Optimizing Recommendations for Clustering Algorithms Using Meta-Learning

Adam Jilling
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI, USA
ajilling@uri.edu

Marco Alvarez
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI, USA
malvarez@cs.uri.edu

***Abstract*—The field of machine learning has seen explosive growth over the past decade, largely due to increases in technology and improvements of implementations. As powerful as machine learning solutions can be, they are still reliant on human input to select the optimal algorithms and parameters. Clustering algorithms, in particular, are typically chosen 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 meta-features can then be used to intelligently recommend an optimal clustering algorithm without the cost of having to manually run the algorithm. To accomplish this, we will experiment using 135 datasets and determine their expected outcomes using only their meta-features. The outcomes being optimized are performance (accuracy) and runtime.

Results are then ranked separately for performance and runtime and we can determine how accurately the learning model was able to choose the optimal algorithm for each objective.

With respect to runtime, we are able to predict the top-performing algorithm 71.1% of the time, one of the top two algorithms 89.6% of the time, and an algorithm in the top three 93.3% of the time. Performance is correctly predicted in the top two 50.4% of the time and in the top three at a rate of 63.7%.

***Index Terms*—clustering, meta-learning, algorithm performance**

I. 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 exponentially as the number of parameters grows. If we can automate the process of algorithm selection, or even 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 are 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 numerous 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 many 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. Its 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 algorithm being chosen is the setting being adjusted. Since we expect the cost of meta-learning to be cheaper than the cost of training, the result is an increase in efficiency. We can also transfer our meta-knowledge to other datasets of similar types.

This paper 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 eight unique clustering algorithms to these datasets and measure their performance (accuracy) and runtime. These results will then be fed into neural networks to predict the performance and runtime for other datasets when using the

same eight 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.

If we measure success as predicting a result in the top two, our system has a success rate of 50.4% for performance and 89.6% for runtime. Using top three as a benchmark, success rates are 63.7% for performance and 93.3% for runtime.

The remainder of the paper is distributed as follows: Section II discusses related work and earlier studies, Section III details the methodology used in our experiment, Section IV visualizes and discusses the results, and Section V summarizes the benefits and any future work that could be done.

II. Related Work

We begin with an overview of AutoML, clustering and its various implementations, and how to apply AutoML to complete clustering tasks. We also present a brief overview of deep learning and neural networks.

A. AutoML

The full 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 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 paper, the structure that we aim to exploit is defined by the metafeatures of each dataset.

Meta-learning aims to improve average performance on new tasks by utilizing experience in past tasks [11], [12] and 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]–[16].

B. 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 [17]. There are many methods that can accomplish this, known as clustering algorithms. Each algorithm is usually classified by how it accomplishes the clustering [18]. Some families of algorithms include:

a) Distribution Models: Data points are modeled based on the probability that they fit into a particular cluster. The number of clusters used is fixed and predefined. Each item is assigned to the cluster to which it has the highest probability of belonging [19]. Gaussian Mixture Models are examples of distributed clustering algorithms. Figure 1(a) shows a visualization of a distribution model.

b) 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 until reaching the specified number of clusters. 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. Figure 1(b) shows Average Agglomerative Clustering, a connectivity model [20].

c) Centroid Models: Sometimes called partitional models, 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 [21]. Centroid models include k-means and fuzzy c-means, as seen in Figure 1(c) [22].

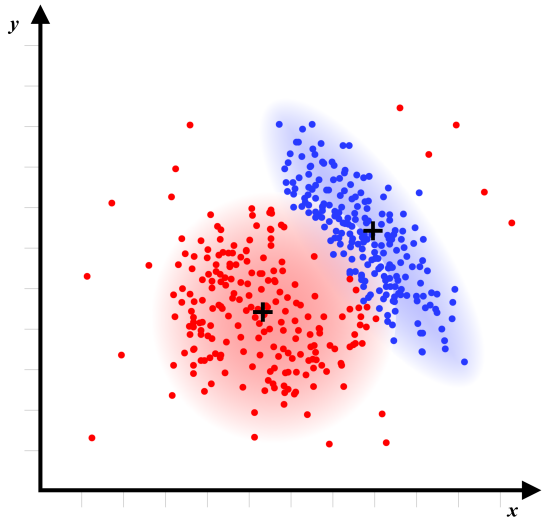
d) Density Models: The data space is scanned for areas of varying density and partitions are made where the density is lower, signifying the edges of a cluster. The number of clusters is not pre-defined. Density-based spatial clustering of applications with noise (DBSCAN) and Mean Shift are two well-known density-based algorithms [23]. Figure 1(d) shows a visualization of a density model.

Though there are other families of algorithms, these four cover most of the algorithms used in this work.

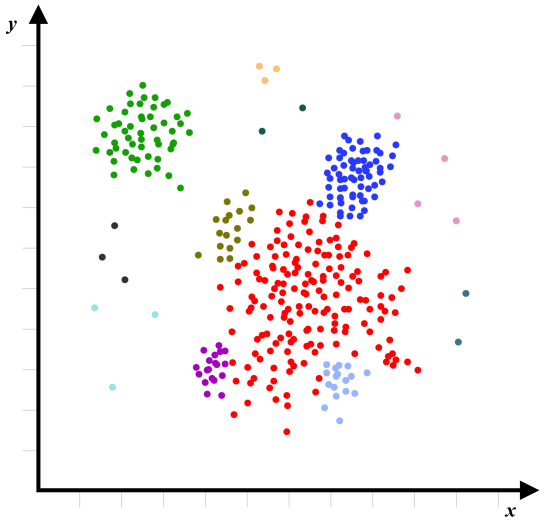
C. AutoML Applied to Clustering

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.

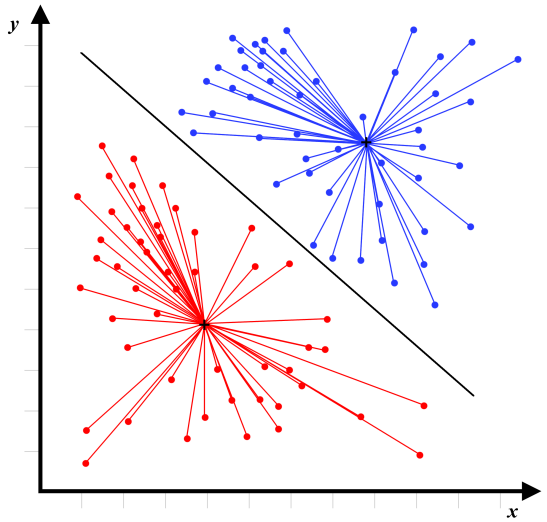
1) Performance Optimization: The most common optimization goal is for accuracy, usually referred to as performance. Performance optimization aims to maximize the number of data points that are assigned to their correct cluster. There are many metrics that attempt to evaluate this in different ways. Table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



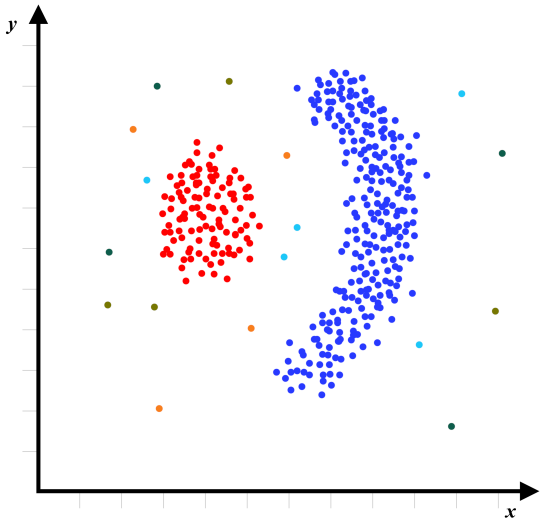
(a) Visualization of expectation-maximization (EM), a distribution model, which uses multivariate normal distributions. Each centroid is marked with a (+).



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



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



(d) Visualization of the 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.

Fig. 1: Visualizations of different approaches for clustering used in this work.

Although clustering is an unsupervised task, performance is often evaluated incorrectly by using the clustering labels as the prediction objective. As pointed out by [24], 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. Conversely, objects with the same class labels might correspond to multiple clusters. For these reasons, in this work, all class labels are dropped from each dataset and we 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 leverage a commonly used technique of specifying that the number of desired clusters be equal to the number of attributes in the dataset.

TABLE I: Clustering performance metrics used in our experiments, each column shows the package and language used to implement, 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

The meta-learning approach to clustering algorithm recommendation was used by [25] 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 [25].

The authors in [26] 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 versus 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" [26].

The study done in [27] 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.

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 [28]. 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 that improvements in runtime aren't completely at the expense of accuracy.

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 [29].

D. 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.

As the ANN is iterated over, a matrix multiplication is performed on each layer based upon 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 [30].

Two ANNs are used in this work, one to predict runtime and one to predict performance. The network to predict runtime has a single output, the expected runtime, while the ANN for performance will output ten distinct performance metrics. Both networks will accept an input containing twenty-five metafeatures and a one-hot encoding representing each of the eight algorithms.

III. Methodology and 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 2 shows a diagram of the entire process.

A. Datasets and Preprocessing

OpenML [31] is a project that provides, among other things, datasets to use in machine learning projects. We use 135 datasets from OpenML, covering a wide range of categories including medical, biological, climatic, and social topics. This library of datasets was mostly compiled and used by the study in [27], 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. Columns are colinear if they are linear combinations of others (either exact or close). This will cause errors when running multivariate analysis so all such columns need to be removed. It is important to note that removing these columns will not affect the predictive power of our model as we are essentially just removing duplicate features. Some sets are found to be *computationally* singular if they have one or more small

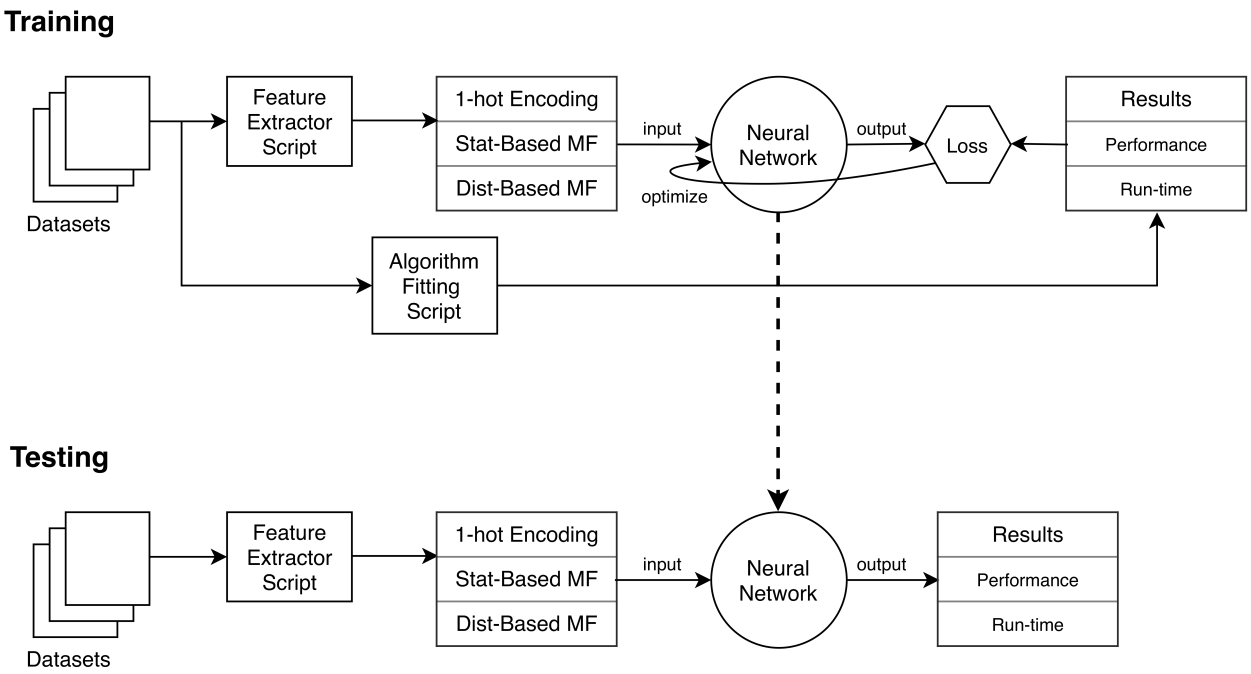


Fig. 2: Overview of the entire process showing feature extraction, fitting, tensor building, training, testing, and output.

values that 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.

B. 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 [25] proposed the use of eight statistical metafeatures. The study in [27] built upon that method, dropping two of the features for being too subject-specific, as the goal is for this to generalize over datasets of all types. They also built upon the work of [26], 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.

1) Statistical-Based Metafeatures: These 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)
   $NE = n$, where n is the number of entries. This indicates the size of the dataset.
2) Number of Entries per Attribute (NEA)
   $NEA = \frac{n}{p}$, where n is the number of entries and p is the number of attributes. This indicates the robustness of the dataset, or how descriptive it is.
3) Percentage of Missing Values (PMV)
   $PMV = \frac{m}{t} \cdot 100$, where m is the number of missing entries and t is the total number of entries. This measures the completeness of the dataset.[1]
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 [32] and Royston's algorithm.
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 to compute multivariate skewness.
6) Percentage of Outliers (PO)
   $PO = \frac{o}{t} \cdot 100$, where o is the number of entries that are labelled as outliers, meaning they are more than two standard deviations from the mean and t is the total number of entries. This is a multivariate metric.

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

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.

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

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

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

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 II.

TABLE II: 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 $[3, \infty)$

For larger datasets, pairwise distance calculations could become prohibitively expensive, so these metafeatures are best suited for smaller to average-size datasets.

C. Recording Algorithm Results

Each dataset is normalized to the interval $[0, 1]$. A widely-used method of dealing with an unknown number of clusters is to set the number of clusters equal to the number of classes in the dataset. This method is used for algorithms that require a set number of clusters. 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 [17]. Eight algorithms will be run, all from Python's *scikit-learn* package, shown in Table III. Each algorithm will be measured for both performance and runtime.

TABLE III: 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
AP	Affinity Propagation	max_iterations=1000, convergence_iterations=10
BI	Birch	threshold=0.01, branching_factor=50
GM	Gaussian Mixture	covariance_type=full, n_init=5
KM	K-Means	init=k-means++, n_init=20, max_iterations=500
MS	Mean Shift	quantile=0.2, cluster_all=False
OP	OPTICS	min_samples=5, metric=minkowski
SC	Spectral Clustering	n_init=20, affinity=rbf, assign_labels=kmeans

1) Performance Data: To calculate performance (accuracy), we use the ten clustering metrics shown in Table I. As 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. The combining and averaging step is done after all ten results are returned from the neural network. Metrics with a minimization objective are flipped by multiplying by -1 to put all metrics on equal footing. Since the objective is to compare among eight algorithms, the actual numeric result is irrelevant, as long as it is consistent among all eight, allowing us to rank relative to one another.

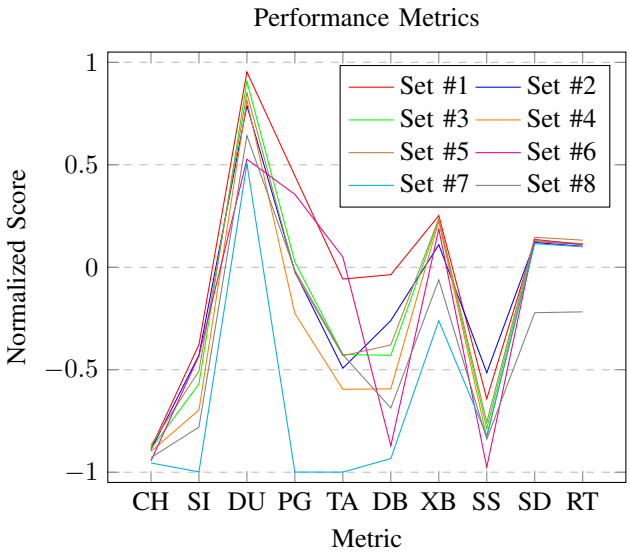
Figure 3 shows the first eight datasets and each of their ten performance metrics. We can see that, in general, all ten metrics seem to agree with each other, so taking the average of all ten should not be an issue. If the graph showed random, inconsistent results, that would be a concern.

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) 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 efficiently 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.

D. Neural Net Training/Testing

We create two neural networks and use leave-one-out cross validation (LOOCV). The input for both will be the metafeatures and a one-hot encoding of the desired algorithm. The output will be the performance predictions for one, and the runtime prediction for the other.

Fig. 3: The first eight datasets and each of their corresponding performance metrics.



There are 1080 input tensors (135 datasets $\times$ 8 algorithms), each with 33 features, shown below in Figure 4. Each time, the ANN is trained with 134 datasets and tested on the held out set. Since LOOCV can potentially have a high variance, we run ten iterations and take the average, while also ensuring the variance is within a reasonable threshold.

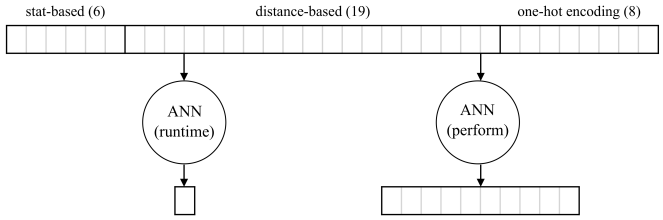


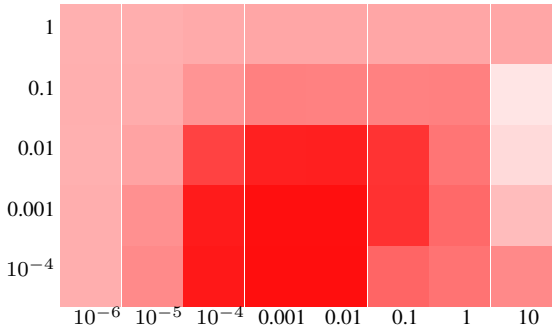
Fig. 4: Diagram showing dimensions of tensors used. Input tensors contain thirty-three values - twenty-five metafeatures and eight values forming a vector to represent each dataset. The output is one value for runtime and ten values for performance.

The process of designing the hidden layers of the ANNs, as previously mentioned, is somewhat of an inexact science. After a lot of testing and tweaking based on feedback and data from trial runs, the runtime network is built with three hidden layers of sizes $32, 24$, and 8. The first two hidden layers use a Leaky ReLU activation function. The third hidden layer uses a sigmoid activation, preventing any negative values as sigmoid, by its nature, produces outputs in the range $(0, 1)$. The performance ANN contains three hidden layers of sizes $32, 24$, and 16, all using Tanh activations.

We begin manually training the networks, and after each pass the average training loss and average testing loss are recorded. We then use a 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.

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

TABLE IV: 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.



A quick look at the heatmaps 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. Using this information, we select a weight decay of 0.001 and a learning rate of 0.001.

Once built, the ANNs are run on all 135 datasets using LOOCV, and all performance and runtime outputs are recorded and output to a text file. Rankings are calculated and then analyzed to obtain a measure of effectiveness.

IV. Results and Analysis

Here we cover the process of analyzing each dataset's results in aggregate. We then use this information to calculate accuracy rates and to provide visuals.

After all numerical values are ranked, we end up with a data structure for each dataset, shown in Table V.

TABLE V: Example of the result structure produced for a single dataset (#8). The performance and runtime results are ranked based on their predicted values and the actual values obtained when run.

		AA	AP	BI	GM	KM	MS	OP	SC
Performance	**Pred**	8	4	7	5	6	1	3	2
	Actual	8	2	4	6	7	3	1	5
Runtime	**Pred**	1	4	2	7	3	8	6	5
	Actual	1	3	2	6	4	8	7	5

In this example we can see (*in blue*) that the predicted top-performing algorithm was Mean Shift (MS), when in fact it was actually the third-best performing. The top-performing was OPTICS (OP). The predicted top-runtime algorithm (*in red*) was Average Agglomerative (AA) and that was indeed the actual top-runtime algorithm.

Since the goal of this project is to identify the top ranked algorithms for each objective, we will focus on all results in the top three. Table VI compares the predictions 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 28.9% of the time, the top performing algorithm was predicted to be in the top two 50.4% of the time, and the actual top performing algorithm was predicted to be in the top three 63.7% of the time.

TABLE VI: 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	28.9%	71.1%
Top 2	50.4%	89.6%
Top 3	63.7%	93.3%

We can also examine the results on a per-rank basis. The figures below each look at a predicted ranking and chart its corresponding actual ranking. For example, in Figures 5, 6, and 7 we look at the top predicted algorithm for each of the top three rankings. For ranking #1, we can see that 39 times, the top predicted algorithm was the top actual algorithm, 29 times the top predicted algorithm was the second best performing, and so on. We can even see that in three cases, the top predicted performer was actually the worst performing. We would hope 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 8 shows that, with respect to performance, other than Average Agglomerative (AA), the algorithms are fairly evenly distributed. Birch (BI) and K-Means (KM) were the next worst performing, while Spectral Clustering (SC) and OPTICS (OP) were the two best. The specialized nature of Birch makes it difficult to generalize. For the purposes of this study, we had to use the same threshold and branching factor across all datasets.

Figure 9 tells a different story, as we can see that certain algorithms consistently have quicker runtimes than others. Average Agglomerative (AA) was easily the fastest running, which may explain its poor performance. The second fastest running, Birch (BI), was also the second worst performing. A quick comparison of both charts shows a pretty clear inverse relationship between performance and runtime, which makes sense. If better results are desired, there will almost always be an additional cost.

Fig. 5: Results attempting to predict ranking #1. The x-axis shows the actual results and how many times each value was predicted.

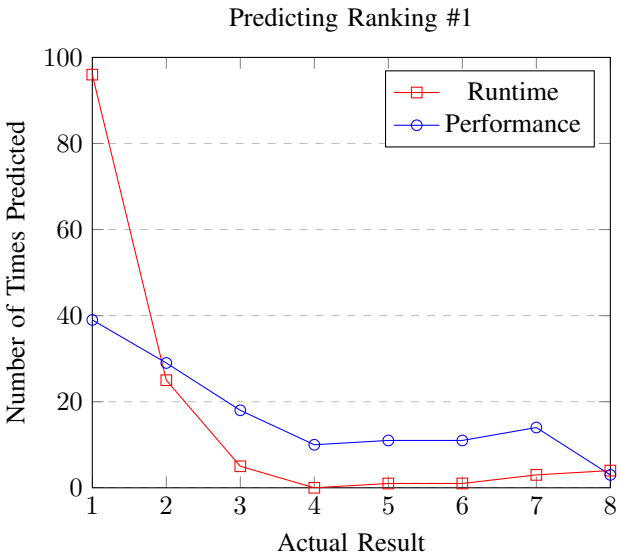
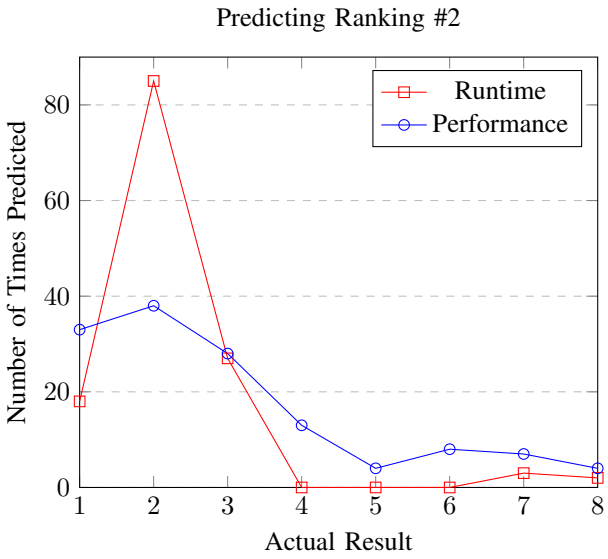


Fig. 6: Results attempting to predict ranking #2. The x-axis shows the actual results and how many times each value was predicted.



V. 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.

With respect to runtime, our meta-learning system was able to predict the top algorithm over 70% of the time. It was able to recommend one of the top two algorithms almost 90%

Fig. 7: Results attempting to predict ranking #3. The x-axis shows the actual results and how many times each value was predicted.

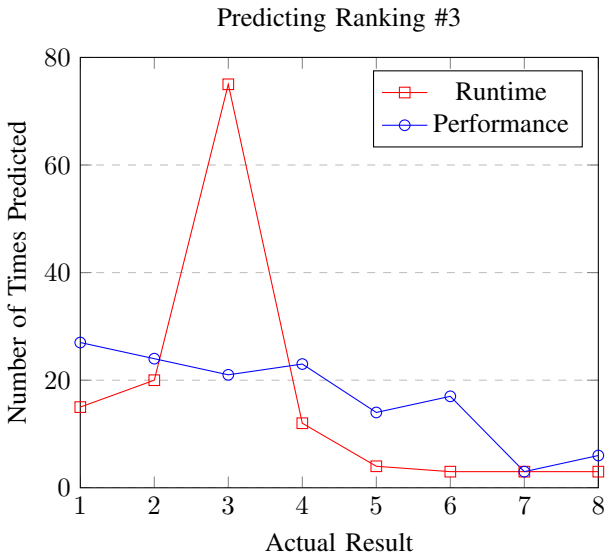
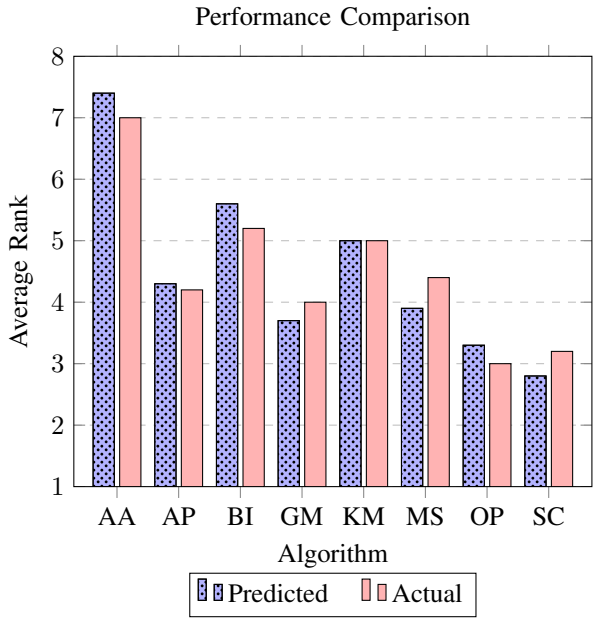


Fig. 8: Difference between the predicted and actual performance average ranking for each algorithm.

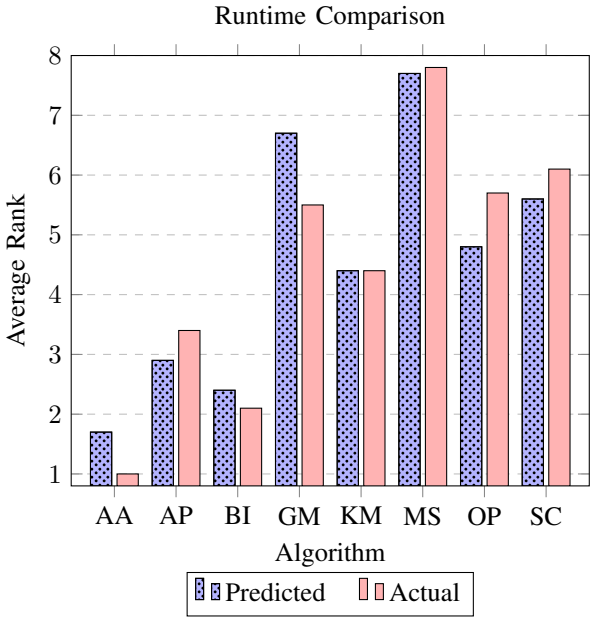


of the time, and in over 93% of cases, the system was able to recommend one of the top three algorithms. If we define success as being in the top three, the system was *unsuccessful* in only 6.7% of cases.

When optimizing for performance, the system was able to identify the top algorithm almost 29% of the time and one of the top three algorithms about 64% 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

Fig. 9: Difference between the predicted and actual runtime average ranking for each algorithm.



relevant towards reaching our objective. Conversely, there are more statistical measures not used here that could be tested to see if they offer any advantage. 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 nets themselves still involved trial and error.

In summary, we have shown that the concept of intelligent algorithm recommendation *does* work, which is exciting as it has the potential to bring an end to the days of guessing and checking randomly selected algorithms. If meta-learning can be leveraged to automate algorithm selection, we can maximize efficiency and accuracy at a much smaller cost than present methods.

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] 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.

[18] S. Priy. (2019) Different types of clustering algorithm. [Online]. Available: https://www.geeksforgeeks.org/different-types-clustering-algorithm/

[19] 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

[20] 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.

[21] M. Halkidi and M. Vazirgiannis, "A data set oriented approach for clustering algorithm selection," in *PKDD*, 2001.

[22] 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.

[23] 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

[24] T. Zhang, L. Zhong, and B. Yuan, "A critical note on the evaluation of clustering algorithms," 2019.

[25] 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.

[26] 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

[27] 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.

[28] J. Vanschoren, "Meta-learning: A survey," 2018.

[29] 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.

[30] M. Zakaria, M. AL-Shebany, and S. Sarhan, "Artificial neural network : A brief overview," in *Artificial Neural Network : A Brief Overview*, 2014.

[31] 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

[32] 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.