ABSTRACT

RACE, SHAINA L. Iterative Consensus Clustering. (Under the direction of Carl Meyer.)

A novel framework for consensus clustering is presented which has the ability to determine both the number of clusters and a final solution using multiple algorithms. A consensus similarity matrix is formed from an ensemble using multiple algorithms and several values for \( k \). A variety of dimension reduction techniques and clustering algorithms are considered for analysis. For noisy or high-dimensional data, an iterative technique is presented to refine this consensus matrix in a way that encourages algorithms to agree upon a common solution. We utilize the theory of nearly uncoupled Markov chains to determine the number, \( k \), of clusters in a dataset by considering a random walk on the graph defined by the consensus matrix. The eigenvalues of the associated transition probability matrix are used to determine the number of clusters. This method succeeds at determining the number of clusters in many datasets where previous methods fail. On every considered dataset, our consensus method provides a final consensus solution with accuracy well above the average of the individual algorithms.
DEDICATION

To my parents, Tamara and Michael Race, and my advisor, Dr. Carl Meyer, for their patience and prodding. Without their support, none of this would have been possible.
ACKNOWLEDGEMENTS

The author would like to thank Dr. Carl Meyer, Dr. Amy Langville, and Ralph Abbey for their help in developing the ideas contained in this thesis. She would also like to thank Dimitrios Zeimpekis and Efstratios Gallopoulos, the creators of the MATLAB Text to Matrix Generator (TMG), which was used to parse many of the documents sets used herein. Additional thanks go to the University of California-Irvine Machine Learning Repository set up by David Aha in 1987 and maintained currently by Arthur Asuncion and David Newman.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
</tbody>
</table>

## Chapter 1 Introduction
1. Clustering
2. Data Clustering
   1.1 Mathematical Notation and Preliminaries
   1.2 Data
   1.3 Text Data
   1.4 The Number of Clusters, \( k \)
3. Partitioning of Graphs and Networks
4. History of Data Clustering

## Chapter 2 Algorithms for Data Clustering
1. Hierarchical Algorithms
   1.1 Agglomerative Hierarchical Clustering
   1.2 Principal Direction Divisive Partitioning (PDDP)
2. Iterative Partitional Algorithms
   2.1 Early Partitional Algorithms
   2.2 \( k \)-means
   2.3 \( k \)-medioids: Partitioning around Mediods (PAM) and Clustering Large Applications (CLARA)
   2.4 The Expectation-Maximization (EM) Clustering Algorithm
3. Density Search Algorithms
   3.1 Density Based Spacial Clustering of Applications with Noise (DBSCAN)

## Chapter 3 Algorithms for Graph Partitioning
1. Spectral Clustering
   1.1 Fiedler Partitioning
   1.2 Other Graph Cuts
   1.3 Power Iteration Clustering
   1.4 Clustering via Modularity Maximization
2. Stochastic Clustering
   2.1 Stochastic Clustering Algorithm (SCA)

## Chapter 4 Dimension Reduction
1. The Curse of Dimensionality
   1.1 Volume
   1.2 Distance and Similarity
2. Dimension Reduction
   2.1 Singular Value Decomposition (SVD)
### Table of Contents

4.2.2 Principal Components Analysis (PCA) .............................................. 52  
4.2.3 Alternative SVD Centerings ............................................................ 54  
4.2.4 Nonnegative Matrix Factorization .................................................. 55  
4.2.5 Other Methods ................................................................................. 56  

**Chapter 5  Cluster Validation:**  
Measuring the Quality of a Clustering ...................................................... 58  
5.1 Internal Validity Metrics ...................................................................... 59  
5.1.1 General Cluster Cohesion and Separation: Graphs vs. Data ............. 59  
5.1.2 Common Measures of Cohesion and Separation ................................. 61  
5.2 External Validity Metrics ..................................................................... 64  
5.2.1 Accuracy ....................................................................................... 64  
5.2.2 Entropy ........................................................................................ 66  
5.2.3 Purity ............................................................................................ 68  
5.2.4 Mutual Information (MI) and Normalized Mutual Information (NMI) . 69  
5.2.5 Other External Measures of Validity .............................................. 70  

**Chapter 6  Determining the Number of Clusters $k$ .......................................................... 74  
6.1 Methods based on Cluster Validity (Stopping Rules) ......................... 74  
6.1.1 Sum Squared Error (SSE) Cohesion Plots ...................................... 75  
6.1.2 Cosine-Cohesion Plots for Text Data ............................................. 76  
6.1.3 Ray and Turi’s Method .................................................................. 78  
6.1.4 The Gap Statistic .......................................................................... 82  
6.2 Graph Methods Based on Eigenvalues (Perron Cluster Analysis) .......... 85  

**Chapter 7  Consensus Clustering ............................................................... 91  
7.1 Consensus Clustering .......................................................................... 92  
7.1.1 Benefits of the Consensus Matrix .................................................. 94  
7.2 Previous Proposals for Consensus Clustering ...................................... 95  
7.3 Iterative Consensus Clustering (ICC) .................................................. 96  
7.3.1 The drop tolerance parameter $\tau$ .................................................. 100  
7.4 Determining the Number of Clusters via Consensus Clustering .......... 101  
7.4.1 Refining the Consensus Matrix through Iteration ............................ 103  
7.5 Conclusions ...................................................................................... 104  

**Chapter 8  Results and Discussion ............................................................ 106  
8.1 Perron Cluster Analysis with Consensus Matrices .............................. 106  
8.1.1 AGBLOG Dataset ........................................................................ 106  
8.1.2 Pendigits17 Dataset ...................................................................... 107  
8.2 Comprehensive Cluster Analysis using Iterative Consensus Clustering . 109  
8.2.1 Wine Dataset ................................................................................ 109  
8.2.2 Newsgroups 6 Collection (NG6) .................................................... 115  
8.3 Reaching Consensus ........................................................................... 121  
8.3.1 Medlars-Cranfield-CISI (MCC) ...................................................... 122  
8.3.2 Benchmark Text Collection by Sinka and Corne ............................ 123
Chapter 9  Conclusions ................................................................. 129
  9.1  Contributions ................................................................. 130
    9.1.1  Determining the Number of Clusters ................................. 130
    9.1.2  The Consensus Matrix ................................................. 130
    9.1.3  Iterated Consensus Clustering ....................................... 131
    9.1.4  Dimension Reduction via NMF ....................................... 131
    9.1.5  The Agreement Metric ................................................. 131
  9.2  Future Work ................................................................. 132

REFERENCES ................................................................. 133
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Some Common External and Relative Indices</td>
<td>71</td>
</tr>
<tr>
<td>5.2</td>
<td>Some Common Measures of Overall Cohesion and Separation [49, 98]</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>Approximated Number of Clusters via Ray and Turi’s Method</td>
<td>81</td>
</tr>
<tr>
<td>7.1</td>
<td>Accuracies of 7 Different Clustering Algorithms on 3 Subsets of Documents from the Same Corpus</td>
<td>92</td>
</tr>
<tr>
<td>7.2</td>
<td>Accuracy of 4 Algorithms on Raw Iris Data</td>
<td>99</td>
</tr>
<tr>
<td>7.3</td>
<td>Pairwise Agreement Between Algorithms on Raw Data</td>
<td>99</td>
</tr>
<tr>
<td>7.4</td>
<td>Accuracy of 4 Algorithms on Consensus Matrix 1</td>
<td>99</td>
</tr>
<tr>
<td>7.5</td>
<td>Accuracy of 4 Algorithms on Consensus Matrix 2</td>
<td>100</td>
</tr>
<tr>
<td>8.1</td>
<td>WINE Data: Summary of Results for Determining the Number of Clusters</td>
<td>114</td>
</tr>
<tr>
<td>8.2</td>
<td>Dataset WINE: Accuracies for individual algorithms on raw data and consensus similarity matrices</td>
<td>114</td>
</tr>
<tr>
<td>8.3</td>
<td>Dataset WINE: Accuracies for spectral algorithms on different similarity matrices</td>
<td>115</td>
</tr>
<tr>
<td>8.4</td>
<td>Dataset WINE: 6 of 8 algorithms find a common solution in one iteration of the final consensus process</td>
<td>115</td>
</tr>
<tr>
<td>8.5</td>
<td>Dataset NG6: Accuracies for individual algorithms on raw data and dimension reductions</td>
<td>120</td>
</tr>
<tr>
<td>8.6</td>
<td>Dataset NG6: Accuracies for individual algorithms on 3 different consensus matrices</td>
<td>121</td>
</tr>
<tr>
<td>8.7</td>
<td>Dataset NG6: Accuracies for spectral algorithms on different similarity matrices</td>
<td>121</td>
</tr>
<tr>
<td>8.8</td>
<td>Dataset NG6: 4 of 7 algorithms find a common solution in one iteration of the final consensus process</td>
<td>121</td>
</tr>
<tr>
<td>8.9</td>
<td>Accuracy Results for 3 Clustering Algorithms on 9 Low Dimensional Representations of the Medlars-Cranfield-CISI text data</td>
<td>122</td>
</tr>
<tr>
<td>8.10</td>
<td>Medlars-Cranfield-CISI text collection: Accuracies for 4 Clustering Algorithms on Consensus Matrices through Iteration</td>
<td>123</td>
</tr>
<tr>
<td>8.11</td>
<td>Dataset BenchmarkDFJX: Accuracy results for raw data and dimension reductions</td>
<td>126</td>
</tr>
<tr>
<td>8.12</td>
<td>Dataset BenchmarkDFJX: Accuracy results for consensus matrices through iteration</td>
<td>126</td>
</tr>
<tr>
<td>8.13</td>
<td>Dataset BenchmarkBCFG: Accuracy statistics for 100 trials of $k$-means algorithm on raw data and dimension reductions</td>
<td>127</td>
</tr>
</tbody>
</table>
Table 8.14  Dataset BenchmarkBCFG: Accuracy statistics for 100 trials of \( k \)-means algorithm on consensus matrix .................................................. 127
Table 8.15  Dataset BenchmarkGHI: Accuracy results for raw data and dimension reductions ................................................................. 127
Table 8.16  Dataset BenchmarkGHI: Accuracy results for consensus matrices through iteration ................................................................. 128
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Three Meaningful Clusterings of One Set of Objects</td>
<td>2</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>How Many Clusters do You See?</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>A Graph with Clusters and its Block Diagonally Dominant Adjacency Matrix</td>
<td>7</td>
</tr>
<tr>
<td>Figure 1.4</td>
<td>Heat Map of Similarity Matrix Exhibiting Cluster Structure</td>
<td>8</td>
</tr>
<tr>
<td>Figure 1.5</td>
<td>Matrix Heat Map Visualization from 1963 Book by Sokal and Sneath</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>A Dendrogram exhibiting linkage/similarity between 9 objects in 3 clusters</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Illustration of Principal Direction Divisive Partitioning</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Failures of Principal Direction Divisive Partitioning</td>
<td>16</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Jancey’s method of reflecting old seed point across the centroid to deter-</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>DBSCAN Illustration</td>
<td>24</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Fiedler Partitions and Zero Valuated Vertices</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Power Iteration Clustering (PIC) Illustrated</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Distribution of $\hat{d}_{ij}$ for 100 random points in $n$-space</td>
<td>47</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Distribution of $\cos(x_i, x_j)$ for 100 random points in $n$-space</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>NMF Feature Extraction Illustrated</td>
<td>56</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>The Difference between Cluster Cohesion Measures in Graph Partitioning vs. Data Clustering</td>
<td>59</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>An Example of a Confusion Matrix</td>
<td>64</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>A More Confusing Confusion Matrix</td>
<td>65</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Matching Predicted Class Labels to Actual Class Labels</td>
<td>65</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>The Two-Dimensional Ruspini Dataset</td>
<td>75</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>5 “Good” $k$-means Clusterings of the Ruspini Dataset and Corresponding SSE Plot</td>
<td>77</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>Example of “Poor” Clusterings’ Effect on SSE Plot</td>
<td>78</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>Spherical $k$-means Objective Function Values for $2 \leq k \leq 20$</td>
<td>79</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>SSE Plots for Medlars-Cranfield-CISI Clusterings using SVD Reduction to $r$ dimensions</td>
<td>80</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>Ray and Turi Validity Plot for Ruspini Data</td>
<td>81</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>Ray &amp; Turi’s Validity Plots for Medlars-Cranfield-CISI Clusterings on Raw Data and SVD Reductions to $r$ dimensions</td>
<td>82</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>Results for gap statistic procedure on Ruspini data. The maximum gap occurs at $k^* = 4$ clusters.</td>
<td>84</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The purpose of this dissertation is to address two major problems that arise in applied cluster analysis: determining the number of clusters and determining a final solution from multiple algorithms. For years researchers have been innovating novel methods for cluster analysis. The vast majority of these algorithms require the user to specify the number of clusters for the algorithm to create. In an applied setting, it is unlikely that the user will know this information before hand. In fact, the number of distinct groups in the data may be the very question that an analyst is attempting to answer. Determining the number of clusters in data has long been considered one of the more difficult aspects of cluster analysis. This fact boils down to basics: what is a cluster? How do we define what should and should not count as two separate clusters? Our approach provides an original answer to this question: A group of points should be considered a cluster when a variety of algorithms agree that they should be considered a cluster. If a majority of algorithms can more or less agree on how to break a dataset into two clusters, but cannot agree on how to partition the data into more than two clusters, then we determine the data has two clusters. This is the essence of our approach.

An additional dilemma arises from the well accepted fact that no method is superior to all others for any type of real data. When many tools exist from which a user may choose, but no hard and fast guidelines are in place for making the choice of one particular tool over another, there is a problem. We propose a method which allows the user to use many of these tools at once, increasing his or her confidence in the final solution.

Before delving into the details of the proposed method, it will be necessary to cover all of the basics of cluster analysis. We start with a preliminary introduction to the field in this chapter.

1.1 Clustering

Clustering is the task of partitioning a set of objects into subsets (clusters) so that objects in the same cluster are similar in some sense. Thus, a “cluster” is a generally a subjective entity determined by how an observer defines the similarity of objects. Take for example
Figure 1.1. Here we have eight objects that differ by shape and color. Depending on the notion of similarity used (color, shape, or both) these objects might be clustered in one of the three ways shown.

![Objects with different shapes and colors arranged in different clusters.](image)

Figure 1.1: Three Meaningful Clusterings of One Set of Objects

### 1.2 Data Clustering

In order to define this problem in a mathematical sense, it is necessary to quantify the attributes of objects using data so that we can mathematically or statistically determine notions of similarity. Data clustering, or cluster analysis, is one of the building blocks of modern data analysis. Data clustering refers to the process of grouping data points *naturally* based on information found in the data which describes its characteristics and relationships. It is not an exact science and, as we will discuss at length later in Chapter 2, there is no *best* method for partitioning data into clusters. First we discuss some notation and preliminaries.

#### 1.2.1 Mathematical Notation and Preliminaries

Throughout this dissertation, we will stick to some common notational traditions. We list them here for your convenience:

- Capitalized, bold-faced letters ($X, S, P$ etc.) will represent matrices.
- Lower-case bold-faced letters ($x, u, v$) will represent vectors.
- Subscripts on vectors indicate their column index inside a matrix ($X = [x_1, \ldots, x_n]$).
- Double matrix subscripts or parenthetical arguments on a vector will be used to represent individual entries in a vector ($V_{i2} = v_2(i)$ is the $i^{th}$ entry in $v_2$).
• All vectors are assumed to be columns, thus $x^T$ will always represent a row vector.
• The vector of all ones will be represented as $e$.
• $\| \ast \|$ will imply the Euclidean norm (2-norm) unless explicitly stated otherwise.
• For a set $S$, $|S|$ is the cardinality or number of elements contained in $S$.

We will also make use of the Singular Value Decomposition (SVD). While many of the pertinent properties of this matrix factorization are given in detail throughout this paper, some familiarity is implied. The SVD is formalized in Definition 1 and a few useful properties are given here. For a more complete treatment of this important mathematical tool, we suggest for example [87, 52].

**Definition 1** (Singular Value Decomposition). For each $A \in \mathbb{R}^{m \times n}$ of rank $r$, there are orthogonal matrices $U_{m \times m}, V_{n \times n}$, and a diagonal matrix $D_{r \times r} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ such that

$$A = U \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} V^T$$

with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$.

The $\sigma_i$’s are called the nonzero singular values of $A$. This factorization is called the singular value decomposition of $A$ and the columns of $U$ and $V$ are called the left-hand and right-hand singular vectors for $A$, respectively [87].

**Some Important Properties of the SVD** [87]

1. The first singular value is equal to the matrix two-norm:

$$\sigma_1 = \max_{\|x\| = 1} \|Ax\|_2 = \|A\|_2$$

2. The Frobenius norm of the matrix is also given by the singular values:

$$\|A\|_F = \sqrt{\sum_{i=1}^{r} \sigma_i^2}$$

3. Singular values represent distances to lower rank matrices:

$$\sigma_{k+1} = \min_{\text{rank}(B) = k} \|A - B\|_2$$

**1.2.2 Data**

Throughout this dissertation the words observation, object, and data point will be used interchangeably to refer to the entities we aim to partition into clusters. These data points could represent any population of interest, be it a collection of documents or a group of Iris flowers.
Each data point will be considered as a column vector, containing measurements of features, attributes, or variables (again, used interchangeably) which characterize it. For example, a column vector characterizing a document could have as many rows as there are words in the dictionary, and each entry in the vector could indicate the number of times each word occurred in the document. An Iris flower, on the other hand, may be characterized by far fewer attributes, perhaps measurements on the size of its petal and sepal. It is assumed we have \( n \) such objects, each represented by a column vector \( \mathbf{x}_j \) containing measurements on \( m \) variables. All of this information is collected in an \( m \times n \) data matrix, \( \mathbf{X} \), which will serve as input to the various clustering methods.

\[
\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]
\]

The aim of data clustering is to automatically determine clusters in these populations based upon the information contained in those vectors. In the document collection, the goal may be to identify clusters of documents which discuss similar subject matter whereas in the Iris population, the goal may be to identify the different species of Iris flowers.

In applied data mining, variables fall into the following four categories: Nominal/Categorical, Ordinal, Interval, or Ratio.

1. **Nominal/Categorical**: Variables which have no ordering, for example ethnicity, color or shape.

2. **Ordinal**: Variables which can be rank-ordered but for which distances have no meaning. For example, scores assigned to levels education (0=no high school, 1=some high school, 2=high school diploma). The distance between 0 and 1 is not necessarily the same as the distance between 1 and 2, but the numbers have some meaning by order.

3. **Interval**: Variables for which differences can be interpreted but for with ratios make no sense. For example if temperature is measured in degrees Fahrenheit the distance from 20 to 30 degrees is the same as the distance from 70 to 80 degrees, however 80 degrees is not “twice as hot” as 40 degrees.

4. **Ratio**: variables for which a meaningful ratio can be constructed. For example height or weight. An absolute zero is meaningful for a ratio variable.

For the algorithms contained herein, it is assumed that the attributes used are ratio variables, although many of the methods have been extended to include other types of data. A majority of experiments in this paper will be conducted using text data because of its high-dimensional nature. In any application of cluster analysis, the interpretation of cluster solutions requires a domain expert who understands the precise nature of the population being studied. Thus, text-mining provides a unique opportunity to work with high-dimensional data in a context where the objects (documents) are easy to interpret (provided they are written in a familiar language, of course!). The next section formalizes how a collection of documents are transformed into data matrix for the purposes of cluster analysis.
1.2.3 Text Data

For this application, the objects to be clustered are called documents. The word document in this context refers to any piece of text - it could be a sentence, a paragraph, a webpage, or an entire article or book. Text is transformed into numerical data by means of a term-document matrix. Since a large number of our experiments involve text data, we formalize this structure in Definition 2.

**Definition 2 (Term-Document Matrix).** Let $n$ be the number of documents in a collection and $m$ be the number of terms appearing in that collection, then we create our term-document matrix $X$ as follows:

$$
X_{m \times n} = \begin{pmatrix}
\text{Term 1} & \text{Doc 1} & \text{Doc j} & \text{Doc n} \\
\text{Term i} & - & - & f_{ij} \\
\text{Term m}
\end{pmatrix}
$$

where $f_{ij}$ is the frequency of term $i$ in document $j$.

Term-document matrices tend to be large and sparse. Term-weighting schemes are often used to downplay the effect of commonly used words and bolster the effect of rare but semantically important words [110, 10]. For the experiments contained herein, the popular “Term Frequency-Inverse Document Frequency” (TF-IDF) weighting method is employed. For this method, the raw term-frequencies $f_{ij}$ in the matrix $X$ are multiplied by global weights (inverse document frequencies), $w_i$, for each term. These weights reflect the commonality of each term across the entire collection. Let $D$ represent the entire document collection, $d_j$ represent the document $j$ and $t_i$ represent term $i$. The inverse document frequency of term $i$ is:

$$w_i = \log \left( \frac{n}{|\{d_j \in D : t_i \in d_j\}|} \right).$$

To put this weight in perspective for a collection of $n = 10,000$ documents we have $0 \leq w_i \leq 9.2$. The document vectors are also normalized to have unit 2-norm, since their directions (not their lengths) in the term-space is what characterizes them semantically.

For the majority of text datasets contained herein, the term-document matrices were formed using the Text to Matrix Generator (TMG) for MATLAB. This tool was created by Dimitrios Zeimpekis and Efstratios Gallopoulos at The University of Patras in Greece [135].

1.2.4 The Number of Clusters, $k$

One of the important problems in cluster analysis is the determination of the number of clusters, $k$. The number of clusters can also be a matter of subjectivity. Take for instance the
2-dimensional scatter plot of points in Figure 1.2. Using the points' proximity in Euclidean space as a measure of their similarity, one could argue that there are any number of clusters in this simple illustration. However, most people would agree that there is indeed cluster-like structure in this data.

Figure 1.2: How Many Clusters do You See?

It is easy to imagine data in which the number of clusters to specify is a matter of debate only because groups of related objects can be meaningfully divided into subcategories or subclusters. For example a collection of webpages may clearly fall into 3 categories: sports, investment banking, and astronomy. If the webpages about sports further divide into 2 categories like baseball and basketball then we’d refer to that as subclustering. One of the aims of the method in Chapter 7 is to determine the most reasonable number of clusters in data based upon whether or not a majority of clustering algorithms can agree upon how exactly to partition the data into that particular number of clusters. Chapter 6 provides a thorough examination of existing procedures for determining the number of clusters.

1.3 Partitioning of Graphs and Networks

Another popular research problem in clustering is the partitioning of graphs. In network applications this problem has become known to many as network community detection [94, 102, 77], although the underlying problem of partitioning graphs has been studied extensively for years [33, 43, 24, 95, 39, 103, 32]. A graph is a set of vertices (or equivalently nodes) $N = \{1, 2, \ldots, n\}$ together with a set of edges $E = \{(i, j) : i, j \in N\}$ which connect vertices together. A weighted graph is one in which the edges are assigned some weight $w_{ij}$ whereas an unweighted graph has binary weights for edges: $w_{ij} = 1$ if $(i, j) \in E$, $w_{ij} = 0$ otherwise. Our focus will be on undirected graphs in which $w_{ij} = w_{ji}$. All algorithms for graph partition-
ing (or network community detection) will rely on an adjacency matrix. An adjacency matrix $A$ for an undirected graph is an $n \times n$ symmetric matrix defined as follows:

$$A_{ij} = \begin{cases} w_{ij}, & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Figure 1.3a is an example of a graph exhibiting cluster or community structure. The weights of the edges are depicted by their thickness. It is expected that edges within the clusters occur more frequently and with higher weight than edges between the clusters. Thus, once the rows and columns of the matrix are reordered according to their cluster membership, we expect to see a matrix that is block-diagonally dominant - that is, one in which values in square diagonal blocks are relatively large compared to those in the off-diagonal blocks. The notion of block-diagonal dominance will be formalized in Chapter 6.

In much of the literature on graph partitioning, it is suggested that the data clustering problem can be transformed into a graph partitioning problem by means of a similarity matrix [32, 124, 103, 39]. A similarity matrix $S$ is an $n \times n$ symmetric matrix where $S_{ij}$ measures some notion of similarity between data points $x_i$ and $x_j$. There are a wealth of metrics available to gauge similarity or dissimilarity, see for example [49]. Common measures of similarity rely on Euclidean or angular distance measures. Two of the most popular similarity functions in the literature are the following:

- **Gaussian Similarity:**

  $$S_{ij} = \exp\left(-\frac{||x_i - x_j||^2}{2\alpha^2}\right)$$

  where $\alpha$ is a tuning parameter.
• Cosine Similarity:
\[
S_{ij} = \cos(x_i, x_j) = \frac{x_i^T x_j}{\|x_i\|\|x_j\|}
\]

Any similarity matrix can be considered an adjacency matrix for a graph, thus transforming the original data clustering problem into a graph partitioning problem. Several algorithms for data clustering and graph partitioning are provided in Chapter 2. Regardless of the method chosen for clustering, similarity matrices can be a useful tool for visualizing cluster results in light of the block diagonal structure shown in Figure 1.3. This block diagonal structure can be visualized using a heat map of a similarity matrix. A matrix heat map represents each value in a matrix by a colored pixel indicating the magnitude of the value. Figure 1.4 is an example of a heat map using real data. The data are a collection of news articles (documents) from the web containing seven different topics of discussion. The rows and columns of the cosine similarity matrix for these documents have been reordered according to some cluster solution. White pixels represent negligible values of similarity. Some of these topics are closely related, which can be seen by the heightened level of similarities between blocks. This data collection, a subset of “twenty newsgroups”, will be revisited in Chapter 8.

![Figure 1.4: Heat Map of Similarity Matrix Exhibiting Cluster Structure](image)

While a heat-map visualization allows the user to get a sense of the quality of a specific clustering, it does not always make it easy to determine which is a better solution given two different clusterings. Since clustering is an unsupervised process, quantitative measures of cluster quality are often used to compare different clusterings. A short survey of these metrics is given in Chapter 5. First, we will take a brief historical look at the roots of cluster...
1.4 History of Data Clustering

According to Anil Jain in [60], data clustering first appeared in an article written by Forrest Clements in 1954 about anthropological data [25]. However, a Google Scholar search provides several earlier publications whose titles also contain the phrase “cluster analysis” [120, 2]. In fact, the discussion of data clustering dates back to the 1930’s when anthropologists Driver and Kroeber [35] and psychologists Zubin [136] and Tryon [1] realized the utility of such analysis in determining cultural or psychological classifications. While the usefulness of such techniques was clear to researchers in many social and biological disciplines at that time, the lack of computational tools made the analysis time consuming and practically impossible for large sets of data.

Cluster analysis exploded into the limelight in the 1960’s and ’70’s after Sokal and Sneath’s 1963 book Principles of Numerical Taxonomy [114]. Although the text is primarily geared toward biologists faced with the task of classifying organisms, it motivated researchers from many different disciplines to consider the problem of data clustering from other angles like computing, statistics, and domain specific applications. Sokal and Sneath’s book presents a detailed discussion of the simple, intuitive, and still popular hierarchical clustering (see Section 2.1) techniques for biological taxonomy. These authors also provided perhaps the earliest mention of the matrix heat map visualizations of clustering that are still popular today. Figure 1.5 shows an example of one of these heat maps, then drawn by hand, from their book.

Prior to the development of modern computational resources, programs for numerical taxonomy were written in machine language and not easily transferred from one computer to another [114]. However, by the mid 1960s, it was clear that the advancement in technology would probably keep pace with advancements in algorithm design and many researchers from various disciplines began to contribute to the clustering literature. The next two chapters present an in-depth view of the most popular developments in data clustering since that time. Chapter 4 will look at a common problem associated with the massive datasets of modern day, and Chapter 7 will present novel techniques for cluster analysis using multiple algorithms.
Figure 1.5: Matrix Heat Map Visualization from 1963 Book by Sokal and Sneath
Chapter 2

Algorithms for Data Clustering

There have been countless algorithms proposed for data clustering. While a complete survey and discussion of clustering algorithms would be nearly impossible, this chapter provides an introduction to some of the most popular algorithms to date. For the purposes of organization, the algorithms are divided into 3 groups: Hierarchical, Iterative Partitional, and Density-based.

2.1 Hierarchical Algorithms

As discussed in Chapter 1, data clustering became popular in the biological fields of phylogeny and taxonomy. Even prior to the advancement of numerical taxonomy, it was common for scientists in this field to communicate relationships by way of a dendrogram or tree diagram as illustrated in Figure 2.1 [114]. Dendrograms provide a nested hierarchy of similarity that allow the researcher to see different levels of clustering that may exist in data, particularly in phylogenetic data. Agglomerative hierarchical clustering has its roots in this domain.

2.1.1 Agglomerative Hierarchical Clustering

The idea behind agglomerative hierarchical clustering is to link similar objects or similar clusters of objects together in a hierarchy where the highest levels of similarity is represented by the lowest level connections. These methods are called agglomerative because they begin with each data point in a separate cluster and at each step they merge clusters together according to some decision rule until eventually all of the points end up in a single cluster. For example, in Figure 2.1, objects 1 and 2 exhibit the highest level of similarity as indicated by the height of the branch that connects them. Also illustrated in the dendrogram is the fact that the blue cluster and green cluster are more similar to each other than they are to the red cluster. One of the advantages to these hierarchical structures is that branches can be cut to achieve any number of clusters desired by the user. For example, in Figure 2.1 if only the
highest branch of the dendrogram is cut, the result is two clusters: \([\{1,2,3\},\{4,5,6,7,8,9\}]\). When the next highest branch is cut, we are left with 3 clusters: \([\{1,2,3\},\{4,5,6\},\{7,8,9\}]\).

Figure 2.1: A Dendrogram exhibiting linkage/similarity between 9 objects in 3 clusters.

There are a number of different systems for determining linkage in hierarchical clustering dendrograms. For a complete discussion, we suggest the classic books by Anderberg [4] or Jain and Dubes [61]. The basic scheme for hierarchical clustering algorithms is outlined in Algorithm 1.

**Algorithm 1** Agglomerative Hierarchical Clustering

**Input:** n objects to be clustered.

1. Begin by assigning each object to its own cluster.

2. Compute the pairwise similarities between each cluster.

3. Find the most similar pair of clusters and merge them into a single cluster. There is now one less cluster.

4. Compute pairwise similarities between the new cluster and each of the old clusters.

5. Repeat steps 3-4 until all objects belong to a single cluster of size n.

**Output:** Dendrogram depicting each merge step.

What differentiates the numerous hierarchical clustering algorithms is the choice of similarity metric used and the way the chosen similarity metric is used to compare clusters in step 4. For example, suppose Euclidean distance is chosen to compute the similarity (or dissim-
ilarity) between objects in step 2. In step 4, the same notion of similarity must be extended to compare clusters of objects. Several methods of computing pairwise distances between clusters have been proposed over the years. The most common approaches are as follows:

**Single-Linkage**
The distance between two clusters is equal to the *shortest* distance from any member of one cluster to any member of the other cluster.

**Complete-Linkage**
The distance between two clusters is equal to the *greatest* distance from any member of one cluster to any member of the other cluster.

**Average-Linkage**
The distance between two clusters is equal to the *average* distance from any member of one cluster to any member of the other cluster.

While many people have been given credit for the methods listed above, it appears that numerical taxonomers Sneath, Sokal and Michener were the first to describe the Single- and Average-linkage protocols, while ecologist Sorenson had previously pioneered Complete-linkage in his ecological studies. These early researchers used correlation coefficients to measure similarity between objects, but they suggest in 1963 that other correlation-like or distance-like measures could also be useful [114]. The paper by Stephen Johnson in 1967 [64] formalized the single- and complete-linkage algorithms in a more general data clustering setting. Other linkage techniques for hierarchical clustering, such as centroid and median linkage, have been proposed as well. We refer interested readers to Anderberg [4] for more on these variants.

The main drawback of agglomerative hierarchical schemes is their computational complexity. In recent years, variations like BIRCH [121] and CURE [54] have been developed in an effort to combat this problem. Another feature which causes problems in some applications is that once a connection between points or clusters is made, it cannot be undone. For this reason, hierarchical algorithms often suffer in the presence of noise and outliers.

### 2.1.2 Principal Direction Divisive Partitioning (PDDP)

While the hierarchical algorithms discussed above were *agglomerative*, it is also possible to create a cluster hierarchy or dendrogram by iteratively *dividing* points into groups until a desired number of groups is reached. Principal Direction Divisive Partitioning (PDDP) is one example of a *divisive hierarchical algorithm*. Other partitional methods which will be discussed in Chapter 3 can also be placed in this hierarchical framework.

PDDP was proposed in [14] by Daniel Boley at the University of Minnesota. PDDP has become popular due to its computational efficiency and ability to handle large data sets. We will explain this algorithm in a different, but equivalent context than is done in the original paper. At each step of this method, data are projected onto the *total least squares line* and split into two groups based upon which side of the mean their projections fall. The *total least
squares line $L$ is the line which minimizes the total sum of squares of orthogonal deviations between the data and $L$ among all lines in $\mathbb{R}^m$. Let $X = [x_1, x_2, \ldots, x_n]$ be the data points and $L(u, p)$ be a line in $\mathbb{R}^m$ where $p$ is a point on a line and $u$ is the direction of the line. The projection of $x_j$ onto $L(u, p)$ is given by

$$\hat{x}_j = uu^T(x_j - p) + p,$$

and therefore the orthogonal distance between $x_j$ and $L(u, p)$ is

$$x_j - \hat{x}_j = (I - uu^T)(x_j - p).$$

Consequently, the total least squares line is the line $L$ which minimizes (over directions $u$ and points $p$)

$$f(u, p) = \sum_{j=1}^{n} \|x_j - \hat{x}_j\|^2_2$$

$$= \sum_{j=1}^{n} \|(I - uu^T)(x_j - p)\|^2_2$$

$$= \|(I - uu^T)(X - pe^T)\|^2_F. \quad (2.1)$$

The following definition precisely characterizes the total least squares line.

**Definition 3. Total Least Squares Line.**

The **total least squares line** for the column data in $X = [x_1, x_2, \ldots, x_n]$ is given by

$$L = \{\alpha u_1(X_c) + \mu | \alpha \in \mathbb{R}\},$$

where $\mu = Xe/n$ is the mean (centroid) of the column data, and $u_1(X_c)$ is the principal left-hand singular vector of the centered matrix

$$X_c = X - \mu e^T = X(I - ee^T/n).$$

$u_1(X_c)$ is also known as the **first principal component** of $X$.

The orthogonal projection of the data onto the total least squares line will capture the maximum amount of directional variance over all possible one dimensional orthogonal projections. This fact is treated in greater detail in Chapter 4.

Boley’s PDDP algorithm partitions the data into two clusters at each step based upon whether their projections onto the total least squares line fall to the left or to the right of $\mu$. This is equivalent to examining the signs of the projections of the centered data, $X_c$, onto the direction $u_1(X_c)$. Conveniently, the signs of the projections are determined by the signs of the entries in the principal right-hand singular vector, $v_1(X_c)$. A simple example motivating this method is illustrated in Figure 2.2.
Once the data are divided, the two clusters are examined to find the one with the greatest variance (scatter). This subset of data is then extracted from the original data matrix, centered and projected onto the span of its own first principal component. The split at zero is made again and the algorithm proceeds iteratively until the desired number of clusters has been produced.

It is necessary to note, however, that the example in Figure 2.2 is truly an ideal geometric configuration of data. Figure 2.3 illustrates configurations in which PDDP would fail. In Figure 2.3a, both clusters would be split down the middle, and in Figure 2.3b the middle cluster would be split in the first iteration. Unfortunately, once data points are separated in an iteration of PDDP, there is no chance for them to be rejoined later. The steps of PDDP are given in Algorithm 2.

Since its initial publication, variations of the PDDP algorithm have been proposed, most notably PDDP(ℓ) [134] and KPDDP [129], both developed by Dimitrios Zeimpekis and Efstathios Gallopoulos from the University of Patras in Greece. PDDP(ℓ) uses the sign patterns in the first ℓ principal components to partition the data into at most $2^\ell$ clusters at each step of the algorithm, whereas KPDDP is a kernal variant which uses $k$-means to steer the cluster assignments at each step.
Algorithm 2 Principal Direction Divisive Partitioning (PDDP)

Input: \( n \) data points \( X = [x_1, x_2, \ldots, x_n] \) and number of clusters \( k \)

1. Center the data to have mean zero: \( X_c = X - \mu e^T \).
2. Compute the first right singular vector of \( X_c \), \( v_1 \).
3. Partition the data into two clusters based upon the signs of the entries in \( v_1 \).
4. Compute the variance of each existing cluster and choose the cluster with largest variance to partition next.
5. Repeat steps 1-4 using only the data in the cluster with largest variance until eventually \( k \) clusters are formed.

Output: Resulting \( k \)-clusters

2.2 Iterative Partitional Algorithms

Iterative partitional algorithms begin with an initial partition of the data into \( k \) clusters and iteratively update the cluster memberships according to some notion of what constitutes a
“better” partition [61, 4]. The k-means algorithm is one example of a partitional algorithm. Before we get into the details of the modern day k-means algorithms, we’ll take a look back at the history that fostered its development as one of the best-known and most widely used clustering algorithms in the world.

2.2.1 Early Partitional Algorithms

Although the name “k-means” was first used by MacQueen in 1967 [78], the partitional method generally referred to by this name today was proposed by Forgy in 1965 [46]. Forgy’s algorithm involves iteratively updating k seed points which, at each pass of the algorithm, define a partitioning of the data by associating each data point with its nearest seed point. The seeds are then updated to represent the centroids (means) of the resulting clusters and the process is repeated. Euclidean distance is the most common metric for measuring the nearness of points in these algorithms, but other metrics, such as Mahalanobis distance and angular distance, can and have been used as well. K-means can also handle binary or categorical variables by using simple matching coefficients found in the data mining literature, for example [98]. Forgy’s method is outlined in Algorithm 3. In 1966, Jancey suggested a variation of this method where the new seeds points in step 2 were computed by reflecting the old seed point across the new centroid, as depicted in Figure 2.4. Jancey argued that the data’s nearness to point 1 grouped them into a cluster initially, and thus using a seed point which exaggerates this movement toward the new centroid ought to help speed up convergence, and possibly lead to a better solution by avoiding local minima [62].

MacQueen’s 1967 partitional process, which he called “k-means”, differs from Forgy’s formulation in that it a) specifies initial seed points and b) assigns data points to clusters one-by-one, updating the seed to be the centroid of the cluster each time a new point is added. The algorithm only makes one pass through the data. MacQueen’s method is presented in Algorithm 4.

Algorithm 3 Forgy’s k-means Algorithm [4]

Input: Data points and an initial cluster configuration of the data, defined by k seed points (start in step 1) or an initial clustering (start in step 2).

1. Assign each data point to the cluster associated with the nearest seed point.

2. Compute new seed points to be the centroids of the clusters.

3. Repeat steps 1 and 2 until no data points change cluster membership in step 2.

Output: Final Clusters
Algorithm 4 MacQueens k-means Algorithm

**Input:** $n$ data points

1. Choose the first $k$ data points as clusters with one member each. Set $i=1$.

2. Assign the $(k + i)^{th}$ data point to the cluster with the closest centroid. Recompute the centroid of the updated cluster. Set $i = i + 1$.

3. Repeat step 2 until $i = n - k$ and all the data points have been assigned. Use final cluster centroids to determine a final clustering by re-assigning each data point to the cluster associated with its nearest centroid.

**Output:** Final Clusters

As you can see, MacQueen’s algorithm, while similar in spirit, is quite different from that proposed by Forgy. The set of clusters found is likely to be dependent upon the order of the data, a property generally undesirable in cluster analysis. MacQueen stated that in his experience, these discrepancies in final solution based upon the order of the data were generally minor, and thus not unlike those caused by the choice of initialization in Forgy’s method. An advantage of MacQueen’s algorithm is the reduced computation load achieved by avoiding the continual processing of the data to convergence. It has also been suggested that MacQueen’s method may be useful to initialize the seeds for Forgy’s algorithm [4] and in fact this option is available in many data mining software packages like SAS’s Enterprise Miner.
2.2.2  \(k\)-means

We will finish our discussion of \(k\)-means with what has become the classical presentation. We begin with a matrix of column data, \(X = [x_1, x_2, \ldots, x_n]\) where \(x_i \in \mathbb{R}^m, 1 \leq i \leq n\). The objective of \(k\)-means is to determine a partitioning of the data into \(k\) sets, \(C = \{C_1, C_2, \ldots, C_k\}\), such that an intra-cluster sum of squares cost function is minimized:

\[
\arg\min_C \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2
\]

Any desired distance metric can be used, according to the applications and whims of the user. Euclidean distance is standard, and leads to the specification \(Euclidean\ k\text{-means}\). In document clustering, it is common to use the cosine of the angle between two data vectors (documents) to measure their distance from each other. This variant is commonly referred to as \textit{spherical} \(k\text{-means}\) and will be discussed briefly in Section 2.2.2. The \(k\)-means algorithm, which is essentially the same as Forgy’s algorithm in Section 2.2, is presented in Algorithm 5.

\begin{algorithm}
\textbf{Input:} Data points \(\{x_1, x_2, \ldots, x_n\}\) and set of initial centroids \(\{\mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_k^{(0)}\}\).

1. Assign each data point to cluster associated with the nearest centroid.

\[
C_j^{(t)} = \{x_i : \|x_i - \mu_j^{(t)}\| \leq \|x_i - \mu_l^{(t)}\| \quad \forall 1 \leq l \leq k\}
\]

If two centroids are equally close, the tie is broken arbitrarily.

2. The new centroid for each cluster is calculated by setting

\[
\mu_j^{(t+1)} = \frac{1}{|C_j^{(t)}|} \sum_{x_i \in C_j^{(t)}} x_i
\]

3. Repeat steps 2 and 3 until the centroids remain stationary.

\textbf{Output:} \(k\) clusters \(C_1, C_2, \ldots, C_k\)
\end{algorithm}

This algorithm is guaranteed to converge because there are a finite number of partitions possible and at each pass of the algorithm the intra-cluster sum of squares cost function is decreased due to the fact that points are reassigned to a new cluster only if they are closer to the existing centroid of the new cluster than they were to the old one. The cost function is further reduced as the new centroids are calculated and the process repeats, lowering the
cost function at each step. However, it is quite common for the algorithm to converge to local minima, particularly with large datasets. The output of \( k \)-means is sensitive to the initialization of the centroids and the choice of distance metric used in step 2. Randomly initialized centroids tend to be the most popular, but one can also seed the algorithm with centroids of clusters determined by another clustering algorithm. We will implement both approaches for our experiments in Chapter 8. One of the objectives of our method in Chapter 7 is to combine results from multiple trials of the algorithm using different initializations.

**Spherical \( k \)-means**

In some applications, such as document clustering, similarity is often measured by the cosine of the angle \( \theta \) between two objects \( x_i \) and \( x_j \) (each normalized to have unit norm),

\[
\cos(\theta) = x_i^T x_j.
\]

This similarity is often transformed into a distance by computing the quantity \( d(x_i, x_j) = 1 - \cos(\theta) \) to formulate the spherical \( k \)-means objective function as follows:

\[
\min_C \sum_{i=1}^{k} \sum_{x \in C_i} 1 - x^T c_i.
\]

Where \( c_i = \frac{1}{||\mu_i||} \mu_i \) is the normalized centroid of the cluster. The spherical \( k \)-means algorithm is the same as the euclidean \( k \)-means algorithm aside from the definition of nearness in step 2.

### 2.2.3 \( k \)-medioids: Partitioning around Mediods (PAM) and Clustering Large Applications (CLARA)

In 1987, Kaufman and Rousseeuw devised another partitional method which searched through data in order to find \( k \) representative points (or mediods) belonging to the dataset which would serve as cluster centers in the same way the centroids do in \( k \)-means. They called these points "representative" because it was thought the points would give some interpretability to the groups by exhibiting some defining characteristics of their associated clusters and distinguishing characteristics from other clusters. The authors' original algorithm, Partitioning around Mediods (PAM), was not suitable for large datasets because of the computation time necessary to search through the data points to build the set of \( k \) representative points. The same authors developed a second algorithm, Clustering Large Applications (CLARA), to combat this problem. The central idea of CLARA was to use PAM on large datasets by sampling the data and applying the algorithm on the smaller sample. Once \( k \) representative points were found in the sample, the remaining data were associated with the mediod to which they were closest. The quality of the clustering is measured by the average distance of every object to its representative point. Five such samples are drawn, and the clustering that
results in the lowest average distance is retained [67].

2.2.4 The Expectation-Maximization (EM) Clustering Algorithm

The Expectation-Maximization (EM) Algorithm, originally proposed by Dempster, Laird, and Rubin in 1977 [5], is one that has been used to solve many types of statistical problems over the years. It is generally used to determine parameters of a statistical model used to describe observations in a dataset. Here we will show how the algorithm is used for clustering, as in [22]. Our discussion is limited to the variant of the algorithm which uses Gaussian mixtures to model the data.

Supposing that our data points, \(x_1, x_2, \ldots, x_n\), each belonging to one of \(k\) clusters (or classes), \(C_1, C_2, \ldots, C_k\). Then there exists some latent variables \(y_i, 1 \leq i \leq n\), which identify the class membership of each \(x_i\). It is assumed that each class label \(C_i\) determines the probability distribution of the data in that class. Here, we assume that this distribution is multivariate Gaussian. The parameters of this model include the a priori probabilities of each of the \(k\) classes, \(P(C_i)\), and the parameters of the corresponding normal distributions \(\mu_i\) and \(\Sigma_i\), which are the mean and covariance matrix respectively. The objective of the EM algorithm is to determine the parameters which maximize the likelihood of the data:

\[
\log L = \sum_i (\log P(y_i) + \log P(x_i|y_i))
\]

The EM algorithm takes as input a set of \(m\)-dimensional data points, \(\{x_i\}_{i=1}^n\), the desired number of clusters \(k\), and an initial set of parameters \(\theta_j\) for each cluster \(C_j\) \(1 \leq j \leq k\). For Gaussian mixtures, \(\theta_j\) consists of mean \(\mu_j\) and an \(m \times m\) covariance matrix \(\Sigma_j\). The a priori probability of each cluster, \(\alpha_j = P(C_j)\) must also be initialized and updated throughout the algorithm. If no information is known about the underlying clusters, then we suggest initialization \(\alpha_j = 1/k\) for all clusters \(C_j\). EM then operates by iteratively executing an expectation step, where the probability that each data point belongs to each of the \(k\) classes is computed, followed by a maximization step, where the parameters for each class are updated to maximize the likelihood of the data [22]. These steps are summarized in Algorithm 6.

The EM Algorithm with Gaussian mixtures works well for clustering when the normality assumption of the underlying clusters holds true. Unfortunately, it is difficult to know if this is the case prior to the identification of the clusters. The algorithm suffers considerable computational drawbacks, particularly with regards to storage of the \(k\) covariance matrices \(\Sigma_j \in \mathbb{R}^{m \times m}\), and is not easily run in parallel. For this reason, the EM algorithm is generally limited in its ability to be used on large datasets, particularly when the number of attributes \(m\) is very large, as it is in document clustering.
Algorithm 6 Expectation-Maximization Algorithm for Clustering [22]

**Input:** \( n \) data points, \( \{x_i\}_{i=1}^n \), number of clusters \( k \), and initial set of parameters for each cluster \( C_j \): \( \alpha_j \) and \( \theta_j = \{\mu_j, \Sigma_j\} \) \( 1 \leq j \leq k \)

1. **Expectation Step:** Compute the probability of each data point \( x_i \) being drawn from each class distribution, \( C_j \):
   \[
p_{ij} = P(x_i | \alpha_j, \mu_j, \Sigma_j) \propto \alpha_j P(x_i | \mu_j, \Sigma_j)
   \]

2. **Maximization Step:** Update the parameters to maximize the likelihood of the data:
   \[
   \alpha_j = \frac{1}{n} \sum_{i=1}^{n} p_{ij}
   \]
   \[
   \mu_j = \frac{\sum_{i=1}^{n} p_{ij} x_i}{\sum_{i=1}^{n} p_{ij}}
   \]
   \[
   \Sigma_j = \frac{\sum_{i=1}^{n} p_{ij} (x_i - \mu_j)(x_i - \mu_j)^T}{\sum_{i=1}^{n} p_{ij}}
   \]

3. Repeat steps 1-2 until convergence.

**Output:** Class label \( j \) for each \( x_i \) such that \( p_{ij} \geq p_{il} \) \( 1 \leq l \leq k \)

2.3 **Density Search Algorithms**

If objects are depicted as data points in a metric space, then one may interpret the problem of clustering as an attempt to find areas of the space that are densely populated by points, separated by less populated areas. A natural approach to the problem is then to search through the space seeking these dense regions. Such algorithms have been referred to as density search algorithms [38]. While these algorithms tend to suffer on real data in both accuracy efficiency, their ability to identify noise and to estimate the number of clusters \( k \) makes them worthy of discussion.

Many density search algorithms have their roots in the single-linkage hierarchical algorithms described in Section 2.1. Individual points are joined together in clusters one-by-one based upon their similarity (or nearness in space). However in this case there exists some criteria for which objects are rejected from joining an existing cluster and instead are set out to form their own cluster. For example, suppose we had two distinct well separated dense regions of points. Beginning with a single point in the first region, we form a cluster and search through the remaining points one by one adding them to the cluster in they satisfy some specified criterion of nearness to the points already in the cluster. Once all the points in the first region are combined into a single cluster, the purpose of the criterion is to reject points from the second region from joining the first cluster, causing them to create a new
cluster.

The conception of density search algorithms dates to the late ‘60s with the taxmap method of Carmichael et al. in [21, 20] and the mode analysis method of Wishart [128]. In taxmap the authors suggested criterion like the drop in average similarity upon adding a new point to a cluster. In mode analysis the criterion was simply containment in a specified radius of points in a cluster. The problem with this approach was that it had trouble finding both large and small clusters simultaneously [38].

All density search algorithms suffer from the inability to find clusters of varying density, no matter how the term is defined in application, because the density of points is used to define the notion of a cluster. High dimensional data adds to this problem as demonstrated in Chapter 4 because as the size of the space grows, the points naturally become less and less dense inside of it. Another problem with density search algorithm is the necessity to search through data again and again, making their implementation difficult if not irrelevant for large data sets. Among the benefits to these methods are the inherent estimation of the number of clusters and their ability to find irregularly shaped (non-convex) clusters. Several algorithms in this category, like Density Based Spacial Clustering of Applications with Noise (DBSCAN) also make an effort to determine outliers or noise in the data. Because of the computational workload of these methods, we will abandon them after the present discussion in favor of more efficient methods. For an in-depth analysis of other density search algorithms and their variants, see [23].

2.3.1 Density Based Spacial Clustering of Applications with Noise (DBSCAN)

Density Based Spacial Clustering of Applications with Noise (DBSCAN) is an algorithm proposed by Ester, Kriegel, Sander, and Xu in 1996 [37], which uses the Euclidean nearness of a group of points in \( m \)-space to define density. The algorithm uses the following definitions and parameters to determine what constitutes a cluster:

**Dense Point and \( \rho_{min} \)**
A point \( x_j \) is called *dense* if there are at least \( \rho_{min} \) other points contained in its \( \epsilon \)-neighborhood.

**Direct Density Reachability**
A point \( x_i \) is called *directly density reachable* from a point \( x_j \) if it is in the \( \epsilon \)-neighborhood surrounding \( x_j \), i.e. if \( x_i \in N(x_j, \epsilon) \), and \( x_j \) is a dense point.

**Density Reachability**
A point \( x_i \) is called *density reachable* from a point \( x_j \) if there is a sequence of points \( x_1, x_2, \ldots, x_p \) with \( x_1 = x_j \) and \( x_p = x_i \) where each \( x_{k+1} \) is directly density reachable from \( x_k \).

**Noise Point**
A point \( x_j \) is called a *noise point* or *outlier* if it contains 0 points in its \( \epsilon \)-neighborhood.
The relationship of density reachability is not symmetric. This fact is illustrated in Figure 2.5. A point in this illustration is dense if its \( \epsilon \)-neighborhood contains at least \( \rho_{\text{min}} = 2 \) other points. The green point \( a \) is density reachable from the blue point \( b \), however the reverse is not true because \( a \) is not a dense point. Because of this, we introduce the notion of *density connectedness*.

\[
\epsilon \quad \rho_{\text{min}} = 2
\]

![DBSCAN Illustration](image)

**Figure 2.5:** DBSCAN Illustration

**Density Connectedness**

Two points \( x_i \) and \( x_j \) are *density-connected* if there exists some point \( x_k \) such that both \( x_i \) and \( x_j \) are density reachable from \( x_k \).

In Figure 2.5, it is clear that we can say points \( a \) and \( b \) are density-connected since they are each density reachable from any of the 4 points in between them. The point \( c \) in this illustration is a noise point or outlier because there are no points contained in its \( \epsilon \)-neighborhood.

Using these definitions, we can formalize the properties that define a cluster in DBSCAN. Given the parameters \( \rho_{\text{min}} \) and \( \epsilon \), a cluster is a set of points that satisfy the two following conditions:

1. All points within the cluster are mutually density-connected.
2. If a point is density-connected to any point in the cluster, it is part of the cluster as well.

Algorithm 7 describes how DBSCAN finds such clusters.
Algorithm 7 Density Based Spacial Clustering of Applications with Noise (DBSCAN) [98]

Input: Set of points $X = [x_1, x_2, \ldots, x_n]$ to be clustered and parameters $\epsilon$ and $\rho_{min}$

1. For each unvisited point $p = x_i$, do:
   I. Mark $p$ as visited.
   II. Let $N$ be the set of points contained in the $\epsilon$-neighborhood around $p$.
      (a) If $|N| < \rho_{min}$ mark $p$ as noise.
      (b) Else let $C$ be the next cluster. Do:
         i. Add $p$ to cluster $C$.
         ii. For each point $p'$ in $N$, do:
            A. If $p'$ is not visited, mark $p'$ as visited, let $N'$ be the set of points contained in the $\epsilon$-neighborhood around $p'$. If $|N'| \geq \rho_{min}$ let $N = N \cup N'$
            B. If $p'$ is not yet a member of any cluster, add $p'$ to cluster $C$.

Output: Clusters found $C_1, \ldots, C_k$

2.4 Conclusion

The purpose of this chapter was to give the reader a basic understanding of hierarchical, iterative partitional, and density search approaches to data clustering. One of the main concerns addressed in this paper is that all of these algorithms have merit, but in application rarely do the algorithms completely agree on a solution. In fact, algorithms with random inputs like $k$-means are not even likely to agree with themselves over a number of different trials. It can be extremely difficult to qualitatively measure the goodness of your clustering when the data cannot be visualized in 2 or 3 dimensions. While there are a number of metrics to help the user get a sense of the compactness of the clusters (see Chapter 5), the effect of noise and outliers can often blur the true picture. It is also common for such metrics to take nearly equivalent values for vastly different cluster solutions, forcing the user to choose a solution in an ad hoc manner. In Chapter 7 we will present a solution to this ambiguity by using multiple algorithms to obtain a solution with which the user can have more confidence. First we will look at another class of clustering methods which aim to solve the graph partitioning problem described in Chapter 1.

The difference between the problems of data clustering and graph partitioning is merely the structure of the input objects to be clustered. In data clustering, the input objects are composed of measurements on $m$ variables or features. If we interpret the graph partitioning problem in such a way that input objects are vertices on a graph and the variables describing them are the weights of the edges by which they are connected to other vertices, then it becomes clear we can use any of the methods in this chapter to cluster the columns of an
adjacency matrix as described in Chapter 1. Similarly if one creates a similarity matrix for objects from a data clustering problem, we can cluster that matrix using the theory and algorithms from graph partitioning. While each problem can be transformed into the other, the design of the algorithms for the two cases is generally quite different. In the next chapter, we provide a thorough overview of some popular graph clustering algorithms.
Chapter 3

Algorithms for Graph Partitioning

3.1 Spectral Clustering

Spectral clustering is a term that data-miners have given to the partitioning problem as it arose in graph theory. The theoretical framework for spectral clustering was laid in 1973 by Miroslav Fiedler [42, 43]. We will begin with a discussion of this early work, and then take a look at how others have adapted the framework to meet the needs of data clustering. In this setting, we have a graph \( G \) on a set of vertices \( N = \{1, 2, \ldots, n\} \) with edge set \( E = \{(i, j) : i, j \in N \text{ and } i \leftrightarrow j\} \). Edges between the vertices are recorded in an adjacency matrix \( A = (a_{ij}) \), where \( a_{ij} \) is equal to the weight of the edge connecting vertex (object) \( i \) and vertex \( j \) and \( a_{ij} = 0 \) if \( (i, j) \not\in E \).

A graph is called connected if there exists some path of edges connecting every pair of vertices, or equivalently if its adjacency matrix is irreducible. A graph that is not connected is said to have \( k \) connected components if there exists a permutation matrix \( P \) such that \( PAP^T \) is block diagonal with \( k \) diagonal blocks. In other words, a connected component is a set of vertices which are connected to each other but disconnected from the rest of the graph.

Spectral clustering algorithms typically involve the Laplacian matrix associated with a graph. A Laplacian matrix is defined as follows:

**Definition 4 (The Laplacian Matrix).** The Laplacian Matrix, \( L \), of an undirected, weighted graph with adjacency matrix \( A = (a_{ij}) \) and diagonal degree matrix \( D = \text{diag}(Ae) \) is:

\[
L = D - A
\]

The Laplacian matrix is symmetric, singular, and positive semi-definite. To see this third property, construct an \( n \times |E| \) “vertex-edge incidence” matrix \( U \) with rows corresponding to vertices and columns corresponding to edges. Allow the edges of the original graph to be
directed arbitrarily, and set
\[
U_{v,e} = \begin{cases} 
+\sqrt{a_{ij}} & \text{if } v \text{ is the head of } e \\
-\sqrt{a_{ij}} & \text{if } v \text{ is the tail of } e \\
0 & \text{otherwise}
\end{cases}
\]

Then \( L = UU^T \) and thus is positive semi-definite [42]. Alternatively, we can simply examine the nice quadratic form to which \( L \) gives rise, keeping in mind that \( a_{ij} \geq 0 \) \( \forall \ i, j \):

\[
y^T L y = \sum_{(i,j) \in E} a_{ij} (y_i - y_j)^2. \tag{3.1}
\]

Let \( \sigma(L) = \{ \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \} \) be the spectrum of \( L \). Since \( L \) is positive semi-definite, \( \lambda_i \geq 0 \) \( \forall i \). Also, since the row sums of \( L \) are zero, \( \lambda_1 = 0 \). Furthermore if the graph, \( G \), is composed of \( k \) connected components then \( \lambda_1 = \lambda_2 = \cdots = \lambda_k = 0 \) and \( \lambda_j \geq 0 \) for \( j \geq k + 1 \).

In [42] Fiedler defined the algebraic connectivity of the graph as the second smallest eigenvalue, \( \lambda_2 \), because its magnitude provides information about how easily the graph can be disconnected into two components by means of an edge cut. In other words, if \( \lambda_2 \) is very close to zero then the graph is almost disconnected or nearly uncoupled. This concept will be expanded upon and formalized in Chapter 7, for now the seed is merely planted for future development. Later work by Fiedler alluded to the utility of the eigenvector associated with \( \lambda_2 \) in determining an optimal two-component decomposition of a graph [43].

### 3.1.1 Fiedler Partitioning

Suppose we wish to decompose our graph into two components (or clusters of vertices) \( C_1 \) and \( C_2 \) where the edges exist more frequently and with higher weight inside the clusters than between the two clusters. In other words, we intend to make an edge-cut disconnecting the graph into two components. It is desired that the edge cut satisfies the following objectives:

1. Minimize the total weight of edges cut (edges between vertices in different components), or equivalently, maximize the total weight of edges remaining (edges between vertices in the same component)

2. Create components (i.e. clusters) of reasonable or balanced size. A cut which isolates a very small number of vertices in the graph is undesirable.

To begin with, let’s take the quadratic form from Eq. 3.1 and let \( y \) be a vector that determines the cluster membership of each vertex as follows:

\[
y_i = \begin{cases} 
+1 & : \text{if vertex } i \text{ belongs in } C_1 \\
-1 & : \text{if vertex } i \text{ belongs in } C_2
\end{cases} \tag{3.2}
\]
Our first objective is then to minimize Eq. 3.1 over all such vectors $y$:

$$
\min_y y^T Ly = \sum_{(i,j) \in E} a_{ij} (y_i - y_j)^2 = 2 \sum_{(i,j) \in E} 4a_{ij}
$$

(3.3)

Note that the final sum is doubled to reflect the fact that each edge connecting $C_1$ and $C_2$ will be counted twice. However, the above formulation is incomplete because it does not take into account the second objective, which is to somehow balance the number of vertices in each component. Indeed it seems the minimum solution to Eq. 3.3 would often involve cutting all of the edges adjacent to a single vertex of minimal degree, disconnecting the graph into components of size 1 and $n - 1$, which is generally undesirable. In addition, the above optimization problem is NP-hard. To solve the latter problem, the objective function is relaxed from discrete to continuous. Rather than partitioning the vertices according to Eq. 3.2, we instead relax the constraint and partition the vertices based upon the sign of their corresponding entry in the relaxed solution. By the Rayleigh theorem,

$$
\min_{\|y\|_2=1} y^T Ly = \lambda_1
$$

with

$$
y^* = \arg \min_{\|y\|_2=1} y^T Ly = v_1
$$

being the eigenvector corresponding to the smallest eigenvalue, $\lambda_1 = 0$. However, for the Laplacian matrix, $v_1 = \frac{1}{\sqrt{n}} e \geq 0$. In context, this makes sense - in order to minimize the weight of edges cut, we should simply assign all vertices to one cluster, leaving the second cluster empty. In order to divide the vertices into two clusters we need an additional constraint on $y$. Since clusters of relatively balanced size are desirable, a natural constraint is $y^T e = 0$. By the Courant-Fischer theorem,

$$
\min_{\|y\|_2=1, y^T e = 0} y^T Ly = \lambda_2
$$

(3.4)

with $y^* = v_2$ being the eigenvector corresponding to the second smallest eigenvalue, $\lambda_2$. This vector is often referred to as the Fiedler vector after the man who identified its usefulness in graph partitioning. We define the Fiedler graph partition as follows:

**Definition 5** (Fiedler Graph Partition). Let $G = (N, E)$ be a connected graph on vertex set $N = \{1, 2, \ldots, n\}$ with adjacency matrix $A$. Let $L = D - A$ be the Laplacian matrix of $G$. Let $v_2$ be an eigenvector corresponding to the second smallest eigenvalue of $L$. The Fiedler partition is:

$$
C_1 = \{i \in N : v_2(i) < 0\}
$$

(3.5)

$$
C_2 = \{i \in N : v_2(i) > 0\}
$$

(3.6)

Vertices $j$, for which $v_2(j) = 0$, can be arbitrarily placed into either cluster.
There is no uniform agreement on how to determine the cluster membership of vertices for which \( v_2(j) = 0 \). The decision to make the assignment arbitrarily comes from experimental results that indicate in some scenarios these zero valuated vertices are equally drawn to either cluster. Situations where there are a large proportion of zero valuated vertices may be indicative of a graph which does not conform well to Fiedler’s partition, and we suggest the user tread lightly in these cases. Figure 3.1 shows the experimental motivation for our arbitrary assignment of zero valuated vertices. The vertices in these graphs are labelled according to the sign of the corresponding entry in \( v_2 \). We highlight the red vertex in the center and watch how its sign in \( v_2 \) changes as nodes and edges are added to the graph.

![Figure 3.1: Fiedler Partitions and Zero Valuated Vertices](image)

In order to create more than two clusters, the Fiedler graph partition can be performed iteratively by examining the subgraphs induced by the vertices in \( C_1 \) and \( C_2 \) and partitioning each based upon their own Fiedler vectors. The iterative method requires a cluster to be chosen for further division, perhaps based upon the algebraic connectivities of the clusters. This approach would lead to a divisive hierarchical algorithm like PDDP (Section 2.1.2). It is also possible to use the sign patterns in subsequent eigenvectors to further partition the graph. This approach is called Extended Fiedler Clustering and is discussed in the next section.
Extended Fiedler Clustering

In the extended Fiedler algorithm, we use the sign patterns of entries in the first $l$ eigenvectors of $L$ to create up to $k = 2^l$ clusters. For instance, suppose we had 10 vertices, and used the $l = 3$ eigenvectors $v_2, v_3,$ and $v_4$. Suppose the sign of the entries in these eigenvectors are recorded as follows:

\[
\begin{pmatrix}
1 & + & + & - \\
2 & - & + & + \\
3 & + & + & + \\
4 & - & - & - \\
5 & - & - & - \\
6 & + & + & - \\
7 & - & - & - \\
8 & - & + & + \\
9 & + & - & + \\
10 & + & + & + \\
\end{pmatrix},
\]

Then the 10 vertices are clustered as follows:

\{1, 6\}, \{2, 8\}, \{3, 10\}, \{4, 5, 7\}, \{9\}.

Extended Fiedler makes clustering the data into a specified number of clusters $k$ difficult, but may be able determine a natural choice for $k$ as it partitions the data along several eigenvectors.

In a 1990 paper by Pothen, Simon and Liou, an alternative formulation of the Fiedler partition is proposed [103]. Rather than partition the vertices based upon the sign of their corresponding entries in $v_2$, the vector $v_2$ is instead divided at its median value. The main motivation for this approach was to split the vertices into sets of equal size. In 2003, Ding et al. derived an objective function for determining an ideal split point for similar partitions using the second eigenvector of the normalized Laplacian, defined in Section 3.1.2 [32]. The basic idea outlined above has been adapted and altered hundreds if not thousands of times in the past 20 years. The present discussion is meant merely as an introduction to the literature.

3.1.2 Other Graph Cuts

The majority of spectral algorithms are derived by altering the objective function in Eq. 3.4. The idea is the same: partition the graph into two components by means of a minimized edge cut, while requiring that the two components remain somewhat balanced in size (i.e. do not simply isolate a small number of vertices). Two common objective functions which embody this idea are the ratio cut (RatioCut) [56], the normalized cut (Ncut) [112]
Ratio Cut

The ratio cut objective function was first introduced by Hagen and Kahng in 1992 [56]. Given a graph \( G(V, E) \) with vertex set \( V \) partitioned into \( k \) disjoint clusters, \( V_1, V_2, \ldots, V_k \), the ratio cut of the given partition is defined as

\[
\text{RatioCut}(V_1, V_2, \ldots, V_k) = \sum_{i=1}^{k} \frac{w(V_i, \bar{V}_i)}{|V_i|}
\]

where \( |V_i| \) is the number of vertices in \( V_i \), \( \bar{V}_i \) is the complement of the set \( V_i \) and, given two vertex sets \( A \) and \( B \), \( w(A, B) \) is the sum of the weights of the edges between vertices in \( A \) and vertices in \( B \). Let \( H \) be an \( n \times k \) matrix indicating cluster membership of vertices by its entries:

\[
H_{ij} = \begin{cases} \frac{1}{\sqrt{|V_j|}}, & \text{if the } i^{th} \text{ vertex is in cluster } V_j \\ 0, & \text{otherwise} \end{cases}
\] (3.7)

Then \( H^TH = I \) and minimizing the ratio cut over all possible partitionings is equivalent to minimizing

\[
f(H) = \text{Trace}(H^TLH)
\]

over all matrices \( H \) described by Eq. 3.7, where \( L \) is the Laplacian matrix from Definition 4. The exact minimization of this objective function is again NP-hard, but relaxing the conditions on \( H \) to \( H^TH = I \) yields a solution \( H^* \) with columns containing the eigenvectors of \( L \) corresponding to the \( k \) smallest eigenvalues.

Unfortunately, after this relaxation it is not necessarily possible to automatically determine from \( H^* \) which vertices belong to each cluster. Instead, it is necessary to look for clustering patterns in the rows of \( H^* \). This is a common conceptual drawback of the relaxation of objective functions in spectral clustering. The best way to proceed after the relaxation is to cluster the rows of \( H^* \) with an algorithm like \( k \)-means to determine a final clustering. The ratio cut minimization method is generally referred to as unnormalized spectral clustering [124]. The algorithm is as follows:

Normalized Cut (Ncut)

The normalized cut objective function was introduced by Shi and Malik in 2000 [112]. Given a graph \( G(V, E) \) with vertex set \( V \) partitioned into \( k \) disjoint clusters, \( V_1, V_2, \ldots, V_k \), the normalized cut of the given partition is defined as

\[
\text{Ncut}(V_1, V_2, \ldots, V_k) = \sum_{i=1}^{k} \frac{w(V_i, \bar{V}_i)}{\text{vol}(V_i)},
\]

where \( \text{vol}(V_i) \) is the sum of the weights of the edges connecting the vertices in \( V_i \). Whereas the size of a subgraph \( V_i \) in the ratio cut formulation is measured by the number of vertices

32
Algorithm 8 Unnormalized Spectral Clustering (RatioCut) [124]

**Input:** $n \times n$ adjacency (or similarity) matrix $A$ for a graph on vertices (or objects) \{1, \ldots, n\} and desired number of clusters $k$

1. Compute the Laplacian $L = D - A$.
2. Compute the first $k$ eigenvectors $V = v_1, v_2, \ldots, v_k$ of $L$ corresponding to the $k$ smallest eigenvalues.
3. Let $y_i$ be the $i^{th}$ row of $V$
4. Cluster the points $y_i \in \mathbb{R}^k$ with the $k$-means algorithm into clusters $\bar{C}_1, \ldots, \bar{C}_k$.

**Output:** Clusters $C_1, \ldots, C_k$ such that $C_j = \{i : y_i \in \bar{C}_j\}$

$|V_i|$, in the normalized cut formulation it is measured by the total weight of the edges in the subgraph. Thus, minimizing the normalized cut is equivalent to minimizing

$$f(H) = \text{Trace}(H^T L H)$$

over all matrices $H$ with the following form:

$$H_{ij} = \begin{cases} 
\frac{1}{\sqrt{\text{vol}(V_j)}}, & \text{if the } i^{th} \text{ vertex is in cluster } V_j \\
0 & \text{otherwise.}
\end{cases}$$

With $H^T D H = I$ where $D$ is the diagonal degree matrix from Definition 4. Thus, to relax the problem, we substitute $G = D^{1/2} H$ and minimize

$$f(G) = G^T L G$$

subject to $G^T G = I$, where $L = D^{-1/2} L D^{-1/2}$ is called the normalized Laplacian. Similarly, the solution to the relaxed problem is the matrix $G^*$ with columns containing eigenvectors associated with the $k$ smallest eigenvalues of $L$. Again, the immediate interpretation of the entries in $G^*$ is lost in the relaxation and so a clustering algorithm like $k$-means is used to determine the patterns.

Other Normalized Cuts

While Algorithm 9 carries “normalized cut” in its title, other researchers have suggested alternative ways to consider normalized cuts in a graph. In a popular 2001 paper, Ng, Jordan, and Weiss made a slight alteration of the previous algorithm which simply normalized the
Algorithm 9 Normalized Spectral Clustering (Ncut) [124]

\textbf{Input}: \( n \times n \) adjacency (or similarity) matrix \( A \) for a graph on vertices (or objects) \( \{1, \ldots, n\} \) and desired number of clusters \( k \)

1. Compute the \textit{normalized} Laplacian \( \mathcal{L} = D^{-1/2}LD^{-1/2} \).
2. Compute the first \( k \) eigenvectors \( V = [v_1, v_2, \ldots, v_k] \) of \( \mathcal{L} \) corresponding to the \( k \) smallest eigenvalues.
3. Let \( y_i \) be the \( i^{th} \) row of \( V \)
4. Cluster the points \( y_i \in \mathbb{R}^k \) with the \( k \)-means algorithm into clusters \( \tilde{C}_1, \ldots, \tilde{C}_k \).

\textbf{Output}: Clusters \( C_1, \ldots, C_k \) such that \( C_j = \{i : y_i \in \tilde{C}_j\}\)

In 2001, Meilă and Shi altered the objective function once again, and derived yet another spectral algorithm using the \textit{normalized random walk} Laplacian, \( \mathcal{L}_{rw} = D^{-1}L = I - D^{-1}A \) [84]. As shown in [125], if \( \lambda \) is an eigenvalue for \( \mathcal{L} \) with corresponding eigenvector \( v \) then \( \lambda \) is also an eigenvalue for \( \mathcal{L}_{rw} \) with corresponding eigenvector \( D^{1/2}v \). This formulation amounts
to a different scaling of the eigenvectors in step 3 of Algorithm 10. This normalized random walk Laplacian will present itself again in Section 3.1.3. Meilă and Shi’s spectral clustering method is outlined in Algorithm 11.

Algorithm 11 Normalized Spectral Clustering according to Meilă and Shi (MS)

**Input:** $n \times n$ adjacency (or similarity) matrix $A$ for a graph on vertices (or objects) \{1, \ldots, n\} and desired number of clusters $k$

1. Compute the normalized random walk Laplacian $L_{rw} = D^{-1}L$.
2. Compute the first $k$ eigenvectors $V = [v_1, v_2, \ldots, v_k]$ of $L_{rw}$ corresponding to the $k$ smallest eigenvalues.
3. Normalize the rows of $V$ to have unit 2-norm.
4. Let $y_i$ be the $i^{th}$ row of $V$.
5. Cluster the points $y_i \in \mathbb{R}^k$ with the $k$-means algorithm into clusters $\tilde{C}_1, \ldots, \tilde{C}_k$.

**Output:** Clusters $C_1, \ldots, C_k$ such that $C_j = \{i : y_i \in \tilde{C}_j\}$

All of the spectral algorithms outlined thus far seem very similar in their formulation, yet in practice they tend to produce quite different results. This presents a problem because while each method has merit in its own right, it is impossible to predict which one will work best on any particular graph. We will discuss this problem further in Chapter 7.

3.1.3 Power Iteration Clustering

In a 2010 paper, Frank Lin and William Cohen propose a fast, scalable algorithm for clustering graphs using the power method (or power iteration) [39]. Let $W = D^{-1}A$ be the $n \times n$ row-normalized (row stochastic) adjacency matrix for a graph, and let $v_0 \neq 0$ be a vector in $\mathbb{R}^n$. A simple method for computing the eigenvector corresponding to the largest eigenvalue of $W$ is the power method, which repeatedly computes the power iteration

$$v_{t+1} = cv_t$$

where $c = 1/\|Wv_t\|_1$ is a normalizing constant to prevent $v_t$ from growing too large.

Applying the power method to convergence on $W$ would result in the uniform vector $\alpha e$ where $\alpha = 1/n$. However, stepping through a small number of power iterations, will result in a vector that contains combined information from the eigenvectors associated with the largest eigenvalues. The formulation of Meilă and Shi’s spectral algorithm in [84] warranted the use
of the eigenvectors corresponding to the \( k \) smallest eigenvalues of the normalized random walk Laplacian \( \mathcal{L}_{rw} = I - W \) which is equivalent to the consideration of the eigenvectors of the largest eigenvalues of \( W \). Thus, the idea behind Power Iteration Clustering (PIC) is to detect and stop the power method at some number of iterations \( t \) such that \( v_t \) is a useful linear combination of the first \( k \) eigenvectors. The analysis in [39] motivates the idea that the power method should pass through some initial stage of local convergence at the cluster level before going on to the stage of global convergence toward the uniform vector. At this stopping point, it is expected that \( v_t \) will be an approximately piecewise constant vector, nearly uniform on each of the clusters. Thus, the clusters at this stage will be revealed by the closeness of their corresponding entries in \( v_t \). See [39] for the complete analysis. The PIC procedure is given in Algorithm 12. An example of a graph, and several iterations of the power method on the matrix \( W \) corresponding to that graph is shown in Figure 3.2.

**Algorithm 12** Power Iteration Clustering (PIC) [39]

**Input:** A row-stochastic matrix \( P = D^{-1}A \) where \( A \) is an adjacency or similarity matrix and the number of clusters \( k \).

1. Pick an initial vector \( v_0 \). [39] suggests the degree vector \( v_0 = Ae \).

2. Set \( v_{t+1} = \frac{Pv_t}{\|Pv_t\|_1} \) and \( \delta_{t+1} = |v_{t+1} - v_t| \).

3. Increment \( t \) and repeat step 2 until \( |\delta_t - \delta_{t+1}| \approx 0 \).

4. Use \( k \)-means to cluster points on \( v_t \) and return clusters \( C_1, C_2, \ldots, C_k \).

**Output:** Clusters \( C_1, C_2, \ldots, C_k \).

### 3.1.4 Clustering via Modularity Maximization

Another technique proposed in the network community detection literature compares the structure of a given graph to what one may expect from a random graph on the same vertices [94, 102]. The motivation for this method was that simply counting edges between clusters as was done in previous spectral methods may not be the best way to define clusters in graph. A better approach may be to somehow measure whether they are fewer edges than *expected* between communities. Let \( A \) be the adjacency matrix of the graph (or network) and let \( P \) be the adjacency matrix of a random graph on the same vertices containing the expected value of weights on that graph. Then the matrix \( B = A - P \) would contain information about how the structure of \( A \) deviates from what is expected. Obviously this formulation relies on some underlying probability distribution of the weights in the random graph, known as the *null*
model. The most common null model uses the degree sequence of the vertices in the given graph, \( \{d_1, d_2, \ldots, d_n\} \), where \( d_i \) is the degree of vertex \( i \) (i.e. \( d_i \) is the sum of the weights of the edges connected to vertex \( i \)), to create the probabilities [102, 94]

\[
p(\text{edge}(i,j)) = \frac{d_j}{\sum_{k=1}^{n} d_k}. \tag{3.8}
\]

Thus, the expected value of the weight of the edge from \( i \) to \( j \) is

\[
P_{ij} = E(w(i,j)) = d_i \left( \frac{d_j}{\sum_{k=1}^{n} d_k} \right).
\]
One may recognize that the probabilities in Eq. 3.8 are precisely the stationary probabilities of the random walk on the graph defined by \( A \), and thus seem a reasonable choice for a null model. This formulation gives us \( E(w(i,j)) = E(w(j,i)) \) as desired for an undirected graph. Using this null model, a *modularity matrix* \( B \) is formed as

\[
B = A - P.
\]

For a division of the data into two clusters, let \( s \) be an \( n \times 1 \) vector indicating cluster membership by

\[
s_i = \begin{cases} 
-1 & : \text{vertex } i \text{ belongs in cluster 1} \\
1 & : \text{vertex } i \text{ belongs in cluster 2}
\end{cases}
\]

Let \( d = \sum_{k=1}^{n} d_k \). The *modularity* of a given partition is defined by

\[
Q = \frac{1}{2d} \sum_{ij} B_{ij} s_i s_j = \frac{1}{2d} s^T Bs. \tag{3.9}
\]

The goal of the algorithm proposed in [94] is to maximize this quantity, thus we can drop the constant \( 1/2d \) and write the objective as

\[
\max_Q Q = s^T Bs \tag{3.10}
\]

**Illustrative Example**

To get an idea of why this is true, consider the case where we have two relatively obvious clusters \( C_1 \) and \( C_2 \) in a graph and reorder the rows and columns of the adjacency matrix to reflect this structure,

\[
A = \begin{bmatrix} 
A_{C_1} & E \\
E^T & A_{C_2}
\end{bmatrix}
\]

Where \( A_{C_1} \) and \( A_{C_2} \) are relatively dense matrices with larger entries representing the weight of edges within the clusters \( C_1 \) and \( C_2 \) respectively and \( E \) is a sparse matrix with smaller entries representing the weight of edges which connect the two clusters. In a random graph with no community or cluster structure, we’d be likely to find just as many edges between the clusters as within clusters. Thus, after subtracting \( P \) our modularity matrix may look something like

\[
B = \begin{bmatrix} 
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix} \approx \begin{bmatrix} 
+ & - \\
- & +
\end{bmatrix}
\]

Where the indicated signs reflect the sign *tendancy* of values in \( B \). In other words, the entries in the diagonal blocks \( B_{11} \) and \( B_{22} \) tend to be positive because the edges within clusters had larger weights than one would expect at random and the entries in the off diagonal blocks \( B_{12} \)
and $B_{21}$ tend to be negative because the edges between clusters had smaller weights than one would expect at random. Thus, the modularity of this graph, $Q = s^T B s$, will be maximized by the appropriate partition $s^T = [s_1^T, s_2^T] = [e_{C_1}^T, -e_{C_2}^T]$.

In order to maximize the modularity objective function given in Eq. 3.10, let $u_1, u_2, \ldots, u_n$ be an orthonormal set of eigenvectors for $B$ corresponding respectively to the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Write the vector $s$ as a linear combination of eigenvectors,

$$s = \sum_{i=1}^{n} \alpha_i u_i$$

where

$$\alpha_i = u_i^T s.$$

Then, the objective function from Eq. 3.10 becomes

$$\max_{s_i = \pm 1} \left( \sum_{i=1}^{n} \alpha_i u_i^T B \right) \left( \sum_{i=1}^{n} \alpha_i u_i \right) = \sum_{i=1}^{n} \lambda_i (u_i^T s)^2.$$

This optimization is NP-hard due to the constraint that $s_i = \pm 1$. It is clear that without this constraint one would choose $s$ proportional to $u_1$, maximizing the first term in the summation (associated with the largest eigenvalue) and terminating the others. A reasonable way to proceed in light of this information is to maximize the leading term and ignore the remaining terms. To accomplish this, it is quite clear that we should choose $s$ so that its entries match the signs of the entries in $u_1$. The placement of vertices corresponding to zero entries in $u_1$ will be decided arbitrarily. However, if the leading eigenvalue of the modularity matrix is negative then the corresponding eigenvector is $e$, leading us to no partition. According to [94] the “no partition” solution in this scenario is in fact the correct result, i.e. a negative leading eigenvalue indicates there is no community or cluster structure in the graph. This gives a clear stopping point for the procedure, which allows it to automatically determine an appropriate number of clusters or communities to create. Unfortunately, the arbitrary placement of vertices corresponding to zero entries in $u_1$ may in some cases affect the determined number of clusters.

To create more than 2 clusters, the above procedure can be repeated on each of the subgraphs induced by the vertices in each cluster found. This leads us to an iterative divisive (hierarchical) algorithm like the iterative Fiedler method in Section 3.1.1 and PDDP in Section 2.1.2. The modularity clustering procedure is formalized in Algorithm 13.
Algorithm 13 Modularity Procedure for Network Community Detection (Newman) [94]

**Input:** $n \times n$ adjacency matrix $A$ for an undirected graph to be partitioned

1. Let $d_i$ be the $i^{th}$ row sum of $A$. Let $d = \sum_{i=1}^{n} d_i$
2. Form the matrix $P$ with $P_{ij} = d_i d_j / d$.
3. Form the modularity matrix $B = A - P$.
4. Compute the largest eigenvalue $\lambda_1$ and corresponding eigenvector $u_1$ of $B$.
5. If $\lambda_1 < 0$, stop. There is no partition of this graph.
6. Otherwise partition the vertices of the graph into 2 clusters as follows

$$C_1 = \{ i : u_1(i) < 0 \}$$
$$C_2 = \{ i : u_1(i) \geq 0 \}$$

(3.11)

7. Determine further partitions by extracting the rows and columns of the original adjacency matrix corresponding to the vertices in each cluster to form $A'$ and repeat the algorithm with $A'$ until each created cluster fails to partition in step 5.

**Output:** Final clusters.
3.2 Stochastic Clustering

An alternative way to interpret a graph is by considering a random walk along the edges. For an undirected graph with adjacency matrix $A$, we can create a transition probability matrix $P$ by dividing each row by the corresponding row sum. Using the degree matrix from Definition 4 we have

$$P = D^{-1}A.$$  

If our graph does indeed have some cluster structure, i.e. sets of vertices $C_1, C_2, \ldots, C_k$ for which the total weight of edges within each set are substantially higher than the total weight of edges between the different sets, then a random walker in a given cluster $C_i$ is more likely to stay in $C_i$ for several steps than he is to transition to another cluster $C_j$. It is well known that for a connected and undirected graph, the long term probability distribution is given by

$$\pi^T = \frac{e^T D e^T}{e^T D e^T}$$

Which is not likely to give any cluster information. However, the short-term evolution of this walk can tell us something about the cluster structure because a random walker is far more likely, in the short-run, to remain inside a cluster than he is to transition between clusters. The Stochastic Clustering Algorithm (SCA) of Wessell and Meyer [85] takes advantage of this fact.

3.2.1 Stochastic Clustering Algorithm (SCA)

In a 2012 paper, Chuck Wessell and Carl Meyer formulated a clustering model by creating a symmetric (doubly stochastic) transition matrix $P$ [85, 127] from the adjacency matrix of a graph. The method in this paper is quite similar to that in PIC except that here the mathematics of the “backward” Markov Chain intuition given in [39] works out in this context because the probability transition matrix is symmetric. One added feature in this algorithm is the automatic determination of the number of clusters in the data, using eigenvalues of the transition matrix $P$. Wessell and Meyer’s formulation is based on theory that was developed by Nobel Laureate economist Herbert Simon and his student Albert Ando. This theory surrounds the mixing rates of resources or wealth in local economies (composed of states in a Markov chain) as part of a global economy (which links together some states from each local economy). It is assumed that the adjacency matrix for the graph is irreducible, or equivalently that the graph is connected.

The basic idea is that resources will be exchanged more frequently at a local level than they will at the global level. Suppose individual companies from a global economy are represented as nodes in a graph with edges between them signifying the amount of trade between each pair of companies. Natural clusters would form in this graph at a local level, represented by the strong and frequent trade relationships of proximal companies. Let $k$ be the number of local economies (clusters), each containing $n_i$ states $i = 1, \ldots, k$, and define the distribution
of resources at time $t$ as $\mathbf{f}_t$, given a starting distribution $\mathbf{f}_0$. Then

$$\mathbf{f}_t^T = \mathbf{f}_0^T \mathbf{P}$$

The heavily localized trade in this global economy leads to a so-called short-term stabilization of the system characterized by a distribution vector at some time $t$ which is nearly constant across each local economy:

$$\mathbf{f}_t^T \approx \left( \frac{\alpha_1}{n_1} \frac{\alpha_1}{n_1} \ldots \frac{\alpha_1}{n_1} | \frac{\alpha_2}{n_2} \frac{\alpha_2}{n_2} \ldots \frac{\alpha_2}{n_2} | \ldots | \frac{\alpha_k}{n_k} \frac{\alpha_k}{n_k} \ldots \frac{\alpha_k}{n_k} \right)$$

After this short-term stabilization, the distribution of goods in the Markov Chain is eventually expected to converge to a constant level across every state. However, in the period following the short-run stabilization, the distribution vector retains its approximately piecewise constant structure for a some time before settling down into its final uniform equilibrium.

Wessell and Meyer’s derivation requires the creation of a symmetric probability transition matrix $\mathbf{P}$ from the adjacency matrix $\mathbf{A}$ by means of a simultaneous row and column scaling. In other words, a diagonal matrix $\mathbf{S}$ is determined for which

$$\mathbf{S} \mathbf{A} \mathbf{S} = \mathbf{P}$$

is a doubly stochastic transition probability matrix. This task turns out to be quite simple, $\mathbf{S}$ is found by iterating a single step until convergence. Letting $\mathbf{S}_{ii} = s(i)$, the diagonal scaling procedure put forth by Ruiz [108] is simply:

$$s_0 = \mathbf{e}$$

$$s_{t+1}(i) = \sqrt{\frac{s_t(i)}{\mathbf{A}_{ii} s_t}}$$

(3.12)

In [127], it is convincingly argued that the diagonal scaling procedure does not change the underlying cluster structure of the data in $\mathbf{A}$, and thus that the desired information is not damaged by this transformation. The clusters in this method are found in a similar manner to PIC, where $k$-means is employed to find the nearly piecewise constant segments of the distribution vector $\mathbf{f}_t$ after a short number of steps. The Stochastic Clustering Algorithm automatically determines the number of clusters in the data by counting the number of eigenvalues whose value is close to $\lambda_1 = 1$. This group of eigenvalues near 1 is referred to as the Perron cluster. We postpone discussion of this matter to Chapter 6 where it will be analyzed in detail. For now, we present the Stochastic Clustering method in Algorithm 14. The eigenvector iterations in SCA are quite similar to those put forth in PIC, and users commonly create visualizations of the iterations that look quite similar to those in Figure 3.2.
Algorithm 14 Stochastic Clustering Algorithm (SCA) [85]

**Input:** Adjacency matrix $A$ for some graph to be partitioned

1. Convert $A$ to a symmetric probability transition matrix $P$ using the diagonal scaling procedure given in Eq. 3.12.
2. Calculate the eigenvalues of $P$ and determine $k$ to be the number of eigenvalues in the Perron cluster.
3. Create a random initial probability distribution $\mathbf{\beta}_0^T$.
4. Track the evolution of $\mathbf{\beta}_{t+1}^T = \mathbf{\beta}_t^T P$. After each multiplication, cluster the entries of $\mathbf{\beta}_t$ using $k$-means. When this clustering has remained the same for a user-preferred number of iterations stop.

**Output:** $k$ clusters found $C_1, C_2, \ldots, C_k$. 

---

43
Chapter 4

Dimension Reduction

A large proportion of modern data problems arise out of high-dimensional data. Common examples include text mining, image processing, and genetic analysis. High-dimensional data is data in which observations exist in high-dimensional space, i.e. each observation or object is defined by a large number of attributes. For example, even a small collection of documents may contain thousands of different words which become their attributes in the Term-Document matrix from Definition 2. In this chapter, we will examine a problematic phenomenon occurring in high-dimensional space, appropriately dubbed the *curse of dimensionality*, and discuss some potential solutions.

4.1 The Curse of Dimensionality

The phrase “*curse of dimensionality*” was first coined by Richard Bellman in 1957 to generally describe the difficulty in determining optimal solutions when the number of variables at play is large [8]. More recent research related to clustering, nearest neighbor search and indexing suggest that the very concepts of distance and proximity may not even be meaningful in high-dimensional spaces [126, 12, 3]. Here, we will try to illustrate the strange behavior of volumes, distances and similarities as the dimensionality of the space increases using some illustrative examples.

4.1.1 Volume

To investigate the behavior of volumes in high-dimensional space, we’ll use two simple examples.
Example 1

The following example is adapted from the “range queries” observation in [126]. Suppose we have a set of equally spaced lattice points sampling the unit hypercube. What happens to the proportion of the data contained in the hypercube with edge length \( l = \frac{1}{2} \) as the dimension of the space increases?

In one dimension the unit hypercube is merely a line segment of length one, so 50% of the data points (lattice points) can be found in a given interval of length \( l = \frac{1}{2} \). In two dimensions, a square with edge length \( l = \frac{1}{2} \) covers only 25% of the data in the unit square. Let \( l_n(p) \) be the edge length of the \( n \)-cube required to contain some fraction, \( p \), of the lattice points of the unit \( n \)-cube. Then

\[
l_n(p) = p^{1/n}.
\]

Thus, to contain only 1% of the data in the unit \( n \)-cube with \( n = 100 \) would require an \( n \)-cube with edge length \( l = 0.95 \)! In other words, 99% of the lattice points in the unit hypercube in 100 dimensions are not contained in the hypercube with edge length \( l = 0.95 \). Unfortunately, \( n = 100 \) is hardly a “high” number compared with the tens or even hundreds of thousands of dimensions in some datasets. This phenomenon clearly gets worse as the dimension of the space increases further, as \( l_n(p) \to 1 \) for any \( p \neq 0 \) as \( n \to \infty \). This gives us a sense for the incredible vastness of \( n \)-space. It’s as though nearly all of the high-dimensional space is far away from the center!

Example 2

Suppose we have a hypersphere with radius \( r \) and a hypercube with sides of length \( 2r \) in \( n \)-space. When \( n = 3 \), the picture looks something like this:

The ratio of the volume of the sphere to the volume of the cube is:

\[
\frac{\text{Sphere Volume}}{\text{Cube Volume}} = \frac{(4/3)\pi r^3}{8r^3} = \frac{\pi}{6}
\]
Now, in higher dimensional space, the volume of an $n$-sphere of radius $r$ is given by:

$$\text{Volume of n-sphere} = \frac{2r^n \pi^{n/2}}{n^\Gamma\left(\frac{n}{2}\right)}$$

whereas the volume of the $n$-cube is $(2r)^n$. Thus, in $n$ dimensional space, the ratio of the volume of the hypersphere to the volume of the cube is:

$$\frac{\text{Sphere Volume}}{\text{Cube Volume}} = \frac{\pi^{n/2}}{n^2 \Gamma\left(\frac{n}{2}\right)^{n-1}} \xrightarrow{n \to \infty} 0.$$

It’s as if nearly all of the volume of the hypercube is contained in the corners!

### 4.1.2 Distance and Similarity

As the dimensionality $n$ of the space increases, distance metrics suffer from the same strange phenomenon. Take for example $p$ points uniformly distributed in the unit hypercube, $[0,1]^n$: \(\{x_1, x_2, \ldots , x_p\}, \ x_j \in \mathbb{R}^{n \times 1}\). Let $\text{dist}_{ij} = \|x_i - x_j\|_2$ and define $\text{dist}_{\text{max}} = \max_{i,j} \text{dist}_{ij}$ and $\text{dist}_{\text{min}} = \min_{i,j} \text{dist}_{ij}$. Then the following is true [12]:

$$\lim_{n \to \infty} \frac{\text{dist}_{\text{max}} - \text{dist}_{\text{min}}}{\text{dist}_{\text{min}}} = 0$$

In order to visualize this situation, let’s do a quick experiment using 100 randomly generated points, $x_1, x_2, \ldots , x_{100}$, uniformly distributed in the unit $n$-cube. Figure 4.1 shows the distributions, for various $n$, of the quantities:

$$\hat{d}_{ij} = \frac{\text{dist}_{ij}}{\text{dist}_{\text{max}}}.$$  

It is clear from Figure 4.1 that as the size of the space grows and the points become further and further apart, the pairwise distance between points becomes close to uniform. A similar effect happens with the angular distance between points, measured by the cosine similarity:

$$\cos(x_i, x_j) = \frac{x_i^T x_j}{\|x_i\| \|x_j\|}.$$  

Figure 4.2 illustrates the distributions of the cosine similarity for the same 100 points, $\{x_i\}$, uniformly distributed in the unit $n$-cube for various $n$.

Since the vast majority of clustering algorithms are based upon distance and similarity measures, this so called dimensionality curse poses a major problem for the analysis of massive datasets. The best way to circumvent this curse is to reduce the dimensionality (number of attributes) of the data in some way. In the next section, we will take a look at some common approaches to dimension reduction.
Figure 4.1: Distribution of $\hat{d}_{ij}$ for 100 random points in $n$-space
Figure 4.2: Distribution of $\cos(x_i, x_j)$ for 100 random points in $n$-space
4.2 Dimension Reduction

The idea of dimension reduction is simple: reduce the number of features (attributes or variables) which define each object or observation. There are many options available to users for this task. Classical techniques for this task fall along 2 different frameworks:

1. Feature Selection
2. Feature Extraction

**Feature selection** is the process of choosing a subset of the existing features which are in some sense more important than other features. The simplest example of feature selection in text-mining might be to discard words in the dictionary which occur only in a single document, as these words do not help the user to identify patterns across multiple documents. It is also common practice to discard common words like “the” and “that”. This will reduce the dimensionality of the data to a certain extent, but will not generally be enough to escape the curse. Some researchers have used term weighting to rank the importance of words, and discard words whose rank was below a certain threshold. Feature selection algorithms tend to be application dependent, and often require domain expertise to implement. While the simplest forms of feature selection will be used in our experiments, more dramatic dimension reductions will result from feature extraction.

**Feature extraction** is the process of creating new features as linear combinations of existing features. Feature extractions can generally be viewed as projections or approximations of the data in lower dimensional spaces. In this section, we will focus on 3 such approximations: Singular Value Decomposition (SVD), Principal Components Analysis (PCA), and Nonnegative Matrix Factorization (NMF).

4.2.1 Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) has long been considered as a technique for noise-reduction in signal processing. We will quickly review this important fact: Let $X = [x_1, x_2, \ldots, x_n]$ be our matrix of column data. Recalling Definition 1, let the SVD of $X$ be

$$X = U \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} V^T$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_R u_R v_R^T$$

$$= \sigma_1 Z_1 + \sigma_2 Z_2 + \cdots + \sigma_R Z_R$$

(4.1)
where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_R > 0$, $R=\text{rank}(X)$, and the matrices $Z_i = u_i v_i^T$ form an orthonormal set because

$$\text{Trace}(Z_i^T Z_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}.$$ 

Interpreting this as a Fourier expansion, $\sigma_i$ gives us the proportion of the signal in $X$ lying in the direction of $Z_i$. Assuming that the noise in the signal is random, or uniformly distributed across the components $Z_i$, then each term $\sigma_i Z_i$ contains approximately the same amount of noise. Thus, the signal-to-noise ratio decreases which each successive term in Eq. 4.1. The implication of this is that if some of the singular values $\sigma_{r+1}, \ldots, \sigma_R$ are small (relative to the amount of noise per component), then truncating the SVD of a signal $X$ to

$$X \approx X_r = U_r S_r V_r^T$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_r u_r v_r^T$$

$$= \sigma_1 Z_1 + \sigma_2 Z_2 + \cdots + \sigma_r Z_r$$

will result in an approximation to the signal, $X_r$, in which some of the noise is filtered out without the loss of significant information [87].

An additional property of the truncated SVD which makes it appealing for dimension reduction is that the rank $r$ truncation, $X_r$, in Eq. 4.2, is the nearest (in the Euclidean sense) rank $r$ matrix to the original matrix, $X$:

$$\min_{\text{rank}(A) = r} \|X - A\|_2 = \sigma_{r+1} \implies \arg \min_{\text{rank}(A) = r} \|X - A\|_2 = X_r$$

Similarly, the approximation $X_r$ gives us the closest entry-wise approximation to $X$ by:

$$\min_{\text{rank}(A) = r} \|X - A\|_F^2 = \sum_{i,j} |A_{ij} - X_{ij}|^2 = \sum_{i=r+1}^R \sigma_i \quad \text{and} \quad \min_{\text{rank}(A) = r} \|X - A\|_F = X_r$$

The truncated SVD provides us with a new coordinate representation of each object or observation (i.e. column vector $x_j$) in the orthonormal basis for the $r$-dimensional subspace spanned by the first $r$ left singular vectors of $X$:

$$x_j \approx (\sigma_1 v_{1j}^T) u_1 + (\sigma_2 v_{2j}^T) u_2 + \cdots + (\sigma_r v_{rj}^T) u_r.$$ (4.3)

Thus, the columns of the $r \times n$ matrix $S_r V_r^T$ form a lower dimensional coordinate representation of the data which we can use as input to our clustering algorithms. The columns of $U_r$ act as the (extracted) features of the data.
Applications to Text Data: Latent Semantic Indexing (LSI)

The noise-reduction property of the SVD was extended to text processing in 1990 by Susan Dumais et al, who named the effect Latent Semantic Indexing (LSI) [28]. It certainly seems logical to view text data in this context as it contains both an informative signal and semantic noise. LSI quickly grew roots in the information retrieval community, where it is often used for query processing. The idea is to remove semantic noise, due to variation in ambiguity in vocabulary and presentation style, without losing significant amounts of information. For example, a human may not differentiate between the words “car” and “automobile”, but indeed the words will become two separate entities in the raw term-document matrix. An additional motivation for LSI is that the realignment of the data into fewer directions should force related documents (like those containing “car” and “automobile”) closer together in an angular sense, thus revealing latent semantic connections.

Purveyors of LSI suggest that the use of the Singular Value Decomposition to project the documents into a lower-dimensional space results in a representation which reflects the major associative patterns of the data while ignoring less important influences [28]. This is done with the simple truncation of the SVD shown in Eq. 4.2. By the very nature of dimension reduction, it becomes possible for two documents with similar semantic properties to be mapped closer together. Unfortunately, the mixture of signs (positive and negative) throughout the linear combination in Eq. 4.3 makes the decomposition difficult to interpret. The proponents of this method seemed unconcerned with this lack of interpretability; as stated in [28], their aim was “merely to be able to represent terms, documents, and queries in a way that escapes the unreliability, ambiguity and redundancy of individual terms as descriptors.” While the major claims of LSI appear to be legitimate by experimentation and experience, this lack of interpretability is still conceptually problematic for some folks. In order to make this point as clear as possible, consider the last phrase in the quote: “individual terms as descriptors.” By this, the authors are referring to the original “term basis” representation for the data, where each document (from a collection containing $m$ total terms in the dictionary) could be written as:

$$x_j = \sum_{i=1}^{m} f_{ij} e_i$$

where $f_{ij}$ is the frequency of term $i$ in the document, and $e_i$ is the $i^{th}$ column of the $m \times m$ identity matrix. The truncated SVD gives us a new set of coordinates and basis vectors (features):

$$x_j \approx \sum_{i=1}^{r} \alpha_i u_i$$

but the features $u_i$ live in the term space, and thus ought to be interpretable as a linear combination of the original “term basis.” However the linear combination, having both positive and negative coefficients, is semantically meaningless in context. This is our motivation for using Nonnegative Matrix Factorization in Section 4.2.4, but first we will look at the dimension reduction problem from a statistician’s perspective.
4.2.2 Principal Components Analysis (PCA)

The goal of Principal Components Analysis is to reduce a large set of correlated variables to a smaller set of uncorrelated variables in a way that incorporates the maximal amount of variation. The idea was originally proposed by Pearson in 1901 ([101]) and developed independently by Hotelling in 1933 ([58]). For a complete treatment of the statistical uses and implications of PCA, see for example [65, 36]. Here we focus on the properties of the PCA transformation as a means of dimension reduction. We define the mean, $\mu$, and total variance, $\text{Var}_T$, for an $m \times n$ matrix of column data, $X = [x_1 | x_2 | \ldots | x_n]$, using $e$ to represent a vector of ones, as follows:

$$
\mu = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{Xe}{n}
$$

$$
\text{Var}_T[X] = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \mu\|^2_2 = \frac{\|X - \mu e^T\|^2_2}{n}
$$

$$
= \text{trace} \left( (X - \mu e^T)^T (X - \mu e^T) \right) \frac{1}{n}
$$

$$
= \frac{\|X\|^2_2}{n} - \|\mu\|^2_2
$$

PCA involves the analysis of eigenvectors of the covariance matrix $X = X^T X_c$ where $X_c = X - \mu e^T$. It is well known that these eigenvectors can be recovered from the Singular Value Decomposition of $X_c$.

With a little matrix theory, the major properties of PCA can easily be derived. Suppose we wanted to project the data in the columns of $X$ into a lower dimensional orthogonal subspace. Let $Q = [q_1, q_2, \ldots, q_r]$ represent an orthonormal basis for an $r$-dimensional subspace of $\mathbb{R}^n$. Then the orthogonal projection of the data in $X$ onto the span($Q$) is given by:

$$
\hat{X} = QQ^T X.
$$

Let $\mu = \mu(X)$ and

$$
\mu_{\hat{X}} = \mu(\hat{X}) = \frac{1}{n} \hat{X} e^T = QQ^T \mu.
$$
Then the total variance of the projected data is:

\[
\text{Var}_T[\hat{X}] = \frac{1}{n} \| \hat{X} - \mu \hat{e}^T \|^2_F
\]
\[
= \| QQ^T (X - \mu e)^T \|^2_F
\]
\[
= \| QQ^T X_c \|^2_F
\]
\[
= \| Q^T X_c \|^2_F
\]
\[
= \sum_{i=1}^r \| X_c^T q_i \|^2_2
\]

(4.4)

So in order to maximize the variance of the projection over all possible orthonormal bases \( \mathbf{Q} \), it is clear that the columns of \( \mathbf{Q} \) should be the first \( r \) left singular vectors of \( X_c \), denoted \( u_1(X_c), \ldots, u_r(X_c) \), since:

\[
\max_{\| q \|=1} \| X_c^T q \|_2 = \sigma_1(X_c) \implies \arg \max_{\| q \|=1} \| X_c^T q \|_2 = u_1(X_c)
\]

and

\[
\max_{\| q \|=1, q \perp u_1} \| X_c^T q \|_2 = \sigma_2(X_c) \implies \arg \max_{\| q \|=1, q \perp u_1} \| X_c^T q \|_2 = u_2(X_c)
\]

and in general

\[
\max_{\| q \|=1, q \perp \{u_1, \ldots, u_{r-1}\}} \| X_c^T q \|_2 = \sigma_r(X_c) \implies \arg \max_{\| q \|=1, q \perp \{u_1, \ldots, u_{r-1}\}} \| X_c^T q \|_2 = u_r(X_c).
\]

Conveniently, the \( r \)-dimensional coordinates of the centered data \( X_c \) projected into this maximum variance space are also recovered from the SVD as in Eq. 4.3. The “centering” of \( \mathbf{X} \) to \( \mathbf{X}_c \) is merely a translation and therefore does not effect any of the spacial relationships between individual observations. It may be noted that the rank \( r + 1 \) SVD approximation contains at least as much variance as the rank \( r \) PCA approximation. However, in the interpretation of their components the two approximations are quite different. In [17], Cadima and Joliffe sum this up quite nicely:

In an uncentred PCA it is variability about the origin, rather than about the centre of gravity of the \( n \)-point scatter in \( \mathbb{R}^m \), that will be of concern. Unless the origin is an important point of reference for the \( m \) variables, an uncentered PCA may be an artificial procedure that merely highlights the size of the variable means relative to the origin. On the other hand, centring [rows] may destroy meaningful patterns in the [columns] of the data matrix, which would therefore also be lost in the eigenvectors of a covariance matrix. This is a relevant concern...

In both variants of PCA, the underlying variables are pre-processed in different
ways, the criterion for choosing a linear combination of the variables is different (maximizing central [PCA] or non-central second moments [SVD]) and the additional requirements are also different (uncorrelatedness [PCA], or zero crossed non-central second moments [SVD]). Despite all this, the relationships between a standard, [row] centered PCA and its uncentred counterpart are strong...

Thus, the configuration of points projected onto the space spanned by singular vectors of $X$ can be quite different from the configuration produced by projection onto principal components of $X$, or singular vectors of $X_c$. For a more complete discussion of the relationships, both theoretical and practical, between SVD and PCA we highly recommend the work by Cadima and Joliffe in [17].

**Principal Points and Other Arguments for PCA**

One of the things that differentiates PCA from the SVD is its statistical interpretation. Because principal components are eigenvectors of the covariance matrix

$$X_cX_c^T = (X - \mu e)^T(X - \mu e^T)$$

it is possible to consider, in theory, the principal components of an entire population, rather than a sample from that population with which we are generally working in practice. Thus, theoretical derivations involving the entire population of interest make sense using PCA. One of these interesting derivations is the notion of *principal points* [45, 118]. Principal points are those points $\xi_1, \xi_2, \ldots \xi_k$ which minimize the expected sum of square distances of each individual in the population $y_1, y_2, \ldots$ and the nearest $\xi_i$. Sound like $k$-means? Well, it is, for the entire population. Statistical analysis by Tarpey, Li, and Flury showed that the principal points of populations having elliptical distributions (a family of distributions which generalize the multivariate normal distribution) lie in the subspace spanned by the first $k$ principal components [119]. This is one strong argument for dimension reduction via principal components analysis.

Another important justification for PCA dimension reduction was given by Chris Ding and Xiaofeng He. In a 2004 paper, they proved that PCA automatically performs data clustering according to the $k$-means objective function and that principal components provide continuous solutions of the cluster membership indicators in the $k$-means clustering method [31].

### 4.2.3 Alternative SVD Centerings

Other work [133, 132, 16, 18, 81, 83] has suggested alternative centerings to create other PCA-like matrix decompositions. In [132] comparisons are made between the column-centered (PCA as outlined here), row-centered, doubly-centered (row-and-column centered) and overall-mean-centered SVD. In [133], the authors present unique visualizations for these different formulations and argue that either one or all of these PCA/SVD variations can provide useful
information.

4.2.4 Nonnegative Matrix Factorization

The Nonnegative Matrix Factorization (NMF) seeks to decompose an \( m \times n \) matrix of nonnegative column data \( X = [x_1, x_2, \ldots, x_n] \) into the product of an \( m \times r \) matrix of nonnegative “extracted feature vectors”, \( W = [w_1, \ldots, w_r] \), and a \( r \times n \) matrix \( H = \{H_{ij}\} \) of nonnegative coefficients. The decomposition is created by solving the following nonlinear optimization problem:

\[
\min_{W,H} \|A - WH\|_F^2 \quad \text{such that} \quad W \geq 0 \text{ and } H \geq 0
\]

The parameter \( r \) is again the level of dimension reduction desired by the user. The NMF thus provides an additive, parts-based approximation to each data column \( x_j \) in the form of a linear combination of extracted feature vectors as follows:

\[
x_j \approx \sum_{i=1}^r H_{ij}w_i
\]  

(4.5)

For the simplicity of discussion, we will refer to the “extracted feature vectors” as **topic vectors** to avoid any confusion with the original features. For text applications, this terminology makes sense because the topic vectors are nonnegative combinations of terms (the original features) and thus may be interpreted as topics being discussed in the text. For other applications, the same idea translates in other fashions. We can think of the linear combination in Eq. 4.5 as a projection of the data onto an \( r \)-dimensional “topic space”. The coefficients in \( H \) are then (approximately) the coordinates of the data points with respect to the spanning vectors of the topic space. Unlike the SVD, NMF does not intend any sort of orthogonality in the topic vectors \( w_i \), nor does it guarantee that these vectors are linearly independent, although in practice this is almost always the case. We point this out as we continue to refer to this as a rank \( r \) approximation, which it usually is.

Figure 4.3 shows a simple example of the feature extraction done by NMF. At the top of the image, four “documents” (perhaps updates from social media) are shown that contain 2 different topics. Terms highlighted in red are chosen for inclusion in the term-document matrix because they appear in more than one document. The NMF of the term-document matrix is seen to depict these two topics in its feature vectors \( w_1 \) and \( w_2 \).

It seems reasonable from the example in Figure 4.3 to think that NMF may have the ability to detect features that are so dominant and distinct that they actually define clusters in the data. In fact, this has been shown to some extent in [111, 15] where NMF was used to determine \( k \) clusters by means of a \( k \)-dimensional decomposition. We present this method formally in Algorithm 15.

Essentially, this clustering algorithm expects the NMF algorithm to be able to pick out \( k \) topics corresponding exactly to each of the \( k \) clusters. Unfortunately, due to overlapping clusters and noise, this may be expecting too much from the algorithm in some cases. Instead, we suggest allowing NMF to break the data into some number, \( r \), of components with \( r > k \).
My cat likes to eat dog food. It's insane. He won't eat tuna, but dog food? He's all over it.

Check out this video of my dog chasing my cat around the house! He never gets tired!

Injured my ankle playing football yesterday. It is bruised and swollen. Maybe sprained?

So tired of being injured. My ankle just won’t get better!

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tired</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>injured</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ankle</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{cat} & \approx \begin{pmatrix}
0 & 0.7 \\
0 & 1.01 \\
0.26 & 0.33 \\
0.73 & 0.02 \\
0.72 & 0.02 
\end{pmatrix} \\
\text{dog} & \approx \begin{pmatrix}
0 & 0.65 \\
0 & 0.76 \\
0.8 & 0.59 \\
0 & 0.05 
\end{pmatrix}
\end{align*} \]

Figure 4.3: NMF Feature Extraction Illustrated

and then using the \( r \)-dimensional coordinate representation of the data as input to other clustering algorithms in the same way that other dimension reductions are used.

There are many NMF algorithms available, the original was proposed by Lee and Seung in 1999 ([74]) and since then several variations and alternatives have been developed. Many of these variants include sparsity constraints which impose some measure of sparsity on the matrices \( W \) or \( H \). These sparsity constraints assist with the interpretability of the resulting topic vectors or coordinate representations. If the topic vectors are restricted to be sparse, then fewer of the original features or variables from the data contribute to each topic, so we can examine the topic vectors (columns of \( W \)) and get an idea of which of the original features or variables are most important to the extracted topic. If instead the coordinate matrix \( H \) is restricted to be sparse, then for each object or observation in our data, we can view its representation in the topic space as a combination of fewer topic vectors. Most NMF algorithms randomly initialize the matrices \( W \) and \( H \) and update them iteratively until convergence. Because of this, NMF solutions are not generally unique and tend to be sensitive to initializations. The variety of NMF algorithms are beyond the scope of this paper. The interested reader should refer to [73, 74, 59, 75] for an introduction.

4.2.5 Other Methods

Much research has been done to develop more efficient methods of dimension reduction. Many of these methods trade interpretability of the new low-dimensional features for a quick
Algorithm 15 Standard NMF Clustering [111]

**Input:** An $m \times n$ matrix of nonnegative column data, $X$. The number of desired clusters, $k$.

1. Compute the nonnegative factorization $X = W_k H_k$, where $W_k$ is $m \times k$ and $H_k$ is $k \times n$.
2. For each object $i = 1, \ldots , n$, put $x_i$ in the cluster associated with the $j^{th}$ topic vector if:
   $$H_{ji} = \max_t H_{ti}$$

**Output:** Resulting clusters

and easy solution. Researchers have shown that using a series of random projections can often provide the same or in some cases better results than classical methods [26, 47, 41]. Our consensus approach, outlined in Chapter 7 is well suited for these types of random projection ensembles. Other researchers have made considerable progress using randomized matrix decompositions which seek to extract important features by sampling data points from sources containing a substantial number of observations [80, 79, 33, 34]. Randomized algorithms show great promise for managing massive datasets.

Another emerging framework for dimension reduction involves *representative methodology*, often using results from clustering algorithms to select representative points from each cluster to collectively capture representative information for the entire collection [30, 99]. These methods have proven extremely effective for dimension reduction in an information retrieval setting, often producing better results than the classical methods outlined previously. We suggest the work in [30, 99, 100, 130] for more on these methods.
Cluster Validation:
Measuring the Quality of a Clustering

The algorithms in Chapters 2 and 3, with the exception of DBSCAN, all suffer from the same drawback: they require the user to input the number of clusters for the algorithm to create. This is problematic because in practice it is unlikely that the user knows exactly how many clusters they are looking for. In fact, this may be precisely the information that the researcher is after. In the next chapter, we will discuss some popular approaches to determining a suitable value for \( k \). Many of these approaches seek to choose the “best” clustering from a set of clusterings containing different numbers of clusters. In order to present these approaches, we must first look at how one might determine a “best” clustering.

In Chapter 1 it was made clear that the concept of a cluster (and hence the “optimal clustering” of a group of data) is a subjective one. Thus, it is impossible to truly quantify the quality or accuracy of a clustering, without first being given a set of categorical labels assumed to be the optimal clustering. Cluster validation metrics which use such labels (i.e. the “answers”) are called \textit{external} metrics because they use additional information that was not contained in the data input to the clustering algorithm.

Clearly such labels are non-existent in practice, or we would not need to do clustering at all. Thus, it is necessary to establish some \textit{internal} metrics, which use only the information contained in the data, in order to get a sense for the quality or validity of a given clustering. In addition, \textit{relative} metrics are established with the aim of comparing two different clusterings. The goal of this chapter is to provide but a brief introduction to internal, external and relative metrics to fit our needs. For a more comprehensive survey of cluster validation see for example [67, 4, 49, 61, 117, 38].
5.1 Internal Validity Metrics

Most internal metrics aim to describe the cohesion of each cluster and the separation between clusters. The cohesion of each cluster is some measure of its compactness, i.e. how proximal or similar the objects in that cluster are to each other. The separation between clusters is some measure of the distance between them or how dissimilar the objects in different clusters are. Some metrics aim to quantify cohesion and separation separately, while others take both ideas into account in one measure.

5.1.1 General Cluster Cohesion and Separation: Graphs vs. Data

Cohesion

Generally, cluster cohesion measures the similarity or proximity of the points within a cluster. The definitions of cohesion for graph partitioning and data partitioning problems differ slightly depending on the similarity measure used. In graph partitioning, the goal is to measure how similar, or close, vertices in a cluster are to one another, whereas in data clustering cohesion is generally measured by the similarity of the points in a cluster to some representative point (usually the mean or centroid) of that cluster [117]. This difference is illustrated in Figure 5.1. The red lines represent the similarity/distance quantities of interest in either scenario, and the red point in Figure 5.1b is a representative point which is not necessarily a data point. In our analysis, representative points will be defined as centroids and thus may be referred to as such.

![Figure 5.1: The Difference between Cluster Cohesion Measures in Graph Partitioning vs. Data Clustering](image)

Depending on the proximity or similarity function used, the two quantities in Figure 5.1 may or may not be the same. Often times for graphs and networks, there is no simple way
to define a centroid or representative point for a cluster. The particular representation for a cohesion metric will always be dependent on a choice of distance or similarity function. Thus, for graphs we merely define the general concept of cohesion as follows:

**Definition 6** (General Cluster Cohesion in Graphs). For a graph \( G(V, E) \) with edge weights \( w_{ij} \), and a partition of the vertices into \( k \) disjoint sets \( C = \{ C_1, C_2, \ldots, C_k \} \), the cohesion of cluster \( C_p \) is

\[
\text{cohesion}(C_p) = \sum_{i,j \in C_p} w_{ij}.
\]

Given this definition, it should be clear that if \( w_{ij} \) is a measure of similarity between vertices \( i \) and \( j \) then higher values of cohesion are desired, whereas if \( w_{ij} \) measures distance or dissimilarity then lower values of cohesion are desired.

For data clustering problems, cluster cohesion is similarly defined, only now the similarities or proximities are measured between each point in the cluster and the cluster’s representative point.

**Definition 7** (General Cluster Cohesion for Data). Let \( X = [x_1, x_2, \ldots, x_n] \) be an \( m \times n \) matrix of column data, and let \( C = \{ C_1, C_2, \ldots, C_k \} \) be a set of disjoint clusters partitioning the data with corresponding representative points \( \{ c_1, c_2, \ldots, c_k \} \). Then the cohesion of cluster \( C_p \) is

\[
\text{cohesion}(C_p) = \sum_{x_i \in C_p} d(x_i, c_p)
\]

Where \( d \) is any distance or similarity function.

Again, the given definitions are not associated with any particular distance or similarity function and thus define a broad classes of metrics for measuring cluster cohesion.

**Separation**

The goal in clustering is not only to form groups of points which are similar or proximal, but also to assure some level of separation or dissimilarity between these groups. Thus, in addition to measuring cluster cohesion, it is also wise to consider cluster separation. Again this concept is a little different for graphs, where the separation is measured pairwise between points in different clusters, than it is for data, where separation is generally measured between the representative points of different clusters. This difference is presented with the following 2 definitions.

**Definition 8** (General Cluster Separation for Graphs). For a graph \( G(V, E) \) with edge weights \( w_{ij} \), and a partition of the vertices into \( k \) disjoint sets \( C = \{ C_1, C_2, \ldots, C_k \} \). The separation between clusters \( C_p \) and \( C_q \) is

\[
\text{separation}(C_p, C_q) = \sum_{\substack{i \in C_p \\ j \in C_q}} w_{i,j}.
\]
Definition 9 (General Cluster Separation for Data). Let \( X = [x_1, x_2, \ldots, x_n] \) be an \( m \times n \) matrix of column data, and let \( C = \{C_1, C_2, \ldots, C_k\} \) be a set of disjoint clusters in the data with corresponding representative points \( \{c_1, c_2, \ldots, c_k\} \). Then the separation between clusters \( C_p \) and \( C_q \) is

\[
\text{separation}(C_p, C_q) = d(c_p, c_q)
\]

where \( d \) is any distance or similarity function.

Averaging Measures of Cohesion and Separation for a Set of Clusters

Definitions 6, 7, 8 and 9 provide simple, well-defined metrics (given a proximity or similarity measure) for individual clusters \( C_p \) or pairs of clusters \( (C_p, C_q) \) that can be combined into overall measures for a clustering \( C = \{C_1, C_2, \ldots, C_k\} \) by some weighted average [117]. The weights for such an average vary according to applications and user-preference, but they typically reflect the size of the clusters in some way. At the end of this chapter, in Table 5.2, we provide a few examples of these overall metrics.

5.1.2 Common Measures of Cohesion and Separation

As stated earlier, the previous definitions were considered “general” in that they did not specify particular functions of similarity or distance. Here we discuss some specific measures which have become established as foundations of cluster validation in the literature.

Sum of Squared Error (SSE)

The sum of squared error (SSE) metric incorporates the squared euclidean distances from each point in a given cluster to the centroid of the cluster, defined as

\[
\mu_j = \frac{1}{n_j} \sum_{x_i \in C_j} x_i.
\]

This is equivalent to measuring the average pairwise distance between points in a cluster, as one would do in a graph having Euclidean distance as a measure of proximity. The SSE of a single cluster is then

\[
\text{SSE}(C_j) = \sum_{x_i \in C_j} \|x_i - \mu_j\|^2
\]

(5.1)

\[
= \frac{1}{2n_j} \sum_{x_i, x_l \in C_j} \|x_i - x_l\|^2
\]

(5.2)
where \( n_j = |C_j| \). The SSE of an entire clustering \( C \) is simply the sum of the SSE for each cluster \( C_j \in C \)

\[
\text{SSE}(C) = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2.
\]

Smaller values of SSE indicate more cohesive or compact clusters. One may recognize Eq. 5.2 as the objective function from Section 2.2.2 because minimizing the SSE is the goal of the Euclidean \( k \)-means algorithm. We can use the same idea to measure cluster separation by computing the Between Group Sums of Squares (SSB), which is a weighted average of the squared distances from the cluster centroids \( \{\mu_1, \mu_2, \ldots, \mu_k\} \) to the over all centroid of the dataset \( \mu_* = \frac{1}{n} \sum_{i=1}^{n} x_i \):

\[
\text{SSB}(C) = \sum_{j=1}^{k} n_j \|\mu_j - \mu_*\|^2.
\]

It is straightforward to show that the total sum of squares (TSS) of the data

\[
\text{TSS}(X) = \sum_{i=1}^{n} \|x_i - \mu_*\|^2,
\]

which is constant, is equal to the sum of the SSE and SSB for every clustering \( C \), i.e.

\[
\text{TSS}(X) = \text{SSE}(C) + \text{SSB}(C),
\]

thus minimizing the SSE (attaining more cohesion) is equivalent to maximizing the SSB (attaining more separation).

Sum of Squared Error is used as a tool in the calculation of the gap statistic, outlined in the next chapter, a popular parameter used to determine the number of clusters in data.

Ray and Turi’s Validity Measure

In [107] a measure of cluster validity is chosen as the ratio of intracluster distance to inter-cluster distance. The authors define these distances as

\[
M_{\text{intra}} = \frac{1}{n} \text{SSE}(C) = \frac{1}{n} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2.
\]

and

\[
M_{\text{inter}} = \min_{1 \leq i \leq j \leq k} \|\mu_i - \mu_j\|^2.
\]

Clearly a good clustering should have small \( M_{\text{intra}} \) and large \( M_{\text{inter}} \). Ray and Turi’s validity measure,

\[
V = \frac{M_{\text{intra}}}{M_{\text{inter}}}
\]
is expected to take on smaller values for a better clustering [49].

**Silhouette Coefficients**

Silhouette coefficients are popular indices that combine the concepts of cohesion and separation [98]. These indices are defined for each object or observation \( x_i, i = 1, \ldots, n \) in the data set using two parameters \( a_i \) and \( b_i \), measuring cohesion and separation respectively. These parameters and the silhouette coefficient for an object \( x_i \) are computed as follows:

Suppose, for a given clustering \( C = \{ C_1, \ldots, C_k \} \) with \( |C_j| = n_j \), that the point \( x_i \) belongs to cluster \( C_p \).

Then \( a_i \) is the average distance (or similarity) of point \( x_i \) from the other points in \( C_p \),

\[
a_i = \frac{1}{n_p} \sum_{x_j \in C_p} d(x_i, x_j)
\]

Define the distance (or similarity) between \( x_i \) and the remaining clusters \( C_q, 1 \leq q \leq k, q \neq p \) to be the average distance (or similarity) between \( x_i \) and the points in each cluster,

\[
d(x_i, C_q) = \frac{1}{n_q} \sum_{x_j \in C_q} d(x_i, x_j).
\]

Then \( b_i \) is defined to be the minimum of these distances (or maximum for similarity):

\[
b_i = \min_{q \neq p} d(x_i, C_q).
\]

The silhouette coefficient for \( x_i \) is then

\[
s_i = \frac{(b_i - a_i)}{\max(a_i, b_i)} \text{ (for distance metrics)}
\]

\[
s_i = \frac{(a_i - b_i)}{\max(a_i, b_i)} \text{ (for similarity metrics)}
\]

The silhouette coefficient takes on values \(-1 \leq s_i \leq 1\), where negative values undesirably indicate that \( x_i \) is closer (or more similar) on the average to points in another cluster than to points in its own cluster, and values close to 1 indicate a good clustering.

Silhouette coefficients are commonly averaged for all points in a cluster to get an overall sense for the validity of that cluster.
5.2 External Validity Metrics

Many of the results presented in Chapter 8 will use data sets for which the class labels of each object are known. Using this information, one can generally create validity metrics that are easier to understand and compare across clusterings. Such metrics are known as external metrics because of their dependence on the external class labels. We will show that most external metrics can be transformed into relative metrics which compute the similarity between two clusterings.

Using the information from external class labels, one can create a so-called confusion matrix (also called a matching matrix). The confusion matrix is simply a table that shows correspondence between predicted cluster labels (determined by an algorithm) and the actual or “true” cluster labels of the data. A simple example is given in Figure 5.2, where the actual class labels (‘science’, ‘math’, and ‘french’) are shown across the columns of the matrix and the clusters determined by an algorithm ($C_1$, $C_2$, and $C_3$) are shown along the rows. The $(i,j)\text{th}$ entry in the confusion matrix is then the number of objects from the dataset that had class label $j$ and were assigned to cluster $i$.

![Confusion Matrix Example](image)

Figure 5.2: An Example of a Confusion Matrix

For this simple example, one may assume that cluster 1 ($C_1$) corresponds to the class ‘Science’, cluster 2 corresponds to the class ‘Math’, and likewise that cluster 3 represents the class ‘French’, even though the clustering algorithm did not split these classes apart perfectly. Most external metrics will rely on the values in the confusion matrix.

5.2.1 Accuracy

Accuracy is a measure between 0 and 1 that simply measures the proportion of objects that were labelled correctly by an algorithm. This is not always a straightforward task, given that the labels assigned by a clustering algorithm are done so arbitrarily in that it does not matter if one refers to the same group of points as ‘cluster 1’ or ‘cluster 2’. In the confusion matrix in Figure 5.2, it is easy to identify which cluster labels corresponds to which class. In this case
it is easy to see that out of a total of 153 objects, only 13 were classified incorrectly, leading to an accuracy of $140/153 \approx 91.5\%$. However with a more confusing confusion matrix, like that shown in Figure 5.3, the answer is not quite as clear and thus it is left to determine exactly how to match predicted cluster labels with assigned class labels in an appropriate way.

![Figure 5.3: A More Confusing Confusion Matrix](image)

This turns out to be a well studied matching problem from graph theory, known as a maximum matching for a bipartite graph. If we transform our confusion matrix from Figure 5.3 into an undirected bipartite graph with edge weights corresponding to edges in the confusion matrix, the result would be the graph in Figure 5.4a. The task is then to find a set of 3 edges, each beginning at distinct vertices on the left and ending at distinct vertices on the right such that the sum of the edge weights is maximal. The solution to this problem is shown in Figure 5.4b and it is clear that the matching of predicted labels to actual labels did not actually change from the simpler version of this confusion matrix in Figure 5.2, it just became less obvious because of the errors made by the algorithm.

![Figure 5.4: Matching Predicted Class Labels to Actual Class Labels](image)
Once the predicted class labels are matched to the actual labels, the accuracy of a clustering is straightforward to compute by

\[
\text{Accuracy}(C) = \frac{\text{# of objects labelled correctly}}{n}.
\]

The accuracy of the second clustering given in Figure 5.3 is 118/153 \(\approx 77\%\), which is sharply lower than the 91.5\% achieved by the clustering in Figure 5.2. The nice thing about accuracy as a metric is it provides a contextual interpretation and thus allows us to quantitate an answer to the question “how much better is this clustering?” This is not necessarily true of other external metrics, as you will see in the next sections.

The aspect of this metric that requires some computation is the determination of the maximum matching as shown in Figure 5.4b. Fortunately, this problem is one that was solved by graph theorist H.W. Kuhn in 1955 [70]. Kuhn’s algorithm was adapted by James Munkres in 1957 and the resulting method was dubbed the Kuhn-Munkres Algorithm, or sometimes the Hungarian Algorithm in honor of the mathematicians who pioneered the work upon which Kuhn’s method was based [92]. This algorithm is fast and computationally inexpensive. The details of the process are not pertinent to the present discussion, but can be found in any handbook of graph theory algorithms.

**Comparing Two Clusterings: Agreement**

The accuracy metric, along with other external metrics, can be used to compute the similarity between two different cluster solutions. Since, in practice, class labels are not available for the data, the user may run two different clustering algorithms (or even the same algorithm with different representations of the data as input or different initializations) and get two different clusterings as a result. The natural question is then “how similar are these two clusterings?” Treating one clustering as class labels and computing the accuracy of the second compared to the first will provide the percentage of data points for which the two clusterings agree on cluster assignment. Thus, when comparing two clusterings, the accuracy metric becomes a measure of agreement between the two clusterings. As such, a value of 90\% agreement indicates that 90\% of the data points were clustered the same way in both clusterings.

**5.2.2 Entropy**

The notion of entropy is associated with randomness. As a clustering metric, entropy measures the degree to which the predicted clusters consist of objects belonging to a single class, as opposed to many classes. Suppose a cluster (as predicted by an algorithm) contains objects belonging to multiple classes (as given by the class labels). Define the quantities

- \(n_i\) = number of objects in cluster \(C_i\)
- \(n_{ij}\) = number of objects in cluster \(C_i\) having class label \(j\)
- \(p_{ij} = n_{ij}/n_i\) = probability that a member of cluster \(C_i\) belongs to class \(j\)
Then the **entropy** of each cluster $C_i$ is

$$\text{entropy}(C_i) = - \sum_{j=1}^{L} p_{ij} \log_2 p_{ij}$$

where $L$ is the number of classes, and the total entropy for a set of clusters, $C$, is the sum of the entropies for each cluster weighted by the proportion of points in that cluster:

$$\text{entropy}(C) = \sum_{i=1}^{k} \frac{n_i}{n} \text{entropy}(C_i).$$

Smaller values of entropy indicate a less random distribution of class labels within clusters [98]. One benefit of using entropy rather than accuracy is that it can be calculated for any number of clusters $k$, whereas accuracy is restricted to the case where $k = L$.

**Sample Calculations for Entropy**

Comparing the two clusterings represented by the confusion matrices in Figures 5.2 and 5.3, we’d see that for the first example,

$$p_{11} = \frac{45}{50} \quad p_{12} = \frac{5}{50} \quad p_{13} = 0$$
$$p_{21} = \frac{8}{48} \quad p_{22} = \frac{40}{48} \quad p_{23} = 0$$
$$p_{31} = 0 \quad p_{32} = 0 \quad p_{33} = 1$$

so that

$$\text{entropy}(C_1) = -( \frac{45}{50} \log_2 \frac{45}{50} + \frac{5}{50} \log_2 \frac{5}{50} ) = 0.469$$
$$\text{entropy}(C_2) = -( \frac{8}{48} \log_2 \frac{8}{48} + \frac{40}{48} \log_2 \frac{40}{48} ) = 0.65$$
$$\text{entropy}(C_3) = -(\log_2 1) = 0$$

and thus the total entropy of the first clustering is

$$\text{entropy}(C) = \frac{50}{153}(0.469) + \frac{48}{153}(0.65) = \boxed{0.357}.$$
And for the second example, we have

\[
\begin{align*}
p_{11} &= \frac{25}{30} & p_{12} &= \frac{5}{30} & p_{13} &= 0 \\
p_{21} &= \frac{30}{68} & p_{22} &= \frac{38}{68} & p_{23} &= 0 \\
p_{31} &= 0 & p_{32} &= 0 & p_{33} &= 1
\end{align*}
\]

yielding

\[
\begin{align*}
\text{entropy}(C_1) &= -\left(\frac{25}{30} \log_2 \frac{25}{30} + \frac{5}{30} \log_2 \frac{5}{30}\right) = 0.65 \\
\text{entropy}(C_2) &= -\left(\frac{30}{68} \log_2 \frac{30}{68} + \frac{38}{68} \log_2 \frac{38}{68}\right) = 0.99 \\
\text{entropy}(C_3) &= -(\log_2 1) = 0
\end{align*}
\]

and finally the total entropy of the second clustering is

\[
\text{entropy}(C) = \frac{30}{153}(0.469) + \frac{68}{153}(0.65) = 0.568
\]

revealing a higher-overall entropy and thus a worse partition of the data compared to the first clustering.

### 5.2.3 Purity

Purity is a simple measure of the extent to which a predicted cluster contains objects of a single class [98]. Using the quantities defined in the previous section, the \textbf{purity} of a cluster is defined as

\[
purity(C_i) = \max_j p_{ij}
\]

and the purity of a clustering \(C\) is the weighted average

\[
purity(C) = \sum_{i=1}^{k} \frac{n_i}{n} \text{purity}(C_i).
\]

The purity metric takes on positive values less than 1, where values of 1 reflect the desirable situation where each cluster only contains objects from a single class. Like entropy, purity can be computed for any number of clusters, \(k\). Purity and accuracy are often confused and used interchangeably but they are not the same. Purity takes no matching of class labels to cluster labels into account, and thus it is possible for the purity of two clusters to count the proportion of objects having the \textit{same} class label. For example, suppose we had only two class labels given, A and B, for a set of 10 objects and set our clustering algorithm to seek 2 clusters in the data and the following confusion matrix resulted:
Then the purity of each cluster would be $\frac{3}{5}$ referring in both cases to the proportion of objects having class label A. High values of purity are easy to achieve when the number of clusters is large. For example, by assigning each object to its own cluster we’d achieve perfect purity. One metric that accounts for such a tradeoff is *Normalized Mutual Information*, presented next.

### Sample Purity Calculations
Again, we’ll compare the two clusterings represented by the confusion matrices in Figures 5.2 and 5.3. For the first clustering,

\[
\begin{align*}
purity(C_1) &= \max(\frac{45}{50}, \frac{5}{50}, 0) = \frac{45}{50} = 0.9 \\
purity(C_2) &= \max(\frac{8}{48}, \frac{40}{48}, 0) = \frac{40}{48} = 0.83 \\
purity(C_2) &= \max(0, 0, 1) = 1
\end{align*}
\]

so the overall purity is

\[
purity(C) = \frac{50}{153}(0.9) + \frac{48}{153}(0.83) + \frac{55}{153}(1) = 0.914
\]

Similarly for the second clustering we have,

\[
\begin{align*}
purity(C_1) &= \max(\frac{25}{30}, \frac{5}{30}, 0) = \frac{25}{30} = 0.83 \\
purity(C_2) &= \max(\frac{30}{68}, \frac{38}{68}, 0) = \frac{38}{68} = 0.56 \\
purity(C_2) &= \max(0, 0, 1) = 1
\end{align*}
\]

And thus the overall purity is

\[
purity(C) = \frac{30}{153}(0.83) + \frac{68}{153}(0.56) + \frac{55}{153}(1) = 0.771
\]

### 5.2.4 Mutual Information (MI) and Normalized Mutual Information (NMI)

Mutual Information (MI) is a measure that has been used in various data applications [98]. The objective of this metric is to measure the amount information about the class labels revealed by a clustering. Adopting the previous notation,

\[n_i = \text{number of objects in cluster } C_i\]
\[ n_{ij} = \text{number of objects in cluster } C_i \text{ having class label } j \]

\[ p_{ij} = \frac{n_{ij}}{n_i} = \text{probability that a member of cluster } C_i \text{ belongs to class } j \]

also let

\[ l_j = \text{the number of objects having class label } j \]

\[ L = \text{the number of classes} \]

\[ \mathcal{L} = \{ \mathcal{L}_1, \ldots, \mathcal{L}_L \} \text{ the “proper” clustering according to class labels} \]

and, as always, let

\[ n = \text{the number of objects in the data} \]

\[ k = \text{the number of clusters in the clustering.} \]

The **Mutual Information** of a clustering \( C \) is then

\[
\text{MI}(C) = \sum_{i=1}^{k} \sum_{j=1}^{L} p_{ij} \log \frac{n_{ij}}{n_i l_j}
\]

and the **Normalized Mutual Information** of \( C \) is

\[
\text{NMI}(C) = \frac{\text{MI}(C)}{[\text{entropy}(C) + \text{entropy}(\mathcal{L})]/2}
\]

Clearly, when \( \mathcal{L} \) corresponds the class labels we have \( \text{entropy}(\mathcal{L}) = 0 \) but if user’s objective is instead to compare two different clusterings, this piece of the equation is necessary. Thus, using the same treatment used for agreement between two clusterings, one can compute the mutual information between two clusterings.

### 5.2.5 Other External Measures of Validity

There are a number of other measures that can either be used to validate a clustering in the presence of class labels or to compare the similarity between two clusterings \( C = \{ C_1, C_2, \ldots, C_k \} \) and \( \hat{C} = \{ \hat{C}_1, \hat{C}_2, \ldots, \hat{C}_k \} \). In our presentation we will consider the second clustering to correspond to the class labels, but in the same way that the accuracy metric can be used to compute agreement, these measures are often used to compare different clusterings. To begin we define the following parameters [49]:

- \( a \) is the number of pairs of data points which are in the same cluster in \( \hat{C} \) and have the same class labels (i.e. are in the same cluster in \( \hat{C} \)).

- \( b \) is the number of pairs of data points which are in the same cluster in \( \hat{C} \) and have different class labels.
$c$ is the number of pairs of data points which are in different clusters in $C$ and have the same class labels.

$d$ is the number of pairs of data points which are in different clusters in $C$ and have different class labels.

These four parameters add up to the total number of pairs of points in the data set, $N$,

$$a + b + c + d = N = \frac{n(n - 1)}{2}.$$ 

From these values we can compute a number of different similarity coefficients, a few of which are provided in Table 5.1 [49].

### Table 5.1: Some Common External and Relative Indices

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaccard Coefficient</td>
<td>$J = \frac{a}{a + b + c}$</td>
</tr>
<tr>
<td>Rand Statistic</td>
<td>$R = \frac{a + b}{N}$</td>
</tr>
<tr>
<td>Folkes and Mallows Index</td>
<td>$\sqrt{\frac{a}{a + b} \frac{a}{a + c}}$</td>
</tr>
<tr>
<td>Odds Ratio</td>
<td>$\frac{ad}{bc}$</td>
</tr>
</tbody>
</table>

**Hubert’s $\Gamma$ Statistic**

Another measure popular in the clustering literature is Hubert’s $\Gamma$ statistic, which aims to measure the correlation between two clusterings, or between one clustering and the class label solution [98, 49]. Here we define an $n \times n$ adjacency matrix for a clustering $C$, denoted $Y$ such that

$$Y_{ij} = \begin{cases} 
1 & \text{object } i \text{ and object } j \text{ are in the same cluster in } C \\
0 & \text{otherwise} 
\end{cases} \quad (5.3)$$

Similarly, let $H$ be an adjacency matrix pertaining to the class label partition (or a different clustering) as follows:

$$H_{ij} = \begin{cases} 
1 & \text{object } i \text{ and object } j \text{ have the same class label} \\
0 & \text{otherwise} 
\end{cases} \quad (5.4)$$
Then Hubert’s $\Gamma$ statistic, defined as

$$\Gamma = \frac{1}{N} \sum_{i=1}^{n-1} \sum_{i+1}^{n} Y_{ij} H_{ij},$$

is a way of measuring the correlation between the clustering and the class label partition [49].
Table 5.2: Some Common Measures of Overall Cohesion and Separation [49, 98]

<table>
<thead>
<tr>
<th>Name</th>
<th>Overall Measure</th>
<th>Cluster Weight</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Cohesion (Graphs)</td>
<td>$\sum_{p=1}^{k} \alpha_p \sum_{i,j \in C_p} w_{ij}$</td>
<td>$\alpha_p = \frac{1}{n_i}$</td>
<td>Graph Cohesion</td>
</tr>
<tr>
<td>$G_{CS}$ (Graph C-S Measure)</td>
<td>$\sum_{p=1}^{k} \alpha_p \sum_{q=1, i \in C_p}^{k} \sum_{q \neq P, j \in C_q} w_{ij}$</td>
<td>$\alpha_p = \frac{1}{\sum_{i,j \in C_p} w_{ij}}$</td>
<td>Graph Cohesion &amp; Separation</td>
</tr>
<tr>
<td>Sum Squared Error (SSE) (Data)</td>
<td>$\sum_{p=1}^{k} \alpha_p \sum_{x_i \in C_p} |x_i - \mu_p|^2$</td>
<td>$\alpha_p = 1$</td>
<td>Data Cohesion</td>
</tr>
<tr>
<td>Ray and Turi’s $M_{intra}$</td>
<td>$\sum_{p=1}^{k} \alpha_p \sum_{x_i \in C_p} |x_i - \mu_p|^2$</td>
<td>$\alpha_p = \frac{1}{n}$</td>
<td>Data Cohesion</td>
</tr>
<tr>
<td>Ray and Turi’s $M_{inter}$</td>
<td>$\min_{1 \leq i \leq k} |\mu_i - \mu_j|^2$</td>
<td></td>
<td>Data Separation</td>
</tr>
<tr>
<td>Ray and Turi’s Validity Measure</td>
<td>$\frac{M_{intra}}{M_{inter}}$</td>
<td>N/A</td>
<td>Data Cohesion &amp; Separation</td>
</tr>
</tbody>
</table>
Determining the Number of Clusters $k$

As previously discussed, one of the major dilemmas faced when using the clustering algorithms from Chapters 2 and 3 is that these algorithms take the number of clusters, $k$, as input. Therefore, it is necessary to somehow determine a reasonable estimate for the number of clusters present. Many methods have been proposed for this task, for a more in-depth summary we suggest the 1999 book by Gordon [53] or the 1985 paper by Milligan and Cooper [89]. The purpose of this chapter is to survey some established methodologies for this task, and to motivate our novel method discussed in Chapter 7.

6.1 Methods based on Cluster Validity (Stopping Rules)

The most popular methods for determining the number of clusters involve observing some internal measure of cluster validity (like those outlined in the previous chapter) as the number, $k$, of clusters increases. Cohesion scores like SSE are expected to be monotonically decreasing as $k$ increases. At some value, $k^*$, the marginal drop in SSE is expected to flatten drastically, indicating that further division of the clusters does not provide a significant improvement in terms of cohesion [89, 122, 61]. Methods based on cluster validity can be implemented with any clustering algorithm the user desires. Unfortunately, if the algorithm used is not working well with the dataset then the resulting determination of the number of clusters will be flawed. Furthermore, it is possible to get different results with different algorithms or different cohesion metrics, which may instil the user with little confidence in a given solution.

In the hierarchical algorithms from Section 2.1, a series of solutions ranging from $k = 1$ to $k = n$ clusters are output, and thus the methods for determining an appropriate value for $k$ in these procedures are often referred to as stopping rules. Since hierarchical algorithms tend to be slow and computationally expensive for large datasets, the stopping rules which cannot be extended to include general partitions of the data will be omitted from the discussion.
6.1.1 Sum Squared Error (SSE) Cohesion Plots

For a simple example for this stopping rule methodology, consider the so-called Ruspini dataset in Figure 6.1, which has been used to demonstrate clustering algorithms in the literature. This dataset consists of 75 two dimensional points in the first Cartesian quadrant, and visually it seems clear that these points fall into $k = 4$ different clusters, using Euclidean distance as a measure of proximity. (Some individuals may argue that these 4 clusters could be meaningfully broken down into smaller clusters. These arguments are certainly valid, but we base our decision to specify 4 on the following assumptions: a) If asked to choose 4 clusters, most human beings would choose the same 4 - this may not be the case with 5 or 6; b) If we consider these points as a sample from a population, then it is reasonable to suspect that the collection of more data may destroy the subcluster appearance - that is, there is more observed evidence of 4 clusters than any other number.) We ought to be able to uncover this “true” number of clusters by observing the level of decrease in the SSE metric as the number of clusters increase, and determining an “elbow” in the curve at $k^* = 4$ where the SSE flattens out for $k \geq 4$.

![Figure 6.1: The Two-Dimensional Ruspini Dataset](image)

Figure 6.2 shows some examples of clusters found in the data using $k$-means and $k = 2, 3, 4, 5, 6$ clusters. The initialization of seed points was done randomly in each case. Fig-
Figure 6.2f shows the SSE (as described in Section 5.2) for the 6 different clusterings. We wish to point out that these 5 clusterings are “good” or reasonable clusterings upon visual inspection. Indeed, this first SSE plot properly depicts $k^* = 4$ as the “elbow” of the curve, where the marginal decrease in SSE for adding additional clusters flattens out.

**User Beware**

As always, with k-means, it is of the utmost importance that the user pay close attention to the output from the algorithm. In our creation of the plot in Figure 6.2f, we also came by the two solutions depicted in Figure 6.3a and Figure 6.3b, associated with $k = 4$ and $k = 5$ clusters respectively. Because we are able to visualize the data in 2 dimensions (which, practically speaking, means we could have identified $k^* = 4$ by visualizing the original data anyhow), we were able to throw away these two solutions upon inspection. If we did not do this, the resulting SSE plot shown in Figure 6.3c would have clearly misled us to choose $k^* = 3$ clusters. Without being able to visually inspect the solutions, it is wise to run several iterations of the k-means algorithm for each $k$ and use some criteria (like lowest SSE, or most frequent SSE [39]) to choose an appropriate clustering for inclusion in the SSE plot. While this is not guaranteed to circumvent problematic SSE plots like that shown in Figure 6.3c, it can help in many situations and certainly won’t hurt in others. This dependence on good clusterings is a glaring drawback of stopping rule methodology, because not all algorithms can produce multiple results for a single value of $k$ to choose from.

6.1.2 Cosine-Cohesion Plots for Text Data

Further complicating the method of cohesion plots is the curse of dimensionality discussed in Chapter 4. For high dimensional data, it is unusual to witness such drastic “elbows” in these plots. To illustrate this effect, we consider a combination of 3 text datasets used frequently in the information retrieval literature: ‘Medlars’, ‘Cranfield’, ‘CISI’[11, 68]. The Medlars-Cranfield-CISI (MCC) collection consists of nearly 4,000 scientific abstracts from 3 different disciplines. These 3 disciplines (Medlars = medicine, Cranfield = aerodynamics, CISI = information science) form 3 relatively distinct clusters in the data, which are not particularly difficult to uncover (For example, spherical k-means frequently achieves 98% accuracy on the full-dimensional data). For this experiment, we ran 25 trials of the spherical k-means algorithm for each value of $k = 2, 3, \ldots, 20$ and from each set of trials chose the solution with the lowest objective value. The resulting SSE plot is shown in Figure 6.4. It is difficult to identify a distinct “elbow” in this curve.

Because of the behavior of distance metrics in high dimensional space, it is often easier (and always faster) to find clusters after reducing the dimensions of a dataset by one of the methods discussed in Chapter 4. Because the singular value decomposition generally works well for text data, we conduct this same experiment on the Medlars-Cranfield-CISI dataset using projections onto the first $r = 8, 12, 20$ singular vectors. Using the correct number of clusters $k^* = 3$, the k-means algorithm is able to achieve the same accuracy of 98% on each
Figure 6.2: 5 “Good” k-means Clusterings of the Ruspini Dataset and Corresponding SSE Plot
of these dimension reductions, indicating that the clustering information is by no means lost in the lower dimensional representations. However, the SSE plots for these lower dimensional representations, shown in Figure 6.5, do no better at clearly indicating an appropriate number of clusters. In fact, these graphs seem to flatten out at $k = r$. Again, 25 trials of the $k$-means algorithm were run for each value of $k$ and the solution with the lowest SSE was chosen to represent that value of $k$ in the plots.

6.1.3 Ray and Turi’s Method

In [107], Ray and Turi suggested the use of their validity metric for determining the number of clusters. Unlike the SSE plots investigated previously, this method does not rely on the subjectivity of the user. Instead, the goal is simply to find the minimum value of their validity metric over the clusterings produced for various values of $k$. Recalling the definition from

![Figure 6.3: Example of “Poor” Clusterings’ Effect on SSE Plot](image)

(c) SSE Plot using the Poor Clusterings Found by $k$-means
Chapter 5 Section 5.1.2, we have the validity of a clustering defined as

\[
v = \frac{M_{\text{intra}}}{M_{\text{inter}}} \tag{6.1}
\]

where

\[
M_{\text{intra}} = \frac{1}{n} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \| \mathbf{x} - \mathbf{\mu}_i \|_2^2 \quad \tag{6.2}
\]

\[
M_{\text{inter}} = \min_{1 \leq i < j \leq k} \| \mathbf{\mu}_i - \mathbf{\mu}_j \|_2^2 \quad \tag{6.3}
\]

and \( \mathbf{\mu}_i \) is the centroid of cluster \( i \). In their original work in [107], the authors’ goal was to cluster images. They noticed for these datasets that the minimum value for the validity metric frequently occurred for small numbers of clusters in the range of 2, 3, or 4 because of the large inter-cluster distances occurring when the number of clusters is small. This was undesirable in their application to image processing because the number of clusters was not expected to be small. To account for this fact, they proposed the following procedural adjustment for determining the number of clusters:

- Specify the maximum number of clusters to be considered, \( k_{\text{max}} \).
- For \( k = 2, \ldots, k_{\text{max}} \) use \( k \)-means to cluster the data into \( k \) clusters.
- For each clustering \( C(k) \) compute the validity metric, \( v(k) \) from Eq. 6.1.
- Locate the first local maximum in the validity measure, \( \hat{k} \) such that

\[
v(\hat{k} - 1) < v(\hat{k}) > v(\hat{k} + 1)
\]
• Choose the optimal number of clusters, $k^*$, to be the modified minimum such that $\tilde{k} < k^* \leq k_{\text{max}}$ is the number which minimizes the validity measure after the first local maximum.

**Ray and Turi Plots for the Ruspini Data**

We applied the above method to the 2-dimensional Ruspini data which was depicted in Figure 6.1. To avoid the type of poor clusterings that were displayed in Figure 6.3, for each value of $k$, the $k$-means algorithm was run 25 times and the best solution (that is, the solution with the lowest objective function) was chosen to represent that value of $k$. Figure 6.6 shows the plot of Ray and Turi’s validity metric computed on each solution. If one were to pick the global minimum from this set of clusterings, the optimal number of clusters would be $k^* = 2$. However, according to the modified minimum favored in the original paper [107], the optimal number of clusters for the Ruspini data is $k^* = 5$. Neither of these solutions impose

![Figure 6.5: SSE Plots for Medlars-Cranfield-CISI Clusterings using SVD Reduction to $r$ dimensions](image)

(c) $r = 20$
quite as obvious a clustering as the true number, 4.

![Ray and Turi Validity Plot for Ruspini Data](image)

**Figure 6.6: Ray and Turi Validity Plot for Ruspini Data**

**Ray and Turi Plots for Medlars-Cranfield-CISI**

We can generate similar plots using the same clusterings found by spherical $k$-means that were used to generate the SSE plots in Figures 6.4 and 6.5. Obviously, the plots of Ray and Turi’s validity metric are far more definitive in their determination of $k^*$, although it is left to the user to determine whether to pick the global minimum or modified minimum [107].

The results from Figures 6.6 and 6.7 are summarized in the following table, which shows the number of clusters that would be chosen if one were to pick the global minimum validity or the modified minimum validity along with the actual number of clusters.

<table>
<thead>
<tr>
<th>Data Input</th>
<th>Global Min</th>
<th>Modified Min.</th>
<th>Actual $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Medlars-Cranfield-CISI</td>
<td>4</td>
<td>6</td>
<td>3 or 5</td>
</tr>
<tr>
<td>Ruspini</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
6.1.4 The Gap Statistic

The gap statistic is an index devised by Tibshirani, Walther, and Hastie in 2000 that has received massive amounts of attention in the literature. This method is quite similar to the stopping methods previously discussed, only now the objective is to compare the cluster cohesion values with what is expected under some null reference distribution [122]. Supposing the \( n \) data points \( x_1, \ldots, x_n \) are clustered in to \( k \) clusters, \( C_1, C_2, \ldots, C_k \) and \( |C_j| = n_j \) some measure of cohesion is defined as

\[
W_k = \sum_{j=1}^{k} \frac{1}{2n_j} \sum_{x_p, x_q \in C_j} d(x_p, x_q)
\]
where \(d(x, y)\) is a distance function. The idea is then to compare the graph of \(\log(W_k), k = 1, \ldots, K\) to its expectation under some null reference distribution and to choose the value of \(k, 1 \leq k \leq K\) for which \(\log(W_k)\) falls the farthest below its expectation. This distance between \(\log(W_k)\) and its expectation under the reference distribution, denoted by \(E^*\), is called the \textit{gap}:

\[
\text{Gap}(k) = E^*(\log(W_k)) - \log(W_k).
\]

This expectation is estimated by drawing a Monte Carlo sample, \(X^*_1, X^*_2, \ldots, X^*_B\) from the reference distribution. Each dataset in the sample is clustered, and the values of \(\log(W^*_k), k = 1, \ldots, K\) are averaged over the samples. The sampling distribution of the gap statistic is controlled using the standard deviation, \(sd_k\), of the \(B\) Monte Carlo replicates of \(\log(W^*_k)\). Accounting for the simulation error in \(E^*(\log(W_k))\) yields the standard error

\[
s_k = sd_k \sqrt{1 + \frac{1}{B}}.
\]

Using the common “one standard error” rule, the number of clusters \(k^*\) is chosen to be the smallest \(k\) such that

\[
\text{Gap}(k) \geq \text{Gap}(k + 1) - s_{k+1}.
\]

The authors in [122] suggest, both for simplicity and performance, using a uniform distribution as the null reference distribution. This process is summarized in Algorithm 16.

In Figure 6.8 we provide the results from the gap statistic procedure on the ruspini data. Our Monte Carlo simulation involved \(B = 10\) generated datasets. The gap statistic indicates the presence of \(k^* = 4\) clusters.
Algorithm 16 Computation of the Gap Statistic [122]

1. Cluster the observed data in $X$ (which contains $n$ objects and $m$ features), varying the total number of clusters from $k = 1, 2, \ldots, K$, recording within dispersion measures (SSE function values) $W_k$, $k = 1, 2, \ldots, K$.

2. Generate $B$ reference datasets, each with $n$ objects having $m$ reference features generated uniformly over the range of the observed values for the original features in $X$. Cluster each of the $B$ reference datasets, recording within dispersion measures (SSE function values) $W^*_{kb}$, $b = 1, 2, \ldots, B$, $k = 1, 2, \ldots, K$. Compute the estimated Gap statistic:

$$Gap(k) = \left( \frac{1}{B} \right) \sum_b \log(W^*_{kb}) - \log(W_k)$$

3. Let $\bar{l} = \left( \frac{1}{B} \right) \sum_b \log(W^*_{kb})$ and compute the standard deviation:

$$sd_k = \sqrt{\left[ \left( \frac{1}{B} \right) \sum_b (\log(W^*_{kb}) - \bar{l})^2 \right]}^{1/2}.$$ 

Define $s_k = sd_k \sqrt{1 + \frac{1}{B}}$. Finally choose the number of clusters to be the smallest value of $k$ such that

$$Gap(k) \geq Gap(k + 1) - s_{k+1}$$

Figure 6.8: Results for gap statistic procedure on Ruspini data. The maximum gap occurs at $k^* = 4$ clusters.
6.2 Graph Methods Based on Eigenvalues (Perron Cluster Analysis)

Another commonly used methodology for determining the number of clusters relies upon the examination of eigenvalues of a graph Laplacian. Keeping with our focus in Chapter 3 we consider only undirected graphs. The methodology contained herein is motivated by the following observation: suppose we had an undirected graph consisting of \( k \) connected components (i.e. \( k \) distinct components, none of which are connected to any other). The adjacency matrix of such a graph would be block diagonal with \( k \) diagonal blocks \( A_1, \ldots, A_k \), and each diagonal block would itself be an adjacency matrix for one connected component.

\[
A = \begin{bmatrix}
A_1 & 0 & 0 & \ldots & 0 \\
0 & A_2 & 0 & \ldots & 0 \\
0 & 0 & A_3 & \ddots & 0 \\
0 & 0 & 0 & \ddots & \vdots \\
0 & 0 & 0 & \ldots & A_k \\
\end{bmatrix}
\]  
(6.4)

Thus, the Laplacian matrix \( L = D - A \) would also be block diagonal and each diagonal block would be the Laplacian matrix for one component of the graph.

\[
L = \begin{bmatrix}
L_1 & 0 & 0 & \ldots & 0 \\
0 & L_2 & 0 & \ldots & 0 \\
0 & 0 & L_3 & \ddots & 0 \\
0 & 0 & 0 & \ddots & \vdots \\
0 & 0 & 0 & \ldots & L_k \\
\end{bmatrix}
\]

with \( L_i e = 0 \) for \( i = 1, \ldots, k \)  
(6.5)

Thus, if each component is connected, the multiplicity of the smallest eigenvalue, \( \lambda_1 = 0 \), will count the number of diagonal blocks and thus the number of components. Of course the situation depicted in Eq. 6.5 is ideal and unlikely to be encountered in practice. However when the graph is nearly decomposable into disconnected components, continuity of the eigenvalues suggests that one may be able to count the number of tightly connected components by counting the number of eigenvalues near \( \lambda_1 = 0 \). In order to be able to characterize eigenvalues as being near \( \lambda_1 = 0 \), it is necessary to transform (normalize) the Laplacian matrix so that its spectrum is contained in the interval \([0, 1]\). This type of analysis is usually done using one of the two normalized Laplacian matrices discussed in Chapter 3 and defined again here.

1. The random-walk Laplacian

\[
L_{rw} = D^{-1}L = I - D^{-1}A = I - P
\]
2. The symmetric Laplacian

\[ L_{\text{sym}} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}AD^{-1/2}. \]

The normalized Laplacians, like the Laplacian matrix itself, are both positive definite. Furthermore, \( L_{\text{rw}} \) and \( L_{\text{sym}} \) have the same spectrum. The following well-known and easily verified fact characterizes the relationship between the eigenvalues and eigenvectors of these two matrices [24].

**Fact 1** (Eigenvalues of \( L_{\text{sym}} \) and \( L_{\text{rw}} \)). \( \lambda \) is an eigenvalue of \( L_{\text{rw}} \) with eigenvector \( v \) if and only if \( \lambda \) is an eigenvalue of \( L_{\text{sym}} \) with eigenvector \( w = D^{1/2}v \).

In light of this fact, we will limit our discussion to the properties of the transition probability matrix of a random walk on the graph associated with the adjacency matrix \( A \), denoted

\[ P = D^{-1}A = I - L_{\text{rw}}, \]

since

\[ \lambda \in \sigma(P) \Rightarrow (1 - \lambda) \in \sigma(L_{\text{rw}}). \]

Random walks on undirected graphs are *reversible Markov chains*, which satisfy the so-called *detailed balance equations* [63, 115]:

\[ QP = P^TQ \quad \text{where} \quad Q = \text{diag}(\beta). \]

The stationary distribution for \( P \) given by \( \beta^T = \frac{\beta^TD\beta}{e^TD\beta} \).

We assume the underlying graph (which we aim to partition) is connected so that the matrix \( P \) is irreducible. If the graph is composed of connected components, like the one associated with Eq. 6.4, the resulting random walk is equivalently referred to as *completely reducible, uncoupled, or completely decomposable* and there simple efficient algorithms available to identify the connected components [57].

In our connected graph, we assume that there exists some cluster or community structure, i.e. that there are \( k \) groups of vertices, \( C_1, C_2, \ldots, C_k \) with \( |C_k| = n_k \), for which edges exist more frequently and with higher weight within each group than between each group. With this assumption, we can reorder the rows and columns of the transition probability matrix \( P \) according to group membership so that the result is *block-diagonally dominant*. By this we essentially mean that \( P \) is a perturbation of a block-diagonal matrix \( B \), such that

\[
\begin{bmatrix}
B_{11} & E_{12} & E_{13} & \cdots & E_{1k} \\
E_{21} & B_{22} & E_{23} & \cdots & E_{2k} \\
E_{31} & E_{32} & B_{33} & \cdots & E_{3k} \\
& \vdots & & \ddots & \vdots \\
E_{k1} & E_{k2} & E_{k3} & \cdots & B_{kk}
\end{bmatrix}
\] (6.6)
where the off-diagonal blocks, $E_{ij}$, are much smaller in magnitude than the diagonal blocks. In fact, the entries in the off-diagonal blocks are small enough that the diagonal blocks are nearly stochastic, i.e. $B_{ii}e \approx 1$ for $i = 1, 2, \ldots, k$. A transition probability matrix taking this form describes a nearly uncoupled or nearly completely reducible Markov Chain. The degree to which a matrix is considered nearly uncoupled is dependent on one’s criteria for measuring the level of coupling (interconnection) between the aggregates (clusters of states) of the Markov chain [97, 86, 127]. In [86], the deviation from complete reducibility is defined as follows:

**Definition 10 (Deviation from Complete Reducibility).** For an $m \times n$ irreducible stochastic matrix with a $k$-level partition

\[
P = \begin{bmatrix}
P_{11} & P_{12} & P_{13} & \cdots & P_{1k} \\
P_{21} & P_{22} & P_{23} & \cdots & P_{2k} \\
P_{31} & P_{32} & P_{33} & \cdots & P_{3k} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
P_{k1} & P_{k2} & P_{k3} & \cdots & P_{kk}
\end{bmatrix}
\]

the number

\[
\delta = 2 \max_i \|P_{ix}\|_\infty
\]

is called the deviation from complete reducibility.

It is important to point out that the parameter $\delta$, or any other parameter that measures the level of coupling between clusters in a graph (like those suggested in [97, 127, 88]) cannot be computed without knowing a priori the clusters in the graph. Such parameters are merely tools for the perturbation analysis, used to present the following important fact regarding the spectrum of block-diagonally dominant stochastic matrices [97, 66, 85, 86, 88, 29, 19]:

87
Fact 2 (The Spectrum of a Block-Diagonally Dominant Stochastic Matrix [97, 86, 29, 85]). For sufficiently small $\delta \neq 0$, the eigenvalues of $P(\delta)$ are continuous in $\delta$, and can be divided into 3 parts:

1. The Perron root, $\lambda_1(\delta) = 1$,

2. a cluster of $k - 1$ eigenvalues $\lambda_2(\delta), \lambda_3(\delta), \ldots, \lambda_k(\delta)$ that approach 1 as $\delta \to 0$, and

3. the remaining eigenvalues, which are bounded away from 1 as $\delta \to 0$.

The cluster of $k$ eigenvalues surrounding and including the Perron root $\lambda_1 = 1$ is known as the Perron cluster [85, 97, 29]. The analysis in [85] explains that if there is no further decomposition (or meaningful sub-clustering) of the diagonal blocks, a relatively large gap between the eigenvalues $\lambda_k$ and $\lambda_{k+1}$ is expected. Thus, we can determine the number of clusters in the state space of a nearly uncoupled Markov chain (i.e. the number of clusters in a graph) by counting the number of eigenvalues in this Perron Cluster.

This method is extremely effective when the graph to be partitioned is sufficiently close to being uncoupled. Problems arise when either high levels of coupling (intercluster linkage) are in play or when some vertices within a cluster are weakly connected to that cluster (for example, dangling nodes - vertices with degree 1).

The examples in Figure 6.9 illustrate this point. In Figures 6.9a and 6.9b we show a synthetic example of a graph exhibiting cluster structure and the eigenvalues of the associated transition probability matrix respectively. The thickness of the edges in the graph correspond to their respective weights. Because there is a limited amount of coupling (intercluster connection) in this first example, the Perron cluster of eigenvalues is easy to identify. Because there are 3 eigenvalues near 1, the user would conclude that the graph has 3 clusters.

Occasionally a user can get a sense of the cluster structure in a graph with an appropriate layout of the nodes and edges. Force-directed graph drawing algorithms are common in this practice. The basic idea behind these algorithms is to model the edges as springs connecting the nodes and then to somehow minimize the total amount of tension in the system. Thus, densely connected groups of nodes are placed proximal to each other and the edges which loosely connect these groups are stretched. The graph drawings in Figures 6.9a, 6.9c, and 6.9e are all examples of force-directed layouts. Graph drawing algorithms are beyond the scope of this paper, but for information the interested reader should see, for example, [50, 7].

Figures 6.9c and 6.9d displays a real-world example using the hyperlink graph between a sample of 1222 American political blogs. Based upon the force-directed drawing of the graph, it is clear that there are 2 large communities or clusters in this graph. These clusters correspond to the liberal and conservative division of American politics. The Perron cluster is not easily identified on the eigenvalue plot in Figure 6.9d, and thus no conclusion should be drawn regarding the number of clusters in this data. However, after removing a large number of dangling nodes from the graph, or blogs which link only to a single neighboring page in the sampled population, a different picture comes to light. In Figures 6.9e and 6.9f we illustrate the effect of removing these dangling nodes (about 200 in total) on the eigenvalues of the transition probability matrix. Luckily, for this particular graph, removing the dangling nodes did not create more, a situation that is not guaranteed in general. The third eigenvalue
in the Perron cluster likely identifies the small group of 3 blogs that is now visible in the force directed drawing of the graph. Such small clusters are generally undesirable in graph partitioning, and since the eigenvalues tell the user nothing about the size or composition of the graph communities counted by the eigenvalues in the Perron cluster, this method must be used with caution!

In the next Chapter, we will introduce a similarity matrix that is well suited for this Perron-cluster analysis. Our method has the ability of estimating the number of clusters in very noisy and high-dimensional data when other methods fail.
Figure 6.9: Some Examples of Perron Cluster Identification
Chapter 7

Consensus Clustering

In Chapters 2 and 3 we presented a number of different clustering algorithms for both data clustering and graph partitioning. As was previously mentioned, any of the data clustering algorithms can be applied to graphs, but in order for the graph partitioning algorithms to be applied to data, a similarity matrix must first be constructed. One of the major objectives of this dissertation is to address one of the most fundamental problems in cluster analysis, which is summed up in the following “theorem” according to Jacob Kogan in [68] who attributed the fact to George Leitmann [76].

**Theorem 1** (The Fundamental Theorem of Cluster Analysis). *There does not exist a best method, that is, one which is superior to all other methods, for solving all problems in a given class of problems.*

One simple demonstration of this theorem is shown in Table 7.1. The table represents the accuracy (percentage of objects clustered correctly) of a number of different clustering algorithms from Chapters 2 and 3. The data used are sampled from a large corpus ($n = 11,000$) of text documents, all collected from webpages and stripped of html. The are 11 dominant topics (clusters) in this collection ranging from astronomy to finance to sport, each having 1,000 relevant documents. The different clusters in the dataset are specified by letters. Each column of Table 7.1 shows the algorithms’ accuracy on the subset of the clusters corresponding to the letters. The “best” performer on each subset is identified by the boxed in accuracy metric. As you can see from this table, even using different samples of data from the same population can cause some uncertainty in a cluster solution. The fundamental problem of clustering boils down to the following dilemma:

*Suppose that, using only the training information contained in Table 7.1, you had to choose an algorithm to cluster documents collected from a similar source and processed in a similar fashion. Which one should you choose?* If you happen to be a marketing executive at a billion dollar corporation, and the quality of the clustering results could have an enormous impact on your company’s performance in the next quarter, this question might make you a little uncomfortable!
Table 7.1: Accuracies of 7 Different Clustering Algorithms on 3 Subsets of Documents from the Same Corpus

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Subset 1 (DFJX)</th>
<th>Subset 2 (BCFG)</th>
<th>Subset 3 (GHI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.32</td>
<td>0.451</td>
<td>0.915</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.577</td>
<td>0.492</td>
<td>0.924</td>
</tr>
<tr>
<td>NMF</td>
<td>0.300</td>
<td>0.595</td>
<td>0.4386</td>
</tr>
<tr>
<td>k-means</td>
<td>0.581</td>
<td>0.607</td>
<td>0.607</td>
</tr>
<tr>
<td>PIC</td>
<td>0.485</td>
<td>0.266</td>
<td>0.643</td>
</tr>
<tr>
<td>NCut</td>
<td>0.449</td>
<td>0.269</td>
<td>0.6504</td>
</tr>
<tr>
<td>NJW</td>
<td>0.624</td>
<td>0.273</td>
<td>0.7257</td>
</tr>
</tbody>
</table>

7.1 Consensus Clustering

In light of the type algorithmic disagreement illustrated in Table 7.1, the goal of consensus methods (also called ensemble methods) is to somehow find an “average” solution using multiple algorithms. Previous approaches to consensus clustering will be outlined in Section 7.2 and a novel method will be presented in Section 7.3. To begin, we introduce some notation. Since consensus methods combine multiple solutions from multiple algorithms (or multiple runs of the same algorithm), we start with a cluster ensemble. A cluster ensemble, \( \mathcal{C} = \{C_1, C_2, \ldots, C_N\} \), is a set of \( N \) clusterings of the \( n \) data objects \( x_1, x_2, \ldots, x_n \). That is, each clustering \( C_j \) in the ensemble is a \( k_j \)-way partition of the data. Using our previous notation from earlier chapters, this means that

\[
C_j = [C_1, C_2, \ldots, C_{k_j}],
\]

where the number of clusters in each clustering may be allowed to vary. In Figure 7.1, we illustrate a simple example with \( N = 3 \) clusterings.

The information from a cluster ensemble is then recorded in a consensus matrix.

**Definition 11 (The Consensus Matrix).** Given a cluster ensemble, \( \mathcal{C} = \{C_1, C_2, \ldots, C_N\} \), of \( n \) data points \( x_1, x_2, \ldots, x_n \), the consensus matrix \( M \) is an \( n \times n \) matrix such that

\[
M(\mathcal{C})_{ij} = \# \text{ of times object } x_i \text{ was placed in the same cluster as } x_j \text{ in the ensemble } \mathcal{C}.
\]

One might prefer to think of the consensus matrix as the sum of individual adjacency matrices for each clustering in the ensemble. For a given clustering \( C_i \) we could define an adjacency matrix, \( A_i \) as

\[
A_{ij} = \begin{cases} 
1 & \text{if object } x_i \text{ was clustered with } x_j \\
0 & \text{otherwise}
\end{cases}
\]
Then the consensus matrix $M$ would be the sum of the adjacency matrices of each clustering in the ensemble:

$$M(C) = \sum_{i=1}^{N} A_i.$$ 

These two definitions are of course equivalent.

As an example, the consensus matrix for the ensemble depicted in Figure 7.1 is given in Figure 7.2.

The consensus matrix from Figure 7.2 is very interesting because the ensemble that was used to create it had clusterings for various values of $k$. The most reasonable number of clusters for the colored circles in Figure 7.1 is $k^* = 3$. The 3 clusterings in the ensemble depict
$k_1 = 3, k_2 = 4,$ and $k_3 = 5$ clusters. However, the resulting consensus matrix is clearly block-diagonal with $k^* = 3$ diagonal blocks! Thus, if we were to use the Perron-cluster method (Chapter 6 Section 6.2) to count the number of clusters in this dataset using the consensus matrix as the adjacency matrix for the graph, we would clearly see $k^* = 3$ eigenvalues equal to 1! Indeed, consensus matrices turn out to be a very good structures for determining the number of clusters in any type of data, as will be demonstrated in Chapter 8. This methodology will be revisited in Section 7.3. First we’d like to consider some practical differences between the consensus matrix and traditional similarity matrices.

### 7.1.1 Benefits of the Consensus Matrix

As a similarity matrix, the consensus matrix offers some benefits overs traditional approaches like the Gaussian or Cosine similarity matrices. One problem with these traditional methods is the curse of dimensionality discussed in Chapter 4. The measures of distance put into these metrics tend to lose meaning when applied to high-dimensional data. The range of values for the pairwise distances tightens as the dimensionality of the space grows, and little has been done to address this fact. In Figure 7.3 we show the distribution of similarity values for the same 1 million entries in a consensus matrix compared to the cosine similarity matrix. As you can see, the consensus approach allows a user to witness some very high levels of similarity in high-dimension data, whereas the cosine similarities have a much smaller range. The dataset is the Medlars-Cranfield-CISI collection discussed in Chapter 6. Such contrast is typical across high-dimensional datasets.

In addition, entries in the consensus matrix have depth. By this, we mean that they result from summing entries in adjacency matrices output by individual clustering algorithms, so more information is available about the meaning of each similarity value. The cosine of the angle between two data vectors $x_i$ and $x_j$ may tell us something about their correlation, but knowing, for instance that these two objects were clustered together by all algorithms with
Figure 7.3: Distribution of Similarity Values in Cosine vs. Consensus Matrix

$k \leq 5$, by some algorithms with $6 \leq k \leq 7$, and never when $k \geq 7$, provides a depth of insight not previously considered. While we do not use this information explicitly in our analysis, it may be beneficial in practical research.

The greatest benefit of using a consensus matrix for clustering is that it provides superior cluster information about the data. This has been demonstrated time and again in the literature [85, 106, 48, 91, 123, 82, 116, 60]. We add to the pile of evidence for this statement with the experiments in Chapter 8.

7.2 Previous Proposals for Consensus Clustering

In recent years, the consensus idea has been promoted by many researchers [9, 91, 82, 85, 96, 131, 93, 123, 40, 44, 116, 109, 48]. The main challenge to ensemble methods using multiple algorithms is generally identified to be the wide variety in the results produced by different algorithms due to the different cluster criteria inherent in each algorithm. Thus any direct combination of results from an ensemble will not often generate a meaningful result [104, 69].

Most often the consensus problem has been formulated as an optimization problem, where the optimal clustering, $\mathcal{C}^*$, minimizes some relative distance metric between $\mathcal{C}^*$ and all of the clusterings $\mathcal{C}_i$ in the ensemble. There are many ways to define the distance between to clusterings, for example one could take the minimum number of elements that need to be deleted for the two partitions to become identical [55]. Using $d(\mathcal{C}_1, \mathcal{C}_2)$ to denote some measure of distance between two different clusterings, we’d write

$$\mathcal{C}^* = \arg\min_{\mathcal{C}} \sum_{i=1}^{N} d(\mathcal{C}_i, \mathcal{C}).$$

(7.1)
This problem is known as the *median partition problem* in the literature and dates back to the 1965-'74 work of Régnier ([51]) and Mirkin ([90]) [44]. Alternatively, some authors use a relative validity metric (see Chapter 5) like the normalized mutual information \( \text{NMI}(\mathcal{C}_i, \mathcal{C}) \) in place of a distance function and attempt to maximize the objective function in Eq. 7.1 [116]. The median partition problem was shown by Krivanek and Moravek, and also by Wakabayashi, to be NP-complete [44], but many heuristics have since been proposed to find approximate solutions [116, 44, 9, 93].

We believe that these methods are bound to suffer because each clustering in the ensemble is given equal importance. Suppose we had 4 perfect clusterings and 1 terribly inaccurate clustering. These methods would not take into account the fact that the majority of the algorithms share 100% agreement on a perfect clustering, and instead may shift the optimal clustering away from perfection towards inaccuracy. Thus, we feel that the optimization in Eq. 7.1 leads to a “middle-of-the-road” solution or a compromise between algorithms, rather than a solution of “agreement” or consensus. In our method, outlined in the next section, the clustering algorithms act as a voting ensemble and continually move through a series of elections until some level of consensus is reached.

### 7.3 Iterative Consensus Clustering (ICC)

The consensus approach outlined herein is based on the work in [82, 123, 105, 85, 48] where the consensus matrix is treated as similarity matrix and used as input to a clustering algorithm to reach a final solution. In [82] the authors suggest using many runs of the \( k \)-means algorithm, initialized randomly, to build the consensus matrix and then using a spectral clustering method, such as normalized cut (NCut), to determine a final solution. In [123, 60, 48], the approach is again to build a consensus matrix using many runs of the \( k \)-means algorithm and then to cluster the consensus matrix with one final run of \( k \)-means. In [85] a consensus matrix is formed via \( k \)-means and then used as input to the stochastic clustering algorithm (SCA). While all these methods can generally provide better results than individual algorithms, they still rely on a single algorithm to make the final decision on cluster membership.

Our method uses a variety of algorithms, rather than just \( k \)-means, to create the initial cluster ensemble. Additionally each algorithm is paired with each of the dimension reductions outlined in Chapter 4, because it is often unclear which dimension reduction gives the best configuration of the data - each lower dimensional representation has the potential to separate different sets of clusters in the data. In this way, we essentially gather an initial round of *votes* for whether or not each pair of objects \((x_i, x_j)\) belong in the same cluster. These votes are collected in a consensus matrix \( M \) as defined in Definition 11.

Each algorithm in our ensemble is assumed to be reasonable, making good choices on cluster membership most of the time. It is inevitable that each of the clustering algorithms will make mistakes, but it is assumed that rarely will the majority of algorithms make the same mistake. To account for this error, we introduce a drop tolerance parameter, \( \tau \), for which entries in the consensus matrix \( M_{ij} \) < \( \tau N \) will be set to zero. In other words, \( \tau \) is the minimum
proportion of algorithms that must agree upon a single cluster relationship \((x_i, x_j)\) in order to keep those “votes” in the consensus matrix. This parameter is the subject of Section 7.3.1.

After the initial consensus matrix is formed, we use it as input to each of the clustering algorithms again. Essentially we start a debate between algorithms, asking each of them to use the collective votes of the ensemble to determine a second solution. Again these solutions are collected in a consensus matrix and the process repeats until a simple majority of algorithms agree upon one solution. Once the majority of algorithms have chosen a common solution, we say the algorithms have reached consensus and call that resulting solution the final consensus solution. This process is illustrated in the flow chart in Figure 7.4.

Before discussing the drop tolerance parameter \(\tau\) we show a simple example of iterated consensus clustering at work (without \(\tau\)) on the well-known Iris data (also called Fisher’s Iris data or Anderson’s Iris data). This dataset contains measurements of 4 attributes on 150 Iris flowers. The attributes are the sepal length, sepal width, petal length, and petal width. Among these flowers are 3 different species (i.e. clusters): setosa, virginica, and versicolor. Figure 7.5 shows every possible two dimensional scatter plot using the 4 given attributes (the x-axis represents the variable indicated in corresponding column, y-axis is variable indicated on
each row). Each point in the scatter plot represents one flower, colored according to its species. Most plots show good separation between the setosa species and other species however the versicolor and virginica species are more difficult to separate.

![Iris Data (red=setosa, green=versicolor, blue=virginica)](image)

Figure 7.5: Iris Data (red = setosa, green = versicolor, blue = virginica)

No dimension reduction is needed for this data. The data vectors were normalized to have unit length and we set 4 algorithms (PDDP, $k$-means (1 trial, initialized randomly), NMFBasic (using the ACLS factorization [72]), EMGM (1 trial, initialized randomly) to recover the $k = 3$ clusters. The results from each of these algorithms are displayed in Table 7.2.

This particular ensemble is a good example of one where median partition methods would likely fail because one algorithm, NMFBasic (see Chapter 4, Algorithm 15) performs quite poorly on this data, while the other 3 algorithms are quite accurate. It is helpful to look at the pairwise agreement (via the accuracy metric discussed in Chapter 5) between algorithms to get a sense of how much discrepancy there is in this initial round of votes. Since the agree-
Table 7.2: Accuracy of 4 Algorithms on Raw Iris Data

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.973</td>
</tr>
<tr>
<td>NMFBasic</td>
<td>0.673</td>
</tr>
<tr>
<td>EMGM</td>
<td>0.967</td>
</tr>
<tr>
<td>k-means</td>
<td>0.967</td>
</tr>
</tbody>
</table>

When the agreement metric is symmetric, we can view the algorithms’ agreement in a triangular array as is shown in Table 7.3.

Table 7.3: Pairwise Agreement Between Algorithms on Raw Data

<table>
<thead>
<tr>
<th></th>
<th>PDDP</th>
<th>NMFBasic</th>
<th>EMGM</th>
<th>k-means</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.693</td>
<td>0.993</td>
<td>0.993</td>
<td></td>
</tr>
<tr>
<td>NMFBasic</td>
<td>0.7</td>
<td>0.7</td>
<td></td>
<td>0.987</td>
</tr>
<tr>
<td>EMGM</td>
<td></td>
<td></td>
<td>0.987</td>
<td></td>
</tr>
</tbody>
</table>

Combining these results into a consensus matrix, and clustering that consensus matrix with each algorithm, we achieve an immediate consensus result because 3 of the 4 algorithms agree upon a common solution as seen in the second table of agreement. The boxed accuracy values in Table 7.4 indicate the consensus solution - this solution achieves the highest accuracy of the 4 algorithms on the raw dataset.

In fact, compiling the clustering results from Table 7.4 into a second consensus matrix (‘Consensus Matrix 2’) and again using all four algorithms results in a unanimous decision for the best solution:

The purpose of this example was only to give the reader some experimental motivation for our iterated consensus approach using a well-known dataset. Many more results on high-
Table 7.5: Accuracy of 4 Algorithms on Consensus Matrix 2

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.973</td>
</tr>
<tr>
<td>NMFBasic</td>
<td>0.973</td>
</tr>
<tr>
<td>EMGM</td>
<td>0.973</td>
</tr>
<tr>
<td>k-means</td>
<td>0.973</td>
</tr>
</tbody>
</table>

dimensional datasets are contained in Chapter 8. Unfortunately, there is no way to guarantee this level of “convergence” of the clustering algorithms in the iterated consensus process, however the phenomenon seems reasonable to expect when our clustering algorithms are working well and expressing a high level of agreement on the majority of the objects in the dataset. When the algorithms behave poorly (due to noisy or high-dimensional nature of the data), a drop tolerance parameter $\tau$ is put to work in an attempt to filter out some of these poor clustering decisions made by algorithms.  

7.3.1 The drop tolerance parameter $\tau$

When clustering high-dimensional datasets, we do not generally expect as great a performance from individual clustering algorithms. To combat some of this error, we introduce a parameter called the drop tolerance. This parameter, $\tau$, is the minimum proportion of algorithms that must agree upon a single cluster relationship $(x_i, x_j)$ in order to keep those “votes” in the consensus matrix. Thus, any entries in the consensus matrix $M_{ij} \leq \tau N$ are dropped (set to zero). Our justification to include a drop tolerance in our method stems from some basic probability notions that are commonly used to motivate ensemble methods from any area of data mining [98].

Suppose, for example, we expect each of the $N$ clustering algorithms in our ensemble to correctly determine the cluster relationship of pairs of points $(x_i, x_j)$ 75% of the time. Then, considering each algorithm’s decision as a Bernoulli trial with probability of successfully determining the cluster relationship $p = 0.75$, we can compute the probability that fewer than $L$ algorithms correctly determined a cluster relationship:

$$P(M_{ij} \leq L \mid (x_i, x_j) \text{ belong in same cluster }) \approx \sum_{i=0}^{L} \binom{N}{i} (0.75)^i (0.25)^{N-i}.$$  

So if we used $N = 16$ clustering algorithms, then the probability that only 4 of these
algorithms correctly put \( x_i \) and \( x_j \) in the same cluster is

\[
    P(M_{ij} \leq 4 \mid (x_i, x_j) \text{ belong in same cluster}) \approx \sum_{i=0}^{4} \binom{16}{i} (0.75)^i (0.25)^{16-i} = 3.8 \times 10^{-5}.
\]

Thus, it is extremely unlikely that only 4 algorithms out of 16 correctly place points \((x_i, x_j)\) in the same cluster, assuming that they belong together. Obviously, there are many simplifying assumptions at play in this analysis that do not necessarily hold up in reality. First, the probability of success of each algorithm is different. Additionally, for a single algorithm the probability of successfully determining the cluster relationship of a pair of points is likely to be different for each pair of points. We also cannot necessarily assume independence between algorithms because quite often our methods are connected. However, the values for the drop tolerance parameter \( \tau \) used in our experiments are generally conservative and thus the probabilities

\[
    P(M_{ij} \leq \tau N \mid (x_i, x_j) \text{ belong in same cluster})
\]

are assumed to be extremely small.

### 7.4 Determining the Number of Clusters via Consensus Clustering

Earlier in Section 7.1 an example was given that alluded to our methodology for determining the number of clusters. We approach this task using the Perron-cluster methodology discussed in the previous chapter, only we use a consensus similarity matrix for the analysis, which has not previously been considered. To build the consensus similarity matrix, we use one or more algorithms to cluster the data into a varying number of clusters. The main motivation behind this idea involves the following reasonable assumptions:

- If there are truly \( k \) distinct clusters in a given dataset, and a clustering algorithm is set to find \( \tilde{k} > k \), then the original \( k \) clusters will be broken apart into smaller clusters to make \( \tilde{k} \) total clusters.

- Further, if there is no clear “subcluster” structure, meaning the original \( k \) clusters do not further break down into meaningful components, then different algorithms (or even different initializations of the same algorithm) will break the clusters apart in different ways.

Based on these assumptions, we set the algorithm(s) in our ensemble to find \( \tilde{k}_1, \tilde{k}_2, \ldots, \tilde{k}_J \) clusters in the data. We then construct the consensus similarity matrix \( M \) from the resulting clusterings, examine the eigenvalues of the transition probability matrix \( P = D^{-1}M \), and estimate the number of eigenvalues in the Perron cluster. It is helpful in this algorithm to include the drop tolerance parameter in the construction of the consensus matrix because we may have allowed the algorithms to partition the data into fewer clusters than actually exist in the data. We return to the Medlars-Cranfield-CISI (MCC) collection of text abstracts.
from 3 different scientific disciplines which was discussed in Chapter 6 as an example where traditional SSE plots generally fail to provide a clear picture of how many clusters may be optimal. Before discussing the results of our method, we first look at the eigenvalues of the transition probability matrix that would result from using cosine as a measure of similarity (this is the most common similarity matrix used for text collections in the spectral clustering literature). The largest eigenvalues of this $3891 \times 3891$ matrix are displayed in Figure 7.6. The plot shows only one eigenvalue in the Perron cluster and therefore, as with the other methods discussed in Chapter 6, no information is gathered about the number of clusters in the data.

![Figure 7.6: Dataset MCC: 20 Largest Eigenvalues Found Using Cosine Similarity Matrix](image)

Now we look at the eigenvalues of the transition probability matrix associated with a consensus similarity matrix. This consensus matrix was build from an ensemble of various algorithms paired with different dimension reductions and different levels of dimension reduction. All 3 of the major dimension reduction techniques discussed in Chapter 4 (SVD, PCA, and NMF) were used to reduce the dimensions of the data to $r = 5, 10, \text{and } 20$ dimensions, creating a total of 10 data inputs (including the raw high-dimensional data) for each clustering algorithm. Three different clustering methods were used to cluster each data input: PDDP, spherical $k$-means initialized randomly, and spherical $k$-means initialized with centroids from the clusters found by PDDP. Counting every combination of dimension reduction and clustering procedure, the ensemble had 30 algorithms at work. For each of the 30 algorithms, $\tilde{k} = 2, 3, 4, \ldots, 10$ clusters were determined and the resulting 270 clusterings were collected in the consensus matrix $M$. We show in Figure 7.7 side-by-side images showing the eigenvalues of the transition probability matrix associated with the consensus similarity matrix with and without use of the drop tolerance parameter $\tau$. Particularly with text and other high-dimensional data, this drop tolerance parameter, by removing extraneous connections in the consensus graph, encourages a nearly uncoupled structure in the clustering results. This
uncoupling effect, even for conservative values of $\tau$, is clearly identified by the widened gap after $\lambda_3$ in the eigenvalue graphs.

![Eigenvalue plots](image)

(a) No Drop Tolerance ($\tau = 0.0$)  
(b) 10% Drop Tolerance ($\tau = 0.1$)

Figure 7.7: Dataset MCC: 20 largest eigenvalues found using consensus similarity matrices with (right) and without (left) the drop tolerance parameter $\tau$. Ensemble of 30 algorithms, each clustering data into $\tilde{k} = 2, 3, \ldots, 10$ clusters

The eigenvalue plots in Figure 7.7 reveal a relatively easily identifiable Perron-cluster containing $k' = 3$ eigenvalues as desired. We will revisit this consensus matrix in Chapter 8 to determine a final solution via Iterative Consensus Clustering.

7.4.1 Refining the Consensus Matrix through Iteration

We have seen with the iris data that the consensus matrix can provide better clustering information than the raw data. Therefore it seems reasonable that iterating the consensus process using multiple values of $\tilde{k}$ may refine the consensus matrix in way that minimizes or eliminates elements outside of the diagonal blocks, revealing a more identifiable Perron cluster. This is most often the case. Iterating the clustering process has an uncoupling effect on the consensus matrix [106]. In Figure 7.8a we show a matrix heat-map of a consensus matrix formed by clustering 700 documents into $\tilde{k} = [10, 11, \ldots, 20]$ clusters with 4 different algorithms and 3 different dimension reductions. Red pixels indicate high levels of similarity while yellow pixels represent lower levels of similarity. There is a considerable amount of noise outside of the diagonal blocks. This consensus matrix was then clustered by the same 4 algorithms and 3 dimension reductions, again into $\tilde{k} = [10, 11, \ldots, 20]$, and a heat map of the resulting consensus matrix (iteration 2) is shown in Figure 7.8b. It is easy to see the refinement of the clusterings by the reduction of noise outside the diagonal blocks. The difference is also clearly shown in the eigenvalue plots displayed in Figure 7.9.
7.5 Conclusions

In this chapter we have presented methods for both determining the number of clusters in a dataset and for reaching a final consensus solution using multiple algorithms. When determining the number of clusters, we employ Perron cluster analysis on the consensus matrix. In the event that the solution is unclear in the initial consensus matrix, we have suggested two courses of action which aim to refine the consensus matrix in a way that encourages a nearly uncoupled structure:
1. Implement a drop tolerance $\tau$ to remove erroneous cluster connections.

2. Iterate the clustering process by applying the same algorithms to the consensus matrix with the same numbers of clusters $\tilde{k}_1, \tilde{k}_2, \ldots, \tilde{k}_J$.

This approach is not specifically designed for any specific algorithm or group of algorithms, it is merely a framework with which a user can proceed as desired. Our objective in Chapter 8 is to demonstrate the flexibility of this approach using a few comprehensive examples on benchmark datasets and to compare the results of this procedure to more traditional methods. The complete framework for Iterative Consensus Clustering (ICC) is presented in Algorithm 17.

**Algorithm 17 Iterative Consensus Clustering (ICC) Framework**

**Part I: Determining the Number of Clusters** *(If desired number of clusters is known, skip to Part II.)*

**Input:** Data Matrix $X$, drop-tolerance $\tau$, and sequence $\tilde{k} = \tilde{k}_1, \tilde{k}_2, \ldots, \tilde{k}_J$

1. Using each clustering method $i = 1, \ldots, N$, partition the data into $\tilde{k}_j$ clusters, $j = 1, \ldots, J$

2. Form a consensus matrix, $M$ with the $JN$ different clusterings determined in step 1.

3. Set $M_{ij} = 0$ if $M_{ij} < \tau JN$.

4. Let $D = \text{diag}(Me)$. Compute the eigenvalues of $P$ using the symmetric matrix $I - D^{-1/2}MD^{-1/2}$.

5. If the Perron Cluster is clearly visible, stop and output the number of eigenvalues in the Perron cluster, $k$. Otherwise, repeat steps 1-5 using $M$ as the data input in place of $X$.

**Part II: Determining the Consensus Solution**

**Input:** Final consensus matrix from part I, drop tolerance $\tau$, and the number of clusters $k$. *(Or if desired number of clusters is known before hand, the raw data matrix X).*

1. Using each clustering method $i = 1, \ldots, N$, partition the matrix into $\tilde{k}_j$ clusters, $j = 1, \ldots, J$

2. If the majority of algorithms agree upon a single solution, stop and output this solution.

3. Form a consensus matrix, $M$ with the $JN$ different clusterings determined in step 1.

4. Set $M_{ij} = 0$ if $M_{ij} < \tau JN$.

5. Repeat steps 1-5.
8.1 Perron Cluster Analysis with Consensus Matrices

In Chapter 6, several methods for determining the number of clusters in data were discussed. In Chapter 7, the Medlars-Cranfield-CISI collection was used to demonstrate how Perron cluster analysis (Chapter 6, Section 6.2) on a consensus matrix might provide a clearer insight than Perron cluster analysis using traditional similarity matrices. Our objective in this section is to back up that claim with evidence from two datasets, upon both of which traditional methods for determining the number of clusters fail.

8.1.1 AGBLOG Dataset

We’ll begin our discussion with the AGBLOG dataset which was introduced in Chapter 6. The data is a network of hyperlinks between American political blogs containing two clusters, ‘liberal’ and ‘conservative’. Figure 6.9d displayed the eigenvalues of the transition probability matrix associated with the hyperlink network itself. For the raw hyperlink network no distinct Perron cluster was identifiable. We clustered this data using 3 of the classic graph clustering algorithms discussed in Chapter 3,

1. Power Iteration Clustering (PIC)
2. Normalized Cuts (NCUT)
3. Ng-Jordan-Weiss (NJW),

into $k = 2, 3, \ldots, 10$ clusters. The resulting 27 clusterings were recorded in a consensus matrix. Figure 8.1a reveals the eigenvalues of the transition probability matrix associated with our consensus matrix. There are 2 eigenvalues in the Perron cluster, correctly indicating the number of clusters or communities in this network. One could “tighten up” the Perron cluster by including a drop tolerance $\tau$ if desired. For completeness, the eigenvalue plot resulting from a drop tolerance of $\tau = 0.1$ is shown in Figure 8.1b.
Figure 8.1: Dataset AGBLOG: Eigenvalues Associated with the Consensus Matrices Without (left) and With (right) a Drop Tolerance of $\tau = 0.1$

8.1.2 Pendigits17 Dataset

PenDigits17 is a dataset, a subset of which was used in [39], which consists of coordinate observations made on handwritten digits. There are roughly 1000 instances each of $k = 2$ digits, ‘1’s and ‘7’s, drawn by 44 writers. Determining $k$ is difficult for this dataset because of the similarity of the two digits and the number of ways to draw each. The complete PenDigits dataset is available from the UCI machine learning repository [6]. In [39], the authors used the cosine similarity matrix to cluster this data using a spectral clustering algorithm. However, as shown in Figure 8.2a, Perron cluster analysis on the cosine similarity matrix does not indicate $k = 2$ clusters in the data. For completeness, we also performed this analysis on the Gaussian similarity matrix, resulting in Figure 8.2b. Here we chose the tuning parameter, $\sigma$, to provide the best possible Perron cluster. This choice required our knowledge of the number of clusters before hand, but stands to show that even in the best case scenario Perron cluster analysis on the Gaussian matrix leaves room for improvement.

We created two consensus matrices for this dataset to convince the reader of the flexibility of our approach.

**Ensemble 1:** First we used only the $k$-means algorithm and varied the number of clusters as $\hat{k} = 2, 3, 4, \ldots, 10$. For each value of $\hat{k}$, 3 iterations of the $k$-means algorithm were done, each with randomly initialized centroids. The eigenvalues associated with this first consensus matrix are shown in Figure 8.3a.

**Ensemble 2:** Next, we used a variety of data clustering algorithms, paired with different dimension reductions to cluster the data as it was projected into various subspaces. The rank of each dimension reduction was varied according to the number of principal components necessary to capture 60%, 75% and 90% of the total variance. Between the raw data and the
various projections there were 10 different data inputs to each of 4 different clustering algorithms as listed below:

Data Inputs:

1. Raw data \((16 \times 1557)\)
2. PCA dimension reductions \((r=3,4,7 \times 1557)\)
3. SVD dimension reductions \((r=3,4,7 \times 1557)\)
4. NMF dimension reductions \((r=3,4,7 \times 1557)\)

Clustering Algorithms:

1. \(k\)-means
2. EMGM
3. PDDP
4. PDDP-seeded \(k\)-means

Using these 40 combinations, the number of clusters were varied as \(\hat{k} = 5,6,\ldots,10\). The eigenvalues associated with this second consensus matrix are shown in Figure 8.3b.

In either case, the Perron cluster is easily identifiable and contains 2 eigenvalues, correctly determining the number of clusters.
8.2 Comprehensive Cluster Analysis using Iterative Consensus Clustering

For the next phase of our discussion, we’ll take a more comprehensive approach by comparing our method to traditional approaches for both determining the number of clusters and for determining a final cluster solution. Our objective is to demonstrate the flexibility of our approach to use many different algorithms together to arrive at a consensus solution with higher accuracy than the average of the algorithms used alone. For this purpose we use two different datasets, one scientific and one textual, both of which are popular in the clustering literature.

8.2.1 Wine Dataset

The wine dataset is one of the most popular collections used to test and evaluate clustering algorithms in the literature. It originated at the Institute of Pharmaceutical and Food Analysis Technologies in Genoa, Italy [6]. The dataset contains measurements on 178 samples of wine grown in the same region in Italy by 3 different cultivators. For each sample, 13 measurements were taken regarding the characteristics of the wine. All of the attributes are continuous. The attributes measured are as follows:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proantocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

The overall objective is to identify 3 different clusters in this data corresponding to the 3 different cultivators involved in the study. To eliminate the units on each variable and put the attributes on the same scale, the rows of the data matrix are normalized to have unit two-norm.

**WINE Data: Determining the Number of Clusters**

Our first step is to try to uncover the number of clusters in the data. To begin, we’ll look at some of the existing techniques discussed in Chapter 6, starting with the ever popular Sum of Squared Error (SSE) Plot. As a reminder this plot compares the $k$-means objective function value for various values of $k$. To avoid the unpredictable impact of $k$-means’s random initialization, the algorithm was run 10 times for each value of $k$ and the solution with the lowest objective function value was chosen. The SSE Plot for the wine data is shown in Figure 8.4.

The SSE plot does not appear to provide a definitive solution for the number of clusters, although it seems to indicate the number is probably in the narrow range $2 \leq k \leq 4$. To get a more definitive answer from the SSE plot, one may use the gap statistic to compare the intracluster dispersion to what one might expect under a uniform reference distribution. In Figure 8.5 we show the comparison of the SSE plot of the data to the average over 10 bootstrap samples from a uniform distribution. Figure 8.5a displays the log of the intracluster dispersion (SSE) for the observed wine data and what may be expected under the null hypothesis that the data is uniformly distributed across the range of the 13 variables. Figure 8.5b shows the width of the gap between these two values. Upon controlling for the sampling distribution and the simulation error according to the procedure discussed in Chapter 6, the final estimate for $k$ via the gap statistic (according to [122]) is 1 cluster.

For another approximation, we generate the plot of the Ray and Turi validity measure also discussed in Chapter 6. Recall for these plots the procedure is to locate the first local maximum and choose $k$ to be the first local minimum after this local maximum. In Figure 8.6,
the Ray and Turi validity measure $V$ was calculated for the wine data. The resulting approximation for $k$ is 4.

With only 13 attributes in the data, dimension reduction may seem unnecessary. However, an analysis of the principal component decomposition shows that 95% percent of the variance in the data can be captured with only 7 principal components. Although the curse of dimensionality is not a concern in this situation, dimension reduction still helps us get a
clearer picture of the differences between objects by comparing them across major axes of variation (such as principal components, singular vectors, or nonnegative topic vectors). To create our consensus matrix, we considered 10 different data inputs:

1. Raw data (13 × 178)
2. PCA dimension reductions (r=3,4,7 × 178)
3. SVD dimension reductions (r=3,4,7 × 178)
4. NMF dimension reductions (r=3,4,7 × 178)

Each data input was paired with 4 different clustering algorithms:

1. $k$-means
2. EMGM
3. PDDP
4. PDDP-seeded $k$-means

For each of these 40 combinations, the data was partitioned first into $\tilde{k} = 5, 6, 7, \ldots, 10$ clusters. The resulting 240 clusterings were compiled into a consensus matrix and the eigenvalues of the associated transition probability matrix are displayed in Figure 8.7a. In a second experiment, the data was partitioned into $\tilde{k} = 10, 11, \ldots, 20$ clusters, resulting in a total of 440 clusterings. The eigenvalues associated with the consensus matrix of the second ensemble are shown in Figure 8.7b.
Figure 8.7: Dataset WINE: Eigenvalues associated with consensus similarity matrices prior to iteration (right) and after iteration (left) using $\tilde{k} = 5, 6, \ldots, 10$ clusters. The $k^* = 3$ eigenvalues in the Perron cluster correctly identify the number of clusters in both ensembles.

The eigenvalue plots in Figure 8.7 show the potential of the consensus similarity matrix to reveal better cluster information than the popular Gaussian similarity matrix. For comparison, the eigenvalues associated with the Gaussian similarity matrix are shown in Figure 8.8. Table 8.1 summarizes the determined numbers of clusters for each method on the wine data.

Figure 8.8: Dataset WINE: Eigenvalues from Gaussian Similarity
Table 8.1: WINE Data: Summary of Results for Determining the Number of Clusters

<table>
<thead>
<tr>
<th>Method</th>
<th>Approximated k</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE Plot</td>
<td>$2 \leq k \leq 4$</td>
</tr>
<tr>
<td>Gap Statistic</td>
<td>1</td>
</tr>
<tr>
<td>Ray and Turi</td>
<td>4</td>
</tr>
<tr>
<td>Gaussian Similarity</td>
<td>2</td>
</tr>
<tr>
<td>ICC</td>
<td>3</td>
</tr>
<tr>
<td>True Number:</td>
<td>3</td>
</tr>
</tbody>
</table>

WINE Data: Determining a Cluster Solution

The next task is to determine a final clustering, assuming the true number of clusters (k=3) is known. In Tables 8.2 and 8.3 we show the results of some common data clustering and graph partitioning algorithms on the consensus matrices compared to traditional data inputs. As discussed in Chapter 3, the most common similarity matrix for graph clustering of scientific data with continuous attributes is the Gaussian similarity matrix, and thus we chose this matrix to compare the individual algorithms with our consensus approach. On the traditional data inputs, the accuracy varies greatly across the 8 algorithms from 61% to 94%, whereas the solutions using the consensus inputs are far more consistent and more accurate on the average. The EMGM algorithm is not well suited for consensus matrices and thus is omitted from these results. This EM algorithm assumes the multivariate input data consists of a mixture of Gaussian distributions and this assumption is not often met by the consensus matrices. For both traditional

Table 8.2: Dataset WINE: Accuracies for individual algorithms on raw data and consensus similarity matrices.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Raw Data</th>
<th>Consensus ($k = 5 : 10$)</th>
<th>Consensus ($k = 10 : 20$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.77</td>
<td>0.88</td>
<td>0.87</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.92</td>
<td>0.88</td>
<td>0.89</td>
</tr>
<tr>
<td>EMGM</td>
<td>0.75</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NMF-Basic</td>
<td>0.61</td>
<td>0.90</td>
<td>0.89</td>
</tr>
<tr>
<td>k-means</td>
<td>0.92</td>
<td>0.90</td>
<td>0.86</td>
</tr>
<tr>
<td>Average</td>
<td>0.79</td>
<td>0.89</td>
<td>0.88</td>
</tr>
</tbody>
</table>

data clustering algorithms and spectral clustering algorithms, using the consensus similarity matrices in place of traditional data inputs is able to increase the average accuracy of the algorithms. The problem of which algorithm to choose still remains. It is important to keep in mind the fundamental “theorem” of clustering which tells us that reliance upon a single algorithm, even with a superior input structure, can be problematic. Our iterated consensus
Table 8.3: Dataset WINE: Accuracies for spectral algorithms on different similarity matrices.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Gaussian Consensus ($k = 5 : 10$)</th>
<th>Consensus ($k = 10 : 20$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC</td>
<td>0.79</td>
<td>0.89</td>
</tr>
<tr>
<td>NCUT</td>
<td>0.94</td>
<td>0.88</td>
</tr>
<tr>
<td>NJW</td>
<td>0.81</td>
<td>0.89</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.85</strong></td>
<td><strong>0.89</strong></td>
</tr>
</tbody>
</table>

This approach, takes the improved solutions from each algorithm on the consensus matrix and iterates the consensus process in an attempt to reach a majority agreement on a single cluster solution.

Taking all of the algorithms’ solutions on the initial consensus matrices (for which the value of $k$ varied) and compiling them into a final consensus matrix, we again run the algorithms on the final consensus matrix in an attempt to have the algorithms sort out any disagreements. In our discussion of this final phase of analysis, we will use boxes around accuracies to reflect a single solution for which pairwise algorithm agreement is 100%. In Table 8.4 we see that 6 of the 7 algorithms have chosen a common solution, and that the accuracy of this consensus solution is 89%.

Table 8.4: Dataset WINE: 6 of 8 algorithms find a common solution in one iteration of the final consensus process

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Final Consensus Iter 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.89</td>
</tr>
<tr>
<td>PDDP-$k$-means</td>
<td>0.89</td>
</tr>
<tr>
<td>NMF-Basic</td>
<td>0.89</td>
</tr>
<tr>
<td>$k$-means</td>
<td>0.89</td>
</tr>
<tr>
<td>PIC</td>
<td>0.67</td>
</tr>
<tr>
<td>NCUT</td>
<td>0.89</td>
</tr>
<tr>
<td>NJW</td>
<td>0.89</td>
</tr>
</tbody>
</table>

The final consensus process results in a single clustering upon which the majority of algorithms agree. This accuracy of this final solution is above the average accuracy of the individual algorithms run on any of the data inputs shown in Tables 8.2 and 8.3.

8.2.2 Newsgroups 6 Collection (NG6)

The Newsgroups 6 (NG6) dataset is a subset from the infamous "Twenty Newsgroups" text corpus that has become a common benchmark for cluster analysis. The Twenty Newsgroups corpus consists of approximately 20,000 news documents (web articles) partitioned somewhat
evenly across 20 different topics. The collection of these documents is attributed to Ken Lang, although it is never mentioned explicitly in his original paper [71]. It is now publicly available via the web [27]. To create the NG6 collection, 300 documents from 6 topics were randomly selected, resulting in a term-document matrix with 11,324 terms and 1800 documents. Our initial consensus matrix (used to determine the number of clusters) was formed using only the k-means algorithm (randomly initialized), performed on the raw data and 3 dimension reductions.

NG6 Data: Determining the Number of Clusters

As usual, we begin our exploration of the data by trying to ascertain how many clusters are present. First we examine the SSE plot to see if there is any clear indication from the k-means objective function as the value of $k$ increases. Since we are working with text data, we implement spherical $k$-means. The resulting SSE plot is displayed in Figure 8.9. The SSE values decrease quite smoothly as $k$ increases, providing the user with little intuition as to how many clusters are present.

![Figure 8.9: Dataset NG6: SSE Plot for Spherical $k$-means](image)

Next, we examine the plot of Ray and Turi’s validity measure, shown in Figure 8.10. From this plot we obtain an estimate of either $k^* = 3$ or $k^* = 9$ clusters.

Because of the size of this dataset, obtaining decent information regarding the gap statistic would be extremely time consuming. Instead, we proceed to our consensus approach. For this data, we show how our method can be implemented successfully using only the $k$-means algorithms and the various dimension reduction inputs. To determine an appropriate rank for dimension reduction we follow convention by observing the screeplot (plot of the singular values) for the NG6 data matrix. The screeplot shown in Figure 8.11, indicates our decision to reduce the dimensions of the data from $m = 11,324$ to $r = 10$. The dimensionality
of the data was then reduced using our 3 preferred dimension reduction algorithms:

1. Principal Components Analysis
2. Singular Value Decomposition
3. Nonnegative Matrix Factorization

and 10 iterations of $k$-means clustering was performed on each dimension reduction to create $\tilde{k} = 10, 11, 12, \ldots, 20$ clusters. Since the clustering is only performed on the reduced data, this phase of the process proceeds extremely fast. The result was a total of 330 clusterings,
combined in an initial consensus matrix. The next step in our analysis is to examine the eigenvalues of the transition probability matrix associated with this initial consensus matrix.

![Figure 8.12: Dataset NG6: Eigenvalues associated with Initial (unadjusted) Consensus Matrix](image)

As seen in Figure 8.12 the Perron cluster is not entirely obvious in this initial consensus matrix. As discussed in Chapter 7, there are two adjustments the user may proceed with.

1. Implement a drop tolerance $\tau$ to distinguish the Perron cluster
2. Iterate the consensus procedure using the initial consensus matrix as input (no dimension reduction was used here).

In Figure 8.13 the results of both adjustments are shown. In either scenario, a Perron cluster with $k^* = 6$ eigenvalues becomes clear.

For the purposes of comparison, we present in Figure 8.14 the eigenvalues of the transition probability matrix associated with the Cosine similarity matrix, which is commonly used to cluster document datasets with spectral algorithms. No convincing information regarding the number of clusters is revealed by these eigenvalues.

**SSE Plots of the Consensus Matrix** Another way to explore the number of clusters is to use our traditional techniques with the consensus matrix as input. In Figure 8.15 we can again see the power of the consensus matrix to show how many clusters belong in the data. The SSE plot shows the total intracluster sum of squares levels off dramatically at $k = 6$ clusters, as desired.
Figure 8.13: Dataset NG6: Eigenvalues associated with consensus similarity matrices adjusted by drop tolerance (right) or adjusted by iteration (left) using $\tilde{k} = 10, 11, 12, \ldots, 20$ clusters. The $k^* = 6$ eigenvalues in the Perron cluster correctly identify the number of clusters.

Figure 8.14: Dataset NG6: Eigenvalues associated with Cosine Similarity Matrix

NG6 Data: Determining a Cluster Solution

Comparing the Consensus Matrix as Input  Tables 8.5, 8.6 and 8.7 demonstrate the superiority of the consensus matrices to traditional data inputs. The accuracies of certain algorithms increase by as much as 60% when the consensus matrix is used as input compared with the raw data. The average accuracies of all the algorithms also increase dramatically. One interesting fact to point out is that in this example, the algorithms perform quite poorly on
the NMF dimension reduction. Even though these results are contained in the consensus matrices, the nature of the process is able to weed out these poor results. The authors have experimentally discovered time and again that the ICC process is not sensitive to a small number of poor results contained in the cluster ensemble. When it comes to clustering the ensemble, such results are essentially “voted out.”

Table 8.5: Dataset NG6: Accuracies for individual algorithms on raw data and dimension reductions.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Raw Data</th>
<th>PCA $r = 10$</th>
<th>SVD $r = 10$</th>
<th>NMF $r = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.32</td>
<td>0.75</td>
<td>0.80</td>
<td>0.31</td>
</tr>
<tr>
<td>PDDP-$k$-means</td>
<td>0.77</td>
<td>0.99</td>
<td>0.99</td>
<td>0.60</td>
</tr>
<tr>
<td>NMF-Basic</td>
<td>0.46</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$k$-means</td>
<td>0.18</td>
<td>0.81</td>
<td>0.68</td>
<td>0.18</td>
</tr>
<tr>
<td>Average</td>
<td>0.43</td>
<td>0.85</td>
<td>0.82</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Dataset NG6: Determining a Final Solution

Our second step in the ICC process, once the number of clusters has been determined via the eigenvalue analysis, is to iterate the consensus procedure using the determined number of clusters in an attempt to witness agreement between algorithms. Combining the clustering results of each algorithm on Consensus B (Column 3 in Table 8.6 and Column 4 in Table 8.7) into another consensus matrix, we run through a second round of “voting”. The accuracies of
Table 8.6: Dataset NG6: Accuracies for individual algorithms on 3 different consensus matrices.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Initial Consensus</th>
<th>Consensus A</th>
<th>Consensus B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.85</td>
<td>0.93</td>
<td>0.98</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.96</td>
<td>0.94</td>
<td>0.98</td>
</tr>
<tr>
<td>NMF-Basic-k-means</td>
<td>0.98</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>k-means</td>
<td>0.75</td>
<td>0.72</td>
<td>0.78</td>
</tr>
<tr>
<td>Average</td>
<td>0.88</td>
<td>0.89</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Table 8.7: Dataset NG6: Accuracies for spectral algorithms on different similarity matrices.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cosine</th>
<th>Initial Consensus</th>
<th>Consensus A</th>
<th>Consensus B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC</td>
<td>0.52</td>
<td>0.86</td>
<td>0.97</td>
<td>0.98</td>
</tr>
<tr>
<td>NCUT</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>0.99</td>
</tr>
<tr>
<td>NJW</td>
<td>0.87</td>
<td>0.82</td>
<td>0.80</td>
<td>0.99</td>
</tr>
<tr>
<td>Average</td>
<td>0.79</td>
<td>0.89</td>
<td>0.92</td>
<td>0.99</td>
</tr>
</tbody>
</table>

the resulting solutions are given in Table 8.8, where boxes have been drawn around accuracies to indicate the final consensus solution upon which those algorithms agree completely.

Table 8.8: Dataset NG6: 4 of 7 algorithms find a common solution in one iteration of the final consensus process

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Final Consensus Iter 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.83</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.99</td>
</tr>
<tr>
<td>NMF-Basic-k-means</td>
<td>0.99</td>
</tr>
<tr>
<td>k-means</td>
<td>0.99</td>
</tr>
<tr>
<td>PIC</td>
<td>0.99</td>
</tr>
<tr>
<td>NCUT</td>
<td>0.99</td>
</tr>
<tr>
<td>NJW</td>
<td>0.82</td>
</tr>
</tbody>
</table>

8.3 Reaching Consensus

In Chapter 7, Section 7.4, the Medlars-Cranfield-CISI text collection was used to demonstrate the ability of ICC to correctly determine the number of clusters, $k = 3$. To complete the cluster
8.3.1 Medlars-Cranfield-CISI (MCC)

The MCC data introduced in Chapter 7 is high-dimensional with \( m \approx 11,000 \) features (words). As a result, clustering algorithms tend to run slowly on the raw data. Thus, we reduce the dimensions of the data using our 3 preferred algorithms:

1. Nonnegative Matrix Factorization (NMF) by Alternating Constrained Least Squares (ACLS) [72]

2. Singular Value Decomposition (SVD) [33, 28]

3. Principal Components Analysis (PCA) [65]

For each dimension reduction technique, the number of features is reduced from \( m = 11,000 \) to \( r = 5, 10, \) and \( 20 \) creating a total of 9 input data sets. On each input dataset, 3 different clustering methods were used to cluster the data:

1. \( k \)-means

2. PDDP

3. PDDP-\( k \)-means

The accuracy of each algorithm on each data input are given in Table 8.9.

Table 8.9: Accuracy Results for 3 Clustering Algorithms on 9 Low Dimensional Representations of the Medlars-Cranfield-CISI text data

<table>
<thead>
<tr>
<th># Features</th>
<th>Algorithm</th>
<th>NMF input</th>
<th>SVD input</th>
<th>PCA input</th>
</tr>
</thead>
<tbody>
<tr>
<td>((r = 5))</td>
<td>(k)-means</td>
<td>0.8741</td>
<td>0.7962</td>
<td>0.8260</td>
</tr>
<tr>
<td></td>
<td>PDDP</td>
<td>0.9599</td>
<td>0.9049</td>
<td>0.9026</td>
</tr>
<tr>
<td></td>
<td>PDDP-(k)-means</td>
<td>0.9599</td>
<td>0.9049</td>
<td>0.9026</td>
</tr>
<tr>
<td>((r = 10))</td>
<td>(k)-means</td>
<td>0.8628</td>
<td>0.8268</td>
<td>0.8286</td>
</tr>
<tr>
<td></td>
<td>PDDP</td>
<td>0.9764</td>
<td>0.9774</td>
<td>0.9481</td>
</tr>
<tr>
<td></td>
<td>PDDP-(k)-means</td>
<td>0.9764</td>
<td>0.9774</td>
<td>0.9481</td>
</tr>
<tr>
<td>((r = 20))</td>
<td>(k)-means</td>
<td>0.8530</td>
<td>0.8263</td>
<td>0.8281</td>
</tr>
<tr>
<td></td>
<td>PDDP</td>
<td>0.9722</td>
<td>0.9802</td>
<td>0.9478</td>
</tr>
<tr>
<td></td>
<td>PDDP-(k)-means</td>
<td>0.6114</td>
<td>0.9802</td>
<td>0.9478</td>
</tr>
</tbody>
</table>

Average Accuracy of All Clusterings: 0.90
The accuracy of the results ranges from 61% (>1,500 misclassified) to 98% (78 misclassified documents). A reasonable question one might ask is this: Why not choose the solution with the lowest $k$-means objective value? The problem with this boils down to the curse of dimensionality: the distance measures used to compute such metrics lose their meaning in high-dimensional space (see Chapter 4). The only comparison we could make between clusterings would be with the full dimensional data, and surprisingly the objective function values for the minimum accuracy solution is approximately equal to the maximum accuracy solution! One must be very careful when attempting to compare high-dimensional clusterings with such methods.

Our suggestion is instead to compile the clusterings from Table 8.9 into a consensus matrix, cluster that consensus matrix with multiple algorithms, and repeat that process until the majority of the algorithms agree upon a solution. This can be done with or without dimension reduction on the consensus matrix. For simplicity, we’ll proceed without reducing the dimensions of the consensus matrix, but we will include an additional clustering algorithm, NMFCluster, which was not well suited for the analysis on the low-dimensional representations in Table 8.9. Table 8.10 provides the accuracy of these 4 clustering algorithms on the consensus matrices through iteration. Boxes are drawn around values to indicate a common solution chosen by algorithms. A final consensus solution is found in the third iteration with 3 of the 4 algorithms agreeing upon a single solution. The accuracy of this final consensus solution is much greater than the average of all the initial clustering results in Table 8.9.

Table 8.10: Medlars-Cranfield-CISI text collection: Accuracies for 4 Clustering Algorithms on Consensus Matrices through Iteration

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Consensus Iter 1</th>
<th>Consensus Iter 2</th>
<th>Consensus Iter 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.939</td>
<td>0.969</td>
<td>0.969</td>
</tr>
<tr>
<td>PDDP-$k$-means</td>
<td>0.954</td>
<td>$0.966$</td>
<td>$0.966$</td>
</tr>
<tr>
<td>NMFcluster</td>
<td>0.969</td>
<td>0.954</td>
<td>0.966</td>
</tr>
<tr>
<td>$k$-means</td>
<td>0.966</td>
<td>$0.966$</td>
<td>$0.966$</td>
</tr>
</tbody>
</table>

### 8.3.2 Benchmark Text Collection by Sinka and Corne

In Chapter 7, the following table (Table 7.1: Accuracies of 6 different clustering algorithms on 3 subsets of documents from the same corpus) was presented as evidence of the uncertainty that comes with the use of a single clustering algorithm:
We will now take a closer look at our consensus solutions to these 3 subsets of documents. The table above is not entirely fair, because the clustering algorithms were run on the raw text data which, due to the curse of dimensionality, is not generally good practice. But the question of which algorithm to use then multiplies because one must first choose a method for dimensionality reduction and then choose a clustering algorithm to cluster the data. Iterative Consensus Clustering eliminates these questions by allowing us to use all the algorithms to reach an above-average solution.

The data at hand was created by Mark Sinka and David Corne from the department of computer science at the University of Reading as a benchmark collection for testing text-mining algorithms [113]. The corpus consists of 11,000 documents extracted from the web, 1000 documents from each of 11 different topics. The collection contains 4 broad topics:

- Banking/Finance
- Programming
- Science
- Sport.

Each of these broad topic breaks down into 2 or 3 subcategories:

- Banking/Finance
  1. Commercial Banks (Group A)
  2. Building Societies (Group B)
  3. Insurance Agencies (Group C)

- Programming
  1. Java (Group D)
  2. C/C++ (Group E)
  3. Visual Basic (Group F)

- Science
1. Astronomy (Group G)
2. Biology (Group H)

- Sports
  1. Soccer (Group I)
  2. Motor Sport (Group J)
  3. Water Sports (Group X)

This dataset is extremely noisy. Many of the documents appear to be nothing more than a list of words, others contain only street addresses and hyperlinks. Some documents are very long, containing comment postings from blogs and consumer sites. Other documents are less than 5 words in length. For more information on how the dataset was engineered and how to obtain the dataset, see [113]. The original documents were in html format and for these experiments we simply removed all the html and javascript tags and parsed the text that remained. We used Zeimpekis and Gallopoulos’s Text-Matrix Generator (TMG) for MATLAB [135] to parse the documents and apply stemming. The term frequencies were weighted using TF-IDF term weighting (see Chapter 1, Section 2) and normalization.

We created 3 subsets of this data, each consisting of all the documents in several categories. The categories chosen for each dataset were meant to contain some overlapping topics.

**BenchmarkDFJX**

The first subset, ‘BenchmarkDFJX’ consists of 4000 documents from groups D, F, J, and X. Groups D and F contain overlapping topics related to programming while groups J and X contain overlapping topics related to sports. For this reason the documents are difficult to cluster into 4 clusters - according to all methods for determining the number of clusters, including our consensus method, the appropriate number of clusters for this collection is 2. For this reason, this dataset is an interesting test as to whether we can achieve agreement among algorithms for 4 clusters.

All of the clusterings from Table 8.11 (16 total) were recorded in a consensus matrix to be clustered. This process was iterated until a final solution was determined by majority rule. Table 8.12 shows the accuracies of the algorithms as the consensus clustering process is iterated. The algorithms begin to agree on the second iteration and reach a majority solution on the third. The majority solution is well above average when compared to the individual algorithms on the raw data and dimension reductions.

**BenchmarkBCFG**

A similar process was done for the BenchmarkBCFG collection, however, instead of using every algorithm combination, we simply used $k$-means to show how the consensus matrix
Table 8.11: Dataset BenchmarkDFJX: Accuracy results for raw data and dimension reductions

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Raw Data</th>
<th>NMF</th>
<th>PCA</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.320</td>
<td>0.424</td>
<td>0.406</td>
<td>0.600</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.577</td>
<td>0.530</td>
<td>0.643</td>
<td>0.759</td>
</tr>
<tr>
<td>NMF -Basic</td>
<td>0.300</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>k-means</td>
<td>0.581</td>
<td>0.726</td>
<td>0.747</td>
<td>0.6365</td>
</tr>
<tr>
<td>PIC</td>
<td>0.485</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NCut</td>
<td>0.449</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NJW</td>
<td>0.624</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 8.12: Dataset BenchmarkDFJX: Accuracy results for consensus matrices through iteration

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Consensus</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.729</td>
<td>0.736</td>
<td>0.733</td>
<td>0.7347</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.7334</td>
<td>0.738</td>
<td><strong>0.738</strong></td>
<td>0.738</td>
</tr>
<tr>
<td>NMF -Basic</td>
<td>0.6922</td>
<td>0.743</td>
<td><strong>0.738</strong></td>
<td>0.738</td>
</tr>
<tr>
<td>k-means</td>
<td>0.6255</td>
<td>0.555</td>
<td>0.747</td>
<td><strong>0.738</strong></td>
</tr>
<tr>
<td>PIC</td>
<td>0.641</td>
<td>0.625</td>
<td>0.625</td>
<td>0.6275</td>
</tr>
<tr>
<td>NCut</td>
<td>0.738</td>
<td>0.739</td>
<td><strong>0.738</strong></td>
<td>0.738</td>
</tr>
<tr>
<td>NJW</td>
<td>0.665</td>
<td>0.666</td>
<td>0.743</td>
<td>0.738</td>
</tr>
</tbody>
</table>

can improve our results. For each data input (raw data and dimension reductions), 100 iterations of k-means clustering was done. For each of these 100 trials, we recorded the minimum/maximum accuracy solution, the average accuracy of the solutions and the accuracy of the most frequent solution - referred to below as the mode accuracy. From the results in Table 8.13 it is clear that dimension reduction increases the values of all 4 of these statistics. The mode frequency reflects the number of times (out of 100) that k-means converged to the mode solution. The final row of the table gives the total time in seconds taken to compute both the dimension reduction and the 100 iterations of k-means clustering.

Next, we took all of these clusterings (400 iterations in total) and compiled them into a consensus matrix and again ran 100 iterations of k-means. The resulting statistics are in Table 8.14.

Using the consensus matrix as input, the most accurate solution occurs more than half the time!
Table 8.13: Dataset BenchmarkBCFG: Accuracy statistics for 100 trials of k-means algorithm on raw data and dimension reductions

<table>
<thead>
<tr>
<th>Dimension Reduction:</th>
<th>Raw Data</th>
<th>NMF</th>
<th>PCA</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0.49</td>
<td>0.51</td>
<td>0.54</td>
<td>0.54</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.93</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>Average</td>
<td>0.77</td>
<td>0.82</td>
<td>0.85</td>
<td>0.80</td>
</tr>
<tr>
<td>Mode</td>
<td>0.62</td>
<td>0.90</td>
<td>0.89</td>
<td>0.91</td>
</tr>
<tr>
<td>Mode Frequency</td>
<td>2</td>
<td>14</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>Total Time (s)</td>
<td>1316</td>
<td>213</td>
<td>284</td>
<td>103</td>
</tr>
</tbody>
</table>

Table 8.14: Dataset BenchmarkBCFG: Accuracy statistics for 100 trials of k-means algorithm on consensus matrix

<table>
<thead>
<tr>
<th>Consensus Matrix</th>
<th>min</th>
<th>max</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>0.59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>0.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>0.81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>0.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode count</td>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total time (s)</td>
<td>594</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BenchmarkGHI**

Proceeding to the next column of Table 7.1, we continue with our original consensus process using multiple algorithms. Here we find a very interesting result. Table 8.15 provides the accuracies of each clustering algorithm on each data input (the raw data and the three dimension reductions).

Table 8.15: Dataset BenchmarkGHI: Accuracy results for raw data and dimension reductions

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Raw Data</th>
<th>NMF</th>
<th>PCA</th>
<th>SVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.915</td>
<td>0.346</td>
<td>0.4975</td>
<td>0.600</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.924</td>
<td>0.4149</td>
<td>0.8665</td>
<td>0.759</td>
</tr>
<tr>
<td>NMF -Basic</td>
<td>0.439</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>k-means</td>
<td>0.607</td>
<td>0.4149</td>
<td>0.8612</td>
<td>0.6365</td>
</tr>
<tr>
<td>PIC</td>
<td>0.643</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NCut</td>
<td>0.650</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NJW</td>
<td>0.726</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
The results in the table leave much to be desired, as most of the algorithms do relatively poor on their own. The clusterings from Table 8.15 were then combined into a consensus matrix to begin our iterative process. The results on the consensus matrices through iteration are given in Table 8.16.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Consensus</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDDP</td>
<td>0.933</td>
<td>0.934</td>
<td>0.933</td>
</tr>
<tr>
<td>PDDP-k-means</td>
<td>0.933</td>
<td>0.933</td>
<td>0.933</td>
</tr>
<tr>
<td>NMF-Basic</td>
<td>0.9267</td>
<td>0.933</td>
<td>0.933</td>
</tr>
<tr>
<td>k-means</td>
<td>0.933</td>
<td>0.933</td>
<td>0.933</td>
</tr>
<tr>
<td>PIC</td>
<td>0.671</td>
<td>0.931</td>
<td>0.933</td>
</tr>
<tr>
<td>NCut</td>
<td>0.933</td>
<td>0.933</td>
<td>0.933</td>
</tr>
<tr>
<td>NJW</td>
<td>0.668</td>
<td>0.673</td>
<td>0.902</td>
</tr>
</tbody>
</table>

In this example, what seems like a very surprising thing happens: the consensus solution actually has higher accuracy than any of the clusterings that were put into the initial consensus matrix. This serendipitous situation seems to occur when the algorithms initially have very high levels of disagreement between solutions, but similar levels of accuracy to the correct solution. In other words, the algorithms are clustering a similar proportion of objects correctly, but the actual objects clustered correctly are varying across algorithms. This allows for the correct solution to become clear as the sum of many inaccurate parts. We believe this special situation is due in large part to the ability of different dimension reductions to represent the data in different spacial configurations. For example, one of the three clusters might be well separated by the nonnegative matrix factorization, whereas another might be more distinct in the singular value configuration. Combining both pictures in the consensus process allows the algorithms to take votes from both configurations into account, settling on a more accurate solution.
Chapter 9

Conclusions

The purpose of this dissertation is to address two of the major problems that arise in applied cluster analysis: determining the number of clusters and determining a final solution from multiple algorithms. The second problem was initially our focus, and our success in this area led to a natural extension to tackle the first.

For years researchers have been innovating novel methods for cluster analysis, but it has been a well accepted fact by all that no method is superior to all others for any type of real data. When many tools exist for a user to choose from, but no hard and fast guidelines are in place for making the choice of one particular tool over the other, there is a problem. We have demonstrated the weight of this problem by showing just how temperamental these algorithms can be, even on subsets of data points sampled from the same population. This inconsistency of performance in cluster analysis is not unique to the clustering algorithms themselves. In fact, the dimension reduction techniques that are expected to aid these algorithms by revealing the cluster tendencies of data also tend to compete unpredictably, and it is difficult to know beforehand which low-dimensional approximation might provide the best separation between clusters. Having many tools and few ways to make an informed decision on which tool to use, high-dimensional cluster analysis was doomed to become an ad hoc science where users blindly reach for a tool and hope for the best. Cluster analysis is not the first type of data analysis to encounter this problem. Data scientists were quick to develop ensemble techniques to escape the unreliability of individual algorithms for tasks like prediction and classification. Ensemble methods have become an integral (perhaps even classical) concept in many areas of data mining, but for cluster analysis such methods have been largely ignored.

Herein we have presented a flexible framework for combining results from multiple clustering algorithms and/or multiple data inputs. Not only does this framework provide the user with an above average clustering solution, it also contains a practical exploratory procedure for determining the number of clusters. We conclude our discussion with a summary of our contributions to and plans for future work.
9.1 Contributions

9.1.1 Determining the Number of Clusters

Determining the number of clusters in data has long been considered one of the more difficult aspects of cluster analysis. This fact boils down to basics: what is a cluster? How do we define what should and should not count as two separate clusters? Our approach provides an original answer to this question. A group of points should be considered a cluster when a variety of algorithms agree that they should be considered a cluster. If all the algorithms can easily agree on how to break a dataset into two clusters, but cannot agree on how to partition the data into more than two clusters, then we determine the data has two clusters. This is the essence of our approach. We have discovered that consensus matrices built using multiple algorithms and multiple values for the number, $k$, of clusters will often allow users to estimate an appropriate number of clusters in data by determining the maximum number of clusters for which algorithms are likely to more or less agree on a common solution. We have provided several examples to show how this approach succeeds at determining the number of clusters in datasets where other methods fail. When the initial consensus matrix does not provide this information, it can be refined through the use of a drop tolerance or iteration to get a clearer picture of how many clusters the algorithms might be able to agree on.

9.1.2 The Consensus Matrix

While the consensus matrix itself is not a new idea, the practice of using multiple algorithms and dimension reductions to create the matrix had not previously been explored, nor had varying the number of clusters for the purposes of approximating $k$. Our approach to building the consensus matrix is novel and improves clustering results from nearly every clustering algorithm on all datasets considered. Our consensus matrix has several advantages over traditional similarity matrices for computation, storage, and performance.

First, the consensus matrix does not require pairwise distance calculation between data points. Not only are such pairwise calculations time consuming, the distances themselves tend to lose meaning in high-dimensional space given the curse of dimensionality discussed in Chapter 4. Second, the consensus matrix is expected to have a much higher level of sparsity since certain pairs of points ought never be clustered together. Additionally, because the clustering for the consensus matrix can be done in parallel, creating similarity matrices for massive datasets may be more feasible than previously thought.

We have thoroughly demonstrated in Chapter 8 that this matrix has the potential to provide better cluster information (both in terms of the number of clusters revealed by the eigenvalues and in terms of the average accuracy of cluster solutions it provides) than the popular Gaussian and cosine similarity matrices. In addition, the entries in the consensus matrix are far more intuitive to understand; they have depth in the sense that each entry in a consensus matrix is really the sum of entries in adjacency matrices from individual clusterings. Consequently, the researcher is afforded an additional avenue of investigation. For example, knowing that the cosine of the angle between two data vectors is 0.751 may or may not help
a user understand how or why they should cluster together. However, knowing a piece of information like \("x_i\) and \(x_j\) were clustered together by every algorithm for \(k = 2, 3, 4,\) and \(5\) by some algorithms with \(k = 6\) and by no algorithms with \(k \geq 7\)’ is far more informative.

### 9.1.3 Iterated Consensus Clustering

We have developed an ensemble clustering methodology that encourages clustering algorithms to agree on a common solution to help escape the unreliability of individual algorithms. While previous consensus methods have aimed to average cluster solutions in one way or another, ours is the first to emphasize agreement between clustering algorithms. After seeing some of the results of the individual algorithms in our ensemble, it should be clear that an average solution could be very poor indeed. Rather than deciding each clustering is equally valid, we simply sum the number of times a cluster relationship was made between two points and let the algorithms decide whether this sum is considerable enough to draw those points together, or whether it might be more reasonable to dissolve the connection in favor of others. Our framework iteratively encourages algorithms to agree upon a common solution because the value of the similarity metric reflects the level of algorithmic agreement at each step. Thus, through iteration cluster relationships upon which the algorithms do not agree are abandoned in favor of relationships with higher levels of agreement.

### 9.1.4 Dimension Reduction via NMF

In Chapter 4 we expanded current ideas of clustering by nonnegative matrix factorization (NMF) by removing the limitation of setting the rank, \(r\), of the dimension reduction equal to the number of clusters \(k\). By allowing \(r > k\), we give the NMF algorithms freedom to determine multiple features for each cluster, and then use the information in the coordinate matrix \(H\) to separate the data according to these new features. In many instances we have seen marked improvement in clustering results by using this method over the traditional NMF clustering from Algorithm 15. To the best of our knowledge, this approach had not previously been proposed.

### 9.1.5 The Agreement Metric

In Chapter 5 we provided a new metric of agreement between two clusterings. This metric is simple and easy to interpret as it is the proportion of objects that are clustered in the same way in both algorithms, i.e. the proportion of objects for which two clustering algorithms agree on their cluster assignment. The interpretability of this metric is of great benefit since existing measures of comparison are often unitless numerical values which are difficult to make sense of in context. In addition to the interpretation of metric itself, the computation of the metric may also provide valuable information. In this process, a user has the ability to determine the specific objects for which the algorithms disagree on cluster membership. These objects, whose removal from the dataset would make the clusterings agree entirely, may be of
interest in and of themselves. The classification of such points may be an interesting problem for further research.

9.2 Future Work

Many avenues of research have opened as a result of this work. The two most likely to be fruitful are the discovery of subcluster structure using sub-dominant eigenvalues and a randomized approach to consensus clustering using bootstrap samples. It has been noted on several occasions that the ICC algorithm has the ability to correctly identify sub-clusters present in the data by examining sub-dominant eigenvalues that fall between the Perron cluster and the remaining eigenvalues. Such eigenvalues, mathematically speaking, indicate that one of the aggregates of the Markov chain (i.e. one of the clusters in the data) is somewhat close to being uncoupled itself. In other words, for one cluster in our dataset, there seems to be some agreement between algorithms on how to further divide that cluster into two. Whether or not one might be able to “divide and conquer” in these scenarios in order to identify and interpret this sub-cluster structure is a question we intend to pursue.

To further enhance the scalability of our approach, we also hope to pursue a randomized approach. Our intuition is that certain forms of bootstrap sampling may be appropriate in forming a randomized consensus matrix. Rather than clustering the entire dataset together, one may be able to sample the data in a methodical fashion to get a fast and accurate approximation of the underlying cluster structure. Little experimentation has been done to verify this claim for consensus matrices, but analogous methods have been sought for other types of similarity matrices.

The results from the dataset ‘benchmarkGHI’ show how a consensus solution can actually exceed the accuracy of the clusterings from which the initial consensus matrix is formed. Such a result is most likely to occur when different low-dimensional configurations of the data are used, as each configuration may do well at separating different clusters. This motivates further investigation into other types of projections, particularly random projections, where the user might find such a scenario. As discussed in Chapter 4, other researchers have had some success conducting cluster analysis in random lower dimensional configurations. Using one such configuration is almost certainly inadequate, as it would be naive to expect that all of the clusters would become separated in one random projection. However it is not inconceivable to believe that each cluster might be visible in one such projection, and as such our consensus approach may be able to capitalize on many random projections.

As mentioned in our discussion of the agreement metric, a final issue that warrants further investigation are the “objects of disagreement” - the objects for which the clustering algorithms tend to conflict in their cluster assignment. Such points, rather than reflecting random error in the clustering models, may in fact be noteworthy observations. Are they outliers? Do they represent border points between clusters? Are they particularly interesting points in networks? Such questions beg investigation in the future.
REFERENCES


[82] Franklina Maria Bragion de Toledo Maria Cristina Vasconcelos Nascimento and Andre C. Ponce Leon Ferreira Carvalho. Consensus clustering using spectral theory.


138


