Department of Digital Systems
University of Piraeus

Master of Science in Information Systems and Services
Area of Study: “Big Data and Analytics”

Unsupervised AutoML

A STUDY ON AUTOMATED MACHINE LEARNING IN THE CONTEXT OF CLUSTERING

MASTER THESIS

BY
POULAKIS GIANNIS

SUPERVISOR: CHRISTOS DOULKERIDIS

Abstract

The modelling of end-to-end Machine Learning processes and methods is not only computationally intensive, but also requires expertise in Data Science and often domain knowledge of the problem. To overcome this adversity, a relatively new field of research has emerged called Automated Machine Learning (AutoML). The main focus of the domain is to discover an automated way to build Machine Learning pipelines given a Machine Learning task and an input data set. While all AutoML systems currently focus on the task of supervised learning, unsupervised learning remains an unexplored and unsolved problem. This thesis aims to provide solutions for automating Machine Learning specifically for the case of unsupervised learning (clustering), in a domain-agnostic manner. This is achieved through a combination of state-of-the-art processes based on Meta Learning for Algorithm Selection and Bayesian Optimization for hyperparameter tuning. Experimentation results on real life datasets provide enough evidence that clustering is a process that can be fully automated.

Table of Contents

Abstract 1
Chapter 1 – Automated Machine Learning 4
1.1 Introduction 4
1.2 Problem Statement 5
1.2.1 Clustering 5
1.2.2 Algorithm Selection 6
1.2.3 Hyperparameter Optimization (HPO) 6
1.2.4 Joint Problem Definition 7
1.2.3 Organization of this Thesis 8
Chapter 2 – Theoretical background 9
2.1 Meta- Learning 9
2.2 Bayesian Optimization 11
Chapter 3 - Related Work 16
3.1 AutoML Systems 16
3.1.1 Auto-Weka 16
3.1.2 Auto-Sklearn 17
3.1.3 Auto-Net 18
3.1.4 TPOT 20
3.1.5 Google Vizier 20
3.1.6 Comparison of AutoML Systems 21
3.2 AutoML in Clustering 23
Chapter 4 - Architecture 26
4.1 Design Specifications 26
4.2 Architecture 26
4.2.1 Training Phase – Analytics Repository Creation Process 27
4.2.2 Selection Phase – “In Action” 30
Chapter 5 - Experimentation 32
5.1 Experimentation Setup 32
5.2 Random Search in Analytics Repository 37
5.3 Cluster Validation Indices 39
5.3.1 Empirical Evidence on the Generalization of Cluster Validation Indices 40

5.3.2 Meta Learning for CVI Selection 41
5.4 Evaluation of Algorithm Selection 42
5.4.1 Prediction of the best Performing Algorithm 43
5.4.2 Algorithm Selection with the use of Novel Meta Features 47
5.5 Evaluation of Hyperparameter Tuning 50
Chapter 6 52
6.1 Proof of Concept 52
6.2 Conclusions 55
6.3 Future Work, Open Challenges 56
References 58

Chapter 1 – Automated Machine Learning

1.1 Introduction

Machine Learning (ML) has seen rapid advancement through the last decade in terms of research and applications thus enabling breakthroughs in a variety of domains. Especially deep neural networks led to advancements in Computer Vision, Natural Language and Speech Processing that consequently enabled the production or improvement of applications such as Autonomous driving, chatbots and customer segmentation analysis, which are very indicative examples of the impact of Machine Learning. Following the many successes of Machine Learning, new challenges emerge, shifting the focus of research on how to use all the available tools that have been developed effectively and collectively.

The constant development of new Machine Learning methods, from data preprocessing to multiple neural network architectures, has created a powerful toolkit for the Data scientist to use in order to solve a large variety of problems encountered. As every method comes with a form of parameterization, such as the number of dimensions to reduce a data set to via dimensionality reduction techniques or the input parameters of an algorithm, the creation of an ML pipeline becomes more difficult even for Data Science experts. The configuration settings available are numerous and selection of the best performing one via manual search in a trial-and-error manner is ineffective due to the time and computational resources required for the training of complex Machine Learning algorithms with combination of large volume Datasets. One of the most representative examples of computational complexity in modern ML models, as of execution time, is that of the ResNet Convolutional Neural Network that was trained on 50.000 images of 32x32 size for settings up to 1202 layers for several days. [1].

In addition, many Machine Learning models tend to be very sensitive to parameterization leading to certain difficulties when comparing efficiency among them. For two scientific studies that include such models to be fairly compared, parameters should receive the same level of tuning at hand for results to be conclusive. Otherwise, the results may be misleading, not revealing the potential of a model because of the difficulty in finding a good parameterization. Towards this goal, a concrete way of parameter tuning independent of domain and architectural choices is to provide a basis for studies to be compared on the same level and add reproducibility over manual searching.

Thus, Machine Learning is moving to a point that it requires a lot of expertise to be applied effectively, even for simpler tasks such as tuning known algorithms, leading to the growing demand for off-the-shelf solutions from non-experts. Ease of use and effective application of Machine Learning from non-experts has the potential to speed up research in other fields, such as psychology, biology, etc., and allow for small scale business to stay competent by improving performance. In short, off-the-shelf Machine Learning is required to enable the collective use of Machine Learning, which in turn will allow for rapid advancements in any data-driven domain.

To accommodate for the aforementioned challenges, the field of study called Automated Machine Learning (AutoML) has emerged. The term refers to the automated construction of ML pipelines given a specific task, while optimizing performance. AutoML aims to alleviate time-consuming procedures, provide off-the-shelf solutions to enable Machine Learning to non-experts, improve performance of algorithms by tailoring them to the problem at hand and provide a better basis for comparing scientific studies by providing a fair amount of tuning among methods of researches. When facing the problem of building Machine Learning pipelines, two major problems arise, namely Algorithm Selection and Hyperparameter Optimization. Although both have been systematically studied individually, AutoML tackles both of them in an intertwined manner.

The AutoML rise in popularity led to the creation of various frameworks that mostly focus on supervised problems. This is generally due to the lack of information in unsupervised problems, to be used for validation purposes often referred to as “ground truth”, such as the true clusters of a dataset’s instances. As a result, no definitive way exists of labeling a clustering as right, leading to obscure objective functions to optimize that rely on user’s definition of good clustering. This thesis aims to tackle some of the problems already mentioned, via an AutoML solution specifically for the case of clustering, dubbed and referred to as *AutoClust.*

1.2 Problem Statement

For an AutoML system to be functional the solution to be provided to the user for a problem instance is crucial to include two things: a Machine Learning Algorithm along with a set of values for its input parameters that are expected to perform well. How to select this solution can be treated as two individual but correlated problems, Algorithm Selection and Hyperparameter Tuning or in an intertwined manner. To gain a better understanding, this chapter provides the necessary definitions to these problems as defined along related research. In addition, the definition of clustering is provided as it is the focus of this thesis and one of the most common Machine Learning practices for exploratory analysis.

1.2.1 Clustering

Machine Learning can be categorized to supervised and unsupervised learning, based on whether a model is trained to learn mappings to already existing labels, often referred to as *ground truth,* on data instances or to explore structural patterns. Applications of unsupervised learning include anomaly detection, association mining, dimensionality reduction and clustering *(also called exploratory data analysis),* the latter being the main focus of this study. Clustering refers to the procedure of splitting a group of objects into homogenous subgroups on the basis of an often subjectively chosen measure of similarity, such that

the similarity between objects within a subgroup is larger than the similarity between objects belonging to different subgroups

Clustering can be split into *hard partitioning* or *fuzzy partitioning.* While the first term refers to the case where every object is exclusively associated with a single cluster the latter considers that objects belong to all clusters with a degree of membership that varies from 0 to 1 [2].The problem of hard partitioning is defined as follows: Given a set of input instances of a Dataset X = $\{x_1, \dots\dots, x_j, \dots\dots, X_N\}$, where $x_j = (x_{j1}, x_{j2}, \dots\dots, x_{jd}) \in R^d$, with each measure x_{ji} called a feature (attribute, dimension, or variable):

1. Hard partitional clustering attempts to seek a K-partition of X, C = $\{\{C_1, \dots\dots, C_K\}$ $(K \leq N)$, such that
    a. $C_i \neq \emptyset, i = 1, \dots, K;$
    b. $U_{i=1}^{K} C_i = X;$
    c. $C_i \cap C_j = \emptyset, i, j = 1, \dots, K \; and \; i \neq j.$

1.2.2 Algorithm Selection

It has been long studied that for a plethora of computational problems with several high - performance algorithms available, no single one performs best in all problem instances. In contrast different algorithms perform best on different types of problem instances [3].This leads to the problem of selecting the appropriate algorithm for a given task, *Algorithm Selection*. The topic of Algorithm selection arises in a wide variety of situations and dates as far as 1976 [4]. Algorithm Selection has core applications in the context of Machine Learning where for a given task, Classification for example, several algorithms exist, and their performance varies depending on the characteristics of the problem instance. Choosing the appropriate algorithm for a problem, can be quite challenging but is crucial for the quality of results.

The problem of Algorithm Selection can be defined as follows: Given a set I of instances of a problem P , a set $A = \{A_1 \dots\dots A_n\}$ of candidate algorithms for P and a metric $m : A \times I \rightarrow R$ that measures the performance of any algorithm $A_j \in A$ on instance set I, construct a selector S that maps any problem instance $i \in I$ to an algorithm $S(i) \in A$ such that the overall performance of S on I is optimal according to metric m .

1.2.3 Hyperparameter Optimization (HPO)

Every Machine Learning system has several hyperparameters to be specified, the choice of which has a great impact on performance. Algorithms are tied to a set of hyperparameters, being integers, binary, real valued or in some cases conditional, that may greatly affect results such as the number of hidden layers in a Deep Neural Network. Consequently, optimizing the set of values for the input parameters of an

algorithm is the main task of any AutoML System and is tied to the algorithm selection problem in that context. Several use cases apply to HPO as it reduces the human effort needed for applying Machine Learning, improves the performance of ML algorithms and improves the reproducibility and fairness of scientific studies [5].

HPO dates as back to the 1990s and has been proven to improve over default settings for hyperparameters suggested by known frameworks. The problem of hyperparameter optimization is defined as follows: Let A denote the ML algorithm with N hyperparameters. We denote the domain of $n-th$ hyperparameter by Λ_n and the overall hyperparameter configuration space as $\Lambda = \Lambda_1 \times \Lambda_2 \ldots\ldots \times \Lambda_N$. A vector of hyperparameters is denoted by $\lambda \in \Lambda$ and A with its hyperparameters instantiated to λ is denoted by A_λ. Given a dataset D the goal is to find:

$$\lambda^* = argmin_{\lambda \in \Lambda} E_{(D_{train}, D_{valid}) \sim D} V(L, A_\lambda, D_{train}, D_{valid}) \quad (1.1)$$

Where $V(L, A_\lambda, D_{train}, D_{valid})$ measures the loss of a model generated by algorithm A with hyperparameters λ on training data D_{train} and evaluated on validation data D_{valid}.

1.2.4 Joint Problem Definition

The former two, Algorithm Selection and HPO, are intertwined and form the core of every AutoML system. Together they can be tackled as a joint, structured optimization problem referred to as *Combined Algorithm Selection and Hyperparameter Optimization (C.A.S.H.)* [6]. Furthermore, if the selection of preprocessing algorithms is also optimized complete ML pipelines are built and the process is referred to as *Full Model Selection (F.M.S).*

The definition of the newly formed problem follows [7]: Let $A = \{A^{(1)}, \ldots\ldots, A^{(R)}\}$ be a set of algorithms, and let the hyperparameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$. Further, let $D_{train} = \{(\chi_1, y_1), \ldots\ldots, (\chi_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{D_{valid}^{(1)}, \ldots\ldots, D_{valid}^{(K)}\}$ and $\{D_{train}^{(1)}, \ldots\ldots, D_{train}^{(K)}\}$ such that $D_{train}^{(1)} = D_{train}^{(1)} / D_{valid}^{(K)}$ for i=1,...,K. Finally, let $L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(K)})$ denote the loss that algorithm $A^{(j)}$ achieves on $D_{valid}^{(i)}$ when trained on $D_{train}^{(i)}$, with hyperparameters λ. Then, the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem is to find the joint algorithm and hyperparameter setting that minimizes this loss:

$$A^*, \lambda^* = argmin_{A^{(j)} \in A, \lambda \in \Lambda^j} \frac{1}{K} \sum_{\iota=1}^{K} L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}) \quad (1.2)$$

1.2.3 Organization of this Thesis

The structure of this thesis is as follows. Chapter 2 provides the necessary theoretical background for Meta Learning (2.1) and Bayesian Optimization (2.2), two of the main techniques used in *AutoClust.* Chapter 3 presents the Related Work for both the cases of supervised (3.1) and unsupervised learning (3.2) in AutoML Systems. Chapter 4 presents the design choices and architecture specifications of the proposed solution in this Thesis, along with detailed explanation of the components. Chapter 5 provides the experimental setup, results and discussion, and Chapter 5 follows with conclusions, AutoML in the context of Big Data and open challenges of the AutoML domain.

Chapter 2 – Theoretical background

The scope of this chapter is to provide information about the two most important features of the system designed for the purpose of this Thesis. The subchapters that follow provide the necessary theoretical background to *Meta Learning* and *Bayesian Optimization*, the two methods that were selected to be implemented for *Algorithm Selection* and *Hyperparameters Tuning* respectively.

2.1 Meta- Learning

It is only logical when building a new machine learning model for a specific task to use experience from related tasks and the understanding of the machine learning techniques to make the right design choices. Task similarity is a key concept for the efficiency of extracting knowledge. Meta Learning in the context of Machine Learning, is the process of using any information stemming from prior experience with similar tasks, called Meta–Data, to improve or speed up the creation of ML models for new tasks. This information varies from algorithm configurations, learned model parameters, model evaluations, as well as measurable properties of the task itself, also known as *Meta–Features*. Which Meta–Features should be used and the reasoning behind using them, differs on the nature of the problem at hand and the task to be performed. A variety of Meta-Features is suggested in the literature, the most common of which can be found under Table 1 and are mostly drawn from statistics, information-theory and landmarking. Statistics and Information Theory include Meta-Features such as the variance of the data instances or the entropy of the features, while Landmarking refers to extracting information from the execution of simple learners, such as the accuracy of a decision tree Classifier. Together they consist of a new dataset often referred as Meta-Features Database.

In AutoML, Meta Learning has two major applications, namely algorithm selection and warm-starting optimization procedures and this is the design approach followed in AutoClust. Algorithm Selection can be described exclusively as a Meta Learning problem; learning intrinsic relationships between a data set characteristics and algorithms performance to create a model that is able to suggest a solution for the dataset at hand. The aim is to build a selector that will map characteristics of new instances of problems to an algorithm appropriate for the task. A simplistic yet representative example is using the percentage of outliers in a dataset. It is commonly known that clustering algorithms such as DBSCAN naturally handle outliers while KMeans does not, information that naturally the Machine Learning practitioner would make use of.

In addition, Meta Learning can be used to initiate optimization procedures. As the number of evaluations for the optimization procedure is limited, knowing which points to evaluate first would lead to the creation of a model that converges faster to an optimum. The history of configuration evaluation for a similar task can provide an initial search space to initiate optimization for problems at hand. This method has been

researched previously to *warm start* genetic algorithms [7] and Bayesian Optimization [8] with a variety of surrogate model choices and to a great success.

Name	Formula	Rationale	Variants
Nr instances	n	Speed, Scalability	$\frac{p}{n}$,$log\ log\ (n)$, $log\ log\ \left(\frac{n}{p}\right)$
Nr features	p	Curse of dimensionality	$log\ log\ (p)$,% co
Nr classes	c	Complexity, imbalance	$ratio\frac{min}{maj}class$
Nr missing values	m	Imputation effects	$\%\ missing$
Nr outliers	o	Data noisiness	o/n
Skewness	$\frac{E(X-\mu\chi)^3}{\sigma_\chi^3}$	Feature normality	$min, max, \mu, \sigma, q_1$
Kurtosis	$\frac{E(X-\mu\chi)^4}{\sigma_\chi^4}$	Feature normality	$min, max, \mu, \sigma, q_1$
Correlation	$\rho X_1 X_2$	Feature interdependence	$min, max, \mu, \sigma, \rho_X$
Covariance	$covX_1X_2$	Feature interdependence	$min, max, \mu, \sigma, co$
Concentration	$\tau X_1 X_2$	Feature interdependence	$min, max, \mu, \sigma, \tau_X$
Sparsity	Sparsity(x)	Degree of discreteness	min, max, μ, σ
Gravity	Gravity(x)	Inter-class dispersion	
ANOVA p-value	$p_{val_{X1,x2}}$	Feature redundancy	$p_{val_{XY}}$
Coeff. Of variation	$\frac{\sigma_Y}{\mu_Y}$	Variation in target	
PCA $\rho\lambda_1$	$\sqrt{\frac{\lambda_1}{1+\lambda_1}}$	Variance in the first PC	$\frac{\lambda_\iota}{\sum_\iota\ \lambda_\iota}$
PCA skewness		Skewness of the first PC	$PCA\ kurtosis$
PCA 95%	$\frac{dim_{95\%var}}{p}$	Intrinsic dimensionality	
Class probability	P(C)	Class distribution	min, max, μ, σ
Class entropy	$H(c)$	Class imbalance	
Norm. entropy	$\frac{H(X)}{n}$	Feature informativeness	min, max, μ, σ
Mutual Inform.	$MI(C,X)$	Feature importance	min, max, μ, σ
Uncertainty Coeff.	$\frac{MI(C,X)}{H(C)}$	Feature importance	min, max, μ, σ
Equiv. nr. Feats	$\frac{H(C)}{MI(C,X)}$	Intrinsic dimensionality	

Noise signal ratio	$\frac{H(X) - MI(C,X)}{MI(C,X)}$	Noisiness of data	
Fisher's discrimin.	$(\mu_{c1} - \mu_{c2})^2$	Separability classes c1, c2	Ref [9]
Volume of overlap		Class distribution overlap	Ref [9]
Concept variation		Task complexity	Ref [10]
Data consistency		Data quality	Ref [11]
Nr nodes, leaves	$\|\eta\|, \|\psi\|$	Concept complexity	Tree Depth
Branch length		Concept complexity	min, max, μ, σ
Nodes per feature	$\|\eta\chi\|$	Feature importance	min, max, μ, σ
Leaves per class	$\frac{\|\psi_c\|}{\|\psi\|}$	Class complexity	min, max, μ, σ
Leaves agreement	$\frac{n_{\psi i}}{n}$	Class separability	min, max, μ, σ
Information gain		Feature importance	$min, max, \mu, \sigma, gi$
Landmarker(1NN)	$P(\theta_{1NN}, t_j)$	Data sparsity	Elite 1NN
Landmarker(Tree)	$P(\theta_{Tree}, t_j)$	Data separability	Stump, Random Tree
Landmarker(Lin)	$P(\theta_{Lin}, t_j)$	Linear separability	Lin.Disciminant
Landmarker(NB)	$P(\theta_{NB}, t_j)$	Feature independence	More Models
Relative LM	$P_{a,j} - P_{b,j}$	Probing performance	
Subsample LM	$P(\theta_i, t_j, s_t))$	Probing performance	

Table 1 Overview of the Most common Meta Features used in the Literature [7]

2.2 Bayesian Optimization

Optimization in general is the idea of maximizing or minimizing, depending on the problem specifics, an objective function. In almost every domain there is a time that people need to optimize a routine to enhance performance. Questions like "*which is the less timely route to take*" or "*which strategy minimizes a certain risk*" often arise and have a great impact on health, economics, production, etc. Decision-making and automation in general can benefit from optimization procedures. In the context of Machine Learning, optimization is very crucial for tuning certain parameters such as the weights of a *Neural Network* or even the choice of kernels for the well-known Support Vector Machines (*SVM)* classification algorithm. AutoML systems usually treat the choice of algorithm to use when creating a Machine Learning pipeline for a certain problem instance, as an additional parameter and its respective input parameters as of conditional parameters. In that manner, the whole procedure of creating a pipeline that corresponds to certain

evaluation metrics *(such as final accuracy of a classification task, or execution time)* can be cast as an optimization problem. Among a set of optimization techniques available, the most prominent and popular choices in the domain are *genetic programming, bandit–based algorithms, Bayesian optimization* and variations that include combinations of the last two. This subchapter provides the theoretical background for Bayesian Optimization, the technique adopted in the *AutoClust* architecture, which is a state-of-the-art technique that has been proven to be effective for various AutoML systems as presented in chapter 3.

BAYESIAN OPTIMIZATION ALGORITHM

1: ***While*** *iteration <= Budget* ***do****:*

2: Select point X_{iter} *to evaluate* $f(x)$ *by optimizing acquisition function* a

$$X_{iter} = argmax\ a(X; D)$$

3: *Evaluate* $f(x)$ *on* X_i

4: *Update Statistical Model*

5: *Iteration = iteration + 1*

Bayesian Optimization is a sequential-model technique to find the global optima for black box functions that are computationally expensive to evaluate, by creating surrogate models. Those black box functions will be assumed to have no closed form but can queried in every point of a configuration space and can only be observed through noisy observations y [8]. In detail, a prior distribution represents the belief over an objective function that is sequentially updated as more information from evaluations is gained (*surrogate model)*. Modelling the objective function allows for the new point to evaluate to be selected in a probabilistic manner via maximizing functions that measure the expected gain for the new point of evaluation also known as *acquisition functions.* The new selected point to evaluate is likely to maximize gain, according to exploration and exploitation trade-offs, and the results update the posterior distribution to improve the already defined model. At this point it should be noted that the number iterations *N* is required to be defined as a termination criterion for the algorithm.

Certain variations of the algorithm exist according to the selection of surrogate model to define posterior distribution or the acquisition function used to select the next point for evaluation. All of the acquisition functions trade off exploration and exploitation, meaning that the values to be explored are selected where the model prediction is high *(exploitation)* and the uncertainty of the surrogate model is large *(exploration)* [9]. Several functions serve that purpose, *Thomson Sampling, probability of improvement and entropy search* to name a few, but the most popular choice used by almost all the state-of-the-art

AutoML systems is the *Expected Improvement Criterion (EI).* Contrary to the similar acquisition function *probability of improvement*, *EI* takes into account the level of the improvement and is defined as follows:

$$a_{EI}(x) = E[u(x)\,|x, D] = \int_{-\infty}^{f'} (f' - f) N\big(f; \mu(\chi), K(\chi, \chi)\big) df$$
$$= \big(f' - \mu(\chi)\big)\Phi\big(f'; \mu(\chi), K(\chi, \chi)\big) + K(\chi, \chi) N\big(f'; \mu(\chi), K(\chi, \chi)\big) \quad (2.1)$$

Where

$$u(x) = \max(0, f' - f(x))$$

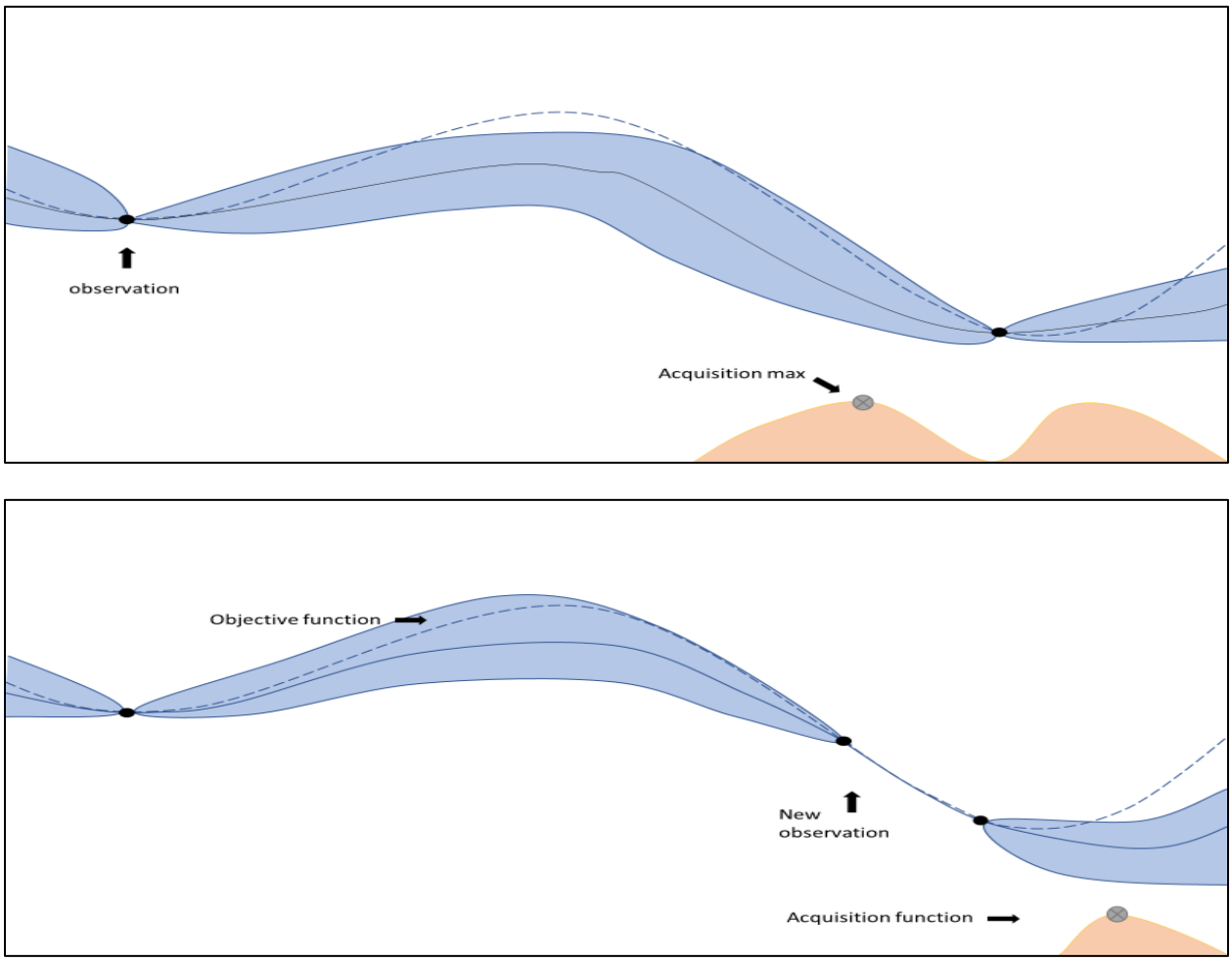
In the equation (2.1), f' refers to the minimal value of f observed so far and *μ(χ), K(χ, χ)* to the mean and variance of the posterior distribution as modeled by a Gaussian Process. Gaussian Processes constitute one of the most common choices for modelling the posterior distribution. Other alternative and frequently used models for Bayesian optimization are Random Forests that can handle natively large, conditional and categorical data *(used by Auto-Weka , see chapter 3.1.1)* and *Tree Parzen Estimator* which has been selected for the Hyperparameter tuning of AutoClust architecture and is also supported by *HyperOpt-Sklearn. Tree Parzen Estimator* (TPE) models the density functions P(λ|y>a) and P(λ|y<a) instead of P(y|λ). Given a percentile α, the two distributions mentioned are modeled by 1-d Parzen Windows. The acquisition function of *Expected improvement Criterion* is transformed to the ratio:

$$ei_{TPE} = \frac{\mathrm{P}(\lambda|\mathrm{y} < \mathrm{a})}{\mathrm{P}(\lambda|\mathrm{y} > \mathrm{a})} \quad (2.2)$$

This choice of surrogate model for Hyperparameter tuning is based on certain studies that demonstrated its adequate performance on parametric spaces with conditional parameters, scaling to higher dimensions and natural parallelization, a very crucial property for the *AutoClust* component that is developed under the scope of Big Data. In the work of Bergstra et. Al [9], TPE is compared to Gaussian Processes in terms of parallel computation efficiency. In this experiment both iterations of optimization are executed asynchronously for 1 hour of GPU – Computing, for the Boston Housing Dataset. The iteration with TPE as surrogate model provided better results in terms of test set classification accuracy dataset indicating the algorithms efficiency when ran in parallel nodes. Furthermore, in the work of Eggensperger et al. [10], three variations of Bayesian Optimization were tested according to the choice of surrogate model (Random Forests, TPE and Gaussian Processes). Random Forests performed overall best, Gaussian Processes performed better for low dimensional continuous problems and TPE better on discrete - low

dimensional spaces. In addition, CPU time was evaluated. Random Forests and TPE's overhead when selecting a point to evaluate next (1s), was negligible compared to Gaussian Processes (>42s).

To gain a better understanding of the optimization procedure, Figure 1 visually presents three consequent iterations of the algorithm for a 1-dimensional function. The goal is to minimize the dashed line, while using a Gaussian Process as surrogate model which is represented by the black line with a blue tube representing the uncertainty around estimations. The orange curve depicts the acquisition function, which in turn indicates the point to be evaluated next, at the point the acquisition function is maximized. Values of the acquisition function are lower around observations and highest where the function value is low. While the variance at the left of the acquisition function is relatively high the predicted mean is much lower to the right indicating the point for the next evaluation.



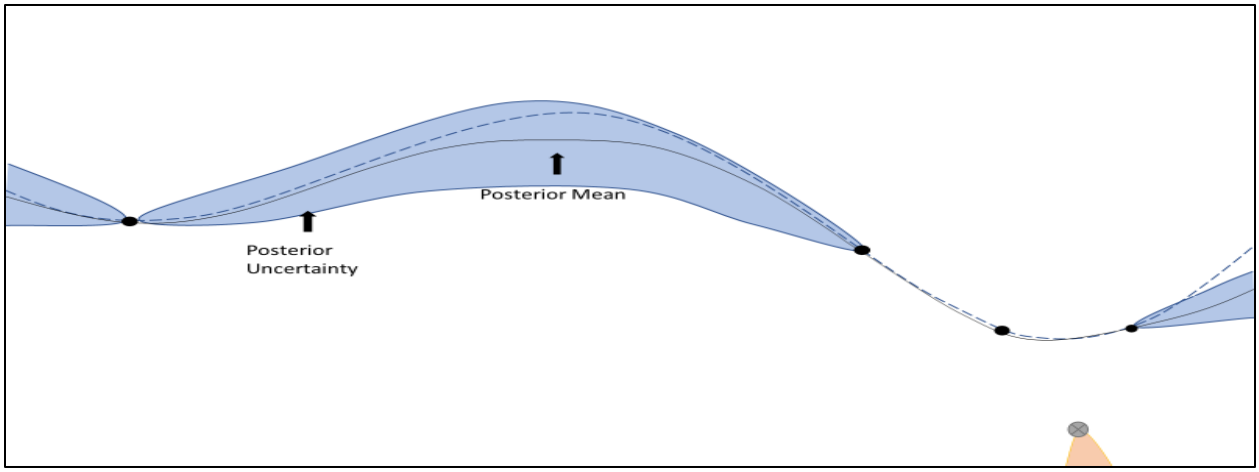


Figure 1 Three Iterations of Bayesian optimization where the dashed line is the goal of minimization and blue tube represents the uncertainty of estimates. The orange curve represents the acquisition function with an X referring to the point where the acquisition function is maximized.

Chapter 3 - Related Work

3.1 AutoML Systems

With the advent of Machine Learning, a variety of packages have been developed to provide solutions in the AutoML domain. The domain is diverse and packages target both researchers and end-users. While the list grows long, in this subsection, a brief review of the most prominent systems is provided.

3.1.1 Auto-Weka

Although algorithm selection and hyperparameter optimization had been previously researched separately, *Auto-Weka* was the first system that proposed to treat them as a joint problem dubbing it as C.A.S.H (1.2). The main focus of Auto Weka is on Classification the proposed method can be applied to any problem that can be treated as optimization problem. It is named after the open source data analysis package known as *WEKA* used for implementation, written in JAVA, and covers all its learners and feature selectors. Auto-Weka tackles the problem of C.A.S.H. by using *Sequential Model Algorithm Configuration – SMAC,* a Bayesian Optimization method with Random Forests that handles adequately discrete and high dimensional input data. Some of the key features of SMAC are the following:

1) It handles conditional parameters by instantiating inactive ones to default values for model training and prediction allowing individual decision trees to create splits that check if a certain hyperparameter is active.
2) The *Expected Improvement Criterion* is used to choose the next configuration setting for evaluation.
3) It implements a certain mechanism to deal with noisy function evaluations. The function to be optimized in the C.A.S.H problem is the mean of a set of losses as calculated by Cross Validation for different pairs of D_{train}, D_{test}. SMAC progressively makes better estimates as it compares these terms one at a time between evaluation points. For a new configuration to outperform the previously best resulting one it must provide better values of 1-fold,2-fold up to the number of folds used previously used. If finally, the new configuration proves to be better, the number of folds for the evaluation method is increased. This procedure, allows for poorly performing configurations to be discarded after just a single fold evaluation.
4) Finally, SMAC selects a random configuration to evaluate every two evaluations in order to make the procedure robust even when the model is misled.

Experiments in the respective Auto-Weka research [6], were conducted on 21 different datasets including popular data sets of images such as *MNIST, CIFAR-10*. In order to create a strong baseline for comparison,

the authors are suggesting three different methods *Grid Search, Random Search* and using the default parameters to the *Weka* algorithms referred to as *Ex-Def*. Results are presented for a 10-fold Cross Validation and to evaluate generalization on a different set of test data. For 10-fold Cross Validation, Grid Search outperforms Ex-Def in 19/21 cases highlighting the importance of Algorithm Selection, Random Search outperforms Grid Search in 9/21 cases providing evidence that larger computational budget does not necessarily mean better results. For the test data *Auto-Weka* outperforms baseline methods in 15/21 cases with reductions to misclassification rates exceeding 16% in 3/21 cases [6].

3.1.2 Auto-Sklearn

Auto-Sklearn is another of the most prominent AutoML packages that won six out of ten phases of *ChaLearn AutoML* challenge, one of the most noticeable challenges in the domain. It is written in Python and makes use of all feature and data preprocessors and learners of Scikit-Learn *(15 Classification Algorithms, 14 preprocessing methods and 4 data preprocessing methods)*, with focus on the supervised problem of Classification. Following *Auto-Weka* and the study of Thornton et al [11]. that provides evidence of random-forest-based SMAC outperforming *Tree Parten Estimators*, Auto-Sklearn implements SMAC for Hyperparameter tuning.

Adding to the AutoML Domain, previously defined by Auto-Weka*, Auto-Sklearn* introduces two novelties; the use of meta-learning to warm-start the Bayesian optimization procedure and an ensemble construction step that uses more than one configuration found in the optimization procedure*. Auto-Sklearn* warm-starts the Bayesian optimization procedure by providing initial instantiations from configurations that produced the best results to similar datasets. In an offline fashion, SMAC is performed on a collection of 140 data sets and the best resulting configurations are stored according to the mean accuracy of a 10-fold Cross Validation. Similarities among datasets are defined by the L_1 distance between 38 Meta Features extracted from each data set. Furthermore, a post-processing method is suggested to utilize models trained during the course of Bayesian Optimization and later discarded. A selection of 50 of those models, that in many cases may yield close to optimal results, construct an ensemble with *ensemble selection* [12] as the method for calculating weights.

As a measure of comparison, the authors [11] use the exact same setting presented in *Auto-Weka* evaluation without the introduction of the two novelties to test performance and compare their framework. *Auto-Sklearn* led to 12/21 ties, 6/21 wins and the rest three as losses, a significant improvement. For the two novelties, results were tested for the vanilla Auto-Sklearn with and without Meta Learning and ensemble building, on a collection of 140 datasets from OpenML according to *balanced classification error rate (BER).* Both of the methods proved to improve performance over vanilla Auto-Sklearn. Most notably, Meta Learning showed significant improvements to all of the configuration evaluations in the Bayesian optimization procedure and ensemble construction proved to benefit from Meta Learning as better performing models were chosen to build the ensemble stack [11].

3.1.3 Auto-Net

Also, worth mentioning is *Auto-Net*, an AutoML framework designed as an extension to Auto-Sklearn specifically for tuning deep learning Neural Networks. Authors of Auto-Net present two versions of the package, version 1.0 and 2.0; The first version of *Auto-Net,* labeled as 1.0, is considered as an extension to *Auto-SKlearn* focusing solely on hyperparameter optimization of *fully connected feed-forward Neural Networks*. It is based on *Lasagne*, a Deep Learning library in Python that is built around the popular framework *Theano*. Following *Auto-Weka* and *Auto-Sklearn* it applies SMAC for hyperparameter optimization in a configuration space of sixty-three hyper-parameters listed in Table 2. The Number of layers is constrained to vary from one to six in order to keep training time of a single configuration low and avoid further complication of the process as each layer adds eight hyper-parameters. Version 2.0 is an alternative version of Auto-Net built upon PyTorch instead of *Lasagne* and thus is also referred to as Auto-PyTorch. Focus of the later version is on modern neural networks, thus supports a variety of different deep learning modules such as network type, learning rate, scheduler etc. At the time of writing Auto-PyTorch supports four different network types: *Multi-Layer Perceptrons, Residual Neural Networks, Shaped Multi-Layer Perceptrons* and *Shaped Residual Networks* tuning them for a set of 112 hyperparameters presented in Table 3.

Network Hyperparameters	*Batch Size, Number of Updates, Number of Layers, Learning rate,* L_2 *regularization, Dropout output layer, Solver type, lr-policy*
Conditioned on solver type	β_1, β_2, *momentum*
Conditioned on lr-policy	*γ, k, s*
Per-layer hyperparameters	*Activation Type, Number of units, Dropout in layer, Weight initialization, Std. normal init., Leakiness, tanh scale in, tanh scale out.*

Table 2 Hyperparameters available for tuning with *Auto-Net 1.0*

In order to evaluate Auto-Net 1.0, the authors present the framework's performance as defined by the rules of the AutoML challenge phases. In order to set a baseline evaluation, the framework is executed both on GPU or CPU and its results are compared to those of Auto-Sklearn, for the five datasets of phase 0 of the challenge. It should be noted that an ensembling method is applied to the models suggested during the optimization procedure to select the final architecture selected by *Auto-Net.* The framework ran on GPU performed best on one, tied in three and lost to one dataset in terms of test error with a five-fold cross validation methodology. For later phases of the AutoML challenge authors combined both frameworks, AutoNet and Auto-Sklearn, winning on a competition dataset of the third phase where results are compared to those of human experts and achieving third place in the fourth phase of the challenge. Finally, to compare both versions of *Auto-Net*, two datasets of different size were chosen as benchmark (*newsgroup with 13k training data points and Dorothea with 800 training data points).* For the

newsgroup dataset, version 2.0 performed better while for the *Dorothea* dataset version 2.0 performs better early on but given enough time the former performs slightly better. This is attributed by the authors to the lack of ensembling and the larger search space of *Auto-PyTorch* [13].

General Hyperparameters	*Batch Size, Use mixup, Mixup alpha, Network, Optimizer, Preprocessor, Imputation, Use loss weight strategy, Learning rate scheduler*
Preprocessor	
Nystroem	*Coef, degree, Gamma, Kernel, Num components*
Kitchen sinks	*Gamma, Num components*
Networks	
MLP	*Activation function, Num Layers, Num units (for layer i), Dropout (for layer i)*
ResNet	*Activation function, Residual block groups, Blocks per group, Num units, Use dropout, Dropout (for group i), Use shake drop, Use shake shake, Shake drop* β_{max}
ShapedMLP	*Activation function, Num layers, Max units per layer, Network shape, Max dropout per layer, Dropout shape*
Shaped ResNet	*Activation function, Num layers, Blocks per layer, Use dropout, Max units per layer, Network shape, Max dropout per layer, Dropout shape, Use shake drop, Use shake shake, Shake drop* β_{max}
Optimizers	
Adam	*Learning rate, Weight decay*
SGD	*Learning rate, Weight decay, Momentum*
Schedulers	
Step	γ*, Step size*
Exponential	γ
OnPlateu	γ*, Patience*
Cyclic	*Cycle length, Max factor, Min factor*
Cosine annealing	T_0, T_{mult}

Table 3 Hyperparameters available for tuning with *Auto-Net 2.0*

3.1.4 TPOT

TPOT, short for *Tree-based Pipeline Optimization Tool,* is an open source Python project that alternate to Bayesian optimization methods, automates machine learning pipelines by using genetic programming *(GP),* a well-known technique for the automatic construction and evolution of programs. The focus of this project is on supervised learning, specifically the task of classification with support of one hundred and fifty of ScikitLearn [14] algorithms including preprocessing ones. A random set of 100 tree-based pipelines creates an initial generation, as referred in genetic programming, and is further optimized according to the Python package DEAP. Twenty of the best pipelines in terms of maximizing cross validation accuracy and minimizing the number of processes, are selected and are mutated to produce a new generation of pipelines. Each of the pipelines produces five more by cross-over techniques or random insertions and shrinks. The algorithm is executed for 100 generations for each of those a *Pareto front of the non-dominated solutions* being updated.

In order to demonstrate the results of *TPOT*, the authors use a variety of benchmark datasets that are subject to 30 replicates of the procedure with different random generator seeds. Datasets are split into *75%* training set and *25%* test set and are evaluated in terms of *balanced accuracy* which corrects for class frequency imbalance. The performance is compared with that of 30 replicates of *Random Forest* with 500 trees on the datasets, a choice that is to simulate a basic machine learning practitioner's choice for classification. As an additional measure of comparison, evaluation on 30 replicates of TPOT with random generation of pipelines was meant to represent a *Random Search* in the TPOT's search space. Following the first scenario of evaluation, TPOT showed improvements over median accuracy varying from 10% to 60% while only degrading in the scale of 2%-5% when performing worse. Furthermore, it allowed for the discovery of useful preprocessors such *RandomizedPCA* for a benchmark dataset leading to a near perfect accuracy. In the second scenario, while *RandomSearch* produced very competitive results compared to those of TPOT it did not take into account the number of pipeline operations leading to complex and computationally intensive solutions [15].

3.1.5 Google Vizier

Finally, *Google Vizier* is a state-of-the-art research, with focus on black-box optimization. It provides core components for the Google's cloud Machine Learning subsystem *HyperTune,* it is highly scalable and is implemented in Python, C++ and Golang.

A key feature of *Google Vizier* is dynamic selection of optimization algorithms; for example, Gaussian Process Bandits, a well-known optimization technique, provides excellent results but scales as $O(n3)$ [16] with the number of training points, thus if it is chosen to initiate the optimization procedure it can be dynamically replaced later on by more scalable algorithms to carry on the process. Although a variety of algorithms for optimization are supported, Google Vizier defaults to *Batched Gaussian Process Bandits* for short scale studies and recommends proprietary local-search algorithms for large scale ones. Another

important feature is *automated early stopping,* a technique that aims to terminate parameter exploration if the next configuration to be evaluated is not predicted to hold any improvements over the last configuration evaluated. The technique mentioned is implemented via two different methods; the first one is *performance curve stopping rule,* where a *Gaussian process regressor* is trained on a partial performance curve and parameters of the optimization procedure to predict whether the optimal value of the objective function, found until any given point, is sufficiently low to terminate the process. The second one is *median stopping rule,* a rule that terminates the optimization process if at any given step the objective found is strictly worse than the running average of the previous ones. Last but not least, *transfer learning* is supported, starting for example optimization procedures for new datasets from resulting configurations of studies for the same task but different data sets. The approach used for *transfer learning* is to build a stack of *Gaussian Process Regressors* where each of the regressors is associated with an optimization procedure that has already taken place and regresses on the residual of its objective relative to the prediction of the regressor below. Each component of the stack is placed according to a chronological order with top stack representing the most recent studies.

The authors provide results by benchmarking samples of different dimensionalities in terms of 4 optimization algorithms implemented in *Google Vizier (multiarmed bandit technique using a Gaussian process regressor, SMAC, Covariance Matrix Adaption Evolution Strategy and a probabilistic search method created by the authors)* compared to vanilla *RandomSearch* and a variation that evaluates two points at each step compared to other methods. Every algorithm showed improvements over *RandomSearch* while Gaussian Process regressors and the probabilistic model of the authors also found the best configurations. Furthermore, both of the *early stopping criteria were tested* and *Median Automated Stopping Rule* held the best results by speeding up *RandomSearch* by a factor of two and always finding the best resulting configurations in samples and popular datasets such as CIFAR10, an image classification dataset [17].

3.1.6 Comparison of AutoML Systems

Many more systems exist, most of them being open-source projects and include some notable properties. Before moving to the comparison of the AutoML systems, a list of some systems with properties worth mentioning follows:

- *TransmogrifAI:* a library written in *Scala* that is scalable as it runs on top of *Apache Spark.*
- *H2O:* An AutoML system supporting Python and R that supports stacked ensembles of previously trained models to create a predictive model
- HyperOpt-Sklearn: An open-source Python library for Bayesian Optimization with *Tree Parzen Estimators (TPE)*
- *Automated Statistician:* A software that provides automatic creation of pipelines through *Gaussian Processes,* supports *Time Series* and automatic reporting of the results.

When it comes to comparing the AutoML solutions that were presented in this chapter, there is no definitive winner. As listed in Table 4 Comparison of the most prominent AutoML Systems

, the frameworks mainly differ in terms of the algorithmic choice for Hyperparameter tuning, scalability and Implementation platforms. Bayesian Optimization is the most popular choice for Hyperparameter Tuning but approaches such as genetic programming and Bandit processes have been also proven to provide sufficient results. Furthermore, it can be perceived that most of the systems are implemented in Python, a choice that is not unreasonable. Scikit Learn provides a wide variety of Machine Learning Algorithms along with many preprocessing ones and libraries such as *Keras* and *Pytorch* are some of the most prominent choices for designing modern Neural Networks. An aspect that all the AutoML solutions share in common is that they support only supervised tasks, a problem that comes from the uncertainty around the choice of objective function to optimize. Meta Learning is used only in AutoSKlearn and as a subsystem to Google Vizier, but in this context provided adequate results improving optimization performance in terms of quality and convergence time.

	Auto-SKlearn	**Auto-Weka**	**TPOT**	**Auto-Net**	**Google Vizier**
ML Tasks	Supervised	Supervised	Supervised	Supervised	
HPO	Bayesian – Random Forests	Bayesian - Random Forests	Genetic Optimization	SMAC	Variety of Algorithms – Batched Gaussian Process
Implementation Platform	Python	Weka	Python	Python	Python, C#, Golang
Key Features	Warm-Starting HPO via Meta Features	First Complete AutoML System	Genetic Optimization, Minimization of the number of processes	Focus on modern Neural Networks	Dynamic Optimization Algorithm Selection, Transfer Learning

Table 4 Comparison of the most prominent AutoML Systems

3.2 AutoML in Clustering

As opposed to the extent of research conducted for the task of classification and regression in the context of AutoML, clustering is less explored. Problems in the context of unsupervised learning are by nature defined by subjectivity and relativity to the domain of the problem thus it is obscure which objective function for results evaluation to use. Different clustering solutions for the same dataset may be perceived as correct depending on the information that needs to be extracted and practitioners often refer to visual representation to validate a clustering schema. For that reason, it is that automatic creation of pipelines for the Machine Learning task of clustering that generalize well across various problem instances is a tedious task. Until the time of writing, few researches address this problem and mostly tackle the problem of *Algorithm Selection (chapter 1.2)* alone. To the best of our knowledge the first study to address the problem of *Algorithm Selection* in the context of clustering was that of *Souto et al* [18]. Authors suggested a *Meta Learning* approach in order to extract knowledge from similar previously tackled problem instances. To explore the relationship of Meta Features and Clustering algorithm performance, each dataset from a set of 32 *microarray gene expression datasets* was characterized by a set of *Meta-Features* as listed in Table 5 Meta Features used in the study of Souto et al [18]

. Seven of the most prominent clustering algorithms were executed on this set of datasets and provided empirical results that create a ranking of the algorithms for each dataset in terms of performance. Quality of results was measured in terms of difference of the clustering schemas provided by the algorithms and external information about the true clusters of the data with the use of *Spearman's rank Correlation coefficient*. Results were compared to an average ranking selected for each dataset and were sufficiently higher.

LgE	log_{10} of the number of examples
$LgREA$	log_{10} of the ratio of the number of examples by the number of attributes
PMV	Percentage of Missing Values
MN	Multivariate normality, which is the proportion of T^2 that are within Chi-squared distribution (with degree of freedom equal to the number of attributes describing the example)
SK	Skewness of the T^2 vector.
$Chip$	Type of microarray technology used.
PFA	Percentage of the attributes that were kept after the application of the attribute selection filter.
PO	Percentage of outliers.

Table 5 Meta Features used in the study of Souto et al [18]

To extend the work of *Souto et al*, a more recent study of *Ferrari et Castro* [18], introduced two novelties; The first one is a set of *Meta Features* tailored specifically for the problem of algorithm selection on clustering, based on the distance distribution of the data set instances as listed in Table 7. In addition to the previously mentioned *Meta Features*, a set of extra 9 *Meta Features* from Statistics and Information Theory were used (Table 6). The second novelty is on the ranking methods the authors experimented with for the algorithm comparison. Contrary to the former research on the topic, *Ferrari et Castro* use datasets without any external information about the true clustering schemas of the data so as to create a selection process that better reflects the nature of unsupervised learning. They used 10 internal validation indices that defined 10 different rankings of the algorithm performance on each dataset. In order to be comparable those rankings were merged into one single result by three different methods; using the *average ranking* , a *score ranking* method were for each index algorithms were assigned points that were summed later on, according to performance and finally a *winner ranking* method that takes into account the number of victories of all the algorithms in each individual ranking. Results were demonstrated by *Spearman's Correlation Coefficient* according to a standard ranking, a system that averages all the rankings across the Meta Features Databases. To model the relationship of *Meta Features, K-Nearest-Neighbors,* along with *Euclidean* distance. Using alternate sets of Meta Features, evidence was provided that the novel distance based and hybrid versions with the attribute based proved more efficient with the use of solely the attribute based [18].

META-FEATURES BASED ON INFORMATION THEORY	
MA-1	Log2 of the No of Objects
MA-2	Log2 of the No of Attributes
MA-3	Percentage of Discrete Attributes
MA-4	Percentage of Outliers
MA-5	Mean Entropy of Discrete Attributes
MA-6	Mean Concentration between Discrete Attributes
MA-7	Mean absolute Correlation between continuous attributes.
MA8	Mean skewness of continuous attributes
MA9	Mean kurtosis of continuous attributes

Table 6 Meta Features from Statistics and Information Theory used in the study of Ferrari et Castro

META-FEATURES BASED ON THE DISTANCE DISTRIBUTION OF INSTANCES	
MD-1	Mean of distances vector
MD-2	Variance of distances vector
MD-3	Standard Deviation of distances vector
MD-4	Skewness of distances vector
MD-5	Kurtosis of the distances vector
MD-6-14	Percentage of distance values in each of ten intervals that equally comprise range [0,1]
MD-15-19	Percentage of distance values with absolute z-score in four intervals of range [0, inf)

Table 7 Novel Meta Features based on the distance distribution of instances of a certain dataset, introduced by Ferrari et Castro

Finally, as part of an AutoML workshop, *Meta learning* was proposed as a technique to select the best cluster validation index (*CVI*) to assess the quality of a clustering schema to make up for the obscurity of selecting an objective function to optimize but adding to the level of complexity of algorithm selection. Experimentation showed adequate results while using a *Random Forests* method as a Meta Learner [19] Consensus clustering has been also researched on how to produce a generated comparison measure instead of "ground truth" leading to algorithm selection and optimization based on a more concrete validity index than internal ones [20].

Chapter 4 - Architecture

4.1 Design Specifications

AutoClust is developed with the use of the programming language Python. This choice complements many required processes of the framework, as it supports a wide variety of tools for Data Science. More specifically, Python libraries such as *Scikit-Learn* offer a large pool of implemented ML algorithms that create a core for the needs of the proposed architecture. Furthermore, state of the art optimization techniques are widely supported in many open source Python projects, thus enabling the addition of Hyperparameter Tuning in AutoClust. To implement Bayesian Optimization with the Tree Parzen estimators as noted in chapter 2.2 *(Theoretical Background-Bayesian Optimization)* the open source Python library HyperOpt is used. HyperOpt enables Parameter Optimization in a parallel manner with the use of *Apache Spark,* a distributed processing software, and *MongoDB* a Scalable NoSQL storage system. This feature along with the respective Python APIs for the softwares mentioned, enables handling and processing of big volumes of data, which in turn broadens the applicability of the prototype presented in this thesis, in use cases related to Big Data.

4.2 Architecture

Focus of AutoClust at the time of writing is Algorithm Selection and Hyperparameter Tuning. Following the state-of-the-art research of Ferrari et Castro a Meta Learning technique is used to select an appropriate algorithm for new unseen Datasets. Expanding on that, a Hyperparameter Optimization component based on Bayesian optimization is responsible for selecting a set of values for the input parameters of algorithms.

In this chapter the architecture of the AutoML solution dubbed as *AutoClust* is presented along with certain design choices such as of programming language, libraries and methods used. *AutoClust* is developed and works in two main phases; The first one will be referred to as *learning phase* (chapter 2.2) and aims to learn to map the characteristics of a problem instance to configuration performance, while

the second referred to as "*in-action"* or *online phase* (chapter 2.2) is responsible for providing results to the user by tailoring the solution at hand.

4.2.1 Training Phase – Analytics Repository Creation Process

Scope of the Training Phase is to extensively search large configuration spaces appropriate for performing a Machine Learning task *(T)*, for each dataset of a collection of datasets, to extract knowledge. This knowledge is to lead to a better understanding on the relationship between performance of a configuration and characteristics of the problem instances *(Meta Features of the datasets)*. The set of information produced is stored to be accessed later in the online phase and will be referred to as the *Analytics Repository (AR).* It should be noted at this point that this procedure although computationally intensive, takes place in an offline fashion meaning that it does not affect the runtime of the system when online providing solutions to the end user.

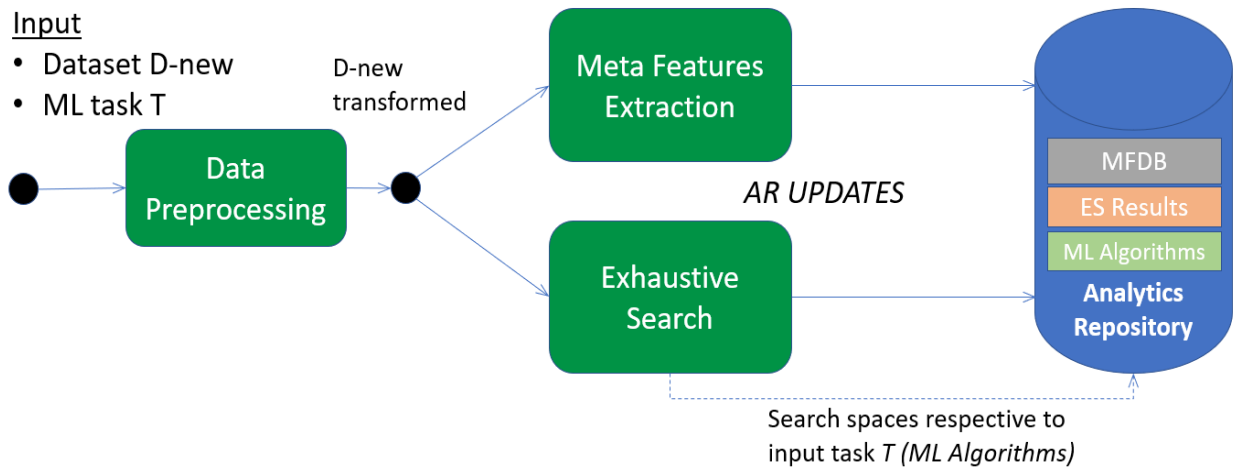


Figure 2 Training phase of the component/ update procedure of the Analytics Repository (AR)

Before moving forward to explaining the individual subcomponents used in the Training phase, it is important to clarify what the Analytics Repository consists of. The Analytics Repository (AR) is a collection of various types of information required for the execution of the individual processes of *AutoClust*. This information is collected and updated from executions of ML algorithms on previously seen datasets. Essentially, the system follows a "learning-to-learn" approach, and exploits knowledge assembled from

its past usage in order to constantly improve its performance with time. As shown in Figure 2, the collection of information in Analytics Repository falls into three distinct categories:

- Meta-Feature database (MFDB)
- Results of Exhaustive Search (ES Results)
- Machine Learning algorithms (ML Algorithms)

which are directly linked to the subcomponents as described next.

A. Data Preprocessing

Data Preprocessing has been proven to have a great impact in terms of performance when creating Machine Learning pipelines. The Data Preprocessing subcomponent is responsible for handling inconsistencies and preprocessing of the input dataset (denoted D_{new}), in order to provide a transformed version that is more suitable as input for task of clustering. Techniques to impute Missing Values allow for data instances to not be wholly discarded if they are corrupted *(missing certain values of dataset's features in a row)* but be included for the extraction of further knowledge. In addition, most of the clustering algorithms require calculating distance metrics among points, which may be sensitive to the scaling difference of the features of a dataset. A representative example being that of *Euclidean Distance,* one of the most commonly used, which can be easily observed in eq. 4.1 *(for p = (* $p_1, p_1 \dots, p_1$*) and* $q = (q_1, q_2 \dots, q_n)$*)* that is affected by the scaling of the features. For that reason, it is important to scale features of a dataset to be of the same range, most usually [0,1].

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \quad (4.1)$$

As an example, the Data Preprocessing is responsible for the following transformations on the input dataset:

- *Drops columns when they consist only of unique integer values*. As they do not hold any information on instance similarity, the key concept for creating partitions in clustering.
- *Drops columns with 30% or more missing values*. Those features lead to a set of instances that are hard to impute missing values correctly and may lead to unexpected, not representative results.
- *Scales data into [0,1] range by applying Min-Max scaling/normalization*. A choice to alleviate the sensitivity of distance metrics, commonly used by most of the clustering algorithms as mentioned before, to features scaling.
- *Replaces missing values with KNN strategies (regressor for continuous attributes and classifier for discrete attributes).* This method has been proven to show adequate results when compared to the well-known imputation techniques such as Mean, Median and Standard deviation Imputation [21]. Number of Neighbors for the algorithms is fixed to 10 and the *Euclidean distance* is used for defining instance similarity algorithm parameters.

It should be mentioned that no restrictive assumptions are made with respect to the original input data. It is expected to consist of *n* rows (records) and *m* columns (attributes) of numeric values. Obviously, both *n* and *m* may vary, depending on the dataset at hand. At the time of writing focus of research is on numeric datasets excluding text and Image datasets.

B. Meta Features Extraction

This subcomponent is responsible for the creation of Dataset descriptors in the form of numerical vectors. The Meta-Features Extraction subcomponent computes several characteristics of a dataset, also known as Meta-Features, that capture information about the instances in the dataset, based on Statistics, Information Theory and the distribution of pairwise distances of the objects (Table 6-Table 7) as used in the research of Ferrari et al. [18]. In addition, an extra set of novel Meta-Features is explored, proposed as part of this thesis, based on Landmarking. This set of Meta Features is a vector comprised of the scoring values, in terms of internal validation indices, of the MeanShift clustering algorithm. This algorithm was found suitable because it requires almost no parameters to be defined before execution and most importantly the number of clusters must not be predefined. This novelty is presented in the experimentation section (chapter 5.4.2) and is based on the idea that internal indices, when compared for certain configurations, can indicate properties of the Dataset structure. This set of vectors for characterizing the datasets is stored in the MFDB (Meta Features Database) in the Analytics Repository (AR), which is subsequently used for finding similar datasets to the dataset at hand during the online phase.

C. Exhaustive Search

The Exhaustive Search subcomponent produces instances of a configuration explored (ML Algorithm and a set of values for input parameters) and a set of proper evaluation metrics that was observed for this setting with respect to the ML Task of clustering. This is achieved using two of the most well-known exploration methods of Hyperparameter Tuning. The first method that is used is a “brute-force” approach called Grid Search that searches every possible combination of values of finite parametric spaces and is computationally intensive. The second one, known as Random Search, randomly selects a handful of combinations from those available for evaluation, is less time-consuming and has proven to outperform Grid Search in certain cases (chapter 5.2).

Both of these methods need a defined search space, which is drawn from ML Algorithms in the Analytics Repository This information, formatted in standard JSON, is responsible for recording the Machine Learning algorithms that are currently supported, their input parameters and the search spaces for the execution of Grid and Random Search. . At the time of writing the algorithms supported by *AutoClust* are those included in the Scikit-Learn 0.22.1 [14] and the reader is referenced to the respective

documentation for more details. For the search space is defined, every combination of ML algorithm and input parameters is executed for a given Dataset in Python (or theoretically any runtime for ML algorithms). The collection of information derived from Exhaustive Search (ES Results) is later used for the selection of ML models for new unseen Datasets, warm starting the Hyperparameter Optimization procedure and as a baseline for evaluating final results.

4.2.2 Selection Phase – "In Action"

In the second phase referred to as online, the goal is to make use of the information already stored in the *Analytics Repository* to provide a solution according to the specifics of the problem instance. Figure 5 presents the architecture design of *AutoClust's* Online phase. The input is provided as a dataset (D_{new}) that is subject to the Machine Learning task of Clustering. A Machine Learning algorithm is then selected (or recommended) along with specified values for the input parameters, both tailored at hand according to the dataset's properties. Based on the ordering presented in Figure 2, a comprehensive explanation of each process of the overall architecture follows.

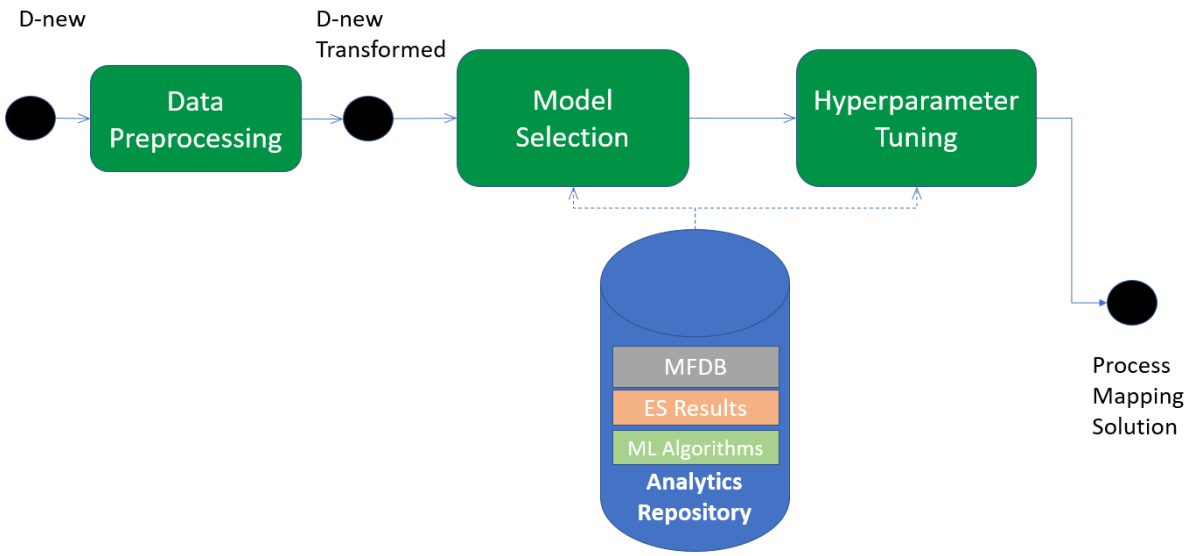


Figure 2 - AutoClust System Architecture

A. Model Selection

The main purpose of this process is to select or recommend a set of ML algorithms for the input dataset D_{new} and the analysis task of clustering. The idea behind this process is that similar datasets should have similar solutions for a given task T. In order to define similarity among datasets, several characteristics,

known as meta-features, are extracted from each dataset and create a feature vector which allows for the calculation of similarity measures between datasets. The construction of the feature vector consists of measurements already referenced in the Meta Features Extraction process description of the Training phase.

When this process is initiated, the meta features of the dataset are extracted and its similarity is computed against the meta features of instances of the Metafeature DB (MFDB), corresponding to previously seen datasets, located in the analytics repository. Then a subset of the most similar datasets is found along with the algorithms that produced the best results, based on the Exhaustive Search that occurred during the training phase. This process is the equivalent of a KNN Meta Learner and the choice of how many similar datasets to consider is further explored in the Experimentation chapter. If more than one dataset, are referenced for similarity, the KNN algorithm uses a majority voting strategy where the most frequent recommendations make up for the solutions.

B. Parameter Tuning

After the completion of Model Selection, AutoClust is able to return a specific ML algorithm that is considered suitable for the dataset at hand. However, it is yet unclear what values should be used for its input parameters. For instance, in the case of K-means the number of returned clusters should be provided as input. As another example in the case of DBSCAN, two input parameters must be set: MinPts and Eps. To address this problem, the Hyperparameter Tuning process is responsible for the optimized selection of the input parameter values. It is based on Bayesian Optimization, a state-of-the-art method that searches the global optima of an objective function f(x) through a surrogate model of the f(x) due to cheaper evaluations (chapter 2.2).

In addition, the Hyperparameter Tuning component uses the information about the exact set of values of input parameters that characterize the selected algorithm for best performance. This information stored in the Analytics Repository is retrieved and points out possible configurations for Bayesian Optimization to initiate the iterative process. This technique also known as warm starting, has been researched to improve efficiency of the Bayesian optimization algorithm and generally leads to a faster convergence in terms of iterations [9].

Chapter 5 - Experimentation

5.1 Experimentation Setup

An experimental study has been conducted to demonstrate the advantages of AutoClust using real-life datasets. As already mentioned, the framework is implemented in Python. This chapter covers all of the technical details of the experimentation such as the datasets used, list of indices for clustering validation, necessary mathematical formulations and so on.

Datasets

To create an experimentation setting that better reflects real life applications and justifies design choices a variety of real-life datasets was used. Two different collections of datasets were used; The first one will be referred to as *group A* and consists of 45 datasets, 40 of which were downloaded from the UCI Machine Learning Repository, while the other 5 from an open to the public collaborative dataset repository namely Data World. The second one referred to as *group B* consists of 25 datasets that include information about the true clustering of the dataset instances. This information allows for various testing purposes as a more accurate picture of the framework's performance can be depicted. The small size of this collection can be considered as a drawback when creating the Meta Learner. Despite the previous mentions these datasets can be further explored to have a more generalized picture on the experimentation scenarios of this chapter. All of the datasets were pre-processed according to the techniques mentioned in chapter 4.1 to be transformed as more appropriate input for the task of clustering. The datasets used are listed as follows:

Group A:

UCI: Absenteism_at_work.txt, ae_train, buddymove_holidayiq, c1r4r_01, c1r4r_02, c1r5r_01, c1r5r_02, c1r6r_01, c1r6r_02, c1r7r_01, c1r7r_02, Frogs_MFCCs, gesture_phase_a2_raw, gesture_phase_b3_raw, gesture_phase_c3_raw, HTRU_2, l1n_01, l1n_02, l1n_03, l1_04, l1_05, l1r_01, l1r_02, l1r_03, l1r_04, l1r_05, LG_G-Watch_1, movement_libras_1, movement_libras_5, movement_libras_8, movement_libras_10, mturk_cluster_data, mturk_data_feature, perfume_dataset,

Sales_Transactions_Dataset_Weekly, SCADI, seeds_dataset, turkiye-student-evaluation-generic, Wholesale_20customers_20data.

Data World: Indian_Premier_League, Customer_Segmentation, Historical_Public_Debt, Asia_Economic_Outlook, Pokemon_Stats.

Group B:

UCI: Arrythmia, Australian_Credit_approval, balance, breast_cancer, cpu, dermatology, e_coli, german, glass, Haberman, heart-statlog, iono, iris, letter, segment, sonar, tae, thy, vehicle, vowel, wdbc, wine, wisc, yeast, zoo.

Algorithms

As already mentioned, the focus is on a specific Machine Learning task (clustering). The certain clustering algorithms that are included at the time of writing are eight, those provided by the Scikit-learn library (version 0.22.1) [14], which is a library used quite extensively by many data scientists. In particular, the clustering algorithms used in the evaluation procedure are: *AffinityPropagation, K-means, SpectralClustering, Agglomerative, DBSCAN, Optics, MeanShift* and *Birch* and the reader is referenced to the library's documentation for further details on the implementation of each algorithm.

Exhaustive Search Configuration Space

The configuration space of the Exhaustive Search process mentioned throughout this thesis is defined as follows: When conducting Random Search, the number of configurations explored for each Dataset, is set to 20 random for each of the available algorithms. For Grid Search the parametric space explored, consists of all possible combinations of values from defined subspaces of values appropriate for the input parameters of each algorithm. For real valued parameters this set includes values that increment with steps of 0.1 while for integers the subspace consists of values incrementing with a step of 1. For example, when running DBSCAN the parameters to be defined are the min pts and eps. For the first one the subspace will be set to [2, 3, …,10] and eps [0.1, 0.2, … ,0.99] and Grid Search will explore all possible pairs from the product of these lists. As for the number of clusters (k) that is a required parameter for some of the clustering algorithms the number of k to search varies from 2 to 30.

Metrics

For the evaluation of Algorithm Selection two main measures are used. The first one is accuracy that refers to the ability of the Meta Learner, or in general any classification algorithm, to suggest the algorithm that performed best on the dataset at hand. The other metric is *Spearman's Rank Correlation* Coefficient and is used to assess similarity of the ranking of two sets [22]. This metric is used to evaluate prediction of the rankings of algorithms performance for the dataset at hand, a technique previously used in Meta Learning Systems, to enable systems that make use of information on more than just the suggestion of the best Algorithm. Definition of *Spearman's Rank Correlation* is as follows:

$$SRC(rr, ir) = 1 - \frac{\sum_{i=1}^{p}(rr_i - ri_i)^2}{p^3 - p} \quad (5.1)$$

Where *rr* and *ri* are the rankings for the two sets and *p* is the number of rank positions. This metric produces values in the range [-1,1] where higher values indicate more similarity *(1 indicates equal rankings, -1 exact opposite rankings).*

The best algorithm or ranking of algorithms is obtained offline by brute-force evaluation of all clustering algorithms. The evaluation of clustering is a long-researched topic, and various cluster quality (validity) indexes have been proposed and are used in practice, without a clear winner. If information about the true clustering of the instances is available as it's the case for group B of datasets, one of the most used and effective quality measures is that of Adjusted Rand Index, which can be used to measure clustering similarity [23].

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_i \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \quad (5.2)$$

This kind of information is rarely available and as result a wide variety of internal indices has been researched and used before that measure either instance similarity inside a cluster and dissimilarity between clusters or measure the or the density of the instances. Five of the most known and prominent indices along with a liner combination of the are used and are described as follows:

<u>Silhouette Coefficient (SL):</u> Validates the clustering performance based on the pairwise difference of between and within-cluster distances. Higher Values indicate better Clustering [24].

$$\frac{1}{NC}\sum_i \{\frac{1}{ni}\sum_{x \in c_i} \frac{b(x) - a(x)}{max\,[b(x), a(x)]}\} \quad (5.3)$$

Calinski - Harabasz Index (CH): Evaluates the cluster validity based on the average between and within cluster sum of squares. Higher values indicate better clustering [2].

$$\frac{\sum_i \ n_i d^2(c_i, c) \ /(NC-1)}{\sum_i \quad \sum_{x \in C_i} \quad d^2(x, c_i)(n-NC)} \quad (5.4)$$

Davies – Bouldin Index (DB): For each cluster C, the similarities between C and all other clusters are computed, and the highest value is assigned to C as its cluster similarity. Then the DB index can be obtained by averaging all the cluster similarities. Smaller values of this index indicate better clustering [2].

$$\frac{1}{NC}\sum_i \quad max_{j \neq i}\left\{\frac{\left[\frac{1}{n_i}\sum_{x \in C_i} \quad d(x, c_i) + \frac{1}{n_j}\sum_{x \in C_i} \quad d\left(x, C_j\right)\right]}{d\left(c_i, c_j\right)}\right\} \quad (5.5)$$

Dunn Index (Dunn): Quality of clustering is measured in terms of the ratio of the maximum distance of points d_{max} within a cluster to the minimal distance d_{min} between points of the dataset. Higher values indicate better clusterings [2].

$$C = \frac{d_{min}}{d_{max}} \quad (5.5)$$

CDBW: A cluster validity method with the use of multi-representatives. Appropriate for datasets of arbitrary geometry. Higher values indicate better clusterings [25].

$$CDbw = Cohesion(c)SC(c), \qquad c > 1 \quad (5.6)$$

Where:

$$SC = \frac{\frac{1}{c}\sum_{i=1}^{c} \min_{j=1\ldots C, i \neq j}\{Dist\left(C_i, C_j\right)\}}{1 + \frac{1}{c}\sum_{i=1}^{c} \max_{j=1\ldots k, j \neq i}\{Dens\left(C_i, C_j\right)\}} \quad (5.7)$$

$$Cohesion(c) = \frac{\sum_s Intra\_dens(C,s)\,/n_s}{1 + \frac{\sum_{i=1\ldots n_s} |Intra\_dens(C,s_i) - Intra\_dens(C,s_{i-1})|}{(n_s - 1)}} \quad (5.8)$$

Where:

$$Intra_{dens(C,S)} = \frac{\frac{1}{r}\sum_{i=1}^{c}\sum_{j=1}^{r} cardinality(v_{ij})}{c * stdev} \quad (5.9)$$

S_Dbw: The S_Dbw index is based on the intra-cluster variance and density between clusters. It is defined as the sum of the mean dispersion in the clusters S and density *G* between cluster the clusters. Lower values indicate better clustering [26].

$$S_{Dbw(c)} = Scat(c) + Dens\_bw(c) \quad (5.10)$$

Where:

$$Dens_{bw(c)} = \frac{1}{c(c-1)}\sum_{i=1}^{c}\left(\sum_{j=1, i\neq j}^{c}\frac{density(u_{ij})}{\max\{density(v_i), density(v_j)\}}\right), \quad (5.11)$$

$$Scat(c) = \frac{\frac{1}{c}\|\sigma(v_i)\|}{\|\sigma(S)\|} \quad (5.12)$$

Composite Score (CS): A linear combination of the 6 indexes presented above, after scaling of them in [0,1] range. Higher values indicate better clustering.

$$CompScore = (SL.Scaled + CH.Scaled - DB.Scaled + Dunn.Scaled + CDBW.Scaled - SDbw.Scaled)/3$$

The experimental methodology that defines the rest of this chapter is as follows:

- 5.2: The first experiments demonstrate empirically the advantages of Random Search over Grid Search to justify its inclusion when producing the Exhaustive Search results and prove it's efficiency as an alternative method for parameter tuning.
- 5.3: This set of experiments explore the properties of the Cluster Validation Indices. First, they ranked when compared to External Validation Indices and then it is attempted to reproduce the work of Muravyov et al [19] to use Meta Learning on selecting the best Internal CVI to capture the performance of an algorithm for a dataset at hand.
- 5.4 In this section is addressed the ability of AutoClust to recommend a subset of algorithms; either a single solution or provide an estimation of the ranking performance of Clustering algorithms for the dataset at hand.
- 5.5: Furthermore, the quality of Bayesian Optimization as a Hyperparameter Tuning technique is assessed both in terms of Performance and Execution Time.

5.2 Random Search in Analytics Repository

The inclusion of Random Search in the Exhaustive Search results stems from the inherent ability of the method to achieve better results compared to Grid Search in certain cases. When the number of input parameters that account for the optimization of a function is relatively small, either because the respective algorithm requires the definition only of a few or because only a handful of them are responsible for the variance of an objective function Figure 3, Random Search is able to achieve better results most of the times in a less computationally intensive manner [27].Strengthening the results of the exhaustive search has great impact in indicating the best performing Algorithm from the Exhaustive Search results that in order affects the performance of Algorithm Selection and the Warm-Starting of Bayesian Optimization. Through the experiments performed and presented in this section enough empirical evidence is provided on the efficiency of Random Search over Grid Search and why those can be used complementary to each other to create a strong baseline of Exhaustive Results.

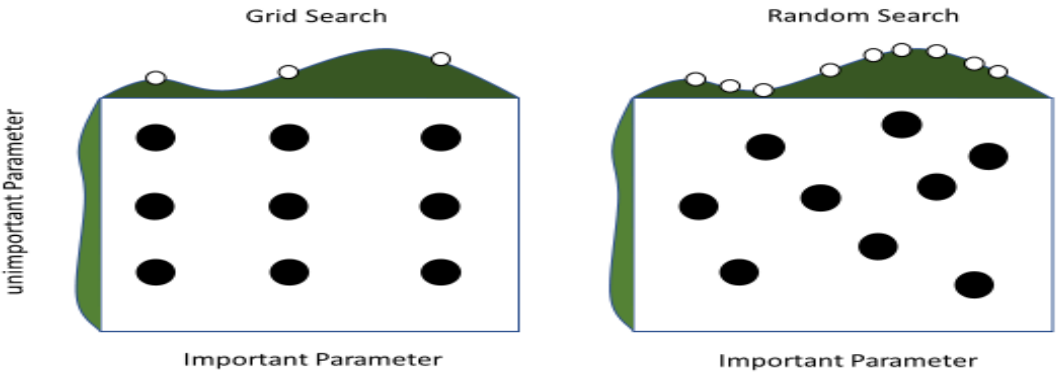


Figure 3 Grid Search and Random Search for 9 different configuration explorations. Through this simple example it is demonstrated that Random Search explores more efficiently a configuration space when only a few of the parameters to be optimized play a major role.

The first experiment of this group of experiments is conducted on group-B of datasets that external information about the true clustering of the data is available. As the Exhaustive search procedure is executed on each dataset of this collection of datasets the respective evaluation results were stored. The two methods results are presented in table 8 in terms of performance and execution time for a set of configurations explored. Note that for Random Search the configuration evaluations for each dataset is less than 140 as described in the experimentation setup section *(20 trials for 20 parametric settings of each algorithms)* due to failures during the execution of the process.

Method	**GridSearch**	**Random Search**
No Wins	21	4
Execution Time (in Hours)	17.56	2.42
Set of Configurations Explored	874	119

Table 8 Results out of Exhaustive Search on group B of Datasets

Table 8 shows the overall performance of both Grid and Random Search over the collection of datasets of group-B. For almost less than half the hours and configurations explored Random Search is still able to outperform Grid Search for 4/25 datasets as of finding the best performing configuration in terms of Adjusted Rand Index (ARI). Results of Table 9 are produced by running Grid Search and Random Search to optimize the input parameters of seven different algorithms for the ML task of clustering for a given dataset as a sample case to further distinguishing the properties of each. In 3 out of 7 cases Random Search is producing better results in terms of the Silhouette Coefficient. Those cases include a few input parameters to be specified, some of them being real-valued.

MODEL	GS_Parameters Suggested	SC (Grid Search)	RS_ParametersSuggested	SC (Random Search)	RS>Gs
Affinity Propagation	damping: 0.5	0.3032	damping: 0.9222286448558172	0.3550	True
Kmeans	N_clusters: 3	0.5857	N_clusters: 2	0.5056	False
Spectral Clustering	N_clusters: 2, Gamma: 0.5	0.5741	N_clusters:2 Gamma: 1.407633	0.5056	False
Agglomerative	N_clusters:3, Affinity: Euclidean, Linkage: Ward	0.58524	N_clusters:2, Affinity: Cosine, Linkage: linkage	0.50555	False
DBSCAN	Eps: 0.2, Min_samples: 5	0.36372	Eps: 0.41511246, Min_samples: 7	0.58526	True
Optics	Cluster_method: "xi", min_samples: 6	0.11201	Cluster_method: dbscan, min_samples: 2	0	False
Birch	N_clusters: 3, Threshold: 0.3	0.58524	N_clusters:210, Threshold: 0.361893	0.58636	True

Table 9 Comparison of the two methods for a single dataset (In terms of Silhouette Coefficient) that comprise Exhaustive Search in the Analytics Repository

.

5.3 Cluster Validation Indices

Selecting the most appropriate Clustering Validation index for result evaluation is a tedious task, with no single solution across different problem instances. Although no evidence suggests of a single best clustering validation index to use, certain researches address the properties of each one [28] and compare them in terms of certain characteristics such as Noise in Data, Monotonicity, Density etc. [29] Most indices define quality in terms of *compactness* and *separation*, two measures that identify similarity between objects in the same cluster and dissimilarity for objects in separate clusters. On the other hand, real life problems may include data that are not naturally partitioned according to those terms but are represented by arbitrary geometries, are very unstructured, include noisy observations and are represented by different densities. For that reason, new indices have been developed such that of *CDBW* and *DBCV* [25] [26] two validation indices that address some of the problems already mentioned appropriate for evaluation of clusters of arbitrary geometries.

5.3.1 Empirical Evidence on the Generalization of Cluster Validation Indices

Goal of the experiment presented in this subsection, is to provide empirical results on the ability of a set of CVIs to indicate the best performing algorithm for different types of Data Sets; the question arises in the form of "*which index when optimized indicates the configuration that holds the best results in terms of external validation"*. This information although perceived according to the data sets used for experimentation, can provide significant insights on the generalization power of metrics.

Similar to the Learning phase of *AutoClust,* defined in Chapter 3, a set of 25 datasets drawn from the *UCI Machine Learning Repository (group B),* are exhaustively searched in terms of different configurations *(Clustering Algorithm, Input Parameter Values)* via the *GridSearch* and *RandomSearch* methods. In order to examine the performance of the indices, for each dataset 7 different configurations are selected from the Exhaustive Search results as indicated by the optimization of every available index. The configurations selected produce a clustering that can be compared to the true clustering of the instances. To measure this relation *Adjusted Rand Index (ARI)* was used as described in the setup section. In turn this value of *ARI* is compared to the maximum *ARI* across all the configurations searched for the specific dataset. This information reveals the ability of an index on indicating a good performing configuration or to quote differently, the ability of an index to indicate a configuration that produces an ARI value as high as possible when compared to the max ARI found with exhaustive configuration exploration. The higher the value of this subtraction the less indicative is a specific CVI for a dataset. To define a metric that corresponds to the generalization of this information on ARI difference across several problem instances and CVI's a measure is formulated as follows:

$$ARI_{DIFF_{CVI_1}} = (\sum_d ARI_{optimal-cvi} - ARI_{max})/N \text{ , (5.1)}$$

$$for\ dataset\ d\ \in D, N: number\ of\ datasets$$

CVI	Ari_diff
Calinski Harabasz	0.19718

Silhouette	0.32214
Composite Score	0.40484
Davies_bouldin	0.41188
Dunn	0.43555
SDbw	0.43856
CDBW	0.47552

Table 10 Results for the proposed measure of CVI generalizationpresents the results of this experiment and the ranking each CVI achieved. It can be perceived that indices like Silhouette that are of great popularity cannot be used as a universal solution. In addition SDbw and CDBW fail to generalize well. On the other hand, the Calinski Harabasz index proved to generalize adequately for the collection of datasets at hand. The optimization of this CVI provided in general configurations that differed from the optimal solutions around 0.2 in terms of Adjusted Rand Index.

Although this observation is linked to the choice of datasets used, it can provide guidelines for several choices to be made while developing or using *AutoClust*. Essentially as the number of datasets grows results will tend to be more concrete as more empirical evidence will be collected on which set of CVI's generalizes better across instance problems already seen. If this information is available and no other properties of a dataset can be extracted, the best performing indices as indicated by this experiment can be selected to indicate the best performing algorithm or set the optimization goal for Hyperparameter Tuning. This process is to simulate, the choice of a Machine Learning practitioner for an index to evaluate a clustering, based on experience. This information is updated as new problems are seen allowing for constant improvement. The question that naturally arises after the conclusion of this experiment, is if the selection of CVI to validate a clustering schema can be related to a set of characteristics of the data set, which leads us to the next section *Meta Learning for CVI Selection.*

Table 10 Results for the proposed measure of CVI generalization

5.3.2 Meta Learning for CVI Selection

The second experiment aims to reproduce the meta learning technique described by Muravyov et al. [19] to further improve the selection of CVI for a problem instance. Experimentation is again conducted on

group B of datasets as they hold information about the true instance clusterings. Each dataset is labeled with the internal validation metric that when optimized by the combination of GridSearch and RandomSearch methods produced the best resulting *ARI* and will be the target variable for classification of the Meta Learner. In addition, the set of Meta Features described in the architecture section is computed for each to create a vector of predictor variables. To capture the relationship of the Meta Features and the index that best fits to indicate performance on a given dataset, a variety of the most prominent classification algorithms has been used as implemented in the Scikit-Learn. Evaluation of the classification algorithms' performance has been measured in terms of the mean accuracy of a Leave-One-Out Validation method to accommodate for the small size of the instances.

Model – Meta Learner	*Accuracy (Leave One Out Validation)*
Random Forests	0.4
Multilayer Perceptron	0.32
Decision Tree	0.24
KNN	0.28

Table 11 Results of selecting the best performing CVI via Meta Learning

As seen in Table 11 none of the Algorithms provides any evidence that Meta Learning can be used to identify the most appropriate index to use. These results may differ from the respective research due to the difference in collections of datasets used for experimentation, the labeling method of the true instance clusterings and the different set of Meta Features for the characterization of the problem instances. Although the accuracy achieved with the Random Forests Classifier is higher than this of attributed to chance *(1/6, where 6 is the number of indices)* results are not considered satisfactory to be taken into consideration when choosing for index as target for the optimization procedure. As a result, the index with the best overall performance among all instances of problems, as presented in the previous section is to be selected when the question of which index to use arises.

5.4 Evaluation of Algorithm Selection

Goal of this set of experiments is to evaluate Algorithms Selection performance when suggesting the best performing Clustering algorithm or predicting the ranking of the algorithms in terms of performance for new unseen datasets.

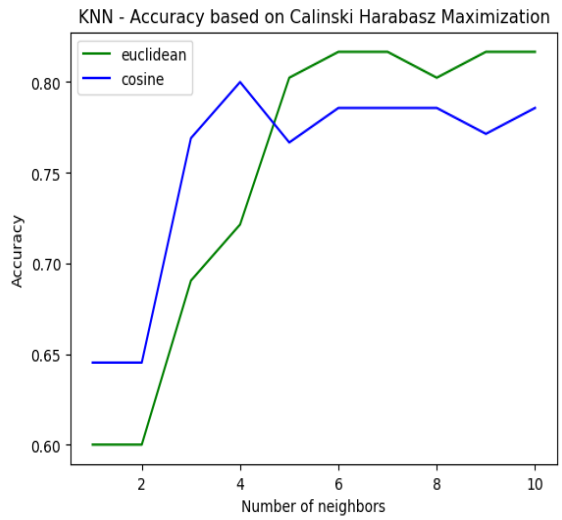
5.4.1 Prediction of the best Performing Algorithm

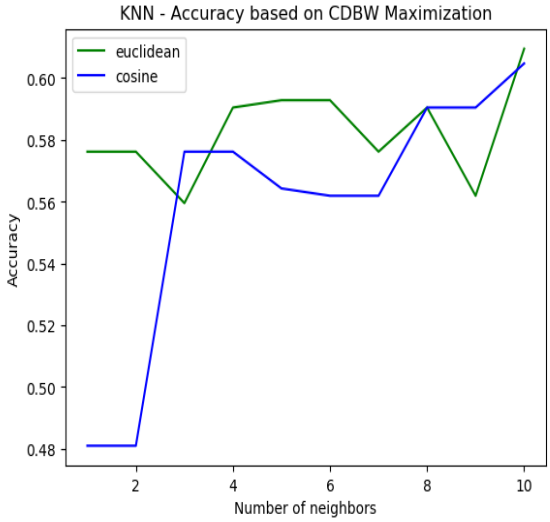
In this experiment both groups of datasets are used together to form a collection of 70 datasets. Each dataset of the collection was used to extract meta-features, which were stored in the MetaFeaturesDB. Then, all clustering algorithms were executed over these datasets for a variety of configurations as defined by the Random Search and Grid Search parametric spaces of exploration. The generated clusterings are evaluated using six different cluster quality indexes (SL, CH, DB, SDbw, CDBW, Dunn's Index), in order to obtain the best performing clustering algorithm for each dataset. Thus, the best performing algorithm can be selected that serves as ground truth for the Algorithm Selection problem based on different cluster quality indexes. In addition, a linear combination of the six indices already mentioned referred to as Composite Score (CS) is used.

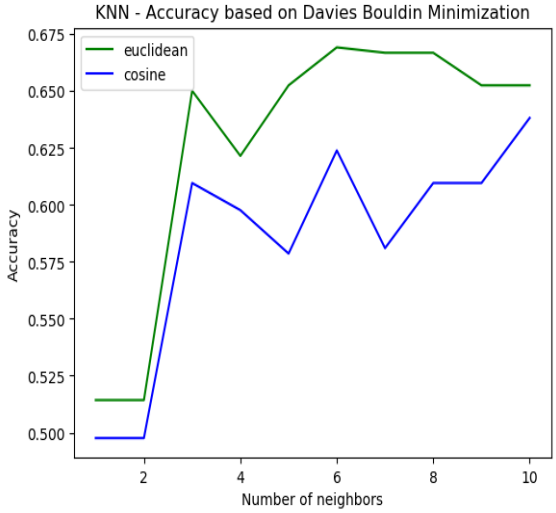
To build a Meta Learner a k-nearest neighbor (kNN) classifier is selected in order to find the k most similar datasets in the training set. Similarity between Meta Features is computed in two different ways: using the Euclidean distance or using the Cosine similarity. Values of k vary from 1 to 10 and the algorithms that performed best for these k nearest neighbors (datasets) are obtained. Then, to merge the suggestions of the k nearest neighbors, a majority voting technique is implemented to define one single solution, i.e., select the algorithm that performed best in most of the k datasets.

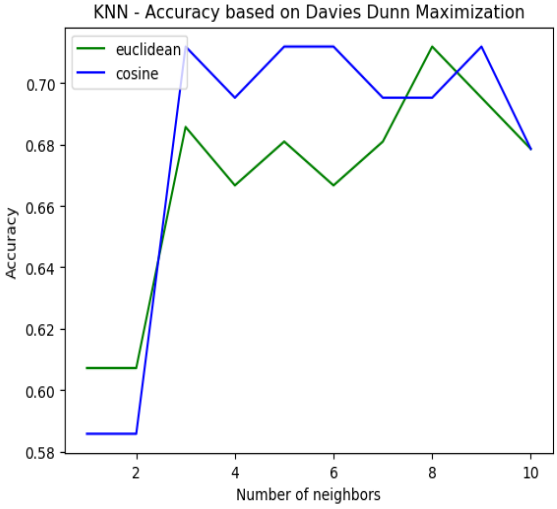
Evaluation of the process is conducted with 10-fold Cross Validation to account for generalization of the model. The figures that follow present the mean accuracy of the 10-fold Cross Validation of model selection for different cluster validity indices: SL, CH, DB, SDbw, CDBW, Dunn's Index and CS. Also, the charts depict the obtained accuracy when using the Euclidean distance and the Cosine similarity. A general trend can be observed, namely that accuracy increases for higher values of k, although in some cases the accuracy drops. This is due to the fact that higher values of k return more datasets that are deemed similar to the one at hand, and (consequently) more algorithms are returned as candidates for selection. When the same algorithm is returned many times, this is strong evidence that it is suitable for the dataset at hand, and this is reflected in the increased accuracy values. However, in some cases (e.g., from k=1 to k=2), two different candidate algorithms are returned, and then the selection is practically random, which may cause a decrease in accuracy values.

When comparing the absolute accuracy values obtained, we observe that the use of SDbw (accuracy 91%) and CH (accuracy 82%) result in the highest values. This means that when we perform model selection based on these cluster quality indexes, we obtain higher accuracy. The SL metric performs worse than all others do, and the rest of the methods are in between these two extremes.


KNN - Accuracy based on Calinski Harabasz Maximization
euclidean
cosine
Accuracy
Number of neighbors


KNN - Accuracy based on CDBW Maximization
euclidean
cosine
Accuracy
Number of neighbors


KNN - Accuracy based on Davies Bouldin Minimization
euclidean
cosine
Accuracy
Number of neighbors


KNN - Accuracy based on Davies Dunn Maximization
euclidean
cosine
Accuracy
Number of neighbors

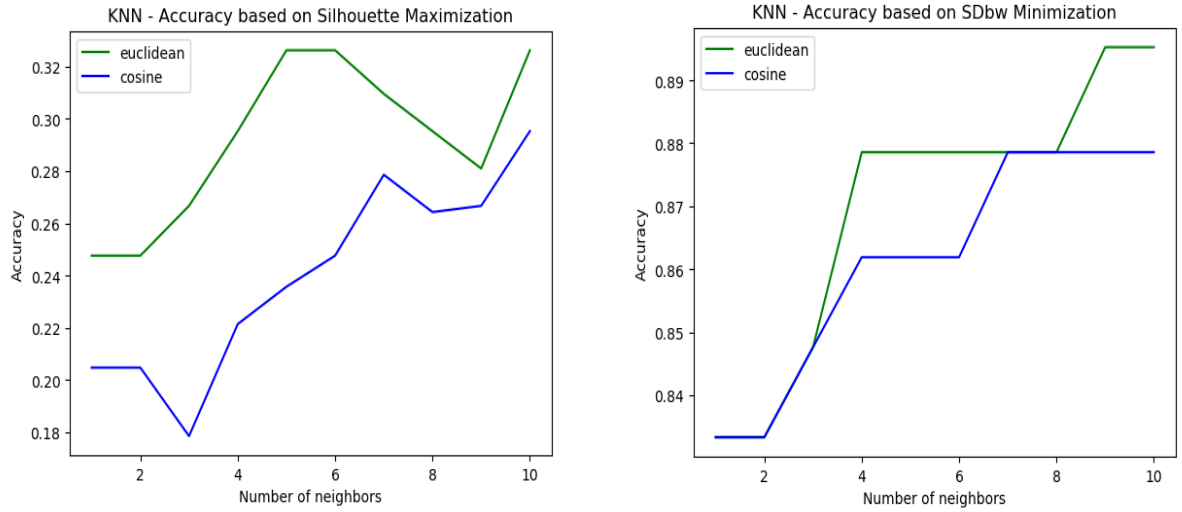


Figure 4 Algorithm Selection Evaluation on a set of 6 Internal Indices

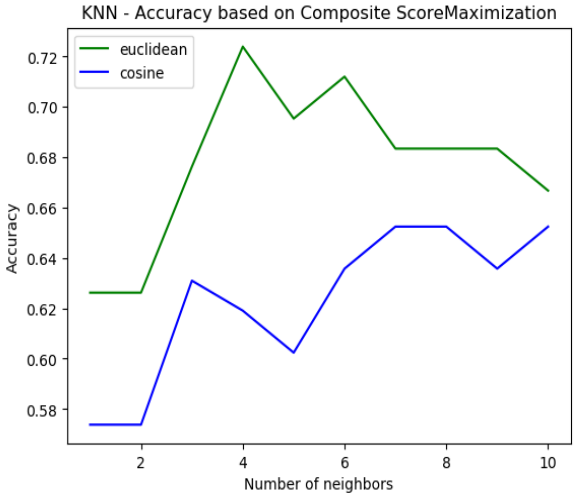


Figure 5 Algorithm Selection Evaluation on a Score that is a linear representation of the 6 previously mentioned

When considering the two alternative similarity measures (Euclidean and Cosine), we observe that only in the case of Dunn Index, there is a clear winner, namely Cosine. In the other cases, the results are mixed, although in most cases the use of the Euclidean distance returns higher accuracy values.

Figure 6 shows how accuracy is affected when the Algorithm Selection process returns the top-N algorithms (1, 2 or 3 algorithms), instead of a single algorithm. This is a common practice for evaluation in multilabel classification, that addresses the ability of a model to include the right solution in a subset of all the possible answers. As the pool of available algorithms contains 8 algorithms, the accuracy of the

model can be explored when returning 2 or 3 algorithms, since this is always much better (in terms of saving time of the Machine Learning Practitioner) than following a brute-force approach that would execute all 8 algorithms and pick the best performing one. The number of neighbors chosen this time as input parameter for the KNN Meta learner is set to the one that performed best for each individual CVI as indicated by the previous experiment. In addition, distance metric chosen is Euclidean as it proved to be more efficient in general than cosine similarity, again as indicated by the previous experiment.

As can be seen in Figure 6, when using Top-3 accuracy a value as high as 98% is achieved when using Dunn's Index followed by the Calinski-Harabasz Index with 96%. For the majority of the indices, Top-3 accuracy is over 90% *(4 out of 7 indices)* while the rest vary from 63% to 81%. Only for Silhouette seems that although the results improved when measuring Top-2 or Top 3 accuracy, they still fall under 70%. Top 2 accuracies also prove to be adequate and they indicate that while using indices that is harder to select the best performing algorithm the model is more able to include it in top two recommendations.

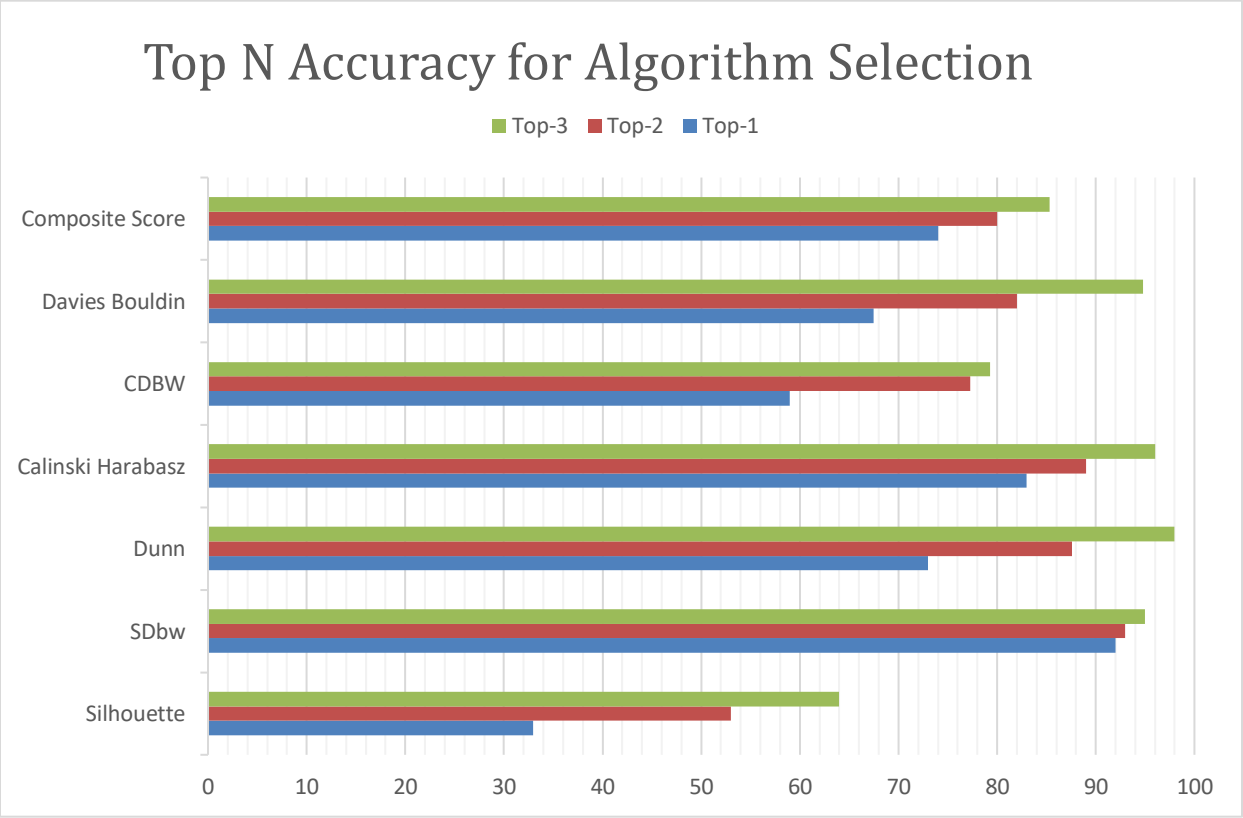


Figure 6 Top-N accuracy of Algorithm Selection, neighbours and distance metric chosen from the best setting presented in figures 4 and 5

In summary, the Algorithm Selection process has been proven to provide adequate results when selecting an algorithm for a dataset at hand. When selecting only one it has achieved high accuracy that of 92% while using the SDbw index. When returning 2 or 3 algorithms the process is able to almost always include

the best performing algorithm in the set of solutions. Accuracy in that manner reaches at most 83 for a subset of two and 98 for a subset of 3.

5.4.2 Algorithm Selection with the use of Novel Meta Features

The experiment in section 4.4.1 evaluated the performance of the Algorithm Selection process on suggesting the best performing algorithm while using only internal indices. An alternate setting to be explored, is if a Meta Builder can be designed to suggest the best performing algorithm as labeled by external information that includes the true clustering of the data instances. This approach has a certain number of advantages over the previous one; The availability of a "ground Truth" allows for better ranking of the Algorithms in terms of performance that better corresponds to real life applications. Furthermore, eliminates the need for certain user input, the CVI to evaluate the algorithms, a choice which is subjective and related to the practitioner's experience and a priori knowledge for a dataset. The challenge that emerges in this approach is the difficulty in finding datasets that include their true clusterings, while also checking their validity to build the Meta-Database.

The methodology for this experiment is as follows: For a set of 25 datasets that information about their true clusterings is available each of the algorithms for examination is executed for a variety of parametric configurations, similar to the Exhaustive Search procedure described in chapter 3.1. The results of these executions are stored both in terms of internal (SL, CH, DB, SDbw, CDBW, Dunn's Index), and external (ARI) validity indices. Afterwards, for each dataset the available algorithms are ranked in correspondence to ARI performance as explored by the Exhaustive Search. In addition, a set of Meta Features is computed for each dataset to form a set of vectors that later are used to asses similarity between the datasets. Those Meta Features stem from Statistics, Information Theory, the distance distribution between the dataset's instances and by landmarking. First the model is evaluated on its ability to predict a ranking of the algorithms performance for each dataset. The most similar dataset to the one at hand suggests the performance ranking. To evaluate the similarity of the two ordered sets the *Spearman's Rank Correlation Coefficient* (SRC) is used as previously described in the Experimentation Setup chapter. Afterwards the model is evaluated on its performance on suggesting a single algorithm. The strategy for evaluation is Leave One Out, which is practically Cross Validation with the number of Folds set equal to the number of instances. This strategy better suits the low number of instances that are under analysis (25 Meta Feature vectors) while also allows to take the models ability of generalization as a factor.

The results presented in table 10 assess the ability of the model to predict the correct performance ranking of clustering algorithms for the new dataset at hand. To gain a better understanding the last two rows present the respective results by always predicting the average ranking of the datasets and suggesting the ranking of another dataset from the collection at random. To define similarity of the datasets two sets of Meta Features have been tested. The first one is suggested by Ferrari et Castro Table 6 and the second one is a novel idea that uses the internal indices of a MeanShift execution. From the SRC it can be observed that predicting the exact ranking of Algorithms is a tedious task, although from using the MeanShift LandMarking Meta Features a 0.43 value of SRC is reported improving from

suggesting random 0.26 or the mean of rankings from the database 0.31. the MeanShift Meta Features improve performance from the Ferrari et Castro by 0.05.

MetaFeatures	**Meta-Learner**	**Datasets**	**Evaluation Methodology**	**SRC (mean of the LOO Validation Process)**
Hybrid-Ferrari	**KNN** (**distance**: *Euclidean*, **Neighbors**:*1*)	25 UCI with External Validation	Leave One Out	0.3699802355238522
Attribute Based Ferrari	**KNN** (**distance**: *Euclidean*, **Neighbors**:*1*)	25 UCI with External Validation	Leave One Out	0.35649196614406614
Distance Based Ferrari	**KNN** (**distance**: *Euclidean*, **Neighbors**:*1*)	25 UCI with External Validation	Leave One Out	0.32332806070179265
Meanshift LandMarking	**KNN** (**distance**: *Euclidean*, **Neighbors**:*1*)	25 UCI with External Validation	Leave One Out	0.4310515873015874
None	**Random Recommendation**	25 UCI with External Validation	Average of 10 LOO	0.26988558087113823
None	**Mean Rankings Recommended**	25 UCI with External Validation	LOO	0.31484856539838746

Table 12 Similarity of Algorithms Performance between datasets

The following diagrams, figures 1 through 3, present the accuracy achieved when selecting one algorithm for the dataset at hand. The model is also evaluated in the ability to include the right solution in a set of algorithms selected, measured by Top 1,2 and 3 accuracy, a common practice in multilabel classification. The new Meta Features presented, that stem from the MeanShift Landmarking method prove to be very competing to the ones already suggested by Ferrari et Castro but not exceeding them in accuracy. A hybrid version of the two sets of Meta Features on the other hand achieves the best Top-3 accuracy of 98 %. And ties with the use solely of the Ferrari Meta Features for Top -1 by 57% accuracy. For the Top-2 accuracy a value of 75% is achieved with the clear winner being the Ferrari Meta Features. When Compared to the results of table 10, it can be observed that although ranking similarity in MeanShift Landmarking Meta Features was better the prediction of the best performing algorithm is better achieved with either the Ferrari Meta Features or a hybrid version of the last two.

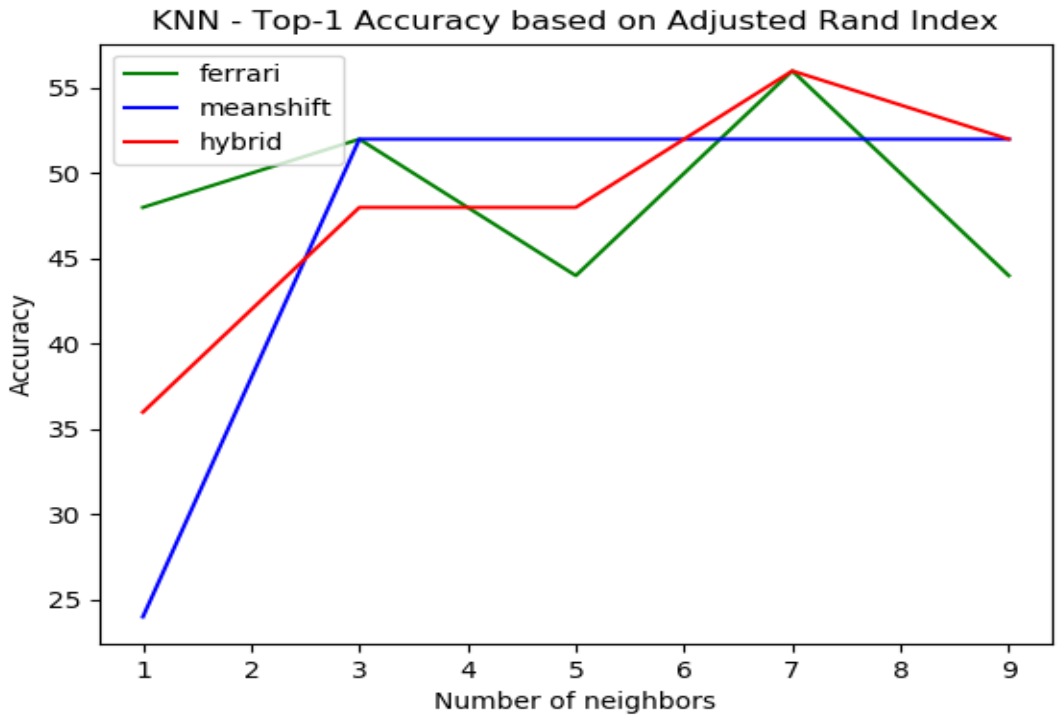


Figure 7 Top-1 Accuracy of the Meta Learner built upon Group-B of datasets

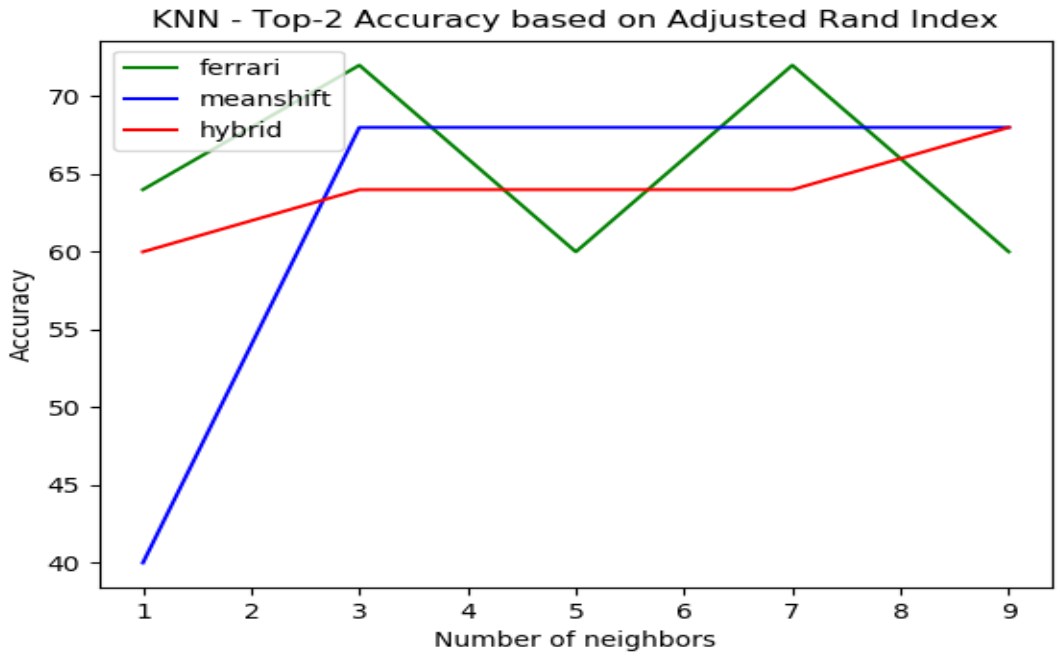


Figure 8 Top-2 accuracy for the Meta Learner built upon Group-B of datasets

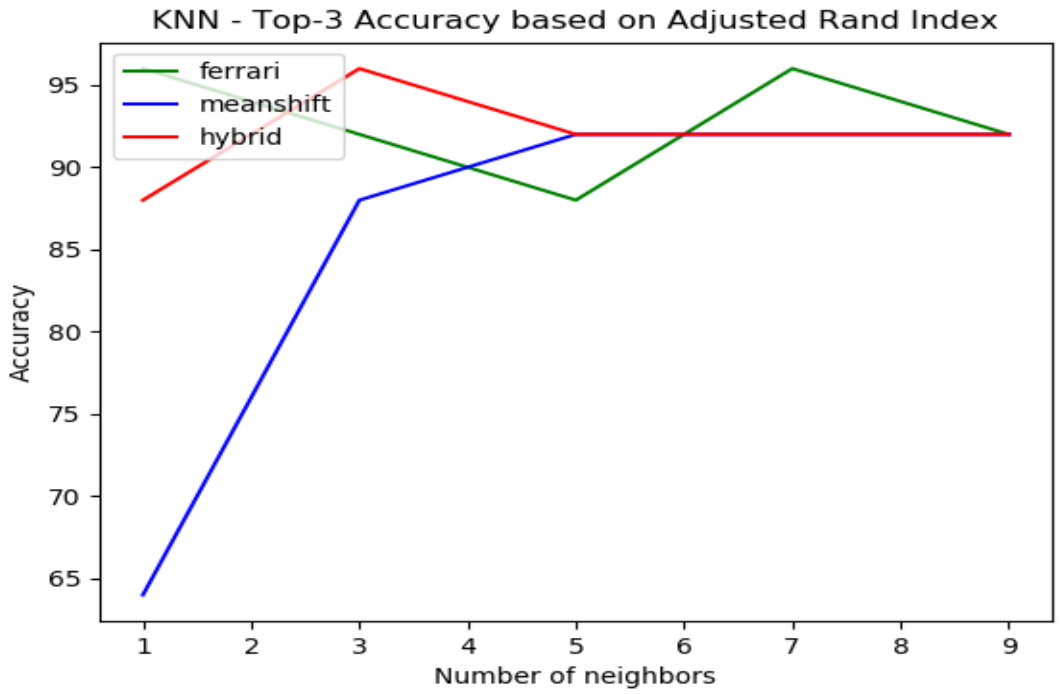


Figure 9 Top-3 accuracy of the Meta Learner built upon Group-B of datasets

5.5 Evaluation of Hyperparameter Tuning

Although the evaluation of Bayesian Optimization as a Hyperparameter Tuning method has already been researched in various previous works, in this experiment it will be compared to the results obtained by Exhaustive Search (ES), using: (a) the number of times that our selection led to better results compared to the best achieved in ES Results and (b) the deviation of the rest from the best achieved in ES Results. Deviation is measured using the Mean Absolute Error (MAE), which is the absolute difference of the scoring of the model suggested after Bayesian Optimization from the best score obtained by Exhaustive Search, formulated as follows.

$$MAE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}} |y_i - \hat{y}_i| \quad (5.13)$$

Low values of the MAE metric indicate that the expected performance of the parameters selected from Hyperparameter Tuning will be approximately close to the best performing parameters of Exhaustive Search. For experimentation purposes, the number of evaluations to performed during Bayesian Optimization is set to seventy (70), for every one of the 25 datasets of GroupB of datasets.

Index	MAE (When Bayesian OPT Results are lower than ES)	No Instances Bayesian OPT outperformed ES
Silhouette Coefficient	0.1134	14/25

Table 13 Evaluation of Bayesian optimization as a Hyperparameter Tuning method

Table 13 demonstrates the ability of Bayesian Optimization to select parameters, that (a) outperform traditional Hyperparameter Tuning methods, or (b) underperform only by a negligible amount compared to the best solutions of ES.

The MAE value 0.1134 can be described as approximately how much lower will be the scoring of the cluster schema selected by our Hyperparameter Tuning compared to the best scoring clustering schema of an offline Exhaustive Search. Taken into consideration the range of Silhouette Coefficient [-1, 1], 0.11 is reasonable difference in evaluation metric, for results produced in a much more timely manner than ES Results. In addition, for 14 out of 25 datasets the Bayesian Optimization method was also able to achieve higher Scorings than ES, showing its capability to outperform Grid Search and Random Search.

Chapter 6

6.1 Proof of Concept

In this chapter will be demonstrated the applicability of AutoClust on a set of 3 of the most known datasets for benchmarking clustering results. The first one is a synthetic 2-dimensional dataset consisting of globular data, a selection to evaluate the ability of the framework to select the right number of clusters for the given data. This dataset consists of 1000 samples that are centered around 5 points as presented in figure 3. To add a layer of complexity samples are set to have a Standard Deviation of 0.3 from each center. The second dataset, again synthetic and 2-Dimensional, known as "Half-Moons" due to its shape, includes data of non-globular geometry. This allows to evaluate on the ability of the framework on finding the right clustering schema when the data geometry is arbitrary. The last dataset to be used is the Iris flower data set. This dataset is a popular dataset for clustering introduced by the British statistician Ronald Fisher in 1936 and consists of 50 samples that correspond to 3 different Iris species (Setosa, Virginica and Versicolor) thus includes the natural clustering of the data. Evaluation on this dataset is to provide a basic understanding on how the framework can handle real-life applications. The exact parametric setup that was used to model the AutoClust procedure is listed in Table 14 Parametric Setup for the execution of AutoClust to be used as Proof of Concept

. The index that is used as an optimization target for the Hyperparameter Tuning is the one that is the most indicative to ARI values for the most similar dataset. From the exhaustive Search results, Spearman's Correlation between ARI and each internal index was calculated and provided insights on the monotonic relationship of the two. The index that had the biggest correlation is considered that when optimized will lead to the better results for the dataset at hand.

AutoClust Parameters

Meta Features Database	Group B Datasets: *25 real life datasets with information on their true clusterings available.*
Meta Features	*As listed in* Table 6*,* Table 7
Meta Learner	KNN: *with majority voting among 7 neighbors selected by Euclidean distance.*
Optimization	Bayesian Optimization: *20 number of evaluations, optimizing the index that had the best Spearmans Correlation for the similar dataset.*

Table 14 Parametric Setup for the execution of AutoClust to be used as Proof of Concept

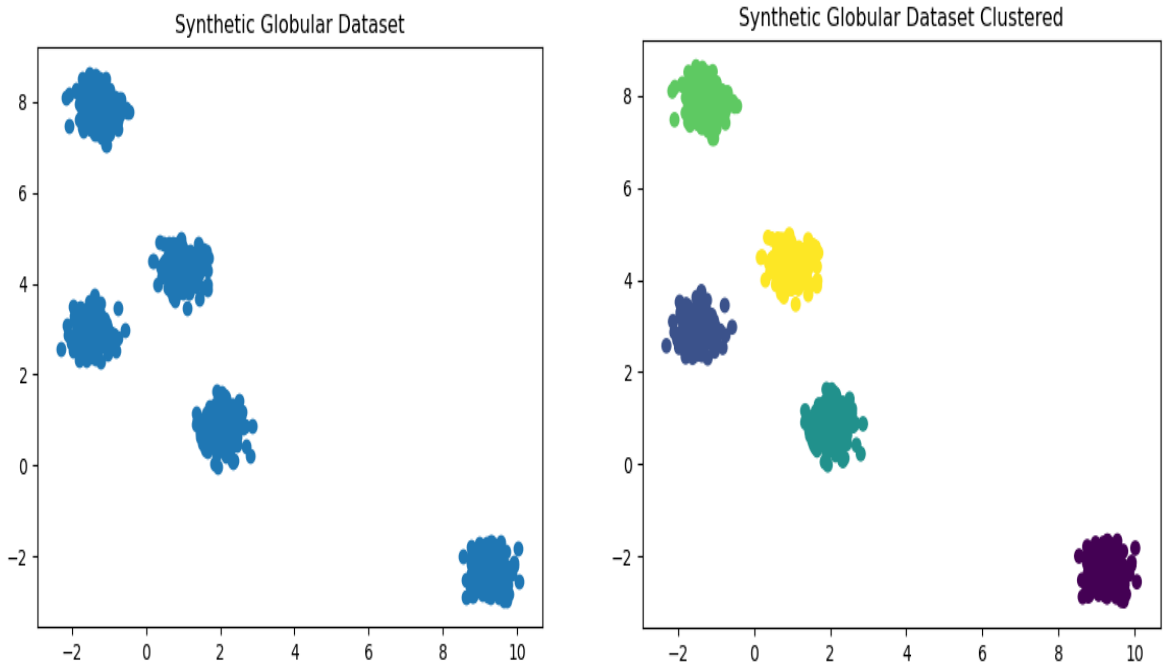


Figure 10 Synthetic 2-Dimensional Data, Data of Globular geometry, clustered by AutoClust

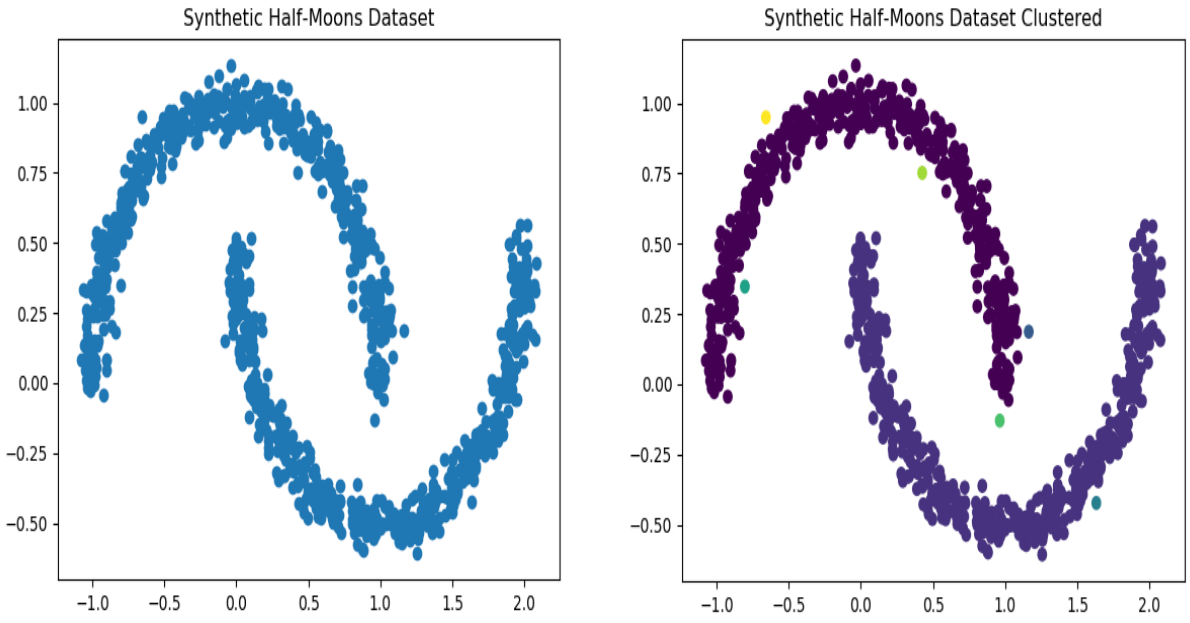
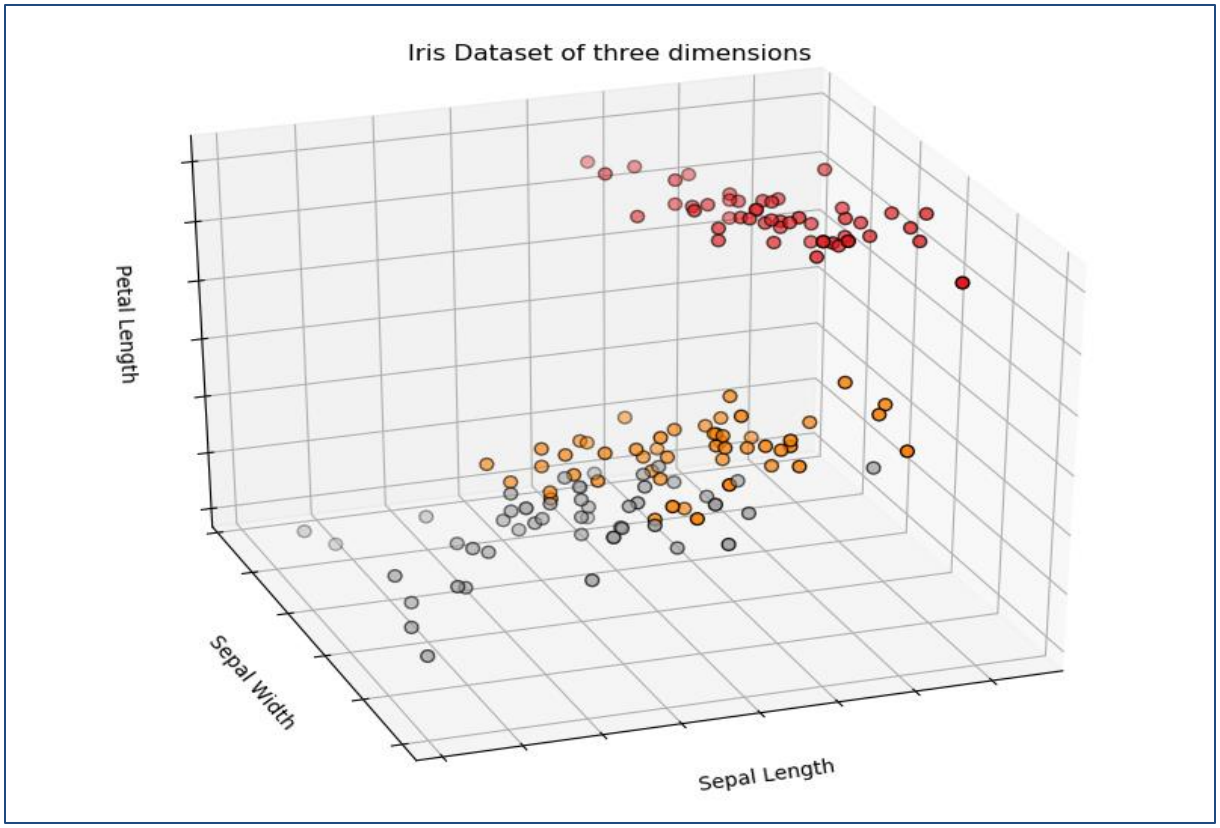


Figure 11 Synthetic 2-Dimensional Half Moons Dataset clustered by AutoClust



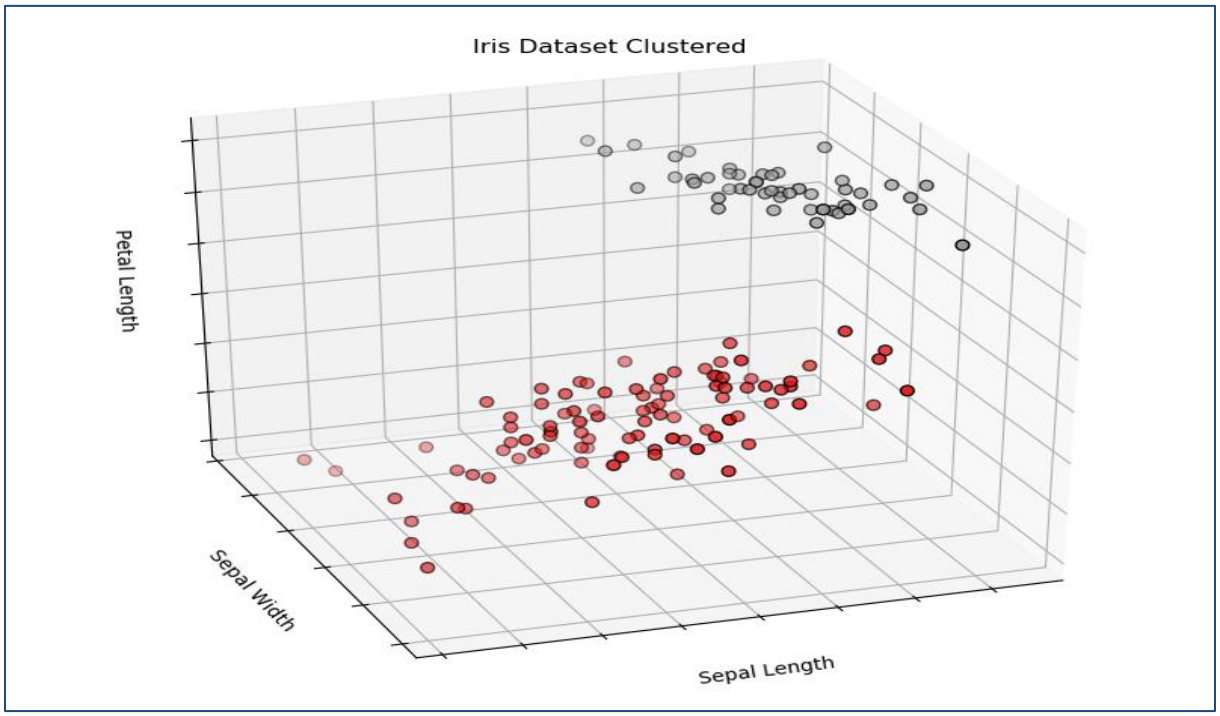


Figure 12 The Iris dataset visualized by 3 of its 4 in total variables, as clustered by AutoClust

Figures 10 to 12 , visually present the datasets along with a colored representation of them according to the clustering provided by running AutoClust with the aforementioned parameters(Table 14). For both the synthetic datasets the suggested framework provides adequate solutions; It finds the right number of clusters for the first one while clustering instances of the second according to its unique “Half-moons” geometric shape. For the Iris dataset the solution included 2 clusters. The one cluster that was not properly indicated was merged with another, a choice that can be justified visually as there is no clear separation from its neighbor in terms of distance. This is something to be attributed to the fact that at the time of writing the framework aims to maximize a given metric and not select the appropriate number for clusters.

6.2 Conclusions

This thesis explored various concepts of Automated Machine Learning, specifically for the task of Clustering. A review of the most prominent AutoML systems has been conducted, that revealed the variety of state-of-the-art techniques that are implemented among them. Clear winner when it comes to optimization, the key-component of those systems, seems to be the Bayesian Optimization method when compared to genetic programming and bandit-based approaches. While all of the systems focus on Supervised Learning tasks such as classification and regression to name a few, clustering is under researched due to its Unsupervised Learning nature. Obstacles arise such as which is the best way to evaluate a clustering schema, a problem that derives from the fact that data instances that are subject to the task are rarely labeled with information about their true clustering, some form of “ground truth”.

In Chapter 3 a proposed architecture is presented as a solution to Automated Machine Learning for Clustering. To provide the user with a configuration setting that consists of an Algorithm and a set of values for its input parameters appropriate for a problem instance, the main methods that are used are Meta Learning and Bayesian Optimization. Both are widely accepted across the AutoML community and can be used together in a complimentary way. Meta Learning not only is a design choice that is flexible but also serves as fast speed up to AutoClust’s Hyperparameter Tuning. Bayesian Optimization takes in turn advantage of information on Exhaustive Search evaluation results of similar datasets to the one at hand and advances the solution in terms of quality in a reliable way.

Design choices of AutoClust are also under the scope of Big Data. Meta Learning is as computationally heavy as the extraction of Meta Features for the dataset at hand combined with a similarity measure computation among two vectors, while provides a solution to Algorithm Selection, which can often be the sole use case for the user. Furthermore, it provides a certain amount of flexibility when providing results,

as suggesting the algorithm of the less computational complexity information that is known a priori. If the solution required, includes a set of Hyperparameters, Bayesian optimization can be initiated with the use of *Apache Spark* and *MongoDB* for parallel evaluations of the procedure allowing for implementation in scalable environments. In addition, certain configurations can be provided as input by the user, such as the number of evaluations for Bayesian optimization, for AutoClust to conclude faster and in a computationally less expensive way to results.

As for results, experimentation on a large collection of real-life datasets, provided enough evidence that the task of Automatic creation of Clustering ML pipelines is a challenging but feasible task. When the user, the ML practitioner, is able to specify certain inputs such as the Cluster Validation Index to evaluate the clustering, the framework is capable of providing a subset that contains the best performing algorithm for the dataset at hand with accuracy as high as 98%. To aid this process of input selection, certain guidelines, have been created based on empirical results, that declare a winner of CVI in terms of generalization among different datasets. Furthermore, an alternative setting has been explored where the user is not required to provide any input other than the data that are subject to clustering. In this setting the Meta Feature Database is created from datasets that include a "ground truth clustering" allowing for the automation of AutoClust end-to-end as it is able to also relate best performing CVIS by dataset similarity.

The Hyperparameter tuning has also been proven to be more efficient than traditional parameter optimization methods. This feature enabled the further customization of the values for input parameters for the Algorithm selected before in a timely manner and adequate results.

Applicability of AutoClust was tested as a proof-of-concept on a set of 3 datasets, very popular for benchmarking clustering results. Each of the datasets allowed for the evaluation of AutoClust from different perspectives. The results provided evidence that the framework is able to automatically cluster the data with a reasonable number of clusters. The arbitrary geometry presented by the Half-Moons Dataset was also identified by the clustering and finally an adequate clustering for Iris was achieved but not with the same number of clusters as dictated by the ground truth.

6.3 Future Work, Open Challenges

The problem of Automated Machine Learning is a complex one and although it has been extensively researched over the past few years, many challenges remain to overcome. AutoML systems need not only to improve in terms of quality but also in execution time. A variety of data are more and more ingested in bigger volumes at a greater velocity, that require new alternative approaches to be handled efficiently by the AutoML procedures. This tradeoff in computation time and quality is a crucial target for optimization, if the systems are to find application in a variety of domains, such as robotics and autonomous driving. For the specific case of clustering that falls under the category of unsupervised learning, the difficulties presented are more compared to supervised, as the term "best clustering" can be subjective, thus requiring different approaches to automate the process. The majority of the processes that were used in AutoClust design can also be individually optimized, complicating the creation of an end-to-end system.

Some of the most major challenges that are presented and dictate future work in order to improve the framework are listed as follows:

- Meta Features, the values used to characterize datasets and consequently define similarity among them, can be chosen from a wide variety of available ones or created anew. As a result, choosing the exact best performing set of Meta Features is very challenging as it differs among problem instances. On top of that the selection of Meta Features to be used should account for computation time.

- The Cluster Validation Index (CVI) to use for clustering evaluation is a very crucial choice with impact on both processes of Algorithm Selection and Hyperparameter Tuning. The fact that no "ground truth" is available makes the clustering evaluation relate on properties that are somewhat subjective. Optimizing a pipeline with some CVI that is not appropriate for the problem instance may lead to false conclusions.

- Bayesian Optimization in the Hyperparameter Tuning, although computationally cheaper and more effective than trial-and-error manual search procedures, remains expensive for big data applications as is still requires dataset evaluations. Speeding up the Hyperparameter Tuning is crucial as it accounts for most of the computation time of the framework, other than extracting Meta Features. Techniques like sampling or alternative optimization methods that parallelize better may lead to a better tradeoff of computation time and quality of results.

- Finally, with the popularity gain of Deep Learning networks, types of data other than numeric *(image, text, sound)* are being more and more explored. In the context of image processing, segmentation by clustering is a very common practice while for text type data, clustering has applications to text identification and information retrieval. For the aforementioned reasons, one of the future work goals is to provide support of these types of data by the AutoClust framework.

References

[1] X. Z. Kaiming He, "Deep Residual Learning for Image Recognition," in *IEEE on Computer Vision and Pattern Recognition*, 2015.

[2] R. Xu, Clustering, IEEE Press, 2008.

[3] H. H. H. F. N. H. T. Pascal Kerschke, "Automated Algorirthm Selection: Survey and Perspectives," Arxiv, 2019.

[4] J. R. Rice, "The Algorithm Selection Problem," *Advances in Computers,* vol. 15, pp. 65-118, 1976.

[5] L. K. J. V. Frank Hutter, AUTOMATED MACHINE LEARNING- Methods, Systems, Challenges, Springer, 2019.

[6] F. H. H. H. H. K. L.-B. Chris Thornton, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," in *KDD*, 2013.

[7] J. Vanschoren, "Meta-Learning: A survey," 2018.

[8] K. S. Z. W. R. P. A. N. d. F. Bobak Sahriari, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," in *IEEE*, 2016.

[9] R. B. Y. B. B. K. Bergstra James, "Algorithms for Hyper-Parameter Optimization," in *NIPS*, 2011.

[10] K. Eggensperger, "Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters," 2013.

[11] A. K. K. E. J. T. S. M. B. F. H. Metthias Feurer, "Efficiend and Robust Automated Machine Learning," in *NIPS*, 2015.

[12] A. N.-M. G. C. A. K. Rich Caruana, "Ensemble Selection from Libraries of Models," in *International Conference of Machine Learning*, 2004.

[13] A. K. M. F. J. T. S. M. U. M. B. M. D. M. L. F. H. Hector Mendoza, "Towards Automatically-Tuned Deep Neural Networks," *Journal of Machine Learning Research,* pp. 1-8, 2016.

[14] P. e. al, "Scikit Learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[15] J. H. M. Randal S. Olson, "TPOT: A Tree-based Pipeline Optimization Tool for Automated Machine Learning," *Journal of Machine Learning Research,* vol. 64, pp. 66-74, 2016.

[16] A. K. V. C. Josip Djolonga, "High-Dimensional Gaussian Process Bandits," in *NIPS*, 2013.

[17] B. S. S. M. G. K. J. K. D. S. Daniel Golovin, "Google Vizier, A service for Black-Box Optimization," in *KDD*, 2017.

[18] L. N. D. C. Daniel Gomes Ferrari, "Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods," *Information Sciences,* no. 301, pp. 181-194, 2015.

[19] A. F. Sergey Muravyov, "Meta-Learning System for Automated Clustering," in *PKDD*, 2017.

[20] H. V. L. M. M. R. B. Luzie Helfmann, "On Hyperparameter Search in Cluster Ensembles," 2018.

[21] S. S. Tahani Aljuaid, "Proper Imputation Techniques for Missing Values in Data Sets," in *ICDSE*, 2016.

[22] V. M. S. L. D. K. G. P. F. W. Kathleen F. Weaver, "Pearson's and Spearman's Correlation," in *Statistical Analysis in Research*, 2017.

[23] G. Bonnacorso, Mastering Machine Learning Algorithms, Packt Publishing, 2018.

[24] Z. L. H. X. X. G. J. W. Yanchi Liu, "Understanding of Internal Clustering Validation Measures," in *IEEE International Conference on Data Mining*, 2010.

[25] M. V. Maria Halkidi, A density-based Cluster Validity Approach using Multi-representatives, 2008.

[26] M. V. Maria Halkidi, "Clustering Validity Assessment: Finding the optimal partitioning of a data set".

[27] Y. B. James Bergstra, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research,* vol. 13, pp. 281-305, 2012.

[28] Y. B. M. V. Maria Halkidi, "On Clustering Validation Techniques," *Journal of Intelligent Information Systems ,* vol. 17, pp. 107-145, 2001.

[29] Z. L. H. X. X. G. J. W. Yanchi Liu, "Understanding of Internal Clustering Validation Measures_," in *IEEE International Conference on Data Mining*, 2010.

[30] P. A. J. R. J. G. B. C. A. Z. Davoud Moulavi, "Density-Based Clustering Validation," in *SIAM International Conference on Data Mining*, 2014.

[31] M.-C. Monard, "A study of K-Nearest Neighbour as an Imputation Method," 2002.

[32] D. S. A. d. A. ,. R. B. C. P. I. G. C. Marcilio Carlos Pereira de Souto, "Ranking and Selecting clustering algorithms using a meta-learning approach," in *IEEE*, 2008.

[33] Y. B. James Bergstra, "Random Search for Hyperparameter Optimization," *Journal of Machine Learning Research ,* 2012.

[34] K. S. Z. W. R. P. A. N. d. F. Bobak Shanhriari, "Taking the Human out of the Loop: A Bayesian Optimization Review," in *IEEE*, 2016.

[35] M. G. Kendall, Rank Correlation Methods, Griffin, 1970.