Pontifícia Universidade Católica
do Rio de Janeiro

**Lucas Saadi Murtinho**

# Theoretical and experimental results in information-theoretic clustering

**Dissertação de Mestrado**

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Eduardo Sany Laber

Rio de Janeiro
April 2020

Pontifícia Universidade Católica
do Rio de Janeiro

**Lucas Saadi Murtinho**

# Theoretical and experimental results in information-theoretic clustering

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee.

**Prof. Eduardo Sany Laber**
Advisor
Departamento de Informática – PUC-Rio

**Prof. Marco Serpa Molinaro**
Departamento de Informática – PUC-Rio

**Prof. Thibaut Victor Gaston Vidal**
Departamento de Informática – PUC-Rio

Rio de Janeiro, April 7th, 2020

**Lucas Saadi Murtinho**

An Economics graduate from PUC-Rio (2004), Lucas also holds a master's degree on Management de la culture et des médias from the Institut d'études politiques de Paris – Sciences Po (2008). Before taking his master's on Computer Science at PUC-Rio, he worked for over ten years as a financial analyst at a fuel distribution company. He has also translated books and articles from English and French to Portuguese.



Bibliographic data

Murtinho, Lucas

Theoretical and experimental results in information-theoretic clustering / Lucas Saadi Murtinho; advisor: Eduardo Sany Laber. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2020.

v., 84 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Otimização e raciocínio automático – Teses. 3. Teoria da informação;. 4. Clusterização;. 5. Medidas de impureza;. 6. Entropia;. 7. Gini.. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To Martin and Laila, for sharing their dad
with study and research.

## Acknowledgments

I started my graduate studies with a keen interest in computer science in general and machine learning in particular, but without a clear plan of how to approach these subjects in a structured manner. I wish to thank my advisor, Eduardo Laber, not only for guiding me in this work, but also for starting to show me how I can build this structure myself. It should go without saying that any mistakes in this thesis are my full responsibility.

All the other professors under which I had the chance of studying during these two years were kind and generous, and I also had the great luck of finding many bright and fun minds among my fellow students. You are too many to mention here, but I hope you know who you are. Thanks.

This study was financed in part by the Coordenação de Aperfeiçoamento Pessoal de Nível Superior(CAPES)–Finance Code 001.

## Abstract

Murtinho, Lucas; Laber, Eduardo Sany (Advisor). **Theoretical and experimental results in information-theoretic clustering**. Rio de Janeiro, 2020. 84p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

We present theoretical and experimental results related to the problem of clustering a set of vectors (which can be interpreted as probability distributions) with the goal of minimizing a weighted impurity measure of the resulting partition. The problem of clustering while minimizing the weighted Gini impurity of the partition is shown to be NP-complete and APX-hard, via a connection with the geometrical $k$-means problem. We also analyze a family of algorithms for information-theoretic clustering that rely on the dominant (largest) component of the vectors to be clustered. These algorithms are shown to be very fast compared to the state of the art, while able to achieve comparable results in terms of the resulting partition's weighted entropy.



## Keywords

Information Theory; Clustering; Impurity measures; Entropy; Gini.

## Resumo

Murtinho, Lucas; Laber, Eduardo Sany. **Resultados teóricos e experimentais em clusterização com métricas de teoria da informação**. Rio de Janeiro, 2020. 84p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esta dissertação apresenta resultados teóricos e experimentais relativos ao problema de clusterização de um conjunto de vetores (que possam ser interpretados como distribuições de probabilidade) com o objetivo de minimizar uma medida de impureza da partição resultante. Por meio de uma conexão entre o problema geométrico de $k$-médias e o problema de clusterização para minimizar a impureza ponderada de Gini da partição, prova-se que este último é NP-completo e APX-difícil. Também analisamos uma família de algoritmos para clusterização com base nas componentes dominantes (as maiores componentes) dos vetores a serem particionados. Mostra-se que, em alguns casos, dois desses algoritmos conseguem obter bons resultados em termos da entropia ponderada da partição resultante, em um tempo bem menor do que os algoritmos considerados como o estado da arte.



## Palavras-chave

Teoria da informação; Clusterização; Medidas de impureza; Entropia; Gini.

# Table of contents

# List of figures

## List of tables

## List of Abreviations

| | |
|---|---|
| KL | Kullback-Leibler divergence |
| PMWEP | Partition with Minimum Weighted Entropy Problem |
| PMWGP | Partition with Minimum Weighted Gini Problem |
| PMWIP | Partition with Minimum Weighted Impurity Problem |
| PTAS | Polynomial-time approximation scheme |



*Information is entropy. This was the strangest*
*and most powerful notion of all.*
**James Gleick**, *The Information: a History, a Theory, a Flood*

# 1
# Introduction

Clustering is one of the fundamental problems in computer science in general and in machine learning in particular. Generally speaking, it consists in partitioning data into groups so that similar items are together and dissimilar items are apart. It can be used, for instance, to reduce the dimensionality of a problem, by clustering similar variables together, thus serving as a preprocessing step in the application of machine learning algorithms; or, as an end in itself, to find previously unknown structure in a data set (for instance, to identify similar customers from a store's database).

From this general description of clustering a great number of particular problems emerge, depending on the specific goal in mind when partitioning the data. Clustering can be hard (each element in the data set may belong to a single cluster) or soft (each element may belong to different clusters with some probability); the end goal may be to maximize the intercluster distance (to make each cluster as dissimilar from the others as possible) or to minimize the intracluster distance (to make all elements in the same cluster as similar to each other as possible); and several dissimilarity measures may be used.

Concerning the latter, clustering a data set while minimizing a *weighted impurity measure* of the resulting partition has been studied for at least over thirty years (Breiman et al. (1984)). In this subset of clustering problems, the goal is to find the partition of the original data set with clusters that are as pure (in an information-theoretic sense) as possible, so that the information retrieved from the clusters is as close as possible to the information retrieved from the full data set. Finding a good partition of the data in the information-theoretic sense is important in fields such as word clustering, node splitting for decision-tree construction, and channel quantization for polar-code construction.

The state of the art when it comes to clustering for minimizing a wide variety of dissimilarity measures — Bregman divergences (Banerjee et al. (2005)) such as the squared Euclidean distance or the Kullback-Leibler divergence — is arguably Lloyd's algorithm (Lloyd (1982)) with the ++ initialization proposed by (Arthur & Vassilvitskii (2007)). Lloyd's algorithm is an expectation-maximization procedure that iteratively improves on the latest partition found,

and the ++ implementation guarantees an $\mathcal{O}(\log k)$ approximation to the best partition, where $k$ is the number of clusters.

One drawback from this method is its computational cost, as exponentially many iterations may be needed for the algorithm to converge, even in the simplest of cases (Vattanni (2005)). In practice, a frequently used heuristic is to stop the algorithm once an iteration provides only a small improvement in the results; however, even a single iteration can be quite costly. This, coupled with the importance of information-theoretic clustering, leads to our interest in finding faster methods that can approximate Lloyd's algorithm's results for hard partitioning using information-theoretic dissimilarity measures.

## 1.1 Problem definition

We are interested in the problem of partitioning a set $S$ of non-negative, real vectors so as to minimize the partition's weighted impurity (for a given impurity measure $i$, and its corresponding weighted impurity $I$). For a partition $\mathcal{P}$ of $S$,

$$I(\mathcal{P}) = \sum_{C \in \mathcal{P}} I(C);$$

that is, the partition's weighted impurity is the sum of weighted impurities of the clusters that comprise it. This, in turn, is defined as

$$I(C) = I\left(\sum_{\mathbf{v} \in C} \mathbf{v}\right);$$

that is, a cluster's weighted impurity is the weighted impurity of the sum vector of all the elements belonging to the cluster.

**Problem definition 1 ($k$-PMWIP)** *For a set $S$, an integer $k$, and a weighted impurity measure $I$, the* ***$k$-Partition with Minimum Weighted Impurity Problem*** *($k$-PMWIP) consists in finding a partition $\mathcal{P}$ of $S$ into $k$ clusters so that $I(\mathcal{P})$ is minimized.*

This general version of the problem can be further specified according to the weighted impurity measure being used. Thus the $k$-Partition with Minimum Weighted Entropy Problem ($k$-PMWEP) is the problem of finding the partition $\mathcal{P}$ of a set $S$ into $k$ clusters such that the partition's weighted entropy is minimized. Substituting the weighted Gini impurity for the weighted entropy in the above definition gives us the $k$-Partition with Minimum Weighted Gini Problem ($k$-PMWGP). We define these measures, as well as the general notion of weighted impurity measures, in Chapter 2.

## 1.2 Our contribution

We present contributions both for the theoretical understanding of information-theoretic clustering and for the development of practical algorithms to tackle the problem.

In the theoretical side, we prove, via a connection with the geometric $k$-means problem, that the PMWGP is both NP-complete and APX-hard. This same connection allows us to find a polynomial-time approximation scheme for this problem (when the number of clusters is fixed). This result has been previously published in (Laber & Murtinho (2019)).

In the practical side, we present an adaptation of the polynomial-time algorithm from (Cicalese et al. (2019)) for the PMWEP. Our experiments on three different data sets show that this practical algorithm, while orders of magnitude faster than Lloyd's algorithm or adaptations thereof, can approximate its performance under some circumstances.

## 1.3 Related work

### 1.3.1 Theoretical results

There have been theoretical investigations on methods to compute the best split efficiently for impurity measures such as the weighted Gini impurity. For $d = k = 2$, where $d$ is the number of dimensions in the vectors, (Breiman et al. (1984)) presents a simple algorithm that finds the best binary partition in $\mathcal{O}(n \log n)$ time for impurity measures in a certain class that includes Gini. The correctness of this algorithm relies on a theorem, also proved in (Breiman et al. (1984)), which is generalized for arbitrary $d$ and $k$ in (Chou (1991)), (Burshtein et al. (1992)), and (Coppersmith et al. (1999)). Basically, these theorems provide necessary conditions for partitions with minimum impurity and can be used to restrict the set of partitions that need to be considered. A connection between Gini and the squared $\ell_2$ distance employed by $k$-means, that we explore here, is mentioned in the appendix of (Chou (1991)).

For the Euclidean $k$-means problem a vast literature is available and the problem is well understood from the perspective of approximation algorithms, in the sense that the gap between the best available approximation factors and the thresholds given by the hardness results is small — see (Awasthi et al. (2015)) and the references therein. The algorithm that we

adapt for minimizing the weighted Gini impurity, and that can be used for many other dissimilarity measures, was presented in (Kumar et al. (2004)) and further studied in (Ackermann et al. (2010)).

### 1.3.2 Experimental results

The PMWEP is a generalization of the problem presented in (Chaudhuri & McGregor (2008)) of clustering elements so as to minimize the Kullback-Leibler divergence between each element and the closest cluster centroid.

As mentioned above, the state of the art in terms of clustering techniques for Bregman divergences is arguably the algorithm from (Lloyd (1982)) with the initialization technique from (Arthur & Vassilvitskii (2007)). This method was initially devised for the geometric $k$-means problem, where the dissimilarity measure to be minimized is the squared Euclidean distance; (Banerjee et al. (2005)) shows that the same algorithm works for any Bregman divergence. We use it as a benchmark against our algorithms in this thesis.

One of the applications for clustering while minimizing the partition's entropy is word clustering, in particular as a preprocessing step for tasks such as text classification. (Baker & McCallum (1998)) presents an algorithm that aims to minimize the "Kullback-Leibler divergence to the mean" (which is symmetrical and bounded, unlike the Kullback-Leibler divergence) between the elements of the data set and the centroids of their clusters. (Dhillon et al. (2003)) uses Lloyd's algorithm for the same preprocessing step, with an initialization that is similar to the algorithms we present here; it is another algorithm whose results we compare against our own.

When the data set to be partitioned is so large as to render Lloyd's algorithm (or variations thereof) impractical due to time constraints, one alternative is to apply the algorithm to coresets of the original data set. (Lucic et al. (2016)) presents a construction method for coresets that, when used to generate cluster centroids, will approximate (with some probability) the results of using the full data set for the same task. (Bachem et al. (2018)), by the same authors, presents a faster, and embarrassingly parallel, method for building such coresets. This second method is the last benchmark we use in our experiments.

## 1.4 Organization

In Chapter 2 we present some definitions deemed important for the understanding of the thesis, including a description of the main weighted impurity measures used in the problems of interest. We also briefly discuss some applications for our algorithms in the previously mentioned areas of word clustering, node splitting for decision-tree construction, and channel quantization for polar-code construction.

Chapter 3 presents our theoretical results related to the PMWGP, which is proven to be NP-complete and APX-hard via a connection with the geometric $k$-means problem. The same connection allows us to adapt an approximation scheme for the PMWGP that runs in polynomial time (while running in linear time for the geometric $k$-means problem).

Chapter 4 presents the dominance-based algorithms that are our main contribution to the study of the PMWIP. Two of the algorithms presented in this chapter, DOMINANCE and POLY (both previously described in (Cicalese et al. (2019))), have approximation guarantees concerning the PMWEP and the PMWGP, but are computationally impractical. The other two algorithms presented in the chapter, RATIO-GREEDY and STAR, borrow ideas from the previous two and are the ones we test in our experiments.

Chapter 5 presents the iteration-based algorithms whose performance we will compare to that of our algorithms: (i) the classical Lloyd's algorithm, with a ++ initialization and the Kullback-Leibler divergence used as a dissimilarity measure; (ii) the algorithm from (Dhillon et al. (2003)), which is Lloyd's algorithm with an initialization that is a simplified version of one of our dominance-based algorithms; and (iii) the algorithm from (Bachem et al. (2018)), which relies on coresets to partition the original data set.

Chapter 6 presents and discusses the results of our experiments on three different data sets: two text collections (20 NEWSGROUPS and RCV1) and a synthetic data set similar to one of the data sets from (Lucic et al. (2016)). Chapter 7 concludes the thesis.

# 2
# Background

In this chapter we present some definitions pertaining to information-theoretic clustering, and also briefly discuss some applications of it as a means of motivating the work ahead.

## 2.1
## Hard clustering

Let $S$ be a set of items. A $k$-partition of $S$ is a collection of $k$ subsets $S_1, \ldots, S_k$ such that

1. $S_1 \cup S_2 \cup \cdots \cup S_k = S$
2. $S_i \cap S_j = \emptyset \; \forall \; i, j \in \{1, \ldots, k\}, i \neq j.$

Hard clustering is the task of finding such a partition of $S$ that has some desired properties, typically involving a *dissimilarity measure* $D$ — a function that maps two items in $S$ to a non-negative real value, such that $D(x, y) = 0 \iff x = y$. A canonical goal of hard clustering is to find a partition such that dissimilar elements are in separate subsets, while similar elements are grouped together.

The idea of *hard clustering* stands in contrast to *soft clustering*, in which an element of $S$ may belong to more than one subset, or to different subsets with different probabilities. In this thesis, we focus on hard versions of the clustering problem, while noting that it is possible to adapt some of the algorithms discussed here — for instance, the coreset algorithm from (Bachem et al. (2018)) — for soft-clustering problems.

The complexity of a clustering problem depends on the specific goal — i.e., on what one intends to achieve by partitioning the original set. For instance, define the dissimilarity between any two clusters as the smallest dissimilarity between two elements that belong one to each of the clusters:

$$D(S_i, S_j) = \min_{x \in S_i, y \in S_j} \{D(x, y)\}, i \neq j.$$

In this case, the goal of maximizing the dissimilarity between clusters can be achieved in polynomial time using, for instance, Kruskal's algorithm for

generating minimum spanning trees (Kruskal (1956)). Minimizing the sum of dissimilarities between elements in the same cluster, however, is NP-complete (Vattanni (2005)).

## 2.2 Information-theoretic clustering

We broadly define *information-theoretic clustering* as a clustering task that relies on dissimilarity measures supported by, and of interest in, information theory. In particular, we are interested in *frequency-weighted impurity measures* as presented in (Laber et al. (2018)). In this instance of information-theoretic clustering, our goal is to find the partition with minimum impurity, where a partition's impurity is defined as the sum of impurities of the clusters that comprise it.

### 2.2.1 Frequency-weighted impurity measures

Given a vector $\mathbf{v} = (v_1, \ldots, v_d)$, an *impurity measure* $i(\mathbf{v})$ is a function of the form

$$i(\mathbf{v}) = \sum_{i=1}^{d} f\left(\frac{v_i}{||\mathbf{v}||_1}\right), \tag{2-1}$$

where $f : \mathbb{R} \mapsto \mathbb{R}$ satisfies the following conditions:

1. $f(0) = f(1) = 0$.
2. $f$ is strictly concave in the interval $[0, 1]$.
3. For all $0 < p \leq q \leq 1$,

$$f(p) \leq \frac{p}{q} \cdot f(q) + q \cdot f\left(\frac{p}{q}\right).$$

There are two main circumstances in which impurity measures may be of interest:

1. $\mathbf{v}$ is a probability distribution. A fully homogeneous vector in this scenario would be equivalent to a uniform distribution, with all possible events having the same probability of happening, and $i(\mathbf{v})$ would be maximized in this case. Conversely, a probability distribution in which the whole probability mass is concentrated on a single event would have $i(\mathbf{v}) = 0$.

2. $\mathbf{v}$ is a counting vector. If $\mathbf{v} \in \mathbb{R}^d_+$, then there are $d$ possible classes, and $v_i$ is the number of elements in $S$ that belong to class $i$. If all elements belong to class $j$, $v_j = |S|$ and $v_i = 0 \; \forall \; i \neq j$, and $i(\mathbf{v}) = 0$. If each class is represented by the same number of elements in $S$, $v_i = \frac{|S|}{d} \; \forall \; i = 1, \ldots, d$, and the impurity of $\mathbf{v}$ is maximized.

The second case above helps explain the importance, in some settings, of scaling (or weighting) an impurity measure according to the vector's magnitude. Let $S_1$ and $S_2$ be two sets whose elements are uniformly distributed among $d$ classes, and such that $|S_1| >> |S_2|$. For a non-weighted impurity measure, both sets would have the same (maximum) impurity assigned to them. However, "purifying" $S_1$ (by partitioning it into $d$ subsets with zero impurity, for instance) would mean classifying more elements into pure sets than doing the same for $S_2$. It is to capture this information that we weight the impurity of a vector $\mathbf{v}$ by its $\ell_1$ norm, $||\mathbf{v}||_1$, so that vectors with the same distribution of mass but different values will have different weighted impurities. Given an impurity measure $i(\mathbf{v})$, a weighted version of it will be given by $I(\mathbf{v}) = ||\mathbf{v}||_1 \cdot i(\mathbf{v})$.

Below we present two of the most used impurity measures in computer science and information theory: entropy and Gini.

#### 2.2.1.1 Entropy

In the information-theoretic context, entropy is defined as the rate of information produced by a stochastic process (Shannon (1948)). Given a vector $\mathbf{v}$ which is a probability distribution, the entropy of $\mathbf{v}$ is the amount of information acquired upon the realization of an event under the probabilities given by $\mathbf{v}$; if $\mathbf{v}$ counts the number of elements from each of $d$ different classes in a set $S$, the entropy of $\mathbf{v}$ is how much information is obtained by selecting one of these elements. Another interpretation is that the entropy of $\mathbf{v}$ corresponds to the uncertainty that a given event will happen under the probabilities defined by $\mathbf{v}$.

The formula for the entropy of $\mathbf{v}$ is

$$i_{entropy}(\mathbf{v}) = -\sum_{i=1}^{d} \frac{v_i}{||\mathbf{v}||_1} \log \frac{v_i}{||\mathbf{v}||_1},$$

and the weighted entropy is defined as $Entropy(\mathbf{v}) = ||\mathbf{v}||_1 \cdot i_{entropy}(\mathbf{v})$. We can see that the weighted entropy is one of the frequency-weighted impurities of the class defined by Equation 2-1 by setting $f(x) = x \log \frac{1}{x}$ (Laber et al. (2018)).

**Problem definition 2 ($k$-PMWEP)** *For a set $S$ of $n$ vectors in $\mathbb{R}^d$ and an integer $k$, the $k$-**Partition with Minimum Weighted Entropy Problem** ($k$-PMWEP) consists in finding a partition $\mathcal{P}$ of $S$ into $k$ clusters so that*

$$\sum_{C\in\mathcal{P}} -||\mathbf{v}^C||_1 \sum_{i=1}^{d} \frac{v_i^C}{||\mathbf{v}^C||_1} \log \frac{v_i^C}{||\mathbf{v}^C||_1}$$

*is minimized, where $\mathbf{v}^C = \sum_{\mathbf{v}\in C} \mathbf{v}$.*

#### 2.2.1.2 Gini

Given a non-negative vector $\mathbf{v} = (v_1, \ldots, v_d)$, the Gini impurity $i_{Gini}$ measures the probability that an object will be misclassified when it is assigned to class $i$ with probability $\frac{v_i}{||\mathbf{v}||_1}$:

$$i_{Gini}(\mathbf{v}) = \sum_{i=1}^{d} \frac{v_i}{||\mathbf{v}||_1} \left(1 - \frac{v_i}{||\mathbf{v}||_1}\right).$$

The weighted Gini impurity is defined as $Gini(\mathbf{v}) = \|\mathbf{v}\|_1 \cdot i_{Gini}(\mathbf{v})$. To see that it belongs to the class defined by Equation 2-1, set $f(x) = x(1-x)$ (Laber et al. (2018)).

**Problem definition 3 ($k$-PMWGP)** *For a set $S$ of $n$ vectors in $\mathbb{R}^d$ and an integer $k$, the $k$-**Partition with Minimum Weighted Gini Problem** ($k$-PMWGP) consists in finding a partition $\mathcal{P}$ of $S$ into $k$ clusters so that*

$$\sum_{C\in\mathcal{P}} ||\mathbf{v}^C||_1 \sum_{i=1}^{d} \frac{v_i^C}{||\mathbf{v}^C||_1} \left(1 - \frac{v_i^C}{||\mathbf{v}^C||_1}\right)$$

*is minimized, where $\mathbf{v}^C = \sum_{\mathbf{v}\in C} \mathbf{v}$.*

### 2.2.2 PMWEP and clustering for minimizing the Kullback-Leibler divergence

As stated in (Cicalese et al. (2019)), the $k$-PMWEP is a generalization of $\text{MTC}_{KL}$ (Chaudhuri & McGregor (2008)), the problem of clustering a set of $n$ probability distributions into $k$ groups minimizing the total Kullback-Leibler divergence (KL) from the distributions to the centroids of their assigned groups. $\text{MTC}_{KL}$ corresponds to the particular case of PMWEP where each vector has the same $\ell_1$ norm.

KL measures how dissimilar two probability distributions are. Given two discrete probability distributions $\mathbf{u}, \mathbf{v} \in \mathbb{R}_+^d$, the KL of $\mathbf{u}$ with respect to $\mathbf{v}$ is

given by

$$KL(\mathbf{v}, \mathbf{u}) = \sum_{i=1}^{d} v_i \log\left(\frac{v_i}{u_i}\right).$$

KL is neither symmetric nor bounded (for a case where $v_i \neq 0$ and $u_i = 0$, it is undefined). In our experiments, we will always consider the centroid of the cluster as the distribution that is trying to approximate the vector (the centroid is $\mathbf{u}$ in the formula above), and KL will indicate how well this approximation is performed.

## 2.3 Applications

Three main applications motivate our work on information-theoretic clustering: i) clustering words for text classification and other tasks related to natural language processing; ii) choosing the best split of a node when building a decision tree; and iii) quantizing channels as a step for the construction of polar codes.



### 2.3.1 Word clustering

The problem of *text classification* can be generally described as follows: let $T$ be a set of texts, each of them classified in one of $d$ possible classes, $C_1, \ldots, C_d$. Our goal is to correctly assign a text $t \in T$ to its class $C_i$, given the contents of the text.

One usual approach for this task is to use a Naive Bayes classifier (Baker & McCallum (1998), Dhillon et al. (2003)), treating the text $t$ as a bag of words and analyzing the relative frequency of words in each class to figure out to which of them $t$ is more likely to belong. Other algorithms used for this task include support vector machines (Dhillon et al. (2003)) and, more recently, deep learning models (Miyato et al. (2017)).

One drawback of this approach is that using words as variables leads to a model of high dimensionality. In a reasonably small data set for current standards, such as the 20 Newsgroups data set (Rennie (2014)), there may be as many as 50,000 words; larger data sets such as RCV1 (Lewis et al. (2004)) have almost 200,000 words; and larger data sets still may have millions or tens of millions of words. The computational cost of running text classification algorithms using words as variables may therefore be quite high.

One solution to this problem starts from recognizing that the information provided by two given words, $w_1$ and $w_2$, may be similar if they appear with the same relative frequency in texts of all classes. In fact, considering a bag-of-

words approach, if $w_1$ and $w_2$ have the same relative frequency for all classes (in the extreme case, if they appear the same amount of time in all texts), they would impart exactly the same information to the model, and using them both would be unnecessary. In many cases, similar words could provide similar information concerning the class of a given text, particularly depending on the context of the classification task.

Therefore, it seems reasonable to perform *word clustering* as a preprocesssing step to reduce the number of variables fed to a classification algorithm. This clustering task would bring the number of variables to a manageable size while keeping each cluster as pure as possible, so that the clusters impart, as much as possible, the same amount of information to the model as the unclustered words would.

Let $k$ be the desired number of variables to be used in a text classification task, and let $n$ be the number of distinct words found in all texts in the data set. Clustering these $n$ words into $k$ clusters with minimum weighted entropy would amount to an instance of the $k$-PMWEP.

$k$ may still be a large absolute number, as partitioning words into too small a number of clusters may lead to very different words being clustered together, with significant increase of the partition's weighted impurity, resulting in a significant decrease in performance from text classification algorithms. However, given the large number of distinct words a text data set may have, it may still be the case that $k << n$, even if $k >> 0$. Some experiments (Baker & McCallum (1998), Dhillon et al. (2003)) have shown that word clustering may be used to significantly reduce the dimensionality of a text classification problem without significantly reducing (and eventually even increasing) the accuracy of text classification models.



### 2.3.2 Node splitting for decision-tree construction

Decision trees, and decision-tree based algorithms such as random forests or gradient boosted trees, are among the most popular models for classification in machine learning. Decision trees are famously easy to interpret, while models based on an ensemble of trees may lead to state-of-the-art results in several tasks.

The main idea behind classification trees is to recursively split the data according to its attributes, so that the leafs of the tree present subsets of the original data that belong to a single class. This way, one can classify a new datum by analyzing to which leaf it belongs, and by assigning it to the same class as the data in that leaf. Figure 2.1 shows an example of a classification

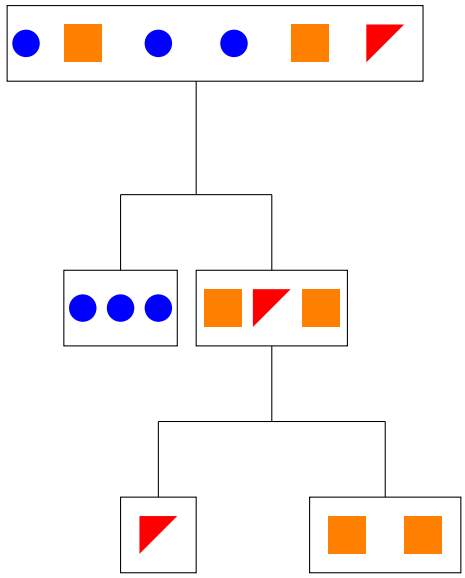


Figure 2.1: An example of decision tree.

tree in a toy data set.

Generally speaking, we want to split the data in a given node so that similar elements will be in the same subnode, while different elements will be apart. In other words, we want to increase the partition's purity when performing the split.

Decision tree implementations have traditionally used weighted impurity measures for determining the best split of a node. For instance, in the CART package (Breiman et al. (1984)), the goal when splitting a node is to minimize the weighted Gini impurity of the subsets generated by the split. In a classification problem, this impurity may be calculated for a counting vector $\mathbf{v} \in \mathbb{R}^d$ representing a node, where $d$ is the number of possible classes and $\mathbf{v}_i$ indicates how many elements of class $i$ are in the node.

In contrast with the word clustering application presented above, in node splitting the partition will usually consist of a small number of clusters — $k$ will be small. Many decision trees are binary, with nodes being split into two subnodes only. Boosted trees usually rely on the mixture of short trees (sometimes called decision stumps), while random forests can deal in practice with very long trees by making them "vote" to decide on the class of a given datum.

### 2.3.3 Channel quantization

Polar codes, introduced in (Arıkan (2009)), are the first family of error-correcting codes to achieve the capacity of discrete memoryless channels, and have since been adapted to achieve the capacity of other channels

(Fayyaz & Barry (2013), Goela et al. (2014)), as well as used in the implementation of 5G wireless standards (Sandberg et al. (2018)). The technique of *channel polarization* consists of splitting the original channel into subchannels that become polarized in the sense that the error rate of a subchannel either grows (turning it into a purely noisy channel) or reduces (turning it into a noiseless channel). Using the noiseless polarized channels, one can transmit information with virtually no error.

One difficulty that arises from the original method for constructing polar codes is that the resulting (noisy and noiseless) channels present an output alphabet exponentially large in relation to the code length. To allow for the efficient construction of polar codes, channels with this intractably large alphabet need to be approximated by channels whose alphabets are of manageable size (Tal & Vardy (2013)).

Quantizing (or clustering) the output of the polarized channels, while preserving as much of the information from the output as possible, prevents the problem from becoming intractable. (Kurkoski & Yagi (2014)) explicitly approaches this problem while noting its similarity with the problem of node splitting presented above; in both cases, the desired partition must be as pure as possible (in the information-theoretic sense).

# 3
# Theoretical results for the PMWGP

This chapter presents theoretical results pertaining to the PMWGP. A connection between this problem and the geometrical $k$-means problem is formalized and allows us to prove that the PMWGP is both NP-complete and APX-hard. Also based on this connection, we present a polynomial-time approximation scheme for the $k$-PMWGP.

## 3.1
## Problem definition

As mentioned in Section 1.1 above, the $k$-PMWGP is the problem of finding a partition $\mathcal{P}$ of a set $V$ of vectors into $k$ clusters so as to minimize the sum of the weighted Gini impurities

$$Gini(\mathcal{P}) = \sum_{i=1}^{k} Gini\left(\sum_{\mathbf{v}\in V_i} \mathbf{v}\right). \tag{3-1}$$

A problem that is equivalent to the above one from the perspective of optimality (but different from the perspective of approximation) is finding the partition $\mathcal{P}$ of $V$ into $k$ groups that minimizes

$$Gini(\mathcal{P}) - \sum_{\mathbf{v}\in V} Gini(\mathbf{v}). \tag{3-2}$$

Using concavity properties of Gini, one can prove that the above expression is always non-negative. An $\alpha$-approximation with respect to goal (3-2) implies an $\alpha$-approximation with respect to goal (3-1), but the converse is not necessarily true, so that approximations with respect to goal (3-2) are stronger (Coppersmith et al. (1999)).

## 3.2
## The geometric $k$-means problem

In the geometric $k$-means problem, we are given a set of vectors $V \subset \mathbb{R}^d$, and the goal is to find a partition $\mathcal{P}$ of $V$ into $k$ groups $V_1, \ldots, V_k$ and a set of $k$ centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$ in $\mathbb{R}^d$ such that

$$Cost_{KM}(\mathcal{P}) = \sum_{i=1}^{k} \sum_{\mathbf{v}\in V_i} \|\mathbf{v} - \mathbf{c}_i\|_2^2$$

is minimized.

It is well known that if $U$ is a set of vectors then the vector $\mathbf{c}$ for which $\sum_{\mathbf{v}\in U} \|(\mathbf{v}-\mathbf{c})\|_2^2$ is minimum is the centroid of $U$, that is, $\mathbf{c} = (\sum_{\mathbf{v}\in U} \mathbf{v})/|U|$.

## 3.3 Connection between the PMWGP and geometrical $k$-means

In this section we show that the following connections between $k$-*PMWGP* and the $k$-means problem hold:

**Proposition 1** *Let $V$ be an instance of $k$-means in which all vectors have the same $\ell_1$ norm. If $\mathcal{P}$ is an optimal partition for instance $V$ of $k$-means, then $\mathcal{P}$ is also an optimal partition for instance $V$ of $k$-PMWGP.*

**Proposition 2** *There exists a pseudo-polynomial time reduction from $k$-PMWGP to the geometric $k$-means problem.*

The key observation for establishing both propositions is the following lemma, which expands on the remark from the appendix of (Chou (1991)) that the weighted Gini impurity and the weighted squared error (dissimilarities used for classification and regression problems, respectively) are built upon the same loss function.

**Lemma 1** *Let $X$ be a set of $n$ vectors, all of them with $\ell_1$ norm equal to $L$. Then,*

$$Gini\left(\sum_{\mathbf{v}\in X}\mathbf{v}\right) - \sum_{\mathbf{v}\in X} Gini(\mathbf{v}) = \frac{1}{L}\times\left(\sum_{\mathbf{v}\in X}\|\mathbf{v}-\mathbf{c}\|_2^2\right),$$

*where $\mathbf{c}$ is the centroid of the set of vectors in $X$.*

*Proof.* Let $\mathbf{u} = \sum_{\mathbf{v}\in X}\mathbf{v}$ and $d$ be the dimension of the vectors in $X$. We have that

$$Gini(\mathbf{u}) - \sum_{\mathbf{v}\in X} Gini(\mathbf{v}) =$$

$$\|\mathbf{u}\|_1 \sum_{i=1}^{d}\left(\frac{u_i}{\|\mathbf{u}\|_1}\right)\left(1-\frac{u_i}{\|\mathbf{u}\|_1}\right) - \sum_{\mathbf{v}\in X}\|\mathbf{v}\|_1\sum_{i=1}^{d}\left(\frac{v_i}{\|\mathbf{v}\|_1}\right)\left(1-\frac{v_i}{\|\mathbf{v}\|_1}\right)$$

$$= \sum_{i=1}^{d}\left(u_i - \frac{(u_i)^2}{L\times n}\right) - \sum_{i=1}^{d}\sum_{\mathbf{v}\in X}\left(v_i - \frac{(v_i)^2}{L}\right).$$

On the other hand,

$$\sum_{\mathbf{v}\in X}\|\mathbf{v}-\mathbf{c}\|_2^2 = \sum_{i=1}^{d}\sum_{\mathbf{v}\in X}(v_i-c_i)^2.$$

Thus, it suffices to show that, for any $i$,

$$u_i - \frac{(u_i)^2}{L \times n} - \sum_{\mathbf{v} \in X} \left( v_i - \frac{(v_i)^2}{L} \right) = \frac{1}{L} \left( \sum_{\mathbf{v} \in X} (v_i - c_i)^2 \right). \tag{3-3}$$

The left side of (3-3) is equal to

$$u_i - \frac{(u_i)^2}{L \times n} - u_i + \frac{\sum_{\mathbf{v} \in X} (v_i)^2}{L} = \frac{1}{L} \times \left( \sum_{\mathbf{v} \in X} (v_i)^2 - \frac{(u_i)^2}{n} \right).$$

Moreover, the right side of (3-3) is equal to

$$\frac{1}{L} \times \left( \sum_{\mathbf{v} \in X} (v_i)^2 - 2c_i \sum_{\mathbf{v} \in X} v_i + \sum_{\mathbf{v} \in X} (c_i)^2 \right)$$

$$= \frac{1}{L} \times \left( \sum_{\mathbf{v} \in X} (v_i)^2 - 2\frac{(u_i)^2}{n} + n\frac{(u_i)^2}{n^2} \right) = \frac{1}{L} \times \left( \sum_{\mathbf{v} \in X} (v_i)^2 - \frac{(u_i)^2}{n} \right),$$

which establishes the lemma. ■

Proposition 1 is a direct consequence of Lemma 1, since it implies that, for all $k$-partitions $\mathcal{P}$ of $V$,

$$Gini(\mathcal{P}) = \frac{1}{L} \cdot Cost_{KM}(\mathcal{P}) + \sum_{\mathbf{v} \in V} Gini(\mathbf{v}).$$

With regard to Proposition 2, let $V$ be an instance of *k-PMWGP* and let $V'$ be the instance of $k$-means obtained from $V$ as follows: for each vector $\mathbf{v} \in V$, we add to the instance set $V'$ exactly $\|\mathbf{v}\|_1$ copies of the vector $\mathbf{v}' = \mathbf{v}/\|\mathbf{v}\|_1$. Using Lemma 1 and also the fact that in any optimal solution for $k$-means identical vectors are in the same partition, we conclude that the optimal value of $V$ and $V'$ differ by exactly $\sum_{\mathbf{v} \in V} Gini(\mathbf{v})$. Note that instance $V'$ is obtained from $V$ in pseudo-polynomial time.



## 3.4 Hardness of PWMGP

From Proposition 1 and the hardness of geometric $k$-means established in (Awasthi et al. (2015)) we obtain:

**Theorem 3.1** *The Partition with Minimum Weighted Gini Problem (PMWGP) is NP-complete with respect to goal* (3-1) *and APX-hard with respect to goal* (3-2).

*Proof.* Theorem 1.1 of (Awasthi et al. (2015)) states that there is a constant $\epsilon$ such that it is NP-hard to approximate the $k$-means problem to a factor better than $(1 + \epsilon)$. In the instance used to prove the theorem in (Awasthi et al. (2015)), all vectors have $\ell_1$ norm of 2; from our Lemma 1,

it follows that the goal (3-2) and the objective function for $k$-means differ by a factor of exactly 2. The NP-completeness of goal (3-1) follows because goals (3-1) and (3-2) differ by an additive constant. ■

## 3.5 Approximating the optimal Gini partition

The reduction given by Proposition 2 allows us to obtain new algorithms for $k$-*PMWGP* with provable approximation factors. As an example, we discuss how to obtain a randomized PTAS for $k$-*PMWGP* with respect to the objective function (3-2) when $k$ is fixed. To do so, we run over instance $V'$ the randomized polynomial-time approximation scheme for the $k$-means problem proposed by (Kumar et al. (2004)). Algorithm 1 (CLUSTER) is a slightly modified version of the algorithm as presented in Figure 1 of (Ackermann et al. (2010)).

In what follows we explain how Algorithm 1 works and how we adapt it to our purposes. However, we do not detail the analysis of its approximation, since it is not necessary to establish our result. We refer the reader to (Ackermann et al. (2010)) for a complete presentation of the PTAS proposed in (Kumar et al. (2004)). This presentation includes a simplified proof of its approximation factor as well as a discussion of how it generalizes to other objective functions.

**Algorithm 1** CLUSTER$(R, l, \tilde{C})$

**Input**:
$R$: set of remaining input points
$l$: number of medians to be found
$\tilde{C}$: set of medians already found
**Procedure**:

**if** $l = 0$ **then** return $\tilde{C}$
**else**
  **if** $l \geq |R|$ **then** return $\tilde{C} \cup R$
  **else**
    $C^{(\tilde{c})} = \{\}$
    */* sampling phase */*
    sample a multiset $S$ of size $\frac{2}{\alpha\gamma\delta}$ from $R$
    $C \leftarrow \left\{ \frac{\sum_{\mathbf{v} \in S'} \mathbf{v}}{|S'|} \; \forall \; S' \subset S, \; |S'| = \frac{1}{\gamma\delta} \right\}$ *//set of median candidates*
    **for all** $\tilde{c} \in C$ **do**
      $C^{(\tilde{c})} \leftarrow C^{(\tilde{c})} \cup$ CLUSTER$(R, l-1, \tilde{C} \cup \{\tilde{c}\})$
    */* pruning phase */*
    let $N$ be the set with the $\frac{1}{2}|R|$ minimal points $p \in R$ w.r.t. $\min_{\tilde{c} \in \tilde{C}} ||p - \tilde{c}||_2^2$
    $C^{(\tilde{c})} \leftarrow C^{(\tilde{c})} \cup$ CLUSTER$(R \setminus N, l, \tilde{C})$
    **return** $C \in C^{(\tilde{c})}$ with minimum cost

The main result of this section is Theorem 3.2, presented below.

**Theorem 3.2** *For any $\epsilon, \delta > 0$ and for a fixed $k$,* Cluster *provides, in polynomial time (in $n$) and with probability $\geq 1 - \delta$, a $(1 + \epsilon)$-approximation to $k-PMWGP$.*

Cluster takes as input the set $R$ of remaining objects to be clustered; the number $l$ of medians yet to be found; and the set $\tilde{C}$ of medians already found. (Initially, $R = V$, $l = k$, and $\tilde{C} = \emptyset$.) In addition, it employs 3 parameters: $\gamma$, which is used to control the approximation factor; $\delta$, which is used to control the probability of returning a good cluster in terms of approximation; and $\alpha$, a positive constant that impacts both the running time and the approximation factor. Cluster returns the set $\tilde{C}$, containing $k$ medians. The points of $V$ can then be clustered according to their closest points in the set $\tilde{C}$.

If there are no medians to be found ($l = 0$), then the algorithm returns the set $\tilde{C}$. In another base case, when $l \geq |R|$, each of the points in $R$ becomes a new median. Otherwise, the algorithm considers two strategies, presented below, to cluster the points in $R$, and returns the best clustering among those built by these strategies.

1. The algorithm builds a set $C$ containing the centroids of all subsets of $S$ with size $1/\gamma\delta$, where $S$ is a random multiset of $R$ with size $2/\alpha\gamma\delta$. Each centroid $\tilde{c} \in C$ is added separately to $\tilde{C}$, and the algorithm performs $|C|$ recursive calls with parameters $(R, l-1, \tilde{C} \cup \tilde{c})$. This strategy corresponds to lines 7-10 of Algorithm 1.

2. The algorithm builds a set $N$ containing the $|R|/2$ points in $R$ that are closest to the medians in $\tilde{C}$. Then, the procedure is recursively called with parameters $(R \setminus N, l, \tilde{C})$. This strategy corresponds to lines 12-13 of Algorithm 1.

Therefore, we can define the number of calls made by Cluster as:

$$T(|R|, l) = \begin{cases} 1 & \text{if } l = 0 \text{ or } |R| \leq l \\ c \times T(|R|, l-1) + T(|R|/2, l) + 1 & \text{otherwise} \end{cases} \quad (3\text{-}4)$$

where $c$ is the cardinality of the set $C$. An important fact is that $c$ is constant with respect to $|V|$ and $k$, since it only depends on $\gamma$, $\delta$, and $\alpha$. In fact, $c \leq 2^{|S|} = 2^{\frac{2}{\alpha\gamma\delta}}$.

Theorem 2.8 of (Ackermann et al. (2010)) shows that Cluster executes at most $n2^{O(k/\gamma\delta \cdot \log(1/\gamma\delta\alpha))}$ arithmetic operations. In addition, Theorem 2.5 of (Ackermann et al. (2010)) shows that, for $\alpha < \frac{1}{4k}$,

$$Pr\left[Cost\left(\tilde{C}\right) \leq \left(1 + 8\alpha k^2\right)(1+\gamma)\, OPT\right] \geq \left(\frac{1-\delta}{5}\right)^k, \qquad (3\text{-}5)$$

where $Cost(\tilde{C})$ is the cost of a clustering induced by the points in $\tilde{C}$ and $OPT$ is the cost of the optimal solution.

**Algorithm 1 for the minimization of Gini.** By applying CLUSTER to the instance $V'$, obtained via Proposition 2, we find with probability at least $((1-\delta)/5)^k$ a partition whose Gini is not larger than $(1+\epsilon)$ times the Gini of an optimal partition, where $\epsilon = 8\alpha k^2(1+\gamma)$. Since the size of $V'$ is pseudopolynomial on the size of $V$, CLUSTER$(V', k, \{\})$ would have an exponential running time with respect to $n = |V|$ in the worst case.

In order to prove Theorem 3.2, we must do the following:

1. Develop a method of constructing instance $V'$ from $V$ in strictly polynomial time, even if $|V'|$ is exponential in $|V|$. We do this by keeping each distinct vector $\mathbf{v}' = \mathbf{v}/\|\mathbf{v}\|_1$ and its multiplicity $||\mathbf{v}||_1$ rather than all the $\sum_{\mathbf{v}\in V} \|\mathbf{v}\|_1$ vectors of instance $V'$.

2. Verify that the running time of a single call to CLUSTER is strictly polynomial in $|V|$. We show in Lemma 2 that this can be achieved by modifying a single step in CLUSTER.

3. Verify that the number of recursive calls performed by CLUSTER$(V', k, \{\})$ is polynomial in $n = |V|$. We prove this in Lemma 3 by induction on recurrence (3-4).

**Lemma 2** *Given a fixed $k$, a single call to* CLUSTER$(V', l, \tilde{C})$ *performs* $\mathcal{O}(n \lg n)$ *operations, where* $n = |V|$.

*Proof.* Let $W = |V'| = \sum_{\mathbf{v}\in V} ||\mathbf{v}||_1$. In the base case when $l = 0$, the algorithm returns the set of medians found, at constant cost. In the base case when $W \leq l$, the algorithm assigns the $W$ remaining points as medians; this would take $\mathcal{O}(W)$ operations, but we are keeping track of the $n$ different vectors in $V'$ and their multiplicities, so assigning them as medians is $\mathcal{O}(n)$.

It remains to evaluate the complexity of calls when no base case has been reached. The only steps in CLUSTER that depend on the size of the input are steps 7 and 12, and we analyze them for input $V'$ as follows:

1. Step 7 samples a constant number of vectors from $V'$. Since the $W$ vectors in $V'$ are being represented as $n$ vectors along with their multiplicities, we can sample these $n$ vectors in proportion to their quantities, and no additional cost is incurred.

2. Step 12 computes the $W/2$ closest vectors to a given set of medians $\tilde{C}$. This is achieved in (Ackermann et al. (2010)) by finding the median element of $V'$ according to their distances to the medians, taking $\mathcal{O}(W)$ operations. Instead, we can order the $n$ unique vectors in $V'$ according to their minimal distance to $\tilde{C}$, and then select the $W/2$ closest vectors by checking the multiplicities of the vectors in $V'$. This will take $\mathcal{O}(n \lg n)$ operations.

Therefore, the number of operations of any call of CLUSTER with $V'$ as input is $\mathcal{O}(n \lg n)$. ■

**Lemma 3** *For the recurrence presented in* (3-4),

$$T(W,k) \leq c^{k+1}(\lg((k+1)W))^k - c^k,$$

*where $c$ is a constant smaller than $2^{\frac{2}{\alpha\gamma\delta}}$.*

*Proof.* For the base cases mentioned above:

- $k = 0$: $T(W,0) = 1 \leq c(\lg W)^0 - c^0$ which holds as long as $c \geq 2$;
- $k \geq W \geq 1$: $T(W,k) = 1 \leq c^{k+1}(\lg((k+1)W))^k - c^k$ which holds because $\frac{1}{c^k} + 1 \leq 2 \leq c \leq c(\lg((k+1)W))^k$.

Let $W > k \geq 1$ and assume that the lemma holds for all $W'$, $k'$ with $W' < W$ or $k' < k$. By induction, we can use recurrence (3-4) to show that:

$$\begin{aligned} T(W,k) &\leq c(c^k(\lg(kW))^{k-1} - c^{k-1}) + c^{k+1}\left(\lg\frac{(k+1)W}{2}\right)^k - c^k + 1 \\ &= c^{k+1}\left(\left(\lg(kW)\right)^{k-1} + \left(\lg\frac{(k+1)W}{2}\right)^k\right) - 2c^k + 1. \quad (3\text{-}6) \end{aligned}$$

We must therefore show that (3-6) $\leq c^{k+1}(\lg((k+1)W))^k - c^k$. Since $1 \leq c^k$ if $c \geq 1$, it suffices to prove

$$\begin{aligned} \left(\lg(kW)\right)^{k-1} + \left(\lg\frac{(k+1)W}{2}\right)^k \leq \\ \left(\lg((k+1)W)\right)^{k-1} + \left(\lg\frac{(k+1)W}{2}\right)^k \leq \\ (\lg((k+1)W))^k \end{aligned}$$

which we can do by showing that

$$1 + \frac{\left(\lg \frac{(k+1)W}{2}\right)^k}{(\lg((k+1)W))^{k-1}} \leq \lg((k+1)W)$$

$$\frac{\left(\lg \frac{(k+1)W}{2}\right)^k}{(\lg((k+1)W))^{k-1}} \leq \lg \frac{(k+1)W}{2}$$

$$\frac{\left(\lg \frac{(k+1)W}{2}\right)^{k-1}}{(\lg((k+1)W))^{k-1}} \leq 1. \tag{3-7}$$

Since (3-7) is always true, the lemma holds. ■

$c$ is exponential in $1/\alpha\gamma$, and $\epsilon = 8\alpha k^2(1+\gamma)$; therefore, Algorithm 1 is exponential in the approximation factor, but polynomial in $n$. The proof of Theorem 3.2 follows directly from both lemmas:

*Proof of Theorem 3.2.* Since, for a fixed $k$, there are at most $\mathcal{O}(n^k)$ calls to Algorithm 1 (Lemma 3), each performing at most $\mathcal{O}(n \lg n)$ operations (Lemma 2), the overall running time of CLUSTER$(V', k, \{\})$ is $\mathcal{O}(n^{k+1} \lg n)$. ■

# 4
# Dominance-based algorithms

The remaining chapters of this thesis will analyze algorithms for information-theoretic clustering, with emphasis on the PMWEP. While some algorithms of purely theoretical interest will be presented, our focus will be on experimental results of practical algorithms, considering both their results in terms of the partition's weighted entropy and their running time.

In this chapter, four different clustering algorithms based on partitioning vectors according to their dominant components will be presented and analyzed. The first two, DOMINANCE and POLY, have already been presented in Sections 4 and 5.2, respectively, of (Cicalese et al. (2019)). The third one, RATIO-GREEDY, is a slightly modified version of the algorithm of the same name presented in Section 7 of (Cicalese et al. (2019)). Both RATIO-GREEDY and STAR, the fourth algorithm presented in this chapter, are practical algorithms whose experimental results will be the subject of analysis in Chapter 6. This thesis is mainly concerned with the results of the experiments for practical dominance-based algorithms, as that was the contribution of the author to the work presented in (Cicalese et al. (2019)).



## 4.1
## The Dominance algorithm

Let $V$ be a collection of $n$ vectors in $\mathbb{R}^d_{>0}$. The DOMINANCE algorithm consists in partitioning the vectors in $V$ into (at most) $d$ clusters according to the dominant components of each vector.

Let $k$ be the desired number of clusters. If $k > d$, DOMINANCE will leave at least $k-d$ clusters empty. Fixing this inefficient use of the available clusters is one motivation behind the RATIO-GREEDY and STAR algorithms presented below.

If $k < d$, on the other hand, DOMINANCE must be applied on a transformed version of $V$ in $\mathbb{R}^k_{>0}$. Let $\mathbf{u} = \sum_{\mathbf{v}\in V} \mathbf{v}$ and assume, without loss of generality, that $u_i \geq u_{i+1}\ \forall\ i = 1, \ldots, d-1$. Then, each vector $\mathbf{v} = (v_1, \ldots, v_d) \in V$ will be transformed into $\mathbf{v}' = (v_1, \ldots, v_{k-1}, \sum_{i=k}^{d} v_i)$. That is, $\mathbf{v}'$ shares the same first $k-1$ components with $\mathbf{v}$, while its $k$-th and

last component is a sum of the $d - k$ remaining components from $\mathbf{v}$. The pseudocode for DOMINANCE is presented below.

**Algorithm 2** DOMINANCE$(V, k)$

**Input**:
$V$: a set of $n$ vectors in $\mathbb{R}^d_{>0}$
$k$: an integer
**Procedure**:

$\mathcal{P} \leftarrow \{C_1, \ldots, C_k\}$, where $C_i = \{\} \; \forall \; i = 1, \ldots, k$
**if** $k \geq d$ **then**
    **for** $\mathbf{v} \in V$ **do**
        $c \leftarrow \arg\max_i \mathbf{v}_i$ (break ties arbitrarily)
        $C_c \leftarrow C_c \cup \{\mathbf{v}\}$
**else**
    $\mathbf{c} \leftarrow \sum_{\mathbf{v} \in V} \mathbf{v}$
    $s \leftarrow$ sorted list of indices $1, \ldots, d$ according to the size of elements in $\mathbf{c}$ (in descending order)
    **for** $\mathbf{v} \in V$ **do**
        $\mathbf{v}' \leftarrow \{\mathbf{v}_{s_1}, \ldots, \mathbf{v}_{s_{k-1}}, \sum_{j=k}^{d} \mathbf{v}_{s_j}\}$
        $c \leftarrow \arg\max_i \mathbf{v}'_i$ (break ties arbitrarily)
        $C_c \leftarrow C_c \cup \{\mathbf{v}\}$
**return** $\mathcal{P}$

### 4.1.1 Running time

If $k \geq d$, DOMINANCE will simply allocate each vector to a cluster corresponding to its dominant component, so it suffices to find the dominant component for each vector, at $\mathcal{O}(nd)$ running time. When $k < d$, there are the additional steps of sorting the indices of the sum vector $\mathbf{u}$, which is $\mathcal{O}(d \log d)$, and creating a vector $\mathbf{v}'$ for each $\mathbf{v} \in V$, which is $\mathcal{O}(nd)$ once all vectors are taken into account. Therefore, the algorithm presents a running time of $\mathcal{O}(d(n + \log d))$.

### 4.1.2 Approximation guarantees

Although very simple, DOMINANCE has some approximation guarantees for relevant impurity measures:

– Theorem 4.2 of (Cicalese et al. (2019)) (Theorems 1 and 2 in the paper's supplementary material) states that DOMINANCE is a 2-approximation algorithm for the $PMWGP$ when $d \leq k$, and a 3-approximation for the same problem in general.

– Theorems 3 and 5 in the supplementary material for (Cicalese et al. (2019)) state that DOMINANCE is a $2p$-approximation for the $PMWEP$, where $p = \log d + \log\left(\sum_{\mathbf{v}\in V} ||\mathbf{v}||_1\right)$ if $d \leq k$ and $p = \min\{\log k, \log d\} + \log\left(\sum_{\mathbf{v}\in V} ||\mathbf{v}||_1\right)$ in the general case.

– Theorem 6 in the supplementary material for (Cicalese et al. (2019)) states that, for the special case where all vectors in $V$ have the same $\ell_1$ norm, DOMINANCE is an $\mathcal{O}(\log n + \log d)$-approximation for the $PMWEP$.

## 4.2 The Poly algorithm

Theorem 7 of (Cicalese et al. (2019)) states the existence of a partition $\mathcal{P}$ of $V$ with the following characteristics:

– It has at most one *mixed* cluster — i.e., at most one cluster with vectors with different dominant components.

– Let $C \in \mathcal{P}$ be a mixed cluster (which implies all other clusters in $\mathcal{P}$ are *pure*, or not mixed). If $\mathbf{v} \in C$ and $\mathbf{v}' \in V \setminus C$ are $i$-dominant vectors (i.e., vectors for which the $i$-th component is dominant), then $\text{ratio}(\mathbf{v}) \leq \text{ratio}(\mathbf{v}')$, where $\text{ratio}(\mathbf{v}) = ||\mathbf{v}||_\infty / ||\mathbf{v}||_1$ — that is, the ratio of a vector $\mathbf{u}$ is its largest element divided by the sum of all of its elements, so that the larger the ratio is, the more representative the largest element of the vector is.

– It is an $\mathcal{O}\left(\log^2 d\right)$-approximation to the optimal solution to the $PMWEP$.

POLY is an algorithm that searches for $\mathcal{P}$ using dynamic programming and pruning to make sure a polynomial number of alternatives are considered. As such, it is a polynomial-time algorithm with a guaranteed $\mathcal{O}\left(\log^2 \min\{d, k\}\right)$-approximation to the $PMWEP$.

The full discussion of POLY is outside the scope of this thesis, and the interested reader is invited to consult Section 5 of (Cicalese et al. (2019)). For our purposes, it suffices to note that the running time of the algorithm, although polynomial, would be impractical in real-world conditions. The two algorithms presented below are adaptations that combine DOMINANCE and POLY to devise a fast, dominance-based algorithm that yields competitive results when compared to the state of the art in terms of information-theoretic clustering.

## 4.3 The Ratio-Greedy algorithm

Ratio-Greedy is a fast algorithm that behaves exactly like Dominance when $k \leq d$ and otherwise aims to approximate the procedure of Poly in an agglomerative, bottom-up fashion. Any partition returned by Ratio-Greedy will have the following characteristics:

1. All clusters will be pure.

2. If there is more than one pure cluster whose dominant components are the same, then the clusters can be sorted according to the importance of the dominant component, in the following sense: Suppose vectors $\mathbf{v}, \mathbf{u}, \mathbf{c}$ have the same dominant component, $j$, and that ratio($\mathbf{v}$) $>$ ratio($\mathbf{u}$) $>$ ratio($\mathbf{c}$). Then $\mathbf{v}, \mathbf{c} \in C \implies \mathbf{u} \in C$. In other words, for any two clusters $C$ and $C'$ of vectors with the same dominant component, there is a real number $r$ such that, without loss of generality, ratio($\mathbf{v}$) $\geq r \geq$ ratio($\mathbf{u}$) $\forall\ \mathbf{v} \in C, \mathbf{u} \in C'$.

The first characteristic is reminiscent of Dominance, which does not allow for mixed clusters, while the second one comes from the observations regarding the partition found when running Poly, in which vectors in the mixed cluster will have smaller ratios than vectors in the pure clusters.

Summarized in words, Ratio-Greedy works as follows: given a data set $V$, with $n$ vectors in $\mathbb{R}^d$, and an integer $k$:

1. Sort all vectors in $V$ in descending order according to their ratios.

2. Subdivide the sorted array of vectors into $d$ subarrays, so that all vectors in a given subarray have the same dominant component.

3. Treat each vector as a cluster, so that there are $n$ initial clusters in the partition. The partition's weighted entropy in this case is the sum of weighted entropies for all vectors in the data set.

4. While the number of clusters is larger than $k$, cluster together the two clusters that are adjacent in the same subarray and whose agglomeration would yield the smallest addition to the partition's weighted entropy.

The pseudocode for the algorithm is presented below.

**Algorithm 3** RATIO-GREEDY$(V, k)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$k$: an integer
**Procedure**:

1: **if** $k \leq d$ **then**
2: **return** DOMINANCE$(V, k)$
3: **else**
4: Start with $n$ clusters, one for each $\mathbf{v} \in V$
5: Sort all vectors $\mathbf{v} \in V$ according to their ratios, ratio$(\mathbf{v}) = \frac{||\mathbf{v}||_\infty}{||\mathbf{v}||_1}$
6: *// 7-10: split vectors into subarrays according to dominant components*
7: $C_i \leftarrow [\,] \; \forall \; i = 1, \ldots, d$
8: **for** $\mathbf{v} \in V$ **do**
9: $c \leftarrow \arg\max_i \mathbf{v}_i$ (break ties arbitrarily)
10: Append $\mathbf{v}$ to $C_c$
11: *// 12-18: cluster vectors in agglomerative fashion*
12: $k' \leftarrow n$
13: **while** $k' > k$ **do**
14: Find the pair of adjacent vectors $\{\mathbf{v}, \mathbf{u}\}$ in any $C_i \; \forall \; i = 1, \ldots, d$ that minimizes entropy$(\mathbf{v} + \mathbf{u})$ − entropy$(\mathbf{v})$ − entropy$(\mathbf{u})$
15: $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{u}$
16: Add $\mathbf{u}$ to the cluster of $\mathbf{v}$
17: Remove $\mathbf{u}$ from its subarray
18: $k' \leftarrow k' - 1$
19: **return** the set of clusters found

### 4.3.1 Implementation analysis

The main concern in the implementation of RATIO-GREEDY is how to efficiently find, among all agglomeration candidates between existing clusters, the one that yields the smallest addition to the partition's weighted entropy.

When two clusters are agglomerated, the additional weighted entropy incurred by agglomerating the new cluster with its neighbors must be recalculated. For example, suppose a list of vectors $\mathbf{v} = [0.9, 0.1]$, $\mathbf{u} = [0.9, 0.1]$, and $\mathbf{c} = [0.6, 0.4]$. Agglomerating $\mathbf{v}$ and $\mathbf{u}$ will yield a new cluster whose sum vector is $[1.8, 0.2]$, and the additional weighted entropy incurred by this agglomeration is 0. However, while agglomerating $\mathbf{c}$ with any of the original clusters would yield an additional weighted entropy of 0.18, agglomerating it with the new cluster would yield an additional weighted entropy of 0.26.

Due to this updating, it is not possible to sort the agglomerations from the outset: the sorted list would have to be modified after each agglomeration is performed. Therefore, to efficiently find the best pair of clusters to agglomerate at each point in the algorithm, these cluster candidates are organized in a heap

according to the additional entropy yielded by their agglomeration.

The algorithm must keep track of where each neighbor to an agglomeration candidate is in the heap. When the heap's root is removed (that is, when the least costly agglomeration is selected), its neighbors must have their additional entropy updated, and the heap must be restored.

Item 4 in the textual description of Ratio-Greedy above implies that each cluster has at most two neighbors: the previous cluster in the subarray of clusters with the same dominant component, and the next one. (The first and last vectors in a subarray have a single neighbor.) Therefore, each agglomeration candidate has at most two neighbors in the heap.

For example, suppose the first agglomeration to be performed is between $C_{3,2}$ and $C_{3,3}$ — that is, the second and third vectors for which the third component is dominant, sorted in descending order by the ratio $||\mathbf{v}||_\infty / ||\mathbf{v}||_1$. Then, the cost of agglomerating vectors $C_{3,1}$ and $C_{3,2}$ must be updated, as well as the cost of agglomerating vectors $C_{3,3}$ and $C_{3,4}$.

### 4.3.2 Running time

Item 1 in the description of Ratio-Greedy above includes:

1. calculating ratio($\mathbf{v}$), which takes $\mathcal{O}(d)$ time for each vector, or $\mathcal{O}(nd)$ time overall.

2. sorting the vectors according to their ratios, which takes $\mathcal{O}(n \log n)$ time.

Items 2 and 3 are achieved in a single pass through the sorted vectors, at $\mathcal{O}(n)$ time.

Since each cluster can be agglomerated with at most two other clusters, and at the outset each vector is in a different cluster, there are $\mathcal{O}(n)$ agglomeration candidates. For each candidate, the cost of the agglomeration must be calculated — which includes calculating the weighted entropy of each vector, at $\mathcal{O}(d)$ time. Therefore, it takes $\mathcal{O}(nd)$ time to add all agglomeration candidates to the heap, and initializing the heap takes $\mathcal{O}(n \log n)$ time.

When clustering two vectors together, the following operations must be performed:

1. Retrieve the pair of clusters whose agglomeration will increase the partition's weighted entropy by the smallest amount. This amounts to removing the heap's root, at $\mathcal{O}(\log n)$ time.

2. Recalculate the additional entropy of clustering the new vector (the sum of both vectors being clustered) with each of its neighbors (at most two). This involves calculating the entropy of the new vector, at $\mathcal{O}(d)$ time.

3. Restore the heap after recalculating the additional entropy of the agglomeration candidates that include the newly formed cluster. Again, there are at most two such candidates, and restoring the heap for each of them takes $\mathcal{O}(\log n)$ time.

These operations will be performed $\mathcal{O}(n)$ times, until there are no more than $k$ clusters.

Therefore, both items 1 and 4 of Ratio-Greedy take $\mathcal{O}(n(\log n + d))$ time — the time complexity of the algorithm, since items 2 and 3 take only $\mathcal{O}(n)$ time.

### 4.3.3 Approximation guarantees

Ratio-Greedy does not allow for mixed clusters, since it will only cluster together vectors that are in the same subarray, and all vectors in a given subarray have the same dominant component. It can be seen as an extension of Dominance that avoids leaving empty clusters — and it does revert to Dominance when $k \leq d$. Splitting a cluster into a number of non-empty subclusters will never increase the partition's weighted entropy, due to the superadditivity properties of weighted impurity measures (see Lemma 1 of (Cicalese et al. (2019))); therefore, Ratio-Greedy enjoys the same approximation guarantees of Dominance for the PMWEP.



## 4.4 The Star algorithm

While Ratio-Greedy does not allow for mixed clusters, Theorem 5.1 from (Cicalese et al. (2019)) states that a partition with at most one mixed cluster will approximate the optimal partition for the PMWIP. Therefore, we would like to expand the search space of our algorithm so that mixed clusters are a possibility.

To do so, we consider an additional characteristic of the partition that Poly returns, namely that vectors in the mixed cluster (if there is one) have lower ratios than those in pure clusters. That is, given two vectors $\mathbf{v}, \mathbf{u}$ with the same dominant component $i$, $\text{ratio}(\mathbf{v}) > \text{ratio}(\mathbf{u})$, and a mixed cluster $C$, then $\mathbf{v} \in C \implies \mathbf{u} \in C$, and $\mathbf{u} \notin C \implies \mathbf{v} \notin C$.

The STAR algorithm is an expansion of RATIO-GREEDY that allows for mixed clusters comprising vectors with the lowest ratios in $V$. We do this by augmenting the neighborhood of the algorithm: all vectors with the lowest ratio among vectors with a given dominant component will be connected to each other. Figure 4.1 compares the neighborhoods defined by RATIO-GREEDY and STAR, and the pseudocode for the latter is presented below.

**Algorithm 4** STAR$(V, k)$

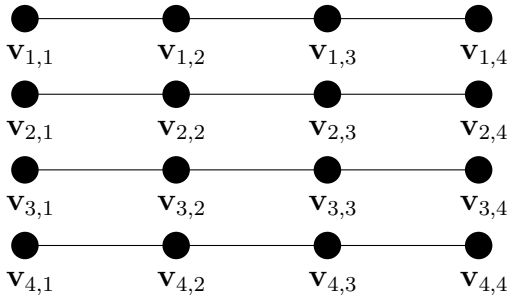
**Input**:

$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$

$k$: an integer

**Procedure**:

1: Start with $n$ clusters, one for each $\mathbf{v} \in V$
2: Sort all vectors $\mathbf{v} \in V$ according to their ratios, ratio$(\mathbf{v}) = \frac{||\mathbf{v}||_\infty}{||\mathbf{v}||_1}$
3: *// 4-7: split vectors into subarrays according to dominant components*
4: $C_i \leftarrow [\,] \; \forall \; i = 1, \ldots, d$
5: **for** $\mathbf{v} \in V$ **do**
6: $\quad c \leftarrow \arg\max_i \mathbf{v}_i$ (break ties arbitrarily)
7: $\quad$ Append $\mathbf{v}$ to $C_c$
8: *// 9-14: find agglomeration candidates*
9: N = {}
10: **for** $i \in \{1, \ldots, d\}$ **do**
11: $\quad$ **for** $j \in \{1, \ldots, |C_i| - 1\}$ **do**
12: $\quad\quad N \leftarrow N \cup \{C_{i,j}, C_{i,j+1}\}$ *// clustering vectors with the same dominant component*
13: $\quad$ **for** $j \in \{i + 1, \ldots, d\}$ **do**
14: $\quad\quad N \leftarrow N \cup \{C_{i,|C_i|}, C_{j,|C_j|}\}$ *// clustering vectors with lowest ratios*
15: *// 16-23: cluster vectors in agglomerative fashion*
16: $k' \leftarrow n$
17: **while** $k' > k$ **do**
18: $\quad$ Find $\{\mathbf{v}, \mathbf{u}\} \in N$ that minimizes entropy$(\mathbf{v} + \mathbf{u})$ − entropy$(\mathbf{v})$ − entropy$(\mathbf{u})$
19: $\quad \mathbf{v} \leftarrow \mathbf{v} + \mathbf{u}$
20: $\quad \mathbf{u} \leftarrow \mathbf{u} + \mathbf{v}$
21: $\quad$ Join the clusters of $\mathbf{u}$ and $\mathbf{v}$ to each other
22: $\quad N \leftarrow N \setminus \{\mathbf{v}, \mathbf{u}\}$
23: $\quad k' \leftarrow k' - 1$
24: **return** the set of clusters found




$\mathbf{v}_{i,j}$ is the vector with dominant component $i$ with the $j$-th largest ratio $||\mathbf{v}||_\infty/||\mathbf{v}||_1$. An edge connecting nodes indicates the corresponding vectors are neighbors for the purposes of the algorithm.



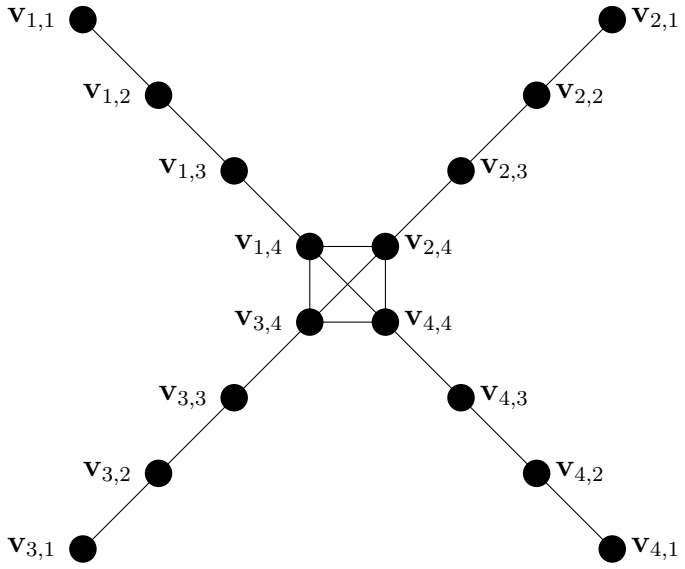



Figure 4.1: A representation of the neighborhoods defined by Ratio-Greedy (above) and Star (below).

### 4.4.1 Running time

Other than the definition of neighborhoods, STAR is identical to RATIO-GREEDY: once the agglomeration candidates are defined, the agglomeration that yields the smallest increase in the partition's weighted entropy is applied at each step, until $k$ clusters remain. However, while in RATIO-GREEDY each vector has at most two neighbors (the previous and next vectors in the subarray of vectors with the same dominant component), in STAR there may be as many as $d$ vectors (the ones with the smallest ratio for each dominant component) with as many as $d$ neighbors (the previous vector in the subarray for vectors with the same dominant component, plus the $d-1$ last vectors in the other subarrays). This means the number of initial agglomeration candidates grows — and, more importantly, this number becomes dependent not only on $n$ but also on $d^2$ (since each of the $d$ last vectors will be connected to the other $d-1$ last vectors).[1]

Since it takes $\mathcal{O}(d)$ time to calculate the additional weighted entropy for each agglomeration candidate, the construction of the heap for STAR takes $\mathcal{O}((n+d^2)d)$ time. There are $\mathcal{O}(n)$ iterations to go from the $n$ initial clusters (one for each vector) to the $k$ clusters in the final partition; in each of these iterations, at most $d$ elements in the heap will be updated, at the cost of $\mathcal{O}(d+\log(n+d^2))$ time (as it takes $\mathcal{O}(d)$ time to update an agglomeration candidate and $\mathcal{O}(\log(n+d^2))$ time to restore the heap once a candidate is updated). Therefore, the number of operations necessary to perform STAR is $\mathcal{O}(d(nd+n\log(n+d^2)+d^2))$.

### 4.4.2 Implementation analysis

The variable number of neighbors in the STAR heap makes its implementation more complicated than that of RATIO-GREEDY. There needs to be an array of the neighbors for each agglomeration candidate: each candidate may have from 1 to $d$ neighbors. Figure 4.4.2 shows a simplified example of how the number of neighbors changes when two elements with the smallest ratio in their respective subarrays are clustered together.

[1] To be more precise, assuming that there are no empty subarrays — i.e., that each component is the dominant one for at least one vector in the data set — then there will be $n+\frac{d^2-3d}{2}$ agglomeration candidates at the outset of the clustering process, when each vector is in a cluster of its own: $n-d$ candidates corresponding to connections of vectors with the same dominant component, plus $\frac{d(d-1)}{2}$ connections between the vectors with the smallest ratio for each dominant component.

$\mathbf{v}_{i,j}$ is the vector with dominant component $i$ with the $j$-th largest ratio $||\mathbf{v}||_\infty/||\mathbf{v}||_1$. An edge connecting nodes indicates the corresponding vectors are neighbors for the purposes of the algorithm.

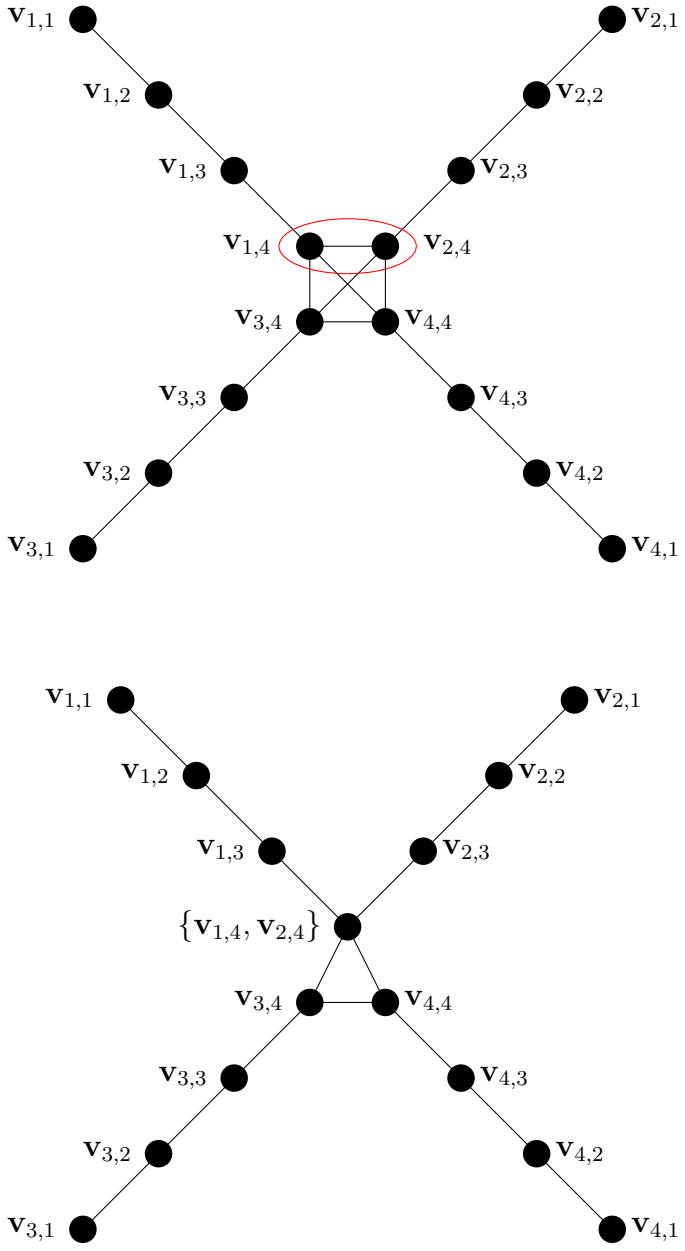



Figure 4.2: Change of number of neighbors after agglomeration in STAR.

Auxiliary arrays are implemented to keep track of the neighbors of each element: for each vector, one array indicates in how many agglomeration candidates the vector is in, another indicates to which other vectors it can be clustered, and a third array indicates in which elements of the heap the vector is represented. The first of these auxiliary arrays has $n$ elements, one for each vector in $V$; the other two have $nd$ elements, since each of the $n$ vectors may have at most $d$ neighbors and therefore may be in at most $d$ candidates in the heap.

When updating the heap after two clusters are agglomerated, we found it more straightforward to identify all elements in the heap to which one of the agglomerated clusters belong, remove them from the heap, and then add new elements considering the neighbors of the new, agglomerated cluster. This procedure still takes $\mathcal{O}(d \log(n + d^2))$ time, as it removes and adds $\mathcal{O}(d)$ elements to the heap, but it is probably less efficient than directly updating the (at most) $d$ elements in the heap that will be affected by agglomerating two clusters. The running time analyses of Ratio-Greedy and Star, presented above, make us suppose that the latter would be slower than the former even if implemented in the fastest possible way; however, this possible inefficiency in its implementation must be taken into account when comparing the running times of both algorithms in Chapter 6.

On the other hand, the running time of Star is very dependent on which agglomeration candidates are selected. If only candidates with one or two neighbors are selected, for instance (that is, if the elements with smallest ratios are never clustered to other elements), then there is an $\mathcal{O}(1)$ number of updated elements per iteration, and each updating takes $\mathcal{O}(\log(n+d^2))$. There would still be, however, a dependence on $d^2$ instead of $d$ when compared to Ratio-Greedy, due to the size of the heap that needs to be constructed and, at each iteration, updated.

One can see the additional running time of Star when compared to Ratio-Greedy as the "price to pay" for allowing mixed clusters instead of only pure ones. It is because vectors with the lowest elements in each subarray are connected to each other that the heap from Star has more elements than the one from Ratio-Greedy, and the additional agglomeration candidates that may need to be updated are those that connect vectors (or clusters) with different dominant components.

### 4.4.3 Approximation guarantees

While Ratio-Greedy may be considered an expanded version of Dominance that takes care of the issue of empty clusters, Star is better thought of as a simplified version of Poly, that allows us to find, in a short amount of time, a good partition with the same characteristics as the best partition from the more complex algorithm.

This simplification, however, means that Star does not enjoy the same approximation guarantees as Poly. And, as the guarantees for Dominance and Ratio-Greedy rely on the fact that only pure clusters are returned by these algorithms, and this is not true for Star, it does not enjoy the approximation guarantees for those algorithms either. We'll see in Chapter 6, however, that in practice the results from Star are typically very close to those of Ratio-Greedy.

# 5
# Iteration-based algorithms

For clustering problems such as the ones of interest in this thesis, the best results in the literature come from algorithms based on an iterative procedure:

1. **Initialization**: Define a set of initial cluster centers.
2. **Expectation**: Assign each element to the cluster whose center is closest to it, according to a pre-defined dissimilarity measure.
3. **Maximization**: Recalculate the cluster centers as an average of the elements in the cluster.
4. Repeat steps 2 and 3 until convergence, or up to a pre-specified number of iterations.

The algorithm was first described in (Lloyd (1982)) with the squared Euclidean distance as dissimilarity measure, but it can be used with any Bregman divergence (Banerjee et al. (2005)). As mentioned in Section 2.2.2, one can minimize the weighted entropy of the partition using the Kullback-Leibler divergence as a metric. Therefore, it is natural, in the context of the PMWEP, to compare the results of the dominance-based algorithms presented in the previous chapter with those of iteration-based algorithms using the Kullback-Leibler divergence.

We present below three such iteration-based algorithms:

– LloydKL++ is the original algorithm from (Lloyd (1982)) with the ++ initialization from (Arthur & Vassilvitskii (2007)), which guarantees an $\mathcal{O}(\log k)$ approximation to the optimal partition, and using the Kullback-Leibler divergence instead of the squared Euclidian distance to measure the dissimilarity between vectors.
– DivisiveClustering (Dhillon et al. (2003)) uses as initialization a procedure that closely resembles the Dominance algorithm.
– CoresetClustering (Bachem et al. (2018)) applies LloydKL++ to a coreset of the original set of vectors — that is, to a subset of the vectors chosen in such a way that results of the algorithm applied to the subset

approximate (with some probability) the results of the algorithm applied to the whole set.

Of these three algorithms, the first two are predominantly concerned with returning the best possible partition, and as such their running time can be quite high. The third one can be seen as a compromise between quality (in terms of the partition's weighted entropy) and efficiency (in terms of the algorithm's running time). By comparison, the dominance-based algorithms used in experiments in this thesis, RATIO-GREEDY and STAR, lean much more heavily on the side of efficiency, trying to return very quickly a partition that is not too far from the optimal one.

## 5.1 Lloyd's algorithm with ++ initialization and Kullback-Leibler as dissimilarity measure

Algorithm 5, presented below, corresponds to the original algorithm from (Lloyd (1982)), with the Kullback-Leibler divergence used instead of the squared Euclidean distance. This algorithm takes as inputs, other than the set of vectors to be clustered, a set of $k$ initial cluster centers; in the original formulation of the algorithm, these centers are chosen at random among the vectors to be partitioned.

**Algorithm 5** LLOYDKL$(V, C, m)$

**Input**:
$V$: a set of $n$ vectors in $\mathbb{R}^d_{>0}$
$C$: a set of $k$ initial cluster centers, represented as vectors in $\mathbb{R}^d_{>0}$
$m$: an integer
**Procedure**:

$i \leftarrow 0$
**repeat**
  *// Expectation step*
  $\mathcal{P} \leftarrow [\{\} \; \forall \; j = 1, \ldots, k]$
  **for** $\mathbf{v} \in V$ **do**
    $j \leftarrow \arg\min_p \{KL(\mathbf{v}, C_p) \; \forall \; p = 1, \ldots, k\}$
    $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{\mathbf{v}\}$
  *// Maximization step*
  **for** $j = 1, \ldots, k$ **do**
    $C_j \leftarrow \frac{1}{|\mathcal{P}_j|} \sum_{\mathbf{v} \in \mathcal{P}_j} \mathbf{v}$
  $i \leftarrow i + 1$
**until** convergence or $i = m$
**return** $\mathcal{P}$

The initialization step described in (Arthur & Vassilvitskii (2007)) and presented in Algorithm 6 below leads to an approximation guarantee of

$\mathcal{O}(\log k)$ to the optimal partition. This algorithm takes as input the original set of vectors $V$ and an integer $k$, and returns a set of $k$ vectors from $V$ such that a partition defined by this set of vectors as cluster centers already enjoys this approximation guarantee. Since Algorithm 5 can only iteratively improve on the sum of dissimilarities from the original partition, the approximation guarantee is achieved.

**Algorithm 6** INIT++$(V, k)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$k$: an integer
**Procedure**:

```
P = [1/n ∀ i = 1, ..., n]
v ← a vector sampled from V with probability P_i ∀ i = 1, ..., n
C = {v}
while |C| < k do
    for i = 1, ..., n do
        P_i ← min_{c∈C}{KL(V_i, c)}
    v ← a vector sampled from V with probability P_i / ∑P ∀ i = 1, ..., n
    C ← C ∪ {v}
return C
```

The main idea of Algorithm 6 is to sample new cluster centers proportionally to their dissimilarity with the cluster centers already defined. Therefore, vectors that are "farther away" (in the sense of the dissimilarity being used) from the current cluster centers will be sampled with higher probability, and the sum of dissimilarities between vectors and the cluster centers to which they are closest will decrease.

The algorithm we propose to compare with our dominance-based algorithms is simply Algorithm 5 with the cluster centers initialized by Algorithm 6, as presented in Algorithm 7.

**Algorithm 7** LLOYDKL++$(V, k, m)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$k, m$: integers
**Procedure**:

```
C ← INIT++(V, k)
return LLOYDKL(V, C, m)
```

### 5.1.1 Implementation analysis

One possible issue with Lloyd's algorithm is the occurrence of empty clusters. Over the iterations performed by the algorithm, it is possible that one or more cluster centers will have no vectors attached to it; once this happens, no vector will ever be assigned to that cluster center again, as the dissimilarity between the cluster center of an empty cluster and a vector is not defined.

To deal with this issue, we added to the implementation of the algorithm a verification step that checks whether a cluster center is empty. If this is the case, one vector at random will be removed from another cluster and defined as a stand-alone cluster (and therefore the cluster center will be the vector itself).

When clustering with the goal of minimizing the sums of the Kullback-Leibler divergences (or of any Bregman divergences) between the elements and the cluster centers, having empty clusters will never mean having a smaller sum of divergences. A similar statement is valid when clustering to minimize impurity instead of the sum of dissimilarities: having empty clusters will never be beneficial. Therefore, this additional step aims to improve the final results of the clustering method implemented.



### 5.1.2 Running time

Init++ takes $\mathcal{O}(nkd)$ time: after each new cluster center is found, which occurs $\mathcal{O}(k)$ times, the Kullback-Leibler divergence between all $\mathcal{O}(n)$ vectors not assigned as centers and the new cluster center must be updated, at $\mathcal{O}(d)$ time. Each iteration of LloydKL++ also takes $\mathcal{O}(nkd)$ time, as the expectation step involves recalculating the Kullback-Leibler divergence between all vectors in $V$ and the $k$ cluster centers, to find the new assignment of each vector.

The number of iterations required for convergence is exponential in the worst case (Vattanni (2005)). However, the smoothed running time for the algorithm is polynomial (Arthur et al. (2009)), and in practice the number of iterations is usually limited by a hard number or by a rule that stops the algorithm once the gain from performing additional iterations falls below a pre-established threshold. In our experiments in Chapter 6, we present the results of LloydKL++ running for up to 100 iterations.

## 5.2 Divisive information-theoretic clustering

DivisiveClustering, presented in (Dhillon et al. (2003)) as Divisive_Information_Theoretic_Clustering, uses the same expectation-maximization procedure as LloydKL++, the only difference between the two algorithms being their initialization steps. While LloydKL++ relies on an adapted version of the ++ initialization using the Kullback-Leibler divergence, DivisiveClustering initializes the clusters with a procedure that is a simplified version of the Dominance algorithm presented in Chapter 4 above:

- Assign every vector $\mathbf{v} \in V$ to $\mathcal{P}_i$ such that $\mathbf{v}_i = \max_{i=1,\dots,d}\{\mathbf{v}_i\}$.
- If $k \geq d$, split each cluster arbitrarily into at least $\lfloor k/d \rfloor$ clusters.
- If $k < d$, merge the $d$ clusters to obtain $k$ initial clusters.

This initialization method is presented in Algorithm 8 below.

**Algorithm 8** InitDivisive$(V, k)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$k$: an integer
**Procedure**:

```
if k ≤ d then
    P ← Dominance(V, k)
else
    P ← {{} ∀ i = 1, ..., k}
    Assign ⌊k/d⌋ clusters from P to each dominant component
    Assign one additional cluster from P to a different dominant component
  until all clusters from P are assigned to a dominant component
    for v ∈ V do
        i = arg max_i v_i (break ties arbitrarily)
        S ← the set of clusters from P assigned to i
        C ← C ∈ S | |C| = min{|C| ∀ C ∈ S} (break ties arbitrarily)
        C ← C ∪ {v}
c ← { (1/|C|) Σ_{v∈C} v ∀ C ∈ P }
return c
```

### 5.2.1 Implementation analysis

In our implementation, $k \leq d$ is treated exactly as in Dominance, by reducing the dimensionality of the original vectors from $V$. If $k > d$, roughly the same number of clusters is reserved to partition vectors with any given dominant component, and the initialization procedure assigns each vector sequentially to one of the clusters for its dominant component.

Once the initial cluster centers are found, DivisiveClustering passes them to LloydKL alongside $V$ to find a suitable partition. Therefore, other than the initialization step, DivisiveClustering behaves the same as LloydKL++. The algorithm for DivisiveClustering is presented below as Algorithm 9.

**Algorithm 9** DivisiveClustering$(V, k, m)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$k, m$: integers
**Procedure**:
1: $C \leftarrow$ InitDivisive$(V, k)$
2: **return** LloydKL$(V, C, m)$

(Dhillon et al. (2003)) suggests iterating until the gain from one iteration to the next is small. In line with our implementation of LloydKL++, we set a maximum number of iterations instead (interrupting the algorithm before the maximum number of iterations if convergence is reached).



### 5.2.2 Running time

The expectation-maximization phase of DivisiveClustering is the same as that of LloydKL++, and therefore the same running-time analysis applies.

In the initialization phase, InitDivisive will run Dominance if $k \leq d$, and in this situation Dominance runs in $\mathcal{O}(d(n + \log d))$ time (see Section 4.1.1). If $k > d$, InitDivisive will find the dominant component of each vector $\mathbf{v} \in V$ and assign it to the least populated cluster of vectors with that dominant component. By adding each vector to a cluster in sequence and keeping track of which cluster, for each dominant component, should receive the next vector, this operation can be performed in $\mathcal{O}(nd)$ time.

Since the running time of Init++ is $\mathcal{O}(nkd)$, the initialization phase of DivisiveClustering will be faster, as long as $\log d < nk$ (which will likely be the case in any practical application of the algorithm). However, the expectation-maximization phase dominates the running time of both algorithms, and in our experiments we don't see much difference between their running times (see Chapter 6).

## 5.3 Clustering via lightweight coresets

One main concern involving algorithms based on iteratively finding better cluster centers and assigning points to them is their running time. Even when limiting the number of iterations performed, such algorithms rely on reevaluating the divergence between each vector and each cluster center, so that even a single iteration runs in $\mathcal{O}(nkd)$ time, which can be quite expensive.

One way to reduce running time is to run the algorithm using as input not the whole set $V$ of size $n$, but a subset $S \subset V$ of size $m$. In doing so, each iteration takes $\mathcal{O}(mkd)$ time to run, which can be more manageable as long as $m << n$. There is, however, the matter that the answer found for $S$ should also apply for $V$.

Given a task to be performed on $V$, a *coreset* of $V$ is a weighted subset $S \subset V$ such that, when performing the same task on $S$, we find an answer whose quality is a good approximation of the quality of the answer that would be retrieved by performing the task on $V$. In our case, a coreset of $V$ is a weighted subset $S \subset V$ such that the partition $\mathcal{P}'$ of $V$ found by applying a clustering algorithm on $S$ has a weighted entropy that approximates that of the partition $\mathcal{P}$ of $V$ found by applying the same algorithm on $V$.

Using the Kullback-Leibler divergence as a dissimilarity measure, the formal definition of a lightweight coreset from (Bachem et al. (2018)) is as follows: let $V$ be a set of $n$ vectors and $Q$ any set of at most $k$ vectors (cluster centers) in $\mathbb{R}^d_{>0}$. Furthermore, let

$$KL_V(Q) = \sum_{\mathbf{v} \in V} \min \left\{ KL(\mathbf{v}, \mathbf{c}) \; \forall \; \mathbf{c} \in Q \right\}.$$

Then, a weighted subset $S \subset V$ is an $(\epsilon, k)$-lightweight coreset of $V$ if

$$|KL_V(Q) - KL_S(Q)| \leq \frac{\epsilon}{2} KL_V(Q) + \frac{\epsilon}{2} KL_V(\{\mu(V)\}), \tag{5-1}$$

where $\mu(V) = \frac{1}{|V|} \sum_{\mathbf{v} \in V} \mathbf{v}$.

The first term in the right-hand side of 5-1 (the *multiplicative error*) "allows the approximation error to scale with the quantization error", while the second term (the *additive error*) "scales with the variance of the data" (Bachem et al. (2018)). In other words, the approximation error may be larger the more variance the full data set presents (additive error) or the larger the quantization error on the full data set is (multiplicative error).

(Lucic et al. (2016)) presents a method for building coresets when clustering for the minimization of Bregman divergences such as the Kullback-Leibler divergence. However, the coreset construction method still involves $k$

passes through the full data set, making finding the coreset itself slow for many practical purposes. In (Bachem et al. (2018)), the same authors present a much faster, and embarrassingly parallel, method for building lightweight coresets for the task of Bregman clustering.

The algorithm CoresetClustering consists of three steps:

1. Construct a weighted coreset $S$ of $V$, $|S| << |V|$.
2. Find $k$ cluster centers by applying a weighted version of LloydKL++ to $S$.
3. Partition $V$ according to the cluster centers found in the previous step.

The coreset construction algorithm, LightCoreset, is presented below as Algorithm 10. It consists of calculating $\mu$, the mean of $V$, and sampling $m$ weighted vectors according to probabilities that depend on the Kullback-Leibler divergence between each vector $\mathbf{v} \in V$ and $\mu$.

**Algorithm 10** LightCoreset$(V, m)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$m$: an integer
**Procedure**:

1: $\mu \leftarrow \frac{1}{|V|} \sum_{\mathbf{v} \in V} \mathbf{v}$
2: $q \leftarrow \left\{ \frac{1}{2|V|} + \frac{KL(\mathbf{v},\mu)^2}{2 \sum_{\mathbf{v} \in V} KL(\mathbf{v},\mu)^2} \; \forall \; \mathbf{v} \in V \right\}$
3: $S \leftarrow m$ vectors from $V$ sampled with probability $\frac{q(\mathbf{v})}{\sum_{\mathbf{v}' \in V} q(\mathbf{v}')} \; \forall \; \mathbf{v} \in V$
4: $W \leftarrow \frac{1}{mq(\mathbf{v})} \; \forall \; \mathbf{v} \in S$
5: **return** $S, W$

Theorem 2 of (Bachem et al. (2018)) states that the size of the subset needed to guarantee (with a certain probability) that LightCoreset returns a coreset is $\mathcal{O}(dk \log k)$. Importantly, it does not depend on $n$.

### 5.3.1 Implementation analysis

Since each vector in the coreset is weighted, we need a version of Lloyd's algorithm that handles weighted vectors. WeightedLloydKL, presented below as Algorithm 11, is a variation of LloydKL in which the maximization step takes into account the weights of each vector when updating the cluster centers.

As shown in the pseudocode for CoresetClustering, presented below as Algorithm 12, clustering via coresets amounts to finding a coreset of the

**Algorithm 11** WeightedLloydKL$(V, W, C, m)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$W$: an array of weights for each vector in $V$
$C$: a set of $k$ initial cluster centers, represented as vectors in $\mathbb{R}^d_{>0}$
$m$: an integer
**Procedure:**

$i \leftarrow 0$
**repeat**
  $\mathcal{P} \leftarrow [\{\} \; \forall \; j = 1, \ldots, k]$
  *// Expectation step*
  **for** $\mathbf{v} \in V$ **do**
    $j \leftarrow \arg\min_p \{KL(\mathbf{v}, C_p) \; \forall \; p = 1, \ldots, k\}$
    $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{\mathbf{v}\}$
  *// Weighted maximization step*
  **for** $j = 1, \ldots, k$ **do**
    $C_j \leftarrow \frac{1}{|\mathcal{P}_j|} \sum_{p \mid V[p] \in \mathcal{P}_j} V[p] \times W[p]$
  $i \leftarrow i + 1$
**until** convergence or $i = m$
**return** $\mathcal{P}$



original data set and passing it to the original algorithms — in this case, Init++ to find the initial cluster centers and WeightedLloydKL to find the partition of the coreset. Then, using the cluster centers thus found, the maximization step is performed once to partition the whole data set.

Our implementation treats the size of the coreset as an input to the algorithm. An alternative implementation would be to derive the necessary size from the data, passing as parameters a desired error $\epsilon$ and the probability $\gamma$ that the error from using the coreset will be $\leq \epsilon$. We follow the presentation of the algorithm in (Bachem et al. (2018)), in which the authors experiment with different sizes of coresets for the same data set, without calculating the size required to achieve the approximation guarantees.

### 5.3.2 Running time

As mentioned above, the main purpose of applying an algorithm on a coreset instead of the full data set is efficiency: since the coreset is smaller than the original data set, the algorithm will run faster.

The analysis of the running time of CoresetClustering is very similar to that of LloydKL++, with the important difference that, when initializing the cluster centers and finding the optimal partition, we are dealing with a coreset of size $m$ instead of a data set of size $n$. Therefore, the initialization

**Algorithm 12** CoresetClustering$(V, k, m, p)$

**Input**:
$V$: an array of $n$ vectors in $\mathbb{R}^d_{>0}$
$k, m, p$: integers
**Procedure**:

$S, W \leftarrow$ LightCoreset$(V, m)$
$C \leftarrow$ Init++$(S, k)$
$\mathcal{P}' \leftarrow$ WeightedLloydKL$(S, W, C, p)$
$C \leftarrow \left[\frac{1}{n}\sum_{\mathbf{v}\in\mathcal{P}'_i}\mathbf{v}\ \forall\ i = 1, \ldots, k\right]$
*// single maximization step*
$\mathcal{P} \leftarrow [\{\}\ \forall\ i = 1, \ldots, k]$
**for** $\mathbf{v} \in V$ **do**
    $i \leftarrow \arg\min_i \{KL(\mathbf{v}, C_i)\ \forall\ i = 1, \ldots, k\}$
    $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \{\mathbf{v}\}$
**return** $\mathcal{P}$

step, as well as each iteration of the algorithm, will take $\mathcal{O}(mkd)$ time.

We must also take into account the time needed to construct the coreset. LightCoreset involves calculating the mean vector of $V$ and then calculating the probability that each vector will be sampled, and both these operations take $\mathcal{O}(nd)$ time. Then, $m$ vectors are sampled from $V$ and the weights of these vectors are calculated, at $\mathcal{O}(m)$ time. Since $m \leq n$ (and in fact, for any efficiency gain to be significant, $m << n$), the overall running time of LightCoreset is $\mathcal{O}(nd)$. We note that (Bachem et al. (2018)) mention that LightCoreset is embarrasingly parallel, but analyzing the performance gain of parallelizing the coreset construction step is outside the scope of this thesis.

The final step of CoresetClustering is assigning each vector in $V$ to a cluster center. This involves calculating the Kullback-Leibler divergence between each $\mathbf{v} \in V$ and each cluster center, at $\mathcal{O}(nkd)$ time.

Therefore, the upper bound for the running time of CoresetClustering ends up being the same as that of LloydKL++. However, there is a single iteration in CoresetClustering that takes $\mathcal{O}(nkd)$ time, and several such iterations in LloydKL++. As the experiments presented in Chapter 6 demonstrate, this makes a great difference in the actual running time of both algorithms.

# 6
# Experimental results

In this chapter we present the results of experiments performed with the dominance-based algorithms Ratio-Greedy (Algorithm 3) and Star (Algorithm 4), both presented and analyzed in Chapter 4. As basis of comparison, we use the iteration-based algorithms LloydKL++ (Algorithm 7), DivisiveClustering (Algorithm 9), and CoresetClustering (Algorithm 12), presented and analyzed in Chapter 5.

There are notable shortcomings in the dominance-based methods when compared to iteration-based ones. The dominance-based methods analyzed here partition the data set based only on the dominant component and its overall importance (measured by the ratio $|\mathbf{v}|_\infty/|\mathbf{v}|_1$), and ignore all other information regarding the vectors to be clustered. The upside of such methods is that they can be much faster than the methods that iteratively improve on the partition until a local minimum is found. Therefore, the main goal of these experiments is to analyze the trade-off between quality (in terms of the partition's weighted entropy) and efficiency (in terms of the algorithm's running time).



## 6.1
## Data sets

We use three data sets in our experiments. The first two, 20 Newsgroups and RCV1, are text data sets traditionally used to evaluate classification models that aim to classify a text given the words it contains. The third data set, Poisson, is a synthetic data set similar to one used in (Lucic et al. (2016)) to evaluate a coreset clustering method using the Kullback-Leibler divergence as dissimilarity measure.

### 6.1.1
### The 20 Newsgroups data set

20 Newsgroups was collected in the 1990's by Ken Lang and has been widely used ever since in experiments involving text classification and feature selection — see, for example, (Joachims (1997)), (Baker & McCallum (1998)), (Slonim & Tishby (1999)), (Dhillon et al. (2003)), (Dasgupta et al. (2007)),

(Dai & Le (2015)), and (Ribeiro et al. (2016)). It includes approximately 18,000 texts (messages to newsgroups), evenly divided among 20 categories (the newsgroups to which the messages were originally sent). The subjects of many newsgroups are similar or overlap (for instance, **comp.os.ms-windows.misc** and **comp.windows.x**, or **rec.autos** and **rec.motorcycles**), which makes correctly classifying the texts a challenging task.

Several versions of the data set have been presented and used over the years. We use the data set as retrieved by the scikit-learn package for Python (Pedregosa et al. (2011)) from Jason Rennie's web-page (Rennie (2014)). We exclude from the data set the English stop-words as defined in scikit-learn, as well as words that appear in no more than 2 documents or in more than 95% of the documents.

We then find how many times each word is found in documents from a given class, smoothing the results by 1 (i.e., each word is considered to appear in each class at least once). The probability of sampling a word $w$ from a document of class $c$ is calculated by dividing the number of times $w$ is found in documents of class $c$ by the total number of words in all documents of class $c$. This probability is then multiplied by the probability that a given document is from class $c$ (the number of documents from class $c$ divided by the total number of documents). The end result is a data set with the probabilities that a document belongs to class $c$ given that word $w$ is in it.

After cleaning the documents as mentioned above, there remain 51840 unique words. The final data set, therefore, has 51840 rows and 20 columns. The smoothing step guarantees that all vectors (each representing a word) are in $\mathbb{R}^{20}_{>0}$.



### 6.1.2 The Reuters (RCV1) data set

The Reuters Corpus Volume 1 (RCV1) data set, first presented in (Lewis et al. (2004)), comprises over 800,000 news-wire stories from the Reuters news agency, divided in 103 non-exclusive categories. As is the case with 20 Newsgroups, RCV1 has been extensively used to benchmark classification and information retrieval models — see, for instance, (Genkin et al. (2007)), (Fan et al. (2008)), (Bottou (2010)), (Duchi et al. (2011)), (Hinton et al. (2012)), (Miyato et al. (2017)), and (Bottou et al. (2018)).

Unlike in 20 Newsgroups, texts in RCV1 can belong to more than a single class (in fact, the classification of texts in RCV1 follows a tree-like hierarchy, with different levels of classification; there are 103 classes in the final

level). For our purposes, we randomly assigned each document to one of the classes to which it originally belongs. As is the case for 20 NEWSGROUPS, our goal is not to ultimately classify the texts in their correct classes, but to find a good partition of the words according to the probability of finding them in a given class.

The original text files were directly downloaded from David D. Lewis's webpage (Lewis (2004)), and vectorized using Python's scikit-learn. Following the same procedure used for 20 NEWSGROUPS, we removed the words appearing in no more than 2 documents or in more than 95% of the documents, as well as the English stop-words as defined in scikit-learn. The end result is a matrix with 170,946 rows (words) and 103 columns (classes), where the element in the $j$-th column of the $i$-th row indicates the probability that a document belongs to class $j$ given that we find word $i$ in it. Smoothing guarantees that all vectors in the final data set are in $\mathbb{R}^{103}_{>0}$.

### 6.1.3 The Poisson data set

The final data set used in our experiments is a synthetic data set like the one used in (Lucic et al. (2016)) to evaluate the results of an initial algorithm based on coresets by the same authors of (Bachem et al. (2018)). The original paper presents a coreset construction method that is much slower than the one found in (Bachem et al. (2018)), leading to smaller gains when comparing the running time of the algorithm with those of state-of-the-art clustering methods such as LLOYDKL++ (Algorithm 7).

Among the seven data sets mentioned between (Lucic et al. (2016)) and (Bachem et al. (2018)), we chose the Poisson data set due to the fact that, in the original paper, the sum of the Kullback-Leibler divergences between each vector and the closest cluster center is used to gauge the quality of the partition returned by the algorithm under analysis. The comparison with an algorithm that aims to minimize the weighted entropy of the partition, such as RATIO-GREEDY and STAR (Algorithms 3 and 4), is therefore more natural than it would be for data sets evaluated using other measures of dissimilarity, such as the squared Euclidean distance.

Following the description in (Lucic et al. (2016)), we built a data set of 10,000 points from a mixture of 50 multivariate Poisson distributions in 10 dimensions. Each dimension is independently sampled, with the parameter of the Poisson distribution being sampled from a Gamma distribution with a shape parameter ($\alpha$) of 10 and a scale parameter ($\beta$) of $10^{-3}$. The data set was built using the Numpy package for Python (Virtanen et al. (2019)).

## 6.2 Results

In this section we present and discuss the results of the experiments performed with the three data sets described above. All experiments were performed in a 2018 MacBook Air with a 1,6 GHz Intel Core i5 and 8 GB of RAM, running the Mojave OS (version 10.14.6). The tables with the full results of the experiments can be found in Appendix A. The code for preparing the data sets and running the experiments can be found in `https://github.com/lmurtinho/thesis_info_clustering`.

### 6.2.1 20 Newsgroups

#### 6.2.1.1 Comparison between "full" iteration-based methods and dominance-based methods

Figure 6.1 shows the average entropy (over five runs) found when partitioning 20 Newsgroups using five different models: the two practical, dominance-based models introduced in Chapter 4 (Ratio-Greedy and Star) and the full versions of the three iteration-based models presented in Chapter 5 (LloydKL++, DivisiveClustering, and CoresetClustering). These are the "full" versions of these models in the sense that, unless the results converge before that, they are running the maximum number of iterations allowed in our experiments (100), and that, in the case of CoresetClustering, the largest coreset (with 5000 members) is being used.

For this data set, iteration-based methods consistently beat dominance-based methods, but by small margins, which tend to decrease as the number of clusters increases. The results from both dominance-based methods are very similar, and the same can be said for the results from the three iteration-based methods. As expected, CoresetClustering, which relies on a weighted sample of the full data set for building the partition, performs slightly worse than both DivisiveClustering and LloydKL++.

Figure 6.2 shows the average running time of the same five algorithms for partitioning 20 Newsgroups. DivisiveClustering and LloydKL++ are by far the most expensive methods, with a running time of up to 100 seconds for larger values of $k$. The fact that the initialization step of DivisiveClustering is much faster than that of LloydKL++ does not seem to make much difference in the running times of the full versions of the algorithms.

The dominance-based methods are much faster, with a small but clear

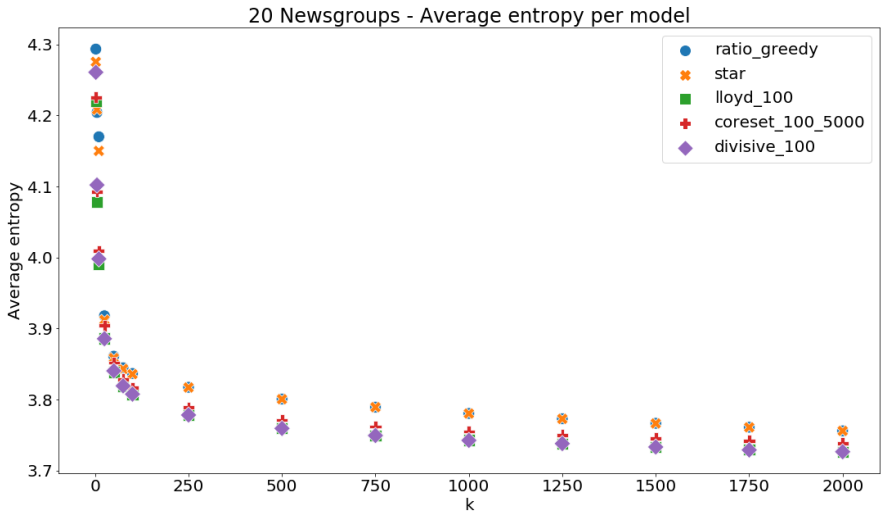



Figure 6.1: Average entropy for the partition of 20 Newsgroups (5 runs per model). Iterative models run for up to 100 iterations. The coreset has 5000 elements (approximately 10% of the elements in the original data set).



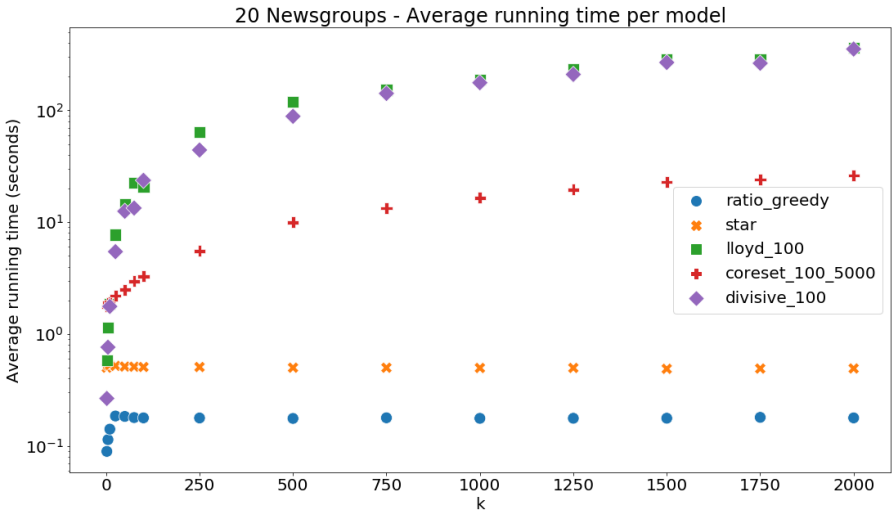



Figure 6.2: Average running time for partitioning 20 Newsgroups (5 runs per model). Iterative models run for up to 100 iterations. The coreset has 5000 elements (approximately 10% of the elements in the original data set).

advantage to Ratio-Greedy over Star — which is to be expected, since, as mentioned in Chapter 4, the latter allows for mixed partitions while the former doesn't. Finally, CoresetClustering is not as fast as the dominance-based methods, but it is much faster than the other iteration-based methods; reducing the size of the set used to find the cluster centers does lead to marked improvements on running times, even if the worst-case analyses for the three methods present the same bound.

#### 6.2.1.2 Comparison between "initial" iteration-based methods and dominance-based methods

Since iteration-based methods return partitions with smaller weighted entropies on average, comparing the full versions of these methods with dominance-based methods may overstate the penalty (in terms of running time) for achieving better results. It may be the case that the initial partition of an iteration-based method is already better than the partitions obtained by dominance-based methods, and that this initial partition is achieved faster. Therefore, we also present below charts comparing the results of dominance-based methods with the "initial" version of the iteration-based methods — that is, the version that simply returns the initial partition obtained, without any iteration to improve on the results.

Figure 6.3 shows the average weighted entropy (over 5 runs) of partitions of 20 Newsgroups obtained by the initial versions of iteration-based models, as well as the results for dominance-based models already presented in Figure 6.1 above.

The weighted entropies for the partitions from LloydKL++ and CoresetClustering are already smaller than those for partitions from dominance-based models, but the difference is smaller than the one found when performing the same comparison with the full version of iteration-based methods, indicating that iterations are important to improve the partitioning of the data.

For DivisiveClustering, however, the initial partitions present larger weighted entropies than the partitions obtained from dominance-based methods. This is notable because the initialization method of DivisiveClustering is itself a dominance-based method, in which the elements of each cluster have the same dominant component. In fact, the initialization of DivisiveClustering can be seen, like Ratio-Greedy, as an adaptation of Dominance to prevent empty clusters; the main difference between both adaptations being that, in Ratio-Greedy, elements are grouped not only according to dominant components but also to the ratio between dominant components and

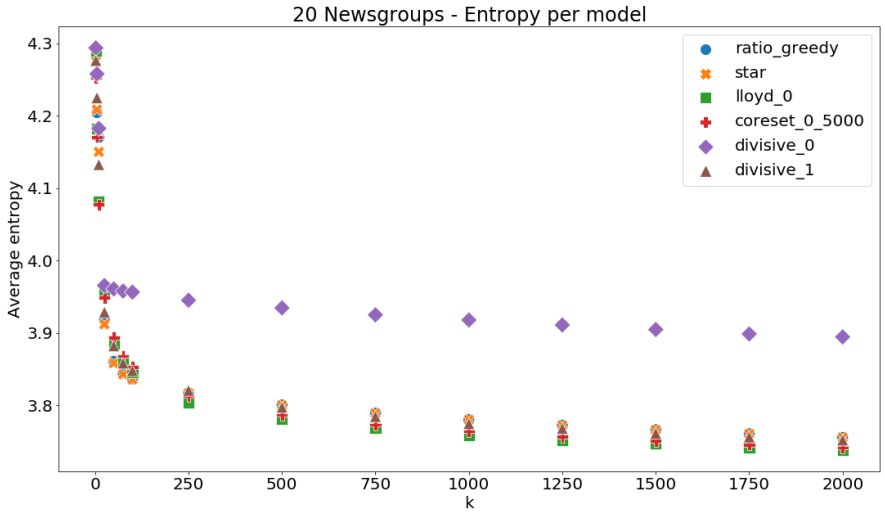



Figure 6.3: Average entropy for the partition of 20 NEWSGROUPS (5 runs per model). Considers the initial partition of iteration-based models, as well as the partition of the DIVISIVECLUSTERING model after a single iteration. The coreset has 5000 elements (approximately 10% of the elements in the original data set).

the elements' $\ell_1$ norm, while in the initialization of DIVISIVECLUSTERING the only concern is that each cluster with the same dominant component will have roughly the same number of elements.

These results suggest that a simpler dominance-based method would not fare as well as the ones presented in Chapter 4. Incorporating information about the ratio between a vector's dominant component and its $\ell_1$ norm achieves significantly better results than relying solely on the dominant components to partition a set.

Because the initialization of DIVISIVECLUSTERING returns partitions with larger entropies, for most values of $k$, than those returned by our dominance-based algorithms, we also present the results of running a single iteration of the expectation-maximization step for this algorithm. As seen in Figure 6.3, this is enough (in this case) for the results of DIVISIVECLUSTERING to be slightly better than those of the dominance-based algorithms.

Figure 6.4 shows the average running times of the models being compared. The difference between the running times of LLOYDKL++ and CORESETCLUSTERING and the dominance-based models is still significant. This, along with the fact that the difference between partitions' weighted entropies is smaller when there is no iterative step in the iteration-based methods, indicates dominance-based methods can be valid alternatives for information

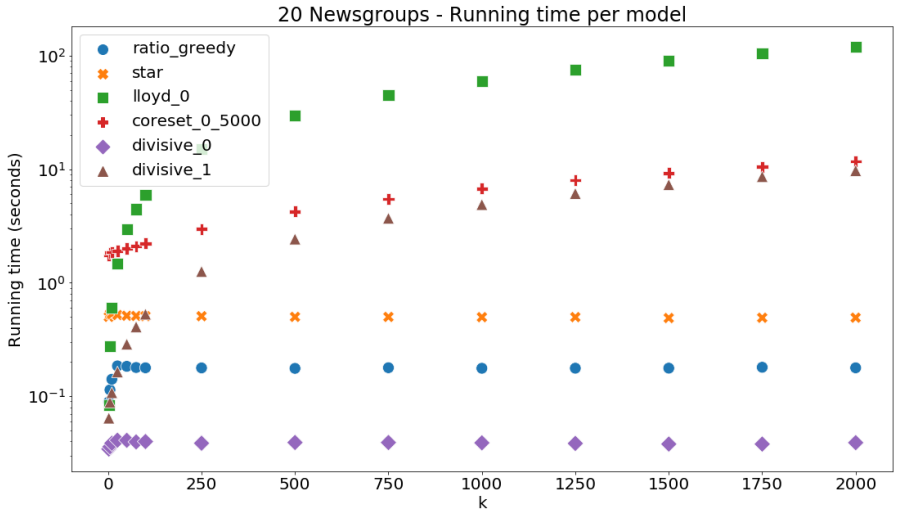



Figure 6.4: Average running time for partitioning 20 Newsgroups (5 runs per model). Considers the initial partition of iteration-based models, as well as the partition of the DivisiveClustering model after a single iteration. The coreset has 5000 elements (approximately 10% of the elements in the original data set).

theoretic clustering when a small running time is of importance.

On the other hand, the initialization step of DivisiveClustering is significantly faster than both dominance-based models under analysis. Coupled with the results for the full DivisiveClustering model, which are comparable to those of LloydKL++, this suggests that using either Ratio-Greedy or Star as the initialization step for an iteration-based model would be unnecessary; a simpler initialization based on dominant components leads to results which are comparable (in the full version of the methods) to those retrieved by using the theoretically sound ++ initialization. However, running a single iteration of DivisiveClustering is enough for this algorithm to become orders of magnitude slower than the dominance-based ones as the number of clusters grows, with running times comparable to CoresetClustering's.

### 6.2.2 RCV1

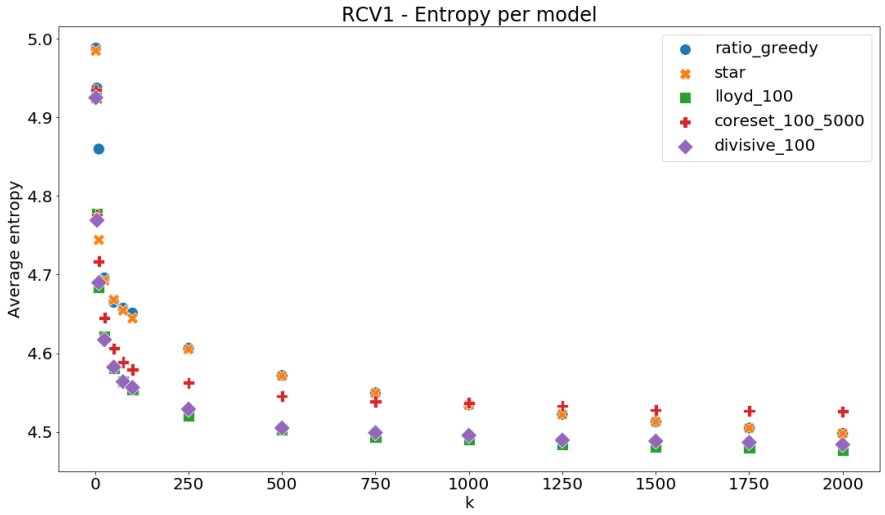



Figure 6.5: Entropy for the partition of RCV1 (1 run per model). Iterative models run for up to 100 iterations. The coreset has 5000 elements (approximately 3% of the elements in the original data set).



#### 6.2.2.1 Comparison between "full" iteration-based methods and dominance-based methods

Figure 6.5 shows the weighted entropy found when partitioning RCV1 using the same five models whose results for 20 Newsgroups were analyzed above, considering the full versions of iteration-based models. Due to time constraints, a single run was performed for each instance of each model.

For this data set, and with a large number of clusters ($k \geq 1000$), the two dominance-based models generate partitions with smaller weighted entropies than those generated by CoresetClustering. One of the reasons may be that, as this data set is larger, a coreset of 5000 elements becomes less representative of it. The margin by which the other two iteration-based models beat the dominance-based models tends to decrease as the number of clusters increases.

Figure 6.6 shows the running time of the same five algorithms for partitioning RCV1. As for 20 Newsgroups, the running times of DivisiveClustering and LloydKL++ are the largest, followed by CoresetClustering and then the dominance-based methods, with a small but clear advantage for Ratio-Greedy over Star. For this particular data set, and as long as the number of clusters is large enough, the dominance-based methods would be a better choice than CoresetClustering, since they perform better in terms of the partition's weighted entropy while taking less time to find the partition.

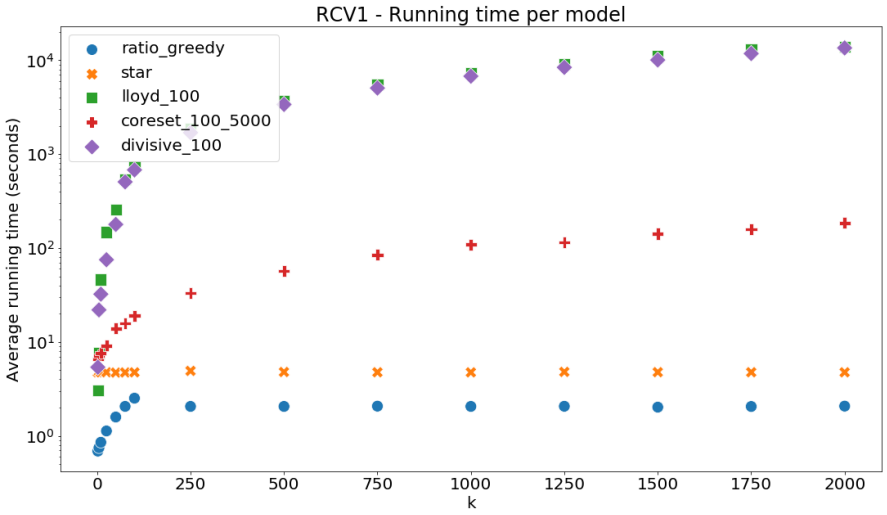



Figure 6.6: Running time for partitioning RCV1 (1 run per model). Iterative models run for up to 100 iterations. The coreset has 5000 elements (approximately 3% of the elements in the original data set).



#### 6.2.2.2 Comparison between "initial" iteration-based methods and dominance-based methods

For the reasons mentioned in Section 6.2.1.2 above, we also present a comparison between partitioning RCV1 with "initial" versions of the iteration-based methods and with dominance-based methods.

Figure 6.7 shows the resulting weighted entropies for each model's partition. As for 20 NEWSGROUPS, the weighted entropies of partitions obtained from the initialization step of DIVISIVECLUSTERING are significantly larger than those of partitions obtained from all other models. When compared to the "initial" versions of the two other iteration-based models, and even to DIVISIVECLUSTERING with a single iteration, dominance-based models present smaller weighted entropies for partitions with a large enough number of clusters. Combined, these two informations seem to further confirm the benefits of using the ratio between dominant components and the $\ell_1$ norm when implementing dominance-based methods.

Figure 6.8 shows the running time for partitioning RCV1 using the two dominance-based methods and the "initial" versions of the three iteration-based methods. Although the running times of the initial versions of iteration-based methods decrease by one or two orders of magnitude when compared to the running times for their full versions, the running time of dominance-based methods is still much smaller. Therefore, when compared to the initial versions

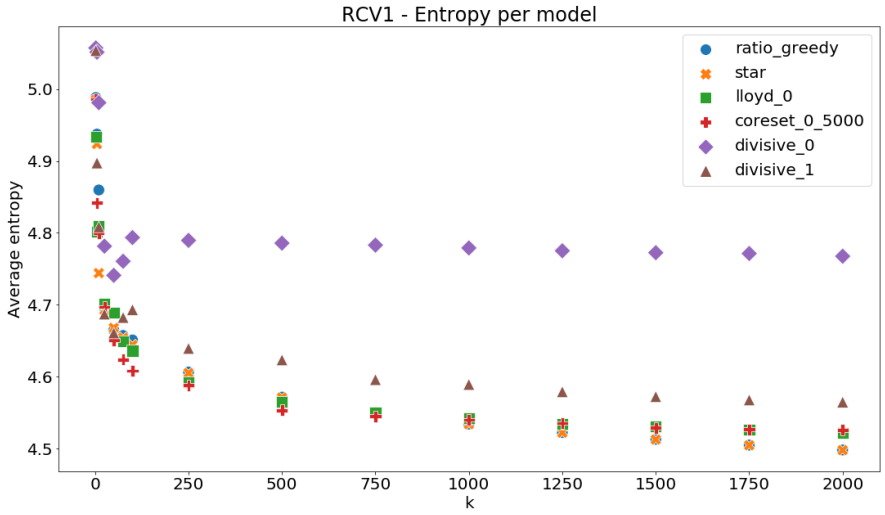



Figure 6.7: Entropy for the partition of RCV1 (1 run per model). Considers the initial partition of iteration-based models, as well as the partition of the DivisiveClustering model after a single iteration. The coreset has 5000 elements (approximately 3% of the elements in the original data set).

of iteration-based methods, dominance-based methods are the best option for this data set, since they are faster and yield partitions with smaller weighted entropies.

The exception, as in the case of 20 Newsgroups, is the running time of the initial version of DivisiveClustering, which is the smallest among all models analyzed. As above, this result is expected due to the fact that the initialization of DivisiveClustering is a simpler dominance-based method of partitioning the data set. Once again we see that, while this simpler method by itself leads to larger weighted entropies than Ratio-Greedy or Star, its use as the initialization step for an iteration-based method is justified, as it leads to results comparable to those of LloydKL++ (while being faster than the more complex dominance-based methods presented here). And again, running a single iteration of DivisiveClustering leads to running times comparables to those of CoresetClustering, and therefore much longer than those of the dominance-based algorithms.

### 6.2.3 Poisson

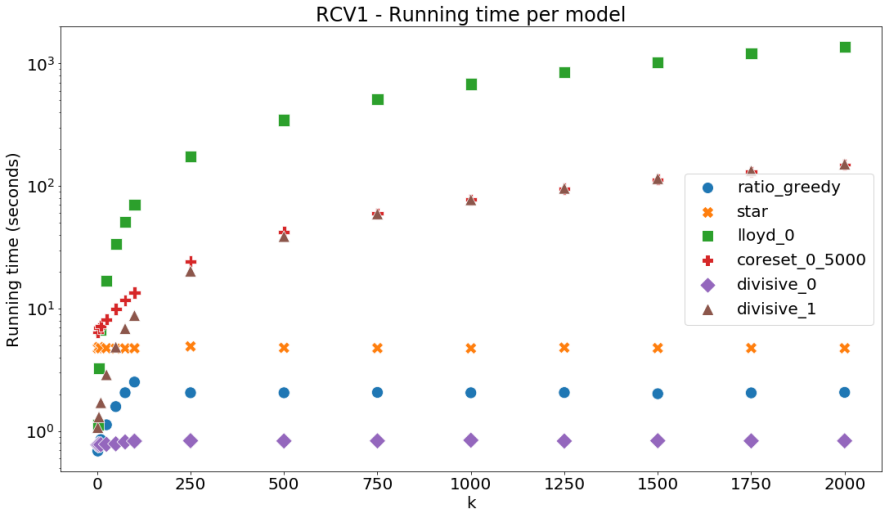



Figure 6.8: Running time for partitioning RCV1 (1 run per model). Considers the initial partition of iteration-based models, , as well as the partition of the DivisiveClustering model after a single iteration. The coreset has 5000 elements (approximately 3% of the elements in the original data set).



#### 6.2.3.1 Comparison between "full" iteration-based methods and dominance-based methods

Figure 6.9 shows the average entropy (over five runs) found when partitioning Poisson using the same five models whose results for 20 Newsgroups and RCV1 were analyzed above, considering the "full" version of iteration-based methods.

For this data set, dominance-based methods yield partitions whose weighted entropies are significantly larger than those of partitions derived from the full versions of iteration-based methods. One additional observation is that, for partitions with a large number of clusters, CoresetClustering tends to produce better partitions (in terms of their weighted entropies) than DivisiveClustering. This data set is smaller than the others, with 10,000 elements, and therefore a coreset with 5,000 members means 50% of the elements in the original set are represented in the coreset, which may explain the good performance of CoresetClustering.

Figure 6.10 shows the average running time of each model. As before, dominance-based methods are orders of magnitude faster than iteration-based methods; among the former, Ratio-Greedy is faster than Star; and, among the latter, CoresetClustering is faster than LloydKL++ and DivisiveClustering.

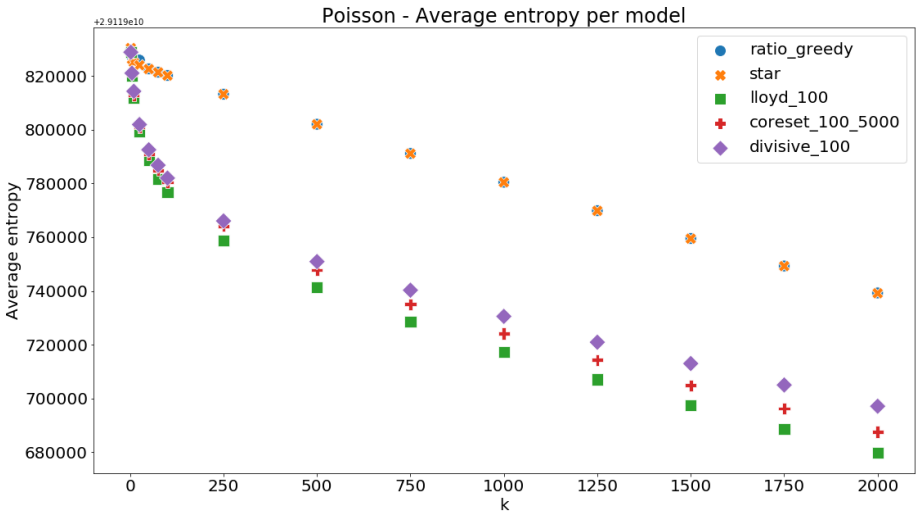



Figure 6.9: Average entropy for the partition of Poisson (5 runs per model). Iterative models run for up to 100 iterations. The coreset has 5000 elements (50% of the elements in the original data set).

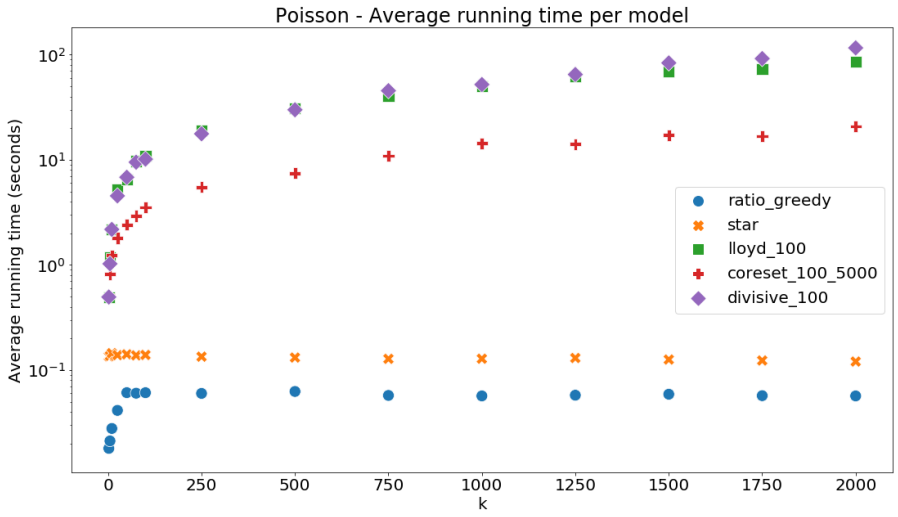



Figure 6.10: Average running time for partitioning Poisson (5 runs per model). Iterative models run for up to 100 iterations. The coreset has 5000 elements (50% of the elements in the original data set).

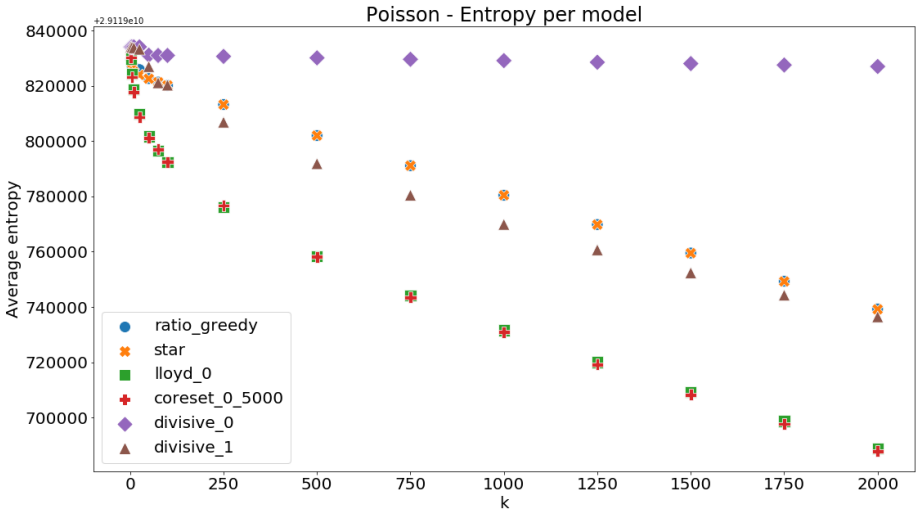



Figure 6.11: Average entropy for the partition of Poisson (5 runs per model). Considers the initial partition of iteration-based models, as well as the partition of the DivisiveClustering model after a single iteration. The coreset has 5000 elements (50% of the elements in the original data set).



#### 6.2.3.2 Comparison between "initial" iteration-based methods and dominance-based methods

As above, we perform below a comparison between the partitions of dominance-based methods and those obtained after the initialization of iteration-based methods. Figure 6.11 shows the weighted entropies of these partitions, according to the number of clusters being used.

Once again, the initialization of DivisiveClustering presents the partitions with higher weighted entropies, followed by the dominance-based models, while the initialization of both CoresetClustering and LloydKL++ yield the partitions with smallest weighted entropies. The same conclusions presented above apply, especially concerning the importance of considering the ratio between dominant components and the $\ell_1$ norm of the vectors when using dominance-based models for clustering. The only change of note regarding DivisiveClustering is that, for this data set, a single iteration is enough for it to return partitions with smaller entropies than those found by the dominance-based algorithms.

Figure 6.12 shows the running time of both dominance-based models, as well as of the initialization steps for the iteration-based models, and for DivisiveClustering with a single iteration. The conclusions are similar to those previously drawn from analyzing results for the other data sets used in

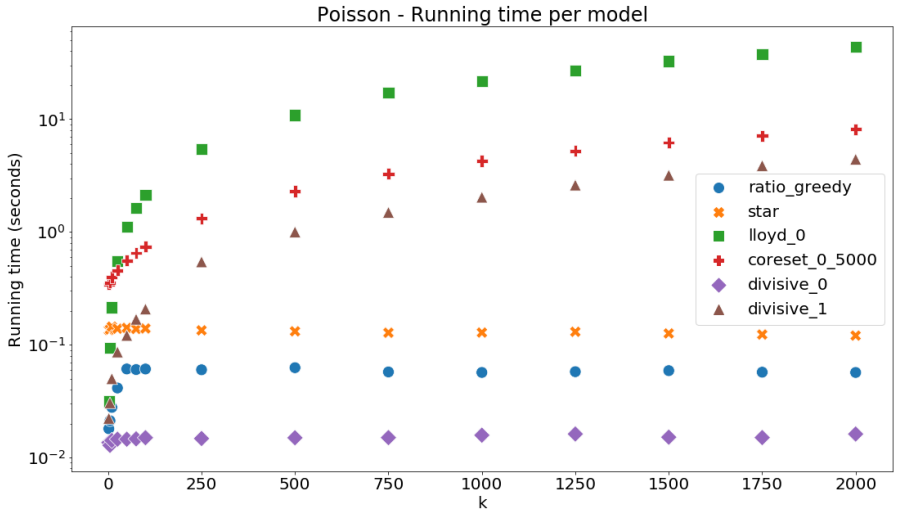



Figure 6.12: Average running time for partitioning Poisson (5 runs per model). Considers the initial partition of iteration-based models, as well as the partition of the DivisiveClustering model after a single iteration. The coreset has 5000 elements (50% of the elements in the original data set).

our experiments.

One issue to keep in mind is that, for this data set in particular, the full version of DivisiveClustering yields partitions with weighted entropies that are significantly higher than those of partitions generated by the full version of LloydKL++. It is possible, therefore, that for this particular data set initializing the cluster centers with a more complex dominance-based model, such as Ratio-Greedy or Star, could be beneficial for an iteration-based model when compared to the simpler dominance-based initialization of DivisiveClustering. However, the results of dominance-based models are relatively worse for Poisson than they are for 20 Newsgroups and RCV1, suggesting any strategy based on dominant components may not be as useful for this particular data set.

# 7
# Conclusions

This thesis has presented both theoretical and experimental results related to information-theoretic clustering. In the theoretical field, we show that the Partition with Minimum Weighted Gini Problem (PMWGP) is closely connected to the geometric $k$-means problem, which leads to a proof that the PMWGP is NP-complete and APX-hard. Furthermore, this connection can be used to derive algorithms for the PMWGP. As an example, we show how one linear-time approximation scheme for the geometric $k$-means problem can be adapted into a PTAS for the PMWGP.

One natural question is whether other algorithms can be adapted to the *PMWGP* via the connection presented here. Since the size of a $k$-means instance built from a *PMWGP* instance can be pseudopolynomial on the size of the *PMWGP* instance, it may be the case that some algorithms do not remain polynomial when used through this connection, or that some known results for the $k$-means problem do not hold for the *PMWGP*. As an example, the algorithm of (Kanungo et al. (2004)), a constant-approximation algorithm for the $k$-means problem, relies on the existence of a set of $\epsilon$-centroids of cardinality $\mathcal{O}(n)$. It is not yet clear whether the pseudopolynomial transformation presented above would allow the *PMWGP* to be (approximately) solved in polynomial time by this algorithm.

In the experimental field, we have presented the results of applying two dominance-based algorithms to the Partition with Minimum Weighted Entropy Problem (PMWEP) for three different data sets, comparing their results with those of iteration-based algorithms. As expected, our algorithms were much faster than the benchmarks, which include state-of-the art Lloyd's algorithm with ++ initialization as well as a version of the same algorithm applied to a coreset of the data. In terms of outcome, our algorithms are able to closely approximate the results of the more involved iteration-based algorithms for two of the three data sets analyzed here, especially as the number of clusters grows.

We have identified three possible applications for information-theoretic clustering: word clustering, node splitting for decision-tree construction, and channel quantization for polar-code construction. Both data sets where our

algorithms have a good performance (for large values of $k$) when compared to the benchmark algorithms are text sets, leading us to cautiously claim that our algorithms may be good candidates for the task of word clustering (where the number of clusters is usually large) when running time is an important concern.

However, there are shortcomings in our analysis that can be addressed by further research. Only two text data sets were used in our experiments, and it would be important to evaluate the results of our algorithms on other data sets from the same domain.[1] Furthermore, it must be remembered that, in most settings, the ultimate goal is not to cluster words, but to properly classify texts; for this reason, additional research may look into whether the word clusters retrieved by dominance-based algorithms yield good results when used by a text classifier.

Generally speaking, and as mentioned at the start of Chapter 6, dominance-based algorithms present several limitations. These can be seen as intentional from a practical point of view, since it is the focus on dominant components and their relative importance that allows such algorithms to be as fast as they are. However, as exemplified by their poor results (compared to iteration-based algorithms) when clustering the third (non-textual) data set analyzed in this thesis, these limitations can also impact their effectiveness.

Therefore, another potential research idea would be to design tests to quickly verify the probability of retrieving a good partition with dominance-based algorithms. It is likely that this would be the case the more representative the dominant components tend to be in vectors from a data set, but further research is needed to clarify this point.

It could be also fruitful to explore whether the good (albeit limited) results presented here when it comes to word clustering can be found in other domains, in particular channel quantization for polar-code construction. On the other hand, the usefulness of dominance-based algorithms as a method for node splitting when building decision trees seems to be limited, since their results for small values of $k$ are significantly worse than those of iteration-based algorithms.

Lastly, there is also the possibility of expanding on the dominance-based

[1] A recent development in textual analysis has been the use of word embeddings to represent words in a vector space, such that similar words are close to each other according to some metric (Mikolov et al. (2013)). Google's word2vec data set (`https://code.google.com/archive/p/word2vec/`), for instance, includes approximately 3 million words, each represented as a vector in $\mathbb{R}^{300}$. The presence of negative values in the embeddings forbids us from applying our algorithms to this data set, but an adaptation that guarantees all embeddings are in $\mathbb{R}^{300}_{+}$ may allow us to further investigate the performance of dominance-based algorithms for word clustering tasks.

algorithms themselves. As mentioned above, both algorithms presented in this thesis were shown to be much faster than their iteration-based counterparts; adding some complexity to them (such as more complex neighborhoods for determining agglomerations of clusters, or the application of randomization techniques) may, at the cost of increasing running time, improve the quality of the partitions they return.

# Bibliography


[Ackermann et al. (2010)] ACKERMANN, M.; BLÖMER, J.; SOHLER, C. **Clustering for metric and nonmetric distances**. In: ACM TRANSACTIONS ON ALGORITHMS (TALG), 2010.

[Arıkan (2009)] ARIKAN, E. **Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels**. IEEE Transactions on Information Theory, 55(7):3051–3073, 2009.

[Arthur & Vassilvitskii (2007)] ARTHUR, D.; VASSILVITSKII, S. **k-means++: The advantages of careful seeding**. In: PROCEEDINGS OF THE EIGHTEENTH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, p. 1027–1035, 2007.

[Arthur et al. (2009)] ARTHUR, D.; MANTHEY, B.; RÖGLIN, H. **$k$-means has polynomial smoothed complexity**. In: 50TH ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, p. 405–414, 2009.

[Awasthi et al. (2015)] AWASTHI, P.; CHARIKAR, M.; KRISHNASWAMY, R.; SINOP, A. K. **The hardness of approximation of euclidean $k$-means**. In: 31ST INTERNATIONAL SYMPOSIUM ON COMPUTATIONAL GEOMETRY (SOCG 2015), 2015.

[Bachem et al. (2018)] BACHEM, O.; LUCIC, M; KRAUSE, A. **Scalable k-means clustering via lightweight coresets**. In: PROCEEDINGS OF THE 24TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY & DATA MINING, p. 1119–1127, 2018.

[Baker & McCallum (1998)] BAKER, L. D.; MCCALLUM, A. K. **Distributional clustering of words for text classification**. In: PROCEEDINGS OF THE 21ST ANNUAL INTERNATIONAL ACM SIGIR CONFERENCE ON RESEARCH AND DEVELOPMENT IN INFORMATION RETRIEVAL, p. 96–103. ACM, 1998.

[Banerjee et al. (2005)] BANERJEE, A. S.; MERUGU, S.; DHILLON, I. S.; GHOSH, J. **Clustering with Bregman divergences**. Journal of Machine Learning Research (JMLR), 6:1705–1749, 2005.

[Bottou (2010)] BOTTOU, L. **Large-scale machine learning with stochastic gradient descent**. In: PROCEEDINGS OF COMPSTAT'2010, p. 177–186. Springer, 2010.

[Bottou et al. (2018)] BOTTOU, L.; CURTIS, F. E.; NOCEDAL, J. **Optimization methods for large-scale machine learning**. Siam Review, 60(2):223–311, 2018.

[Breiman et al. (1984)] BREIMAN, L; FRIEDMAN, J. H.; OLSHEN, R.; STONE, C. J. **Classification and Regression Trees**. 1984.

[Burshtein et al. (1992)] BURSHTEIN, D.; DELLA PIETRA, V.; KANEVSKY, D.; NADAS, A. **Minimum impurity partitions**. The Annals of Statistics, 20(3):1637–1646, 1992.

[Chaudhuri & McGregor (2008)] CHUDHURI, K.; MCGREGOR, A. **Finding metric structure in information theoretic clustering**. In: CONFERENCE ON LEARNING THEORY, 2008.

[Chou (1991)] CHOU, P. A. **Optimal partitioning for classification and regression trees**. IEEE Transactions on Pattern Analysis & Machine Intelligence, 4:340–354, 1991.

[Cicalese et al. (2019)] CICALESE, F.; LABER, E.; MURTINHO, L. **New results on information theoretic clustering**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING – ICML, p. 1242–1251, 2019.

[Coppersmith et al. (1999)] COPPERSMITH, D.; HONG, S. J.; HOSKING, J. R. **Partitioning nominal attributes in decision trees**. Data Mining and Knowledge Discovery, 3(2):197–217, 1999.

[Dai & Le (2015)] DAI, A. M.; LE, Q. V. **Semi-supervised sequence learning**. In: PROCEEDINGS OF THE 28TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS-VOLUME 2, p. 3079–3087. MIT Press, 2015.

[Dasgupta et al. (2007)] DASGUPTA, A.; DRINEAS, P.; HARB, B.; JOSIFOVSKI, V.; MAHONEY, M. W. **Feature selection methods for text classification**. In: PROCEEDINGS OF THE 13TH ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 230–239. ACM, 2007.

[Dhillon et al. (2003)] DHILLON, I.S.; MALLELA, S.; KUMAR, R. **A divisive information-theoretic feature clustering algorithm for text classification**. Journal of machine learning research, 3 (Mar):1265–1287, 2003.

[Duchi et al. (2011)] DUCHI, J.; HAZAN, E.; SINGER, Y. **Adaptive subgradient methods for online learning and stochastic optimization**. Journal of Machine Learning Research, 12(Jul):2121–2159, 2011.

[Fan et al. (2008)] FAN, R.; CHANG, K.; HSIEH, C.; WANG, X.; LIN, C. **LIBLINEAR: A library for large linear classification**. Journal of machine learning research, 9:1871–1874, 2008.

[Fayyaz & Barry (2013)] FAYYAZ, U. U.; BARRY, J. R. **Polar codes for partial response channels**. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS, p. 4337–4341, 2013.

[Genkin et al. (2007)] GENKIN, A.; LEWIS, D. D.; MADIGAN, D. **Large-scale bayesian logistic regression for text categorization**. Technometrics, 49(3):291–304, 2007.

[Goela et al. (2014)] GOELA, N.; ABBE, E.; GASTPAR, M. **Polar codes for broadcast channels**. IEEE Transactions on Information Theory, 61 (2):758–782, 2014.

[Hinton et al. (2012)] HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. **Improving neural networks by preventing co-adaptation of feature detectors**. arXiv preprint arXiv:1207.0580, 2012.

[Joachims (1997)] JOACHIMS, T. **A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization**. In: PROCEEDINGS OF THE 14TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING – ICML, p. 143–151, 1997.

[Kanungo et al. (2004)] KANUNGO, T.; MOUNT, D. M.; NETANYAHU, N. S.; PIATKO, C. D.; SILVERMAN, R. ; WU, A. Y.. **A local search approximation algorithm for k-means clustering**. Computational Geometry, 28(2-3):89–112, 2004.

[Kruskal (1956)] KRUSKAL, J. B. **On the shortest spanning subtree of a graph and the traveling salesman problem**. Proceedings of the American Mathematical society, 7(1):48–50, 1956.

[Kumar et al. (2004)] KUMAR, A.; SABHARWAL, Y.; SEN, S. **A simple linear time (1+ epsilon)-approximation algorithm for $k$-means clustering in any dimensions**. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 2004.

[Kurkoski & Yagi (2014)] KURKOSKI, B. M.; YAGI, H. **Quantization of binary-input discrete memoryless channels**. IEEE Transactions on Information Theory, 60 (8):4544–4552, 2014.

[Laber & Murtinho (2019)] LABER, E.; MURTINHO, L. **Minimization of gini impurity: NP-completeness and approximation algorithm via connections with the k-means problem.** Electronic Notes in Theoretical Computer Science, 346:567–576, 2019.

[Laber et al. (2018)] LABER, A.; MOLINARO, M.; PEREIRA, F. M. **Binary partitions with approximate minimum impurity**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 2018.

[Lewis (2004)] LEWIS, D. D. **RCV1-v2/LYRL2004: The LYRL2004 distribution of the RCV1-v2 text categorization test collection (12-apr-2004 version)**. http://www.jmlr.org/papers/volume5/lewis04a/lyrl2004_rcv1v2_README.htm, 2004 (last access: 10/31/2019).

[Lewis et al. (2004)] LEWIS, D. D.; YANG, Y.; ROSE, T.; LI, F. **RCV1: A new benchmark collection for text categorization research**. Journal of Machine Learning Research, 5:361–397, 2004.

[Lloyd (1982)] LLOYD, S. P. **Least squares quantization in PCM**. IEEE Transactions on Information Theory, 28 (2):129–137, 1982.

[Lucic et al. (2016)] LUCIC, M.; BACHEM, O.; KRAUSE, A.. **Strong coresets for hard and soft Bregman clustering with applications to exponential family mixtures**. In: PROCEEDINGS OF THE 19TH INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND STATISTICS, p. 1–9, volume 51 of Proceedings of Machine Learning Research, 2016.

[Mikolov et al. (2013)] MIKOLOV, T.; SUTSKEVER, I.; CHEN, K.; CORRADO, G. S. ; DEAN, J.. **Distributed representations of words and phrases and their compositionality**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 3111–3119, 2013.

[Miyato et al. (2017)] MIYATO, T.; DAI, A. M.; GOODFELLOW, I. J. **Adversarial training methods for semi-supervised text classification**.

In: 5TH INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, ICLR, 2017.

[Pedregosa et al. (2011)] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. **Scikit-learn: Machine learning in Python**. Journal of Machine Learning Research, 12:2825–2830, 2011.

[Rennie (2014)] RENNIE, J. **20 Newsgroups**. http://qwone.com/∼jason/20Newsgroups/, 2014 (last access: 10/31/2019).

[Ribeiro et al. (2016)] RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. **Why should I trust you?: Explaining the predictions of any classifier**. In: PROCEEDINGS OF THE 22ND ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, p. 1135–1144. ACM, 2016.

[Sandberg et al. (2018)] HUI, D.; SANDBERG, S.; BLANKENSHIP, Y.; ANDERSSON, M.; GROSJEAN, L. **Channel coding in 5g new radio: A tutorial overview and performance comparison with 4g lte**. IEEE Vehicular Technology Magazine, 13 (4):60–69, 2018.

[Shannon (1948)] SHANNON, C. E. **A mathematical theory of communication**. Bell system technical journal, 27(3):379–423, 1948.

[Slonim & Tishby (1999)] SLONIM, N.; TISHBY, N.. **Agglomerative information bottleneck**. In: PROCEEDINGS OF THE 12TH INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, p. 617–623. MIT Press, 1999.

[Tal & Vardy (2013)] TAL, I.; VARDY, A. **How to construct polar codes**. IEEE Transactions on Information Theory, 59(10):6562–6582, 2013.

[Vattanni (2005)] VATTANI, A. **$k$-means requires exponentially many iterations even in the plane**. Discrete & Computational Geometry, 45 (4):596–616, 2005.

[Virtanen et al. (2019)] VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M.; REDDY, T.; COURNAPEAU, D.; BUROVSKI, E.; PETERSON, P.; WECKESSER, W.; BRIGHT, J.; VAN DER WALT, S. J.; BRETT, M.; WILSON, J.; JARROD MILLMAN, K.; MAYOROV, N.; NELSON, A. R. J.; JONES, E.; KERN, R.; LARSON, E.; CAREY, C.; POLAT, İ.;

FENG, Y.; MOORE, E. W.; VAND ERPLAS, J.; LAXALDE, D.; PERKTOLD, J.; CIMRMAN, R.; HENRIKSEN, I.; QUINTERO, E. A.; HARRIS, C. R.; ARCHIBALD, A. M.; RIBEIRO, A. H.; PEDREGOSA, F. ; VAN MULBREGT, P.. **SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python**. arXiv e-prints, 2019.

# A
# Tables

| k | ratio greedy | star | lloyd (full) | divisive (full) | coreset (full) | lloyd (initial) | divisive (initial) | coreset (initial) | divisive (1 iteration) |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4.293432 | 4.275437 | 4.219599 | 4.260723 | 4.225287 | 4.289268 | 4.293432 | 4.251928 | 4.275954 |
| 5 | 4.204171 | 4.208214 | 4.077699 | 4.102027 | 4.092606 | 4.181965 | 4.257562 | 4.170591 | 4.224650 |
| 10 | 4.169997 | 4.150054 | 3.989964 | 3.998027 | 4.009254 | 4.081381 | 4.182388 | 4.077401 | 4.132395 |
| 25 | 3.917957 | 3.911994 | 3.886013 | 3.885630 | 3.904378 | 3.955967 | 3.965535 | 3.948950 | 3.928808 |
| 50 | 3.861007 | 3.858353 | 3.838683 | 3.840378 | 3.852128 | 3.885519 | 3.960809 | 3.893700 | 3.882149 |
| 75 | 3.844733 | 3.843040 | 3.819141 | 3.819205 | 3.828496 | 3.861447 | 3.957870 | 3.868111 | 3.858348 |
| 100 | 3.836762 | 3.835763 | 3.807509 | 3.807591 | 3.816524 | 3.845289 | 3.956388 | 3.853085 | 3.848018 |
| 250 | 3.817300 | 3.817011 | 3.778452 | 3.778251 | 3.788812 | 3.803782 | 3.945220 | 3.813429 | 3.820643 |
| 500 | 3.800762 | 3.800524 | 3.760166 | 3.759344 | 3.771318 | 3.781114 | 3.934656 | 3.787048 | 3.797431 |
| 750 | 3.789368 | 3.789202 | 3.749751 | 3.749329 | 3.761857 | 3.768601 | 3.925004 | 3.773288 | 3.784651 |
| 1000 | 3.780444 | 3.780276 | 3.742896 | 3.742497 | 3.755105 | 3.759012 | 3.917938 | 3.764189 | 3.774740 |
| 1250 | 3.772976 | 3.772751 | 3.737468 | 3.738040 | 3.749855 | 3.752442 | 3.911065 | 3.756809 | 3.768240 |
| 1500 | 3.766577 | 3.766336 | 3.733184 | 3.733193 | 3.745451 | 3.747087 | 3.904769 | 3.751009 | 3.761351 |
| 1750 | 3.760994 | 3.760855 | 3.729401 | 3.728988 | 3.741847 | 3.742387 | 3.898670 | 3.746041 | 3.756389 |
| 2000 | 3.755996 | 3.755863 | 3.726173 | 3.726515 | 3.738544 | 3.738344 | 3.894607 | 3.741716 | 3.752262 |

Table A.1: Average entropy results for 20 Newsgroups

| k | ratio greedy | star | lloyd (full) | divisive (full) | coreset (full) | lloyd (initial) | divisive (initial) | coreset (initial) | divisive (1 iteration) |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.089270 | 0.500183 | 0.583147 | 0.265434 | 1.762526 | 0.083043 | 0.034218 | 1.737987 | 0.064091 |
| 5 | 0.113707 | 0.528462 | 1.144575 | 0.762250 | 1.903266 | 0.275538 | 0.035883 | 1.840957 | 0.088464 |
| 10 | 0.141082 | 0.528866 | 1.854220 | 1.765205 | 1.931072 | 0.599352 | 0.038493 | 1.860257 | 0.107388 |
| 25 | 0.184904 | 0.519034 | 7.723546 | 5.448819 | 2.198037 | 1.484309 | 0.040957 | 1.906503 | 0.163712 |
| 50 | 0.183287 | 0.511797 | 14.413080 | 12.510153 | 2.494366 | 2.974803 | 0.040863 | 2.002048 | 0.287502 |
| 75 | 0.178988 | 0.509649 | 22.377388 | 13.379707 | 2.972268 | 4.440341 | 0.039556 | 2.097930 | 0.408989 |
| 100 | 0.178002 | 0.506903 | 20.700294 | 23.664588 | 3.285300 | 5.921286 | 0.039845 | 2.219427 | 0.528466 |
| 250 | 0.177566 | 0.507161 | 63.523320 | 44.180532 | 5.545954 | 14.978410 | 0.038535 | 2.980419 | 1.255622 |
| 500 | 0.175891 | 0.499894 | 119.569385 | 88.411377 | 9.998801 | 29.855319 | 0.039088 | 4.249595 | 2.425079 |
| 750 | 0.178235 | 0.499234 | 152.173178 | 141.512505 | 13.351993 | 44.848930 | 0.039045 | 5.476371 | 3.695966 |
| 1000 | 0.176228 | 0.497160 | 188.366974 | 176.487439 | 16.515814 | 59.763221 | 0.038815 | 6.740068 | 4.882431 |
| 1250 | 0.176496 | 0.497809 | 231.659353 | 209.507502 | 19.561351 | 74.868846 | 0.038382 | 7.996400 | 6.113360 |
| 1500 | 0.176668 | 0.488512 | 287.966453 | 267.870829 | 22.861389 | 90.081415 | 0.037885 | 9.243964 | 7.309602 |
| 1750 | 0.179830 | 0.490912 | 287.732299 | 263.449267 | 24.074901 | 104.860897 | 0.037844 | 10.498560 | 8.605804 |
| 2000 | 0.177908 | 0.492027 | 361.577766 | 353.064525 | 26.181137 | 119.677433 | 0.038955 | 11.750335 | 9.718906 |

Table A.2: Average running time results (in seconds) for 20 Newsgroups



| k | ratio greedy | star | lloyd (full) | divisive (full) | coreset (full) | lloyd (initial) | divisive (initial) | coreset (initial) | divisive (1 iteration) |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4.988122 | 4.984583 | 4.924933 | 4.924933 | 4.934793 | 4.933324 | 5.057129 | 4.985904 | 5.053361 |
| 5 | 4.937212 | 4.923494 | 4.777805 | 4.768933 | 4.776362 | 4.801898 | 5.051237 | 4.841608 | 4.897173 |
| 10 | 4.859612 | 4.744035 | 4.684260 | 4.689716 | 4.716866 | 4.809369 | 4.980996 | 4.799576 | 4.808251 |
| 25 | 4.695792 | 4.692511 | 4.620766 | 4.617084 | 4.645058 | 4.701911 | 4.781484 | 4.696867 | 4.687030 |
| 50 | 4.664759 | 4.667937 | 4.580996 | 4.582244 | 4.606087 | 4.688920 | 4.740935 | 4.650629 | 4.660805 |
| 75 | 4.657290 | 4.654225 | 4.564263 | 4.563625 | 4.588504 | 4.649285 | 4.760473 | 4.623981 | 4.682322 |
| 100 | 4.651063 | 4.644092 | 4.554052 | 4.556294 | 4.579158 | 4.636144 | 4.793506 | 4.608345 | 4.693167 |
| 250 | 4.606472 | 4.605026 | 4.520945 | 4.528798 | 4.562408 | 4.592505 | 4.789517 | 4.588212 | 4.639259 |
| 500 | 4.571615 | 4.570989 | 4.503039 | 4.504820 | 4.545267 | 4.564554 | 4.785697 | 4.553299 | 4.623322 |
| 750 | 4.549617 | 4.549229 | 4.493682 | 4.498790 | 4.538645 | 4.550672 | 4.782943 | 4.544805 | 4.595886 |
| 1000 | 4.534191 | 4.533896 | 4.490114 | 4.495489 | 4.536682 | 4.542177 | 4.778984 | 4.539831 | 4.589146 |
| 1250 | 4.522348 | 4.522145 | 4.483655 | 4.489465 | 4.533012 | 4.534037 | 4.775023 | 4.535391 | 4.579001 |
| 1500 | 4.512843 | 4.512648 | 4.481085 | 4.488185 | 4.527667 | 4.530565 | 4.772407 | 4.528954 | 4.572260 |
| 1750 | 4.504851 | 4.504622 | 4.480290 | 4.486530 | 4.526521 | 4.526442 | 4.771216 | 4.526816 | 4.567651 |
| 2000 | 4.498036 | 4.497856 | 4.476803 | 4.483781 | 4.525842 | 4.522161 | 4.767508 | 4.525887 | 4.564712 |

Table A.3: Average entropy results for Reuters

| k | ratio greedy | star | lloyd (full) | divisive (full) | coreset (full) | lloyd (initial) | divisive (initial) | coreset (initial) | divisive (1 iteration) |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.687496 | 4.750072 | 3.062775 | 5.374690 | 6.623090 | 1.120811 | 0.772890 | 6.392863 | 1.071491 |
| 5 | 0.747023 | 4.852910 | 7.585375 | 21.927801 | 7.162562 | 3.274089 | 0.775982 | 6.952784 | 1.306023 |
| 10 | 0.853291 | 4.744305 | 46.147143 | 32.276245 | 7.618352 | 6.704050 | 0.776410 | 7.194540 | 1.707210 |
| 25 | 1.126180 | 4.748391 | 147.931616 | 75.014111 | 9.130253 | 16.896152 | 0.780436 | 8.165270 | 2.885525 |
| 50 | 1.584945 | 4.709388 | 256.239682 | 178.433209 | 13.936328 | 33.773199 | 0.787496 | 9.931513 | 4.850950 |
| 75 | 2.057063 | 4.729634 | 535.673108 | 505.931080 | 15.823710 | 50.843226 | 0.816143 | 11.686668 | 6.862048 |
| 100 | 2.514964 | 4.744296 | 747.911316 | 678.946728 | 19.022464 | 70.396615 | 0.828753 | 13.481888 | 8.781648 |
| 250 | 2.056307 | 4.915030 | 1859.953555 | 1695.899428 | 33.099919 | 173.732464 | 0.834977 | 24.292476 | 20.211499 |
| 500 | 2.055205 | 4.782807 | 3714.137374 | 3381.468338 | 56.857849 | 345.947968 | 0.830837 | 42.096571 | 38.740886 |
| 750 | 2.069842 | 4.753813 | 5571.168237 | 5055.220161 | 84.858647 | 511.415335 | 0.832131 | 59.757162 | 59.283889 |
| 1000 | 2.055573 | 4.740188 | 7325.872212 | 6760.536644 | 108.935517 | 679.866750 | 0.842635 | 77.489931 | 77.254765 |
| 1250 | 2.063251 | 4.798266 | 9200.137220 | 8390.788803 | 114.862795 | 848.238285 | 0.829937 | 95.055134 | 95.816122 |
| 1500 | 2.018976 | 4.761795 | 11136.846356 | 10059.742257 | 141.812546 | 1023.996069 | 0.834826 | 112.849625 | 114.992715 |
| 1750 | 2.051474 | 4.757773 | 13064.705580 | 11788.731241 | 158.292310 | 1207.790818 | 0.832128 | 131.059607 | 133.099780 |
| 2000 | 2.071375 | 4.746367 | 13869.995693 | 13516.243217 | 185.187180 | 1366.155830 | 0.832740 | 149.572981 | 151.044055 |

Table A.4: Average running time results (in seconds) for Reuters

| k | ratio greedy | star | lloyd (full) | divisive (full) | coreset (full) |
|---|---|---|---|---|---|
| 2 | 29119829693.401482 | 29119830397.911232 | 29119828845.371777 | 29119828852.263649 | 29119829093.008369 |
| 5 | 29119827288.443542 | 29119826220.500111 | 29119820086.507175 | 29119821068.915714 | 29119820619.066845 |
| 10 | 29119826794.779438 | 29119825066.033375 | 29119811920.029652 | 29119814312.318764 | 29119812883.256794 |
| 25 | 29119825906.549324 | 29119823984.723381 | 29119799200.823807 | 29119801936.536282 | 29119800904.446255 |
| 50 | 29119822648.437248 | 29119822587.012482 | 29119788711.031487 | 29119792556.176655 | 29119790940.394451 |
| 75 | 29119821393.597008 | 29119821347.639568 | 29119781706.828308 | 29119786798.797436 | 29119784905.467094 |
| 100 | 29119820194.168568 | 29119820155.003777 | 29119776888.406258 | 29119782003.303452 | 29119780432.155682 |
| 250 | 29119813250.976227 | 29119813227.192162 | 29119758778.434326 | 29119766045.035027 | 29119764199.685219 |
| 500 | 29119802048.116505 | 29119802023.946526 | 29119741504.356808 | 29119750935.945408 | 29119747812.814293 |
| 750 | 29119791154.169895 | 29119791116.945995 | 29119728563.577705 | 29119740270.605389 | 29119735050.742474 |
| 1000 | 29119780455.956036 | 29119780425.543526 | 29119717274.769855 | 29119730540.325325 | 29119724144.387264 |
| 1250 | 29119769847.897606 | 29119769808.466480 | 29119707071.964062 | 29119720921.451519 | 29119714296.717850 |
| 1500 | 29119759530.007401 | 29119759467.673637 | 29119697484.822086 | 29119712995.427292 | 29119704900.284531 |
| 1750 | 29119749299.591496 | 29119749234.745605 | 29119688568.674271 | 29119705017.458752 | 29119696266.545338 |
| 2000 | 29119739264.851070 | 29119739145.362511 | 29119679698.255993 | 29119697107.035686 | 29119687570.261848 |

Table A.5: Average entropy results for Poisson (dominance-based models and full iteration-based models)

| k | lloyd (initial) | divisive (initial) | coreset (initial) | divisive (1 iteration) |
|---|---|---|---|---|
| 2 | 29119829841.823570 | 29119834118.582336 | 29119830339.725380 | 29119834083.729424 |
| 5 | 29119824460.485817 | 29119834118.582336 | 29119823182.610420 | 29119833981.821205 |
| 10 | 29119818755.771263 | 29119834118.582336 | 29119817706.371761 | 29119833794.261883 |
| 25 | 29119809880.295830 | 29119834118.582336 | 29119808716.826767 | 29119833272.061081 |
| 50 | 29119801746.639599 | 29119831134.924324 | 29119801195.335762 | 29119826999.676563 |
| 75 | 29119796621.898232 | 29119831026.037468 | 29119797102.872581 | 29119821134.404926 |
| 100 | 29119792476.469379 | 29119831026.037468 | 29119792505.151337 | 29119820287.086292 |
| 250 | 29119776163.638206 | 29119830692.222801 | 29119776747.970688 | 29119806875.761356 |
| 500 | 29119758278.573223 | 29119830147.308483 | 29119758135.027264 | 29119791864.316280 |
| 750 | 29119744179.503807 | 29119829631.516975 | 29119743452.907707 | 29119780448.654339 |
| 1000 | 29119731565.728317 | 29119829153.366417 | 29119730926.447243 | 29119769887.850761 |
| 1250 | 29119720023.775257 | 29119828574.153561 | 29119719169.674782 | 29119760677.726326 |
| 1500 | 29119709137.791508 | 29119828076.959927 | 29119708235.133408 | 29119752364.448837 |
| 1750 | 29119698724.562111 | 29119827571.016731 | 29119697669.351452 | 29119744271.348927 |
| 2000 | 29119688818.625908 | 29119827027.752338 | 29119687819.115295 | 29119736441.474251 |

Table A.6: Average entropy results for Poisson (initial iteration-based models and DivisiveClustering with a single iteration)

| k | ratio greedy | star | lloyd (full) | divisive (full) | coreset (full) | lloyd (initial) | divisive (initial) | coreset (initial) | divisive (1 iteration) |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.018000 | 0.135116 | 0.495420 | 0.495039 | 0.511791 | 0.031417 | 0.013560 | 0.344416 | 0.022249 |
| 5 | 0.021190 | 0.136633 | 1.172796 | 1.025198 | 0.820930 | 0.093801 | 0.012818 | 0.355642 | 0.030559 |
| 10 | 0.027839 | 0.144701 | 2.179667 | 2.177951 | 1.230562 | 0.214628 | 0.014171 | 0.397860 | 0.049974 |
| 25 | 0.041363 | 0.138769 | 5.232155 | 4.538169 | 1.801705 | 0.552225 | 0.014547 | 0.456867 | 0.086270 |
| 50 | 0.060990 | 0.141193 | 6.528851 | 6.826835 | 2.407553 | 1.109491 | 0.014525 | 0.558623 | 0.121002 |
| 75 | 0.060260 | 0.137981 | 9.724374 | 9.494791 | 2.925528 | 1.630370 | 0.014630 | 0.651176 | 0.168263 |
| 100 | 0.060939 | 0.139276 | 10.888260 | 10.143591 | 3.522485 | 2.146698 | 0.014989 | 0.740014 | 0.207255 |
| 250 | 0.059992 | 0.134116 | 18.989656 | 17.654201 | 5.494470 | 5.412886 | 0.014750 | 1.321167 | 0.543587 |
| 500 | 0.062596 | 0.131529 | 31.079115 | 29.880638 | 7.456691 | 10.876284 | 0.014964 | 2.296382 | 1.000713 |
| 750 | 0.057396 | 0.127812 | 40.678406 | 45.429981 | 10.937243 | 17.243368 | 0.015012 | 3.268727 | 1.492379 |
| 1000 | 0.056794 | 0.128239 | 50.310647 | 51.976858 | 14.322862 | 21.723420 | 0.015779 | 4.266434 | 2.033195 |
| 1250 | 0.057677 | 0.130346 | 62.213431 | 64.797581 | 14.107402 | 27.048870 | 0.016158 | 5.228098 | 2.605786 |
| 1500 | 0.058932 | 0.125964 | 69.047410 | 83.430343 | 17.136119 | 32.716511 | 0.015149 | 6.203094 | 3.186513 |
| 1750 | 0.057100 | 0.123272 | 73.152869 | 91.824522 | 16.818815 | 37.607582 | 0.015025 | 7.116022 | 3.873707 |
| 2000 | 0.056771 | 0.120528 | 85.461097 | 115.924309 | 20.810354 | 43.749034 | 0.016156 | 8.156780 | 4.412736 |

Table A.7: Average running time results (in seconds) for Poisson