Greedy Representative Selection for Unsupervised Data Analysis

by

Ahmed Khairy Farahat Helwa

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In recent years, the advance of information and communication technologies has allowed the storage and transfer of massive amounts of data. The availability of this overwhelming amount of data stimulates a growing need to develop fast and accurate algorithms to discover useful information hidden in the data. This need is even more acute for unsupervised data, which lacks information about the categories of different instances.

This dissertation addresses a crucial problem in unsupervised data analysis, which is the selection of representative instances and/or features from the data. This problem can be generally defined as the selection of the most representative columns of a data matrix, which is formally known as the Column Subset Selection (CSS) problem. Algorithms for column subset selection can be directly used for data analysis or as a pre-processing step to enhance other data mining algorithms, such as clustering. The contributions of this dissertation can be summarized as outlined below.

First, a fast and accurate algorithm is proposed to greedily select a subset of columns of a data matrix such that the reconstruction error of the matrix based on the subset of selected columns is minimized. The algorithm is based on a novel recursive formula for calculating the reconstruction error, which allows the development of time and memory-efficient algorithms for greedy column subset selection. Experiments on real data sets demonstrate the effectiveness and efficiency of the proposed algorithms in comparison to the state-of-the-art methods for column subset selection.

Second, a kernel-based algorithm is presented for column subset selection. The algorithm greedily selects representative columns using information about their pairwise similarities. The algorithm can also calculate a Nyström approximation for a large kernel matrix based on the subset of selected columns. In comparison to different Nyström methods, the greedy Nyström method has been empirically shown to achieve significant improvements in approximating kernel matrices, with minimum overhead in run time.

Third, two algorithms are proposed for fast approximate k-means and spectral clustering. These algorithms employ the greedy column subset selection method to embed all data points in the subspace of a few representative points, where the clustering is performed. The approximate algorithms run much faster than their exact counterparts while achieving comparable clustering performance.

Fourth, a fast and accurate greedy algorithm for unsupervised feature selection is proposed. The algorithm is an application of the greedy column subset selection method presented in this dissertation. Similarly, the features are greedily selected such that the reconstruction error of the data matrix is minimized. Experiments on benchmark data sets

show that the greedy algorithm outperforms state-of-the-art methods for unsupervised feature selection in the clustering task.

Finally, the dissertation studies the connection between the column subset selection problem and other related problems in statistical data analysis, and it presents a unified framework which allows the use of the greedy algorithms presented in this dissertation to solve different related problems.

Acknowledgements

All praise goes to God for giving me the strength and knowledge to complete this work.

The completion of this dissertation was not possible without the support of many people, to whom I would like to express my gratitude.

I would like to express my sincere gratitude to my supervisors, Prof. Mohamed Kamel and Prof. Ali Ghodsi for their support, guidance and encouragement. I especially thank them for their valuable contributions to the work presented in this dissertation. I would like also to express my appreciation to the committee members: Prof. Jian Pei, Prof. Fakhri Karray, Prof. Otman Basir and Prof. Mu Zhu, for their valuable comments and suggestions.

I would like also to thank my colleagues at the Centre of Pattern Analysis and Machine Intelligence (CPAMI) and University of Waterloo for the useful and stimulating discussions. My thanks goes to Moataz El-Ayadi, Mostafa Hassan, Ahmed Othman, Mohamed AbdelRazik, Ahmed Gawish, Shady Shehata, Nabil Drawil, Mehrdad Gangeh, Miao Yun Qian, Ahmed Elgohary, Pooyan Khajehpour, Fatemeh Dorri, Daniel Severn, Mohammad Ashtiani and Han Sheng Sun. I would like also to extend my appreciation to my friends in Egypt and Canada for their support and encouragement.

Many thanks also go to the Graduate Studies Office at the University of Waterloo and the Ontario Graduate Scholarship (OGS) program for their financial support.

My heartfelt gratitude goes to my parents, Khairy and Samia, for their encouragement and never ending support and to my brother, Mohamed, for his support. I would also like to dedicate special thanks to my wife, Amira, for her patience, understanding, and having faith in me, and my daughters, Salma and Maryam, for brightening my life.

Dedication

To my parents, wife and daughters.

Table of Contents

List of Tables **xv**

List of Figures **xvii**

1 Introduction **1**

1.1 Motivations 1

1.2 Summary of Contributions 3

1.3 Organization of the Dissertation 4

1.4 Notations 4

2 Background **7**

2.1 Low-rank Matrix Approximation 7

2.1.1 Singular Value Decomposition (SVD) 8

2.1.2 Stochastic Singular Value Decomposition (SSVD) 10

2.1.3 QR Decomposition 11

2.2 Data Clustering 13

2.2.1 Hierarchical Clustering 14

2.2.2 k-Means 14

2.2.3 Spherical k-Means 18

2.2.4 Spectral Clustering 18

2.2.5 Affinity Propagation 21

2.2.6 Evaluating the Quality of Clusters 22

3 Greedy Column Subset Selection **27**
3.1 Column Subset Selection (CSS) . . . 27
3.2 Recursive Selection Criterion . . . 31
3.3 Greedy Selection Algorithm . . . 35
3.3.1 Memory-Efficient Criterion . . . 36
3.3.2 Partition-Based Criterion . . . 41
3.4 Related Work . . . 43
3.4.1 Randomized Methods . . . 43
3.4.2 Deterministic Methods . . . 45
3.4.3 Hybrid Methods . . . 46
3.4.4 Comparison to Related Work . . . 46
3.5 Experiments and Results . . . 47

4 Greedy Nyström Approximation **55**
4.1 Nyström Approximation . . . 55
4.1.1 The Basic Nyström Method . . . 56
4.1.2 Applications of the Nyström Method . . . 57
4.2 Kernel Column Subset Selection . . . 58
4.3 Recursive Nyström Approximation . . . 61
4.4 Greedy Sampling Criterion . . . 65
4.4.1 Memory-Efficient Criterion . . . 66
4.4.2 Partition-Based Criterion . . . 66
4.5 Related Work . . . 69
4.5.1 Random Sampling . . . 69
4.5.2 Deterministic Sampling . . . 69
4.5.3 Comparison to Related Work . . . 71
4.6 Experiments and Results . . . 72

5 Fast Approximate Data Clustering **83**
5.1 Fast Approximate k-Means Clustering . . . 83
5.2 Fast Approximate Spectral Clustering . . . 85
5.3 Related Work . . . 87
5.3.1 k-Means Clustering . . . 87
5.3.2 Spectral Clustering . . . 89
5.4 Experiments and Results . . . 90
5.4.1 Fast Approximate k-Means Clustering . . . 91
5.4.2 Fast Approximate Spectral Clustering . . . 92

6 Greedy Unsupervised Feature Selection **99**
6.1 Unsupervised Feature Selection . . . 99
6.2 Greedy Feature Selection . . . 101
6.3 Related Work . . . 101
6.3.1 PCA-based Methods . . . 101
6.3.2 Methods Preserving Data Similarity . . . 103
6.3.3 Methods Preserving Data Cluster Structure . . . 103
6.3.4 Feature Selection using Feature Similarity . . . 104
6.3.5 Comparison to Previous Work . . . 104
6.4 Experiments and Results . . . 105

7 Generalized Greedy CSS **117**
7.1 Generalized Column Subset Selection (CSS) . . . 117
7.2 Generalized CSS Problems . . . 119

8 Conclusions and Future Work **123**
8.1 Conclusions . . . 123
8.2 Future Work . . . 124

Permissions **127**

References **133**

List of Tables

3.1 The properties of the data sets used to evaluate different CSS methods. . . 47

3.2 The relative accuracy measures of the best performing CSS methods for the **Reuters-21578**, **Reviews**, and **LA1** data sets. 53

3.3 The relative accuracy measures of the best performing CSS methods for the **MNIST-4K**, **PIE-20**, and **YaleB-38** data sets. 54

4.1 The relative accuracy measures of the best performing Nyström methods for the **Reuters-21578**, **Reviews**, and **LA1** data sets. 79

4.2 The relative accuracy measures of the best performing Nyström methods for the **MNIST-4K**, **PIE-20**, and **YaleB-38** data sets. 80

4.3 The relative accuracy measures of the proposed greedy Nyström methods compared to the ensemble Nyström method. 81

4.4 The run times of the proposed greedy Nyström methods compared to the ensemble Nyström method. 82

5.1 The properties of the data sets used to evaluate the fast approximate clustering algorithms. 90

5.2 The clustering performance measures of the best performing fast k-means algorithms. 95

5.3 The clustering performance measures of the best performing fast spectral clustering algorithms. 98

6.1 The properties of the data sets used to evaluate different feature selection methods. 106

6.2 The clustering performance of the k-means algorithm for the top performing unsupervised feature selection methods for the *ORL*, *COIL20*, and *ISOLET* data sets 114

6.3 The clustering performance of the k-means algorithm for the top performing unsupervised feature selection methods for the *USPS*, *TDT2-30*, and *20NG* data sets 115

7.1 The connection between the generalized column subset selection and related problems. 120

List of Figures

3.1 The relative accuracy measures and run times of different column-based low-rank approximations $\tilde{A}_{\mathcal{S}}$ for the *Reuters-21578*, *Reviews* and *LA1* data sets. 51

3.2 The relative accuracy measures and run times of different column-based low-rank approximations $\tilde{A}_{\mathcal{S}}$ for the *MNIST-4K*, *PIE-20* and *YaleB-38* data sets. 52

4.1 The relative accuracy measures and run times of different rank-l Nyström approximations ($\tilde{K}_{\mathcal{S}}$) for the *Reuters-21578*, *Reviews* and *LA1* data sets. . 74

4.2 The relative accuracy measures and run times of different rank-l Nyström approximations ($\tilde{K}_{\mathcal{S}}$) for the *MNIST-4K*, *PIE-20* and *YaleB-38* data sets. 75

4.3 The relative accuracy measures and run times of different rank-k Nyström approximations ($\tilde{K}_{\mathcal{S},k}$) for the *Reuters-21578*, *Reviews* and *LA1* data sets. 76

4.4 The relative accuracy measures and run times of different rank-k Nyström approximations ($\tilde{K}_{\mathcal{S},k}$) for the *MNIST-4K*, *PIE-20* and *YaleB-38* data sets. 77

5.1 The clustering performance measures and run times of the fast k-means algorithms. 93

5.2 The clustering performance measures and run times of the fast spectral clustering algorithms using linear kernels. 96

5.3 The clustering performance measures and run times of the fast spectral clustering algorithms using Gaussian kernels. 97

6.1 The k-means clustering performance of different feature selection methods. 110

6.2 The affinity propagation (AP) clustering performance of different feature selection methods for the data sets *ORL*, *COIL20* and *ISOLET*. 111

6.3 The run times of different feature selection methods. 112
6.4 The run times of the **PCA-LRG** and **LS** methods in comparison to the proposed greedy algorithms. 113

Chapter 1

Introduction

This dissertation is principally concerned with the problem of selecting representative data instances and features, and the use of such representatives to enhance various algorithms for unsupervised data analysis. The selection of representative data instances and features is a crucial task in unsupervised data mining, which can be used to learn about the insights of the data, or to preprocess data for further analysis.

This chapter introduces the research problems addressed in the dissertation and summarizes the contributions made toward the solution of these problems. The chapter is organized as follows: Section 1.1 introduces the representative selection problem and presents the motivations behind this work. Section 1.2 summarizes the contributions of the dissertation. Section 1.3 describes how the dissertation is organized. Lastly, Section 1.4 describes the notations used throughout the dissertation.

1.1 Motivations

Today's information systems are typically characterized by the transfer, storage and processing of a massive amount of data. The acute need to analyze this data motivates the development of fast and accurate algorithms to understand the insights of the data and explore its hidden structures. One of the challenging tasks in unsupervised data analysis is the selection of representative data instances and/or features. This problem is formally known as the Column Subset Selection (CSS) problem. The CSS problem has many useful applications in unsupervised data analysis which are summarized below.

Enhanced data analysis. In information retrieval systems, when the data analyst submits a query to retrieve a subset of data instances (e.g., documents, images, records), thousands of search results are retrieved. The retrieval engine usually ranks the returned results according to their relevance to the query, as well as their centrality with respect to other search results. However, many application domains lack information about the relations between different instances. In this case, the selection of representative data instances based solely on their contents is a potentially powerful tool to allow the data analyst to browse the search results and gain insight into their meanings. In this application, the analyst will be presented a few representative data instances, while being allowed to display how other instances are related to the selected representatives. Moreover, this tool can also be used to allow the browsing of a large collection of data instances, such that the data analyst can navigate through different views of the data and show the representative instances for each view.

Representative-based data embedding. Low-dimensional embedding of data instances is a powerful tool in unsupervised data analysis, which allows the data analyst to visualize the data instances in a space with very few dimensions, and apply different clustering and classification techniques to the low-dimensional data. However, most of the existing techniques for low-dimensional embedding produce dimensions whose meanings are difficult to interpret. For instance, Latent Semantic Analysis (LSA) [1] is a well-known technique in text mining which allows the learning of different hidden concepts from a text corpus. One of the problems of LSA is that each learned concept is represented by a dense vector which combines thousands of terms with positive and negative weights. This makes it difficult to understand the meaning of these concepts. Another example is the Eigenfaces method for face recognition [2] in which face images are represented in the space of the leading eigenvectors of the image-pixel matrix. Although this technique has been successfully used in the face recognition task, the meaning of the resulting Eigenfaces is hard to interpret. The selection of representative instances and features provides an effective solution to the latent concepts problem. This solution depends on learning a low-dimensional embedding of data instances whose basis vectors represent a set of representative instances or features. This allows the data analyst to interpret the meaning of the learned concepts and understand the performance of different data mining and machine learning algorithms.

Fast data clustering. Data clustering is a core algorithm in supervised data analysis which is concerned with the organization of data instances into groups. Different algorithms have been proposed for data clustering, such as the well-known k-means algorithm [3] and the effective spectral clustering methods [4]. Most of the existing data clustering algorithms

do not scale well to handle data sets with very large numbers of instances and/or features. An effective way to reduce the complexity of clustering algorithms is to choose a small set of representative data points (also known as landmarks) and use those landmarks to enhance the efficiency of the clustering algorithms while maintaining their effectiveness. Although different methods have been proposed to perform landmark-based clustering [5–7], the development of more accurate algorithms for representative selection will definitely lead to enhancing the effectiveness of different landmark-based clustering algorithms.

In attempting to tackle the above-mentioned problems, this dissertation presents different algorithms for unsupervised data analysis whose core component is the greedy selection of representative data instances or features. The contributions of this dissertation are summarized in the following section.

1.2 Summary of Contributions

The contributions of the dissertation can be summarized as follows:

- A fast and accurate algorithm is proposed for column subset selection. The algorithm depends on minimizing the reconstruction error of the data matrix based on the subset of selected columns. Time and memory-efficient algorithms are presented to greedily select columns which minimize this reconstruction error.
- A kernel-based algorithm for column subset selection is presented. The algorithm can also be used to calculate the Nyström approximation of the kernel matrix based on the subset of selected columns. A comparison to different methods for Nyström approximation shows the superiority of the proposed algorithm.
- The greedy algorithm is used to enhance the efficiency of different clustering algorithms. This is done by learning a low-dimensional embedding of data points in the space of representative data instances, and then applying the data clustering algorithm to data points in that low-dimensional space.
- A greedy algorithm is presented for unsupervised feature selection. The algorithm is an instance of the greedy column subset selection presented in this dissertation. A comparison to state-of-the-art methods of unsupervised feature selection shows the superiority of the proposed feature selection algorithm.
- A framework for generalized column subset selection is presented, and an efficient greedy algorithm is proposed to select columns from a source matrix such that the

reconstruction error of a target matrix is minimized. The connection between the generalized column subset selection problem and related problems in statistical data analysis has been studied.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows: Chapter 2 reviews the necessary background of the proposed work. Chapter 3 studies the problem of Column Subset Selection (CSS), and presents a fast and effective algorithm to greedily select a compact set of representative columns such that the reconstruction error of the data matrix based on the selected columns is minimized. Next, Chapter 4 presents a novel greedy algorithm for Nyström approximation, which extends the greedy CSS algorithm to work on positive semi-definite kernel matrices. In Chapter 5, different fast algorithms for approximate data clustering are proposed. In these algorithms, a compact column-based representation is first learned, and then data points are clustered in the new compact space. In Chapter 6, the greedy CSS algorithms are applied to the problem of selecting representative features for data clustering. Chapter 7 presents a framework that generalizes the CSS problem and allows the use of the proposed greedy algorithms to solve different related problems. Finally, Chapter 8 concludes the dissertation and discusses future work.

1.4 Notations

Throughout the dissertation, scalars are denoted by small letters (e.g., l, m, n), sets are denoted in script letters (e.g., $\mathcal{S}$, $\mathcal{R}$), vectors are denoted by small bold italic letters (e.g., $\boldsymbol{f}$, $\boldsymbol{g}$), and matrices are denoted by capital letters (e.g., A, K). In addition, the following notations are used:

For a set $\mathcal{S}$:

$\lvert\mathcal{S}\rvert$	the cardinality of the set.

For a vector $\boldsymbol{x} \in \mathbb{R}^p$:

$\boldsymbol{x}_i$	i-th element of $\boldsymbol{x}$.
$\lVert\boldsymbol{x}\rVert$	the Euclidean norm (ℓ_2-norm) of $\boldsymbol{x}$.

For a matrix $A \in \mathbb{R}^{p \times q}$:

A_{ij} (i,j)-th entry of A.
$A_{i:}$ i-th row of A.
$A_{:j}$ j-th column of A.
$A_{\mathcal{S}:}$ the sub-matrix of A which consists of the set $\mathcal{S}$ of rows.
$A_{:\mathcal{S}}$ the sub-matrix of A which consists of the set $\mathcal{S}$ of columns.
A^T the transpose of A.
$\tilde{A}$ a low rank approximation of A.
$\tilde{A}_k$ the best rank-k approximation of A obtained using singular value decomposition.
$\tilde{A}_{\mathcal{S}}$ a rank-l approximation of A based on the set $\mathcal{S}$ of columns, where $|\mathcal{S}| = l$.
$\tilde{A}_{\mathcal{S},k}$ a rank-k approximation of A based on the set $\mathcal{S}$ of columns, where $|\mathcal{S}| = l$ and $k \leq l$.
$\|A\|_F$ the Frobenius norm of A: $\|A\|_F = \sqrt{\Sigma_{i,j} A_{ij}^2} = \sqrt{\Sigma_i \sigma_i^2}$, where σ_i is the i-th leading singular value of A.
$\|A\|_2$ the spectral norm of A: $\|A\|_2 = \max_{\|\boldsymbol{x}\| \neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \sigma_1$, where σ_1 is the leading singular value of A.

For a kernel matrix $K \in \mathbb{R}^{p \times p}$:

$\tilde{K}$ a low rank approximation of K.
$\tilde{K}_k$ the best rank-k approximation of K obtained using singular value decomposition.
$\tilde{K}_{\mathcal{S}}$ rank-l Nyström approximation of K based on the set $\mathcal{S}$ of columns, where $|\mathcal{S}| = l$.
$\tilde{K}_{\mathcal{S},k}$ rank-k Nyström approximation of K based on the set $\mathcal{S}$ of columns, where $|\mathcal{S}| = l$ and $k \leq l$.

Chapter 2

Background

This chapter presents a review of the necessary background for two basic topics that are discussed throughout the dissertation: Low-rank Matrix Approximation and Data Clustering.

2.1 Low-rank Matrix Approximation

Given an $m \times n$ matrix A, the rank of A is defined as the maximum number of linearly independent rows or columns. The problem of finding a low-rank approximation of a matrix is defined as:

Problem 2.1 (Low-rank Matrix Approximation) *Given an $m \times n$ matrix A and a positive integer k, find an $m \times n$ matrix $\tilde{A}$ such that:*

$$\tilde{A} = \underset{B,\, rank(B) \leq k}{\arg\min} \|A - B\|_F \ ,$$

where $\|.\|_F$ is the Frobenius norm of a matrix.

The objective function of Problem 2.1 quantifies the discrepancy between the data matrix A and its low-rank approximation B. This discrepancy is referred to as the approximation error or the reconstruction error of the data matrix. In this objective function, the Frobenius norm of the discrepancy matrix is used to quantify the reconstruction error. Other types of norms such as the spectral norm can also be used to quantify this error.

Throughout this dissertation, the Frobenius norm will be used to quantify the discrepancy between the data matrix and its low-rank approximation.

As the target rank k increases, the reconstruction error of the data matrix decreases. If the target rank k is greater or equal to the rank of the original matrix A, the reconstruction error will be zero.

The low-rank approximation matrix $\tilde{A}$ can be decomposed as

$$\tilde{A} = CT,$$

where C is an $m \times k$ matrix whose columns represent a basis for the column space of $\tilde{A}$ and T is a $k \times n$ matrix whose elements represent the coefficients of the columns of $\tilde{A}$ in the basis C:

$$\tilde{A}_{:i} = CT_{:i} = \sum_{j=1}^{k} T_{ji} C_{:j} \; .$$

The problem of finding a low-rank approximation of a matrix has many useful applications in data mining and machine learning. Examples include the discovery of latent concepts in a text corpus [1], the learning of basic components in images (e.g., Eigenfaces [2]), reducing the dimensionality of data for visualization and fast analysis [8], and the completion of matrices for recommendation systems [9].

Without additional constraints, the optimal solution of the low-rank approximation problem can be obtained by calculating the truncated singular value decomposition [10] of A as discussed in Section 2.1.1. In addition, different methods for incomplete matrix decomposition can be used to obtain low-rank approximations that satisfy additional constraints. The following sections discuss different matrix decomposition methods that are related to the algorithms presented in this dissertation. The reader is referred to [10] for more details about different methods for matrix decomposition.

2.1.1 Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) is a factorization of a rectangular matrix into its left and right singular vectors [10].

Definition 2.1 (Singular Value Decomposition) *Given an $m \times n$ matrix A, the singular value decomposition of A is*

$$A = U \Sigma V^T,$$

where U and V are $m \times m$ and $n \times n$ orthonormal matrices whose columns represent the left and right singular vectors of A respectively and Σ is an $m \times n$ rectangular diagonal matrix whose diagonal elements represent the singular values of A.

For a non-negative singular value σ, the corresponding left and right singular vectors $\boldsymbol{u}$, $\boldsymbol{v}$ are non-zero vectors such that

$$A\boldsymbol{v} = \sigma\boldsymbol{u}\,,$$
$$A^T\boldsymbol{u} = \sigma\boldsymbol{v}\,.$$

The SVD is closely related to the eigendecomposition [10]. Given an $n \times n$ square matrix B, the eigendecomposition of B is defined as

$$B = \Omega\Lambda\Omega^T\,,$$

where Ω is an $n \times n$ orthonormal matrix whose columns represent the eigenvectors of B and Λ is an $n \times n$ diagonal matrix whose diagonal elements represent the eigenvalues of B. For an eigenvalue λ, the corresponding eigenvector is a non-zero vector $\boldsymbol{\omega}$ such that

$$B\boldsymbol{\omega} = \lambda\boldsymbol{\omega}\,.$$

It can be easily shown that the left and right singular vectors of a matrix A are also the eigenvectors of AA^T and A^TA respectively, and the singular values of a matrix A are the square roots of the eigenvalues of the matrices AA^T and A^TA.

The SVD is also related to Principal Component Analysis (PCA) [11] which is a well-known method for unsupervised data analysis. The principal component of a data matrix is the direction in the space of features with the maximum variance. Given an $m \times n$ data matrix whose columns represent data points, the principal components of a matrix can be obtained by calculating the leading eigenvectors of the covariance matrix, or equivalently the leading left singular values of the column-centered data matrix.

The largest k singular values are referred to as the leading k singular values of A and the corresponding singular vectors are referred to as the leading k singular vectors. Those k leading singular values and vectors can be used to calculate a rank-k approximation of A. This decomposition is referred to as the truncated SVD (also known as partial SVD).

Definition 2.2 (Truncated Singular Value Decomposition) *Given an $m \times n$ matrix A, the truncated singular value decomposition of A is*

$$A \approx U_k\Sigma_kV_k^T,$$

where U_k and V_k are orthogonal matrices whose columns represent the leading k left and right singular vectors of A respectively and Σ_k is a $k \times k$ diagonal matrix whose diagonal elements represent the leading k singular values of A.

The matrix $\tilde{A}_k = U_k \Sigma_k V_k^T$ is the best rank-k approximation of A in terms of minimizing the Frobenius and spectral norms of the discrepancy matrix $A - \tilde{A}_k$ and it is accordingly the optimal solution of Problem 2.1. In this case, the Frobenius and spectral norms of the discrepancy matrix are

$$\|A - \tilde{A}_k\|_F = \sqrt{\sum_{i=1}^{r-k} \sigma_{k+i}^2} \,,$$
$$\|A - \tilde{A}_k\|_2 = \sigma_{k+1} \,,$$

where σ_i is the i-th leading singular value of A and r is the rank of A.

2.1.2 Stochastic Singular Value Decomposition (SSVD)

The Stochastic Singular Value Decomposition (SSVD) [12, 13] is a fast algorithm for approximating the singular value decomposition of a data matrix. The SSVD is a randomized algorithm which first calculates an approximate basis for the range of the data matrix, and then calculates an approximate SVD decomposition of the matrix using this approximate basis.

In order to calculate an approximate basis for the range of the data matrix, the SSVD algorithm first generates an $n \times (k+p)$ random matrix Ω with Gaussian distribution where n is the number of columns of the data matrix, k is the target rank and p is an oversampling parameter. The algorithm then multiplies the random matrix Ω with the data matrix A:

$$Y = A\Omega \,.$$

The columns of Y are then orthogonalized to obtain an approximate orthonormal basis Q for the range of A.

After the approximate basis Q is calculated, the algorithm embeds the columns of the data matrix into the subspace of the columns of Q:

$$B = Q^T A \,.$$

The leading k singular values and vectors of B are then calculated using an exact algorithm for truncated SVD.

$$B \approx U_k^{(B)} \Sigma_k^{(B)} V_k^{(B)} .$$

This step is very efficient as the size of the matrix B is $(k+p) \times n$, which is very small compared to the size of the original matrix A. The leading singular values and vectors of A are then approximated using those of B as follows:

$$\tilde{U}_k^{(A)} = Q U_k^{(B)}, \qquad \tilde{\Sigma}_k^{(A)} = \Sigma_k^{(B)}, \qquad \tilde{V}_k^{(A)} = V_k^{(B)} . \tag{2.1}$$

Although the SSVD algorithm in this simple form is very time and memory efficient, it is not effective in approximating the singular values and vectors of a data matrix when the singular spectrum of the data matrix decays slowly. In order to address this problem, Halko et al. [12] suggested the use of a few steps of a power iteration. In this case, the matrix Y is calculated iteratively as

$$Y = \left(AA^T\right)^q A\Omega .$$

This multiplication enhances the accuracy of the SVD approximation but increases the computational complexity of the algorithm and the number of passes over the data matrix. When implementing the power iteration, the authors suggested orthogonalizing the columns of Y after each power iteration in order to avoid the effect of rounding errors on extinguishing the information associated with small singular values.

Halko et al. [12] showed that the SSVD algorithm achieves good approximation accuracies when using a very small number of power iterations ($q = 1$ or 2) and a small oversampling parameter ($p = 5$ to 10). Algorithm 1 shows the steps of the SSVD algorithm [13, Algorithm 4.3]. The function *orth* orthogonalizes the columns of an input matrix, and the function *tsvd* calculates the truncated singular value decomposition.

2.1.3 QR Decomposition

The QR decomposition is a factorization of a matrix into the product of an orthogonal matrix Q and an upper-triangular matrix R.

Definition 2.3 (QR Decomposition) *Given an $m \times n$ matrix A where $m \geq n$, the QR decomposition is*

$$A = QR,$$

where Q is an $m \times n$ orthogonal matrix and R is an $n \times n$ upper-triangular matrix.

Algorithm 1 Stochastic Singular Value Decomposition (SSVD) [13, Algorithm 4.3]

Input: Data matrix A, Target rank k, Number of iterations q, Oversampling parameter p
Output: Approximate truncated SVD of A
Steps:

1. Generate an $n \times (k+p)$ random matrix Ω with Gaussian distribution
2. $Y = A\Omega$
3. $Q = orth(Y)$
4. Repeat q times
   - $Y = AA^T Q$
   - $Q = orth(Y)$
5. $B = Q^T A$
6. $\left[U_k^{(B)}, \Sigma_k^{(B)}, V_k^{(B)}\right] = tsvd\,(B, k)$
7. $\tilde{U}_k^{(A)} = QU_k^{(B)}, \qquad \tilde{\Sigma}_k^{(A)} = \Sigma_k^{(B)}, \qquad \tilde{V}_k^{(A)} = V_k^{(B)}\ .$

Different methods have been proposed to calculate the QR decomposition of a matrix. These methods include the use of the GramSchmidt process, Householder reflections, and the Givens rotations [10, Chapter 5]. A column pivoting strategy [10] can also be employed to improve the numerical stability, and to calculate the QR decomposition of rank-deficient matrices. In this case, a column permutation matrix P is introduced and the QR factorization is calculated as

$$PA = QR,$$

where P is an $m \times m$ matrix which permutes the columns of A.

Rank-revealing QR (RRQR) decomposition [14–17] is a category of QR decomposition in which additional constraints are imposed on the sub-matrices of R. These constraints separate the linearly-independent columns of the matrix from the dependent ones. This separation reveals the rank of the matrix.

2.2 Data Clustering

Data clustering is an unsupervised learning task which aims at organizing data instances into groups based on their similarity. Data instances are usually represented as points in a multidimensional space of features, and the similarity between these data instances is calculated based on their closeness in that space.

Data clustering is a crucial task in unsupervised data analysis, which has many useful applications. One category of these applications is concerned with the organization of data instances for end-user analysis. For instance, document clustering has been used to allow users to browse large collections of documents [18], organize new stories [19], and group related search results [20]. Another category of applications is concerned with the use of clustering as a pro-processing step for other data mining and machine learning tasks. An example of such application is the clustering of data instances prior to calculating a low-rank approximation of their matrix [21].

Data clustering algorithms can be generally categorized into hierarchical and partitional [3, 22]. Hierarchical clustering constructs a hierarchy of nested clusters, while partitional clustering (also known as flat clustering) divides data points into non-overlapped clusters such that a specific criterion function is optimized.

Besides centralized clustering algorithms, different algorithms have been proposed to cluster data that are distributed across different computing nodes [23–25].

The rest of this section reviews some of the basic data clustering algorithms. Some of these algorithms have been used to evaluate the algorithms presented in this dissertation.

2.2.1 Hierarchical Clustering

Hierarchical algorithms for data clustering construct a hierarchy of nested clusters. The top cluster in the hierarchy contains all data points while each cluster at the bottom contains a single data point (also called singleton clusters). The data points of each intermediate cluster are divided into a set of sub-clusters (usually two). The output of a hierarchical clustering algorithm can be graphically represented as a tree (also called a dendogram) with the root node representing the top cluster, and the leaf nodes representing singleton clusters.

In contrast to partitional algorithms, hierarchical clustering does not require the number of clusters to be specified. However, a flat partitioning of the data points can be obtained by traversing the cluster hierarchy from the root node until a predefined number of clusters is obtained. Another method to obtain a flat partitioning is by cutting off the hierarchy based on a predefined threshold of the combination similarity between clusters.

Hierarchical algorithms are either agglomerative (bottom-up) or divisive (top-down). Agglomerative algorithms start with singleton clusters, and then successively merge the most similar pair of clusters until all data points are grouped in one cluster. Divisive algorithms start with all data points in one cluster and then successively partition the data points into two dissimilar groups until singleton clusters are obtained.

Agglomerative algorithms are more commonly used, especially in document clustering, and they are usually referred to as hierarchical agglomerative clustering (HAC). Instances of HAC algorithms differ in the way they calculate the similarity/distance between pair of clusters. The most commonly used instances are single-link, complete-link, and average-link clustering [26, Section 5.5], which measure the similarity between two clusters based on the maximum, minimum, and average of the similarities between their data instances, respectively.

2.2.2 k-Means

The k-means algorithm [22] is the most widely used algorithm for data clustering. The goal of the algorithm is to group the data points into k clusters such that the Euclidean distances between data points in each cluster and that cluster's centroid is minimized. Given a set

of n data points described by m features, let A be an $m \times n$ data matrix whose column vectors $A_{:i}$'s represent the data points in the feature space, the corresponding optimization problem can be written as:

$$\mathbf{P}^{\star} = \arg\min_{\mathbf{P}} \sum_{j=1}^{k} \sum_{i \in \mathcal{P}_j} \left\| A_{:i} - \boldsymbol{\mu}_{(j)} \right\|^2 \tag{2.2}$$

where $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$ is a possible partitioning of the data points, $\mathcal{P}_j$ is the set of data points that belong to cluster j, and $\boldsymbol{\mu}_{(j)}$ is the column vector that represents the centroid of cluster j.

The k-means clustering problem is NP-hard [27]. An iterative algorithm, namely Lloyd's algorithm [28], is usually used for the optimization of this criterion function. The steps of the Lloyd's algorithm are shown in Algorithm 2.

The Lloyd's algorithm is guaranteed to converge to a local minimum of the criterion function, and the quality of the obtained solution depends on the initial partitioning of the data points $\mathbf{P}^{(0)}$. One heuristic to obtain better solutions is to repeat the algorithm starting from different initial solutions, and then take the final solution with the minimum value of the criterion function. The output of another clustering algorithm (e.g. hierarchical clustering) can also be used to obtain an initial partitioning of the data points.

Another way to optimize the k-means criterion function is to use an iterative algorithm that incrementally moves data points from one cluster to another such that a maximum improvement in the criterion function is achieved [29, 30]. This algorithm can be used in a stand-alone manner to optimize the criterion function starting from a random partitioning, or to refine the output of the Lloyd's algorithm. The incremental optimization technique, however simple, has been successively used to optimize many clustering criterion functions [31–33].

There are many variants of the basic k-means algorithm. Some of these variants are outlined below:

- k-medoids [34] uses the medoid of the data points in each cluster, instead of the mean, as the cluster centroid.
- Bisecting k-means [30] starts with all data points in one cluster, and then successively apply the k-means algorithm to divide data points into two clusters (namely bisections) until k clusters are obtained.

Algorithm 2 The Lloyd's Algorithm for k-Means Clustering

Inputs: Data matrix A, Number of clusters k, Maximum number of iterations $t_{\max}$
Outputs: Data partitions $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$
Steps:

1. Partition the data points into k random clusters: $\mathbf{P}^{(0)} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$, Initialize $t = 1$

2. Compute the cluster centroids $M = \left[\boldsymbol{\mu}_{(1)}, \boldsymbol{\mu}_{(2)}, ...\boldsymbol{\mu}_{(k)}\right]$:

$$\boldsymbol{\mu}_{(j)} = \frac{1}{|\mathcal{P}_j|} \sum_{i \in \mathcal{P}_j} A_{:i}$$

3. Assign each data point i to the cluster with the closest centroid:

$$\boldsymbol{y}_i = \arg\min_j \left\| A_{:i} - \boldsymbol{\mu}_{(j)} \right\|^2$$

4. Obtain the new partitioning: $\mathbf{P}^{(t)} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$:

$$\mathcal{P}_j = \{i : \ \boldsymbol{y}_i = j\}$$

5. If ($\mathbf{P}^{(t)} \neq \mathbf{P}^{(t-1)}$ and $t < t_{\max}$), increment t and go to Step 2. Otherwise, return $\mathbf{P}^{(t)}$

- Kernel k-means [35, 36] is a kernel-based version of the k-means algorithm in which the distance between a data point and a centroid is calculated in terms of the kernel matrix K which encodes the inner-products between data points in the feature space:

$$\left\|\phi\left(A_{:i}\right)-\phi\left(\boldsymbol{\mu}_{(j)}\right)\right\|^{2}=K_{ii}-\frac{2}{\left|\mathcal{P}_{j}\right|}\sum_{a\in\mathcal{P}_{j}}K_{ia}+\frac{1}{\left|\mathcal{P}_{j}\right|^{2}}\sum_{a,b\in\mathcal{P}_{j}}K_{ab}\,,$$

  where $\phi\left(.\right)$ is the function that maps data vectors to the high-dimensional feature space implicitly defined by the kernel.

- Weighted k-means [35] assigns weights to different data points according to their importance. The new clustering criterion can be written as:

$$\mathbf{P}^{\star}=\arg\min_{\mathbf{P}}\sum_{j=1}^{k}\sum_{i\in\mathcal{P}_{j}}\lambda_{i}\left\|A_{:i}-\boldsymbol{\mu}_{(j)}\right\|^{2}\,,$$

  where λ_i is the weight of data point $A_{:i}$. The kernel k-means formula can be generalized to include weights for different data points. It has been shown that weighted kernel k-means with specific weights is equivalent to finding the normalized-cut of the similarity graph of data points [35, 36].

- Fuzzy c-means [37–39] allows overlap between clusters by assigning membership values to data points with respect to each cluster: $U=\left[U_{ij}\right]_{n\times k}$ such that $0\leq U_{ij}\leq 1$ and $\sum_{j}U_{ij}=1$. The new clustering criterion can be written as:

$$U^{\star}=\arg\min_{U}\sum_{j=1}^{k}\sum_{i=1}^{n}U_{ij}^{\gamma}\left\|A_{:i}-\boldsymbol{\mu}_{(j)}\right\|^{2}\,,$$

  where γ is any real number greater than 1. As γ approaches 1, the clustering becomes more crisp, and as it increases, the clustering becomes more fuzzy. A variant of the Lloyd's algorithm can be used to update the cluster centroids and membership values as follows:

$$\boldsymbol{\mu}_{(j)}=\frac{\sum_{i=1}^{n}U_{ij}^{\gamma}A_{:i}}{\sum_{i=1}^{n}U_{ij}^{\gamma}},\quad U_{ij}=\left(\sum_{c=1}^{k}\left(\frac{\left\|A_{:i}-\boldsymbol{\mu}_{(j)}\right\|}{\left\|A_{:i}-\boldsymbol{\mu}_{(c)}\right\|}\right)^{2/(\gamma-1)}\right)^{-1}\,.$$

2.2.3 Spherical k-Means

Spherical k-means [40] is a variant of the basic k-means algorithm that uses cosine similarity between data points instead of the Euclidean distance. The use of cosine similarities with k-means has been proposed by Rasmussen [41] for the purpose of document clustering. However, the name spherical k-means was suggested by Dhillon et al [40, 42] as the algorithm partitions the high-dimensional hypersphere on which data points lie using a collection of great hyper-circles.

Spherical k-means was originally proposed for document clustering, but it can be generally used for other directional data instances (where the direction is relevant but not the magnitude). The criterion function of spherical k-means is the sum of inner-products between vectors of data points that belong to each cluster and that cluster's centroid vector. The corresponding optimization problem can be written as:

$$\mathbf{P}^{\star} = \max_{\mathbf{P}} \sum_{j=1}^{k} \sum_{i \in \mathcal{P}_j} {A_{:i}}^T \boldsymbol{\mu}_{(j)} \, . \tag{2.3}$$

The criterion function of spherical k-means can be minimized using a variant of Lloyd's algorithm (Algorithm 2) in which Steps 2 and 3 are replaced with

$$\boldsymbol{\mu}_{(j)} = \frac{\sum_{i \in \mathcal{P}_j} A_{:i}}{\left\| \sum_{i \in \mathcal{P}_j} A_{:i} \right\|}, \qquad \boldsymbol{y}_i = \arg\max_j {A_{:i}}^T \boldsymbol{\mu}_{(j)} \, .$$

Dhillon et al. [43, 44] observed that the Lloyd's algorithm tends to get stuck at a poor local maximum when applied to document data sets. They suggested that the output of spherical k-means could be refined by using the incremental optimization technique described in Section 2.2.2.

2.2.4 Spectral Clustering

Spectral clustering is a family of clustering algorithms that are based on spectral partitioning of a graph whose adjacency matrix encodes measures of similarity between data points. A general framework for spectral clustering algorithms is shown in Algorithm 3.

Given an $n \times n$ matrix S whose elements represent measures of similarity between data points, the algorithm starts by constructing an undirected similarity graph. The vertices

Algorithm 3 Spectral Clustering

Inputs: Similarity matrix S, Number of clusters k
Outputs: Data partitions $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$
Steps:

1. Construct a similarity graph whose vertices V represent the data points and calculate the adjacency matrix: $W = H(S)$
2. Compute the (normalized) graph Laplacian $L = F(W)$
3. Solve an eigendecomposition problem of L and compute the second k smallest eigenvectors: $\boldsymbol{q}_{(2)}, ..., \boldsymbol{q}_{(k+1)}$
4. Represent data points in a k-dimensional space $\mathbb{R}^k$ using the eigenvectors $\boldsymbol{q}_{(2)}, ..., \boldsymbol{q}_{(k+1)}$
5. Cluster the data points in the new space into k clusters $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$ using the k-means algorithm

of the graph V represent the data points, and its adjacency matrix W (where $W_{ij} \geq 0$) is constructed based on the pairwise-similarity matrix S. The algorithm then calculates the graph Laplacian matrix L, and solves an eigenvalue decomposition problem of L. The eigenvectors that correspond to the second k smallest eigenvectors are used to represent data points in a k-dimensional space where a traditional clustering algorithm (like k-means) is applied.

The graph Laplacian (un-normalized) is defined as: $L = D - W$ where D is a $n \times n$ diagonal matrix, also called the degree matrix, whose elements are the sum of weights at each node of the similarity graph: $D_{ii} = \sum_{j=1}^{n} W_{ij}$. In some spectral clustering algorithms, normalized forms of graph Laplacian are used.

The clustering criterion function of spectral clustering can be interpreted as the minimization of the cut between partitions of the graph vertices normalized by the weights of these partitions. In the case of bi-partitions, this weighted cut between two vertex sets A, B can be generally defined as [45]:

$$\mathrm{C}(A, B) = \frac{\mathrm{cut}(A, B)}{\mathrm{weight}(A)} + \frac{\mathrm{cut}(A, B)}{\mathrm{weight}(B)} = \frac{\boldsymbol{q}^T L \boldsymbol{q}}{\boldsymbol{q}^T P \boldsymbol{q}}, \tag{2.4}$$

where $\mathrm{cut}(A, B) = \sum_{i \in A, j \in B} W_{ij}$ defines the sum of edge weights between A and B, and $\mathrm{weight}(A)$ defines the weight of a vertex set A. Two vertex sets A, B are balanced if

weight(A) = weight(B). $\boldsymbol{q}$ is an $n \times 1$ cluster (partition) indicator vector. $\boldsymbol{q}_i = 1$ if vertex i belongs to A, and -1 if it belongs to B. L is the graph Laplacian and P is an $n \times n$ matrix such that weight$(A) = \sum_{i,j \in A} P_{ij}$.

The optimization problem that corresponds to minimizing the cut can be written as [45]:

$$\min_{q \neq 0} \frac{\boldsymbol{q}^T L \boldsymbol{q}}{\boldsymbol{q}^T P \boldsymbol{q}}, \quad \text{s.t.} \quad \boldsymbol{q}^T P \boldsymbol{e} = 0 \,,$$

where $\boldsymbol{e}$ is the all-ones vector. A relaxed solution to this problem is equal to the eigenvector corresponding to the second smallest eigenvalue of the generalized eigendecomposition problem:

$$L\boldsymbol{q} = \lambda P \boldsymbol{q} \,. \tag{2.5}$$

Different definitions of the matrix P leads to different types of weighted cuts. Ratio cut $\mathrm{R_{cut}}$ [46], normalized cut $\mathrm{N_{cut}}$ [47], and min-max cut $\mathrm{MM_{cut}}$ [48] could be obtained from Equation (2.4) by setting P to I, D, and W respectively:

$$\mathrm{R_{cut}}(A,B) = \frac{\mathrm{cut}(A,B)}{|A|} + \frac{\mathrm{cut}(A,B)}{|B|} = \frac{\boldsymbol{q}^T L \boldsymbol{q}}{\boldsymbol{q}^T \boldsymbol{q}} \,,$$

$$\mathrm{N_{cut}}(A,B) = \frac{\mathrm{cut}(A,B)}{\mathrm{cut}(A,A) + \mathrm{cut}(A,B)} + \frac{\mathrm{cut}(A,B)}{\mathrm{cut}(B,B) + \mathrm{cut}(A,B)} = \frac{\boldsymbol{q}^T L \boldsymbol{q}}{\boldsymbol{q}^T D \boldsymbol{q}} \,,$$

$$\mathrm{MM_{cut}}(A,B) = \frac{\mathrm{cut}(A,B)}{\mathrm{cut}(A,A)} + \frac{\mathrm{cut}(A,B)}{\mathrm{cut}(B,B)} = \frac{\boldsymbol{q}^T L \boldsymbol{q}}{\boldsymbol{q}^T W \boldsymbol{q}} \,.$$

In a similar way, a k-way partitioning of the graph can be obtained by computing the eigenvectors with the second k smallest eigenvalues, and then applying the k-means clustering as shown in Algorithm 3. Let Y be an $k \times n$ matrix whose rows are the k eigenvectors of L corresponding to the second k smallest eigenvalues. The columns of Y are usually used to represent the data points in a k-dimensional space $\mathbb{R}^k$ where the k-means is applied.

Other well-known algorithms for spectral clustering are those proposed by Meila and Shi [49] and Ng et al [4]. Meila and Shi [49] present a random-walk view of spectral clustering. In this algorithm, the eigendecomposition problem $L\boldsymbol{q} = \lambda \boldsymbol{q}$ is solved with a form of normalized graph Laplacian:

$$L = I - D^{-1} W \,.$$

Ng et al [4] proposed the use of another form of normalized graph Laplacian:

$$L = I - D^{-1/2} W D^{-1/2} \,. \tag{2.6}$$

In this algorithm, the eigenvalue decomposition problem $L\boldsymbol{q} = \lambda \boldsymbol{q}$ is solved, and the vectors of data points are normalized to unit vector before applying the k-means algorithm.

In all spectral clustering algorithms, the adjacency matrix W can constructed from the similarity measures in different ways. One way is to use a fully-connected graph in which all data points are connected to each other (i.e., $W = S$). This graph is usually used if the similarity matrix represents the local neighborhood of data points [50]. Example of such a similarity is the Gaussian similarity function: $W = \exp\left(-E_{ij}^2/2\sigma^2\right)$ where E_{ij} is the distance between data points i and j, σ is a parameter that controls the width of the neighborhood area. The graph can also be constructed by connecting each data point to its most similar k data points (k-nearest neighbor graph) or by connecting two data points if their similarity is above a predefined threshold (ϵ-neighborhood graph).

The advantage of spectral clustering compared to other clustering algorithms, like k-means, is that the solution is deterministic and corresponds to the global minimum of the clustering criterion function. On the other hand, spectral clustering is computationally expensive, as it is based on the eigenvalue decomposition of the graph Laplacian matrix. This makes its application to large data sets infeasible.

2.2.5 Affinity Propagation

The Affinity Propagation (AF) [51] is a state-of-the-art clustering method which depends on propagating real-valued messages between data points on their similarity graph. The basic idea of the algorithm is to simultaneously consider all data points as potential cluster centroids (also called exemplars) and keep propagating real-valued messages between data points until a set of good exemplars emerge.

Two types of messages are propagated between data points: the availability messages and the responsibility messages. The availability message $a\,(i,k)$ is sent from a data point i to a candidate exemplar k to reflect how well-suited k is to serve as the exemplar for i based on the information collected from other potential exemplars. On the other hand, the responsibility message $r\,(k,i)$ is sent from a candidate exemplar k to a data point i to reflect how appropriate it is for i to choose k as its exemplar based on the information collected from other supporting data points.

The AF algorithm defines update rules for availability and responsibility scores. These update rules are based on the similarity between data points as well as other availability

and responsibility scores. The responsibility update rule makes the candidate exemplars compete for the ownership of a data point while the availability update rule gathers information about how good a candidate exemplar is going to be for each data point. The AF algorithm keeps updating the responsibility and availability scores and propagating them between data points until some stopping criterion is met. At this point, the availability and responsibility scores are combined to identify the exemplar for each data point.

The affinity propagation algorithm achieves good clustering results on various data sets, especially those with a large number of clusters. It also does not require the number of clusters to be specified in advance, as it considers all data points as potential exemplars. The AF algorithm, however, has some limitations. First, it requires the specification of two parameters: a preference vector which reflects how likely each data point is to be an exemplar, and a damping factor that is used in the update rules to avoid numerical oscillations. It is usually difficult to determine the optimal values for these parameters. Adaptive algorithms have been proposed to select values for these parameters to improve clustering performance and avoid oscillations [52, 53]. In addition, the AF algorithm does not scale well to handle large data sets [54]. Recent work has been proposed to speed up the algorithm by reducing the number of messages to be propagated between data points [54, 55].

2.2.6 Evaluating the Quality of Clusters

Different quality measures have been proposed to quantify the goodness of the clusters obtained by different algorithms. These quality measures (also called quality indexes) are either internal or external. Internal measures of clustering quality quantify the compactness of data points inside clusters and the separation between different clusters without using any external knowledge about the true partitioning of the data points. The internal quality indexes are usually used in the absence of external knowledge about the labels of data points. Examples of internal quality indexes include overall similarity, partition index, and separation index [56–58] .

Let $\mathbf{P} = \{\mathcal{P}_1, ..\mathcal{P}_k\}$ be a partitioning of data points obtained by a clustering algorithm, n be the total number of data points, and $|\mathcal{P}_j|$ be the number of data points that belong to cluster j.

The overall similarity $S_{\mathbf{P}}$ measures the pairwise similarities between all points that belong to the same cluster. The higher the overall similarity, the better is the clustering

solution.

$$S_{\mathbf{P}} = \sum_{j=1}^{k} \left(\frac{1}{|\mathcal{P}_j|^2} \sum_{a,b \in \mathcal{P}_j} \text{sim}(a,b) \right) .$$

The partition index SC is the sum of the ratios of compactness of data points inside each cluster to the separation of this cluster from other clusters. The compactness of a cluster is estimated as the sum of distances between data points in the cluster and its centroid, while the separation is estimated as the distances between the centroid of the cluster and the centroids of other clusters. The ratio for each cluster is weighted by the size of that cluster:

$$\text{SC} = \sum_{j=1}^{k} \frac{1}{|\mathcal{P}_j|} \left(\frac{\sum_{i \in \mathcal{P}_j} \left\| A_{:i} - \boldsymbol{\mu}_{(j)} \right\|^2}{\sum_{l=1}^{k} \left\| \boldsymbol{\mu}_{(l)} - \boldsymbol{\mu}_{(j)} \right\|^2} \right) .$$

The separation index SI is a variant of the partition index which uses the minimum distance between the centroids of all clusters to estimate the separation between clusters:

$$\text{SI} = \frac{1}{n} \left(\frac{\sum_{j=1}^{k} \sum_{i \in \mathcal{P}_j} \left\| A_{:i} - \boldsymbol{\mu}_{(j)} \right\|^2}{\min\limits_{a,b} \left\| \boldsymbol{\mu}_{(a)} - \boldsymbol{\mu}_{(b)} \right\|^2} \right) .$$

Lower values of SC and SI indicate better partitioning. The reader is referred to [56–58] for more details about internal quality measures.

External measures, on the other hand, compare the partitioning obtained by the clustering algorithm with a ground-truth partitioning created by human annotators. Examples of external measures include F-measure [59], entropy [60], purity [60], and the Normalized Mutual Information (NMI) [61].

Let $\mathbf{C} = \{\mathcal{C}_1, ..\mathcal{C}_c\}$ be the ground-truth partitioning of the data points, and C_{ij} be the number of data points that belong to class i and cluster j. The external measures use the contingency matrix $C = [C_{ij}]_{c \times k}$ to estimate the quality of $\mathbf{P}$.

The F-measure is a well-known measure of quality in information retrieval which combines precision and recall [59]. The higher the F-measure, the better is the clustering solution. Given the contingency matrix C, the precision and recall matrices P and R can be calculated as:

$$P_{ij} = \frac{C_{ij}}{|\mathcal{C}_i|}, \;\; R_{ij} = \frac{C_{ij}}{|\mathcal{P}_j|}$$

where $|\mathcal{C}_i| = \sum_j C_{ij}$ is the number of data points in class i, and $|\mathcal{P}_j| = \sum_i C_{ij}$ is the number of data points in cluster j. The F-measure matrix can be calculated as:

$$F_{ij} = \frac{2P_{ij}R_{ij}}{P_{ij} + R_{ij}}$$

To evaluate the quality of clusters, each class i is mapped to cluster j which has the maximum F_{ij} measure, and the F-measure of this class is set to this maximum value: $F_i = \max_j \{F_{ij}\}$. The overall F-measure of the partitioning is calculated as the weighted sum of the F-measures of individual classes.

The entropy is another external quality index that measures the homogeneity of clusters with respect to classes [60]. The lower the entropy, the more homogeneous are the clusters and the better is the clustering algorithm. Let $P(\mathcal{P}_j|\mathcal{C}_i) = C_{ij}/|\mathcal{C}_i|$ be the probability that a member of cluster j belongs to class i, the entropy of a cluster j is calculated as

$$E_j = -\sum_{i=1}^{c} P(\mathcal{P}_j|\mathcal{C}_i) \log\left(P(\mathcal{P}_j|\mathcal{C}_i)\right) \ .$$

The entropy of a partitioning $\mathbf{P}$ is then calculated as the sum of cluster entropy measures weighted by the cluster sizes.

The third external quality index presented is the purity which measures the average precision of clusters relative to their best matching classes [60]. The higher the purity, the better is the clustering solution. The purity of a cluster j is calculated by first assigning cluster j to the most dominant cluster j, and then dividing the number of data points that belong to cluster j and class i by the total number of data points in cluster j:

$$P_j = \frac{1}{|\mathcal{P}_j|} \max_i \{C_{ij}\} \ .$$

The overall purity is then calculated as the weighted average of purity measures of clusters.

The Normalized Mutual Information (NMI) [61] is a well-known external quality index, which measures the amount of information shared between the cluster and class labels. Let X and Y be discrete random variables which represent the class and cluster labels respectively. The Normalized Mutual Information (NMI) is calculated as

$$\mathrm{NMI}(Y;X) = \frac{\mathrm{MI}(Y;X)}{\sqrt{H(Y)\,H(X)}}$$

where $\mathrm{MI}(Y;X)$ is the mutual information of random variables Y and X; and $H(Y)$ and $H(X)$ are the entropies of the random variables Y and X respectively. The mutual information $\mathrm{MI}(Y;X)$ measures the amount of information that can be obtained from the cluster labels by observing the class labels. $\mathrm{MI}(Y;X)$ can be calculated as

$$\begin{aligned}\mathrm{MI}(Y;X) &= \sum_{j=1}^{k}\sum_{i=1}^{c} P(\mathcal{P}_j,\mathcal{C}_i)\log\frac{P(\mathcal{P}_j,\mathcal{C}_i)}{P(\mathcal{P}_j)P(\mathcal{C}_i)} \\ &= \sum_{j=1}^{k}\sum_{i=1}^{c}\frac{C_{ij}}{n}\log\frac{nC_{ij}}{|\mathcal{C}_i||\mathcal{P}_j|}\,,\end{aligned}$$

where $P(\mathcal{P}_j)$, $P(\mathcal{C}_i)$ and $P(\mathcal{P}_j,\mathcal{C}_i)$ are the probabilities that a data point belongs to cluster j, class i and their intersection respectively. The entropies $H(Y)$ and $H(X)$ measure the amount of information contained in the cluster and class labels respectively. The entropy measures can be calculated as

$$\begin{aligned}H(Y) &= -\sum_{j=1}^{k}P(\mathcal{P}_j)\log P(\mathcal{P}_j) = -\sum_{j=1}^{k}\frac{|\mathcal{P}_j|}{n}\log\frac{|\mathcal{P}_j|}{n}\,, \\ H(X) &= -\sum_{i=1}^{c}P(\mathcal{C}_i)\log P(\mathcal{C}_i) = -\sum_{i=1}^{c}\frac{|\mathcal{C}_i|}{n}\log\frac{|\mathcal{C}_i|}{n}\,.\end{aligned}$$

The value of the NMI measure is between 0 and 1 and the higher the NMI measure, the better is the clustering solution.

The NMI measure have been used to evaluate the clustering performance in much recent work. It has also been observed that NMI obtains results that are strongly correlated with other well-known measures such as the F-measure [62]. Therefore, the NMI measures will be used to evaluate the quality of the clustering methods in the experiments conducted in this dissertation.

Chapter 3

Greedy Column Subset Selection

This chapter presents a fast and accurate algorithm for Column Subset Selection (CSS). The algorithm minimizes an objective function which measures the reconstruction error of the data matrix based on the subset of selected columns. The chapter first presents a novel recursive formula for calculating the reconstruction error of the data matrix, and then proposes a fast and accurate algorithm which selects the most representative columns in a greedy manner. The presented algorithm can be used to calculate a column-based low-rank approximation of the data matrix, as well as a low-dimensional embedding of all columns in the subspace of selected ones.

The chapter is organized as follows: Section 3.1 gives an overview of the Column Subset Selection (CSS) problem. Section 3.2 presents a novel recursive formula for the selection criterion. Section 3.3 proposes an effective greedy algorithm for column subset selection as well as memory and time efficient variants of the algorithm. Section 3.4 reviews previous work on the CSS problem. Section 3.5 presents an empirical evaluation of the proposed method compared to other randomized and deterministic methods.

3.1 Column Subset Selection (CSS)

The Column Subset Selection (CSS) problem can be generally defined as the selection of the most representative columns of a data matrix [63–65]. The CSS problem generalizes the problem of selecting representative data instances as well as the unsupervised feature selection problem. Both are crucial tasks, that can be directly used for data analysis or as pre-processing steps for developing fast and accurate algorithms in data mining and machine learning. The CSS problem can be formally defined as follows:

Problem 3.1 (Column Subset Selection) *Given an $m \times n$ matrix A and an integer l, find a subset of columns $\mathcal{L}$ such that $|\mathcal{L}| = l$ and*

$$\mathcal{L} = \arg\min_{\mathcal{S}} F(\mathcal{S}),$$

where $F(\mathcal{S})$ is the column subset selection criterion, $\mathcal{S}$ is the set of the indices of the candidate columns and $\mathcal{L}$ is the set of the indices of the selected columns.

Although different criteria for column subset selection can be defined, a common criterion that has been used in much recent work measures the discrepancy between the original matrix and the approximate matrix reconstructed from the subset of selected columns [63–73]. Most of the recent work either develops CSS algorithms that directly optimize this criterion or uses this criterion to assess the quality of the proposed CSS algorithms. In the present work, the following criterion is optimized:

Definition 3.1 (Column Subset Selection Criterion) *Let A be an $m \times n$ matrix and $\mathcal{S}$ be the set of the indices of the candidate columns, the column subset selection criterion is defined as:*

$$F(\mathcal{S}) = \|A - P^{(\mathcal{S})}A\|_F^2,$$

where $P^{(\mathcal{S})}$ is an $m \times m$ projection matrix which projects the columns of A onto the span of the set $\mathcal{S}$ of columns.

The criterion $F(\mathcal{S})$ represents the sum of squared errors between the original data matrix A and its rank-l column-based approximation (where $l = |\mathcal{S}|$),

$$\tilde{A}_{\mathcal{S}} = P^{(\mathcal{S})}A\,. \tag{3.1}$$

In other words, the criterion $F(\mathcal{S})$ calculates the Frobenius norm of the residual matrix $E = A - \tilde{A}_{\mathcal{S}}$. Other types of matrix norms can also be used to quantify the reconstruction error. Some of the recent work on the CSS problem [63–65] derives theoretical bounds for both the Frobenius and spectral norms of the residual matrix. The present work, however, focuses on developing algorithms that minimize the Frobenius norm of the data matrix.

The projection matrix $P^{(\mathcal{S})}$ can be calculated as:

$$P^{(\mathcal{S})} = A_{:\mathcal{S}} \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}}\right)^{-1} A_{:\mathcal{S}}^T\,, \tag{3.2}$$

where $A_{:\mathcal{S}}$ is the sub-matrix of A which consists of the columns corresponding to $\mathcal{S}$.

It should be noted that if the subset of columns $\mathcal{S}$ is known, the projection matrix $P^{(\mathcal{S})}$ can be derived as follows. The columns of the data matrix A can be approximated as linear combinations of the subset of columns $\mathcal{S}$:

$$\tilde{A}_{\mathcal{S}} = A_{:\mathcal{S}} T \,,$$

where T is an $l \times n$ matrix of coefficients which can be found by solving the following optimization problem.

$$T^* = \arg\min_{T} \|A - A_{:\mathcal{S}} T\|_F^2 \,.$$

This is a least-squares problem whose closed-form solution is

$$T^* = \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}}\right)^{-1} A_{:\mathcal{S}}^T A \,.$$

Substituting with T^* in $\tilde{A}_{\mathcal{S}}$ gives

$$\tilde{A}_{\mathcal{S}} = A_{:\mathcal{S}} T = A_{:\mathcal{S}} \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}}\right)^{-1} A_{:\mathcal{S}}^T A = P^{(\mathcal{S})} A \,.$$

The set of selected columns (i.e., data instances or features) can be directly presented to a data analyst to learn about the insights of the data, or they can be used to preprocess the data for further analysis. For instance, the selected columns can be used to obtain a low-dimensional representation of all columns into the subspace of selected ones. This representation can be calculated as follows.

1. Calculate an orthonormal basis Q for the selected columns,

   $$Q = orth\left(A_{:\mathcal{S}}\right) \,,$$

   where $orth\left(.\right)$ is a function that orthogonalizes the columns of its input matrix and Q is an $m \times l$ orthogonal matrix whose columns span the range of $A_{:\mathcal{S}}$. The matrix Q can be obtained by applying an orthogonalization algorithm such as the Gram-Schmidt algorithm to the columns of $A_{:\mathcal{S}}$, or by calculating the Singular Value Decomposition (SVD) or the QR decomposition of $A_{:\mathcal{S}}$ [10].

2. Embed all columns of A into the subspace of Q,

   $$W = Q^T A \,, \tag{3.3}$$

   where W is an $l \times n$ matrix whose columns represent an embedding of all columns into the subspace of selected ones.

The selected columns can also be used to calculate a column-based low-rank approximation of A [69]. Given a subset $\mathcal{S}$ of columns with $|\mathcal{S}| = l$, a rank-l approximation of the data matrix A can be calculated as:

$$\tilde{A}_{\mathcal{S}} = P^{(\mathcal{S})}A = A_{:\mathcal{S}} \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}}\right)^{-1} A_{:\mathcal{S}}^T A \,. \tag{3.4}$$

In order to calculate a rank-k approximation of the data matrix A where $k \leq l$, the following procedure suggested by Boutsidis et al. [65] can be used.

1. Calculate an orthonormal basis Q for the columns of $A_{:\mathcal{S}}$ and embed all columns of A into the subspace of Q:

$$Q = orth\left(A_{:\mathcal{S}}\right) \,,$$
$$W = Q^T A \,,$$

   where Q is an $m \times l$ orthogonal matrix whose columns span the range of $A_{:\mathcal{S}}$ and W is an $l \times n$ matrix whose columns represent an embedding of all columns into the subspace of selected ones.

2. Calculate the best rank-k approximation of the embedded columns using Singular Value Decomposition (SVD):

$$\tilde{W}_k = U_k^{(W)} \Sigma_k^{(W)} V_k^{(W)T} \,,$$

   where $U_k^{(W)}$ and $V_k^{(W)}$ are $l \times k$ and $n \times k$ matrices whose columns represent the leading k left and right singular vectors of W respectively, $\Sigma_k^{(W)}$ is a $k \times k$ matrix whose diagonal elements are the leading k singular values of W, and $\tilde{W}_k$ is the best rank-k approximation of W.

3. Calculate the column-based rank-k approximation of A as:

$$\tilde{A}_{\mathcal{S},k} = Q\tilde{W}_k \,,$$

   where $\tilde{A}_{\mathcal{S},k}$ is a rank-k approximation of A based on the set $\mathcal{S}$ of columns.

This procedure results in a rank-k approximation of A within the column space of $A_{:\mathcal{S}}$ that achieves the minimum reconstruction error in terms of Frobenius norm [65]:

$$T^* = \underset{T,\, rank(T)=k}{\arg\min} \|A - A_{:\mathcal{S}}T\|_F^2 \,.$$

Moreover, the leading singular values and vectors of the low-dimensional embedding W can be used to approximate those of the data matrix as follows:

$$\tilde{U}_k^{(A)} = QU_k^{(W)}, \qquad \tilde{\Sigma}_k^{(A)} = \Sigma_k^{(W)}, \qquad \tilde{V}_k^{(A)} = V_k^{(W)} \tag{3.5}$$

where $\tilde{U}_k^{(A)}$ and $\tilde{V}_k^{(A)}$ are $l \times k$ and $n \times k$ matrices whose columns approximate the leading k left and right singular vectors of A respectively, and $\tilde{\Sigma}_k^{(A)}$ is a $k \times k$ matrix whose diagonal elements approximate the leading k singular values of A.

In Section 3.2, a recursive formula for the selection criterion is presented. This formula allows the development of an efficient algorithm to greedily minimize $F(\mathcal{S})$. The greedy algorithm is presented in Section 3.3.

3.2 Recursive Selection Criterion

The column subset selection criterion presented in Section 3.1 measures the reconstruction error of a data matrix based on the subset of selected columns. The minimization of this criterion is a combinatorial optimization problem whose optimal solution is very difficult to obtain. In this section, a recursive formula for the CSS criterion is presented. This formula allows the development of an efficient greedy algorithm that approximates the optimal solution of the column subset selection problem. The recursive formula of the CSS criterion is based on a recursive formula for the projection matrix $P^{(\mathcal{S})}$ which can be derived as follows.

Lemma 3.1 *Given a set of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,*

$$P^{(\mathcal{S})} = P^{(\mathcal{P})} + R^{(\mathcal{R})} ,$$

where $R^{(\mathcal{R})}$ is a projection matrix which projects the columns of $E = A - P^{(\mathcal{P})}A$ onto the span of the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns,

$$R^{(\mathcal{R})} = E_{:\mathcal{R}} \left(E_{:\mathcal{R}}^T E_{:\mathcal{R}} \right)^{-1} E_{:\mathcal{R}}^T .$$

Proof Define a matrix $B = A_{:\mathcal{S}}^T A_{:\mathcal{S}}$ which represents the inner-product over the columns of the sub-matrix $A_{:\mathcal{S}}$. The projection matrix $P^{(\mathcal{S})}$ can be written as:

$$P^{(\mathcal{S})} = A_{:\mathcal{S}} B^{-1} A_{:\mathcal{S}}^T . \tag{3.6}$$

Without loss of generality, the columns and rows of $A_{:\mathcal{S}}$ and B in Equation (3.6) can be rearranged such that the first sets of rows and columns correspond to $\mathcal{P}$:

$$A_{:\mathcal{S}} = \begin{bmatrix} A_{:\mathcal{P}} & A_{:\mathcal{R}} \end{bmatrix}, \quad B = \begin{bmatrix} B_{\mathcal{P}\mathcal{P}} & B_{\mathcal{P}\mathcal{R}} \\ B_{\mathcal{P}\mathcal{R}}^T & B_{\mathcal{R}\mathcal{R}} \end{bmatrix},$$

where $B_{\mathcal{P}\mathcal{P}} = A_{:\mathcal{P}}^T A_{:\mathcal{P}}$, $B_{\mathcal{P}\mathcal{R}} = A_{:\mathcal{P}}^T A_{:\mathcal{R}}$ and $B_{\mathcal{R}\mathcal{R}} = A_{:\mathcal{R}}^T A_{:\mathcal{R}}$.

Let $S = B_{\mathcal{R}\mathcal{R}} - B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}}$ be the Schur complement [74] of $B_{\mathcal{P}\mathcal{P}}$ in B. Using the block-wise inversion formula [74], B^{-1} can be calculated as:

$$B^{-1} = \begin{bmatrix} B_{\mathcal{P}\mathcal{P}}^{-1} + B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} & -B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} \\ -S^{-1} B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} & S^{-1} \end{bmatrix}$$

Substitute with $A_{:\mathcal{S}}$ and B^{-1} in Equation (3.6):

$$\begin{aligned} P^{(\mathcal{S})} = & \begin{bmatrix} A_{:\mathcal{P}} & A_{:\mathcal{R}} \end{bmatrix} \begin{bmatrix} B_{\mathcal{P}\mathcal{P}}^{-1} + B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} & -B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} \\ -S^{-1} B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} & S^{-1} \end{bmatrix} \begin{bmatrix} A_{:\mathcal{P}}^T \\ A_{:\mathcal{R}}^T \end{bmatrix} \\ = & A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T + A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T - A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} A_{:\mathcal{R}}^T \\ & - A_{:\mathcal{R}} S^{-1} B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T + A_{:\mathcal{R}} S^{-1} A_{:\mathcal{R}}^T . \end{aligned}$$

Take out $A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1}$ as a common factor from the 2^{nd} and 3^{rd} terms, and $A_{:\mathcal{R}} S^{-1}$ from the 4^{th} and 5^{th} terms:

$$\begin{aligned} P^{(\mathcal{S})} = & A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T - A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} S^{-1} \left(A_{:\mathcal{R}}^T - B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T \right) \\ & + A_{:\mathcal{R}} S^{-1} \left(A_{:\mathcal{R}}^T - B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T \right) . \end{aligned}$$

Take out $S^{-1} \left(A_{:\mathcal{R}}^T - B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T \right)$ as a common factor from the 2^{nd} and 3^{rd} terms:

$$P^{(\mathcal{S})} = A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T + \left(A_{:\mathcal{R}} - A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} \right) S^{-1} \left(A_{:\mathcal{R}}^T - B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T \right) . \tag{3.7}$$

The first term of Equation (3.7) is the projection matrix which projects the columns of A onto the span of the subset $\mathcal{P}$ of columns: $P^{(\mathcal{P})} = A_{:\mathcal{P}} \left(A_{:\mathcal{P}}^T A_{:\mathcal{P}} \right)^{-1} A_{:\mathcal{P}}^T = A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T$. The second term can be simplified as follows. Let E be an $m \times n$ residual matrix which is calculated as: $E = A - P^{(\mathcal{P})} A$. The sub-matrix $E_{:\mathcal{R}}$ can be expressed as:

$$E_{:\mathcal{R}} = A_{:\mathcal{R}} - P^{(\mathcal{P})} A_{:\mathcal{R}} = A_{:\mathcal{R}} - A_{:\mathcal{P}} \left(A_{:\mathcal{P}}^T A_{:\mathcal{P}} \right)^{-1} A_{:\mathcal{P}}^T A_{:\mathcal{R}} = A_{:\mathcal{R}} - A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} .$$

Since projection matrices are idempotent, then $P^{(\mathcal{P})}P^{(\mathcal{P})} = P^{(\mathcal{P})}$ and the inner-product $E_{:\mathcal{R}}^T E_{:\mathcal{R}}$ can be expressed as:

$$\begin{aligned} E_{:\mathcal{R}}^T E_{:\mathcal{R}} =& \left(A_{:\mathcal{R}} - P^{(\mathcal{P})}A_{:\mathcal{R}}\right)^T \left(A_{:\mathcal{R}} - P^{(\mathcal{P})}A_{:\mathcal{R}}\right) \\ =& A_{:\mathcal{R}}^T A_{:\mathcal{R}} - A_{:\mathcal{R}}^T P^{(\mathcal{P})} A_{:\mathcal{R}} - A_{:\mathcal{R}}^T P^{(\mathcal{P})} A_{:\mathcal{R}} + A_{:\mathcal{R}}^T P^{(\mathcal{P})} P^{(\mathcal{P})} A_{:\mathcal{R}} \\ =& A_{:\mathcal{R}}^T A_{:\mathcal{R}} - A_{:\mathcal{R}}^T P^{(\mathcal{P})} A_{:\mathcal{R}} \,. \end{aligned}$$

Substituting with $P^{(\mathcal{P})} = A_{:\mathcal{P}} \left(A_{:\mathcal{P}}^T A_{:\mathcal{P}}\right)^{-1} A_{:\mathcal{P}}^T$ gives

$$E_{:\mathcal{R}}^T E_{:\mathcal{R}} = A_{:\mathcal{R}}^T A_{:\mathcal{R}} - A_{:\mathcal{R}}^T A_{:\mathcal{P}} \left(A_{:\mathcal{P}}^T A_{:\mathcal{P}}\right)^{-1} A_{:\mathcal{P}}^T A_{:\mathcal{R}} = B_{\mathcal{R}\mathcal{R}} - B_{\mathcal{P}\mathcal{R}}^T B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}} = S \,.$$

Substituting $\left(A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} A_{:\mathcal{P}}^T\right)$, $\left(A_{:\mathcal{R}} - A_{:\mathcal{P}} B_{\mathcal{P}\mathcal{P}}^{-1} B_{\mathcal{P}\mathcal{R}}\right)$ and S with $P^{(\mathcal{P})}$, $E_{:\mathcal{R}}$ and $E_{:\mathcal{R}}^T E_{:\mathcal{R}}$ respectively, Equation (3.7) can be expressed as:

$$P^{(\mathcal{S})} = P^{(\mathcal{P})} + E_{:\mathcal{R}} \left(E_{:\mathcal{R}}^T E_{:\mathcal{R}}\right)^{-1} E_{:\mathcal{R}}^T \,.$$

The second term is the projection matrix which projects the columns of E onto the span of the subset $\mathcal{R}$ of columns:

$$R^{(\mathcal{R})} = E_{:\mathcal{R}} \left(E_{:\mathcal{R}}^T E_{:\mathcal{R}}\right)^{-1} E_{:\mathcal{R}}^T \,. \tag{3.8}$$

This proves that $P^{(\mathcal{S})}$ can be written in terms of $P^{(\mathcal{P})}$ and R as: $P^{(\mathcal{S})} = P^{(\mathcal{P})} + R^{(\mathcal{R})}$ ■

This means that projection matrix $P^{(\mathcal{S})}$ can be constructed in a recursive manner by first calculating the projection matrix which projects the columns of A onto the span of the subset $\mathcal{P}$ of columns, and then calculating the projection matrix which projects the columns of the residual matrix onto the span of the remaining columns. Based on this lemma, a recursive formula can be developed for $\tilde{A}_{\mathcal{S}}$.

Corollary 3.2 *Given a matrix A and a subset of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,*

$$\tilde{A}_{\mathcal{S}} = \tilde{A}_{\mathcal{P}} + \tilde{E}_{\mathcal{R}} \,,$$

where $E = A - P^{(\mathcal{P})}A$, and $\tilde{E}_{\mathcal{R}}$ is the low-rank approximation of E based on the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns.

Proof Using Lemma (3.1), and substituting with $P^{(\mathcal{S})}$ in Equation (3.1) gives:

$$\tilde{A}_{\mathcal{S}} = P^{(\mathcal{P})}A + E_{:\mathcal{R}} \left(E_{:\mathcal{R}}^T E_{:\mathcal{R}}\right)^{-1} E_{:\mathcal{R}}^T A \,. \tag{3.9}$$

The first term is the low-rank approximation of A based on $\mathcal{P}$: $\tilde{A}_{\mathcal{P}} = P^{(\mathcal{P})}A$. The second term is equal to $\tilde{E}_{\mathcal{R}}$ as $E_{:\mathcal{R}}^T A = E_{:\mathcal{R}}^T E$. To prove that, multiplying $E_{:\mathcal{R}}^T$ by $E = A - P^{(\mathcal{P})}A$ gives:

$$E_{:\mathcal{R}}^T E = E_{:\mathcal{R}}^T A - E_{:\mathcal{R}}^T P^{(\mathcal{P})} A \,.$$

Using $E_{:\mathcal{R}} = A_{:\mathcal{R}} - P^{(\mathcal{P})} A_{:\mathcal{R}}$, the expression $E_{:\mathcal{R}}^T P^{(\mathcal{P})}$ can be written as:

$$E_{:\mathcal{R}}^T P^{(\mathcal{P})} = A_{:\mathcal{R}}^T P^{(\mathcal{P})} - A_{:\mathcal{R}}^T P^{(\mathcal{P})} P^{(\mathcal{P})} \,.$$

This is equal to 0 as $P^{(\mathcal{P})} P^{(\mathcal{P})} = P^{(\mathcal{P})}$ (an idempotent matrix). This means that $E_{:\mathcal{R}}^T A = E_{:\mathcal{R}}^T E$. Substituting $E_{:\mathcal{R}}^T A$ with $E_{:\mathcal{R}}^T E$ in Equation (3.9) proves the corollary. ■

This means that the column-based low-rank approximation of A based on the subset $\mathcal{S}$ of columns can be calculated in a recursive manner by first calculating the low-rank approximation of A based on the subset $\mathcal{P} \subset \mathcal{S}$, and then calculating the low-rank approximation of the residual matrix E based on the remaining columns.

Based on Corollary (3.2), a recursive formula for the column subset selection criterion can be developed as follows.

Theorem 3.3 *Given a set of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,*

$$F(\mathcal{S}) = F(\mathcal{P}) - \|\tilde{E}_{\mathcal{R}}\|_F^2 \,,$$

where $E = A - P^{(\mathcal{P})}A$, and $\tilde{E}_{\mathcal{R}}$ is the low-rank approximation of E based on the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns.

Proof Using Corollary (3.2), the CSS criterion can be expressed as:

$$\begin{aligned} F(\mathcal{S}) &= \left\| A - \tilde{A}_{\mathcal{S}} \right\|_F^2 = \left\| A - \tilde{A}_{\mathcal{P}} - \tilde{E}_{\mathcal{R}} \right\|_F^2 \\ &= \left\| E - \tilde{E}_{\mathcal{R}} \right\|_F^2 = \left\| E - R^{(\mathcal{R})} E \right\|_F^2 \,. \end{aligned}$$

Using the relation between the Frobenius norm and the trace function,[1] the right-hand side can be expressed as:

$$\begin{aligned} \left\| E - R^{(\mathcal{R})} E \right\|_F^2 &= trace\left(\left(E - R^{(\mathcal{R})} E \right)^T \left(E - R^{(\mathcal{R})} E \right) \right) \\ &= trace\left(E^T E - 2E^T R^{(\mathcal{R})} E + E^T R^{(\mathcal{R})} R^{(\mathcal{R})} E \right) \,. \end{aligned}$$

[1] $\|A\|_F^2 = trace\left(A^T A\right)$

As $R^{(\mathcal{R})}R^{(\mathcal{R})} = R^{(\mathcal{R})}$ (an idempotent matrix), $F(\mathcal{S})$ can be expressed as:

$$F(\mathcal{S}) = trace\left(E^T E - E^T R^{(\mathcal{R})} R^{(\mathcal{R})} E\right) = trace\left(E^T E - \tilde{E}_{\mathcal{R}} \tilde{E}_{\mathcal{R}}\right) = \|E\|_F^2 - \|\tilde{E}_{\mathcal{R}}\|_F^2 .$$

Replacing $\|E\|_F^2$ with $F(\mathcal{P})$ proves the theorem. ■

The term $\|\tilde{E}_{\mathcal{R}}\|_F^2$ represents the decrease in reconstruction error achieved by adding the subset $\mathcal{R}$ of columns to $\mathcal{P}$. In the following section, a novel greedy heuristic is presented to optimize the column subset selection criterion based on this recursive formula.

3.3 Greedy Selection Algorithm

This section presents an efficient greedy algorithm to optimize the column subset selection criterion presented in Section 3.1. The algorithm selects at each iteration one column such that the reconstruction error for the new set of columns is minimized. This problem can be formulated as follows:

Problem 3.2 *At iteration t, find column p such that,*

$$p = \arg\min_i \quad F(\mathcal{S} \cup \{i\}) \tag{3.10}$$

where $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

A naïve implementation of the greedy algorithm is to calculate the reconstruction error for each candidate column, and then select the column with the smallest error. This implementation is, however, computationally very complex, as it requires $O(m^2n^2)$ floating-point operations per iteration. A more efficient approach is to use the recursive formula for calculating the reconstruction error. Using Theorem 3.3,

$$F(\mathcal{S} \cup \{i\}) = F(\mathcal{S}) - \|\tilde{E}_{\{i\}}\|_F^2,$$

where $E = A - \tilde{A}_{\mathcal{S}}$ and $\tilde{E}_{\{i\}}$ is the rank-1 approximation of E based on the candidate column i. Since $F(\mathcal{S})$ is a constant for all candidate columns, an equivalent criterion is:

$$p = \arg\max_i \quad \|\tilde{E}_{\{i\}}\|_F^2 \tag{3.11}$$

This formulation selects the column p which achieves the maximum decrease in reconstruction error. Using the properties that: $trace(AB) = trace(BA)$ and $trace(aA) =$

a trace (A) where a is a scalar, the new objective function $\left\|\tilde{E}_{\{i\}}\right\|_F^2$ can be simplified as follows:

$$\begin{aligned}\left\|\tilde{E}_{\{i\}}\right\|_F^2 &= trace\left(\tilde{E}_{\{i\}}^T\tilde{E}_{\{i\}}\right) = trace\left(E^T R^{(\{i\})}E\right)\\ &= trace\left(E^T E_{:i}\left(E_{:i}^T E_{:i}\right)^{-1}E_{:i}^T E\right)\\ &= \frac{1}{E_{:i}^T E_{:i}} trace\left(E^T E_{:i}E_{:i}^T E\right) = \frac{\left\|E^T E_{:i}\right\|^2}{E_{:i}^T E_{:i}}.\end{aligned}$$

This defines the following equivalent problem.

Problem 3.3 (Greedy Column Subset Selection) *At iteration t, find column p such that,*

$$p = \arg\max_i \quad \frac{\left\|E^T E_{:i}\right\|^2}{E_{:i}^T E_{:i}} \tag{3.12}$$

where $E = A - \tilde{A}_{\mathcal{S}}$, and $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

The computational complexity of this selection criterion is $O\left(n^2 m\right)$ per iteration, and it requires $O\left(nm\right)$ memory to store the residual of the whole matrix, E, after each iteration. In the rest of this section, two novel techniques are proposed to reduce the memory and time requirements of this selection criterion.

3.3.1 Memory-Efficient Criterion

This section proposes a memory-efficient algorithm to calculate the column subset selection criterion without explicitly calculating and storing the residual matrix E at each iteration. The algorithm is based on a recursive formula for calculating the residual matrix E.

Let $\mathcal{S}^{(t)}$ denote the set of columns selected during the first $t-1$ iterations, $E^{(t)}$ denote the residual matrix at the start of the t-th iteration (i.e., $E^{(t)} = A - \tilde{A}_{\mathcal{S}^{(t)}}$), and $p^{(t)}$ be the column selected at iteration t. The following lemma gives a recursive formula for residual matrix at the start of iteration $t+1$, $E^{(t+1)}$.

Lemma 3.4 *$E^{(t+1)}$ can be calculated recursively as:*

$$E^{(t+1)} = \left(E - \frac{E_{:p}E_{:p}^T}{E_{:p}^T E_{:p}}E\right)^{(t)}. \tag{3.13}$$

Proof Using Corollary 3.2, $\tilde{A}_{\mathcal{S}\cup\{p\}} = \tilde{A}_{\mathcal{S}} + \tilde{E}_{\{p\}}$. Subtracting both sides from A, and substituting $A - \tilde{A}_{\mathcal{S}\cup\{p\}}$ and $A - \tilde{A}_{\mathcal{S}}$ with $E^{(t+1)}$ and $E^{(t)}$ respectively gives:

$$E^{(t+1)} = \left(E - \tilde{E}_{\{p\}}\right)^{(t)}$$

Using Equations (3.1) and (3.2), $\tilde{E}_{\{p\}}$ can be expressed as $\left(E_{:p}(E_{:p}^T E_{:p})^{-1} E_{:p}^T\right) E$. Substituting $\tilde{E}_{\{p\}}$ with this formula in the above equation proves the lemma. ■

Let G be an $n \times n$ matrix which represents the inner-products over the columns of the residual matrix E: $G = E^T E$. The following corollary is a direct result of Lemma 3.4.

Corollary 3.5 *$G^{(t+1)}$ can be calculated recursively as:*

$$G^{(t+1)} = \left(G - \frac{G_{:p} G_{:p}^T}{G_{pp}}\right)^{(t)}.$$

Proof This corollary can be proved by substituting with $E^{(t+1)^T}$ (Lemma 3.4) in $G^{(t+1)} = E^{(t+1)^T} E^{(t+1)}$, and using the fact that $R^{(\{p\})} R^{(\{p\})} = R^{(\{p\})}$ (an idempotent matrix). ■

To simplify the derivation of the memory-efficient algorithm, at iteration t, define $\boldsymbol{\delta} = G_{:p}$ and $\boldsymbol{\omega} = G_{:p}/\sqrt{G_{pp}} = \boldsymbol{\delta}/\sqrt{\boldsymbol{\delta}_p}$. This means that $G^{(t+1)}$ can be calculated in terms of $G^{(t)}$ and $\boldsymbol{\omega}^{(t)}$ as follows:

$$G^{(t+1)} = \left(G - \boldsymbol{\omega}\boldsymbol{\omega}^T\right)^{(t)}, \tag{3.14}$$

or in terms of A and previous $\boldsymbol{\omega}$'s as:

$$G^{(t+1)} = A^T A - \sum_{r=1}^{t} \left(\boldsymbol{\omega}\boldsymbol{\omega}^T\right)^{(r)}. \tag{3.15}$$

$\boldsymbol{\delta}^{(t)}$ and $\boldsymbol{\omega}^{(t)}$ can be calculated in terms of A and previous $\boldsymbol{\omega}$'s as follows:

$$\begin{aligned} \boldsymbol{\delta}^{(t)} &= A^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\omega}^{(r)}, \\ \boldsymbol{\omega}^{(t)} &= \boldsymbol{\delta}^{(t)}/\sqrt{\boldsymbol{\delta}_p^{(t)}}. \end{aligned} \tag{3.16}$$

The column subset selection criterion can be expressed in terms of G as:

$$p = \arg\max_i \quad \frac{\|G_{:i}\|^2}{G_{ii}}$$

The following theorem gives recursive formulas for calculating the column subset selection criterion without explicitly calculating E or G.

Theorem 3.6 *Let $\boldsymbol{f}_i = \|G_{:i}\|^2$ and $\boldsymbol{g}_i = G_{ii}$ be the numerator and denominator of the criterion function for column i respectively, $\boldsymbol{f} = [\boldsymbol{f}_i]_{i=1..n}$, and $\boldsymbol{g} = [\boldsymbol{g}_i]_{i=1..n}$. Then,*

$$\begin{aligned}
\boldsymbol{f}^{(t)} &= \left(\boldsymbol{f} - 2\left(\boldsymbol{\omega} \circ \left(A^T A\boldsymbol{\omega} - \Sigma_{r=1}^{t-2}\left(\boldsymbol{\omega}^{(r)^T}\boldsymbol{\omega}\right)\boldsymbol{\omega}^{(r)}\right)\right) + \|\boldsymbol{\omega}\|^2(\boldsymbol{\omega}\circ\boldsymbol{\omega})\right)^{(t-1)}, \\
\boldsymbol{g}^{(t)} &= \left(\boldsymbol{g} - (\boldsymbol{\omega}\circ\boldsymbol{\omega})\right)^{(t-1)}.
\end{aligned}$$

where $\circ$ represents the Hadamard product operator.

Proof Based on Equation (3.14), $\boldsymbol{f}_i^{(t)}$ can be calculated as:

$$\begin{aligned}
\boldsymbol{f}_i^{(t)} &= \left(\|G_{:i}\|^2\right)^{(t)} = \left(\|G_{:i} - \boldsymbol{\omega}_i\boldsymbol{\omega}\|^2\right)^{(t-1)} \\
&= \left((G_{:i} - \boldsymbol{\omega}_i\boldsymbol{\omega})^T(G_{:i} - \boldsymbol{\omega}_i\boldsymbol{\omega})\right)^{(t-1)} \\
&= \left(G_{:i}^T G_{:i} - 2\boldsymbol{\omega}_i G_{:i}^T\boldsymbol{\omega} + \boldsymbol{\omega}_i^2\|\boldsymbol{\omega}\|^2\right)^{(t-1)} \\
&= \left(\boldsymbol{f}_i - 2\boldsymbol{\omega}_i G_{:i}^T\boldsymbol{\omega} + \boldsymbol{\omega}_i^2\|\boldsymbol{\omega}\|^2\right)^{(t-1)}.
\end{aligned} \tag{3.17}$$

Similarly, $\boldsymbol{g}_i^{(t)}$ can be calculated as:

$$\begin{aligned}
\boldsymbol{g}_i^{(t)} &= G_{ii}^{(t)} = \left(G_{ii} - \boldsymbol{\omega}_i^2\right)^{(t-1)} \\
&= \left(\boldsymbol{g}_i - \boldsymbol{\omega}_i^2\right)^{(t-1)}.
\end{aligned} \tag{3.18}$$

Let $\boldsymbol{f} = [\boldsymbol{f}_i]_{i=1..n}$and $\boldsymbol{g} = [\boldsymbol{g}_i]_{i=1..n}$, $\boldsymbol{f}^{(t)}$ and $\boldsymbol{g}^{(t)}$ can be expressed as:

$$\begin{aligned}
\boldsymbol{f}^{(t)} &= \left(\boldsymbol{f} - 2(\boldsymbol{\omega}\circ G\boldsymbol{\omega}) + \|\boldsymbol{\omega}\|^2(\boldsymbol{\omega}\circ\boldsymbol{\omega})\right)^{(t-1)}, \\
\boldsymbol{g}^{(t)} &= \left(\boldsymbol{g} - (\boldsymbol{\omega}\circ\boldsymbol{\omega})\right)^{(t-1)},
\end{aligned} \tag{3.19}$$

where $\circ$ represents the Hadamard product operator, and $\|.\|$ is the ℓ_2 norm.

Based on the recursive formula of G (Eq. 3.15), the term $G\boldsymbol{\omega}$ at iteration $(t-1)$ can be expressed as:

$$\begin{aligned}
G\boldsymbol{\omega} &= \left(A^T A - \Sigma_{r=1}^{t-2}\left(\boldsymbol{\omega}\boldsymbol{\omega}^T\right)^{(r)}\right)\boldsymbol{\omega} \\
&= A^T A\boldsymbol{\omega} - \Sigma_{r=1}^{t-2}\left(\boldsymbol{\omega}^{(r)^T}\boldsymbol{\omega}\right)\boldsymbol{\omega}^{(r)}
\end{aligned} \tag{3.20}$$

Substitute with $G\boldsymbol{\omega}$ in Equation (3.19) gives the update formulas for $\boldsymbol{f}$ and $\boldsymbol{g}$. ■

This means that the greedy criterion can be memory-efficient by only maintaining two score variables for each column, $\boldsymbol{f}_i$ and $\boldsymbol{g}_i$, and updating them at each iteration based on their previous values and the columns selected so far.

In addition to the selection of representative columns, the memory-efficient procedure directly learns an l-dimensional embedding of all columns into the subspace of selected ones. This can be illustrated as follows. As Equation (3.13) projects the columns of the residual matrix at iteration t onto the subspace orthogonal to $E_{:p}$, the column selected at iteration $t+1$ will be orthogonal to the column selected at iteration t and accordingly the columns selected during the first $t-1$ iterations. This means that the proposed procedure implicitly performs the Gram-Schmidt orthogonalization process. Accordingly, the matrix Q whose columns represent an orthonormal basis for the subspace of selected columns can be expressed as:

$$Q = \begin{bmatrix} \left(\frac{1}{\|E_{:p}\|}E_{:p}\right)^{(1)} & \left(\frac{1}{\|E_{:p}\|}E_{:p}\right)^{(2)} & \dots & \left(\frac{1}{\|E_{:p}\|}E_{:p}\right)^{(l)} \end{bmatrix}.$$

Using Equation (3.3), the embedding of columns into the subspace of Q is given by

$$W = Q^T A = \begin{bmatrix} \left(\frac{1}{\|E_{:p}\|}E_{:p}^T A\right)^{(1)} & \left(\frac{1}{\|E_{:p}\|}E_{:p}^T A\right)^{(2)} & \dots & \left(\frac{1}{\|E_{:p}\|}E_{:p}^T A\right)^{(l)} \end{bmatrix}^T.$$

Since at iteration t, $E_{:p}^T A = E_{:p}^T E$ (see the proof of Corollary 3.2) and $E_{:p}^T E/\|E_{:p}\| = G_{:p}/G_{pp} = \omega$, then

$$\begin{aligned} W = Q^T A &= \begin{bmatrix} \left(\frac{1}{\|E_{:p}\|}E_{:p}^T E\right)^{(1)} & \left(\frac{1}{\|E_{:p}\|}E_{:p}^T E\right)^{(2)} & \dots & \left(\frac{1}{\|E_{:p}\|}E_{:p}^T E\right)^{(l)} \end{bmatrix}^T \\ &= \begin{bmatrix} \boldsymbol{\omega}^{(1)} & \boldsymbol{\omega}^{(2)} & \dots & \boldsymbol{\omega}^{(l)} \end{bmatrix}^T. \end{aligned}$$

Although the proposed procedure does not directly calculate the orthogonal matrix Q, the calculated $\boldsymbol{\omega}$'s can be used to construct the embedding matrix W with no extra computational cost. Given W, the singular values and right singular vectors of A can be approximated using those of W (Eq. 3.5). However, if the goal is to calculate a low-rank approximation of A or approximate its left singular vectors, additional steps will be required to calculate Q. Algorithm 4 shows the complete memory-efficient greedy algorithm.

Algorithm 4 Greedy Column Subset Selection

Input: Data matrix A, Number of columns l
Output: Selected columns $\mathcal{S}$, Low-dimensional embedding W
Steps:

1. Initialize $\mathcal{S} = \{\ \}$
2. Initialize $\boldsymbol{f}_i^{(0)} = \|A^T A_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = A_{:i}^T A_{:i}$ for $i = 1,\ 2,\ \ldots\ n$
3. Repeat $t = 1 \rightarrow l$:
   (a) $p = \arg\max_i \ \boldsymbol{f}_i^{(t)} / \boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{p\}$
   (b) $\boldsymbol{\delta}^{(t)} = A^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\omega}^{(r)}$
   (c) $\boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)} / \sqrt{\boldsymbol{\delta}_p^{(t)}}$
   (d) Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s

$$\boldsymbol{f}^{(t)} = \left(\boldsymbol{f} - 2 \left(\boldsymbol{\omega} \circ \left(A^T A \boldsymbol{\omega} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{\omega}^{(r)^T} \boldsymbol{\omega} \right) \boldsymbol{\omega}^{(r)} \right) \right) + \|\boldsymbol{\omega}\|^2 \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$
$$\boldsymbol{g}^{(t)} = \left(\boldsymbol{g} - \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$

   where $\circ$ represents the Hadamard product operator.

4. $W = \left[\ \boldsymbol{\omega}^{(1)} \quad \ldots \quad \boldsymbol{\omega}^{(l)}\ \right]^T$

3.3.2 Partition-Based Criterion

The column subset selection criterion calculates, at each iteration, the inner-products between each candidate column $E_{:i}$ and other columns E. The computational complexity of these inner-products is $O(nm)$ per candidate column (or $O(n^2m)$ per iteration). When the memory-efficient update formulas are used, the computational complexity is reduced to $O(nm)$ per iteration (that of calculating $A^TA\boldsymbol{\omega}$). However, the computational complexity of calculating the initial value of $\boldsymbol{f}$ is still $O(n^2m)$.

In order to reduce this computational complexity, a novel partition-based criterion is proposed, which reduces the number of inner-products to be calculated at each iteration. The criterion partitions columns into $c \ll n$ random groups, and selects the column which best represents the centroids of these groups. Let $\mathcal{P}_j$ be the set of column that belong to the j-th partition, $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ...\mathcal{P}_c\}$ be a random partitioning of columns into c groups, and B be an $m \times c$ matrix whose element j-th column is the sum of column vectors that belong to the j-th group: $B_{:j} = \sum_{r \in \mathcal{P}_j} A_{:r}$. The use of the sum function (instead of mean) weights each column of B with the size of the corresponding group. This avoids any bias towards larger groups when calculating the sum of inner-products.

The selection criterion can be written as:

Problem 3.4 (Partition-Based Greedy Column Selection) *At iteration t, find column p such that,*

$$p = \arg\max_i \quad \frac{\left\|F^T E_{:i}\right\|^2}{E_{:i}^T E_{:i}} \tag{3.21}$$

where $E = A - \tilde{A}_{\mathcal{S}}$, $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations, $F_{:j} = \sum_{r \in \mathcal{P}_j} E_{:r}$, and $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ...\mathcal{P}_c\}$ is a random partitioning of columns into c groups.

Similar to E (Lemma 3.4), F can be calculated in a recursive manner as follows:

$$F^{(t+1)} = (F - \frac{E_{:p}E_{:p}^T}{E_{:p}^T E_{:p}} F)^{(t)}.$$

This means that random partitioning can be done once at the start of the algorithm. After that, F is initialized to B and then updated recursively using the above formula. The computational complexity of calculating B is $O(nm)$ if the data matrix is full. However, this complexity can be considerably reduced if the data matrix is very sparse.

Further, a memory-efficient variant of the partition-based algorithm can be developed as follows. Let H be an $c \times n$ matrix whose element H_{ji} is the inner-product of the centroid of the j-th group and the i-th column, weighted with the size of the j-th group: $H = F^T E$. Similarly, H can be calculated recursively as follows:

$$H^{(t+1)} = (H - \frac{H_{:p} G_{:p}^T}{G_{pp}})^{(t)}.$$

Define $\boldsymbol{\gamma} = H_{:p}$ and $\boldsymbol{\upsilon} = H_{:p}/\sqrt{G_{pp}} = \boldsymbol{\gamma}/\sqrt{\boldsymbol{\delta}_p}$. $H^{(t+1)}$ can be calculated in terms of $H^{(t)}$, $\boldsymbol{\upsilon}^{(t)}$ and $\boldsymbol{\omega}^{(t)}$ as follows:

$$H^{(t+1)} = (H - \boldsymbol{\upsilon}\boldsymbol{\omega}^T)^{(t)}, \tag{3.22}$$

or in terms of A and previous $\boldsymbol{\omega}$'s and $\boldsymbol{\upsilon}$'s as:

$$H^{(t+1)} = B^T A - \sum_{r=1}^{t} (\boldsymbol{\upsilon}\boldsymbol{\omega}^T)^{(r)}. \tag{3.23}$$

$\boldsymbol{\gamma}^{(t)}$ and $\boldsymbol{\upsilon}^{(t)}$ can be calculated in terms of A, B and previous $\boldsymbol{\omega}$'s and $\boldsymbol{\upsilon}$'s as follows:

$$\boldsymbol{\gamma}^{(t)} = B^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\upsilon}^{(r)},$$
$$\boldsymbol{\upsilon}^{(t)} = \boldsymbol{\gamma}^{(t)}/\sqrt{\boldsymbol{\delta}_p^{(t)}}.$$

The partition-based selection criterion can be expressed in terms of H and G as:

$$p = \arg\max_i \quad \frac{\|H_{:i}\|^2}{G_{ii}}$$

Similar to Theorem 3.6, the following theorem derives recursive formulas for the partition-based criterion function.

Theorem 3.7 *Let $\boldsymbol{f}_i = \|H_{:i}\|^2$ and $\boldsymbol{g}_i = G_{ii}$ be the numerator and denominator of the partition-based criterion function for column i respectively, $\boldsymbol{f} = [\boldsymbol{f}_i]_{i=1..n}$, and $\boldsymbol{g} = [\boldsymbol{g}_i]_{i=1..n}$. Then,*

$$\boldsymbol{f}^{(t)} = \left(\boldsymbol{f} - 2\left(\boldsymbol{\omega} \circ \left(A^T B \boldsymbol{\upsilon} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{\upsilon}^{(r)T} \boldsymbol{\upsilon}\right) \boldsymbol{\omega}^{(r)}\right)\right) + \|\boldsymbol{\upsilon}\|^2 (\boldsymbol{\omega} \circ \boldsymbol{\omega})\right)^{(t-1)},$$
$$\boldsymbol{g}^{(t)} = \left(\boldsymbol{g} - (\boldsymbol{\omega} \circ \boldsymbol{\omega})\right)^{(t-1)},$$

where $\circ$ represents the Hadamard product operator.

Proof The proof is similar to that of Theorem 3.6. It can be easily derived by using the recursive formula for $H_{:i}$ instead of that for $G_{:i}$. ■

In these update formulas, A^TB can be calculated once and then used in different iterations. This makes the computational complexity of the new update formulas is $O(nc)$ per iteration. Algorithm 5 shows the complete partition-based greedy algorithm. The computational complexity of the algorithm is dominated by that of calculating $A^TA_{:p}$ in Step (b) which is of $O(mn)$ per iteration. The other complex step is that of calculating the initial $\boldsymbol{f}$, which is $O(mnc)$. However, these steps can be implemented in an efficient way if the data matrix is sparse. The total computational complexity of the algorithm is $O(\max(mnl, mnc))$, where l is the number of columns and c is the number of random partitions.

3.4 Related Work

Different approaches have been proposed for selecting a subset of representative columns from a data matrix. These approaches can be categorized into randomized, deterministic and hybrid.

3.4.1 Randomized Methods

The randomized methods sample a subset of columns from the original matrix using carefully chosen sampling probabilities. The main focus of this category of methods is to develop fast algorithms for column subset selection and then derive a bound for the reconstruction error of the data matrix based on the selected columns relative to the best possible reconstruction error obtained using Singular Value Decomposition (SVD).

Frieze et al. [66] was the first to suggest the idea of randomly sampling l columns from a matrix and using these columns to calculate a rank-k approximation of the matrix (where $l \geq k$). The authors derived an additive bound for the reconstruction error of the data matrix. This work of Frieze et al. was followed by different papers [67, 68] that enhanced the algorithm by proposing different sampling probabilities and deriving better error bounds for the reconstruction error. Drineas et al. [69] proposed a subspace sampling method which samples columns using probabilities proportional to the norms of the rows of the top k right singular vectors of A. The subspace sampling method allows the development of a relative-error bound (i.e., a multiplicative error bound relative to the best rank-k approximation). However, the subspace sampling depends on calculating

Algorithm 5 Partition-based Greedy Column Selection

Input: Data matrix A, Number of columns l
Output: Selected columns $\mathcal{S}$, Low-dimensional embedding W
Steps:

1. Initialize $\mathcal{S} = \{\ \}$, Generate a random partitioning $\mathbf{P}$, Calculate B: $B_{:j} = \sum_{r \in \mathcal{P}_j} A_{:r}$
2. Initialize $\boldsymbol{f}_i^{(0)} = \|B^T A_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = A_{:i}^T A_{:i}$ for $i = 1,\ 2,\ \ldots n$
3. Repeat $t = 1 \rightarrow l$:
   (a) $p = \arg\max_i \ \boldsymbol{f}_i^{(t)} / \boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{p\}$
   (b) $\boldsymbol{\delta}^{(t)} = A^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\omega}^{(r)}$
   (c) $\boldsymbol{\gamma}^{(t)} = B^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{v}^{(r)}$
   (d) $\boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)} / \sqrt{\boldsymbol{\delta}_p^{(t)}}$, $\boldsymbol{v}^{(t)} = \boldsymbol{\gamma}^{(t)} / \sqrt{\boldsymbol{\delta}_p^{(t)}}$
   (e) Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s

$$\boldsymbol{f}^{(t)} = \left(\boldsymbol{f} - 2 \left(\boldsymbol{\omega} \circ \left(A^T B \boldsymbol{v} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{v}^{(r)T} \boldsymbol{v} \right) \boldsymbol{\omega}^{(r)} \right) \right) + \|\boldsymbol{v}\|^2 \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$
$$\boldsymbol{g}^{(t)} = \left(\boldsymbol{g} - \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$

   where $\circ$ represents the Hadamard product operator.

4. $W = \left[\ \boldsymbol{\omega}^{(1)} \quad \ldots \quad \boldsymbol{\omega}^{(l)}\ \right]^T$

the leading singular vectors of a matrix which is computationally very complex for large matrices.

Deshpande et al. [70, 71] proposed an adaptive sampling method which updates the sampling probabilities based on the columns selected so far. This method is computationally very complex, as it depends on calculating the residual of the data matrix after each iteration. In the same paper, Deshpande et al. also proved the existence of a volume sampling algorithm (i.e., sampling a subset of columns based on the volume enclosed by their vectors) which achieves a multiplicative $(l+1)$-approximation. However, the authors did not present a polynomial time algorithm for this volume sampling algorithm.

3.4.2 Deterministic Methods

The deterministic methods employ a deterministic algorithm for selecting columns which minimizes some criterion function. This criterion function usually quantifies the reconstruction error of the data matrix based on the subset of selected columns. The deterministic methods are slower, but more accurate, than the randomized ones.

In the area of numerical linear algebra, the column pivoting method exploited by the QR decomposition [10] permutes the columns of the matrix based on their norms to enhance the numerical stability of the QR decomposition algorithm. The first l columns of the permuted matrix can be directly selected as representative columns. The Rank-Revealing QR (RRQR) decomposition [14–17] is a category of QR decomposition methods which permute columns of the data matrix while imposing additional constraints on the singular values of the two sub-matrices of the upper-triangular matrix R corresponding to the selected and non-selected columns. It has been shown that the constrains on the singular values can be used to derive an theoretical guarantee for the column-based reconstruction error according to spectral norm [63].

Besides methods based on QR decomposition, different recent methods have been proposed for directly selecting a subset of columns from the data matrix. Boutsidis et al. [63] proposed a deterministic column subset selection method which first groups columns into clusters and then selects a subset of columns from each cluster. The authors proposed a general framework in which different clustering and subset selection algorithms can be employed to select a subset of representative columns. Çivril and Magdon-Ismail [72, 73] presented a deterministic algorithm which greedily selects columns from the data matrix that best represent the right leading singular values of the matrix. This algorithm, however accurate, depends on the calculation of the leading singular vectors of a matrix, which is computationally very complex for large matrices.

Recently, Boutsidis et al. [65] presented a column subset selection algorithm which first calculates the top-k right singular values of the data matrix (where k is the target rank) and then uses deterministic sparsification methods to select $l \geq k$ columns from the data matrix. The authors derived a theoretically near-optimal error bound for the rank-k column-based approximation. Deshpande and Rademacher [75] presented a polynomial-time deterministic algorithm for volume sampling with a theoretical guarantee for $l = k$. Quite recently, Guruswami and Sinop [76] presented a deterministic algorithm for volume sampling with theoretical guarantee for $l > k$. The deterministic volume sampling algorithms are, however, more complex than the algorithms presented in this chapter, and they are infeasible for large data sets.

3.4.3 Hybrid Methods

A third category of CSS techniques is the hybrid methods which combine the benefits of both the randomized and deterministic methods. In these methods, a large subset of columns is randomly sampled from the columns of the data matrix and then a deterministic step is employed to reduce the number of selected columns to the desired rank.

For instance, Boutsidis et al. [64] proposed a two-stage hybrid algorithm for column subset selection which runs in $O\left(\min\left(n^2m, nm^2\right)\right)$. In the first stage, the algorithm samples $c = O\left(l \log l\right)$ columns based on probabilities calculated using the l-leading right singular vectors. In the second phase, a Rank-revealing QR (RRQR) algorithm is employed to select exactly l columns from the columns sampled in the first stage. The authors suggested repeating the selection process 40 times in order to provably reduce the failure probability. The authors proved a good theoretical guarantee for the algorithm in terms of spectral and Frobenius term. However, the algorithm depends on calculating the leading l right singular vectors which is computationally complex for large data sets.

3.4.4 Comparison to Related Work

The greedy column subset selection algorithm presented in Section 3.3.1 belongs to the category of deterministic algorithms. In comparison to QR-based methods, the greedy CSS algorithm can be implicitly used to calculate a Q-less incomplete QR factorization of the data matrix A:

$$A = QW,\ A\Pi = QW\Pi = QR$$

where Π is a permutation matrix which sorts the first l columns according to their selection order. The permutation of the columns of the embedding matrix W produces an upper

Table 3.1: The properties of the data sets used to evaluate different CSS methods.

Data set	Type	# Instances	# Features
Reuters-21578	Documents	5946	18933
Reviews	Documents	4069	36746
LA1	Documents	3204	29714
MNIST-4K	Digit Images	4000	784
PIE-20	Face Images	3400	1024
YaleB-38	Face Images	2414	1024

triangular matrix.

The greedy CSS algorithm differs from the greedy algorithm proposed by Çivril and Magdon-Ismail [72, 73] in that the latter depends on first calculating the Singular Value Decomposition of the data matrix, which is computationally complex, especially for large matrices. The proposed algorithm is also more efficient than the recently proposed volume sampling algorithms [75, 76].

On the other hand, the partition-based greedy algorithm presented in Section 3.3.2 belongs to the category of hybrid algorithms in the sense that it combines a randomized step with a deterministic selection. However, the randomized step in the partition-based algorithm depends on grouping columns into random partitions rather than selecting a set random samples; this random partitioning speeds up the algorithm without losing much information about the span of the columns.

3.5 Experiments and Results

Experiments have been conducted on six benchmark data sets, whose properties are summarized in Table 3.1.[2] The *Reuters-21578* is the training set of the Reuters-21578 collection [77]. The *Reviews* and *LA1* are document data sets from TREC collections.[3] The pre-processed versions of *Reviews* and *LA1* that are distributed with the CLUTO Toolkit [78] were used. The *MNIST-4K* is a subset of the MNIST data set of handwritten

[2] The data sets *Reuters-21578*, *MNIST-4K*, *PIE-20* and *YaleB-38* are available in MAT format at: http://www.cad.zju.edu.cn/home/dengcai/Data/data.html. *PIE-20* is a subset of *PIE-32x32* with the images of the first 20 persons.

[3] http://trec.nist.gov

digits.[4] The *PIE-20* and *YaleB-38* are pre-processed subsets of the CMU PIE [79] and Extended Yale Face [80] data sets respectively. The *PIE-20* and *YaleB-38* data sets have been used by He et al. [81] to evaluate different face recognition algorithms.

The following CSS methods are compared[5].

- **UniNoRep**: is uniform sampling of columns without replacement.
- **qr**: is the QR decomposition with column pivoting [10] implemented by the MATLAB *qr* function.[6]
- **SRRQR**: is the strong rank-revealing QR decomposition [15]. Algorithm 4 of [15] was implemented in MATLAB. In this implementation, the MATLAB *qr* function is first used to calculate the QR decomposition with column pivoting and then the columns are swapped using the criterion specified by Gu and Eisenstat [15].[7]
- **ApproxSVD**: is the sparse approximation of Singular Value Decomposition (SVD) [72, 73]. The algorithm was implemented in MATLAB. The generalized CSS algorithm presented in Chapter 7 is used to select columns that best approximates the leading singular vectors. The use of the generalized CSS algorithm is equivalent to, but more efficient than, the algorithm proposed by Çivril and Magdon-Ismail [72,73]. Since the calculation of exact SVD is computationally complex, the Stochastic SVD algorithm [12] is used to approximate the leading singular values and vectors of the data matrix. This significantly reduces the run time of the original algorithm proposed by Çivril and Magdon-Ismail while achieving comparable accuracy.
- **HybridCSS**: is the hybrid column subset selection algorithm proposed by Bousidis et al. [64]. The number of selected columns in the randomized phase is set to $l \log(l)$. The algorithm was implemented in MATLAB. In the randomized phase, the Stochastic SVD is first used to calculate the leading singular vectors, and the approximated singular vectors are then used to calculate the sampling probabilities. In the deterministic phase, the MATLAB *qr* function is used to select columns.[8]

[4] http://yann.lecun.com/exdb/mnist

[5] The CSS algorithm of Boutsidis et al. [65] was not included in the comparison as its implementation is not available.

[6] Revision: 5.13.4.7

[7] In the implemented code, the efficient recursive formulas in Section 4 of [15] are used to implement the update of QR decomposition and the swapping criterion.

[8] In [82] (a newer version of [64]), Boutsidis et al. suggested the use of the SRRQR algorithm [15, Algorithm 4] for the deterministic phase. Although the SRRQR algorithm achieves the theoretical guarantee

- **GreedyCSS**: is the greedy column subset selection method described in Algorithm 4.

- **PartGreedyCSS**: is the partition-based greedy column subset selection method described in Algorithm 5. For all experiments, the number of partitions is set to 100.

The different CSS methods are evaluated according to their ability to minimize the reconstruction error of the data matrix based on the subset of selected columns (Definition 3.1). In order to quantify the reconstruction error across different data sets, a relative accuracy measure is defined as

$$\text{Relative Accuracy} = \frac{\|A - \tilde{A}_l\|_F}{\|A - \tilde{A}_{\mathcal{S}}\|_F}, \tag{3.24}$$

where $\tilde{A}_{\mathcal{S}}$ is the rank-l approximation of the data matrix calculated based on the subset $\mathcal{S}$ of columns and $\tilde{A}_l$ is the best rank-l approximation of the data matrix calculated using the Singular Value Decomposition (SVD). Since $\|A - \tilde{A}_l\|_F$ is the minimum possible reconstruction error for a rank-l approximation, the relative accuracy is between 0 and 1 with higher values indicating better CSS methods.

For all the data sets, the percentage of selected columns l/n is changed from 1% to 25% with increments of 2% and the relative accuracies and run times are measured.[9] Experiments with randomness were repeated ten times, and the average and standard deviation of measures were calculated.

Figures 3.1 and 3.2 show the relative accuracy measures and run times for different CSS methods on the six benchmark data sets.[10] In addition, Tables 3.2 and 3.3 show the relative accuracy measures for the best performing CSS methods (**ApproxSVD**, **HybridCSS**, **GreedyCSS**, and **PartGreedyCSS**). Each sub-table represents a data set and each column represents a percentage of selected columns. The relative accuracy measures in each sub-column are divided into groups according to their statistical significance. The best group of methods is highlighted in bold, and the second best group is underlined.

presented in [64], the MATLAB *qr* function is used in the conducted experiments as it is much faster and it achieves comparable accuracy for the experimented data sets.

[9] For the MNIST4K data set, the range of l/n values is smaller since the rank of the matrix is very low (i.e., less than the number of pixels).

[10] The **qr** and **SRRQR** methods both depend on the MATLAB qr function. For the document data sets, the MATLAB qr function takes very long times compared to other methods and accordingly they are not reported in the shown figures.

The tests of statistical significance were performed as follows: the methods in each sub-column are first sorted in a descending order according to their average accuracy measures, then a one-tailed t-test is used to assess the significance of each method with respect to its successor. The t-test uses the null-hypothesis that two methods are equivalent, and the alternative hypothesis that the method is superior to its successor. For each pair of methods, the t-statistic is calculated as:

$$t = \frac{\overline{q_1} - \overline{q_2}}{\sqrt{\frac{s_1^2}{r_1} + \frac{s_2^2}{r_2}}},$$

where $\overline{q_1}$ and $\overline{q_2}$ are the average accuracy measures for the two methods, s_1 and s_2 are the standard deviations of the accuracy measures, and r_1 and r_2 are the number of runs used to estimate $\overline{q_1}$ and $\overline{q_2}$ respectively. The value of t-statistic is then compared to the critical value $t_{critical}$ obtained from the t-distribution table for a 95% confidence interval. If $t > t_{critical}$, the null-hypothesis is rejected and the method is considered superior to its successor.

It can be observed from the figures and tables that for all data sets, the **GreedyCSS** method significantly outperforms the **UniNoRep**, **qr**, **SRRQR**, and **HybridCSS** methods in terms of relative accuracy, and it shows comparable accuracy to the **ApproxSVD** method. In terms of run times, for most of the data sets, the **GreedyCSS** scales better than the **HybridCSS** and **ApproxSVD** methods.

On the other hand, the **PartGreedyCSS** outperforms the **UniNoRep**, **qr**, and **SRRQR** methods in terms of relative accuracy, and shows comparable accuracy to the **HybridCSS** method. In terms of run times, the **PartGreedyCSS** is much more efficient than the **HybridCSS** method and other methods for all data sets. It should also be noted that the **SRRQR** method achieves comparable accuracy to the **qr** method and both methods demonstrate lower approximation accuracies than other deterministic and hybrid methods.

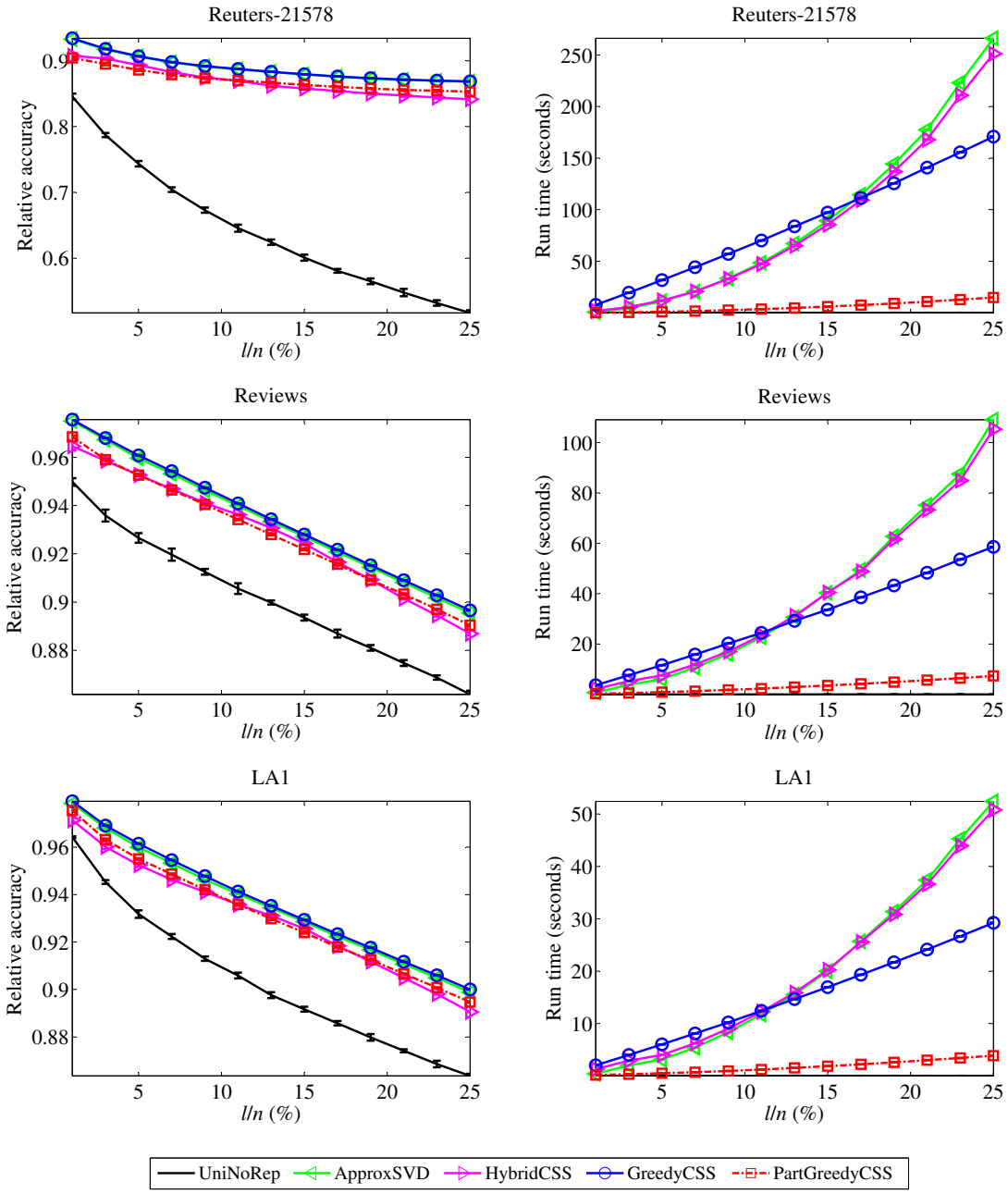

Figure 3.1: The relative accuracy measures and run times of different column-based low-rank approximations $\tilde{A}_{\mathcal{S}}$ for the *Reuters-21578*, *Reviews* and *LA1* data sets.

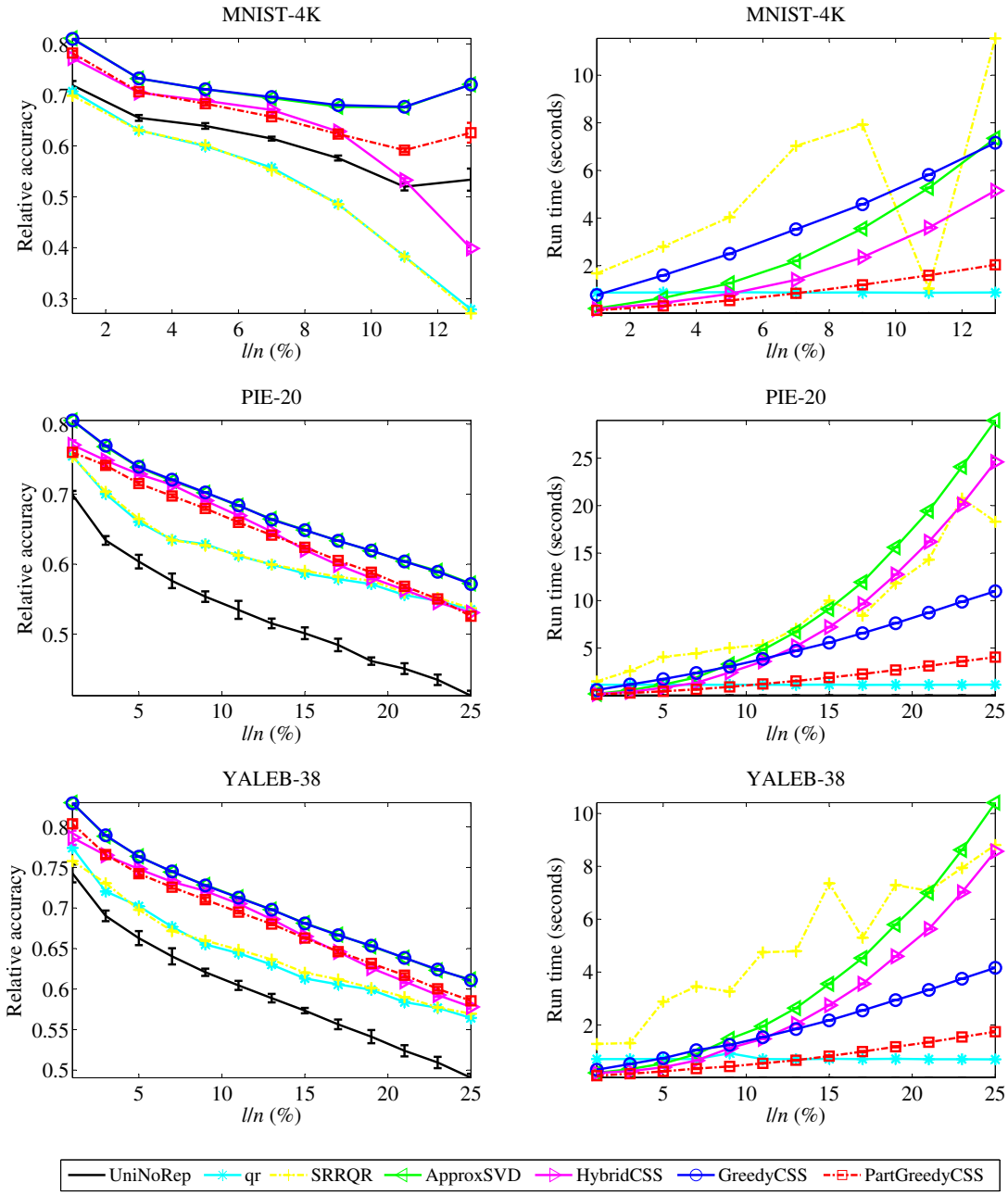

Figure 3.2: The relative accuracy measures and run times of different column-based low-rank approximations $\tilde{A}_{\mathcal{S}}$ for the *MNIST-4K*, *PIE-20* and *YaleB-38* data sets.

Table 3.2: The relative accuracy measures of the best performing CSS methods for the **Reuters-21578**, **Reviews**, and **LA1** data sets. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

Reuters-21578				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
ApproxSVD	0.9064±0.0000	0.8913±0.0000	0.8830±0.0000	0.8759±0.0000
HybridCSS	0.8935±0.0013	0.8748±0.0011	0.8619±0.0007	0.8540±0.0014
GreedyCSS	**0.9069±0.0000**	**0.8920±0.0000**	**0.8834±0.0000**	**0.8762±0.0000**
PartGreedyCSS	0.8861±0.0015	0.8736±0.0009	0.8669±0.0011	0.8604±0.0007
Reviews				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
ApproxSVD	0.9596±0.0000	0.9463±0.0000	0.9333±0.0000	0.9207±0.0000
HybridCSS	0.9526±0.0006	0.9411±0.0005	0.9307±0.0004	0.9165±0.0002
GreedyCSS	**0.9608±0.0000**	**0.9474±0.0000**	**0.9343±0.0000**	**0.9217±0.0000**
PartGreedyCSS	0.9526±0.0004	0.9405±0.0003	0.9280±0.0004	0.9156±0.0004
LA1				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
ApproxSVD	0.9599±0.0000	0.9464±0.0000	0.9344±0.0000	0.9223±0.0000
HybridCSS	0.9524±0.0007	0.9411±0.0006	0.9311±0.0003	0.9184±0.0003
GreedyCSS	**0.9614±0.0000**	**0.9478±0.0000**	**0.9352±0.0000**	**0.9233±0.0000**
PartGreedyCSS	0.9551±0.0003	0.9421±0.0004	0.9298±0.0005	0.9178±0.0004

Table 3.3: The relative accuracy measures of the best performing CSS methods for the **MNIST-4K**, **PIE-20**, and **YaleB-38** data sets. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

MNIST-4K				
Method	$l/n = 1\%$	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$
ApproxSVD	**0.8112±0.0000**	0.7111±0.0000	0.6768±0.0000	**0.7206±0.0000**
HybridCSS	0.7723±0.0059	0.6887±0.0027	0.6283±0.0039	0.3986±0.0058
GreedyCSS	0.8099±0.0000	**0.7112±0.0000**	**0.6799±0.0000**	**0.7203±0.0000**
PartGreedyCSS	0.7821±0.0039	0.6827±0.0033	0.6233±0.0041	0.6258±0.0196
PIE-20				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
ApproxSVD	0.7385±0.0000	0.7021±0.0000	**0.6646±0.0000**	**0.6334±0.0000**
HybridCSS	0.7285±0.0019	0.6908±0.0013	0.6463±0.0013	0.5989±0.0013
GreedyCSS	**0.7391±0.0000**	**0.7025±0.0000**	**0.6636±0.0000**	**0.6333±0.0000**
PartGreedyCSS	0.7153±0.0023	0.6797±0.0021	0.6416±0.0013	0.6052±0.0015
YaleB-38				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
ApproxSVD	**0.7638±0.0000**	**0.7285±0.0000**	**0.6981±0.0000**	**0.6672±0.0000**
HybridCSS	0.7481±0.0029	0.7210±0.0016	0.6861±0.0026	0.6458±0.0016
GreedyCSS	0.7634±0.0000	**0.7278±0.0000**	0.6980±0.0000	**0.6665±0.0000**
PartGreedyCSS	0.7423±0.0031	0.7103±0.0029	0.6802±0.0013	0.6461±0.0025

Chapter 4

Greedy Nyström Approximation

This chapter addresses the problem of selecting a set of representative data instances when only information about their pairwise similarities is available. The algorithms presented in this chapter extend the greedy Column Subset Selection (CSS) methods presented in Chapter 3 to work on a kernel matrix that represents the inner-products over the data instances in a high-dimensional feature space. The objective of the kernel CSS algorithms is to select a subset of data instances that minimizes the reconstruction error of the data vectors in the high-dimensional space implicitly defined by the kernel function. The chapter also explores the connection between the low-rank approximations obtained using the CSS methods and a well-known low-rank approximation of kernel matrices, namely the Nyström approximation. The chapter presents a novel recursive formula for calculating the Nyström approximation and uses this formula to derive an effective greedy criterion for kernel CSS.

The chapter is organized as follows: Section 4.1 reviews the basic Nyström method. Section 4.2 defines a criterion for kernel column subset selection and shows the connection between the Nyström approximation and the column-based approximations presented in Chapter 3. Section 4.3 proposes a recursive algorithm for calculating the Nyström approximation. Section 4.4 presents the greedy selection criteria. Related work is then discussed in Section 4.5. Section 4.6 presents an empirical evaluation of the proposed algorithms.

4.1 Nyström Approximation

Kernel methods [83] are widely-used algorithms that work on matrices whose elements represent the inner-products between data points in some vector space. These matrices

are referred to as the kernel matrices. Examples of kernel methods include Support Vector Machines (SVM) [84], Gaussian Processes (GP) [85], and kernel k-means [35, 36]. The use of kernels allows the application of different machine learning algorithms to complex data structures for which it is difficult to represent data instances in a feature space. Examples include strings, trees and graphs [83]. In addition, kernel methods can be used to explore non-linear relationships between data points in the feature space.

The Nyström method [86] is an efficient technique for obtaining a low-rank approximation of a large kernel matrix, using only a subset of its columns. It can also be used to efficiently approximate the singular values and vectors of a kernel matrix [86, 87]. The Nyström method has been successfully used in many large-scale applications including efficient learning of kernel-based models such as Gaussian processes [86] and support vector machines [88], fast multi-dimensional scaling [89], approximate spectral clustering [5], and large-scale manifold learning [90].

The quality of the Nyström approximation is highly dependent on the subset of selected columns. Although uniform sampling has been the most common technique for column selection [86], a considerable amount of research work has been conducted to theoretically and empirically study other sampling techniques. These techniques include: non-uniform sampling, using probabilities calculated based on the kernel matrix [68, 91, 92]; adaptive sampling, in which probabilities are updated based on intermediate approximations of the kernel matrix [87, 93]; and deterministic sampling, where columns are selected such that some criterion function is optimized [21, 94].

4.1.1 The Basic Nyström Method

The Nyström method obtains a low-rank approximation of a kernel matrix using a subset of its columns. Let K be an $n \times n$ symmetric positive semi-definite (SPSD) kernel matrix defined over n data instances. The Nyström method starts by selecting a subset of $l \ll n$ columns of K (usually by random sampling). These columns represent the similarities between the subset of l data instances and all data instances. Let $\mathcal{S}$ be the set of the indices of selected columns, and $\mathcal{R}$ be the set of the indices of remaining columns. Without loss of generality, the columns and rows of K can be arranged as follows:

$$K = \begin{bmatrix} K_{\mathcal{SS}} & K_{\mathcal{SR}} \\ K_{\mathcal{SR}}^T & K_{\mathcal{RR}} \end{bmatrix}, \tag{4.1}$$

where $K_{\mathcal{SS}}$, $K_{\mathcal{SR}}$ and $K_{\mathcal{RR}}$ are sub-matrices of K whose elements are $\{K_{ij} : i, j \in \mathcal{S}\}$, $\{K_{ij} : i \in \mathcal{S}, j \in \mathcal{R}\}$, and $\{K_{ij} : i, j \in \mathcal{R}\}$ respectively, and K_{ij} denotes the element of K at row i and column j.

The Nyström method calculates a rank-l approximation of K as [86]:

$$\tilde{K}_{\mathcal{S}} = K_{:\mathcal{S}} K_{\mathcal{S}\mathcal{S}}^{-1} K_{:\mathcal{S}}^T, \tag{4.2}$$

where $K_{:\mathcal{S}} = \begin{bmatrix} K_{\mathcal{S}\mathcal{S}} & K_{\mathcal{S}\mathcal{R}} \end{bmatrix}^T$ is an $n \times l$ matrix which consists of the selected columns of K.

The Nyström method can also be used to approximate the leading singular values and vectors of K using those of $K_{\mathcal{S}\mathcal{S}}$ [86], which is sometimes referred to as the approximate spectral decomposition [87].The k leading singular values and vectors of K can be approximated as:

$$\tilde{\Sigma}_k = \frac{n}{l}\Lambda_k, \quad \tilde{U}_k = \sqrt{\frac{l}{n}} K_{:\mathcal{S}} V_k \Lambda_k^{-1}. \tag{4.3}$$

where $k \leq l \ll n$. V_k and U_k are $l \times k$ and $n \times k$ matrices whose columns are the k leading singular vectors of $K_{\mathcal{S}\mathcal{S}}$ and K respectively. Λ_k and Σ_k are $k \times k$ matrices whose diagonal elements are the k leading singular values of $K_{\mathcal{S}\mathcal{S}}$ and K respectively.

The approximate singular values and vectors of K can be used to map data points to a k-dimensional space:

$$Y = \tilde{\Sigma}_k^{1/2} \tilde{U}_k^T = \Lambda_k^{-1/2} V_k^T K_{:\mathcal{S}}^T, \tag{4.4}$$

where Y is a $k \times n$ matrix whose columns represent data instances in the k-dimensional space. The kernel matrix over data points in the k-dimensional space represents a rank-k approximation of K which can be calculated as:

$$\tilde{K}_{\mathcal{S},k} = Y^T Y = K_{:\mathcal{S}} V_k \Lambda_k^{-1} V_k^T K_{:\mathcal{S}}^T. \tag{4.5}$$

Throughout the rest of the chapter, "Nyström approximation" and "rank-l Nyström approximation" are used interchangeably to refer to $\tilde{K}_{\mathcal{S}}$, while "rank-k Nyström approximation" refers to $\tilde{K}_{\mathcal{S},k}$.

The computational complexity of calculating $K_{\mathcal{S}\mathcal{S}}^{-1}$ is $O(l^3)$, and those of calculating Y and $\tilde{K}_{\mathcal{S},k}$ are $O(l^3 + nlk)$ and $O(l^3 + nlk + n^2k)$, respectively. It should be noted that the approximate singular vectors, as well as the basis of the k-dimensional space are, however, non-orthonormal [87]. In some applications, additional steps might be required to obtain orthonormal vectors. This, however, increases the computational complexity.

4.1.2 Applications of the Nyström Method

The Nyström method has been successfully used in many large-scale applications in machine learning and data mining. As the Nyström method obtains an approximate spectral

decomposition of a large kernel matrix, it has been used to speed up the training of kernel-based learning models, such as Support Vector Machines (SVM) [84] and Gaussian Processes (GP) [85]. These models are computationally very complex, as their training phase scales up to $O(n^3)$, where n is the number of instances. Williams and Seeger [86] suggested that the low-rank factorization obtained by the Nyström method can be efficiently used to calculate the inverse of the regularized kernel matrix (e.g., $(K + \alpha I)^{-1}$) using the Woodbury formula [10]. This formula significantly reduces the computational complexity of learning Gaussian processes to $O(nk^2)$, where k is the rank of the approximated kernel. The same observation has been made by Fine and Scheinberg [95], who suggested that using a low-rank approximation of the kernel matrix reduces the training time of support vector machines. The authors, however, used a method based on incomplete Cholesky Decomposition to obtain the low-rank approximation.

The Nyström method has also been used for fast dimension reduction, where approximate singular values and vectors are used to map instances to a low-dimensional space (Eq. 4.4). In particular, Platt [89] showed that different large-scale algorithms for multi-dimensional scaling are in essence Nyström approximation. Similarly, Talwalkar et al. [90] used the Nyström method for large-scale manifold learning. Efficient techniques for dimension reduction have many practical applications, such as latent semantic analysis [1], which is useful in analyzing documents based on their semantics; the analysis of DNA microarray data [96]; and face recognition based on eigenfaces [2]. The Nyström method has also been used to speed up spectral clustering by approximating the leading eigenvectors of the graph Laplacian matrix [5].

4.2 Kernel Column Subset Selection

This section defines a criterion for selecting representative data instances when only information about their pairwise similarities is available. These pairwise similarities are encoded by a kernel matrix whose elements represent the inner-products over the data points in a high-dimensional feature space. The selection criterion is defined in terms of the Nyström approximation of the kernel matrix based on the columns corresponding to the selected representatives.

It should be noted that the greedy Column Subset Selection (CSS) methods presented in Chapter 3 can be directly applied to the columns of the kernel matrix. In this case, the greedy CSS methods will minimize the reconstruction error of the kernel matrix based on the selected columns. This is different from the objective of the algorithms presented in this chapter which minimize the reconstruction error of the data vectors in the high-dimensional

feature space implicitly defined by the kernel function. The decision of whether to select columns from a kernel matrix using the CSS methods presented in Chapter 3 or the kernel methods presented in this chapter depends on the definition of good representatives in the application domain and the aspects of similarity captured by the kernel function.

Let $k(i,j)$ be a kernel function that calculates the inner-product between data points i and j. Let $\phi(.)$ be a function that maps a data point to a vector in the high-dimensional feature space implicitly defined by the kernel function. $k(i,j)$ can be expressed as

$$k(i,j) = \phi(i)^T \phi(j) \ .$$

Given a set of n data points, let K be an $n \times n$ matrix whose (i,j)-th element is $k(i,j)$. Let Φ be a matrix whose columns represent the vectors of the data points in the high-dimensional feature space implicitly defined by the kernel.

$$\Phi = [\phi(1), \ \phi(2), \ ..., \ \phi(n)] \ .$$

The kernel matrix K can be expressed as:

$$K = \Phi^T \Phi \ .$$

The objective of the CSS algorithm is to select a subset $\mathcal{S}$ of columns of Φ such that the reconstruction error of Φ based on the selected columns is minimized. Since Φ is implicitly defined by K, the greedy CSS algorithms presented in Chapter 3 cannot be directly applied to Φ. To implicitly select representative columns of Φ, a selection criterion needs to be defined in terms of the kernel matrix K. In order to define this criterion, the following theorem shows the connection between the Nyström approximation and the low-rank approximation of Φ based on the selected representatives.

Theorem 4.1 *Let K be an $n \times n$ kernel matrix, Φ be a matrix whose columns represent the vectors of the data points in the high-dimensional space implicitly defined by the kernel. For any subset of columns $\mathcal{S}$,*

$$\tilde{K}_{\mathcal{S}} = \tilde{\Phi}_{\mathcal{S}}^T \tilde{\Phi}_{\mathcal{S}}$$

where $\tilde{K}_{\mathcal{S}}$ is the Nyström approximation of K based on the subset $\mathcal{S}$ of columns and $\tilde{\Phi}_{\mathcal{S}}$ is the rank-l approximation of Φ calculated by projecting all columns of Φ onto the subspace of $\Phi_{:\mathcal{S}}$.

Proof Given a set $\mathcal{S}$ of columns, the column-based low-rank approximation of Φ based on $\mathcal{S}$ is defined as:

$$\tilde{\Phi}_{\mathcal{S}} = P^{(\mathcal{S})}\Phi,$$

where $P^{(\mathcal{S})}$ is a projection matrix which projects the columns of Φ onto the span of $\Phi_{:\mathcal{S}}$.

The inner-products over the columns of $\tilde{\Phi}_{\mathcal{S}}$ define the following kernel matrix:

$$\tilde{\Phi}_{\mathcal{S}}^T\tilde{\Phi}_{\mathcal{S}} = \Phi^T P^{(\mathcal{S})}P^{(\mathcal{S})}\Phi = \Phi^T P^{(\mathcal{S})}\Phi,$$

as $P^{(\mathcal{S})}P^{(\mathcal{S})} = P^{(\mathcal{S})}$ (an idempotent matrix). Substituting with $P^{(\mathcal{S})} = \Phi_{:\mathcal{S}}\left(\Phi_{:\mathcal{S}}^T\Phi_{:\mathcal{S}}\right)^{-1}\Phi_{:\mathcal{S}}^T$ in $\tilde{\Phi}_{\mathcal{S}}^T\tilde{\Phi}_{\mathcal{S}}$:

$$\tilde{\Phi}_{\mathcal{S}}^T\tilde{\Phi}_{\mathcal{S}} = \Phi^T\Phi_{:\mathcal{S}}\left(\Phi_{:\mathcal{S}}^T\Phi_{:\mathcal{S}}\right)^{-1}\Phi_{:\mathcal{S}}^T\Phi.$$

Using $K_{:\mathcal{S}} = \Phi^T\Phi_{:\mathcal{S}}$, $K_{\mathcal{S}\mathcal{S}} = \Phi_{:\mathcal{S}}^T\Phi_{:\mathcal{S}}$:

$$\tilde{\Phi}_{\mathcal{S}}^T\tilde{\Phi}_{\mathcal{S}} = K_{:\mathcal{S}}K_{\mathcal{S}\mathcal{S}}^{-1}K_{:\mathcal{S}}^T.$$

The right-hand side is the Nyström approximation of K based on the set $\mathcal{S}$ of columns. ■

Based on this theorem, the CSS criterion of Definition 3.1 on page 28 can be expressed in terms of the kernel matrix and its Nyström approximation as follows:

$$\begin{aligned}
F(\mathcal{S}) &= \|\Phi - P^{(\mathcal{S})}\Phi\|_F^2 \\
&= trace\left(\left(\Phi - P^{(\mathcal{S})}\Phi\right)^T\left(\Phi - P^{(\mathcal{S})}\Phi\right)\right) \\
&= trace\left(\Phi^T\Phi - 2\Phi^T P^{(\mathcal{S})}\Phi + \Phi^T P^{(\mathcal{S})}P^{(\mathcal{S})}\Phi\right) \\
&= trace\left(\Phi^T\Phi - \Phi^T P^{(\mathcal{S})}\Phi\right) \\
&= trace\left(K - \tilde{K}_{\mathcal{S}}\right),
\end{aligned}$$

as $P^{(\mathcal{S})}P^{(\mathcal{S})} = P^{(\mathcal{S})}$ and $\|\Phi\|_F^2 = trace\left(\Phi^T\Phi\right)$.

This defines the following kernel-based criterion and the corresponding kernel column subset selection problem.

Definition 4.1 (Kernel Column Subset Selection Criterion) *Let K be an $n \times n$ kernel matrix and $\mathcal{S}$ be the set of the indices of the candidate columns, the kernel column subset selection criterion is defined as:*

$$F(\mathcal{S}) = trace\left(K - \tilde{K}_{\mathcal{S}}\right) \tag{4.6}$$

where $\tilde{K}_{\mathcal{S}}$ is the Nyström approximation of K based on the subset $\mathcal{S}$ of columns.

Problem 4.1 (Kernel Column Subset Selection) *Given an $n \times n$ kernel matrix K and an integer l, find a subset of columns $\mathcal{L}$ such that $|\mathcal{L}| = l$ and*

$$\mathcal{L} = \arg\min_{\mathcal{S}} F(\mathcal{S}),$$

where $F(\mathcal{S})$ is the kernel column subset selection criterion, $\mathcal{S}$ is the set of the indices of the candidate columns, and $\mathcal{L}$ is the set of the indices of the selected columns.

The following sections define recursive formulas for calculating the Nyström approximation as well as the kernel CSS criterion. In addition, a greedy algorithm is presented to efficiently select columns for Nyström approximation.

4.3 Recursive Nyström Approximation

The Nyström approximation of a kernel matrix can be calculated in a recursive manner as shown by the following theorem.

Theorem 4.2 *Given a set of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,*

$$\tilde{K}_{\mathcal{S}} = \tilde{K}_{\mathcal{P}} + \tilde{G}_{\mathcal{R}}$$

where $\tilde{K}_{\mathcal{S}}$, $\tilde{K}_{\mathcal{P}}$ are the Nyström approximations of K based on the subsets $\mathcal{S}$ and $\mathcal{P}$ respectively, $G = K - \tilde{K}_{\mathcal{P}}$ is the residual matrix of K after subtracting $\tilde{K}_{\mathcal{P}}$, and $\tilde{G}_{\mathcal{R}}$ is the Nyström approximation of G based on the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns.

Proof Using Theorem 4.1, $\tilde{K}_{\mathcal{S}} = \tilde{\Phi}_{\mathcal{S}}^T \tilde{\Phi}_{\mathcal{S}}$. Let $E = \Phi - P^{(\mathcal{P})}\Phi = \Phi - \tilde{\Phi}_{\mathcal{P}}$. Using Corollary 3.2 on page 33 and substituting in $\tilde{K}_{\mathcal{S}} = \tilde{\Phi}_{\mathcal{S}}^T \tilde{\Phi}_{\mathcal{S}}$ gives:

$$\begin{aligned} \tilde{K}_{\mathcal{S}} &= \left(\tilde{\Phi}_{\mathcal{P}} + \tilde{E}_{\mathcal{R}}\right)^T \left(\tilde{\Phi}_{\mathcal{P}} + \tilde{E}_{\mathcal{R}}\right) \\ &= \tilde{\Phi}_{\mathcal{P}}^T \tilde{\Phi}_{\mathcal{P}} + \tilde{\Phi}_{\mathcal{P}}^T \tilde{E}_{\mathcal{R}} + \tilde{E}_{\mathcal{R}}^T \tilde{\Phi}_{\mathcal{P}} + \tilde{E}_{\mathcal{R}}^T \tilde{E}_{\mathcal{R}} \end{aligned} \tag{4.7}$$

The first term of the right-hand side is the Nyström approximation of K based on the subset $\mathcal{P}$ of columns: $\tilde{K}_{\mathcal{P}} = \tilde{\Phi}_{\mathcal{P}}^T \tilde{\Phi}_{\mathcal{P}}$ (Theorem 4.1). The second term $\tilde{\Phi}_{\mathcal{P}}^T \tilde{E}_{\mathcal{R}} = \Phi^T P^{(\mathcal{P})} R^{(\mathcal{R})} E$. Using Lemma 3.1 on page 31, the product $P^{(\mathcal{P})} R^{(\mathcal{R})} = P^{(\mathcal{P})} \left(P^{(\mathcal{S})} - P^{(\mathcal{P})}\right) = P^{(\mathcal{P})} P^{(\mathcal{S})} - P^{(\mathcal{P})}$. Since $\mathcal{P} \subset \mathcal{S}$, then $P^{(\mathcal{P})} P^{(\mathcal{S})} = P^{(\mathcal{P})}$ and accordingly $P^{(\mathcal{P})} R^{(\mathcal{R})} = 0$ and

$\tilde{\Phi}_{\mathcal{P}}^T \tilde{E}_{\mathcal{R}} = 0$. Similarly, the third term $\tilde{E}_{\mathcal{R}}^T \tilde{\Phi}_{\mathcal{P}} = 0$. The fourth term can be simplified as follows: the inner-product $E^T E$ can be expressed as:

$$\begin{aligned} E^T E = &\left(\Phi - P^{(\mathcal{P})}\Phi\right)^T \left(\Phi - P^{(\mathcal{P})}\Phi\right) \\ =&\Phi^T\Phi - \Phi^T P^{(\mathcal{P})}\Phi - \Phi^T P^{(\mathcal{P})}\Phi + \Phi^T P^{(\mathcal{P})} P^{(\mathcal{P})}\Phi \\ =&\Phi^T\Phi - \Phi^T P^{(\mathcal{P})}\Phi = K - \tilde{K}_{\mathcal{P}} = G\,, \end{aligned}$$

since $P^{(\mathcal{P})} P^{(\mathcal{P})} = P^{(\mathcal{P})}$ (an idempotent matrix). Using Theorem 4.1, $\tilde{E}_{\mathcal{R}}^T \tilde{E}_{\mathcal{R}} = \tilde{G}_{\mathcal{R}}$.

In Equation (4.7), removing the terms $\tilde{\Phi}_{\mathcal{P}}^T \tilde{E}_{\mathcal{R}}$, $\tilde{E}_{\mathcal{R}}^T \tilde{\Phi}_{\mathcal{P}}$ and substituting $\tilde{\Phi}_{\mathcal{P}}^T \tilde{\Phi}_{\mathcal{P}}$ and $\tilde{E}_{\mathcal{R}}^T \tilde{E}_{\mathcal{R}}$ with $\tilde{K}_{\mathcal{P}}$ and $\tilde{G}_{\mathcal{R}}$ respectively proves the theorem. ■

Theorem 4.2 indicates that the Nyström approximation of K based on $\mathcal{S}$ can be constructed by first calculating the Nyström approximation based on any subset $\mathcal{P} \subset \mathcal{S}$, and then recursively applying the Nyström method to the residual matrix $K - \tilde{K}_{\mathcal{P}}$ based on $\mathcal{S} \setminus \mathcal{P}$. If $\mathcal{P}$ consists of a single element: $\mathcal{P} = \{p\}$, the following corollary directly follows from Theorem 4.2.

Corollary 4.3 *Given a set of columns $\mathcal{S}$. For any $p \in \mathcal{S}$,*

$$\tilde{K}_{\mathcal{S}} = \tilde{K}_{\{p\}} + \tilde{G}_{\mathcal{R}}$$

where $\tilde{K}_{\mathcal{S}}$ is the Nyström approximations of K based on the subset $\mathcal{S}$ of columns, $\tilde{K}_{\{p\}}$ is the rank-1 Nyström approximation of K based on the p-th columns:

$$\tilde{K}_{\{p\}} = \frac{1}{K_{pp}} K_{:p} K_{:p}^T,$$

$G = K - \tilde{K}_{\{p\}}$ is the residual matrix of K after subtracting $\tilde{K}_{\{p\}}$, and $\tilde{G}_{\mathcal{R}}$ is the Nyström approximation of G based on the subset $\mathcal{R} = \mathcal{S} \setminus \{p\}$ of columns.

Proof Using Theorem 4.2 and replacing $\mathcal{P}$ with $\{p\}$ gives the rank-1 recursive formula. Using Equation (4.2) when K_{pp} is a scalar and $K_{:p}$ is a column vector gives the rank-1 formula for Nyström approximation. ■

This means that rank-l Nyström approximation of K (where $l = |\mathcal{S}|$) can be constructed in a recursive manner by first calculating a rank-1 Nyström approximation of K based on the column corresponding to p, and then calculating the rank-$(l-1)$ Nyström approximation of the residual matrix based on the columns corresponding to the remaining elements

of $\mathcal{S}$. Based on this recursion, the rank-l Nyström approximation of K can be expressed as a summation of rank-1 approximations calculated at different iterations of the recursive formula.

Let $G^{(t)}$ be the residual matrix of K at the start of iteration t (where $G^{(1)} = K$), $p^{(t)}$ be the index of the column sampled at iteration t, $\delta^{(t)} = (G_{:p})^{(t)}$ be the column sampled from G and $\alpha^{(t)} = (G_{pp})^{(t)}$ be the corresponding diagonal element. Define $\boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)}/\sqrt{\alpha^{(t)}}$. The rank-1 Nyström approximation of the residual matrices at different iterations can be expressed as:

$$\begin{aligned}
\tilde{K}_{\{p^{(1)}\}} = \tilde{G}^{(1)}_{\{p^{(1)}\}} &= \boldsymbol{\omega}^{(1)}\boldsymbol{\omega}^{(1)T}\,, \\
\tilde{G}^{(2)}_{\{p^{(2)}\}} &= \boldsymbol{\omega}^{(2)}\boldsymbol{\omega}^{(2)T}\,, \\
&\cdots \\
\tilde{G}^{(l)}_{\{p^{(l)}\}} &= \boldsymbol{\omega}^{(l)}\boldsymbol{\omega}^{(l)T}\,.
\end{aligned}$$

The rank-l Nyström approximation can be expressed as:

$$\begin{aligned}
\tilde{K}_{\mathcal{S}} &= \boldsymbol{\omega}^{(1)}\boldsymbol{\omega}^{(1)T} + \tilde{G}^{(2)}_{\mathcal{S}\setminus\{p^{(1)}\}} \\
&= \boldsymbol{\omega}^{(1)}\boldsymbol{\omega}^{(1)T} + \boldsymbol{\omega}^{(2)}\boldsymbol{\omega}^{(2)T} + \tilde{G}^{(3)}_{\mathcal{S}\setminus\{p^{(1)},\,p^{(2)}\}} \\
&= \sum_{t=1}^{l} \boldsymbol{\omega}^{(t)}\boldsymbol{\omega}^{(t)T}
\end{aligned} \tag{4.8}$$

Similar to Equation (3.16) on page 37, $\boldsymbol{\delta}^{(t)}$ and $\alpha^{(t)}$ can be efficiently calculated as:

$$\begin{aligned}
\boldsymbol{\delta}^{(t)} &= K_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)}\boldsymbol{\omega}^{(r)}, \\
\alpha^{(t)} &= \boldsymbol{\delta}_p^{(t)},
\end{aligned} \tag{4.9}$$

where $K_{:p}$ denotes the p-th column of K, and $\boldsymbol{\delta}_p$ denotes the p-th element of $\boldsymbol{\delta}$.

Let W be an $l \times n$ matrix whose t-th row is $\boldsymbol{\omega}^{(t)T}$:

$$W = \begin{bmatrix} \boldsymbol{\omega}^{(1)} & \ldots & \boldsymbol{\omega}^{(l)} \end{bmatrix}^T .$$

The rank-l Nyström approximation $\tilde{K}_{\mathcal{S}}$ can be expressed in a matrix form as:

$$K = W^T W. \tag{4.10}$$

The columns of W can be directly used to represent the data instances in an l-dimensional space. This space is implicitly defined by an orthonormal basis that spans the subspace of selected columns in the high-dimensional feature space defined by the kernel: $Q = orth\left(\Phi_{:\mathcal{S}}\right)$.

In order to calculate a rank-k Nyström approximation where $k \leq l$, the proposed algorithm first calculates the k leading left singular vectors of W (or equivalently, the eigenvectors of WW^T), and then uses these vectors to represent the columns of W in a k-dimensional space as follows:

$$Y = \Omega_k^T W, \tag{4.11}$$

where Y is a $k \times n$ matrix whose columns represent data instances in the k-dimensional space, and Ω_k is a $k \times k$ matrix whose columns are the k leading left singular vectors of W.

The corresponding rank-k Nyström approximation of K is:

$$\tilde{K}_{\mathcal{S},k} = Y^T Y = W^T \Omega_k \Omega_k^T W. \tag{4.12}$$

Although the recursive Nyström algorithm calculates the same rank-l Nyström approximation $\tilde{K}_{\mathcal{S}}$ as the traditional Nyström formula (Eq. 4.2), it calculates different estimates of Y and $\tilde{K}_{\mathcal{S},k}$. The advantage of the recursive algorithm is that the basis of low-dimensional representation is orthogonal, and that $\tilde{K}_{\mathcal{S},k}$ is the best rank-k approximation of $\tilde{K}_{\mathcal{S}}$.

The computational complexity of calculating $\boldsymbol{\delta}^{(t)}$ (Eq. 4.9) in terms of previous $\boldsymbol{\omega}$'s is $O\left(nt\right)$, and that of W is $O\left(nl^2\right)$. The computational complexity of orthogonalization steps is $O\left(l^3 + nl^2\right)$. Thus, the computational complexity of calculating Y is $O\left(l^3 + nlk + nl^2\right)$ and that of $\tilde{K}_{\mathcal{S},k}$ is $O\left(l^3 + nlk + n^2k + nl^2\right)$. This is the same complexity as the traditional Nyström method with orthogonalization.

In addition to calculating the Nyström approximation in a recursive manner, Theorem 4.2 can be directly used to derive a recursive formula for the kernel column subset selection criterion.

Theorem 4.4 *Given a set of columns $\mathcal{S}$. For any $\mathcal{P} \subset \mathcal{S}$,*

$$F\left(\mathcal{S}\right) = F\left(\mathcal{P}\right) - trace\left(\tilde{G}_{\mathcal{R}}\right)$$

where $G = K - \tilde{K}_{\mathcal{P}}$, and $\tilde{G}_{\mathcal{R}}$ is the Nyström approximation of G based on the subset $\mathcal{R} = \mathcal{S} \setminus \mathcal{P}$ of columns.

Proof Substituting with $\tilde{K}_{\mathcal{S}} = \tilde{K}_{\mathcal{P}} + \tilde{G}_{\mathcal{R}}$ (Theorem 4.2) in Equation (4.6) gives:

$$F(\mathcal{S}) = trace\left(K - \tilde{K}_{\mathcal{P}} - \tilde{G}_{\mathcal{R}}\right) = trace\left(K - \tilde{K}_{\mathcal{P}}\right) - trace\left(\tilde{G}_{\mathcal{R}}\right)$$

Substituting $trace\left(K - \tilde{K}_{\mathcal{P}}\right)$ with $F(\mathcal{P})$ proves the theorem. ■

The recursive formula of Theorem 4.4 allows the development of the efficient greedy algorithm presented in the following section.

4.4 Greedy Sampling Criterion

The recursive nature of the Nyström method can be used to develop an efficient greedy algorithm for sampling columns while calculating the low-rank approximation of the kernel matrix. The basic idea here is to select, at each iteration, the column that minimizes the column subset selection criterion. Let $\mathcal{S}$ be the subset of columns selected at the first $t-1$ iterations. The greedy sampling can be formally defined as:

Problem 4.2 *At iteration t, find column p such that,*

$$p = \arg\min_{i} \quad F(\mathcal{S} \cup \{i\}) \tag{4.13}$$

where $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

Using Theorem 4.4, $F(\mathcal{S} \cup \{i\})$ can be calculated in a recursive manner as:

$$F(\mathcal{S} \cup \{i\}) = F(\mathcal{S}) - trace\left(\tilde{G}_{\{i\}}\right),$$

where $G = K - \tilde{K}_{\mathcal{S}}$ is the residual matrix after subtracting the Nyström approximation of K based on $\mathcal{S}$. Using the properties that: $trace(aA) = a\,trace(A)$, $trace(AB) = trace(BA)$ and $trace(a) = a$, where a is a scalar, the second term can be simplified as follows.

$$\begin{aligned} trace\left(\tilde{G}_{\{i\}}\right) &= trace\left(\frac{1}{G_{ii}} G_{:i} G_{:i}^T\right) = \frac{1}{G_{ii}} trace\left(G_{:i}^T G_{:i}\right) \\ &= \frac{1}{G_{ii}} G_{:i}^T G_{:i} = \frac{\|G_{:i}\|^2}{G_{ii}}, \end{aligned}$$

The greedy selection criterion can be formally defined as:

Problem 4.3 (Greedy Kernel Column Subset Selection) *At iteration t, find column p such that,*

$$p = \arg\max_i \quad \frac{\|G_{:i}\|^2}{G_{ii}} \tag{4.14}$$

where $G = K - \tilde{K}_{\mathcal{S}}$, and $\mathcal{S}$ is the set of columns selected during the first $t-1$ iterations.

To evaluate this selection criterion, at each iteration, the term $\|G_{:i}\|^2/G_{ii}$ has to be evaluated for all the columns of the current residual matrix G, and the column with the maximum criterion function is selected.

The computational complexity of the selection criterion is $O(n^2 + n)$ per iteration, and it requires $O(n^2)$ memory to store the residual of the whole kernel matrix after each iteration. In the rest of this section, two novel techniques are proposed to reduce the memory and time requirements of the greedy selection criterion.

4.4.1 Memory-Efficient Criterion

To reduce the memory requirements of the greedy algorithm, the sampling criterion for each data instance can be calculated in a recursive manner as follows. Let $\boldsymbol{f}_i = \|G_{:i}\|^2$ and $\boldsymbol{g}_i = G_{ii}$ be the numerator and denominator of the criterion function for data point i respectively, $\boldsymbol{f} = [\boldsymbol{f}_i]_{i=1..n}$, and $\boldsymbol{g} = [\boldsymbol{g}_i]_{i=1..n}$. Similar to Theorem 3.6 on page 38, it can be shown that $\boldsymbol{f}$ and $\boldsymbol{g}$ can be calculated recursively as follows:

$$\begin{aligned}
\boldsymbol{f}^{(t)} &= \left(\boldsymbol{f} - 2\left(\boldsymbol{\omega} \circ \left(K\boldsymbol{\omega} - \Sigma_{r=1}^{t-2}\left(\boldsymbol{\omega}^{(r)^T}\boldsymbol{\omega}\right)\boldsymbol{\omega}^{(r)}\right)\right) + \|\boldsymbol{\omega}\|^2(\boldsymbol{\omega}\circ\boldsymbol{\omega})\right)^{(t-1)}, \\
\boldsymbol{g}^{(t)} &= \left(\boldsymbol{g} - (\boldsymbol{\omega}\circ\boldsymbol{\omega})\right)^{(t-1)}.
\end{aligned} \tag{4.15}$$

where $\circ$ represents the Hadamard product operator, $\|.\|$ is the ℓ_2 norm, $\boldsymbol{f}_i^{(0)} = \|K_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = K_{ii}$ for $i = 1, 2, \ldots n$. This means that the greedy criterion can be memory-efficient by only maintaining two score variables for each data point, $\boldsymbol{f}_i$ and $\boldsymbol{g}_i$, and updating them at each iteration based on their previous values and the selected columns so far.

4.4.2 Partition-Based Criterion

The memory-efficient selection criterion eliminates the complexity of calculating and storing a new residual matrix at each iteration. This reduces the computational complexity

Algorithm 6 Greedy Nyström Approximation

Inputs: Kernel matrix K, Number of columns l, Target rank k

Outputs: Selected columns $\mathcal{S}$, Rank-l Nyström approximation $\tilde{K}_{\mathcal{S}}$, Rank-k Nyström approximation, $\tilde{K}_{\mathcal{S},k}$, l-dimensional embedding W, k-dimensional embedding Y

Steps:

1. Initialize $\mathcal{S} = \{\ \}$
2. Initialize $\boldsymbol{f}_i^{(0)} = \|K_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = K_{ii}$ for $i = 1,\ 2,\ ...\ n$
3. Repeat $t = 1 \rightarrow l$:
   (a) $q = \arg\max_i \boldsymbol{f}_i^{(t)} / \boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{q\}$
   (b) $\boldsymbol{\delta}^{(t)} = K_{:q} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_q^{(r)} \boldsymbol{\omega}^{(r)}$, $\alpha^{(t)} = \boldsymbol{\delta}_q^{(r)}$
   (c) $\boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)} / \sqrt{\alpha^{(t)}}$
   (d) Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s

$$\boldsymbol{f}^{(t)} = \left(\boldsymbol{f} - 2 \left(\boldsymbol{\omega} \circ \left(K\boldsymbol{\omega} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{\omega}^{(r)^T} \boldsymbol{\omega} \right) \boldsymbol{\omega}^{(r)} \right) \right) + \|\boldsymbol{\omega}\|^2 (\boldsymbol{\omega} \circ \boldsymbol{\omega}) \right)^{(t-1)},$$
$$\boldsymbol{g}^{(t)} = \left(\boldsymbol{g} - (\boldsymbol{\omega} \circ \boldsymbol{\omega}) \right)^{(t-1)},$$

   where $\circ$ represents the Hadamard product operator.

4. $W = \left[\ \boldsymbol{\omega}^{(1)} \quad ... \quad \boldsymbol{\omega}^{(l)}\ \right]^T, \ \ \tilde{K}_{\mathcal{S}} = W^T W$
5. $\Omega = eigvec(WW^T), \ \ Y = \Omega_k^T W, \ \ \tilde{K}_{\mathcal{S},k} = Y^T Y$

of the greedy selection algorithm to $O(n^2)$ per iteration (that of calculating $K\boldsymbol{\omega}$). In addition, the computational complexity of calculating the initial value of $\boldsymbol{f}$ given the kernel matrix K is still $O(n^2)$.

In order to reduce this computational complexity, a partition-based criterion is proposed. The criterion partitions data points into $c \ll n$ random groups, and selects the columns of K which best represent the centroids of these groups in the high-dimensional feature space defined by the kernel function. Let $\mathcal{P}_j$ be the set of data points that belong to the j-th partition, $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ...\mathcal{P}_c\}$ be a random partitioning of data points into c groups, and M be an $c \times n$ matrix whose element M_{ji} is the inner-product of the centroid of the j-th group and the i-th data point. M_{ji} can be calculated as follows.

$$\begin{aligned} M_{ij} &= \phi(i)^T \left(\frac{1}{|\mathcal{P}_j|} \sum_{r \in \mathcal{P}_j} \phi(r) \right) \\ &= \frac{1}{|\mathcal{P}_j|} \sum_{r \in \mathcal{P}_j} \left(\phi(i)^T \phi(r) \right) = \frac{1}{|\mathcal{P}_j|} \sum_{r \in \mathcal{P}_j} K_{ir} \end{aligned}$$

where $\phi(.)$ is a function that maps a data point to a vector in the high-dimensional feature space implicitly defined by the kernel function. In the proposed partition-based algorithm, the j-th column of the matrix M is weighted by the size of the j-th partition. The use of weighted columns avoids any bias towards larger groups.

Let L be an $c \times n$ matrix whose element M_{ji} is

$$L_{ji} = |\mathcal{P}_j| M_{ij} = \sum_{r \in \mathcal{P}_j} K_{ir} .$$

Let $H^{(t)}$ be the residual of L at iteration t, and $\boldsymbol{\gamma}^{(t)}$ be the column of H corresponding to the selected column at iteration t, which can be calculated as: $\boldsymbol{\gamma}^{(t)} = L_{:q} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_q^{(r)} \boldsymbol{\upsilon}^{(r)}$. Similar to Theorem 3.7 on page 42, it can be shown that for partition-based sampling, $\boldsymbol{f}$ and $\boldsymbol{g}$ can be calculated as:

$$\begin{aligned} \boldsymbol{f}^{(t)} &= \left(\boldsymbol{f} - 2 \left(\boldsymbol{\omega} \circ \left(L^T \boldsymbol{\upsilon} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{\upsilon}^{(r)T} \boldsymbol{\upsilon} \right) \boldsymbol{\omega}^{(r)} \right) \right) + \|\boldsymbol{\upsilon}\|^2 (\boldsymbol{\omega} \circ \boldsymbol{\omega}) \right)^{(t-1)}, \\ \boldsymbol{g}^{(t)} &= \left(\boldsymbol{g} - (\boldsymbol{\omega} \circ \boldsymbol{\omega}) \right)^{(t-1)}. \end{aligned} \tag{4.16}$$

where $\boldsymbol{\upsilon}^{(t)} = \boldsymbol{\gamma}^{(t)} / \sqrt{\alpha^{(t)}}$, $\boldsymbol{f}_i^{(0)} = \|L_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = K_{ii}$ for $i = 1, 2, ...n$. The computational complexity of the new update formulas is $O(ncl + nl^2)$ (or $O(nc + nt)$ per iteration). In

general, it requires $O(n^2)$ to calculate L given K. However, L needs to be calculated only once during the calculation of K. In the case of a linear kernel, this complexity can be significantly reduced by calculating the centroids of each group in the feature space, and then the inner-products between each centroid and all data points. This computational complexity can be reduced further if the data matrix is very sparse. In addition, there is no need to calculate and store the whole kernel matrix in order to calculate L. Algorithm 7 shows the complete greedy Nyström algorithm.

4.5 Related Work

4.5.1 Random Sampling

Different sampling schemes have been used with the Nyström method. Williams and Seeger [86], who first proposed the use of Nyström approximation for kernel methods, used uniform sampling without replacement to select columns. This has been the most commonly used sampling scheme for Nyström methods. Non-uniform sampling has also been used with Nyström methods. This includes non-uniformly sampling columns based on the corresponding diagonal elements of the kernel matrix [91], or the norms of its columns [68]. Recently, Kumar et al. [92] showed that uniform sampling without replacement outperforms other random sampling techniques on real data sets.

Adaptive sampling has also been used with Nyström methods. These techniques update sampling probabilities based on intermediate approximations of the kernel matrix. Deshpande et al. [93] suggested an adaptive sampling algorithm which iteratively samples subsets of columns using probabilities calculated based on the low-rank approximation error so far. This adaptive mechanism is more effective than fixed sampling. However, it is computationally more complex, as it requires the calculation of the Nyström approximation at each iteration of the algorithm. A more efficient algorithm for adaptive sampling [87] calculates sampling probabilities based on the approximation error of a small part of the kernel matrix. A more recent work [97] uses an ensemble of Nyström approximations to obtain a better low-rank approximation of the kernel matrix.

4.5.2 Deterministic Sampling

Besides random sampling, deterministic sampling has also been used with Nyström methods. Sparse greedy matrix approximation (SGMA) [94] is a related algorithm, which selects

Algorithm 7 Partition-based Greedy Nyström Approximation

Inputs: Kernel matrix K, Number of columns l, Target rank k

Outputs: Selected columns $\mathcal{S}$, Rank-l Nyström approximation $\tilde{K}_{\mathcal{S}}$, Rank-k Nyström approximation, $\tilde{K}_{\mathcal{S},k}$, l-dimensional embedding W, k-dimensional embedding Y

Steps:

1. Initialize $\mathcal{S} = \{\ \}$, Generate a random partitioning $\mathbf{P}$, Calculate L: $L_{ji} = \sum_{r \in \mathcal{P}_j} K_{ir}$
2. Initialize $\boldsymbol{f}_i^{(0)} = \|L_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = K_{ii}$ for $i = 1,\ 2,\ \ldots\ n$
3. Repeat $t = 1 \rightarrow l$:
   (a) $q = \arg\max_i \ \boldsymbol{f}_i^{(t)} / \boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{q\}$
   (b) $\boldsymbol{\delta}^{(t)} = K_{:q} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_q^{(r)} \boldsymbol{\omega}^{(r)}$, $\alpha^{(t)} = \boldsymbol{\delta}_q^{(r)}$
   (c) $\boldsymbol{\gamma}^{(t)} = L_{:q} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_q^{(r)} \boldsymbol{v}^{(r)}$
   (d) $\boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)} / \sqrt{\alpha^{(t)}}$, $\boldsymbol{v}^{(t)} = \boldsymbol{\gamma}^{(t)} / \sqrt{\alpha^{(t)}}$
   (e) Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s

$$\boldsymbol{f}^{(t)} = \left(\boldsymbol{f} - 2 \left(\boldsymbol{\omega} \circ \left(L^T \boldsymbol{v} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{v}^{(r)T} \boldsymbol{v} \right) \boldsymbol{\omega}^{(r)} \right) \right) + \|\boldsymbol{v}\|^2 \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$

$$\boldsymbol{g}^{(t)} = \left(\boldsymbol{g} - \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$

   where $\circ$ represents the Hadamard product operator.

4. $W = \left[\ \boldsymbol{\omega}^{(1)} \quad \ldots \quad \boldsymbol{\omega}^{(l)}\ \right]^T$, $\tilde{K}_{\mathcal{S}} = W^T W$
5. $\Omega = eigvec(WW^T)$, $Y = \Omega_k^T W$, $\tilde{K}_{\mathcal{S},k} = Y^T Y$

a set of basis kernel functions, and represents other kernel functions as a linear combination of these basis functions: $\tilde{K} = K_{:\mathcal{S}}T$, where $K_{:\mathcal{S}}$ denotes the subset of selected columns, and T is a matrix of coefficients. The authors showed that low-rank approximation which optimizes the approximation error in the reproducing kernel Hilbert space (RKHS) is equal to Nyström approximation [98, Chapter 10]. They also proposed a greedy algorithm to select columns of the kernel matrix and recursively update T based on the newly selected columns. The selection criterion used by SGMA is based on maximizing the improvement of the low-rank approximation error in RKHS. To reduce the complexity of this algorithm, a probabilistic speedup was suggested by the authors to evaluate the criterion function for only a random subset of columns. This makes the complexity of the selection criterion $O(Nnl^2)$ (or $O(Nnt)$ per iteration), where N is the size of the random subset.

Ouimet and Bengio [99] proposed another greedy sampling algorithm which recursively selects samples that are far (using some threshold) from the subspace spanned by previously selected samples. Incomplete Cholesky decomposition [95] can also be used to greedily select columns for the Nyström method. Zhang et al. [21] recently proposed an algorithm which first applies the k-means algorithm to cluster data points, and then uses the centroids of clusters for calculating the Nyström approximation. As the k-means algorithm scales as $O(nmlt)$, where m is the number of features and t is the number of iterations, this algorithm is computationally infeasible for data sets with large numbers of features.

4.5.3 Comparison to Related Work

Like adaptive sampling, the greedy algorithm presented in this chapter selects columns based on intermediate approximations of the kernel matrix. However, at each iteration, the greedy algorithm deterministically selects one column, while adaptive methods randomly sample a subset of columns. In term of computational complexity, the complexity of adaptive sampling based on the full kernel [93] is $O(n^2v + nv^2 + v^3)$ per iteration, where v is the number of samples selected so far. The greedy selection criterion without partitioning ($O(n^2l)$) is therefore less complex than the last iteration of the adaptive algorithm with the full kernel (when $v = l$). On the other hand, the greedy criterion without partitioning is more complex than adaptive sampling based on part of the kernel [87], which is $O(nv^2+v^3)$ per iteration.

In comparison to SGMA [94], it can be shown that maximizing the improvement in approximation error is equivalent to minimizing the squared reconstruction error in the feature space (as shown in Problem 4.3). However, the basic selection criterion presented here ($O(n^2l)$) is more efficient than that of SGMA with probabilistic speedup ($O(nNl^2)$)

when $l/n \geq 1/N$. In addition, the approximation of K as $K_{:\mathcal{S}}T$ does not allow SGMA to be directly applied to approximate spectral decomposition and dimension reduction. In this case, the Nyström method has to be applied to columns selected by SGMA, which requires extra computational cost. In comparison to k-means [21], which is $O(nmlt)$, the greedy selection criterion is computationally less complex for data sets with large numbers of features (when $mt > n$). On the other hand, the partition-based selection criterion ($O(ncl + nl^2)$) is much less complex than the two adaptive sampling methods, SGMA, and sampling based on k-means centroids.

The greedy Nyström algorithm is also different from the greedy algorithm proposed by Çivril and Magdon-Ismail [72], as the latter depends on calculating the leading singular vectors to select columns.

4.6 Experiments and Results

Experiments have been conducted on six benchmark data sets whose properties are described in Table 3.1 and Section 3.5.

Similar to previous work [92], the low-rank approximations obtained by greedy Nyström algorithm are compared to those obtained by other Nyström methods relative to the best low-rank approximation obtained by singular value decomposition. In particular, the following quality measure is used:

$$\text{Relative Accuracy } = \frac{\|K - \tilde{K}_r\|_F}{\|K - \tilde{K}_{Nys}\|_F}, \tag{4.17}$$

where K is the kernel matrix, $\tilde{K}_r$ is the best rank-r approximation obtained using singular value decomposition, $\tilde{K}_{Nys}$ is the rank-r Nyström approximation (i.e., $\tilde{K}_{\mathcal{S}}$, or $\tilde{K}_{\mathcal{S},k}$), and $\|.\|_F$ is the Frobenius norm. The relative accuracy is between 0 and 1 with higher values indicating a better low-rank approximation. The run times of different algorithms are also compared.

The basic greedy Nyström algorithm (**GreedyNyström**) and its partition-based variant (**PartGreedyNys**) are compared to six well-known Nyström methods:

- **UniNoRep**: is the uniform sampling method without replacement, which has been shown to outperform other random sampling methods [92].

- **AdaptFull**: is the adaptive sampling method based on the full kernel matrix [93]. Similar to Kumar et al. [87], 10 iterations were used (i.e., $l/10$ columns were sampled in each iteration).
- **AdaptPart**: is the adaptive sampling method based on a part of the kernel matrix [87] with the same number of iterations as **AdaptFull**.
- **k-means**: is the Nyström method based on k-means centroids [21]. The implementation of the k-means algorithm by Zhang et al. [21] was used, with the same number of iterations.
- **SGMA**: is the SGMA algorithm with probabilistic speedup [94]. A random subset size of 59 was used, as suggested by Smola and Schölkopf [94]
- **ICD**: is the incomplete Cholesky decomposition with symmetric pivoting [95]

Experiments with randomness were repeated ten times, and the average and standard deviation of measures were calculated. Linear kernels were used for document data sets, and Gaussian kernels with $\sigma = 10$ for image data sets.

Two sets of experiments were conducted to evaluate the quality of rank-l and rank-k Nyström approximations ($\tilde{K}_{\mathcal{S}}$ and $\tilde{K}_{\mathcal{S},k}$) for different sampling methods. For $\tilde{K}_{\mathcal{S},k}$, as SGMA and ICD do not directly allow the calculation rank-k approximation from $\tilde{K}$, SGMA and ICD were used for selecting columns and then a traditional Nyström formula was applied.

Figures 4.1-4.4 show the relative accuracies and run times for the two experiments. In addition, Tables 4.1 and 4.2 show the relative accuracy measures for the best performing methods in obtaining rank-l Nyström approximation (k-**means**, **SGMA**, **GreedyNyström**, and **PartGreedyNys** (with $c = 100$). Each sub-table represents a data set and each column represents a percentage of selected columns. The relative accuracy measures in each sub-column are divided into groups according to their statistical significance. The best group of methods is highlighted in bold, and the second best group is underlined. The statistical significance tests were performed as explained in Section 3.5 on page 50.

It can be observed from results that the greedy Nyström method (**GreedyNyström**) achieves significant improvement in estimating low-rank approximations of a kernel matrix, compared to other sampling-based methods. It also achieves better accuracy than **SGMA** and k-**means** for most data sets. Although the k-**means** achieves better accuracy for some data sets, it obtains much worse accuracy for others. This inconsistency could be due to the nature of the k-means algorithm, which might obtain a poor local minimum. It

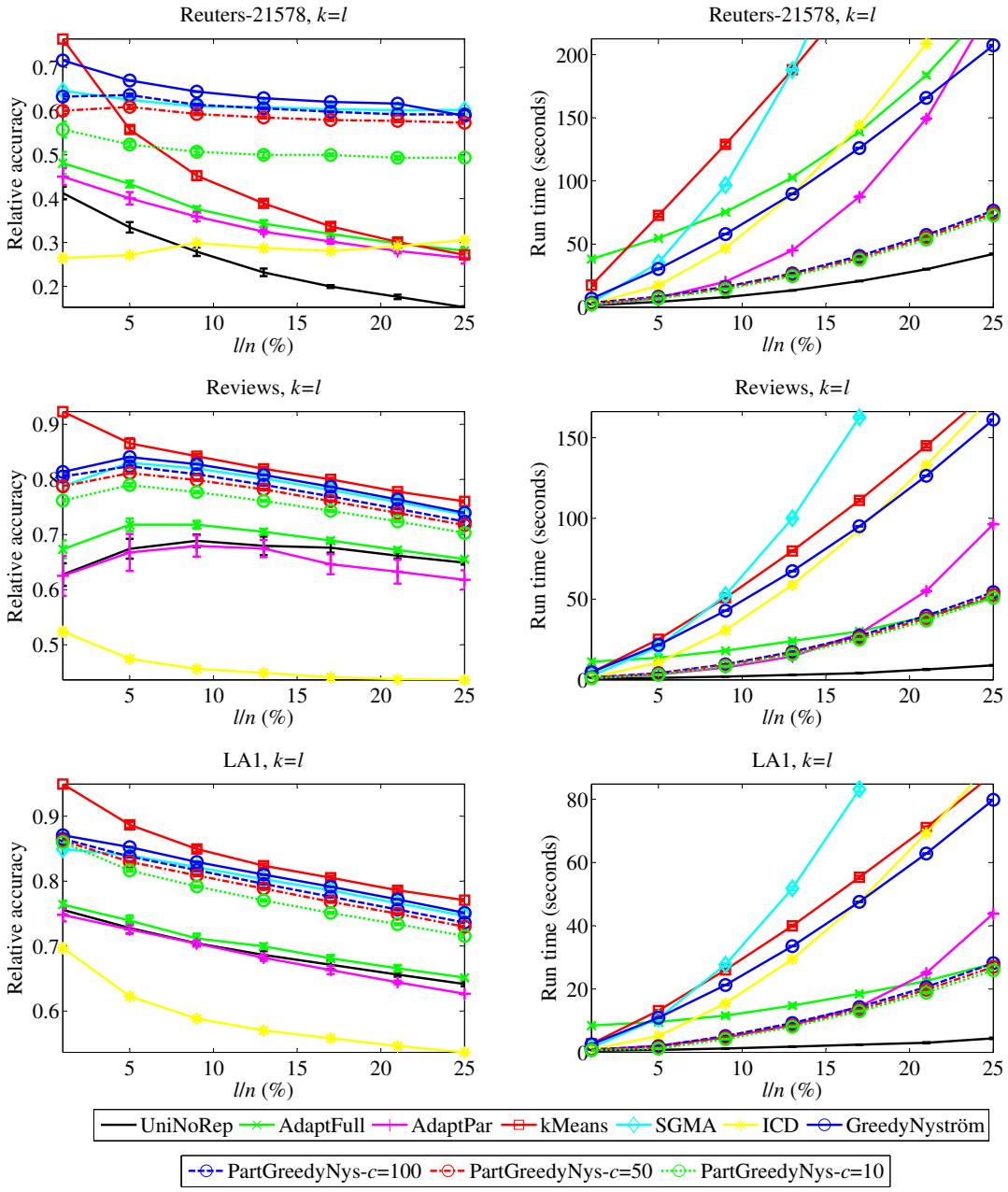

Figure 4.1: The relative accuracy measures and run times of different rank-l Nyström approximations ($\tilde{K}_{\mathcal{S}}$) for the *Reuters-21578*, *Reviews* and *LA1* data sets.

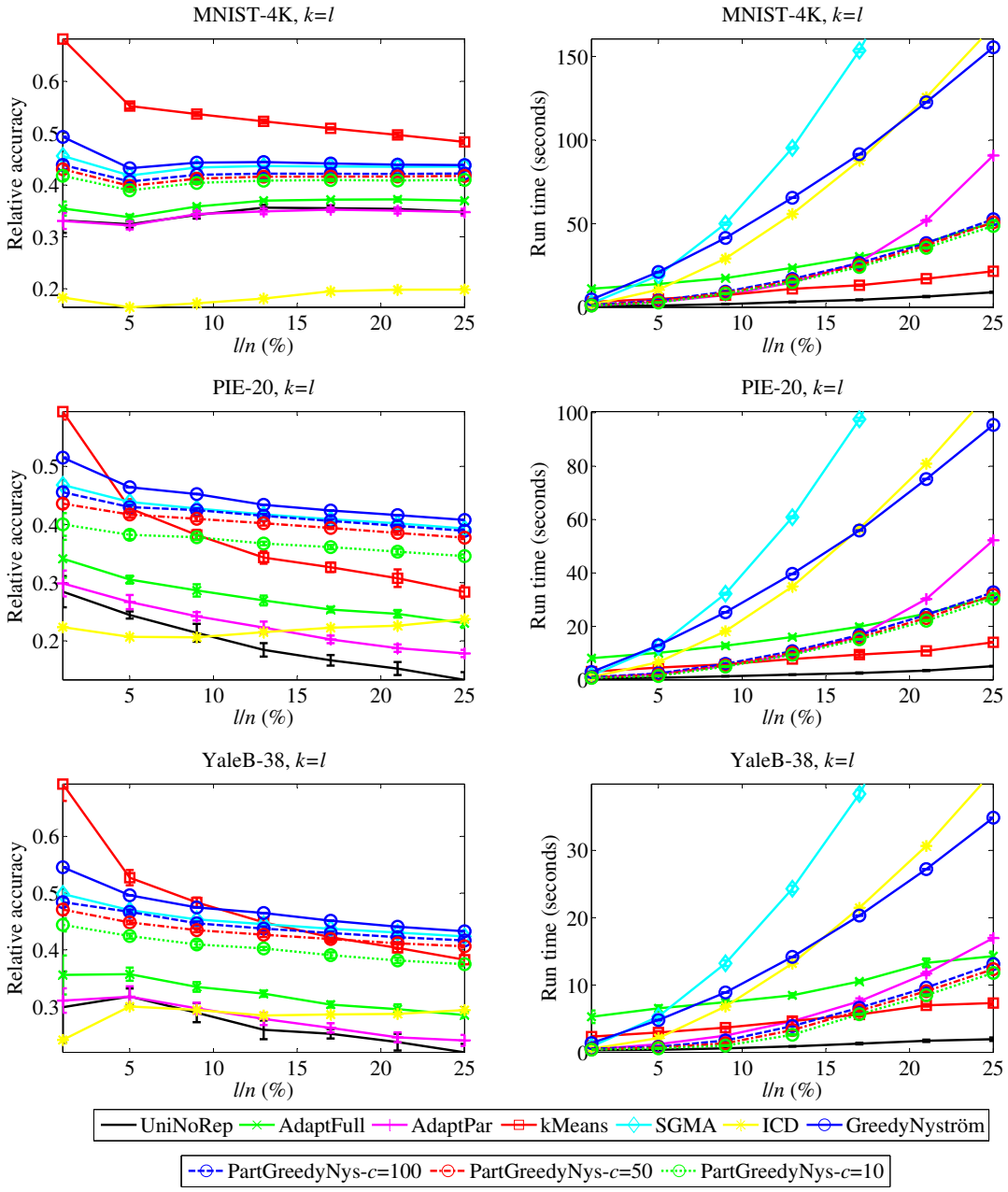

Figure 4.2: The relative accuracy measures and run times of different rank-l Nyström approximations ($\tilde{K}_{\mathcal{S}}$) for the *MNIST-4K*, *PIE-20* and *YaleB-38* data sets.

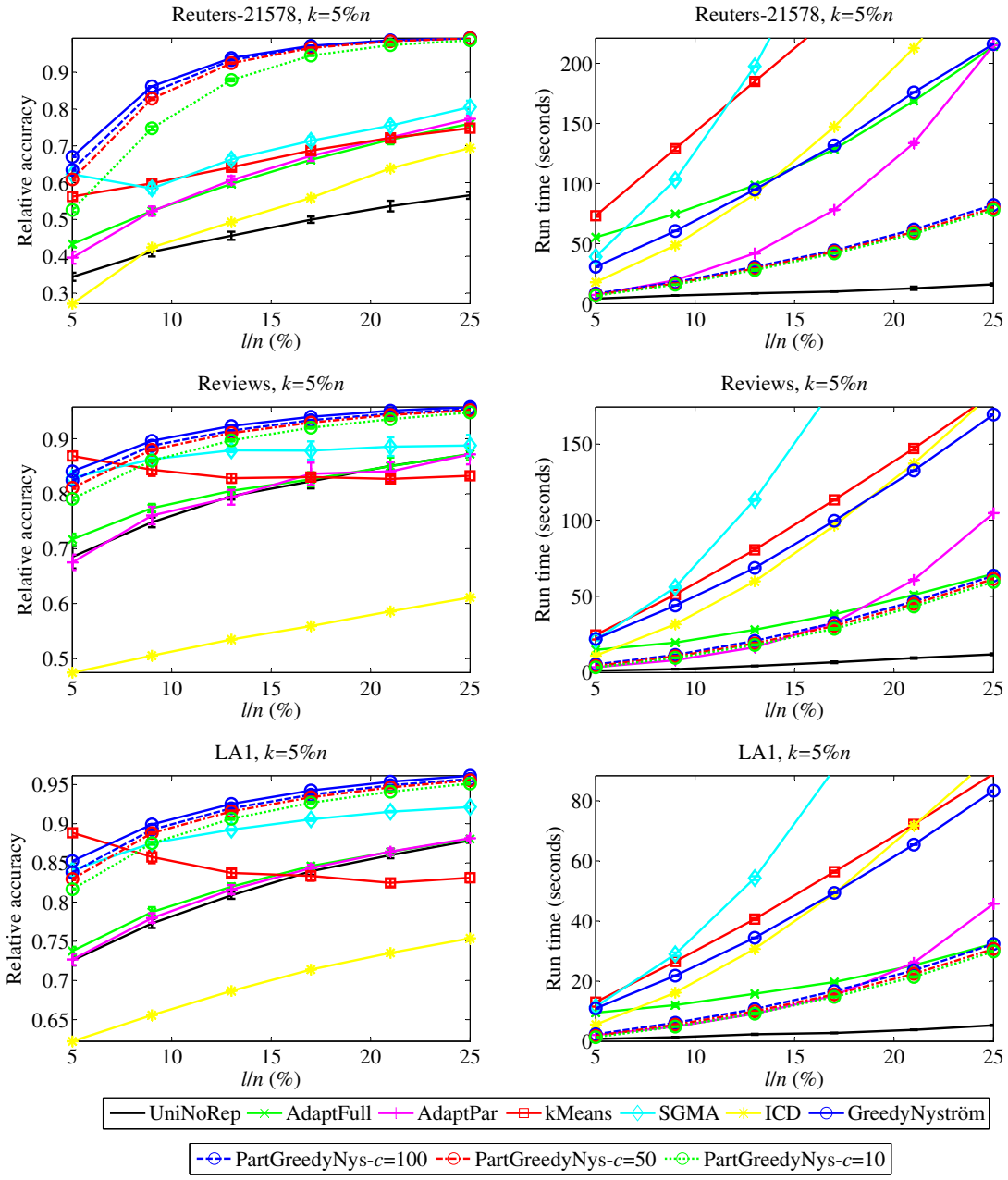

Figure 4.3: The relative accuracy measures and run times of different rank-k Nyström approximations ($\tilde{K}_{\mathcal{S},k}$) for the *Reuters-21578*, *Reviews* and *LA1* data sets.

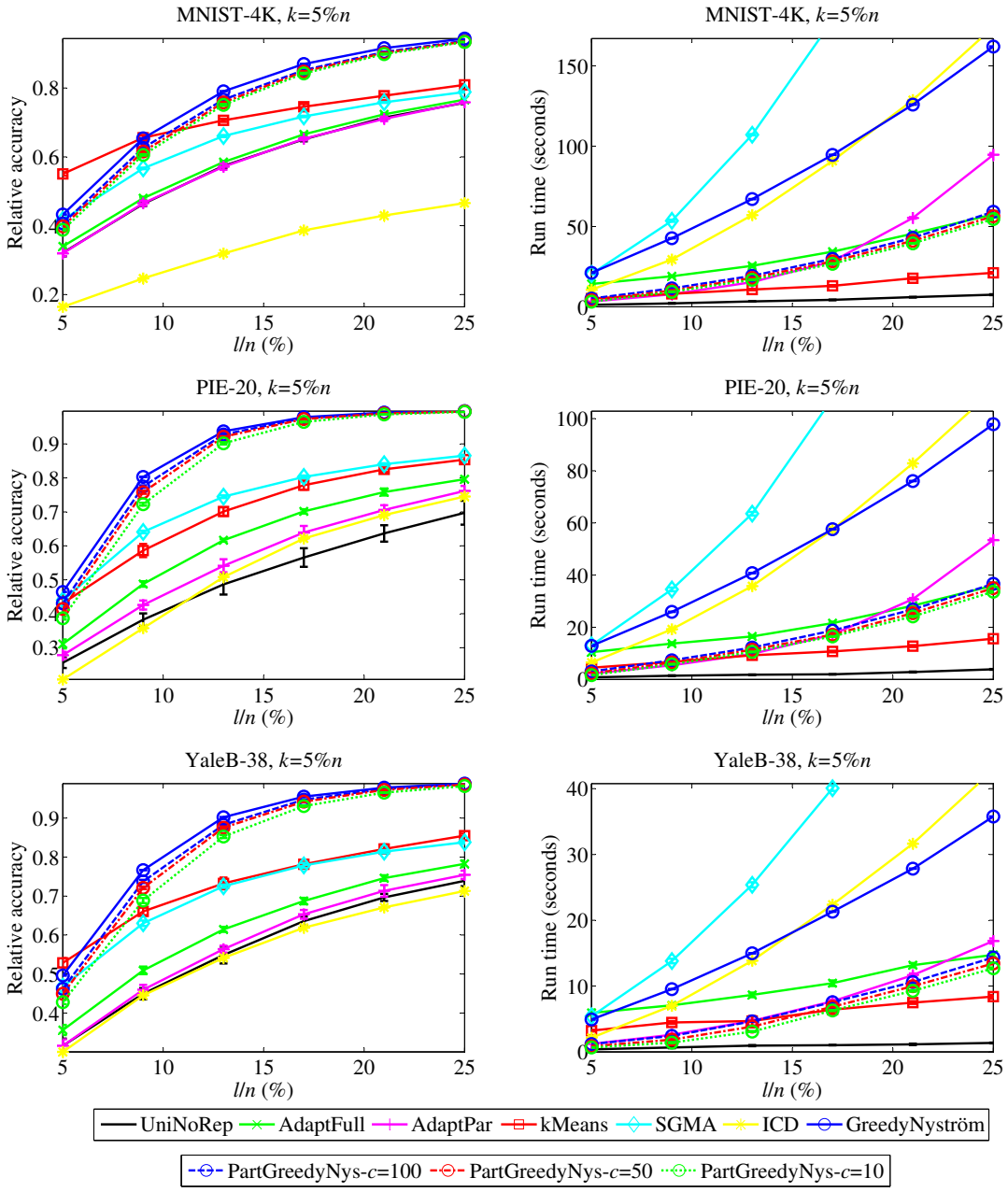

Figure 4.4: The relative accuracy measures and run times of different rank-k Nyström approximations ($\tilde{K}_{\mathcal{S},k}$) for the *MNIST-4K*, *PIE-20* and *YaleB-38* data sets.

can also be seen that **GreedyNyström** is more efficient than **SGMA** and **AdaptFull**, but is computationally more complex than **UniNoRep** and **AdaptPart**. The latter two methods, however, achieve inferior accuracies. **GreedyNyström** is also computationally less complex than k**-means** for data sets with large numbers of features. On the other hand, the partition-based algorithm (**PartGreedyNys**) outperforms the two adaptive sampling methods in obtaining low-rank approximations, and it requires small overhead in run time compared to **UniNoRep**. **PartGreedyNys** obtains slightly lower accuracies than **GreedyNyström** and **SGMA** when calculating $\tilde{K}_{\mathcal{S}}$, but in much less time, and it outperforms all other deterministic methods when calculating $\tilde{K}_{\mathcal{S},k}$. **PartGreedyNys** is also not sensitive to the number of random partitions used. It can also be noted that **ICD** obtains inferior approximation accuracies compared to other methods.[1]

The proposed greedy Nyström methods (**GreedyNyström** and **PartGreedyNys**) have also been compared to the ensemble Nyström algorithm (**Ensemble**) proposed by Kumar et al. [97]. As suggested by Kumar et al. [97], the ridge regression algorithm can be used to learn the mixture weights of different approximations using a validation set of columns sampled from the original kernel matrix. In this experiment, an ensemble of $p = 10$ Nyström approximations is used, and l columns are sampled to calculate each low-rank approximation. A validation set of $s = 20$ columns is used for estimating the mixture weights of the ensemble, and a hold-out set of $s' = 20$ columns is used to estimate the ridge parameter. Tables 4.3 and 4.4 show the relative accuracies and run time of different methods. Two values are used for l: $l = 3\%n$ and $l = 5\%n$, with $k = 1\%n$. It can be observed that both **PartGreedyNys** and **GreedyNyström** outperforms the ensemble method (**Ensemble**) in term of approximation accuracy for most data sets.

[1] This was also observed by Zhang et al. [21] and Talwalkar [100]

Table 4.1: The relative accuracy measures of the best performing Nyström methods (rank-l approximations) for the **Reuters-21578**, **Reviews**, and **LA1** data sets. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

Reuters-21578				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
kMeans	0.5580±0.0072	0.4527±0.0100	0.3894±0.0062	0.3368±0.0069
SGMA	0.6255±0.0038	0.6106±0.0035	<u>0.6090±0.0041</u>	<u>0.6042±0.0016</u>
GreedyNyström	**0.6701±0.0000**	**0.6446±0.0000**	**0.6296±0.0000**	**0.6210±0.0000**
PartGreedyNys	<u>0.6367±0.0034</u>	<u>0.6146±0.0023</u>	0.6065±0.0020	0.5987±0.0009
Reviews				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
kMeans	**0.8655±0.0093**	**0.8422±0.0035**	**0.8192±0.0039**	**0.8000±0.0026**
SGMA	0.8302±0.0019	0.8199±0.0010	0.8018±0.0007	0.7807±0.0006
GreedyNyström	<u>0.8405±0.0000</u>	<u>0.8276±0.0000</u>	<u>0.8081±0.0000</u>	<u>0.7865±0.0000</u>
PartGreedyNys	0.8239±0.0015	0.8094±0.0009	0.7901±0.0012	0.7692±0.0009
LA1				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
kMeans	**0.8866±0.0047**	**0.8495±0.0043**	**0.8237±0.0032**	**0.8052±0.0030**
SGMA	0.8399±0.0011	0.8214±0.0011	0.8030±0.0007	0.7856±0.0006
GreedyNyström	<u>0.8523±0.0000</u>	<u>0.8294±0.0000</u>	<u>0.8101±0.0000</u>	<u>0.7914±0.0000</u>
PartGreedyNys	0.8378±0.0013	0.8169±0.0013	0.7958±0.0009	0.7760±0.0014

Table 4.2: The relative accuracy measures of the best performing Nyström methods (rank-l approximations) for the **MNIST-4K**, **PIE-20**, and **YaleB-38** data sets. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

MNIST-4K				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
kMeans	**0.5522±0.0050**	**0.5369±0.0040**	**0.5228±0.0036**	**0.5092±0.0021**
SGMA	0.4182±0.0020	0.4338±0.0014	0.4365±0.0009	0.4357±0.0006
GreedyNyström	0.4323±0.0000	0.4431±0.0000	0.4441±0.0000	0.4414±0.0000
PartGreedyNys	0.4067±0.0019	0.4197±0.0019	0.4220±0.0015	0.4217±0.0007
PIE-20				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
kMeans	0.4277±0.0158	0.3821±0.0087	0.3431±0.0106	0.3266±0.0083
SGMA	0.4387±0.0024	0.4274±0.0020	0.4174±0.0029	0.4091±0.0015
GreedyNyström	**0.4640±0.0000**	**0.4523±0.0000**	**0.4340±0.0000**	**0.4241±0.0000**
PartGreedyNys	0.4299±0.0038	0.4249±0.0030	0.4151±0.0031	0.4060±0.0030
YaleB-38				
Method	$l/n = 5\%$	$l/n = 9\%$	$l/n = 13\%$	$l/n = 17\%$
kMeans	**0.5273±0.0135**	**0.4837±0.0086**	0.4488±0.0065	0.4223±0.0093
SGMA	0.4702±0.0055	0.4538±0.0024	0.4453±0.0032	0.4373±0.0021
GreedyNyström	0.4962±0.0000	**0.4748±0.0000**	**0.4650±0.0000**	**0.4516±0.0000**
PartGreedyNys	0.4670±0.0041	0.4469±0.0034	0.4376±0.0038	0.4298±0.0026

Table 4.3: The relative accuracy measures of the proposed greedy Nyström methods compared to the ensemble Nyström method and Nyström method with uniform sampling (rank-k approximations $\tilde{K}_{\mathcal{S},k}$, $k = 1\%n$). In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

Data sets	**UniNoRep**	**EnsembleNys**	**GreedyNyström**	**PartGreedyNys**
		l/n= **3%**		
Reuters-21578	0.5306	0.5965	**0.9074**	<u>0.8897</u>
Reviews	0.7329	0.8383	<u>0.9096</u>	**0.9135**
LA1	0.8194	**0.9320**	<u>0.9302</u>	0.9294
MNIST-4K	0.5749	0.7199	**0.8476**	<u>0.8174</u>
PIE-20	0.5382	0.6918	**0.9102**	<u>0.8848</u>
YaleB-38	0.5676	0.7119	**0.8820**	<u>0.8655</u>
		l/n= **5%**		
Reuters-21578	0.6076	0.6593	**0.9603**	<u>0.9548</u>
Reviews	0.7998	0.8774	<u>0.9446</u>	**0.9454**
LA1	0.8546	0.9445	**0.9534**	<u>0.9523</u>
MNIST-4K	0.7131	0.8669	**0.9433**	<u>0.9314</u>
PIE-20	0.6797	0.8492	**0.9792**	<u>0.9735</u>
YaleB-38	0.6937	0.8435	**0.9637**	<u>0.9562</u>

Table 4.4: The run times (in seconds) of the proposed greedy Nyström methods compared to the ensemble Nyström method and Nyström method with uniform sampling (rank-k approximations $\tilde{K}_{\mathcal{S},k}$, $k = 1\%n$).

Data sets	**UniNoRep**	**EnsembleNys**	**GreedyNyström**	**PartGreedyNys**
		l/n= **3%**		
Reuters-21578	1.63	33.29	17.24	4.82
Reviews	0.86	26.02	12.61	2.31
LA1	0.57	18.54	6.41	1.21
MNIST-4K	0.79	23.93	11.98	2.13
PIE-20	0.59	19.72	7.58	1.40
YaleB-38	0.29	11.64	2.89	0.65
		l/n= **5%**		
Reuters-21578	1.73	35.86	28.53	6.79
Reviews	0.87	27.04	21.38	4.08
LA1	0.63	19.52	10.84	1.95
MNIST-4K	0.88	24.83	21.19	3.87
PIE-20	0.57	20.28	12.77	2.35
YaleB-38	0.28	11.98	4.69	0.82

Chapter 5

Fast Approximate Data Clustering

This chapter presents two algorithms for fast approximate data clustering. The algorithms use the Column Subset Selection (CSS) methods presented in Chapters 3 and 4 to first select a set of representative data points, and then embed all data points in the subspace of selected points. The clustering algorithm is then applied to data points in the new space. The algorithms have been shown to achieve clustering performance comparable to their exact counterparts in much less run time.

The chapter is organized as follows: Sections 5.1 and 5.2 present the fast approximate algorithms for k-means and spectral clustering respectively. Section 5.3 gives an overview of related work for developing fast approximate clustering algorithms. Section 5.4 shows different experiments that have been conducted to evaluate the proposed algorithms.

5.1 Fast Approximate k-Means Clustering

The k-means algorithm [22] is the most widely used algorithm for data clustering, whose goal is to minimize the sum of distances between the data points in each class and their centroids. In this section, a fast approximate algorithm is proposed to perform k-means clustering. The algorithm uses the greedy column subset selection algorithm presented in Chapter 3 to select a compact set of representative data points, and then represents all points in the subspace of selected points. The k-means clustering is then conducted in that low-dimensional space. Algorithm 8 shows the steps of the fast approximate k-means.

The following theorem shows that applying the k-means algorithm on the column-based low-dimensional embedding of data points W is equivalent to applying the k-means

Algorithm 8 Fast Approximate k-Means

Input: Data matrix A, Number of clusters k, Number of representative points l
Output: Data partitions: $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$
Steps:

1. Greedily select l data points (i.e., columns) from A, and calculate W whose columns represent the embedding of data points into the subspace of selected points:

$$[\mathcal{S}, W] = \text{GreedyCSS}(A, l)$$

2. Run the k-means algorithm on the columns of W

$$\mathbf{P} = \text{k-Means}(W, k)$$

algorithm on the column-based low-rank approximation of A. It should be noted that this theorem is general for any representation of data points in an orthonormal basis.

Theorem 5.1 *Given a data matrix $A \in \mathbb{R}^{m\times n}$ whose columns represent data points. Let $\tilde{A} = QW$ be a rank-l approximation of A where $Q \in \mathbb{R}^{m\times l}$ is a matrix with orthonormal columns and $W \in \mathbb{R}^{m\times l}$ is the embedding of data points into the basis formed by Q's columns, then applying the Lloyd's algorithm for k-means clustering on the columns of $\tilde{A}$ is equivalent to applying it on the columns of W.*

Proof The Lloyd's algorithm has two steps: (1) the calculation of the new centroids of clusters and, (2) the assignment of data points to the new clusters based on the distances between all data points and each centroid.

In the first step, applying the Lloyd's algorithm to $\tilde{A}$, the centroid of the i-th cluster will be:

$$\boldsymbol{\mu}_{(i)} = \frac{1}{|\mathcal{P}_i|} \sum_{j\in\mathcal{P}_i} \left(\tilde{A}\right)_{:j} = \frac{1}{|\mathcal{P}_i|} \sum_{j\in\mathcal{P}_i} QW_{:j}$$

where $\mathcal{P}_i$ is the set of data points that belong to the i-th class. Let $\boldsymbol{\sigma}_{(i)}$ be the centroid of the i-the cluster if the algorithm is applied to W:

$$\boldsymbol{\sigma}_{(i)} = \frac{1}{|\mathcal{P}_i|} \sum_{j\in\mathcal{P}_i} W_{:j}$$

It can be easily shown that $\boldsymbol{\mu}_{(i)} = Q\boldsymbol{\sigma}_{(i)}$.

In the second step, applying the Lloyd's algorithm to $\tilde{A}$, the distance between the j-th point and the i-th cluster will be:

$$D_{ij}^2 = \left\| \boldsymbol{\mu}_{(i)} - \left(\tilde{A}_{\mathcal{S}}\right)_{:j} \right\|^2$$

This can be simplified to:

$$\begin{aligned} D_{ij}^2 &= \left\| \boldsymbol{\mu}_{(i)} - \left(\tilde{A}_{\mathcal{S}}\right)_{:j} \right\|^2 = \left\| Q\boldsymbol{\sigma}_{(i)} - QW_{:j} \right\|^2 \\ &= trace\left(\left(Q\boldsymbol{\sigma}_{(i)} - QW_{:j}\right)^T \left(Q\boldsymbol{\sigma}_{(i)} - QW_{:j}\right) \right) = \text{trace}\left(\left(\boldsymbol{\sigma}_{(i)} - W_{:j}\right)^T Q^T Q \left(\boldsymbol{\sigma}_{(i)} - W_{:j}\right) \right) \\ &= trace\left(\left(\boldsymbol{\sigma}_{(i)} - W_{:j}\right)^T \left(\boldsymbol{\sigma}_{(i)} - W_{:j}\right) \right) = \left\| \boldsymbol{\sigma}_{(i)} - W_{:j} \right\|^2 \end{aligned}$$

The right-hand side is equal to the distance between the j-th point and the i-th cluster if the algorithm is applied to W. This means that at each iteration both algorithms will make the same assignment of data points to clusters, and accordingly they are equivalent. ■

The computational complexity of the original Lloyd's algorithm is $O(nmkt)$ where n, m, k, and t are the number of data instances, features, clusters and iterations, respectively. Reducing the dimensions of the data vectors to c columns reduces the computational complexity of the Lloyd's algorithm to $O(nckt)$. Since $c \ll m$, the fast approximate k-means algorithm achieves a considerable reduction in the run time.

5.2 Fast Approximate Spectral Clustering

Spectral clustering [4, 47] is a state-of-the-art algorithm for data clustering that builds a graph over the data points and finds the minimum cut between graph partitions. In this section, a fast approximate algorithm is proposed for spectral clustering. The algorithm uses the greedy Nyström method presented in Chapter 4 to approximate the leading eigenvectors of the graph Laplacian matrix. The k-means clustering algorithm is then applied to the leading eigenvectors to obtain the final clusters.

Algorithm 9 shows the steps of the algorithm. First, the algorithm uses the greedy Nyström method to greedily select l columns from the graph similarity matrix K. The greedy Nyström also calculates W whose columns represent a low-dimensional embedding

Algorithm 9 Fast Approximate Spectral Clustering

Input: Graph similarity matrix K, Number of clusters k, Number of representatives l
Output: Data partitions: $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, ..\mathcal{P}_k\}$
Steps:

1. Greedily select l data points (i.e., columns) from K, and calculate W whose columns represent the embedding of data points into the subspace of selected points in the high-dimensional space implicitly defined by the kernel:

$$[\mathcal{S}, W] = \text{GreedyNyström}(K, l)$$

2. $\tilde{D} = diag\left(W^T\left(W\boldsymbol{e}\right)\right)$
3. $\bar{W} = W\tilde{D}^{-1/2}$
4. $[U_k,\ \Sigma_k,\ V_k] = svd\left(\bar{W}, k\right)$
5. Run the k-means algorithm on the columns of V_k^T

$$\mathbf{P} = \text{k-Means}(V_k^T, k)$$

of data points in the subspace of selected points in the high-dimensional space implicitly defined by the kernel. The low-rank approximation of the graph similarity matrix K can be expressed as:

$$\tilde{K}_{\mathcal{S}} = W^T W.$$

In order to calculate the Laplacian matrix, the algorithm first approximates the node degree matrix D as:

$$\tilde{D} = diag\left(\tilde{K}_{\mathcal{S}}\boldsymbol{e}\right) = diag\left(W^T(W\boldsymbol{e})\right)$$

where $\boldsymbol{e}$ is an $n \times 1$ all-ones vector, and $diag$ is a function which takes an $n \times 1$ vector and transform it to an $n \times n$ diagonal matrix.

While calculating $\tilde{D}$, there is no need to explicitly calculate $\tilde{K}_{\mathcal{S}}$. The matrix vector multiplications $u = W\boldsymbol{e}$, $W^T u$ can be calculated instead.

The algorithm then calculates $\bar{W}$ as:

$$\bar{W} = W\tilde{D}^{-1/2}$$

Accordingly, the approximate graph Laplacian can be expressed in terms of $\bar{W}$ as:

$$\tilde{L} = \tilde{D}^{-1/2}\tilde{K}_{\mathcal{S}}\tilde{D}^{-1/2} = \tilde{D}^{-1/2}W^T W\tilde{D}^{-1/2} = \bar{W}^T\bar{W}$$

As the eigenvectors of $\tilde{L}$ are equal to the right singular values of $\bar{W}$, the algorithm calculates the leading k right singular of $\bar{W}$ and then apply the k-means clustering on the columns of those singular vectors to obtain the final clustering.

5.3 Related Work

This section gives an overview of different techniques that have been proposed to speed up the k-means and spectral clustering algorithms.

5.3.1 k-Means Clustering

The Lloyd's algorithm [28] is the most commonly used algorithm to optimize the k-means clustering criterion. It is an iterative algorithm whose computational complexity is $O(nmk)$ per iteration where n, m and k are the numbers of data instances, features, and clusters, respectively. This computational complexity is mainly due to the cluster assignment step

in which the distances between all data points and centroids are calculated, and then each data point is assigned to the cluster of the nearest centroid. For large or high-dimensional data sets (i.e., large n or m), the calculation of these distances becomes computationally complex.

Different methods have been proposed to reduce the computational complexity of the assignment step. One category of methods depends on reducing the number of distances that need to be calculated at each iteration. Some of these methods are fast but exact implementations of the Lloyd's algorithm. For instance, Kanungo et al. [101] present an algorithm which first builds a k-d tree for the data points, and then maintains for each node of the tree a set of candidate centers. The use of these candidate centers reduces the number of distances to be calculated at each iteration. This implementation, however, works well when both the dimension of the data and the number of clusters are small. Elkan [102] proposes the use of the triangle inequality to avoid the calculations of unnecessary distances. This method is only suitable for low-dimensional data and it requires $O(k^3)$ extra storage, where k is the number of clusters.

Other methods for fast k-means clustering depend on sub-sampling the data points to reduce the number of distances to be calculated. For instance, Har-Peled and Kushal [103, 104] propose an algorithm which clusters a small portion of the data points, namely a coreset, to obtain a constant factor approximation to the optimal clustering criterion. Frahling and Sohler [105] extends this work by iteratively running the Lloyd's algorithm on coresets while increasing their size. Quite recently, Wang et al. [106] proposes a fast approximate k-means algorithm that depends on identifying a set of active points that change their cluster assignment at each iteration. These active points are identified by grouping neighboring data points using multiple random partition trees [107].

Another category of methods depends on first reducing the dimensionality of the data and then applying the k-means algorithm in the reduced space. Both feature selection and extraction methods can be used for reducing the dimension of the data vectors. This review focuses on feature extraction methods as they are related to the proposed method. A survey of related feature selection methods is a topic of Section 6.3.

Traditional feature extraction techniques, such as Singular Value Decomposition (SVD) [10] and Principal Component Analysis (PCA) [11] have been used to reduce the dimensionality of the data as a preprocessing step for the k-means algorithm [108, 109]. The use of SVD and PCA, however, increases the computational complexity of the clustering algorithm, especially for large or high-dimensional data. The use of fast dimension reduction techniques has also been explored. For instance, Boutsidis et al. [110] propose the use of random projections to reduce the dimensionality of the data before applying the k-means

algorithm. The authors present a theoretical analysis for the approximation error of the k-means objective function. Quite recently, Cardoso and Wichert [111] proposed the idea of applying the k-means algorithm on random projections many times, while increasing the dimensionality of the data points after each convergence of the k-means algorithm.

5.3.2 Spectral Clustering

The spectral clustering algorithm mainly depends on calculating the eigenvalue decomposition of the graph Laplacian matrix. The computational complexity of this eigenvalue decomposition is $O(n^3)$ where n is the number of data points to be clustered.

Some methods have been proposed to directly reduce the computational complexity of the eigenvalue decomposition of the graph Laplacian matrix. For instance, Fowlkes et al. [5] propose the use of the Nyström method to speed up the spectral clustering algorithm. The basic idea is to randomly select a subset of data points and then use the Nyström approximation method to estimate the leading eigenvectors of the graph Laplacian matrix using the eigenvalues of a small similarity matrix defined over the subset of sampled points. The Nyström method has also been used to implement the spectral clustering algorithm on distributed systems [112].

Another category of methods for fast approximate spectral clustering depends on reducing the size of data points. For instance, Shinnou and Sasaki [113] present an algorithm which reduces the size of the similarity matrix by constructing committees of data points. In this algorithm, the k-means algorithm is first applied to the data points and then the data points that are closed to each cluster centroid form a committee. The committees along with the remaining data points are clustered using the traditional spectral clustering algorithms and the members of each committee are assigned to the cluster of its new centroid. Yan et al. [6] present a fast algorithm for spectral clustering which first selects a small number of representative points (or pseudo-points), and then clusters these points using the traditional spectral clustering algorithm. The cluster labels are then propagated to the original points. The authors present two methods for selecting the representative points: the k-means clustering algorithm and the random projection trees. Recently, Chen and Cai [7] presented a Landmark-based Spectral Clustering (LSC) algorithm which first selects a set of representative data points (through random sampling or using the centroids of the k-means algorithm) and then represents other data points as a sparse combination of these representatives. The spectral embedding of data points are then calculated based on the landmark-based representation.

Table 5.1: The properties of the data sets used to evaluate the fast approximate clustering algorithms.

Data set	Type	# Instances	# Features	# Classes
TDT2-30	Documents	9394	19677	30
20NG	Documents	18774	29360	20
MNIST-70K	Digit Images	70000	784	10
PIE-68	Face Images	11554	4096	68

5.4 Experiments and Results

Experiments have been conducted on four benchmark data sets, whose properties are summarized in Table 5.1.[1] The data sets have been used in previous work to evaluate different clustering and classification tasks. The *TDT2-35* data set is a subset of the NIST Topic Detection and Tracking corpus [114] which consists of the top 30 categories. The *20NG* data set is the 20 newsgroups data.[2] The *MNIST-70K* is the MNIST data set of handwritten digits.[3] The *PIE-68* is a pre-processed subset of the of the CMU PIE [79] data set.

The data sets were preprocessed as follows. For document data sets (*TDT2-35* and *20NG*), the terms that appear in less than five documents were removed and the normalized term frequency - inverse document frequency (tf-idf) weighting scheme was used to encode the importance of terms inside documents. For image data sets (*MNIST-70K* and *PIE-68*), the intensity values of each image were scaled to lie in the range [0, 1].

Two sets of experiments were conducted to evaluate the fast approximate k-means and spectral clustering algorithms. For all experiments, after the clustering is performed using different methods, the cluster labels are compared to ground-truth labels provided by human annotators and the Normalized Mutual Information (NMI) [61] between clustering labels and the class labels is calculated. In addition to clustering performance, the run times of different clustering algorithms are compared. The rest of this section describes the details of different experiments.

[1] Data sets are available in MATLAB format at:
http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html
http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html
http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html

[2] http://people.csail.mit.edu/jrennie/20Newsgroups/

[3] http://yann.lecun.com/exdb/mnist

5.4.1 Fast Approximate k-Means Clustering

In this set of experiments, the fast approximate k-means algorithm presented in Section 5.1 is evaluated. The focus of this empirical validation is on algorithms that use feature extraction to reduce the dimension of the data before applying the traditional Lloyd's algorithm. The following representations of the data points are compared.

- **Data**: is the original data matrix.
- **SSVD**: is the low-dimensional embedding obtained by applying the Stochastic SVD [115] algorithm to the data matrix to obtain the leading l singular vectors. The SSVD algorithm is the state-of-the-art algorithm in approximating singular values and vectors. In all the conducted experiments, the number of iterations and the oversampling parameter are set to 2 and 10 respectively.
- **RandCSS**: is the low-dimensional embedding obtained by applying the CSS method to select l random columns (Eq. 3.3 on page 29). This method has been chosen to study how the proposed greedy CSS compares to random selection of columns.
- **PGreedyCSS**: is the low-dimensional embedding obtained by applying the partition-based greedy CSS algorithm to select l columns.
- **PGreedyCSS-Hybrid-r=5**: is the low-dimensional embedding obtained by applying a hybrid variant of the partition-based greedy algorithm in which $rl \log(l)$ columns are first selected from the data matrix using uniform sampling, and then the partition-based greedy algorithm is applied to the reduced matrix to select l columns. For this method, r is set to 5.
- **PGreedyCSS-Hybrid-r=1**: is the same as the previous method with $r = 1$.

For the partition-based greedy CSS algorithms, the number of partitions used in all experiments is 10.

In each experiment, the Lloyd's algorithm is repeated five times with different initial centroids and the solution with the minimum objective function is selected. All experiments are repeated ten times and the average performance measures are calculated.

Figure 5.1 shows the average NMI measures and run times of the fast approximate k-means algorithm with different feature extraction methods. Table 5.2 also shows the average NMI measures for the best performing methods (**SSVD**, **PGreedyCSS**, and **PGreedyCSS-Hybrid-r=5**). Each sub-table represents a data set and each column

represents a percentage of selected columns. The NMI measures in each sub-column are divided into groups according to their statistical significance. The best group of methods is highlighted in bold, and the second best group is underlined. The statistical significance tests were performed as explained in Section 3.5 on page 50.

It can be observed from the figure that the **PGreedyCSS** and **PGreedyCSS-Hybrid** embedding achieves better clustering performance than the **RandCSS** for the data sets with large numbers of features (*TDT2-30* and *20NG*), and comparable clustering performance for the data sets with small numbers of features (*MNIST-70K* and *PIE-68*). The **SSVD** method achieves better clustering performance than all other methods, and it approaches the clustering performance of the k-means algorithm on the original data matrix. However, the run times of the **SSVD** method are much longer than other methods, especially as the number of selected columns increases. On the other hand, the run times of the **PGreedyCSS-Hybrid** methods are comparable to that of the **RandCSS** method. It can also be observed that increasing the parameter r of the **PGreedyCSS-Hybrid** method increases the clustering performance, but also increases the run time of the method. This parameter can be effectively used to control the trade-off between the clustering performance of the **PGreedyCSS-Hybrid** method and its run time. It should also be noted that although the **SSVD** method achieves good clustering performance using very few dimensions, it is hard for the data analysts to understand the meaning of these dimensions. On the other hand, the dimensions obtained by the CSS methods directly represent data instances, which can be easily presented to the analyst. The representation of cluster centroids in the space of few data instances is more appealing for data analysts, and it allows them to learn about the contents of each data cluster.

5.4.2 Fast Approximate Spectral Clustering

In this set of experiments, the proposed fast approximate spectral clustering algorithm presented in Section 5.2 is evaluated. The following algorithms for approximate spectral clustering are compared:[4]

- **NyströmSC**: is the spectral clustering algorithm using the Nyström extension proposed by Fowlkes et al. [5]. The MATLAB implementation by Chen et al. [112] is used.[5]

[4]The fast SC method by Yan et al. [6] was not included in the comparison as applying the k-means algorithm to the data matrices requires very long run times compared to the run times of other fast SC algorithms (as shown in Figure 5.1).

[5]http://alumni.cs.ucsb.edu/~wychen/download/spectralclustering-1.1.zip

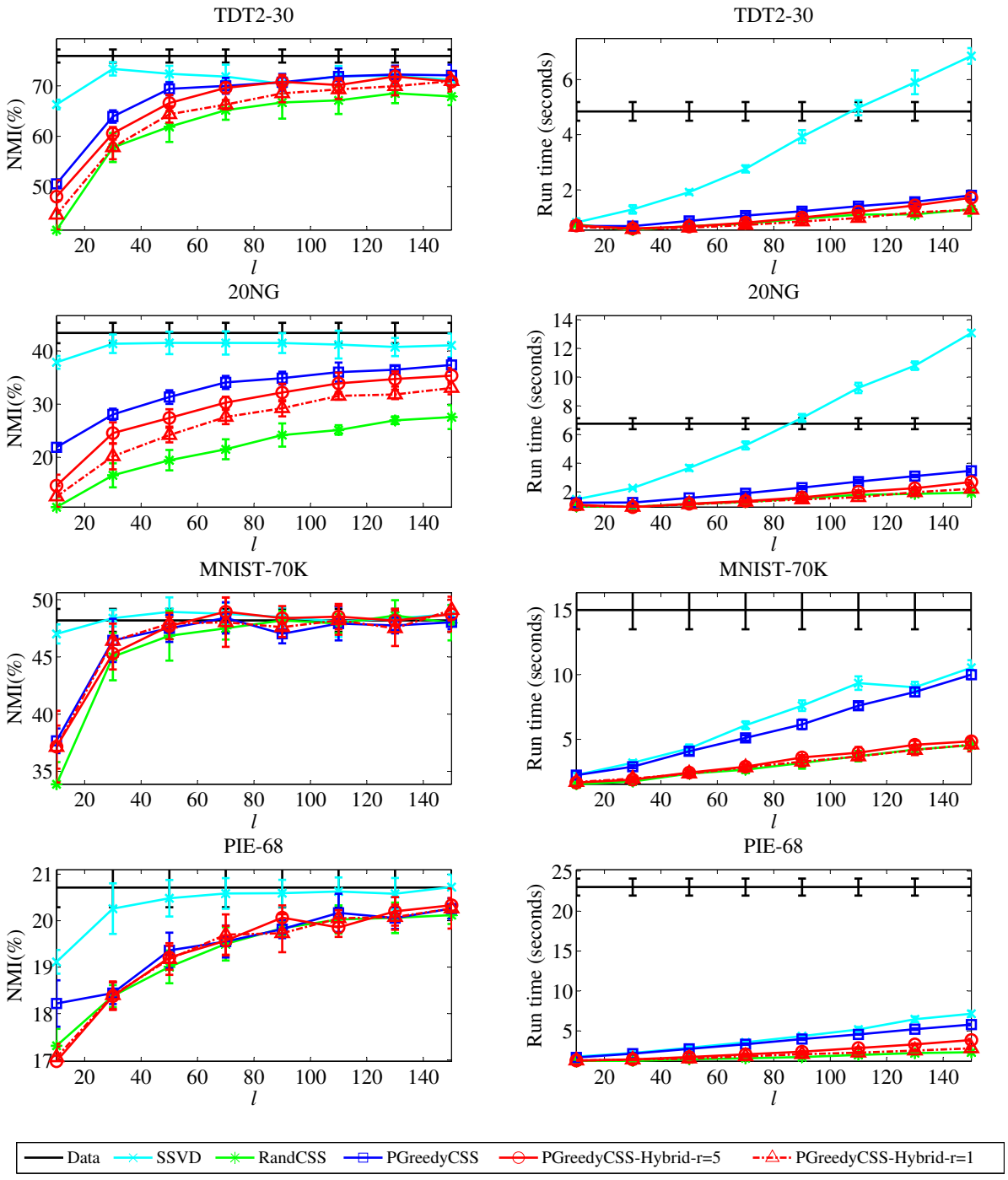

Figure 5.1: The clustering performance measures and run times of the fast k-means algorithms.

- **LSC**: is the Landmark-based Spectral Clustering (LSC) method proposed by Chen and Cai [7] with the landmarks selected using random sampling. The MATLAB implementation provided by the authors is used.[6]
- **PGreedyNysSC**: is the fast approximate spectral clustering method of Algorithm 9 in which the partition-based greedy Nyström algorithm is used to select the columns.
- **PGreedyNysSC-Hybrid-r=5**: is the fast approximate spectral clustering method with a hybrid variant of the partition-based greedy Nyström algorithm in which $rl \log(l)$ columns are first selected from the kernel matrix using uniform sampling, and then the partition-based greedy algorithm is applied to the reduced matrix to select l columns. For this method, r is set to 5.
- **PGreedyNysSC-Hybrid-r=1**: is the same as the previous method with $r = 1$.

For the partition-based greedy Nyström algorithms, the number of partitions used in all experiments is 10.

Two sets of experiments are conducted on linear and Gaussian kernels. For Gaussian kernels, the σ parameter is estimated for each pair of data points as:

$$\sigma_{ij}^2 = \left(\sum_k D_{ik}\right)\left(\sum_k D_{jk}\right),$$

where D is an $n \times n$ matrix of Euclidean distances between data instances. This is a variant of the Gaussian kernel used with self-tuning spectral clustering [116] which has been shown to achieve better clustering performance than manually tuned kernel parameters. This variant was used by Chen et al. [112] in their work on parallel spectral clustering.

Figures 5.2 and 5.3 show the NMI measures and run times of different fast approximate spectral clustering algorithms. Table 5.2 also shows the average NMI measures for the best performing methods (**NyströmSC**, **PGreedyNysSC**, and **PGreedyNysSC-Hybrid-r=5**). Each sub-table represents a data set and each column represents a percentage of selected columns. The NMI measures in each sub-column are divided into groups according to their statistical significance. The best group of methods is highlighted in bold, and the second best group is underlined. The statistical significance tests were performed as explained in Section 3.5 on page 50.

It can be observed from the figure that the proposed **PGreedyNysSC** methods achieve better clustering performance than the **NyströmSC** methods for the for the data sets with

[6] http://www.cad.zju.edu.cn/home/dengcai/Data/code/LSC.m

Table 5.2: The clustering performance measures of the best performing fast k-means algorithms. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

TDT2-30				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
SVD	**73.38±01.32**	**71.84±02.35**	**71.94±02.02**	**71.19±01.63**
PGreedyCSS	63.92±01.23	70.01±01.58	**71.88±01.60**	**72.09±02.10**
PGrCSS-Hybrid-r=5	60.59±01.24	69.62±00.99	70.15±01.41	**70.83±01.93**
20NG				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
SVD	**41.36±01.74**	**41.51±02.19**	**41.20±02.61**	**41.06±02.22**
PGreedyCSS	28.07±01.09	34.08±01.23	36.02±01.79	37.36±01.26
PGrCSS-Hybrid-r=5	24.55±02.00	30.28±01.11	33.91±01.99	35.35±01.46
MNIST-70K				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
SVD	**48.38±00.73**	**48.77±00.95**	**48.08±01.33**	**48.73±00.83**
PGreedyCSS	46.44±01.89	**48.41±01.36**	**47.91±01.47**	**48.03±00.61**
PGrCSS-Hybrid-r=5	45.29±01.39	**48.95±01.26**	**48.52±01.10**	**48.73±01.54**
PIE-68				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
SVD	**20.26±00.54**	**20.59±00.33**	**20.63±00.30**	**20.72±00.27**
PGreedyCSS	18.44±00.23	19.55±00.34	20.17±00.41	20.26±00.22
PGrCSS-Hybrid-r=5	18.38±00.30	19.57±00.32	19.85±00.20	20.33±00.15

large numbers of features (*TDT2-30* and *20NG*), and comparable clustering performance for the data sets with small numbers of features (*MNIST-70K* and *PIE-68*). The run times of the **PGreedyCSS-Hybrid** methods are comparable to that of the **NyströmSC** method and the r parameter can be used to control the trade-off between the clustering performance and the run time. On the other hand, the clustering performance of the **LSC** method is worse than other methods, except for the *MNIST-70K* data set with large numbers of selected columns. The run times of the **LSC** method are also longer than that of the Nyström-based methods.

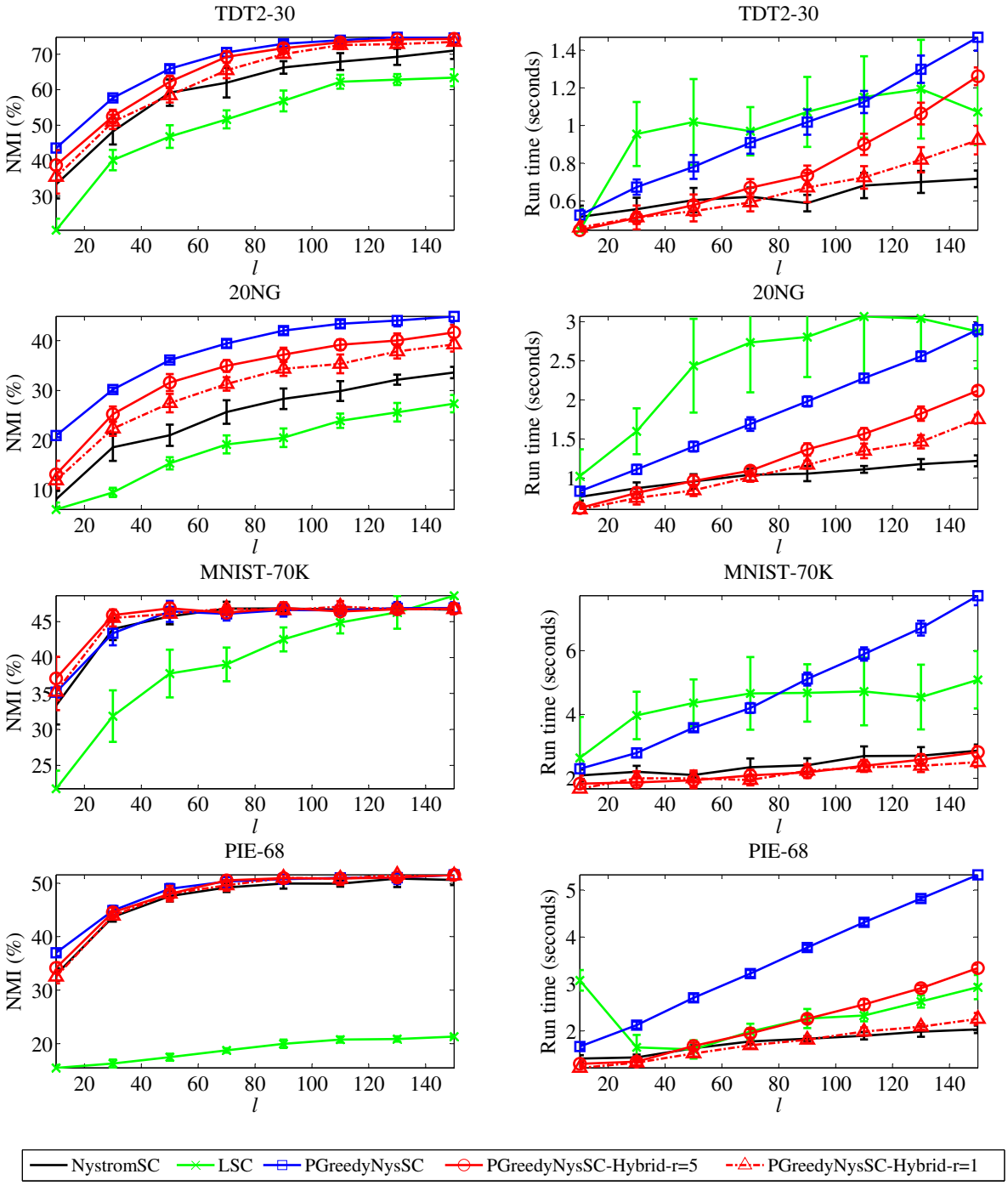

Figure 5.2: The clustering performance measures and run times of the fast spectral clustering algorithms using linear kernels.

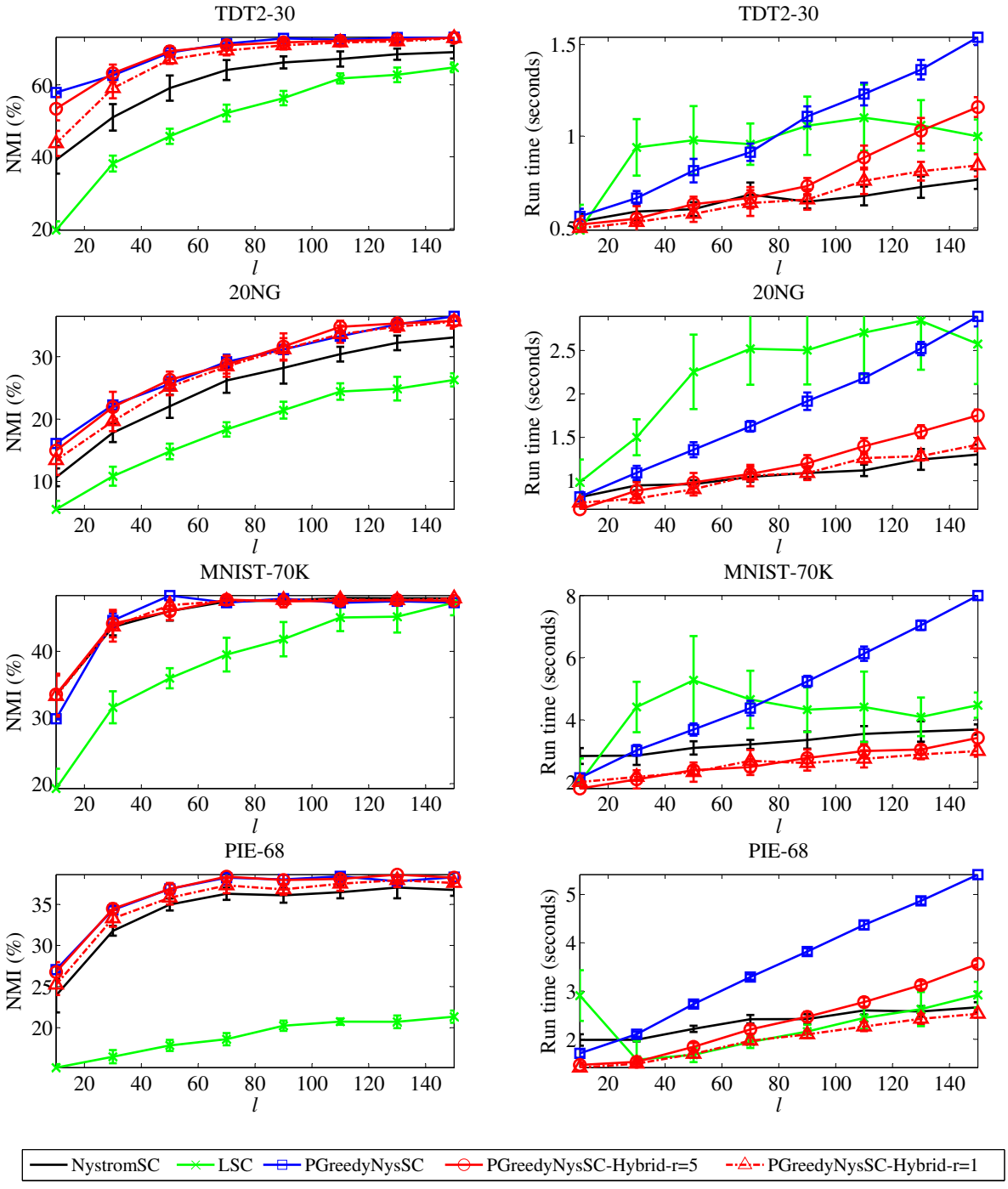

Figure 5.3: The clustering performance measures and run times of the fast spectral clustering algorithms using Gaussian kernels.

Table 5.3: The clustering performance measures of the best performing fast spectral clustering algorithms. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

TDT2-30				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
NyströmSC	48.25±03.69	61.99±04.18	67.93±02.39	71.03±02.35
PGreedyNysSC	**57.70±00.59**	**70.49±01.21**	**73.95±01.03**	**74.60±01.50**
PGrNysSC-Hybrid-r=5	52.46±01.89	69.29±01.57	**73.35±01.06**	**74.32±01.24**
20NG				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
NyströmSC	18.58±02.72	25.67±02.37	29.89±01.99	33.61±01.13
PGreedyNysSC	**30.19±00.41**	**39.44±00.79**	**43.38±00.78**	**44.89±01.21**
PGrNysSC-Hybrid-r=5	25.24±01.55	34.94±01.18	39.19±01.05	41.65±01.51
MNIST-70K				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
NyströmSC	43.97±01.54	**46.82±00.93**	**46.69±00.55**	**46.66±00.50**
PGreedyNysSC	43.38±01.66	46.10±00.94	**46.57±00.23**	**46.90±00.81**
PGrNysSC-Hybrid-r=5	**45.91±00.70**	46.25±00.33	**46.42±00.37**	**46.81±00.78**
PIE-68				
Method	$l = 30$	$l = 70$	$l = 110$	$l = 150$
NyströmSC	43.70±00.89	49.20±00.85	49.92±00.57	50.59±00.95
PGreedyNysSC	**44.87±00.90**	**50.30±01.19**	**51.01±01.22**	**51.57±00.75**
PGrNysSC-Hybrid-r=5	**44.57±00.86**	**50.52±00.96**	**50.86±00.91**	**51.54±00.94**

Chapter 6

Greedy Unsupervised Feature Selection

This chapter presents an effective filter method for unsupervised feature selection. The method is an application of the greedy column subset selection method presented in Chapter 3. The proposed method has been applied to different benchmark data sets and compared to different state-of-the-art algorithms for unsupervised feature selection.

The chapter is organized as follows: Section 6.1 introduces the unsupervised feature selection problem. Section 6.2 presents the greedy feature selection approach. Section 6.3 discusses previous work on filter methods for unsupervised feature selection. Section 6.4 presents an empirical evaluation of the proposed method.

6.1 Unsupervised Feature Selection

Data instances are typically described by a huge number of features. Most of these features are either redundant, or irrelevant to the data mining task at hand. Having a large number of redundant and irrelevant features negatively affects the performance of the underlying learning algorithms, and makes them more computationally demanding. Therefore, reducing the dimensionality of the data is a fundamental task for machine learning and data mining applications.

Throughout past years, two approaches have been proposed for dimension reduction; feature selection, and feature extraction. Feature selection (also known as variable selection

or subset selection) searches for a relevant subset of existing features, while feature extraction (also known as feature transformation) learns a new set of features which combines existing features. These methods have been employed with both supervised and unsupervised learning, where in the case of supervised learning class labels are used to guide the selection or extraction of features.

Feature extraction methods produce a set of continuous vectors which represent data instances in the space of the extracted features. Accordingly, most of these methods obtain unique solutions in polynomial time, which make these methods more attractive in terms of computational complexity. On the other hand, feature selection is a combinatorial optimization problem which is too difficult to be solved optimally, and most feature selection methods depend on heuristics to obtain a subset of relevant features in a manageable time. Nevertheless, feature extraction methods usually produce features which are difficult to interpret, and accordingly feature selection is more appealing in applications where understanding the meaning of features is crucial for data analysis.

Feature selection methods can be categorized into wrapper and filter methods. Wrapper methods wrap feature selection around the learning process and search for features which enhance the performance of the learning task. Filter methods, on the other hand, analyze the intrinsic properties of the data, and select highly-ranked features according to some criterion before doing the learning task. Wrapper methods are computationally more complex than filter methods as they depend on deploying the learning models many times until a subset of relevant features are found.

This chapter presents an effective filter method for unsupervised feature selection. The method is an application of the greedy Column Subset Selection (CSS) method presented in Chapter 3. The criterion used for feature selection measures the reconstruction error of the data matrix based on the subset of selected features. This criterion can be calculated in a recursive manner which allows the development of an efficient greedy algorithm for feature selection. The greedy algorithm selects at each iteration the most representative feature among the remaining features, and then eliminates the effect of the selected features from the data matrix. This step makes it less likely for the algorithm to select features that are similar to previously selected features, which accordingly reduces the redundancy between the selected features. In addition, the use of the recursive criterion makes the algorithm computationally feasible and memory efficient compared to the state of the art methods for unsupervised feature selection.

6.2 Greedy Feature Selection

In this section, the unsupervised feature selection problem is formulated as a column subset selection problem and the algorithms presented in Chapter 3 are used to select features.

Definition 6.1 (Unsupervised Feature Selection Criterion) *Let A be an $m \times n$ data matrix whose rows represent the set of data instances and whose columns represent the set of features. The feature selection criterion is defined as:*

$$F(\mathcal{S}) = \|A - P^{(\mathcal{S})}A\|_F^2$$

where $\mathcal{S}$ is the set of the indices of selected features, and $P^{(\mathcal{S})}$ is an $m \times m$ projection matrix which projects the columns of A onto the span of the set $\mathcal{S}$ of columns.

The criterion $F(\mathcal{S})$ represents the sum of squared errors between the original data matrix A and its rank-l approximation based on the selected set of features (where $l = |\mathcal{S}|$):

$$\tilde{A}_{\mathcal{S}} = P^{(\mathcal{S})}A.$$

The goal of the feature selection algorithm presented in this chapter is to select a subset $\mathcal{S}$ of features such that $F(\mathcal{S})$ is minimized.

Problem 6.1 (Unsupervised Feature Selection) *Find a subset of features $\mathcal{L}$ such that,*

$$\mathcal{L} = \arg\min_{\mathcal{S}} F(\mathcal{S}).$$

This problem is an instance of the Column Subset Selection (CSS) problem presented in Chapter 3. In order to select representative features, Algorithms 4 and 5 can be applied to data matrices whose columns represent features.

6.3 Related Work

6.3.1 PCA-based Methods

Many filter methods for unsupervised feature selection depend on the Principal Component Analysis (PCA) method [11] to search for the most representative features. PCA is the best-known method for unsupervised feature extraction which finds directions with maximum

variance in the feature space (namely principal components). The principal components are also those directions that achieve the minimum reconstruction error for the data matrix.

Jolliffe [11] suggests different algorithms to use PCA for unsupervised feature selection. In these algorithms, features are first associated with principal components based on the absolute value of their coefficients, and then features corresponding to the first (or last) principal components are selected (or deleted). This can be done once or recursively (i.e., by first selecting or deleting some features and then recomputing the principal components based on the remaining features).

Similarly, sparse PCA [117], a variant of PCA which produces sparse principal components, can also be used for feature selection. This can be done by selecting for each principal component the subset of features with non-zero coefficients. However, Masaeli et al. [118] showed that these sparse coefficients may be distributed across different features and accordingly are not always useful for feature selection.

Another iterative approach is suggested by Cui and Dy [119], in which the feature that is most correlated with the first principal component is selected, and then other features are projected onto the direction orthogonal to that feature. These steps are repeated until the required number of features are selected. Lu et al. [120] suggests a different PCA-based approach which applies k-means clustering to the principal components, and then selects the features that are close to clusters' centroids.

Boutsidis et al. [121, 122] propose a feature selection method that randomly samples features based on probabilities calculated using the k-leading singular values of the data matrix. In [121], random sampling is used to reduce the number of candidate features, and then the required number of features is selected by applying a complex subset selection algorithm on the reduced matrix. In [122], the authors derive a theoretical guarantee for the error of the k-means clustering when features are selected using random sampling. However, theoretical guarantees for other clustering algorithms were not explored in this work.

Recently, Masaeli et al. [118] propose an algorithm called Convex Principal Feature Selection (CPFS). CPFS formulates feature selection as a convex continuous optimization problem which minimizes the mean-squared-reconstruction error of the data matrix (a PCA-like criterion) with sparsity constraints. This is a quadratic programming problem with linear constraints, which was solved using a projected quasi-Newton method.

6.3.2 Methods Preserving Data Similarity

Another category of unsupervised feature selection methods are based on selecting features that preserve similarities between data instances. Most of these methods first construct a k nearest neighbor graph between data instances, and then select features that preserve the structure of that graph. Examples of these methods include the Laplacian score (LS) [123] and the spectral feature selection method (a.k.a., SPEC) [124].

The Laplacian score (LS) [123] calculates a score for each feature based on the graph Laplacian and degree matrices. This score quantifies how each feature preserves similarity between data instances and their neighbors in the graph. Spectral feature selection [124] extends this idea and presents a general framework for ranking features on a k nearest neighbor graph.

6.3.3 Methods Preserving Data Cluster Structure

Some methods directly select features which preserve the cluster structure of the data. The $Q-\alpha$ algorithm [125] measures the goodness of a subset of features based on the clustering quality (namely cluster coherence) when data is represented using only those features. The authors define a feature weight vector, and propose an iterative algorithm that alternates between calculating the cluster coherence based on current weight vector and estimating a new weight vector that maximizes that coherence. This algorithm converges to a local minimum of the cluster coherence and produces a sparse weight vector that indicates which features should be selected.

Recently, Cai et al. [126] propose an algorithm called Multi-Cluster Feature Selection (MCFS) which selects a subset of features such that the multi-cluster structure of the data is preserved. To achieve that, the authors employ a method similar to spectral clustering [4], which first constructs a k nearest neighbor graph over the data instances, and then solves a generalized eigenproblem over the graph Laplacian and degree matrices. After that, for each eigenvector, an $L1$-regularized regression problem is solved to represent each eigenvector using a sparse combination of features. Features are then assigned scores based on these coefficients, and highly scored features are selected. The authors show experimentally that the MCFS algorithm outperforms Laplacian score (SC) and the $Q-\alpha$ algorithm.

6.3.4 Feature Selection using Feature Similarity

Another well-known approach for unsupervised feature selection is the Feature Selection using Feature Similarity (FSFS) method suggested by Mitra et al. [127]. The FSFS method groups features into clusters and then selects a representative feature for each cluster. To group features, the algorithm starts by calculating pairwise similarities between features, and then it constructs a k nearest neighbor graph over the features. The algorithm then selects the feature with the most compact neighborhood and removes all its neighbors. This process is repeated on the remaining features until all features are either selected or removed. The authors also suggested a new feature similarity measure, namely maximal information compression, which quantifies the minimum amount of information loss when one feature is represented by the other.

6.3.5 Comparison to Previous Work

The greedy feature selection method proposed in this chapter uses a PCA-like criterion which minimizes the reconstruction error of the data matrix based on the selected subset of features. In contrast to traditional PCA-based methods, the proposed algorithm does not calculate the principal components, which is computationally demanding.

Unlike Laplacian score (LS) [123] and its extension [124], the greedy feature selection method does not depend on calculating pairwise similarity between instances. It also does not calculate eigenvalue decomposition over the similarity matrix as the $Q - \alpha$ algorithm [125] and Multi-Cluster Feature Selection (MCFS) [126] do.

The feature selection criterion presented in this chapter is similar to that of Convex Principal Feature Selection (CPFS) [118] as both minimize the reconstruction error of the data matrix. While the method presented here uses a greedy algorithm to minimize a discrete optimization problem, CPFS solves a quadratic programming problem with sparsity constraints. In addition, the number of features selected by the CPFS depends on a regularization parameter λ which is difficult to tune.

Similar to the method proposed by Cui and Dy [119], the method presented in this chapter removes the effect of each selected feature by projecting other features to the direction orthogonal to that selected feature. However, the method proposed by Cui and Dy is computationally very complex, as it requires the calculation of the first principal component for the whole matrix after each iteration.

The Feature Selection using Feature Similarity (FSFS) [127] method employs a similar greedy approach which selects the most representative feature, and then eliminates its

neighbors in the feature similarity graph. The FSFS method, however, depends on a computationally complex measure for calculating similarity between features.

As shown in Section 6.4, experiments on real data sets show that the proposed algorithm outperforms the Feature Selection using Feature Similarity (FSFS) method [127], Laplacian score (SC) [123], and Multi-Cluster Feature Selection (MCFS) [126] when applied with different clustering algorithms.

6.4 Experiments and Results

Experiments have been conducted on six benchmark data sets, whose properties are summarized in Table 6.1.[1] The *ORL* data set consists of 400 face images, and has been used in the face identification task. The *COIL'20* is the Columbia University Image Library which consists of images of different objects. The *ISOLET* is a set of spoken letters, and the USPS consists of handwritten digits. The first four data sets were recently used by Cai et al. [126] to evaluate different feature selection methods in comparison to the Multi-Cluster Feature Selection (MCFS) method. The *TDT2-35* data set is a subset of the NIST Topic Detection and Tracking corpus [114] which consists of top 30 categories. The *20NG* data set is the 20 newsgroups data.[2] The *TDT2-35* and *20NG* data sets have been used in previous work to evaluate different clustering and classification tasks.

The data sets were preprocessed as follows. For image data sets (*ORL*, *COIL20* and *USPS*), the intensity values of each image were scaled to lie in the range [0, 1]. For document data sets (*TDT2-35* and *20NG*), the terms that appear in less than five documents were removed and the normalized term frequency - inverse document frequency (tf-idf) weighting scheme was used to encode the importance of terms inside documents.

In the conducted experiments, seven methods for unsupervised feature selection are compared:[3]

[1] Data sets are available in MATLAB format at:
http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html
http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html
http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html

[2] http://people.csail.mit.edu/jrennie/20Newsgroups/

[3] The following implementations were used:
FSFS: http://www.facweb.iitkgp.ernet.in/~pabitra/paper/fsfs.tar.gz
LS: http://www.cad.zju.edu.cn/home/dengcai/Data/code/LaplacianScore.m
SPEC: http://featureselection.asu.edu/algorithms/fs_uns_spec.zip
MCFS: http://www.cad.zju.edu.cn/home/dengcai/Data/code/MCFS_p.m

Table 6.1: The properties of the data sets used to evaluate different feature selection methods.

Data set	# Instances	# Features	# Classes	Data Types	Feature Types
ORL	400	1024	40	Face images	Pixels
COIL20	1440	1024	20	Object images	Pixels
ISOLET	1560	617	26	Speech signals	Different properties [128]
USPS	9298	256	10	Digit images	Pixels
TDT2-30	9394	19677	30	Documents	Terms
20NG	18774	29360	20	Documents	Terms

- **PCA-LRG**: is a PCA-based method that selects features associated with the first l principal components [11]. It has been shown that by Masaeli et al. [118] that this method achieves a low reconstruction error of the data matrix compared to other PCA-based methods.[4]
- **FSFS**: is the Feature Selection using Feature Similarity [127] method with the maximal information compression as the feature similarity measure.
- **LS**: is the Laplacian score (LS) [123] method.
- **SPEC**: is the spectral feature selection method [124] using all the eigenvectors of the graph Laplacian.
- **MCFS**: is the Multi-Cluster Feature Selection [126] method which has been shown to outperform other methods that preserve the cluster structure of the data.
- **GreedyFS**: is the basic greedy algorithm presented in this chapter (Algorithm 4).
- **PartGreedyFS**: is the partition-based greedy algorithm (Algorithm 5). In the conducted experiments, the number of partitions was set to 1% of the number of features.

For methods that depend on constructing a k-nearest neighbor graph over the data instances (i.e., **LS**, **SPEC**, and **MCFS**), a five-nearest neighbor graph is constructed for

[4]The CPFA method was not included in the comparison as its implementation details were not completely specified in [118].

each data set, and the weights of the graph are calculated as follows:

$$W_{ij} = \exp\left(-\frac{D_{ij}^2}{2\left(\sum_k D_{ik}\right)\left(\sum_k D_{jk}\right)}\right),$$

where D is an $n \times n$ matrix of Euclidean distances between data instances, and W_{ij} is the weight between nodes i and j of the graph. This weighting function is a variant of the Gaussian kernel used with self-tuning spectral clustering [116] which has been shown to achieve better clustering performance than manually tuned kernel parameters. This variant was used by Chen et al. [112] in their work on parallel spectral clustering.

Similar to previous work [123] [126], the feature selection methods were compared based on their performance in clustering tasks. Two clustering algorithms were used to compare different methods:

- The well-known k-means algorithm [129]: For each feature selection method, the k-means algorithm is applied to the rows of the data matrix whose columns are the subset of the selected features. For document data sets, the spherical k-mean algorithm [40] is applied, where the cosine similarity between data points is used instead of the Euclidean distance. Each run of the k-means algorithm is repeated ten times with different initial centroids and the clustering with the minimum objective function is selected.

- The state-of-the-art affinity propagation (AP) algorithm [51]: The distance matrix between data instances is first calculated based on the selected subset of features, and then the AP algorithm is applied to the negative of this distance matrix. The preference vector, which controls the number of clusters, is set to the median of each column of the similarity matrix, as suggested by Frey and Dueck [51].

After the clustering is performed using the subset of selected features, the cluster labels are compared to ground-truth labels provided by human annotators and the Normalized Mutual Information (NMI) [61] between clustering labels and the class labels is calculated. The clustering performance with all features is also calculated and used as a baseline. In addition to clustering performance, the run times of different feature selection methods are compared. This run time includes the time for selecting features only, and not the run time of the clustering algorithm. For all data sets, the number of selected features were changed from 1% to 10% of the total number of features.

Figures 6.1 and 6.2 show the clustering performance for the k-means and affinity propagation (AP) algorithms respectively.[5] Tables 6.2 and 6.3 show the k-means clustering performance for the best performing feature selection methods (**LS**, **MCFS**, **GreedyFS**, and **GreedyFSPart**). Each sub-table represents a data set and each column represents a percentage of selected features. The NMI measures in each sub-column are divided into groups according to their statistical significance. The best group of methods is highlighted in bold, and the second best group is underlined. The statistical significance tests were performed as explained in Section 3.5 on page 50.

It can be observed from Figures 6.1-6.2 and Tables 6.2-6.3 that the greedy feature selection methods (**GreedyFS** and **PartGreedyFS**) outperform the **PCA-LRG**, **FSFS**, **LS**, and **SPEC** methods for almost all data sets. The **GreedyFS** method outperforms **MCFS** for the *COIL20* data set as well as the three large data sets (*USPS*, *TDT2-30* and *20NG*), while its partition-based variant, **PartGreedyFS**, outperforms **MCFS** for the two document data sets (*TDT2-30* and *20NG*) and gives comparable performance for the *COIL20* and *USPS* data sets. The **MCFS** method mostly outperforms the two greedy algorithms for the *ORL* and *ISOLET* data sets.

Figures 6.3 show the run times of different feature selection methods. It can be observed that **FSFS** is computationally more expensive than other methods as it depends on calculating complex similarities between features. The **FSFS** method does not even scale to run on the document data sets. The **MCFS** method, however efficient, is more computationally complex than Laplacian score (**LS**) and the proposed greedy methods. It can be also observed that for data sets with large numbers of instances (like *USPS*, *TDT2-30* and *20NG*), the **MCFS** method becomes very computationally demanding as it depends on solving a generalized eigenproblem over a data similarity matrix, and then solving an $L1$-regularized regression problem for each eigenvector.

Figure 6.4 shows the run times of the **PCA-LRG** and Laplacian score (**LS**) methods in comparison to the proposed greedy methods. It can be observed that the **PCA-LRG** method is computationally more demanding than the proposed greedy methods for the first four data sets, and it does not scale to run on data sets with large numbers of features as it depends on the calculation of the principal components of the data matrix. On the other hand, the **LS** method is computationally efficient relative to greedy methods when the number of data instances is comparable to the number of features. However, the **LS** method becomes very computationally demanding for data sets with very large number of

[5]The implementations of AP and SPEC algorithms do not scale to run on the **USPS** data set, and those of AP, PCA-LRG, FSFS, and SPEC do not scale to run on the **TDT2-30** and **20NG** data sets on the used simulation machines.

data instances (like the *USPS* data set). It can also be observed that the partition-based greedy feature selection (**PartGreedyFS**) is more efficient than the basic greedy feature selection (**GreedyFS**).

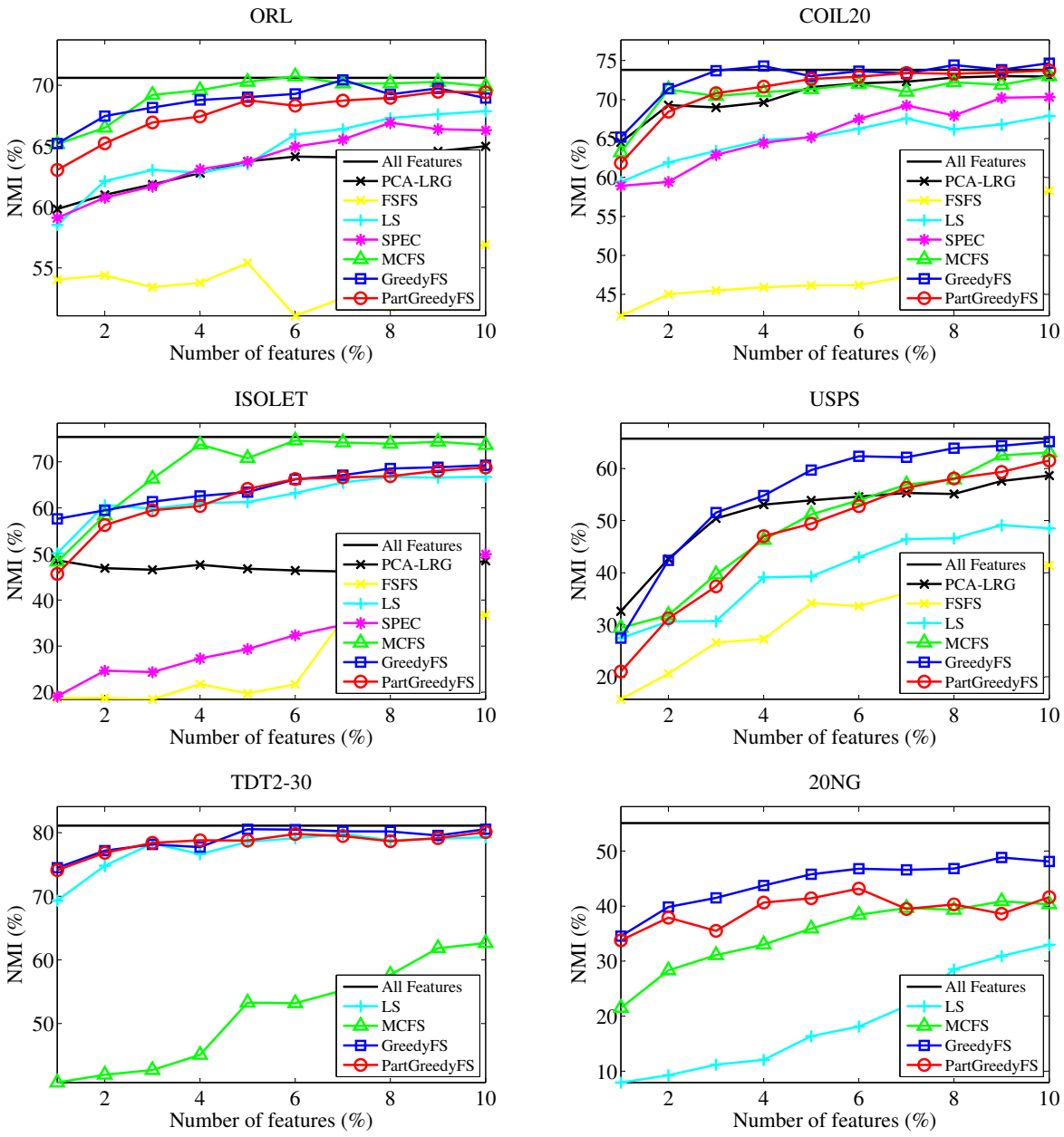

Figure 6.1: The k-means clustering performance of different feature selection methods.

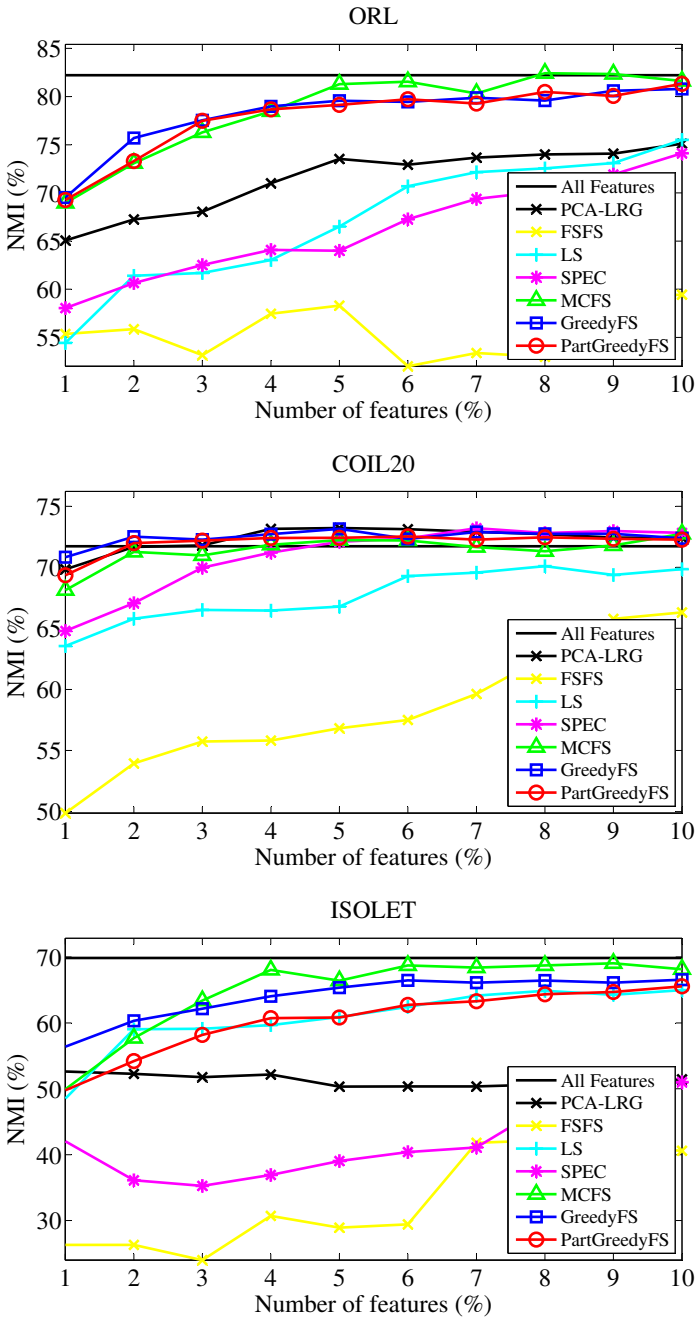

Figure 6.2: The affinity propagation (AP) clustering performance of different feature selection methods for the data sets *ORL*, *COIL20* and *ISOLET*.

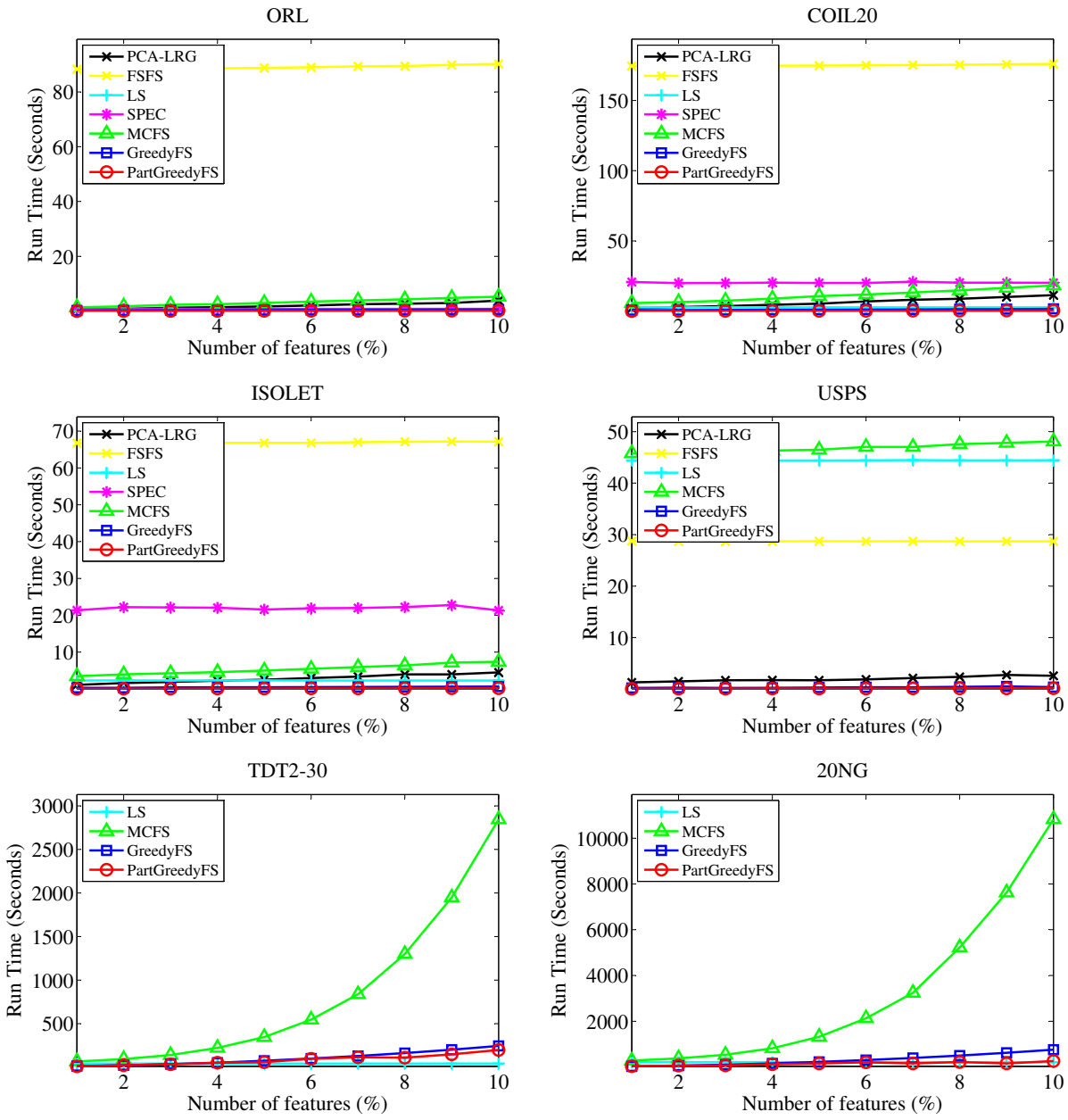

Figure 6.3: The run times of different feature selection methods.

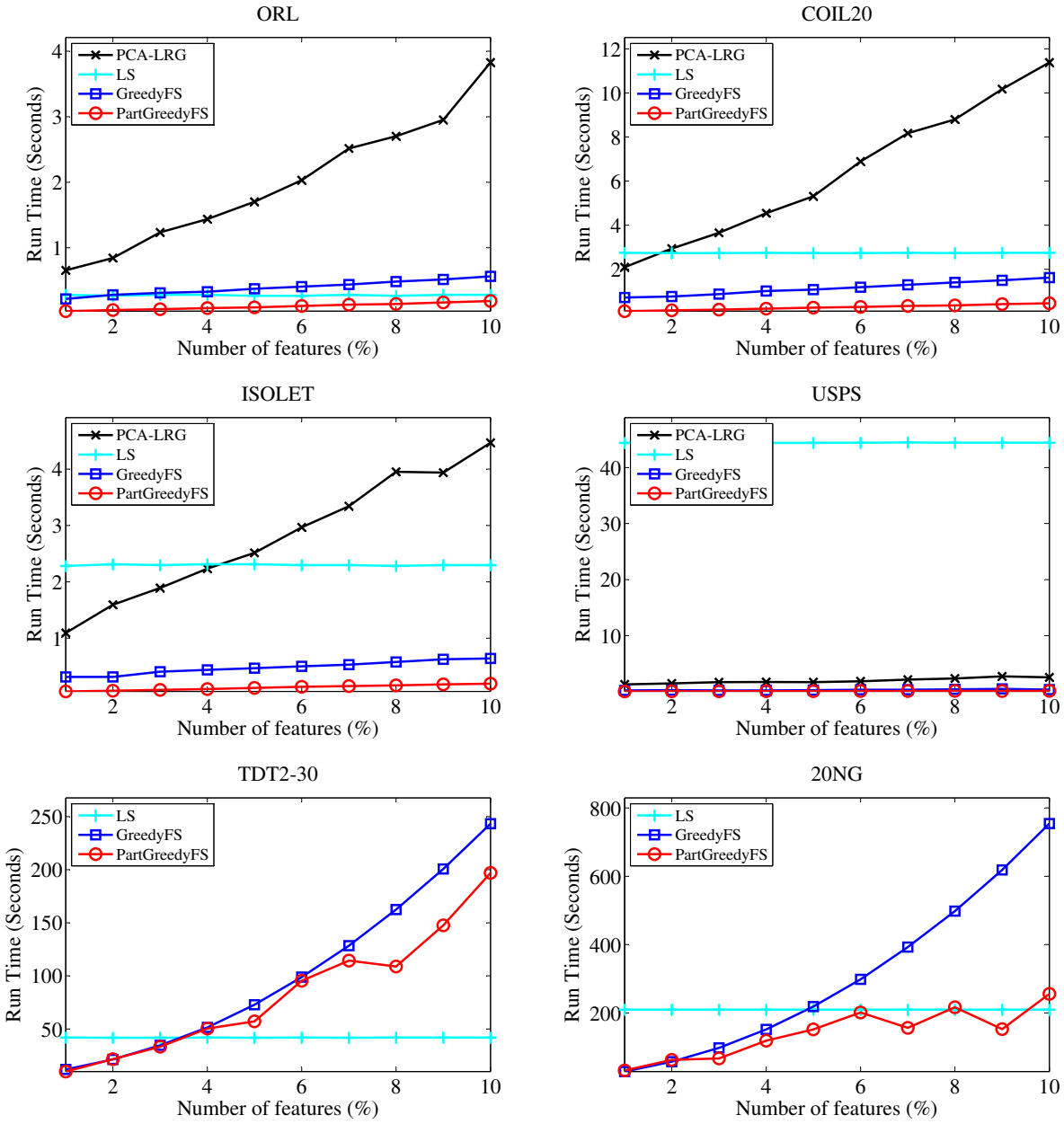

Figure 6.4: The run times of the **PCA-LRG** and **LS** methods in comparison to the proposed greedy algorithms.

Table 6.2: The clustering performance of the k-means algorithm for the top performing methods for the *ORL*, *COIL20*, and *ISOLET* data sets. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

ORL				
Method	$l/n = 1\%$	$l/n = 4\%$	$l/n = 7\%$	$l/n = 10\%$
All Features	70.61±01.51	70.61±01.51	70.61±01.51	70.61±01.51
LS	58.52±00.85	62.83±00.69	66.39±01.21	67.87±01.28
MCFS	**65.17±01.09**	**69.59±01.10**	**70.15±01.45**	**69.93±01.63**
GreedyFS	**65.22±00.74**	68.78±01.42	**70.43±01.64**	**68.96±01.60**
PartGreedyFS	63.05±00.98	67.43±00.84	68.74±00.61	**69.42±00.64**
COIL20				
Method	$l/n = 1\%$	$l/n = 4\%$	$l/n = 7\%$	$l/n = 10\%$
All Features	73.80±02.20	73.80±02.20	73.80±02.20	73.80±02.20
LS	59.44±01.36	64.81±01.57	67.57±01.48	67.90±01.28
MCFS	63.22±01.42	70.94±01.32	71.00±01.48	72.98±01.29
GreedyFS	**65.18±01.91**	**74.30±01.49**	**73.34±02.20**	**74.66±01.43**
PartGreedyFS	61.84±01.98	71.65±00.74	**73.41±00.75**	73.73±00.66
ISOLET				
Method	$l/n = 1\%$	$l/n = 4\%$	$l/n = 7\%$	$l/n = 10\%$
All Features	75.40±01.82	75.40±01.82	75.40±01.82	75.40±01.82
LS	50.13±00.63	61.03±00.68	65.51±00.91	66.75±01.13
MCFS	48.32±00.92	**73.77±01.19**	**74.20±00.88**	**73.68±00.89**
GreedyFS	**57.62±00.81**	62.59±01.43	67.09±01.94	69.24±01.49
PartGreedyFS	45.66±01.75	60.39±03.55	66.64±02.73	68.77±01.84

Table 6.3: The clustering performance of the k-means algorithm for the top performing methods for the *USPS*, *TDT2-30*, and *20NG* data sets. In each sub-column, the best group of methods (according to t-test) is highlighted in bold, and the second best group is underlined.

USPS				
Method	$l/n = 1\%$	$l/n = 4\%$	$l/n = 7\%$	$l/n = 10\%$
All Features	65.73±00.58	65.73±00.58	65.73±00.58	65.73±00.58
LS	27.43±00.14	39.12±00.73	46.47±00.87	48.51±00.74
MCFS	**29.41±00.67**	46.31±01.80	56.91±01.02	63.08±01.27
GreedyFS	27.44±00.59	**54.81±01.04**	**62.15±01.28**	**65.17±00.88**
PartGreedyFS	21.01±01.12	47.02±01.75	56.30±02.35	61.54±01.61
TDT2-30				
Method	$l/n = 1\%$	$l/n = 4\%$	$l/n = 7\%$	$l/n = 10\%$
All Features	81.10±01.65	81.10±01.65	81.10±01.65	81.10±01.65
LS	69.33±01.53	76.63±02.51	**79.79±01.08**	79.26±00.95
MCFS	40.74±00.95	45.07±00.97	55.22±00.88	62.65±01.09
GreedyFS	**74.48±01.34**	77.73±02.07	**80.21±01.88**	**80.55±01.48**
PartGreedyFS	**74.10±00.62**	**78.79±01.15**	**79.47±00.54**	**80.09±00.90**
20NG				
Method	$l/n = 1\%$	$l/n = 4\%$	$l/n = 7\%$	$l/n = 10\%$
All Features	55.08±01.75	55.08±01.75	55.08±01.75	55.08±01.75
LS	07.95±00.45	12.08±00.48	21.85±01.00	32.97±00.81
MCFS	21.50±00.68	33.03±00.87	39.64±01.12	40.33±01.03
GreedyFS	**34.54±02.45**	**43.74±01.43**	**46.58±02.25**	**48.11±01.09**
PartGreedyFS	**33.78±00.49**	40.62±02.45	39.48±04.92	41.60±05.76

Chapter 7

Generalized Greedy CSS

This chapter presents a formulation for a generalized Column Subset Selection (CSS) problem in which a subset of columns of a source matrix is selected, such that the reconstruction error of a target matrix based on the selected columns is minimized. The chapter then presents a greedy algorithm for solving the generalized CSS problem, and describes different problems that can be solved using the proposed algorithm.

The chapter is organized as follows: Section 7.1 defines the generalized column subset selection problem and presents the generalized greedy CSS algorithm, while Section 7.2 describes different problems that can be solved using the proposed algorithm.

7.1 Generalized Column Subset Selection (CSS)

The column subset selection problem is concerned with the selection of a subset of columns from a data matrix which best represent other columns of the same matrix. This problem can be generalized to select a subset of columns from a source matrix which best represent the columns of a different target matrix. The generalized column subset selection problem can be formally defined as follows.

Problem 7.1 (Generalized Column Subset Selection) *Given an $m \times n$ source matrix A, an $m \times q$ target matrix B, and an integer l, find a subset of columns $\mathcal{L}$ from A such that $|\mathcal{L}| = l$ and*

$$\mathcal{L} = \arg\min_{\mathcal{S}} \left\| B - A_{:\mathcal{S}} \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}} \right)^{-1} A_{:\mathcal{S}}^T B \right\|_F ,$$

where $\mathcal{S}$ is the set of the indices of the candidate columns.

Algorithm 10 Generalized Greedy Column Subset Selection

Input: Source matrix A, Target matrix B, Number of columns l
Output: Selected columns $\mathcal{S}$
Steps:

1. Initialize $\boldsymbol{f}_i^{(0)} = \|B^T A_{:i}\|^2$, and $\boldsymbol{g}_i^{(0)} = A_{:i}^T A_{:i}$ for $i = 1,\ 2,\ \ldots n$
2. Repeat $t = 1 \rightarrow l$:
   (a) $p = \arg\max_i \ \boldsymbol{f}_i^{(t)} / \boldsymbol{g}_i^{(t)}, \quad \mathcal{S} = \mathcal{S} \cup \{p\}$
   (b) $\boldsymbol{\delta}^{(t)} = A^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\omega}^{(r)}$
   (c) $\boldsymbol{\gamma}^{(t)} = B^T A_{:p} - \sum_{r=1}^{t-1} \boldsymbol{\omega}_p^{(r)} \boldsymbol{\upsilon}^{(r)}$
   (d) $\boldsymbol{\omega}^{(t)} = \boldsymbol{\delta}^{(t)} / \sqrt{\boldsymbol{\delta}_p^{(t)}}$, $\boldsymbol{\upsilon}^{(t)} = \boldsymbol{\gamma}^{(t)} / \sqrt{\boldsymbol{\delta}_p^{(t)}}$
   (e) Update $\boldsymbol{f}_i$'s, $\boldsymbol{g}_i$'s:

$$\boldsymbol{f}^{(t)} = \left(\boldsymbol{f} - 2 \left(\boldsymbol{\omega} \circ \left(A^T B \boldsymbol{\upsilon} - \Sigma_{r=1}^{t-2} \left(\boldsymbol{\upsilon}^{(r)T} \boldsymbol{\upsilon} \right) \boldsymbol{\omega}^{(r)} \right) \right) + \|\boldsymbol{\upsilon}\|^2 \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$
$$\boldsymbol{g}^{(t)} = \left(\boldsymbol{g} - \left(\boldsymbol{\omega} \circ \boldsymbol{\omega} \right) \right)^{(t-1)},$$

   where $\circ$ represents the Hadamard product operator.

The objective function of Problem 7.1 represents the reconstruction error of the target matrix B based on the selected columns from the source matrix. and the term $A_{:\mathcal{S}} \left(A_{:\mathcal{S}}^T A_{:\mathcal{S}} \right)^{-1} A_{:\mathcal{S}}^T$ is the projection matrix which projects the columns of B onto the subspace of the columns selected from A.

The generalized CSS problem can be solved using a variant of the greedy algorithm presented in Chapter 3. Algorithm 10 shows the steps of the method. The generalized greedy CSS algorithm is a direct extension of the partition-based greedy algorithm. The reader is referred to Section 3.3.2 for the derivations of different steps of the algorithm.

7.2 Generalized CSS Problems

This section describes some of the problems that can be formulated as a generalized column subset selection. It starts by presenting the problems already addressed in this dissertation, and then describes some related problems in the literature. It should be noted that for some of these problems, the use of greedy algorithms has been explored in the literature. However, identifying the connection between these problems and the problems presented in this dissertation gives more insight about those problems, and allows the efficient algorithms presented here to be explored in other interesting domains. The rest of this section draws the connection between the generalized CSS problem and these related problems. The application of the proposed generalized greedy CSS algorithm to these problems is a subject of a future work.

Column Subset Selection. The basic column subset selection is clearly an instance of the generalized CSS problem. In this instance, the target matrix is the same as the source matrix and the goal is to select a subset of columns from a data matrix that best represent other columns. The greedy CSS algorithm presented in Chapter 3 (Algorithm 4) is an instance of Algorithm 10 with $B = A$.

Partition-based Column Subset Selection. The partition-based column subset selection depends on first partitioning the columns of the data matrix into random groups and then selecting a subset of columns from the data matrix that best represent the centroids of these random partitions. The partition-based CSS problem is an instance of the generalized CSS problem in which the columns of the target matrix represent the centroids of random partitions weighted by their sizes. The partition-based greedy CSS algorithm presented in Chapter 3 (Algorithm 5) is an instance of Algorithm 10 with $B_{:j} = \sum_{r \in \mathcal{P}_j} A_{:r}$.

SVD-based Column Subset Selection. Çivril and Magdon-Ismail [72, 73] proposed a column subset selection which first calculates the Singular Value Decomposition (SVD) of the data matrix, and then selects the subset of columns which best approximates the leading singular values of the data matrix. The formulation of this CSS method is an instance of the generalized CSS problem, in which the columns of the target matrix are the embedding of data points in the subspace of the leading singular vectors. The greedy algorithm presented by Çivril and Magdon-Ismail can be implemented using Algorithm 10 by setting $B = U_k \Sigma_k$ where U_k is a matrix whose columns represent the leading left singular vectors of the data matrix, and Σ_k is a matrix whose diagonal elements represent the corresponding singular

Table 7.1: The connection between the generalized column subset selection and related problems.

Method	**Source**	**Target**
Generalized CSS	A	B
Column subset selection (CSS)	Data matrix A	Data matrix $B = A$
Partition-based CSS	Data matrix A	Random partitions $B = \sum_{r \in \mathcal{P}_j} A_{:r}$
SVD-based CSS	Data matrix A	SVD-based embedding $B = U_k \Sigma_k$
Sparse Approximation	Atoms D	Target vector $B = y$

values. This implementation is more efficient than the implementation proposed by Çivril and Magdon-Ismail.

Sparse Approximation. Given a target vector and a set of basis vectors, also called atoms, the goal of sparse approximation is to represent the target vector as a linear combination of a few atoms [130]. Different instances of this problem have been studied in the literature under different names, such as variable selection for linear regression [131], sparse coding [132, 133], and dictionary selection [134, 135]. If the goal is to minimize the error between the target vector and its projection onto the subspace of selected atoms, the sparse approximation can be considered an instance of the generalized CSS problem in which the target matrix is a vector and the columns of the source matrix are the atoms. Several greedy algorithms have been proposed for sparse approximation, such as basic matching pursuit [136], orthogonal matching pursuit [137], orthogonal least square [138], and their kernelized variants [139, 140]. Besides greedy selection, other algorithms for sparse coding include the well-known Lasso method [141]. The generalized greedy CSS algorithm presented in this chapter is closely related to the orthogonal matching pursuit and the orthogonal least square algorithms. Algorithm 10 can be used for sparse approximation by setting $B = y$, where y is the target vector.

Simultaneous Sparse Approximation. A more general problem is the selection of atoms which represent a group of target vectors. This problem is referred to as simultaneous sparse approximation [142]. Different greedy algorithms have been proposed for simultaneous sparse approximation with different constraints [134, 142]. If the goal is to select a subset of atoms to represent different target vectors without imposing sparsity constraints on each representation, simultaneous sparse approximation will be an instance of

the greedy CSS problem, where the source columns are the atoms and the target columns are the input signals.

Table 7.1 summarizes the connection between the generalized column subset selection problem and the related problems described in this section.

Chapter 8

Conclusions and Future Work

This chapter concludes the dissertation and discusses future work.

The chapter is organized as follows: Section 8.1 summarizes the work studied in this dissertation and presents the conclusions of the study. Section 8.2 discusses future extensions of the proposed algorithms.

8.1 Conclusions

This dissertation explores the problem of selecting a subset of representative data instances and/or features from a data matrix. This problem is formally known as Column Subset Selection (CSS).

The dissertation first presented a novel algorithm which greedily selects a subset of columns from a data matrix such that reconstruction error of the data matrix is minimized. The algorithm depends on a novel recursive formula for the reconstruction error of the data matrix, which allows a greedy selection criterion to be calculated efficiently at each iteration. The dissertation then presented a kernelized algorithm to select representative instances/features when only information about their inner-products is available.

The dissertation also presented two interesting applications of the proposed column subset selection algorithms. It first presented two fast approximate algorithms for data clustering, in which column-based embedding of data instances is used to speed up the k-means and spectral clustering respectively. The dissertation then presented a novel algorithm for unsupervised feature selection which depends on the greedy selection of relevant features.

The dissertation finally presented a generalized framework for column subset selection which allows the proposed algorithms to be explored in different domains.

From what has been presented in this dissertation, it can be concluded that the column subset selection is a challenging problem which has many useful applications in data mining and machine learning. These applications include data analysis, low-rank approximation of data matrices, fast approximate data clustering and unsupervised feature selection. The CSS problem can be tackled by developing fast and accurate greedy algorithms. These algorithms can work on data matrices that represent data points in some space, as well as on kernel matrices that represent similarity over data points or features. The greedy algorithms allow the development of fast approximate clustering algorithms as well as fast and accurate unsupervised feature selection algorithms, and they can also be extended to solve a generalized column subset problem in which columns from a source matrix are selected to accurately represent the columns of a target matrix.

Much of the work presented in this dissertation has appeared in peer-reviewed publications. The greedy column subset selection method presented in Chapter 3 and its application to unsupervised feature selection of Chapter 6 are based on the work published in [143, 144]. Also, the greedy Nyström approximation method of Chapter 4 has been published in [145, 146].

8.2 Future Work

The algorithms presented in this dissertation can be extended in different directions and employed in many useful applications. Some of the possible extensions of the current work are:

Column Subset Selection from Big Data. Recent years have witnessed the development of new architectures and programming models to handle big data. In these models, data storage and processing are typically distributed across clusters of computers. The MapReduce framework [147] is an example of such a programming model, which allows the developments of efficient algorithms that work on big data. One of the promising future directions is to develop efficient MapReduce algorithms for selecting representative data instances and/or features from big data. The greedy algorithms presented in this dissertation can be extended to handle big data distributed across thousands of computers. This extension, however, requires the development of an efficient mechanism to simultaneously select groups of columns from different sub-matrices, and then combine the selected groups

in an effective manner to reduce the redundancy between different groups and maximize the coverage of the whole data matrix.

Data Summarization. One of the interesting applications for representative selection is data summarization. The greedy algorithms proposed in this dissertation can be applied to different data summarization problems. For instance, the greedy Nyström can be used to identify the representative keywords in a document corpus by applying it to a kernel matrix over these keywords. The greedy CSS can also be used to extract representative sentences by applying it to a matrix whose columns represent sentences in the space of documents. Moreover, the proposed methods can be used to summarize other types of data based on their contents such as images, speech signals, and biological sequences. The success of the greedy algorithms in data summarization is subject to the design of effective data representations and kernel functions.

Mining Useful Patterns. Extracting useful patterns from big data is one of the interesting problems in data mining. These patterns consist of a set of attribute value pairs. Exploring all possible combinations of pairs is a combinatorial problem. One of the common solutions is to construct patterns in a greedy manner such that the coverage of the data records is maximized. The greedy algorithms presented in this dissertation can be used to extract patterns in a fast and efficient way. The basic idea is to define a kernel function over the patterns and data records and then use the greedy Nyström method to extract patterns that best approximates the data records. The existing methods, however, need to be modified to work on patterns on varying lengths.

Permissions

Parts of Chapters 3 and 6 are reprinted from the following papers.

- A. K. Farahat, A. Ghodsi, and M. S. Kamel, "An efficient greedy method for unsupervised feature selection," in Proceedings of the Eleventh IEEE International Conference on Data Mining (ICDM'11), 2011, pp. 161-170. DOI: 10.1109/ICDM.2011.22. Copyright © 2011 IEEE. The IEEE Thesis/Dissertation Reuse statement is included on page 128.

- A. K. Farahat, A. Ghodsi, and M. S. Kamel, "Efficient greedy feature selection for unsupervised learning," Knowledge and Information Systems, pp. 1-26, 2012, Appeared Online. DOI: 10.1007/s10115-012-0538-1. Copyright © Springer-Verlag London Limited 2012. Reprinted with kind permission from Springer Science and Business Media. The Springer Licence is included on page 129.

SPRINGER LICENSE
TERMS AND CONDITIONS

Jan 18, 2013

This is a License Agreement between Ahmed K Farahat ("You") and Springer ("Springer") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Springer, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	3071900045986
License date	Jan 18, 2013
Licensed content publisher	Springer
Licensed content publication	Knowledge and Information Systems
Licensed content title	Efficient greedy feature selection for unsupervised learning
Licensed content author	Ahmed K. Farahat
Licensed content date	Jan 1, 2012
Type of Use	Thesis/Dissertation
Portion	Excerpts
Author of this Springer article	Yes and you are the sole author of the new work
Order reference number	None
Title of your thesis / dissertation	Greedy Representative Selection for Unsupervised Data Analysis
Expected completion date	Jan 2013
Estimated size(pages)	140
Total	**0.00 CAD**

Terms and Conditions

Introduction

The publisher for this copyrighted material is Springer Science + Business Media. By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at http://myaccount.copyright.com).

Limited License

With reference to your request to reprint in your thesis material on which Springer Science and Business Media control the copyright, permission is granted, free of charge, for the use indicated in your enquiry.

Licenses are for one-time use only with a maximum distribution equal to the number that you identified in the licensing process.

This License includes use in an electronic form, provided its password protected or on the university's intranet or repository, including UMI (according to the definition at the Sherpa website: http://www.sherpa.ac.uk/romeo/). For any other electronic use, please contact Springer at (permissions.dordrecht@springer.com or permissions.heidelberg@springer.com).

The material can only be used for the purpose of defending your thesis, and with a maximum of 100 extra copies in paper.

Although Springer holds copyright to the material and is entitled to negotiate on rights, this license is only valid, provided permission is also obtained from the (co) author (address is given with the article/chapter) and provided it concerns

References

[1] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science and Technology*, vol. 41, no. 6, pp. 391–407, 1990.

[2] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[3] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[4] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems 14 (NIPS'01)*. MIT Press, 2001, pp. 849–856.

[5] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the Nyström method," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 2, pp. 214–225, 2004.

[6] D. Yan, L. Huang, and M. Jordan, "Fast approximate spectral clustering," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. New York, NY, USA: ACM, 2009, pp. 907–916.

[7] X. Chen and D. Cai, "Large scale spectral clustering with landmark-based representation," in *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'11)*. AAAI Press, 2011, pp. 313–318.

[8] I. Fodor, "A survey of dimension reduction techniques," Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Tech. Rep. UCRL-ID-148494, 2002.

[9] X. Su and T. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, vol. 2009, Article ID 421425, 19 pages, 2009.

[10] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed. Johns Hopkins Univ Pr, 1996.

[11] I. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer Verlag, 2002.

[12] N. Halko, P.-G. Martinsson, Y. Shkolnisky, and M. Tygert, "An algorithm for the principal component analysis of large data sets," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2580–2594, 2011.

[13] N. P. Halko, "Randomized methods for computing low-rank approximations of matrices," Ph.D. dissertation, University of Colorado at Boulder, 2012.

[14] T. Chan, "Rank revealing QR factorizations," *Linear Algebra and Its Applications*, vol. 88, pp. 67–82, 1987.

[15] M. Gu and S. C. Eisenstat, "Efficient algorithms for computing a strong rank-revealing QR factorization," *SIAM Journal on Scientific Computing*, vol. 17, no. 4, pp. 848–869, 1996.

[16] C. Bischof and G. Quintana-Ortí, "Computing rank-revealing QR factorizations of dense matrices," *ACM Transactions on Mathematical Software (TOMS)*, vol. 24, no. 2, pp. 226–253, 1998.

[17] C. Pan, "On the existence and computation of rank-revealing LU factorizations," *Linear Algebra and its Applications*, vol. 316, no. 1, pp. 199–222, 2000.

[18] D. Cutting, D. Karger, J. Pedersen, and J. Tukey, "Scatter/gather: A cluster-based approach to browsing large document collections," in *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 1992, pp. 318–329.

[19] K. McKeown, R. Barzilay, D. Evans, V. Hatzivassiloglou, J. Klavans, A. Nenkova, C. Sable, B. Schiffman, and S. Sigelman, "Tracking and summarizing news on a daily basis with Columbia's Newsblaster," in *Proceedings of the Second International Conference on Human Language Technology Research*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2002, pp. 280–285.

[20] O. Zamir and O. Etzioni, "Grouper: A dynamic clustering interface to Web search results," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 31, no. 11, pp. 1361–1374, 1999.

[21] K. Zhang, I. Tsang, and J. Kwok, "Improved Nyström low-rank approximation and error analysis," in *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*. New York, NY, USA: ACM, 2008, pp. 1232–1239.

[22] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[23] K. Hammouda and M. Kamel, "Hierarchically distributed peer-to-peer document clustering and cluster summarization," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 5, pp. 681–698, 2009.

[24] R. Kashef and M. Kamel, "Cooperative clustering," *Pattern Recognition*, vol. 43, no. 6, pp. 2315–2329, 2010.

[25] K. Hammouda and M. Kamel, "Models of distributed data clustering in peer-to-peer environments," *Knowledge and Information Systems*, pp. 1–27, 2012, Appeared Online.

[26] P. Sneath and R. Sokal, *Numerical Taxonomy*. Springer, 1973.

[27] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of Euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, no. 2, pp. 245–248, 2009.

[28] S. Lloyd, "Least squares quantization in PCM," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[29] B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*. New York, NY, USA: ACM, 1999, pp. 16–22.

[30] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining*, 2000.

[31] Y. Zhao and G. Karypis, "Comparison of agglomerative and partitional document clustering algorithms," University of Minnesota, Department of Computer Science, Tech. Rep. 02-014, 2002.

[32] ——, "Empirical and theoretical comparisons of selected criterion functions for document clustering," *Machine Learning*, vol. 55, no. 3, pp. 311–331, 2004.

[33] A. Smola, A. Gretton, and K. Borgwardt, "A dependence maximization view of clustering," in *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*. New York, NY, USA: ACM, 2007, pp. 815–822.

[34] L. Kaufman and P. Rousseeuw, "Clustering by means of medoids," Technische Hogeschool, Delft (Netherlands). Deptartment of Mathematics and Informatics., Tech. Rep., 1987.

[35] I. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means, spectral clustering and normalized cuts," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*. New York, NY, USA: ACM, 2004, pp. 551–556.

[36] ——, "A unified view of kernel k-means, spectral clustering and graph cuts," *The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-04-25*, 2005.

[37] J. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Cybernetics and Systems*, vol. 3, no. 3, pp. 32–57, 1973.

[38] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1981.

[39] J. Bezdek and R. Ehrlich, "FCM: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2, pp. 191–203, 1984.

[40] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine Learning*, vol. 42, no. 1/2, pp. 143–175, 2001.

[41] E. M. Rasmussen, "Clustering algorithms," in *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, Inc., 1992, pp. 419–442.

[42] I. Dhillon, J. Fan, and Y. Guan, "Efficient clustering of very large document collections," in *Data mining for scientific and engineering applications*. Kluwer Academic Publishers, 2001, pp. 357–381.

[43] I. Dhillon, Y. Guan, and J. Kogan, "Iterative clustering of high dimensional text data augmented by local search," in *Proceedings of the Second IEEE International Conference on Data Mining (ICDM'02)*. IEEE, 2002, pp. 131–138.

[44] ——, "Refining clusters in high-dimensional text data," in *Proceedings of the Workshop on Clustering High Dimensional Data and its Applications at the Second SIAM International Conference on Data Mining (SDM'02)*, 2002, pp. 71–82.

[45] I. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*. New York, NY, USA: ACM, 2001, pp. 269–274.

[46] L. Hagen and A. Kahng, "New spectral methods for ratio cut partitioning and clustering," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, no. 9, pp. 1074–1085, 1992.

[47] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000.

[48] C. Ding, X. He, H. Zha, M. Gu, and H. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proceedings of the First IEEE International Conference on Data Mining (ICDM'01) 2001*, 2001, pp. 107–114.

[49] M. Meila and J. Shi, "A random walks view of spectral segmentation," *Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics (AISTATS'01)*, 2001.

[50] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[51] B. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, p. 972, 2007.

[52] K. Wang, J. Zhang, D. Li, X. Zhang, and T. Guo, "Adaptive affinity propagation clustering," *CoRR*, vol. abs/0805.1096, 2008.

[53] C. Sun, C. Wang, S. Song, and Y. Wang, "A local approach of adaptive affinity propagation clustering for large scale data," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'09)*. IEEE, 2009, pp. 2998 –3002.

[54] Y. Fujiwara, G. Irie, and T. Kitahara, "Fast algorithm for affinity propagation," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*. AAAI Press, 2011, pp. 2238–2243.

[55] F. Shang, L. Jiao, J. Shi, F. Wang, and M. Gong, "Fast affinity propagation clustering: A multilevel approach," *Pattern recognition*, vol. 45, no. 1, pp. 474–486, 2012.

[56] J. Bezdek and N. Pal, "Some new indexes of cluster validity," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 28, no. 3, pp. 301–315, 1998.

[57] U. Maulik and S. Bandyopadhyay, "Performance evaluation of some clustering algorithms and validity indices," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 12, pp. 1650–1654, 2002.

[58] M. Pakhira, S. Bandyopadhyay, and U. Maulik, "Validity index for crisp and fuzzy clusters," *Pattern Recognition*, vol. 37, no. 3, pp. 487–501, 2004.

[59] C. Manning, P. Raghavan, and H. Schtze, *Introduction to information retrieval*. Cambridge University Press New York, NY, USA, 2008.

[60] Y. Zhao and G. Karypis, "Hierarchical clustering algorithms for document datasets," *Data Mining and Knowledge Discovery*, vol. 10, no. 2, pp. 141–168, 2005.

[61] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *Journal on Machine Learning Research*, vol. 3, pp. 583–617, 2003.

[62] S. Basu, A. Banerjee, and R. Mooney, "Active semi-supervision for pairwise constrained clustering," in *Proceedings of the Fourth SIAM International Conference on Data Mining (SDM'04)*, 2004, pp. 333–344.

[63] C. Boutsidis, J. Sun, and N. Anerousis, "Clustered subset selection and its applications on it service metrics," in *Proceedings of the 17th ACM conference on Information and knowledge management*, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 599–608.

[64] C. Boutsidis, M. W. Mahoney, and P. Drineas, "An improved approximation algorithm for the column subset selection problem," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*. SIAM Philadelphia, PA, USA, 2009, pp. 968–977.

[65] C. Boutsidis, P. Drineas, and M. Magdon-Ismail, "Near optimal column-based matrix reconstruction," in *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS'11)*, 2011, pp. 305 –314.

[66] A. Frieze, R. Kannan, and S. Vempala, "Fast Monte-Carlo algorithms for finding low-rank approximations," in *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS'98)*, 1998, pp. 370 –378.

[67] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay, "Clustering large graphs via the singular value decomposition," *Machine Learning*, vol. 56, pp. 9–33, 2004.

[68] P. Drineas, R. Kannan, and M. Mahoney, "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 158–183, 2007.

[69] P. Drineas, M. Mahoney, and S. Muthukrishnan, "Subspace sampling and relative-error matrix approximation: Column-based methods," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer Berlin / Heidelberg, 2006, pp. 316–326.

[70] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, "Matrix approximation and projective clustering via volume sampling," in *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*. New York, NY, USA: ACM, 2006, pp. 1117–1126.

[71] ——, "Matrix approximation and projective clustering via volume sampling," *Theory of Computing*, vol. 2, no. 1, pp. 225–247, 2006.

[72] A. Çivril and M. Magdon-Ismail, "Deterministic sparse column based matrix reconstruction via greedy approximation of SVD," in *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC'08)*. Springer-Verlag, 2008, pp. 414–423.

[73] ——, "Column subset selection via sparse approximation of SVD," *Theoretical Computer Science*, vol. 421, no. 0, pp. 1 – 14, 2012.

[74] H. Lütkepohl, *Handbook of Matrices*. John Wiley & Sons Inc, 1996.

[75] A. Deshpande and L. Rademacher, "Efficient volume sampling for row/column subset selection," in *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, 2010, pp. 329 –338.

[76] V. Guruswami and A. K. Sinop, "Optimal column-based low-rank matrix reconstruction," in *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*. SIAM, 2012, pp. 1207–1214.

[77] D. Lewis, "Reuters-21578 text categorization test collection distribution 1.0," 1999.

[78] G. Karypis, "CLUTO - a clustering toolkit," University of Minnesota, Department of Computer Science, Tech. Rep. #02-017, 2003.

[79] T. Sim, S. Baker, and M. Bsat, "The CMU pose, illumination, and expression database," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 12, pp. 1615–1618, 2003.

[80] K. Lee, J. Ho, and D. Kriegman, "Acquiring linear subspaces for face recognition under variable lighting," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 5, pp. 684–698, 2005.

[81] X. He, S. Yan, Y. Hu, P. Niyogi, and H. Zhang, "Face recognition using Laplacianfaces," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 3, pp. 328–340, 2005.

[82] C. Boutsidis, M. W. Mahoney, and P. Drineas, "An improved approximation algorithm for the column subset selection problem," *CoRR*, vol. abs/0812.4293, 2008.

[83] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[84] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer Verlag, 2000.

[85] C. Williams and C. Rasmussen, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[86] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13 (NIPS'00)*. MIT Press, 2000, pp. 682–688.

[87] S. Kumar, M. Mohri, and A. Talwalkar, "On sampling-based approximate spectral decomposition," in *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*. New York, NY, USA: ACM, 2009, pp. 553–560.

[88] C. Cortes, M. Mohri, and A. Talwalkar, "On the impact of kernel approximation on learning accuracy," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, 2010, pp. 113–120.

[89] J. Platt, "Fastmap, MetricMap, and Landmark MDS are all Nyström algorithms," in *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS'06)*, 2005, pp. 261–268.

[90] A. Talwalkar, S. Kumar, and H. Rowley, "Large-scale manifold learning," in *Proceedings of the 21st IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*, 2008, pp. 1–8.

[91] P. Drineas and M. W. Mahoney, "On the Nyström Method for approximating a Gram matrix for improved kernel-based learning," *Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[92] S. Kumar, M. Mohri, and A. Talwalkar, "Sampling techniques for the Nyström method," in *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, 2009, pp. 304–311.

[93] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang, "Matrix approximation and projective clustering via volume sampling," in *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*. New York, NY, USA: ACM, 2006, pp. 1117–1126.

[94] A. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proceedings of 17th International Conference on Machine Learning (ICML'00)*. New York, NY, USA: ACM, 2000, pp. 911–918.

[95] S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representations," *Journal of Machine Learning Research*, vol. 2, pp. 243–264, 2002.

[96] O. Alter, P. Brown, and D. Botstein, "Singular value decomposition for genome-wide expression data processing and modeling," *Proceedings of the National Academy of Sciences*, vol. 97, no. 18, pp. 10 101–10 106, 2000.

[97] S. Kumar, M. Mohri, and A. Talwalkar, "Ensemble Nyström Method," in *Advances in Neural Information Processing Systems 22 (NIPS'09)*, 2009, pp. 1060–1068.

[98] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[99] M. Ouimet and Y. Bengio, "Greedy spectral embedding," in *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, 2005, pp. 253–260.

[100] A. Talwalkar, "Matrix Approximation for Large-scale Learning," Ph.D. dissertation, New York University, 2010.

[101] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 881 –892, 2002.

[102] C. Elkan, "Using the triangle inequality to accelerate k-means," in *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, T. Fawcett and N. Mishra, Eds. AAAI Press, 2003, pp. 147–153.

[103] S. Har-Peled and S. Mazumdar, "On coresets for k-means and k-median clustering," in *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04)*. New York, NY, USA: ACM, 2004, pp. 291–300.

[104] S. Har-Peled and A. Kushal, "Smaller coresets for k-median and k-means clustering," *Discrete & Computational Geometry*, vol. 37, no. 1, pp. 3–19, 2007.

[105] G. Frahling and C. Sohler, "A fast k-means implementation using coresets," *International Journal of Computational Geometry & Applications*, vol. 18, no. 6, pp. 605–625, 2008.

[106] J. Wang, J. Wang, Q. Ke, G. Zeng, and S. Li, "Fast approximate k-means via cluster closures," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'12)*. IEEE, 2012, pp. 3037–3044.

[107] N. Verma, S. Kpotufe, and S. Dasgupta, "Which spatial partition trees are adaptive to intrinsic dimension?" in *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligenc (UAI'09)*. Arlington, VA, USA: AUAI Press, 2009, pp. 565–574.

[108] H. Schütze and C. Silverstein, "Projections for efficient document clustering," in *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '97. New York, NY, USA: ACM, 1997, pp. 74–81.

[109] C. H. Q. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the 21st International Conference on Machine Learning (ICML 2004)*, 2004.

[110] C. Boutsidis, A. Zouzias, and P. Drineas, "Random projections for k-means clustering," in *Advances in Neural Information Processing Systems 23 (NIPS'10)*, 2010, pp. 298–306.

[111] Â. Cardoso and A. Wichert, "Iterative random projections for high-dimensional data clustering," *Pattern Recognition Letters*, vol. 33, no. 13, pp. 1749 – 1755, 2012.

[112] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Chang, "Parallel spectral clustering in distributed systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 3, pp. 568 –586, 2011.

[113] H. Shinnou and S. M., "Spectral clustering for a large data set by reducing the similarity matrix size," in *Proceedings of the Sixth International Language Resources and Evaluation (LREC)*, 2008.

[114] C. Cieri, D. Graff, M. Liberman, N. Martey, and S. Strassel, "The TDT-2 text and speech corpus," in *Proceedings of the DARPA Broadcast News Workshop*, 1999, pp. 57–60.

[115] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[116] L. Zelnik-Manor and P. Perona, "Self-tuning spectral clustering," *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pp. 1601–1608, 2004.

[117] H. Zou, T. Hastie, and R. Tibshirani, "Sparse principal component analysis," *J. Comput. Graph. Stat.*, vol. 15, no. 2, pp. 265–286, 2006.

[118] M. Masaeli, Y. Yan, Y. Cui, G. Fung, and J. Dy, "Convex principal feature selection," in *Proceedings of SIAM International Conference on Data Mining (SDM)*, 2010, pp. 619–628.

[119] Y. Cui and J. Dy, "Orthogonal principal feature selection," in *the Sparse Optimization and Variable Selection Workshop at the International Conference on Machine Learning (ICML)*, 2008.

[120] Y. Lu, I. Cohen, X. Zhou, and Q. Tian, "Feature selection using principal feature analysis," in *Proceedings of the 15th International Conference on Multimedia*. New York, NY, USA: ACM, 2007, pp. 301–304.

[121] C. Boutsidis, M. W. Mahoney, and P. Drineas, "Unsupervised feature selection for principal components analysis," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*, Y. Li, B. Liu, and S. Sarawagi, Eds. ACM, 2008, pp. 61–69.

[122] ——, "Unsupervised feature selection for the k-means clustering problem," in *Advances in Neural Information Processing Systems 22 (NIPS'09)*, 2009, pp. 153–161.

[123] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in Neural Information Processing Systems 18 (NIPS'05)*. Cambridge, MA, USA: MIT Press, 2005, pp. 507–514.

[124] Z. Zhao and H. Liu, "Spectral feature selection for supervised and unsupervised learning," in *Proceedings of the 24th International Conference on Machine learning (ICML'07)*. New York, NY, USA: ACM, 2007, pp. 1151–1157.

[125] L. Wolf and A. Shashua, "Feature selection for unsupervised and supervised inference: The emergence of sparsity in a weight-based approach," *Journal of Machine Learning Research*, vol. 6, pp. 1855–1887, 2005.

[126] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'10)*. New York, NY, USA: ACM, 2010, pp. 333–342.

[127] P. Mitra, C. Murthy, and S. Pal, "Unsupervised feature selection using feature similarity," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 3, pp. 301–312, 2002.

[128] R. Cole and M. Fanty, "Spoken letter recognition," in *Proceedings of the Third DARPA Speech and Natural Language Workshop*, 1990, pp. 385–390.

[129] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[130] J. Tropp, "Greed is good: Algorithmic results for sparse approximation," *Information Theory, IEEE Transactions on*, vol. 50, no. 10, pp. 2231–2242, 2004.

[131] A. Das and D. Kempe, "Algorithms for subset selection in linear regression," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*. New York, NY, USA: ACM, 2008, pp. 45–54.

[132] B. Olshausen and D. Field, "Sparse coding with an overcomplete basis set: A strategy employed by VI?" *Vision Research*, vol. 37, no. 23, pp. 3311–3326, 1997.

[133] H. Lee, A. Battle, R. Raina, and A. Ng, "Efficient sparse coding algorithms," in *Advances in Neural Information Processing Systems 19 (NIPS'06)*. MIT, 2006, pp. 801–808.

[134] V. Cevher and A. Krause, "Greedy dictionary selection for sparse representation," *Journal of Selected Topics in Signal Processing*, vol. 5, no. 5, pp. 979–988, 2011.

[135] A. Das and D. Kempe, "Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection," in *Proceedings of the 28th International Conference on Machine Learning, (ICML'11)*, 2011, pp. 1057–1064.

[136] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *Signal Processing, IEEE Transactions on*, vol. 41, no. 12, pp. 3397–3415, 1993.

[137] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4655–4666, 2007.

[138] S. Chen, C. Cowan, and P. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *Neural Networks, IEEE Transactions on*, vol. 2, no. 2, pp. 302 –309, 1991.

[139] P. Vincent and Y. Bengio, "Kernel matching pursuit," *Machine Learning*, vol. 48, no. 1, pp. 165–187, 2002.

[140] Z. Hussain and J. Shawe-Taylor, "Theory of matching pursuit," in *Advances in Neural Information Processing Systems 21 (NIPS'08)*, 2008, pp. 721–728.

[141] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[142] J. Tropp, A. Gilbert, and M. Strauss, "Algorithms for simultaneous sparse approximation. Part I: Greedy pursuit," *Signal Processing*, vol. 86, no. 3, pp. 572–588, 2006.

[143] A. K. Farahat, A. Ghodsi, and M. S. Kamel, "An efficient greedy method for unsupervised feature selection," in *Proceedings of the Eleventh IEEE International Conference on Data Mining (ICDM'11)*. IEEE, 2011, pp. 161 –170.

[144] ——, "Efficient greedy feature selection for unsupervised learning," *Knowledge and Information Systems*, pp. 1–26, 2012, Appeared Online.

[145] ——, "Greedy Nyström approximation." in *Workshop on Low-rank Methods for Large-scale Machine Learning, Advances in Neural Information Processing Systems 23 (NIPS'10)*, 2010.

[146] ——, "A novel greedy algorithm for Nyström approximation." in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*, 2011, pp. 269–277.

[147] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.