PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
GRADUATE PROGRAM IN COMPUTER SCIENCE

ESTIMATION OF DISTRIBUTION ALGORITHMS FOR CLUSTERING AND CLASSIFICATION

HENRY EMANUEL LEAL CAGNINI

Dissertation presented as partial requirement for obtaining the degree of Master in Computer Science at Pontifical Catholic University of Rio Grande do Sul.

Advisor: Prof. Dr. Rodrigo Coelho Barros

Porto Alegre
2017

Ficha Catalográfica

C131e Cagnini, Henry Emanuel Leal

Estimation of Distribution Algorithms for Clustering and Classification / Henry Emanuel Leal Cagnini . – 2017.
40 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rodrigo Coelho Barros.

1. estimation of distribution algorithm. 2. decision-tree induction. 3. clustering. 4. optimization. I. Barros, Rodrigo Coelho. II. Título.

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Estimation of Distribution Algorithms for Clustering and Classification" apresentada por Henry Emanuel Leal Cagnini como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 20 de março de 2017 pela Comissão Examinadora:

[illegible]

Prof. Dr. Rodrigo Coelho Barros (PPGCC/PUCRS - Orientador)

[illegible]

Prof. Dr. Duncan Dubugras Alcoba Ruiz (PPGCC/PUCRS)

[illegible]

Prof. Dr. Leonardo Ramos Emmendofer (FURG)

Homologada em 06/04/17, conforme Ata No. 05 pela Comissão Coordenadora.

[illegible]

Prof. Dr. Luiz Gustavo Leão Fernandes

PUCRS

Av. Ipiranga, 6681 – Prédio 32 – Sala 507 | CEP 90619-900 | Porto Alegre, RS – Brasil

ACKNOWLEDGMENTS

To my parents, Marlene Leal Cagnini and Luiz Sadi Cagnini, for always giving me support when I most needed it.

To my fiancé, Juliana de Oliveira Mozzaquatro, for helping me overcome my insecurities and pointing out my qualities.

To my friends, and colleagues at PUCRS, for all the moments we shared together.

To my former advisor, Ana Trindade Winck, for making me believe that I was capable of great achievements.

To my current advisor, Rodrigo Coelho Barros, for pushing me to always give my best.

To Motorola Mobility Inc., for funding this research.

ESTIMATION OF DISTRIBUTION ALGORITHMS FOR CLUSTERING AND CLASSIFICATION

RESUMO

Extrair informações relevantes a partir de dados não é uma tarefa fácil. Tais dados podem vir a partir de lotes ou em fluxos contínuos, podem ser completos ou possuir partes faltantes, podem ser duplicados, e também podem ser ruidosos. Ademais, existem diversos algoritmos que realizam tarefas de mineração de dados e, segundo o teorema do "Almoço Grátis", não existe apenas um algoritmo que venha a solucionar satisfatoriamente todos os possíveis problemas. Como um obstáculo final, algoritmos geralmente necessitam que hiper-parâmetros sejam definidos, o que não surpreendentemente demanda um mínimo de conhecimento sobre o domínio da aplicação para que tais parâmetros sejam corretamente definidos. Já que vários algoritmos tradicionais empregam estratégias de busca local gulosas, realizar um ajuste fino sobre estes hiper-parâmetros se torna uma etapa crucial a fim de obter modelos preditivos de qualidade superior. Por outro lado, Algoritmos de Estimativa de Distribuição realizam uma busca global, geralmente mais eficiente que realizar uma buscam exaustiva sobre todas as possíveis soluções para um determinado problema. Valendo-se de uma função de aptidão, algoritmos de estimativa de distribuição irão iterativamente procurar por melhores soluções durante seu processo evolutivo. Baseado nos benefícios que o emprego de algoritmos de estimativa de distribuição podem oferecer para as tarefas de agrupamento e indução de árvores de decisão, duas tarefas de mineração de dados consideradas NP-difícil e NP-difícil/completo respectivamente, este trabalho visa desenvolver novos algoritmos de estimativa de distribuição a fim de obter melhores resultados em relação a métodos tradicionais que empregam estratégias de busca local gulosas, e também sobre outros algoritmos evolutivos.

Palavras Chave: algoritmos de estimativa de distribuição, indução de árvores de decisão, agrupamento, otimização.

ESTIMATION OF DISTRIBUTION ALGORITHMS FOR CLUSTERING AND CLASSIFICATION

ABSTRACT

Extracting meaningful information from data is not an easy task. Data can come in batches or through a continuous stream, and can be incomplete or complete, duplicated, or noisy. Moreover, there are several algorithms to perform data mining tasks, and the no-free lunch theorem states that there is not a single best algorithm for all problems. As a final obstacle, algorithms usually require hyper-parameters to be set in order to operate, which not surprisingly often demand a minimum knowledge of the application domain to be fine-tuned. Since many traditional data mining algorithms employ a greedy local search strategy, fine-tuning is a crucial step towards achieving better predictive models. On the other hand, Estimation of Distribution Algorithms perform a global search, which often is more efficient than performing a wide search through the set of possible parameters. By using a quality function, estimation of distribution algorithms will iteratively seek better solutions throughout its evolutionary process. Based on the benefits that estimation of distribution algorithms may offer to clustering and decision tree-induction, two data mining tasks considered to be NP-hard and NP-hard/complete, respectively, this works aims at developing novel algorithms in order to obtain better results than traditional, greedy algorithms and baseline evolutionary approaches.

Keywords: estimation of distribution algorithm, decision-tree induction, clustering, optimization.

LIST OF ACRONYMS

EDA – Estimation of Distribution Algorithm

GM – Probabilistic Graphical Model

DGM – Directed Probabilistic Graphical Model

UGM – Undirected Probabilistic Graphical Model

GA – Genetic Algorithm

EA – Evolutionary Algorithm

LIST OF SYMBOLS

CONTENTS

1. INTRODUCTION

Data mining is a field which seeks to extract meaningful information from data in a way that such information will help in the process of decision-making in the future [TSK05, c. 1, p 2]. With regard to information retrieval, both of them differ in the way that retrieving the number of goals for each player from the Brazilian National Football Team does not help predicting the probability that Brazil will win the next World Cup. It is also different from machine learning, since their aims are intrinsically different. Machine learning algorithms seeks to "change their behavior in a way that makes them perform better in the future" [WF05, c. 1, p. 8]. More formally, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [Mit97, chap. 1, p. 2].

The existence of machine learning algorithms that seek to analyze, process, learn, and make predictions for data mining tasks is what bring those areas together. With the ever increasing amount of available data [WF05, Mur12, TSK05], it is unlikely that a single algorithm will always perform better for all application domains; hence, new machine learning algorithms for data mining tasks are constantly being developed.

Data mining performs its information extracting process by working with datasets. Datasets may describe a wide range of situations: satellite imagery, characteristics of cancerous cells, data of customers from a bank, and so on. Datasets may be available either as batches or streams of data. Since in this work we only use batches, we will use those terms interchangeably. Batch datasets are roughly $M \times N$ tables. Its N rows are commonly referred as instances (X), records, or objects, and are observations of some or all of its M columns, which in turn are also referred as attributes or features (A). Attributes are subdivided into two categories: predictive attributes ($\mathbf{X}$) and class attribute (Y). Predictive attributes provide some information about an aspect $x_{i,j}$ of the data – for satellite imagery, it can be information such as localization, altitude, etc. A class attribute denotes the category, target, or group for respectively classification, regression, and clustering tasks. The categorical dataset presented in Table 1.1 was presented in [WF05] and describes several aspects of the weather, which may or may not collaborate in the process of deciding whether to play tennis or not. Table 1.2 describes the same dataset in math notation.

Machine learning performs roughly four kinds of learning: supervised, semi-supervised, unsupervised, and by reinforcement, with several algorithms available for each one of those tasks. Popular algorithms perform a greedy local-search through the search space. Take for example K-means [M$^+$67], an unsupervised algorithm for clustering unknown objects into groups. It progressively optimizes the sum of squared errors (SSE) in the hope that such heuristic will lead to groups with high intra-cluster similarity and high inter-cluster dissimilarity [XW05].

Regarding classification, decision trees – one of the most popular methods for both classification and regression, due to its innate comprehensibility feature [BBdC$^+$09] – are often induced through a greedy top-down induction strategy such as Hunt's algorithm [TSK05, c. 4, p.152]. Based on one characteristic of the data, it subdivides it into two or more subsets that present a "purer"

Outcome	Temperature	Humidity	Wind	Play
sunny	hot	high	no	no
sunny	hot	high	yes	no
cloudy	hot	high	no	yes
rainy	mild	high	no	yes
rainy	cold	regular	no	yes
rainy	cold	regular	yes	no
cloudy	cold	regular	yes	yes
sunny	mild	high	no	no
sunny	cold	regular	no	yes
rainy	mild	regular	no	yes
sunny	mild	regular	yes	yes
cloudy	mild	high	yes	yes
cloudy	hot	regular	no	yes
rainy	mild	high	yes	no
Predictive Attributes				Class Attribute

Table 1.1 – An example of a dataset with the class attribute (play) available. Each row is an instance, describing a day, and each column a characteristic of that day. Adapted from [WF05].

	a_1	a_2	a_3	a_4	Y
X_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	y_1
X_2	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	y_2
			$\dots$		
X_n	$x_{n,m-4}$	$x_{n,m-3}$	$x_{n,m-2}$	$x_{n,m-1}$	y_n

Table 1.2 – Math notation for the weather dataset, displayed in Table 1.1. The set of all categories is $\{\text{yes}, \text{no}\} \in C$. Adapted from [WF05].

distribution over the class attribute. It repeats that procedure until a stopping criterion is reached. Generating optimal trees is a NP-hard [BdCF15] problem, whereas inducing a minimal binary tree is NP-complete [TSK05]. Algorithms that employ a greedy strategy tend to be faster than global-search procedures. Albeit comprehensible, considering only a portion of the data at a given split does not capture the whole data distribution. Hence, greedy top-down induction algorithms also employ a pruning procedure in order to avoid overfitting the training data [TSK05, c. 4, p. 172].

Instead of performing a local search, one may employ a meta-heuristic to perform a robust global search through the solution space. In this sense, Estimation of Distribution Algorithms (EDAs) seem to be a natural choice, capable of achieving near-optimal solutions within a reasonable time window [HP11]. In a nutshell, an EDA starts by generating random solutions to the given problem. It then verifies the quality of each solution through a user-defined function. Once it identifies the set of most promising solutions, it then propagates their characteristics to the following generations. It repeats this procedure until either a sufficient number of iterations or a convergence in the quality of solutions is reached.

EDAs belong to the field of evolutionary computation, and as such have some overlapping with data mining tasks. Several evolutionary algorithms have already been proposed for dealing with both clustering [XW05, HCFDC09], or with decision-tree induction[BBDCF12]. However very few EDAs have been proposed for tackling such tasks, even though EDAs surpasses other evolutionary algorithms in both performance and comprehensibility of the evolutionary process [HP11].

With that in mind, this work aims at developing estimation of distribution algorithms for both decision tree induction and data clustering in order to achieve better results than traditional greedy strategies and also other baseline evolutionary algorithms.

1.1 Contributions

The contributions of this work are the development of EDAs for the aforementioned tasks: Clus-EDA (Section 3.3) and PASCAL (Section 3.4) are effective clustering algorithms based on medoid and density paradigms; Ardennes (Section 4.4) is a decision-tree induction algorithm which surpasses its evolutionary baseline and achieves competitive results regarding its traditional greedy baseline.

1.2 Outline

The rest of this work is organized as follows. Chapter 2 presents in further detail the aspects of Estimation of Distribution Algorithms. Chapter 3 describes clustering, the data mining task of grouping objects into similar groups, whereas Chapter 4 explains classification and reviews why decision-trees are popular methods for performing this task. Finally, the final remarks are in Chapter 5.

2. ESTIMATION OF DISTRIBUTION ALGORITHMS

According to Hauschild and Pelikan, "Estimation of Distribution Algorithms (EDAs) are stochastic optimization techniques that explore the space of potential solutions by building and sampling explicit probabilistic models of promising candidate solutions" [HP11]. EDAs are explore the complete set of solutions available to a given problem, and are employed in searches where performing a stressful verification through all possible solutions is infeasible due to time constraints. EDAs rely on solution sampling, probabilistic graphical model updating and quality evaluation in order to achieve satisfactory results within a predefined time window [HP11].

In EDAs, solutions are also called individuals (s), and a set of individuals comprises a population (S). The quality of an individual is called fitness (ψ) and is a problem-specific metric: it can only be asserted by testing each individual in the proposed problem. Fitness may present any range of values, but are often normalized for ranges $[0,1]$, $[0,+\infty]$ or $[-1,1]$, with higher values as better ones. EDAs perform their search through a predefined amount of time, such as number of iterations or generations (G). From a given generation to another, a set of most promising individuals (Φ) is selected to update the probabilistic graphical model's variables ($\mathbf{V}$).

As a meta-heuristic, user-defined parameters in EDAs are more likely to be related to the search procedure rather than directly to the problem being solved. EDAs do not present a canonical way of updating their probabilistic graphical models, replacing and storing individuals across generations nor how to evaluate individuals. However, the pseudo-code of Figure 2.1 succinctly summarizes the overall process of an EDA.

```
1: function EDA(population_size, problem)
2:     model ← initializes probabilistic graphical model (GM) with a uniform distribution
3:     while stopping criteria not met do
4:         population ← sample individuals from GM
5:         fittest ← individuals in the population which better solve the problem
6:         model ← update model from the fittest individuals in a way that is more likely to generate
   them in following generations
7:     return the best individual found so far
```

Figure 2.1 – Generic pseudo-code for EDAs.

In order to better explain how an EDA work, let's assume a hypothetical optimization problem. It consists in assigning colors to several nodes. Those nodes reside within a graph structure, which connects nodes through a set of edges. Two connected nodes are considered to be neighbors. The quality of an individual is given by the number of neighboring edges that *do not* present the same color. Figure 2.2 shows two possible color sets.

An hypothetical EDA assigned to solve this problem could start by sampling individuals from a uniform distribution – that is, all color sets are equally likely to occur in the population – encapsulated in a Probabilistic Graphical Model (GM). This is a particular optimization problem, where GMs' variables are coincident with the individual's structure – that is, each node in the graph is also a variable

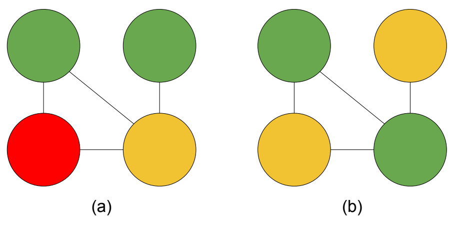


Figure 2.2 – Two possible color sets (individuals) for the graph: (a) is a high-quality configuration, since no node has a neighbor with the same color; (b) is a low-quality configuration. Best viewed in colors.

in the GM. Each variable comprises a Probabilistic Mass Function, since a node can only be instantiated with discrete colors, and not a set of them.

Once the EDA samples $|S|$ individuals from the GM, it is necessary to measure their quality. In this case, its simply the number of nodes connected with different colors, normalized by the number of total edges. Once this is done, it is interesting to propagate the characteristics of the fittest individuals to the following generations. This is done by updating the variables' probabilities. In Figure 2.3, the univariate inference of probabilities is showed based on two individuals.

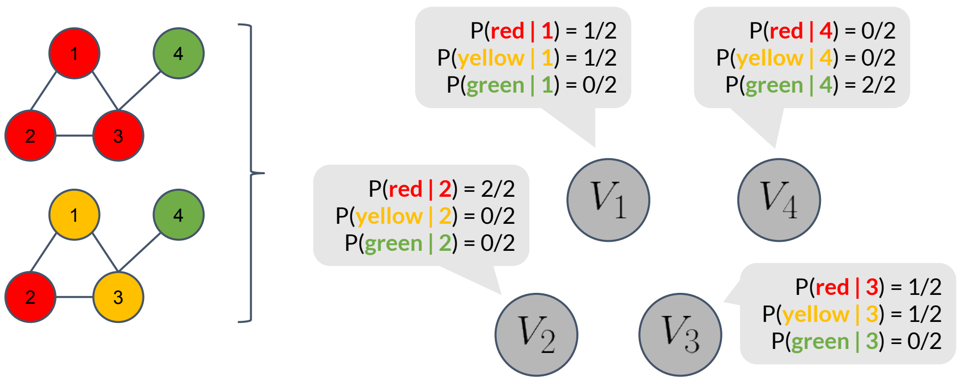


Figure 2.3 – Based on two individuals (left colored graphs), the EDA's GM updating step will update its variables' probabilities in a way that is more likely to generate those individuals. Best viewed in colors.

The EDA will repeat this process of sampling, fitness evaluation and GM updating until a stopping criteria is reached: often a sufficient amount of time has passed (which may be measured by the number of iterations) or the lack of diversity in the population (all individuals have the same fitness, or the probabilities in the GM are locked into either 0 or 1 for all possible selections). Since probabilistic graphical models play an important role in the EDAs process, the following sections further explains their concepts, variants and procedures.

2.1 Probabilistic Graphical Models

Imagine an environment composed of several variables; it may be, as pointed by Murphy [Mur12, c. 10, p. 310], pixels in an image, words in a document or genes in a microarray. One way to represent the interactions of all variables is to use the chain rule, which states that one variable V_1 is independent, the subsequent variable V_2 is dependent on V_1, V_3 on V_1 and V_2, and so on until the $|\mathbf{V}|$-th variable, where $|\mathbf{V}|$ the number of variables in the system. It may also be expressed as

$$P(V_{|\mathbf{V}|-1}) = P(V_1)P(V_2|V_1)P(V_3|V_1,V_2)...P(V_{|\mathbf{V}|-1}|V_1,V_2,...,V_{|\mathbf{V}|-2}) \tag{2.1}$$

However, the computational cost of representing this chain grows with the number of variables, eventually requiring an unfeasible amount of processing power to compute a single variable probability. A Probabilistic Graphical Model (or GM) is a way to represent interaction between variables of a given environment without the need of writing the full chain of relationships. In a GM, variables are represented by nodes and edges connect two interacting variables.

GMs may be either directed or undirected. A directed probabilistic graphical model (DGM) is sometimes referred as a Bayesian network, belief network or causal network [Mur12, c. 10, p. 310]. In DGMs, a variable V_i may have one or more parents – a parent is a variable V_j which influences the outcome value of V_i, but is not influenced by V_i – and one or more children – when V_i is the parent of another variable V_k. Another characteristic of a DGM is that it must not have cycles in its layout, thus requiring independent variables. Figure 2.4 represents a possible DGM layout.

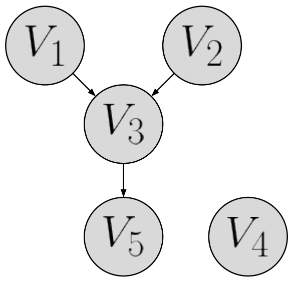


Figure 2.4 – Layout of a DGM. V_1 and V_2 are parents of V_3, which in turn is parent of V_5. V_4 does not interact with any other variable, thus being independent.

An undirected probabilistic graphical model (UGM), on the other hand, is a GM where a variable V_i may influence and be influenced by any other node connected to it. This GM project is required since not all domains can be represented by a DGM. Murphy [Mur12, p. 663, chap. 19] gives the example of an image: it makes no sense to think that a pixel may influence its neighboring pixels but is not influenced by them.

Using a DGM or UGM depends directly on the domain's data distribution. Some distributions may be represented perfectly by both DGMs and UGMs, and are called **chordals**. In Figure 2.5 an UGM and two failed attempts to represent it as a DGM are presented; Figure 2.6 presents a Venn diagram of the representational possibilities of distributions. Since it is most common to find EDAs based on DGMs than UGMs [HP11], the following section describes possible Directed Architectures proposed in literature for EDAs.

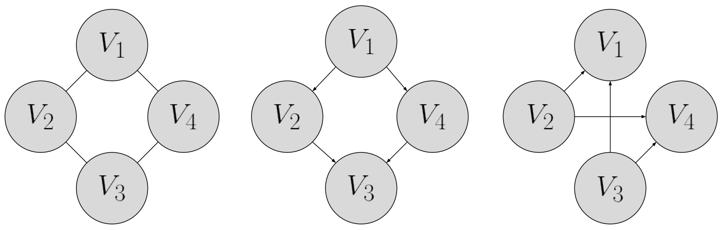


Figure 2.5 – An UGM (left) and two failed attempts to model it as a DGM, since they do not capture all conditional dependencies. Adapted from [Mur12, c. 19, p. 667].

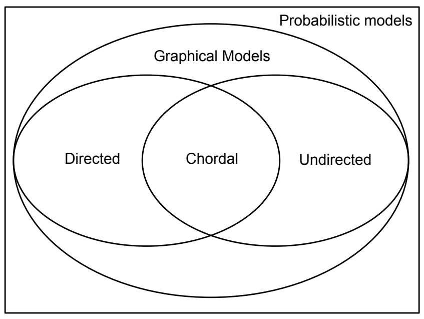


Figure 2.6 – All representational possibilities over distributions. Adapted from [Mur12, c. 19, p. 666].

2.1.1 Directed Architectures

Independently from the GM used, EDAs have to model the relationship of variables. EDAs do this by correlating variables with one another. Since GMs represent variables as nodes in their structure, from now on the terms node and variable will be used interchangeably to refer to the same concept. Also, since this work does not develop EDAs based on UGMs, its architectures will not be presented, and whenever referring to graphical models in other chapters we will refer to DGMs. The next sections present some DGM architectures previously used in conceptual EDAs. For the interested reader, more DGM architectures can be found in [HP11].

Univariate Models

Univariate models [HP11] assume that the generation of values for a given variable V_i is independent from the generating process of any other variable V_j, $j \neq i$ and $i, j \in \{1, \ldots, |\mathbf{V}|\}$. A representative example of an EDAs using this approach is the Univariate Marginal Distribution Algorithm, or UMDA [MP96]. Figure 2.7 presents its GM layout.

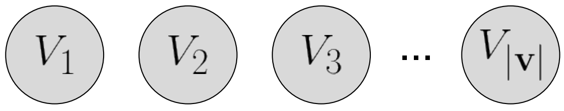


Figure 2.7 – The architecture of a univariate distribution.

The updating of probabilities associated with each variable is done as follows. The UMDA selects the best-quality individuals from a given generation. Since it does not assume dependence between variables, its updating process can be done in parallel: for each variable V_i it verifies the sampled value for each individual in the fittest population, groups them by value and uses this relative frequency as the new probabilities of sampling those values. The previously showed Figure 2.3 depicts this procedure.

Due to its parallel nature, UMDAs tend to perform faster than EDAs that assume dependence between variables or even infer their relationship. However, univariate EDAs may fall in local optima since they do not capture the underlying interactions between variables. Since modelling the full relationship between variables is prohibitive due to processing costs, other strategies must be explored. One of such strategies is the one of rearranging GMs in a way that to resemble the structure of trees.

Tree-based Models

Tree-based models are constructed so the DGM resembles a tree, containing root, leaf nodes and inner nodes. A particularity of those models is that each variable may have at most one parent. The number of children, however, may vary depending on the algorithm.

The first algorithm presented here is MIMIC [DBIJV96]. It starts with a univariate GM. Once it samples its initial population, it works by finding the variable V_i with lowest entropy in the Φ subpopulation to be the tree root:

$$H(V_i) = -\sum_{v_{i,k} \in V_i} P(v_{i,k}) \log P(v_{i,k}) \tag{2.2}$$

where $v_{i,k}$ is a possible value for V_i. Once the lowest entropy variable V_i is found, it then calculates the mutual information between V_i and V_j, $j \neq i$ and $i, j \in \{1, \dots, |\mathbf{V}|\}$:

$$I(V_i, V_j) = \sum_{v_{i,k} \in V_i} \sum_{v_{j,l} \in V_j} P(v_{i,k}, v_{j,l}) \frac{P(v_{i,k}, v_{j,l})}{P(v_{i,k})P(v_{j,l})} \tag{2.3}$$

where $P(v_{i,k}, v_{j,l})$ is the joint probability of variables V_i and V_j defined as $P(V_i = v_{i,k}, V_j = v_{j,l}) = P(V_i = v_{i,k} | V_j = v_{j,l}) P(V_j = v_{j,l})$. It chooses the variable V_j with highest mutual information with

V_i to be the next variable in the chain. It then repeats this process with V_j and so on, until no variable is left outside. The resulting probability to generate a set of variables is

$$\hat{P}_\pi = P(V_{k_1}|V_{k_2})P(V_{k_2}|V_{k_3})...P(V_{k_{|\mathbf{V}|-1}}|V_{k_{|\mathbf{V}|}}) \tag{2.4}$$

where $k_1, k_2, \dots k_M$ are the indexes of variables, and π the order of such indexes. Once the tree is built, the algorithm then samples individuals from this tree and repeat the aforementioned GM building process until a stopping criteria is met. Figure 2.8 presents the DGM layout of MIMIC.

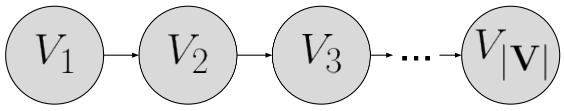


Figure 2.8 – DGM layout of MIMIC.

Another tree-based EDA is the one proposed by Baluja and Davies in [BD97]. In their work the authors do not name their method, but for clarity purposes we will dub it as TreeMIMIC. TreeMIMIC performs an inference process similar to MIMIC in the way it correlates variables: using entropy to find the root and mutual information to establish links between them. It also forces each variable to have at most one parent (except for the root), but contrarily to MIMIC does not restrict the number of children. Figure 2.9 exemplifies a hypothetical DGM generated by TreeMIMIC.

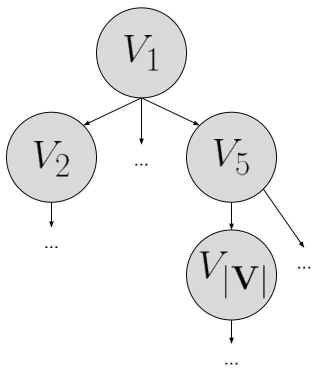


Figure 2.9 – Hypothetical DGM generated by TreeMIMIC.

Finally, the last tree-based EDA covered in this section is BEDA – Bivariate Marginal Distribution Algorithm, proposed by Pelikan and Mühlenbein [PM99]. BEDA analyzes bivariate and univariate distributions to determine whether two variables are correlated or not. This is done by using Pearson's chi-square statistics, which can be defined as

$$\chi^2 = \sum \frac{(observed - expected)^2}{expected} \tag{2.5}$$

This can be interpreted as follows: given Υ observations of random variables $V_1, V_2, ..., V_{|\mathbf{V}|}$, we want to find out whether two arbitrary variables V_i and V_j are correlated or not. When a set of instantiations of variables occurs, the probability of the combination is

$$P(V_i = v_{i,k}, V_j = v_{j,l}) = P(V_i = v_{i,k} | V_j = v_{j,l}) P(V_j = v_{j,l}) \tag{2.6}$$

The probability that V_i will assume values independently from V_j can be written as

$$P(V_i = v_{i,k}) P(V_j = v_{j,l}) \tag{2.7}$$

if we interpret the *observed* value as the *conditional probability* of V_i given V_j and the *expected* value as the unconditional probability of V_i and V_j, we can merge equations 2.5, 2.6 and 2.7, obtaining

$$\chi^2_{i,j} = \sum_{v_{i,k} \in V_i} \sum_{v_{j,l} \in V_j} \frac{(\Upsilon \cdot P(v_{i,k}, v_{j,l}) - \Upsilon \cdot P(v_{i,k}) P(v_{j,l}))^2}{\Upsilon \cdot P(v_{i,k}) P(v_{j,l})} \tag{2.8}$$

$\chi^2_{i,j}$ will result in a value in the chi-squared distribution. If $\chi^2_{i,j} < 3.84$, then the two variables are independent for 95% of the observed occurrences and will not be linked in the GM.

It is important to note that, because of the nature of the DGM building strategy used by BEDA, the resulting DGM layout may be composed by several trees which are not necessarily correlated. BEDA also restricts variables to have at most one parent. Figure 2.10 shows a possible layout of the DGM generated by BEDA.

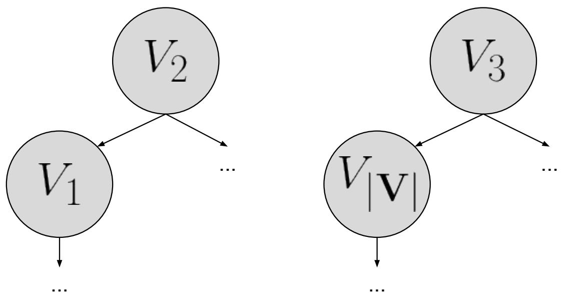


Figure 2.10 – A possible DGM layout generated by BEDA.

3. CLUSTERING

Imagine that you work for a big, colorful tech company that provides a web search engine. As the years pass by and more Internet users perform searches using your service, you know more what the users are searching: "cake recipe", "how to surf", etc. Then, one day you are asked to draw several profiles of users that used the tool: How do you start?

Clustering is a field where data mining and machine learning crosses its paths: to group data is to cluster, and it can be done by unsupervised machine learning algorithms. Technically, datasets used in clustering have only predictive attributes, and is up to the data scientist to define a group for each instance, since no prior information about the number or type of groups is known [KR90]. Examples of real-world applications that benefit from data clustering include image segmentation [CTC$^+$06] and bioinformatics [HCdC06, BCFdC13].

It is important to stress that clustering does not mainly attempts to assign a *class* to unknown data, but rather a group; it is up to the data scientist to whether treat it as a class or not. Take for example the problem of grouping tumor cells into similar groups [SBL11]. Tumor cells may present similar characteristics and thus belong to the same group; however similar characteristics do not guarantee that a treatment is appropriate to same-group tumors.

The set of group assignments for each instance is called partition. The objective of clustering is to obtain groups with high intra-group similarity and low inter-group similarity. There is no consensus in the way of measuring this concept, as viewed by the several quality criteria currently available [VCH10]. More formally, clustering is a two-step task, where it is first necessary to establish the desired number of groups, and then assign instances to each of of those groups. However, the number of groups K[1] ranges between 1 (the trivial partition, where every object belong to the same group) to N (one object per group, i.e a singleton). Once it is selected, the number of possible partitions is $\frac{1}{K!}\sum_{i=0}^{K}(-1)^i\binom{K}{i}(K-i)^N$ [TSK05].

From an optimisation viewpoint, clustering is a particular kind of NP-hard problem. Traditional clustering algorithms perform a greedy, local-search through the space of possible partitions. However, many works also approach it by using evolutionary strategies [XW05, HCFDC09], general-purpose meta-heuristics believed to be effective in tackling NP-hard problems, although few works employ EDAs for this task.

The rest of this chapter is organized as follows. Section 3.1 presents the criteria used to evaluate quality of partitions, whereas Section 3.2 reviews some popular clustering algorithms. Section 3.3 presents Clus-EDA, our first EDA for tackling clustering using medoids, and Section 3.4 presents PASCAL, the EDA for performing density-based clustering.

[1]Due to its popularity, this chapter will use K to define the number of groups that objects within a dataset may be assigned. However, other chapters may use K as a generic indexer.

3.1 Clustering analysis

According to Tan, Steinbach and Kumar [TSK05, chap. 8, p. 487], cluster analysis is the task of dividing data into clusters that are meaningful, useful or both. Cluster analysis may be used for two purposes: utility and understandability.

When clusters are required to be understandable, they must try their best to represent a possible class. This translates to a high similarity between objects, in a way that each one of them present a characteristic common to all. Some tasks which require understandability are biology taxonomy, consumer's profile clustering and illness grouping.

On the other hand, utility requires that a cluster provides a representative prototype. Some of those tasks are data summarization, compression and nearest neighbor localization. Prototypes come in two forms: centroids and medoids. A centroid is a hypothetical object which better represents the instances within a group. It may or may not be placed above an actual instance, in which case it becomes a medoid. Some partitional algorithms restrict cluster prototypes to use only medoids, since they diminish the impact of outliers in the dataset. Figure 3.1 demonstrates how a cluster with several outliers can be prejudiced from using a centroid-based clustering algorithm.

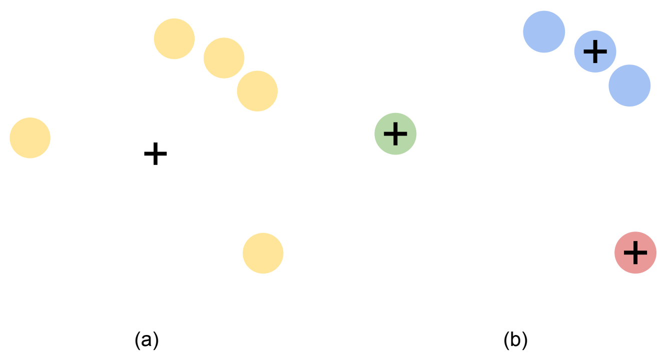


Figure 3.1 – (a) A set of 5 objects (circles) belonging to the same group, due to the use of centroids (cross) as group prototypes. (b) The same 5 objects from (a), but grouped in 3 different groups, due to the use of medoids (cross inside circles) as cluster prototypes. Best viewed in colors.

The general idea of clustering is to produce partitions which maximize the intra-cluster similarity and reduce the inter-cluster similarity. However, this notion may be difficult, since a cluster may be partitioned into smaller clusters while still keeping the previous metrics. Figure 3.2 is based on the figure presented in [TSK05, chap. 8, p. 491] and exemplifies this concept.

Clusters may be evaluated using different validity criteria, each one measuring a desirable aspect of clusters. Validation of clustering structures is said to be most difficult and frustrating part of cluster analysis, to the point in which it is compared to a "black art" [JD88]. We refer the interested reader to a thorough survey on clustering validity criteria by Vendramin et al. [VCH10]. Most of these criteria, however, are computationally costly. The next sections present some of those measures that will be used in this work.

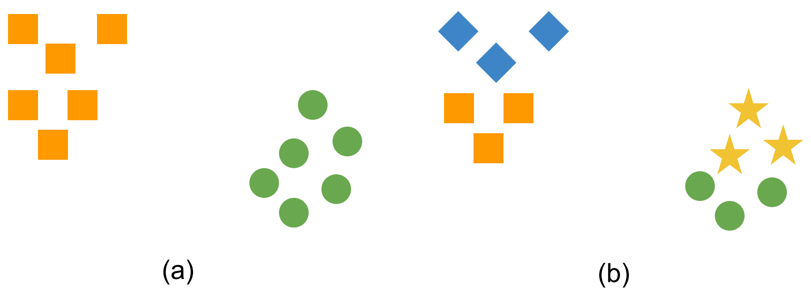


Figure 3.2 – A three-dimensional dataset with 12 instances, described with both (a) 2 clusters and (b) 4 clusters. Best viewed in colors.

3.1.1 Silhouette Width Criterion

According to [VCH10], the Silhouette Width Criterion (SWC) [Rou87] is a measure based on separate and compact clusters, defined as

$$SWC = \frac{1}{N} \sum_{i=1}^{N} \frac{b(i) - a(i)}{max(b(i), a(i))} \tag{3.1}$$

where $a(i)$ is the mean distance between an object i and all other objects belonging to the same cluster, and $b(i)$ the mean distance between the object i and all objects of the closest cluster but its own. For singletons, the ratio is not computed, being replaced by zero.

SWC values range from $[-1, 1]$, where -1 describes overlapping clusters with sparse objects, and 1 are well separated and compact clusters. Although SWC provides a value which can be compared between several partitions, it is not guaranteed that the clustering with the best SWC provides a reasonable understandability or utility. Since one needs, for each object i, the distance between it and all other objects in the dataset, the computational cost of SWC is $O(M \cdot N^2)$.

3.1.2 Simplified Silhouette Width Criterion

In order to reduce the computational cost of SWC, the Simplified Silhouette Width Criterion (SSWC) [VCH10] makes two replacements: $a(i)$ becomes the distance between an object and the prototype of the cluster it belongs, and $b(i)$ the distance to the prototype of the closest cluster but its own. As inferred in the work of Vendramin et al. [VCH10], the overall cost of SSWC is $O(M \cdot N \cdot K)$. Note that $SSWC$ can become costly for $K \approx N$, though in practice $K << N$.

3.1.3 Dunn

Dunn Index [Dun74], similarly to SWC and SSWC, is a measure based on cluster's separation and compactness. It is defined as

$$DN = \min_{\substack{i,j \in \{1,\dots,K\} \\ i \neq j}} \left\{ \frac{\delta_{i,j}}{\max\limits_{l \in \{1,\dots,K\}} \Delta_l} \right\} \tag{3.2}$$

where Δ_l is the diameter of the lth cluster or, in other words, the maximum distance between two objects that belong to the same cluster, and $\delta_{i,j}$ is the distance between clusters i and j.

Since $\delta_{i,j}$ is the mean distance between two clusters, it is required the computation of the distance between all objects of the dataset, thus being of same complexity of SWC, $O(M \cdot N^2)$.

3.1.4 Density-Based Clustering Validation

As pointed out by Moulavi et al. [MJC+14], previously developed density-based criterion fail in several aspects such as correctly analysing arbitrarily-shaped clusters due to the use of the euclidean distance – which, as previously said, favors the generation of spherical groups – or requiring a parameter such as the number of nearest neighbors to calculate the density of a neighborhood. Density-Based Clustering Validation (DBCV) [MJC+14], on the other hand, is a parameterless criterion which defines a new distance to calculate such density.

The calculation of the DBCV starts by taking as input a partition. It then computes $\Gamma(X_i)$, which is the inverse of density of the object X_i in its cluster:

$$\Gamma(X_i) = \left(\frac{\sum_{j=2}^{n_i} \left(\frac{1}{d(X_i, X_j)} \right)^M}{|\mathbf{X}_\pi| - 1} \right)^{-\frac{1}{M}} \tag{3.3}$$

where M is the dimensionality of the data, $|\mathbf{X}_\pi|$ the number of objects in the π-th cluster and $d(X_i, X_j)$ the distance between objects X_i and the j-th closest neighbor of X_i in $\mathbf{X}_\pi$. Note that j starts at 2 (i.e, the first closest neighbor), since it makes no sense in calculating the inverse density between an object and itself. PASCAL uses the squared euclidean distance for d, i.e $d^2(X_i, X_j)$.

Once all objects have their Γ computed, it builds a mutual reachability matrix of the $\mathbf{X}_\pi$ cluster objects using the definition of mutual reachability distance:

$$d_{mreach}(X_i, X_j) = \max(\Gamma(X_i), \Gamma(X_j), d(X_i, X_j)) \tag{3.4}$$

From the $mreach$ matrix of each cluster, a MST_{MRD} is built, which captures the underlying structure of the data. Using each MST_{MRD} it is possible to calculate the Density Sparseness of a Cluster, $DSC(\mathbf{X}_\pi)$, which is the edge with maximum weight in MST_{MRD}, and Density Separation of a Pair of Clusters, $DSPC(\mathbf{X}_\pi, \mathbf{X}_\varepsilon)$, which is the minimum mutual reachability distance that separates the inner nodes (e.g nodes with degree 2 or above) of two MST_{MRD}.

The validity of a cluster is calculated using both DSC and $DSPC$ values:

$$VC(\mathbf{X}_\pi) = \frac{\min_{1\leq\varepsilon\leq K, \varepsilon\neq\pi}(DSPC(\mathbf{X}_\pi, \mathbf{X}_\varepsilon)) - DSC(\mathbf{X}_\pi)}{\max\big(\min_{1\leq\varepsilon\leq K, \varepsilon\neq\pi}(DSPC(\mathbf{X}_\pi, \mathbf{X}_\varepsilon)), DSC(\mathbf{X}_\pi)\big)} \tag{3.5}$$

Finally, the index can be calculated as the weighted average of each $VC(\mathbf{X}_\pi)$:

$$DBCV(C) = \sum_{r=1}^{K} \frac{|\mathbf{X}_\pi|}{|\mathbf{X}|} VC(\mathbf{X}_\pi) \tag{3.6}$$

where K is the number of clusters, $|\mathbf{X}|$ is the total number of data objects, and $VC(\mathbf{X}_\pi)$ the validity of cluster $\mathbf{X}_\pi$. DBCV ranges from -1 to $+1$, going from partitions with sparse and overlapping clusters to partitions with dense and well-separated groups, respectively.

3.1.5 Adjusted Rand Index

The Adjusted Rand Index (ARI) [HA85] analyses the conformation of a partition Q to the original data distribution R (ground truth). It takes into account the probability that the partition has been generated from a random distribution of objects into clusters rather than by any intelligent mechanism. The unadjusted index ranges from 0 to 1, with larger values meaning better conformations, but ARI may yield negative values if the found partition is less attractive than the random expected partition:

$$ARI = \frac{a - \frac{(a+c)(a+b)}{E}}{\frac{(a+c)+(a+b)}{2} - \frac{(a+c)(a+b)}{E}} \tag{3.7}$$

where:

- a: number of pairs of data objects from the same class in R and same cluster in Q;
- b: number of pairs of data objects from the same class in R and different clusters in Q;
- c: number of pairs of data objects from different classes in R and to the same cluster in Q;
- d: number of pairs of data objects from different classes in R and to different clusters in Q;
- $E = a + b + c + d$

3.2 Baseline Algorithms

When referring to algorithms, David Wolpert once said [Mur12] that "there is no free lunch", which roughly captures the nature of machine learning algorithms: albeit one may be the best performer for a given application domain, it is very unlikely that it will be the best one for *any* domain. Based on this, data scientists come with new machine learning algorithms from time to time.

3.2.1 K-means

K-means [For65, M$^+$67] is a partitional algorithm that iteratively optimises the sum of squared errors (SSE) in an expectation-maximisation procedure. In its simplest form, it takes two parameters as input: the number of K groups (which is up to the user to specify) and a similarity measure, such as the euclidean distance[2]. It then randomly places K prototypes in the data space. Objects will be assigned to their closest prototype, and prototypes will be moved to match the center of the objects assigned to them. Since their positions change, other objects may become within range of a prototype, thus repeating the assigning-moving process until no object changes from group. The pseudo-code for K-means may be viewed in Figure 3.3, whereas Figure 3.4b details a single expectation-maximization step over the dataset of Figure 3.4a.

```
1: function $K$-MEANS($K$)
2:     randomly initializes $K$ prototypes
3:     while prototypes changing do
4:         form $K$ clusters by assigning each point to its closest prototype
5:         recalculate the centroid of each cluster
6:     return partition
```

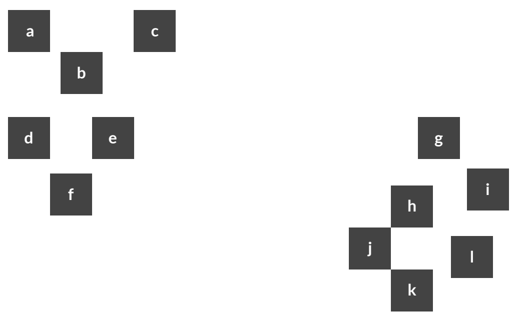
Figure 3.3 – Pseudo-code for K-means (omitting the similarity function for clarity purposes). Adapted from [TSK05].

3.2.2 DBSCAN

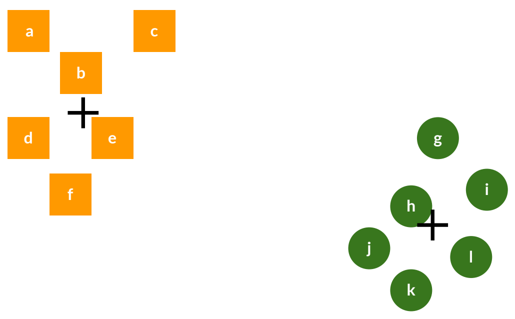
As the name suggests, Density Based Spatial Clustering of Applications with Noise (or DBSCAN for short) [EKS$^+$96] tends to analyze the neighborhood of an object before assigning it to a group. DBSCAN requires two user-defined parameters: a radius eps, which determines the maximum distance of a neighborhood, and a minimum number of neighbors to configure a group. If an object does not have a sufficient number of neighbors in its neighborhood, then it is considered noise.

The authors propose [EKS$^+$96] the following strategy to set DBSCAN parameters. The minimum number of objects in a neighborhood is up to the user to be defined (in the user profile example

[2]When using the euclidean distance, K-means tends to favour hyper-spherical groups.



(a)



(b)

Figure 3.4 – (a) A dataset of 12 objects. (b) A single run of expectation-maximization of K-means on the dataset of Figure 3.4a. Squares and circles are data objects, whereas crosses denote the group centroids (prototypes). Best viewed in colors.

of the beginning of this chapter, is it useful to have a profile with only 4 users?), and eps is set using the following procedure:

1. find the j-th closest neighbor for each object in the dataset;
2. store the j-th closest neighbor distance in a unidimensional array with length N;
3. sort the array in crescent order;
4. uses the distance before the biggest relative increase in distance required to set a neighborhood.

The idea is that the value before the biggest relative leap is the maximum value which can be used before having to increase the radius too much in order to obtain different partitions. Figure 3.5 illustrates the unidimensional array of distances, as well as the distance before the biggest relative increase.

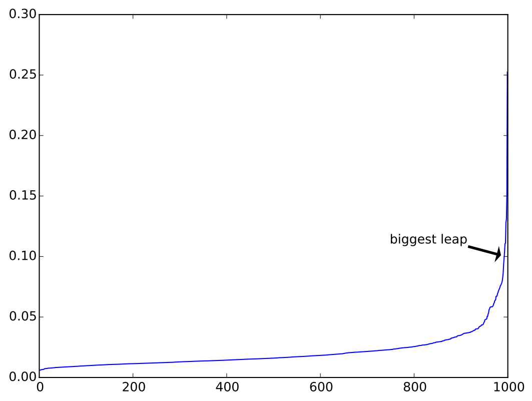


Figure 3.5 – For every object in a dataset, this graph describes the distance to its closest j neighbor, sorted in ascending order.

3.2.3 Hierarchical Clustering

Hierarchical algorithms perform the task of clustering by iteratively building hierarchic structures that correlate all objects within a dataset. At the end of the building procedure, this structure can yield $N-2$ possible partitions (excluding the partition in which all objects are singletons, clusters of only one object, and the trivial partition). For this reason, those algorithms present an advantage of not requiring setting the number of groups *a priori* [MC12, XW05].

There are two kinds of hierarchical clustering approaches: agglomerative and divisive. An agglomerative algorithm starts by treating each object in the dataset as a singleton. It then proceeds to merge clusters into bigger ones based on their similarity. A popular measure for similarity is the euclidean distance. In the process of merging clusters, a dendrogram records all merging performed so far. Between a merging and another, their similarity is then recomputed. This process is repeated until all objects belong to the same, trivial cluster. Divisive algorithms take the exact opposite procedure, but are unpopular due to its $O(2^N)$ computational cost – as opposed to $O(N^2 \log N)$ of agglomerative algorithms [BYL+15].

Agglomerative algorithms are further defined by the way that they recompute the similarity between groups. Single linkage [Sne57] selects the minimum distance between two clusters; complete linkage [Sør48], the maximum; average and median compute respectively the average and median distances, and so on.

Unlike K-means, hierarchical algorithms do not require the definition of the number of clusters beforehand; it is assigned later by analyzing the dendrogram and cutting it where the vertical distance is the largest, which very likely indicates a significant dissimilarity between two clusters. By analyzing the dendrogram of Figure 3.6, it is apparent that the dataset contains two groups, since the distance between clusters $\{f,e,d,b,a,c\}$ and $\{g,i,l,h,j,k\}$ is larger than any other inter-cluster distance. However, when a dendrogram presents several dissimilar clusters at more than one height, the task of selecting the correct the number of groups becomes more difficult.

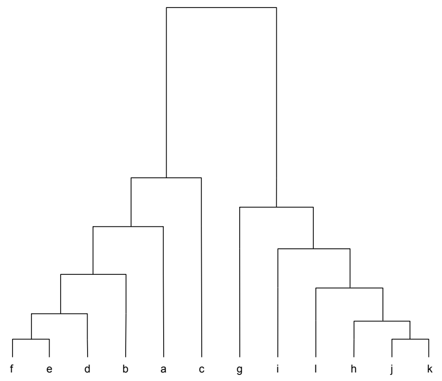


Figure 3.6 – A dendrogram made for the dataset of Figure 3.4a by complete linkage. The height of the vertical edges are the distances between the horizontally linked clusters.

3.2.4 EDAs for clustering

To the best of our knowledge, there is only one other strategy to perform clustering with EDAs, proposed by Santana et al. [SBL11]. The authors use three architectures of GM – a univariate marginal distribution, a tree-based distribution and a bayesian distribution – to initialize a matrix of similarities between objects. This matrix is then fed to affinity propagation [FD07], a clustering algorithm based on message exchange which aims to detect meaningful cluster representatives (i.e medoids). The authors apply this strategy to group human cancerous cells, which will be later used in a classification task.

The authors do not directly measure quality of clusters, but rather employ a penalized exemplar-based accuracy (E_{acc}), which measures the accuracy of the classification procedure by using clusters found by EDA-enhanced affinity propagation algorithm. The authors also assume that groups are classes. We seek to perform a comparison between our methods and the one of Santana et al. in future work, although we acknowledge that their EDA do not directly performs clustering, nor its paradigm (message-passing) is similar to the ones employed by our EDAs – medoid-based and density-based.

3.3 Clus-EDA

Clus-EDA is the first of our proposed algorithms for performing clustering through EDAs. It's name is an acronym for "Clustering with Estimation of Distribution Algorithms". It was submitted and accepted for presentation in the 31st ACM/SIGAPP Symposium on Applied Computing (SAC 2016), from April 4 to 8 in Pisa, Italy [CBQB16].

Clus-EDA is an EA for data clustering following the medoid-based approach with a variable number of clusters. It makes use of a univariate estimation of distribution algorithm (EDA) for evolving clustering partitions following the binary string encoding. Our hypothesis is that a medoid-based EDA is capable of achieving better results than traditional data clustering algorithms such as K-means [M$^+$67] and hierarchical agglomerative clustering [KR90]. Furthermore, we believe our approach is capable of outperforming a label-based EA called F-EAC [HCdC06], without the knowledge of number of clusters prior to its execution.

3.3.1 Individual Encoding

In Clus-EDA, any object may be sampled to become a medoid. The graphical model for Clus-EDA has N variables, with N as the number of objects in the dataset. The probability of an object X_i becoming a medoid is p_i, following a distribution $[0, 1] \in \mathbb{R}$. The variables in the GM do not interact with one another – that is, the probability that an object becomes a medoid does not affect the probability of neighboring objects.

For initialising the variables' probabilities, we run K-means [M$^+$67] multiple times varying K from 2 to $\sqrt{N}$ and select the number of clusters K^* from the partition that optimises a given clustering validity index. This heuristic is a thumb rule for defining the optimal value of K for methods that require this definition *a priori*. Even though Clus-EDA does not require setting a fixed number of clusters prior to its execution, we set $p_i = K^*/N$, $\forall i \in \{1, 2, ..., N\}$. By doing so, we potentially reduce the search-space of Clus-EDA, even though it will automatically adjust the probability vector throughout evolution, being capable of discovering partitions with any number of clusters. The clustering validity criterion used to define K^* is the Silhouette Width Criterion [Rou87], a widely-used index to validate data clustering partitions described in Section 3.1.1.

An individual is a binary string of N positions. An 1 in a given i position denotes that the i-th object in the dataset is a medoid for this individual. Non-medoid objects are assigned to the closest (using the euclidean distance) medoid object, thus forming clusters. In this encoding, the number of medoids for a given individual denotes the number its groups. Figure 3.7 demonstrates the encoding adopted by Clus-EDA.

The binary encoding adopted by Clus-EDA has several advantages over other typical encodings in evolutionary clustering problems. For instance, let us consider the case of the integer encoding

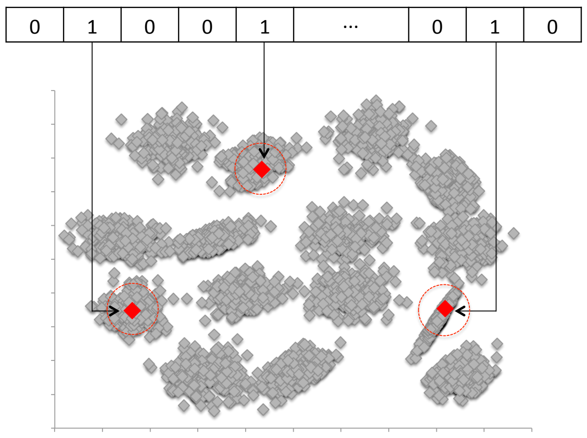


Figure 3.7 – Mapping of individuals adopted by Clus-EDA. Each individual is an array of length N. If an object is sampled to be a medoid, all non-medoid objects are assigned to the closest medoid object.

in which each gene (object) has a value over the alphabet $\{1, 2, ..., K\}$. Such an encoding is naturally redundant (1-to-many), since there are $k!$ different genotypes that represent the same solution [HCFDC09]. Furthermore, it assumes the number of clusters k is previously known, which is often not the case in real world applications.

3.3.2 Fitness function

Once all individuals are sampled, its fitness shall be asserted. The fitness function in Clus-EDA should be capable of evaluating the quality of the data clustering partition. Let N be the number of objects and M the number of attributes in the dataset. The cost of most validity criteria is quadratic in the number of objects – e.g., Dunn's ($O(M \cdot N^2)$), Silhouette Width Criterion ($O(M \cdot N^2)$), Gamma ($O(M \cdot N^2 + N^4/k)$), McClain-Rao ($O(M \cdot N^2)$), just to name a few. Hence, we decided to choose as fitness function a validity criterion whose complexity is linear in N, namely the Simplified Silhouette Width Criterion ($SSWC$) [HCdC06]. It is an efficient implementation of the traditional Silhouette Width Criterion (SWC) [Rou87], and is presented in further detail in Section 3.1.2.

For updating the probabilities, we pick the Φ% fittest individuals (i.e, highest SSWC) from the current generation, where the size of Φ is defined by the user. Thus, the probability of an object becoming a medoid in the next iteration of Clus-EDA is $1/|\Phi|$, with $|\Phi|$ as the number of fittest individuals from the current generation. Once all probabilities are updated, we resample the whole population. This procedure is repeated until a maximum number of iterations is reached.

3.3.3 Experimental Setup

In this section we detail the datasets that are employed in the experiments, as well as the clustering algorithms that participate in the analysis, the parameters used in Clus-EDA and in the baseline algorithms, and the evaluation measures to validate the results.

Datasets

For validating our results, we make use of a total of 9 datasets. The first 4 of them, namely s1, s2, s3, and s4, are artificial datasets proposed by Fränti and Virmajoki [FV06]. These datasets are 2-d data with $N = 5000$ and $K = 15$ Gaussian clusters with different degrees of cluster overlapping. The advantage of using artificial data is that we possess the “golden truth", i.e., the partition with the correct cluster assignments, so we can evaluate the clustering algorithms more objectively. Those datasets are presented in Figure 3.8.

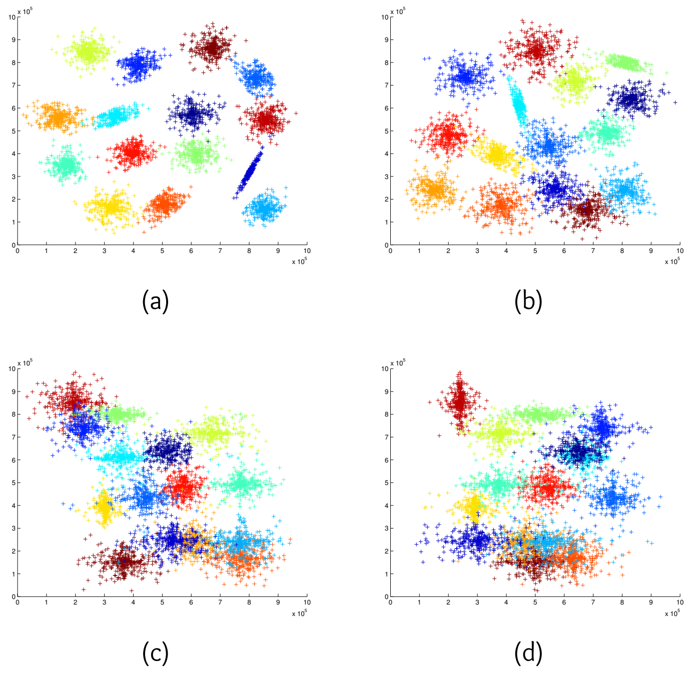


Figure 3.8 – "S" datasets: s1 (a), s2 (b), s3 (c) and s4 (d). Best viewed in colors.

The second set of datasets we make use were created by Yeung et al. [YMB03], which simulate data from microarray. The 5 synthetic microarray datasets, namely sin1, sin2, sin3, sin4, and sin5, are formed by $N = 400$ genes and $M = 20$ measurements (attributes). There are approximately 6 clusters with equal size in each of these dataset.

Baseline Algorithms

For comparison purposes in the empirical analysis, we make use of well-known clustering algorithms, namely K-means [M$^+$67] and UPGMA [KR90], as well as F-EAC [NCHdC11], which is a mutation-based EA (no crossover is performed whatsoever), with specialised mutation operators for the clustering task.

Parameters

Both K-means and UPGMA have a single parameter: the final number of clusters K. For deciding which value of K to use, we executed both algorithms for each dataset varying the number of clusters within $[2, \sqrt{N}]$, selecting the value of K from the partition that optimised the SWC. This thumb rule is often used for defining the number of clusters in algorithms such as K-means.

Regarding F-EAC and Clus-EDA, we executed both within a cycle of 500 individuals and 500 generations. We kept the remaining default parameters from EAC [NCHdC11]. For Clus-EDA, the only parameters are the value of the truncation selection, which we set to $t = 50\%$, and the value of the initial probability for each gene in the probabilistic vector (for generating a uniform distribution). As detailed in Section 3.3.1, we defined the initial probability as K^*/N, where k^* is the same value of K found by K-means in the thumb rule. Hence, the values of initial probability for Clus-EDA in the 9 datasets are presented in Table 3.1.

Table 3.1 – Characteristics of datasets used in this work along the initial probability of an object becoming a medoid in Clus-EDA.

Dataset	# objects	# attributes	# groups	Initial probability $p_i \forall i, i \in [1, N]$
s1	5000	2	15	0.0032
s2	5000	2	15	0.0028
s3	5000	2	15	0.0028
s4	5000	2	15	0.0034
sin1	400	20	6	0.0125
sin2	400	20	6	0.0150
sin3	400	20	6	0.0100
sin4	400	20	6	0.0150
sin5	400	20	6	0.0150

Evaluation Measures

Considering that all datasets that are used during the empirical analysis are synthetic, one of the evaluation measures we compute is the Adjusted Rand Index (ARI). ARI verifies the compatibility between the generated partition (henceforth called "clusters") and the reference partition (henceforth called "classes"). It is a measure *adjusted for chance*, i.e., when comparing two random partitions it yields a value close to zero. We also evaluate the results according to other two criteria, namely SWC

and DB. SWC is the original Silhouette Width Criterion [Rou87] without the prototype simplification for speeding it up, the latter being used in Clus-EDA's fitness function. Criterion DB is the Davies-Bouldin index [DB79], which is also an internal validity criterion that analyses the data alone. All measures are discussed in further detail in Section 3.1.

3.3.4 Experimental Results

We executed Clus-EDA and F-EAC 30 times by varying the seed of each execution, since they are evolutionary non-deterministic approaches. UPGMA and K-means were executed once per number of clusters, which was varied within $[2, \sqrt{N}]$. Then, we selected the partition that optimised the SWC validity index [Rou87] for each one of them. Table 3.2 presents a summary with the experimental results.

Table 3.2 – Results summary. Values for the real number of clusters (K), the estimated number of clusters (K^*), Simplified Silhouette Width Criterion, Davies-Bouldin index, and Adjusted Rand Index for K-means, UPGMA, F-EAC, and Clus-EDA. Values for Clus-EDA and F-EAC are averages of 30 executions.

		K-means				UPGMA				F-EAC				Clus-EDA			
Dataset	K	K^*	SWC	DB	ARI	K^*	SWC	DB	ARI	K^*	SWC	DB	ARI	K^*	SWC	DB	ARI
s1	15	16.00	0.63	0.61	0.90	19.00	0.51	0.63	0.85	15.00	**0.71**	0.46	0.87	15.07	**0.71**	**0.42**	**0.99**
s2	15	14.00	0.61	**0.48**	0.89	15.00	0.52	0.68	0.91	15.00	**0.63**	0.57	0.87	15.07	0.62	0.53	**0.93**
s3	15	14.00	0.41	**0.69**	0.62	15.00	0.19	0.91	0.69	15.00	**0.49**	0.76	**0.86**	14.73	**0.49**	0.70	0.73
s4	15	17.00	0.47	**0.68**	0.64	18.00	0.10	0.98	0.61	15.00	**0.48**	0.77	**0.85**	15.20	0.47	0.73	0.65
sin1	6	5.00	0.63	0.53	0.67	6.00	**0.65**	0.71	**0.84**	6.00	**0.65**	0.60	0.67	6.00	**0.65**	**0.52**	**0.84**
sin2	6	6.00	0.54	1.11	0.43	5.00	**0.69**	0.62	**0.67**	5.00	**0.69**	0.48	0.55	5.00	**0.69**	**0.43**	**0.67**
sin3	6	4.00	0.45	0.89	0.44	5.00	**0.73**	0.47	**0.54**	4.00	0.72	0.44	0.44	4.00	0.72	**0.43**	**0.54**
sin4	6	6.00	0.51	1.06	0.64	8.00	**0.70**	0.91	0.83	6.93	0.69	0.62	0.67	6.00	0.69	**0.43**	**0.84**
sin5	6	6.00	0.55	0.78	0.65	5.00	**0.64**	0.70	0.67	8.00	**0.64**	0.57	0.67	6.00	0.63	**0.56**	**0.83**

Our first analysis in this round of experiments was regarding the number of clusters found by each method. Clus-EDA presents the lowest average absolute error (0.40) regarding the estimated number of clusters, followed by F-EAC (0.66), K-means (0.89) and UPGMA (1.33). In other words, Clus-EDA is the algorithm that best estimates the number of clusters, though we are aware that simply estimating the proper number of clusters is not enough for a clustering algorithm to be deemed *effective*.

Therefore, our second analysis is regarding the Adjusted Rand Index (ARI). Note once again that Clus-EDA seems to be the best option among the algorithms that were executed. It provides the largest ARI value in 7 of the 9 datasets, even though it ties with UPGMA in three of them. By presenting the best ARI values, Clus-EDA demonstrates it has the greatest potential to approximate the golden truth provided by each of these datasets.

Our next analysis was regarding the internal validity criteria SWC and DB. Regarding SWC, Clus-EDA together with F-EAC and UPGMA shared wins, with a small advantage to F-EAC overall. In terms of the DB index, Clus-EDA once again has shown to be the best option, winning in 6 out of 9

datasets (with K-means winning in the remaining three datasets). Hence, we have showed that Clus-EDA is not only the best clustering algorithm in estimating the correct number of clusters, but that it also is the best algorithm regarding both external and internal clustering validity criteria.

3.3.5 Final Remarks

This section described Clus-EDA, an estimation of distribution algorithm for medoid-based clustering. As previously said in Section 3.3, Clus-EDA was submitted and accepted for presentation in the 31st ACM/SIGAPP Symposium on Applied Computing (SAC 2016), from April 4 to 8 in Pisa, Italy [CBQB16].

The proposed approach employs a simple but efficient and effective evolutionary framework that estimates a univariate marginal distribution model to define cluster prototypes. To guide the iterative refinement of the probabilistic model, Clus-EDA employs a clustering internal validity criterion that has complexity $O(M \cdot N)$, i.e., linear in the number of objects and attributes.

We compared Clus-EDA with k-means [M^{+}67], an hierarchical agglomerative clustering [KR90], and also with an evolutionary algorithm F-EAC [NCHdC11]. For comparison purposes, we employed 9 clustering datasets: 4 of them were artificially generated based on Gaussian clusters [FV06], and 5 of them simulate microarray gene expression data [YMB03]. Results show that Clus-EDA can generate data partitions that provide a greater agreement regarding the reference partitions, outperforming the baseline clustering algorithms in both external and internal clustering validity criteria. As future work, we intend to verify whether more sophisticated probabilistic models would wield improved results for medoid-based clustering.

3.4 PASCAL

PASCAL is the second of our proposed algorithms for performing clustering through the use of EDAs. Its name stands for "Parameterless Shape-Independent Clustering Algorithm". It was submitted and accepted for presentation at the 2016 IEEE Congress on Evolutionary Computation, held at Vancouver, Canada from 24 to 29 july [CB16]. PASCAL relies on the assumption that, although clustering is a task done without prior knowledge on the data, geometrical aspects of it may give a hint into how to assimilate objects into groups.

Let us assume that all objects are connected to each other through E edges. An edge weight is given by the distance between a pair of objects (say the Euclidean distance, without loss of generality). Hence, we have a dense graph with objects representing the vertices. If we now consider that each edge links two objects that belong to the same cluster, the task of finding clusters is reduced to the one of removing unnecessary edges from the graph. This is a decrease in the search space from $\frac{1}{K!}\sum_{i=0}^{K}(-1)^i\binom{K}{i}(K-i)^N$ possibilities [TSK05] to $N!/(N-K)!$, for a given set of N M-dimensional points and K number of partitions.

3.4.1 Individual encoding

Besides considering that geometrical aspects, other assumption which can be made is that far-away objects are much less likely to belong to the same cluster than closer ones. In fact, this assumption may be expanded to consider the closest neighbor only. If the closest neighbor for a given object does not belong to the same cluster than it, why would farther objects be more likely to do? This obviously holds true for scenarios where minimizing distance is equivalent to maximising similarity.

Hence, instead of building a fully dense graph, PASCAL builds a minimum spanning tree (MST), which connects the set of N vertices with a subset E_π of $N-1$ edges from E that minimise the overall weight of the tree while preventing cycles. The distance measure used for calculating the weight of edges can be any one which satisfies the symmetry ($d(X_i, X_j) = d(X_j, X_i)$), positivity ($d(X_i, X_j) \geq 0 \; \forall i, j \in [0, \ldots, N]$), and reflexivity ($d(X_i, X_j) = 0$ iff $X_i = X_j$) properties, with $i \neq j$ [XW05]. A well-known distance measure that satisfies all these properties plus the triangle inequality ($d(X_i, X_j) \leq D(X_i, X_l) + D(X_l, X_j)$ for all X_i, X_j, and X_l, $i \neq j$, $j \neq l$, $i \neq l$) is the Euclidean distance, which is used as the default distance measure in PASCAL.

PASCAL employs the Kruskal's algorithm [CLRS09] for generating the MST based on a pre-computed distance matrix. Kruskal's algorithm starts by considering each vertex as a candidate subtree, and also all possible edges in the graph. It then iteratively removes an edge with minimum weight from the set of edges, and if that edge connects different trees it combines them into a single tree. At the end of the process, since the graph is connected the single resulting tree is an MST containing all N vertices (i.e., objects) and $N - 1$ edges. Figure 3.9 shows the output of Kruskal's algorithm at every iteration for a dataset with 3 objects.

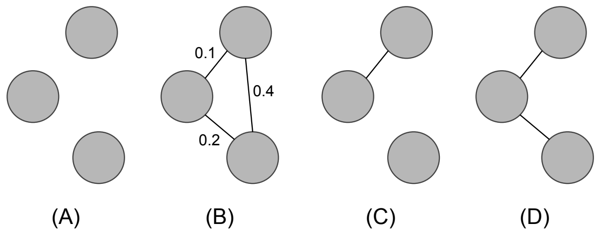
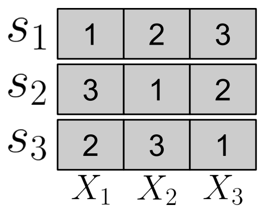


Figure 3.9 – (A) A bidimensional dataset with 3 objects. (B) The fully-connected graph that links all objects and the respective distances. (C) Forest of subtrees in the first iteration of Kruskal's algorithm. (D) Final subtree found in the second iteration of the Kruskal's algorithm, which is also a MST.

PASCAL approaches the problem of generating clusters from this MST as the one of removing its edges in a way that disconnected subtrees represent different groups. As previously said, there are $2^{(N-1)} - 2$ valid clustering partitions to be found, meaning that the number of solutions grows exponentially with the number of objects. Nonetheless, considering that each individual is the set of removed edges presents an advantage over the label-based approach, usually employed in evolutionary clustering algorithms (e.g., [HCdC06]), which assigns the group label to each one of the objects, as depicted in Figure 3.10. Note that this approach allows the encoding of multiple genotypes (i.e sets of labels) which, in fact, represent the same phenotype (i.e partition).



	X_1	X_2	X_3
s_1	1	2	3
s_2	3	1	2
s_3	2	3	1

Figure 3.10 – Three individuals (s_1, s_2 and s_3) with different label assignments for objects X_1, X_2, X_3, but with the same phenotype.

Once the MST is built, it is mapped into a probabilistic graphical model (GM) [Mur12]. A GM is a resource used by EDAs to sample new individuals for comprising the population of candidate solutions. GMs may come in two types, directed and undirected graphical models (DGM and UGM, respectively) [Mur12]. In a directed graphical model, a parent variable may influence the outcome of a child variable, but otherwise is impossible; this behavior is allowed when using UGMs. Nonetheless, the type of GM dictates how variables should interact between themselves, but does not explicitly group variables into interaction groups (i.e, it does not say that the outcome of variable V_i affects the outcome of V_j). PASCAL uses a DGM as its mechanism for sampling values of variables; however, there is no interdependence between variables.

In PASCAL, we convert MSTs undirected nature to a directed one, assigning a random node in the tree to its root. We then assign a variable to each one of the *edges* of the MST. The initial probability of connected objects X_i and X_j belonging to the same cluster C_l is given by

$$p(\{X_i, X_j\} \in C_l) = 1 - \frac{d(X_i, X_j)}{\sum\limits_{e \in E_\pi} w_e} \tag{3.8}$$

where $d(X_i, X_j)$ is the distance between objects X_i and X_j and w_e is the weight of edge e that belongs to the set of edges E_π of the MST. Hence, closer objects have a higher initial probability to be in the same cluster than farther ones. In this way, we have $N-1$ variables, with N as the number of objects in the dataset. Figure 3.11 depicts the arrangement of variables.

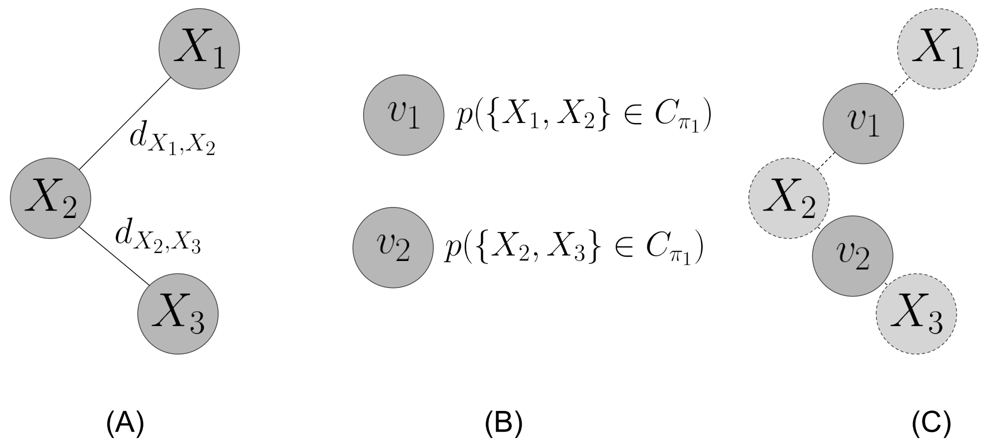


Figure 3.11 – (A) MST with objects and euclidean distance between neighboring objects. (B) Encoding of variables in PASCAL's DGM. Distances are converted to probabilities following Equation 3.8. Note that each variable encodes the probability that two neighboring objects belong to the same group. Also note that the outcome of a variables does not affect the outcome of other variables (i.e, group of other pairs of objects), configuring a univariate distribution. (C) Superposition of MST and GM.

Once the MST is converted to a GM, it is possible to sample individuals from each variables' distribution. Since the outcome of each variable is a boolean value denoting a must-link or cannot-link condition, an individual is encoded as a list of boolean values. This gives PASCAL enhanced agility and speed when sampling from and updating the distribution. Note that chains of linked objects translate to larger groups, whereas pairs of objects translates to smaller – and possibly outlier – groups.

PASCAL does not accept any parameter regarding the clustering process, such as number of groups K, minimum distance between objects to be considered from the same group (as opposed to DBSCAN [EKS+96]), etc. Its only parameters are related to the search procedure, which are hyper-parameters in relation to clustering: maximum number of individuals to keep in memory at the same time; maximum time of search (i.e iterations/generations); amount of individuals to use for updating the GM; and individuals to be replaced in the next generation. No attempt was made to tune these parameters during the experimental analysis, presented in Section 3.4.6. In fact, PASCAL is robust and insensitive to these parameter choices in terms of cluster quality, and that well-established common-sense values for these search parameters are rather sufficient for performing high-quality clustering with PASCAL. Its pseudocode is presented in Figure 3.12.

```
1: function PASCAL($data$, $iter$, $frac\_induce$, $frac\_replace$)
2:     compute the distance matrix from $data$
3:     build $MST$ from $data$
4:     initialise GM with objects as variables, edges from $MST$ as relationships, and edge weights as probabilities
5:     sample an entire population with GM
6:     evaluate population
7:     while $g < G$ and GM has not converged yet do
8:         update GM based on $frac\_induce$ individuals with best fitness
9:         remove $frac\_replace$ of worst individuals from the population
10:        sample $frac\_replace$ individuals from GM and add to the population
11:        evaluate population
12:        $g \leftarrow g + 1$
13:    return best individual from the population
```

Figure 3.12 – Pseudo-code of PASCAL.

3.4.2 Fitness function

For evaluating the quality of generated partitions, PASCAL uses density-based clustering validation (DBCV) [MJC+14] as its fitness function, as described in Section 3.1.4. By using a density-based criterion, PASCAL is capable of detecting arbitrarily-shaped clusters, since its assumption regarding what a cluster is will be based on the concept of finding dense areas separated by sparse regions.

3.4.3 Time complexity

PASCAL's time complexity is computed as follows. The calculation of the distance matrix between N objects takes $O(N^2)$. Finding the MST through Kruskal's algorithm takes $O(N \log N)$. The main loop of the EDA runs in the worst case for G times, where G is the number of max iterations. The number of individuals to have its fitness calculated is proportional to the fraction used as input for the EDA, $|S|$. Since the whole population must be sampled in the first iteration, it has a complexity of $O(|S|(1+G))$. The worst case for calculating DBCV is when all objects belong to the same cluster, since γ will be calculated using all $N-1$ objects as neighbours and at most $N-2$ objects will be an inner node for finding the MST. Hence, it has a complexity of $O((N-2)^2+(N-2)\log(N-2)+N+1)$, which corresponds respectively for calculating γ, finding the MST of the trivial cluster, finding DSC, and calculating the DBCV index itself. Updating the GM based on fittest function takes $O(|\Phi|)$, where $|\Phi|$ is the number of fittest individuals. Hence, our algorithm has a complexity of $O(N(1+\log N) + I(1+G)((N-2)^2 + (N-2)^2 \log (N-2)^2 + N + 1 + |\Phi|)$, which is $\approx O(N^2)$ for large values of N.

3.4.4 Related Work

Using minimum spanning trees for clustering is not a novel approach. For instance, the dendrogram produced by single linkage (introduced in Section 3.2.3) is in fact a MST. The difference between PASCAL and single linkage is two-fold. First, PASCAL does not require setting the value of K (or, in other words, a horizontal cut in the dendrogram) in order to assign objects to clusters. Second, PASCAL allows the "regretting" from a cluster assignment for a given object. In the dendrogram, there is only one partition which yields K clusters. In order to obtain different partitions for the same number of groups, the user must run other hierarchical agglomerative clustering algorithms such as complete linkage. PASCAL, on the other hand, has $N!/(N-K)!$ possible partitions for each value of K ranging in $[2, N-1]$.

The work of Zhou et al. [ZGH11] propose two procedural algorithms for performing clustering with minimum spanning trees, one K-constrained and the other unconstrained. Since PASCAL automatically detects the number of clusters, we describe here the unconstrained algorithm. It starts by building a MST from the dataset, and then it iteratively removes edges from the MST. By removing an edge, it produces two clusters of data objects. The approach used for removing edges is to remove those that contribute the most for increasing the weighted standard deviation of all edges in the set of subtrees. It then performs a 6-th order regression analysis with the information of how much is reduced in terms of standard deviation with the number of removed edges. When removing an edge ceases to decrease the standard deviation of the partition, the corresponding value of K is chosen and the subtrees generated from that configuration is returned.

Regarding evolutionary algorithms for clustering, Alves et al. [ACH06, HCdC06] propose a Fast Evolutionary Algorithm for Clustering (F-EAC), a mutation-based algorithm which further improves the EAC [HdCC04]. F-EAC encodes all N objects of the dataset in one array of N positions. For each position, it randomly assigns a value in the range $[1, K]$ during the first generation, where K is the number of partitions. The initial value of K is a starting point, since it is constantly changed by the algorithm within its procedure. F-EAC proceeds to mutate individuals in order to update cluster assignments for each object, either by splitting or merging clusters. Two functions are used for fitness evaluation across the several variations of F-EAC presented in the work: Simplified Silhouette Width Criterion (SSWC), which is a simplification over the original silhouette; and Rand Index, the original unadjusted version. The authors develop a set of F-EAC variants, which can accurately predict the correct number of clusters and are faster than the original EAC implementation. As mentioned before, F-EAC requires an initial value of K to initiate the clustering process. Furthermore, since it uses the SSWC as fitness function, it tends to favour spherical clusters. The variation in which it uses the Rand Index is not realistic since in most domains there is no known ground truth.

3.4.5 Experimental Setup

The following sections are an excerpt of the text found in [CBQB16], which describes our first experiments with PASCAL. In this work, we use the following hyper-parameters for running PASCAL: 500 individuals, maximum of 100 iterations, updating the GM based on 10% of the fittest individuals and full replacement of the population. We did not attempt to tune these parameters. We run

PASCAL 10 times on each dataset, varying the random seed that controls evolution. In the following sections, we comment on the datasets, algorithms, and evaluation criteria that were used during the empirical analysis of this work.

Baseline Algorithms

For validating the performance of PASCAL, we compare it with 4 well-known clustering algorithms: K-means [Llo06], DBSCAN [EKS$^+$96], single and complete linkage [JD88, MC12]. All those algorithms are further explained in Section 3.2.

For parametrising the baseline algorithms, we employed two different strategies. For DBSCAN, we set a neighbourhood of 4 objects and follow the strategy described in Section 3.2.2. For K-means, single linkage and complete linkage, we executed them selecting K from 2 to $\sqrt{N}$. The K used for generating the partition with the best DBCV index is used as input for the algorithms. We decided to use DBCV as the validity criterion for choosing the best partition since it is the same criterion optimised by PASCAL during its evolutionary search. K-means was executed 10 times by varying the random seed for defining the initial prototypes for each possible value of K in the interval $[2, \sqrt{N}]$.

Datasets

During the empirical analysis, we verify the performance of the algorithms in 10 datasets: 9 of them were artificially generated and one is a real-world labeled dataset. For the real dataset, we make the further (probably naïve) assumption that the classes are equivalent to clusters. The real dataset in question is the well-known Fischer's Iris data, which contains information about petal length and width, and sepal length and width of three specimens of flowers [Lic13]. The artificial datasets blobs2, circles0 and moons0 were generated using the Scikit Learn toolkit for the Python programming language [PVG$^+$11]. The rest of the datasets were generated in Octave [JWEW15] using the code made available by Kools [Koo16]. All artificial datasets are two-dimensional for the sake of visualisation. Table 3.3 presents the characteristics of all datasets, and the 9 artificial datasets can be seen in the first column of Figure 3.13.

Table 3.3 – Artificial and real datasets used in the empirical analysis.

	dataset	# features	# objects	# clusters
Real	Iris	4	150	3
	blobs2	2	1,000	2
	circles0	2	1,000	2
	moons0	2	1,000	2
	outlier	2	1,000	2
Artificial	clusterincluster (cinc)	2	1,012	2
	corners	2	1,000	4
	crescentfullmoon (cfm)	2	1,000	2
	halfkernel	2	1,000	2
	twospirals	2	1,000	2

Evaluation Measures

Unlike classification, clustering is a subjective task and the quality of the result may vary according to the prior assumptions of the validity criterion that is used for evaluating the partitions. For analysing the clustering results, we decided to employ 2 clustering validity criteria: DBCV, which is the criterion optimised by PASCAL during evolution, and the Adjusted Rand Index. Note that DBCV is an *internal* validity index, which means it takes into account only the data itself to compute the partition's quality. The Adjusted Rand Index, on the other hand, is an *external* validity criterion, which compares the resulting partition with the *ground truth*, i.e., an external partition that is allegedly the expected result. Since we are using 9 artificial datasets and one labeled real-world dataset, we do have the external partitions to properly evaluate the clustering quality, and thus the *internal* validity index is just presented for the sake of completeness. DBCV and ARI are further explained in Section 3.1.4.

3.4.6 Experimental Results

We present all results of this experimental analysis in Table 3.4, and in Figure 3.13 we show the comparison between the ground truth and the partitions provided by each algorithm. Note that PASCAL and all baseline algorithms are capable of correctly choosing the number of clusters in 8 out of the 10 datasets. Recall that we had to execute a multiple-runs procedure followed by the evaluation of an internal validity criterion to define the number of clusters for K-means and complete/single linkage, since they require the user to set the value of K. We can infer that using DBCV as an internal validity criterion for estimating the number of clusters for algorithms that need to set that parameter is indeed a good alternative. Yet, we should give emphasis to the fact that neither DBSCAN nor PASCAL require any sophisticated procedure to properly estimate the number of clusters. Moreover, note that PASCAL is, in fact, the only algorithm that does not require any procedure at all to define a set of parameters so it can be successfully executed.

The main evaluation criterion in this experimental analysis is ARI, which indicates the level of conformity between the provided partitions and the real distribution of the data. Note that both PASCAL and DBSCAN can perfectly reproduce the ground truth in 8 out of the 9 artificial datasets, substantially outperforming both hierarchical agglomerative methods and K-means. Given the variety of shapes present in the artificial datasets, it was expected that K-means would fail in reproducing the ground truth, since it is only capable of generating hyper-spherical clusters. In terms of functioning, the most similar algorithm to PASCAL is Single Linkage, but note that it fails in providing the ground truth for the outlier dataset – whereas PASCAL can reproduce it correctly. This dataset has two "real" groups and two outlier groups – very compact clusters where all objects have the same characteristics, which is a strong evidence towards treating them as noise. We employ the following methodology to evaluate algorithms in this dataset: if the algorithm finds either the two "real" groups or all the four groups, we deem it to be correct (note, for instance, the difference in ARI values between K-means and DBSCAN for a further insight in Table 3.4). From the Figure 3.13, it is evident that Single Linkage detects an outlier group and puts all other instances in another group, which is obviously incorrect.

Table 3.4 – Results for all baseline algorithms and PASCAL. Bold numbers indicate the best results for the given dataset.

Dataset	Complete Linkage			Single Linkage			K-means			DBSCAN			PASCAL		
	ARI	DBCV	K	ARI	DBCV	K	ARI	DBCV	K	ARI	DBCV	K	ARI	DBCV	K
blobs2	**1.00**	-0.23	**2**	**1.00**	-0.23	2	**1.00±0.00**	-0.23±0.00	**2±0**	**1.00**	**-0.09**	**2**	**1.00±0.00**	**-0.09±0.00**	**2±0**
moons0	0.59	-0.81	**2**	**1.00**	**0.74**	2	0.53±0.00	-0.82±0.00	**2±0**	**1.00**	**0.74**	**2**	**1.00±0.00**	**0.74±0.00**	**2±0**
circles0	0.00	-0.39	**2**	**1.00**	**0.30**	2	0.00±0.00	-0.40±0.00	**2±0**	**1.00**	**0.30**	**2**	**1.00±0.00**	**0.30±0.00**	**2±0**
outlier	**1.00**	0.09	**2**	0.00	-0.33	2	0.99±0.00	-0.37±0.00	**2±0**	**1.00**	**0.54**	**2**	**1.00±0.00**	**0.54±0.00**	**2±0**
cinc	0.53	-0.25	**2**	**1.00**	**0.38**	2	0.00±0.00	-0.93±0.02	**2±0**	**1.00**	**0.38**	**2**	**1.00±0.00**	**0.38±0.00**	**2±0**
corners	0.37	-0.73	**4**	**1.00**	**0.28**	4	0.70±0.30	-0.23±0.55	**4±0**	**1.00**	**0.28**	**4**	**1.00±0.00**	**0.28±0.00**	**4±0**
cfm	0.65	-0.75	**2**	**1.00**	**0.03**	2	0.21±0.01	-0.67±0.04	**2±0**	**1.00**	**0.03**	**2**	**1.00±0.00**	**0.03±0.00**	**2±0**
halfkernel	0.02	-0.97	**2**	**1.00**	**0.35**	2	0.09±0.13	-0.88±0.00	**2±0**	0.68	0.13	5	**1.00±0.00**	**0.35±0.00**	**2±0**
twospirals	0.03	-0.58	24	0.31	-0.33	24	0.05±0.00	-0.47±0.04	24±0	**1.00**	-0.57	**2**	0.46±0.06	**-0.25±0.04**	5±1
iris	0.22	-0.65	2	**0.57**	**0.22**	2	**0.57±0.00**	**0.22±0.00**	2±0	**0.57**	-0.05	2	**0.57±0.00**	**0.22±0.00**	2±0
Victories	2	0	8	8	7	8	2	1	8	9	7	8	9	10	8

It was also expected that PASCAL would outperform all baseline algorithms in terms of DBCV, since it is the very own criterion optimised during its evolution. Indeed, the values of DBCV for the partitions provided by PASCAL were the best for all 10 datasets. Both DBSCAN and Single Linkage had the best DBCV values in 7 datasets, substantially better than K-means (one dataset) and Complete Linkage (0 datasets). It seems safe to affirm that the choice of a density-based validity criterion such as DBCV proved to be a good option for looking for partitions in arbitrarily-shaped datasets, specially considering the correlation between ARI and DBCV values.

Another point worth mentioning in the experimental analysis is regarding the halfkernel dataset, which is formed by two semi-circle structures. Even though it can be hard to visualise on image, there are changes in density across the structure of each semi-circle. That is probably the reason for DBSCAN failing to detect these two semi-circles, considering that it detects the lower-density regions as inter-cluster areas, and thus ends up generating more clusters than necessary. PASCAL, on the other hand, is perfectly capable of detecting the two clusters thus yielding the best ARI value.

Notwithstanding, PASCAL did fail to properly detect the two spirals in the twospirals dataset, whereas DBSCAN was the only algorithm to correctly detect the groups. More interestingly, the twospirals dataset was the only one in which PASCAL had a variety of behaviour in its 10 runs, finding partitions ranging from 4 to 6 clusters, instead of the real value of 2. We believe that happened because of a particularity with the DBCV criterion, as follows. The centre of the dataset is a low-density area with objects from the two spirals. By following the course of the spirals, the objects condense into a high-density distribution, misleading DBCV to understand that clustering the whole centre is a good idea. Indeed, PASCAL achieves the largest DBCV value for this dataset, clearly indicating that DBCV is not particularly suited for this problem. We are already studying new strategies for modifying DBCV so it can cope with this scenario while keeping the good results achieved so far.

Finally, it is worth mentioning that PASCAL is quite stable across multiple runs. Only for the twospirals dataset, which seems to deceive the behaviour of DBCV, PASCAL ended up generating different results. Perhaps this is a particular case in which tuning the search parameters of the EDA

Ground Truth	PASCAL	DBSCAN	K-means	Single Linkage	Complete Linkage

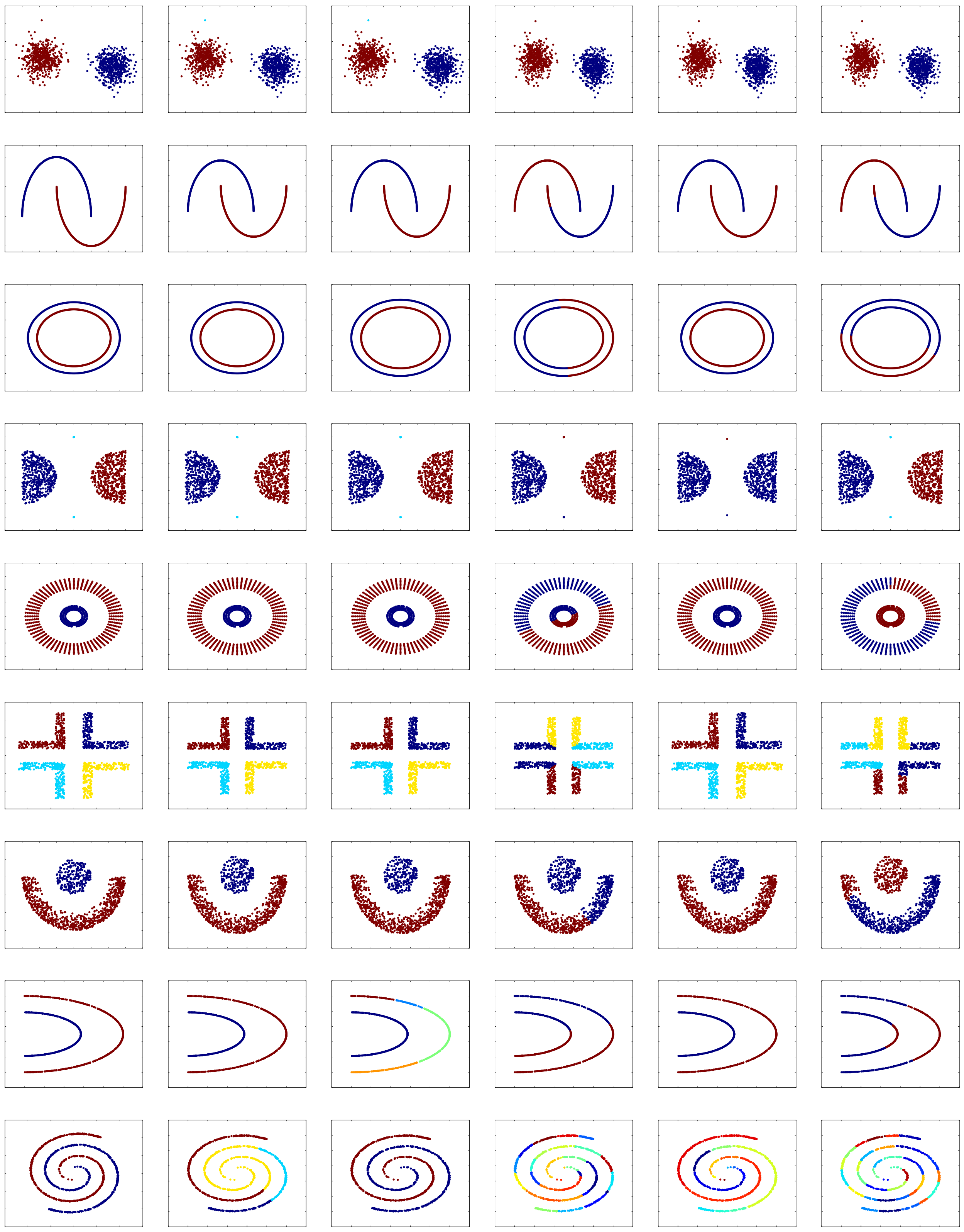


Figure 3.13 – Ground truth and partitions found by each algorithm. Each cluster is identified by a given color. Best viewed in colors.

would yield more interesting results. For instance, we assume that perhaps with a larger number of individuals (and perhaps a larger budget) PASCAL would be capable of identifying more interesting conformations that could lead to larger DBCV values. Nevertheless, since the whole point of PASCAL is to be *parameterless*, suggesting to tune the search parameters may not be the right solution for the problem, so we prefer to investigate modifications of DBCV or perhaps the inclusion of multiple validity criterion within the fitness function so the final user does not need to worry with setting parameters for PASCAL at all.

3.4.7 Final Remarks

This section described PASCAL, a novel and effective EDA for performing data clustering that is capable of addressing arbitrarily-shaped clusters without the need of setting specific and decisive parameters such as the number of clusters, minimum density, radius, etc. PASCAL makes use of a MST to identify possible constraints of must-link/cannot link between pairs of objects, and it optimises a density-based validity criterion during its search for the best partition. As previously said in Section 3.4, it was submitted and accepted for presentation at the 2016 IEEE Congress on Evolutionary Computation, held at Vancouver, Canada from 24 to 29 july [CB16].

PASCAL is compared to well-known clustering algorithms that employ different strategies to perform clustering, such as K-means [Llo06], Single Linkage [JD88], Complete Linkage [JD88], and DBSCAN [EKS$^+$96]. By performing an empirical analysis with 10 datasets whose ground truth partitions were previously known, we show that PASCAL is capable of not only correctly identifying the number of clusters but also of presenting the largest possible conformation between the predicted partitions and the real ones in 8 out of the 10 datasets. Indeed, PASCAL seems to perform as strongly as DBSCAN, though with the further advantage of not requiring any critical clustering parameters, whereas DBSCAN requires two of them. As future work, we aim to investigate which modifications are required in PASCAL's density-based fitness function so it can successfully deal with scenarios that PASCAL failed to identify the ground truth. Moreover, we intend to verify whether a multi-objective fitness function would be well-suited for addressing this issue.

4. CLASSIFICATION

Classification is the data mining task of assigning a category to unknown data. One way of performing it is by using supervised machine learning algorithms. Those algorithms will build a predictive model based on known data in order to predict the class of unknown data. Thus, there must exist available labeled training data.

The difference between classification and regression, a similar task, lies in the domain of the class attribute. When one seeks to assign a class to an instance, the task is classification; when the objective is to estimate a real value (for example, based on land size and number of floors, how much costs a house) the task is regression.

There are several algorithms within machine learning to perform classification. Decision trees are among the most popular due to its easiness of comprehension, robustness to noise, speed regarding both training and prediction and ability to deal with redundant attributes [BBDCF12]. Although some algorithms achieve state-of-the-art results (Support Vector Machines [VC95, Vap99] and Deep neural networks [GBC16], just to name a few), most of them perform a black-box predictive process: it is unknown how the data is being treated. Decision trees, on the other hand, provide a transparent, white-box predictive process. Consider for example the case of a medical doctor (MD) who needs to analyze symptoms from a patient in order to make a diagnosis. It is unadvised for the MD to simply rely on the prediction of an algorithm, as it may be incorrect; it is rather more useful to analyze the predictive process, correct it if wrong or, if right, incorporate this knowledge into his or her experience.

There are exponential many decision trees that can be built from the same data, with different levels of predictive quality and compactness. Generating the optimal decision tree regarding a given quality criterion is a combinatorial optimization problem. For instance, the problem of inducing an optimal decision-tree from decision tables is NP-hard, whereas evolving a minimum binary tree is NP-complete. In practice, current decision-tree induction algorithms are heuristic-driven, employing a greedy local-search for generating reasonably-accurate albeit suboptimal decision trees [BdCF15].

There are at least two major problems related to the greedy local-search employed by traditional decision-tree induction algorithms [BBdC$^+$09]: (i) the greedy strategy often produces locally (rather than globally) optimal solutions, (ii) recursive partitioning iteratively degrades the quality of the dataset for the purpose of statistical inference, contributing to the problem of data overfitting. To address these problems, distinct strategies have been proposed in the last decade or so, in particular focusing on meta-heuristics based on global search [BBdC$^+$09, BBDCF12].

The rest of this chapter is organized as follows. Section 4.1 reviews the concepts of decision-trees. Section 4.2 reviews how classification algorithms are evaluated, whereas Section 4.3 describes the data preparation for this task. Ardennes, the EDA for decision-tree induction, is presented in Section 4.4.

4.1 Decision-trees

Decision trees are predictive models capable of performing both classification and regression tasks. A decision tree may be viewed as a directed acyclic graph (DAG) [BdCF15] with inner and leaf nodes and edges. Inner nodes split data according to a test over a single predictive attribute into two or more subsets. The edges in this DAG are the possible test outcomes, i.e the value which the instance must have to follow one of the outgoing edges. Finally, a leaf node encodes one of the possible values for the class attribute. A decision tree which classifies instances into one of the two categories of the dataset of Table 1.1 is presented in Figure 4.1.

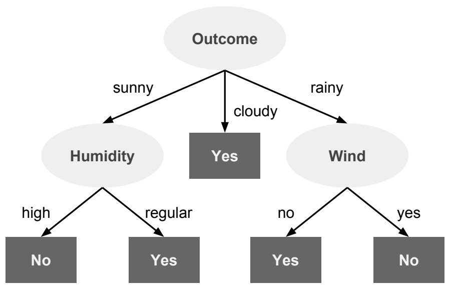


Figure 4.1 – A top-down induced decision-tree which classifies unknown records of the dataset presented in Table 1.1 into one of the two possibles classes: whether to play or not tennis. Generated using the J48 algorithm under the Weka Toolkit [WF05].

Besides the aforementioned benefit of a transparent classification process, decision trees are also fast: one object does not have to travel more than H levels (with H as the maximum height of a given tree) in order to receive a prediction. Decision trees are also robust to noise when anti-overfitting measures are employed [BA97], and do not present disadvantages when two attributes are strongly correlated [TSK05, chap. 4, p. 169].

4.1.1 Top-down induction

Decision trees may be generated by either top-down or bottom-up approaches. Top-down induction is based on, given a set of heterogeneous data arriving at the root of a decision-tree, evolve the tree's branches (i.e nodes) until a stopping criterion is met. Bottom-up goes the other way around: from a set of homogeneous data at the bottom of the tree, build branches in order to reach a single, heterogeneous root node.

Top-down induction is by far the most popular decision-tree induction method [BdCF15], and is better translated by the Hunt's Algorithm [HMS66], which acts as a "core" strategy to several deterministic top-down induction strategies. In order to understand its procedure, let us consider a dataset of only numerical predictive attributes and a categorical class attribute. Hunt's algorithm proceeds as following:

1. Given the set of data which reaches the current node (starting by the root), see if all instances belong to the same class. If they do, assign this class to the current node and halt the process; if not, see which attribute, when selecting a threshold value, maximizes a given splitting criterion;

2. select such attribute for the current node;

3. generate two children nodes, where the edges contain the condition that an instance must have to follow such path (commonly $<$ and $\geq$ when dealing with numerical attributes, followed by the threshold value: < 5 and ≥ 5, for example);

4. repeat 1 through 3 for each one of the children and so on, until a stopping criterion is met.

The following sections will describe in further details the splitting criteria used to split sets of data into more pure subsets, as well as evaluation methodology to measure how good a decision tree performs when dealing with unknown data.

4.1.2 Splitting Criteria

In order to split sets of data into more pure subsets, a decision-tree induction algorithm may make use of one out of several information theory methods [TSK05, chap. 4, p. 158]. Those methods vary in range of values, methodology and advantages, but all of them describe data in terms of homogeneous (i.e, pure) and heterogeneous (i.e, impure) distributions.

Entropy

Entropy [Sha01] is not a splitting criterion *per se*, but a function which measures impurity in sets used in other splitting criteria. It is defined as

$$H(\mathbf{X}_\pi) = -\sum_{c \in C} p(c|\mathbf{X}_\pi) \log_2 p(c|\mathbf{X}_\pi) \tag{4.1}$$

where C is the set of class values and $\mathbf{X}_\pi$ the set of instances at the current node. A set of homogeneous instances (that is, evenly distributed across all classes) achieves minimum entropy, whereas pure subsets yield maximum entropy. Since Equation 4.1 is not normalized by the number of instances, the upper bound varies according to the number of classes, but the lower bond is at 0.

Information Gain

Information gain [Bar13] aims to measure the decrease in impurity (or, in other words, the gain in information) before and after splitting the data at a given threshold. It is defined as

$$IG(\mathbf{X}_\pi) = H(\mathbf{X}_\pi) - \sum_{\mathbf{X}_\varepsilon \in \mathbf{X}_\pi} \frac{|\mathbf{X}_\varepsilon \in \mathbf{X}_\pi|}{|\mathbf{X}_\pi|} H(\mathbf{X}_\varepsilon) \tag{4.2}$$

where $\mathbf{X}_\varepsilon$ is a subset of $\mathbf{X}_\pi$.

Information Gain Ratio

In order to mitigate the issue presented by Information Gain, Information Gain Ratio [Qui93] was proposed. It compensates the decrease in entropy in multiple partitions by dividing the information gain by the entropy of subset $\mathbf{X}_\pi$ [Bar13]:

$$IGR(\mathbf{X}_\pi) = \frac{IG(\mathbf{X}_\pi)}{SI(\mathbf{X}_\pi)} \tag{4.3}$$

where $SI(\mathbf{X}_\pi)$ is the entropy of subsets generated by splitting $\mathbf{X}_\pi$:

$$SI(\mathbf{X}_\pi) = -\sum_{\mathbf{X}_\varepsilon \in \mathbf{X}_\pi} \frac{|\mathbf{X}_\varepsilon|}{|\mathbf{X}_\pi|} \log_2 \frac{|\mathbf{X}_\varepsilon|}{|\mathbf{X}_\pi|} \tag{4.4}$$

where $\mathbf{X}_\varepsilon$ is a subset of $\mathbf{X}_\pi$.

4.2 Evaluation Criteria

A evaluation criterion aims to capture an aspect of the classifier: be it the capability to correctly predict the class of instances (accuracy), the capability to differentiate between different classes (sensitivity and specificity), etc. Most of these criteria are derived from a confusion matrix (e.g Table 4.1), which depicts the number of correctly classified instances for each class. The interested reader can refer to a list of such criteria in [WF05, chap. 5.7, p. 161].

		Predicted label			
		virginica	setosa	versicolor	$\sum$
True label	virginica	33	10	7	50
	setosa	21	17	12	50
	versicolor	5	24	21	50
	$\sum$	59	51	40	150

Table 4.1 – Confusion matrix for a classifier trained on the Iris dataset [TSK05, chap. 3, p. 98]. Numbers in the main diagonal are correctly classified instances. For each row, the last column $\sum$ shows the real distribution of data among the classes. For each column, the last row $\sum$ shows the predicted classes by the classifier. This classifier has 47.33% of accuracy ($\frac{33+17+21}{150}$). Best viewed in colors.

4.3 Evaluation Methodology

Independently from the category of model used for performing predictions, its quality is assured through a common methodology: based on known data, it must perform well also on unknown data. In a real-world scenario, one would not hide a portion of the data from the model; take for example the prediction of a disease based on its symptoms. It is not interesting to train a model over only 80% of the data and then use it in a hospital. However, when validating the model's quality, this strategy is necessary, since known data hidden from the model for estimating the generalization error. Two of such strategies are holdout and cross-validation.

4.3.1 Holdout

Holdout [TSK05, WF05, chap. 4, p. 186; chap. 5 p. 144] is a technique which splits the whole dataset into two disjoint sets: training and test. The percentage of data which goes to each one of the sets is up to the user to define, but some common values are $\frac{1}{2}$-$\frac{1}{2}$ or $\frac{2}{3}$ for training and $\frac{1}{3}$ for test. Holdout is more commonly used when making several tests is unpractical due to the extent of the dataset. However, it presents several flaws. First, the quality of the model is measured in terms of data available at the training set, making crucial that training and test sets are configured in a way that the distribution of classes between they are similar. Secondly, the amount of data going to training and test sets has a large impact in the predictions: the more data in the training set, the less reliable the test predictions are; the more data in the test, the less reliable is the model, since it is trained on a subsample.

4.3.2 Cross-validation

A more robust strategy is cross-validation [TSK05, WF05, chap. 4, p. 187; chap. 5 p. 149], which splits the dataset into several equal-size subsets: for example, when performing a 10-fold cross-validation procedure, the dataset will be split into 10 folds, each one containing $\frac{1}{10}$ of the data. Each fold is used once for test and the rest of the time for training. As well as it is recommended for holdout, when generating the folds it is important that the distribution of classes follows a similar pattern as the one presented for the whole dataset, preventing that predictions made for a fold to be too different from predictions for the other folds (for example, $\approx 50\%$ of accuracy in one fold and $\approx 95\%$ for the other 2 folds in a three-fold cross-validation).

4.4 Ardennes

Ardennes is our proposed EDA for induction of decision-trees for the task of classification. It deals with numerical predictive attributes with no missing values. There are several ways of dealing with missing data [TSK05], and we leave to the user to find the one which best fits the problem. Ardennes achieves competitive results for 10 datasets, and to the best of our knowledge it is the first EDA for this task. In this section we describe how it was developed, which algorithms are compared to it and the results found for the 10 evaluated datasets. It was submitted and accepted for publication in the 2017 IEEE Congress on Evolutionary Computation, to be held in San Sebastián, Spain, between 5 and 8 of june, 2017 [CBB17].

4.4.1 Individual Encoding

In Ardennes individuals are binary trees of at most H height, which may or may not be complete. They are sampled from a graphical model (GM) which resembles a complete binary tree with H levels, modelled as following. Each one of the GM variables is a probabilistic mass function (PMF) over the attributes (both predictive A and class C) of the dataset. The number of variables at a given level h is 2^{h-1}, and the total number of variables in the GM are $2^{H} - 1$. Since we assume there's no interdependence between variables (i.e, the probability of an attribute being chosen for a given node does not affect neighboring probabilities), the GM does not present edges between them, although we acknowledge that such encoding may be naïve.

We start the probabilities for the predictive attributes uniformly across all variables. The probability of selecting the class attribute, however, is zero from the root to level $H-1$; at H it assumes 100%. Although we adopt this approach, the probability of selecting the class may be increased in further generations by the updating procedure. Figure 4.2 depicts the initial GM.

The sampling of nodes is done in a *on-demand* fashion: the root is sampled first, then its left child, the left child of this child and so on. If the class attribute is sampled, the current node is turned into a leaf and its label is the most frequent class found in X_{π}, the subset of instances reaching that node. If instead a predictive attribute is selected, we perform a binary split on the instances reaching that node. We use information gain ratio as splitting criterion, which is the same criterion used by J48 [Qui93] and is explained in further detail in Section 4.1.2.

The sampling goes on until one of the following leaf-turning conditions are met: (1) The class attribute is sampled; (2) None of the possible threshold values for the sampled attribute provides a gain in information, which occurs when all instances have the same value (a scenario possible due to the stochastic nature of the induction process); or (3) the current node is already pure. Since the probability of sampling the class at level H is always 100%, we can assume that either one of those conditions will be met.

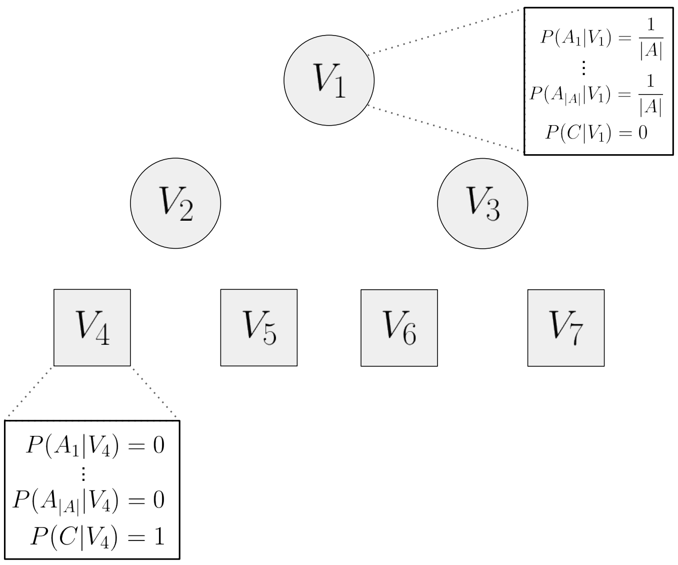


Figure 4.2 – An initial GM with $H = 3$ and 7 variables. Note that the probability of sampling the class attribute C is 100% at the last level and 0% in the other levels.

4.4.2 GM updating process

We employ a lexicographic approach [BBDCF12, BBdC+09] for updating GM's variables. We sort individuals based on three criteria: fitness (descending), tree height (ascending) and number of nodes (ascending). If individuals present the same fitness, then the tree height is used to break the tie; if a tie persists, then the individual with least nodes in its tree is prioritized. After sorting we select the first $|\Phi|$ fittest individuals (with $|\Phi|$ provided by the user) for updating GM's variables and resample the $|S| - |\Phi|$ remaining individuals using the updated GM.

For each variable in the GM, we count how many times its node appeared in the fittest population Φ and group the sampled attributes. We complement the remaining attributes (since not all fittest individuals may present this node in their structure) by sampling attributes from a uniform distribution U over A. Then, we update the variable probability as following:

$$P(V_i) = \frac{\sum_{\phi \in \Phi} (\exists \phi_{V_i} \to 1) \vee (\nexists \phi_{V_i} \to U(A))}{|\Phi|} \tag{4.5}$$

where V_i is the i-th variable in the GM and ϕ_{V_i} the corresponding node in individual ϕ. If a variable is not sampled in all of the fittest population, then by Equation 4.5 its probabilities will be updated to a uniform distribution.

By using a lexicographic approach plus the uniform component we can simulate a exploration-exploitation behavior. In the first generations fitness equality is expected to be small, with the EDA performing an exploration. When individuals start to present similar fitness, the exploratory procedure is replaced by a exploitation one, since smaller individuals are benefited. The EDA ends by refining equal-fitness and equal-sized trees to its more compact form, which may be viewed as a pruning step. There is a chance that this process of exploration-exploitation-pruning may repeat itself several times throughout evolution, as explained in Section 4.4.7.

The evolutionary process is stopped when either the whole population is practically equal (i.e same fitness, tree height and number of nodes) or when a maximum number of G generations is reached, with G provided by the user. From the fittest population of the last generation we then select the individual which presents the best validation accuracy, explained in Section 4.4.3.

4.4.3 Fitness Function

Ardennes receives as input two sets: a training set used for threshold setting, and a validation set used for verifying how well the tree is performing on unseen data. The fitness of an individual *until* the last generation is given by

$$fitness(s_i) = acc_{s_i}(training) \tag{4.6}$$

with s_i as the i-th individual for a given generation g and $acc_{s_i}(training)$ the accuracy for the training set, with $acc = (TP + TN)/(TP + TN + FP + FN)$[1]. Fitness ranges from 0 to 1, with higher values as better ones.

After the evolutionary process is completed, the *quality* of an individual is given by

$$quality(s_i) = \frac{acc_{s_i}(training) + acc_{s_i}(validation)}{2} \tag{4.7}$$

with $acc_{s_i}(validation)$ as the accuracy in the validation set. The individual which maximizes the *quality* index is then deemed to be the solution for the problem.

4.4.4 Complexity Analysis

We analyze Ardennes' complexity as follows. Since all variables are updated independently of the fittest subpopulation configuration, the cost of updating the GM is at most $G \cdot (2^H - 1) \cdot |\Phi|$. Sampling the rest of population at each generation is $(|S| - |\Phi|) \cdot 2^H - 1$, in the worst case of always generating complete trees. Let τ be the cost of calculating thresholds for each node and γ the cost of calculating the fitness function. Hence, the cost of Ardennes is given by

$$G \cdot \left[(\tau \cdot (2^H - 1) \cdot \gamma \cdot (|S| - |\Phi|)) + |\Phi| \cdot (2^H - 1)\right] \tag{4.8}$$

[1] TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative.

4.4.5 Experimental Setup

In this section we describe the datasets, algorithms and criteria used to evaluate Ardennes.

Datasets

We use 10 datasets available at the UCI Machine Learning Repository [Lic13]. All datasets contain only numerical predictive attributes and a categorical class. Those datasets describe a wide variety of scenarios: from characteristics of a collection of flowers to heart diagnosis. The characteristics of the datasets are presented in Table 4.2.

Table 4.2 – Datasets from the UCI Machine Learning Repository [Lic13].

name	# instances	# attributes	# classes
column-2c	310	6	2
column-3c	310	6	3
ionosphere	351	33	2
iris	150	4	3
liver-disorders	345	6	2
sonar	208	60	2
tep-fea	3572	7	3
transfusion	748	4	2
vehicle	846	18	2
wine	178	13	3

4.4.6 Parameters and Baseline Algorithms

For generating Ardennes' results we use $|S| = 200$, $|\Phi| = 100$ (i.e 50%), $G = 100$ and $H = 9$. We do not make any attempt at optimizing those parameters, leaving this task for future work. We perform a stratified 10-fold cross-validation on each of the datasets, using one fold for test, one for validation and 8 for setting thresholds per round of the cross-validation. Since Ardennes is a stochastic algorithm, we run it 10 times for each fold. We compare Ardennes with two distinct approaches, J48 [WF05] and LEGAL-Tree [BBdC$^+$09].

J48 is the Java version of the C4.5 algorithm [Qui93]. It performs deterministic top-down inference of decision-trees, and is available at the Weka Toolkit[2] [WF05]. We run it using its default parameters. Since J48 does not make use of a validation set, it uses both training and validation sets for threshold setting, and because it is a *deterministic* algorithm, we run it only once for each fold.

LEGAL-Tree [BBdC$^+$09] is a genetic algorithm for performing lexicographic induction of decision trees; that is, it can use more than one criterion for evaluating how good solutions are, prioritizing one over another when a tie is found. Since LEGAL-Tree is a GA, a strategy for pre-setting splitting

[2]Available at http://www.cs.waikato.ac.nz/ml/weka/downloading.html. Accessed in 2016-12-06.

thresholds must be defined prior to the evolutionary process. In order to achieve this, for each attribute, the authors divide the training set into 10 smaller subsets and run a decision stump algorithm for each one of the subsets. In this way, the initial population is made of several binary trees of three nodes (one internal and two leaves), which will be combined and perturbed by respectively crossover and mutation operators. The authors use accuracy as fitness function, with tree height (smaller trees are preferred) for breaking ties. LEGAL-Tree also makes use of two distinct sets during its training process (a training set and a validation set), which are the same as the ones used by Ardennes. Since LEGAL-Tree is a stochastic algorithm, we use the same framework from Ardennes to test it – 10 runs for each fold – with the same parameters as the ones proposed by the authors in their work [BBdC+09].

4.4.7 Experimental Results

Evolution Analysis

In order to evaluate the assumptions made in Section 4.4.2 about the exploration-exploitation-pruning capability of Ardennes, we conduct an analysis over a run of Ardennes in the Iris dataset. The progression of probabilities within the GM is showed in Figure 4.3. Colder colors (i.e blue-ish) are closer to a uniform distribution, whereas hotter (i.e red-ish) describe a perturbation. From the first generation (a) all variables are initialized uniformly. By the 9-th generation (b) most of the individuals within the fittest population present the same structure, as denoted by a preponderance of hotter colors for some nodes. This is corroborated by Figure 4.5, where at the 9-th generation most of the individuals present a tree of height 6. However, since the fitness stalls at this point, in the 13-th generation (c) smaller same-fitness trees are prioritized. However, in the last generation (d), possibly another set of predictive attributes are arranged in the same structure of the 9-th generation (b), thus halting the evolutionary process. As expected, the fittest individual from the last generation – showed in Figure 4.4 – presents the same structure as the one present in (d). It achieves 93.33% accuracy in the test fold and 100% accuracy in both training and validation sets, confirming that Ardennes is properly optimizing the inputted data.

By the analysis of Figure 4.5 (b) it is evident that Ardennes is overfitting the training set. For this particular case, however, there is a significant correlation between this set and the test set, so this behavior is actually beneficial to the evolutionary process. Observe in Figure 4.5 (a) that, from the second to the last generation, the mean test accuracy (showed here only for analysis purposes, since Ardennes does not use it during evolution) is 100%. this was confirmed when we verified individuals' test accuracy in the last generation: 196 individuals had 100% in the test set, whereas the rest presented 93.33%. However, this seems misleading, since the reported test accuracy was 93.33%. In order to solve this we selected the individuals which presented maximum fitness in the last generation and show them in Table 4.3. It is evident that, at this point in the evolutionary process, Ardennes doesn't have any other criterion to discern individuals from one another, so the choice for a best individual to report the results becomes random.

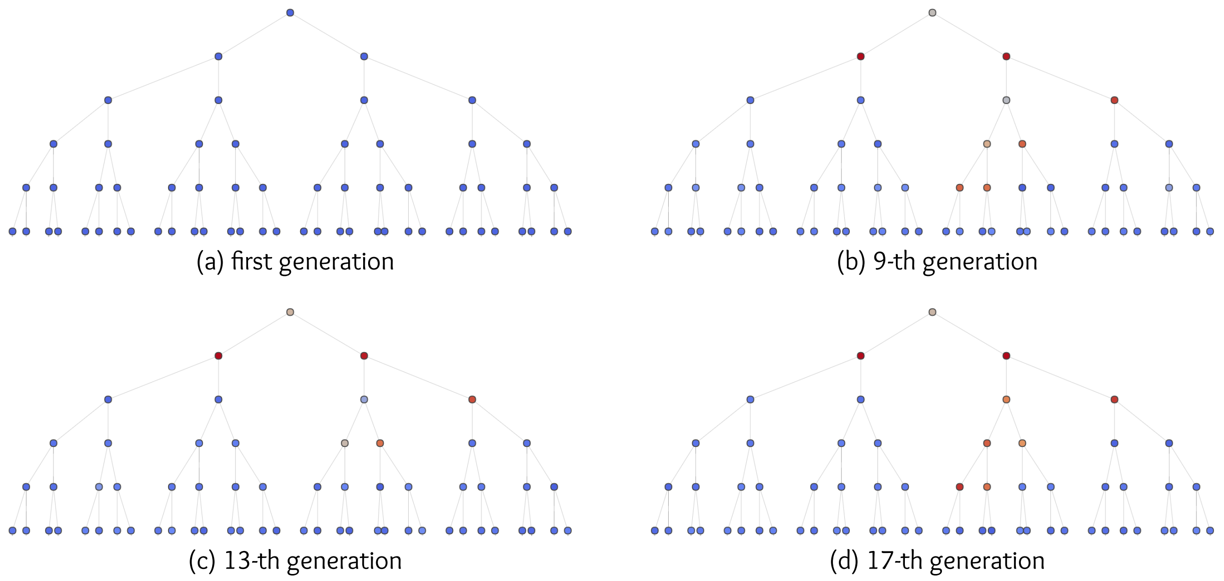


Figure 4.3 – Evolution of GM's probabilities throughout first (a), 9-th (b), 13-th (c) and last (d) generation for a given run of Ardennes in dataset Iris. Hot colors mean that one of the attributes is becoming more frequent than others. Best viewed in colors.

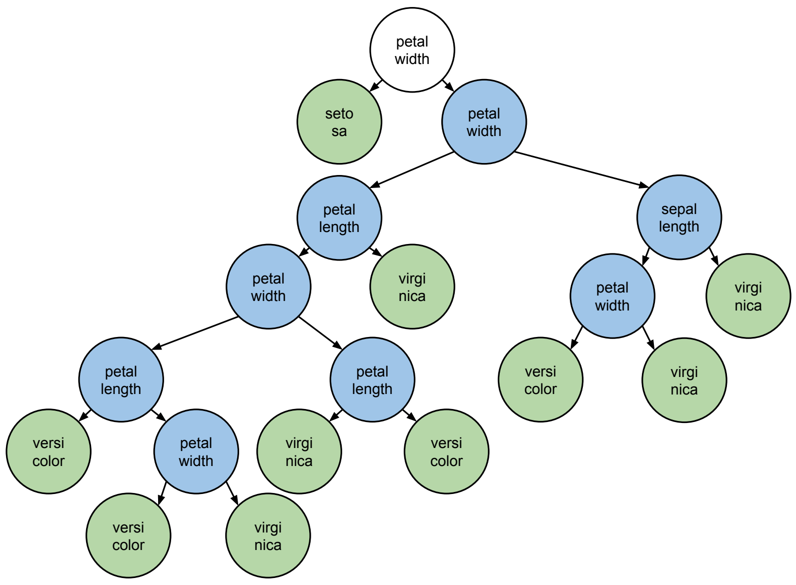


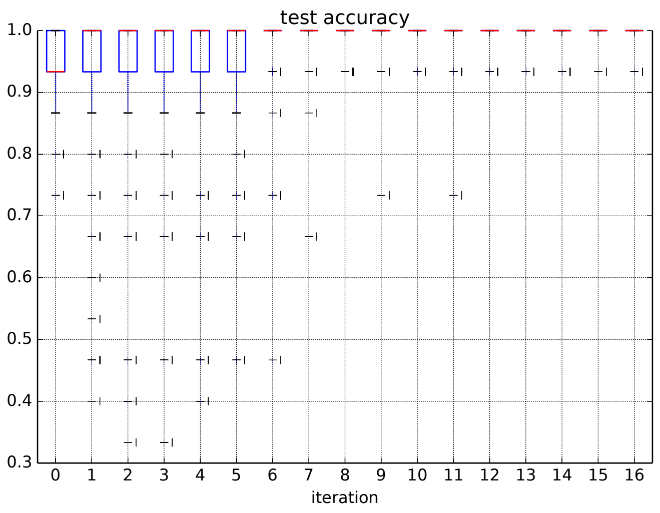
Figure 4.4 – Best individual found in the last generation of Ardennes for a given run in Iris dataset. Note that its structure is in conformity with the one presented in the last generation (d) of Figure 4.3.

Table 4.3 – Best performing individuals from the last iteration of Ardennes for the Iris dataset.

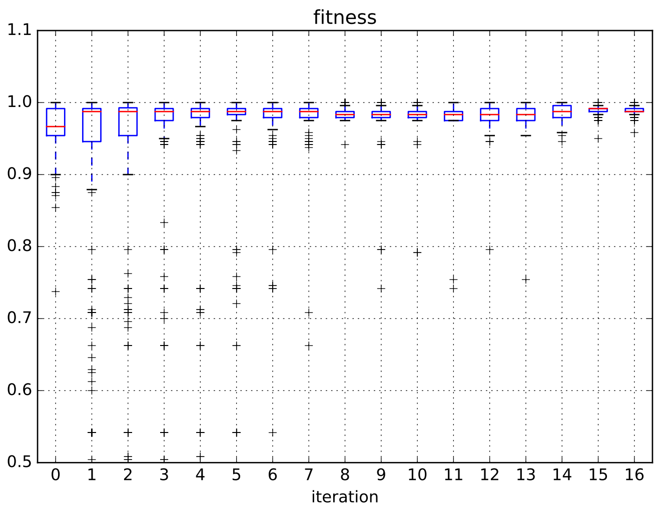
Individual	Fitness	Tree Height	Number of Nodes	Validation Accuracy	Test Accuracy
6	1.0	7	19	1.000	0.933
17	1.0	7	19	1.000	1.000
44	1.0	7	19	1.000	1.000

Results

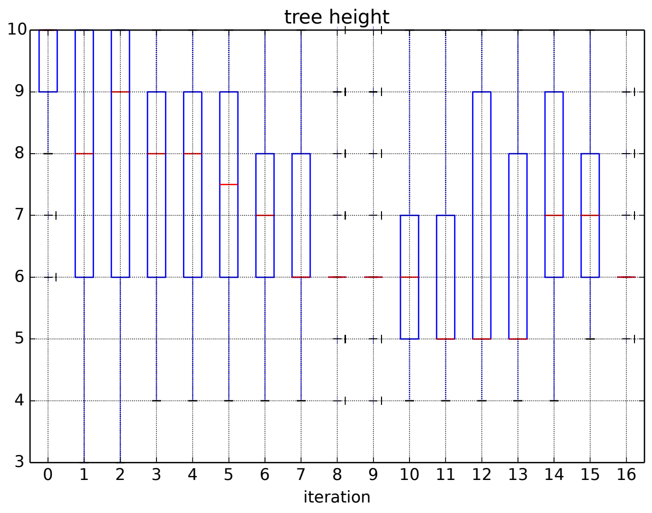
Our first analysis is regarding LEGAL-Tree. Table 4.4 shows that Ardennes is capable of outperforming it in 8 datasets and tying in iris and tep_fea. Such results contribute to our vision that



(a)



(b)



(c)

Figure 4.5 – Test accuracy, fitness and tree height across 17 iterations of Ardennes when optimizing the Iris dataset. Best viewed in colors.

EDAs are better suited for decision-tree induction than genetic algorithms, since both Ardennes and LEGAL-Tree induce it using the same (lexicographic) approach. In order to consolidate this view it is necessary to perform tests with more evolutionary approaches.

Table 4.4 – J48, LEGAL-Tree and Ardennes results for all evaluated algorithms. Best accuracy results are shown underlined and in bold, whereas ties are shown only in bold.

	J48			Legal-tree			Ardennes		
Dataset	Test Accuracy	Tree Height	Nodes	Test accuracy	Tree Height	Nodes	Test Accuracy	Tree Height	Nodes
column_2c	0.82	7.10 ± 1.73	17.60 ± 6.76	0.79 ± 0.05	10.66 ± 0.47	83.98 ± 4.81	**0.83 ± 0.01**	8.59 ± 0.17	28.10 ± 1.77
column_3c	0.80	7.70 ± 0.95	22.60 ± 5.64	0.79 ± 0.03	11.25 ± 0.54	86.18 ± 3.8	**0.82 ± 0.01**	8.66 ± 0.15	40.54 ± 3.28
ionosphere	0.87	9.70 ± 1.16	27.00 ± 3.69	0.88 ± 0.04	11.54 ± 0.79	55.36 ± 5.36	**0.92 ± 0.01**	7.93 ± 0.34	18.42 ± 1.06
iris	**0.95**	4.70 ± 0.48	8.40 ± 0.92	**0.95 ± 0.03**	4.71 ± 0.44	13.7 ± 1.84	**0.95 ± 0.01**	5.14 ± 0.32	10.92 ± 1.34
liver-disorders	**0.63**	9.60 ± 0.97	44.40 ± 8.58	0.6 ± 0.03	12.91 ± 0.47	190.74 ± 5.45	0.62 ± 0.02	8.98 ± 0.04	39.00 ± 1.61
sonar	**0.74**	8.00 ± 0.67	28.00 ± 3.00	0.67 ± 0.05	9.76 ± 0.28	82.78 ± 3.36	0.73 ± 0.02	7.79 ± 0.17	34.60 ± 1.64
tep_fea	**0.65**	4.40 ± 0.97	7.80 ± 1.83	**0.65 ± 0.03**	2.95 ± 0.48	7.04 ± 1.11	**0.65 ± 0.00**	2.00 ± 0.00	3.00 ± 0.00
transfusion	**0.78**	6.70 ± 2.16	12.80 ± 4.51	0.75 ± 0.03	12.32 ± 1.13	138.28 ± 26.72	**0.78 ± 0.01**	8.21 ± 0.32	20.64 ± 3.08
vehicle	**0.74**	14.60 ± 1.78	137.80 ± 26.35	0.66 ± 0.02	17.14 ± 0.67	422.8 ± 11.29	0.73 ± 0.01	8.85 ± 0.07	67.92 ± 2.64
wine	0.91	4.10 ± 0.32	9.80 ± 0.98	0.93 ± 0.04	5.48 ± 0.54	19.98 ± 2.61	**0.95 ± 0.01**	4.27 ± 0.24	8.96 ± 0.64
Average rank	1.85	**1.6**	1.6	2.6	2.8	2.9	**1.55**	**1.6**	**1.5**

We continue by comparing Ardennes to J48. From the results we can assume that it is the strongest baseline, since Ardennes is capable of outperforming it in 4 datasets, tying in another 3 and losing in 3 within a margin of 1%. However, in two of the losing datasets (vehicle and liver-disorders) Ardennes is producing smaller trees than J48, which may be viewed as an exchange between accuracy and comprehensibility. Also note that for some of the winning datasets (ionosphere and wine) Ardennes presents an advantage up to 5%. This can be viewed as an evidence that Ardennes performs an effective global-search procedure in a solution-space larger than the one presented by J48.

4.4.8 Related Work

Deterministic top-down inference is largely one of the most popular approaches for evolving decision trees [BdCF15]. Under Hunt's algorithm [TSK05] one seeks to maximize purity of generate nodes by splitting instances into purer subsets. Although maximizing purity is a desirable feature of a decision tree, it presents two flaws. First, partitioning towards purer subsets is a greedy heuristic, and as such performs a local optimization rather than a global one [BBdC+09]. Secondly, it may lead to overfitting, since iteratively partitioning the training set reduces the significance of subsets in relation to the complete distribution. For instance, leaves with only one object achieve maximum purity for any measure. C4.5, a popular decision-tree induction algorithm proposed by Quinlan [Qui93], uses tree pruning to reduce complexity and avoid overfitting.

An active field for evolving decision-trees is evolutionary computation [BBDCF12], most notably by the use of genetic algorithms. The behavior of a genetic algorithms (GA) may be summarized as follows: by randomly sampling S individuals in the first iteration, the GA will rely on operators such as mutation and crossover in order to generate novel individuals in following iterations. Although

EDAs and GAs share a fair amount of similarities in their evolutionary processes, their differences are evident in the way that novel individuals are generated. Whereas EDAs use explicit evolutionary operators [BD97, HP11], which are easy to keep track and less prone to yield irrelevant individuals [DBIJV96, PM99], GAs use crossover and mutation. Hence, GAs need more iterations in order to overcome the random nature of its operators [BD97].

To the best of our knowledge, the use of EDAs for decision-tree induction is a completely novel research field. However, we take the inspiration for using a GM which resembles a tree from the work of Salustowicz and Schmidhuber [SS97], which presents a similar implementation but different application. In their work, the authors sample programs from a univariate distribution. Each node at the GM encodes functions such as $\sin$, $\cos$, $\div$, $\times$ and operands. The probabilities at each node are initialized uniformly, accordingly to the need of a given instruction at a given position in the individual program. Fitness is measured in runtime (with lower values being better), since a valid program must necessarily solve a problem (i.e, approximate a function). The authors do not initialize a complete n-ary tree of H levels; it has its size dynamically modified throughout the evolutionary process. The authors note however that pruning the GM is required in order to reduce memory consumption.

4.4.9 Final remarks

This section presented Ardennes, an Estimation of Distribution Algorithm for performing top-down induction of decision trees. Ardennes was submitted and accepted for publication in the 2017 IEEE Congress on Evolutionary Computation, to be held in San Sebastián, Spain, between 5 and 8 of june, 2017 [CBB17]. Ardennes is capable of overcoming LEGAL-Tree, a genetic algorithm, which employs the same lexicographic strategy towards induction. The results shown here temporally assure our view that EDAs are better suited for this task. In order to further investigate this we seek to test Ardennes against more evolutionary approaches in a wider set of datasets. With such testing it will be possible to determine whether EDAs are overall better than other approaches or if it performs well only in a niche of applications.

Ardennes presents competitive results regarding J48, a traditional, deterministic top-down induction algorithm. However, the work presented here is the first step towards developing EDAs for this task. By tackling the the issue presented in Section 4.4.7 we believe that Ardennes can have its performance further improved. Two possibilities for solving it would be to use more criteria as tie breaking – F1 measure, precision, etc – or employing a Pareto approach [HP11]. As for implementation improvements, Ardennes is well suited for parallelization, since several steps are independent from one another. In fact we employ parallelism for verifying the quality of thresholds when splitting predictive attributes. However, several other aspects may be paralellized, such as (1) sampling of individuals, (2) sampling of nodes within an individual, (3) initialization and updating of weights in the GM (since it is a univariate) and possibly more.

5. CONCLUSIONS

Typically, machine learning algorithms rely on a heuristic to perform their tasks, be it clustering objects into groups or evolving a model to perform classification. Although heuristics significantly speed-up the optimization process and may yield good results, employing a local-search strategy does not guarantee a good performance in all scenarios, as outlined by David Wolpert in his "No Free Lunch" Theorem [Mur12]. It also comes at the cost of minimally knowing about the data being processed: K-means (Section 3.2.1) requires a pre-defined number of groups to perform clustering; DBSCAN (Section 3.2.2) expects the user to input the minimum density in which a group may occur; Hunt's algorithm (Section 4.1.1) successively splits the data space in order to classify instances, thus reducing the robustness of generated models and requiring anti-overfitting measures. The use of EDAs, on the other hand, provides a robust strategy to perform both clustering and decision-tree induction with little to no prior knowledge, since they employ an efficient global-search strategy: Clus-EDA (Section 3.3) requires an initial value for the number of clusters, but its not a rigid limit on the number of groups and can vary throughout the clustering process; PASCAL (Section 3.4) does not require any prior knowledge, making use only of the hyper-parameters related to the evolutionary process; and Ardennes (Section 4.4) only requires the maximum number of levels in which a decision-tree can span, which may be viewed as a compactness parameter.

In order to achieve good results, the design of an EDA is an important task to be developed. Take PASCAL for example: its success in performing clustering relies on the Density-Based Clustering Validation (Section 3.1.4), an internal criterion for validating cluster quality which presents a good correspondence [MJC+14] to the Adjusted Rand Index, which is an external validity criterion. Another key aspect for achieving good results is designing the probabilistic graphical model in a way that it is capable of capturing the underlying dependence between variables. Since in our problems the structure of the GM dictates the encoding of individuals, we consider that the smart design of GMs is closely related to the smart design of individuals as well. None of our algorithms (Clus-EDA, PASCAL nor Ardennes) considers interdependence between variables; however, by the combination of design, fitness function, and updating procedures, they are capable of finding satisfactory results.

Finally, updating probabilities in a way that it is more likely to generate good solutions in the following generations allows EDAs to efficiently navigate through the search space. All EDAs presented in this work use a simple updating process, where values from the fittest individuals from a given generation are propagated to the following generations. Once the probabilities are updated, the amount of individuals that will replace the previous generation also plays an important role: Clus-EDA replaces the whole population in the following generations; PASCAL replaces only half through its median procedure (individuals with fitness above or equal to the median are preserved, whereas individuals with fitness below are replaced). Finally, Ardennes employs a similar strategy to PASCAL, preserving individuals with fitness above a user-defined percentile and replacing individuals with fitness below or equal to it.

5.1 Contributions

This work presents Estimation of Distribution Algorithms for the tasks of decision-tree induction and clustering. The advantages of using EDAs over other evolutionary algorithms are two-fold. First, evolutionary analysis is made clearer due to the use of the probabilistic graphical model as a sampling mechanism, allowing one to look at the variables' probabilities and analyze how variables interact with each other, how much they are affected by the updating process, for each and every generation of the EDA. This wide analysis cannot be done with genetic algorithms, for example, due to the use of the crossover operator, whose stochasticy makes them hard to analyze. Second, due to the use of probabilistic graphical models, good solutions are much easier to be generated and preserved in the following generations. This claim is backed by the superior results presented by EDAs in clustering and competitive ones in decision-tree induction for classification. It is important to note that EDAs for both clustering and decision-tree induction is a relatively new field, with few studies exploring its capabilities; we believe that by employing some of the mechanisms described in the next section this area may advance further.

5.2 Limitations and Future Work

Although EDAs achieve a good performance for both clustering and decision-tree induction, they share some disadvantages with other evolutionary algorithms for the same tasks, in relation to traditional methods. First PASCAL is the only which does not use any parameter directly related to the problem being optimized. Additionally, all EDAs require hyper-parameters to be set, such as population size and number of iterations, and finding a good set of parameters may add yet another layer of difficulty.

This work does not present a runtime analysis, since we believe this is not a fair method of comparing methods given the substantial differences between traditional local-search algorithms, GAs, and EDAs methods. However, we acknowledge that evolutionary procedures are much slower than traditional ones, since they do not employ a greedy search procedure. In a wider sense, complexity of EDAs are mainly inherited from the fitness function and sampling and updating procedures of graphical models. In our work, since all EDAs use univariate probabilistic graphical models, their runtime performance is more affected by their fitness functions.

As a side note to one seeking to implement EDAs for clustering or decision-tree induction, most of the code of both Ardennes and PASCAL were ported to a parallel architecture (CUDA [NVI14]) in order to take advantage from several parallel procedures executed by an EDA: individual sampling, fitness evaluation, and probabilistic graphical model update (when using univariate models).

We also employ one of the simplest models, the univariate marginal distribution, to develop our EDAs for both tasks. However, interdependence between variables may be inferred in future EDAs.

By designing the interaction between variables, it would be possible to capture more complex solution spaces. We also only employ discrete variables in our GMs.

With regard to specific improvements to each of the EDAs, they can be found in their respective Sections: 4.4.9 for Ardennes; 3.3.5 for Clus-EDA; and 3.4.7 for PASCAL. In a wider sense, EDAs for clustering and decision-tree induction may benefit from the use of more sophisticated probabilistic graphical models, which assume dependence between variables, be it in a static way (variables interactions are hand-made prior to EDA execution) or dynamically inferred, using a schema similar to the one presented in Section 2.1.1. Furthermore, in this work we only use discrete variables; it is yet to be researched whether continuous variables may provide an improvement in THE EDAs' performance for those tasks.

BIBLIOGRAPHY

[ACH06] Alves, V. S.; Campello, R. J.; Hruschka, E. R. "Towards a fast evolutionary algorithm for clustering". In: IEEE Congress on Evolutionary Computation, 2006, pp. 1776–1783.

[BA97] Breslow, L. A.; Aha, D. W. "Simplifying decision trees: A survey", *The Knowledge Engineering Review*, vol. 12–1, 1997, pp. 1–40.

[Bar13] Barros, R. C. "On the automatic design of decision-tree induction algorithms", Ph.D. Thesis, University of São Paulo, 2013, 204p.

[BBdC$^+$09] Basgalupp, M. P.; Barros, R. C.; de Carvalho, A. C.; Freitas, A. A.; Ruiz, D. D. "Legal-tree: a lexicographic multi-objective genetic algorithm for decision tree induction". In: ACM Symposium on Applied Computing, 2009, pp. 1085–1090.

[BBDCF12] Barros, R. C.; Basgalupp, M. P.; De Carvalho, A. C.; Freitas, A. "A survey of evolutionary algorithms for decision-tree induction", *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 42–3, 2012, pp. 291–312.

[BCFdC13] Barros, R. C.; Cerri, R.; Freitas, A. A.; de Carvalho, A. C. "Probabilistic clustering for hierarchical multi-label classification of protein functions". In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2013, pp. 385–400.

[BD97] Baluja, S.; Davies, S. "Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space", Technical Report, Defense Technical Information Center, 1997, 20p.

[BdCF15] Barros, R. C.; de Carvalho, A. C.; Freitas, A. A. "Automatic Design of Decision-Tree Induction Algorithms". Springer, 2015, 176p.

[BYL$^+$15] Bouguettaya, A.; Yu, Q.; Liu, X.; Zhou, X.; Song, A. "Efficient agglomerative hierarchical clustering", *Expert Systems with Applications*, vol. 42–5, 2015, pp. 2785–2797.

[CB16] Cagnini, H. E. L.; Barros, R. C. "PASCAL: an EDA for parameterless shape-independent clustering". In: IEEE Congress on Evolutionary Computation, 2016, pp. 3433–3440.

[CBB17] Cagnini, H. E. L.; Barros, R. C.; Basgalupp, M. "Estimation of distribution algorithms for decision-tree induction". In: IEEE Congress on Evolutionary Computation, 2017, pp. 1–8.

[CBQB16] Cagnini, H. E. L.; Barros, R. C.; Quevedo, C. V.; Basgalupp, M. P. "Medoid-based data clustering with estimation of distribution algorithms". In: ACM Symposium on Applied Computing, 2016, pp. 112–115.

[CLRS09] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. “Introduction to algorithms”. MIT Press, 2009, 3 ed., 944p.

[CTC+06] Chuang, K.-S.; Tzeng, H.-L.; Chen, S.; Wu, J.; Chen, T.-J. “Fuzzy c-means clustering with spatial information for image segmentation”, *Computerized Medical Imaging and Graphics*, vol. 30–1, 2006, pp. 9–15.

[DB79] Davies, D. L.; Bouldin, D. W. “A cluster separation measure”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1–2, 1979, pp. 224–227.

[DBIJV96] De Bonet, J. S.; Isbell Jr, C. L.; Viola, P. “Mimic: finding optima by estimating probability densities”. In: International Conference on Neural Information Processing Systems, 1996, pp. 424–430.

[Dun74] Dunn, J. C. “Well-separated clusters and optimal fuzzy partitions”, *Journal of Cybernetics*, vol. 4–1, 1974, pp. 95–104.

[EKS+96] Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.; et al.. “A density-based algorithm for discovering clusters in large spatial databases with noise”. In: International Conference on Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[FD07] Frey, B. J.; Dueck, D. “Clustering by passing messages between data points”, *science*, vol. 315–5814, 2007, pp. 972–976.

[For65] Forgy, E. W. “Cluster analysis of multivariate data: efficiency versus interpretability of classifications”, *Biometrics*, vol. 21–3, 1965, pp. 768–769.

[FV06] Fränti, P.; Virmajoki, O. “Iterative shrinking method for clustering problems”, *Pattern Recognition*, vol. 39–5, 2006, pp. 761–775.

[GBC16] Goodfellow, I.; Bengio, Y.; Courville, A. “Deep Learning”. MIT Press, 2016, 800p.

[HA85] Hubert, L.; Arabie, P. “Comparing partitions”, *Journal of classification*, vol. 2–1, 1985, pp. 193–218.

[HCdC06] Hruschka, E. R.; Campello, R. J. G. B.; de Castro, L. N. “Evolving clusters in gene-expression data”, *Information Sciences*, vol. 176–13, 2006, pp. 1898–1927.

[HCFDC09] Hruschka, E. R.; Campello, R. J. G. B.; Freitas, A. A.; De Carvalho, A. C. P. L. F. “A survey of evolutionary algorithms for clustering”, *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 39–2, 2009, pp. 133–155.

[HdCC04] Hruschka, E. R.; de Castro, L. N.; Campello, R. J. “Evolutionary algorithms for clustering gene-expression data”. In: IEEE International Conference on Data Mining, 2004, pp. 403–406.

[HMS66] Hunt, E. B.; Marin, J.; Stone, P. J. “Experiments in induction”. Academic Press, 1966, 247p.

[HP11] Hauschild, M.; Pelikan, M. “An introduction and survey of estimation of distribution algorithms”, *Swarm and Evolutionary Computation*, vol. 1–3, 2011, pp. 111–128.

[JD88] Jain, A. K.; Dubes, R. C. “Algorithms for Clustering Data”. Prentice Hall, 1988, 304p.

[JWEW15] John W. Eaton, David Bateman, S. H.; Wehbring, R. “Gnu octave version 4.0.0 manual: a high-level interactive language for numerical computations”. Available at http://www.gnu.org/software/octave/doc/interpreter, 2016-11-25.

[Koo16] Kools, J. “6 Functions for generating artificial datasets”. Available at https://goo.gl/g31ksb, 2016-01-12.

[KR90] Kaufman, L.; Rousseeuw, P. J. “Finding Groups in Data: An Introduction to Cluster Analysis”. John Willey & Sons, 1990, 342p.

[Lic13] Lichman, M. “UCI machine learning repository”. Available at http://archive.ics.uci.edu/ml, 2017-03-28.

[Llo06] Lloyd, S. “Least squares quantization in pcm”, *IEEE Transactions on Information Theory*, vol. 28–2, 2006, pp. 129–137.

[M$^+$67] MacQueen, J.; et al.. “Some methods for classification and analysis of multivariate observations”. In: Berkeley Symposium on mathematical statistics and probability, 1967, pp. 281–297.

[MC12] Murtagh, F.; Contreras, P. “Algorithms for hierarchical clustering: an overview”, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2–1, 2012, pp. 86–97.

[Mit97] Mitchell, T. M. “Machine learning”. McGraw-Hill, 1997, 421p.

[MJC$^+$14] Moulavi, D.; Jaskowiak, P. A.; Campello, R.; Zimek, A.; Sander, J. “Density-based clustering validation”. In: SIAM International Conference on Data Mining, 2014, pp. 839–847.

[MP96] Mühlenbein, H.; Paass, G. “From recombination of genes to the estimation of distributions I. Binary parameters”. In: International Conference on Parallel Problem Solving from Nature, 1996, pp. 178–187.

[Mur12] Murphy, K. P. “Machine learning: a probabilistic perspective”. MIT Press, 2012, 1049p.

[NCHdC11] Naldi, M. C.; Campello, R. J. G. B.; Hruschka, E. R.; de Carvalho, A. C. P. L. F. “Efficiency issues of evolutionary k-means”, *Applied Soft Computing Journal*, vol. 11–2, 2011, pp. 1938–1952.

[NVI14] NVIDIA. “CUDA Toolkit Documentation”. Available at http://docs.nvidia.com/cuda/, 2017-03-28.

[PM99] Pelikan, M.; Mühlenbein, H. “The bivariate marginal distribution algorithm”. In: *Advances in Soft Computing*, Roy, R.; Furuhashi, T.; Chawdhry, P. K. (Editors), Springer, 1999, chap. 10, pp. 521–535.

[PVG$^+$11] 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*, vol. 12–10, 2011, pp. 2825–2830.

[Qui93] Quinlan, J. R. “C4. 5: Programming for machine learning”. Morgan Kauffmann, 1993, 38p.

[Rou87] Rousseeuw, P. J. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”, *Journal of Computational and Applied Mathematics*, vol. 20–1, 1987, pp. 53–65.

[SBL11] Santana, R.; Bielza, C.; Larrañaga, P. “Affinity propagation enhanced by estimation of distribution algorithms”. In: Annual conference on Genetic and evolutionary computation, 2011, pp. 331–338.

[Sha01] Shannon, C. E. “A mathematical theory of communication”, *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5–1, 2001, pp. 3–55.

[Sne57] Sneath, P. H. “The application of computers to taxonomy”, *Microbiology*, vol. 17–1, 1957, pp. 201–226.

[Sør48] Sørensen, T. “A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons”, *Biologiske Skrifter*, vol. 5–4, 1948, pp. 1–34.

[SS97] Salustowicz, R.; Schmidhuber, J. “Probabilistic incremental program evolution”, *Evolutionary Computation*, vol. 5–2, 1997, pp. 123–141.

[TSK05] Tan, P.-N.; Steinbach, M.; Kumar, V. “Introduction to Data Mining”. Addison Wesley, 2005, 769p.

[Vap99] Vapnik, V. “An overview of statistical learning theory”, *IEEE Transactions on Neural Networks*, vol. 10–5, 1999, pp. 988–999.

[VC95] Vapnik, V.; Cortes, C. “Support vector networks”, *Machine Learning*, vol. 20–3, 1995, pp. 273–297.

[VCH10] Vendramin, L.; Campello, R. J.; Hruschka, E. R. “Relative clustering validity criteria: A comparative overview”, *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 3–4, 2010, pp. 209–235.

[WF05] Witten, I. H.; Frank, E. “Data Mining: Practical machine learning tools and techniques”. Morgan Kaufmann, 2005, 664p.

[XW05] Xu, R.; Wunsch, D. “Survey of clustering algorithms”, *IEEE Transactions on Neural Networks*, vol. 16–3, 2005, pp. 645–678.

[YMB03] Yeung, K. Y.; Medvedovic, M.; Bumgarner, R. E. “Clustering gene-expression data with repeated measurements”, *Genome Biology*, vol. 4–5, 2003, pp. 1–17.

[ZGH11] Zhou, Y.; Grygorash, O.; Hain, T. F. “Clustering with minimum spanning trees”, *International Journal on Artificial Intelligence Tools*, vol. 20–01, 2011, pp. 139–177.