème de la minimisation de la grande somme des fonctions DC, et ne nécessite qu'un sous-ensemble de fonctions DC à chaque itération. DCA-Like relaxe la condition de convexité de la deuxième composante DC en assurant la convergence. Accelerated DCA-Like intègre la technique d'accélération de Nesterov dans DCA-Like pour améliorer sa performance. Les expériences numériques sur plusieurs jeux de données benchmark de grande taille montrent l'efficacité de tous les algorithmes proposés en termes de temps d'exécution et de qualité de la solution.

La deuxième partie concerne t-SNE, une technique efficace de réduction de dimension non linéaire. t-SNE est modélisé sous forme d'un problème d'optimisation non-convexe. Motivés par le caractère novateur de DCA-Like et Accelerated DCA-Like, nous développons ces deux algorithmes pour résoudre le problème t-SNE. La supériorité de nos algorithmes, appliqués à la visualisation de données, par rapport aux méthodes existantes est illustrée via des expériences numériques réalisées sur les jeux de données de très grande taille.

La troisième partie est consacrée à la classification non-supervisée par l'apprentissage en profondeur. Dans la première application, nous proposons deux algorithmes basés sur DCA pour combiner t-SNE avec MSSC (Minimum Sum-of-Squares Clustering) par ces deux approches : « tandem analysis » et joint-clustering. La deuxième application considère le clustering en utilisant l'auto-encodeur. Nous avons proposé une extension d'une classe d'algorithmes de joint-clustering pour résoudre le

problème de mise à l'échelle de données (« scaling problem » en anglais), et appliqué pour un cas spécifique de joint-clustering avec MSSC. Les résultats numériques sur plusieurs jeux de données benchmark montre l'efficacité de notre algorithme comparé aux méthodes existantes.

Abstract

Big Data has become gradually essential and ubiquitous in all aspects nowadays. Therefore, there is an urge to develop innovative and efficient techniques to deal with the rapid growth in the volume of data. This dissertation considers the following problems in Big Data: group variable selection in multi-class logistic regression, dimension reduction by t-SNE (t-distributed Stochastic Neighbor Embedding), and deep clustering. We develop advanced DCAs (Difference of Convex functions Algorithms) for these problems, which are based on DC Programming and DCA – the powerful tools for non-smooth non-convex optimization problems.

Firstly, we consider the problem of group variable selection in multi-class logistic regression. We tackle this problem by using recently advanced DCAs – Stochastic DCA and DCA-Like. Specifically, Stochastic DCA specializes in the large sum of DC functions minimization problem, which only requires a subset of DC functions at each iteration. DCA-Like relaxes the convexity condition of the second DC component while guaranteeing the convergence. Accelerated DCA-Like incorporates the Nesterov's acceleration technique into DCA-Like to improve its performance. The numerical experiments in benchmark high-dimensional datasets show the effectiveness of proposed algorithms in terms of running time and solution quality.

The second part studies the t-SNE problem, an effective non-linear dimensional reduction technique. Motivated by the novelty of DCA-Like and Accelerated DCA-Like, we develop two algorithms for the t-SNE problem. The superiority of proposed algorithms in comparison with existing methods is illustrated through numerical experiments for visualization application.

Finally, the third part considers the problem of deep clustering. In the first application, we propose two algorithms based on DCA to combine t-SNE with MSSC (Minimum Sum-of-Squares Clustering) by following two approaches: "tandem analysis" and joint-clustering. The second application considers clustering with auto-encoder (a well-known type of neural network). We propose an extension to a class of joint-clustering algorithms to overcome the scaling problem and applied for a specific case of joint-clustering with MSSC. Numerical experiments on several real-world datasets show the effectiveness of our methods in rapidity and clustering quality, compared to the state-of-the-art methods.

Introduction générale

Cadre général et motivations

De nos jours, le Big Data est devenu essentiel et omniprésent dans tous les domaines. Dans la nouvelle ère de l'Internet où les données peuvent être facilement collectées (comme les documents, les textes et les vidéos de l'Internet, les données sensorielles obtenues à partir de machines), les données obtenues ont souvent de nombreuses variables (c'est-à-dire des données de grande dimension) qui posent de nombreuses difficultés aux techniques classiques d'apprentissage automatique. L'une des façons les plus naturelles et les plus populaires de traiter les données de grande dimension est d'utiliser des méthodes de réduction dimensionnelle. Dans cette thèse, nous sommes intéressés par le développement de techniques innovantes et efficaces, bien adaptées au Big Data en utilisant des techniques de réduction dimensionnelle : sélection de groupes de variables et réduction non linéaire de la dimension.

La sélection des variables est une approche simple mais efficace, qui a été bien étudiée dans la littérature. Cependant, les travaux existants montrent qu'il y a un chevauchement dans les variables sélectionnées. De plus, dans de nombreuses applications, nous pouvons avoir des connaissances préalables sur des groupes de variables, il est donc scientifiquement significatif d'incorporer la sélection des variables de groupe dans les modèles. La sélection de groupes de variables basée sur la norme zéro-mixte ("mixed-norm"-$\ell_{q,0}$ en anglais) est une façon naturelle de modéliser ce problème. Cependant, cette dernière n'est pas souvent utilisée en raison de la non-convexité de la ℓ_0-norme. Dans ce travail, nous considérons une application de la $\ell_{q,0}$-norme pour les données de grande dimension en apprentissage automatique. La difficulté mentionnée est surmontée en utilisant une fonction DC pour approximer la ℓ_0-norme, et le problème résultant est aussi un problème d'optimisation DC. Une autre façon préférée de représenter des données de grande dimension est d'utiliser des méthodes de réduction non linéaire de la dimension. Cependant, la plupart des problèmes d'optimisation issus de cette direction sont non convexes. Motivés par le succès de la programmation DC et DCA pour plusieurs problèmes de réduction non linéaires de la dimension, tels que la carte auto-organisatrice ("Self-Organizing Map" en anglais) et le positionnement multidimensionnel ("Multidimensional Scaling" en anglais), nous choisissons volontairement la programmation DC et DCA comme la base de méthodologie dans cette thèse.

Sur le plan algorithmique, la thèse a proposé une approche unifiée, fondée sur la nouvelle génération de la programmation DC et DCA. La programmation DC et DCA (voir [62]) sonts des outils puissants d'optimisation non convexe qui connaissent un grand succès, au cours de trois dernières décennies, dans la modélisation et la résolution de nombreux problèmes d'application dans divers domaines de sciences appliquées, en particulier en apprentissage automatique et fouille de données (MLDM – "Machine Learning and Data Mining" en anglais) (voir par example [18, 42, 45, 47, 48, 49, 55, 56, 57, 58, 59, 60, 63, 64, 66, 67, 71, 84, 85, 86, 87, 125, 111, 50, 52] et la liste des références dans [4]). Récemment, la croissance rapide du Big Data a provoqué plusieurs nouveaux challenges d'optimisation, d'où une nouvelle génération de DCA voit le jour pour leur confronter d'une façon efficace. Cette thèse est dédiée au développement de nouveaux algorithmes basés sur cette nouvelle génération de DCA. De nombreuses expérimentations numériques sur différents types de données dans divers domaines réalisées dans cette thèse ont prouvé l'efficacité, la scalabilité, la rapidité des algorithmes proposés et leur supériorité par rapport aux méthodes standards.

La programmation DC et DCA considèrent le problème DC de la forme

$$\alpha = \inf\{f(x) = g(x) - h(x) : x \in \mathbb{R}^n\} \quad (P_{dc})$$

òu g et h sont des fonctions convexes définies sur $\mathbb{R}^n$ et à valeurs dans $\mathbb{R} \cup \{+\infty\}$, semi-continues inférieurement et propres. La fonction f est appelée fonction DC avec les composantes DC g et h, et $g - h$ est une décomposition DC de f. DCA est basé sur la dualité DC et des conditions d'optimalité locale. La construction de DCA implique les composantes DC g et h et non la fonction DC f elle-même. Chaque fonction DC admet une infinité des décompositions DC qui influencent considérablement sur la qualité (la rapidité, l'efficacité, la globalité de la solution obtenue ...) de DCA. Ainsi, au point de vue algorithmique, la recherche d'une "bonne" décomposition DC et d'un "bon" point initial est très importante dans le développement de DCA pour la résolution d'un programme DC.

L'utilisation de la programmation DC et DCA dans cette thèse est justifiée par de multiple arguments [97]:

- La programmation DC et DCA fournissent un cadre très riche pour les problèmes MLDM: MLDM constituent une mine des programmes DC dont la résolution appropriée devrait recourir à la programmation DC et DCA. En effet la liste indicative (non exhaustive) des références dans [4] témoigne de la vitalité, la puissance et la percée de cette approche dans la communauté de MLDM.
- DCA est une philosophie plutôt qu'un algorithme. Pour chaque problème, nous pouvons concevoir une famille d'algorithmes basés sur DCA. La flexibilité de DCA sur le choix des décomposition DC peut offrir des schémas DCA plus performants que des méthodes standards.
- L'analyse convexe fournit des outils puissants pour prouver la convergence de DCA dans un cadre général. Ainsi tous les algorithmes basés sur DCA bénéficient (au moins) des propriétés de convergence générales du schéma DCA générique qui ont été démontrées.

— DCA est une méthode efficace, rapide et scalable pour la programmation non convexe. A notre connaissance, DCA est l'un des rares algorithmes de la programmation non convexe, non différentiable qui peut résoudre des programmes DC de très grande dimension.

Il est important de noter qu'avec les techniques de reformulation en programmation DC et les décompositions DC appropriées, on peut retrouver la plupart des algorithmes existants en programmation convexe/non convexe comme cas particuliers de DCA. En particulier, pour la communauté d'apprentissage automatique et fouille de données, les méthodes très connus comme Expectation–Maximisation (EM) [23], Succesive Linear Approximation (SLA) [10], ConCave–Convex Procedure (CCCP) [126], Iterative Shrinkage–Thresholding Algorithms (ISTA) [15] sont des versions spéciales de DCA.

Nos contributions

L'objectif principal de cette thèse est de développer de nouveaux modèles et méthodes d'apprentissage automatique dans le Big Data, en particulier pour les trois classes de problèmes suivantes : la sélection de groupes de variables pour la régression logistique multi-classes, la réduction de dimension par t-SNE, et l'apprentissage en profondeur pour la classification non-supervisée. Les principales réalisations de cette thèse seront décrites en détail ci-dessous.

La premier partie concerne le problème de la sélection de groupes de variables en régression logistique multi-classe. Dans la littérature, la norme $\ell_{q,0}$ a été développée pour formuler le problème de la sélection de groupes de variables. Cependant, le problème qui en résulte est difficile à cause de la non-convexité du $\ell_{q,0}$. Nous étudions le DCA pour ce problème par deux approches. La première approche considère le Stochastic DCA. Ce dernier est spécifiquement développé pour le problème *"large sum of DC functions"*. L'avantage principal de cette méthode par rapport à la méthode DCA standard est qu'elle ne nécessite que le calcul du sous-gradient d'un sous-ensemble de la deuxième composante DC. Dans la deuxième approche, nous avons considéré DCA-Like et Accelerated DCA-Like. Contrairement à DCA standard, le DCA-Like relaxe la condition de convexité de la deuxième composante DC tout en assurant la convergence. DCA-Like accéléré intègre la technique d'accélération de Nesterov dans DCA-Like pour améliorer sa performance tout en bénéficiant de propriétés de convergence similaires à celles de DCA-Like. Les expériences numériques sur les jeux de données de très grande taille (synthétiques et réelles) montrent que nos méthodes sont efficaces en terms de qualité et de temps de calcul.

La deuxième partie étudie le problème de la réduction dimensionnelle par t-SNE. Cette dernière est une méthode populaire, en particulier pour la visualisation. Elle a été utilisée dans de nombreuses applications telles que la bioinformatique, la recherche sur le cancer, la visualisation de "features" dans les réseaux neuronaux, etc. Cependant, il s'agit d'un problème difficile d'optimisation non convexe. Nous étudions le DCA pour ce problème et nous proposons deux algorithmes efficaces basés sur deux DCA avancés – DCA-like et Accelerated DCA-Like. Nous montrons également que la minimisation de la majorisation [123], le meilleur algorithme pour t-SNE, n'est rien d'autre que DCA-Like appliqué au t-SNE. Nous effectuons soigneusement les expériences numériques et

fournissons une comparaison des algorithmes proposés sur plusieurs jeux de données benchmarks pour l'application de visualisation.

La troisième partie concerne l'apprentissage en profondeur ("Deep Clustering" en anglais) pour la classification non-supervisée (clustering) pour les données de grande dimension. Dans la littérature, les problèmes "deep clustering" sont souvent abordés en utilisant des techniques de réduction dimensionnelle, qui ont donné les meilleurs résultats dans de nombreuses applications de clustering, en particulier pour les données complexes de grande dimension comme les images et les documents. Nous considérons deux directions basées sur deux méthodes de réduction de la dimension – t-SNE et auto-encodeur. Dans la première direction, le t-SNE est considéré par deux approches : *"tandem analysis"* et joint-clustering. Nous avons développé deux algorithmes basés sur la programmation DC et DCA. Le premier algorithme, appelé `MSSC-2S`, suit l'approche de *"tandem analysis"*, qui utilise t-SNE pour la réduction de la dimensionnalité avant le clustering. Puisque chaque étape du `MSSC-2S` est effectuée par un DCA bien adapté aux jeux de données de grande dimension, le `MSSC-2S` est applicable aux contextes de grande dimension. La deuxième direction considère le problème de l'optimisation non-convexe pour les tâches de clustering et de réduction de la dimension. Nous développons une méthode efficace, `MSSC-JDR`, par la programmation DC et DCA. Nous effectuons des expériences extensives pour des jeux de données de grande dimension et à grande dimension avec plusieurs méthodes récentes pour démontrer l'efficacité de nos algorithmes. Dans la deuxième direction, nous considérons l'auto-encodeur sur l'approche de joint-clustering. Nous montrons d'abord qu'il existe une classe d'algorithmes qui sont affectés par le problème d'échelle dans l'espace latent. Ensuite, nous proposons des extensions en utilisant deux fonctions de distance d'invariance d'échelle. En tant qu'application, les extensions proposées sont appliquées au problème de joint-clustering avec MSSC (Minimum Sum-of-Square Clustering). Les expérimentations numériques sont effectuées sur des ensembles de données de grande dimension démontrent la qualité des extensions proposées par rapport aux méthodes existantes.

Organisation de la Thèse

La thèse est composée de cinq chapitres. Le premier chapitre décrit brièvement les concepts fondamentaux et les principaux résultats de l'analyse convexe, la programmation DC et DCA, et ses versions avancées, ce qui fournit la base théorique et algorithmique pour les autres chapitres. Chacun des trois chapitres suivants est consacré à la présentation d'une classe de problèmes abordée ci-dessus. Le chapitre 2 concerne le problème de la sélection de groupes de variables pour la régression logistique multi-classe. Ensuite, le chapitre 3 considère au problème du t-SNE. Dans le chapitre 4, nous abordons les problèmes clustering en d'apprentissage en profondeur. La conclusion et les perspectives de nos travaux sont données au chapitre 5.

Chapter 1

Methodology

This chapter summarizes some basic concepts and results that will be the groundwork of the dissertation.

1.1 DC programming and DCA

DC programming and DCA, which constitute the backbone of nonconvex programming and global optimization, were introduced by Pham Dinh Tao in their preliminary form in 1985 [88]. Important developments and improvements on both theoretical and computational aspects have been completed since 1994 throughout the joint works of Le Thi Hoai An and Pham Dinh Tao. The readers are referred to the seminal paper [62] for the thirty years of developments of DCA.

In this section, we present some basic properties of convex analysis and DC optimization and DC Algorithm that computational methods of this thesis are based on. The materials of this section are extracted from [44, 85, 60].

1.1.1 Fundamental convex analysis

First, let us recall briefly some notions and results in convex analysis related to the dissertation (refer to the references [9, 85, 93] for more details). Let denote X the Euclidean space $\mathbb{R}^n$.

A subset C of X is said to be *convex* if $(1-\lambda)x + \lambda y \in C$ whenever x, $y \in C$ and $\lambda \in [0,1]$.

Let f be a function whose values are in $\overline{\mathbb{R}}$ and whose domain is a subset S of X. The set

$$\{(x,t) : x \in S,\ t \in \mathbb{R}, f(x) \leq t\}$$

is called the *epigraph* of f and is denoted by epi$f$.

We define f to be a *convex function* on S if epi$f$ is convex set in $X \times \mathbb{R}$. This is equivalent to that S is convex and

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y), \quad \forall x, y \in S, \ \forall \lambda \in [0,1].$$

The function f is *strictly convex* if the inequality above holds strictly whenever x and y are distinct in S and $0 < \lambda < 1$.

The *effective domain* of a convex function f on S, denoted by dom$f$, is the projection on X of the epigraph of f

$$\text{dom}f = \{x \in X : \exists t \in \mathbb{R}, (x,t) \in \text{epi}f\} = \{x \in X : f(x) < +\infty\}$$

and obviously, it is convex.

The convex function f is called *proper* if dom$f \neq \emptyset$ and $f(x) > -\infty$ for all $x \in S$.

The function f is said to be *lower semi-continuous* at a point x of S if

$$f(x) \leq \liminf_{y \to x} f(y).$$

Denote by $\Gamma_0(X)$ the set of all proper lower semi-continuous convex functions on X.

Let ρ be a nonnegative number and C be a convex subset of X. One says that a function $\theta : C \to \mathbb{R} \cup \{+\infty\}$ is ρ–convex if

$$\theta[\lambda x + (1-\lambda)y] \leq \lambda\theta(x) + (1-\lambda)\theta(y) - \frac{\lambda(1-\lambda)}{2}\rho\|x-y\|^2$$

for all $x, y \in C$ and $\lambda \in (0,1)$. It amounts to say that $\theta - (\rho/2)\|\cdot\|^2$ is convex on C. The modulus of strong convexity of θ on C, denoted by $\rho(\theta, C)$ or $\rho(\theta)$ if $C = X$, is given by

$$\rho(\theta, C) = \sup\{\rho \geq 0 : \theta - (\rho/2)\|\cdot\|^2 \text{ is convex on } C\}.$$

One says that θ is *strongly convex* on C if $\rho(\theta, C) > 0$.

A vector y is said to be a *subgradient* of a convex function f at a point x^0 if

$$f(x) \geq f(x^0) + \langle x - x^0, y\rangle, \qquad \forall x \in X.$$

The set of all subgradients of f at x^0 is called the *subdifferential* of f at x^0 and is denoted by $\partial f(x^0)$. If $\partial f(x)$ is not empty, f is said to be *subdifferentiable* at x.

For $\varepsilon > 0$, a vector y is said to be an ε*–subgradient* of a convex function f at a point x^0 if

$$f(x) \geq (f(x^0) - \varepsilon) + \langle x - x^0, y\rangle, \qquad \forall x \in X.$$

The set of all ε–subgradients of f at x^0 is called the ε*–subdifferential* of f at x^0 and is denoted by $\partial_\varepsilon f(x^0)$.

For $\epsilon \geq 0$, a point x_ϵ is called an ϵ-solution of the problem $\inf\{f(x) : x \in \mathbb{R}^d\}$ if

$$f(x_\epsilon) \leq f(x) + \epsilon \ \forall x \in \mathbb{R}^d.$$

Let us describe two basic notations as follows.

$$\text{dom } \partial f = \{x \in X : \partial f(x) \neq \emptyset\} \ \text{ and } \ \text{range } \partial f(x) = \cup\{\partial f(x) : x \in \text{dom } \partial f\}.$$

Proposition 1.1. *Let f be a proper convex function. Then*

1. *$\partial_\varepsilon f(x)$ is a closed convex set, for any $x \in X$ and $\varepsilon \geq 0$.*
2. $\text{ri}(\text{dom} f) \subset \text{dom } \partial f \subset \text{dom} f$
   where $\text{ri}(\text{dom} f)$ *stands for the* relative interior *of* $\text{dom} f$.
3. *If f has a unique subgradient at x, then f is differentiable at x, and $\partial f(x) = \{\nabla f(x)\}$.*
4. *$x_0 \in \text{argmin}\{f(x) : x \in X\}$ if and only if $0 \in \partial f(x_0)$.*

Conjugates of convex functions

The *conjugate* of a function $f : X \to \overline{\mathbb{R}}$ is the function $f^* : X \to \overline{\mathbb{R}}$, defined by

$$f^*(y) = \sup_{x \in X}\{\langle x, y\rangle - f(x)\}.$$

Proposition 1.2. *Let $f \in \Gamma_0(X)$. Then we have*

1. *$f^* \in \Gamma_0(X)$ and $f^{**} = f$.*
2. *$f(x) + f^*(y) \geq \langle x, y\rangle$, for any $x, y \in X$.*
   Equality holds if and only if $y \in \partial f(x) \Leftrightarrow x \in \partial f^*(y)$.
3. *$y \in \partial_\varepsilon f(x) \Longleftrightarrow x \in \partial_\varepsilon f^*(y) \Longleftrightarrow f(x) + f^*(y) \leq \langle x, y\rangle + \varepsilon$, for all $\varepsilon > 0$.*

1.1.2 DC optimization

DC program

In the sequel, we use the convention $+\infty - (+\infty) = +\infty$.

For $g, h \in \Gamma_0(X)$, a standard *DC program* is of the form

$$(P) \qquad \alpha = \inf\{f(x) = g(x) - h(x) : x \in X\}$$

and its dual counterpart

$$(D) \qquad \alpha^* = \inf\{h^*(y) - g^*(y) : y \in X\}.$$

There is a perfect symmetry between primal and dual programs (P) and (D): the dual program to (D) is exactly (P), moreover, $\alpha = \alpha^*$.

Remark 1.1. *Let C be a nonempty closed convex set. Then, the constrained problem*

$$\inf\{f(x) = g(x) - h(x) : x \in C\}$$

can be transformed into an unconstrained DC program by using the indicator function χ_C, i.e.,

$$\inf\{f(x) = \phi(x) - h(x) : x \in X\}$$

where $\phi := g + \chi_C$ belongs to $\Gamma_0(X)$.

We will always keep the following assumption that is deduced from the finiteness of α

$$\operatorname{dom} g \subset \operatorname{dom} h \quad \text{and} \quad \operatorname{dom} h^* \subset \operatorname{dom} g^*. \tag{1.1}$$

Optimality conditions for DC optimization

A point x^* is said to be a *local minimizer* of $g - h$ if $x^* \in \operatorname{dom} g \cap \operatorname{dom} h$ (so, $(g - h)(x^*)$ is finite) and there is a neighborhood U of x^* such that

$$g(x) - h(x) \geq g(x^*) - h(x^*), \qquad \forall x \in U. \tag{1.2}$$

A point x^* is said to be a *critical point* of $g-h$ if it verifies the generalized Kuhn–Tucker condition

$$\partial g(x^*) \cap \partial h(x^*) \neq \emptyset. \tag{1.3}$$

Let $\mathcal{P}$ and $\mathcal{D}$ denote the solution sets of problems (P) and (D) respectively, and let

$$\mathcal{P}_\ell = \{x^* \in X : \partial h(x^*) \subset \partial g(x^*)\}, \quad \mathcal{D}_\ell = \{y^* \in X : \partial g^*(y^*) \subset \partial h^*(y^*)\}.$$

In the following, we present some fundamental results on DC programming [85].

Theorem 1.1. **i)** *Global optimality condition: $x \in \mathcal{P}$ if and only if*

$$\partial_\varepsilon h(x) \subset \partial_\varepsilon g(x), \ \forall \varepsilon > 0.$$

ii) *Transportation of global minimizers: $\cup\{\partial h(x) : x \in \mathcal{P}\} \subset \mathcal{D} \subset \operatorname{dom} h^*$. The first inclusion becomes equality if g^* is subdifferentiable in $\mathcal{D}$. In this case, $\mathcal{D} \subset (\operatorname{dom} \partial g^* \cap \operatorname{dom} \partial h^*)$.*

iii) *Necessary local optimality: if x^* is a local minimizer of $g - h$, then $x^* \in \mathcal{P}_\ell$.*

iv) *Sufficient local optimality: Let x^* be a critical point of $g - h$ and $y^* \in \partial g(x^*) \cap \partial h(x^*)$. Let U be a neighborhood of x^* such that $(U \cap \operatorname{dom} g) \subset \operatorname{dom} \partial h$. If for any $x \in U \cap \operatorname{dom} g$, there is $y \in \partial h(x)$ such that $h^*(y) - g^*(y) \geq h^*(y^*) - g^*(y^*)$, then x^* is a local minimizer of $g - h$. More precisely,*

$$g(x) - h(x) \geq g(x^*) - h(x^*), \qquad \forall x \in U \cap \operatorname{dom} g.$$

v) *Transportation of local minimizers: Let* $x^* \in \operatorname{dom}\partial h$ *be a local minimizer of* $g - h$. *Let* $y^* \in \partial h(x^*)$ *and a neighborhood* U *of* x^* *such that* $g(x) - h(x) \geq g(x^*) - h(x^*),\ \forall x \in U \cap \operatorname{dom} g$. *If*

$$y^* \in \operatorname{int}(\operatorname{dom} g^*) \quad \text{and} \quad \partial g^*(y^*) \subset U$$

then y^* *is a local minimizer of* $h^* - g^*$.

Remark 1.2. *a) By the symmetry of the DC duality, these results have their corresponding dual part. For example, if* y *is a local minimizer of* $h^* - g^*$, *then* $y \in \mathcal{D}_\ell$.

b) The properties ii), v) and their dual parts indicate that there is no gap between the problems (P) *and* (D). *They show that globally/locally solving the primal problem* (P) *implies globally/locally solving the dual problem* (D) *and vice–versa. Thus, it is useful if one of them is easier to solve than the other.*

c) The necessary local optimality condition $\partial h^*(x^*) \subset \partial g^*(x^*)$ *is also sufficient for many important classes programs, for example [60], if* h *is polyhedral convex, or when* f *is locally convex at* x^*, *i.e. there exists a convex neighborhood* U *of* x^* *such that* f *is finite and convex on* U. *We know that a polyhedral convex function is almost everywhere differentiable, that is to say, it is differentiable everywhere except on a set of measure zero. Thus, if* h *is a polyhedral convex function, then a critical point of* $g - h$ *is almost always a local solution to* (P).

d) If f *is actually convex on* X, *we call* (P) *a* "false" DC program. *In addition, if* $\operatorname{ri}(\operatorname{dom} g) \cap \operatorname{ri}(\operatorname{dom} h) \neq \emptyset$ *and* $x^0 \in \operatorname{dom} g$ *such that* g *is continuous at* x^0, *then* $0 \in \partial f(x^0) \Leftrightarrow \partial h(x^0) \subset \partial g(x^0)$ *[60]. Thus, in this case, the local optimality is also sufficient for the global optimality. Consequently, if in addition* h *is differentiable, a critical point is also a global solution.*

1.1.3 DC Algorithm (DCA)

The DCA consists in the construction of the two sequences $\{x^k\}$ and $\{y^k\}$ (candidates for being primal and dual solutions, respectively) which are easy to calculate and satisfy the following properties:

i) The sequences $(g - h)(x^k)$ and $(h^* - g^*)(y^k)$ are decreasing.

ii) Their corresponding limits x^∞ and y^∞ either satisfy the local optimality condition $(x^\infty, y^\infty) \in \mathcal{P}_\ell \times \mathcal{D}_\ell$ or are critical points of $g - h$ and $h^* - g^*$, respectively.

From a given initial point $x^0 \in \operatorname{dom} g$, the DCA generates these sequences by the scheme

$$y^k \in \partial h(x^k) = \arg\min\{h^*(y) - \langle y, x^k\rangle : y \in X\}, \tag{1.4a}$$

$$x^{k+1} \in \partial g^*(y^k) = \arg\min\{g(x) - \langle x, y^k\rangle : x \in X\}. \tag{1.4b}$$

The interpretation of the above scheme is simple. At iteration k of DCA, one replaces the second component h in the primal DC program by its affine minorant

$$h_k(x) = h(x^k) + \langle x - x^k, y^k\rangle, \tag{1.5}$$

where $y^k \in \partial h(x^k)$. Then the original DC program is reduced to the *convex program*

$$(P_k) \qquad \alpha_k = \inf\{f_k(x) := g(x) - h_k(x) : x \in X\}$$

that is equivalent to (1.4b). It is easy to see that f_k is a majorant of f which is exact at x^k i.e. $f_k(x^k) = f(x^k)$. Similarly, by replacing g^* with its affine minorant

$$g_k^*(y) = g^*(y^{k-1}) + \langle y - y^{k-1}, x^k \rangle \tag{1.6}$$

where $x^k \in \partial g^*(y^{k-1})$, it leads to the convex program

$$(D_k) \qquad \inf\{h^*(y) - g_k^*(y) : y \in X\}$$

whose solution set is $\partial h(x^k)$.

Well definiteness of DCA

DCA is well defined if one can construct two sequences $\{x^k\}$ and $\{y^k\}$ as described above from an arbitrary initial point. The following lemma is the necessary and sufficient condition for this property.

Lemma 1.1 ([85]). *The sequences $\{x^k\}$ and $\{y^k\}$ in DCA are well defined if and only if*

$$\operatorname{dom}\partial g \subset \operatorname{dom}\partial h \quad \text{and} \quad \operatorname{dom}\partial h^* \subset \operatorname{dom}\partial g^*.$$

Since for $\varphi \in \Gamma_0(X)$ one has $\operatorname{ri}(\operatorname{dom}\varphi) \subset \operatorname{dom}\partial\varphi \subset \operatorname{dom}\varphi$ (Proposition 1.1). Moreover, under the assumptions $\operatorname{dom} g \subset \operatorname{dom} h$, $\operatorname{dom} h^* \subset \operatorname{dom} g^*$, one can say that DCA in general is well defined.

Convergence properties of DCA

Complete convergence of DCA is given in the following results [85].

Theorem 1.2. *Suppose that the sequences $\{x^k\}$ and $\{y^k\}$ are generated by DCA. Then we have*

i) *The sequences $\{g(x^k) - h(x^k)\}$ and $\{h^*(y^k) - g^*(y^k)\}$ are decreasing and*

- *$g(x^{k+1}) - h(x^{k+1}) = g(x^k) - h(x^k)$ if and only if $\{x^k, x^{k+1}\} \subset \partial g^*(y^k) \cap \partial h^*(y^k)$ and $[\rho(h) + \rho(g)]\|x^{k+1} - x^k\| = 0$.*
- *$h^*(y^{k+1}) - g^*(y^{k+1}) = h^*(y^k) - g^*(y^k)$ if and only if $\{y^k, y^{k+1}\} \subset \partial g(x^k) \cap \partial h(x^k)$ and $[\rho(h^*) + \rho(g^*)]\|y^{k+1} - y^k\| = 0$.*

DCA terminates at the kth iteration if either of the above equalities holds.

ii) *If $\rho(h) + \rho(g) > 0$ (resp. $\rho(h^*) + \rho(g^*) > 0$), then the sequence $\{\|x^{k+1} - x^k\|^2\}$ (resp. $\{\|y^{k+1} - y^k\|^2\}$) converges.*

iii) *If the optimal value α is finite and the sequences $\{x^k\}$ and $\{y^k\}$ are bounded, then every limit point x^∞ (resp. y^∞) of the sequence $\{x^k\}$ (resp. $\{y^k\}$) is a critical point of $g - h$ (resp. $h^* - g^*$).*

iv) *DCA has a linear convergence for general DC program.*

v) *In polyhedral DC programs, the sequences $\{x^k\}$ and $\{y^k\}$ contain finitely many elements and DCA has a finite convergence.*

vi) *If DCA converges to a point x^* that admits a convex neighborhood in which the objective function f is finite and convex (i.e. the function f is locally convex at x^*) and if the second DC component h is differentiable at x^*, then x^* is a local minimizer to the problem (P).*

Remark 1.3. *a) Finding y^k, x^{k+1} based on the scheme 1.4 amounts to solving the problems (D_k) and (P_k). Thus, DCA works by reducing a DC program to a sequence of convex programs which can be solved efficiently.*

b) In practice, the calculation of the subgradient of the function h at a point x is usually easy if we know its explicit expression. But, the explicit expression of the conjugate of a given function g is unknown, so calculating x^{k+1} is done by solving the convex problem (P_k). For the large-scale setting, the solutions to the problem (P_k) should be either in an explicit form or achieved by efficient algorithms with inexpensive computations.

c) DCA's distinctive feature relies upon the fact that DCA deals with the convex DC components g and h but not with the DC function f itself. Moreover, a DC function f has infinitely many DC decompositions which have crucial implications for the qualities (e.g. convergence speed, robustness, efficiency, globality of computed solutions) of DCA. For a given DC program, the choice of optimal DC decompositions is still open. Of course, this depends strongly on the very specific structure of the problem being considered.

d) Similarly to the effect of DC decompositions on DCA, searching the good initial points for DCA is also an open question to be studied.

1.2 Stochastic DCA [1]

The *large sum of DC functions minimization* problem takes the form

$$\min_{x\in\mathbb{R}^d}\left\{F(x) := \frac{1}{n}\sum_{i=1}^{n} F_i(x)\right\}, \tag{1.7}$$

where F_i are DC functions, i.e., $F_i(x) = g_i(x) - h_i(x)$ with g_i and h_i being lower semi-continuous proper convex functions, and n is a very large integer number. The problem of minimizing F under a convex set Ω is also of the type (1.7), as the convex constraint $x \in \Omega$ can be incorporated into the objective function F via the indicator function χ_Ω on Ω defined by $\chi_\Omega = 0$ if $x \in \Omega$, $+\infty$ otherwise.

A natural DC formulation of the problem (1.7) is

$$\min\left\{F(x) = G(x) - H(x) : x \in \mathbb{R}^d\right\}, \tag{1.8}$$

1. The material of this section is from the following work: H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for minimizing a large sum of DC functions and its application in Multi-class Logistic Regression. Submitted & Available on arvXiv [arXiv:1911.03992].*

where

$$G(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x) \text{ and } H(x) = \frac{1}{n}\sum_{i=1}^{n} h_i(x).$$

According to the generic DCA scheme, DCA for solving the problem (1.8) consists of computing, at each iteration l, a subgradient $v^l \in \partial H(x^l)$ and solving the convex subproblem of the form

$$\min\left\{G(x) - \langle v^l, x\rangle : x \in \mathbb{R}^d\right\}. \tag{1.9}$$

As $H = \sum_{i=1}^{n} h_i$, the computation of subgradients of H requires the one of all functions h_i. This may be expensive when n is very large. The main idea of SDCA is to update, at each iteration, the minorant of only some randomly chosen h_i while keeping the minorant of the other h_i. Hence, only the computation of such randomly chosen h_i is required.

SDCA for solving the problem (1.8) [54] is described in Algorithm 1.1.

Algorithm 1.1 SDCA for solving the problem (1.7)

Initialization: Choose $x^0 \in \mathbb{R}^d$, $s_0 = \{1, ..., n\}$, and $l \leftarrow 0$.
Repeat
1. Compute $v_i^l \in \partial h_i(x^l)$ if $i \in s_l$ and keep $v_i^l = v_i^{l-1}$ if $i \notin s_l$, $l > 0$. Set $v^l = \frac{1}{n}\sum_{i=1}^{n} v_i^l$.
2. Compute x^{l+1} by solving the convex problem (1.9).
3. Set $l \leftarrow l + 1$ and randomly choose a small subset $s_l \subset \{1, ..., n\}$.

Until Stopping criterion.

Convergence properties of SDCA: The following theorem shows that the convergence properties of SDCA are guaranteed with probability one. The complete proof proof of Theorem 1.3 is given in [54].

Theorem 1.3. *Assume that $\alpha^* = \inf F(x) > -\infty$, and $|s_l| = b$ for all $l > 0$. Let $\{x^l\}$ be a sequence generated by SDCA. The following statements are hold [54].*

a) $\{F(x^l)\}$ is the almost sure convergent sequence.
b) If $\min_i \rho(h_i) > 0$, then $\sum_{l=1}^{\infty} \|x^l - x^{l-1}\|^2 < +\infty$ and $\lim_{l\to\infty} \|x^l - x^{l-1}\| = 0$, almost surely.
c) If $\min_i \rho(h_i) > 0$, then every limit point of $\{x^l\}$ is a critical point of F with probability one.

1.3 DCA-Like and Accelerated DCA-Like [1]

First, let us considers the *sum of a nonconvex differentiable function and composite functions* minimization problem, which takes the form

$$\min_{\mathbf{x}\in X}\left\{F(x) := f(\mathbf{x}) + \sum_{i=1}^{m} h_i(g_i(\mathbf{x}_i))\right\}, \tag{1.10}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonconvex differentiable function with L-Lipschitz continuous gradient; $g_i : \mathbb{R}^{n_i} \to \mathbb{R}$ $(i = 1 \dots n)$ are continuous convex functions (possibly nonsmooth) with $\sum_{i=1}^{m} n_i = n$; real functions h_i are concave increasing and $\partial(-h_i)(t) \subset \mathbb{R}_-$ if $t \geq g_i(\mathbf{x}_i)$ and X is a closed convex subset of $\mathbb{R}^n$.

1.3.1 DCA for the problem (1.10)

This section outlines the DCA based algorithm to solve the sum of nonconvex and composite functions minimization problem (1.10) in [53]. The problem (1.10) is formulated as follows:

$$\min_{(\mathbf{x},\mathbf{z})}\left\{\varphi(\mathbf{x},\mathbf{z}) := \chi_\Omega(\mathbf{x},\mathbf{z}) + f(\mathbf{x}) + \sum_{i=1}^{m} h_i(z_i)\right\}, \tag{1.11}$$

where $\Omega = \{(\mathbf{x},\mathbf{z}) : \mathbf{x} \in X, g_i(\mathbf{x}_i) \leq z_i, i = 1, ..., m\}$ and χ_Ω is the indicator function of Ω. Let $g(\mathbf{x})$ be the function defined by $g(\mathbf{x}) = (g_1(\mathbf{x}_1), ..., g_m(\mathbf{x}_m))$. The problems (1.10) and (1.11) are equivalent in the following sense.

Proposition 1.3. *[53] A point $\mathbf{x}^* \in X$ is a global (resp. local) solution to the problem* (1.10) *if and only if $(\mathbf{x}^*, g(\mathbf{x}^*))$ is a global (resp. local) solution to the problem* (1.11).

Since the problems (1.10) and (1.11) are equivalent, in the remaining of the section 1.3, we consider the problem (1.11) instead of (1.10). Furthermore, the problem (1.11) can be rewritten as

$$\min_{(\mathbf{x},\mathbf{z})}\left\{\varphi(\mathbf{x},\mathbf{z}) = G_\mu(\mathbf{x},\mathbf{z}) - H_\mu(\mathbf{x},\mathbf{z})\right\}, \tag{1.12}$$

where $G_\mu(\mathbf{x},\mathbf{z}) := \frac{\mu}{2}\|\mathbf{x}\|^2 + \chi_\Omega(\mathbf{x},\mathbf{z})$ and $H_\mu(\mathbf{x},\mathbf{z}) := \frac{\mu}{2}\|\mathbf{x}\|^2 - f(\mathbf{x}) - \sum_{i=1}^{m} h_i(z_i)$ with $\mu > 0$. It is easy to see that $G_\mu(\mathbf{x},\mathbf{z})$ is convex since Ω is a convex set. On the other hand, ∇f is Lipschitz continuous with a constant L, hence $\frac{\mu}{2}\|\mathbf{x}\|^2 - f(\mathbf{x})$ is convex if $\mu \geq L$. Consequently, $H_\mu(\mathbf{x},\mathbf{z})$ is convex, and (1.11) is a standard DC program with $\mu \geq L$. Thus DCA can be investigated to solve the problem (1.11). At each

1. The material of this section is from the following work: H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Novel DCA Based Algorithms for Minimizing the Sum of a Nonconvex Function and Composite Functions with Applications in Machine learning. Submitted & Available on arvXiv [arXiv:1806.09620].*

iteration k, DCA approximates the second DC component H_μ by its affine minorant $H_\mu^{(\mathbf{x}^k,\mathbf{z}^k)}(\mathbf{x},\mathbf{z}) = H_\mu(\mathbf{x}^k,\mathbf{z}^k)+\langle(\mathbf{x},\mathbf{z})-(\mathbf{x}^k,\mathbf{z}^k),(\mathbf{y}^k,\xi^k)\rangle$ with $(\mathbf{y}^k,\xi^k)\in\partial H_\mu(\mathbf{x}^k,\mathbf{z}^k)$ and computes $(\mathbf{x}^{k+1},\mathbf{z}^{k+1})$ by solving the following convex problem

$$\min\left\{\varphi_\mu^k(\mathbf{x},\mathbf{z}) := G_\mu(\mathbf{x},\mathbf{z}) - H_\mu^k(\mathbf{x},\mathbf{z})\right\}. \tag{1.13}$$

Note that φ_μ^k is a majorant of φ. The convex sub-problem (1.13) can be rewritten as follows

$$\min_{(\mathbf{x},\mathbf{z})\in\Omega}\left\{\frac{\mu}{2}\|\mathbf{x}\|^2 - \langle\mathbf{y}^k,\mathbf{x}\rangle + \sum_{i=1}^{m}(-\xi_i^k)z_i\right\}, \tag{1.14}$$

where $\xi_i^k \in \partial(-h_i)(z_i^k)$.

Lemma 1.2 [53] shows that an optimal solution of (1.14) can be obtained by solving a strongly convex problem without the variables $\mathbf{z}$.

Lemma 1.2. *[53] If $\mathbf{x}^{k+1}$ is an optimal solution of the following strongly convex problem*

$$\min_{\mathbf{x}\in X}\{\frac{\mu}{2}\|\mathbf{x}\|^2 - \langle\mathbf{y}^k,\mathbf{x}\rangle + \sum_{i=1}^{m}(-\xi_i^k)g_i(\mathbf{x}_i)\}, \tag{1.15}$$

then $(\mathbf{x}^{k+1},\mathbf{z}^{k+1})$, where $\mathbf{z}^{k+1} = g(\mathbf{x}^{k+1})$, is an optimal solution of (1.14).

Finally, DCA for solving (1.11) is described in Algorithm 1.2.

Algorithm 1.2 DCA for solving (1.11)

Initialize:
Choose $\mathbf{x}^0$, $\mu \geq L$, and $k \leftarrow 0$.
repeat
1. Compute $\xi_i^k \in \partial(-h_i)(z_i^k)$ with $z_i^k = g_i(\mathbf{x}_i^k)$ and $\mathbf{y}^k = \mu\mathbf{x}^k - \nabla f(\mathbf{x}^k)$
2. Compute $\mathbf{x}^{k+1}$ by solving the strongly convex problem (1.15)
3. $k \leftarrow k+1$.

until Stopping criterion

The convergence properties of Algorithm 1.2 are provided in the Theorem 1.4 [53]. We recall that a point $(\mathbf{x}^*,\mathbf{z}^*)\in\mathbb{R}^n\times\mathbb{R}^m$ is called a critical point of the problem (1.11) if and only if

$$[(\nabla f(\mathbf{x}^*),0_m)+\mathcal{N}_\Omega(\mathbf{x}^*,\mathbf{z}^*)]\cap[0_n\times\partial(-h_1)(z_1^*)\times\ldots\times\partial(-h_m)(z_m^*)]\neq\emptyset,$$

where 0_d denotes the zero vector in $\mathbb{R}^d$, and $\mathcal{N}_\Omega(\mathbf{u}^*)$ is the normal cone of Ω at $\mathbf{u}^*$ defined by

$$\mathcal{N}_\Omega(\mathbf{u}^*) = \{\mathbf{v} : \langle\mathbf{v},\mathbf{u}-\mathbf{u}^*\rangle \leq 0, \forall\mathbf{u}\in\Omega\}.$$

Theorem 1.4. *[53] Let $\{\mathbf{x}^k\}$ be the sequence generated by Algorithm 1.2. The following statements hold.*

i) The sequence $\{\varphi(\mathbf{x}^k,g(\mathbf{x}^k))\}$ is decreasing.

ii) If $\alpha = \inf\varphi(\mathbf{x},\mathbf{z}) > -\infty$ then $\sum_{k=0}^{+\infty}\|\mathbf{x}^{k+1}-\mathbf{x}^k\|^2 < +\infty$, and therefore $\lim_{k\to+\infty}\|\mathbf{x}^{k+1}-\mathbf{x}^k\| = 0$.

iii) If $\alpha = \inf\varphi(\mathbf{x},\mathbf{z}) > -\infty$, then any limit point of $\{(\mathbf{x}^k,g(\mathbf{x}^k))\}$ is a critical point of (1.11).

1.3.2 DCA-Like for solving the problem (1.11)

This section presents DCA-Like for for solving (1.11) [53]. As we have mentioned above, the goal of DCA-Like is to avoid bad approximations of the objective function with a too large value of μ (c.f. Algorithm 1.2). The main idea of DCA-Like is to keep the parameter μ as small as possible while finding a convex approximation of objecive function. By using a small value of μ_k, we can get a closer majorant of the objective function which could lead to a better solution. DCA-Like relaxes the key requirement of standard DCA that the minorant $H_\mu^{(\mathbf{x}^k,\mathbf{z}^k)}(\mathbf{x},\mathbf{z}) = H_\mu(\mathbf{x}^k,\mathbf{z}^k) + \langle(\mathbf{x},\mathbf{z}) - (\mathbf{x}^k,\mathbf{z}^k),(\mathbf{y}^k,\xi^k)\rangle$ must be a lower bound of the second component $H_\mu(x,z)$ on the whole space. More precisely, at each iteration k, we only need to find μ_k such that $H_{\mu_k}^{(\mathbf{x}^k,\mathbf{z}^k)}(\mathbf{x},\mathbf{z})$ is a lower bound of $H_{\mu_k}(\mathbf{x},\mathbf{z})$ at x^{k+1}, z^{k+1}, i.e.,

$$H_{\mu_k}(\mathbf{x}^{k+1},\mathbf{z}^{k+1}) \geq H_{\mu_k}^{(\mathbf{x}^k,g(\mathbf{x}^k))}(\mathbf{x}^{k+1},\mathbf{z}^{k+1}). \tag{1.16}$$

where $(\mathbf{x}^{k+1},\mathbf{z}^{k+1}) \in \arg\min \varphi_{\mu_k}^k(\mathbf{x},\mathbf{z})$.

The DCA-Like algorithm for solving (1.11) is described in Algorithm 1.3.

Algorithm 1.3 DCA-Like for solving (1.11)

Initialize:
Choose $\mathbf{x}^0$, $\eta > 1, 0 < \delta < 1$, $\mu_0 > 0$ and $k \leftarrow 0$.
repeat
Compute $\xi_i^k \in \partial(-h_i)(z_i^k)$ with $z_i^k = g_i(\mathbf{x}_i^k)$ and $\nabla f(\mathbf{x}^k)$;
Set $\mu_k = \max\{\mu_0, \delta\mu_{k-1}\}$ if $k > 0$;
Compute $\mathbf{x}^{k+1}$ by solving (1.15) with $\mu = \mu_k$ and $\mathbf{y}^k = \mu_k\mathbf{x}^k - \nabla f(\mathbf{x}^k)$;
while $H_{\mu_k}(\mathbf{x}^{k+1}, g(\mathbf{x}^{k+1})) < H_{\mu_k}^{(\mathbf{x}^k,g(\mathbf{x}^k))}(\mathbf{x}^{k+1}, g(\mathbf{x}^{k+1}))$ **do**
$\mu_k \leftarrow \eta\mu_k$;
Compute $\mathbf{x}^{k+1}$ by solving (1.15) with $\mu = \mu_k$ and $\mathbf{y}^k = \mu_k\mathbf{x}^k - \nabla f(\mathbf{x}^k)$;
end while
$k \leftarrow k + 1$.
until Stopping criterion

Convergence properties of DCA-Like: The complete proof is given in [53].

Theorem 1.5. *[53] Let $\{\mathbf{x}^k\}$ be the sequence generated by DCA-Like (Algorithm 1.3). The following statements hold.*

(i) The sequence $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ is decreasing. More precisely, we have

$$\varphi(\mathbf{x}^k, g(\mathbf{x}^k)) - \varphi(\mathbf{x}^{k+1}, g(\mathbf{x}^{k+1})) \geq \frac{\mu_k}{2}\|\mathbf{x}^{k+1} - \mathbf{x}^k\|^2.$$

(ii) If $\alpha = \inf\varphi(\mathbf{x},\mathbf{z}) > -\infty$ then $\sum_{k=0}^{+\infty}\|\mathbf{x}^{k+1} - \mathbf{x}^k\|^2 < +\infty$, and therefore $\lim_{k\to+\infty}\|\mathbf{x}^{k+1} - \mathbf{x}^k\| = 0$.

(iii) If $\alpha = \inf\varphi(\mathbf{x},\mathbf{z}) > -\infty$, then any limit point of $\{(\mathbf{x}^k, g(\mathbf{x}^k))\}$ is a critical point of (1.11).

Theorem 1.6. *[53] Suppose that* $\inf \varphi(\mathbf{x}, \mathbf{z}) > -\infty$*, and* h_i *is differentiable with locally Lipschitz derivative. Assume further that* φ *has the KL property at any point* $(\mathbf{x}, \mathbf{z}) \in dom\ \partial^L \varphi$*. If* $\{\mathbf{x}^k\}$ *generated by DCA-Like is bounded, then the whole sequence* $\{\mathbf{x}^k\}$ *converges to* $\mathbf{x}^*$*, which* $(\mathbf{x}^*, g(\mathbf{x}^*))$ *is a critical point of* (1.11)*. Moreover, if the function* ψ *appearing in the KL inequality has the form* $\psi(s) = cs^{1-\theta}$ *with* $\theta \in [0, 1)$ *and* $c > 0$*, then the following statements hold:*

(i) If $\theta = 0$*, then the sequences* $\{\mathbf{x}^k\}$ *and* $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ *converge in a finite number of steps to* $\mathbf{x}^*$ *and* φ^**, respectively.*

(ii) If $\theta \in (0, 1/2]$*, then the sequences* $\{\mathbf{x}^k\}$ *and* $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ *converge linearly to* $\mathbf{x}^*$ *and* φ^**, respectively.*

(iii) If $\theta \in (1/2, 1)$*, then there exist positive constants* δ_1*,* δ_2*, and* N_0 *such that* $\|\mathbf{x}^k - \mathbf{x}^*\| \leq \delta_1 k^{-\frac{1-\theta}{2\theta-1}}$ *and* $\varphi(\mathbf{x}^k, g(\mathbf{x}^k)) - \varphi^* \leq \delta_2 k^{-\frac{1}{2\theta-1}}$ *for all* $k \geq N_0$*.*

1.3.3 Accelerated DCA-Like Algorithm for problem (1.11)

According to the DCA-Like scheme, at each iteration, one computes $\mathbf{x}^{k+1}$ from $\mathbf{x}^k$ by solving the convex sub-problem (1.15). The idea of ADCA-Like (Accelerated DCA-Like) [53], in order to accelerate DCA-Like, is to find a point $\mathbf{w}^k$ which is better than $\mathbf{x}^k$ for the computation of $\mathbf{x}^{k+1}$. In this work, we consider $\mathbf{w}^k$ as an extrapolated point of the current iterate $\mathbf{x}^k$ and the previous iterate $\mathbf{x}^{k-1}$:

$$\mathbf{w}^k = \mathbf{x}^k + \frac{t_k - 1}{t_{k+1}} \left(\mathbf{x}^k - \mathbf{x}^{k-1}\right),$$

where $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$. If $\mathbf{w}^k$ is better than the last iterate $\mathbf{x}^k$, i.e., $\varphi(\mathbf{w}^k, g(\mathbf{w}^k)) \leq \varphi(\mathbf{x}^k, g(\mathbf{x}^k))$ then $\mathbf{w}^k$ will be used instead of $\mathbf{x}^k$ to compute $\mathbf{x}^{k+1}$. Note that, ADCA-Like does not require any particular property of the sequence $\{t_k\}$. ADCA-Like algorithm for solving (1.11) is described in Algorithm 1.4.

In Theorem 1.7, we show that any limit point of the sequence generated by ADCA-Like is a critical point of (1.11). The complete proof of the convergence is given in [53].

Theorem 1.7. *[53] Let* $\{x^k\}$ *be the sequence generated by Algorithm 1.4. The following statements hold*

(i) The sequence $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ *is decreasing. More precisely, we have*

$$\varphi(\mathbf{x}^k, g(\mathbf{x}^k)) - \varphi(\mathbf{x}^{k+1}, g(\mathbf{x}^{k+1})) \geq \frac{\mu_k}{2} \|\mathbf{x}^{k+1} - \mathbf{v}^k\|^2.$$

(ii) If $\alpha = \inf \varphi(\mathbf{x}, \mathbf{z}) > -\infty$ *then* $\sum_{k=0}^{+\infty} \|\mathbf{x}^{k+1} - \mathbf{v}^k\|^2 < +\infty$ *and therefore* $\lim_{k \to +\infty} \|\mathbf{x}^{k+1} - \mathbf{v}^k\| = 0$*.*

(iii) If $\alpha = \inf \varphi(\mathbf{x}, \mathbf{z}) > -\infty$*, then any limit point of* $\{(\mathbf{x}^k, g(\mathbf{x}^k))\}$ *is a critical point of* (1.11)*.*

Algorithm 1.4 ADCA-Like for solving (1.11)

Initialize:
Choose $\mathbf{x}^0$, $\mathbf{w}^0 = \mathbf{x}^0$, $\eta > 1, 0 < \delta < 1$, $\mu_0 > 0$, $t_0 = (1+\sqrt{5})/2$, and $k \leftarrow 0$.
repeat
if $\varphi(\mathbf{w}^k, g(\mathbf{w}^k)) \leq \varphi(\mathbf{x}^k, g(\mathbf{x}^k))$ **then**
set $\mathbf{v}^k = \mathbf{w}^k$.
else
set $\mathbf{v}^k = \mathbf{x}^k$.
end if
2: Compute $\xi_i^k \in \partial(-h_i)(z_i^k)$ with $z_i^k = g_i(\mathbf{v}_i^k)$ and $\nabla f(\mathbf{v}^k)$;
3: Set $\mu_k = \max\{\mu_0, \delta\mu_{k-1}\}$ if $k > 0$;
4: Compute $\mathbf{x}^{k+1}$ by solving (1.15) with $\mu = \mu_k$ and $\mathbf{y}^k = \mu_k\mathbf{v}^k - \nabla f(\mathbf{v}^k)$;
while $H_{\mu_k}(\mathbf{x}^{k+1}, g(\mathbf{x}^{k+1})) < H_{\mu_k}^{(\mathbf{v}^k, g(\mathbf{v}^k))}(\mathbf{x}^{k+1}, g(\mathbf{x}^{k+1}))$ **do**
$\mu_k \leftarrow \eta\mu_k$;
Update $\mathbf{x}^{k+1}$ by STEP 4.
end while
6: Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$;
7: Compute $\mathbf{w}^{k+1} = \mathbf{x}^{k+1} + \frac{t_k - 1}{t_{k+1}}\left(\mathbf{x}^{k+1} - \mathbf{x}^k\right)$;
8: $k \leftarrow k + 1$.
until Stopping criterion

The sufficient descent property (i) of Theorem 1.7 is different from Theorem 1.5 due to the intermediate variable $\mathbf{v}^k$. Hence, neither the convergence of the whole sequence $\{\mathbf{x}^k\}$ nor convergence rate for $\{\|\mathbf{x}^k - \mathbf{x}^*\|\}$ can be achieved. However, we can still obtain some interesting results for the sequence $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ under the KL assumption. These properties are presented in the Theorem 1.8.

Theorem 1.8. *[53] Suppose that* $\inf \varphi(\mathbf{x}, \mathbf{z}) > -\infty$, *and* h_i *is differentiable with locally Lipschitz derivative. Assume further that* φ *has the KL property at any point* $(\mathbf{x}, \mathbf{z}) \in$ *dom* $\partial^L\varphi$ *with* $\psi(s) = cs^{1-\theta}$ *for some* $\theta \in [0, 1)$ *and* $c > 0$. *If* $\{\mathbf{x}^k\}$ *generated by accelerated DCA-Like is bounded, then the following statements hold.*

i) If $\theta = 0$, *then the sequence* $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ *converges in a finite number of steps to* φ^*.

ii) If $\theta \in (0, 1/2]$, *then the sequence* $\{\varphi(\mathbf{x}^k, g(\mathbf{x}^k))\}$ *converges linearly to* φ^*.

iii) If $\theta \in (1/2, 1)$, *then there exist positive constants* δ *and* N_0 *such that* $\varphi(\mathbf{x}^k, g(\mathbf{x}^k)) - \varphi^* \leq \delta k^{-\frac{1}{2\theta-1}}$ *for all* $k \geq N_0$.

Chapter 2

Group Variable Selection in Multi-class Logistic Regression[1]

Abstract: *In this chapter, we aim at developing efficient methods to solve the group variable selection in multi-class logistic regression for high-dimensional data. To deal with a large number of features, we consider feature selection method evolving the $\ell_{q,0}$ ($q \in \{1, 2, \infty\}$) regularization. The resulting optimization problem is non-convex, and is tackled by two approaches based on DC programming and DCA. The first approach based on Stochastic DCA, exploits the special structure of the problem in data with a large number of samples. The second approach is based on DCA-Like and its accelerated version Accelerated DCA-Like. DCA-Like relaxes the convexity condition of the second DC component while ensuring the convergence. Accelerated DCA-Like in-cooperates the Nesterov's acceleration technique, which potentially leads to better solution in comparison to DCA-Like. The efficiency of proposed algorithms are empirically demonstrated on both real-world and synthetic datasets. It turns out that both approaches reduce the running time significantly with recent algorithms for the problem of group variable selection in multi-class logistic regression.*

1. The material of this chapter is developed from the following works:
[1] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for the large-sum of non-convex functions problem. Application to group variables selection in multiclass logistic regression.* International Conference on Machine Learning ICML, pp. 3394-3403, 2017.
[2] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for Sparse Multiclass Logistic Regression.* Advances in Intelligent Systems and Computing, 629, pp. 1-12, 2017.
[3] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Novel DCA Based Algorithms for Minimizing the Sum of a Nonconvex Function and Composite Functions with Applications in Machine learning. Submitted & Available on arvXiv [arXiv:1806.09620].*
[4] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Stochastic DCA for minimizing a large sum of DC functions and its application in Multi-class Logistic Regression. Submitted & Available on arvXiv [arXiv:1911.03992].*

2.1 Introduction

In machine learning, supervised learning consists in building a predictor function, based on a labeled training data, which can identify the label of new instances with the highest possible accuracy. Logistic regression, introduced by D. Cox in 1958 [19], is undoubtedly one of the most popular supervised learning methods. Logistic regression can be seen as an extension of linear regression where the dependent variable is categorical. Instead of estimating the outcome by a continuous value like linear regression, logistic regression tries to predict the probability that an instance belongs to a category. Logistic regression has been successfully applied in various real-life problems such as cancer detection [39], medical [8, 6, 103], social science [40], etc. Especially, logistic regression combined with feature selection has been proved to be suitable for high dimensional problems, for instance, document classification [27] and microarray classification [70, 39].

In this chapter, we deal with the multi-class logistic regression problem where the dependent variable has more than two outcome categories. We aim at developing an efficient method for solving the multi-class logistic regression problem to deal with data that has not only a large number of features but also a large number of instances.

The multi-class logistic regression problem can be described as follows. Let $\{(x_i, y_i) : i = 1, ..., n\}$ be a training set with observation vectors $x_i \in \mathbb{R}^d$ and labels $y_i \in \{1, ..., Q\}$ where Q is the number of classes. Let W be the $d \times Q$ matrix whose columns are $W_{:,1}, ..., W_{:,Q}$ and $b = (b_1, ..., b_Q) \in \mathbb{R}^Q$. The couple $(W_{:,i}, b_i)$ forms the hyperplane $f_i := W_{:,i}^T x + b_i$ that separates the class i from the other classes.

In the multi-class logistic regression problem, the conditional probability $p(Y = y|X = x)$ that an instance x belongs to a class y is defined as

$$p(Y = y|X = x) = \frac{\exp(b_y + W_{:,y}^T x)}{\sum_{k=1}^{Q} \exp(b_k + W_{:,k}^T x)}. \tag{2.1}$$

We aim to find (W, b) for which the total probability of the training observations x_i belonging to its correct classes y_i is maximized. A natural way to estimate (W, b) is to minimize the negative log-likelihood function which is defined by

$$\mathcal{L}(W, b) := \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, W, b), \tag{2.2}$$

where $\ell(x_i, y_i, W, b) = -\log p(Y = y_i|X = x_i)$.

In many applications such as information retrial, face recognition, microarray analysis, etc., datasets contain a very large number of features. In such datasets, we often encounter the problem of redundant features (information already presented by other features) and irrelevant features (features that do not contain useful information). Feature selection methods that try to select only useful features for the considered task,

is a popular and efficient way to deal with this problem. A natural way to deal with feature selection problem is to formulate it as a minimization of the ℓ_0-norm (or $\|.\|_0$) where the ℓ_0-norm of $x \in \mathbb{R}^d$ is defined by the number of non-zero components of x, namely, $\|x\|_0 = |i = 1, \ldots, n : x_i \neq 0|$. It is well-known that the problem of minimizing ℓ_0-norm is NP-hard [3] due to the discontinuity of ℓ_0-norm. The minimization of ℓ_0-norm has been extensively studied on both theoretical and practical aspects for individual variable selection in many practical problems. An extensive overview of existing approaches for the minimization of ℓ_0-norm can be found in [63].

In our problem, as mentioned above, the class i $(i = 1, \ldots, Q)$ is separated from the other classes by the hyperplane $f_i = W_{:,i}^T x + b_i$. If the weight $W_{j,i}$ is equal to zero then we can say that the feature j $(j = 1, \ldots, d)$ is not necessary to separate class i from the other classes. Hence, the feature j is to be removed if and only if it is not necessary for any separator f_i $(i = 1, \ldots, Q)$, i.e. all components in the row j of W are zero. Therefore, it is reasonable to consider rows of W as groups. Denote by $W_{j,:}$ the j-th row of the matrix W. The $\ell_{q,0}$-norm of W, i.e., the number of non-zero rows of W, is defined by

$$\|W\|_{q,0} = |\{j \in \{1, ..., d\} : \|W_{j,:}\|_q \neq 0\}|.$$

Hence, the $\ell_{q,0}$ regularized multi-class logistic regression problem is formulated as follows

$$\min_{W,b} \left\{ \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, W, b) + \lambda \|W\|_{q,0} \right\}. \tag{2.3}$$

Our contributions

In this chapter, we investigate DC programming and DCA and propose two approaches for solving the problem (2.3). As mentioned above, the regularization term $\ell_{q,0}$ makes the problem (2.3) non-smooth non-convex. Hence, we first reformulated the problem as a DC program, which DCA-based algorithms will be developed to solve it. We exploit the special structure of the problem to propose an efficient DC decomposition for which the corresponding DCA scheme is very inexpensive: it only requires the projection of points onto balls that is explicitly computed. Next, we considers two approaches based on advanced DC algorithms to efficiently solve it.

The first approach is suitable for large-scale setting, as it can exploit the advantage of the sum structure of (2.3). The proposed alogrithm, `SDCA`, is a Stochastic DCA for the multi-class logistic regression with group variable selection. `SDCA` requires only a small subset of components $\ell(x_i, y_i, W, b)$ at each iteration instead of using all n components for the calculation of W and b. We perform an empirical comparison of `SDCA` with standard methods on very large synthetic and real-world datasets, and show that `SDCA` is efficient in group variable selection ability and classification accuracy as well as running time.

In the second approach, we considered another advanced DC algorithms (DCA-Like and Accelerated DCA-Like) for the problem (2.3). In standard DCA, the second

DC component H must convex. DCA-Like relaxes this condition by using the DCA-Like condition (1.16), which leads to a closer majorant of the objective function and potentially better solution. In addition, Accelerated DCA-Like – a variance of DCA-Like with Nesterov's acceleration techniques – potentially reduces the running time and having better solution in compare with DCA-Like. Hence, the proposed algorithms could potentially reduces the running time and having better solution in compare with standard DCA. We conduct numerical experiment to compare proposed methods with DCA and standard methods on synthetic and real-world datasets, which demonstrates the efficiency of proposed algorithms in both terms of solution quality and running time.

The rest of the chapter is organized as follows. Section 2.2 presents the standard DCA for solving the problem (2.3). Next, section 2.3 shows the stochastic DCA applied for the reformulation of the problem (2.3), including numerical experiments on large-scale datasets. Section 2.4 applies DCA-Like and ADCA-like for the reformulation of the problem (2.3), with numerical experiments. Finally, section 2.6 concludes this chapter.

2.2 Standard DCA for the group variable selection in multi-class logistic regression

An approximation of problem (2.3)**:** In this application, we use a non-convex approximation of the $\ell_{q,0}$-norm ($q \in \{1, 2, \infty\}$) based on the following two penalty functions $\eta_\alpha(s)$:

$$\begin{aligned} \text{Exponential:} \quad \eta_\alpha^{\exp}(s) &= 1 - \exp(-\alpha s), \\ \text{Capped-}\ell_1\text{:} \quad \eta_\alpha^{\text{cap-}\ell_1}(s) &= \min\{1, \alpha s\}. \end{aligned}$$

These penalty functions have shown to be efficient in several problems, for instance, individual variables selection in SVM [11, 46], sparse optimal scoring problem [65], sparse covariance matrix estimation problem [90]. The corresponding approximate problem of (2.3) takes the form:

$$\min_{W,b} \left\{ \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, W, b) + \lambda \sum_{j=1}^{d} \eta_\alpha(\|W_{j,:}\|_q) \right\}. \tag{2.4}$$

Since η_α is increasing on $[0, +\infty)$, the problem (2.4) can be equivalently reformulated as follows

$$\min_{(W,b,t)} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left[\ell(x_i, y_i, W, b) + \chi_\Omega(W, b, t) + \lambda \sum_{j=1}^{d} \eta_\alpha(t_j) \right] \right\}, \tag{2.5}$$

where $\Omega = \{(W, b, t) \in \mathbb{R}^{d \times Q} \times \mathbb{R}^Q \times \mathbb{R}^d : \|W_{j,:}\|_q \leq t_j, j = 1, ..., d\}$.

A DC formulation for problem (2.5)**:** In the sequel, we consider problem (2.5). Since $\ell(x_i, y_i, W, b)$ is differentiable with L-Lipschitz continuous gradient and η_α is

concave, the problem (2.5) takes the form of (1.11) where $f(W,b) = \sum_{i=1}^{n} \ell(x_i, y_i, W, b)$ and $h_j(t_j) = \lambda \eta_\alpha(t_j)$

DCA scheme for problem (2.5)**:** According to DCA scheme (1.2), applying DCA to problem (2.5) consists of computing, at each iteration l, $(U^l, v^l, z^l) \in \partial H(W^l, b^l, t^l)$ where $H(W,b,t) = \sum_{i=1}^{n} h_i(W,b,t)$, and solving the convex sub-problem

$$\min_{(W,b,t)} \left\{ \frac{\rho}{2} \|(W,b)\|^2 + \chi_\Omega(W,b,t) - \langle U^l, W \rangle - \langle v^l, b \rangle - \langle z^l, t \rangle \right\}. \tag{2.6}$$

The computation of (U^l, v^l, z^l) is explicitly defined as follows.

$$(U^l, v^l, z^l) = \frac{1}{n} \sum_{i=1}^{n} (U_i^l, v_i^l, z_i^l),\ (U_i^l, v_i^l, z_i^l) \in \partial h_i(W^l, b^l, t^l).$$

More precisely

$$\begin{aligned}
(U_i^l)_{:,k} &= \rho W_{:,k}^l - \left(p_k^l(x_i) - \delta_{k y_i}\right) x_i, k = 1, ...Q, \\
(v_i^l)_k &= \rho b_k^l - \left(p_k^l(x_i) - \delta_{k y_i}\right), k = 1, ...Q, \\
(z_i^l)_j &= \begin{cases} -\lambda\alpha \exp(-\alpha t_j^l), & j = 1, \ldots, d, \quad \text{if } \eta_\alpha = \eta_\alpha^{\exp} \\ -\lambda\alpha \text{ if } t_j^l \le 1, \text{ and 0 otherwise}, & j = 1, \ldots, d, \quad \text{if } \eta_\alpha = \eta_\alpha^{\text{cap}-\ell_1} \end{cases}
\end{aligned} \tag{2.7}$$

with $p_k^l(x_i) = \exp(b_k^l + (W_{:,k}^l)^T x_i) / (\sum_{h=1}^{Q} b_h^l + (W_{:,h}^l)^T x_i))$, $\delta_{k y_i} = 1$ if $k = y_i$ and 0 otherwise.

The convex sub-problem (2.6) can be solved as follows (note that $z_j^l \le 0$ for $j = 1, ..., d$)

$$W^{l+1} = \arg\min_{W} \left\{ \frac{\rho}{2} \|W\|^2 + \sum_{j=1}^{d} (-z_j^l) \|W_{j,:}\|_q - \langle U^l, W \rangle \right\}, \tag{2.8}$$

$$b^{l+1} = \arg\min_{b} \left\{ \frac{\rho}{2} \|b\|^2 - \langle v^l, b \rangle \right\} = \frac{1}{\rho} v^l, \tag{2.9}$$

$$t_j^{l+1} = \|W_{j,:}^{l+1}\|_q, j = 1, ..., d. \tag{2.10}$$

Since the problem (2.8) is separable in rows of W, solving it amounts to solving d independent sub-problems

$$W_{j,:}^{l+1} = \arg\min_{W_{j,:}} \left\{ \frac{\rho}{2} \|W_{j,:}\|^2 + (-z_j^l) \|W_{j,:}\|_q - \langle U_{j,:}^l, W_{j,:} \rangle \right\}.$$

Moreover, $W_{j,:}^{l+1}$ is computed via the following proximal operator

$$W_{j,:}^{l+1} = \mathbf{prox}_{(-z_j^l)/\rho \|\cdot\|_q} \left(U_{j,:}^l / \rho \right),$$

where the proximal operator $\mathbf{prox}_f(\nu)$ is defined by

$$\mathbf{prox}_f(\nu) = \arg\min_{t} \left\{ \frac{1}{2} \|t - \nu\|^2 + f(t) \right\}.$$

The proximal operator of $(-z_j^l)/\rho \|\cdot\|_q$ can be efficiently computed [83]. Computation of $\mathbf{prox}_{(-z_j^l)/\rho \|\cdot\|_q}(\nu/\rho)$ is detailed in Appendix A.1, and is summarized in Table 2.1. DCA based algorithms for solving (2.5) with $q \in \{1, 2, \infty\}$ are described in Algorithm 2.1.

Table 2.1 – Computation of $W_{j,:}^{l+1} = \mathbf{prox}_{(-z_j^l)/\rho\|.\|_q}\left(U_{j,:}^l/\rho\right)$ corresponding to $q \in \{1, 2, \infty\}$.

q	$\mathbf{prox}_{(-z_j^l)/\rho\|.\|_q}\left(U_{j,:}^l/\rho\right)$
1	$\left(\lvert U_{j,:}^l\rvert/\rho - (-z_j^l)/\rho\right)_+ \circ \operatorname{sign}(U_{j,:}^l)$
2	$\begin{cases}\left(1 - \frac{-z_j^l}{\|U_{j,:}^l\|_2}\right) U_{j,:}^l/\rho & \text{if } \|U_{j,:}^l\|_2 > -z_j^l \\ 0 & \text{if } \|U_{j,:}^l\|_2 \le -z_j^l.\end{cases}$
∞	$\begin{cases}U_{j,:}^l/\rho - \left(\frac{1}{-z_j^l}\lvert U_{j,:}^l\rvert - \delta\right)_+ \circ \operatorname{sign}(U_{j,:}^l) & \text{if } \|U_{j,:}^l\|_1 > -z_j^l \\ 0 & \text{if } \|U_{j,:}^l\|_1 \le -z_j^l,\end{cases}$ where δ satisfies $\sum_{k=1}^Q \left(\frac{1}{-z_j^l}\lvert U_{j,k}^l\rvert - \delta\right)_+ = 1$.

Algorithm 2.1 DCA-$\ell_{q,0}$: DCA for solving (2.5) with $q \in \{1, 2, \infty\}$

Initialization: Choose $(W^0, b^0) \in \mathbb{R}^{d\times Q} \times \mathbb{R}^Q$, $\rho > L$ and $l \leftarrow 0$.

Repeat

1. Compute $(U^l, v^l, z^l) = \frac{1}{n}\sum_{i=1}^n (U_i^l, v_i^l, z_i^l)$, where (U_i^l, v_i^l, z_i^l), $i = 1, ..., n$ are computed by defined in (2.7).
2. Compute $(W^{l+1}, b^{l+1}, t^{l+1})$ according to Table 2.1, (2.9) and (2.10), respectively.
3. $l \leftarrow l + 1$.

Until Stopping criterion.

2.3 SDCA for the group variable selection in multi-class logistic regression

Rewrite problem (2.5) **into the form of** (1.7)**:** Since $\ell(x_i, y_i, W, b)$ is differentiable with L-Lipschitz continuous gradient and η_α is concave, the problem (2.5) takes the form of (1.7) where the function $F_i(W, b, t)$ is given by

$$F_i(W, b, t) := g_i(W, b, t) - h_i(W, b, t)$$

where

$$g_i(W, b, t) = \frac{\rho}{2}\|(W, b)\|^2 + \chi_\Omega(W, b, t),$$

$$h_i(W, b, t) = \frac{\rho}{2}\|(W, b)\|^2 - \ell(x_i, y_i, W, b) - \lambda\sum_{j=1}^d \eta_\alpha(t_j),$$

with $\rho > L$.

SDCA scheme for problem (2.5)**:** Following SDCA in section 1.2, at each iteration l, we have to compute $(U_i^l, v_i^l, z_i^l) \in \partial h_i(W^l, b^l, t^l)$ for $i \in s_l$ and keep $(U_i^l, v_i^l, z_i^l) = (U_i^{l-1}, v_i^{l-1}, z_i^{l-1})$ for $i \notin s_l$, where s_l is a randomly chosen subset of the indexes, and solve the convex sub-problem taking the form of (2.6). This is the main speed-up of SDCA in comparison with DCA. In standard DCA in section 2.7, the most expensive operation is the computing the subgradient $(U^l, v^l) = \sum_i^n (U_i, v_i)$, which is linearly scale with n. In contrast, SDCA only uses a subset s_l to compute $(U^l, v^l) = \sum_{i \in s_l} (U_i, v_i)$, which significantly reduces the computing time of this bottleneck. It is clear that each iteration of SDCA is much faster than DCA, hence SDCA is potentially faster than DCA.

SDCA for solving (2.5) is described in Algorithm 2.2.

Algorithm 2.2 SDCA-$\ell_{q,0}$: SDCA for solving (2.5) with $q \in \{1, 2, \infty\}$

Initialize:

Choose $(W^0, b^0) \in \mathbb{R}^{d \times Q} \times \mathbb{R}^Q$, $t_j^0 = \|W_{j,:}^0\|_q$, $\rho > L$, $s_0 = \{1, ..., n\}$ and $l \leftarrow 0$.

repeat

1. Compute (U_i^l, v_i^l, z_i^l) by (2.7) if $i \in s_l$ and keep $(U_i^l, v_i^l, z_i^l) = (U_i^{l-1}, v_i^{l-1}, z_i^{l-1})$ if $i \notin s_l$. Set $(U^l, v^l, z^l) = \frac{1}{n} \sum_{i=1}^n (U_i^l, v_i^l, z_i^l)$.

2. Compute $(W^{l+1}, b^{l+1}, t^{l+1})$ according to Table 2.1, (2.9) and (2.10), respectively.

3. $l \leftarrow l + 1$ and randomly choose a small subset $s_l \subset \{1, ..., n\}$.

until Stopping criterion.

2.3.1 Numerical experiment

2.3.1.1 Datasets

To evaluate the performances of algorithms, we performed numerical experiments on two types of data: real datasets (*covertype*, *madelon*, *miniboone*, *protein*, *sensit* and *sensorless*) and simulated datasets (*sim_1*, *sim_2* and *sim_3*). All real-world datasets are taken from the well-known UCI and LibSVM data repositories. We give below a brief description of real datasets:

— *covertype* belongs to the Forest Cover Type Prediction from strictly cartographic variables challenge [2]. It is a very large dataset containing 581, 012 points described by 54 variables.

— *madelon* is one of five datasets used in the NIPS 2003 feature selection challenge [3]. The dataset contains 2600 points, each point is represented by 500 variables. Among 500 variables, there are only 5 informative variables and 15 redundant variables (which are created by linear combinations of 5 informative

2. `https://archive.ics.uci.edu/ml/datasets/Covertype`
3. `https://archive.ics.uci.edu/ml/datasets/Madelon`

variables). The 480 others variables were added and have no predictive power. Notice that *madelon* is a highly non-linear dataset.
— *miniboone* is taken form the MiniBooNE experiment to observe neutrino oscillations [4], containing $130,065$ data points.
— *protein* [5] is a dataset for classifying protein second structure state (α, β, and coil) of each residue in amino acid sequences, including $24,387$ data points.
— *sensit* [5] dataset obtained from distributed sensor network for vehicle classification. It consists of $98,528$ data points categorized into 3 classes: Assault Amphibian Vehicle (AAV), Dragon Wagon (DW) and noise.
— *sensorless* measures electric current drive signals from different operating conditions, which is classified into 11 different classes [6]. It is a huge dataset, which contains $58,509$ data points, described by 48 variables.

We generate three synthetic datasets (*sim_1*, *sim_2* and *sim_3*) by the same process proposed in [115]. In the first dataset (*sim_1*), variables are independent and have different means in each class. In dataset (*sim_2*), variables also have different means in each class, but they are dependent. The last synthetic dataset (*sim_3*) has different one-dimensional means in each class with independent variables. Detail produces to generate three simulated datasets are described as follows:

— For *sim_1*: we generate a four-classes classification problem. Each class is assumed to have a multivariate normal distribution $\mathcal{N}(\mu_k, I)$, $k = 1, 2, 3, 4$ with dimension of $d = 50$. The first 10 components of μ_1 are 0.5, $\mu_{2j} = 0.5$ if $11 \leq j \leq 20$, $\mu_{3j} = 0.5$ if $21 \leq j \leq 30$, $\mu_{4j} = 0.5$ if $31 \leq j \leq 40$ and 0 otherwise. We generate $100,000$ instances with equal probabilities.
— For *sim_2*: this synthetic dataset contains three classes of multivariate normal distributions $\mathcal{N}(\mu_k, \Sigma)$, $k = 1, 2, 3$, each of dimension $d = 50$. The components of $\mu_1 = 0$, $\mu_{2j} = 0.4$ and $\mu_{3j} = 0.8$ if $j \leq 40$ and 0 otherwise. The covariance matrix Σ is the block diagonal matrix with five blocks of dimension 10×10 whose element (j, j') is $0.6^{|j-j'|}$. We generate $150,000$ instances.
— For *sim_3*: this synthetic dataset consists of four classes. For class $k = 1, 2, 3, 4$, $i \in C_k$ then $X_{ij} \sim \mathcal{N}(0, 1)$ for $j > 100$, and $X_{ij} \sim \mathcal{N}(\frac{k-1}{3}, 1)$ otherwise, where $\mathcal{N}(\mu, \sigma^2)$ denotes the Gaussian distribution with mean μ and variance σ^2. We generate $62,500$ data points for each class.

The number of points, variables and classes of each dataset are summarized in the first column of Table 2.2.

2.3.1.2 Comparative algorithms

To the best of our knowledge, there is no existing method in the literature for solving the group variable selection in multi-class logistic regression using $\ell_{q,0}$ regularization. However, closely connected to the Lasso (ℓ_1-norm), [109] proposed to use the convex

4. `https://archive.ics.uci.edu/ml/datasets/MiniBooNE+particle+identification`
5. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`
6. `https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis`

regularization $\ell_{2,1}$ instead of $\ell_{2,0}$. Thus, the resulting problem takes the form

$$\min_{W,b} \left\{ \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, W, b) + \lambda \|W\|_{2,1} \right\}. \tag{2.11}$$

A coordinate gradient descent method, named `msgl`, was proposed in [109] to solve the problem (2.11). `msgl` is a comparative algorithm in our experiment.

On another hand, we are interested in a comparison between our algorithms and a stochastic based method. A stochastic gradient descent algorithm to solve (2.11), named `SPGD-`$\ell_{2,1}$, is developed for this purpose. `SPGD-`$\ell_{2,1}$ is described as follows.

Algorithm 2.3 SPGD-$\ell_{2,1}$: Stochastic Proximal Gradient Descent for solving (2.11)

Initialization: Choose $(W^0, b^0) \in \mathbb{R}^{d\times Q} \times \mathbb{R}^Q$, and $l \leftarrow 0$.

Repeat

1. Randomly choose a small subset $s_l \subset \{1, ..., n\}$. Set $\alpha_l = \frac{n}{10l}$. Compute $\bar{U}^l_{:,k} = W^l_{:,k} - \frac{\alpha_l}{|s_l|} \sum_{i\in s_l} \left(p^l_k(x_i) - \delta_{ky_i}\right) x_i, k = 1, ...Q$.
2. Compute (W^{l+1}, b^{l+1}) by

$$\begin{aligned} W^{l+1}_{j,:} &= \left(\|\bar{U}^l_{j,:}\|_2 - \alpha_l\lambda\right)_+ \frac{\bar{U}^l_{j,:}}{\|\bar{U}^l_{j,:}\|_2}, j = 1, ..., d \\ b^{l+1}_k &= b^l_k - \frac{\alpha_l}{|s_l|} \sum_{i\in s_l} \left(p^l_k(x_i) - \delta_{ky_i}\right), k = 1, ..., Q. \end{aligned} \tag{2.12}$$

3. $l \leftarrow l + 1$.

Until Stopping criterion.

2.3.1.3 Experiment setting

We randomly split each dataset into a training set and a test set. The training set contains 80% of the total number of points and the remaining 20% are used as test set.

In order to evaluate the performance of algorithms, we consider the following three criteria: the classification accuracy (percentage of well classified point on test set), the sparsity of obtained solution and the running time (measured in seconds). The sparsity is computed as the percentage of selected variables. Note that a variable $j \in \{1, \ldots, d\}$ is considered to be removed if all components of the row j of W are smaller than a threshold, i.e., $|W_{j,i}| \leq 10^{-8}, \forall i \in 1, \ldots, Q$. We perform each algorithm 10 times and report the mean and standard deviation of each criterion.

We use the early-stopping condition for `SDCA` and `SPGD-`$\ell_{2,1}$. Early-stopping is a well-know technique in machine learning, especially in stochastic learning which permits to avoid the over-fitting in learning. More precisely, after each epoch, we compute the classification accuracy on a validation set which contains 20% randomly

chosen data points of training set. We stop SDCA and SPGD-$\ell_{2,1}$ if the classification accuracy is not improved after $n_{patience} = 5$ epochs. The batch size of stochastic algorithms (SDCA and SPGD-$\ell_{2,1}$) is set to 10%. DCA is stopped if the difference between two consecutive objective functions is smaller than a threshold $\epsilon_{stop} = 10^{-6}$. For msgl, we use its default stopping parameters as in [109]. We also stop algorithms if they exceed 2 hours of running time in the training process.

The parameter α for controlling the tightness of zero-norm approximation is chosen in the set $\{0.5, 1, 2, 5\}$. We use the solution-path procedure for the trade-off parameter λ. Let $\lambda_1 > \lambda_2 > ... > \lambda_l$ be a decreasing sequence of λ. At step k, we solve the problem (2.3) with $\lambda = \lambda_k$ from the initial point chosen as the solution of the previous step $k-1$. Starting with a large value of λ, we privilege the sparsity of solution (i.e. selecting very few variables) over the classification ability. Then by decreasing the value λ decreases, we select more variables in order to increase the classification accuracy. In our experiments, the sequence of λ is set to $\{10^4, 3 \times 10^3, 10^3, \ldots, 3 \times 10^{-3}, 10^{-3}\}$.

All experiments are performed on a PC Intel (R) Xeon (R) E5-2630 v2 @2.60 GHz with 32GB RAM.

2.3.1.4 Experiment 1

In this experiments, we study the effectiveness of SDCA. For this purpose, we choose the $\ell_{2,0}$ regularization, and perform a comparison between SDCA-$\ell_{2,0}$-exp and DCA-$\ell_{2,0}$-exp. Furthermore, we will compare SDCA-$\ell_{2,0}$-exp with msgl and SPGD-$\ell_{2,1}$, two algorithms for solving the multi-class logistic regression using $\ell_{2,1}$ regularization (c.f Section 2.3.1.2).

The comparative results between are reported in Table 2.2 and Figure 2.1. Note that the running time is plotted in logarithmic scale.

Comparison between SDCA-$\ell_{2,0}$ and DCA-$\ell_{2,0}$-exp

In term of classification accuracy, SDCA-$\ell_{2,0}$-exp produces fairly similar result comparing with DCA-$\ell_{2,0}$-exp. DCA-$\ell_{2,0}$-exp is better than SDCA-$\ell_{2,0}$-exp on 4 datasets (*covertype*, *sensit*, *sensorless* and *sim_3*) while SDCA-$\ell_{1,0}$-exp gives better results on 2 datasets (*madelon* and *protein*). The two biggest gaps (3.49% and 1.17%) occur on dataset *sensorless* and *sensit* respectively.

As for the sparsity of solution, DCA-$\ell_{2,0}$-exp and SDCA-$\ell_{2,0}$-exp provide the same results on 4 datasets (*miniboon*, *sim_1*, *sim_2* and *sim_3*). DCA-$\ell_{2,0}$-exp suppresses more variables than SDCA-$\ell_{2,0}$-exp on 3 datasets (*protein*, *sensit* and *sensorless*), while SDCA-$\ell_{2,0}$-exp gives better sparsity on *covertype* and *madelon*. The gain of DCA-$\ell_{2,0}$-exp on this criterion is quite high, up to 22.3% on dataset *protein*.

Concerning the running time, SDCA-$\ell_{2,0}$-exp clearly outperforms DCA-$\ell_{2,0}$-exp. Except for *miniboone* where DCA-$\ell_{2,0}$-exp is 1.11 second faster, the gain of SDCA-$\ell_{2,0}$-exp is huge. SDCA-$\ell_{2,0}$-exp is up to 19.58 times faster than DCA-$\ell_{2,0}$-exp (dataset *cover-*

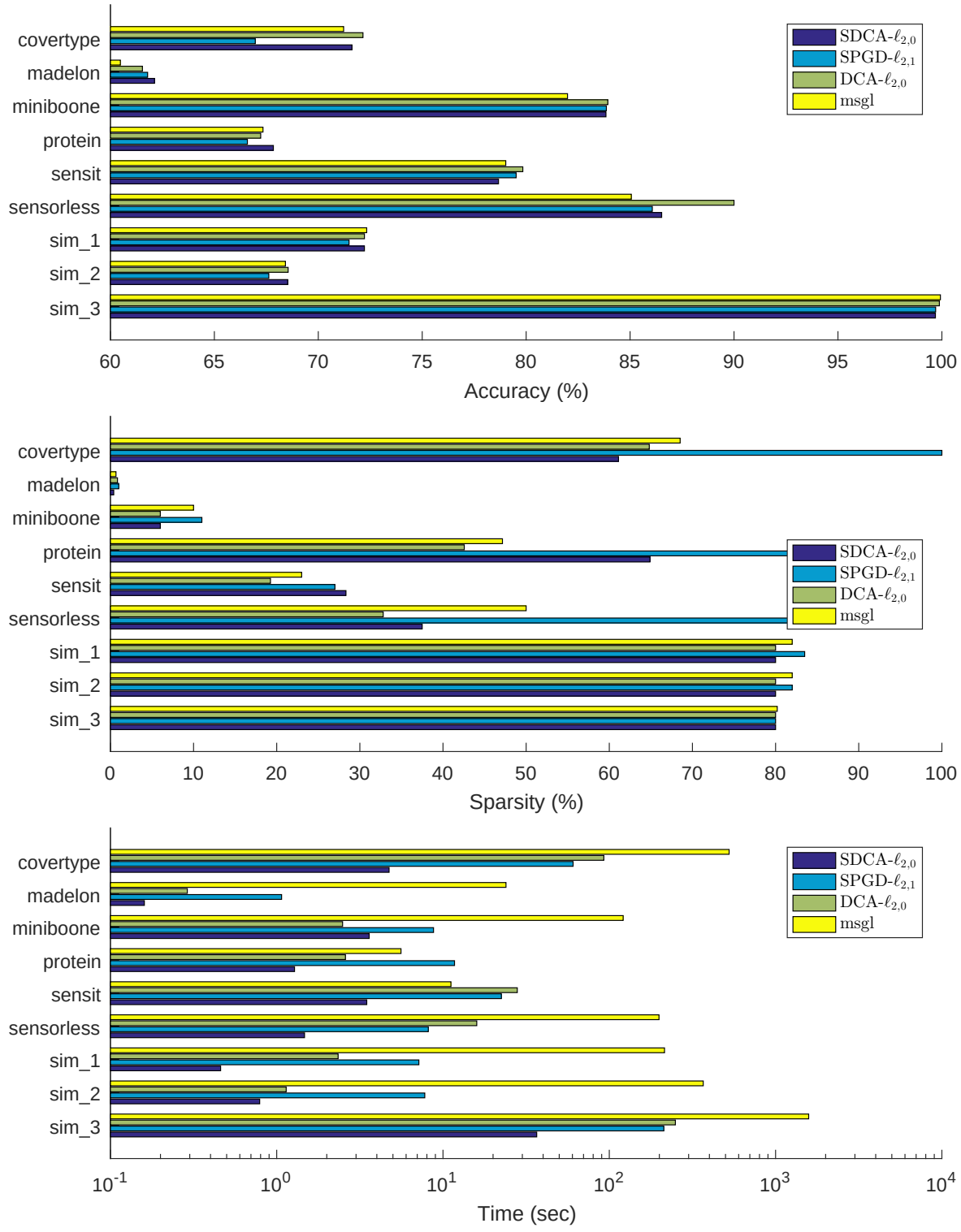


Figure 2.1 – Comparative results between `SDCA-`$\ell_{2,0}$`-exp`, `DCA-`$\ell_{2,0}$`-exp`, `SPGD-`$\ell_{2,1}$ and `msgl` (running time is plotted on a logarithmic scale).

type).

Overall, `SDCA-`$\ell_{2,0}$`-exp` is able to achieve equivalent classification accuracy with a

running time much smaller than DCA-$\ell_{2,0}$-exp.

Comparison between SDCA-$\ell_{2,0}$-exp and msgl.

SDCA-$\ell_{2,0}$-exp provides better classification accuracy on 6 out of 9 datasets with a gain up to 1.85%. For the 3 remaining datasets, the gain of msgl in accuracy is smaller than 0.3%. As for the sparsity of solution, the two algorithms are comparable. SDCA-$\ell_{2,0}$-exp is by far faster than msgl on all datasets, from 3.2 times to 470 time faster.

Comparison between SDCA-$\ell_{2,0}$-exp and SPGD-$\ell_{2,1}$.

In term of classification accuracy, SDCA is better on 6 datasets with a gain up to 4.65%, whereas SPGD only gives better result on *sensit*. Moreover, the number of selected variables by SPGD-$\ell_{2,1}$ is considerably higher. SPGD-$\ell_{2,1}$ chooses from 2% to 51.39% more variables than SDCA in 6 over 9 cases (*covertype*, *miniboone*, *protein*, *sensorless*, *sim_1*, and *sim_2*), and $> 27\%$ more in 3 over 9 cases (*covertype*, *protein* and *sensorless*). As for the running time, SDCA-$\ell_{2,0}$-exp is up to 15.68 times faster than SPGD-$\ell_{2,1}$. Overall, SDCA-$\ell_{2,0}$-exp clearly outperforms SPGD-$\ell_{2,1}$ on all three criteria.

In conclusion, as expected, SDCA-$\ell_{2,0}$-exp reduces considerably the running time of DCA-$\ell_{2,0}$ while achieving equivalent classification accuracy. Moreover, SDCA-$\ell_{2,0}$-exp outperforms the two related algorithms msgl and SPGD-$\ell_{2,1}$.

2.3.1.5 Experiment 2

This experiment studies the effectiveness of different non-convex regularizations $\ell_{q,0}$. We compare three algorithms SDCA-$\ell_{1,0}$-exp, SDCA-$\ell_{2,0}$-exp and SDCA-$\ell_{\infty,0}$-exp. The results are reported in Table 2.2 and plotted in Figure 2.2.

In term of classification accuracy, SDCA-$\ell_{1,0}$-exp and SDCA-$\ell_{2,0}$-exp are comparable and are slightly better than SDCA-$\ell_{\infty,0}$-exp. SDCA-$\ell_{1,0}$-exp produces similar results with SDCA-$\ell_{2,0}$-exp on 6 out of 9 datasets, where the gap is lower than 0.3% in classification accuracy. For *protein*, *sensorless* and *sensit*, SDCA-$\ell_{\infty,0}$-exp provides slightly better classification accuracy than SDCA-$\ell_{1,0}$-exp and SDCA-$\ell_{2,0}$-exp. This is due to the fact that SDCA-$\ell_{\infty,0}$-exp selects much more variables than the two others.

As for the sparsity of solution, SDCA-$\ell_{2,0}$-exp is the best on 8 out of 9 datasets (except for *protein*). SDCA-$\ell_{1,0}$-exp selects moderately more variables than SDCA-$\ell_{2,0}$-exp, from 5.67% to 17.19%. In contrast to SDCA-$\ell_{2,0}$-exp, SDCA-$\ell_{\infty,0}$-exp suppresses less variables than SDCA-$\ell_{1,0}$-exp and SDCA-$\ell_{2,0}$-exp on all datasets, except *covertype*. Especially, on dataset *sensorless*, SDCA-$\ell_{\infty,0}$-exp selects 60.42% (resp. 43.23%) more variables than SDCA-$\ell_{2,0}$-exp (resp. SDCA-$\ell_{1,0}$-exp).

In term of running time, SDCA-$\ell_{1,0}$-exp is the fastest and SDCA-$\ell_{2,0}$-exp is the slowest among the three algorithms. SDCA-$\ell_{1,0}$-exp is up to 3.4 time faster than SDCA-$\ell_{2,0}$-exp and 2.06 times faster than SDCA-$\ell_{\infty,0}$-exp.

Overall, SDCA-$\ell_{1,0}$-exp and SDCA-$\ell_{2,0}$-exp provide comparable results and realize a better trade-off between classification and sparsity of solution than SDCA-$\ell_{\infty,0}$-exp.

2.3.1.6 Experiment 3

This experiment studies the effect of different types of sparsity approximations (capped-ℓ_1 and exponential approximation). We compare two algorithms: SDCA-$\ell_{2,0}$-exp and SDCA-$\ell_{2,0}$-capℓ_1. The results are reported in Figure 2.3 and Table 2.2.

For *sensit*, *madelon*, *sim_1*, *sim_2* dataset, both algorithms have similar performance in all three criteria. The differences in terms of accuracy are negligible ($< 0.1\%$), while the gaps of sparsity and running time are mostly the same.

For *sim_3* and *miniboone* dataset, both algorithms choose the same number of features. However, SDCA-$\ell_{2,0}$-capℓ_1 is faster than SDCA-$\ell_{2,0}$-exp (by 41% and 67% respectively), while SDCA-$\ell_{2,0}$-exp gives better (or similar) result in terms of classification accuracy.

For *covertype*, *sensorless* and *protein* dataset, SDCA-$\ell_{2,0}$-exp provides better results than SDCA-$\ell_{2,0}$-capℓ_1. SDCA-$\ell_{2,0}$-exp furnishes results with higher classification accuracy in 2 out of 3 cases (*covertype* and *sensorless*) while having lower lower sparsity in 2 out of 3 cases (*protein* and *sensorless*). In terms of running time, SDCA-$\ell_{2,0}$-exp is faster than SDCA-$\ell_{2,0}$-capℓ_1 by at least 1.5 times.

Overall, SDCA-$\ell_{2,0}$-exp clearly shows better results SDCA-$\ell_{2,0}$-capℓ_1 in three criteria.

Table 2.2 – Comparative results on both synthetic and real datasets.
Bold values correspond to best results for each dataset. n, d and Q is the number of instances, the number of variables and the number of classes respectively.

Dataset	Algorithm	Accuracy (%)		Time (s)		Sparsity (%)	
		Mean	STD	Mean	STD	Mean	STD
covertype	SDCA-$\ell_{2,0}$-exp	71.62	0.05	**4.74**	**0.07**	61.11	3.21
$n = 581,012$	SDCA-$\ell_{1,0}$-exp	71.34	0.07	10.27	1.25	69.91	1.77
$d = 54$	SDCA-$\ell_{\infty,0}$-exp	69.92	0.38	11.93	0.88	60.49	1.51
$Q = 7$	SDCA-$\ell_{2,0}$-capℓ_1	70.40	0.03	7.47	5.69	57.41	1.85
	SDCA-$\ell_{1,0}$-capℓ_1	68.60	0.29	8.98	2.03	**25.93**	**0.00**
	SDCA-$\ell_{\infty,0}$-capℓ_1	70.16	0.03	14.80	3.63	56.79	3.85
	DCA-$\ell_{2,0}$-exp	72.15	0.08	92.73	0.51	64.81	1.51
	DCA-$\ell_{1,0}$-exp	**72.28**	**0.07**	57.93	2.87	73.61	0.93
	DCA-$\ell_{\infty,0}$-exp	69.39	0.10	57.22	5.36	42.13	0.93
	DCA-$\ell_{2,0}$-capℓ_1	70.40	0.03	61.15	3.34	57.41	1.85
	DCA-$\ell_{1,0}$-capℓ_1	69.41	0.39	37.20	1.23	69.14	1.07
	DCA-$\ell_{\infty,0}$-capℓ_1	72.09	0.13	19.99	0.10	49.38	5.35
	SPGD-$\ell_{2,1}$	66.97	0.51	60.59	7.09	100.00	0.00

	msgl	71.22	0.02	525.49	1.10	68.52	0.00
madelon	SDCA-$\ell_{2,0}$-exp	**62.12**	**1.00**	0.16	0.02	**0.40**	**0.12**
$n = 2,600$	SDCA-$\ell_{1,0}$-exp	61.92	0.80	**0.14**	**0.03**	0.65	0.10
$d = 500$	SDCA-$\ell_{\infty,0}$-exp	61.68	1.05	0.16	0.01	0.70	1.47
$Q = 2$	SDCA-$\ell_{2,0}$-capℓ_1	62.18	1.35	0.15	0.09	0.40	0.00
	SDCA-$\ell_{1,0}$-capℓ_1	61.73	1.26	0.16	0.02	1.53	0.12
	SDCA-$\ell_{\infty,0}$-capℓ_1	61.99	1.06	0.16	0.31	10.60	0.20
	DCA-$\ell_{2,0}$-exp	61.54	0.79	0.29	0.27	0.85	0.19
	DCA-$\ell_{1,0}$-exp	61.83	1.12	0.32	0.02	0.55	0.10
	DCA-$\ell_{\infty,0}$-exp	61.88	1.03	2.17	0.01	4.65	0.25
	DCA-$\ell_{2,0}$-capℓ_1	61.28	2.23	0.21	0.00	0.93	0.12
	DCA-$\ell_{1,0}$-capℓ_1	61.54	1.57	0.41	0.00	1.07	0.31
	DCA-$\ell_{\infty,0}$-capℓ_1	60.58	1.07	0.35	0.01	2.73	0.23
	SPGD-$\ell_{2,1}$	61.79	0.80	1.07	0.03	1.00	0.20
	msgl	60.48	2.37	23.92	0.12	0.67	0.00
miniboone	SDCA-$\ell_{2,0}$-exp	83.84	0.08	3.60	0.04	6.00	0.00
$n = 130,065$	SDCA-$\ell_{1,0}$-exp	83.90	0.10	1.57	0.06	8.00	0.00
$d = 50$	SDCA-$\ell_{\infty,0}$-exp	83.10	0.22	1.62	0.04	8.00	0.00
$Q = 2$	SDCA-$\ell_{2,0}$-capℓ_1	83.31	0.15	**1.18**	**0.01**	6.00	0.00
	SDCA-$\ell_{1,0}$-capℓ_1	82.50	0.06	2.96	0.19	8.00	0.00
	SDCA-$\ell_{\infty,0}$-capℓ_1	83.77	0.10	4.22	0.28	16.00	4.00
	DCA-$\ell_{2,0}$-exp	83.93	0.12	2.49	0.31	6.00	0.00
	DCA-$\ell_{1,0}$-exp	**84.19**	**0.15**	9.42	0.09	8.00	0.00
	DCA-$\ell_{\infty,0}$-exp	81.54	0.12	9.81	3.45	8.00	0.00
	DCA-$\ell_{2,0}$-capℓ_1	83.74	0.07	7.04	0.01	6.00	0.00
	DCA-$\ell_{1,0}$-capℓ_1	83.11	0.05	7.54	0.00	**4.00**	**0.00**
	DCA-$\ell_{\infty,0}$-capℓ_1	82.81	0.09	7.14	0.00	15.33	1.15
	SPGD-$\ell_{2,1}$	83.86	0.13	8.77	0.41	11.00	1.15
	msgl	81.99	0.21	121.17	4.30	10.00	0.00
protein	SDCA-$\ell_{2,0}$-exp	67.84	1.11	1.28	0.06	64.89	1.95
$n = 24,387$	SDCA-$\ell_{1,0}$-exp	67.23	0.90	1.47	0.02	63.67	2.39
$d = 357$	SDCA-$\ell_{\infty,0}$-exp	68.13	0.57	1.36	0.06	92.79	0.86
$Q = 3$	SDCA-$\ell_{2,0}$-capℓ_1	66.41	1.12	**1.13**	**0.12**	**22.64**	**0.47**
	SDCA-$\ell_{1,0}$-capℓ_1	67.25	1.24	1.33	0.14	65.73	1.09
	SDCA-$\ell_{\infty,0}$-capℓ_1	**68.19**	**1.06**	**1.13**	**0.10**	77.47	0.42
	DCA-$\ell_{2,0}$-exp	67.23	0.75	2.59	0.02	42.56	1.66
	DCA-$\ell_{1,0}$-exp	66.19	0.96	3.77	0.41	33.36	1.87
	DCA-$\ell_{\infty,0}$-exp	66.93	0.75	13.53	2.12	54.21	0.61
	DCA-$\ell_{2,0}$-capℓ_1	67.04	0.72	3.35	0.00	50.47	1.27
	DCA-$\ell_{1,0}$-capℓ_1	67.89	0.60	3.43	0.00	79.68	0.58
	DCA-$\ell_{\infty,0}$-capℓ_1	66.90	0.84	3.66	1.04	58.43	1.46
	SPGD-$\ell_{2,1}$	66.59	1.82	11.73	2.80	92.70	2.50
	msgl	67.34	0.48	5.59	0.36	47.15	1.32

sensit	SDCA-$\ell_{2,0}$-exp	78.67	0.11	3.48	0.21	28.33	8.50
$n = 98,528$	SDCA-$\ell_{1,0}$-exp	79.64	0.22	3.11	0.96	34.00	17.35
$d = 100$	SDCA-$\ell_{\infty,0}$-exp	79.73	0.28	**1.61**	**0.07**	53.67	6.81
$Q = 3$	SDCA-$\ell_{2,0}$-capℓ_1	78.59	0.08	2.94	0.17	33.80	5.31
	SDCA-$\ell_{1,0}$-capℓ_1	79.71	0.23	2.94	2.12	100.00	0.00
	SDCA-$\ell_{\infty,0}$-capℓ_1	78.83	0.24	2.91	0.20	35.00	2.74
	DCA-$\ell_{2,0}$-exp	**79.84**	**0.11**	27.97	0.80	19.25	0.50
	DCA-$\ell_{1,0}$-exp	79.65	0.21	18.31	4.90	**17.50**	**0.58**
	DCA-$\ell_{\infty,0}$-exp	79.16	0.17	42.91	5.24	91.50	2.38
	DCA-$\ell_{2,0}$-capℓ_1	78.92	0.15	26.36	2.22	56.67	1.53
	DCA-$\ell_{1,0}$-capℓ_1	78.91	0.38	27.05	2.80	57.33	0.58
	DCA-$\ell_{\infty,0}$-capℓ_1	79.20	0.17	35.78	2.69	91.67	7.23
	SPGD-$\ell_{2,1}$	79.52	0.27	22.44	2.41	27.00	1.00
	msgl	79.02	0.13	11.16	0.53	23.00	0.00
sensorless	SDCA-$\ell_{2,0}$-exp	86.52	0.78	1.47	0.16	37.50	5.10
$n = 58,509$	SDCA-$\ell_{1,0}$-exp	87.33	0.27	1.40	0.09	54.69	10.67
$d = 48$	SDCA-$\ell_{\infty,0}$-exp	86.91	0.19	1.41	0.38	97.92	2.08
$Q = 11$	SDCA-$\ell_{2,0}$-capℓ_1	84.77	0.08	2.45	0.13	68.06	1.20
	SDCA-$\ell_{1,0}$-capℓ_1	82.89	0.30	2.69	0.62	72.92	2.08
	SDCA-$\ell_{\infty,0}$-capℓ_1	87.12	0.72	**1.36**	**0.09**	**25.69**	**1.20**
	DCA-$\ell_{2,0}$-exp	90.00	0.31	15.96	0.65	32.81	1.04
	DCA-$\ell_{1,0}$-exp	89.11	0.18	16.28	0.97	31.25	0.00
	DCA-$\ell_{\infty,0}$-exp	**90.76**	**0.14**	18.99	0.81	100.00	0.00
	DCA-$\ell_{2,0}$-capℓ_1	89.60	1.15	24.75	1.39	53.47	1.20
	DCA-$\ell_{1,0}$-capℓ_1	88.87	1.04	16.28	0.44	47.92	0.80
	DCA-$\ell_{\infty,0}$-capℓ_1	81.06	3.9	14.99	3.22	41.67	0.70
	SPGD-$\ell_{2,1}$	86.07	1.39	8.16	1.05	88.89	2.41
	msgl	85.06	0.31	199.00	41.75	50.00	0.00
sim_1	SDCA-$\ell_{2,0}$-exp	72.22	0.46	0.46	0.02	**80.00**	**0.00**
$n = 100,000$	SDCA-$\ell_{1,0}$-exp	72.24	0.43	0.46	0.03	**80.00**	**0.00**
$d = 50$	SDCA-$\ell_{\infty,0}$-exp	72.24	0.47	0.56	0.06	**80.00**	**0.00**
$Q = 4$	SDCA-$\ell_{2,0}$-capℓ_1	72.24	0.52	0.50	0.04	**80.00**	**0.00**
	SDCA-$\ell_{1,0}$-capℓ_1	72.24	0.58	0.42	0.06	**80.00**	**0.00**
	SDCA-$\ell_{\infty,0}$-capℓ_1	72.21	0.58	0.51	0.07	**80.00**	**0.00**
	DCA-$\ell_{2,0}$-exp	72.22	0.40	2.34	0.05	**80.00**	**0.00**
	DCA-$\ell_{1,0}$-exp	72.25	0.38	0.26	0.01	**80.00**	**0.00**
	DCA-$\ell_{\infty,0}$-exp	72.22	0.40	9.79	0.10	**80.00**	**0.00**
	DCA-$\ell_{2,0}$-capℓ_1	72.25	0.52	0.32	0.00	**80.00**	**0.00**
	DCA-$\ell_{1,0}$-capℓ_1	72.24	0.52	**0.30**	**0.00**	**80.00**	**0.00**
	DCA-$\ell_{\infty,0}$-capℓ_1	72.24	0.51	3.00	0.00	**80.00**	**0.00**
	SPGD-$\ell_{2,1}$	71.48	0.81	7.16	0.91	83.50	2.52
	msgl	**72.33**	0.18	214.83	25.40	82.00	0.00

sim_2	SDCA-$\ell_{2,0}$-exp	68.53	0.29	0.79	0.00	**80.00**	**0.00**
$n = 150,000$	SDCA-$\ell_{1,0}$-exp	68.48	0.34	0.73	0.16	**80.00**	**0.00**
$d = 50$	SDCA-$\ell_{\infty,0}$-exp	**68.71**	**0.23**	0.97	0.12	**80.00**	**0.00**
$Q = 3$	SDCA-$\ell_{2,0}$-capℓ_1	68.50	0.29	1.02	0.14	**80.00**	**0.00**
	SDCA-$\ell_{1,0}$-capℓ_1	67.42	0.40	**0.71**	**0.23**	**80.00**	**0.00**
	SDCA-$\ell_{\infty,0}$-capℓ_1	68.38	0.28	1.40	0.18	**80.00**	**0.00**
	DCA-$\ell_{2,0}$-exp	68.55	0.22	1.14	0.26	**80.00**	**0.00**
	DCA-$\ell_{1,0}$-exp	68.31	0.23	13.51	1.93	**80.00**	**0.00**
	DCA-$\ell_{\infty,0}$-exp	**68.71**	**0.18**	2.75	2.80	**80.00**	**0.00**
	DCA-$\ell_{2,0}$-capℓ_1	67.70	0.31	4.29	0.03	**80.00**	**0.00**
	DCA-$\ell_{1,0}$-capℓ_1	68.43	0.24	0.93	0.16	**80.00**	**0.00**
	DCA-$\ell_{\infty,0}$-capℓ_1	67.49	0.35	0.69	0.10	**80.00**	**0.00**
	SPGD-$\ell_{2,1}$	67.62	0.48	7.77	0.28	82.00	0.00
	msgl	68.42	0.03	367.29	53.52	82.00	0.00
sim_3	SDCA-$\ell_{2,0}$-exp	99.69	0.04	36.61	1.48	**80.00**	**0.00**
$n = 250,000$	SDCA-$\ell_{1,0}$-exp	**99.93**	**0.01**	**10.74**	**0.42**	**80.00**	**0.00**
$d = 500$	SDCA-$\ell_{\infty,0}$-exp	99.56	0.07	22.11	3.43	80.73	0.64
$Q = 4$	SDCA-$\ell_{2,0}$-capℓ_1	99.69	0.01	21.45	0.93	**80.00**	**0.00**
	SDCA-$\ell_{1,0}$-capℓ_1	99.00	0.01	23.10	0.12	**80.00**	**0.00**
	SDCA-$\ell_{\infty,0}$-capℓ_1	99.67	0.01	21.05	1.06	**80.00**	**0.00**
	DCA-$\ell_{2,0}$-exp	99.88	0.02	249.74	10.73	**80.00**	**0.00**
	DCA-$\ell_{1,0}$-exp	99.88	0.02	202.67	33.27	**80.00**	**0.00**
	DCA-$\ell_{\infty,0}$-exp	97.74	2.05	431.13	26.47	**80.00**	**0.00**
	DCA-$\ell_{2,0}$-capℓ_1	99.92	0.01	178.89	7.83	**80.00**	**0.00**
	DCA-$\ell_{1,0}$-capℓ_1	99.87	0.01	270.69	17.64	**80.00**	**0.00**
	DCA-$\ell_{\infty,0}$-capℓ_1	99.85	0.03	24.40	4.29	80.40	0.40
	SPGD-$\ell_{2,1}$	99.70	0.12	212.71	21.79	**80.00**	**0.00**
	msgl	**99.93**	**0.01**	1581.44	14.76	80.20	0.00

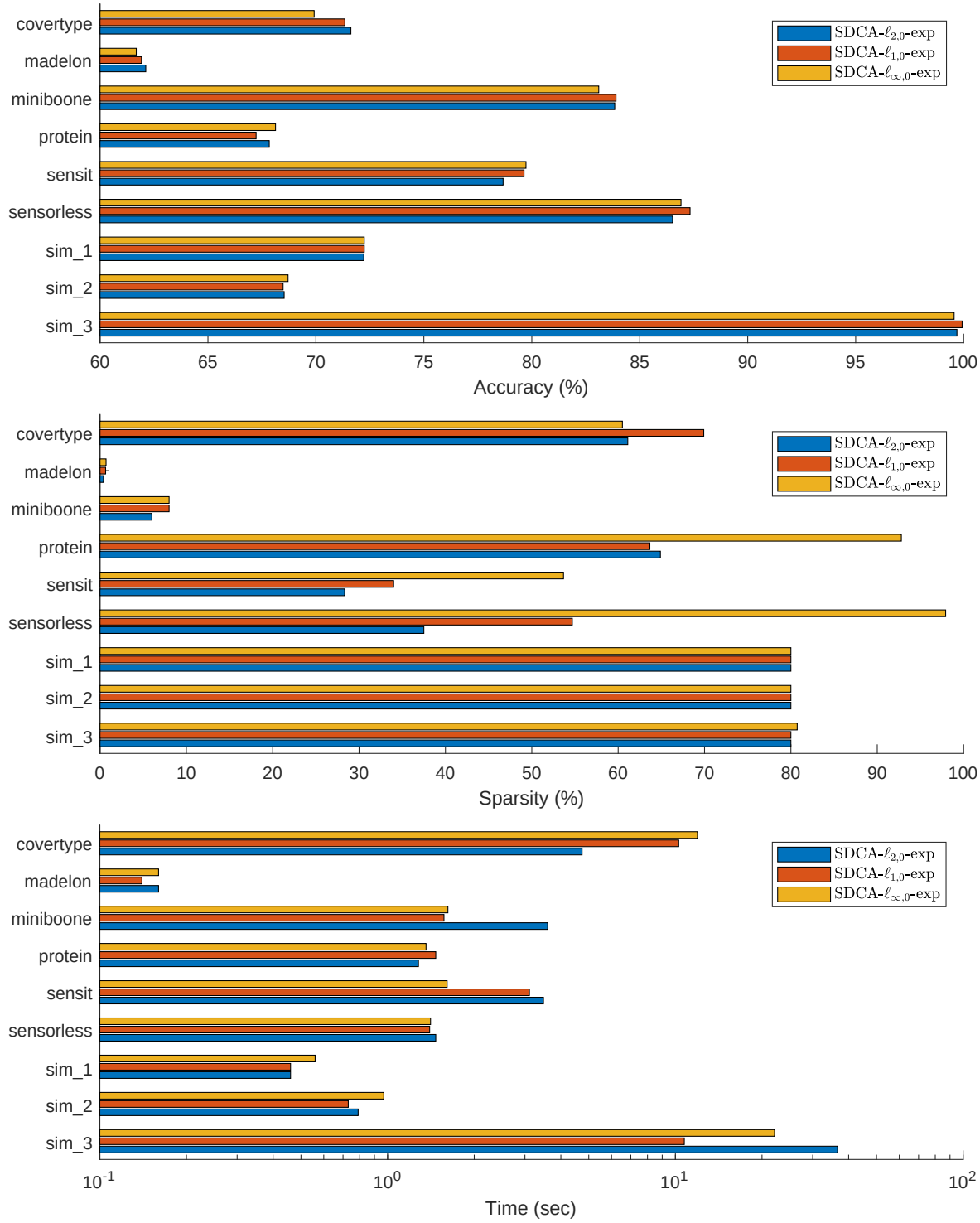


Figure 2.2 – Comparative results between `SDCA-`$\ell_{1,0}$`-exp`, `SDCA-`$\ell_{2,0}$`-exp` and `SDCA-`$\ell_{\infty,0}$`-exp` (running time is plotted on a logarithmic scale).

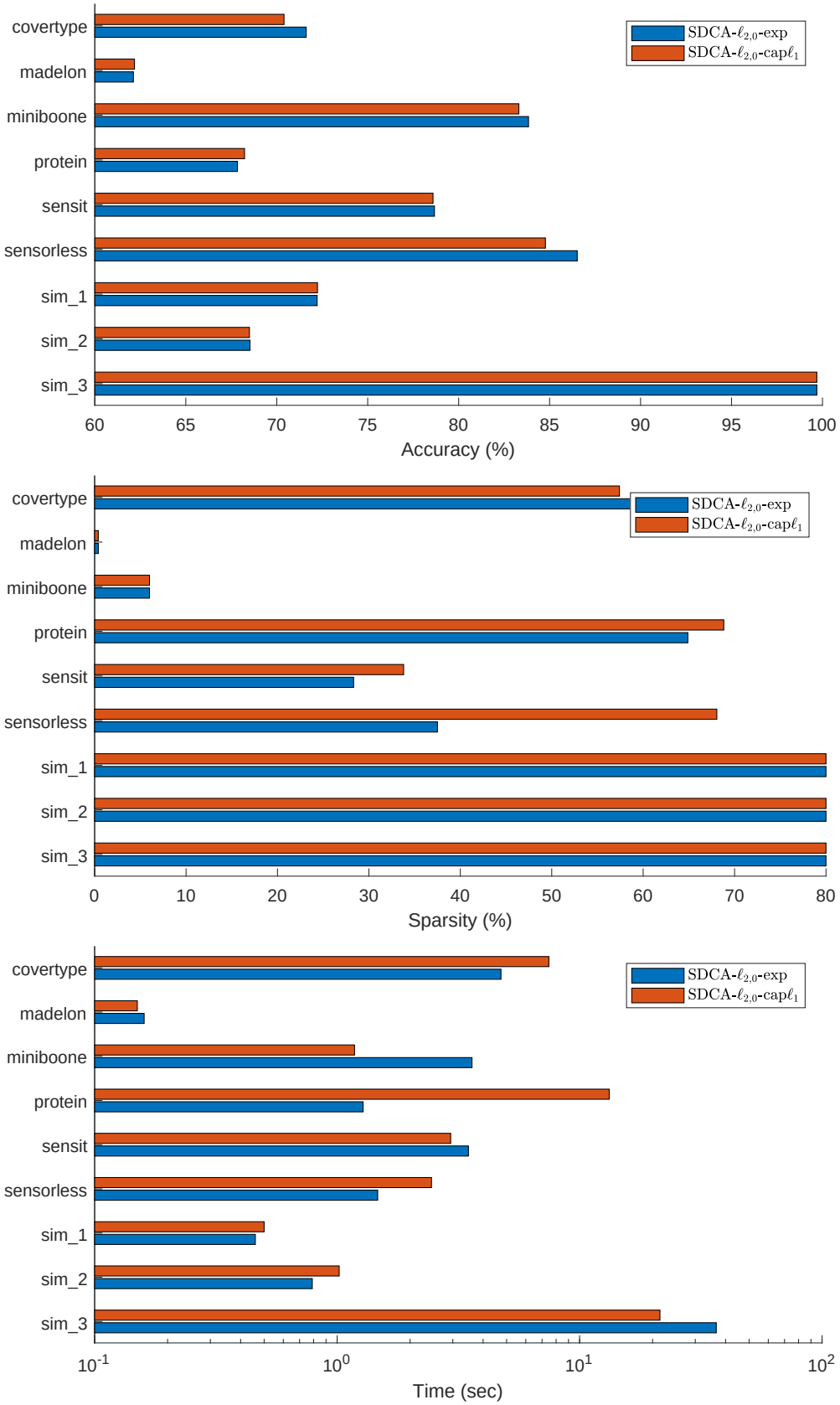


Figure 2.3 – Comparative results between `SDCA-`$\ell_{2,0}$`-exp` and `SDCA-`$\ell_{2,0}$`-cap`ℓ_1 (running time is plotted on a logarithmic scale).

2.4 DCA-Like and ADCA-Like for the group variable selection in multi-class logistic regression

Standard DCA scheme (Algorithm 2.1) requires computing a parameter μ. In practice, it is often estimate by estimating $\mu \geq L$ – the Lipschitz constant of $\mathcal{L}$, which leads to bad approximations of the objective function since μ is usually quite large. DCA-Like applied to the problem (2.3) eliminates the needs to compute μ while keeping μ small, which leads to closer majorant of the objective function and potentially to a better solution.

DCA-Like scheme for problem (2.4)**:** Applying DCA-Like to (2.4), we observe that the while loop in DCA-Like (Algorithm 1.3) stops if the following inequality holds

$$\mathbf{U}_{\mu_k}^{LR}((W,b)^{k+1},(W,b)^k) \geq F((W,b)^{k+1}), \tag{2.13}$$

where

$$\begin{aligned}\mathbf{U}_{\mu_k}^{LR}((W,b)^{k+1},(W,b)^k) = \quad & F((W,b)^k) + \langle \nabla f((W,b)^k), (W,b)^{k+1} - (W,b)^k \rangle \\ & + \tfrac{\mu_k}{2}\|(W,b)^{k+1} - (W,b)^k\|^2 - \langle o^k, g(W^{k+1}) - g(W^k)\rangle.\end{aligned}$$

Thus, DCA-Like for solving problem (2.4) is presented in Algorithm 2.4.

Algorithm 2.4 `DCA-Like`: DCA-Like for solving (2.4)

Initialize:
Choose $(W^0, b^0) \in \mathbb{R}^{d\times Q} \times \mathbb{R}^Q$, $\mu_0 > 0$ and $k \leftarrow 0$.
repeat
Compute $(U, v, o)^k$ by (2.7).
Set $\mu_k = \max\{\mu_0, \delta\mu_{k-1}\}$ if $k > 0$.
Compute $(W, b, z)^{k+1}$ according to Table 2.1, (2.9) and (2.10), respectively.
while $\mathbf{U}_{\mu_k}^{LR}((W,b,z)^{k+1},(W,b,z)^k) < F((W,b,z)^{k+1})$ **do**
$\mu_k \leftarrow \eta\mu_k$.
Compute $(W, b, z)^{k+1}$ according to Table 2.1, (2.9) and (2.10), respectively.
end while
$k \leftarrow k + 1$.
until Stopping criterion

ADCA-Like scheme for problem (2.4)**:** According to ADCA-Like scheme in Alogrithm (1.4), ADCA-Like for solving (2.4) is obtained by adding the acceleration step of Algorithm 1.4 to Algorithm 1.2. The alogrithm is described in 2.5.

2.4.1 Numerical experiment

2.4.1.1 Experiment setting

To evaluate the performances of algorithms, we performed numerical experiments on several benchmark datasets taken from UCI and LIBSVM data repositories. The

Algorithm 2.5 `ADCA-Like`: ADCA-Like for solving (2.4)

Initialize:
Choose $(W^0, b^0) \in \mathbb{R}^{d\times Q} \times \mathbb{R}^Q$, $\omega^0 = W^0$, $\beta^0 = b^0$,
$\eta > 1, 0 < \delta < 1$, $\mu_0 > 0$ and $k \leftarrow 0$.
repeat
if $F(\omega^k, \beta^k) \leq F(W^k, b^k)$ **then**
set $\mathbf{w}^k = \omega^k$ and $\mathbf{b}^k = \beta^k$
else
set $\mathbf{w}^k = W^k$ and $\mathbf{b}^k = b^k$
end if
Compute $(U^k, v^k) = \mu(\mathbf{w}^k, \mathbf{b}^k) - \frac{1}{n}\sum_{i=1}^n \nabla\ell(x_i, y_i, \mathbf{w}^k, \mathbf{b}^k)$
Compute $o_j^k \in \partial\left(-\lambda \sum_{j=1}^d \eta_\alpha(\|\mathbf{w}_{j,:}^k\|_2)\right)$ for $j = 1, \ldots, d$ (similar to (2.7)).
Set $\mu_k = \max\{\mu_0, \delta\mu_{k-1}\}$ if $k > 0$.
Compute $(W, b, z)^{k+1}$ according to Table 2.1, (2.9) and (2.10), respectively.
while $\mathbf{U}_{\mu_k}^{LR}((W, b, z)^{k+1}, (W, b, z)^k) < F((W, b, z)^{k+1})$ **do**
$\mu_k \leftarrow \eta\mu_k$.
Compute $(W, b, z)^{k+1}$ according to Table 2.1, (2.9) and (2.10), respectively.
end while
Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$.
Compute $(\omega, \beta)^{k+1} = (W, b)^{k+1} + \frac{t_k - 1}{t_{k+1}}\left((W, b)^{k+1} - (W, b)^k\right)$.
$k \leftarrow k + 1$.
until Stopping criterion

detailed information of used datasets is summarized in the first column of Table 2.3. n represents the number of points in dataset, d is the number of features, and Q is the number of classes. We randomly take 80% of the whole dataset as a training set and the rest is used as test set (20%).

The comparisons are performed between 6 algorithms: `DCA`, `DCA-Like`, `ADCA`, `ADCA-Like`, `nm-APG` (non-monotone APG) [69] and `DC-PN` (DC Proximal Newton) [91]. Recall that `nm-APG` is an accelerated proximal gradient based method for minimizing $f(\mathbf{x}) + r(\mathbf{x})$ where $f(\mathbf{x})$ is a differentiable function with L-Lipschitz gradient and $r(\mathbf{x})$ is a nonconvex function. `nm-APG` requires to compute the proximal mapping of the DC function η_α. However, this proximal mapping does not have a closed form. We therefore use DCA to compute the proximal mapping of η_α in `nm-APG`. `ADCA` is Accelerated DCA for solving problem (2.4), which is described in Algorithm 2.6.

For `DCA` and `ADCA`, the Lipschitz constant L is estimated by an upper bound of Hessian matrix of logistic loss function which clearly too large. Hence, similarly as in the first application, we incorporate a μ-updating procedure into `DCA` and `ADCA`. We set the initial value of μ to $\mu_0 = 10^{-1}$.

The parameter α for controlling the tightness of zero-norm approximation capped-ℓ_1 is set to 5. All the algorithms are terminated if the change of two consecutive objective function values is less than 10^{-5}. We also stop the algorithms after either

Algorithm 2.6 `ADCA`: Accelerated DCA for solving (2.4)

Initialize:
Choose $(W^0, b^0) \in \mathbb{R}^{d\times Q} \times \mathbb{R}^Q$, $\omega^0 = W^0$, $\beta^0 = b^0$,
$\eta > 1, 0 < \delta < 1$, $\mu_0 > L$ and $k \leftarrow 0$.
repeat
if $F(\omega^k, \beta^k) \leq F(W^k, b^k)$ **then**
set $\mathbf{w}^k = \omega^k$ and $\mathbf{b}^k = \beta^k$
else
set $\mathbf{w}^k = W^k$ and $\mathbf{b}^k = b^k$
end if
Compute $(U^k, v^k) = \mu(\mathbf{w}^k, \mathbf{b}^k) - \frac{1}{n}\sum_{i=1}^n \nabla\ell(x_i, y_i, \mathbf{w}^k, \mathbf{b}^k)$
Compute $o_j^k \in \partial\left(-\lambda \sum_{j=1}^d \eta_\alpha(\|\mathbf{w}_{j,:}^k\|_2)\right)$ for $j = 1, \ldots, d$ (similar to (2.7)).
Compute $(W, b, z)^{k+1}$ according to Table 2.1, (2.9) and (2.10), respectively.
Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$.
Compute $(\omega, \beta)^{k+1} = (W, b)^{k+1} + \frac{t_k - 1}{t_{k+1}}\left((W, b)^{k+1} - (W, b)^k\right)$.
$k \leftarrow k + 1$.
until Stopping criterion

one hour of running time or $10,000$ iterations.

In order to evaluate the performance of algorithms, we consider the following criteria: the classification accuracy (percentage of well-classified point on test set), the sparsity of obtained solution (percentage of selected features), the running time (measured in seconds) and the number of iterations. The numerical results are reported in Table 2.3. In Figure 2.4, we plot the curves of objective function values versus the running time.

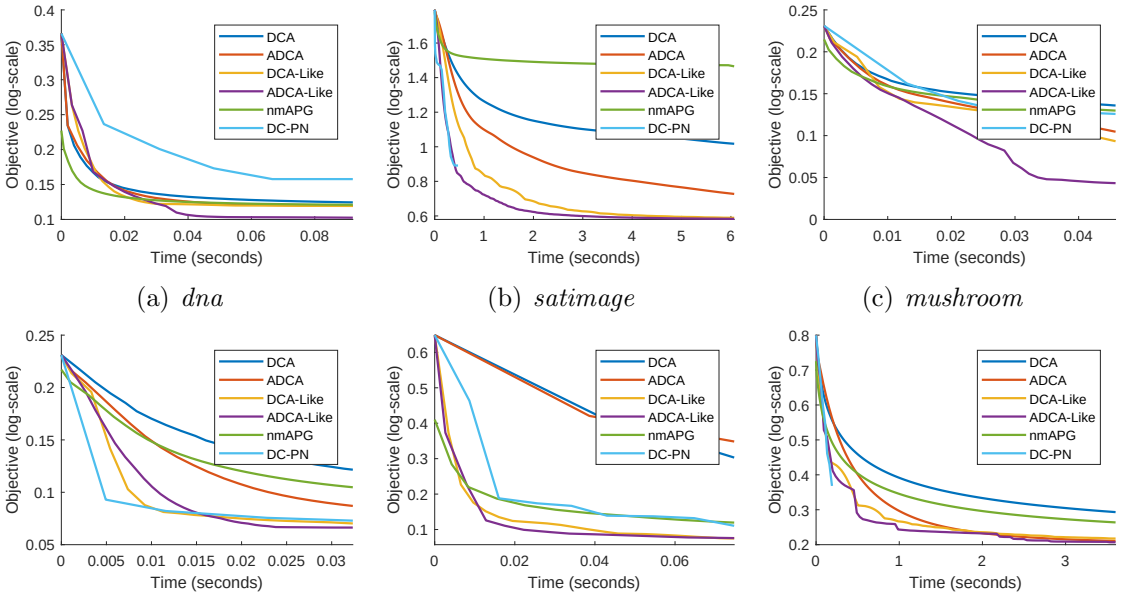
2.4.1.2 Comments on numerical results

Comparison between `DCA` and `DCA-Like`. Concerning the running time, `DCA-Like` is clearly faster than `DCA`, with the gains from 1.6 to 8.8 times. In term of classification accuracy and sparsity, `DCA-Like` is slightly better than `DCA`. In 4 cases (*satimage*, *mushroom*, *shuttle* and *sensorless*), `DCA-Like` achieves better classification accuracy (up to 1%) whereas choosing the same number of features. In the remaining 2 cases, the difference between them are neglectable.

Comparison between `ADCA` and `ADCA-Like`. In terms of classification accuracy, `ADCA-Like` is slightly better than `DCA-Like`. In five over six datasets (except *mushroom* dataset), `ADCA-Like` shows the superior over `DCA-Like` in both classification accuracy and sparsity: `ADCA-Like` yields higher classification accuracy while choosing a smaller subset of features.

Comparison between our algorithms and existing methods (nm-APG and DC-PN). In term of classification accuracy, `ADCA-Like` produces the best re-

sults in all six datasets. As for the sparsity of solution, `ADCA-Like` also furnishes good results: the best result on four datasets and the second-best result on the other two datasets. It is worth to mention that in two datasets (*sensorless* and *shuttle*), while most algorithms choose the same percentage of features, `ADCA-Like` has the highest classification accuracy. We also observe that `DC-PN` often chooses the biggest subset of features while giving lower classification accuracy. In term of computing time, `DCA-Like` and `ADCA-Like` are arguably faster than the other algorithms, whereas `nm-APG` is the slowest. Overall, both `ADCA-Like` and `ADCA` achieve better results than `nm-APG` and `DC-PN`.



(a) *dna* (b) *satimage* (c) *mushroom*

(d) *phishing* (e) *shuttle* (f) *sensorless*

Figure 2.4 – Objective value versus running time (average of ten runs)

2.5 Comparison between proposed algorithms

This section focuses on evaluating the performance of three proposed algorithms (`SDCA`, `DCA-Like` and `ADCA-Like`) for high-dimensional datasets.

Datasets: We performed numerical experiments on two types of data: real datasets (*CLL_SUB_111*, *Carcinom*, *lung*, *ORL*, *BASEHOCK*, *protein* and *miniboone*) and simulated datasets (*sim_1* and *sim_3*). All real-world datasets are taken from the well-known UCI, LibSVM and Feature Selection Datasets [7] data repositories. Two simulated datasets (*sim_1* and *sim_3*) are generated from the same procedures in section 2.3.1.1.

Experiment setting Unless stated, this experiment's settings are the same as in section 2.3.1. We employ grid search for $\lambda \in \{10^4, 3 \times 10^3, 10^3, \ldots, 3 \times 10^{-3}, 10^{-3}\}$.

7. `http://featureselection.asu.edu/datasets.php`

All experiments are performed on a PC Intel (R) i5-6300HQ CPU @2.30 GHz (4 CPUs) with 8GB RAM.

The numerical experiment is reported in Table 2.4. We notice that:

— The classification accuracy of `DCA-Like` and `ADCA-Like` are similar ($< 0.1\%$ in 8 over 9 cases); and both are slightly higher than `SDCA` by up to 0.5% (in `protein` and `sim_1`). Overall, the differences are neglectable.

— Similarly, in terms of sparsity, `DCA-Like` and `ADCA-Like` often choose the least number of variables, and is much smaller than `SDCA`. The gap between `DCA-Like` (reps. `ADCA-Like`) and `SDCA` is up to 9%, which is equivalent to 2 to 6 times bigger subset of features in some datasets (*CLL_SUB_111*, *ORL* and *miniboone*).

— In terms of running time, it is clear that `SDCA` is faster than `DCA`, especially for large dataset. Except the three smallest datasets (*CLL_SUB_111*, *Carcinom* and *lung*) where `DCA-Like` (reps. `ADCA-Like`) is faster than `SDCA`; `SDCA` is faster. The reduction in running time is from 1.3 to 8 times; especially in the two biggest datasets where `SDCA` is faster than the second fastest ones by 4 and 8 times respectively.

In conclusion, `ADCA-Like` is faster than `DCA-Like`, while having similar classification accuracy and sparsity. `SDCA` is much faster than `ADCA-Like` (reps. `DCA-Like`) while having similar accuracy; however `ADCA-Like` selects more compact subset of variables than `SDCA`.

2.6 Conclusion

We have rigorously studied the DC programming and DCA for the problem of group variables selection in multi-class logistic regression. Using the $\ell_{q,0}$ regularization, the resulting optimization problem is non-convex. The $\ell_{q,0}$-norm is approximated by piecewise exponential function and capped-ℓ_1 function. We have developed DCA-based algorithms to solve it by two approaches based on recent advanced DC algorithms.

The first approach developed an algorithm for multi-class logistic regression based on Stochastic DCA. The proposed algorithm, `SDCA`, is very inexpensive and suitable for large-scale datasets. `SDCA` exploits the special structure of the problem: not only all computation of `SDCA` are explicit, but `SDCA` also only requires a small subset of data at each iteration for computing the subgradient of the second DC component, which significantly reduces the running time. Numerical results showed that the proposed algorithm reduces the running time of `DCA` (standard DCA for this problem) by more than 19 times while achieving equivalent classification accuracy. Moreover, `SDCA` outperforms the two related algorithms (`msgl` and `SPGD-`$\ell_{2,1}$): it reduces the running time by more than 15 times while giving more accurate result. We are convinced that `SDCA` is fast and scalable method for the problem of group variable selection in multi-class logistic regression.

In the second approach, we developed two algorithms for the multi-class logistic regression with group variables selection, `DCA-Like` and `ADCA-Like`. `DCA-Like` elimi-

nates the needs to compute μ, and iterative updating μ while keeping μ small, hence it potentially reduces the running time with better solution in compare with standard DCA. `ADCA-Like` improves `DCA-Like` by in-cooperating Nesterov's acceleration. Both algorithms are efficient, all computation are explicit. `DCA-Like` and `ADCA-Like` improves the running time of `DCA` (by up to 8 times and 18.6 times respectively) while having equivalent classification accuracy and sparsity. In addition, both of them deliver better results than the two related algorithms (`nm-APG` and `DC-PN`) in both term of classification accuracy and running time. We can undoubtedly conclude that `DCA-Like` and `ADCA-Like` improves DCA in this application.

In comparison between two proposed approaches, `SDCA` is more suitable for large datasets as it is faster while achieving similar classification accuracy. In contrasts, `ADCA-Like` and `DCA-Like` is more suitable if we prefer a more compact subset of variables. And `ADCA-Like` is better than `DCA-Like` while achieving the similar classification accuracy and sparsity. Hence, the choice for one method is depend on the compactness of the solution and the size of the dataset. For small dataset, the suitable choice is clearly `ADCA-Like`; whereas the choice for large dataset is based on the trade-off between the compactness of the solution versus training time.

Table 2.3 – Comparative results on group variable selection for multi-class logistic regression. Bold values correspond to best results for each dataset. NA means that the algorithm fails to furnish a result. n, d and Q are the number of instances, dimensions and classes respectively. Unit of time is second.

Dataset	Algorithm	Accuracy (%)		Time (sec.)		Sparsity (%)	
		Mean	STD	Mean	STD	Mean	STD
dna	DCA	93.41	0.54	3.95	0.19	8.89	0.56
$n = 3186$	ADCA	93.30	0.18	**0.60**	0.02	8.89	0.56
$d = 180$	DCA-Like	93.20	0.74	0.98	0.13	**7.78**	**0.96**
$Q = 3$	ADCA-Like	**93.88**	0.83	0.92	0.05	**7.78**	2.00
	nm-APG	93.30	0.65	3.30	0.05	8.52	0.32
	DC-PN	93.56	0.16	1.20	0.13	29.07	2.51
satimage	DCA	84.25	0.20	4.02	1.15	**44.44**	2.78
$n = 6435$	ADCA	83.61	0.95	2.90	2.58	62.04	25.66
$d = 36$	DCA-Like	84.51	0.25	0.46	0.05	**44.44**	2.78
$Q = 6$	ADCA-Like	**84.67**	0.43	**0.28**	0.03	49.07	1.60
	nm-APG	81.84	0.70	1.98	0.05	100.00	0.00
	DC-PN	78.63	0.88	4.24	0.26	47.22	16.90
mushroom	DCA	98.44	0.30	5.71	0.08	4.50	0.00
$n = 8124$	ADCA	98.24	0.30	0.48	0.03	6.31	3.12
$d = 112$	DCA-Like	**99.41**	**0.22**	0.25	0.02	4.50	0.00
$Q = 2$	ADCA-Like	**99.41**	**0.22**	0.20	0.09	**3.60**	0.00
	nm-APG	98.44	0.30	12.07	0.47	4.50	0.00
	DC-PN	97.81	0.66	**0.17**	0.04	27.93	1.80
phishing	DCA	92.52	0.18	2.60	0.08	33.82	2.55
$n = 11055$	ADCA	92.64	0.07	0.38	0.01	28.92	0.85
$d = 68$	DCA-Like	92.36	0.32	0.19	0.01	27.94	1.47
$Q = 2$	ADCA-Like	**92.66**	0.15	**0.14**	0.01	**27.45**	0.85
	nm-APG	92.40	0.32	6.95	0.18	28.43	1.70
	DC-PN	92.66	0.11	0.38	0.03	61.76	5.30
shuttle	DCA	95.97	0.11	16.50	5.13	**59.26**	6.42
$n = 58000$	ADCA	96.11	0.26	15.93	4.43	**59.26**	6.42
$d = 9$	DCA-Like	96.06	0.05	1.81	0.20	**59.26**	6.42
$Q = 7$	ADCA-Like	**96.13**	0.08	**1.43**	0.21	**59.26**	6.42
	nm-APG	96.06	0.05	21.27	0.30	66.67	0.00
	DC-PN	92.56	0.61	3.04	0.73	92.59	6.42
sensorless	DCA	78.55	0.45	58.76	0.15	**12.50**	0.00
$n = 58509$	ADCA	79.03	0.44	12.47	0.27	**12.50**	0.00
$d = 54$	DCA-Like	79.51	0.41	55.10	1.32	**12.50**	0.00
$Q = 11$	ADCA-Like	**79.56**	0.36	36.02	0.79	**12.50**	0.00
	nm-APG	78.72	0.41	46.53	0.86	**12.50**	0.00
	DC-PN	77.64	1.20	**8.43**	1.11	78.47	12.73

Table 2.4 – Comparative results on group variable selection for multi-class logistic regression. Bold values correspond to best results for each dataset. NA means that the algorithm fails to furnish a result. n, d and Q are the number of instances, dimensions and classes respectively. Unit of time is second.

Dataset	Algorithm	Accuracy (%)		Time (sec.)		Sparsity (%)	
		Mean	STD	Mean	STD	Mean	STD
CLL_SUB_111	DCA-Like	**78.79**	**6.94**	0.75	0.03	0.32	0.18
$n \times d = 111 \times 11340$	ADCA-Like	**78.79**	**6.94**	**0.63**	**0.06**	**0.18**	**0.04**
$Q = 3$	SDCA	**78.79**	**5.25**	5.25	1.64	1.95	1.09
Carcinom	DCA-Like	89.52	8.25	3.06	0.05	**0.41**	**0.01**
$n \times d = 174 \times 9182$	ADCA-Like	**90.48**	**6.60**	**1.81**	**0.11**	**0.41**	**0.01**
$Q = 11$	SDCA	**90.48**	**6.60**	4.16	2.74	0.66	0.25
lung	DCA-Like	**91.06**	**3.73**	0.71	0.01	**0.47**	**0.02**
$n \times d = 203 \times 3312$	ADCA-Like	**91.06**	**3.73**	**0.42**	**0.08**	0.50	0.02
$Q = 5$	SDCA	**91.06**	**3.73**	0.86	0.10	0.57	0.11
ORL	DCA-Like	**94.58**	**5.05**	10.76	0.56	**6.02**	**0.31**
$n \times d = 400 \times 1024$	ADCA-Like	**94.58**	**2.89**	7.43	0.76	7.71	0.34
$Q = 40$	SDCA	**94.58**	**5.05**	**3.04**	**1.38**	16.76	6.69
BASEHOCK	DCA-Like	94.65	1.26	5.72	2.53	4.24	0.87
$n \times d = 1993 \times 4862$	ADCA-Like	94.65	1.38	5.47	1.51	**2.28**	**0.69**
$Q = 2$	SDCA	**94.74**	**0.25**	**3.54**	**0.28**	2.73	0.16
protein	DCA-Like	68.26	0.50	2.92	0.43	**77.90**	**2.03**
$n \times d = 24387 \times 357$	ADCA-Like	**68.29**	**0.51**	4.16	0.90	78.00	2.13
$Q = 3$	SDCA	67.94	0.62	**1.17**	**0.35**	80.43	0.58
sim_1	DCA-Like	**72.22**	**0.50**	0.66	0.00	**80.00**	**0.00**
$n \times d = 100000 \times 50$	ADCA-Like	**72.22**	**0.50**	0.63	0.02	**80.00**	**0.00**
$Q = 4$	SDCA	72.18	0.57	**0.49**	**0.06**	**80.00**	**0.00**
miniboone	DCA-Like	83.78	0.07	4.82	0.42	**6.00**	**0.00**
$n \times d = 130064 \times 50$	ADCA-Like	83.78	0.07	3.68	0.60	**6.00**	**0.00**
$Q = 2$	SDCA	**84.25**	**0.34**	**0.92**	**0.23**	11.33	1.15
sim_3	DCA-Like	**99.93**	**0.01**	358.40	7.38	**80.00**	**0.00**
$n \times d = 250000 \times 500$	ADCA-Like	**99.93**	**0.01**	159.95	16.26	**80.00**	**0.00**
$Q = 4$	SDCA	**99.93**	**0.02**	**17.86**	**3.28**	**80.00**	**0.00**

Chapter 3

t-distributed Stochastic Neighbor Embedding [1]

Abstract: *t-distributed Stochastic Neighbor Embedding (t-SNE) is a popular method for dimensional reduction and data visualization, especially for high-dimensional data. In this chapter, we developed two alogrithms based on advanced DCAs – DCA-Like and ADCA-Like – for the t-SNE problem. DCA-Like relaxes the convexity condition of the second DC component while ensuring the convergence; whereas ADCA-Like in-cooperates the Nesterov's acceleration technique into DCA-Like, which potentially leads to better solution in comparison to DCA-Like. It turns out that Majorization Minimization, the best state-of-the-art algorithm for t-SNE, is DCA-Like applied for t-SNE. Numerical experiments on several benchmark datasets for visualizing high-dimensional data illustrate the efficiency of our algorithms.*

1. The material of this chapter is developed from the following work:
[1] H.A. Le Thi, H.M. Le, D.N. Phan, B. Tran. *Novel DCA Based Algorithms for Minimizing the Sum of a Nonconvex Function and Composite Functions with Applications in Machine learning. Submitted & Available on arvXiv [arXiv:1806.09620].*

3.1 Introduction

Dimensional reduction for high-dimensional data, especially in data visualization, is an important task in data mining. Due to the nature of high-dimensional data where only pairwise distances are reliable, many techniques exploit this property by trying to preserve the small pairwise distance in low-dimensional embedding. Among them, t-SNE is a gaining popular method from the family of stochastic neighbor embedding (SNE) methods [34], operates by retaining local pairwise distances. t-SNE was first introduced in [106] as a visualization technique for high dimensional data. Later, it has been applied in many applications such as bioinformatic [114], cancer research, visualize features in neural networks [77], etc.

The t-SNE problem is formulated as the minimization of the divergence between two distributions: (1) a distribution that measures pairwise similarities of the input objects and (2) a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding. This problem is described as follows. Given a data set of n objects $\mathcal{D} = \{\mathbf{a}_1, ..., \mathbf{a}_n\}$ with $\mathbf{a}_i \in \mathbb{R}^d$, t-SNE aims to find their low-dimensional representation $\mathcal{E} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^s$ $(d \gg s)$ which preserves the pairwise similarities of input objects. To this end, t-SNE defines joint probabilities p_{ij} that measure the pairwise similarity between objects $\mathbf{x}_i$ and $\mathbf{x}_j$ by symmetrizing two conditional probabilities as $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$, where

$$p_{j|i} = \begin{cases} \frac{\exp(-\|\mathbf{a}_i - \mathbf{a}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{a}_i - \mathbf{a}_k\|^2 / 2\sigma_i^2)} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

In the embedding subspace $\mathcal{E}$, the similarities between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ are measured using a normalized heavy-tailed kernel. Specifically, the embedding similarity q_{ij} between the two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is computed as a normalized Student-t kernel with a single degree of freedom:

$$q_{ij} = \begin{cases} \frac{(1 + \|\mathbf{x}_i - \mathbf{x}_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|\mathbf{x}_k - \mathbf{x}_l\|^2)^{-1}} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

The locations of the embedding points $\mathbf{x}_i$ are determined by minimizing the Kullback-Leibler divergence between the joint distributions P and Q:

$$\min_{\mathbf{x}} \left\{ \mathrm{KL}(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \right\}, \tag{3.1}$$

which is equivalent to

$$\min_{\mathbf{x}} \left\{ F(\mathbf{x}) = f(\mathbf{x}) + \sum_{i,j} p_{ij} \log(1 + \|\mathbf{x}_i - \mathbf{x}_j\|^2) \right\}, \tag{3.2}$$

where $f(\mathbf{x}) = \log(\sum_{i \neq j} (1 + \|\mathbf{x}_i - \mathbf{x}_j\|^2)^{-1})$.

The nonconvex optimization problem (3.1) has been studied in several works [106, 122, 110], but the most noticeable was presented in [123]. Yang et al. [123] presented and compared Majorization Minimization algorithm (MM) with five state-of-the-arts methods, such as gradient descent, gradient descent with momentum [106], spectral direction [110], FPHSSNE [122] and Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [81]. The numerical results in Yang et al. [123] showed that MM for t-SNE outperforms all five state-of-the-art optimization methods.

Our contributions

This chapter investigates DC programming and DCA to solve the t-SNE problem. First, we study the standard DCA for the t-SNE problem (3.2). As shown in Chapter 2, the main advantage of DCA-Like (reps. ADCA-Like) over standard DCA is using the DCA-Like condition (1.16), which leads to closer majorant of the objective function and potentiality a better solution. Hence, we develop two algorithms by DCA-Like and ADCA-Like for the t-SNE problem. It turns out that Majorization Minimization [123], the best state-of-the-art algorithm for t-SNE, is nothing else but DCA-Like applied for t-SNE. In addition, its accelerated version – ADCA-Like – further reduces the running time of DCA-Like. We carefully conduct the numerical experiments and provide a comparison of proposed algorithms on several benchmark datasets.

The remainder of this chapter is organized as follows. The solution method based on DCA, DCA-Like and ADCA-Like is developed in section 3.2. The numerical experiments on the t-SNE problem are reported in section 3.4. Finally, section 3.5 concludes the chapter.

3.2 Standard DCA for t-SNE problem

A DC formulation for problem (3.1)**:**

Let us rewrite the problem (3.1) from (3.2) into

$$\min_{\mathbf{x}\in X}\left\{F(x) := f(\mathbf{x}) + \sum_{i,j} h_{ij}(g_{ij}(\mathbf{x}))\right\}, \tag{3.3}$$

where $f(\mathbf{x}) = \log(\sum_{i\neq j}(1+\|\mathbf{x}_i-\mathbf{x}_j\|^2)^{-1})$, $h_{ij}(t) = p_{ij}\log(1+t)$ and $g_{ij}(\mathbf{x}) = \|\mathbf{x}_i-\mathbf{x}_j\|^2$. It is obvious that g_{ij} are convex functions, and h_{ij} are concave increasing functions whose derivatives are non-negatives and Lipschitz continuous on $[0,+\infty)$. Moreover, the function $f(\mathbf{x})$ is differentiable with L-Lipschitz continuous gradient as shown in the following proposition.

Proposition 3.1. *The function* $f(\mathbf{x}) = \log(\sum_{i\neq j}(1+\|\mathbf{x}_i-\mathbf{x}_j\|^2)^{-1})$ *is smooth with Lipschitz gradient, where we can choose a Lipschitz constant* $L = 6n\sqrt{s}$.

Therefore, the optimization problem (3.3) takes the form of (1.10), where we can use the DC decomposition in (1.12).

DCA scheme for problem (3.1)**:**

Since the optimization problem (3.3) takes the form of (1.10), then according to DCA scheme (1.2), at each iteration, we have to compute

$$\xi_{ij}^k = \nabla(-h_{ij})(g_{ij}(\mathbf{x}_i^k, \mathbf{x}_j^k)) = -\frac{p_{ij}}{1 + \|\mathbf{x}_i^k - \mathbf{x}_j^k\|^2}$$

and $\nabla f(\mathbf{x}^k)$ by

$$\nabla_{\mathbf{x}_i} f(\mathbf{x}^k) = \sum_{j=1}^{n} \frac{-4(\mathbf{x}_i^k - \mathbf{x}_j^k)(1 + \|\mathbf{x}_i^k - \mathbf{x}_j^k\|^2)^{-2}}{\sum_{l \neq m}(1 + \|\mathbf{x}_l^k - \mathbf{x}_m^k\|^2)^{-1}}, \tag{3.4}$$

and solve the following convex problem to compute $\mathbf{x}^{k+1}$

$$\min_{\mathbf{x}} \left\{ \frac{\mu_k}{2}\|\mathbf{x} - \mathbf{x}^k\|^2 + \langle \nabla f(\mathbf{x}^k), \mathbf{x} \rangle + \sum_{i,j} -\xi_{ij}^k \|\mathbf{x}_i - \mathbf{x}_j\|^2 \right\}.$$

The solution $\mathbf{x}^{k+1}$ of the above problem is given by

$$\mathbf{x}^{k+1} = (2\mathcal{L}_{-\xi^k - (\xi^k)^T} + \mu_k I)^{-1}(-\nabla f(\mathbf{x}^k) + \mu_k \mathbf{x}^k), \tag{3.5}$$

where the matrix ξ^k is defined by the elements ξ_{ij}^k and $\mathcal{L}_A$ denotes the matrix with $(\mathcal{L}_A)_{ij} = -A_{ij}$ if $i \neq j$ and $-A_{ii} + \sum_{l=1}^{n} A_{il}$ otherwise.

The standard DCA for solving the t-SNE problem (3.1) is presented in Algorithm 3.1.

Algorithm 3.1 `DCA`: DCA for solving t-SNE problem (3.1)

Initialize:
Choose $\mathbf{x}^0$, $\mu_0 > L$ and $k \leftarrow 0$.
repeat
Compute $\xi_{ij}^k = -\frac{p_{ij}}{1+\|\mathbf{x}_i^k - \mathbf{x}_j^k\|^2}$ and $\nabla f(\mathbf{x}^k)$ by (3.4).
Compute $\mathbf{x}^{k+1}$ by (3.5).
$k \leftarrow k + 1$.
until Stopping condition

3.3 DCA-Like and ADCA-Like for t-SNE problem

The `DCA` (Algorithm 3.1) requires setting $\mu_0 \geq L$, where we have estimated $L = 6n\sqrt{s}$. This value is clearly too large, which could lead to a bad convex approximation of F, then potentially a bad solution. Hence, we consider DCA-Like to solve this problem. As we have mentioned in Section 1.3.2, the goal of DCA-Like is to avoid bad approximations of the objective function with a too large value of μ (c.f.

Algorithm 1.2). The main idea of DCA-Like is to keep the parameter μ as small as possible while finding a convex approximation of objecive function. By using a small value of μ_k, we can get a closer majorant of the objective function which could lead to a better solution. According to the Algorithm 1.3, stops if the following inequality holds

$$\mathbf{U}_{\mu_k}^{tsne}(\mathbf{x}^{k+1}, \mathbf{x}^k) \geq F(\mathbf{x}^{k+1}), \tag{3.6}$$

where $\mathbf{U}_{\mu_k}^{tsne}(\mathbf{x}^{k+1}, \mathbf{x}^k) = F(\mathbf{x}^k) + \langle \nabla f(\mathbf{x}^k), \mathbf{x}^{k+1} - \mathbf{x}^k \rangle + \frac{\mu_k}{2}\|\mathbf{x}^{k+1} - \mathbf{x}^k\|^2 - \langle \xi^k, g(\mathbf{x}^{k+1}) - g(\mathbf{x}^k) \rangle$. From the update rule (3.5) for $\mathbf{x}^{k+1}$ and this stopping criterion for searching μ_k, we can conclude that MM [123] for (3.1) is a special version of DCA-Like. DCA-Like for solving t-SNE problem (3.1) is described in Algorithm 3.2.

Algorithm 3.2 `DCA-Like`: DCA-Like for solving t-SNE problem (3.1)

Initialize:
Choose $\mathbf{x}^0$, $\eta > 1, 0 < \delta < 1$, $\mu_0 > 0$ and $k \leftarrow 0$.
repeat
Compute $\xi_{ij}^k = -\frac{p_{ij}}{1+\|\mathbf{x}_i^k - \mathbf{x}_j^k\|^2}$ and $\nabla f(\mathbf{x}^k)$ by (3.4).
Set $\mu_k = \max\{\mu_0, \delta\mu_{k-1}\}$ if $k > 0$.
Compute $\mathbf{x}^{k+1}$ by (3.5).
while $\mathbf{U}_{\mu_k}^{tsne}(\mathbf{x}^{k+1}, \mathbf{x}^k) < F(\mathbf{x}^{k+1})$ **do**
$\mu_k \leftarrow \eta\mu_k$.
Compute $\mathbf{x}^{k+1}$ by (3.5).
end while
$k \leftarrow k + 1$.
until Stopping criterion

ADCA-Like for solving (3.1) is obtained by adding the acceleration steps of Algorithm 1.4 to Algorithm 3.2, and is presented in Algorithm 3.3.

3.4 Numerical experiment

Experiment setting

To evaluate the performances of our methods, we perform numerical experiments on 13 datasets taken from UCI data repository [2] and Feature Selection repository [3]. *20news* and *rcv1* dataset are downloaded and pre-processed using the same detailed procedure in [26, 117].

The comparison is realized on three criteria: the objective value $F(\mathbf{x})$, the number of iterations and the computation time (measured in seconds). Each experiment is repeated 10 times, then the final result is the average value of each criterion. The results are reported in the Table 3.1.

2. `http://archive.ics.uci.edu/ml/index.php`
3. `http://featureselection.asu.edu/datasets.php`

Algorithm 3.3 `ADCA-Like`: Accelerated DCA-Like for solving t-SNE problem (3.1)

Initialize:
Choose $\mathbf{x}^0$, $\mathbf{w}^0 = \mathbf{x}^0$, $\eta > 1, 0 < \delta < 1$, $\mu_0 > 0$ and $k \leftarrow 0$.
repeat
if $F(\mathbf{w}^k) \leq F(\mathbf{x}^k)$ **then**
set $\mathbf{v}^k = \mathbf{w}^k$;
else
set $\mathbf{v}^k = \mathbf{x}^k$.
end if
Compute $\xi_{ij}^k = -\frac{p_{ij}}{1+\|\mathbf{v}_i^k - \mathbf{v}_j^k\|^2}$ and $\nabla f(\mathbf{v}^k)$ by (3.4).
Set $\mu_k = \max\{\mu_0, \delta\mu_{k-1}\}$ if $k > 0$.
Compute $\mathbf{x}^{k+1}$ by (3.5).
while $\mathbf{U}_{\mu_k}^{tsne}(\mathbf{x}^{k+1}, \mathbf{x}^k) < F(\mathbf{x}^{k+1})$ **do**
$\mu_k \leftarrow \eta\mu_k$.
Compute $\mathbf{x}^{k+1}$ by (3.5).
end while
Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$.
Compute $\mathbf{w}^{k+1} = \mathbf{x}^{k+1} + \frac{t_k - 1}{t_{k+1}}\left(\mathbf{x}^{k+1} - \mathbf{x}^k\right)$.
$k \leftarrow k + 1$.
until Stopping criterion

Algorithm 3.4 `ADCA`: Accelerated DCA for solving t-SNE problem (3.1)

Initialize:
Choose $\mathbf{x}^0$, $\mathbf{w}^0 = \mathbf{x}^0$, $\eta > 1, 0 < \delta < 1$, $\mu_0 > L$ and $k \leftarrow 0$.
repeat
if $F(\mathbf{w}^k) \leq F(\mathbf{x}^k)$ **then**
set $\mathbf{v}^k = \mathbf{w}^k$;
else
set $\mathbf{v}^k = \mathbf{x}^k$.
end if
Compute $\xi_{ij}^k = -\frac{p_{ij}}{1+\|\mathbf{v}_i^k - \mathbf{v}_j^k\|^2}$ and $\nabla f(\mathbf{v}^k)$ by (3.4).

Compute $\mathbf{x}^{k+1}$ by (3.5).
Compute $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$.
Compute $\mathbf{w}^{k+1} = \mathbf{x}^{k+1} + \frac{t_k - 1}{t_{k+1}}\left(\mathbf{x}^{k+1} - \mathbf{x}^k\right)$.
$k \leftarrow k + 1$.
until Stopping criterion

The comparisons are performed between 5 algorithms: `DCA`, `DCA-Like`, `ADCA`, `ADCA-Like` and `DC-PN` (DC Proximal Newton [91]). `DC-PN` is a DCA-based algorithm for minimizing $f(x) + h(x)$, where $f(x) = f_1(x) - f_2(x)$ is a DC function, twice differentiable, with $f(x)$ verifying the L-Lipschitz gradient property; $h(x) = h_1(x) - h_2(x)$ where both h_1 and h_2 are convex functions and (possibly) non-differential. Note that, in `DC-PN`, L-BFGS is employed for approximating the Hessian matrix H_k; and

the sub-problem is solved by `minFunc` solver [4]. `ADCA`, the accelerate version of Algorithm 3.1 [89], is described in Algorithm 3.4.

As mentioned before, in the `DCA` (Algorithm 3.1) and `ADCA` (Algorithm 3.4), we have to estimate the L-Lipschitz constant of f. According to Proposition 3.1, we can choose $L = 6n\sqrt{s}$. This value is clearly too large. Hence, we incorporate a μ updating procedure into `DCA` and `ADCA`. We start with a small value of μ and increase μ if the objective value increases in `DCA`/`ADCA` scheme. For all algorithms, the initial value of μ is set to be $\mu_0 = 10^{-6}$.

Stopping conditions of all algorithms are the same, by either (1) number of iterations exceeds $10,000$ or (2) $\|\mathbf{x}^k - \mathbf{x}^{k-1}\|/\|\mathbf{x}^{k-1}\| \leq 10^{-8}$. Throughout our experiment, the number of embedding dimension is set to $s = 2$. All experiments are performed on a PC Intel (R) Xeon (R) E5-2630 v4 @2.20 GHz of 32GB RAM.

For large datasets, Barnes-Hut tree approximation is used for reducing computing cost [74]. This technique is well-known in Neighbor Embedding problems, which provides a good trade-off small loss in gradients and cost function against huge reduction in computation time. We set the parameter $\theta_{\text{Barnes-Hut}} = 0.5$.

We adopt the same test process as described in [123]. For all datasets, k-Nearest Neighbor (with $k = 10$) is employed to construct $\bar{p}_{ij}$, where $\bar{p}_{ij} = 1$ if data point j (reps. i) is one of k nearest neighbors of data point j (reps. i), and $\bar{p}_{ij} = 0$ otherwise. p_{ij} is then computed by $p_{ij} = \frac{\bar{p}_{ij}}{\sum_{k,l}\bar{p}_{kl}}$. The initial point $\mathbf{x}^0$ is drawn from normal distribution $\mathcal{N}(0, 10^{-8})$ for all methods. Early exaggeration technique [106] is deployed for first 20 iterations with the constant value of 4.

Table 3.1 – Comparative results on datasets. Bold values correspond to best results for each dataset, n and d are the number of instances and dimensions respectively. Unit of time is second.

Dataset	Algorithm	Objective		Iteration		Time (sec.)	
		Mean	STD	Mean	STD	Mean	STD
tox_171	`DCA`	1.52	0.03	**22**	**0**	0.64	0.01
$n = 171$	`ADCA`	1.62	0.64	37	22	1.95	1.38
$d = 5748$	`DCA-Like` (MM)	**1.25**	**0.00**	46	3	2.92	0.21
	`ADCA-Like`	**1.25**	**0.00**	49	21	3.54	1.40
	`DC-PN`	**1.25**	**0.00**	83	5	**0.62**	**0.04**
orl	`DCA`	1.86	0.20	**23**	**2**	5.50	4.77
$n = 400$	`ADCA`	**1.59**	**0.00**	59	23	15.10	3.59
$d = 1024$	`DCA-Like` (MM)	**1.59**	**0.00**	54	13	17.45	3.49
	`ADCA-Like`	**1.59**	**0.01**	68	30	20.43	8.18
	`DC-PN`	1.62	0.01	177	47	**3.25**	**0.82**

4. `minFunc`: unconstrained differentiable multivariate optimization in Matlab. `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`

relathe	DCA	3.82	0.00	**22**	**0**	**2.98**	**0.04**
$n = 1427$	ADCA	3.05	0.02	61	23	18.69	7.09
$d = 4322$	DCA-Like (MM)	3.03	0.01	71	15	26.74	9.86
	ADCA-Like	**3.02**	**0.00**	67	2	28.85	1.98
	DC-PN	3.10	0.04	1,044	313	117.36	32.09
pcmac	DCA	4.35	0.00	**22**	**0**	**1.87**	**0.07**
$n = 1943$	ADCA	3.60	0.01	103	10	11.10	0.45
$d = 3289$	DCA-Like (MM)	**3.59**	**0.00**	103	11	14.88	1.07
	ADCA-Like	**3.59**	**0.01**	81	16	13.78	2.67
	DC-PN	3.69	0.01	1,363	124	232.60	39.44
gisette	DCA	3.52	0.04	**27**	**0**	**15.3**	**0.2**
$n = 7000$	ADCA	3.33	0.00	118	31	39.5	7.9
$d = 5000$	DCA-Like (MM)	3.34	0.01	283	81	209.0	57.9
	ADCA-Like	**3.32**	**0.02**	133	24	62.9	10.0
	DC-PN	3.53	0.02	2187	186	1218.3	194.3
usps	DCA	2.41	0.01	**29**	**1**	**23.1**	**1.8**
$n = 9298$	ADCA	3.92	1.36	70	107	27.9	39.0
$d = 256$	DCA-Like (MM)	**2.34**	**0.00**	106	18	62.5	11.0
	ADCA-Like	**2.34**	**0.01**	107	15	64.9	6.0
	DC-PN	2.57	0.07	2083	538	1315.2	171.7
rcv1	DCA	5.39	0.41	**30**	**13**	**17.43**	**10.31**
$n = 10000$	ADCA	4.86	0.05	259	179	119.11	57.89
$d = 2000$	DCA-Like (MM)	4.84	0.00	295	88	135.77	45.39
	ADCA-Like	**4.82**	**0.00**	130	22	78.53	20.99
	DC-PN	5.13	0.03	5,083	582	9,783.73	2,539.40
20news	DCA	3.67	0.05	**78**	**27**	**160.57**	**84.05**
$n = 18846$	ADCA	3.64	0.09	221	145	207.45	114.95
$d = 2000$	DCA-Like (MM)	3.45	0.01	687	147	1,378.75	353.95
	ADCA-Like	**3.43**	**0.01**	145	5	208.51	8.62
	DC-PN	3.93	0.02	3,156	79	6,843.18	280.67
magic	DCA	2.40	0.00	850	373	413.4	170.4
$n = 19020$	ADCA	2.46	0.01	215	9	150.5	9.8
$d = 10$	DCA-Like (MM)	**2.36**	**0.00**	168	39	135.2	31.1
	ADCA-Like	**2.36**	**0.00**	**106**	**7**	**101.7**	**4.8**
	DC-PN	2.80	0.07	3498	237	4321.0	557.8
letters	DCA	1.58	0.02	1186	159	652.8	98.2
$n = 20000$	ADCA	1.49	0.02	369	104	268.4	86.8
$d = 16$	DCA-Like (MM)	**1.48**	**0.01**	164	24	149.0	19.3
	ADCA-Like	**1.48**	**0.01**	**90**	**7**	**96.8**	**7.4**

	DC-PN	2.14	0.07	1326	253	1997.1	539.0
shuttle	DCA	1.60	0.02	3520	459	6056.5	782.9
$n = 58000$	ADCA	1.48	0.04	760	96	1781.7	215.1
$d = 9$	DCA-Like (MM)	1.45	0.02	333	9	981.7	33.1
	ADCA-Like	**1.42**	**0.00**	**152**	**23**	**553.7**	**123.6**
	DC-PN	2.54	0.03	4693	68	26784.4	3633.2
sensorless	DCA	3.18	0.02	1788	454	3232.1	837.3
$n = 58509$	ADCA	**3.13**	**0.01**	418	24	912.8	50.0
$d = 48$	DCA-Like (MM)	3.21	0.02	313	41	953.7	114.5
	ADCA-Like	3.18	0.02	**153**	**28**	**499.3**	**91.2**
	DC-PN	3.81	0.04	4255	144	20586.9	4488.6
mnist	DCA	3.44	0.00	3894	111	13752.7	644.9
$n = 70000$	ADCA	3.45	0.01	960	60	2659.1	63.3
$d = 784$	DCA-Like (MM)	3.46	0.01	371	67	1576.8	259.7
	ADCA-Like	**3.43**	**0.01**	**196**	**27**	**834.1**	**127.5**
	DC-PN	4.12	0.01	3703	308	21486.7	3643.1
miniboone	DCA	3.55	0.05	3401	1151	20473.3	6471.2
$n = 130064$	ADCA	**3.47**	**0.06**	454	338	2917.1	2050.2
$d = 50$	DCA-Like (MM)	3.53	0.05	469	61	4841.7	691.5
	ADCA-Like	3.53	0.02	**170**	**12**	**1560.6**	**209.1**
	DC-PN	5.36	0.15	471	29	7071.7	844.0
covertype	DCA	2.14	0.00	3998	35	71591.8	1407.2
$n = 581012$	ADCA	1.88	0.00	1670	0	29954.8	0.0
$d = 54$	DCA-Like (MM)	2.10	0.04	1217	66	37123.1	3546.5
	ADCA-Like	**1.68**	**0.01**	**330**	**10**	**8338.4**	**191.8**
	DC-PN	4.24	0.00	5741	45	140122.8	5126.4

Comments on numerical results

— We first interest in the performance of DCA-Like in compare to DCA. It is clear that DCA-Like is better than DCA. DCA-Like gives a lower or equal objective value than DCA in 13 out of 15 datasets (all except *sensorless* and *mnist*). For dataset with more than $19,000$ datapoints, the running time of DCA-Like is from 1.9 to 8.7 times less than DCA. For other cases, although DCA is faster than DCA-Like, the objective value of DCA higher than than DCA-Like by a large margin.

— We study now the benefit of using acceleration technique. The comparison is made between 2 pairs: ADCA versus DCA and ADCA-Like versus DCA-Like. As we can see, ADCA-Like improves the performance of DCA-Like. The gains in computing time are huge, as ADCA-Like is faster than DCA-Like from 1.0 to 6.6 times (for datasets with more than 1000 datapoints, the reset are neglectable).

Concerning objective, ADCA-Like performs the best in 13 out of 15 datasets among all algorithms. Similarly, ADCA also improves the performance of DCA. ADCA is significantly faster than DCA by 2.7 to 7.1 times (for datasets with more than 1000 datapoints) while having a lower objective value in 11 over 15 datasets.

— Overall, ADCA-Like gives the best results among the 5 comparative algorithms. In term of objective value, ADCA-Like gives the lowest in 13 out of 15 datasets among all algorithms. As for running time, ADCA-Like is the fastest for dataset with more than 19000 datapoints. Note that for the remaining cases, although ADCA-Like does not have the smallest running time, it still has the best objective value overall.

In Figure 3.1, we plot the value of objective function as time progress. Surprisingly, DCA performs thoroughly at the beginning but then it is left behind; while DCA-Like, ADCA and ADCA-Like improve swiftly over time. It is noticeable that, although ADCA-Like struggle at first in many cases (*rcv1*, *20news*, *gissete*, *magic*, *letters*, and *shuttle*), it soon catches up and surpasses others algorithms to produces the best result while being the fastest. Also, for the biggest dataset (*covertype*), the effectiveness of accelerated algorithms is clearly demonstrated: ADCA-Like (reps. ADCA) surpass DCA-Like (reps. DCA) after short amount of time.

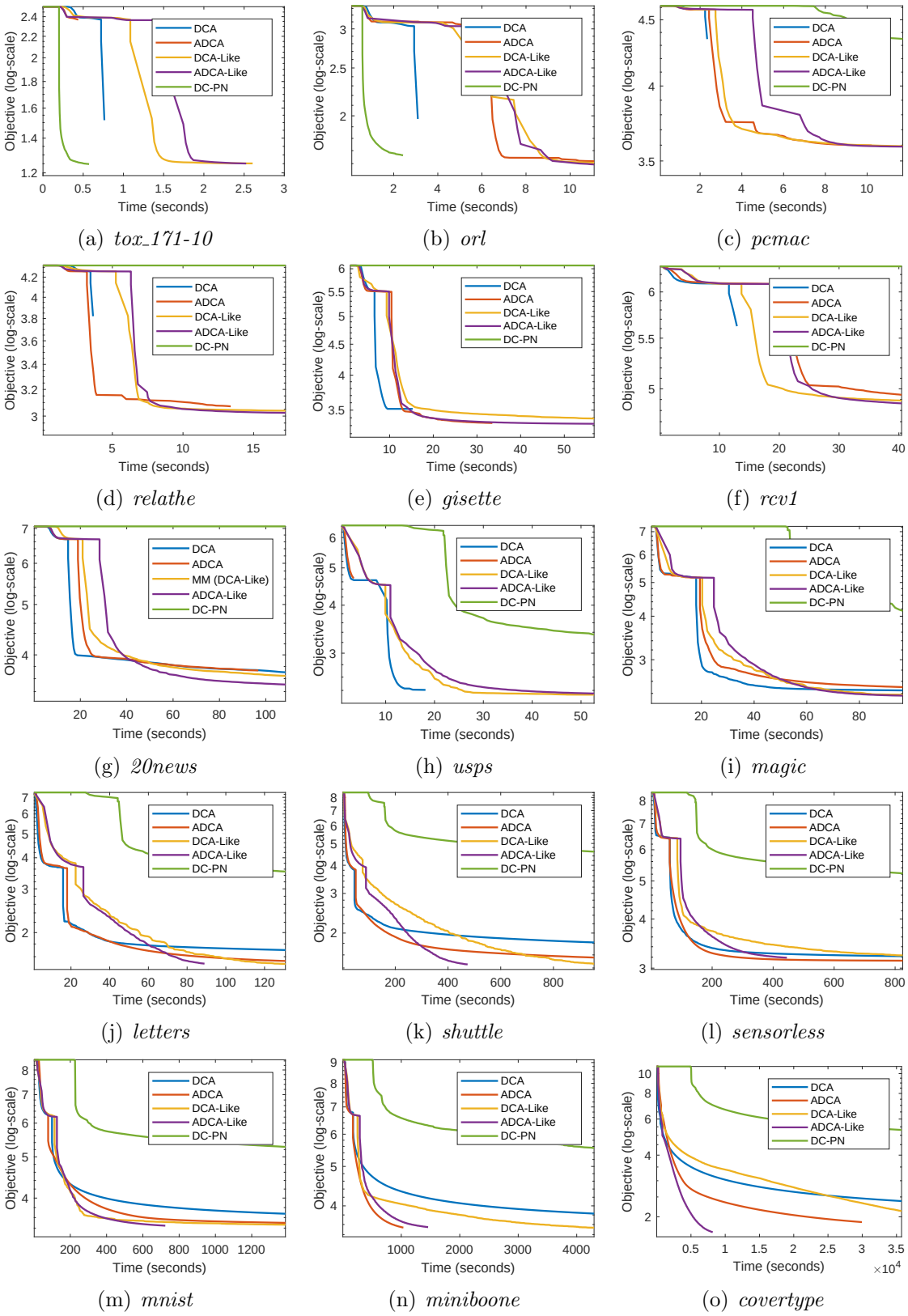
Figure 3.2 visualizes the results on *mnist* dataset given by all five algorithms. This dataset consists of 70000 gray-scale 28×28 images over 10 classes of handwritten digits. *mnist* can be considered as the benchmark dataset for SNE-based algorithms, since they are able to capture both local and global structure of this dataset, especially in 2D embedding space. As we can see, in the embedding space, all four DCA-based algorithms managed to keep the structure of dataset of the original space, whereas DC-PN failed to maintain the structure. Four images of DCA-based algorithms in Figure 3.2 are quite similar since the objective values of all algorithms in this case are fairly similar.

In conclusion, ADCA-Like gives the best results in all three comparison criteria. DCA-Like is clearly an improvement of DCA, and ADCA-Like is undoubtedly an improvement over DCA-Like.

3.5 Conclusion

We proposed two algorithms for the t-SNE problem, DCA-Like and ADCA-Like, which are both inexpensive: the solution of the convex sub-problem can be explicitly computed. We showed that the Majorization-Minimization algorithm, the best state-of-the-art algorithm for t-SNE, is nothing else but DCA-Like. Both proposed algorithms enjoy the convergence properties of DCA. Numerical experiments on high-dimensional and large dataset were carefully conducted on several benchmark datasets for visualization. The numerical results show that DCA-Like greatly improves the running time of DCA while giving similar or better solution. DCA-Like is up to 9.2 times faster than DCA while giving better objective value. ADCA-Likes improves further the

running time (by up to 5 times) as well as the objective value of `DCA-Like`. Thus, we can undoubtedly conclude the two proposed method, `DCA-Like` and `ADCA-Like`, are the state-of-the-art algorithms for solving the t-SNE problem.



(a) *tox_171-10* (b) *orl* (c) *pcmac*

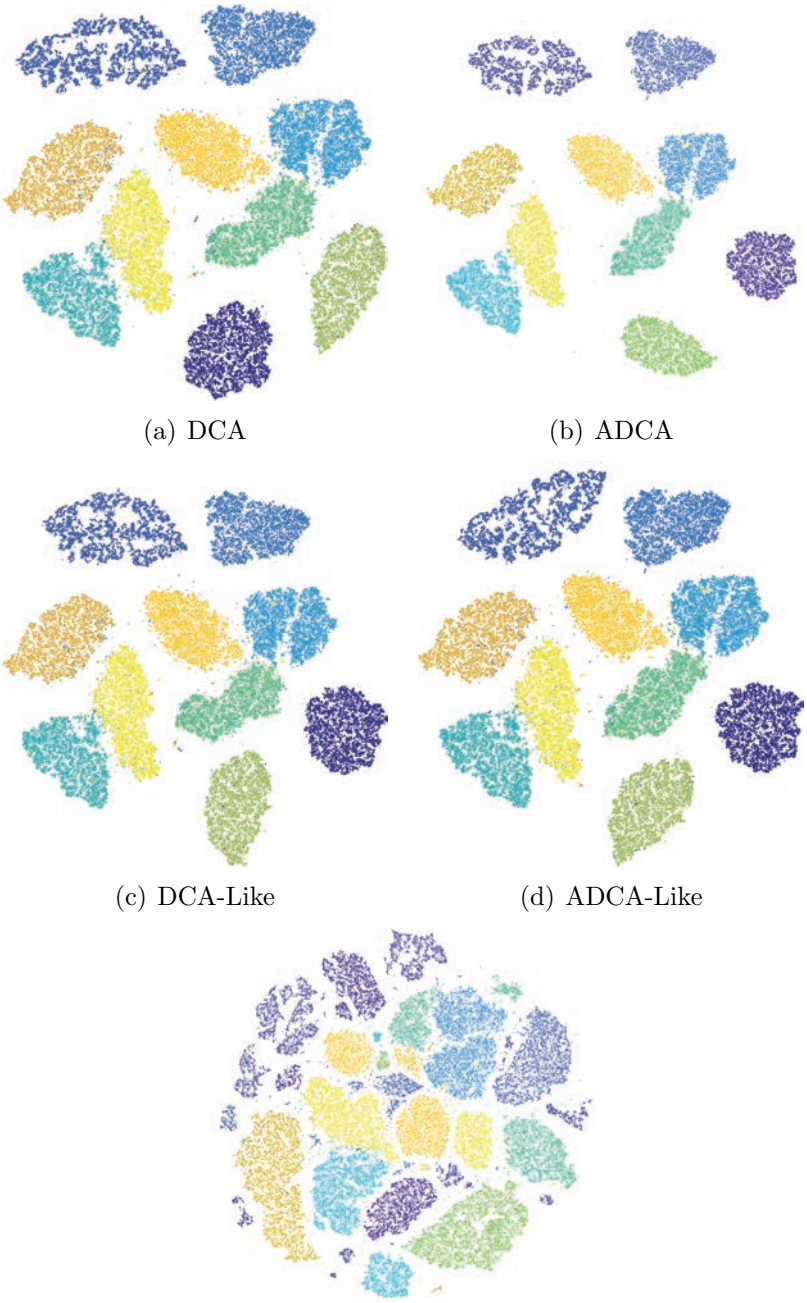
(d) *relathe* (e) *gisette* (f) *rcv1*

(g) *20news* (h) *usps* (i) *magic*

(j) *letters* (k) *shuttle* (l) *sensorless*

(m) *mnist* (n) *miniboone* (o) *covertype*

Figure 3.1 – Objective value versus running time (average of ten runs)



(a) DCA (b) ADCA

(c) DCA-Like (d) ADCA-Like

(e) DC-PN

Figure 3.2 – Visualization of embedding space on *mnist* dataset. Colors represent classes of data (0 – 9).

Chapter 4

Deep Clustering[1]

Abstract: *In this chapter, we propose new approaches for clustering high-dimensional data. The first approach considers the usage of t-SNE (t-Distributed Stochastic Neighbor Embedding) in clustering. We propose two clustering algorithms based on DC Programming and DCA. In the second approach, we consider the problem of joint-clustering using auto-encoder. We propose an extension by using a scale invariance distance measures for a class of joint-clustering problem using auto-encoder, and apply for a particular case of joint-clustering with MSSC (minimizing sum-of-square clustering). The efficiency of the proposed algorithms in comparison with the state-of-the-art algorithms on real-world high-dimensional datasets are empirically demonstrated in terms of clustering quality and running time.*

1. The material of this chapter is developed from the following works:
[1] H.A. Le Thi, B. Tran. *A DCA-based approach for Joint Clustering and Dimensional Reduction by t-SNE. Submitted.*
[2] B. Tran, H.A. Le Thi. *Deep Clustering with Spherical Distance in Latent Space.* In: Advanced Computational Methods for Knowledge Engineering. ICCSAMA 2019. Advances in Intelligent Systems and Computing, Springer, Cham. *Accepted.*
[3] G. Da Silva, H.M. Le, H.A. Le Thi, V. Lefieux, B. Tran. *Customer Clustering of French Transmission System Operator (RTE) Based on Their Electricity Consumption.* In: Le Thi H., Le H., Pham Dinh T. (eds) Optimization of Complex Systems: Theory, Models, Algorithms and Applications. WCGO 2019. Advances in Intelligent Systems and Computing, vol 991, pp. 893–905. Springer, Cham.

4.1 Introduction and related works

Clustering is an important task in data mining with the aim of segmenting datapoints into groups which pose similarity. Despite decades of research, clustering high-dimensional datasets is still a difficult problem due to the *"curse of dimensionality"* phenomenon. In general terms, it is "the widely observed phenomenon that data analysis techniques (including clustering), which work well at lower dimensions, often perform poorly as the dimensionality of analyzed data increases" [101]. Using dimension reduction techniques is a popular way to overcome this problem, where the original features are presented by a smaller but informative set of features. For clustering methods using dimensional reduction techniques, we can categorize them by two components: clustering approaches and dimensionality reduction methods.

There are two main approaches of using dimensional reduction techniques in clustering: "tandem analysis" approach (or two-step approach) and joint-clustering approach. The first approach, "tandem analysis", first introduced in [7], refers to a clustering procedure of two steps. The first step transforms the high-dimensional data to a lower-dimensional space (embedding space) that has preferable properties, where a clustering algorithm is performed on such space in the second step. This approach has extensively studied in the literature and has proved to be effective (better clustering results), such as Principal Component Analysis (PCA) [16, 128], Non-negative Matrix Factorization (NMF) [118], Volume Minimization-based Factorization [119], Auto-encoder, Isomap [124, 132], etc. However, since these two steps are applied separately, the obtained embedding data from the dimension reduction step do not receive any information from the clustering step which gives better adaptation of the latent structure.

Joint-clustering approach has been developed to overcome the mentioned limitation of the "tandem analysis" approach. Instead of employing dimension reduction and clustering seperately, joint-clustering executes simultaneously both tasks, usually by combining them into an optimization problem to solve. In the earliest works, [108] and [22] considered integrating K-means clustering with components analysis for dimension reduction, which is closely related to PCA, and demonstrated the advantages of the proposed joint-clustering method (readers can refer to [78] for an extensively list of joint-clustering by PCA-based techniques). NMF and auto-encoder with their variances have been considered in the joint-clustering framework ([119, 112], and [2, 76], to name a few). The joint-clustering approach often obtains more accurate results than considering each problem independently, which has been demonstrated in the literature.

Beside clustering approaches, one could categorize clustering methods based on reduction techniques. Dimensionality reduction methods have been extensively studied (see [107] and [12] for an extensive review of dimensionality reduction methods). In short, they can be divided into two main categories: linear and non-linear methods. As the name suggested, linear methods performs linear mapping of data points from original space into low-dimension space. In contrasts, non-linear methods assume that

data lie on a non-linear manifold within then high-dimensional space, where they seek the projection/mapping of data points on that space. Despite clustering methods and algorithms are frequently developed for linear methods by both clustering approaches (i.e. PCA [22, 108, 78] and NMF [119, 112]), non-linear methods have mostly been considered only in "tandem analysis" approach (Isomap [100, 124, 132], t-SNE [127, 94], and auto-encoder [2, 76]). For joint-clustering approach, most proposed methods are based on auto-encoder, where they empirically show significant improvement in clustering quality in compare with state-of-the-art methods (readers can refer to [2, 76] for a comprehensive review). The lack of joint-clustering method based on non-linear dimensionality reduction techniques may due to their complexity: they often result in a non-convex optimization problem in most cases, which is practically discouraged. Hence, the usage of non-linear techniques in clustering are not common, except auto-encoder.

With the recent advances in optimization, the non-convexity resulted in joint-clustering methods using non-linear dimensional reduction techniques is no longer a big problem. Motivated by the success of auto-encoder for clustering high-dimensional data, the development of new methods based on non-linear dimensional reduction techniques may lead to better clustering result. Hence, this chapter focuses on developing clustering methods based on two non-linear dimensionality reduction methods: t-SNE and auto-encoder.

We first consider the non-linear dimensionality reduction by t-SNE in clustering. Motivated by the superior of t-SNE over standard methods such as auto-encoder and PCA [105, 107, 106] with efficient non-convex optimization methods for t-SNE [123, 51], we consider the combination of t-SNE with clustering. t-SNE [106] is a relatively new, based on Stochastic Neighbor Embedding family, which based on preserving the pairwise distance between high and low-dimensional data, well-suited for visualization high-dimensional data. According to our knowledge, although there exist several works where t-SNE is considered as the dimensional reduction step for "tandem clustering" such as spectral clustering [94] and Gaussian Mixture Model [127]; attempted to jointly combine t-SNE and clustering in the joint-clustering approach. In this work, we consider the combination of t-SNE with clustering in both joint-clustering and standard "tandem analysis" approach. The proposed algorithms are applicable in many applications, as we can in-cooperate domain knowledge from experts for the clustering tasks by using an appropriate measure that captures the similarities between objects in the original data space. One of the most popular clustering techniques is MSSC (minimizing sum-of-squares clustering), which is well-known by the popularity of K-means [75, 72]. The rapidity and efficiency of MSSC on large-scale datasets have been demonstrated in several works [45, 47], but it still suffers the same limitation when applying to high-dimensional data. Hence, our proposed method would bring the advantage of t-SNE as a powerful dimension reduction method with the fast and effective of MSSC. The resulting problem is a difficult non-convex problem, where we tackle by DC (Difference of Convex functions) programming and DCA (DC Algorithm), which are powerful tools in non-convex optimization literature.

For the second considered non-linear dimensionality reduction – auto-encoder, we

consider a class of joint-clustering methods with auto-encoder. Auto-encoder seeks a mapping function f_θ to project data x_i into $z_i := f_\theta(x_i)$ in lower-dimensional space. Since f_θ is often a composition of non-linear transformations parametrized by θ, f can approximate a large class of function. Learning a high-quality mapping f_θ requires a massive amount of data, which enjoys the growing volume of unlabeled data, hence explained its popularity recently. Auto-encoder have demonstrated its effectiveness over standard methods for high-dimensional data such as PCA [35, 107], and its application in clustering recently empirically show significant improvement in clustering quality in compare with state-of-the-art methods (readers can refer to [2, 76] for a comprehensive review). They can be classified into two main types: (1) simultaneously minimizing the auto-encoder reconstruction and clustering in a joint framework by linear scalarization techniques (i.e. min $F^{joint} = F^{AE} + \lambda F^{clustering}$) [104, 99, 120, 129, 32, 38, 130, 131, 96], or (2) progressively updating neural network mapping and clustering assignment in order to match a target distribution/pseudo label, which is updated during the optimization process [117, 33, 32, 121, 14, 28]. Generally speaking, it can be viewed as solving an optimization problem or a sequence of problems where each has the form similar to (1). However, the derived problem in several works is not well-defined, because clustering is applied on data's representation z_i in the latent space, the clustering objective $F^{clustering}$ is affected by the scaling whereas the reconstruction objective of the auto-encoder F^{AE} is not. This affects a class of joint-clustering algorithm realized on Euclidean distance or ℓ_p norm such as K-means (MSSC) [104, 99, 120, 129, 32, 43], latent space clustering [38, 130, 131]; or integrating additional tasks on the latent space depends on Euclidean distance/ℓ_p norm [21, 37, 17, 97]. To solve the mentioned problem, one could consider regularization techniques for neural network, such as $\|\theta\|_p$ or $\|z_i\|_p$ [37]. However, it introduces another trade-off parameter that needs to be tuning, which is not encouraged in unsupervised setting. Hence, instead of using Euclidean distance or ℓ_p norm, we could choose another distance function that is invariance to scaling. Motivated by the success of cosine distance in document clustering [24], which is a well-known measure that satisfies our mentioned property, we consider two variants of cosine distance. As a specific case, we considered the specific problem of deep joint-clustering with MSSC, and proposed an algorithm to solve the mentioned reformulation by taking advantage of state-of-the-art first-order stochastic methods for neural networks such as Adam [41].

This chapter contains two main parts divided into two sections. Section 4.2 dedicates to the clustering problems using t-SNE, and the section 4.3 considers the problem of clustering using auto-encoder. Finally, section 4.4 concludes the chapter.

In this chapter, we use the following notations:

Notation	Meaning
n	Number of datapoints
k	Number of clusters
m	Number of features of raw data
d	Number of dimensions in low-dimensional space
a_i	Row i-th of the matrix a
$x_i \in \mathbb{R}^m$	The i^{th} original datapoint
$x := \{x_1, \ldots, x_n\} \in \mathbb{R}^{n \times m}$	Original dataset
$z_i \in \mathbb{R}^d$	The i^{th} datapoint in low-dimensional space
$z := \{z_1, \ldots, z_N\} \in \mathbb{R}^{n \times d}$	Data matrix of dataset in low-dimensional space

4.2 Two-step and joint-clustering by t-SNE and MSSC

4.2.1 Our contributions

In the first part, we study the problem of clustering by t-SNE by following the two approaches of "tandem analysis" and joint-clustering. In the joint-clustering approach, we propose a model for simultaneously clustering and dimensional reduction by t-SNE. Later on, we study DC Programing and DCA for both approaches and propose algorithms that are suitable for large-scale setting. We conduct extensive experiments for high-dimensional large-scale datasets with several recent clustering methods to study the quality of the proposed algorithm.

The remaining of this chapter is organized as following. Section 4.2.2 introduces the "tandem analysis" clustering by t-SNE. Next, in Section 4.2.3, we formulate the problem of joint-clustering between t-SNE and MSSC, and develop an efficient method to solve the proposed model by DC Programming and DCA. Numerical experiment's results are report in Section 4.2.4.

4.2.2 Two-step clustering by t-SNE and MSSC

The two-step clustering approach has two main steps: (1) applying a dimensional reduction algorithm to data for mapping high-dimensional data to a lower-dimensional space that suitable for performing clustering, then (2) performing clustering on embedded space. In this work, we consider the t-SNE as the dimensional reduction algorithm. This choice is motivated by its effectiveness for high-dimensional setting, which has been widely used for visualizing data ([106, 114, 77], to cite a few) and for clustering [94, 127]. In addition, t-SNE offers the liberty to choose the distance measures the similarity between objects in the datasets. In detail, the original work [106] defines joint probabilities p_{ij} that measure the pairwise similarity between objects $\mathbf{x}_i$ and $\mathbf{x}_j$

by symmetrizing two conditional probabilities as $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$, where

$$p_{j|i} = \begin{cases} \frac{\exp(-d_{\text{original}}(x_i,x_j)/2\sigma_i^2)}{\sum_{k\neq i}\exp(-d_{\text{original}}(x_i,x_k)/2\sigma_i^2)} & \text{if } i \neq j, \\ 0 & \text{otherwise,} \end{cases}$$

d_{original} is squared Euclidean distance. However, (squared) Euclidean distance is not always the suitable to measure the similarity between two objects in the original space. For example, Dynamic Time Warping distance has been proved in several works [30, 92, 82, 20] for measuring distance between time-series (see the surveys [1] and [113] for a complete list of similarity measures for time-series data). Hence, clustering high-dimensional data by t-SNE is suitable for a wide range of applications.

Minimize sum-of-squares clustering (MSSC)

Given a dataset $z := \{z_1, \ldots, z_n\}$ where $z_i \in \mathbb{R}^d$ $(i = 1, \ldots, n)$, our goal is to find k points $\{u_1, \ldots, u_k\}$, known as "centroid", and assign each data point in z to its closest centroid in u by a distance measure. The MSSC problem is formulated as minimizing the sum of assignment distance, which is the distance from z_i to the assigned/closest centroid, by squared Euclidean distance. The formulation of MSSC is

$$\min_{u\in\mathbb{R}^{k\times d}} \left\{ F_C(u) = \frac{1}{2}\sum_{i=1}^{n} \min_{l=1,\ldots,k} \|u_l - z_i\|_2^2 \right\}. \tag{4.1}$$

The problem has been investigated, and solve efficiently by several methods, including classical method such as K-means [72], or more recent methods such as DCA [45, 61, 47]. In [45], the authors have formulated the MSSC problem as a DC program, and proposed an efficient algorithm. In detail, according to the property

$$\min_{l=1,\ldots,k} \|u_l - z_i\|_2^2 = \sum_{l=1}^{k} \|u_l - z_i\|_2^2 - \max_{r=1,\ldots,k} \sum_{l=1,l\neq r}^{k} \|u_l - z_i\|_2^2,$$

we can derive $F_C(u)$ as a DC decomposition $F_C(u) = G_C(u) - H_C(u)$, with

$$G_C(u) := \sum_{i=1}^{n}\sum_{l=1}^{k} \frac{1}{2}\|u_l - z_i\|_2^2 \tag{4.2}$$

$$H_C(u) := \sum_{i=1}^{n} \max_{r=1,\ldots,k} \sum_{l=1,l\neq r}^{k} \frac{1}{2}\|u_l - z_i\|_2^2, \tag{4.3}$$

where both $G_C(u, z)$ and $H_C(u, z)$ are convex [45].

According to the general scheme of DCA in Section 1.1.3, for each iteration, DCA computes $\bar{u_C}^q \in \partial H_C(u^q)$ and obtain u^{q+1} by solving the convex sub-problem

$$\min\{G_c(u) - \langle u, \bar{u_C}^q\rangle : u \in \mathbb{R}^{k\times d}\}. \tag{4.4}$$

The calculation of $\bar{u_C}^q$ is as following. For $i \in \{1, \ldots, n\}$, denote:

$$h_i(u) = \max_{r=1,\ldots,k} h_{i,r}(u), \text{ where } h_{i,r}(u) = \sum_{l=1, l\neq r}^{k} \frac{1}{2}\|u_l - z_i\|_2^2 \text{ for } r \in \{1, \ldots, k\},$$

$$\begin{aligned} V_i(u) := \arg\max_{r=1,\ldots,k} h_{i,r}(u) &= \arg\max_{r=1,\ldots,k} \frac{1}{2}\left(\sum_{l=1}^{k} \|u_l - z_i\|_2^2 - \|u_r - z_i\|_2^2\right) \\ &= \arg\min_{r=1,\ldots,k} \|u_r - z_i\|_2^2. \end{aligned}$$

Hence, $\bar{u_C}^q \in \partial H_C(u^q) \Leftrightarrow \bar{u_C}^q = \sum_{i=1}^{n} \bar{u}_i^q$ where $\bar{u}_i^q \in \partial h_i(u^q)$. Then we have [36]

$$\partial h_i(u) = co\left\{\bigcup_{r \in V_i(u)} \partial h_{i,r}(u)\right\},$$

where *co* stands for the convex hull. Hence, $\partial h_i(u)$ is a convex combination of $\{\nabla h_{i,r}(u) : r \in V_i(u)\}$, i.e,

$$\partial h_i(u) = \sum_{r \in V_i(u)} \lambda_r^{[i]} \nabla h_{i,r}(u) \text{ with } \lambda_r^{[i]} \geq 0 \text{ and } \sum_{r \in V_i(k)} \lambda_r^{[i]} = 1.$$

The gradient of $\nabla_{u_l} h_{i,r}(u)$ is computed as (for $r, l \in \{1, \ldots, k\}$):

$$\nabla_{u_l} h_{i,r}(u) = (u_l - z_i) \text{ if } r \neq l, 0 \text{ otherwise.}$$

For simplicity, at each iteration q, we choose $p^q(u^q) = (p_i^q(u^q))_{i=1,\ldots,n}$ where $p_i^q(u^q)$ is an element from $V_i(u^q)$, which results $\bar{u}_i^q = \nabla h_{i,p_i^q(u^q)}(u^q)$. $\bar{u_C}^q$ is calculated as

$$(\bar{u_C}^q)_l = (n - |c_l^q|)u_l^q + \sum_{i \in c_l^q} z_i - \sum_{i=1}^{n} z_i \text{ for } l = 1, \ldots, k,$$

where $(\bar{u_C}^q)_l$ is the l-th row of $\bar{u_C}^q$ and $c_l^q = \{i = 1, \ldots, n : l = p_i^q(u^q)\}$.

The solution of the convex sub-problem (4.4) is obtained as following. Since the objective of the problem (4.4) is convex, and according to first order optimality condition, u^{q+1} can be obtained by solving the following system of equations

$$\bar{u_C}^q = \nabla_u G_C(u). \tag{4.5}$$

The gradient of $G_C(u)$ is computed as

$$\nabla_{u_l} G_C(u, z) = nu_l - \sum_{i=1}^{n} z_i \text{ for } l = 1, \ldots, k.$$

Algorithm 4.1 `MSSC`: DCA for solving problem (4.1)

Input: Dataset x and the number of cluster k.
Output: Centroids u^* and clustering assignment p^*.
Initialization: Choose u^0 and $q \leftarrow 0$.
repeat
  Calculate u^{q+1} according to (4.6).
  $q \leftarrow q + 1$.
until Stopping condition.
Output $u^* \leftarrow u^q$ and $p^* \leftarrow u^q$.

Consequently, equation (4.5) can be simplified to

$$u_l = \left(1 - \frac{|c_l^q|}{n}\right) u_l^q + \frac{1}{n} \sum_{i \in c_l^q} z_i \text{ for } l = 1, \ldots, k. \tag{4.6}$$

The DCA for Problem (4.1) is described in Algorithm 4.1.

Two-step clustering by t-SNE

Given a high-dimensional dataset $x := \{x_1, \ldots, x_n\}$ in $\mathbb{R}^m$. Our task consists of finding $z := \{z_1, \ldots, z_n\}$ where $z_i \in \mathbb{R}^d$ $(i = 1, \ldots, n)$ and k points $\{u_1, \ldots, u_k\}$, known as "centroid", and assign each data point in z to its closest centroid in u. The two-step MSSC clustering with t-SNE, named `MSSC-2S` (2-step) is described in Algorithm 4.2.

Algorithm 4.2 `MSSC-2S`: two-step MSSC clustering with t-SNE

Input: Dataset x, number of cluster k.
Output: Low-dimensional data representation z, centroids u^* and clustering assignment p^*.
Step 1: Apply t-SNE to the data
  Compute proximity matrix P from x.
  Apply algorithm 3.2 with input P to obtain z.
Step 2: Apply MSSC clustering
  Initialize u_0 from z.
  Apply `MSSC` (algorithm 4.1) with initial point u_0 and data z to obtain the clustering result (u^*, p^*).

As an application, we use two-step clustering algorithm for customer clustering of French transmission system operator (RTE) based on their electricity consumption in [20]. The ultimate goal of customer clustering is to automatically detect patterns for understanding the behaviors of customers in their evolution. It will allow RTE to better know its customers and consequently to propose them more adequate services, to optimize the maintenance schedule, to reduce costs, etc. We tackle three crucial issues in high-dimensional time-series data clustering for pattern discovery: appropriate similarity measures, efficient procedures for high-dimensional setting, and fast/scalable clustering algorithms. For that purpose, we use the DTW (Dynamic Time Warping)

distance in the original time-series data space, the t-distributed stochastic neighbor embedding (t-SNE) method to transform the high-dimensional time-series data into a lower dimensional space, and DCA (Difference of Convex functions Algorithm) based clustering algorithms. We conduct numerical experiment on a dataset contains 462 clients, where each is represented by a curve of electricity consumption for each 10 minutes over two years (from 01 January 2016 to 31 December 2017); hence it results a dataset of 462 time-series in $\mathbb{R}^{105,120}$. We observe that the two-step algorithm only takes approximately 10 minutes, whereas the whole process (including other steps) takes 42 minutes in total. The numerical results on real-data of RTE's customer have shown that our clustering result is coherent: customers in the same group have similar consumption curves and the dissimilarity between customers of different groups are quite clear.

4.2.3 Joint-clustering by t-SNE and MSSC

In this section, we proposed a formulation for joint-clustering by MSSC and t-SNE problem, and developed an optimization algorithm to the proposed problem.

4.2.3.1 Problem formulation and solution methods

Given a dataset $x := \{x_1, \ldots, x_n\}$ in $\mathbb{R}^m$. Let $z := \{z_1, \ldots, z_n; z_i \in \mathbb{R}^d\}$, $d \ll m$; and $u := \{u_1, \ldots, u_k\}$ are k centroids in $\mathbb{R}^d$. Recall that the objective function of MSSC in section 4.2.2 and t-SNE in section 3.2 are

$$\begin{aligned} F_C(u,z) &= \frac{1}{2}\sum_{i=1}^{n} \min_{l=1,\ldots,k} \|u_l - z_i\|_2^2, \\ F_E(z) &= \log \sum_{i\neq j} (1 + \|z_i - z_j\|_2^2)^{-1} + \sum_{i,j} p_{ij} \log(1 + \|z_i - z_j\|_2^2). \end{aligned}$$

We aim to jointly learn an embedding z in $\mathbb{R}^d$ (of x) by t-SNE, that clustered around k centroids in $\mathbb{R}^d$. One nature way to formulate the joint-clustering problem is by adding the objective functions $F_C(u,z)$ and $F_E(z)$. This gives us the following optimization problem

$$\min_{u,z} F_{Joint}(u,z) = F_E(z) + \lambda F_C(u,z), \tag{4.7}$$

where λ controls the trade-off parameter between clustering and dimension reduction. Since both $F_E(z)$ and $F_C(u,z)$ are non-convex, the problem (4.7) is a non-convex optimization problem. In the next section, we will employ DCA-based algorithms to efficiency solve this problem.

4.2.3.2 DC Decomposition for the Problem (4.7)

Recall from section 4.2.2 and 1.3.1, the objective function $F_E(z)$ can be reformulated as $F_E(z,v) = G_E^\mu(z,v) - H_E^\mu(z,v)$, with

$$G_E^\mu(z,v) := \frac{\mu}{2}\|z\|_2^2 + \chi_\Omega(z,v), \tag{4.8}$$

$$H_E^\mu(z,v) := \frac{\mu}{2}\|z\|_2^2 - F_E(z,v) = \frac{\mu}{2}\|z\|_2^2 - f_E(z) - \sum_{i,j} h_{ij}(v), \tag{4.9}$$

where $\Omega = \{(z,v) : g_{ij}(z) \leq v_{ij} \,|\, i,j = 1,\ldots,n\}$; $g_{ij}(z)$, $f_E(z)$ and $h_{ij}(v)$ are $g_{ij}(\mathbf{x})$, $f(\mathbf{x})$ and $h_{ij}(\mathbf{z})$ in section 4.2.2 respectively; $f_E(z)$ is differentiable with L-Lipschitz constant and $\mu > L$.

Using the same reformation as in section 1.3.1, the problem (4.7) is equivalent to the following optimization problem

$$\min_{(u,z,v)\in\Psi} F_J(u,z,v) = F_E(z,v) + \lambda F_C(u,z), \tag{4.10}$$

where $F_C(u,z)$ is defined in (4.7), $F_E(z,v)$ is defined as $\Psi = \mathbb{R}^{k\times d} \times \Omega$.

This gives us the following DC decomposition of $F_J(u,z,v)$:

$$F_J(u,z,v) = G_J(u,z,v) - H_J(u,z,v) \tag{4.11}$$

where

$$G_J(u,v,z) = \lambda G_C(u,z) + G_E^\mu(z,v),$$
$$H_J(u,v,z) = \lambda H_C(u,z) + H_E^\mu(z,v),$$

$G_C(u,z)$ and $H_C(u,z)$ take the form

$$G_C(u,z) := \sum_{i=1}^{n}\sum_{l=1}^{k} \frac{1}{2}\|u_l - z_i\|_2^2$$

$$H_C(u,z) := \sum_{i=1}^{n} \max_{r=1,\ldots,k} \sum_{l=1,l\neq r} \frac{1}{2}\|u_l - z_i\|_2^2;$$

$G_E^\mu(z,v)$ and $H_E^\mu(z,v)$ are defined as in (4.8) and (4.9).

According to the convexity property of $\|u_l - z_i\|^2$, it is clear that $G_C(u,z)$ and $H_C(u,z)$ are convex. With $\mu > L$ (in Section 4.2.2), $G_E(u,z)$ and $H_E(u,z)$ are convex. Hence, the decomposition (4.11) is a valid DC decomposition.

4.2.3.3 Optimization algorithm for (4.7)

Followed the DC decomposition (4.11), this section develops a solution method based on DCA to solve the problem (4.10).

Calculation of ∂H_J:

Denote $(\bar{z_E}^q, \bar{v_E}^q) \in \partial H_E^\mu(z^q, v^q) = \nabla_z H_E^\mu(z^q, v^q)$ and $(\bar{u_C}^q, \bar{z_C}^q) \in \partial H_C(u^q, z^q)$. $\bar{z_E}^q$ and $\bar{v_E}^q$ are calculated as

$$\begin{aligned}
\bar{z_E}^q = \nabla_z H_E^\mu(z^q, v^q) &= \mu z^q - \nabla_z f_E(z^q), \\
\nabla_{z_i} f_E(z^q) &= \sum_{j=1}^{n} \frac{-4(z_i^q - z_j^q)(1 + \|z_i^q - z_j^q\|_2^2)^{-2}}{\sum_{l \neq m}(1 + \|z_l^q - z_m^q\|_2^2)^{-1}}, \text{ for } i = 1 \ldots, n, \\
\bar{v_E}^q = \nabla_v H_E^\mu(z^q, v^q) &= \frac{p_{ij}}{1 + \|z_i^q - z_j^q\|_2^2};
\end{aligned}$$

and $\bar{u_C}^q$ and $\bar{z_C}^q$ are computed as

$$\begin{aligned}
(\bar{u_C}^q)_l = &\quad (n - |c_l^q|)u_l^q + \textstyle\sum_{i \in c_l^q} z_i^q - \sum_{i=1}^n z_i^q \quad \text{for } l = 1, \ldots, k, \\
(\bar{z_C}^q)_i = &\quad (k-1)z_i^q + u_{p_i^q(u^q, z^q)} - \textstyle\sum_{l=1}^k u_l^q \quad \text{for } i = 1, \ldots, n,
\end{aligned}$$

where $c_l^q = \{i = 1, \ldots, n : l = p_i^q(u^q, z^q)\}$ and $p_i^q(u^q, z^q)$ are chosen from $V_i(u^q, z^q) := \arg\min_{j=1,\ldots,k} \|u_j^q - z_i^q\|_2^2$.

Solving the convex sub-problem by alternating:

According to DCA scheme (section 1.1.3), we solve the following convex sub-problem at each iteration q,

$$\min_{(u,z,v) \in \Psi} \left\{ G_J(u, v, z) - \lambda \left\langle \bar{u_C}^q, u \right\rangle - \lambda \left\langle \bar{z_C}^q, z \right\rangle - \left\langle \bar{z_E}^q, z \right\rangle + \left\langle -\bar{v_E}^q, v \right\rangle \right\}. \tag{4.12}$$

The solution of the problem (4.12) is given by:

$$(u, z)^{q+1} \in \operatorname*{argmin}_{u,z} \{F_S(u, z)\} \tag{4.13}$$

$$v_{ij}^{q+1} = g_{ij}(z) \tag{4.14}$$

where

$$F_S(u, z) = \lambda G_J(u, v, z) + \sum_{i,j} (-\bar{v_E}_{ij}^q) g_{ij}(z) - \lambda \left\langle \bar{u_C}^q, u \right\rangle - \lambda \left\langle \bar{z_C}^q, z \right\rangle - \left\langle \bar{z_E}^q, z \right\rangle.$$

Solving (4.13) effectively for large-scale setting faces a huge obstacle. Since the objective function of problem (4.13) is convex, then the most straigh-forward way is solving the linear system $\nabla F_S(u, z) = 0$ by using the first order optimality condition. This linear system has the form of

$$(\mathbb{A}^q - \mathbb{B}^q) z = \mathbb{C}^q, \tag{4.15}$$

where $\mathbb{A}^q = (\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T})$ and $\mathbb{B}^q = \frac{\lambda k}{n} \mathbb{1}_{n \times n}$ (detail about the expansion is presented in appendix A.2).

In practice, $\mathbb{A}^q$ is a sparse matrix in case the proximity matrix P is sparse, but the $\mathbb{A}^q - \mathbb{B}^q$ is a full matrix as $\mathbb{B}^q$ is a full matrix. Storing $\mathbb{A}^q - \mathbb{B}^q$ is memory-intensive, as the space complexity for storing it is $O(n^2)$ regardless of the sparsity of matrix P (hence $\mathbb{A}^q$). As for $n = 70,000$ yields the total size of $36,5$ GB of RAM for the matrix $\mathbb{A}^q - \mathbb{B}^q$, hence this algorithm can not scale for bigger dataset. One may argue that $\mathbb{A}^q - \mathbb{B}^q$ could be stored in custom "sparse" matrix format where the "sparse" element is $\lambda k/n$ (or 1 if we divide both sides of the linear system by $\lambda k/n$) instead of 0 in standard sparse matrix, but it is complex and difficult to apply existing powerful tools for solving the huge system of equation $(\mathbb{A}^q - \mathbb{B}^q)z = \mathbb{C}^q$.

Hence, in order to exploit the sparsity structure of the matrix P, we employ alternating optimization algorithm to solve the convex sub-problem (4.13). Each alternating iteration t of problem (4.13) requires solving two convex sub-problems as

$$u^{q,t+1} \in \underset{u}{\operatorname{argmin}} \left\{ G_C(u, z^{q,t+1}) - \langle \bar{u_C}^q, u \rangle \right\}, \tag{4.16}$$

and

$$z^{q,t+1} \in \underset{z}{\operatorname{argmin}} \left\{ \lambda G_C(u^{q,t}, z) + \frac{\mu}{2}\|z\|^2 - \lambda \langle \bar{z_C}^q, z \rangle - \langle \bar{z_E}^q, z \rangle + \sum_{i,j} (-\bar{v_E}^q_{ij}) g_{ij}(z) \right\}. \tag{4.17}$$

Since the objective function of problem (4.16) is convex, then according to the first order optimality condition, its solution can be obtained by solving $\nabla_u G_C(u, z^{q,t+1}) - \bar{u_C}^q = 0$. Hence, $u^{q,t+1}$ is computed as

$$u^{q,t+1} = \frac{1}{n} \left(\bar{u_C}^q + \mathbb{1}_{k \times n} z^{q,t+1} \right). \tag{4.18}$$

Solving (4.17) is equivalent to solving the following linear system

$$\mathbb{D}^{q,t} z = \left(\lambda \bar{z_C}^q + \bar{z_E}^q + \lambda \mathbb{1}_{n \times k} u^{q,t} \right), \tag{4.19}$$

where $\mathbb{D}^{q,t} = \left((\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T}) \right)$, $\mathcal{L}_A$ is a $\mathbb{R}^{n \times n}$ matrix with $(\mathcal{L}_A)_{ij} = -A_{ij}$ if $i \neq j$ and $-A_{ii} + \sum_{l=1}^{n} A_{il}$ otherwise. For the detailed expansion to acquire (4.19), refer to section A.3. Algorithm for solving the problem (4.12) is detailed in Algorithm 4.3.

Solving (4.13) has double advantages. Beside the memory issue, algorithm 4.3 is potentially faster than solving (4.15). The best alogrithm for solving the linear system (4.15) has the complexity $\mathcal{O}(n^{2.376})$ since $(\mathbb{A}^q - \mathbb{B}^q)$ is a full matrix. On the other hand, since the sparse matrix P has at most $(2kn)$ non-zero elements (where k is the k-nearest neighbor paramter for computing P), hence the matrix $\mathbb{D}^{q,t}$ of (4.19) has at most $(2k+1)n$ non-zero elements. Algorithm 4.3 runs for t iterations, each has the complexity of $\mathcal{O}((2k+1)\sqrt{\kappa}n))$ ([98], by Conjugate Gradient method), where κ is the condition number of $\mathbb{D}^{q,t}$. Since we add $(\lambda k + \mu)$ to the diagonal of $\mathbb{D}^{q,t}$, the κ of $\mathbb{D}^{q,t}$ is small. In addition, the number of neighbor are often small ($k \ll n$, i.e $k = 5$). Hence, for a large value of n, this is an huge improvement in terms of running time.

Algorithm 4.3 Algorithm for solving the problem (4.12)

Input: $u^{q,0}$, $z^{q,0}$, λ
Output: u^*, z^*
Initialization: $t \leftarrow 0$.
repeat
  Compute $z^{q,t+1}$ by solving (4.19).
  Compute $u^{q,t+1} = \frac{1}{n}\left(\bar{u_C}^q + \mathbb{1}_{k\times n} z^{q,t+1}\right)$.
  $t \leftarrow t+1$.
until Stopping condition.
Output $(z^*, u^*) = (z^{q,t}, u^{q,t})$

DCA-Like for problem (4.7)**:**

The DCA for solving the problem (4.7) is described in Algorithm 4.4. However, based on the advantages of DCA-Like over DCA for t-SNE problem in chapter 3, we consider DCA-Like for solving the problem (4.7).

Algorithm 4.4 DCA for solving problem (4.7)

Initialization: Choose u^0, z^0, $\mu > L$ and $q \leftarrow 0$.
repeat
  Compute $(\bar{z_E}^q, \bar{v_E}^q) \in \partial H_E^{\mu_q}(z^q, v^q)$ and $(\bar{u_C}^q, \bar{z_C}^q) \in \partial H_C(u^q, z^q)$.
  Obtain (z^{q+1}, u^{q+1}) by using Algorithm (4.3).
  $q \leftarrow q+1$.
until Stopping condition.

At each iteration q, DCA-Like requires μ_q satisfies the following condition:

$$\begin{aligned} H_J(u^{q+1}, z^{q+1}, v^{q+1}) \geq \quad & H_J(u^q, z^q, v^q) + \langle \bar{u_J}^q, u^{q+1} - u^q \rangle \\ & + \langle \bar{v_J}^q, v^{q+1} - v^q \rangle + \langle \bar{z_J}^q, z^{q+1} - z^q \rangle \end{aligned} \tag{4.20}$$

where $(\bar{u_J}^q, \bar{v_J}^q, \bar{z_J}^q) \in \partial H_J(u^q, v^q, z^q)$. Hence, in compare to Algorithm 4.4, we only need to search for $\mu = \mu_q$ such that the inequality (4.20) holds at $(z^{q+1}, u^{q+1}, v^{q+1})$. The condition (4.20) is detailed in (4.21). The strategy to update μ_q is described in Alogrithm 4.5.

In detail, condition (4.20) is expanded as

$$U_E^{\mu_q}(z^q, z^{q+1}) - \lambda U_C(u^q, z^q, u^{q+1}, z^{q+1}) \geq F_E(x^{q+1}) - \lambda H_C(z^{q+1}, u^{q+1}), \tag{4.21}$$

where

$$\begin{aligned} U_E^{\mu_q}(z^q, z^{q+1}) &= F_E(z^q) + \left\langle \nabla f_E(z^q), z^{q+1} - z^q \right\rangle \\ &\quad + \frac{\mu_q}{2}\|z^{q+1} - z^q\|^2 - \left\langle \bar{v_E}^q, g(z^{q+1}) - g(z^q) \right\rangle, \\ U_C(u^q, z^q, u^{q+1}, z^{q+1}) &= H_C(u^q, z^q) + \left\langle \nabla H_C(z^q, u^q), (z^{q+1} - z^q, u^{q+1} - u^q) \right\rangle. \end{aligned}$$

Proposition 4.1. *Let $g(z) = \{g_{i,j}(z)\}_{i,j=1,\dots,n}$, $\phi = (u, z, g(z))$. The sequence $\{\phi^q\}$ generated by Algorithm 4.5 enjoys the following properties:*

Algorithm 4.5 `MSSC-JDR`: DCA-Like for solving problem (4.7)

Initialization: Choose u^0, z^0, $\eta > 1$, $0 < \delta < 1$, $\mu_0 > 0$ and $q \leftarrow 0$.
repeat
  Compute $(\bar{z_E}^q, \bar{v_E}^q) \in \partial H_E^{\mu_q}(z^q, v^q)$ and $(\bar{u_C}^q, \bar{z_C}^q) \in \partial H_C(u^q, z^q)$.
  Set $\mu_q = \max\{\mu_0, \delta\mu_{q-1}\}$.
  Obtain (z^{q+1}, u^{q+1}) by using Algorithm (4.3).
  while $U_E^{\mu_q}(z^q, z^{q+1}) - \lambda U_C(u^q, z^q, u^{q+1}, z^{q+1}) < F_E(x^{q+1}) - \lambda H_C(z^{q+1}, u^{q+1})$ **do**
    $\mu_q \leftarrow \eta\mu_q$
    Obtain (z^{q+1}, u^{q+1}) by using Algorithm (4.3).
  end while
  $q \leftarrow q + 1$.
until Stopping condition.

(i) The sequence $\{F_J(\phi^q)\}$ is decreasing;

(ii) If $F_J(\phi^{q+1}) = F_J(\phi^q)$, then ϕ^q is a critical point of (4.10) *and Algorithm 4.5 terminates at the q-th iteration.*

(iii) If $\mu(G) + \mu(H) > 0$ then the series $\{\|\phi^{q+1} - \phi^q\|^2\}$ converges.

Proof. Direct consequences of the convergence properties of DCA-Like in [51] and [80]. □

4.2.4 Numerical experiment

4.2.4.1 Experiment settings and Datasets

Evaluation criteria:

Given an input x_i with ground-truth label l_i; and p_i is the assignment label from clustering algorithm, we measure the following criteria to evaluate experimental results:

— **Clustering Accuracy (ACC [13])**: ACC is calculated as $ACC(l, p) = \frac{1}{n}\sum_{i=1}^{n} 1_{m(p_i)=l_i}$, where $m(x_i)$ is the function which maps each clustering assign in p_i to the equivalent label l_i. In our case, we use the mapping by using the Kuhn-Munkres algorithm [73].
— **Normalized mutual information (NMI [102])**: The NMI criterion is calculated as $\text{NMI}(l, p) = \frac{I(l,p)}{\sqrt{H(l)H(p)}}$, where $I(l, p)$ is the mutual information of l and p.

In addition, we measure running time (in seconds) to evaluate the speed of algorithms. All experiments are conducted on a Intel(R) Xeon (R) CPU E5-2630 v4 @2.20 GHz with 32GB of RAM and a GTX 1080 GPU.

We repeat the experiment 10 times, and report the average with standard deviation of each criterion.

Setting for `MSSC-2S` and `MSSC-JDR`: following the work of [51, 123], we set $\mu_0 = 10^{-6}$. For all datasets, proximity matrix P is constructed by k-Nearest Neighbor with $k_{\text{NN}} = 10$. z_0 is draw from $\mathcal{N}(0, 10^{-8})$. Early exaggeration technique [106] is deployed for first 20 iterations with the constant value of 4. For `MSSC-2S`, we execute t-SNE until convergence, then clustering on z by MSSC 10 times, choosing the solution which has the best objective value. For `MSSC-JDR`, we initialize $z_0^{\texttt{MSSC-JDR}}$ from the 5th iteration of t-SNE after early exaggeration. $u_0^{\texttt{MSSC-JDR}}$ is obtained by executing MSSC 10 times on $z_0^{\texttt{MSSC-JDR}}$ and choose the solution with the lowest objective value. The linear system of the sub-problem is solved by Conjugate Gradient solver with the complexity of $O(n)$.

Datasets: Datasets used in the experiment are downloaded from Feature Selection Repository [2] and UCI Repository [3].

4.2.4.2 Experiment 1: Hyper-parameters of `MSSC-JDR` and `MSSC-2S`

In this experiment, we interest in the performance of `MSSC-JDR` and `MSSC-2S` as their hyper-parameters vary: λ for `MSSC-JDR` and d for both algorithms.

For this experiment, we vary $\lambda \in \{10^{-4}, 10^{-3}, \ldots, 1\}$ and $d \in \{5, 10, 20, 30\}$. Clustering accuracy is reported in Figure 4.1 for 6 datasets (*leukemia*, *GLI_85*, *Carcinom*, *coil20*, *isolet* and *gisette*), due to the large grid-search used in this experiment.

We observe that:

- For `MSSC-2S`, the number of dimensions has a strong effect on the clustering accuracy. In detail, for $d \in \{20, 30\}$, `MSSC-2S` produces the highest clustering accuracy in 4 over 6 cases (*leukemia*, *coil20*, *gissete* and *usps*); which is higher than $d \in \{5, 10\}$ by 1 case.

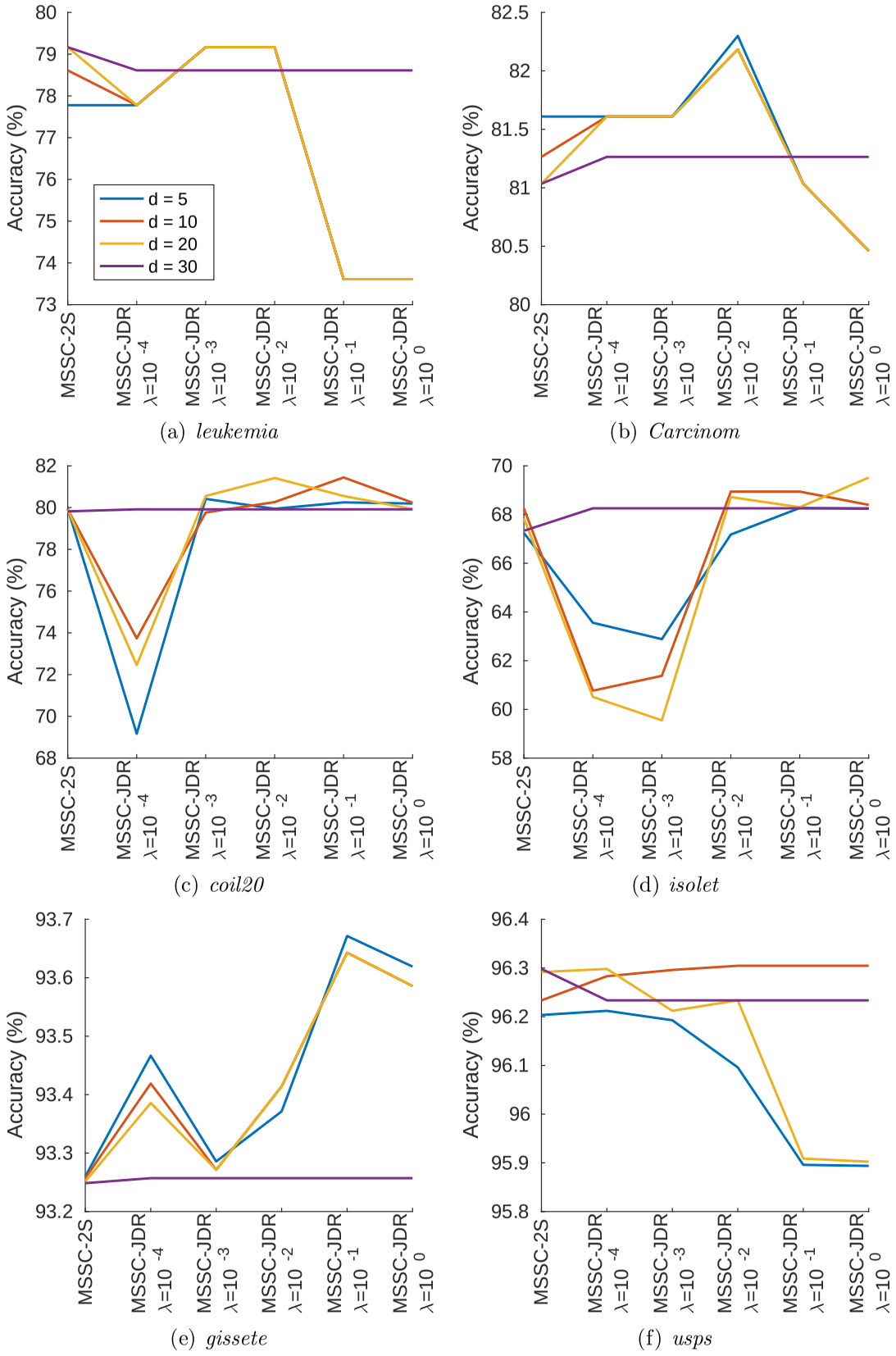
- For `MSSC-JDR`, both λ and d strongly affect clustering accuracy. In detail, the highest accuracy is mostly obtained for $d = 20$ in 4 over 6 cases (*leukemia*, *coil20*, *isolet*, and *usps*); followed by $d = 10$ for 3 over 4 cases (*leukemia*, *coil20* and *usps*). For $d = 20$, $\lambda = 10^{-2}$ produces the best result with the highest accuracy in 4 over 6 cases (*leukemia*, *Carcinom*, *coil20*, *isolet*, *gissete*, and *usps*); followed by $\lambda = 10^{-1}$ with 3 over 4 cases (*leukemia*, *Carcinom*, and *gissete*).

From these observations, we choose $d = 10$ (for both `MSSC-JDR` and `MSSC-2S`) and $\lambda = 10^{-2}$ (for `MSSC-JDR`) in the next experiments.

4.2.4.3 Experiment 2: Comparasion between `MSSC-2S`, `MSSC-JDR` and standard `MSSC`

In this experiment, we interest in studying the effectiveness of `MSSC-2S` and `MSSC-JDR` in compare with the standard `MSSC` algorithm [45], to demonstrate the ef-

2. `http://featureselection.asu.edu/index.php`
3. `https://archive.ics.uci.edu/ml/datasets/isolet`



(a) *leukemia* (b) *Carcinom*

(c) *coil20* (d) *isolet*

(e) *gissete* (f) *usps*

Figure 4.1 – Clustering accuracy of `MSSC-2S` and `MSSC-JDR` as λ varies.

fectiveness of our proposed method.

Parameters of `MSSC-2S` and `MSSC-JDR` are the same as in the Experiment 1, but with the number of reduced dimension $d = 10$ for both algorithms and $\lambda = 10^{-2}$ for `MSSC-JDR`.

The comparative results between algorithms are reported in Table 4.1 and Figure 4.2.

In comparison between `MSSC-2S` and `MSSC-JDR`:
— In terms of clustering accuracy, both `MSSC-2S` and `MSSC-JDR` are clearly better than MSSC in all datasets. The gain is high, from 6.95% (*isolet*) to 48.42% (*emnist-digits*).
— In terms of running time, in general, both `MSSC-2S` and `MSSC-JDR` are slower than `MSSC`. However, `MSSC-JDR` is faster than `MSSC` for 2 biggest datasets (*mnist* and *emnist-digits*). It is worthy to note that the running time for small-size datasets is quite small (lower than 15 minutes), but the save-up in terms of running time for large-scale datasets is enormous: from 8.8 hours (reps. 80 minutes) to 1.5 hours (reps. 10.5 minutes) for *emnist-digits* dataset (reps. *mnist* dataset).

In comparison between `MSSC-JDR` and `MSSC-2S`, `MSSC-JDR` proved to be more efficient, in both terms of clustering quality and running time. In terms of the clustering accuracy, `MSSC-JDR` is better than `MSSC-2S` in 7 over 9 datasets (from 0.01% to 1.09%), and equal in the remaining 2 datasets (*leukemia* and *GLI_85*). As for running time, except for 2 datasets (*isolet* and *gisette*), `MSSC-JDR` is faster than `MSSC-2S` by a large margin, up to 14 times. Most notably, in the three largest datasets (*usps*, *mnist* and *emnist-digits*), the gains are from 3.17 to 14 times, with no drop in accuracy.

In short, both `MSSC-2S` and `MSSC-JDR` out-perform MSSC in term of accuracy (up to 48.42%) but with a trade-off in term of computing time. This result is aligned with previous result on clustering high-dimensional data: the dimension reduction technique improves the clustering accuracy by reducing the number of dimension to cope with *"curse of dimensionality"* phenomenal. `MSSC-JDR` is clearly an enhancement of MSSC for high-dimensional datasets with an agreeable trade-off in terms of computing time; and `MSSC-JDR` also clearly is an improvement over `MSSC-2S`.

4.2.4.4 Experiment 3: Comparison with NMF and VolMin-based factorization joint-clustering algorithms

Comparative algorithms and settings: This experiment compares the performance between our proposed method with jointly dimension reduction and MSSC-based clustering algorithms. We consider the following algorithms:
— `NMFKM` [119]: Two-stage combination of NMF and K-means.
— `JNKM` [119]: Joint NMF and K-means.
— `JVKM` [119]: Joint VolMin and K-means.

For `JNKM` and `JVKM`, the parameters for this experiments is the number of reduced

dimensions F. We choose $F \in \{5, 10, 20, 30\}$ and report the best average result. Others parameters are set as default provided by the authors' source code.

Parameters of `MSSC-JDR` are the same as in Experiment 1, but with the number of reduced dimension $d = 10$ and $\lambda = 10^{-2}$.

Experiment result and comments: The comparative results between algorithms are reported in Table 4.1 and Figure 4.2. We observe that `MSSC-JDR` outperforms others by a large margin in term of clustering accuracy but requires longer running time. The gains in clustering accuracy depends on the datasets, but `MSSC-JDR` is always better or equal: from no gains at all (as in *GLI_85* dataset) up to 38% (as in *mnist* dataset). In terms of computing time, the same situation as in comparison with `MSSC`: the improvement in term of clustering accuracy comes with a trade-off but reasonable computing time.

To sum up, `MSSC-JDR` is the most accurate clustering algorithm among `MSSC-JDR`, `MSSC-2S` and `MSSC`.

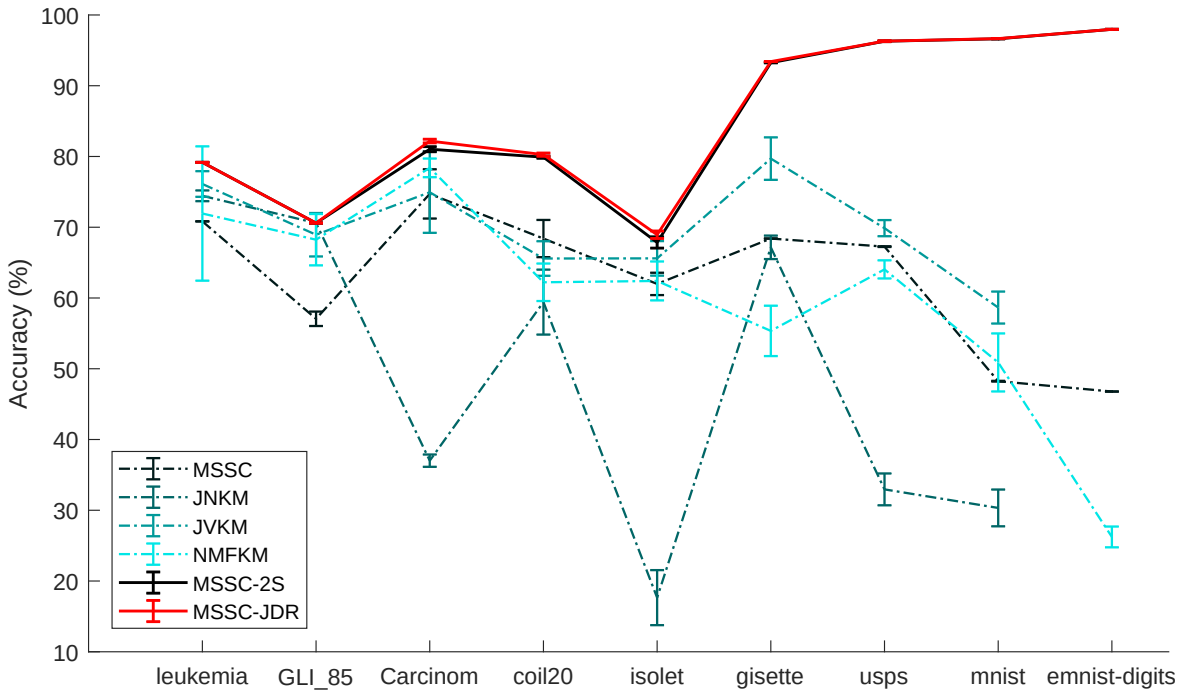


Figure 4.2 – Clustering accuracy of all algorithms. For the last dataset (*emnist-digits*), `JNKM` and `JVKM` encounter the Out Of Memory error.

Table 4.1 – Comparative result between algorithms over all datasets. Bold values correspond to best results for each dataset. NA means the algorithm failed to procedure the result (i.e. Out of Memory).

Dataset	Algorithm	NMI		Accuracy		Time (sec.)	
		Average	STD	Average	STD	Average	STD
leukemia	`MSSC`	12.54%	0.00%	70.83%	0.00%	1.44	0.05

$n = 72$	JNKM	16.47%	0.91%	74.44%	0.76%	1.78	0.30
$d = 7070$	JVKM	18.34%	3.02%	76.11%	1.81%	37.31	0.08
$k = 2$	NMFKM	17.25%	18.21%	71.94%	9.49%	**1.10**	**0.06**
	MSSC-2S	**23.14%**	**0.00%**	**79.17%**	**0.00%**	7.36	0.15
	MSSC-JDR	**23.14%**	**0.00%**	**79.17%**	**0.00%**	3.82	0.05
GLI_85	MSSC	5.00%	0.58%	57.06%	1.02%	6.12	0.21
$n = 85$	JNKM	**25.09%**	**0.00%**	**70.59%**	**0.00%**	**0.77**	**0.02**
$d = 22283$	JVKM	5.27%	2.21%	68.94%	3.07%	340.97	0.18
$k = 2$	NMFKM	22.96%	3.27%	68.24%	3.63%	15.90	0.46
	MSSC-2S	**25.09%**	**0.00%**	**70.59%**	**0.00%**	5.41	0.33
	MSSC-JDR	**25.09%**	**0.00%**	**70.59%**	**0.00%**	3.98	0.16
Carcinom	MSSC	76.23%	3.33%	74.71%	3.48%	7.88	0.23
$n = 174$	JNKM	33.02%	1.48%	37.01%	0.87%	**0.56**	**0.01**
$d = 9182$	JVKM	75.23%	3.93%	74.94%	5.73%	62.13	0.19
$k = 11$	NMFKM	**81.20%**	**2.11%**	78.39%	1.32%	9.74	0.09
	MSSC-2S	79.93%	0.39%	81.03%	0.36%	10.16	0.19
	MSSC-JDR	81.10%	0.28%	**82.18%**	**0.26%**	9.77	0.11
coil20	MSSC	78.37%	0.62%	68.40%	2.63%	13.94	0.21
$n = 1440$	JNKM	74.47%	3.03%	59.42%	4.59%	3.06	2.56
$d = 1024$	JVKM	79.17%	2.13%	65.58%	2.44%	19.47	1.16
$k = 20$	NMFKM	74.07%	1.22%	62.22%	2.65%	**2.58**	**0.32**
	MSSC-2S	86.70%	0.04%	79.90%	0.16%	73.82	4.74
	MSSC-JDR	**86.97%**	**0.01%**	**80.26%**	**0.23%**	71.07	3.82
isolet	MSSC	76.27%	0.66%	61.99%	1.58%	9.80	0.14
$n = 1560$	JNKM	43.64%	0.66%	17.65%	3.89%	**0.71**	**0.08**
$d = 617$	JVKM	77.50%	1.04%	65.60%	2.44%	16.83	7.14
$k = 26$	NMFKM	77.24%	1.47%	62.41%	2.75%	2.86	0.13
	MSSC-2S	**80.30%**	**0.41%**	67.85%	0.80%	63.28	5.50
	MSSC-JDR	80.14%	0.33%	**68.94%**	**0.53%**	80.53	0.68
gisette	MSSC	12.16%	0.01%	68.40%	0.01%	815.42	18.33
$n = 7000$	JNKM	9.80%	1.89%	67.16%	1.66%	162.89	1.40
$d = 5000$	JVKM	30.17%	6.07%	79.70%	3.00%	61.74	1.36
$k = 2$	NMFKM	1.17%	1.47%	55.35%	3.56%	19.90	**0.55**
	MSSC-2S	64.45%	0.03%	93.25%	0.01%	330.37	0.83
	MSSC-JDR	**65.06%**	**0.00%**	**93.41%**	**0.00%**	**27.34**	**3.13**
usps	MSSC	61.35%	0.01%	67.26%	0.05%	83.88	2.40
$n = 9298$	JNKM	29.28%	4.48%	32.95%	2.25%	**2.02**	**0.16**
$d = 256$	JVKM	67.75%	0.85%	69.87%	1.14%	58.17	1.50
$k = 10$	NMFKM	60.11%	0.92%	64.04%	1.28%	4.94	0.14
	MSSC-2S	90.91%	0.11%	96.29%	0.06%	1862.05	95.57

	MSSC-JDR	**90.92%**	**0.13%**	**96.30%**	**0.07%**	583.97	41.56
mnist	MSSC	44.25%	0.02%	48.24%	0.05%	4774.27	171.50
$n = 70000$	JNKM	23.73%	2.26%	30.33%	2.60%	**31.96**	**10.55**
$d = 784$	JVKM	49.94%	1.00%	58.65%	2.26%	201.61	3.50
$k = 10$	NMFKM	44.36%	2.21%	50.89%	4.10%	95.36	3.25
	MSSC-2S	91.41%	0.01%	96.63%	0.00%	2294.60	49.17
	MSSC-JDR	**91.46%**	**0.03%**	**96.66%**	**0.01%**	630.52	24.49
emnist-digits	MSSC	42.86%	0.01%	46.78%	0.00%	31816.56	1,013.58
$n = 280000$	JNKM	NA	NA	NA	NA	NA	NA
$d = 784$	JVKM	NA	NA	NA	NA	NA	NA
$k = 10$	NMFKM	36.67%	0.45%	26.23%	0.63%	274.79	3.61
	MSSC-2S	94.35%	0.01%	**97.97%**	**0.01%**	78962.85	5078.14
	MSSC-JDR	**94.36%**	**0.00%**	**97.97%**	**0.00%**	**5623.78**	691.53

4.2.4.5 Experiment 4: Compare with joint-clustering algorithms using auto-encoder

This experiment compares our proposed approach with auto-encoder-based joint-clustering algorithms on image datasets. We consider the following ones:

— DC-Kmeans [104] proposes an algorithm for solving the joint-clustering of Deep Auto-Encoder and K-means by Alternating Direction of Multiplier Method (ADMM).
— DEC [117] uses a two-step procedure: the first step consists of training the auto-encoder using reconstruction loss; then the second step jointly optimizes the encoder and soft assignment clustering loss function for dimension reduction and clustering respectively.
— IDEC [33] (Improved DEC) argues the two-step procedure in DEC could weaken the representatives of the embedded features and hurts clustering performances as DEC strips out the decoder part during the optimization the joint-clustering; hence it simultaneously optimizes the full auto-encoder and clustering by soft assignment clustering as in DEC.

Setting for DC-Kmeans: we use the authors' implementation [4]. We set the number of neural in the bottle-neck layer equals to the number of classes for all datasets. For training the auto-encoder, we use Matlab's neural network toolbox. We also notice that the author's scheme for initializing K-means' centroids often produces bad results (especially for *coil20* dataset), so we use to the same procedure for initial point as MSSC-JDR.

4. https://github.com/JennyQQL/DeepClusterADMM-Release

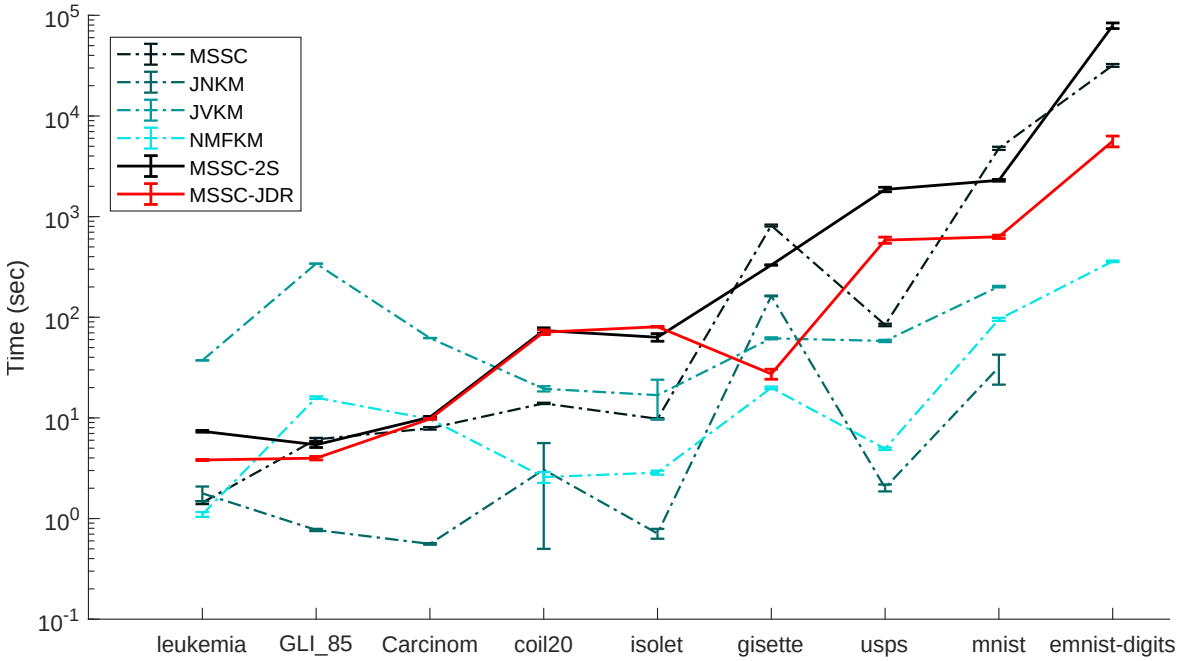


Figure 4.3 – Running time (in seconds) of all algorithms. For the last dataset (*emnist-digits*), `JNKM` and `JVKM` encounter the Out Of Memory error.

Setting for `IDEC` and `DEC`: we use the implementation from `IDEC`'s authors [5] [6] for pre-train the auto-encoder and for applying the joint-clustering algorithm respectively.

Setting for `MSSC-JDR` is the same as in the Experiment 1, but with the number of reduced dimension $d = 10$ and $\lambda = 10^{-2}$.

Note that, there would be different from the original work's result due to the randomness.

Result in this experiment is summarized in Table 4.2. We observe that `MSSC-JDR` is very competitive in general. In terms of both NMI and clustering accuracy, `MSSC-JDR` has the best result in all 4 datasets. The improvement is huge: the gap to the second-best alogrithm in terms of clustering accuracy is from 5.92% to 20.32%. In terms of running time, DEC is the fastest in 2 smaller datasets (*coil20* and *usps*), whereas `MSSC-JDR` is the fastest in the 2 bigger ones (*mnist* and *emnist-digits*). It is clear that `MSSC-JDR` outperforms considering deep clustering algorithms (`DC-Kmeans`, `DEC` and `IDEC`) in clustering quality and running time, especially in large-scale dataset.

5. `https://github.com/XifengGuo/IDEC`
6. `https://github.com/XifengGuo/IDEC-toy`

Table 4.2 – Comparative result between `MSSC-JDR` and Auto-encoder-based joint-clustering algorithms. Bold values correspond to best results for each dataset. *NA* means the algorithm failed to produced a result under 12 hours.

Dataset	Algorithms	NMI		Accuracy		Time		Environment
		Average	STD	Average	STD	Average	STD	
coil20	`DC-Kmeans`	75.35%	0.85%	62.43%	2.09%	810.39	0.04	CPU
$n = 1440$	`DEC`	67.16%	0.77%	41.25%	1.69%	**28.51**	**0.87**	GPU
$d = 1024$	`IDEC`	67.58%	1.18%	40.68%	1.83%	36.84	1.28	GPU
$k = 20$	`MSSC-JDR`	**86.97%**	**0.01%**	**80.26%**	**0.23%**	71.07	3.82	CPU
usps	`DC-Kmeans`	55.26%	0.11%	60.55%	0.16%	5,006.83	19.35	CPU
$n = 9298$	`DEC`	77.75%	0.81%	75.98%	0.54%	**20.06**	**1.70**	GPU
$d = 256$	`IDEC`	77.38%	0.86%	75.78%	0.62%	218.79	13.29	GPU
$k = 10$	`MSSC-JDR`	**90.92%**	**0.13%**	**96.30%**	**0.07%**	583.97	41.56	CPU
mnist	`DC-Kmeans`	75.65%	0.02%	78.04%	0.03%	39,840.73	635.35	CPU
$n = 70000$	`DEC`	84.65%	0.64%	87.19%	3.21%	2,377.82	520.98	GPU
$d = 784$	`IDEC`	88.98%	1.53%	90.74%	4.56%	2,135.05	98.4	GPU
$k = 10$	`MSSC-JDR`	**91.46%**	**0.03%**	**96.66%**	**0.01%**	**630.52**	**24.49**	GPU
emnist-digits	`DC-Kmeans`	NA	NA	NA	NA	NA	NA	CPU
$n = 280000$	`DEC`	86.05%	-	82.85%	-	5,899.98	-	GPU
$d = 784$	`IDEC`	89.05%	0.12%	84.74%	0.17%	8,762.10	135.31	GPU
$k = 10$	`MSSC-JDR`	**94.36%**	**0.00%**	**97.97%**	**0.00%**	**5,623.78**	**691.53**	CPU

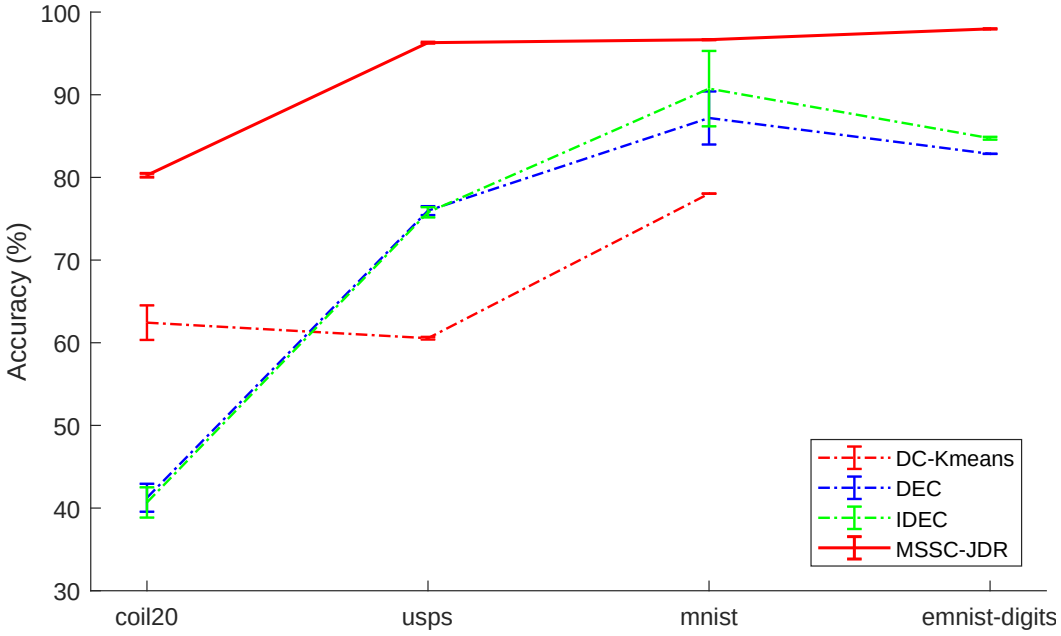


Figure 4.4 – Clustering accuracy of `MSSC-JDR` and AE-based joint-clustering algorithms. For the last dataset (*emnist-digits*), `DC-Kmeans` failed to give result due to the time limitation of 24 hours.

4.3 An approach for the scaling problem in a class of joint-clustering algorithms by auto-encoder

4.3.1 Auto-encoder

According to the definition from [31], auto-encoder is *"a neural network that is trained to attempt to copy its input to its output"*. It is a powerful tool to train a mapping from original data to code, which minimizes the reconstruction error from such code. Usually, the dimension of the code is smaller than the one of data, hence auto-encoder traditionally has been used for dimension reduction or feature learning. In this study, we focus on stacked auto-encoder - a variance of auto-encoder which is widely used in clustering.

The standard stacked auto-encoder consists of two components: an encoder and a decoder. An encoder f^E (reps. decoder f^D) is a neural network parametrized by parameter θ_E (reps. θ_D), maps data from $\mathbb{R}^m \to \mathbb{R}^d$ (reps. $\mathbb{R}^d \to \mathbb{R}^m$). Given a raw input data point $x_i \in \mathbb{R}^m$ (i.e. an image, a document, etc.), the encoder first produces code $z_i \in \mathbb{R}^d$, then the decoder reconstructs $\hat{x_i}$ of x_i only from code z_i (figure 4.5 illustrates the corresponding auto-encoder). By reconstructing the input, the network can learn the under-complete but informative representation from the raw input data. Mathematically, given a data-set $\{x_i\}_{i=1,\dots,n}$ where $x_i \in \mathbb{R}^m$, the problem of auto-encoder reconstruction can be defined as

$$\min_{\theta_E,\theta_D} \left\{ F^{AE}(\theta_E,\theta_D) = \sum_{i=1}^{n} \ell(x_i, f^D(\theta_D, f^E(\theta_E, x_i))) \right\}, \tag{4.22}$$

where ℓ measures the reconstruction error. Common choices for ℓ are mean squared error, binary cross-entropy, ℓ_1 norm, etc. In this work, we consider the square error $\ell(x,y) = \|x-y\|_2^2$.

The autoencoder $f^{AE} = f^D \circ f^E$ can be seen as a neural network of L layers, where $f^D = f^{(L)} \circ \cdots \circ f^{(l)}$ and $f^E = f^{(l)} \circ \cdots \circ f^{(1)}$. Each function $f^{(i)}$ represents a layer of the neural network, which maps the output of the previous layer $z^{(i-1)}$ into new code $z^{(i)} = h(z, \theta^{(i-1)})$ where h is the activation function. A typical choice for h is linear function ($h_{\text{linear}}(z,\theta) = \phi(z,\theta)$) or non-linear function such as ReLU [79] ($h_{\text{ReLU}}(z,\theta) = \max(\phi(z,\theta), 0)$, where max is a element-wise operator), where $\phi(z,\theta)$ is the matrix multiplication operators (dense layer in neural network) or convolution

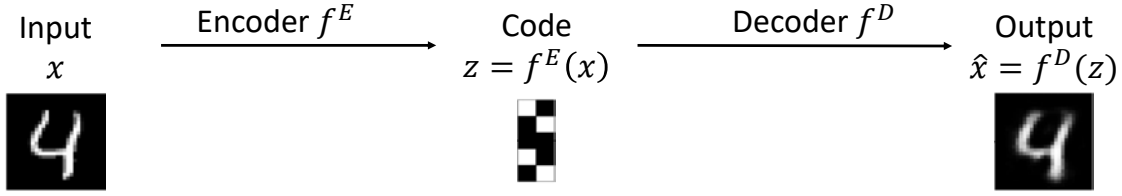


Figure 4.5 – Illustration of an auto-encoder.

operation (convolution layer in convolution neural network).

4.3.2 Scaling problem of joint-clustering by auto-encoder

Let us consider the problem of deep joint-clustering by MSSC. Several works [99, 104, 120, 129, 32, 43] combines auto-encoder with K-means (MSSC) into an optimization problem by using the linear scalarization technique of multi-objective programing. Formally, this problem is defined as

$$\begin{aligned}
\min_{\theta_E,\theta_D,u,s} \quad & F^{\text{J-MSSC}}(\theta_E,\theta_D,u,s) := F^{AE}(\theta_E,\theta_D) + \lambda F^{\text{IP-MSSC}}(\theta_E,u,s) \qquad (4.23)\\
& = \sum_{i=1}^{n} \|x_i - f^D(\theta_D, f^E(\theta_E,x_i))\|_2^2 + \lambda \|us - \ f^E(\theta_E,x_i)\|_2^2\\
\text{s.t.} \quad & s_{j,i} \in \{0,1\} \quad \text{for} \quad i = 1,\dots,n \text{ and } j = 1,\dots,k,\\
& \sum_j s_{i,j} = 1 \quad \text{for} \quad i = 1,\dots,n,
\end{aligned}$$

or

$$\begin{aligned}
\min_{\theta_E,\theta_D,u} \quad & F^{\text{J-MSSC}}(\theta_E,\theta_D,u,s) := F^{AE}(\theta_E,\theta_D) + \lambda F^{\text{Bi-MSSC}}(\theta_E,u) \qquad (4.24)\\
& = \sum_{i=1}^{n} \left(\|x_i - f^D(\theta_D, f^E(\theta_E,x_i))\|_2^2 + \lambda \min_{l=1\dots k} \|u_l - \ f^E(\theta_E,x_i)\|_2^2 \right),
\end{aligned}$$

where λ controls the trade-off between two terms; $u \in \mathbb{R}^{k\times m}$ are k centroids in latent space.

However, problem (4.23) and (4.24) both suffer the scaling problem in latent space, which is explained as following. Let l is the bottle-neck layer of the L-layer auto-encoder, i.e. $f^E = f^{(l)} \circ f^{(l-1)} \circ \cdots \circ f^{(1)}$ and $f^D = f^{(L)} \circ \cdots \circ f^{(l+1)}$, where $f^{(i)}$ is the transformation function of i-th layer/block. In modern auto-encoder in general and in deep clustering in particular, $f^{(l)}$ and $f^{(l+1)}$ are linear function or sometimes are nonlinear activation function (such as ReLU):

$$\begin{aligned}
\text{ReLU:} \quad & f^{(i)}(\theta^{(i)}, z^{(i-1)}) &&= \text{ReLU}(\theta^{(i)} z^{(i-1)}),\\
\text{Linear activation:} \quad & f^{(i)}(\theta^{(i)}, z^{(i-1)}) &&= \theta^{(i)} z^{(i-1)},
\end{aligned}$$

for $i \in \{l, l-1\}$, and $z^{(i)}$ is the output of the i-th layer. For these networks, the first terms F^{AE} is invariance to the scaling of $z_i^{(l)} = f^E(\theta_E, x_i)$, but the second term F^{MSSC} scale quadratically to $z_i^{(l)}$.

4.3.3 Proposed solution

4.3.3.1 Spherical distance

Instead of computing the Euclidean distance on $\mathbb{R}^m$ space, we measure the distance on the projection of data points on the surface of the unit hypersphere $\mathbb{S}^{m-1}$ instead.

By only considering the distance between projections, we eliminate the magnitude of z_i but only its direction. Among them, the cosine distance function is a popular distance function, which has been used in several clustering algorithms, notably by Spherical K-means [24]. It is defined as

$$d_{\text{cosine}}(z_i, z_j) = 1 - \left\langle \frac{z_i}{\|z_i\|_2}, \frac{z_j}{\|z_j\|_2} \right\rangle.$$

This distance function has recently been used in deep clustering problem [5], and have demonstrated its effectiveness in comparison with other regularization methods such as Batch Normalization and Layer Normalization. However, the cosine distance is not a metric because it violates the triangular inequality. In addition, consider the case where comparing the distance between two projects on the surface of the hypersphere $\mathbb{S}^{m-1}$. Let us consider the example in Figure 4.6. Since $\bar{x}_i$ and $\bar{x}_j$ are the projection of x_i and x_j onto the hypersphere, measuring the arc length between $\bar{x}_i$ and $\bar{x}_j$ (length of the dashed orange arc) is more suitable than the Euclidean distance between them (length of the solid blue line segment). Hence, instead of using the d_{cosine}, we employ the $d_{\text{spherical}}$ as

$$d_{\text{spherical}}(z_i, z_j) = \frac{\arccos(s_{\text{cosine}}(z_i, z_j))}{\pi} = \frac{1}{\pi}\arccos\left\langle \frac{z_i}{\|z_i\|_2}, \frac{z_j}{\|z_j\|_2} \right\rangle,$$

where $\arccos(\alpha)$ is the inverse cosine 1function for $\alpha \in [-1, 1]$. To avoid numerical problem of $\|z_i\|_2 = 0$, we slightly modify $d_{\text{spherical}}$ to

$$d^{\epsilon}_{\text{spherical}}(z_i, z_j) = \frac{1}{\pi}\arccos\left\langle \frac{z_i}{\|z_i\|_2 + \epsilon}, \frac{z_j}{\|z_j\|_2 + \epsilon} \right\rangle, \tag{4.25}$$

for a very small value of ϵ.

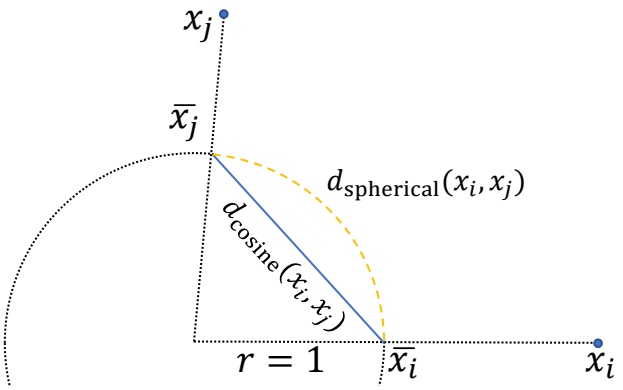


Figure 4.6 – Illustration of the spherical distance. The solid blue line segment (reps. dashed orange arc) represents the Euclidean distance between $\bar{x}_i$ and $\bar{x}_i$ (reps. spherical distance), which are the projection of x_i and x_j onto the hypersphere (i.e. $\bar{x} = \frac{x}{\|x\|_2}$).

4.3.3.2 Application for deep joint-clustering with MSSC

In this section, we applied the proposed extension in section 4.3.2 for the specific problem (4.24). Modifying the distance in problem (4.24) from squared Euclidean distance to $d^{\epsilon}_{\text{spherical}}$ leads to the

$$\begin{aligned}\min_{\theta_E,\theta_D,u} \quad & F^{\text{J-Spherical}_\epsilon}(\theta_E,\theta_D,u,s) := F^{AE}(\theta_E,\theta_D) + \lambda F^{\text{Spherical}_\epsilon}(\theta_E,u) \\ & = \sum_{i=1}^{n} \|x_i - f^D(\theta_D, f^E(\theta_E,x_i))\|_2^2 + \lambda \sum_{i=1}^{n} \min_{l=1,\dots,k} d^{\epsilon}_{\text{spherical}}(f^E(\theta_E,x_i),u_l),\end{aligned} \tag{4.26}$$

where λ is the trade-off parameter.

Motivated by the success of Adam algorithm [41] in deep learning, especially for solving the first term $F^{AE}(\theta_E,\theta_D)$ in (4.26), we adopt Adam to solve our problem. The problem (4.26) is difficult due to the non-differentiable of the second term $F^{\text{Spherical}_\epsilon}$. We apply the following smoothing technique for min function with $\alpha_s > 0$,

$$\min_{l=1,\dots,k} r_l \approx \text{LSE}_{\alpha_s}(r) = -\alpha_s \log \sum_{l=1}^{k} \exp(-\alpha_s r_l),$$

which turns problem (4.26) into following optimization problem

$$\min_{\theta_E,\theta_D,u} \quad F^{\text{J-Spherical}_\epsilon}(\theta_E,\theta_D,u,s) := F^{AE}(\theta_E,\theta_D) + \lambda F^{\text{Smooth-Spherical}_\epsilon}(\theta_E,u) \tag{4.27}$$

$$= \sum_{i=1}^{n} \|x_i - f^D(\theta_D, f^E(\theta_E,x_i))\|_2^2 + \sum_{i=1}^{n} \frac{-\lambda\alpha}{\pi} \log \sum_{l=1}^{k} \exp -\alpha d^{\epsilon}_{\text{spherical}}(f^E(\theta_E,x_i),u_l).$$

Each iteration of Adam for solving the problem (4.27) requires computing gradient $\nabla_{\theta_E,\theta_D,u} F^J$. The computation of $\nabla_{\theta_E,\theta_D} F^J$ can be calculated by the back-propagation algorithm [95], and the computation for $\nabla_u F^J$ is computed by

$$\frac{\partial F^J(u)}{\partial u_l} = \frac{\alpha^2\lambda}{\pi} \sum_{i=1}^{n} \text{LSE}_1(\alpha t_i)\, \text{softmax}(\alpha t_i) \frac{\frac{1}{\|u_l\|_2+\epsilon}\mathbb{I} - \frac{1}{\|u_l\|_2(\|u_l\|_2+\epsilon)^2} u_l u_l^T}{\sqrt{1 - \left\langle z_i, \frac{u_l}{\|u_l\|_2+\epsilon} \right\rangle}} z_i, \tag{4.28}$$

where $\text{softmax}(\bar{t}) = \frac{\exp(-\bar{t})}{\sum_l \exp(-\bar{t}_l)}$, $t_i = (t_l^{(i)})_{l=1,\dots,k} = \left(\arccos \left\langle z_i, \frac{u_l}{\|u_l\|_2} \right\rangle\right)_{l=1,\dots,k}$ and $z_i = \frac{F^E(\theta_E,x_i)}{\|F^E(\theta_E^0,x_i)\|}$, $i = 1,\dots,n$.

The Adam applied for problem (4.26) is outlined in Algorithm 4.6.

Similar to `MSSC-JAE-Sphere`, `MSSC-JAE-Cosine` solves the joint-clustering problem of auto-encoder with MSSC using the cosine distance (problem (4.26) with

Algorithm 4.6 `MSSC-JAE-Sphere`: Adam applied for problem (4.26)

Input: Data x, number of clusters k, trade-off parameter λ, smooth parameter α, batch-size b, Adam's parameter $(\alpha, \beta_1, \beta_2)$

Initialization:

Initialize θ_E, θ_D (by pre-train of by random initialization).

Initialize u (by random initialization or by clustering on $z = f^E(\theta_E, x)$).

repeat

Sample a batch x^t.

Compute $G^t = \nabla_{\theta_E,\theta_D,u} F^{\text{J-Spherical}}(\theta_E^t, \theta_D^t, u^t)$ with data x^t where

$\nabla_{\theta_E,\theta_D} F^{\text{J-Spherical}_\epsilon}(\theta_E^t, \theta_D^t, u^t)$ is computed by back-propagation and

$\nabla_u F^{\text{J-Spherical}_\epsilon}(\theta_E^t, \theta_D^t, u^t)$ is computed by (4.28).

Update $(\theta_E, \theta_D, \mu)$ using Adam with G^t.

$t \leftarrow t + 1$.

until Stopping condition.

d_{cosine} instead of $d^{\epsilon}_{\text{spherical}}$) by Adam. The difference is the step of computing $\nabla_u F^{\text{J-Spherical}_\epsilon}(\theta_E^t, \theta_D^t, u^t)$, which is done automatically by autograd [7].

4.3.4 Numerical experiment

4.3.4.1 Datasets

To study the performances of clustering algorithms, we consider the following image and text dataset(s), which are all widely used to benchmark clustering algorithms:

— `mnist`: The `mnist` dataset [68] consists of 70000 gray-scale 28×28 images over 10 classes of handwritten digits.

— `fashion`: The `fashion` dataset [116] consists of 70000 gray-scale 28×28 images. `fashion` contains 10 classes of clothing items (shirt, dress, shoe, bag, etc.).

— `usps`: Similar to `mnist` dataset, the `usps` dataset consists of 9298 gray-scale 16×16 images over 10 classes of handwritten digits.

— `rcv1`: Similar to [117, 120], we used a subset of 10000 documents from the full RCV1–v2 corpus (Reuters Corpus Volume 1 Version 2) of the four largest classes. Following the procedure in [117], we represent each document by a tf-idf vector of the 2000 most frequently occurring words.

4.3.4.2 Comparative algorithms

We compare the proposed methods (`MSSC-JAE-Sphere` and `MSSC-JAE-Cosine`) with the following baselines:

— `MSSC`: MSSC clustering for raw data.

7. `https://pytorch.org/docs/stable/autograd.html`

— `AE-MSSC`: The 2-step approach, which an auto-encoder is first applied for dimensionality reduction, followed by MSSC for clustering
— `DC-Kmeans` [104] solves an alternative of problem (4.23) by Alternating Direction of Multiplier Method (ADMM).
— `DCN` [120] solves the problem (4.23) by their proposed alternating stochastic gradient algorithm.

Auto-encoder settings: For a fair comparison, we following the setting from [120]. For all algorithms, we follow the standard architecture for clustering with auto-encoder: the number of node in the encoder is $m - 500 - 500 - 2000 - k$, where m is the number of features in the dataset and k is the number of clusters; and the decoder is the mirror of the encoder. The activation function of embedding layer and the last layer are linear, whereas the rest are ReLU. Unless specified otherwise, the auto-encoder is initialized follow the "Xavier Uniform" scheme [29] and pre-train by Adam with learning rate $\alpha = 10^{-3}$ and $(\beta_1, \beta_2) = (0.9, 0.999)$ for 50 epochs with batch-size of 256.

Setting for `MSSC-JAE-Cosine`: we use the Adam optimizer with learning rate $\alpha = 3 \times 10^{-4}$, $(\beta_1, \beta_2) = (0.9, 0.999)$ and batch-size of 256. The smooth parameter $\alpha_s = 16$. The algorithm stop when either of the following criteria is met: (1) 500 epochs or (2) convergence ($\frac{F^J(\theta^k,u^k)-F^J(\theta^{k-1},u^{k-1})}{F^J(\theta^{k-1},u^{k-1})} < \epsilon$ or $\|(\theta,u)^k - (\theta,u)^{k-1}\|_2 < \epsilon$). For initial point, the weighted $\theta^0 = (\theta_E^0, \theta_D^0)$ is obtained by the procedure above. u^0 is initialized as the best result (by objective value) among 10 runs of K-means on extracted feature $\left\{\frac{F^E(\theta_E^0,x_i)}{\|F^E(\theta_E^0,x_i)\|}\right\}_{i=1,\dots,n}$. The activation in auto-encoder is Soft Plus - a smooth approximation of ReLU: $\text{Softplus}(x) = \frac{1}{\beta}\log(1+\exp(\beta x))$, with $\beta = 256$.

Setting for `MSSC-JAE-Sphere`: the setting is the same as `MSSC-JAE-Cosine`'s except u^0. We set $\epsilon = 10^{-4}$ for the $d^{\epsilon}_{\text{spherical}}$. For u^0, we solve the `MSSC-Sphere` problem (MSSC with $d^{\epsilon}_{\text{spherical}}$ instead of $d_{\text{Squared Euclidean}}$) by Adam optimizer (default parameters) for 10 runs for extracted data $\left\{F^E(\theta_E^0, x_i)\right\}_{i=1,\dots,n}$. u^0 is selected as the result whose objective value is smallest. The algorithm is implemented in PyTorch [8].

Setting for `DC-Kmeans`: We use the authors' implementation [9]. For training the auto-encoder, we use Matlab's neural network toolbox. We also notice that the initial point scheme for K-means from the author's implementation often produces bad results, so we use to the same procedure for initial point as `MSSC-JAE-Cosine` but with extracted feature $\{F^E(\theta_E^0, x_i)\}_{i=1,\dots,n}$. For `DC-Kmeans`, the authors set $\lambda = 1$.

Setting for `DCN`: We use the source code available at [10].

8. `https://pytorch.org/`
9. `https://github.com/JennyQQL/DeepClusterADMM-Release`
10. `https://github.com/MaziarMF/deep-k-means`

4.3.4.3 Experiment setting

Evaluation criteria: Given an input x_i with ground-truth label l_i; and p_i is the assignment label from clustering algorithm, we measure the following criteria to evaluate experimental results:

— **Clustering Accuracy (ACC [13])**: ACC is calculated as $ACC(l, p) = \frac{1}{n}\sum_{i=1}^{n} 1_{m(p_i)=l_i}$, where $m(x_i)$ is the function which maps each clustering assign in p_i to the equivalent label l_i. In our case, we use the mapping by using the Kuhn-Munkres algorithm [73].

— **Normalized mutual information (NMI [102])**: The NMI criterion is calculated as $\text{NMI}(l, p) = \frac{I(l,p)}{\sqrt{H(l)H(p)}}$, where $I(l, p)$ is the mutual information of l and p.

All algorithms in our experiment (except `MSSC` and `AE+KM`) has a hyper parameter λ controls the trade-off between the auto-encoder and the clustering. For choosing the λ, we performs a grid search $\lambda \in \{10^{-7}, 10^{-6}, \ldots, 10^{2}\}$. We repeat the experiment 10 times, select the results which has the highest accuracy among 10 runs, and report the average with standard deviation of each criterion.

All experiments are conducted on a Intel(R) Xeon (R) CPU E5-2630 v4 @2.20 GHz with 32GB of RAM and a GTX 1080 GPU.

4.3.4.4 Experiment results

The experiment result of all datasets is reported in Table 4.3.

The results show that reducing the number of dimension by auto-encoder facilitates the clustering process: the increase in accuracy by using the 2-step approach (`AE+MSSC`) over clustering on raw data (`MSSC`) is significant. The gap in clustering accuracy is up to 32.56% as in *mnist* dataset. However, the final result of both `MSSC` and `AE+MSSC` do not exceed the joint-clustering approach (`DCN`, `MSSC-JAE-Cosine`, and `MSSC-JAE-Sphere`). `DCN` further improves the accuracy of `AE+MSSC` among all 4 datasets, range from 0.33% to 2.68%.

The proposed methods (`MSSC-JAE-Cosine` and `MSSC-JAE-Sphere`) further improve the clustering quality over `DCN`. The increase in terms of clustering accuracy is consistent, ranging from 1.21% to 7.63% (reps. from 2.48% to 8.17%) for `MSSC-JAE-Cosine` (reps. `MSSC-JAE-Sphere`). Both `MSSC-JAE-Cosine` and `MSSC-JAE-Sphere` produces better results than `DCN`. The NMI of both results of `MSSC-JAE-Cosine` and `MSSC-JAE-Sphere` is higher than `DCN`'s, up to 3.90% and 6.72% respectively. This result demonstrates the importance of regularization on the latent space, which is achieved in this case by projection the data point's representation in the latent space onto the ℓ_2 ball. In addition, among two methods that utilize the ℓ_2 projection in the latent space, `MSSC-JAE-Sphere` is undoubtedly better than `MSSC-JAE-Cosine`: the gap in clustering accuracy is from 0.22% (*usps* dataset) to 2.5% (*mnist* dataset). This

Table 4.3 – Comparative result between Auto-encoder-based joint-clustering algorithms. Bold values correspond to best results for each dataset. NA means that the algorithm fails to furnish a result.

Dataset	Algorithms	NMI		Accuracy	
		Average	STD	Average	STD
usps	`MSSC`	61.35%	0.01%	67.26%	0.05%
$n = 9298$	`AE+MSSC`	65.41%	1.09%	69.14%	0.52%
$d = 256$	`DC-Kmeans`	55.26%	0.11%	60.55%	0.16%
$k = 10$	`DCN`	69.68%	0.60%	71.21%	0.29%
	`MSSC-JAE-Cosine`	**70.59%**	**1.73%**	73.46%	0.62%
	`MSSC-JAE-Sphere`	69.98%	1.11%	**73.68%**	**0.79%**
rcv1	`MSSC`	31.30%	5.40%	50.80%	2.90%
$n = 10000$	`AE+MSSC`	35.99%	5.47%	55.36%	4.70%
$d = 2000$	`DC-Kmeans`	NA	NA	NA	NA
$k = 4$	`DCN`	31.54%	4.58%	58.05%	4.74%
	`MSSC-JAE-Cosine`	34.80%	10.26%	61.69%	9.27%
	`MSSC-JAE-Sphere`	**38.27%**	**5.55%**	**64.19%**	**6.09%**
mnist	`MSSC`	44.25%	0.02%	48.24%	0.05%
$n = 70000$	`AE+MSSC`	75.63%	0.54%	81.23%	1.83%
$d = 784$	`DC-Kmeans`	75.65%	0.02%	78.04%	0.02%
$k = 10$	`DCN`	76.96%	0.70%	83.83%	1.31%
	`MSSC-JAE-Cosine`	80.86%	0.75%	85.04%	2.30%
	`MSSC-JAE-Sphere`	**82.81%**	**2.04%**	**86.85%**	**6.44%**
fashion	`MSSC`	51.24%	0.01%	53.99%	0.07%
$n = 70000$	`AE+MSSC`	55.48%	0.72%	53.04%	1.99%
$d = 784$	`DC-Kmeans`	51.64%	2.96%	47.61%	2.44%
$k = 10$	`DCN`	56.51%	0.58%	53.37%	1.18%
	`MSSC-JAE-Cosine`	60.21%	1.15%	61.01%	2.58%
	`MSSC-JAE-Sphere`	**61.34%**	**0.67%**	**61.54%**	**3.25%**

increase indicates the importance of using the appropriate distance measure in the latent space.

In conclusion, both `MSSC-JAE-Cosine` and `MSSC-JAE-Sphere` are the improvement over `DCN` and `DC-Kmeans`. In addition, `MSSC-JAE-Sphere` is better than `MSSC-JAE-Cosine`, which demonstrates the importance of using the appropriate distance measure.

4.4 Conclusion

In this chapter, we tackle the deep clustering problems in two directions. The first direction considers the usage of t-SNE (t-Distributed Stochastic Neighbor Embedding) in clustering. We proposed two clustering methods for high-dimensional data based on DC Programming and DCA, named `MSSC-2S` and `MSSC-JDR`. `MSSC-2S` is a straight-forward application of two high-efficiency DCA methods, Minimum Sum-of-Squares Clustering and t-SNE, where both have proved their effectiveness in the literature. `MSSC-JDR` follows the joint-clustering approach to By exploiting the particular structure of the problem, we have recast the joint-clustering problem as a DC program and proposed `MSSC-JDR` – a DCA-Like algorithm. `MSSC-JDR` solves two convex sub-problems at each iteration, both are relatively fast: one requires only elementary operations on vectors, the remaining one is solving a sparse linear system. Both methods are applicable for many applications given an appropriate distance function. The numerical results show that the proposed methods significantly improve the clustering quality in comparison with state-of-the-art methods in deep clustering, up to 38% higher than the second-best method. Furthermore, `MSSC-JDR` significantly reduces the running time of `MSSC-2S` (up to 14 times) while having similar clustering accuracy. We are convinced that the proposed methods are efficient, fast, and scalable for clustering high-dimensional data.

The second direction considers joint-clustering using auto-encoder. We proposed an extension for deep joint-clustering problems by using cosine and spherical distance measure, which is applicable when the derived optimization problems suffer from the scaling of data's representation in the latent space. Both distance measures are invariance to scaling since they compute the distance between projections of data points onto the surface of the unit hypersphere $\mathbb{S}^{m-1}$ instead of between the data points in $\mathbb{R}^m$. As an application, we considered the specific problem of deep joint-clustering with MSSC and proposed two algorithms (`MSSC-JAE-Cosine` and `MSSC-JAE-Sphere`) to solve the mentioned problem. The numerical results present the effectiveness of proposed algorithms in comparison with state-of-the-art methods in joint-clustering by K-means. The clustering accuracy is higher in comparison with the second-best method (from 3.90% to 6.72%). In addition, `MSSC-JAE-Sphere` improves the clustering accuracy `MSSC-JAE-Cosine` by up to 2.5%, which indicates the importance of using the appropriate distance measure in the latent space.

Each developed method has its advantages and disadvantages. The first method – clustering based on t-SNE – works well in case we can define a distance function to measure the similarity between data-points in the original space. On the other hand, since the second approach based on the auto-encoder, it seeks the mapping function solely based on input data, hence eliminating the needs of the similarity function. However, the auto-encoder-based approach requires a massive amount of data to work well, so in case we can gather or generate data [32], this is a preferable choice. Hence, the choice is the trade-off between expert knowledge and data-driven approach: how can we define the similarity function versus the amount of the data we have.

In conclusion, the proposed methods have demonstrated their effectiveness in both terms of clustering accuracy and running time in comparison to the relevance methods. In the future, we could extend `MSSC-JAE-Sphere` to consider other unit spheres other than ℓ_2 such as ℓ_p ball ($p \in \{1, \infty\}$). On another direction, we could consider the combination between t-SNE and neural network for deep joint-clustering problems. `MSSC-JDR` based on t-SNE is better than `MSSC-JAE-Sphere` in clustering accuracy, nevertheless, `MSSC-JDR` requires a distance measure to capture the similarity between data points in the original whereas `MSSC-JAE-Sphere` does not need. The combination between them is interesting since it could take the advantages from both methods.

Chapter 5

Conclusion

This dissertation focus on three problems in machine learning on big data: group variable selection in multi-class logistic regression, dimension reduction by t-SNE and deep clustering. The principal methodologies of this dissertation are DC (Difference of Convex functions) programming and DCA (DC Algorithm) and their advanced variances, which are well-known powerful tools due to their effectiveness and efficiency for non-smooth and non-convex optimization problems.

The first part rigorously studied the DC programming and DCA for the problem of group variables selection in multi-class logistic regression. Using the $\ell_{q,0}$ regularization, the resulting optimization problem is non-convex. The $\ell_{q,0}$-norm is approximated by the piecewise exponential function and the capped-ℓ_1 function. We studied DCA for this problem and developed DCA-based algorithms to solve the approximated problem by two approaches based on recent advanced DC algorithms. The proposed algorithm in the first approach, `SDCA`, is based on Stochastic DCA. `SDCA` is very inexpensive and suitable for large-scale datasets. In the second approach, we developed two algorithms based on DCA-Like and its accelerated version ADCA-Like. Both algorithms extend DCA by relaxing the DCA's convexity condition on the second DC component, while still ensuring the convergence. Unlike standard DCA where we need to compute the parameter μ greater or equal to the L-Lipschitz constant, which is difficult (or even impossible) in practice; DCA-Like iteratively updates the parameter μ and consequently the decomposition of the objective function. The proposed algorithms are efficient: all computations are explicit.

The numerical experiments have shown that all proposed algorithms significantly improve the running time in comparison with standard DCA while having equivalent classification accuracy and sparsity. They also give better results than the related algorithms in the literature in both terms of classification accuracy and running time. Among the proposed algorithms, choosing an appropriate algorithm depends on the dataset's size and the number of the selected variables. `SDCA` is more suitable for large datasets, whereas `ADCA-Like` and `DCA-Like` are ideal if we prefer a more compact subset of variables. We are convinced that proposed algorithms are efficient for the problem of group variable selection in multi-class logistic regression.

In the second part, we studied DC programming and DCA for the t-SNE problem. The two proposed algorithms are based on DCA-Like and ADCA-Like, which are inexpensive: the solution of the convex sub-problem can be explicitly computed. We showed that the Majorization-Minimization algorithm, the best state-of-the-art algorithm for t-SNE is nothing else but DCA-Like applied for t-SNE. Numerical experiments were carefully conducted on several benchmark datasets for visualization. The numerical results show that DCA-Like dramatically improves the running time of standard DCA while giving a similar or better solution. Besides, ADCA-Likes improves further the running time as well as the objective value of DCA-Like. Thus, we can conclude that the two proposed methods are the best for the t-SNE problem.

Finally, the last part tackles the deep clustering problems in two directions. The first direction considered the usage of t-SNE in clustering for large-scale datasets. Two algorithms based on DC Programming and DCA, `MSSC-2S` and `MSSC-JDR`, have been developed. `MSSC-2S` follows the standard "tandem analysis" by employing two well-performance DCAs (for t-SNE and MSSC), whereas `MSSC-JDR` is an extension of `MSSC-2S` by following the joint-clustering approach. We developed a fast and scalable DCA-Like algorithm for `MSSC-JDR`. The numerical results show that both methods significantly improve the clustering quality and running time in comparison with existing methods. Furthermore, `MSSC-JDR` significantly reduces the running time of `MSSC-2S` while having similar accuracy. Hence, our methods are efficient, fast and scalable for clustering large-scale data. The second direction considers the problem of joint-clustering with auto-encoder. We considered a class of methods that suffers from the scaling problem in latent space; and proposed an extension by employing the cosine distance and spherical distance, which are invariance to scaling. We applied the proposed extension on a specific problem of joint-clustering with MSSC. The numerical results show the superiority of our methods in comparison with existing methods.

Each developed clustering method has its advantages and disadvantages. The main distinction between them are based on the distance function measures the similarity of data points in the original space. The first method – clustering based on t-SNE – works well in case we can define such function; whereas the second method – based on auto-encoder – eliminates that requirement. However, the auto-encoder-based method requires a massive amount to learn a good mapping function, so it is preferable if we can gather a huge volume of data or generate more data [32]. Hence, the choice is the trade-off between expert knowledge and data-driven approach: how can we define the distance function versus the amount of the data we have.

The works in this dissertation raise the following topics that can be considered in the near future. In terms of modeling, for the deep clustering problem, we could extend `MSSC-JAE-Sphere` to consider other unit spheres other than ℓ_2 such as ℓ_p ball ($p \in \{1, \infty\}$). On another direction, we could consider the combination between t-SNE and neural network for deep joint-clustering problems. `MSSC-JDR` based on t-SNE is better than `MSSC-JAE-Sphere` in clustering accuracy. Nevertheless, `MSSC-JDR` requires a distance measure to capture the similarity between data points in the original, whereas `MSSC-JAE-Sphere` does not need since the reconstruction error in the auto-encoder does not require external information. The combination between them is

interesting since it could take advantages of both methods. Furthermore, [32] considers data augmentation in deep clustering improves the accuracy of the based model significantly. This technique further exploits the self-supervised nature of the auto-encoder, which is a promising research direction for deep clustering. In terms of optimization, it is possible to develop DCAs for the neural network in general, and deep clustering problem in particular.

Appendix A

Appendix

A.1 Computation of $\mathbf{prox}_{(-z_j^l)/\rho\|.\|_q}(\nu/\rho)$

Denote by $\mathcal{B}_q$ the unit ball for the dual norm of the ℓ_q norm. By Moreau decomposition, we have

$$\mathbf{prox}_{(-z_j^l)/\rho\|.\|_q}(\nu/\rho) = \nu/\rho - (-z_j^l)/\rho \mathbf{proj}_{\mathcal{B}_q}\left(\frac{1}{-z_j^l}\nu\right), \tag{A.1}$$

where $\mathbf{proj}_{\mathcal{B}_q}$ denotes the projection onto $\mathcal{B}_q$ and is computed as follows.

For $q = 1$, the unit ball of the ℓ_1 norm is the box $\mathcal{B}_1 = \{x : \|x\|_1 \leq 1\}$. Hence, the projection onto this box is given by

$$\mathbf{proj}_{\mathcal{B}_1}\left(\frac{1}{-z_j^l}\nu\right)_k = \begin{cases} 1 & \text{if } \frac{1}{-z_j^l}\nu_k > 1 \\ \frac{1}{-z_j^l}\nu_k & \text{if } |\frac{1}{-z_j^l}\nu_k| \leq 1 \\ -1 & \text{if } \frac{1}{-z_j^l}\nu_k < -1. \end{cases} \tag{A.2}$$

From (A.1) and (A.2), we obtain

$$\mathbf{prox}_{(-z_j^l)/\rho\|.\|_q}(\nu/\rho)_k = \begin{cases} \nu_k/\rho - (-z_j^l)/\rho & \text{if } \nu_k > -z_j^l \\ 0 & \text{if } |\nu_k| \leq -z_j^l \\ \nu_k/\rho + (-z_j^l)/\rho & \text{if } \nu_k < z_j^l. \end{cases}$$

This is also known as the elementwise soft thresholding given by

$$\mathbf{prox}_{(-z_j^l)/\rho\|.\|_q}(\nu/\rho) = \left(|\nu|/\rho - (-z_j^l)/\rho\right)_+ \circ \operatorname{sign}(\nu),$$

where $\circ$ is the elementwise product.

For $q = 2$, the unit ball of the ℓ_2 norm is the Euclidean unit ball $\mathcal{B}_2 = \{x : \|x\|_2 \leq 1\}$. Hence, the projection onto this ball is given by

$$\mathbf{proj}_{\mathcal{B}_2}\left(\frac{1}{-z_j^l}\nu\right) = \begin{cases} \frac{\nu}{\|\nu\|_2} & \text{if } \frac{1}{-z_j^l}\|\nu\|_2 > 1 \\ \frac{1}{-z_j^l}\nu & \text{if } \frac{1}{-z_j^l}\|\nu\|_2 \leq 1. \end{cases} \tag{A.3}$$

It follows from (A.1) and (A.3) that

$$\mathbf{prox}_{(-z_j^l)/\rho\|\cdot\|_q}(\nu/\rho) = \begin{cases} \left(1 - \frac{-z_j^l}{\|\nu\|_2}\right)\nu/\rho & \text{if } \|\nu\|_2 > -z_j^l \\ 0 & \text{if } \|\nu\|_2 \leq -z_j^l. \end{cases}$$

For $q = \infty$, the unit ball of the ℓ_∞ norm is the ball $\mathcal{B}_\infty = \{x : \|x\|_\infty \leq 1\}$. Hence, the projection onto this ball is given by

$$\mathbf{proj}_{\mathcal{B}_\infty}\left(\frac{1}{-z_j^l}\nu\right) = \begin{cases} \left(\frac{1}{-z_j^l}|\nu| - \delta\right)_+ \circ \mathrm{sign}(\nu) & \text{if } \frac{1}{-z_j^l}\|\nu\|_1 > 1 \\ \frac{1}{-z_j^l}\nu & \text{if } \frac{1}{-z_j^l a}\|\nu\|_1 \leq 1, \end{cases} \tag{A.4}$$

where δ satisfies the following equation

$$\sum_{k=1}^{Q}\left(\frac{1}{-z_j^l}|\nu_k| - \delta\right)_+ = 1. \tag{A.5}$$

For computing δ in (A.5), some efficient algorithms are available. Among them, we use the very inexpensive algorithm developed in [25]. From (A.1) and (A.4), we get

$$\mathbf{prox}_{(-z_j^l)/\rho\|\cdot\|_q}(\nu/\rho) = \begin{cases} \nu/\rho - \left(\frac{1}{-z_j^l}|\nu| - \delta\right)_+ \circ \mathrm{sign}(\nu) & \text{if } \|\nu\|_1 > -z_j^l \\ 0 & \text{if } \|\nu\|_1 \leq -z_j^l. \end{cases}$$

A.2 Solving problem (4.13) by first order optimality condition

The sub-problem (4.13) is a convex problem. Hence, we can effectively obtain the solution by solving the system of equation $\nabla F_S(u, z) = 0$. It leads to

$$0 = \nabla_u G_C(u, z) - \bar{u_C}^q \tag{A.6}$$

$$0 = \lambda(\nabla_z G_C(u, z) - \bar{z_C}^q) + \mu z - \bar{z_E}^q + \nabla_z\left(\sum_{i,j}(-\bar{v_E}_{ij}^q)g_{ij}(z)\right) \tag{A.7}$$

By expanding both (A.6) and (A.7), we obtained the following solution:

$$\begin{aligned} \text{Eq (A.6)} &\Leftrightarrow nu - \mathbb{1}_{k\times n}z = \bar{u_C}^q \\ &\Leftrightarrow u = \frac{1}{n}\left(\bar{u_C}^q + \mathbb{1}_{k\times n}z\right) \end{aligned} \tag{A.8}$$

Expanding Eq (A.7)

$$
\begin{aligned}
&\Leftrightarrow \quad \lambda(\nabla_z G_C(u,z) - \bar{z_C}^q) + \mu z - \bar{z_E}^q + \nabla_z \left(\sum_{i,j} (-\bar{v_E}_{ij}^q) \|z_i - z_j\|^2 \right) = 0 \\
&\Leftrightarrow \quad \lambda(kz - \mathbb{1}_{n\times k} u - \bar{z_C}^q) + \mu z - \bar{z_E}^q + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T}) z = 0 \\
&\Leftrightarrow \quad \lambda(kz - \mathbb{1}_{n\times k} u) + \mu z + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T}) z = \lambda \bar{z_C}^q + \bar{z_E}^q \\
&\Leftrightarrow \quad \left((\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T})\right) z - \lambda \mathbb{1}_{n\times k} u = \lambda \bar{z_C}^q + \bar{z_E}^q \\
&\overset{(A.8)}{\Leftrightarrow} \quad \left((\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T})\right) z - \frac{\lambda}{n} \mathbb{1}_{n\times k} \left(\bar{u_C}^q + \mathbb{1}_{k\times n} z\right) = \lambda \bar{z_C}^q + \bar{z_E}^q \\
&\Leftrightarrow \quad \left((\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T})\right) z - \frac{\lambda k}{n} \mathbb{1}_{n\times n} z = \lambda \bar{z_C}^q + \bar{z_E}^q + \frac{\lambda}{n} \mathbb{1}_{n\times k} \bar{u_C}^q \\
&\Leftrightarrow \quad \left(\lambda k \left(I_n - \frac{1}{n} \mathbb{1}_{n\times n} \right) + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T}) + \mu I_n \right) z = \lambda \bar{z_C}^q + \bar{z_E}^q + \frac{\lambda}{n} \mathbb{1}_{n\times k} \bar{u_C}^q \\
&\Leftrightarrow \quad \left(\underbrace{(\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T})}_{\mathbb{A}^q} - \underbrace{\frac{\lambda k}{n} \mathbb{1}_{n\times n}}_{\mathbb{B}^q} \right) z = \underbrace{\lambda \bar{z_C}^q + \bar{z_E}^q + \frac{\lambda}{n} \mathbb{1}_{n\times k} \bar{u_C}^q}_{\mathbb{C}^q} \\
&\Leftrightarrow \quad (\mathbb{A}^q - \mathbb{B}^q) z = \mathbb{C}^q
\end{aligned}
\tag{A.9}
$$

A.3 Solving convex sub-problem (4.17)

Let $F_{S_2}(z)$ denotes objective function of problem (4.17). Since the problem (4.17) is convex, then by applying the first order optimality condition, we can obtained the solution by solving $\nabla F_{S_2}(z) = 0$, which is equivalent to

$$
\begin{aligned}
&\Leftrightarrow \quad \lambda(\nabla_z G_C(u^{q,t}, z) - \bar{z_C}^q) + \mu z - \bar{z_E}^q + \nabla_z \left(\sum_{i,j} (-\bar{v_E}_{ij}^q) g_{ij}(z) \right) = 0 \\
&\Leftrightarrow \quad \lambda(kz - \mathbb{1}_{n\times k} u^{q,t}) + \mu z + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T}) z = \lambda \bar{z_C}^q + \bar{z_E}^q \\
&\Leftrightarrow \quad \lambda k z + \mu z + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T}) z = \lambda \bar{z_C}^q + \bar{z_E}^q + \lambda \mathbb{1}_{n\times k} u^{q,t} \\
&\Leftrightarrow \quad \left((\lambda k + \mu) I_n + 2(\mathcal{L}_{-\bar{v_E}^q - (\bar{v_E}^q)^T})\right) z = \lambda \bar{z_C}^q + \bar{z_E}^q + \lambda \mathbb{1}_{n\times k} u^{q,t}
\end{aligned}
\tag{A.10}
$$

Bibliography

[1] Aghabozorgi, S., Seyed Shirkhorshidi, A., and Ying Wah, T. (2015). Time-series clustering – A decade review. *Information Systems*, 53:16–38.

[2] Aljalbout, E., Golkov, V., Siddiqui, Y., and Cremers, D. (2018). Clustering with Deep Learning: Taxonomy and New Methods. *arXiv:1801.07648 [cs, stat]*.

[3] Amaldi, E. and Kann, V. (1998). On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theor. Comput. Sci.*, 209(1):237–260.

[4] An(Homepage), L. T. H. (2005). DC Programming and DCA: `http://www.lita.univ-lorraine.fr/~lethi/index.php/en/research/dc-programming-and-dca.html`.

[5] Aytekin, C., Ni, X., Cricri, F., and Aksu, E. (2018). Clustering and Unsupervised Anomaly Detection with ℓ_2 Normalized Deep Auto-Encoder Representations. *arXiv:1802.00187 [cs]*.

[6] Bagley, S. C., White, H., and Golomb, B. A. (2001). Logistic regression in the medical literature: Standards for use and reporting, with particular attention to one medical domain. *Journal of Clinical Epidemiology*, 54(10):979–985.

[7] Bagozzi, R. P. (1994). *Advanced Methods of Marketing Research*. Blackwell Business.

[8] Boyd, C. R., Tolson, M. A., and Copes, W. S. (1987). Evaluating trauma care: The TRISS method. Trauma Score and the Injury Severity Score. *The Journal of Trauma*, 27(4):370–378.

[9] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge university press.

[10] Bradley, P. S. and Mangasarian, O. L. (1998a). Feature selection via concave minimization and support vector machines. In *ICML*, volume 98, pages 82–90.

[11] Bradley, P. S. and Mangasarian, O. L. (1998b). Feature selection via concave minimization and support vector machines. In *Machine Learning Proceedings of the Fifteenth International Conference (ICML 1998)*, pages 82–90. Morgan Kaufmann.

[12] Burges, C. J. C. (2009). Dimension Reduction: A Guided Tour. *FNT in Machine Learning*, 2(4):275–364.

[13] Cai, D., He, X., and Han, J. (2011). Locally Consistent Concept Factorization for Document Clustering. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):902–913.

[14] Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2018). Deep Clustering for Unsupervised Learning of Visual Features. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 139–156. Springer International Publishing.

[15] Chambolle, A., Vore, R. A. D., Nam-Yong Lee, and Lucier, B. J. (1998). Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage. *IEEE Transactions on Image Processing*, 7(3):319–335.

[16] Chang, W.-C. (1983). On Using Principal Components Before Separating a Mixture of Two Multivariate Normal Distributions. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 32(3):267–275.

[17] Chen, D., Lv, J., and Zhang, Y. (2017). Unsupervised Multi-Manifold Clustering by Learning Deep Representation. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

[18] Collobert, R., Sinz, F., Weston, J., and Bottou, L. (2006). Trading convexity for scalability. In *In ICML '06: Proceedings of the 23rd International Conference on Machine Learning*, pages 201–208. ACM Press.

[19] Cox, D. (1958). The regression analysis of binary sequences (with discussion). *J Roy Stat Soc B*, 20:215–242.

[20] Da Silva, G., Le, H. M., Le Thi, H. A., Lefieux, V., and Tran, B. (2020). Customer Clustering of French Transmission System Operator (RTE) Based on Their Electricity Consumption. In Le Thi, H. A., Le, H. M., and Pham Dinh, T., editors, *Optimization of Complex Systems: Theory, Models, Algorithms and Applications*, Advances in Intelligent Systems and Computing, pages 893–905. Springer International Publishing.

[21] Das, D., Ghosh, R., and Bhowmick, B. (2019). Deep Representation Learning Characterized by Inter-Class Separation for Image Clustering. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 628–637.

[22] De Soete, G. and Carroll, J. D. (1994). K-means clustering in a low-dimensional Euclidean space. In Diday, E., Lechevallier, Y., Schader, M., Bertrand, P., and Burtschy, B., editors, *New Approaches in Classification and Data Analysis*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 212–219. Springer Berlin Heidelberg.

[23] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

[24] Dhillon, I. S. and Modha, D. S. (2001). Concept Decompositions for Large Sparse Text Data Using Clustering. *Machine Learning*, 42(1):143–175.

[25] Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pages 272–279. ACM.

[26] Fard, M. M., Thonet, T., and Gaussier, E. (2018). Deep k-Means: Jointly clustering with k-Means and learning representations. *arXiv:1806.10069 [cs, stat]*.

[27] Genkin, A., Lewis, D. D., and Madigan, D. (2007). Large-scale Bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304.

[28] Ghasedi Dizaji, K., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5736–5745.

[29] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[30] Goldin, D. Q. and Kanellakis, P. C. (1995). On similarity queries for time-series data: Constraint specification and implementation. In Montanari, U. and Rossi, F., editors, *Principles and Practice of Constraint Programming — CP '95*, Lecture Notes in Computer Science, pages 137–153. Springer Berlin Heidelberg.

[31] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[32] Guo, X. (2018). Deep Embedded Clustering with Data Augmentation. In *ACML*, page 16.

[33] Guo, X., Gao, L., Liu, X., and Yin, J. (2017). Improved deep embedded clustering with local structure preservation. In *International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 1753–1759.

[34] Hinton, G. E. and Roweis, S. T. (2003). Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, pages 857–864.

[35] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507.

[36] Hiriart-Urruty, J.-B. and Lemarechal, C. (1993). *Convex Analysis and Minimization Algorithms*. Grundlehren Der Mathematischen Wissenschaften, Convex Analysis and Minimization Algorithms. Springer-Verlag, Berlin Heidelberg.

[37] Huang, P., Huang, Y., Wang, W., and Wang, L. (2014). Deep Embedding Network for Clustering. In *2014 22nd International Conference on Pattern Recognition*, pages 1532–1537.

[38] Ji, P., Zhang, T., Li, H., Salzmann, M., and Reid, I. (2017). Deep Subspace Clustering Networks. In *NIPS*.

[39] Kim, J., Kim, Y., and Kim, Y. (2008). A Gradient-Based Optimization Algorithm for LASSO. *Journal of Computational and Graphical Statistics*, 17(4):994–1009.

[40] King, G. and Zeng, L. (2001). Logistic Regression in Rare Events Data. *Political Analysis*, 9:137–163.

[41] Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*.

[42] Le, H. M., Le Thi, H. A., and Nguyen, M. C. (2015). Sparse semi-supervised support vector machines by DC programming and DCA. *Neurocomputing*, 153:62–76.

[43] Le Tan, D.-K., Le, H., Hoang, T., Do, T.-T., and Cheung, N.-M. (2018). DeepVQ: A deep network architecture for vector quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2579–2582.

[44] Le Thi, H. A. (1997). Contribution à l'optimisation non convexe et l'optimisation globale: Théorie, algorithmes et applications. *Habilitation à Diriger des Recherches, Université de Rouen*.

[45] Le Thi, H. A., Belghiti, M. T., and Pham Dinh, T. (2007). A new efficient algorithm based on DC programming and DCA for clustering. *Journal of Global Optimization*, 37(4):593–608.

[46] Le Thi, H. A., Le, H. M., Nguyen, V. V., and Pham Dinh, T. (2008). A DC programming approach for feature selection in support vector machines learning. *Advances in Data Analysis and Classification*, 2(3):259–278.

[47] Le Thi, H. A., Le, H. M., and Pham Dinh, T. (2014a). New and efficient DCA based algorithms for minimum sum-of-squares clustering. *Pattern Recognition*, 47(1):388–401.

[48] Le Thi, H. A., Le, H. M., and Pham Dinh, T. (2015a). Feature selection in machine learning: An exact penalty approach using a Difference of Convex function Algorithm. *Mach Learn*, 101(1):163–186.

[49] Le Thi, H. A., Le, H. M., Pham Dinh, T., and Van Huynh, N. (2013). Binary classification via spherical separator by DC programming and DCA. *J Glob Optim*, 56(4):1393–1407.

[50] Le Thi, H. A., Le, H. M., Phan, D. N., and Tran, B. (2017). Stochastic DCA for the Large-sum of Non-convex Functions Problem and its Application to Group Variable Selection in Classification. In *Proceedings of the 34th International Conference on Machine Learning* , volume 70, pages 3394–3403.

[51] Le Thi, H. A., Le, H. M., Phan, D. N., and Tran, B. (2018a). A DCA-Like Algorithm and its Accelerated Version with Application in Data Visualization. In *arXiv:1806.09620 [Cs, Math]*.

[52] Le Thi, H. A., Le, H. M., Phan, D. N., and Tran, B. (2018b). Stochastic DCA for sparse multiclass logistic regression. In Le, N.-T., van Do, T., Nguyen, N. T., and Le Thi, H. A., editors, *Advanced Computational Methods for Knowledge Engineering*, pages 1–12.

[53] Le Thi, H. A., Le, H. M., Phan, D. N., and Tran, B. (2019a). Novel DCA Based Algorithms for Minimizing the Sum of a Nonconvex Function and Composite Functions. Applications in Machine learning. *Submitted & Available on arvXiv [arXiv:1806.09620]*.

[54] Le Thi, H. A., Le, H. M., Phan, D. N., and Tran, B. (2019b). Stochastic DCA for minimizing a large sum of DC functions with application to Multi-class Logistic Regression. *Submitted & Available on arvXiv [arXiv:1911.03992]*.

[55] Le Thi, H. A. and Moeini, M. (2014). Long-Short Portfolio Optimization Under Cardinality Constraints by Difference of Convex Functions Algorithm. *J Optim Theory Appl*, 161(1):199–224.

[56] Le Thi, H. A. and Nguyen, M. C. (2014). Self-organizing maps by difference of convex functions optimization. *Data Min Knowl Disc*, 28(5):1336–1365.

[57] Le Thi, H. A. and Nguyen, M. C. (2017). DCA based algorithms for feature selection in multi-class support vector machine. *Ann Oper Res*, 249(1):273–300.

[58] Le Thi, H. A., Nguyen, M. C., and Pham Dinh, T. (2014b). A DC Programming Approach for Finding Communities in Networks. *Neural Computation*, 26(12):2827–2854.

[59] Le Thi, H. A. and Pham Dinh, T. (1997). Solving a Class of Linearly Constrained Indefinite Quadratic Problems by D.C. Algorithms. *Journal of Global Optimization*, 11(3):253–285.

[60] Le Thi, H. A. and Pham Dinh, T. (2005). The DC (Difference of Convex Functions) Programming and DCA Revisited with DC Models of Real World Nonconvex Optimization Problems. *Annals of Operations Research*, 133(1):23–46.

[61] Le Thi, H. A. and Pham Dinh, T. (2009). Minimum sum-of-squares clustering by DC programming and DCA. In *International Conference on Intelligent Computing*, pages 327–340. Springer.

[62] Le Thi, H. A. and Pham Dinh, T. (2018). DC programming and DCA: thirty years of developments. *Mathematical Programming*, pages 1–64.

[63] Le Thi, H. A., Pham Dinh, T., Le, H. M., and Vo, X. T. (2015b). DC approximation approaches for sparse optimization. *European Journal of Operational Research*, 244(1):26–46.

[64] Le Thi, H. A., Pham Dinh, T., and Thiao, M. (2016a). Efficient approaches for ℓ_2-ℓ_0 regularization and applications to feature selection in SVM. *Applied Intelligence*, 45(2):549 – 565.

[65] Le Thi, H. A. and Phan, D. N. (2016). DC Programming and DCA for Sparse Optimal Scoring Problem. *Neurocomput.*, 186(C):170–181.

[66] Le Thi, H. A. and Phan, D. N. (2017). DC Programming and DCA for Sparse Fisher Linear Discriminant Analysis. *Neural Comput. Appl.*, 28(9):2809–2822.

[67] Le Thi, H. A., Vo, X. T., and Pham Dinh, T. (2016b). Efficient Nonnegative Matrix Factorization by DC Programming and DCA. *Neural Comput*, 28(6):1163–1216.

[68] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[69] Li, H. and Lin, Z. (2015). Accelerated proximal gradient methods for nonconvex programming. In *Advances in Neural Information Processing Systems*, pages 377–387.

[70] Liao, J. G. and Chin, K.-V. (2007). Logistic regression for disease classification using microarray data: Model selection in a large p and small n case. *Bioinformatics*, 23(15):1945–1951.

[71] Liu, Y. and Shen, X. (2006). Multicategory ψ-Learning. *Journal of the American Statistical Association*, 101(474):500–509.

[72] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[73] Lovász, L. and Plummer, M. D. (2009). *Matching Theory*. American Mathematical Soc.

[74] Maaten, L. v. d. (2014). Accelerating t-sne using tree-based algorithms. *Journal of machine learning research*, 15(1):3221–3245.

[75] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297. The Regents of the University of California.

[76] Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., and Long, J. (2018). A Survey of Clustering With Deep Learning: From the Perspective of Network Architecture. *IEEE Access*, 6:39501–39514.

[77] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[78] Mori, Y., Kuroda, M., and Makino, N. (2016). Joint Dimension Reduction and Clustering. In Mori, Y., Kuroda, M., and Makino, N., editors, *Nonlinear Principal Component Analysis and Its Applications*, SpringerBriefs in Statistics, pages 57–64. Springer Singapore, Singapore.

[79] Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress.

[80] Nguyen, T. M. T. (2018). *DCA Based Approaches for Mathematical Programs with Equilibrium Constraints*. PhD thesis, Université de Lorraine.

[81] Nocedal, J. (1980). Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, 35(151):773–782.

[82] Paparrizos, J. and Gravano, L. (2015). K-Shape: Efficient and Accurate Clustering of Time Series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1855–1870. ACM Press.

[83] Parikh, N. and Boyd, S. (2014). Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239.

[84] Pham Dinh, T., Le, H. M., Le Thi, H. A., and Lauer, F. (2014). A Difference of Convex Functions Algorithm for Switched Linear Regression. *IEEE Transactions on Automatic Control*, 59(8):2277–2282.

[85] Pham Dinh, T. and Le Thi, H. A. (1997). Convex analysis approach to dc programming: Theory, algorithms and applications. *Acta Mathematica Vietnamica*, 22(1):289–355.

[86] Pham Dinh, T. and Le Thi, H. A. (1998). A D. C. Optimization Algorithm for Solving the Trust-Region Subproblem. *SIAM Journal of Optimization*, 8(2):476–505.

[87] Pham Dinh, T. and Le Thi, H. A. (2014). Recent Advances in DC Programming and DCA. In Nguyen, N. T. and Le Thi, H. A., editors, *Transactions on Computational Intelligence XIII*, Lecture Notes in Computer Science, pages 1–37. Springer Berlin Heidelberg.

[88] Pham Dinh, T. and Souad, E. B. (1986). Algorithms for Solving a Class of Nonconvex Optimization Problems. Methods of Subgradients. In Hiriart-Urruty, J. B., editor, *North-Holland Mathematics Studies*, volume 129 of *Fermat Days 85: Mathematics for Optimization*, pages 249–271. North-Holland.

[89] Phan, D. N., Le, H. M., and Le Thi, H. A. (2018). Accelerated Difference of Convex functions Algorithm and its Application to Sparse Binary Logistic Regression. In *Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1369–1375.

[90] Phan, D. N., Le Thi, H. A., and Pham Dinh, T. (2017). Sparse covariance matrix estimation by DCA-Based Algorithms. *Neural Computation*, 29(11):3040–3077.

[91] Rakotomamonjy, A., Flamary, R., and Gasso, G. (2016). DC Proximal Newton for Nonconvex Optimization Problems. *IEEE Transactions on Neural Networks and Learning Systems*, 27(3):636–647.

[92] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '12*, page 262, Beijing, China. ACM Press.

[93] Rockafellar, R. T. (1970). Convex Analysis. *Princeton, NJ.*

[94] Rogovschi, N., Kitazono, J., Grozavu, N., Omori, T., and Ozawa, S. (2017). T-Distributed stochastic neighbor embedding spectral clustering. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1628–1632.

[95] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533.

[96] Shah, S. A. and Koltun, V. (2018). Deep Continuous Clustering. *arXiv:1803.01449 [cs]*.

[97] Shaol, X., Ge, K., Su, H., Luo, L., Peng, B., and Li, D. (2018). Deep Discriminative Clustering Network. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, Rio de Janeiro. IEEE.

[98] Shewchuk, J. R. (1994). An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA.

[99] Song, C., Liu, F., Huang, Y., Wang, L., and Tan, T. (2013). Auto-encoder Based Data Clustering. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Lecture Notes in Computer Science, pages 117–124. Springer, Berlin, Heidelberg.

[100] Souvenir, R. and Pless, R. (2005). Manifold clustering. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 648–653 Vol. 1, Beijing, China. IEEE.

[101] Steinbach, M., Ertöz, L., and Kumar, V. (2004). The Challenges of Clustering High Dimensional Data. In Wille, L. T., editor, *New Directions in Statistical Physics: Econophysics, Bioinformatics, and Pattern Recognition*, pages 273–309. Springer Berlin Heidelberg, Berlin, Heidelberg.

[102] Strehl, A. and Ghosh, J. (2002). Cluster Ensembles - A Knowledge Reuse Framework for Combining Multiple Partitions. *Journal of Machine Learning Research*, 3:583–617.

[103] Subasi, A. and Erçelebi, E. (2005). Classification of EEG signals using neural network and logistic regression. *Comput. Methods Programs Biomed.*, 78(2):87–99.

[104] Tian, K., Zhou, S., and Guan, J. (2017). DeepCluster: A General Clustering Framework Based on Deep Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 809–825. Springer.

[105] Van Der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. *RBM*, 500(500):26.

[106] van der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

[107] Van Der Maaten, L., Postma, E., and Van den Herik, J. (2009). Dimensionality Reduction: A Comparative Review. *J Mach Learn Res*, 10(66-71):13.

[108] Vichi, M. and Kiers, H. A. (2001). Factorial k-means analysis for two-way data. *Computational Statistics & Data Analysis*, 37(1):49–64.

[109] Vincent, M. and Hansen, N. R. (2014). Sparse group lasso and high dimensional multinomial classification. *Comput. Stat. Data Anal.*, 71:771–786.

[110] Vladymyrov, M. and Carreira-Perpinan, M. (2012). Partial-Hessian strategies for fast learning of nonlinear embeddings. *arXiv preprint arXiv:1206.4646*.

[111] Vo, X. T., Bach, T., Le Thi, H. A., and Pham Dinh, T. (2017). Ramp Loss Support Vector Data Description. In Nguyen, N. T., Tojo, S., Nguyen, L. M., and Trawiński, B., editors, *Intelligent Information and Database Systems*, volume 10191, pages 421–431. Springer International Publishing, Cham.

[112] Wang, S., Chang, T.-H., Cui, Y., and Pang, J.-S. (2019). Clustering by Orthogonal Non-negative Matrix Factorization: A Sequential Non-convex Penalty Approach. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5576–5580, Brighton, United Kingdom. IEEE.

[113] Warren Liao, T. (2005). Clustering of time series data—a survey. *Pattern Recognition*, 38(11):1857–1874.

[114] Wilson, N. K., Kent, D. G., Buettner, F., Shehata, M., Macaulay, I. C., Calero-Nieto, F. J., Sánchez Castillo, M., Oedekoven, C. A., Diamanti, E., Schulte, R., Ponting, C. P., Voet, T., Caldas, C., Stingl, J., Green, A. R., Theis, F. J., and

Göttgens, B. (2015). Combined Single-Cell Functional and Gene Expression Analysis Resolves Heterogeneity within Stem Cell Populations. *Cell Stem Cell*, 16(6):712–724.

[115] Witten, D. M. and Tibshirani, R. (2011). Penalized classification using Fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B*, 73(5):753–772.

[116] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]*.

[117] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*, pages 478–487.

[118] Xu, W., Liu, X., and Gong, Y. (2003). Document Clustering Based on Non-negative Matrix Factorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 267–273, New York, NY, USA. ACM.

[119] Yang, B., Fu, X., and Sidiropoulos, N. D. (2017a). Learning from hidden traits: Joint factor analysis and latent clustering. *IEEE Transactions on Signal Processing*, 65(1):256–269.

[120] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017b). Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In *International Conference on Machine Learning*, pages 3861–3870.

[121] Yang, J., Parikh, D., and Batra, D. (2016). Joint Unsupervised Learning of Deep Representations and Image Clusters. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5147–5156.

[122] Yang, Z., King, I., Xu, Z., and Oja, E. (2009). Heavy-tailed symmetric stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, pages 2169–2177.

[123] Yang, Z., Peltonen, J., and Kaski, S. (2015). Majorization-minimization for manifold embedding. In *Artificial Intelligence and Statistics*, pages 1088–1097.

[124] Yankov, D. and Keogh, E. (2006). Manifold Clustering of Shapes. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 1167–1171, Hong Kong, China. IEEE.

[125] Yin, P., Lou, Y., He, Q., and Xin, J. (2015). Minimization of ℓ_{1-2} for Compressed Sensing. *SIAM J. Sci. Comput.*, 37(1):A536–A563.

[126] Yuille, A. L. and Rangarajan, A. (2002). The Concave-Convex Procedure (CCCP). In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 1033–1040. MIT Press.

[127] Zeng, H. and Cheung, Y.-m. (2014). Learning a mixture model for clustering with the completed likelihood minimum message length criterion. *Pattern Recognition*, 47(5):2011–2030.

[128] Zha, H., He, X., Ding, C., Gu, M., and Simon, H. D. (2002). Spectral Relaxation for K-means Clustering. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 1057–1064. MIT Press.

[129] Zhang, P., Gong, M., Zhang, H., and Liu, J. (2017). DRLnet: Deep Difference Representation Learning Network and An Unsupervised Optimization Framework. In *IJCAI*.

[130] Zhang, T., Ji, P., Harandi, M., Hartley, R., and Reid, I. (2019a). Scalable Deep k-Subspace Clustering. In Jawahar, C., Li, H., Mori, G., and Schindler, K., editors, *Computer Vision – ACCV 2018*, Lecture Notes in Computer Science, pages 466–481. Springer International Publishing.

[131] Zhang, T., Ji, P., Harandi, M., Huang, W., and Li, H. (2019b). Neural Collaborative Subspace Clustering. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7384–7393. PMLR.

[132] Zhao Zhang, Chow, T. W. S., and Mingbo Zhao (2013). M-Isomap: Orthogonal Constrained Marginal Isomap for Nonlinear Dimensionality Reduction. *IEEE Trans. Cybern.*, 43(1):180–191.