2019

Extending K-means

Nicholas S. Berry

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Statistics and Probability Commons

Recommended Citation

Extending K-means

by

Nicholas S. Berry

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Program of Study Committee:
Ranjan Maitra, Major Professor
Alicia Carriquiry
William Meeker
Daniel Nettleton
Vivekananda Roy

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2019

Copyright © Nicholas S. Berry, 2019. All rights reserved.
DEDICATION

To my wonderful family - Mom and Dad, Lindsay, Ben, and Cooper.


**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ix</td>
</tr>
<tr>
<td>CHAPTER 1. GENERAL INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2. VARIABLE SELECTION IN K-MEANS CLUSTERING</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Methodology</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Notational Preliminaries</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 A general framework for $K$-means clustering in the presence of redundant and noisy variables</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Examples</td>
<td>21</td>
</tr>
<tr>
<td>2.3.1 Ruspini Dataset</td>
<td>21</td>
</tr>
<tr>
<td>2.3.2 100 km Race Runners</td>
<td>22</td>
</tr>
<tr>
<td>2.3.3 Simulated Datasets</td>
<td>23</td>
</tr>
<tr>
<td>2.3.4 Written Digits</td>
<td>29</td>
</tr>
<tr>
<td>2.4 Conclusion</td>
<td>38</td>
</tr>
<tr>
<td>References</td>
<td>39</td>
</tr>
<tr>
<td>CHAPTER 3. TiK-MEANS: A TRANSFORMATION-INFUSED K-MEANS ALGORITHM FOR SKEWED GROUPS</td>
<td>44</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>44</td>
</tr>
<tr>
<td>3.1.1 Finding Types of Gamma Ray Bursts</td>
<td>46</td>
</tr>
<tr>
<td>3.2 Methodology</td>
<td>47</td>
</tr>
<tr>
<td>3.2.1 Preliminaries</td>
<td>47</td>
</tr>
<tr>
<td>3.2.2 The TiK-means algorithm</td>
<td>49</td>
</tr>
<tr>
<td>3.2.3 Additional Issues</td>
<td>52</td>
</tr>
<tr>
<td>3.3 Performance Evaluations</td>
<td>54</td>
</tr>
<tr>
<td>3.3.1 Illustrative Example</td>
<td>55</td>
</tr>
</tbody>
</table>
CHAPTER 4. STATISTICAL ANALYSIS OF HANDWRITTEN GLYPHS FOR WRITER IDENTIFICATION

4.1 Introduction
4.2 Document Processing
  4.2.1 Pre-processing
  4.2.2 Data Extraction
4.3 Deterministic Groupings
4.4 Clustering
  4.4.1 Graph Distances and Prototypes
  4.4.2 Distance Measure for Glyphs
  4.4.3 Weighted Mean of Glyphs
  4.4.4 K-means Type Algorithm
4.5 Applications
  4.5.1 MNIST Digits
  4.5.2 Handwriting Clustering
4.6 Discussion
4.7 Supplementary Materials
  4.7.1 Path Finding for Glyph Edge Identification
  4.7.2 Writer Identification for Other K
References

CHAPTER 5. GENERAL CONCLUSION
References
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>$K$-selection comparison for simulation study</td>
<td>30</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Dimension selection and clustering accuracy comparison for simulation study</td>
<td>31</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Confusion matrix for handwritten digit clustering</td>
<td>33</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Performance, in terms of ARI of $K$-means, MBC and TiK-means on datasets for known and estimated $K$</td>
<td>57</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Confusion matrix of the true (rows), and the groups (columns) estimated by TiK-means for (a) $\hat{K} = 8$ and (b) $\hat{K} = 13$ for the scaled and unscaled datasets. The column names denote the author-specified digits that can be used to best characterize the TiK-means solutions.</td>
<td>61</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>(a) TiK-means results for the scaled and unscaled wines dataset (results are identical) with a $p$-dimensional $\lambda$. TiK-means results with the $k \times p$-dimensional $\lambda$ for the (b) scaled and (c) unscaled datasets.</td>
<td>68</td>
</tr>
<tr>
<td>Table 3.4</td>
<td>Confusion matrix for Iris dataset when clustered using TiK-means with (a) $p$-dimensional $\lambda$ and (b) $k \times p$-dimensional $\lambda$.</td>
<td>70</td>
</tr>
<tr>
<td>Table 3.5</td>
<td>Confusion matrix for the Olive Oils dataset when clustered using TiK-means with $K = 3$ and (a) $p$-dimensional $\lambda$ (b) $k \times p$-dimensional $\lambda$, for the scaled dataset. Corresponding results for the scaled dataset are in (c) and (d). (e) Results obtained on the scaled (and unscaled) dataset for $\hat{K} = 5$ groups.</td>
<td>71</td>
</tr>
<tr>
<td>Table 3.6</td>
<td>Confusion matrix for the Olive Oils dataset when clustered using TiK-means with $K = 9$ and (a) $o$-dimensional $\lambda$ and (b) $k \times p$-dimensional $\lambda$, for the unscaled and (c, d) scaled datasets.</td>
<td>72</td>
</tr>
<tr>
<td>Table 3.7</td>
<td>Confusion matrices for Seeds dataset when clustered with TiK-means at (a-c) true and (d) estimated $K$.</td>
<td>74</td>
</tr>
<tr>
<td>Table 3.8</td>
<td>Confusion matrices for the SDSS dataset when clustered with TiK-means with true $K=2$ and estimated $\hat{K}$.</td>
<td>75</td>
</tr>
</tbody>
</table>
Table 3.8  Confusion matrix for the (a) scaled and (b) unscaled datasets showing differences between the true clusters, indicated by rows, and the TiK-means estimated clusters for $K = 10$, indicated by rows. The column names denote the author specified digits that the TiK-means clusters seem to cover.
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Scatter plots demonstrating redundant and irrelevant dimensions.</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Ruspini dataset</td>
<td>21</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>100km race runners dataset</td>
<td>22</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Results for $K$-selection in the simulation study.</td>
<td>25</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Box plots of ARI of clustering after variable selection</td>
<td>27</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Q-map quantitation plot for handwritten digits</td>
<td>32</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Grid plot of estimated versus true clusters for handwritten digits</td>
<td>35</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Simulated dataset for illustrating TiK-means.</td>
<td>56</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>The Jump selection plot for the wine dataset</td>
<td>57</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>The jump selection plot for the scaled and unscaled pen digits datasets.</td>
<td>60</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>The jump selection plot for the GRB dataset that shows support for 3 groups but more support for $\hat{K} = 5$.</td>
<td>62</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>Three-dimensional radial visualization plots of the gamma ray burst clusters.</td>
<td>63</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>TiK-means clustering of the GRB dataset. Summary of duration ($T_{90}$), total fluence $F_{total}$, and spectral hardness $H_{321}$ for each of the five groups obtained using TiK-means.</td>
<td>64</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Iris Jump Statistic Calculation</td>
<td>69</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Pixel Neighborhood</td>
<td>89</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Partitioning of the cursive word ‘London’</td>
<td>91</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>The letter ‘x’ showing vertices and edge labels from handwriter</td>
<td>92</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Glyphs and their adjacency matrices for two common deterministic groups</td>
<td>96</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Distance measure comparison glyphs</td>
<td>99</td>
</tr>
</tbody>
</table>
Figure 4.6  Edge distance measure calculation companion figures. ............. 109
Figure 4.7  Weighted mean transition examples for 4 pairs of glyphs ........... 110
Figure 4.8  Examples of typical outlier glyphs .................................. 112
Figure 4.9  Cluster exemplars for MNIST dataset ................................. 117
Figure 4.10 Grid plot of how MNIST digit types are grouped into estimated clusters 118
Figure 4.11 Glyph shapes for the four most common deterministic groupings .... 121
Figure 4.12 Cluster exemplars for CVL handwriting data .......................... 125
Figure 4.13 Cluster raw means for CVL handwriting data .......................... 126
Figure 4.14 Subset of the outlier cluster for CVL handwriting data ............... 128
Figure 4.15 Densities of posterior samples for multinomial parameter vectors ... 130
Figure 4.16 Writer identification results for estimated clusters ..................... 134
Figure 4.17 Writer identification results for deterministic adjacency groupings 135
Figure 4.18 Writer identification results for deterministic connectivity groupings 136
Figure 4.19 Writer identification results for estimated clusters with $K = 30$ ... 140
Figure 4.20 Writer identification results for estimated clusters with $K = 60$ ... 141
ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Ranjan Maitra. His support throughout my graduate school tenure made the completion of this work possible. His interest in my life outside of school, encouragement in the face of adversity, and patience with regards to my shortcomings made our partnership eminently enjoyable. Additionally, I’d like to thank my committee: Dr. Alicia Carriquiry, Dr. William Meeker, Dr. Dan Nettleton, and Dr. Vivekananda Roy. I’d like to express gratitude to Dr. Carriquiry for her support in my research with CSAFE, and her unwavering personal support no matter the circumstance. I appreciate Dr. Nettleton’s and Dr. Roy’s dedicated instruction during my course work at Iowa State, as well as serving on my master’s committee.

Finally, I would like to thank my friends here at Iowa State. They all made the graduate school experience great, from our first classes together to the end. A special thanks to Amy Crawford for her constant support during the writing of this dissertation... During my most stressful times, she was always there.

There are certainly people that I did not mention here that I am indebted to. Each of those people has helped to make this entire process more pleasant than any process so long and grueling should be.
CHAPTER 1. GENERAL INTRODUCTION

Clustering is an unsupervised learning technique that separates observations into groups based on similarities among those observations. It is called unsupervised because there are no class labels available to inform an algorithm about what groups should look like. Instead, a clustering algorithm uses the data to detect the dominating structure of a dataset that it will identify. This latent structure is often difficult to find in the clustering setting, and the variety of potential shapes in the underlying structure make a universally good approach for all clustering tasks next to impossible. This component of difficulty in clustering results in a vast array of clustering techniques that each excel at detecting a certain type of structure.

Hierarchical clustering (Ward, 1963; Johnson, 1967) successively merges or splits a dataset based on distances between points. Distribution based clustering algorithms (Titterington et al., 1985; Melnykov and Maitra, 2010) use Expectation Maximization (Dempster et al., 1977) or Markov Chain Monte Carlo to estimate parametric distributions for each cluster. Density based methods (Ester et al., 1996; Ankerst et al., 1999) search for and group areas of high observational density. Centroid based algorithms, like $K$-means (Forgy, 1965; Lloyd, 1982; MacQueen, 1967; Hartigan and Wong, 1979), group observations into clusters designated by their centers. Those centers can be means, medians, or modes, which result in traditional $K$-means, $K$-medoids (Kaufman and Rousseeuw, 1987), or $K$-modes (Carreira-Perpiñán and Wang, 2013), respectively. This dissertation focuses on $K$-means, which will now be expanded on.

The earliest, and most simple version of $K$-means is the Forgy-Lloyd algorithm. This algorithm starts with a set of initial cluster centers and iteratively modifies the provided
cluster centers until the algorithm converges to an optimal clustering. The iterations happen in two stages. First, each observation is assigned to the cluster whose mean is closest to it. Second, the cluster means are updated to be the mean of the set of observations that were assigned to the cluster in the first step. This algorithm continues until no observations change cluster assignments for an iteration. The Hartigan-Wong algorithm is a commonly used alternative to the Forgy-Lloyd algorithm that implements heuristic checks within the algorithm to improve the compactness of the resulting clusters. As any implementation of $K$-means iterates to convergence it minimizes the within cluster sum of squared distances (WSS), a measure of the compactness of the clusters.

$K$-means requires that the number of clusters are provided by the user. Often, $K$ is not known a priori, making this a potential difficulty. There are many existing methodologies to estimating the optimal number of clusters (Sugar and James, 2003; Tibshirani et al., 2003; Wang, 2017) that work with varying degrees of success. Each of these methods attempts to determine whether the improvement in the compactness of $K$-means clusters is worth the additional complexity of the additional cluster.

$K$-means is only guaranteed to converge to a local optimum. Since a global minimum is preferred, running $K$-means for a variety of starting points and choosing the initializations that produce the minimum WSS is common. Two common initialization strategies are to choose random observations to be starting cluster means and to randomly assign points to clusters and use the means of the random clusters as the initial means. Another common initialization strategy is $K$-means++ (Arthur and Vassilvitskii, 2007), which seeks to choose observations as initial means that are as far from each other as possible.

Since $K$-means makes no explicit distributional assumptions it is commonly seen as an assumption free algorithm, especially by users who may be new to clustering. This is true in the sense that $K$-means will converge to the locally best solution without any conditions. However, there are implicit assumptions that reveal themselves in the groupings produced
by $K$-means. Due to the Euclidian distance measure used, clusters in $K$-means tends to produce homogeneous, spherical clusters. Also, large differences in cluster membership may result in centers biased towards the large cluster. So, the implicit assumptions of $K$-means are that the clusters are homogeneously and spherically distributed and are roughly equal in size, and deviations from these assumptions may result in nonintuitive clustering results. Common violations of these assumptions are in datasets with correlated features, datasets with outliers, and datasets with dimensions of very different scales.

Clustering is often used as a staring point for data analysis, when very little is known about the characteristics of a dataset. In order to be useful in these situations, it is important that the clustering algorithm be robust with regards to the inputs it can process coherently. This dissertation contributes adaptations of the $K$-means algorithm that allow it to succeed in situations when it otherwise would not.

Chapter 1 implements a simultaneous variable selection and $K$ selection process. This procedure solves the perpetual clustering problem of selecting $K$, and solves the additional problem of removing noise dimensions that can degrade clustering. Our algorithm is compared to existing methods and applied to a dataset of handwritten digits. Chapter 2 provides a modified $K$-means algorithm called TiK-means that estimates a normalizing transformation for the data within the $K$-means framework. This algorithm allows for the estimation of non-spherical clusters, and accommodates skewed data and data with dimensions on different scales. TiK-means also provides a strategy for estimating $K$. TiK-means is applied to a gamma ray burst dataset with skewed dimensions, providing insight to a discussion with far reaching astronomical implications. Finally, Chapter 3 provides a $K$-means type algorithm for the clustering of handwriting samples. A distance measure and mean calculation are introduced to make the clustering possible. Writer identification is performed on handwritten documents by modeling a writer based on their propensity for emitting characters belonging
to certain clusters. Results for writer identification are provided for samples from a database of handwritten documents.
CHAPTER 2. VARIABLE SELECTION IN K-MEANS CLUSTERING

A paper to be submitted to the Journal of Computational and Graphical Statistics

Nicholas S. Berry and Ranjan Maitra

Abstract Clustering is a difficult problem that is further challenged in higher dimensions where some of the information can be redundant. Such redundancy can be in the form of dimensions that have group information that is already present in other variables, or are simply irrelevant and contribute no useful information with regard to clustering. The K-means algorithm is arguably the most widely used clustering tool, but its performance is degraded by the presence of redundant dimensions. We provide a formal approach to identifying and removing these redundant features and demonstrate improved performance, as well as interpretability of the derived groupings. Our methodology also simultaneously estimates the number of groups while selecting dimensions informative for clustering. We evaluate performance on datasets simulated under many complexities and conditions as well as on a set of handwritten digits, and is used to identify different running styles among participants in a 100 km ultra-marathon race.

2.1 Introduction

Cluster analysis (Murtagh, 1985; Ramey, 1985; McLachlan and Basford, 1988; Kaufman and Rousseuw, 1990; Everitt et al., 2001; Fraley and Raftery, 2002; Tibshirani and Walther,
2005; Kettenring, 2006; Xu and Wunsch, 2009) is a common practice in data mining and statistical learning whereby observations are partitioned into groups (or classes) containing data points with similar properties. These similar properties can be measured by anything from distances between points to parametric likelihoods specified for clusters. Clustering is generally used in unsupervised situations, meaning that before a dataset is clustered there are no previously-known groupings for any of those data points. An exception is when some of the observation labels are known, as in the case of semi-supervised clustering (Demiriz et al., 1999; Wagstaff et al., 2001; Basu et al., 2002, 2004; Wang et al., 2011). This is useful in a host of applications such as taxonomical classification Michener and Sokal (1957), market segmentation (Hinneburg and Keim, 1999) and software management (Maitra, 2001) among many others. It is also useful in situations where it is of interest to obtain some preliminary understanding of the underlying structure of a dataset.

There are many different clustering algorithms, each making assumptions about the dataset and its underlying cluster structure. These clustering methods range from the formal model-based approaches that require parametric distributions to be specified for each cluster (Titterington et al., 1985; McLachlan and Peel, 2000; Fraley and Raftery, 2002; Melnykov and Maitra, 2010) to the more heuristic, distance-based methods (Johnson, 1967; Everitt et al., 2001; Jain and Dubes, 1988; Forgy, 1965; MacQueen, 1967; Kaufman and Rousseuw, 1990) that make no such assumption (however, see Maitra et al. (2012) for attempts to cast some of these methods within a nonparametric distributional framework). Additional approaches (Heyer et al., 1999; Tseng and Wong, 2005; Maitra and Ramler, 2009; Dib and Carbone, 2012) also make allowances for the presence of scatter points while clustering.

The $K$-means algorithm (MacQueen, 1967; Lloyd, 1982) is perhaps the most popular partitioning method in use owing to its simplicity and computational speed. For a given data matrix $X = \{X_1, X_2, \ldots, X_n\}'$ (where $X_1, X_2, \ldots, X_n$ are the $n$ $p$-variate observation vectors) and number of clusters $K$, the $K$-means algorithm partitions the rows of $X$ into
$K$ groups ($C_1, C_2, \ldots, C_K$), with each observation $X_i$ assigned to the cluster $C_k$ whose mean $\mu_k$ is closest in Euclidean distance. The clusters may each be indexed by the corresponding centers (or means): that is, $C_1, C_2, \ldots, C_K$ may be characterized by $\mu_1, \mu_2, \ldots, \mu_K$ respectively. Mathematically, the $K$-means algorithm chooses $C_1, C_2, \ldots, C_K$ and $\mu_1, \mu_2, \ldots, \mu_K$ in order to minimize

$$W_z = \sum_{i=1}^{n} \sum_{k=1}^{K} \varphi_{ik} (X_i - \mu_k)'(X_i - \mu_k)$$

(2.1)

where $\varphi_{ik} = I(X_i \in C_k)$. $W_z$ is called the within cluster sum of squares (WSS) after partition. The WSS is a quantification (or goodness-of-fit measure) of how compact the clusters are, with smaller values corresponding to more compact clusters. The standard $K$-means algorithm (MacQueen, 1967; Lloyd, 1982) starts with initial seeds $\mu_1, \mu_2, \ldots, \mu_K$, and iteratively and alternately assigns each observation $X_i$ to the closest $\mu_k$ and then updates each $\mu_k$ from all observations obtained from the partition formed by the assignment of the $X_i$s to each $C_k$. The algorithm terminates upon convergence. Additional reductions to the $K$-means algorithm obtained by eliminating redundant calculations were provided by Hartigan and Wong (1979) whose algorithm is the one in most common use in statistics and packages like R (R Core Team, 2018).

Finding a global minimizing solution for (2.1) is an NP-hard problem because (2.1) can have multiple local optima for $K > 1$. Hitting the global minimum can depend on the choice of initializing seeds $\mu_1, \mu_2, \ldots, \mu_K$. The importance of good initialization has been recognized (Steinley, 2003; Maitra, 2009), but there is no obvious best choice and many initialization strategies (for example, Hartigan and Wong (1979); MacQueen (1967); Faber (1994); Maitra (2009); Steinley and Brusco (2007)) have been proposed. One of the most widely used methods randomly selects $K$ observations to serve as the initial centers of each cluster (Faber, 1994). The selection is repeated several times, and the best final clustering solution optimizing (2.1) reported. There are many variations on this simple idea,
including methods that refine the random centers Bradley and Fayyad (1998) or that insure
that they are well-separated (Arthur and Vassilvitskii, 2007). There are also deterministic
methods that select a single initialization given a configuration of observations (Astrahan,

The $K$-means algorithm assumes a given number of groups $K$. This is a potential dif-
ficulty in an unsupervised situation, since it is common for the number of clusters to be
unknown. Important also, and pertaining to the central issue addressed in this paper, is
that the $K$-means algorithm naturally uses every dimension available to it, disregarding the
fact that in many cases, some dimensions may offer redundant or irrelevant information that
do not significantly improve (and indeed, in many cases, substantially degrade) the quality
of the clustering. This problem is exacerbated with larger number of dimensions but is well-
illustrated via a small three-dimensional sample dataset (Figure 2.1) with both redundant
and irrelevant dimensions. We turn to this example in a bit, but first fix some definitions
for the sake of clarity. Specifically, we define a redundant dimension to be one that offers
information that could be informative, but that is so similar to another dimension that it is
not worth including in the clustering algorithm that also includes the previous dimension.
On the other hand, an irrelevant dimension offers no information about clustering and com-
plicates the search for a good partitioning. An example of such a dimension is one that has
small signal and large noise. We are now ready to discuss Figure 2.1.

Figure 2.1 shows that choosing only one of the first two dimensions is enough to obtain
a good clustering solution. It should not, however, present a major problem in this case
if both these dimensions were used to cluster the dataset, but the added complexity is
not needed. (This example has very little variability in these two dimensions relative to
the clustering information so including both variables while unnecessary does not degrade
clustering performance, however, this may not in general hold.) If the third dimension
(which contains noise and is irrelevant for finding the groups in the data) is included in
Raftery and Dean, 2006; Maugis et al., 2009; C et al., 2009) has been in the context of redundant and irrelevant information while clustering. Some of it (Dy and Brodley, 2004; Raftery and Dean, 2006; Maugis et al., 2009; C et al., 2009) has been in the context of model-based clustering where an appropriate number of clusters and features are selected clustering, however, the quality of the resulting grouping is degraded as seen in the figure which shows the results obtained using all three dimensions via different fill-types. Clearly, the grouping is worse than that obtained using either of the first two dimensions (with resulting grouping results displayed via color). We see, therefore, that the noise variable introduces information that has a negative impact on the clustering, so it would be beneficial to cluster without including that variable. Sifting out and accounting for the redundant and irrelevant dimensions while clustering would be important in these situations and is the focus of this paper.

There has been some attention paid in the literature to the issue of accounting for redundant and irrelevant information while clustering. Some of it (Dy and Brodley, 2004; Raftery and Dean, 2006; Maugis et al., 2009; C et al., 2009) has been in the context of model-based clustering where an appropriate number of clusters and features are selected.

Figure 2.1: Pairwise scatter plots of a 2-class simulated dataset. Dimensions 1 and 2 are redundant as they provide the same class information. Dimension 3 is irrelevant as it offers no clustering information but contributes only to noise. Color and shape of an observation indicate its true generating cluster, while its filled/unfilled status indicates how it was clustered by 2-means clustering.
simultaneously using penalized likelihood methods such as the Scatter Separability Criterion and Maximum Likelihood Criterion (Dy and Brodley, 2004), or Bayes Information Criterion (BIC) or Bayes Factors (Raftery and Dean, 2006) are used to evaluate the “interestingness” (and inclusion) of dimensions. Witten and Tibshirani (2010) addressed the issue of feature selection for the K-means algorithm using an adaptation of K-means that assigned weights to each coordinate based on its importance to clustering. The weights are controlled via a tuning parameter that is selected by bootstrapping. Further, the number of groups K is assumed known.

In this paper, we revisit variable selection in K-means clustering in order to improve its performance in the presence of redundant and irrelevant information while also assuming an unknown K. Our methodology proceeds by identifying whether a more complicated (parameter-rich) model as identified by the total number of estimable parameters (K × p) provides for a significantly better fit (lower goodness-of-fit measure) than a simpler, less complicated model. We develop and extend the methodology of Maitra et al. (2012) for this purpose in Section 2.2. We illustrate and evaluate our suggested approach on an illustrative example and in a large-scale simulation study in Section 2.3. The paper concludes with some discussion.

2.2 Methodology

2.2.1 Notational Preliminaries

As before, let the dataset be given by \( X = \{X_1, X_2, \ldots, X_n\}' \) where each \( X_i \) is a \( p \)-dimensional vector. Each \( X_i \) is comprised of elements \( X_{ij} \) for \( i = 1, 2, \ldots, n \) and \( j = 1, 2, \ldots, p \), which represents the value of the \( j \)th feature for the \( i \)th observation. The \( z \)th dimension of the dataset is the set \( X_{\cdot z} \) containing the \( z \)th coordinate values \( X_{iz} \) of the \( i \)th observation for all \( i = 1...n \). When clustering a dataset \( X \), denote \( K \) as the the number of
clusters. Whenever a subset of the dimensions in $X$ is needed, denote $Z$ as the dimensions in the subset and $X^{\{Z\}}$ as the resulting data matrix. When adding a new dimension $z$ to an existing subset, denote the new data matrix as $X^{\{Z \cup z\}}$ and the new set of dimensions as $Z^{+z} = \{Z, z\}$.

### 2.2.2 A general framework for $K$-means clustering in the presence of redundant and noisy variables

Our proposed framework uses a sequential forward testing approach to decide on both the number of groups and also the variables that are useful for clustering with the $K$-means algorithm. Specifically, the three main steps are as follows:

1. For each $K = 2, \ldots, K_{\text{max}}$, choose the subset of dimensions that should be included for clustering based on that fixed value of $K$.

2. For each of the $\binom{K_{\text{max}}}{2}-1$ possible combinations of pairs of $K$ values that can be chosen, conduct a significance test to decide if the more complex clustering is decidedly better than the simpler clustering.

3. Convert the obtained $p$-values for each pairwise test to $q$-values to control the proportion of false discoveries (Benjamini and Hochberg, 1995). Visualize results using the $q$-value quantitation map (Maitra et al., 2012) and choose the optimal number of dimensions and that number’s corresponding dimension set.

We will develop and explain each of the steps shortly, but we first provide a short discussion on the goodness-of-fit measure used to both select dimensions and choose the number of clusters.

Suppose that for any given and fixed $K$, we have a subset of dimensions, $Z$, and that same subset with 1 extra dimension added, $Z^{+}$. Then we cluster those two sets of dimensions
with $K$-means, giving two sets of inclusion matrices $\varphi_{ik}^Z$ and $\varphi_{ik}^{Z^+}$ each being $n \times K$ and whose elements are 1 if the $i$th observation falls into the $k$th cluster and 0 otherwise. We can use these inclusion matrices to impose the clustering from a subset of the data onto the entire dataset. For example, the within cluster sum of squares for $X$ based on the clustering of $X^{(Z)}$ is

$$W_Z = \sum_{i=1}^{n} \sum_{k=1}^{K} \varphi_{ik}^Z (X_i - \mu_k)' (X_i - \mu_k)$$

$W_Z$ represents a measure of the compactness of clusters of an entire dataset based on only a subset of its dimensions, essentially elucidating a measure of the quality of clustering of a dataset based on a subset. Despite the general indifference of this paper’s methodology to the goodness-of-fit measure used, for consistency and its general effectiveness, we use the sum of the log determinant of the estimated covariance matrices corresponding to the cluster estimated for each observation, or:

$$D_z = \sum_{i=1}^{n} \sum_{k=1}^{K} \varphi_{ik}^Z \log(\left| \hat{\Sigma}_k \right|)$$

(2.2)

where $\hat{\Sigma}_k$ is the covariance matrix estimated on the entirety of $X$ for the $k$th cluster. Like the Within Sum of Squares, a smaller value of $T_z$ indicates a smaller spread of points around the means, and thus a better clustering.

Thus, our goodness-of-fit measure to determine the clustering discrepancy between two models is then $T_{Z,Z^+} = D_Z - D_{Z^+}$, which represents the improvement in the test statistic for the complex model over the simpler one.

For the forward selection algorithms that follow, we use this measure as our test statistic and resample to determine whether the increase in the test statistic significantly asserts that the considered model complexity is worth incorporating.
2.2.2.1 Choosing the dimension subset for each $K$

For a known $K \in \{2, \ldots, K_{\max}\}$ there are $2^p$ different subsets of $\mathbf{X}$ that could be chosen as dimensions considered appropriate for clustering. Unfortunately, in all but the smallest of data sets comparing this many subset combinations is intractable, especially when considering comparing across different values of $K$. Instead, we fix $K$, and implement a forward selection procedure to iterate through the dimensions of the dataset. This narrows the number of comparisons down to $p$ for each value of $K$.

The fixed $K$ forward selection procedure follows in two general steps. First, sequentially look at each dimension in the dataset that has not already been added or tested and propose for inclusion the dimension with the largest improvement in the test statistic $T_{Z,Z+}$ on $\mathbf{X}$. Then, using $T_{Z,Z+}$, perform a bootstrap hypothesis test to determine whether the new dimension significantly improves the test statistic of the dataset over the subset without that new dimension. If a dimension is significantly beneficial, it is added to the subset for future proposals. If not, it is thrown out and the next dimension is proposed against the same dimension subset. After these steps are repeated and each dimension has been proposed and tested, the resulting dimension subset is fixed as the subset of dimensions for its number of clusters. These steps are presented in more detail below, first dimension selection and then the bootstrap procedure.

Selecting a Dimension: Before starting our forward selection procedure for choosing dimensions we make two adjustments to our general algorithm outline to attempt to limit some of the well documented errors that can happen early in all forward selection procedures. As dimensions are correctly incorporated into the selection procedure it gets better and is at less risk for making bad decisions, but when very few dimensions are included in the dataset the procedure is vulnerable to making mistake inclusions that cannot be undone. As a first measure, we sort our dimensions into obvious noise and non-noise groups. To
do this, we conduct an Anderson Darling test for Uniformity on each dimension with a significance level of $0.05/p$. The dimensions that reject uniformity are put into one group and the other dimensions are put into a noise block. While we will still propose every dimension eventually, we guarantee that every non-noise dimension will be tested before every noise blocked dimension.

In addition to the dimension blocking we will make one more change when adding the first dimension to our dataset. Due to the nature of our forward selection procedure when there are no dimensions in the dataset we cannot conduct a test for significant improvement. Because of this, the first proposed dimension will be automatically included in the approved subset, and should be chosen with care. Additionally problematic, is that the first dimension has the least information about the rest of the dataset to work with, so is the most vulnerable to forward selection pitfalls. To make the first dimension selection better, we take every pair of dimensions in the non-noise block and calculate the test statistic for the entire dataset based on the clustering of those two dimensions. Then, we choose the single best dimension from among the pair that provides the best improvement in clustering, denoted by the largest test statistic. Clustering based on pairs of dimensions offers significantly more information to make the initial decision than looking at each dimension individually, and does so without costing too much in the way of computation. After the first dimension is included from the winning pair, the forward selection procedure takes over for the remaining dimensions. In the fringe case where there is only one non-noise dimension, that dimension is included as the first and forward selection starts on the non-noise dimensions. If there are 0 non-noise dimensions, we begin with forward selection, and the first dimension chosen for testing is automatically added to the subset.

When choosing dimensions for forward selection it is guaranteed that there will be at least 1 dimension in the approved subset. From this point on to choose a dimension we iterate through every untested dimension $i$ and calculate $T_{Z, Z_i^+} = D_Z - D_{Z_i^+}$, where $Z$ is
the current subset and $\mathcal{Z}_i^+$ is the current subset augmented with dimension $i$. Whichever dimension $i$ produces the largest $T_{\mathcal{Z},\mathcal{Z}_i^+}$ is the proposed dimension for the next round of testing. Since $T_{\mathcal{Z},\mathcal{Z}_i^+}$ is the improvement in the test statistic based on including dimension $i$, this proposed dimension is the dimension with the largest potential to improve the clustering of the existing subset.

In addition to being used to choose dimensions, $T_{\mathcal{Z},\mathcal{Z}_i^+}$ is used as the test statistic for the subsequent test of significance. Since a large value of the test statistic represents a large increase in the quality of the clustering, if the observed $T_{\mathcal{Z},\mathcal{Z}_i^+}$ is larger than a large enough proportion of the resampled test statistics we will conclude that the new dimension should be added to the inclusion subset. In this case all later dimension proposals and significance tests will include this dimension in the included subset. In the case when $T_{\mathcal{Z},\mathcal{Z}_i^+}$ is not larger than a high enough proportion of our resampled test statistics we throw out the proposed dimension and start over with dimension selection with the same base subset of dimensions to compare against.

Analyzing Significance: When testing the significance of a proposed dimension, we know the improvement in the test statistic of the entire dataset based on the addition of the dimension, but must determine the significance of this improvement.

Again, let $\mathcal{Z}$ represent the currently included subset of dimensions and $\mathcal{Z}_i^+$ indicate that same set of dimensions augmented with the newly proposed dimension based on the criteria above. As we test for the significance of the improvements made based on this dimension we work under the assumption that it does not significantly improve the clustering of $\mathbf{X}$, and thus should not be added to the subset $\mathcal{Z}$. If our test statistic $T_{\mathcal{Z},\mathcal{Z}_i^+}$ is large enough we reject this assumption and favor the belief that the proposed dimension does belong in $\mathcal{Z}$.
Based on these hypotheses, we provide a bootstrap methodology and calculate a p-value corresponding to $T_{Z,Z^+}$. The resampling scheme for testing the $i^{th}$ dimension is similar to the method used by Maitra et al. (2012).

As in every bootstrapping scheme, the bootstrapped datasets are made under the assumption that the null hypothesis is the truth. In our case this means that we create new datasets such that the newly proposed dimension is not important for clustering.

To do this, start by running K-means on only the dimensions in $Z$ and use this grouping to calculate the mean for observations in $X$. We then subtract out these means from $X$, scale the residuals by the estimated cluster specific covariances, and then calculate the Euclidean norms of the residual vectors of those scaled values. Then, randomly assign new directions to each normed residual to get a new residual vector, re-scale the vectors by the covariance matrices, and add the previous means back into the newly permuted residuals.

Mathematically, the process is as follows. First, using the cluster assignments $\hat{\varphi}^Z$ and means $\hat{\mu}_1,\ldots,\hat{\mu}_k$ estimated from the base subset, obtain $\hat{\epsilon}$ where

$$\hat{\epsilon}_i = \sum_{j=1}^k \hat{\varphi}_{ij} (X_i - \hat{\mu}_j).$$

Then the scaled residual matrix is $\Xi$ where $\Xi_i = \sum_{j=1}^k \hat{\varphi}_{ij} \left( \left( \sum_{j=1}^k \hat{\epsilon}_j \right)^{-\frac{1}{2}} \hat{\epsilon}_i \right)$ and $\hat{\Sigma}_k = \sum_{i=1}^n \hat{\varphi}_{ik} (X_i - \hat{\mu}) (X_i - \hat{\mu})' / \sum_{i=1}^n \hat{\varphi}_{ik}^2$.

After this scaling, $\Xi$ will have spherical clusters and the norms of each $\Xi_i$ for $i = 1,\ldots,n$ are permuted around the origin forming $\Xi^*$ via $\Xi^*_i = \|\Xi_i\| \frac{Y_i}{\|Y_i\|}$ where each $Y_i$ is a $p$-dimensional vector of independent standard normals. Each $\frac{Y_i}{\|Y_i\|}$ is a randomly directed unit vector, which when multiplied by a normed residual gives a vector with the same magnitude as the residual, but in a different direction.

Finally, the complete resampled dataset $X^*$ is made up of $X^*_i = \sum_{j=1}^k \hat{\varphi}_{ij} \left( \hat{\mu}_j + \hat{\Sigma}_k^{\frac{1}{2}} \Xi_i^* \right)$.

The resampling algorithm permutes the residuals around exactly K means, so no matter how many clusters $X$ appears to have, the resampled datasets will have K. Additionally, the dimensions included in $Z$ will contain information useful to clustering the resampled dataset,
but the other dimensions in the dataset are all using mean estimates based on a clustering
that didn’t involve any information about that dimension.

For each resampled dataset $X^*$, we calculate a test statistic identically to how $T_{Z,Z+i}$ was
calculated on $X$, by clustering $X^*$ on $Z$ and $Z^+$, calculating $D^*_Z, D^*_Z$, and then $T^*_Z,Z+i$.

This resampling and calculation of resampled test statistics process is repeated for enough
iterations to satisfy the desired accuracy of the resampled p-value. We then compare the
resampled test statistics $T^*_Z,Z+i$ to the original test statistic $T_{Z,Z+i}$, and our p-value is the
proportion of the resampled test statistics that are larger than $T_{Z,Z+i}$.

As always, the significance level of the test must be chosen beforehand. If the goal is to
choose to add as few dimensions as possible, then setting a small $\alpha$ is appropriate, but if
dimensions are cheap and we don’t mind including many dimensions, then setting a larger
$\alpha$ is acceptable. In the examples provided later $\alpha = 0.05$. For a p-value greater than $\alpha$,
sufficient evidence against $H_0$ is not provided and $Z$ remains the same for the next proposed
and tested dimension; however, if the p-value is less than $\alpha$ it suggests that the proposed
dimension is significantly beneficial and while proposing and testing the next dimension
our future $Z$ should be changed to the current $Z$ augmented with the dimension that just
underwent testing.

2.2.2.2 Choosing the Best $(K, Z_K)$ Pair

After the selection procedure is performed on $K = 2$ through $K = K_{max}$, we are left
with a set of $(K, Z_K)$ pairs, where each pair is number of clusters and the corresponding
best dimensions for that number of clusters. These pairs make up the set of candidates for
the final decision on the best way to cluster the target dataset. In order to choose which of
these $K_{max} - 1$ pairs is best, we again rely on the bootstrap to perform hypothesis tests to
decide between different pairs.
We start by ordering the pairs by complexity. We call a pair \((K^*, Z_{K^*})\) more complex than \((K, Z_K)\) if \(K^* \times |Z_{K^*}| > K \times |Z_K|\). When the pairs are ordered, we take each pair in the list and compare it to every pair more complex than it. The comparison procedure for pairs is similar to the procedure used to select dimensions for a fixed \(K\), with the distinct difference that \(K\) is now allowed to change.

We again perform a hypothesis test, using the bootstrap to assess the significance of the test. The null hypothesis is that the simpler pair is sufficient for clustering with the alternative being that the more complex pair is necessary for proper clustering of the dataset.

To test \((K, Z_K)\) vs. \((K^*, Z_{K^*})\), where \((K^*, Z_{K^*})\) is the more complex pair, start by computing \(T\{K, z_K\},\{K^*, z_{K^*}\} = D\{K, z_K\} - D\{K^*, z_{K^*}\}\), which is the improvement in the test statistic on \(X\) when moving from clustering the dimensions \(Z_K\) with \(K\) clusters to clustering \(Z_{K^*}\) with \(K^*\) clusters. As in 2.2.2.1, \(T\) is the test statistic that will be used in the resampling test.

After calculating the test statistic, we create bootstrap datasets and use them to get a p-value for our test statistic. In the same fashion as before, we create the new datasets by clustering based on the simpler pair, removing the means, scaling to spherical, permuting the residuals, re-scaling to ellipses, and adding the means back in again.

When calculating the resampled test statistic \(T^*\), the test statistic must be calculated on \(X^*\) for both the simple and complex models. For the simple model, since it was used to create the resampled dataset, \(Z_K\) is adequate for the set of dimensions best for clustering \(X^*\) with \(K\) clusters, however, this is not true for \(K^*\). Instead, \(Z_{K^*}^*\) must be re-chosen based on which dimensions in \(X^*\) are best for clustering with \(K^*\) clusters. Note that testing for significance of those dimensions is not necessary as we know exactly how many dimensions must be included in \(Z_{K^*}^*\). While time consuming, this is necessary for obtaining comparable test statistics from the resampled datasets.
We then cluster the resampled dataset with \((K, Z_K)\) and \((K^*, Z_{K^*})\), and calculate the resampled test statistic as \(T^*_{\{K, Z_K\},\{K^*, Z_{K^*}\}} = D^*_{\{Z_K, K\}} - D^*_{\{Z_{K^*}, K^*\}}\). The p-value for our test is the proportion of resampled test statistics that are smaller than the original test statistic.

We repeat this test for every pair of possible outcomes, and plot the results on a quantitation plot.

**Creating and Reading Quantitation Plots:** The q-value quantitation map, created by (Maitra and Melnykov, 2010), is a convenient way to view the results from the preceding procedure, as well as a way to easily identify the results suggested by the algorithm. The map shows the path that should be taken in interpreting the results from the bootstrap procedures. It indicates whether each step from simple to a more complex model is a significant improvement in clustering.

For simplicity’s sake, we provide a simple example of a Quantitation plot in Figure 2.2b, and will use that as an aide in explaining how the plots are designed to work.

We start on the top row at the left side. Each cell represents a pairwise test that was performed, where the row label represents the simpler null hypothesis and the column label represents the more complicated alternative. Each color represents a level of significance, where a darker color represents a higher level of significance, and more evidence against the null hypothesis. For some pre-specified significance level, we either reject or fail to reject the hypothesis specified by the cell. If we fail to reject the test, then we keep moving to the right along the row, testing the same simple hypothesis against a new more complex alternative hypothesis. If we reject at any cell, we move from our current row to the row specified by the alternative hypothesis and start the same process over from the leftmost cell of that row. If we ever go across an entire row without rejecting that null, then we have found our final result from the algorithm.
Before using the p-values obtained in 2.2.2.2, we must control for the many comparisons done in the testing process. We use Benjamini and Hochberg (1995)’s method (Benjamini and Hochberg, 1995) for controlling false discovery rate to compute q-values. The p-values being adjusted have complicated dependencies with each other, so this correction is more of a heuristic than a theoretically complete FDR correction. The q-value are certainly more conservative than their original p-values, and are used in the quantitation map rather than the potentially over aggressive p-values.

To read Figure 2.2b, we start at the pair of $K = 2$ and $|J_2| = 1$ on the left side of the top row. We will use $|J_K|$, rather than $J_K$ in the quantitation plots for brevity. The dark red colored block on the left side of the top row represents a test of $(K, |J_K|) = (2,1)$ vs. the alternative $(K^*, |J_{K^*}|) = (3,1)$ since the column label for that cell is $(3,1)$. The dark red box means that our q-value is less than 0.01, so we reject $(2,1)$ in that case. Since $(3,1)$ is determined to be better than $(2,1)$ we immediately move down to the second row of the plot rather than continuing along the first. The left most cell of the second row represents the test of $(3,1)$ vs. $(4,1)$, and again is dark red indicating a rejection of the simpler model. We now move down the the third row labelled $(4,1)$ and notice that all of the cells in this row are tan colored. Tan cells indicate q-values of larger than 0.1 meaning that no model more complex than $(4,1)$ was significantly better at clustering the dataset, and thus, $(4,1)$ is the model that should be used to cluster the dataset.

While the q-map uses $K$ and the number of dimensions for convenience, we are actually testing the pair of $K$ and the set of dimensions included, so the actual dimensions needed to cluster $X$ are fully known when the result is obtained from the quantitation plot.
2.3 Examples

2.3.1 Ruspini Dataset

Our examples begin with a case that can be easily visualized and which should be easy to choose the number of clusters for. Figure 2.2 shows the two dimensions of the Ruspini dataset forming a reasonably clear four clusters.

Figure 2.2: The q-value quantitation plot in Figure 2.2b demonstrates the choice of our bootstrap algorithm as $K = 4$ with one dimension. The clustering with this data is shown via the color and shape of the points in Figure 2.2a.

The q-value quantitation plot for this dataset was presented in Figure 2.2b and used as an example in Section 2.2.2.2 where it was explained that the algorithm choses $K = 4$ with 1 dimension as the appropriate way to cluster the dataset. Additionally, although the quantitation plot doesn’t show it, the 1 dimension selected for the dataset is Dimension 2.

Looking closer at Figure 2.2’s point shapes, which represent the clustering resulting from $K = 4$ and using only dimension 2, shows a coherent clustering with the only 2 questionable
points being between the upper 2 clusters. In order to obtain a clustering without these points being switched dimension 1 and 2 would both need to be clustered on. Our algorithm determined that switching the cluster assignments of those 2 points was not worth the added complexity of a new dimension. In this example the two at odds clusters are fairly close together meaning the mix up is not extremely expensive, but in a case where the distance between those clusters was bigger the algorithm could easily decide to incorporate the first dimension to assign those points to their closest cluster centers.

2.3.2 100 km Race Runners

The 100 km Race Runners dataset contains times for 80 runners in a 100 km Ultramarathon. Rather than giving only the total race time, the runners dataset provides information for the time of each runner on each tenth of the race.

Figure 2.3: The q-value quantitation plot in Figure 2.3b demonstrates the choice of our bootstrap algorithm as $K = 2$ with one dimension. The clustering with this setup is shown via the color of the lines in Figure 2.3a. The wide, dark lines represent the cluster centers.

Figure 2.3b shows the q-value quantitation plot created for the runners dataset. This quantitation plot shows that no model more complex than $K = 2$ with 1 dimension is
preferred. The one dimension chosen is the 7th, representing kilometers 60 through 70 of the 100 km race, which is shown as the large dots on the mean lines in Figure 2.3a.

The runners are separated into two groups where one is a fast group, containing 37 runners, and the other is the slower group containing 43 runners. Since most of the runners in the dataset are clustered well, it seems that using only the 7th of the 10 times does an adequate job of partitioning the runners into two groups. One runner in particular has very large values for the first 4 tenths, but has a low time for the 7th tenth so it is grouped with the faster runners. Apparently, neither this discrepancy nor some of the very late race time inconsistencies were enough to warrant adding more dimensions.

One interesting feature of this dataset that is not demonstrated in Figure 2.3a is that the dimensions of the dataset are all positively correlated. This inter-dependency indicates that the various dimensions offer redundant information for clustering, and offer some insight as to why only 1 of the 10 dimensions is used.

2.3.3 Simulated Datasets

We used the CARP package (Melnykov and Maitra, 2011) to generate datasets with known structures and compare the results of different algorithms on them. The structure features that change are the number of dimensions, including noise ($Q$) and informative ($P$) ones, the number of clusters ($K$), and the overlap of the clusters. The overlap is specified as the general overlap in the CARP package, which is a proxy to the misclassification probability for the dataset. The larger the overlap the more difficult to correctly cluster the dataset is. In the simulation study we created 6 different scenarios in which each have 4 different overlap settings and simulated 20 datasets for each scenario. Each simulated dataset has 1000 observations and the noise dimensions are unimodal normally sampled. In ($K$, $P$, $Q$) format the scenarios presented in this study are (2, 5, 0), (2, 3, 2), (4, 6, 4), (4, 10, 0), (6, 4, 6), (6, 15, 5).
We consider and compare results from various algorithms for two different problems. First, the number of clusters that algorithms choose, and second, the accuracy of clustering after dimension selection. For the $K$ selection problem we will compare the algorithm presented in this paper which we call *Boot* for bootstrap, the gap statistic (Tibshirani et al., 2003), the slope statistic (Wang, 2017), and the jump statistic (Sugar and James, 2003) across the different dataset structures and overlaps. For the dimension selection and accuracy problem we compare with *Sparcl* (Witten and Tibshirani, 2010). Table 2.1 and Figure 2.4 show the results of the different $K$ selection algorithms.

### 2.3.3.1 $K$ Estimation

The left most two plots of Figure 2.4 show results of the four algorithms on datasets with no noise dimensions. In the $P = 5$ and $K = 2$ setting all of the algorithms do well, and our algorithm makes the only 2 mistakes. Both of these mistakes were situations in which the algorithm decided to include one of the five dimensions for $K = 2$ and more dimensions for $K = 3$ and the $K$ selection part of the algorithm decided that the increase in $K$ was worth getting 2 new dimensions. That is not to say that the algorithm was right, but that the error was more from the selection of the proper dimensions rather than the selection of $K$. In the $(K = 4, P = 10, Q = 0)$ setting there is more variability in the $K$'s selected. As the overlap increases the results get expectedly worse for every algorithm. The gap statistic seems to perform the best in this example and unlike the other three algorithms presented, which underestimate, ours seems to incorrectly overestimate $K$ in difficult situations. These examples have no noise dimensions, so the algorithms that use every dimension are not using bad information and they excel. The rest of the examples have noise dimensions present and will test the robustness of the algorithms.

The $(K = 2, P = 3, Q = 2)$ situation shows that our bootstrap algorithm out performs the others in the face of noise dimensions. Even in the simple case with a general overlap
Figure 2.4: This figure presents the results of the $K$ selection aspect of the simulation study. Each pane represents a different simulation setup. Within a pane moving from left to right the results of 4 different algorithms are shown for general clustering overlaps of 0.01, 0.05, 0.1, and 0.2. The size of the box represents the number out of 20 dataset resulting in a choice of that $K$. The horizontal black lines denote the true $K$ for the simulation scenario of that pane.

of 0.01 the algorithms mistake the cluster counts frequently. Our bootstrapping algorithm gets slightly worse as the overlap increases, but still maintains its superiority over the other algorithms. Results from ($K = 4, P = 6, Q = 4$) show a similar pattern, although our algorithm seems to deteriorate in the 0.2 overlap situation. The ($K = 6, P = 15, Q = 5$) example shows more of the same as the other examples already discussed. The 15:5 good to noise dimension ratio means that using all of the dimensions is less penalizing to the algorithms that use every dimension than, for example, the $p = 3, q = 2$ example. This is reflected in the relative quality of results. Like the examples with no noise the gap statistics performs the best and all of the algorithms get worse as the overlap increases.
The final scenario is the \((K = 6, P = 4, Q = 6)\) example. This is the simulated setup with the lowest information to noise ratio at 4:6, and this is reflected in the poor results from the algorithms that do not perform variable selection. Even at the easiest overlap of 0.01 the other algorithms fail. The gap statistic and jump statistic each correctly chose \(K = 6\) for 3 of the 20 datasets and the slope chose 0 of the 20 correctly, while our bootstrap algorithm chose 19 of the 20 correctly. As the overlap increases our algorithm does expectedly do worse, but the other already poor algorithms get worse as well. With the overlap of 0.2 no other algorithm chose 6 clusters for a single dataset, while ours was correct for 8 of 20.

On the whole, as the informative to noise dimension ratio decreases our algorithm starts to outpace the others. This outpacing appears especially in the \((K = 2, P = 3, Q = 2)\) and \((K = 6, P = 4, Q = 6)\) scenarios. In the simulation setups with no noise our algorithm performs slightly worse than the best of the others in general, especially the \((K = 4, P = 10, Q = 0)\) scenario with an overlap of 0.2.

### 2.3.3.2 Dimension Selection and Clustering Accuracy

Table 2.2 lists results from the dimension selection for our bootstrapping algorithm as well as Sparcl and the ARI results for clustering the simulated datasets in different dimension selection paradigms. Figure 2.4 shows an alternate view of those same ARI results as box plots. It’s worth mentioning that Sparcl does not choose the number of clusters while it selects variables, so we manually set \(K\) to the actual number of groups used to create the dataset when running it. For the ARI comparison we provide results for our algorithm had it been given the true value of \(K\) in the Known column and the \(K\)-means results for the known true \(K\) and every dimension in the dataset in the All column.

In the Dimension Selection columns of Table 2.2 we show in the Boot column the mean and standard deviation of the number of columns chosen by our bootstrapping algorithm and the \(\omega_{\text{Boot}}\) and \(\omega_{\text{Sparcl}}\) columns are the proportion of the columns that were chosen and were
Figure 2.5: This set of box plots shows the accuracy, measured by the Adjusted Rand Index, of different variable selection enabled clustering algorithms. The methods in the figure are our standard bootstrap algorithm, our algorithm with a fixed $K$, Sparcl, and base $K$-means using all dimensions and known $K$. In general, as the number of noise dimensions increases our algorithm starts to improve relatively, but for datasets with little to no noise other algorithms perform better.

In the Boot column we see that within a dataset size and number of clusters the number of dimensions included expectedly increases as the overlap increases. For a more difficult clustering problem the number of dimensions needed increases. Also expectedly the larger the number of dimensions in the dataset the more dimensions are chosen by our algorithm.

Examining the $\omega_{\text{Boot}}$ and $\omega_{\text{Sparcl}}$ columns we see that Sparcl does very well for all of the datasets except for $(K = 2, p = 3, q = 2)$. In this simulation situation 83% or 84% of the...
total sum of feature weights belong to dimensions that were simulated noise. In the \((K = 2, P = 3, Q = 2)\) scenario we chose a smaller percentage of noise dimensions. Despite that, in other categories with noise dimensions our algorithm chose more noise dimensions than Sparcl, although almost all of the percentages are very large.

Despite \textit{sparcl} choosing a lower proportion of noise dimensions, our algorithm generally obtains a higher clustering accuracy than \textit{sparcl}. Figure 2.5 shows box plots of the adjusted rand index for the four clustering scenarios discussed previously. The Boot and Known methods are both results for versions of our bootstrapping algorithm. Boot denotes the standard algorithm that does variable selection and estimates the number of clusters. Known assumes the actual value of \(K\) and does variable selection with that assumption. \textit{Sparcl} assumes the true \(K\) and weights dimensions based on the information they provide. Finally, the method denoted All is simply the \(K\)-means algorithm on every dimension in the dataset with known value of \(K\).

Like in the \(K\) estimation results the differences in the algorithm’s results reveal themselves in two scenarios – the case with little noise and the case with a large proportion of noise dimensions. In the \((K = 2, P = 5, Q = 0)\), \((K = 4, P = 10, Q = 0)\), \((K = 6, P = 15, Q = 5)\) simulation setups the \(K\)-means algorithm on all dimensions generally exceeds the other algorithms. Especially in the cases with no noise dimensions there is no penalty to including every dimension. When the datasets contain a higher proportion of noise dimensions, as in \((K = 2, P = 3, Q = 2)\), \((K = 4, P = 6, Q = 4)\), and especially \((K = 6, P = 4, Q = 4)\), our algorithms tend to outperform the others.

\textit{Sparcl} is generally the least accurate method except in cases with low overlaps where it can perform well. In the \((K = 2, P = 3, Q = 2)\) simulations there are points at around 0 in the ARI box plots, which doesn’t appear in any other situation. This happens to every algorithm when noise dimensions are used to cluster. There are not enough true dimensions to correct the clustering when a noise dimension is included.
In conclusion, when the simulated datasets contain none or small amounts of noise it is better to use an algorithm that doesn’t eliminate any of those dimensions. On the other hand, when there is moderate to large amounts of noise in the dataset it is beneficial to both the accuracy of clustering to use our variable selection algorithm.

2.3.4 Written Digits

One of the more iconic applications of statistical learning is that of classifying handwritten digits. Between poor handwriting and different styles of numbers, it’s an exceedingly difficult task. These difficulties are exacerbated by the fact that we have no prior knowledge of what the clusters should look like; this is a truly unsupervised environment. Even in a supervised situation, when an algorithm can be trained on what the digits look like, it’s a difficult problem, but our algorithm makes an attempt without that prior knowledge.

Our dataset has 7291 handwritten digits, where each digit is divided into 256 ‘pixels’ on a 16 × 16 grid. Each pixel’s value is on a scale of −1 to 1 where −1 represents a completely white pixel and 1 represents a solidly black pixel. In order to reduce the dimensionality to a more computationally tractable size, we find the principal components that account for 80% of the variation in the dataset and use those components as our dataset. This will remove some of the interpretability of knowing which areas of the image are important, but in the end we are only interested in which cluster each image is sorted into, which is not damaged by the PCA conversion. After the dimensionality reduction we are left with a 7291 × 29 dataset to cluster.

In the quantitation plot provided as Figure 2.6, we can see that the algorithm recommends clustering the dataset into 17 groups using 5 principal components. The components chosen by the algorithm were components 1, 2, 3, 4, and 8. When creating principal components, the first component explains the most variability in the dataset as possible and each time a new orthogonal component is included it also explains as much of the remaining variability
Table 2.1: This table displays the mean and standard deviation of the $K$ selection algorithms from the simulation study. The information in this table is presented more efficiently and eloquently in Figure 2.4.

<table>
<thead>
<tr>
<th>K</th>
<th>P</th>
<th>Q</th>
<th>$\tilde{\omega}$</th>
<th>Dataset Features</th>
<th>K Selection Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0.01</td>
<td>2.10 ± 0.31</td>
<td>Boot Gap Slope Jump</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0.05</td>
<td>2.00 ± 0.00</td>
<td>2.00 ± 0.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0.1</td>
<td>2.00 ± 0.00</td>
<td>2.00 ± 0.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0.2</td>
<td>2.00 ± 0.00</td>
<td>2.00 ± 0.00</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.01</td>
<td>2.00 ± 0.00</td>
<td>3.65 ± 2.92</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.05</td>
<td>2.00 ± 0.00</td>
<td>2.65 ± 2.18</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.1</td>
<td>2.25 ± 1.12</td>
<td>2.15 ± 1.57</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.2</td>
<td>2.65 ± 1.84</td>
<td>1.75 ± 0.44</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>0.01</td>
<td>4.00 ± 0.00</td>
<td>5.75 ± 2.00</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>0.05</td>
<td>4.15 ± 0.67</td>
<td>4.10 ± 0.97</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>0.1</td>
<td>4.20 ± 0.83</td>
<td>4.20 ± 0.89</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>0.2</td>
<td>4.90 ± 1.45</td>
<td>3.60 ± 0.50</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0.01</td>
<td>4.05 ± 0.22</td>
<td>4.00 ± 0.00</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0.05</td>
<td>4.15 ± 0.49</td>
<td>4.00 ± 0.00</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0.1</td>
<td>4.25 ± 0.64</td>
<td>4.00 ± 0.00</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>0.2</td>
<td>5.70 ± 1.66</td>
<td>3.85 ± 0.37</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>0.01</td>
<td>6.05 ± 0.22</td>
<td>3.95 ± 1.39</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>0.05</td>
<td>6.05 ± 0.76</td>
<td>4.25 ± 1.65</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>0.1</td>
<td>5.80 ± 0.83</td>
<td>4.25 ± 1.62</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>0.2</td>
<td>6.50 ± 1.00</td>
<td>3.40 ± 0.82</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>5</td>
<td>0.01</td>
<td>6.10 ± 0.31</td>
<td>6.00 ± 0.00</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>5</td>
<td>0.05</td>
<td>6.10 ± 0.31</td>
<td>6.00 ± 0.00</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>5</td>
<td>0.1</td>
<td>6.25 ± 0.64</td>
<td>6.00 ± 0.00</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>5</td>
<td>0.2</td>
<td>6.65 ± 0.75</td>
<td>5.90 ± 0.31</td>
</tr>
</tbody>
</table>
Table 2.2: This table presents the results for the variable selection portion of the Boot algorithm as well as quantifies the clustering accuracy results for various clustering paradigms. The left third of the table describes the dataset simulation parameters. The middle third provides the mean and standard deviation for the number of dimensions chosen by our bootstrap algorithm and the proportion of the selected dimensions that are simulated noise for both our algorithm and sparcl. The rightmost third provides the mean and standard deviation of the adjusted rand index for the standard bootstrap algorithm presented in this paper, Sparcl, a modification of the bootstrap algorithm with $K$ known, and base $K$-means on all dimensions.

<table>
<thead>
<tr>
<th>Dataset Features</th>
<th>Dimension Selection</th>
<th>ARI Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$ $P$ $Q$ $\tilde{\omega}$</td>
<td>$\omega_{\text{Boot}}$ $\omega_{\text{Sparcl}}$</td>
<td>$\text{Boot}$ $\text{Known}$ $\text{Sparcl}$ $\text{All}$</td>
</tr>
<tr>
<td>2 5 0 0.01</td>
<td>2.00 ± 0.56 1.00 1.00</td>
<td>0.92 ± 0.07 0.94 ± 0.03 0.96 ± 0.01 0.98 ± 0.01</td>
</tr>
<tr>
<td>2 5 0 0.05</td>
<td>2.65 ± 0.81 1.00 1.00</td>
<td>0.87 ± 0.04 0.87 ± 0.04 0.85 ± 0.03 0.90 ± 0.02</td>
</tr>
<tr>
<td>2 5 0 0.1</td>
<td>2.55 ± 0.60 1.00 1.00</td>
<td>0.77 ± 0.04 0.77 ± 0.04 0.72 ± 0.05 0.81 ± 0.03</td>
</tr>
<tr>
<td>2 5 0 0.2</td>
<td>2.95 ± 0.69 1.00 1.00</td>
<td>0.62 ± 0.05 0.62 ± 0.05 0.49 ± 0.08 0.64 ± 0.04</td>
</tr>
<tr>
<td>2 3 2 0.01</td>
<td>1.80 ± 0.62 0.95 0.84</td>
<td>0.96 ± 0.02 0.96 ± 0.02 0.73 ± 0.43 0.73 ± 0.43</td>
</tr>
<tr>
<td>2 3 2 0.05</td>
<td>1.95 ± 0.51 0.93 0.83</td>
<td>0.83 ± 0.20 0.83 ± 0.20 0.66 ± 0.39 0.67 ± 0.40</td>
</tr>
<tr>
<td>2 3 2 0.1</td>
<td>2.35 ± 0.59 0.96 0.83</td>
<td>0.73 ± 0.22 0.72 ± 0.25 0.57 ± 0.34 0.60 ± 0.35</td>
</tr>
<tr>
<td>2 3 2 0.2</td>
<td>2.70 ± 0.80 0.90 0.83</td>
<td>0.51 ± 0.24 0.57 ± 0.20 0.42 ± 0.25 0.48 ± 0.28</td>
</tr>
<tr>
<td>4 6 4 0.01</td>
<td>3.70 ± 0.66 0.99 0.99</td>
<td>0.95 ± 0.02 0.95 ± 0.02 0.92 ± 0.10 0.92 ± 0.12</td>
</tr>
<tr>
<td>4 6 4 0.05</td>
<td>4.95 ± 1.05 0.95 1.00</td>
<td>0.81 ± 0.07 0.82 ± 0.04 0.69 ± 0.16 0.83 ± 0.06</td>
</tr>
<tr>
<td>4 6 4 0.1</td>
<td>5.35 ± 0.75 1.00 1.00</td>
<td>0.69 ± 0.06 0.71 ± 0.04 0.43 ± 0.16 0.70 ± 0.05</td>
</tr>
<tr>
<td>4 6 4 0.2</td>
<td>5.95 ± 1.00 0.94 1.00</td>
<td>0.43 ± 0.09 0.49 ± 0.04 0.21 ± 0.07 0.50 ± 0.04</td>
</tr>
<tr>
<td>4 10 0 0.01</td>
<td>4.90 ± 0.91 1.00 1.00</td>
<td>0.91 ± 0.04 0.91 ± 0.04 0.96 ± 0.02 0.97 ± 0.01</td>
</tr>
<tr>
<td>4 10 0 0.05</td>
<td>6.35 ± 1.42 1.00 1.00</td>
<td>0.78 ± 0.04 0.79 ± 0.04 0.66 ± 0.19 0.85 ± 0.03</td>
</tr>
<tr>
<td>4 10 0 0.1</td>
<td>7.55 ± 1.28 1.00 1.00</td>
<td>0.66 ± 0.06 0.67 ± 0.05 0.35 ± 0.13 0.71 ± 0.03</td>
</tr>
<tr>
<td>4 10 0 0.2</td>
<td>8.05 ± 1.15 1.00 1.00</td>
<td>0.38 ± 0.10 0.46 ± 0.04 0.15 ± 0.06 0.50 ± 0.03</td>
</tr>
<tr>
<td>6 4 6 0.01</td>
<td>3.35 ± 0.67 1.00 0.99</td>
<td>0.96 ± 0.02 0.96 ± 0.02 0.92 ± 0.10 0.56 ± 0.18</td>
</tr>
<tr>
<td>6 4 6 0.05</td>
<td>3.95 ± 0.76 0.95 1.00</td>
<td>0.79 ± 0.08 0.82 ± 0.05 0.69 ± 0.16 0.51 ± 0.16</td>
</tr>
<tr>
<td>6 4 6 0.1</td>
<td>4.15 ± 0.49 0.95 1.00</td>
<td>0.65 ± 0.07 0.63 ± 0.09 0.43 ± 0.16 0.46 ± 0.13</td>
</tr>
<tr>
<td>6 4 6 0.2</td>
<td>4.45 ± 0.60 0.90 1.00</td>
<td>0.38 ± 0.07 0.40 ± 0.05 0.21 ± 0.07 0.36 ± 0.06</td>
</tr>
<tr>
<td>6 15 5 0.01</td>
<td>7.90 ± 1.41 0.99 1.00</td>
<td>0.86 ± 0.03 0.86 ± 0.04 0.94 ± 0.02 0.96 ± 0.01</td>
</tr>
<tr>
<td>6 15 5 0.05</td>
<td>10.20 ± 1.36 0.99 1.00</td>
<td>0.71 ± 0.04 0.71 ± 0.03 0.42 ± 0.24 0.80 ± 0.02</td>
</tr>
<tr>
<td>6 15 5 0.1</td>
<td>12.15 ± 1.63 0.98 1.00</td>
<td>0.56 ± 0.04 0.56 ± 0.05 0.15 ± 0.04 0.62 ± 0.02</td>
</tr>
<tr>
<td>6 15 5 0.2</td>
<td>11.65 ± 2.16 0.97 1.00</td>
<td>0.30 ± 0.06 0.26 ± 0.09 0.07 ± 0.02 0.39 ± 0.02</td>
</tr>
</tbody>
</table>
Figure 2.6: This figure shows the q-map quantitation plot for the clustering of the first 29 principal components of the handwritten digits dataset. By reading the plot as explained in Section 2.2.2.2 we can see that the recommended clustering setup is 17 groups using 5 dimensions.
in the data as possible. Because of this we expect to see early components be deemed the most important to clustering, and indeed that is the case for our algorithm.

There are two different estimations being done by our algorithm here. First is the choice of the correct number of clusters in the dataset and second is the choice of dimensions to use in clustering. We will attempt to compare each of these procedures to alternative approaches that attempt the same task.

Table 2.3: Table showing a comparison of the true clusters of observations in the handwritten digits dataset to how the observations were clustered based on our algorithm’s suggestion. Shows classification frequencies, along with total sizes of true clusters.

<table>
<thead>
<tr>
<th>Truth</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>m</th>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>331</td>
<td>285</td>
<td>298</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>99</td>
<td>90</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1194</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>979</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>234</td>
<td>209</td>
<td>16</td>
<td>10</td>
<td>59</td>
<td>14</td>
<td>68</td>
<td>8</td>
<td>24</td>
<td>8</td>
<td>22</td>
<td>23</td>
<td>731</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>30</td>
<td>344</td>
<td>0</td>
<td>1</td>
<td>63</td>
<td>25</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>161</td>
<td>8</td>
<td>658</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>4</td>
<td>102</td>
<td>0</td>
<td>202</td>
<td>145</td>
<td>0</td>
<td>2</td>
<td>37</td>
<td>3</td>
<td>4</td>
<td>52</td>
<td>2</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>27</td>
<td>26</td>
<td>52</td>
<td>4</td>
<td>11</td>
<td>179</td>
<td>106</td>
<td>45</td>
<td>32</td>
<td>15</td>
<td>4</td>
<td>44</td>
<td>556</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>23</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>23</td>
<td>27</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>102</td>
<td>359</td>
<td>115</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>664</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>8</td>
<td>67</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>370</td>
<td>2</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>11</td>
<td>7</td>
<td>10</td>
<td>10</td>
<td>6</td>
<td>77</td>
<td>95</td>
<td>2</td>
<td>0</td>
<td>9</td>
<td>257</td>
<td>42</td>
<td>542</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>69</td>
<td>75</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>131</td>
<td>3</td>
<td>349</td>
<td>644</td>
</tr>
</tbody>
</table>

As previously stated and shown in Figure 2.6 our algorithm chooses $K = 17$ as the best number of clusters. Ideally the algorithm would choose 10, which is the true number of digits, but there are subtleties and overlaps in the images that lie outside of those 10 simple clusters. The similarities across digit types and differences within those types lead to a nearly mandatory estimation of more than 10 clusters. Of the three $K$ estimation algorithms that were compared against in the Simulation Study in Section 2.3.3.1, Gap and Jump both chooses the maximum specified number of clusters of 20 and Jump chooses 16. Every algorithm overestimates the true number of clusters in the dataset.

The large grid of plots in Figure 2.7 displays how digits and features of those digits were separated among the 17 clusters obtained from clustering with the first, second, third, fourth,
and eighth principal components. Table 2.3 also gives the confusion matrix corresponding to the same clustering as Figure 2.7.

### 2.3.4.1 Description of clusters

In Figure 2.7 the rows represent the true digit type (denoted by numbers from (0) - (9)) and the columns represent the clusters estimated with $K$-means resulting from our algorithm (denoted by letters (a) - (q)). The last two rows of the plot show the mean and standard deviation of each pixel for the images sorted into each of 17 different clusters, which represent the general type of digit in the corresponding clusters. For finer feedback on which digits went into each cluster the rows above the last two show overlaid plots of digits that fall into each intersection of true digits and estimated clusters.

There are several interesting and informative features of Figure 2.7 that are worth investigating further. First of all, the clusters (a), (b), and (c) each seem to show that they contain mostly zeros. Upon closer inspection of the zeros it is clear that there are actual differences between the digits. Cluster (a) is the skinniest, while (b) is slightly wider, and (c) is wider and rounder still. Cluster (n) also has an additional set of zeros that seem to be very wide and short. This partitioning demonstrates some of the difficulties of clustering these digits and in estimating the number of clusters. While it is clear to the human eye that each of these clusters contain zeros, it’s not immediately clear what patterns the clustering algorithms can take advantage of after PCA has obfuscated some of the intuition. In these different zeros, as well as in each example moving across the plot, it will be clear that overlap of the digits is a large part of how the images are grouped. Clusters (a)-(c) clearly have similar structure and a fair amount of overlap, but it was deemed that they were different enough to be separated into different groups.

The ones were basically the only easy to cluster digit. 979 of the 1005 ones were correctly put into cluster (d) and of the 26 remaining, 25 of those were put into cluster (l). These
Figure 2.7: This figure shows the mean of the observations in each of our 17 clusters, the variance of pixels for plots in each cluster, and a grid of figures showing the mean image for observations in the intersection of the true value (rows) and the cluster our algorithm sorted the observations into (columns).

Extra 25 were systematically thicker than the 979 in cluster (d). The single remaining one was put into cluster (q), and looking at its faint image on the last column of Figure 2.7 it can be seen that the digit has a branch on top like that of a (7), which are abundant in cluster (q). There were 33 non ones grouped into cluster (d), all of which were extremely skinny versions of their digit, as can be seen in the (d) column outside of the (1) row.

The twos are grouped mostly in clusters (e) and (f) with (e) being the taller set of twos and (f) being shorter and wider, following the pattern of partitioning seen in the zeros group.
The threes are sorted mostly into (g) with (p) holding a fairly large subset of threes with shorter left edge lines. The fours are split between groups (h) and (i) with the dividing characteristic being width and slant of the digits. Group (f) also includes some fours that have a longer crossing line and a more meridian vertical line.

The fives have been divided into (j) and (k) mostly, separated by the way the top of the five is written with (k) being more deliberate and wider and (j) being skinnier and faster to descend. The pattern in the sixes duplicates the width partitioning pattern seen with the zeros almost identically. Sevens can be found in mostly (o) and (q) with the difference being in the width of the digit drawn. Eights are found primarily in (p), (l), and to a lesser extent (k). Again, width is the defining characteristic with (l), (p), and (k) ascending in width. The eights in (k) match up and are clustered with the set of wide fives and the eights in (l) are grouped with the skinniest sixes. Finally, the nines are grouped into (q), matching the skinny sevens, with an additional chunk in (o) matching the wider nines. The slant of the stems on the sevens and nines also match closely within clusters.

Instead of looking at the individual cells, only looking at the cluster means can reveal a large part of what will be sorted into the cluster. Most of the cluster means show a single fuzzy digit meaning that there was one basic type of character sorted into that group, but clusters (h), (i), (k), (n), and (o) show what looks like a primary character and in the background the extension of that digit that matches what a subcategory of digits would be in that cluster. Cluster (h) for example shows a majority of fours, but the bar across the top indicates that there are also some nines in that group. Additionally the fact that only the top curve of the nines is visible in that cluster mean suggests that the fours and nines in (h) are very overlapped except for the top bar.
2.3.4.2 Assessment of Dimension Selection

Up to this point we have done a detailed review of how the dataset was partitioned, and saw minor differences in digits that dictated which cluster those digits belong to. It was not discussed, however, that the clustering presented and discussed was done using only 5 of 256 possible principal components (when selecting dimensions this 256 is pared down to 29, but this information is still not available to $K$-means). This makes some of the minor differences and the consistency by which those minor differences are held to fairly impressive. It also suggests that the dimensions chosen by our algorithm are diverse and spanning with respect to the features they describe.

Again, we will use adjusted rand index to quantify the accuracy of different clustering setups as well as compare to sparcl. With $K = 17$ and using the five chosen components we obtain an ARI of 0.422. Using all of the 29 components in the parsed down dataset and $K = 17$ produces and ARI of 0.473, which is higher than ours, but still within the probable range of variability. Using $K = 17$, sparcl results in an adjusted rand index of 0.388, which is slightly lower than that of our algorithm.

Expanding on the results of sparcl, we can compare their weighting of components against the components chosen by our bootstrapping algorithm. Our algorithm favored components 1, 2, 3, 4, and 8 for both $K = 10$ and $K = 17$. Sparcl also chose components 1 through 4 as the most important, accounting for 85% of the total weights, but weighted components 6 and 7 above component 8. The fairly large gap between the weights of components 1 – 4 and the rest of the components verifies that sparcl finds these dimensions to be the most important.

On the whole, this is a difficult dataset to cluster and it would be an overstatement to declare any clustering as superior to the others. Each algorithm partitions the data differently according to the information it includes. To demonstrate this, the ARI of clusters obtained
from different algorithms can suggest similarity between the responses. For our algorithm and \textit{sparcl}'s clusters the ARI is 0.576, meaning that there is a substantial difference between the two clusterings. Additionally between our algorithm and \textit{K}-means with every dimension included the ARI is 0.498, which is a largely different set of differences since the ARI between \textit{K}-means with all dimensions and \textit{sparcl} is only 0.428.

2.4 Conclusion

Clustering is a difficult problem that is made more difficult by not knowing the number of clusters or incorporating noisy dimensions that can be harmful to the results. In this paper, we apply a bootstrap approach to a hypothesis testing methodology designed to estimate the number of clusters in a dataset while simultaneously selecting and clustering with only the important variables. The algorithm works in two stages, first iterating through the possible cluster counts and choosing the relevant dimensions for that number of clusters, and secondly testing each of these possible solutions for the one that offers the necessary clustering information, while still remaining as simple as possible.

We showed proof of concept and used the Ruspini dataset as an introductory example to the algorithm, and clustered a set of 100 km race runners with 1/10 of the original correlated dimensions. We then simulated datasets from Gaussian mixtures and compared our algorithm to alternative methods. In general, our algorithm outperformed others as the clustering difficulty increased both in terms of overlap and number of noise dimensions. Finally, our algorithm was used to classify hand-written digits. We took a large dataset of digits and allowed the algorithm to decide how to deal with clustering the difficult to cluster digits. It responded by suggesting 17 clusters with 5 of the principal components, and provided coherent results when we clustered as it suggested.
There is plenty of room for improvement and extension of this algorithm. It seems that the most common mistake made by the algorithm is to not incorporate dimensions that could have been beneficial. This mistake usually reveals itself when the correct number of clusters is passed over and a larger than necessary $K$ is estimated. When the number of dimensions is small the algorithm can be sensitive to violations of the $K$-means algorithm's assumptions. An example of this is when there is very little signal in a dataset, so clustering its noise results in the best clustering of the complete dataset. The algorithm presented here is only useful for dimension selection and $K$ estimation for $K > 2$. Choosing dimensions for $K = 1$ does not make sense, at least in the context of this algorithm, but choosing 0 or 1 clusters is an eminently reasonable clustering outcome. Maitra et al. (2012) introduced a test for 0 groups, it would have to be extended to fit here. Finally, the process of estimating covariance matrices of clusters is very difficult for hard clustering algorithms like $K$-means. These covariance matrices are systematically underestimated when clusters overlap, which can result in resampled datasets that do not simulate from null distribution as expected. A correction factor based on cluster overlap for this estimation procedure would result in better estimation both for $K$ and variable selection.

References


R Core Team (2018). *R: A Language and Environment for Statistical Computing*. Vienna, Austria.


CHAPTER 3. TiK-MEANS: A TRANSFORMATION-INFUSED 
K-MEANS ALGORITHM FOR SKewed GROUPS

A paper to appear in Statistical Analysis and Data Mining – The ASA Data Science 
Journal

Nicholas S. Berry and Ranjan Maitra

Abstract The $K$-means algorithm is extended to allow for partitioning of skewed 
groups. Our algorithm is called TiK-Means and contributes a $K$-means type algorithm 
that assigns observations to groups while estimating their skewness-transformation parame-
ters. The resulting groups and transformation reveal general-structured clusters that can be 
explained by inverting the estimated transformation. Further, a modification of the jump 
statistic chooses the number of groups. Our algorithm is evaluated on simulated and real-
life datasets and then applied to a long-standing astronomical dispute regarding the distinct 
kinds of gamma ray bursts.

3.1 Introduction

Clustering observations into distinct groups of homogeneous observations (Murtagh, 1985; 
Ramey, 1985; McLachlan and Basford, 1988; Kaufman and Rousseeuw, 1990; Everitt et al., 
2001; Fraley and Raftery, 2002; Tibshirani and Walther, 2005; Kettenring, 2006; Xu and 
Wunsch, 2009) is important for many applications e.g. taxonomical classification (Michener 
and Sokal, 1957), market segmentation (Hinneburg and Keim, 1999), color quantization of
images (Celebi et al., 2013; Maitra et al., 2012), and software management (Maitra, 2001). The task is generally challenging with many methods developed that range from the heuristic (Johnson, 1967; Everitt et al., 2001; Jain and Dubes, 1988; Forgy, 1965; MacQueen, 1967; Kaufman and Rousseuw, 1990) to more formal statistical-model-based approaches (Titterington et al., 1985; McLachlan and Peel, 2000; Fraley and Raftery, 2002; Melnykov and Maitra, 2010). However, the most commonly used clustering algorithm is $K$-means (MacQueen, 1967; Lloyd, 1982; Hartigan and Wong, 1979) which finds partitions locally minimizing the within-sums-of-squares (WSS). The algorithm depends on good initialization (Maitra, 2009; Celebi et al., 2013) and is also ideally suited to find homogeneous spherically-shaped and dispersed groups. Further, $K$-means itself does not provide the number of groups $K$, for which many methods (Krzanowski and Lai, 1985; Milligan and Cooper, 1985; Kaufman and Rousseuw, 1990; Hamerly and Elkan, 2003; Pelleg and Moore, 2000; Sugar and James, 2003; Maitra et al., 2012) exist. Nevertheless, it is used (rather, more commonly abused) extensively because of its computational speed and simplicity.

Many adaptations of $K$-means exist. The algorithm was recently adapted to partially-observed datasets (Chi et al., 2016; Lithio and Maitra, 2018). The $K$-medoids algorithm (Kaufman and Rousseuw, 1987; Park and Jun, 2009) is a robust alternative that decrees each cluster center to be a medoid (or exemplar) that is an observation in the dataset. The $K$-medians (Jain and Dubes, 1988; Bradley et al., 1997) and $K$-modes (Carreira-Perpiñán and Wang, 2013) algorithms replace the cluster means in $K$-means with medians and modes. A different $K$-modes algorithm (Chaturvedi et al., 2001; Huang, 1997) partitions categorical datasets.

At its heart, the $K$-means algorithm uses Euclidean distances between observations to decide on those with similar characteristics. As a result, it partitions well when the distinguishing groups of a dataset have features on about the same scale, i.e. when the dataset has homogeneous spherical clusters. One common approach to accommodate this assump-
tion involves scaling each dimension of a dataset before applying $K$-means. It is easy to see that this remedy can have potentially disastrous consequences, e.g. when the grouping phenomenon is what caused the differential scaling in a dimension. An alternative approach to remove skewness transforms the entire dataset as a whole. This method has potentially unusable results when using $K$-means, as seen in the showcase application of this paper.

### 3.1.1 Finding Types of Gamma Ray Bursts

Gamma Ray Bursts (GRBs) are high-energy electromagnetic explosions from supernovas or kilonovas. Observable physical properties have established multiple types of GRBs (Mazets et al., 1981; Norris et al., 1984; Dezalay et al., 1992) with one group of researchers claiming 2 kinds (Kouveliotou et al., 1993) of GRBs and another claiming 3 types (Horvath, 2002; Huja et al., 2009; Tarnopolski, 2015; Horvath and Toth, 2016; Zitouni et al., 2015; Mukherjee et al., 1998). These analyses used only a few of the available features. The 25-year-old controversy between 2 or 3 GRB types is perhaps academic because careful model-based clustering and variable selection (Chattopadhyay and Maitra, 2017, 2018; Almodóvar-Rivera and Maitra, 2018) showed all available features as necessary and found 5 kinds of GRBs.

The BATSE 4Br catalog has 1599 GRBs fully observed in 9 features, namely $T_{50}$, $T_{90}$ (measuring the time to arrival for the first 50% and 90% of the flux for a burst), $F_1$, $F_2$, $F_3$, $F_4$ (the time-integrated fluences in the $20-50$, $50-100$, $100-300$, and $>300$ keV spectral channels), $P_{64}$, $P_{256}$ and $P_{1024}$ (the peak flux recorded in intervals of 64, 256 and 1024 milliseconds). The features all skew heavily to the right and are summarily log$_{10}$-transformed before clustering. On this dataset, $K$-means with the Jump statistic (Sugar and James, 2003) was initially shown (Chattopadhyay et al., 2007) to find 3 groups but careful reapplication (Chattopadhyay and Maitra, 2017) found it to be indeterminate. Model-based clustering (MBC) (Chattopadhyay and Maitra, 2017, 2018) found 5 ellipsoidally-dispersed groups in the log$_{10}$-transformed dataset. Rather than applying $K$-means on the log$_{10}$-transformed
features, we investigate a data-driven approach to choose feature-specific transformations before applying $K$-means.

This paper incorporates dimension-specific transformations into $K$-means. Our Transformation-Infused-$K$-means (TiK-means) algorithm learns the transformation parameters from a dataset by massaging the features to allow for the detection of skewed clusters within a $K$-means framework. Section 3.2 details our strategy that also includes initialization and a modification of the Jump statistic (Sugar and James, 2003) for finding $K$. Section 3.3 evaluates the algorithm while Section 3.4 applies it to the GRB dataset to find five distinct groups (Chattopadhyay and Maitra, 2017, 2018). We conclude with some discussion. A supplement with sections, figures, and tables is also available.

3.2 Methodology

3.2.1 Preliminaries

3.2.1.1 The $K$-means Algorithm

The $K$-means algorithm (Lloyd, 1982; Forgy, 1965; MacQueen, 1967; Hartigan and Wong, 1979) starts with $K$ initial cluster centers and alternately partitions and updates cluster means, continuing until no substantial change occurs, converging to a local optimum that minimizes the WSS. The minimization criterion means the use of Euclidean distance in identifying similar groups and implicit assumption of homogeneous spherically-dispersed clusters. Disregarding this assumption while using $K$-means, as for the BATSE 4Br GRB dataset can yield unintuitive or unusable results. So we propose to relax the assumption of homogeneous spherical dispersions for the groups, allowing for both heterogeneity and skewness. We do so by including, within $K$-means, multivariate normalizing transformations that we discuss next.
3.2.1.2 Normalizing Transformations

The Box-Cox (power) transform (Box and Cox, 1964) was originally introduced in the context of regression to account for responses that are skewed or not Gaussian distributed, but fit model assumptions after appropriate transformation. The commonly-used Box-Cox transformation only applies to positive values, even though the original suggestion (Box and Cox, 1964) was a two-parameter transform that also shifted the response to the positive half of the real line before applying the one-parameter Box-Cox transform. Unfortunately, this shift parameter is difficult to optimize and basically ignored. Exponentiating the observations before applying the one-parameter transformation (Manly, 1976) can have very severe effects. The signed Box-Cox transform (Bickel and Doksum, 1981) multiplies the sign of an observation with the one-parameter Box-Cox transform of its absolute value. Other power transformations (Yeo and Johnson, 2000) exist but our specific development uses the simpler Inverse Hyperbolic Sine (IHS) transformation (Burbidge et al., 1988) that has one parameter, is a smooth function and is of the form

\[ y_i = \Psi^{-1}_\lambda(x_i) = \frac{1}{\lambda} \log \left( \lambda x_i + \sqrt{\lambda^2 x_i^2 + 1} \right) \text{ for } \lambda \neq 0. \]  

(3.3)

As \( \lambda \to 0 \), \( y_i \to x_i \), so \( \Psi^{-1}_0(x_i) \equiv x_i \). Also, the transformation is defined for \( \lambda \in \mathbb{R} \) but is symmetric around 0, so we only consider \( \lambda \in [0, \infty) \) when finding the optimal transformation. (3.3) is for univariate data: for multi-dimensional datasets, we simply use (3.3) on each coordinate with dimension-specific parameters.

IHS transformation parameters are usually estimated by finding the \( \lambda \) that best normalizes the data. For us, however, is it enough to find a transformed space where the groups satisfy the underlying assumptions of \( K \)-means (i.e., homogeneous spherical clusters). Our algorithm, described next, estimates the transformation in concert with \( K \)-means clustering.
3.2.2 The TiK-means algorithm

We first describe our algorithm for the homogeneous case where all groups in our dataset $X$ of $n$ $p$-dimensional records have the same transformation and then consider the non-homogeneous case with cluster-specific transformations.

3.2.2.1 The homogeneous case

The TiK-means algorithm is designed to follow (in large part) the traditional $K$-means algorithm (Lloyd, 1982; Forgy, 1965), with similar iterations, but for an additional step estimating the transformation parameters and a modified distance metric that accounts for the transformation. To fix ideas, we present a simple high-level outline of our algorithm and then highlight the differences between $K$-means and our modification.

1. Provide initial values for the $K$ centers and $\lambda$.

2. Repeat as long as $\lambda$ or cluster assignments change:
   
   (a) Step $\lambda$ in the direction most improving clustering.
   
   (b) Assign each observation to the group with closest mean (as per distance in transformed space).
   
   (c) Update the group centers to be the mean of the transformed observations in that group.

Beyond the usual parameters governing $K$-means, TiK-means also has a parameter vector of interest $\lambda$ that takes values from a (for convenience) discrete set $\Lambda_p$. An element of $\lambda$, $\lambda_i$, denotes the scalar transformation parameter for dimension $i$. Also, Step 2b is the same as in $K$-means, except that that the dataset is transformed using the current $\lambda$ before using Euclidean distance. Step 2c calculates group means in the transformed space.
The most substantial difference of TiK-means over K-means is in Step 2a, where \( \lambda = ((\lambda_i))_{i=1,2,...,p} \). For practical reasons, we choose the set, \( \Lambda_i \), of values for \( \lambda_i \) to be discrete, and decide on updating \( \lambda_i \) by checking one rung up and one rung down from the current value to decide if the update improves our clustering fit. If no eligible steps improve the current clustering for any \( i \), then every element of \( \lambda \) remains unchanged.

A reviewer asked about the case when changing one \( \lambda_i \) does not improve the clustering, but moving a set of two or more would. The TiK-means default is to step only for only one dimension per iteration, but it contains an option to allow for one step per dimension or one step per dimension per cluster for the nonhomogeneous case (Section 3.2.2.2). For the homogeneous algorithm these saddle points have not been observed to occur, but allowing for more than one step per iteration has been seen to help with finding a global optimum in the nonhomogeneous case. Since this is not a well studied practice, using the results from TiK-means with a large step type as initializations for the algorithm with one step per iteration is a safe way to proceed.

\( K \)-means minimizes the sum of the squared distance from each point to its closest cluster mean, called the within cluster sum of squares (WSS). TiK-means uses the same metric in the transformed space but needs to account for the transformation. We use a standard result (see (20.20) of Billingsley (1986)) on the distribution of transformations of random vectors:

**Result 1** (Billingsley, 1986) For a continuous \( p \)-dimensional random vector \( Y \) with probability density function (PDF) \( f_Y(y) \), \( X = \Psi(Y) \), for \( \Psi : \mathbb{R}^p \to \mathbb{R}^p \), has PDF \( f_X(x) = f_Y(\Psi^{-1}(x)) |J_{\Psi^{-1}}(x)| \), where \( |J_{\Psi^{-1}}(x)| \) is the Jacobian of \( \Psi^{-1}(x) \).

The IHS transformation exemplifies using \( \Psi^{-1} \). From Result 1, \( X_i \) contributes to the log-likelihood the additional term \( \log |J_{\Psi^{-1}}(X_i)| \) that involves \( \lambda \). For the case where we have transformed observations from \( K \) groups, the \( k \)th one having mean \( \mu_k^{(\lambda)} \) and common dis-
perspective of $\sigma^2 I$, the optimized loglikelihood function is, but for additive constants

$$
\ell(\hat{\sigma}^2, \hat{\lambda}, \hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_K; X)
= -\frac{np}{2} \log(WSS) + \sum_{i=1}^{n} \log |J_{\hat{\lambda}}(X_i)|.
$$

(3.4)

Equation (3.4) is also the setting for hard partitioning of observations into homogeneous Gaussian clusters in the transformed space: we propose using its negative value as our objective function. For the IHS transformation, $\log |J_{\hat{\lambda}}(X_i)| = -\frac{1}{2} \sum_{j=1}^{p} \log(\lambda_j^2 X_{ij}^2 + 1)$, so (3.4) reduces to

$$
\text{Obj}_{\text{IHS}} = \frac{np}{2} \log(WSS) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} \log(\lambda_j^2 X_{ij}^2 + 1).
$$

(3.5)

3.2.2.2 The nonhomogenous case

Section 3.2.2.1 outlined TiK-means for dimension-specific but cluster-agnostic transformation parameters $\lambda_i$s. A more general version allows for cluster- and dimension-specific transformations. This results in $K$ separate $p$-dimensional cluster-specific $\lambda$ vectors, or a $(k \times p)$-dimensional $\lambda$ matrix. The objective function, using similar arguments as in (3.5), is

$$
\text{Obj}_{\text{IHS},\lambda} = \frac{np}{2} \log(WSS) + \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{K} \sum_{j=1}^{p} \zeta_{ik} \log(\lambda_{kj}^2 X_{ij}^2 + 1),
$$

(3.6)

where $\lambda_{kj}$ is the transformation parameter for the $k$th group in the $j$th dimension and $\zeta_{ik}$ is an indicator variable that is 1 if $X_i$ is in the $k$th group and 0 otherwise. $\zeta_{ik}$ differentiates (3.5) from (3.6) by dictating which of the $K$ clusters’ $\lambda$ vectors should be applied to transform $X_i$. Optimizing (3.6) proceeds similarly as before, but requires more calculations and iterations before convergence. Further, initializations have a bigger role because of the higher dimensional $\lambda$, which leads to more potential local, but not global, minima.
3.2.3 Additional Issues

3.2.3.1 Initialization and convergence

Like K-means, our TiK-means algorithm converges to a local, but not necessarily global, optimum. Our remedy is to run our algorithm to convergence from many random initial values for $\lambda$ and means in the transformed space. The minimizer of (3.5) or (3.6) (as applicable) is our TiK-means solution. Our algorithm is similar to K-means, but has an additional layer of complexity because of the estimation of $\lambda$ at each iteration. Also, Steps 2a-2c of TiK-means are inter-dependent as orientation of clusters at the current iteration determines the next $\lambda_i$ step which also depends on the current partitioning. The additional complexity over K-means slows down convergence, especially in the non-homogeneous case for large $\Lambda$. So we first obtain cluster-agnostic $\lambda$s and initialize the non-homogeneous algorithm with these TiK-means solutions. Our experience shows this approach to have good convergence.

In general, the homogeneous TiK-means algorithm has never been observed to fall into a cycle that does not converge. For complex problems however, the nonhomogeneous TiK-means algorithm, with poorly initialized $k \times p$-dimensional $\lambda$ sometimes finds itself in a loop that it can not escape. This generally happens in sub-optimal areas of the objective function surface, and is addressed with another initializer such as, for instance, the homogeneous TiK-means solution.

In addition to using many random starting points, the $\Lambda$ grid must be specified so that it contains the true $\lambda$ values. If it does not, the estimated $\lambda$ may pile up near the bounds of the grid and result in not actually finding a local optimum. Of course, increasing the upper bound will allow for large $\lambda$ s to be estimated. The objective function rewards spherical clusters so the $\lambda$ values will move based on their relative size with other dimensions’ $\lambda$s. Increasing the density of $\Lambda$ at lower values can allow for the larger $\lambda$s to be estimated more
easily. Finally, if a $\Lambda$ grid that allows for a good estimate of $\lambda$ is difficult to find, then scaling the dataset before clustering can normalize the magnitude of $\lambda$.

### 3.2.3.2 Data Scaling and Centering

Datasets with features on very different scales are often standardized prior to using $K$-means or other clustering algorithms to dampen the influence of high-variability coordinates. This approach can err (Kettenring, 2006), for instance, when the widely-varying values in some dimensions are the result of cluster separation, rather than genuine scaling differences. TiK-means is more robust to scaling than $K$-means because the $\lambda$ needed to sphere groups are allowed to vary while optimizing (3.5). However, scaling is sometimes still operationally beneficial because larger-scaled features may need larger-valued $\lambda$s. While conceptually not an issue because our $\lambda$s are allowed to vary by coordinate, specifying a grid $\Lambda$ for choosing the appropriate $\lambda$s becomes difficult in such situations. This affects performance when $K$ is not known \textit{a priori} and needs to be estimated. We therefore recommend scaling when the variables for clustering are on very different scales.

The IHS transformation is symmetric about the origin so centering of the data affects its fit. We do not center the data in our implementation. (Centering is not an issue with transformations that include a location parameter, such as the two-parameter Box-Cox or Yeo-Johnson transformations.)

### 3.2.3.3 Choosing $K$

The jump statistic (Sugar and James, 2003) is an information-theoretic formalization of the “elbow method” that analyzes the decrease in WSS (distortion, more precisely) with increasing $K$. The statistic is

$$J_k = \left[ \frac{1}{np} WSS_k \right]^{-\eta} - \left[ \frac{1}{np} WSS_{k-1} \right]^{-\eta}$$

where WSS$_k$ is the WSS from the $K$-means algorithm with $k$ groups (WSS$_0 \equiv 0$). Here, $J_k$ is the improvement from $k - 1$ to $k$ groups, so the optimal $K = \arg\max_k J_k$. The jump statistic’s performance is
impacted by the transformation power $\eta$, recommended (Sugar and James, 2003) to be the number of effective dimensions in the dataset, with the examples in that paper (Sugar and James, 2003) using $\eta = p/2$.

Extending the jump statistic to the TiK-means setting is not immediate, because the WSS does not accurately represent the distortion of the clustered data in either the transformed or the original space. Since the $\lambda$ is learned differently for different $K$, the data for each $K$ is in different transformed spaces. We propose as our distortion measure (3.5) or (3.6) for the homogeneous or nonhomogeneous cases. Neither objective function necessarily increases monotonically with $K$, but it functions more similarly to the $K$-means WSS than the transformed WSS does with respect to the behavior of the jump statistics.

Selecting the $\eta$ in $J_k$ generally requires care, that is even more acute with TiK-means. The case of non-homogeneous, non-spherical clusters (which forms the setting for TiK-means) is the one where the out-of-the-box $\eta = p/2$ consistently performs poorly. Any single across-the-board prescription of $\eta$ is unreliable, so we propose calculating our $J_k$ across a range of $\eta$s and choosing the $K$ inside the range of candidate $K$s that maximizes the Jump statistic for the largest number of $\eta$-values. We call the display of argmax$_K J_K$ against $\eta$ the *jump selection plot*.

### 3.3 Performance Evaluations

We illustrate our algorithm on a dataset with skewed groups simulated using CARP (Maitra and Melnykov, 2010; Melnykov and Maitra, 2011). Our first example is a case where $K$-means performs poorly and demonstrates the value of TiK-means. We then evaluate performance on several standard classification datasets. Our comparisons are with $K$-means and Gaussian MBC (Fraley and Raftery, 2006). In all cases, numerical performance evaluations are by the Adjusted Rand Index, or ARI, which compares the estimated clustering assignment
to the true and takes values no more than 1, with 1 indicating perfect agreement in the partitioning and 0 being no better than that expected from a random clustering assignment. All performance results are summarized in Table 3.1.

3.3.1 Illustrative Example

Our illustrative example uses a two-dimensional simulated dataset of two clusters (Figure 3.1a) that are easily separated in transformed space (Figure 3.1b). This is a simulated dataset with known groupings and IHS transformation with \( \lambda = \{1.4, 0.9\} \). The \( K \)-means algorithm with known \( K \) does poorly (Figure 3.1c), with an ARI of just 0.013. Clearly, the horizontal axis dominates the WSS because of its larger scale, so that \( K \)-means is pretty much only influenced by this coordinate in the clustering. Scaling the coordinates and then applying \( K \)-means visually improves performance (Figure 3.1d) but still has a low ARI of 0.055. The scaling helped \( K \)-means use both dimensions, but was not enough to isolate the small-valued cluster. Our algorithm perfectly (Figure 3.1e) recovered the original clustering with both homogeneous and non-homogeneous transformations. Therefore, allowing \( K \)-means to transform the dataset as it assigned clusters helped TiK-means find and isolate the distinct but skewed groups: Figure 3.1f displays the homogeneous TiK-means solution in terms of the back-transformed dataset and shows perfectly separated and homogeneous spherically-dispersed clusters.

3.3.2 Performance on Classification Datasets

Our evaluations are on classification datasets with different \((n, p, K)\) that are popular for evaluating clustering algorithms. Our datasets have \( n \) ranging from \( n = 150 \) to \( n = 10992 \) and \( p \) from \( p = 3 \) to \( p = 16 \). We set \( K_{\text{max}} = \min(2K_{\text{true}} + 1, 20) \) when the \( K \) needs to be estimated. In all cases, we evaluated performance on both scaled and unscaled datasets.
Figure 3.1: 3.1b shows the latent generated data. After transformation the observable data is shown in 3.1a. Two versions of $K$-means are shown in 3.1c and 3.1d. 3.1e shows the TiK-means results and 3.1f shows the inverse transformation of the data in 3.1a estimated by TiK-means.

### 3.3.2.1 Wines

This dataset (Forina et al., 1988; S. Aeberhard and de Vel, 1992) has measurements on $p = 13$ chemical components of 178 wines from $K = 3$ (Barolo, Grignolino and Barbera) cultivars. TiK-means correctly estimates $K$ (see Figure 3.2 for the jump selection plot) with very good performance (Table 3.1) that (in the non-homogeneous case) almost matches MBC which is known to perform well on this dataset. However, the much-abused $K$-means algorithm, while reasonably competitive for $K = 3$ on the scaled dataset, chooses 8 groups
Table 3.1: Performance, in terms of ARI of $K$-means, MBC and TiK-means on datasets for known and estimated $K$.

<table>
<thead>
<tr>
<th></th>
<th>K</th>
<th>Known K</th>
<th></th>
<th>Estimated K</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$K$-means</td>
<td>MBC</td>
<td>$\lambda_p$</td>
<td>$\lambda_{k \times p}$</td>
</tr>
<tr>
<td>Wine</td>
<td>Unscaled</td>
<td>3</td>
<td>0.371</td>
<td>0.967</td>
<td>0.854</td>
</tr>
<tr>
<td></td>
<td>Scaled</td>
<td>0.898</td>
<td>0.930</td>
<td>0.854</td>
<td>0.933</td>
</tr>
<tr>
<td>Olive Oils</td>
<td>Unscaled</td>
<td>3</td>
<td>0.318</td>
<td>0.535</td>
<td>0.809</td>
</tr>
<tr>
<td></td>
<td>Scaled</td>
<td>0.448</td>
<td>0.535</td>
<td>0.815</td>
<td>0.814</td>
</tr>
<tr>
<td>SDSS</td>
<td>Unscaled</td>
<td>3</td>
<td>0.444</td>
<td>0.649</td>
<td>0.761</td>
</tr>
<tr>
<td></td>
<td>Scaled</td>
<td>0.632</td>
<td>0.659</td>
<td>0.629</td>
<td>0.622</td>
</tr>
<tr>
<td>Seeds</td>
<td>Unscaled</td>
<td>2</td>
<td>0.024</td>
<td>0.393</td>
<td>0.946</td>
</tr>
<tr>
<td></td>
<td>Scaled</td>
<td>0.994</td>
<td>0.391</td>
<td>1</td>
<td>0.997</td>
</tr>
<tr>
<td>Pen Digits</td>
<td>Unscaled</td>
<td>3</td>
<td>0.717</td>
<td>0.737</td>
<td>0.675</td>
</tr>
<tr>
<td></td>
<td>Scaled</td>
<td>0.773</td>
<td>0.712</td>
<td>0.675</td>
<td>0.785</td>
</tr>
</tbody>
</table>

Figure 3.2: The Jump selection plot for the wines dataset. $\hat{K} = 3$ is chosen, as every $\eta$ between the $\eta$s that choose $K = 1$ and the $\eta$s that choose $K = 8$ has a maximum jump statistic at 3.

for both the raw and scaled versions of the dataset when $K$ is not known. Performance is substantially poorer for such a high choice of $K$. However, TiK-means performs creditably and nearly recovers the MBC grouping.

### 3.3.2.2 Olive Oils

The olive oils dataset (Forina and Tiscornia, 1982; Forina et al., 1983) contains the concentration of 8 different fatty acids in olive oils produced in 9 regions in 3 areas (North, South and Sardinia) of Italy. Clustering algorithms can be evaluated at a finer level (corresponding
to regions) or a macro areal level. Observations are on similar scales so scaling is not recommended. However, we evaluate performance both with scaling and without scaling. With known $K = 3$, TiK-Means recovers the true clusters better than $K$-means or MBC on both the scaled and unscaled datasets, while for known $K = 9$, TiK-means is the best performer (Table 3.1) on the unscaled dataset. All algorithms perform similarly on the scaled dataset with $K = 9$ groups.

Non-homogeneous TiK-means betters its homogeneous cousin on the scaled dataset but is similar on the unscaled dataset.

With $K$ unknown, $K$-means and MBC both choose $K = 10$ and 11 for the unscaled and scaled data. Without scaling TiK-means chooses $K = 9$, which matches the true number of regions, but with scaling TiK-MEANS chooses $K = 5$. In either case, TiK-means betters the other algorithms for both unscaled and scaled datasets (Table 3.1).

**3.3.2.3 Sloan Digital Sky Survey**

The Sloan Digital Sky Survey (SDSS) documents physical measurements on hundreds of millions of astronomical objects. The small subset of data made available by Wagstaff and Laidler (2005) contains 1465 complete records quantifying brightness, size, texture, and two versions of shape (so $p = 5$). Our objective is to differentiate galaxies from stars. Before applying TiK-means the dataset is slightly shifted in the texture and shape features so that each coordinate is positive. We do so to address the issue that the IHS transformation does not have a shift parameter and naturally splits dimensions at 0 irrespective of whether the split is coherent or not for clustering.

Table 3.1 indicates that $K$ is difficult to identify for all of the algorithms - none can correctly estimate it. Without scaling, the nonhomogeneous TiK-means algorithm chooses $K = 3$ and gets the highest ARI of 0.958 followed by the homogeneous algorithm with $K = 5$ and an ARI of 0.927. On the scaled data, the standard $K$-means jump statistic
chooses \( K = 7 \) and gets an ARI of 0.830. When \( K \) is known and fixed to be 2 the results are more palatable. Model-based clustering, not surprisingly, struggles, having the lowest ARIs except for the unscaled \( K \)-means algorithm. On the unscaled dataset non-homogeneous TiK-means almost recovered the true grouping, with homogeneous TiK-means being close. With scaling, homogeneous TiK-means is perfect while non-homogeneous TiK-means and \( K \)-means performed well.

3.3.2.4 Seeds

The seeds dataset (Charytanowicz et al., 2010) contains seven features to differentiate 3 different kinds of wheat kernels. On this dataset, each method performs similarly with respect to clustering accuracy for known \( K \) (Table 3.1). With unknown \( K \), only TiK-means correctly estimates \( K \) on the unscaled dataset and is the best performer, while MBC at \( \hat{K} = 4 \) is close but performs substantially worse, and \( K \)-means does even worse. All methods perform indifferently on the scaled dataset.

3.3.2.5 Pen Digits

The pen digits dataset (Alimoglu et al., 1996) contains \( p = 16 \) similar-scaled features extracted from each of \( n = 10992 \) handwritten numerical digits. (Because \( Kp \) is reasonably large for this dataset, we only report homogeneous TiK-means because simultaneously optimizing over \( Kp \) transformation parameters over such a large parameter space was not successful.) For this dataset, and with known \( K = 10 \), scaling does not impact performance of TiK-means, \( K \)-means or MBC. MBC is the best performer with an ARI of 0.64 (Table 3.1) while TiK-means marginally betters \( K \)-means. For unknown \( K \), Figure 3.3 indicates that for the scaled dataset, \( K = 8 \) is the preference for every value of the exponent that chooses a \( K \) between \( K = 1 \) or \( K = 20 \). For the unscaled dataset, \( K = 13 \) is estimated for all but a couple \(-\eta\) values. MBC chooses \( K = 19 \) for the unscaled dataset but is indeterminate
for the scaled version, returning the maximum $K = 20$. $K$-means with the jump statistic (Sugar and James, 2003) also can not estimate $K$ for both the scaled and unscaled datasets.

In terms of performance, TiK-means at $K = 8$ is marginally beaten by $K$-means at $K = 20$ for the scaled dataset but is better than MBC. On the unscaled dataset, TiK-means is the indisputable best performer, almost recovering in ARI the performance of MBC with known $K = 10$.

Table 3.3a shows the confusion matrix for the TiK-means clustering with 8 groups on the scaled dataset. The column names show the digit truths that the TiK-means clusters seem to recover. The diagonal elements of the table show that digits are grouped correctly together, with 0s, 1s, 2s, 3s, 4s, 6s, 7s, and 9s grouped well, even though 3s and 9s are grouped together. Many 5s and 8s share a cluster, but other 5s tend to be grouped with 3s.
Table 3.2: Confusion matrix of the true (rows), and the groups (columns) estimated by TiK-means for (a) $\hat{K} = 8$ and (b) $\hat{K} = 13$ for the scaled and unscaled datasets. The column names denote the author-specified digits that can be used to best characterize the TiK-means solutions.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3,9</th>
<th>4</th>
<th>5,8</th>
<th>6</th>
<th>7,8</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>0</th>
<th>5,9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1032</td>
<td>3</td>
<td>10</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>85</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>664</td>
<td>355</td>
<td>120</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>19</td>
<td>1121</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>33</td>
<td>1</td>
<td>1020</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>6</td>
<td>1118</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>427</td>
<td>0</td>
<td>626</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1011</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>145</td>
<td>25</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>56</td>
<td>904</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>215</td>
<td>10</td>
<td>44</td>
<td>26</td>
<td>0</td>
<td>323</td>
<td>34</td>
<td>403</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>243</td>
<td>0</td>
<td>672</td>
<td>122</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) Scaled Dataset, $\hat{K} = 8$

(b) Unscaled Dataset, $\hat{K} = 13$

and 9s, while other 8s are usually grouped with 7s. For true $K = 10$, the confusion matrix (Table 3.8) indicates not much benefit in pre-specifying $K = 10$. Though the additional clusters available to TiK-means succeeds in separating the $\{7,8\}$ group, 0s are also split into two different groups with the other extra cluster rather than resolving the mistakes made with clustering 5s. On the unscaled dataset (Table 3.3b), and with $K = 13$, every digit has a cluster mostly to itself. The three additional groups split the 0s into another cluster, one splits the 8s into a new cluster, and one contains a mixture of 5s and 9s. Expectedly, the common defined clusters for both the scaled and unscaled TiK-means results match almost perfectly. The better performance with $K = 13$ indicates a commonly-observed trait in handwriting classification, that is, that there is substantial variation in writing digits with more curves or edges.

The results of our experiments show the success of TiK-means in overcoming the homogeneous-dispersions assumption of $K$-means while remaining within the scope of the algorithm. It performs competitively with regards to clustering accuracy on all of the datasets presented, sometimes performing much better than $K$-means and MBC. TiK-means is generally as good or better at recovering the true number of clusters as MBC, and is much better than the jump statistic with the default transformation power. TiK-means estimates
more closely when the data is not scaled before hand, reinforcing the previous suggestion of not scaling unless necessitated by \( \lambda \) search. We now use it to analyze the GRB dataset.

### 3.4 Application to Gamma Ray Bursts dataset

As discussed in Section 3.1.1, GRB datasets have so far primarily been analyzed on the \( \log_{10} \) scale to remove skewness before clustering. This summary transformation reduces skewness in the marginals but is somewhat arbitrary with no regard for any potential effects on analysis. TiK-means allows us to obtain data-driven transformations of the variables while clustering. In this application, the features have vastly different scales, so we have scaled the data on the lines of the discussion in Section 3.2.3.2 to more readily specify the \( \Lambda \) grid for estimating the IHS transformation parameters.

Figure 3.4 displays the jump selection plot for the GRB dataset that shows support for 3 groups but more support for \( \hat{K} = 5 \).

![Jump selection plot](image)

Figure 3.4: The jump selection plot for the GRB dataset that shows support for 3 groups but more support for \( \hat{K} = 5 \).

For \( \eta \in (-4.31, -4.83) \), we are recommended a 3-groups solution, but a much larger interval for \( \eta \in (-4.83, -6.27) \), recommends 5 kinds of GRBs in the BATSE 4Br database. So the jump selection plot suggests some support for the 3-groups TiK-means solution but has a clear and overwhelming preference for the partitioning with
Figure 3.5: Three-dimensional radial visualization plots (Dai et al., 2019) of obtained groups in transformed space.

\[ K = 5. \] (A similar observation holds for TiK-means on the unscaled dataset.) Thus, even with \( K \)-means, but done on objectively-chosen transformed spaces, our results demonstrate support the presence of five distinct types of GRBs (Chattopadhyay and Maitra, 2018) than the previously disputed two-or-three types of GRBs which summarily used log-transformed variables and used \( K \)-means and the Jump statistic (Chattopadhyay et al., 2007), but which upon careful examination (Chattopadhyay and Maitra, 2017) was actually indeterminate. Our five groups are distinct, as per the three-dimensional radial visualization (Dai et al., 2019) plot in Figure 3.5 which displays the most-separated projections of the transformed dataset. We now discuss the characteristics of the obtained groups in terms of three summary characterizations commonly used in the astrostatistics literature (Mukherjee et al., 1998) to describe the physical properties of GRBs. These characterizations are the duration \((T_{90})\), total fluence \((F_{\text{total}} = F_1 + F_2 + F_3 + F_4)\), and spectral hardness \(H_{321} = F_3/(F_1 + F_2)\).

Figure 3.6 summarizes performance in terms of these three summary statistics. We use the traditional \(\log_{10}\) scale for these summaries in describing our groups in order to pre-
Figure 3.6: TiK-means clustering of the GRB dataset. Summary of duration ($T_{90}$), total fluence $F_{total}$, and spectral hardness $H_{321}$, introduced in Mukherjee et al. (1998), for each of the five groups obtained using TiK-means.

The burst duration variable $T_{90}$ shows three groups clumped in the long duration range, and an intermediate and a short duration cluster that has some separation from the large group. The total fluence variable shows 5 homogeneous and approximately equally-spaced groups with fairly good separation between each other. The spectral hardness shows less separation than the other values, but has a soft group and a hard group that are partially separated. The three summary statistics characterize our groups in terms of the paradigm of duration/fluence/spectrum as follows: (1) long/intermediate/hard, (2) intermediate/intermediate/soft, (3) short/faint/soft,
(4) long/bright/hard, and (5) long/bright/soft. The fourth and fifth groups have high fluence, but the latter is brighter and could belong to its own very bright fluence class.

We end our discussion here by noting that Figure 3.6b also provides an indication of the reason behind the controversy between 2 and 3 GRB groups in the astrophysics community. When restricting attention only to the most commonly used duration variables, it is easy to see why there would be differing interpretation between 2 or 3 groups. However, incorporating the additional information separates out Groups 2, 4 and 5, as can be seen from the plots of total fluence and spectral hardness. Our TiK-means solution presents groups that are distinct, interpretable and in line with newer results obtained using carefully done Gaussian (Chattopadhyay and Maitra, 2017) or t-mixtures (Chattopadhyay and Maitra, 2018) model-based or nonparametric syncytial (Almodóvar-Rivera and Maitra, 2018) clustering.

3.5 Conclusions

In this paper, we present a modification of $K$-means called the TiK-means algorithm that preserves the simplicity and computational advantages of $K$-means while recovering groups of varying complexities. The efficient and ubiquitous $K$-means clustering algorithm has inherent underlying distributional assumptions of homogeneous spherically dispersed groups. Such assumptions are crucial to its good performance and several alternatives exist, but practitioners routinely use it without regard to its applicability in the given context. These assumptions are relaxed in the TiK-means framework, allowing for more robust application of the $K$-means algorithm.

We provide a modified Jump statistic (Sugar and James, 2003) to determine the number of groups. The jump statistic is dependent on a parameter ($\eta$) that represents the number of effective dimensions in the dataset (Sugar and James, 2003) and can greatly impact performance. We desensitize our method to the choice of $\eta$ by calculating our modified
jump-selected $\hat{K}$ for a range of $\eta$s in a display called the jump selection plot and choosing the one that is supported by the largest range of $\eta$s. Our algorithm is made available to the scientific community through a publicly available R (R Core Team, 2018) package at https://github.com/berryni/TiKmeans and performs creditably on several examples where the groups are not homogeneous and spherically dispersed. Finally, we weigh in on the distinct kinds of GRBs in the BATSE 4Br catalog. Most astrophysicists have hitherto used $K$-means on a few features to arrive at two or three groups in findings that had split the community between two camps, one advocating for either solution. Recent work (Chattopadhyay and Maitra, 2017, 2018) showed that the $K$-means solutions were neither tenable nor determinate in terms of finding the kinds of GRBs and that in reality, MBC with the Bayesian Information Criterion indicates five groups of clusters. Our analysis of the GRB dataset using TiK-means confirms this finding and shows five well-separated groups, shedding more light into this long-standing question with implications on the origin of the universe.

There are a number of areas that could benefit from further work. One potential downside to our algorithm when compared to $K$-means is the running time which can quickly increase as $p$ and/or the fineness of the search grid for $\lambda$ increases. We also need to have a better understanding of the convergence properties of the algorithm. While the convergence of the algorithm has not been proven like in the case of traditional $K$-means, for any reasonable number of starting points the algorithm converges to consistent values of both $\lambda$ and cluster assignments. However, it is conceivable that there exist cases where $\lambda$ and the cluster assignments reach a point where they never converge, and instead oscillate back and forth. We know that $K$-means alone will converge to a local minimum and that a hill climbing $\lambda$ step will as well, but it would be worth investigating if both happening simultaneously guarantees local convergence. Additionally, the current algorithm is implemented in conjunction with a Lloyd’s algorithm (Lloyd, 1982). While easy to follow and modify, later algorithms do a better job at reducing the number of computations and limiting computing
time. Attempting a Hartigan-Wong-style algorithm (Hartigan and Wong, 1979) could speed up TiK-means. Here, care should be taken with regards to the live set because as $\lambda$ changes from iteration to iteration, some of the assumptions made to allow for the shortcuts in the Hartigan-Wong algorithm may no longer be immediate and may need rework. Increasing $n$ has a smaller impact on running time than increasing $p$ because the expensive optimization step is over the $\lambda$ space and relates to $p$. Finally, the TiK-means algorithm has been developed and illustrated in the context of the IHS transformation. While the performance of TiK-means using the IHS is quite good, additional flexibility provided by, for instance, the Yeo-Johnson and the two-parameter Box-Cox transformations may further improve performance. Our development is general enough to allow for such transformations, but it would be worth evaluating their performance. Thus, we see that notwithstanding our comprehensive approach to finding skewed and heterogeneous-dispersed groups within the context of $K$-means, issues meriting further attention remain.

## 3.6 Supplementary Materials

We include information about TiK-means clustering results that were not included in the paper.

### 3.6.1 Wines

Confusion matrices for the Wine dataset TiK-means results are shown in Tables 3.3.

### 3.6.2 Iris

#### 3.6.2.1 Iris

The celebrated Iris dataset (Anderson, 1935; Fisher, 1936) has lengths and widths of petals and sepals on 50 observations each from three Iris’ species: *I. setosa*, *I. virginica* and
Table 3.3: (a) TiK-means results for the scaled and unscaled wines dataset (results are identical) with a $p$-dimensional $\lambda$. TiK-means results with the $k \times p$-dimensional $\lambda$ for the (b) scaled and (c) unscaled datasets.

<table>
<thead>
<tr>
<th>Unscaled/Scaled ($\lambda_p$)</th>
<th>Unscaled ($\lambda_{kxp}$)</th>
<th>Scaled ($\lambda_{kxp}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TiK-means</td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td>1 59 0 0</td>
<td>1 59 0 0</td>
<td>1 59 0 0</td>
</tr>
<tr>
<td>2 0 62 7</td>
<td>2 2 63 6</td>
<td>2 1 67 3</td>
</tr>
<tr>
<td>3 0 0 48</td>
<td>3 0 0 48</td>
<td>3 0 0 48</td>
</tr>
</tbody>
</table>

I. versicolor. The first species is very different from the others that are less differentiated in terms of sepal and petal lengths and widths. At the true $K$, MBC has the highest ARI at 0.904, with 5 misclassifications. Non-homogeneous TiK-means is close, with 6 misclassifications and an ARI of 0.886. Homogeneous TiK-means has 8 misclassifications and an ARI of 0.851. $K$-means performs poorly on both the scaled and unscaled data, although if the Iris dataset is scaled and not centered a priori, then it matches the performance of non-homogeneous TiK-means. Despite this improvement, there is little reason to scale the dataset before $K$-means without knowing the true cluster assignments in advance. Only non-homogeneous TiK-means on the scaled dataset recovers the true number of clusters, while MBC and homogeneous TiK-means both choose $K = 2$. $K$-means fails to choose $K$ with the default $\eta = 2$, although using the jump statistic with $\eta = 4$ chooses $K = 3$ Sugar and James (2003).

Despite the $p$-dimensional version of TiK-means missing two data points more than the best version of the base $K$-means algorithm, TiK-means was run on the dataset without manipulation whereas base $K$-means only performed well on the dataset that was scaled by Root Mean Square (\(\text{RMS}(x) = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} x^2_i}\)) without centering. While manipulating a dataset before clustering is generally fine it is difficult to know exactly which way it should
be scaled in the case that the true clusters are unknown, and there is little reason to scale
the iris dataset by RMS unless the true cluster assignments are known.

![Figure 3.7](image-url)

Figure 3.7: The selection of $K$ using the jump statistic on the iris dataset: (a) Plot of Jump statistic $K$ against $\eta$ and (b) Jump Statistic slice at $\eta = -3$. $K = 2$ is the value chosen.

Because the labels of the iris dataset are known, it is common to cluster the dataset with
3 clusters. Without the labels it is less obvious how many clusters should be used, although
2 or 3 are the obvious answers. Using the algorithm for number of cluster selection outlined
in Section 3.2.3.3, our method chooses 2 clusters as the best way to cluster the data. In
Figure 3.7a, as you follow the dots from left to right, the selection of number of clusters goes
from 1 to 2, and then to 8.

At $\eta = -3$ the method chooses $k = 2$. The plot in Figure 3.7b shows the jump statistics
from the transformed objective functions that were chosen between. In this case $k = 2$ had
the largest value, followed by $k = 3$. This decision between $k = 2$ and $k = 3$ matches our
intuition for the problem.

Tables 3.5a and 3.5b show the confusion matrices from the TiK-means clustering results
for $K = 3$ and $p$ and $k \times p$-dimensional $\lambda$s, respectively.
Table 3.4: Confusion matrix for Iris dataset when clustered using TiK-means with (a) $p$-dimensional $\lambda$ and (b) $k \times p$-dimensional $\lambda$.

<table>
<thead>
<tr>
<th></th>
<th>Iris Dataset ($\lambda_p$)</th>
<th>Iris Dataset ($\lambda_{k \times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td>Setosa</td>
<td>50 0 0</td>
<td>50 0 0</td>
</tr>
<tr>
<td>Versicolor</td>
<td>0 46 4</td>
<td>0 48 2</td>
</tr>
<tr>
<td>Virginica</td>
<td>0 4 46</td>
<td>0 4 46</td>
</tr>
</tbody>
</table>

(a) (b)

3.6.3 Olive Oils

3.6.3.1 Macro Regions ($K = 3$)

Tables 3.5 show the confusion matrices from the TiK-means clustering results for $K = 3$ and $p$ and $k \times p$-dimensional $\lambda$s, respectively for the scaled and unscaled datasets.

3.6.3.2 Micro Regions ($K = 9$)

Tables 3.6 show the confusion matrices from the TiK-means clustering results for $K = 9$ and $p$ and $k \times p$-dimensional $\lambda$s, respectively.

3.6.4 Seeds

Confusion matrices for the Seeds dataset using TiK-means vis-a-vis the true grouping are in Table 3.6.

3.6.5 SDSS

Confusion matrices for the SDSS dataset using TiK-means vis-a-vis the true grouping are in Table 3.7.
Table 3.5: Confusion matrix for the Olive Oils dataset when clustered using TiK-means with $K = 3$ and (a) $p$-dimensional $\lambda$ (b) $k \times p$-dimensional $\lambda$, for the scaled dataset. Corresponding results for the scaled dataset are in (c) and (d). (e) Results obtained on the scaled (and unscaled) dataset for $K = 5$ groups.

<table>
<thead>
<tr>
<th></th>
<th>Olive Oils Dataset - Unscaled ($k = 3$, $\lambda_p$)</th>
<th>Olive Oils Dataset - Unscaled ($k = 3$, $\lambda_{p \times k}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td>South</td>
<td>323 0 0</td>
<td>322 1 0</td>
</tr>
<tr>
<td>Sardinia</td>
<td>0 98 0</td>
<td>0 98 0</td>
</tr>
<tr>
<td>Centre.North</td>
<td>0 114 37</td>
<td>0 67 84</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th></th>
<th>Olive Oils Dataset - Scaled ($k = 3$, $\lambda_p$)</th>
<th>Olive Oils Dataset - Scaled ($k = 3$, $\lambda_{p \times k}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td>South</td>
<td>323 0 0</td>
<td>323 0 0</td>
</tr>
<tr>
<td>Sardinia</td>
<td>0 98 0</td>
<td>0 98 0</td>
</tr>
<tr>
<td>Centre.North</td>
<td>0 98 53</td>
<td>0 100 51</td>
</tr>
</tbody>
</table>

(c)

<table>
<thead>
<tr>
<th></th>
<th>Olive Oils Dataset - Scaled ($k = 5$, $\lambda_p$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
</tr>
<tr>
<td>South</td>
<td>218 0 0 0 105 0</td>
</tr>
<tr>
<td>Sardinia</td>
<td>0 98 0 0 0</td>
</tr>
<tr>
<td>Centre.North</td>
<td>0 2 99 0 50</td>
</tr>
</tbody>
</table>

(e)

3.6.6 Pen Digits

Table 3.8 shows the confusion matrix of the TiK-means solution at the true $K = 10$. 
Table 3.6: Confusion matrix for the Olive Oils dataset when clustered using TiK-means with $K = 9$ and (a) $o$-dimensional $\lambda$ and (b) $k \times p$-dimensional $\lambda$, for the unscaled and (c, d) scaled datasets.

### Olive Oils Dataset - Unscaled ($k = 9$, $\lambda_p$)

<table>
<thead>
<tr>
<th></th>
<th>TiK-means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apulia.north</td>
<td>23 2 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Calabria</td>
<td>0 56 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Apulia.south</td>
<td>0 13 193 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sicily</td>
<td>14 15 7 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sardinia.inland</td>
<td>0 0 0 0 65 0 0 0 0</td>
</tr>
<tr>
<td>Sardinia.coast</td>
<td>0 0 0 0 33 0 0 0 0</td>
</tr>
<tr>
<td>Liguria.east</td>
<td>0 0 0 0 0 3 11 0 36</td>
</tr>
<tr>
<td>Liguria.west</td>
<td>0 0 0 7 0 8 16 19 0</td>
</tr>
<tr>
<td>Umbria</td>
<td>0 0 0 0 0 0 0 0 51</td>
</tr>
</tbody>
</table>

(a)

### Olive Oils Dataset - Unscaled ($k = 9$, $\lambda_{p \times k}$)

<table>
<thead>
<tr>
<th></th>
<th>TiK-means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apulia.north</td>
<td>23 2 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Calabria</td>
<td>0 56 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Apulia.south</td>
<td>0 15 191 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sicily</td>
<td>14 15 7 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Sardinia.inland</td>
<td>0 0 0 0 65 0 0 0 0</td>
</tr>
<tr>
<td>Sardinia.coast</td>
<td>0 0 0 0 33 0 0 0 0</td>
</tr>
<tr>
<td>Liguria.east</td>
<td>0 0 0 0 0 11 34 3 2</td>
</tr>
<tr>
<td>Liguria.west</td>
<td>0 0 0 7 0 16 0 27 0</td>
</tr>
<tr>
<td>Umbria</td>
<td>0 0 0 0 0 0 0 0 51</td>
</tr>
</tbody>
</table>

(b)
<table>
<thead>
<tr>
<th>Region</th>
<th>TiK-means</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apulia.north</td>
<td>23</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Calabria</td>
<td>0</td>
<td>54</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Apulia.south</td>
<td>0</td>
<td>7</td>
<td>139</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sicily</td>
<td>14</td>
<td>15</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sardinia.inland</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>65</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sardinia.coast</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Liguria.east</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>43</td>
<td>3</td>
</tr>
<tr>
<td>Liguria.west</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>Umbria</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>51</td>
</tr>
</tbody>
</table>

(c) Olive Oils Dataset - Scaled ($k = 9$, $\lambda_p$)

<table>
<thead>
<tr>
<th>Region</th>
<th>TiK-means</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apulia.north</td>
<td>24</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Calabria</td>
<td>1</td>
<td>54</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Apulia.south</td>
<td>0</td>
<td>6</td>
<td>137</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sicily</td>
<td>17</td>
<td>14</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sardinia.inland</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sardinia.coast</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Liguria.east</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>37</td>
<td>3</td>
</tr>
<tr>
<td>Liguria.west</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>Umbria</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>47</td>
</tr>
</tbody>
</table>

(d) Olive Oils Dataset - Scaled ($k = 9$, $\lambda_{k,p}$)
Table 3.6: Confusion matrices for Seeds dataset when clustered with TiK-means at (a-c) true and (d) estimated $K$.

<table>
<thead>
<tr>
<th>Seeds - Scaled/Unscaled ($K = 3$, $\lambda_p$)</th>
<th>Seeds - Scaled ($K = 3$, $\lambda_{k \times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td>Kama</td>
<td>55</td>
</tr>
<tr>
<td>Rosa</td>
<td>9</td>
</tr>
<tr>
<td>Canadian</td>
<td>2</td>
</tr>
</tbody>
</table>

(a) Seeds - Scaled ($K = 3$, $\lambda_{k \times p}$)

| TiK-means |
| Kama | 63 | 1 | 6 |
| Rosa | 5 | 65 | 0 |
| Canadian | 4 | 0 | 66 |

(b)

<table>
<thead>
<tr>
<th>Seeds - Unscaled ($K = 3$, $\lambda_{k \times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TiK-means</td>
</tr>
<tr>
<td>Kama</td>
</tr>
<tr>
<td>Rosa</td>
</tr>
<tr>
<td>Canadian</td>
</tr>
</tbody>
</table>

(c)

<table>
<thead>
<tr>
<th>Seeds - Scaled ($\hat{K} = 7$, $\lambda_p$)</th>
<th>Seeds - Scaled ($\hat{K} = 6$, $\lambda_{k \times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td>Kama</td>
<td>27</td>
</tr>
<tr>
<td>Rosa</td>
<td>0</td>
</tr>
<tr>
<td>Canadian</td>
<td>3</td>
</tr>
</tbody>
</table>

(d) Seeds - Scaled ($\hat{K} = 6$, $\lambda_{k \times p}$)

(e)
Table 3.7: Confusion matrices for the SDSS dataset when clustered with TiK-means with true $K=2$ and estimated $\hat{K}$.

<table>
<thead>
<tr>
<th></th>
<th>SDSS - Unscaled ($k = 2, \lambda_p$)</th>
<th>SDSS - Unscaled ($K = 2, \lambda_{k\times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td></td>
<td>0 1178</td>
<td>1 1177</td>
</tr>
<tr>
<td></td>
<td>270 17</td>
<td>287 0</td>
</tr>
</tbody>
</table>

(a) (b)

<table>
<thead>
<tr>
<th></th>
<th>SDSS - Scaled ($K = 2, \lambda_p$)</th>
<th>SDSS - Scaled ($K = 2, \lambda_{k\times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td></td>
<td>0 1178</td>
<td>1 1177</td>
</tr>
<tr>
<td></td>
<td>287 0</td>
<td>287 0</td>
</tr>
</tbody>
</table>

(c) (d)

<table>
<thead>
<tr>
<th></th>
<th>SDSS - Scaled ($\hat{K} = 5, \lambda_p$)</th>
<th>SDSS - Unscaled ($\hat{K} = 5, \lambda_p$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td></td>
<td>1 2 1164 11 0</td>
<td>3 2 0 1165 1</td>
</tr>
<tr>
<td></td>
<td>0 0 0 201 86</td>
<td>6 0 86 0</td>
</tr>
</tbody>
</table>

(e) (f)

<table>
<thead>
<tr>
<th></th>
<th>SDSS - Scaled ($\hat{K} = 4, \lambda_{k\times p}$)</th>
<th>SDSS - Unscaled ($\hat{K} = 3, \lambda_{k\times p}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TiK-means</td>
<td>TiK-means</td>
</tr>
<tr>
<td></td>
<td>31 0 1139 8</td>
<td>17 1161 0</td>
</tr>
<tr>
<td></td>
<td>0 96 10 181</td>
<td>0 0 287</td>
</tr>
</tbody>
</table>

(g) (h)
Table 3.8: Confusion matrix for the (a) scaled and (b) unscaled datasets showing differences between the true clusters, indicated by rows, and the TiK-means estimated clusters for \( K = 10 \), indicated by rows. The column names denote the author specified digits that the TiK-means clusters seem to cover.

### Pen Digits - Scaled

<table>
<thead>
<tr>
<th>TiK-means</th>
<th>{0,}</th>
<th>{1}</th>
<th>{2}</th>
<th>{3,5,9}</th>
<th>{4}</th>
<th>{5,8}</th>
<th>{6}</th>
<th>{7}</th>
<th>{8}</th>
<th>{0,}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>553</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>46</td>
<td>519</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>660</td>
<td>344</td>
<td>136</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>22</td>
<td>1118</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>40</td>
<td>1</td>
<td>1014</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>5117</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>427</td>
<td>0</td>
<td>624</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1027</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>142</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>18</td>
<td>960</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>10</td>
<td>22</td>
<td>22</td>
<td>0</td>
<td>332</td>
<td>6</td>
<td>97</td>
<td>410</td>
<td>129</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>190</td>
<td>0</td>
<td>662</td>
<td>179</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>23</td>
</tr>
</tbody>
</table>

### Pen Digits - Unscaled - \( K = 10 \)

<table>
<thead>
<tr>
<th>TiK-means</th>
<th>{0,}</th>
<th>{1}</th>
<th>{2}</th>
<th>{3,9}</th>
<th>{4}</th>
<th>{5}</th>
<th>{6}</th>
<th>{7,8}</th>
<th>{0,8}</th>
<th>{5,9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>693</td>
<td>3</td>
<td>14</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>410</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>652</td>
<td>322</td>
<td>57</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>14</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>15</td>
<td>1128</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>24</td>
<td>1</td>
<td>1012</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>34</td>
<td>8</td>
<td>13</td>
<td>1046</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>94</td>
<td>0</td>
<td>627</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>333</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1044</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>156</td>
<td>9</td>
<td>30</td>
<td>0</td>
<td>4</td>
<td>49</td>
<td>894</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>1</td>
<td>18</td>
<td>12</td>
<td>0</td>
<td>126</td>
<td>45</td>
<td>345</td>
<td>432</td>
<td>58</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>70</td>
<td>0</td>
<td>598</td>
<td>98</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>272</td>
</tr>
</tbody>
</table>

(a)

(b)
References


R Core Team (2018). *R: A Language and Environment for Statistical Computing*. Vienna, Austria.


CHAPTER 4. STATISTICAL ANALYSIS OF HANDWRITTEN GLYPHS FOR WRITER IDENTIFICATION

A paper to be prepared and submitted for publication in due course

Nicholas S. Berry and Amy M. Crawford

Abstract This paper presents a method for processing handwritten documents and clustering components of the writing into groups based on structural attributes. The obtained cluster membership information is used to develop a statistical model for writer identification. The presented clustering algorithm creates a grouping structure for glyphs, which are small pieces of handwriting extracted using the handwriter R package developed by Berry. To facilitate the clustering of glyphs, a distance measure inspired by the graph edit distance and a method for calculating the center of a set of glyphs are both introduced. The clustering algorithm is applied to the MNIST dataset for demonstration and exploratory purposes. Various behaviors of the algorithm are explored using its relatively simple digit glyphs. We also establish a Bayesian hierarchical model for modeling a set of writers based on their propensity for writing glyphs that are assigned to certain clusters. We then perform a full scale writer identification analysis on handwritten documents from 27 writers in the Computer Vision Lab dataset.

4.1 Introduction

The processing of handwritten text is important to many fields, and in recent years there has been a shift towards the automation of this processing. At its heart, automatic
handwriting processing is the task of converting an image of scanned handwriting into usable data. This data serves as the foundation for an analysis of the handwriting. When done algorithmically, analysis of handwriting usually falls into one of two categories. The more common analysis objective is to recognize the characters (letters or digits) that are written on the page. This task is called Optical Character Recognition, or OCR, and is useful in applications such as mail sorting or essay topic recognition. The second major category of handwriting analysis is the determination of who wrote the sample, which we call writer identification. For writer identification, the goal does not concern investigating what is written, but the way in which it is written.

Within the writer identification framework, it is important to make the distinction between authorship and writership. Authorship analysis can utilize natural language processing tools to analyze the word choice or punctuation used in a document and often involves a mix of the two objectives described above. Writership identification is limited to an analysis of the shapes that a writer emits via their writing style.

One large and active area of writership identification is in forensic statistics. Such analyses can be used to determine the source of a piece of handwritten evidence such as a potentially forged signature on a legal document, a bank robbery note, or a bomb threat, just to name a few. Traditional handwriting analysis of this nature is done by trained forensic practitioners, who are trained in what to look for when searching for similarities and differences in documents. A forensic examiner must then come to a conclusion of “same source,” “different source,” or “inconclusive” when making a comparison between a document from an unknown writer and one of a known writer.

This paper is concerned with automating portions of this process. We seek to partition writing and focus on how to filter each piece of parsed ink into groups where meaningful statistical comparisons can be made. The resulting groupings are used for exploratory analyses and in statistical modeling to draw probabilistic conclusions for writership determination.
Algorithms and computations we implement here are completely automated, so they require no human intervention or assistance.

There are existing software packages for handwriting analysis of this nature, like the proprietary product FLASH ID® (Sciometrics LLC, Chantilly, VA, USA). Programs like this are designed to automatically process a handwritten document of questioned source and match it with a known writer that exists in a self contained database. These software packages take in a full scanned writing sample and break the handwriting down from potentially large connected structures (words) into smaller components of more manageable sizes (letters or strokes). They record many measurements for each piece, then those measurement are compared across documents. A determination is made about the writer based on the similarity of the questioned document’s measurements and measurements for known writers.

These products are limited by the manner in which they make measurement comparisons. Small components (often characters) are grouped together in a deterministic fashion and measurement comparisons are made only within groups. The deterministic way in which characters are grouped is based on their structures. Small differences in structure can cause otherwise very similar characters to be placed in separate groups. Once separated, they will never be compared in downstream analysis. This leaves valuable comparisons that should be captured, unutilized.

In this paper, we address the major pitfalls of deterministic groupings by introducing a dynamic grouping procedure through a \( K \)-means type clustering algorithm that relies on a distance measure designed specifically for handwritten structures. We begin by processing a scanned handwritten document and segmenting the writing into small graphical structures that we call *glyphs*. Glyphs are the smallest units of writing we consider and they often, but not always, correspond to roman characters. Rather than sort the glyphs into groups in a deterministic fashion, we cluster them based on the similarities of their graphical attributes (nodes, edges, etc.). The clustering algorithm creates robust, error tolerant clusters of glyphs,
which are more descriptive and repeatable for writers, thereby lending to more accurate writership analyses.

To perform a writer identification analysis, we must characterize a person’s unique writing style. We assert that a writer’s style can be sufficiently characterized by the way in which they emit glyphs to a set of clusters. Consider a small example with two writers. Suppose that writer A favors formal cursive, and so they tend to make loops when forming characters such as ‘l’ and ‘h’. Conversely, writer B uses a more broken writing style and tends to make a single stroke ‘l’ and a loopless ascending segment in an ‘h’. Then, when writer A’s documents are processed, their glyphs will be clustered into groups that are characterized by wide cursive-style loops at a higher rate. When writer B’s glyphs are considered, their distribution over the clusters will differ from that of writer A in the sense that writer B will have significantly fewer glyphs that are characterized by wide loops (maybe some ‘h’s wind up with a looped stem, but not many), and more that are simple single strokes.

Section 4.2 discusses the document processing pipeline taking a handwritten document from a scanned image to usable data (glyphs). We introduce our structure based clustering algorithm for glyphs in Section 4.4. The method is applied to two publicly available datasets. First, the relatively simple nature of the handwritten digits in the MNIST dataset (Liu et al., 2003) provide a nice context to investigate the behaviors of the resulting clusters. Then we move to the motivating problem of writer identification with writers from the Computer Vision Lab (CVL) handwriting dataset (Kleber et al., 2013). We compare writership analysis results for the dynamic groupings of clustered glyphs to results using the same model and writers, but based on deterministic groupings of glyphs.
4.2 Document Processing

In Section 4.2 we outline the pipeline for document pre-processing and data extraction. This procedure begins with a scanned handwritten document and results in a set of what we will call glyphs. Glyphs are small pieces of connected ink that serve as individual observations in the analyses performed later in the paper. Processing is done using the R (R Core Team, 2018) package handwriter, available on github https://github.com/CSAFE-ISU/handwriter. This R package, written by Nick Berry for the Center for Statistics and Applications in Forensics (CSAFE) at Iowa State University, provides a toolkit for full handwritten document processing. It has no statistical or fundamental ties to the clustering application, presented in Section 4.4, except that it serves as our glyph extraction tool before clustering can be performed.

4.2.1 Pre-processing

Preparation of a scanned document is an important first step for any software that aims to process handwritten text. Mistakes made in pre-processing can take a signal that is already difficult to extract, and bury it in noise resulting from scanning artifacts or mistaken letter breaks from pen skips. For this reason, careful consideration and implementation of document processing operations is essential for accurate downstream handwriting analyses. In the handwriter pipeline, pre-processing entails binarization to convert the grayscaled image to pure black and white, cleaning of a potentially messy document image, and skeletonization of the writing into a one pixel wide representation of itself.

We begin by converting a scanned image of handwriting into a pure black and white image. If the original image is colored, say an RGB image, those colors are converted to grayscale based on perceived intensity of the image using the simple linear combination $0.2126R + 0.7152G + 0.0722B$ on each pixel. Otsu’s binarization method (Otsu, 1979) is then
applied to the grayscale image to partition each pixel into one of two groups in a fashion that maximizes between-class variance. Those groups are then appropriately normalized to black and white.

After an image is binarized, a cleaning algorithm is applied to detect and fix scanning artifacts or binarization mistakes. Cleaning is done by handwriter using Stentiford and Mortimer (1983)’s procedure, which introduces a set of masks designed to ensure that the black pixels that remain after binarization properly represent pen strokes of the writing. The masks aim to isolate and correct spurious pixels, fill white holes likely caused by mistakes in binarization rather than intentional writing strokes, widen holes that were likely intentionally made, and pinch acute angles in an attempt to preempt future skeletonization inconsistencies.

On the clean binary image, we use skeletonization, specifically the Zhang-Suen thinning algorithm (Zhang and Suen, 1984), to reduce the image to a one pixel wide structure that maintains the shape and connections of the original. The resulting skeleton frame is easier to process downstream, but by choosing to work with the skeleton we sacrifice all information about the width of lines. The width may contain some useful information, such as a possible way to detect the direction of a line via taper, but most of the information it conveys is a function of the writing utensil used, the softness of the writing surface, or even the density of the pixels in the scanned image.

The greatest benefit of thinning is that it facilitates our ability to find structures in the handwritten images. Figure 4.2a shows an example of a binary image (displayed in gray) with its thin counterpart overlayed (black pixel wide line). The overall shape of the written text remains in tact, and key structural features like terminal nodes and intersections are easy to detect. Additionally, between each terminal node and intersection there is a single pixel wide path that we can traverse to map edges of the writing. Section 4.2.2 expands on this idea of using the simple structural elements of the skeleton to extract the requisite
information from the document. Moving forward, only the skeleton of the original scanned writing is kept. The full image is no longer used.

4.2.2 Data Extraction

The goal of the data extraction process is to take a scanned and pre-processed hand-written document and decompose the writing into small manageable pieces called glyphs. Each of these glyphs is considered an attributed graph $G_i$ with vertices $V_i$ that represent terminal points and intersections of paths in the $i^{th}$ glyph. Similarly, the set of edges $E_i$ represents paths within glyph $i$ that connect the terminal points and intersections. These graph structures, similar to those in (Saunders et al., 2011; Miller et al., 2017; Walch and Gantz, 2004; Gantz et al., 2005), will be the basis for evaluating the structural styles of a writer and making comparisons between them.

A document is composed of many disjoint glyphs and, as such, can be expressed as $G = \bigcup_{i=1}^{n} G_i$. Before a document has been processed the divisions needed to isolate glyphs are unknown. The unprocessed document can be thought of as one large graph $G^*$ with vertices $V^*$ and edges $E^*$ yet unknown. These $V^*$ and $E^*$ are analogous to the $V_i$ and $E_i$ of a particular glyph $i$, but include graphical attributes across the entire document rather than one small piece of writing. Under this conceptualization, the task becomes separating one large graph into disjoint subgraphs (glyphs) which can be used directly for analysis.

The first, and most readily available graphical characteristic of $G^*$ is the set of vertices $V^*$. Since our image has been thinned to a one pixel wide skeleton, vertices are straightforward to find. Begin by defining the 8-neighborhood of a pixel to be the 8 pixels surrounding it, shown in Figure 4.1 as the cells numbered 0 – 7. A pixel is considered a vertex if it has either less than 2, or more than 2, 4-connected components in its surrounding 8-neighborhood. The number of 4-connected components in an 8-neighborhood can be found
by counting the number of white to black transitions when traversing the 8-neighborhood path around the pixel, designated by \((0, 1, 2, 3, 4, 5, 6, 7, 0)\) in Figure 4.1. If the number of white to black transitions is less than 2, then that pixel represents a terminal pixel in a path, as there is only a path coming out of it in one direction. If its 4-connected component count is more than 2, then it represents an intersection of multiple paths.

Due to intricacies of the thinning procedure, many intersections in thick pen paths are actually recorded as two intersections in the thinned image. This phenomenon, often called necking, can be seen in the connection of the bottom loop in the cursive ‘L’ in Figure 4.2a and to a lesser extent at the center of the ‘x’ in Figure 4.3. This phenomenon is undesirable when the writer intends to make a single crossing, because it misrepresents the writing structure. To remedy this, any intersection points that are within a small range of each other are merged together to form a single intersection point halfway between them. Figure 4.3 shows an example of merged vertices, where the middle vertex is placed between the two adjacent pixels which are the original intersection points. handwriter uses 2 pixels as the maximum merge distance.

By checking each pixel in the skeleton for satisfaction of the vertex criteria and merging nearby intersection points where appropriate, we get \(V^*\). The elements of \(V^*\) are the vertices of the full document graph. When the document is divided into subgraphs, vertex information will remain the same, but each will become associated with the subgraph that encompasses its pixel. Unlike a general graph data structure, the graph \(G^*\) has a spatial component on the page, so each element of \(V^*\) has a coordinate location. This location is kept as a vertex attribute, which will be useful for finding distances between glyphs.

After identifying \(V^*\) the set of connections between its members, called \(E^*\), will complete our graphical definition of a document. Unfortunately, \(E^*\) is not as easy to construct as iterating across black pixels in the thinned image. Instead, the skeleton is converted into a traversable graph (different than \(G^*\)), and a path finding algorithm is used to compute edges.
The path finding procedure, including creation of the traversable object, is explained in full detail in 4.7.1 for the interested reader.

The path finding algorithm returns the set of edges $E^*$ that connect elements of $V^*$ in $G^*$. Figure 4.3 shows a glyph of the letter ‘x’ with its four edges labeled by different numbers along the traversed path. Like the vertices keep their spatial location as an attribute, an edge keeps the pixel path that connects its two vertices.

The resulting graph structure $G^* = \{V^*, E^*\}$ contains the vertices and edges for an entire document, but in reality, the disconnected subgraphs $G_i$ are of interest. The first step towards extracting the glyphs $G_i$ is to decompose $G^*$ into sub-components corresponding to different blots of ink. Blots of ink separated by white space cannot share nodes or edges, so this separation is trivial. In cursive writing this generally simplifies each sub-graph into a connected word, and in traditional print each sub-graph is a single character or small
connected set of characters. Each time a graph is divided into subgraphs, its vertices and edges are assigned to the subgraph in which they fall.

Once individual blots of ink are being analyzed, they are candidates to be decomposed further into glyphs. This is done by breaking edges that satisfy a set of conditions. Any edge that is roughly horizontal (slope between -2 and 2), longer than 10 pixels, does not have another path connecting its end points, is not a loop, and does not have a terminal node is broken. When an edge is broken, the two newly formed end points are added as vertices in their respective glyph’s attributed graph. The location of a break point on an edge depends on the edge’s shape. If it has a clear trough, then it is broken at the base of that trough, and if not then it is broken at its midpoint.

One exception to the breaking rule is that if a break would separate a graph into two extremely simple sub-graphs, then it is left intact. Here, a sub-graph is considered sufficiently simple if its structure is simpler than 4 vertices and 3 edges. This prevents over-breaking of certain character types (‘m’s and ‘H’s, mostly) into glyphs that are substantially smaller than the character.

After this breaking, each glyph represents roughly one character. The document graph $\mathbf{G}^*$ has been fully decomposed into a disjoint set of glyphs $\{\mathbf{G}_i\}$, each with their respective
vertices and edges. Although the glyphs are not truly characters, they do represent partitions of the writing that are generally small and comparably sized structures.

The last step of processing the image is to give the vertices in each glyph an ordering so that we can coherently compare vertices across glyphs in sections that follow. The vertices are first ranked by connectivity, and the single node with the most connections is called node number 1. If there is a tie for the most connected vertex, then the left most one is considered first. If there is still a tie, then both vertices are necessarily in the same column, and the upper vertex is first.

After the first vertex is found, the ordering of the remaining nodes is based off of the first. A more connected node will always be given a smaller rank than a less connected one. The nodes with the same connectivity are ordered by starting from directly left of the first node and looking in a clockwise rotation. The nodes are ranked in the order they are reached.

Recall that each glyph is an attributed graph that retains the list of pixels in each edge, the pixel locations of the vertices in the graph, and we have just added the ordering of the nodes. The adjacency matrix for each glyph follows naturally from the ordered nodes and edges. It is important to mention here that although we only use handwriter for document processing, it also has feature extraction capabilities such as finding centroids, slants, and other measurables for each glyph. These features are undeniably important for downstream handwriting analysis, but will not be used directly for clustering and will be ignored for now.

4.3 Deterministic Groupings

In this section we introduce two deterministic methods to group glyphs before moving on to the dynamic structure based clustering method in Section 4.4. These deterministic groups are based only on the adjacency matrices of a glyph, making them readily available after processing is done.
The first deterministic group set is determined by collecting all glyphs with the same adjacency matrix and placing them in a group together. Since each group has a unique adjacency matrix in these groups we call it the adjacency grouping. Walch and Gantz (2004); Miller et al. (2017); Saunders et al. (2011); Gantz et al. (2005) utilize a similar method, which they call “isocode”. Adjacency grouping is sensitive to small changes in glyph structure, because even a tiny accidental edge changes the adjacency matrix, and thus the adjacency grouping. Additionally, the ordering of the vertices in the glyph effects the resulting clusters. Although there are many potential ways to order vertices, it is very difficult to create an ordering without some sort of common pitfall. Since this grouping method is so sensitive to small changes, the number of resulting groups is very large.

For the CVL dataset we analyze 162 documents, and handwriter partitions these documents into a total of 52451 glyphs. The number of unique adjacency groups for those glyphs is 1636. The two most common groups, shown in Figure 4.4, account for approximately 60% of all glyphs in the CVL documents.

The main advantage of this grouping method is the strict structural similarity between the elements of a group. Within a group, each glyph has the same number of edges and vertices, allowing for easy feature extraction and comparison on small components of the graphs. (Gantz et al., 2005) utilize this by collecting a large number of measurements on each edge and comparing within groups with the same isocode (and shape code, which they define).

The second way to group deterministically is similar to the adjacency grouping, but is less restrictive and independent of the node ordering. Rather than give each unique adjacency matrix its own group, this method relies only on the connectivity of each vertex (the row or column sums of the adjacency matrix) ordered from lowest to highest. The vector formed from this sorting represents a grouping of glyphs based on their connectivity,
without regards for order of nodes or the pattern of connectivity outside the count. We call this the connectivity grouping method.

Recall that the CVL document glyphs formed 1636 unique groups under the adjacency method. On the same data, the connectivity grouping results in only 319 groups. This makes for substantially fewer groups to compare, but sacrifices the direct edge comparability that the adjacency grouping provides. While the total number of groups is drastically smaller using the connectivity method, the two groups shown in Figure 4.4 remain the first and second most populated groups respectively. Note that there is no way to get a connectivity group assignments of “1113” (4.4a) or “11” (4.4b) other than by being isomorphisms of the graphs shown.

Both deterministic grouping systems suffer from the same major pitfall, which is that glyphs with similar, but slightly different structures will never be compared. It is clear that under both deterministic routines, glyphs will only be grouped with others having the same number of vertices. In spite of containing a potentially large amount of information about writing style, informative comparisons can be missed because of very minor differences.

### 4.4 Clustering

All $K$-means type clustering algorithms hinge on the ability to calculate two essential quantities: a distance (or discrepancy, or similarity) measure and a measure of center. The distance measure must be able to calculate the discrepancy between two observations as well as between an observation and a center. The measure of center should summarize a set of observations by creating a prototype that falls as near as possible to all of the set elements simultaneously.

When observations are numeric vectors, as is standard in traditional clustering, Euclidean distance and mean (or median) are natural choices for measures of distance and center,
Figure 4.4: 4.4a shows an example of the most common graph structure. 4.4b shows an example of the next most common. (c) and (d) show the adjacency matrices, including the row sums which define node connectivity.

respectively. When clustering glyphs, neither the discrepancy measure nor the measure of center are as easily defined. Section 4.4.1 introduces examples of these two essential measures that have been established in the general graph framework. Then, for glyphs, which are a kind of attributed graph, we develop both a distance measure and a mean calculation. Finally, we will outline an accompanying clustering algorithm for glyphs.

4.4.1 Graph Distances and Prototypes

With the general motivation of grouping structurally similar graphs together, it is instructive and informative to delve into existing clustering methods with similar objectives. Clustering graphs (where individual graphs are the observations, rather than clustering vertices within a graph based on connectivity) is a difficult concept due to the lack of an easily
defined discrepancy measure, potential complexity in dimensionality, and the lack of a coherent measure of center.

One example of a discrepancy measure for graphs is the error-correcting graph matching of Messmer and Bunke (1998). In their paper, the resulting distance measure for two graphs is a sequence of steps that are necessary to transition from one graph to the other, called the edit distance. The edit distance for graphs is related to a much simpler concept used to quantify the difference in two strings (Levenshtein, 1966). In the string edit context, operations are sequentially applied to a string \( S_1 \) until it matches another string \( S_2 \). The available edit operations are \textit{Change}, \textit{Insert}, \textit{Delete}. Expectedly, \textit{Change} switches one character to another, \textit{Insert} inserts a new character, and \textit{Delete} removes a character. The string edit distance calculation can be posed and solved as a dynamic programming problem (Wagner and Fischer, 1974) where the difference between each of the pairwise combinations of letters across strings is computed before the minimum distance letter matching is calculated.

In the graph edit distance calculation, like with strings, the goal is to sequentially apply edit operations to transition one graph to another. The necessary operations for graph editing are conceptually the same as in the string context, except they can be applied to both edges and vertices. Each of these edit operations is given a cost, and the difference between two graphs is the sum of the minimal cost edit sequence between them. So given two graphs, \( G_1 \) and \( G_2 \), the edit path \( \{E(G_1, G_2)\} \) denotes the sequence of cheapest edits necessary to transition \( G_1 \) to \( G_2 \). Then \( \sum \{C(E(G_1, G_2))\} \) is the sum of the sequential edit costs and defines the edit distance between \( G_1 \) and \( G_2 \). Two graphs with a small \( \sum \{C(E(G_1, G_2))\} \) are thus considered close together. To use the edit distance, one must specify the cost of the various edit operations. The costs are context dependent, and usually determined based on experience, such that common operations are given a relatively smaller cost.

The graph edit distance is used in (Günter and Bunke, 2002) to implement a Self Organizing Map (Kohonen, 2012) for graphs. In addition to using the graph edit distance
as their discrepancy measure, they use the sequence of cheapest edits \( \{E(G_1, G_2)\} \), to calculate a weighted mean of \( G_1 \) and \( G_2 \). Their weighted mean results from applying a sub-portion of graph edits from \( \{E(G_1, G_2)\} \), say \( \{E(G_1, G_2)\}_1 \) to \( G_1 \). The resulting graph lies somewhere between \( G_1 \) and \( G_2 \), and if the remaining, unapplied edit operations \( \{\{E(G_1, G_2)\}_1^C \cap \{E(G_1, G_2)\}\} \) are applied to the mean graph, the resulting graph would be \( G_2 \).

The graph edit distance provides a method for calculating the similarity between graphs that accounts for, and is robust in allowing, small changes in the structure. Graphs that are compared using this algorithm do have labelled vertices, but generally do not have edge attributes such as lengths or curvatures, or vertex attributes like locations. The graphs that coincide with handwritten glyphs do have these attributes, which will be leveraged in their distance calculations. In the following section, we exploit edge attributes to develop a distance measure for two glyphs and a mean calculation for a set of glyphs. These measures will emulate aspects of the graph edit distance measure, with costs corresponding to the size of the changes necessary to transition one glyph into the other.

### 4.4.2 Distance Measure for Glyphs

Figure 4.5 shows a side by side comparison of two different versions of the letter ‘f’. Each letter has been processed and segmented into individual paths. Figure 4.5a has 4 paths, each emerging from a central node, and Figure 4.5b has only 3. Figure 4.5b is missing the path extending out of the right side of the central node. Despite this difference, the letters are close together and should be eligible to be grouped into the same cluster for comparison. The following framework we develop allows for this opportunity. Neither of the deterministic methods discussed are able to group these letters together, since they have a different number of nodes and edges.
Figure 4.5: Two similarly shaped ‘f’ s that help demonstrate the calculation of the distance measure. Section 4.4.2 walks through the calculation with these letters. The weighted means of these letters are provided in Figures 4.7g-4.7l.

The distance measure for clustering glyphs is based on ideas related to the graph edit distance. Like the edit distance, the distance between two glyphs is the sum of a set of changes that have to be made to get from one to the other. However, there are distinct differences in how the set of changes is calculated. First, only edges of the glyphs are taken into account. There is no cost directly associated with vertex edits because every edge has two vertices, so differences in the vertices of the glyphs can be reflected by altering an edge. Second, the attributes of the edges will be used to determine the cost associated with each change.

Now, we consider each glyph as completely described by its collection of edges. Differences in the edges of glyphs directly reflect differences in the structure of the glyphs. With this in mind, the problem of calculating glyph distances can be simplified to calculating edge distances. We present a distance measure for two edges, then provide the mechanism for computing full glyph distances by combining edge distances.
4.4.2.1 Distance between Two Edges

To develop the edge distance measure we keep two things in mind. First, that the distance should connect back to the graph edit distance. Recall that the edit distance is the sum of the cost of the set of changes that must be made to one graph to equalize it with another. Second, the distance between glyphs should be, in part, characterized by their structures, while allowing glyphs with similar but not identical structures to be grouped together.

Consider the two glyphs in Figure 4.5a and 4.5b, which we call $G_1$ and $G_2$. The set of edges in the two graphs are called $E_1$ and $E_2$, respectively, and $|E_i|$ is the number of edges in $G_i$. Then, choose one edge from each graph to be compared, and denote them as $e_1 \in E_1$ and $e_2 \in E_2$. Denote the distance between these edges as $d(e_1, e_2)$. This edge distance is comprised of three distinct components:

1. difference in **endpoint locations** of the edges

2. difference in the **straight-line distance** between edge endpoints

3. a rough estimation of the difference in **edge shapes**.

Before $d(e_1, e_2)$ is calculated, we place both comparison glyphs on the same 2-dimensional coordinate system with their center of gravities overlapping on the origin. This allows for meaningful edge location comparisons. In addition, information necessary to make the above comparisons is not readily available from edges as they are currently recorded, and so we must also prepare each edge $e_1$ and $e_2$ before $d(e_1, e_2)$ can be calculated.

Section 4.4.2.2 details the standalone preparation of an edge, Section 4.4.2.3 gives the edge distance calculation, and in Section 4.4.2.4 we combine the edge distances to complete the process by which we measure the distance between two glyphs.
4.4.2.2 Prepare Edge Quantities

In this section, we will outline the preparation of a single edge, denoted as $e$, including notational preliminaries for defining key components. Figure 4.6 provides a set of companion plots for the edge distance calculation. For the purpose of Figure 4.6 we assume that the matching direction of the edges is known to be left to right in each edge. Since these glyphs both have a single edge, the edge distance is equal to the glyph distance. As components of the distance are presented, Figure 4.6 will be referenced back to as a visual aid.

For endpoint locations, the first component from the list in Section 4.4.2.1, only pixel locations of the end points of the paths are needed. We denote the location of the first terminal pixel, the starting point of edge $e$, as $p_e^1$. The second terminal pixel location, the end point of the edge, is $p_e^2$. These locations are recorded as $(x,y)$ points on the coordinate plane where the glyph centroid is anchored at $(0,0)$. The points connected by the red lines in Figure 4.6c represent the edge end points.

The second and third components will make use of the straight line that connects $p_e^1$ and $p_e^2$, call it $\ell_e$. Then the second component, the straight-line distance between endpoints, is the simple length of $\ell_e$. The red line segments in Figure 4.6c depict $\ell$ for the two example single edge glyphs.

In the third component of edge distance, we need a rough estimation of the shape of the edge $e$. To accomplish this, we characterize the shape of an edge as the way that it deviates from the straight line $\ell_e$. If an edge is relatively straight (think the number one), we do not expect to see much deviation of the edge pixels from $\ell_e$. If an edge has more curvature (think the number five) we expect more deviation of the edge from the straight line between its endpoints. The direction and magnitude in which the $e$ deviates from $\ell_e$ is how we capture its shape. The black dots on the interior of the edges in Figure 4.6c represent the seven shape points dividing the edges into eight segments.
We call this a rough estimation of shape because rather than capture deviation from the straight line for every pixel in the edge, we split the path into eight equal length mini-paths and record the deviation for only the seven internal equally spaced points that act as dividers of the mini-paths. The number of mini-paths was arbitrarily chosen, and of course can be increased or decreased depending on the desired precision of the measurement. Denote the equally spaced quantile points on the edge $e$ as $q^e_1, q^e_2, q^e_3, \ldots, q^e_7$. Similarly, place seven equally spaced points on the straight line that connects the edge endpoints and call them $q^\ell_1, q^\ell_2, q^\ell_3, \ldots, q^\ell_7$. Then, subtract each of the line points from each of the corresponding edge points to obtain seven calculated shape points that are a rough representation of the shape of edge $e$. These shape points are $s^e_1 = q^e_1 - q^\ell_1, \ldots, s^e_7 = q^e_7 - q^\ell_7$.

Together, the endpoint locations $p^e_1$ and $p^e_2$, the length of the straight line $\ell^e$ between endpoints, and the shape points $s_1^e, s_2^e, \ldots, s_7^e$ are the set of components needed from each edge $e$ for an edge distance comparison.

### 4.4.2.3 Edge Distance Calculation

We now return to the edges $e_1 \in \mathcal{E}_1$ and $e_2 \in \mathcal{E}_2$ introduced above and recall that distance between them, $d(e_1, e_2)$, is comprised of three distinct comparisons.

The first comparison is of end point locations in order to determine how similar the positions of the edges are within their glyphs. This measure of edge location is grounded in the endpoint pixel locations only. The shape and orientation of the edge will be measured by later distance components, so measuring location independently of the other physical attributes of the edge prevents double counting information. The location component of the edge distance measure is denoted by $d_{loc}(e_1, e_2)$, and is defined below.

The assignment of $p^e_1$ and $p^e_2$ to each of the endpoints of $e$ is done arbitrarily, so their location comparisons need to be conducted in both directions. To demonstrate, we revisit Figure 4.5. Comparing edge 4 in 4.5a from bottom to top with edge 3 in 4.5b from bottom
to top is how they should be compared, because that is the order in which they are the most similar. Sometimes, because of the arbitrary endpoint numbering, path 3 in 4.5b happens to start at the top, so the order of comparison should be reversed. When relevant for edge distance component calculations introduced in this Section, a plus sign subscript \( d_+ \) indicates a distance component taken in the original assignment order of the paths, and a minus sign subscript \( d_- \) indicates that the order of the second path is reversed. Choosing the correct order of comparison will be addressed in Equation 4.7 after all three of the edge distance components have been established.

Now, as promised above, we define the location component of the edge distance measure, \( d_{loc}(e_1, e_2) = \{d_{loc+}(e_1, e_2), d_{loc-}(e_1, e_2)\} \) as

\[
\begin{align*}
    d_{loc+}(e_1, e_2) &= \min\{p_1^{e_1} - p_1^{e_2}, p_2^{e_1} - p_2^{e_2}\} \\
    d_{loc-}(e_1, e_2) &= \min\{p_1^{e_1} - p_2^{e_2}, p_2^{e_1} - p_1^{e_2}\}.
\end{align*}
\]

Since the minimum of the endpoint differences is used, two paths that either begin or end at the same pixel location will have a \( d_{loc} = 0 \) regardless of their shapes, angles, or lengths. Figure 4.6d shows \( p_1^{e_1} - p_1^{e_2} \) and \( p_2^{e_1} - p_2^{e_2} \) for the example glyphs. After taking the minimum, the location shift distance component is only 1.4. The rest of the differences in the edges will be accounted for by other distance components.

The second distance component concerns the difference in the lengths of \( \ell^{e_1} \) and \( \ell^{e_2} \). We can think of this piece of the edge distance measure as adding a penalty that reflects the difference in the straight line displacement of the endpoints in each edge. Define this straight line distance component \( d_{dist}(e_1, e_2) \) as

\[
d_{dist}(e_1, e_2) = \left| \|\ell^{e_1}\|_2 - \|\ell^{e_2}\|_2 \right|
\]
where $|| \cdot ||_2$ denotes the Euclidean norm of a line segment. Note that there is no need to make this comparison in both directions since the specification of the straight line between end points for an edge, $\ell^e$ is unchanged by the ordering of the terminal pixel locations $d_1^e$ and $d_2^e$. Figure 4.6e shows the difference in the straight line distances. On the diagonal segment, the solid portion has the same length as the shorter segment, and the dotted portion, with a length of 9.9, represents the difference in the distances of $\ell^{e_1}$ and $\ell^{e_2}$.

The final component of the distance measure is the average difference in the shape points of $e_1$ and $e_2$. Here, we certainly need to consider both the forward ($d_{shape+}$) and backward ($d_{shape-}$) orderings of the shape points. Let $d_{eu}(\cdot, \cdot)$ denote the Euclidean distance between two points. Then, the shape contribution to the edge distance measure $d_{shape}(e_1, e_2) = \{d_{shape+}(e_1, e_2), d_{shape-}(e_1, e_2)\}$ is

$$d_{shape+}(e_1, e_2) = \frac{1}{7}[d_{eu}(s_{1}^{e_1}, s_{1}^{e_2}) + d_{eu}(s_{2}^{e_1}, s_{2}^{e_2}) + \cdots + d_{eu}(s_{7}^{e_1}, s_{7}^{e_2})]$$ and

$$d_{shape-}(e_1, e_2) = \frac{1}{7}[d_{eu}(s_{1}^{e_1}, s_{7}^{e_2}) + d_{eu}(s_{2}^{e_1}, s_{6}^{e_2}) + \cdots + d_{eu}(s_{7}^{e_1}, s_{1}^{e_2})].$$

Figures 4.6f and 4.6g depict different stages of the calculation of $d_{shape}$. In 4.6f, vectors are drawn depicting $s_1$ through $s_7$ for each edge. Then, Figure 4.6g shows the comparison of these shape points. The differences in the shape points are given, and the average of these 7 distances make up the shape component of the edge distance, $d_{shape}$.

Now, with all three edge comparison components ($d_{loc}$, $d_{dist}$, and $d_{shape}$) in-hand, the final step is to combine them in a coherent manner. We must take care to scale the edge distance so edges that make up a large proportion of their glyphs are weighted more heavily. This gives way to an edge distance calculation that is robust to differences in small edges, but demands that large edges be similar. This is a favorable property since larger edges generally dominate the structure of a glyph, and to find similar glyphs, we seek similar dominating structures. The small edge distance down weighting also provides error-tolerance by preventing small
edges from drastically influencing the distance measure. The amount that a distance is down
weighted is based on the average proportion of the two glyphs that the edges \( e_1 \) and \( e_2 \) make up.

The distance measure between two edges \( d(e_1, e_2) \) is the final culmination of the three
comparison components and weighting results. It is given in Equation 4.7 below.

\[
d(e_1, e_2) = \frac{1}{2} \left[ \frac{|e_1|}{\sum_{i=1}^{|E_j|} |e_{i}^{E_j}|} + \frac{|e_2|}{\sum_{i=1}^{|E_j|} |e_{i}^{E_j}|} \right] \times \min \{ d_{loc}^+ + d_{dist}^- + d_{shape}^+, \; d_{loc}^- + d_{dist}^+ + d_{shape}^- \}, \tag{4.7}
\]

where we let \( e_{i}^{E_j} \) denote the \( i^{th} \) edge in \( E_j \), which is the set of edges comprising glyph \( j \). The
leading fraction in (4.7) is the weighting component, where \( |e_i| \) denotes the length of path
\( i \) and \( |E_j| \) is the total number of edges in glyph \( j \). The minimum statement on line 2 of
Equation 4.7 is the mechanism for choosing the direction of edge comparison that yields the
smallest distance measure.

Recall that we would like the edge distance calculation to relate back to fundamentals of
the graph edit distance. If we could force the edge distance to \( d(e_1, e_2) = 0 \), then the two
edges would match. We can think of the steps necessary to achieve this (in the sense of all
three comparison components) as the edits it would take to transition from one edge to the
other.

The endpoint location component \( d_{loc} \) emulates an edit operation that shifts an edge
in space. The cost of this edit is equivalent to the distance that an edge must travel to
align itself with its counterpart’s nearest endpoint. The straight-line distance component
\( d_{dist} \) measures the difference in displacement of the edge endpoints. This is comparable to
stretching or compressing edges as an edit operation. The final component \( d_{shape} \) represents
the amount that two paths have to be straightened, curved, or twisted in order to match each other.

### 4.4.2.4 Glyph Distance Measure from Edge Calculations

Combining edge distances to form a complete graph distance happens in two phases. First, we consider a method for handling potential differences in edge counts between two glyphs. Then, the optimal matching between the edge sets is chosen via evaluation of all available edge distance comparisons. The edge distance for the resulting matches are summed to form the final glyph distance measure.

To equalize the number of edges between two glyphs, dummy edges are added to the graph with smaller $|\mathcal{E}|$ until the number of edges is equal. These dummy paths do not have any physical structure, and thus cannot be compared with real edges in the usual way (using endpoints, shape points, etc.). Instead, the distance between a real path and a dummy path is assigned solely based on the length of the real path. For a real edge $e_r$ and a dummy edge $e_d$, we define the distance as

$$d(e_r, e_d) = \frac{1}{2} \left[ \frac{|e_r|}{\sum_{i=1}^{\mathcal{E}_1} |e_{r_i}|} + \frac{0}{\sum_{i=1}^{\mathcal{E}_2} |e_{d_i}|} \right] |e_r|^2 = \frac{1}{2} \left[ \frac{|e_r|^3}{\sum_{i=1}^{\mathcal{E}_1} |e_{r_i}|} \right].$$

This calculation uses the same weighting factor as in Equation 4.7 with $|e_2| = 0$ and the distance component set to the squared length of the real path, $e_r$. In the graph edit distance framework, the distance between a real edge and a dummy edge $d(e_r, e_d)$ directly relates to the cost of adding a new edge of a particular size.

Including the dummy edge comparisons, we make $\max(|\mathcal{E}_1|, |\mathcal{E}_2|)^2$ unique pairwise edge distance calculations for $\mathcal{G}_1$ and $\mathcal{G}_2$. During this process, every path in $\mathcal{G}_1$ has been matched with and compared to every path in $\mathcal{G}_2$. Like in the graph edit distance, the final glyph distance is the set of edit operations that minimize the total edit cost.
To calculate this minimal edit cost, we match the edges in $G_1$ to edges in $G_2$ in a $1-1$ fashion such that the total edge distance is as small as possible. The edge matching between glyphs is calculated by formulating the optimization problem as constrained minimization, which is solved via linear programming.

We complete Section 4.4.2 by giving a formal expression for the complete glyph distance measure. Define the indices $I^* = \{i_1^*, \ldots, i_{\text{max}\{|E_1|,|E_2|\}}^*\}$ to be the set of indices that reorder the edges of $G_2$ to reflect the optimal minimum distance matching with respect to the original ordering of $G_1$. The distance between the two glyphs is

$$D(G_1, G_2) = \sum_{i=1}^{\text{max}\{|E_1|,|E_2|\}} d(e_{i_1}^{E_1}, e_{i_2}^{E_2}).$$

(4.8)

### 4.4.3 Weighted Mean of Glyphs

The introduction to Section 4.4 explains the necessity for both a distance measure and a measure of center for creating a $K$-means type clustering algorithm. Section 4.4.2 outlines the procedure for calculating the distance measure. In order to simplify the cluster mean calculation, it is sufficient to describe the methodology of taking the weighted mean of two glyphs. Once this procedure has been established, it can be applied in sequence to produce the mean of a larger set of glyphs by iteratively taking weighted means with decreasing weight on each newly introduced glyph.

Like the glyph distance measure relied on combining individual edge distances, the glyph mean calculation will rely on combining individually calculated edge means. Prior to the mean calculation of two glyphs, it is assumed that the edges have been matched as described in Section 4.4.2.4.

In cases that do not require dummy edges, an edge is completely characterized by its end points, shape points, and length. Similarly, the weighted mean of two edges is completely
characterized by the weighted mean of each of these components. These weighted mean calculations are immediate, because of the edge matching and comparison direction obtained previously.

Figures 4.7a-4.7f demonstrate this weighted mean calculation in the simple situation in which both glyphs have one edge.

Figures 4.7g-4.7l show the weighted mean of the two glyphs used in Figure 4.5. Note the dummy edge on the left side of the blue ‘f.’ Although the images do not show it, the length of that edge is weighted between the length of the blue edge it represents and 0.

In the case when one of the edges in the mean calculation is a dummy edge the mean must be calculated slightly differently. A real edge that is matched with a dummy edge lends its spatial components to the mean edge, so the resulting mean has the same end points and shape points as the real edge. However, the length of the resulting edge is a weighted average of the length of the real edge and 0.

Due to repeated down weighting of an edge across a set of glyphs, it is possible that the length of certain edges get very small, below 1, for example. The last part of taking the weighted mean is to delete any edge from the mean that is shorter than 1 pixel long. This prevents the number of edges in the mean graph from being equal to the largest glyph in the set that the mean is being calculated on. This pruning keeps computation times down and makes the mean a more realistic representation of the entire set of glyphs.

Figures 4.7s-4.7x show the transition of the weighted mean where early in the transition (4.7s-4.7u) the small top left edge is present in the mean, but later in the transition (4.7v-4.7x) the edge has been pruned as it is shorter than 1 pixel long due to down weighting.

With this above process, we can calculate the mean of two glyphs for any value of the weight. The iterative process for calculating the mean of a larger set of glyphs \( \mathcal{G} = \{ \mathcal{G}_1, \ldots, \mathcal{G}_n \} \) is as follows. Define a function to calculate the weighted mean \( w'(\mathcal{G}_1, \mathcal{G}_2, p) \) where \( p \) is the \([0, 1]\) weight put on \( \mathcal{G}_1 \). Then, the mean of the first two elements of \( \mathcal{G} \) is
Figure 4.6: Edge distance measure calculation companion figures.
Figure 4.7: Figures showing the transition of the weighted mean of two letters. 4.7a shows the mean weighted completely on the ‘n’, then left to right transitions from ‘n’ to ‘v’, while 4.7f shows the mean weighted completely on the ‘v’.
\( m_2 = w'(G_1, G_2, 0.5) \). To incorporate the third element into the mean, we must take care that the included glyphs are each weighted evenly in \( m_3 \), so \( m_3 = w'(m_2, G_3, \frac{2}{3}) \). For each subsequent mean calculation the update formula is \( m_i = w'(m_{i-1}, G_i, \frac{i-1}{i}) \). Applying this update formula iteratively across a set of glyphs provides a formula for calculating a group mean.

While this glyph set mean does a good job of summarizing a set of similar glyphs, it has some properties that are sub-optimal. First, and most egregious, the value of the set mean depends on the order that the glyphs are introduced with the weighted mean. This is due to the path matching part of the algorithm, and is unavoidable. Second, the mean of two glyphs is only equi-distant from both if they have the same number of paths. This leads to a mean that, in the case of a set of glyphs with differing edge counts, may not minimize the sum of the squared distances to those edges. Both of these properties apply for means of more standard data types like vectors, but do not follow to the glyph mean presented here. Despite these weaknesses, we will see that as the elements in a cluster start to look similar, the mean is a good template for the glyphs it summarizes.

4.4.4 \( K \)-means Type Algorithm

With a method of computing a distance measure and a calculation of the mean of a set of letters, a \( K \)-means type algorithm can be created. Before actually implementing the \( K \)-means framework, we must investigate one more difficulty that comes with this data. Although we have attempted to construct an error tolerant distance metric, the \( K \)-means algorithm has no way to deal with outliers, and the presence of outliers can have undesirable effects on cluster groupings (Tseng and Wong, 2005). The glyphs extracted by handwriter are certainly very diverse, and there will be characters that are very different from all of the others. Additionally, there may be sets of glyphs that are not frequent enough to form a cluster, but are not close to any of the cluster centers. These points can be considered
outliers, as there is no cluster that adequately summarizes that glyph group. Figure 4.8 shows two examples of unconventional glyph types that will be far from every group mean. Figure 4.8a is a large complex glyph that is not repeatable, while Figure 4.8b is a simple, but uncommon structure in handwriting. When K-means encounters an observation very far from any cluster it may either pull the center of a cluster toward it, thereby mistakenly summarizing the rest of the cluster poorly, or it may create its own cluster with only that point. Neither of these options is a desirable result in clustering.

To accommodate these outliers, we implement the method of (Gan and Ng, 2017). They use a cutoff distance derived from the average distance from each observation to their cluster mean to decide which observations are too far from any center. These observations are grouped as outliers, meaning they will not contribute to any cluster centers and as a result will not bias cluster centers. The number of outliers is capped by a parameter, but is learned based on the data. In the case of no outliers the modified algorithm simplifies to standard K-means.

The clustering framework of (Lloyd, 1982; Forgy, 1965), the simplest formulation of K-means, starts with a set of initial clusters and iterates through two steps until convergence.
First, each point in a dataset is assigned to the cluster whose mean is closest to it, then the cluster means are recalculated with all of the points that were just assigned to that cluster. These steps are both readily available for use on glyphs with the tools we introduced in Sections 4.4.2 and 4.4.3, but before implementing them we make a modification to improve performance.

Instead of using the cluster means calculated as described in Section 4.4.3, we increase the stability of the clustering by using the cluster exemplar rather than mean as the cluster center. An exemplar is defined as the point closest to the calculated cluster mean. This allows for small perturbations in the cluster mean to be largely ignored, so long as the same exemplar graph is closest to the mean. This change is most beneficial in clusters that contain a more variable set of glyphs.

The outlier removing modifications and use of exemplar rather than mean leave the Forgy-Lloyd algorithm skeleton largely in tact, but add a couple of steps to each iteration. The complete algorithm is now introduced.

Consider a set of glyphs $\mathbf{X} = \{X_1, \ldots, X_n\}$, a pre-specified number of clusters $K$, a set of initial cluster exemplars $C = \{C_1, \ldots, C_K\}$, a maximum number of outliers $n_o$, and a parameter $\gamma$ controlling the allowable distance from a center to an observation before it is classified as an outlier. We also define the variable $T_o$ to be the distance threshold by which outliers are determined. $T_o$ will be learned and adjusted within each iteration of the algorithm, but the initial value of $T_o = \infty$. The cluster assignment of observation $i$ is $\varphi_i$, and $\varphi_i = -1$ denotes that the $i^{th}$ observation is an outlier.

An outlier tolerant $K$-means type algorithm for clustering characters proceeds as follows.

- Iterate through the following steps until the cluster assignments do not change:
1. Assign each glyph $X_i \in \mathbf{X}$ to the cluster whose exemplar is nearest to $X_i$ with regard to the distance measure in Section 4.4.2.

$$[\varphi_i = \arg\min_{j=1,\ldots,K} \{D(X_i, C_j)\}]$$

2. Remove the cluster assignment of $X_i$ and call it an outlier (set $\varphi_i = -1$) if its distance to its cluster exemplar ($D(X_i, C_{\varphi_i})$) is larger than $T_o$. If more than $n_o$ observations would be outliers, only move the observations with the $n_o$ largest distances.

3. Calculate the mean of the glyphs in each cluster. Update $C_1, \ldots, C_K$ to be the $X_i$ closest to each calculated mean. Outlier glyphs contribute to none of the means.

$$[C_j = \arg\min_{X_i | \varphi_i = j} D(X_i, \text{Mean}(\{X_m | \varphi_m = j\}))]$$

4. Update $T_o$ with

$$T_o = \frac{\gamma}{n - \sum_{i=1}^{n} I(\varphi_i = -1) \sum_{i=1}^{n} I(\varphi_i \neq -1) D(X_i, C_{\varphi_i})}$$

When the algorithm has converged, each glyph is grouped into the cluster with the nearest mean except a subset of the observations, which are in the outlier cluster. Like all versions of $K$-means, only a local optimum is guaranteed. To try to reach the global optimum, running the algorithm with different starting values is suggested.

The input parameters $n_o$ and $\gamma$ are set with the default values of $n_o = .25n$ and $\gamma = 3$, which allows for up to 25% of the observations to be called outliers and requires a distance 3 times the average within cluster distance to call an observation an outlier. For this algorithm, $K$ is fixed before running, and throughout the paper, $K$ will be assumed to be known, so no $K$ selection criterion are provided.
4.5 Applications

To demonstrate the use of our distance measure and clustering algorithm we begin by applying our clustering methodology to the MNIST digits dataset. The simple character types found in these clean digits provide a convenient platform to investigate behaviors of the algorithm and serve as a spring board into the fundamental handwritten text application. Then, using the CVL dataset we process, cluster, and conduct a full pipeline writer identification analysis.

4.5.1 MNIST Digits

In total, the MNIST dataset consists of 70000 handwritten numerical digits saved as images. The samples are divided into a training set with 60000 samples and a test set with 10000 samples. In this paper, we limit analyses to the test set. The images in this dataset are all 28x28 pixels, and the size of the original written digits have already been properly normalized to fit the common image dimensions. This results in a set of very clean images that are passed to the handwriter package for glyph extraction. There are only 10 different numerical digits and they are written individually without any connections, so the glyphs are relatively simple and in this data, correspond to a single digit. All of these things combine to make the MNIST dataset a natural starting point for investigating results of our clustering algorithm.

While this is a perfectly respectable application of our algorithm, it is significantly simpler than the case of general writing. When traditional handwriting is clustered, the clusters must encompass a wider range of glyph types, so a cluster for handwriting might contain ‘glyphs with tails and a broken loop on top,’ which could describe a variety of letters under a variety of writing styles. In contrast, the cluster centers resulting from clustering MNIST more closely resemble specific 0-9 digits.
Digit labels are available for the MNIST data, so we begin by looking at how the partitions compare to the known digits labels. We pay special attention to digits that have more than one common styles (such as the 2 with and without a loop) and observe the separation of these styles within and across clusters. Then, we compare our cluster assignments with the deterministic grouping methods discussed in Section 4.3. Since there are so many groups formed by the deterministic methods, we focus on the most populated and least populated groups in our analysis. We conclude by discussing behaviors and implications that will become important in the handwritten text context, and places where our method performs poorly on the digits.

4.5.1.1 Digit Clusters

Figure 4.9 shows the cluster centers and a visual summary of glyph assignments that result from dynamically assigning the MNIST digits into 15 clusters. For each of 0 – 9 there is a cluster that is largely dominated by that digit, and some digits span multiple clusters. For a closer inspection of the contents of the clusters, see Figure 4.10 which shows the glyphs within each cluster that have ground truth in each of the respective numeric digits. The columns represent the calculated clusters, and the rows represent the true digit label of each glyph. Darker marking indicate that more glyphs fall into that cell, but note that differences in uncommon cells may not be visible as we have forced every cell to be dark enough to be seen, regardless of its population.

Figure 4.9a and 4.9b represent sets of 0s and 1s. The 1s cluster also contains digits that are not 1s. Figure 4.10 shows that these digits (images in the 2nd column that are not in the 1s row) are generally not identical to 1s, but have one path and are mostly vertical.

Figure 4.9c contains 2s with a loop at the turnaround point, but excludes 2s without a loop. The single path 2s are mostly sorted into the cluster in Figure 4.9h, with the 7s. There is also a set of 2s that may have intended to loop, but did not actually loop, so there
Figure 4.9: Cluster exemplars for each of the 15 clusters estimated on the MNIST dataset in red. Glyphs assigned to each cluster over-plotted in light grey behind the exemplars. The first 10 correspond largely to single digits, followed by 5 glyph sets that allow for deviation in style from the standard digit forms. The number of glyphs in each cluster is listed next to the subfigure label.

is just a segment where the loop should have been; these characters tend to be sorted in the 4s cluster, shown in Figure 4.9e.

Figure 4.9d’s cluster contains 3s that have a breaking point in the middle and a stem at the turnaround point, where as 3s that were written as one line without any retrace are placed with the 7s in Figure 4.9h. Most of the 4s are in the cluster depicted in 4.9e, with a small portion of very slanted 4s in 4.9k, and 4s with a closed top in 4.9j.
Figure 4.10: This figure shows, for each ground truth digit type, how glyphs are distributed into our clusters. Each row corresponds to a known digit type, and the columns represent our clusters. Each cell is a set of digits plotted overlaying each other.

The 5s are mostly in the 4.9f cluster, but a handful get grouped with 3s if they have an extra appendage at the corner before the loop, and some with the 1s if they are more vertical. The 6s are separated into three main groups. The largest is cluster 4.9g, which contains 6s with long stems and loops that begin near the lowest point of the character. The two smaller sets of 6s are in the cluster of 4.9n which contains 6s with shorter stems and loops that begin closer to the midpoint of the letter, and either cluster 4.9b or 4.9d, where 6s without loops are sorted depending on their path’s shape.

The 7s are almost all put into the cluster in Figure 4.9h, with the small number of 7s that have a crossing being grouped mostly with the 3s or 4s. The 8s are interesting, because
they are divided almost equally into three clusters. The first type of 8, in Figure 4.9i, are those with an intersection of two loops that happens basically on a single point, maybe with a short neck in between. The second cluster of 8s, Figure 4.9o, contains 8s that have a wider separation between loops. The 8s in Figure 4.9m are the most different from the other two groups. These 8s all have a broken top loop, usually on the top right hand side.

The final digit type is 9s, of which most are a simple top loop and descending stem. These glyphs are grouped into the cluster shown in Figure 4.9j. The largest set of different 9s are those in which the top loop does not connect all the way back to the stem. Since we are clustering largely based on graph structure, it is not surprising that these disconnected 9s are grouped with the 4s in Figure 4.9e.

The number of elements in each cluster are provided in the captions of Figure 4.9. The clusters shown in Figures 4.9b, 4.9e, and 4.9h are the most populated groups.

Upon analyzing the clustering results for the MNIST digits, there are two key features of the clustering that may be undesirable. First, the handling of loops could be better. Looking at the 4.9g and 4.9n clusters, the only difference is the orientation of the loop of the 6s. The structure of the glyphs is otherwise very similar and it would be nice to allow for comparisons between these digits. The path in 4.9g is generally above and to the right of the expected eighth points line, while in 4.9n the path points are generally to the right and below the originating point. This is not a structural difference, but a difference in path orientation. A possible fix would be to align the edges before comparing eighth points, which would work nicely for loops, but would have to be reconciled with paths that are almost full loops, but do not originate at the same point. No obvious solution presents itself for the loop alignment problem.

Next, it seems that single path glyphs may carry a higher distance penalty than their more complex counterparts. This behavior can be seen by examining glyphs in the outlier cluster in Figure 4.10. There is one 0 there that had the bottom cut off, which makes sense
as an outlier, but the other outlier digits are all single path 2s. While these 2s do not fit obviously in another cluster, it is surprising that they are not grouped into the Figure 4.9b cluster with the 1s, or more likely with the 7s in 4.9h. This propensity for single path glyphs as outliers will be revisited in the handwriting application.

4.5.1.2 Comparison of Deterministic and Dynamic Grouping Methods

When handwriter processes the MNIST digits, it outputs each glyph along with its membership assignments under each of the two deterministic grouping methods discussed in Section 4.3. We just presented dynamic clusters assignments obtained from our algorithm, and will now compare those learned groupings to the fixed groupings. The first and most noticeable difference is the number of groups and the distribution of digits within those groups. We fixed $K = 15$ for our algorithm, and the clusters were learned accordingly. The deterministic methods do not have a set number of groups, instead, the number is determined by the diversity of the characters.

Recall that under the adjacency grouping method, glyphs are placed in groups together based on the entirety of their adjacency matrices. For the MNIST digits dataset the number of different adjacency groupings is only 71. Compared to the 1791 adjacency groups of the CVL handwriting dataset, 71 is quite small, and is due entirely to the simplicity of the MNIST digits relative to parsed handwriting. Together, the groupings defined by the adjacency matrices of Figure 4.11a and 4.11b account for 6009 of the 10000 digits. Interestingly, these are the same adjacency matrices as those of Figure 4.4 which are the two most common deterministic groups for the CVL dataset. Additionally, 30 of the 71 adjacency groups contain only a single element. This combination of high count clusters, which are hard to compare within because they are so diverse, and low count clusters, which are hard to compare at all because of lack of information, make analysis based on this grouping potentially difficult.
The connectivity grouping, which will always have fewer groups than the adjacency grouping, has 35 groups of MNIST digits. We noted previously that which clusters occur most frequently will remain unchanged between the adjacency grouping and the connectivity grouping, so 6009 digits still fall into the most frequent connectivity groups. In fact, all four of the most common adjacency groupings coincide with the four most popular connectivity groupings. The difference in the two deterministic groupings lies in the more complex glyphs. Only 10 of the connectivity clusters contain a single element, which means that the 30 singleton clusters in the adjacency grouping merged into more populated groups for the connectivity groups.

The four most common deterministic groupings are shown in Figure 4.11. The first set, in Figure 4.11a, holds 39% of the digits, and every element in this group is a single path glyph with two end points. This cluster consists of most of the 1s, unlooped 2s, 3s without a retrace branch, most of the 5s, 6s with an unfinished loop, most of the 7s, unconnected 9s, and unconnected 0s. Figure 4.11b contains characters that have three paths which connect
at a single middle point. This cluster is made up of mostly 2s with a retrace intersection but no loop, 3s with a stem at their turning point, and 4s without a full cross. It also contains a smaller, but still present, amount of 5s with a stem at their sharp turnaround point and 7s with a partial cross. Figure 4.11c has a connect code of “12”, which corresponds to any glyph with a loop and a stem. Expectedly, this set is made up of largely 6s and 9s, with a smaller amount of 8s that did not connect one of their loops. The fourth, and easiest, of the most frequent deterministic groups contains characters with a single path, but no intersections or end points. This can only happen in perfect loops, so this cluster contains all 0s.

In the handwriting analysis context the large clusters are difficult to work with, because the characters that go into them are so diverse. The comparisons made within these groups are not always meaningful. For example, consider comparing measurements taken on the elements of Figure 4.11a where we had just about every digit represented as a member. Similar issues would likely arise with with Figure 4.11b and 4.11c groups. Features recorded on the Figure 4.11d glyphs, such as slant or aspect ratio, are likely informative measurements, because they are comparing similarly shaped objects.

Additionally, on the opposite end of the spectrum, the many small singleton clusters that we observe are not informative. Many of the small deterministic clusters are formed due to practically insignificant differences in the writing of complex glyphs, making the glyphs in these groups more about random perturbations in writing and less about how a writer forms their writing. These small and unrepeatable groups contain very little information while the large popular groups contain such diverse collections that meaningful comparisons within cluster are diluted and much of the repeatable signal in writing is left uncaptured.

### 4.5.2 Handwriting Clustering

For the seminal handwriting example of this paper, we use a subset of the CVL handwriting dataset (Kleber et al., 2013) to perform a writership analysis on a set of documents
with known origins. The CVL dataset includes 310 writers in total, but we use only the 27 writers who wrote all 7 of the CVL writing prompts. Unfortunately, one of the prompts is in German, and although German writing uses roman letters, we remove those samples from consideration for our application. The resulting data set is a collection of 162 writing samples by 27 writers. Immediately, we set aside the fourth writing sample of each writer to create a holdout set of documents. These will act as questioned documents in the writer identification exercise to come.

Before processing, the documents are cropped to the smallest bounding box containing writing and down sampled to contain a maximum of 1750000 pixels without changing the aspect ration of the document. Then, handwriter is used to extract glyphs from each of the documents. The resulting dataset contains 52451 glyphs.

For computational purposes, we use glyphs from one document from each writer to perform clustering, effectively creating cluster templates from a diverse collection of writing. After the clusters are created, the glyphs that were not used during clustering are sorted into groups based on their similarity to cluster exemplars, in the sense of the distance measure developed in Section 4.4.2.

4.5.2.1 Cluster Assignments

We group the 52451 glyphs into 40 clusters. 40 was chosen because it walks the line of being able to accommodate the complexity of handwriting characters without having groups that are rarely observed. Supplementary materials Section 4.7.2 provides results for the analysis provided here for other cluster counts with very similar results. The outcomes presented here are not strictly dependent on the number of clusters chosen.

Figure 4.12 shows the set of exemplars resulting from the clustering ordered based on the number of observations in the clusters. In addition, Figure 4.13 shows the calculated mean for the clusters in the same order. While the exemplars are used for all functional purposes,
the combination of these two measures of center will be used to investigate behaviors of the clusters built from handwritten documents. For brevity, we will not discuss each of the 40 clusters individually. Instead, we note some general trends and focus on specific clusters of interest.

Most of the highly populated clusters are very compact, and the exemplars are good descriptors for the rest of the glyphs. Many of these clusters describe very simply glyphs like vertical segments and single path glyphs like those that represent ‘c’s. The first cluster, shown as Figure 4.12a, is interesting, in that the exemplar is very small, but the elements in the background are larger. A closer inspection reveals that the glyphs in this cluster all have crosses, but the edges leading out of the intersection extend out in different directions. This leads to matching edges that are dissimilar, and means that are closer to the center than most observations. The compact mean means that the exemplar for the cluster will also be compact.

The clusters that are not made up of glyphs with very few paths form clusters that are more variable than the compact clusters. The more complex glyphs have more edges, so they have more natural variability. This can be seen in the less populated clusters like 4.12ab and 4.12an. Examining the mean of the clusters, rather than the exemplar, can reveal patterns in the clusters of complex glyphs. Figure 4.13al, for example, reveals that in the complex cluster shown in Figure 4.12al the common component is the loop at the bottom of the glyph. The dots above that loop signify that there are more edges in the glyph, but they are not important in grouping glyphs into that cluster. Figure 4.13an reveals a similar characteristic for the cluster in 4.12an, which has a loop on the bottom left and the top right. The exemplar in Figure 4.12an shows an example of a character with these loops.

A large number of the clusters align directly with known letter types. Cluster 4.12h is a group a group of ‘u’s, cluster 4.12i contains ‘c’s, cluster 4.12m is largely ‘f’s, cluster 4.12r contains ‘n’s, clusters 4.12s, 4.12w, and 4.12aa are groups of ‘e’s, cluster 4.12ae contains
Figure 4.12: Each subplot represents 1 of the 40 clusters obtained for the 162 CVL documents. The black background figures are the glyphs which were grouped into the cluster. The exemplars are shown in red on top of the glyphs.
Figure 4.13: These plots show the raw calculated mean for each of the clusters. The panes are ordered to match the exemplars in Figure 4.12. These means show the important characteristics for each cluster with the full drawn edges. The small dots represent paths that often differ within the cluster.
largely ‘s’s, and finally clusters 4.12ac and 4.12aj contain ‘o’s. If all of a character type were grouped into a single cluster, then that cluster would represent not a characteristic of handwriting, but the frequency in which that letter appears in words. This would make it mostly useless in our writer identification setting. Luckily, even though these clusters contain mostly single character types, they do not contain all characters of the type, so these clusters can still be informative for writer identification.

The $K$-means algorithm discussed in Section 4.4 was altered to allow for the detection and isolation of glyphs that are not particularly close to any cluster exemplar. We expected that large, complex glyphs would be called outliers. This is true to an extent, but a large proportion of the outliers are simple graphs that do not line up with any of the existing cluster types. Instead of being outliers, the complicated glyphs get a cluster to accommodate them. The cluster shown in Figure 4.12ab represents an unexpected, but not unwelcome cluster type. Looking at the raw weighted mean in Figure 4.13ab, we see that it is essentially a large set of points scattered around the character space. This cluster absorbs the glyphs with a large number of edges, making it very diverse.

The outlier cluster is mostly made up of large simple glyphs, which certain types of writers are more likely to produce. This makes the frequency with which outlier glyphs are observed a potentially useful piece of information for writer identification. Thus, instead of throwing it out, we include it as the 41st cluster in the analysis that follows. This decision is justified by separation in the writer specific posterior density of outlier cluster membership probabilities, shown in the bottom panel of Figure 4.15.

4.5.2.2 Writer Identification

Recall that there are 135 documents in the training data set (5 prompts, 27 writers), and 27 documents in the holdout dataset (1 prompt, 27 writers). We will index model
elements from 1 to 27, but for the sake of visuals and tables, it is worth noting that the writer identification names from CVL are not consecutive and span the range of 1 to 50.

With cluster examplars established in Section 4.5.2.1 serving as glyph templates, CVL writers are characterized by the way in which their glyphs are distributed into the clusters. For each document, we tally the number of glyphs that fall into each of the 40 clusters and the outlier cluster. A Bayesian hierarchical model for this compositional count data is developed to model the data. The desired model result is a probabilistic conclusion about writer identification for each of the held out documents. We compare model results from our dynamic cluster groupings with results using the same model, but with deterministic groupings discussed in Section 4.3.

Model Formulation  Let \( Y_{w(d),c} \) be the number of glyphs assigned to cluster \( c \), for \( c = \{1, \ldots, K+1\} \) (where \( K+1 \) arises from the fixed number of clusters \( K \) plus one group of outliers), for document \( d \) nested in writer \( w \). Then, \( Y_{w(d)} = \{Y_{w(d),1}, \ldots, Y_{w(d),K}, Y_{w(d),K+1}\} \) characterizes the number of glyphs assigned to each cluster for document \( w(d) \). We consider these cluster assignments as samples from a multinomial distribution. Each writer’s glyph emissions are modeled by their own multinomial distribution using a different set of cluster probabilities for every writer. Let \( \pi_w \) denote the simplex of size \( K + 1 \) for writer \( w \). The
simplices $\pi_w$ are modeled within each writer with the Dirichlet distribution. The $K + 1$ hyperparameters of the Dirichlet distribution, denoted here as a vector $\alpha = \{\alpha_1, \ldots, \alpha_{K+1}\}$, are given independent gamma priors. The mathematical formulation of the model is

$$
Y_{w(d)} \overset{\text{ind}}{\sim} \text{Multi}(\pi_w),
$$

$$
\pi_w \overset{\text{ind}}{\sim} \text{Dirichlet}(\alpha),
$$

$$
\alpha_c \overset{iid}{\sim} \text{Gamma}(a = 3, b = 0.25),
$$

(4.9)

where the gamma distribution is parameterized such that $E(\alpha_c) = \frac{a}{b}$.

Markov chain Monte Carlo estimates of the posteriors for the $\alpha$ and $\pi_w$ parameter vectors are obtained using the rstan R package (Stan Development Team, 2018). In the analysis that follows 4500 iterations were used with a burn-in period of 2500. Then, 1000 of the remaining 2000 samples are used for plotting and to calculate posterior predictive quantities (discussed below). Denote these samples as $\alpha^{(m)}$ and $\pi_w^{(m)}$, for $m = 1, \ldots, M (= 1000)$.

Figure 4.15 shows posterior densities of the $\pi_{w,c}$ components for all writers and seven different clusters. Within a pane of the figure, the different density curves represent different writers from the CVL database, $w = \{1, \ldots, 27\}$. The curves can be conceptualized as the posterior cluster inclusion rate for glyphs from each writer. The separation of the density curves across the writers reinforces our assertion that a writer’s style can be sufficiently captured and characterized by the rate in which their glyphs are assigned to various clusters. The top three panes of Figure 4.15 present the cluster proportion estimates for the three largest clusters. The next three panes, labelled 10, 11, and 12, show the same for less frequently occurring clusters. While this plot shows only seven clusters, the separation trend shown here exists across all clusters. Keep in mind that when a posterior probability vector $\pi_w$ is sampled it must sum to one, and that there are 41 elements of the vector. So, a
Figure 4.15: This plot shows densities of the posterior samples for $\pi$. The different panes represent different clusters, and the overlapping densities represent different writers’ assignment probabilities to that cluster.

A writer that has an unusually high probability of inclusion to one cluster must necessarily have lower inclusion probabilities to others. Although it is not shown here, clusters with lower populations will reflect posterior $\pi_w$ densities with increased overlap as they are pushed towards 0.
As discussed previously, the bottom pane of Figure 4.15 shows the posterior samples for the probability of observing outlier glyphs for each writer. Overlap of the densities in the outlier pane resemble those in the traditional cluster panes. The separation of writers observed in the outlier densities is again a suggestion that there is identifying information carried in the frequency of outliers.

**Prediction** With posterior samples for the $\alpha$ and $\pi_w$ parameters in hand, we can use the model defined in equation 4.9 to evaluate a holdout document written by unknown writer, $w^*$. The holdout document is processed by the *handwriter* R package to extract glyphs. Every glyph is assigned to a cluster based on which exemplar it is closest to. We count the number of glyphs in each cluster to obtain the multinomial response vector for the new document $Y_w^* = \{Y_w^*1, \ldots Y_w^*K+1\}$. We drop the document nested within writer notation $w(d)$ here since we consider only one questioned document at a time.

We can now use the posterior predictive distribution to retrieve posterior probabilities of writership for each of the 27 writers in the training data. We do this by evaluating the likelihood of the vector $Y_w^*$ under the multinomial distribution for each writer using their respective posterior probability vectors $\pi_w$ as parameters. For any particular writer in the training data $w'$ the posterior probability of writership for writer $w'$ on the new document $Y_w^*$ is

$$p_{w'} = P(w^* = w'|Y_w^*, \ldots)$$

$$\propto P(Y_w^*|w^* = w', \ldots)P(w^* = w'|\ldots)$$

$$\propto P(Y_w^*|w^* = w', \ldots)$$

$$= P(Y_w^*|\pi_{w'})$$

$$= \text{Multi}(Y_w^*; \pi_{w'}).$$
A density for \( p_{w'} \) can be obtained with the MCMC samples of the \( \pi_w \)'s with the formula

\[
p_{w'}^{(m)} = \frac{\text{Multi}(Y^{*}\pi_{w'}^{(m)})}{\sum_{w=1}^{27} \text{Multi}(Y^{*}\pi_{wi}^{(m)})}.
\]  

(4.10)

Performing this calculation for every known writer \( w = \{1, \ldots, 27\} \) yields the probability vector

\[
\bar{p}^{(m)} = \{p_{w_1}^{(m)}, \ldots, p_{w_{27}}^{(m)}\}.
\]  

(4.11)

then we summarize the probability of writership for writers over all mcmc samples as

\[
\bar{\bar{p}} = \{\bar{p}_{w_1}, \ldots, \bar{p}_{w_{27}}\},
\]  

(4.12)

where \( \bar{p}_{w_i} = \frac{1}{M} \sum_{m=1}^{M} p_{w_i}^{(m)} \). In addition to the mean, we calculate lower and upper 90% credible bounds for the \( p_{w_i} \) posterior probabilities.

We apply this posterior predictive procedure to each of the 27 holdout documents.

Figure 4.16 provides a graphical summary of results for the holdout documents using our estimated clustering for glyphs. We see that the posterior predictive analysis places most (more than 0.5) of the average posterior probability on the true writer for all but one of the questioned documents. The holdout document written by writer 42 only places 0.182 average probability on the true author, while writer 23 gets most of the rest. In the two cells with non-zero probabilities in the questioned document 42 row, the small bands below the number represent the 90% credible intervals for our prediction. Despite the incorrect prediction, the confidence bands in this row are wide for both highlighted cells, indicating the model model acknowledges the uncertainty associated with the questioned document for writer 42.
Figures 4.17 and 4.18 reflect similar information as in Figure 4.16, but the modeling results are products of the two deterministic glyph grouping methods described in Section 4.3. Compared to the cluster grouping results of Figure 4.16, it is easy to see that there is much more posterior probability assigned off-diagonal when the deterministic groupings are used. All three grouping methods do well on holdout documents for writers 6 – 22 and 25. These writers have unique writing styles and are relatively easy to separate out regardless of glyph grouping method. It is the less easily separated writers that the deterministic groupings struggle to identify.

Under the adjacency grouping method (Figure 4.17) the model places more than 0.5 of the average probabilities on the true writer for only 23 of the 27 holdout documents. That number degrades for the connectivity grouping results (Figure 4.18) where only 20 of the 27 true writers are properly identified. Additionally, we should note that in a forensic statistics context, one would never use the cutoff of 0.5 probability for actual evidence as corresponding conclusions would be fraught with false positives. Many of the on-diagonal cells in both the deterministic grouping results reflect probabilities between 0.5 – 0.8, in which case we would not be comfortable making an identification. Neither of the deterministic grouping methods do particularly well with the questioned document for writer 42, which is the one that the cluster method struggles to identify.

In the supplementary material, Section 4.7.2, we provide figures similar to Figure 4.16, but when using $K = 30$ and $K = 60$ to cluster instead of 40. Both of these clusters also fail to correctly identify writer 42 as the writer of their questioned document. In each of these examples the held out questioned document is the same, so it is possible that the held out document simply does not line up with the documents used in the training set.
Figure 4.16: Results of the posterior predictive analysis defined in equations (4.10) - (4.12) for each of the 27 holdout documents in which glyphs were grouped by our clustering algorithm. Columns represent known writers. There is one row to summarize each questioned document analysis with true writer labeled on the left-hand side. Cells in the row are colored and labeled by elements of their $\bar{p}$ vector defined in equation (4.12). Rows are independent and sum to one, as all of the posterior writership probability is spread across the writers. Diagonal cells indicate the correct identification for the questioned documents. A point for the average probability, and 90% credible interval band is plotted below the cell label. Cell contents are omitted for any cell whose average probability rounds to 0 in three decimal places.
Figure 4.17: Results of the posterior predictive analysis defined in equations (4.10) - (4.12) for each of the 27 holdout documents using the deterministic adjacency grouping of glyphs. Columns represent known writers. There is one row to summarize each questioned document analysis with true writer labeled on the left-hand side. Cells in the row are colored and labeled by elements of their $\bar{p}$ vector defined in equation (4.12). Rows are independent and sum to one, as all of the posterior writership probability is spread across the writers. Diagonal cells indicate the correct identification for the questioned documents. A point for the average probability, and 90% credible interval band is plotted below the cell label. Cell contents are omitted for any cell whose average probability rounds to 0 in three decimal places.
Figure 4.18: Results of the posterior predictive analysis defined in equations (4.10) - (4.12) for each of the 27 holdout documents using the deterministic connectivity grouping of glyphs. Columns represent known writers. There is one row to summarize each questioned document analysis with true writer labeled on the left-hand side. Cells in the row are colored and labeled by elements of their $\vec{p}$ vector defined in equation (4.12). Rows are independent and sum to one, as all of the posterior writership probability is spread across the writers. Diagonal cells indicate the correct identification for the questioned documents. A point for the average probability, and 90% credible interval band is plotted below the cell label. Cell contents are omitted for any cell whose average probability rounds to 0 in three decimal places.
4.6 Discussion

In this paper, we create a clustering algorithm for grouping small pieces of handwriting, and use an individual's propensity for creating glyphs that belong to particular clusters to identify the writer of questioned source documents. Development of the clustering algorithm hinges on defining measures of distance and center for handwriting graphs. Our glyph distance measure emulates the graph edit distance measure while leveraging additional attributes that the glyphs possess. The $K$-means type clustering algorithm is used to cluster the MNIST handwriting dataset, with results demonstrating positive and negative qualities of the algorithm. Then, on the CVL handwriting dataset, we create clusters for glyphs derived from handwriting samples and show that writers can be characterized and identified by examining the frequency with which they emit glyphs to the various clusters. Identification statements are made using posterior probability summaries from a multinomial-dirichlet hierarchical model. We compare these identification results to those obtained using two deterministic groupings of the glyphs, showing that our clusters assignments improve upon the easily available, but volatile deterministic assignments and constitute a great improvement in model results.

In addition to the clustering algorithm, we describe the procedure by which a document is processed and turned to data. The handwriter R package reads, processes, and parses the document into glyphs, the observations we work with during clustering. This processing takes the extremely complex data source that is a scanned handwritten document, and makes usable information readily available in the form of attributed graphs. The clustering and writer identification as we did it would not be possible without this extraction.

In our two applications, we discuss times when our clustering algorithm came to unexpected conclusions. Of particular interest to us are the seemingly inflated distance contributions associated with loops with different orientations, and the calculation of a robust
mean to serve as the cluster centers. No immediate solutions present themselves for either of these interest cases. The distance measure presented here could be modified to rotate loops for better comparisons, but it is difficult to modify the distance measure without negatively effecting outcomes for other comparison situations. For diverse clusters, the mean tends to shrink towards the centroid of the observations, and the order that the observations are introduced to the weighted mean calculation can have small yet meaningful impacts on the resulting measure of center. This issue is mitigated in this paper by using exemplars, rather than the raw weighted mean, as the measure of center for the $K$-means algorithm, but finding a way to calculate a stable mean for any cluster type could improve or simplify our algorithm.

As with every distance measure, the one presented in this paper favors certain properties over others in assessing similarity. It would be interesting, and easy, to implement distance measures that prioritize other features of handwriting. For example, distance measures that do not take edge shape into account would prioritize edge location more heavily. On the other hand, discounting the location of an edge within a glyph would lead to a measure that cares only about how similar the edge shapes are and not at all about where the edges lie in space. The distance measure that we implement is used because it leans evenly on many types of distance without over emphasizing any specific aspects of the glyphs.

4.7 Supplementary Materials

4.7.1 Path Finding for Glyph Edge Identification

To create a traversable object from the thinned image, each pixel in the skeleton is considered the vertex of a new graph. To form the edges of this new graph, each pixel’s vertex is connected to vertices corresponding to pixels that are adjacent to the original pixel, with the exception that if there is a 4-neighborhood adjacent pixel then no 8-neighborhood
pixel adjacent to that 4-neighborhood neighbor is considered connected to the original. This exception is to make sure that a step is not taken diagonally if moving over and up or up and over would have led to the same movement.

With this traversable object, finding $\mathcal{E}^*$ is simplified to finding paths between the $\mathcal{V}^*$ pixels that don’t use any other pixel in $\mathcal{V}^*$. *handwriter* uses the *igraph* R package for path finding (Csardi and Nepusz, 2006). Care must be taken in path finding, as many vertices have multiple connecting paths. *handwriter* solves this problem by breaking a path after using it, and searching again for a path between the same two vertices. Additionally, loops exist in writing which start and end at the same node, so the path finding search must take care to look for edges that start and end at the same vertex. $\mathcal{E}^*$ is made up of the entire set of edges from one element of $\mathcal{V}^*$ to another. Like the attributed vertices, the pixel path traversed by an edge is kept as an edge attribute.

### 4.7.2 Writer Identification for Other $K$

The number of clusters used in the writer identification was chosen because it seemed like a good compromise between flexibility to model various character types without getting overly complicated or having very rare clusters. This section is included to verify that the writer identification results are robust to the number of clusters used. While the accuracy does downgrade slightly when moving down to $K = 30$, for $K = 60$ the results are very similar to the $K = 40$ results presented in full in Section 4.5.2.2.
Figure 4.19: This plot shows the results of predicting the writer for 27 different questioned documents using our clustering algorithm as the grouping mechanism with $K = 30$. The rows represent questioned documents. Columns represent writers. The diagonal cells represent the correct identification for the questioned documents.
Figure 4.20: This plot shows the results of predicting the writer for 27 different questioned documents using our clustering algorithm as the grouping mechanism with $K = 60$. The rows represent questioned documents. Columns represent writers. The diagonal cells represent the correct identification for the questioned documents.
References


R Core Team (2018). *R: A Language and Environment for Statistical Computing*. Vienna, Austria.


CHAPTER 5. GENERAL CONCLUSION

In this collection of work, we have extended the $K$-means algorithm to situations that are outside the general scope of the original algorithm. The $K$-means algorithm and its results are accompanied by a set of implicit assumptions that reduce the range of viable application of $K$-means. These assumptions include homogeneity of cluster spread, uncorrelated data points within cluster assignments, and a similar number of observations in each cluster. Disregard for these requirements leads to unintuitive and potentially unusable results. To extend the usability of $K$-means we present a set of algorithms that alter the traditional use case of $K$-means in order to provide coherent clustering results in non-standard clustering settings.

In Chapter 1, we present an algorithm for estimating the number of clusters in a dataset while also including information from only relevant dimensions from the dataset. Variable selection within $K$ selection insures that the estimated number of clusters is based only on important information, and that it is not biased by noisy or redundant dimensions. We provided a large scale simulation study to assess the performance of the dataset both on choosing the number of clusters in a dataset and on the accuracy of the resulting clusters after variable selection. We apply our algorithm to a collection of handwritten digits, which were clustered coherently using only 5 of 28 principal components on the estimated 17 clusters.

In Chapter 2, we introduced the TiK-means algorithm as a modification of $K$-means. TiK-means, a Transformation infused $K$-means algorithm, can be applied to data with skewed cluster groups to recover underlying clusters that can remain hidden to clustering
algorithms that do not correct the skew. We presented a stable homogeneous algorithm along with a more volatile, but potentially rewarding non-homogeneous methodology. TiK-means is applied to a range of common classification datasets with non-spherical clusters with generally positive results. It compares favorably with model based clustering in most of the classification settings. We also use TiK-means to cluster gamma ray burst data, which has dramatically skewed groups. TiK-means estimated the number of groups in the data to be 5, which agrees with many recent analyses of the data, but goes against traditional wisdom. TiK-means was able to perform clustering on the entire dataset by estimating the skew, rather than using a simplification of the data that creates dimensions more appropriate for traditional clustering.

Chapter 3 extends $K$-means by introducing and entirely new data type to the clustering framework. Handwriting is parsed into small pieces, called glyphs, which are clustered into groups based on their structures and edge characteristics. The distance formula and a method for estimating the mean of a set of glyphs are both developed. With the resulting clusters it is possible to create a model for a writer based on their propensity for writing glyphs that are assigned to different clusters. Using this writer model, we create a hierarchical model that learns the writing style for 27 writers and is able to accurately identify the writer of documents that are of questioned source.
References


