2019

OPTIMIZING RECOMMENDATIONS FOR CLUSTERING ALGORITHMS USING META-LEARNING

Adam Jilling
University of Rhode Island, ajilling@gmail.com

Follow this and additional works at: <https://digitalcommons.uri.edu/theses>

Recommended Citation

Jilling, Adam, "OPTIMIZING RECOMMENDATIONS FOR CLUSTERING ALGORITHMS USING META-LEARNING" (2019). *Open Access Master's Theses*. Paper 1728.
<https://digitalcommons.uri.edu/theses/1728>

This Thesis is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Master's Theses by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons-group@uri.edu.

OPTIMIZING RECOMMENDATIONS FOR CLUSTERING ALGORITHMS
USING META-LEARNING

BY
ADAM JILLING

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2019

MASTER OF SCIENCE THESIS
OF
ADAM JILLING

APPROVED:

Thesis Committee:

Major Professor Marco Alvarez

Lutz Hamel

William Ohley

Nasser H. Zawia

DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2019

ABSTRACT

The field of machine learning (ML) has seen explosive growth over the past decade, largely due to increases in technology and improvements of implementations. As powerful as ML solutions can be, they are still reliant on human input to select the optimal algorithms and parameters. This process is typically done by trial and error, as researchers will select a number of algorithms and choose whichever provides the most desirable result.

This study will use a process called meta-learning to evaluate and analyze datasets and extract a series of meta-features. These features can then be used to intelligently recommend an optimal algorithm, without the cost of having to manually run the algorithm. To accomplish this, we will experiment using 230 datasets and determine their expected outcomes using only the meta-features. The outcomes being optimized are performance (accuracy) and runtime.

Results are ranked in terms of performance and runtime and we can determine how accurately the learning model was able to choose the optimal algorithm for each objective. Additionally, we also run tests to determine the optimal learning rate and weight decay to use when training.

TABLE OF CONTENTS

ABSTRACT	ii
TABLE OF CONTENTS	iii
LIST OF FIGURES	v
LIST OF TABLES	vii
CHAPTER	
1 Introduction	1
2 Related Work	4
2.1 AutoML	4
2.2 Meta-Learning	5
2.3 Clustering	5
2.4 AutoML Applied to Clustering	9
2.4.1 Performance Optimization	9
2.4.2 Runtime Optimization	11
2.5 Neural Networks and Deep Learning	12
3 Methodology & Experiments	14
3.1 Datasets and Preprocessing	14
3.2 Feature Extraction	15
3.2.1 Statistical-Based Metafeatures	16
3.2.2 Distance-Based Metafeatures	17
3.3 Data Collection	19

	Page
3.3.1 Performance Data	19
3.3.2 Runtime Data	20
3.4 Neural Net Training/Testing	20
3.5 Results and Analysis	24
4 Conclusion	34
LIST OF REFERENCES	36
BIBLIOGRAPHY	39

LIST OF FIGURES

Figure		Page
1	Visualization of a distribution model, expectation-maximization (EM), which uses multivariate normal distributions. Each centroid is marked with a (+).	6
2	Visualization of single-linkage clustering, an agglomerative connectivity model. At each step, two clusters that have not yet been categorized are combined. Here we can see three primary clusters (red, green, blue) and other smaller clusters (purple, gold, aqua).	7
3	Visualization of k-means clustering, showing cluster vectors and centroids (+). We can see that clusters can never overlap. . . .	8
4	Visualization of DBSCAN algorithm. Points that are tightly packed are assumed to be members of the same class. When the density of the points lessens, we are likely reaching the cluster's boundary.	8
5	General structure of an artificial neural network showing an input layer with ten values, three hidden layers of sizes eight, six, and six, and two outputs	13
6	Overview of the entire process showing feature extraction, fitting, tensor building, training, testing, and output	14
7	Diagram showing dimensions of tensors used. Input tensors contain thirty-two values - nineteen metafeatures and seven values forming a vector to represent each dataset. The output is two values - runtime and performance	21
8	Results attempting to predict ranking #1. The x-axis shows the actual results and how many times each value was predicted. . .	26
9	Results attempting to predict ranking #2. The x-axis shows the actual results and how many times each value was predicted. . .	26
10	Results attempting to predict ranking #3. The x-axis shows the actual results and how many times each value was predicted. . .	27

Figure		Page
11	Results attempting to predict ranking #4. The x-axis shows the actual results and how many times each value was predicted. . .	27
12	Results attempting to predict ranking #5. The x-axis shows the actual results and how many times each value was predicted. . .	28
13	Results attempting to predict ranking #6. The x-axis shows the actual results and how many times each value was predicted. . .	28
14	Results attempting to predict ranking #7. The x-axis shows the actual results and how many times each value was predicted. . .	29
15	Difference between the predicted and actual performance average ranking for each algorithm.	30
16	Difference between the predicted and actual runtime average ranking for each algorithm.	31
17	Number of times each algorithm was predicted to be in the top three for performance.	32
18	Number of times each algorithm was predicted to be in the top three for runtime.	33

LIST OF TABLES

Table		Page
1	Clustering performance metrics used, the package and language used to implement them, the range of outputs, and the optimization objective.	9
2	Comparison showing previously done related studies and the scope of each. The eleventh metric in this work is referring to runtime tracking.	11
3	Distance-Based Metafeatures and Descriptions	18
4	Clustering algorithms used and the values of any customizable parameters. For algorithms needing a set number of clusters, the number of attributes was used	19
5	Average training loss when running model with different parameters. The x-axis shows different weights applied to learning rate. The y-axis shows different weights applied to weight decay.	23
6	Average testing loss when running model with different parameters. The x-axis shows different weights applied to learning rate. The y-axis shows different weights applied to weight decay.	23
7	Example of the result structure produced for a single dataset. The performance and runtime results are ranked based on their predicted values and the actual values obtained when run.	24
8	Accuracy predicting the top algorithms over all datasets. Top 1 means the actual best algorithm was predicted, top 2 means the actual best algorithm was predicted in first or second place, etc.	25

CHAPTER 1

Introduction

Machine learning is a very expensive process, both from a human and machine perspective. From a human perspective, a great deal of time is required to find, test, and tweak algorithms. For instance, testing just four algorithms, each with three customizable parameters, using any of three different values for each parameter, results in thirty-six distinct tests to run. From a machine perspective, a huge amount of processing power or memory consumption is required for each run. These costs grow linearly with larger datasets and exponentially as the number of parameters grows. If we can automate the process of algorithm selection, or even just help narrow down the selection, we can prevent a great deal of unnecessary work.

Cluster analysis provides a powerful way of automating the grouping and classification of different sets of objects. There is a large number of clustering algorithms with an even larger number of customizable parameters. Selection of an optimal algorithm is often determined by factors such as accuracy, speed, resources required, or other metrics. However, the process of testing different algorithms is often slow and largely trial-and-error based. The goal of algorithm selection is to choose a clustering algorithm based upon the structural properties of the problem [1]. If the process of algorithm recommendation could be automated based on the feature set of the problem, it would become much more efficient.

There are a number of ways to determine which algorithm is the most desirable. The most common metric is accuracy, also referred to as performance. We can choose to optimize for performance or for runtime. Runtime is important for the large number of researchers who may not have access to large GPU clusters.

Sometimes a wise trade-off of a method’s speed and efficiency may be more important than its accuracy. Such cases might involve privacy concerns, high latency, or network connectivity issues and are best resolved by training being done locally on the device itself [2]. Other examples could include modeling real-time traffic flows, short-term stock market pricing trends, medical symptom evaluation, and real-time marketing/advertising. While meta-learning and the creation of meta-features itself will carry a cost, that cost can be neglected if it is amortized enough to result in a net positive across the entire application [3].

Meta-learning is the process of analyzing past results to choose future settings dynamically. The contrast is base-learning where the settings are fixed [4]. By leveraging predefined meta-features and their performance results, we can select algorithms that we know are likely to perform better than others. In this case, the setting being adjusted is the algorithm selected.

This thesis proposes the use of *metadata* — data that describes other data — to automate the process of algorithm recommendation. In this case, the metadata will describe the characteristics of the problem, specifically, various metrics of a dataset. A series of *meta-features* will be defined and their values calculated for a given number of datasets. We then apply seven unique clustering algorithms to these datasets and measure their performance (accuracy) and runtime. These results will then be fed into a neural network to predict the performance and runtime for other datasets when using the same seven algorithms. A recommendation can then be made for which algorithm would optimize performance and which would optimize runtime without the cost of having to run the algorithms.

Chapter 2 discusses related work and the background of other meta-learning studies. We give an overview of clustering and a general introduction to neural networks and deep learning. Most prior work has focused solely on performance

optimization. This work will introduce runtime optimization as well.

In chapter 3 we focus on the methodology and how the study is conducted. We detail how each meta-feature is chosen and how it is calculated. An overview of the system and process is given, both from an architectural and code-based perspective. We then go into detail regarding how experiments are run and analyze the results. Decisions on how to construct the neural network are discussed and we detail the process of training and testing. We then determine and graph how well our model was able to predict actual results using only the meta-features of a dataset.

The thesis concludes in chapter 4 where we discuss an overview of the findings and consider what might be done in future work.

CHAPTER 2

Related Work

In this chapter, we discuss the background research involving AutoML, clustering, and deep learning with neural networks. Previous meta-learning studies are also discussed and compared.

2.1 AutoML

Automated machine learning, more commonly referred to as AutoML, is the concept of automating the full machine learning process. The machine learning pipeline includes data preparation, feature engineering, model generation, and model evaluation [5]. Performing all these steps manually can take a great deal of time and expertise, so instead, we leverage existing tools to improve both the speed and accuracy of the process, resulting in much greater efficiency [6]. Additionally, this opens up the field of ML to those without ML domain specific knowledge [7]. Attempts have even been made to crowdsource and benchmark previous ML studies to use as a reference for future work [8].

According to the *no-free-lunch* theorem [9] it is impossible for there to be a single ML pipeline that is optimal for every application. It follows that for each new problem, a new pipeline would need to be constructed, which is a very tedious and time-consuming process. The goal of AutoML is to automate these processes, such as data cleaning, feature engineering, or hyperparameter selection [10].

Most classes of problems will have some structure that, if known, can be exploitable. To justify its use, that structure must be known and be directly reflected in the choice of algorithm [9]. In this thesis, the structure that we aim to exploit is defined by the metafeatures of each dataset.

2.2 Meta-Learning

The goal of meta-learning is to improve average performance on new tasks by utilizing experience in past tasks [11, 12]. The authors in [11] have grouped current meta-learning approaches into three categories:

- Gradient-based: learns parameter initialization from past experiences on the distribution of related tasks.
- Memory-enabled neural network: takes advantage of the characteristics of network structure to memorize past performance.
- Hierarchical: divides neural network into two layers, a high level and low level, to learn different levels of task knowledge. The high level provides sub-goals to decide which low level network to choose. The low level network contains primitive actions.

Meta-learning has been used to help fill incomplete models in space missions that have highly variable or even completely unknown parameters [13]. It has also been used to augment zero-shot learning (ZSL), the process of classifying unseen class examples at runtime [14, 15, 16].

A meta-learning system is comprised of two parts. The first part is concerned with the acquisition of knowledge while the second part handles the application of meta-knowledge to new problems [17].

2.3 Clustering

Clustering is the process of separating groups of objects in such a way that objects within a group are more similar to each other than objects outside the group, or cluster. It is not a one-shot process and usually requires a series of trials and repetitions [18]. There are a number of methods that can accomplish this,

known as clustering algorithms. Each algorithm is usually classified by how it accomplishes the clustering [19]. Some families of algorithms include:

Distribution Models Data points are modeled based on the probability they fit into a particular cluster. The number of clusters used is fixed and predefined. Each item is assigned to the cluster for which it has the highest probability of belonging [20]. Gaussian Mixture Models are examples of distributed clustering algorithms.

Figure 1. Visualization of a distribution model, expectation-maximization (EM), which uses multivariate normal distributions. Each centroid is marked with a (+).

Connectivity/Hierarchical Models This approach can either be top-down (divisive) or bottom-up (agglomerative). In a divisive approach, all observations begin in a single cluster and divisions form as the data is analyzed. An agglomerative approach begins with each observation as its own cluster. Similar clusters are then merged together. Clusters are defined based on distance. The idea is that data points closer to each other have more in common than those spaced farther apart. The function used to calculate distance can vary. Average

Agglomerative Clustering is a connectivity model [21].

Figure 2. Visualization of single-linkage clustering, an agglomerative connectivity model. At each step, two clusters that have not yet been categorized are combined. Here we can see three primary clusters (red, green, blue) and other smaller clusters (purple, gold, aqua).

Centroid Models Sometimes called partitional models, here a series of centroids are predefined and each observation is paired with the centroid to which it lies closest. Each cluster is represented by a single mean vector. A drawback is that the number of clusters must be specified beforehand. Also, centroid models are unable to handle noise or deal with clusters with non-convex shapes [22]. Centroid models include k-means and fuzzy c-means [23].

Density Models Here the data space is scanned for areas of varying density and partitions are made where the density is lower, signifying the edges of a cluster. Density-based spatial clustering of applications with noise (DBSCAN) and Mean-shift are two well-known density-based algorithms [24].

There are other types of algorithms but these four cover most of the algorithms used in this work.

Figure 3. Visualization of k-means clustering, showing cluster vectors and centroids (+). We can see that clusters can never overlap.

Figure 4. Visualization of DBSCAN algorithm. Points that are tightly packed are assumed to be members of the same class. When the density of the points lessens, we are likely reaching the cluster's boundary.

2.4 AutoML Applied to Clustering

Running clustering algorithms and extracting meaningful results involves more than running an algorithm — an entire process is needed. Our goal is to find a way to optimize the process, by optimizing one or more specific steps in it. There are a number of metrics that can be used to define “optimal”, such as memory consumption, performance, CPU use, or runtime. We will focus on performance optimization and runtime optimization.

2.4.1 Performance Optimization

The most common optimization goal is for accuracy, usually referred to as performance. Performance optimization aims to maximize the number of data points that are assigned to their correct cluster. There are many metrics that attempt to evaluate this in different ways. Table 1 shows ten of these metrics along with the software used to implement them and their performance objectives.

Table 1. Clustering performance metrics used, the package and language used to implement them, the range of outputs, and the optimization objective.

Index	Package	Interval	Objective
Calinski-Harabasz	scikit-learn (Python)	$[0, \infty)$	max
Silhouette	scikit-learn (Python)	$[-1, 1]$	max
Dunn	fpc (R)	$[0, \infty)$	max
Pearson Gamma	fpc (R)	$[-1, 1]$	max
Tau	clusterCrit (R)	$[0, \infty)$	max
Davies-Bouldin	clusterCrit (R)	$[0, \infty)$	min
Xie-Beni	clusterCrit (R)	$[0, \infty)$	min
SD-Scat	clusterCrit (R)	$[0, \infty)$	min
SD-Dis	clusterCrit (R)	$[0, \infty)$	min
Ray-Turi	clusterCrit (R)	$[0, \infty)$	min

Although clustering is an unsupervised task, often times performance is incorrectly evaluated by using the clustering labels as the prediction objective. As pointed out by [25], this can lead to incorrect or misleading results, since labels are intended for classification tasks, not clustering. By using classification labels, we

focus only on a specific property rather than the distribution of the entire dataset. For example, there might be a situation where groups of data with different class labels overlap. These labels might be better represented as a single cluster, yet using existing class labels as the ground truth objective would deem the results incorrect. Another example would be objects with the same class label corresponding to multiple clusters. For these reasons, in this work, all class labels are dropped from each dataset and we will rely solely on these performance metrics for evaluation. This does provide a slight disadvantage as class labels are often used as a way to “cheat” and specify the number of desired clusters for centroid models. Instead, we specify the number of desired clusters to be equal to the number of attributes in the dataset.

The meta-learning approach to clustering algorithm recommendation was used by [26] to optimize for performance. They limited the number to thirty-two cancer gene expression datasets and used seven unique algorithms - single linkage, complete linkage, average linkage, k-means, mixture model clustering, spectral clustering, and shared nearest neighbors algorithm. Eight statistical metafeatures were chosen, the six used here, plus two more. They then run the algorithms and evaluate the performance by comparing results to the ground truth classification label. They found that their method provided a significant advantage over using the default ranking [26].

The authors in [27] used thirty datasets and ten metafeatures. The five algorithms used were K-Means, Single Linkage, Complete Linkage, Medium Linkage, and a Self-Organizing Feature Map. Accuracy was again measured by comparing predictions vs. ground truth labels. They found that meta-learning can “provide a guide for designing experiments and choosing suitable algorithms for each type of problem based on its features” [27].

The study done in [28] improved upon earlier attempts by expanding the number of datasets, algorithms, metafeatures, and the metrics used to evaluate performance. They also sought to determine which *types* of metafeatures, statistical or distance-based, are the most suitable for a given problem. Table 2 shows an overview of the studies.

Table 2. Comparison showing previously done related studies and the scope of each. The eleventh metric in this work is referring to runtime tracking.

Study	Datasets	Meta-features	Algorithms	Metrics
de Souto	32	8	7	1
Ferrari	30	10	5	1
Pimentel	218	25	10	10
Jilling	230	25	7	11

2.4.2 Runtime Optimization

Since the desirability of clustering algorithms is largely driven by which is the most accurate, the area of runtime optimization has seen fewer contributions. Some previous works have been able to leverage meta-knowledge to predict training time, some by using only the number of instances and features [29]. There are many real-world scenarios where an algorithm’s runtime could be more important than its performance, provided the performance loss is an amount deemed acceptable. For that reason, this work will still track performance to ensure improvements in runtime aren’t completely at the expense of accuracy.

Some works have attempted to quantify the trade-off between performance and runtime. The authors in [30] defined the formula:

$$ARR_{a_p, a_q}^{d_i} = \frac{\frac{SR_{a_p}^{d_i}}{SR_{a_q}^{d_i}}}{1 + AccD \times \log \left(\frac{T_{a_p}^{d_i}}{T_{a_q}^{d_i}} \right)} \quad (1)$$

where $SR_{a_p}^{d_i}$ and $T_{a_p}^{d_i}$ represent the success rate and time, respectively, of al-

gorithm a_p on dataset d_i and $AccD$ is a user-defined measure of the importance of accuracy and time. The user can then tune the algorithm accordingly. For example, supplying an $AccD$ of 10% means the user is willing to sacrifice 10% of accuracy for a ten times speedup [30].

Certain algorithms, by their nature, are naturally inclined to run at different speeds than others. In one study involving bank data, it was determined that hierarchical models take the most time while k-means and density-based algorithms were significantly faster [31].

2.5 Neural Networks and Deep Learning

An Artificial Neural Network (ANN) is a system of connected nodes designed to emulate the human brain. Much like how a human brain contains billions of neurons connected by synapses, an ANN is comprised of nodes connected by a series of weighted edges. An ANN contains an input layer, an output layer, and a number of hidden layers in between. Each layer is comprised of a number of nodes and each node transforms an input into an output via an *activation function*. Widely-used activations include step, sigmoid, rectified linear unit (ReLU), and tanh. The aim of the hidden layers is to transform the input into some kind of useful output. The input is transformed by iteratively tweaking the weights of the edges. Deep learning is an area of machine-learning that involves ANNs with two or more hidden layers. An ANN with a single hidden layer is called a *shallow* network [32].

As the ANN is iterated over, a matrix multiplication is performed on each layer based on the given weights. The average of the mistakes is tracked, called the loss. After each iteration, the weights are tweaked by back-propagating through the network. Changes can then be made to the training model to find a more desirable result. Adjusting and tweaking an ANN's parameters and choosing a

suitable classifier is still more art than science [33].

Figure 5 shows an ANN with ten inputs, three hidden layers, and a two-dimensional output. There are a total of 32 nodes and 176 edges.

Figure 5. General structure of an artificial neural network showing an input layer with ten values, three hidden layers of sizes eight, six, and six, and two outputs

The ANN used in this work has an input layer of 32 values, three hidden layers, and two outputs. The input contains twenty-five metafeatures and a one-hot encoding representing each of the seven algorithms as input. The output will be two values representing the expected performance and runtime for the specified clustering algorithm.

CHAPTER 3

Methodology & Experiments

Here we describe the processes used in this work, starting with data pre-processing and feature extraction. This continues with the training and timing of each algorithm for each dataset and the recording of results. We then discuss the design of the ANN, the decisions behind it, and the training and testing process. Finally, the results are visualized and analyzed. Figure 6 shows a diagram of the entire process.

Figure 6. Overview of the entire process showing feature extraction, fitting, tensor building, training, testing, and output

3.1 Datasets and Preprocessing

OpenML [34] is a project that provides, among other things, datasets to use in machine learning projects. 230 datasets from OpenML are used¹, covering a

¹<https://www.openml.org/s/88/data>

wide range of categories, including medical, biological, climatic, astrological, and social topics. This library of datasets was mostly compiled and used by the study in [28], although some additions and removals have been done for this study.

The process begins with normalizing all values on the interval $[0, 1]$. Next, we find and remove any columns that are computationally or exactly singular to other columns. Any columns that are linear combinations of others (either exact or close) will result in errors when running multivariate analysis and need to be removed. Some sets are found to be computationally singular if they have very small values which can be rounded to zero, leading to the assumption that it is a singular matrix. Since a singular matrix is not invertible, it would then prevent a number of algorithms in the *MVN* package from running. The R package *caret* is able to clean any datasets with a high correlation among dependent variables. About a quarter of our datasets fall into this category.

3.2 Feature Extraction

The objective of meta-feature characterization is to capture the identifying characteristics of a dataset and use that information to group other similar datasets. This work will rely primarily on the metafeatures of datasets to make intelligent recommendations. Therefore, the features chosen and how they are calculated become extremely important. The authors in [26] proposed the use of eight statistical metafeatures. The study in [28] built upon that method, dropping two of the features due to being too subject-specific, as the goal is for this to generalize over datasets of all types. They also built upon the work of [27], who proposed the use of distance-based metafeatures where the Euclidean distance between objects is used to obtain a measure of dissimilarity.

This work will leverage these previous metrics using six statistical-based features and nineteen distance-based metrics. The result will be a twenty-five item

vector characterizing each dataset.

3.2.1 Statistical-Based Metafeatures

Statistical-based metafeatures are macro-level observations of a dataset. Here we will quantify information such as the size of the dataset – both the number of entries and the number of parameters for each entry – and we will look at normality, variance, and the overall distribution of the data. These features will provide a rough indication of the size, quality, and behavior of each dataset.

1. Number of Entries (NE)

$$NE = n \quad (2)$$

n is the number of entries. This indicates the size of the dataset.

2. Number of Entries per Attribute (NEA)

$$NEA = \frac{n}{p} \quad (3)$$

n is the number of entries, p is the number of attributes. This indicates the robustness of the dataset, or how descriptive it is.

3. Percentage of Missing Values (PMV)

$$PMV = \frac{m}{t} \cdot 100 \quad (4)$$

m is the number of missing entries, t is the total number of entries. This measures the completeness of the dataset².

4. Multivariate Normality (MN)

A measure of how close the dataset is to a normal distribution. This value is computed using R's *MVN* package [35] and Royston's algorithm.

²every dataset used in this paper is fully complete so this value will be 0 for all

5. Skewness (SK)

A measure of how far a distribution is pushed left or right. This measures the dataset's asymmetry. This value is computed using R's *MVN* package and Mardia's Test.

6. Percentage of Outliers (PO)

$$PO = \frac{o}{t} \cdot 100 \quad (5)$$

o is the number of entries that are labelled as outliers, meaning they are more than two standard deviations from the mean, t is the total number of entries.

3.2.2 Distance-Based Metafeatures

The goal here is to calculate the pairwise Euclidean distance between entries (rows). Given a dataset X containing n entries described by p variables, we use the following formula to calculate the distance, d , between entries i and j .

$$d(X_i, X_j) = \sqrt{\sum_{c=1}^p (x_{i,c} - x_{j,c})^2} \quad (6)$$

We then create a vector of size $n(n-1)/2$ listing all pairwise distances:

$$d = [d_{1,2}, d_{1,3}, d_{1,4}, \dots, d_{2,3}, d_{2,4}, \dots, d_{n-1,n}] \quad (7)$$

Min-Max Feature Scaling is then implemented to normalize the vector on the interval $[0, 1]$. The resulting vector is labeled m' and is used to calculate the nineteen metafeatures shown in Table 3.

Table 3. Distance-Based Metafeatures and Descriptions

Metafeature	Description
MF ₁	Mean of m'
MF ₂	Variance of m'
MF ₃	Standard deviation of m'
MF ₄	Skewness of m'
MF ₅	Kurtosis of m'
MF ₆	% of values in $[0, 0.1]$
MF ₇	% of values in $(0.1, 0.2]$
MF ₈	% of values in $(0.2, 0.3]$
MF ₉	% of values in $(0.3, 0.4]$
MF ₁₀	% of values in $(0.4, 0.5]$
MF ₁₁	% of values in $(0.5, 0.6]$
MF ₁₂	% of values in $(0.6, 0.7]$
MF ₁₃	% of values in $(0.7, 0.8]$
MF ₁₄	% of values in $(0.8, 0.9]$
MF ₁₅	% of values in $(0.9, 1.0]$
MF ₁₆	% of values with absolute Z-score in $[0, 1)$
MF ₁₇	% of values with absolute Z-score in $[1, 2)$
MF ₁₈	% of values with absolute Z-score in $[2, 3)$
MF ₁₉	% of values with absolute Z-score in $[3, \infty)$

3.3 Data Collection

Next we will run and time algorithms and fit models to the datasets. Each dataset is again normalized on the interval $[0, 1]$. For algorithms that require a set number of clusters, the number of clusters is equal to the number of classes in the dataset. Admittedly, this is somewhat of a shortcoming as selecting the optimal number of clusters is a problem in itself. Selecting too many clusters can over-complicate the result while selecting too few clusters can result in information loss and over-generalization [18]. Seven algorithms will be run, all from Python’s *scikit-learn* package, shown in Table 4. Each algorithm will be measured for both performance and runtime.

Table 4. Clustering algorithms used and the values of any customizable parameters. For algorithms needing a set number of clusters, the number of attributes was used

Label	Algorithm	Parameters
AA	Average Agglomerative	affinity=euclidean, linkage=average
CA	Complete Agglomerative	affinity=euclidean, linkage=complete
GMD	Gaussian Mixture Diagonal	covariance_type= diagonal
GMF	Gaussian Mixture Full	covariance_type=full
KM	K-Means	init=k-means++, n_init=10
MK	Mini Batch K-Means	init=k-means++, n_init=10
WA	Ward Agglomerative	affinity=euclidean, linkage=ward

3.3.1 Performance Data

To calculate performance (accuracy), we use the ten clustering metrics shown in Table 1. Since this is unsupervised, we use internal indices to evaluate performance, meaning the quality of the clustering structure uses features already inherent in the dataset. Since each metric uses unique scales and objectives, these

results will need to be normalized and averaged to ensure that all ten metrics are weighted equally. Metrics with a maximization objective are normalized on the interval $[0, 1]$. Metrics with a minimization objective are normalized on the same interval and then “flipped” by subtracting one and taking the absolute value. For example, a result of 0.28 on a min scale would become 0.72 on a max scale. This puts all metrics on equal footing allowing them to be averaged. Since the objective is to compare among seven results, the actual numeric result is irrelevant, as long as it’s consistent among all seven allowing us to rank relative to one another.

3.3.2 Runtime Data

To calculate runtime, a dedicated CPU (Intel Xeon E5-1603 V3 @ 2.80GHz, 4 cores, 4 threads, 8GB RAM running Ubuntu 18.04) in the Alvarez Lab is used to measure the exact time it takes to train each algorithm. In order to remove any unrelated factors, the machine has no network connection and minimal concurrent processes. Since an algorithm’s runtime could be influenced by how efficient a package is implemented, the *scikit-learn* package for Python is used for all to ensure consistency. We will do ten runs total and take the average, while also ensuring the variance in each run is relatively low. If distinct runtime results vary by a significant amount, there is likely an external condition that needs to be addressed.

3.4 Neural Net Training/Testing

We will then leverage past knowledge about how these datasets performed and use that to predict the future results. To accomplish this, we create a neural network using Python’s *PyTorch* package and use leave-one-out cross validation (LOOCV). The input will be the meta-features and the output will be the performance and runtime predictions. There are 1610 input tensors ($230 \text{ datasets} \times 7$

algorithms), each with 32 features, shown below in Figure 7. Each input tensor has a corresponding output tensor with two features. A Python script will be run to loop through the input tensors in blocks of seven. Each time, the ANN is trained with 229 datasets and tested on the held out set.

Figure 7. Diagram showing dimensions of tensors used. Input tensors contain thirty-two values - nineteen metafeatures and seven values forming a vector to represent each dataset. The output is two values - runtime and performance

The next steps involve architecting the ANN and the decisions and tests involved to determine its design. We have the input and output content for our network, now the question is how to design the hidden layers and which parameters to use. A sequential network is selected, meaning that layers are added in the order in which they are passed in the constructor. This allows us to have complete control over the path of hidden layers.

A Python class is created with user-defined parameters specifying the number of training iterations, learning rate, weight decay, and the number of datasets to train. After each pass, the average training loss and average testing loss are outputted, allowing us to gauge the relative success of the algorithm. We can run and eyeball these results for awhile, which we do, but a more thorough method is to write a bash script using a wide range of values for weight decay and learning rate and record the loss value from each. Eight values $[10^{-6}, 10^{-5}, 10^{-4}, 0.001, 0.01, 0.1, 1, 10]$ are selected for learning rate and five $[10^{-4}, 0.001, 0.01, 0.1, 1]$ for weight decay. The script shown below then runs and records the loss from all forty combinations giving us a visual guide of the best

places to set these values.

Listing 3.1. Bash script used to test for optimal parameter values. The four command line arguments are, in order, learning rate, weight decay, number of iterations, and number of datasets.

```
python3 train.py .000001 .0001 1500 20
python3 train.py .000001 .001 1500 20
python3 train.py .000001 .01 1500 20
...
python3 train.py .00001 .0001 1500 20
python3 train.py .00001 .001 1500 20
python3 train.py .00001 .01 1500 20
...
```

We record both training loss and testing loss, even though we expect them to be similar given the same parameters. Table 5 shows the average loss for each set of values during training. Table 6 shows the average loss during testing. A darker shade of red indicates a lower average loss (more optimal) while a lighter shade indicates a higher loss (less optimal).

A quick look at the heat maps shows the optimal range for learning decay to lie somewhere between 10^{-4} and 0.01 and the optimal range for weight decay to lie between 10^{-4} and 0.001. As expected, the training and testing losses show little difference. Based on this information, we select a weight decay of 0.001 and a learning rate of 0.001 and continue building the model.

The rest of the process involves designing the hidden layers of the ANN, and as previously mentioned, is somewhat of an inexact science. After a lot of trial and tweaking based on feedback and data from trial runs, we end up with a network

Table 5. Average training loss when running model with different parameters. The x-axis shows different weights applied to learning rate. The y-axis shows different weights applied to weight decay.

Table 6. Average testing loss when running model with different parameters. The x-axis shows different weights applied to learning rate. The y-axis shows different weights applied to weight decay.

with three hidden layers of sizes 32, 24, and 8. The first two hidden layers use a ReLU (rectified linear unit) activation function. ReLU takes the positive part of each argument and sets all negative values to zero. The third hidden layer uses a sigmoid activation, preventing any negative values as sigmoid, by its nature, produces outputs in the range $(0, 1)$.

Once built, the ANN is run on all 230 datasets using LOOCV and all performance and runtime output is recorded and output to a text file. A Python script then parses the file and converts raw data into rankings. These rankings are then analyzed to obtain a measure of effectiveness.

3.5 Results and Analysis

After all numerical values are ranked, we end up with the following data structure for each dataset. Below is the result from dataset # 27:

Table 7. Example of the result structure produced for a single dataset. The performance and runtime results are ranked based on their predicted values and the actual values obtained when run.

		AA	CA	GMD	GME	KM	MK	WA
Predicted	Perf	3	5	1	4	7	2	6
	RT	1	3	4	6	7	5	2
Actual	Perf	3	6	1	4	7	2	5
	RT	2	1	4	6	7	5	3

In this example we can see (*in blue*) the predicted top-performing algorithm was GMD and that was indeed the actual top-performing algorithm. The predicted top runtime algorithm (*in red*) was AA but it was actually CA, with AA finishing second.

Since the goal of this project is to identify the top ranked algorithm, we will primarily focus on all results in the top three. The following table compares the predicting of top algorithms to the ground truth results obtained from running the algorithms. Looking at the Performance column, we see the top performing

algorithm was predicted correctly 43.9% of the time, the top performing algorithm was predicted to be in the top two 63.0% of the time, and the actual top performing algorithm was predicted to be in the top three 75.7% of the time.

Table 8. Accuracy predicting the top algorithms over all datasets. Top 1 means the actual best algorithm was predicted, top 2 means the actual best algorithm was predicted in first or second place, etc.

	Performance	Runtime
Top 1	43.9%	31.3%
Top 2	63.0%	85.7%
Top 3	75.7%	98.2%

We can also examine the results on a per-rank basis. The tables below each look at a predicted ranking and chart its corresponding actual ranking. For example, the chart for ranking #1 looks at the top predicted algorithm for each of the 230 datasets. With respect to performance, we can see that 101 times, the top predicted algorithm was the top actual algorithm, 44 times the top predicted algorithm was the second best performing, and so on. We can even see that in 9 cases, the top predicted performer was actually the worst performing. We would hope that the chart for ranking #1 peaks at 1, the chart for ranking #2 peaks at 2, and so on.

Finally, we can visualize sorted by algorithm rather than ranking. This allows us to see if some algorithms are just naturally better performing or faster running. Figure 15 shows that, with respect to performance, the algorithms are fairly evenly distributed. Ward Agglomerative (WA) and K-Means (KM) are the two weakest performing while Gaussian Mixture Diagonal (GMD) and Mini Batch K-Means (MK) are two of the best. Interestingly, Mini Batch K-Means outperforms K-Means. This would be expected with regard to runtime as Mini Batch K-Means is just K-Means with a smaller number of randomly selected observations (batch size). It is possible that the smaller batch size prevents over-fitting which leads to

Figure 8. Results attempting to predict ranking #1. The x-axis shows the actual results and how many times each value was predicted.

Figure 9. Results attempting to predict ranking #2. The x-axis shows the actual results and how many times each value was predicted.

Figure 10. Results attempting to predict ranking #3. The x-axis shows the actual results and how many times each value was predicted.

Figure 11. Results attempting to predict ranking #4. The x-axis shows the actual results and how many times each value was predicted.

Figure 12. Results attempting to predict ranking #5. The x-axis shows the actual results and how many times each value was predicted.

Figure 13. Results attempting to predict ranking #6. The x-axis shows the actual results and how many times each value was predicted.

Figure 14. Results attempting to predict ranking #7. The x-axis shows the actual results and how many times each value was predicted.

the increase in performance.

Figure 16 tells a different story, as we can see that certain algorithms fairly consistently have quicker runtimes than others. The agglomerative algorithms (AA, CA, WA) provide the fastest training times while K-Means (KM) and Gaussian Mixture Full (GMF) provide the slowest. Since Gaussian Mixture Diagonal (GMD) is a more lightweight version of Gaussian Mixture Full (GMF), we would expect that to be faster, and it is. The same holds true for Mini Batch K-Means outperforming K-Means. It's also important to reiterate that both K-Means algorithms are run with their default parameter of ten iterations.

The last two charts show how many times each algorithm was predicted in each of the top three rankings for both performance and runtime. This is important because it allows us to get a feel for what the neural network learned. In Figure 17 we see a reasonable distribution and that, in terms of performance predictions, the network thought most highly of Average Agglomerative (AA) and Mini Batch K-Means (MK).

Figure 15. Difference between the predicted and actual performance average ranking for each algorithm.

Figure 16. Difference between the predicted and actual runtime average ranking for each algorithm.

Figure 18 shows that the network was *really* able to learn the quicker runtime nature of the agglomerative algorithms. This would account for the 98.2% success rate when selecting the top 3.

Figure 17. Number of times each algorithm was predicted to be in the top three for performance.

Figure 18. Number of times each algorithm was predicted to be in the top three for runtime.

CHAPTER 4

Conclusion

In this study, we have presented a method for using meta-learning to intelligently recommend clustering algorithms. The process of defining and calculating each meta-feature is detailed. We also reference and use a number of clustering performance metrics and detail how to effectively measure runtime when training algorithms.

Tensor construction and construction of a neural network are shown using *PyTorch*. We also create a series of tests designed to try various combinations of parameters and to track average training and testing losses. This allows us to determine the optimal parameters to use in creating the network.

With respect to runtime, our meta-learning system was able to predict the top algorithm over 30% of the time. It was able to recommend one of the top two algorithms more than 85% of the time, and in over 98% of cases, the system was able to recommend one of the top three algorithms. If we define success as being in the top three, the system was unsuccessful in only 1.8% of cases.

When optimizing for performance, the system was able to identify the top algorithm almost 44% of the time and one of the top three algorithms over 75% of the time.

In the future, we hope to do more work to decipher which of the twenty-five metafeatures used are the most important. It is possible that of the twenty-five, only a handful are actually relevant towards reaching our objective. Conversely, there are many more statistical measures we have not used here that could be tried as well to see if they offer any advantage.

We have shown that the concept of intelligent algorithm recommendation *does*

work, which is exciting as it implies an end to the days of guessing and checking random algorithms. Future work could also include shifting the recommendation process farther back in the AutoML chain. While we were able to get suggestions for the algorithm to use, the work of tweaking and designing the neural net itself still involved trial and error. Perhaps it's possible to use metadata to recommend the design and features of the network itself? If meta-learning can be leveraged to automate the entire AutoML process, we could ensure maximum efficiency and maximize accuracy at a very small cost.

LIST OF REFERENCES

- [1] K. A. Smith-Miles, “Cross-disciplinary perspectives on meta-learning for algorithm selection,” *ACM Computing Surveys*, vol. 41, no. 1, pp. 1–25, 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1456650.1456656>
- [2] B. Taylor, V. S. Marco, W. Wolff, and Z. Wang, “Adaptive Deep Learning Model Selection on Embedded Systems,” *LCTES 2018 Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pp. 31–43, 2018.
- [3] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier, “Meta-Learning by Landmarking Various Learning Algorithms,” *ICML 00 Proceedings of the Seventeenth International Conference on Machine Learning*, vol. 951, no. 2000, pp. 743–750, 2000. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.1272{%&}rep=rep1{%&}type=pdf>
- [4] R. Vilalta and Y. Drissi, “A perspective view and survey of meta-learning,” *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.
- [5] X. He, K. Zhao, and X. Chu, “AutoML : A Survey of the State-of-the-Art,” *arXiv preprint arXiv:1908.00709*, no. August, 2019.
- [6] Z. Weng, “From Conventional Machine Learning to AutoML,” in *Journal of Physics Conference Series*, ser. Journal of Physics Conference Series, vol. 1207, Apr 2019, p. 012015.
- [7] H. Al-Sahaf, Y. Bi, q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, “A survey on evolutionary machine learning,” *Journal- Royal Society of New Zealand*, 05 2019.
- [8] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Frechette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, “Aslib: A benchmark library for algorithm selection,” *Artificial Intelligence*, vol. 237, 06 2015.
- [9] D. H. Wolpert and W. G. Macready, “No Free Lunch Theorems for Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.
- [10] M. F. Huber and M.-A. Zöller, “Survey on Automated Machine Learning,” *arXiv preprint arXiv:1904.12054*, no. May, 2019.

- [11] Z. Xu, L. Cao, and X. Chen, “Meta-learning via weighted gradient update,” *IEEE Access*, vol. 7, pp. 110 846–110 855, 2019.
- [12] I. Drori, L. Liu, Y. Nian, S. C. Koorathota, J. S. Li, A. Moretti, J. F. Freire, and M. Udell, “Automl using metadata language embeddings,” *ArXiv*, vol. abs/1910.03698, 2019.
- [13] B. Gaudet and R. Linares, “Adaptive guidance with reinforcement meta-learning,” 2019.
- [14] V. K. Verma, D. Brahma, and P. Rai, “A Meta-Learning Framework for Generalized Zero-Shot Learning,” *arXiv e-prints*, p. arXiv:1909.04344, Sep 2019.
- [15] F. G. Mohammadi, M. H. Amini, and H. R. Arabnia, “An introduction to advanced machine learning : Meta learning algorithms, applications and promises,” *ArXiv*, vol. abs/1908.09788, 2019.
- [16] L. Liu, T. Zhou, G. Long, J. Jiang, and C. Zhang, “Learning to propagate for graph meta-learning,” *ArXiv*, vol. abs/1909.05024, 2019.
- [17] P. Brazdil, R. Vilalta, C. Giraud-Carrier, and C. Soares, *Metalearning*. Springer, 01 2011, pp. 662–666.
- [18] R. Xu and Wunsch II, D. C., “{S}urvey of {C}lustering {A}lgorithms,” *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.
- [19] S. Priy. “Different types of clustering algorithm.” 2019. [Online]. Available: <https://www.geeksforgeeks.org/different-types-clustering-algorithm/>
- [20] D. Reynolds, *Gaussian Mixture Models*. Boston, MA: Springer US, 2015, pp. 827–832. [Online]. Available: https://doi.org/10.1007/978-1-4899-7488-4_196
- [21] P. Berkhin, “Survey of clustering data mining techniques,” *A Survey of Clustering Data Mining Techniques. Grouping Multidimensional Data: Recent Advances in Clustering.*, vol. 10, 08 2002.
- [22] M. Halkidi and M. Vazirgiannis, “A data set oriented approach for clustering algorithm selection,” in *PKDD*, 2001.
- [23] L. Morissette and S. Chartier, “The k-means clustering technique: General considerations and implementation in mathematica,” *Tutorials in Quantitative Methods for Psychology*, vol. 9, pp. 15–24, 02 2013.
- [24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*,

- ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3001460.3001507>
- [25] T. Zhang, L. Zhong, and B. Yuan, “A critical note on the evaluation of clustering algorithms,” 2019.
 - [26] M. C. P. de Souto, R. B. C. Prudencio, R. G. F. Soares, D. S. A. de Araujo, I. G. Costa, T. B. Ludermir, and A. Schliep, “Ranking and selecting clustering algorithms using a meta-learning approach,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 3729–3735.
 - [27] D. G. Ferrari and L. N. de Castro, “Clustering algorithm selection by meta-learning systems,” *Inf. Sci.*, vol. 301, no. C, pp. 181–194, Apr. 2015. [Online]. Available: <https://doi.org/10.1016/j.ins.2014.12.044>
 - [28] B. A. Pimentel, “Statistical versus Distance-Based Meta-Features for Clustering Algorithm recommendation Using,” *2018 International Joint Conference on Neural Networks (IJCNN)*, no. 2008, pp. 861–868, 2018.
 - [29] J. Vanschoren, “Meta-learning: A survey,” 2018.
 - [30] P. B. Brazdil, C. Soares, and J. P. da Costa, “Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results,” *Machine Learning*, vol. 50, no. 3, pp. 251–277, Mar 2003. [Online]. Available: <https://doi.org/10.1023/A:1021713901879>
 - [31] R. Bala, S. Sikka, and J. Singh, “A comparative analysis of clustering algorithms,” *International Journal of Computer Applications*, vol. 100, pp. 35–39, 08 2014.
 - [32] J. M. Johnson and T. M. Khoshgoftaar, “Survey on deep learning with class imbalance,” *Journal of Big Data*, 2019. [Online]. Available: <https://doi.org/10.1186/s40537-019-0192-5>
 - [33] M. Zakaria, M. AL-Shebany, and S. Sarhan, “Artificial neural network : A brief overview,” in *Artificial Neural Network : A Brief Overview*, 2014.
 - [34] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, “Openml: Networked science in machine learning,” *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013. [Online]. Available: <http://doi.acm.org/10.1145/2641190.2641198>
 - [35] S. Korkmaz, D. Goksuluk, and G. Zararsiz, “Mvn: An r package for assessing multivariate normality,” *The R Journal*, vol. 6, pp. 151–162, 12 2014.

BIBLIOGRAPHY

- Al-Sahaf, H., Bi, Y., Chen, q., Lensen, A., Mei, Y., Sun, Y., Tran, B., Xue, B., and Zhang, M., “A survey on evolutionary machine learning,” *Journal- Royal Society of New Zealand*, 05 2019.
- Bala, R., Sikka, S., and Singh, J., “A comparative analysis of clustering algorithms,” *International Journal of Computer Applications*, vol. 100, pp. 35–39, 08 2014.
- Berkhin, P., “Survey of clustering data mining techniques,” *A Survey of Clustering Data Mining Techniques. Grouping Multidimensional Data: Recent Advances in Clustering.*, vol. 10, 08 2002.
- Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., and Vanschoren, J., “Aslib: A benchmark library for algorithm selection,” *Artificial Intelligence*, vol. 237, 06 2015.
- Brazdil, P., Vilalta, R., Giraud-Carrier, C., and Soares, C., *Metalearning*. Springer, 01 2011, pp. 662–666.
- Brazdil, P. B., Soares, C., and da Costa, J. P., “Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results,” *Machine Learning*, vol. 50, no. 3, pp. 251–277, Mar 2003. [Online]. Available: <https://doi.org/10.1023/A:1021713901879>
- de Souto, M. C. P., Prudencio, R. B. C., Soares, R. G. F., de Araujo, D. S. A., Costa, I. G., Ludermir, T. B., and Schliep, A., “Ranking and selecting clustering algorithms using a meta-learning approach,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 3729–3735.
- Drori, I., Liu, L., Nian, Y., Koorathota, S. C., Li, J. S., Moretti, A., Freire, J. F., and Udell, M., “Automl using metadata language embeddings,” *ArXiv*, vol. abs/1910.03698, 2019.
- Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., “A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD’96. AAAI Press, 1996, pp. 226–231. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3001460.3001507>

- Ferrari, D. G. and de Castro, L. N., “Clustering algorithm selection by meta-learning systems,” *Inf. Sci.*, vol. 301, no. C, pp. 181–194, Apr. 2015. [Online]. Available: <https://doi.org/10.1016/j.ins.2014.12.044>
- Gaudet, B. and Linares, R., “Adaptive guidance with reinforcement meta-learning,” 2019.
- Halkidi, M. and Vazirgiannis, M., “A data set oriented approach for clustering algorithm selection,” in *PKDD*, 2001.
- He, X., Zhao, K., and Chu, X., “AutoML : A Survey of the State-of-the-Art,” *arXiv preprint arXiv:1908.00709*, no. August, 2019.
- Huber, M. F. and Zöller, M.-A., “Survey on Automated Machine Learning,” *arXiv preprint arXiv:1904.12054*, no. May, 2019.
- Johnson, J. M. and Khoshgoftaar, T. M., “Survey on deep learning with class imbalance,” *Journal of Big Data*, 2019. [Online]. Available: <https://doi.org/10.1186/s40537-019-0192-5>
- Korkmaz, S., Goksuluk, D., and Zararsiz, G., “Mvn: An r package for assessing multivariate normality,” *The R Journal*, vol. 6, pp. 151–162, 12 2014.
- Liu, L., Zhou, T., Long, G., Jiang, J., and Zhang, C., “Learning to propagate for graph meta-learning,” *ArXiv*, vol. abs/1909.05024, 2019.
- Mohammadi, F. G., Amini, M. H., and Arabnia, H. R., “An introduction to advanced machine learning : Meta learning algorithms, applications and promises,” *ArXiv*, vol. abs/1908.09788, 2019.
- Morissette, L. and Chartier, S., “The k-means clustering technique: General considerations and implementation in mathematica,” *Tutorials in Quantitative Methods for Psychology*, vol. 9, pp. 15–24, 02 2013.
- Pfahring, B., Bensusan, H., and Giraud-Carrier, C., “Meta-Learning by Landmarking Various Learning Algorithms,” *ICML 00 Proceedings of the Seventeenth International Conference on Machine Learning*, vol. 951, no. 2000, pp. 743–750, 2000. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.1272{%&}rep=rep1{%&}type=pdf>
- Pimentel, B. A., “Statistical versus Distance-Based Meta-Features for Clustering Algorithm recommendation Using,” *2018 International Joint Conference on Neural Networks (IJCNN)*, no. 2008, pp. 861–868, 2018.
- Priy, S. “Different types of clustering algorithm.” 2019. [Online]. Available: <https://www.geeksforgeeks.org/different-types-clustering-algorithm/>

- Reynolds, D., *Gaussian Mixture Models*. Boston, MA: Springer US, 2015, pp. 827–832. [Online]. Available: https://doi.org/10.1007/978-1-4899-7488-4_196
- Smith-Miles, K. A., “Cross-disciplinary perspectives on meta-learning for algorithm selection,” *ACM Computing Surveys*, vol. 41, no. 1, pp. 1–25, 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1456650.1456656>
- Taylor, B., Marco, V. S., Wolff, W., and Wang, Z., “Adaptive Deep Learning Model Selection on Embedded Systems,” *LCTES 2018 Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pp. 31–43, 2018.
- Vanschoren, J., “Meta-learning: A survey,” 2018.
- Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L., “Openml: Networked science in machine learning,” *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013. [Online]. Available: <http://doi.acm.org/10.1145/2641190.2641198>
- Verma, V. K., Brahma, D., and Rai, P., “A Meta-Learning Framework for Generalized Zero-Shot Learning,” *arXiv e-prints*, p. arXiv:1909.04344, Sep 2019.
- Vilalta, R. and Drissi, Y., “A perspective view and survey of meta-learning,” *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.
- Weng, Z., “From Conventional Machine Learning to AutoML,” in *Journal of Physics Conference Series*, ser. Journal of Physics Conference Series, vol. 1207, Apr 2019, p. 012015.
- Wolpert, D. H. and Macready, W. G., “No Free Lunch Theorems for Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.
- Xu, R. and Wunsch II, D. C., “{S}urvey of {C}lustering {A}lgorithms,” *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.
- Xu, Z., Cao, L., and Chen, X., “Meta-learning via weighted gradient update,” *IEEE Access*, vol. 7, pp. 110 846–110 855, 2019.
- Zakaria, M., AL-Shebany, M., and Sarhan, S., “Artificial neural network : A brief overview,” in *Artificial Neural Network : A Brief Overview*, 2014.
- Zhang, T., Zhong, L., and Yuan, B., “A critical note on the evaluation of clustering algorithms,” 2019.