EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

Eindhoven University of Technology

BACHELOR

Better clustering evaluation for the OpenML Evaluation Engine

Kaandorp, Otto J.

Award date:
2022

Awarding institution:
Tilburg University

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

University of Technology Eindhoven
The Department of Mathematics and Computer Science

Bachelor Final Project

Better clustering evaluation for the OpenML Evaluation Engine

Otto Kaandorp - 1508865
o.j.kaandorp@student.tue.n

supervised by:
J. Vanschoren
P. Gijsbers
P. Singh

June 26, 2022

Abstract - This report talks about the creation of a Python based evaluation engine for OpenML. The creation of this evaluation engine in Python is necessary for OpenML to remain relevant. Furthermore, the report discusses the applications of the Pareto optimal front for clustering evaluation. It looks at the possibility of using the Pareto front to evaluate clustering models and to find the optimal combination of metrics for clustering model evaluation. To do this, a set of models was made and evaluated with a set of metrics. The results of metrics were then plotted against each other and using those points Pareto fronts were constructed. To find useful combinations of clustering metrics, the Pareto hypervolume and the amount of Pareto efficient points were analysed. However, in the end these methods for metric selection did not turn out to be a better alternative to the use of correlation. Using the Pareto front it self for clustering evaluation turned out to be more promising. The plotted fronts can give OpenML users a lot more insight into how models perform a certain task. It is a lot harder to quantify this result though. These results could be used by the OpenML team to improve their website and evaluation engine.

Contents

1 Introduction **3**
1.1 Research questions . . . 3

2 Background information **4**
2.1 Basic Idea of the evaluation engine . . . 4
2.2 Clustering and evaluation . . . 4
2.3 The Pareto front . . . 4

3 Methodology **5**
3.1 Creating the evaluation engine . . . 5
3.1.1 Package structure . . . 5
3.1.2 Testing . . . 6
3.1.3 Documentation . . . 6
3.2 Clustering metric selection . . . 6
3.2.1 Preparation and models . . . 6
3.2.2 Evaluation . . . 7
3.2.3 Scaling . . . 8
3.2.4 Correlation analysis . . . 9
3.2.5 Analysis with the Pareto front . . . 9

4 Results **11**
4.1 Evaluation engine . . . 11
4.2 Clustering metric selection . . . 11

5 Discussion **12**
5.1 Limitations . . . 12

6 Conclusion **12**
6.1 Recommendations . . . 13

A Appendix **15**
A.1 Personal contribution . . . 15
A.2 tables . . . 15
A.3 Pareto front code . . . 16

1 Introduction

OpenML is an online Machine Learning(ML) platform, to which people can upload datasets, ML models, pipelines and more. An OpenML team member describes the idea of the platform as follows: 'The core idea is to have a single repository of datasets and results of ML experiments on them.' [7]. Using the provided datasets and tasks, people can perform their own machine learning experiments, and compare their results to those of others on the platform. This can be very useful when studying-, experimenting with-, and working with- machine learning.

The machine learning experiments are uploaded in the format of runs. Creating such a run is done as follows. Firstly, OpenML creates a task, which specifies the train and test data and what should be returned by the model. Secondly, an OpenML user downloads the task and applies their machine learning algorithm on it. Any machine learning algorithm. Lastly, the user publishes their run, which uploads their predictions to the OpenML server. When a run is published, The evaluation engine is executed on the incoming results. This evaluation engine is a small software library that assesses results with a series of evaluation metrics such as accuracy, precision etc. After this, the results of these metrics are automatically uploaded to the OpenML server.

Currently the evaluation engine is implemented in Java, but to make sure it stays relevant within the machine learning community it needs to be rewritten in Python. OpenML components are being rewritten in Python because it is more modern and it is starting to become a more widely used programming language than Java [2] [11]. Rewriting the evaluation engine in Python will be the focus of this project. The end goal of the project is a fully functional evaluation engine for OpenML that is well-documented, extensible, practical, and scalable.

In addition to the creation of the evaluation engine, this report will also focus on finding a better way to evaluate clustering tasks. To do this, it will be determined which metrics have an interesting trade-off and deviate the most from each other for a certain clustering task. Meaning, a 'good' score for one of these metrics often causes a 'bad' score for the other. To find these 'most interesting' metrics, an approach using the Pareto optimal front will be explored. Using a combination of the found metrics can provide a better way of evaluating the model. Furthermore, the Pareto front can help give more insight into how well a model performs, because clustering metrics are often not as straight forward as a percentage of correct predictions. In turn, this could help the users of OpenML create better models.

1.1 Research questions

For the analysis with the Pareto front a research question is defined, together with two sub-questions.

Research question: How can the Pareto front by implemented by OpenML to improve the evaluation of clustering metrics?
Sub-question 1: How can the Pareto front be used for clustering metric selection?
Sub-question 2: How can the Pareto front it self be used for clustering evaluation?

2 Background information

2.1 Basic Idea of the evaluation engine

The idea of the evaluation engine is to have a Python software package/ library, that can easily be used to evaluate machine learning models. In turn, the OpenML website will then be able to publish the evaluations, using this package. The package has to be a standalone product, meaning more parties than just the OpenML website must be able to use it. Outside parties that wish to use the package will still have to use parts of the existing OpenML library though, as the package is designed to evaluate OpenML runs. To ensure the ability for others to use the package, it will be open source. Because the package is designed to specifically evaluate OpenML runs, it will have to communicate with OpenML servers to receive and publish data. This communication will run through the existing OpenML Python API.

In addition to the OpenML software package, the evaluation engine package will also build on a variety of different existing libraries such as: pandas, numpy and scikit-learn. The use of established packages will help improve extensibility and maintainability, and it will make the package easier to understand. This is because it allows the package to use functions that are known to be correct and efficient. Furthermore, it prevents extensive coding to define and update functions from scratch.

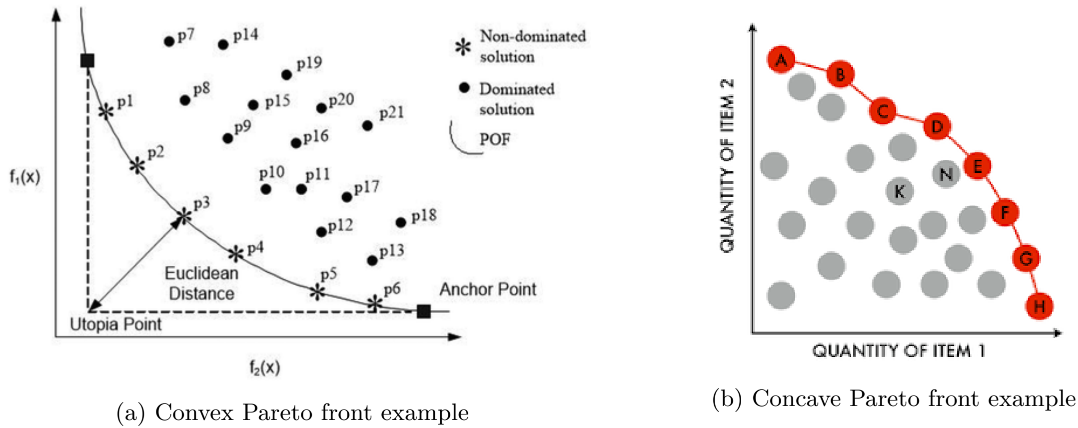
2.2 Clustering and evaluation

Clustering can be described as follows: 'clustering, or cluster analysis, refers to the process of organizing objects into groups whose members are similar with respect to a similarity or distance criterion. As such, a cluster is a collection of similar objects that are distant from the objects of other clusters. Unlike most classification techniques that aim to assign new observations to one of the many existing groups, clustering is an exploratory procedure that attempts to group objects based on their similarities or distances without relying on any assumptions regarding the number of groups.' [5]

Clustering falls under unsupervised learning, which means that, during training, models do not have access to the variable they are trying to predict. Furthermore, the true values of this target variable sometimes do not even exist. This creates the need for other types of metrics that also don not require a target variable to be present. Interpretation of those types of metrics is often less intuitive. The book 'Introduction to Information Retrieval' gives a general description of such metrics: "Typical objective functions in clustering formalize the goal of attaining high intra-cluster similarity (documents within a cluster are similar) and low inter-cluster similarity (documents from different clusters are dissimilar). This is an internal criterion for the quality of a clustering." [6]

2.3 The Pareto front

During multi-objective optimization (MOO) there is always a trade-off between a set of objective functions. For example, making an algorithm more accurate also makes it slower and vice versa. A situation, in which increasing one objective function is not possible without decreasing another, is called a non-dominated solution or Pareto efficient. "A solution where an objective function can be improved without reducing the objective function of the other is called non-Pareto optimal solution" [4]. The Pareto front is the set of Pareto efficient solutions to the optimization problem. The best of an objective function is given on a graph as an anchor point. The best possible situation is given by the utopia point. The point on the Pareto front that is closest to the utopia point, is considered to be the optimal value of the Pareto front (point 'p3' in figure 2.1a). Depending on the scaling of the metrics and preference of the creator, the front can be concave or convex. A visualization of such a convex Pareto front can be seen in figure 21a. A visualization of a concave Pareto front can be seen in 2.1b.
Looking at the curve of the Pareto front can help show how correlated objective functions are. In addition, a steep curve makes it fairly easy to find the optimal solution, while a more gradual trade-off might require the user to think more about how they want the model to perform. Therefore, if evaluation metrics were taken as objective functions, the Pareto front could help select clustering evaluation metrics and optimize models.



(a) Convex Pareto front example

(b) Concave Pareto front example

Figure 2.1: Examples of Pareto fronts

3 Methodology

3.1 Creating the evaluation engine

During the development of the machine learning engine there are a few important factors to take into account. The engine needs to be well-designed, well-implemented, well-tested, well-documented, scalable, and extensible. These key points were taken into account throughout the creation of the evaluation engine and have helped achieve a working Python package.

3.1.1 Package structure

The package loads all of the data from a certain run into a placeholder object, using the existing OpenML Python API. This placeholder object firstly saves all of the data from the run, for example predictions, true values, and row-indices. Secondly, the placeholder object stores a lot of information provided by OpenML, like run ID, task type, etc. The object it self is created using a Python class that takes a run ID as argument. Furthermore, this class also contains the function used to run evaluation metrics for the inputted run. Running this function outputs all of the relevant metrics for the inputted OpenML run. The results of all metrics can be outputted at once, or the result of one specific metric can be called. To do this all of the metrics are categorized by what type of task they can be used for.

Next to the placeholder object class, there is a second very important class in the package. Namely, the class containing all of the evaluation metrics. This evaluation metrics class is a child class to the placeholder object class (Parent class). That means the class inherits all of the properties and methods from the placeholder object class. The evaluation metrics class inheriting all of these properties and methods, means it is very easy to use data from the loaded run. The metric functions can then very easily be applied to the data.

To maintain consistency in the structure of the evaluation engine and to keep the structure as simple as possible, Cookiecutter is used. According to the website Cookiecutter works roughly as follows: It takes an example directory and a settings file that have been provided and then outputs a directory structure [10]. The structure provided by Cookiecutter was used to make a start to the project and it is used to keep consistency throughout development. A consistent structure is key for the extensibility and scalability of the package.

To ensure that the package does not become extensively hard coded and to make sure it remains easily updatable, metrics from established packages are used as much as possible. These metrics largely originate from the scikit-learn package. For the rest a variety of libraries, like pandas and arff, are used for data loading and processing. scipy and numpy are used for statistical and mathematical operations

3.1.2 Testing

To make sure the software package is robust and has consistent output, extensive testing is performed. This testing is done with the Pytest package. Figure 3.1 shows the structure in which the testing is performed.
Firstly, the Placeholder object is tested. The test consists of multiple checks to see if the object's attributes are valid, for example, to see whether a run's task type is supported by OpenML. Once the placeholder testing is complete, a test on the evaluation metrics class is performed to see which metrics have been implemented. This is necessary because all of the metrics from the OpenML website have been included in this class, but not all of them are fully implemented. Checking implementation is very straight forward, since all unimplemented metrics will raise a 'not implemented' error when they are used. With this second test, a list can be constructed of all implemented metrics that can be tested further. Lastly, a test is performed to see whether the metrics of the new evaluation engine have the same output as those of the old evaluation engine. To do this, the test iterates over random runs and evaluates them until all metrics have at least been run once. Iterating over runs is necessary because not all metrics are compatible with all task types. Iterating over random runs makes it so that runs from all task types can be evaluated.

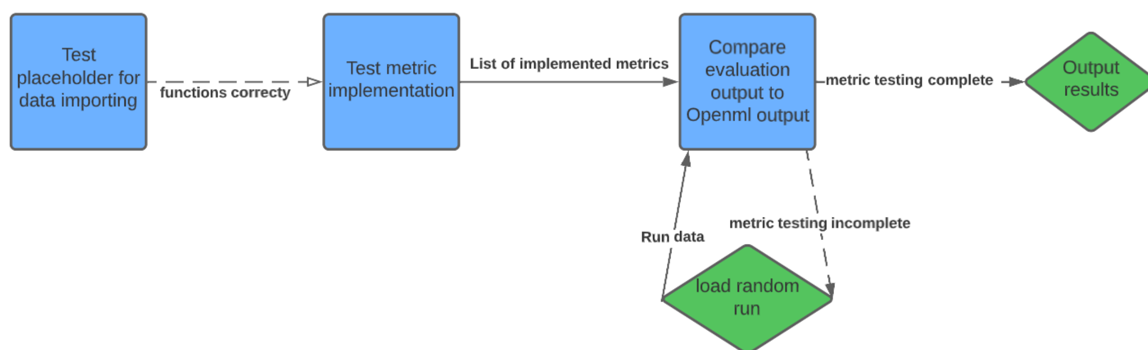


Figure 3.1: Structure of performed pytests

3.1.3 Documentation

Documentation is mainly done using docstrings. These docstrings contain a short description of the function or class, the input it takes and the output it gives. This makes it easiest for others to understand and use these functions. Using docstrings also conforms to the coding standards required by OpenML.
To make sure all documentation can be found in one place and to make it easily accessible, sphinx is used. "Sphinx is a tool that makes it easy to create intelligent and beautiful documentation" [1]. Sphinx allows the output of the package's documentation in HTML, LaTeX, ePub, Texinfo, manual pages, and plain text.

3.2 Clustering metric selection

When working on clustering tasks either one evaluation metric or a combination of multiple metrics can be used to evaluate the model. Using a combination of metrics that have a large correlation will be very uninformative, as the output of both metrics will change similarly when the model is altered. This also means that optimizing the model, using highly correlated metrics, might lead to a sub optimal model. Therefore, a combination of metrics that have a small correlation will work best for effective evaluation and optimization of the model. In this section the selection process of the metrics will be discussed

3.2.1 Preparation and models

Throughout the duration of this Bachelor End Project, only one existing clustering run was available on the OpenML website. Furthermore, it was impossible to create new clustering runs or tasks with the OpenML API due to errors in the OpenML system. This meant fitted clustering algorithms had to be

created and evaluated 'manually'. The models had to cluster the iris dataset. This dataset can easily be imported into any Python project, using the Scikit-learn package. The iris dataset contains information on different types of irises, a type of flower. The datasets contains 'Sepal Length', 'Sepal Width', 'Petal Length' and 'Petal Width' as variables and the actual type of iris as target variable. A few different types of algorithms were used to perform the clustering task on this dataset. Also for each different algorithm a few models were created with different parameters. A table of all algorithms and their sets of parameters can be found in the appendix in table A.1.

The first algorithm is 'K-Means clustering', which consists of the following 3 steps according to Scikit-learn: "The first step chooses the initial centroids, with the most basic method being to choose k samples from the dataset X. After initialization, K-means consists of looping between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all of the samples assigned to each previous centroid. The difference between the old and the new centroids are computed and the algorithm repeats these last two steps until this value is less than a threshold. In other words, it repeats until the centroids do not move significantly." [9].

The second clustering algorithm used is DBSCAN (Density-Based Spatial Clustering of Applications with Noise). Unlike k-means clustering, which only really allows clusters to be convex shaped, DBSCAN allows clusters to have any shape. This is because DBSCAN views clusters as areas of high density separated by areas of low density. According to DBSCAN, a cluster is a set of core points combined with set of non-core points that are close to a core point. To be considered a core point, a sample needs to be less than Epsilon away from a minimum amount of other points. Epsilon and the minimal amount of other points are parameters defined during creation of the model. Epsilon stands for the maximum distance between two points. The number of clusters does not need to be specified before running the model. The algorithm works by iterating over all points and checking whether it is a core point, non-core points are labelled as noise. If a sample is a core point, the algorithm iterates over all points that are close to the core point. Noise close to the core point will simply receive the same label as the core point. If a point close to the newly discovered core point is a core point as well, the algorithm will recursively iterate over all points close to that core point.

The third clustering method that was used is the MeanShift algorithm. MeanShift aims to find clusters in smooth distributions of samples by finding local density maxima. To do this a kernel moves through the feature space. The kernel moves by moving it's centre to the mean of all of the points within the kernel. The kernel no longer moving means that a candidate centroid has been found. After the discovery of all candidate centroids, post-processing is performed. During this process, candidate centroids that are too close to each other are removed so that only one of them remains. The remaining candidates form the final set of cluster centroids.

The last used clustering algorithm is Affinity Propagation. This is a graph based clustering algorithm. It does not require the number of clusters to be specified before running the algorithm. The functioning of the algorithm can, according to Scikit-learn be described as: "AffinityPropagation creates clusters by sending messages between pairs of samples until convergence. A dataset is then described using a small number of exemplars, which are identified as those most representative of other samples. The messages sent between pairs represent the suitability for one sample to be the exemplar of the other, which is updated in response to the values from other pairs. This updating happens iteratively until convergence, at which point the final exemplars are chosen, and hence the final clustering is given."[9]

3.2.2 Evaluation

Because of the nature of unsupervised learning, it is possible to fit a model on the same data you are predicting. Unsupervised learning methods will not run into the same over-fitting issues that occur during supervised learning. This is because, during unsupervised learning, the target value is not provided. For the scope of this project, that means it is fine to predict the labels of the data on which the model was fitted. Which is exactly what was done. Each model's predictions were put into a data frame. Each column in this data frame represents a model. This allows for easy comparison between the actual labels and the assigned labels.

All Scikit-learn clustering metrics, that could be run with just true values and predictions, were ran on the predictions data frame and the target values. This created the list of used metrics seen in figure 3.2. The descriptions of these metrics are provided by the Scikit-learn website [8]. The outputs of these metrics were stored in a data frame as well. This data frame was constructed as follows. Each metric was run on the predicted labels of all models, forming a list of all results of that metric. Such a list was then added to the data frame in a column named after the metric. The final data frame then contains all metrics with their outputs for all models.

Metric	Description
'adjusted_mutual_info_score'	Adjusted Mutual Information (AMI) is an adjustment of the Mutual Information (MI) score to account for chance. It accounts for the fact that the MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared.
'adjusted_rand_score',	Rand index adjusted for chance.
'completeness_score'	A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster.
'fowlkes_mallows_score'	The Fowlkes-Mallows index (FMI) is defined as the geometric mean between of the precision and recall.
'homogeneity_completeness_v_measure'	Compute the homogeneity and completeness and V-Measure scores at once.
'homogeneity_score'	Homogeneity of a cluster labelling given a ground truth. A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class.
'mutual_info_score'	The Mutual Information is a measure of the similarity between two labels of the same data.
'normalized_mutual_info_score'	Normalized Mutual Information (NMI) is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation).
'pair_confusion_matrix'	The pair confusion matrix C computes a 2 by 2 similarity matrix between two clusterings by considering all pairs of samples and counting pairs that are assigned into the same or into different clusters under the true and predicted clusterings. Considering a pair of samples that is clustered together a positive pair, then as in binary classification the count of true negatives is C_{00}, false negatives is C_{10}, true positives is C_{11} and false positives is C_{01}.
'rand_score'	The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings.
'v_measure_score'	This score is identical to 'normalized_mutual_info_score' with the 'arithmetic' option for averaging. The V-measure is the harmonic mean between homogeneity and completeness.

Figure 3.2: Table of used metrics with description

3.2.3 Scaling

To find out which metrics could be used for further analysis, the type and range of the metric outputs need to be checked. All columns are checked to confirm they only contain numbers between 0 and 1. The algorithm that performs these checks, outputs the names of the columns that do not pass them. These columns can then be scaled or removed form the data frame. In this case, three metrics were returned: 'homogeneity_completeness_v_measure', 'mutual_info_score', and 'pair_confusion_matrix'.

The 'homogeneity_completeness_v_measure' simultaneously returns three metrics, that have also been included in the evaluation individually. Therefore, it does not make sense to scale this metric and include it in the analysis.
The 'mutual_info_score' metric had outputs greater than one, which would mean it had to be scaled. However, a metric that was also computed is the 'normalized_mutual_info_score', which is the scaled version of the 'mutual_info_score'. This means it is also unnecessary to perform scaling on the 'mutual_info_score', as it would just lead to a duplicate column.
The last metric that could not be used as it is, is the 'pair_confusion_matrix'. While there are operations that turn a metric into one number, there did not seem to be a feasible way to scale the matrix to one number. This is also because some very important information given by a matrix, is determined by ratio's and positions of numbers. This information would no longer be accessible after scaling the matrix to one number. For these reasons, the 'pair_confusion_matrix' was also not scaled.

This means that in the end none of the metrics, that were not usable in their original form, were used for further analysis. Consequentially, they were dropped from the data frame.

3.2.4 Correlation analysis

Before finding the most interesting trade-offs using the Pareto front, some preliminary analysis is done. This preliminary analysis basically meant checking the correlations between all metrics. To do this, a correlation matrix is made. This was done using a standard Pandas function. This matrix was turned into a heatmap, which can be seen in figure 3.3.

The correlation metric can be turned into a table with three columns: one for the first metric, one for the second metric and one containing the correlation between these metrics. This table can then be ordered so that the smallest correlations are at the top of the table. This helps when selecting which metrics to plot the Pareto front for. A function is used to remove duplicates from this table. As mentioned before, highly correlated metrics will likely not have a very interesting trade-off. Therefore, knowing correlations helps determine which combinations of metrics are actually useful to see plotted as a Pareto front.

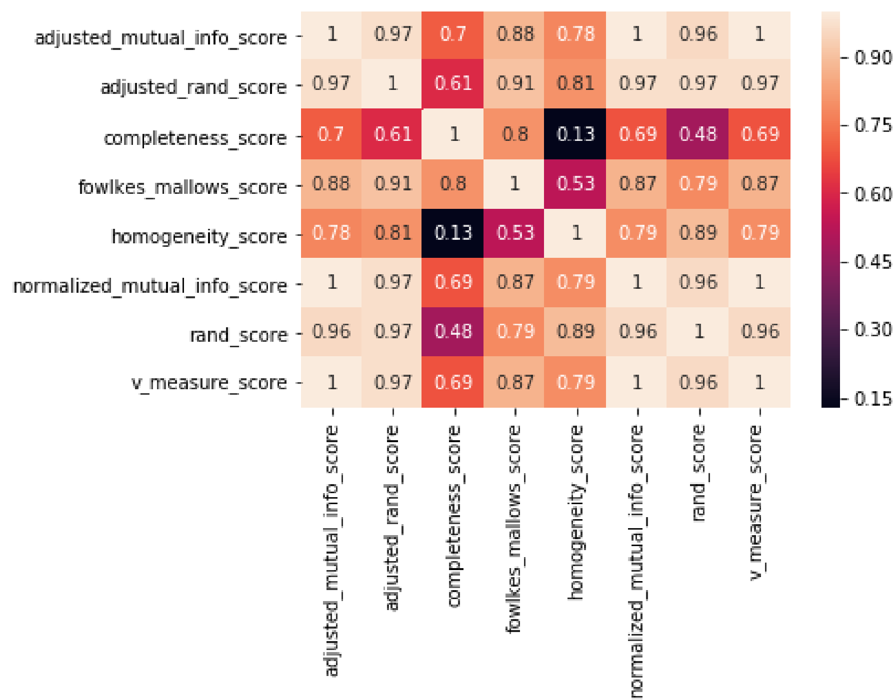


Figure 3.3: Correlation matrix as heatmap

3.2.5 Analysis with the Pareto front

The metrics can be used to represent an x and y coordinate, thus allowing clustering runs to be plotted as points. One metric is plotted on the x-axis and another on the y-axis. These points can be visualized using a scatter plot, which will in turn also make visualization of the Pareto optimal front possible.

For the construction of the Pareto optimal front, Pareto efficient points(clustering models) need to be found. Finding the Pareto efficient points is done as follows. An algorithm iterates over all of the points in the set. Each iteration the algorithm checks whether there are points with both a higher x- and y-coordinate, than the point that is being iterated over. If there are no points more efficient, the iterated over point is added to the list of Pareto efficient points. Next, anchor points need to be established. These anchor points are found by taking the best score for one objective function as one of the coordinates and zero as the other (see definition in section 2.3). Together, the anchor points and the

Pareto efficient points for the Pareto optimal front. Visualizations of these fronts can be seen in figure 4.1

After construction, Pareto fronts can be compared using their hypervolumes, the area under the curve. To do this, all points in the front are exported along with the (0, 0) point. All of these points together are inputted into the 'geometry.Polygon()' function from the shapely package. The purpose of the shapely package is "set-theoretic analysis and manipulation of planar features" [3]. In this case, the 'geometry.Polygon()' function turns the Pareto front into a 2d shape. Most importantly, it also calculates the area of that shape, the Pareto hypervolume, and saves it as an element of the shape. This area can then be outputted very easily as result or for further analysis.

Another way of analysing the Pareto front is by looking at the amount of points that lie on the front. This could give an insight into how many points are considered to be efficient and if a lot of points are efficient the trade-off will very likely be more interesting. To do this the length of the vectors containing the coordinates are obtained using the function that was also used for creating the Pareto front. All code used during construction of the Pareto front can be found in section A.1 in the appendix.

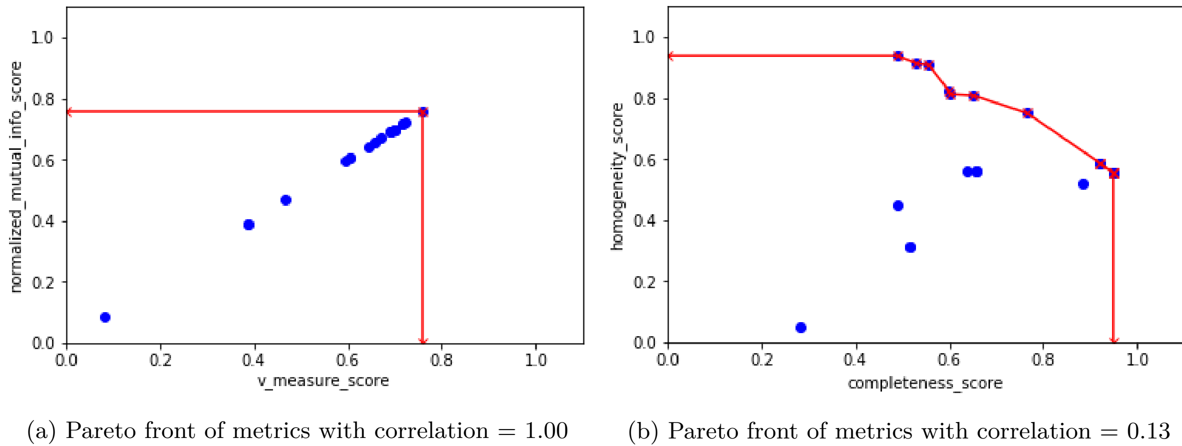
4 Results

4.1 Evaluation engine

The results of the creation of the package are quite straightforward. The package was created and could in theory be used by the OpenML website. Not all are (fully) functional, however more metrics can be integrated fairy easily as the evaluation engine was created to be scalable.

4.2 Clustering metric selection

Two of the constructed Pareto fronts are shown in Figure 4.1. A clear difference can be seen between the front of two metrics with a very high correlation and the front of the metrics with a low correlation. The front of two metrics with a high correlation shows no trade-off between the two metrics, as there is one point performing best with regard to both metrics. The Pareto front of the two metrics with the least correlation shows a trade-off that is a lot more diverse. It shows that plotting the Pareto front of barely correlated metrics gives a lot more insight into the different performances of models. Plotting the Pareto front of highly correlated metrics on the other hand, only showcases which models are 'better' than others.



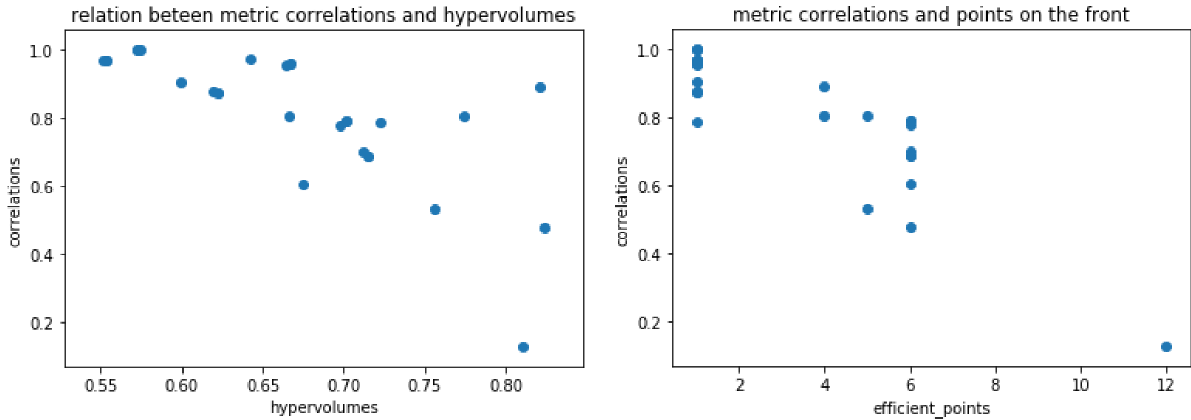
(a) Pareto front of metrics with correlation = 1.00

(b) Pareto front of metrics with correlation = 0.13

Figure 4.1: Pareto fronts for metrics

There is also a difference in the hypervolumes. The low correlation front has an area of 0.81 while the high correlation front has an area of 0.66 (no measurement unit provided because area is calculated through graph coordinates). This does however not mean that highly correlated metrics automatically have a small hyper volume. For example, when a model performs very well for one metric and that metric is highly correlated with another, the area of their Pareto front could be very large. The relation between the correlations of metrics and the area of their Pareto front is show in a scatter plot in figure 4.2a. The correlation between the correlations of metrics and the hypervolumes of the same metrics is -0.72. This does indicate that metrics with a high correlation will generally have a Pareto front with a smaller hypervolume. All of the hypervolumes and correlations per metric combination can be seen in Table A.2 in the Appendix.

Another way the Pareto front was analysed, was through the amount of points on the Pareto front. The relation of the metric correlations and the amount of points on the Pareto front can be seen in figure 4.2b. The correlation in this case is -0.88, which means it is an even stronger negative correlation than that of the hypervolumes. Having more points on the Pareto front also means that the front is likely to be more curved, as the efficient points can not all be in the same spot. This means finding the amount of points on the front is likely a better way of evaluating a trade-off than calculating the Pareto hypervolume.



(a) Relation between metric correlations and hypervolumes

(b) Relation between metric correlations and the amount of points on the pareto front

Figure 4.2: Pareto fronts for metrics

5 Discussion

The clustering metric selection analysis has created some useful insights. Firstly, the Pareto hypervolumes are quite highly correlated with the correlations between metrics. According to this analysis a higher hypervolume will very likely indicate a more interesting trade-off between metrics. However, using Pareto hypervolumes to find out which metrics have an interesting trade-off is not a fail safe method. This is because metrics without an interesting trade-off can still generate a large Pareto hypervolume. Moreover, using Pareto hypervolumes is less efficient than just using correlations between metrics, while it does not seem like a better way of finding interesting trade-offs between metrics.

Looking at the amount of Pareto efficient points, seems like a better way of finding interesting trade-offs as it is less susceptible to highly correlated metrics causing outliers. Also, counting the amount of Points on the Pareto efficient front actually gives some insight into what the front looks like. This would make it a better option for metric selection than using hypervolumes, though it does not look that much more promising than just using the correlation.

In the end, constructing and plotting the Pareto front seemed most useful for the OpenML user, because they can look at it and see how different models perform. This will allow them to work on a tasks with some useful knowledge about how their model and models of others perform.

5.1 Limitations

During both the analysis of clustering metrics and the creation of the package, there were some issues with the OpenML API. Most of them were resolved swiftly, however some them were not. An issue with clustering tasks meant all models had to be newly created and were not in a real 'run' format. Consequentially, the used data set was fairly small and contained the results of only twenty models. API issues also remove the possibility to create some integration of the Pareto front for the OpenML website.

Some limitations of the OpenML API also made it so that not all metrics could be implemented. Furthermore, some metrics could not be compared to correct results, because they had no existing OpenML implementation.

6 Conclusion

To bring this final bachelor project report to an end, the evaluation engine and research questions will be discussed. The evaluation engine does not warrant much of a conclusion, since no 'real research' was

conducted. However, the development of the evaluation engine did bring many challenges with it. In the end almost all of these challenges were overcome and a satisfactory evaluation engine was created.

This research provided the following answers to the research question and sub-questions stated in the introduction. Firstly, The Pareto front can be used for the selection of clustering metrics by calculating and comparing Pareto hypervolume or by counting and comparing the number of Pareto efficient points. However, these methods do not seem to be optimal.
Secondly, Plotting the Pareto front seemed to be a very promising way to evaluate clustering models, as it provides the OpenML user with useful insights into model performance.
Finally, answering the main research question, the Pareto front can best be used by OpenML to give insight into the performance of clustering models. This can be done through construction and visualization of the Pareto front.

6.1 Recommendations

Selecting combinations of clustering metrics can best be done using correlations between metrics. It could be very useful to integrate some function into the OpenML website, that shows the correlation of all metrics that have been used to evaluate runs for a specific task. Such a function could be very helpful when selecting the metrics with which to optimize the model.
Building on this function that shows correlations, a function that constructs the Pareto front for selected metrics can really help give insight into any machine learning task. It will show what models work well and what limitations and trade-offs to keep in mind when starting or working on a task.

References

[1] Georg Brandl. "Sphinx documentation". In: *URL http://sphinx-doc. org/sphinx. pdf* (2021).

[2] Brian Eastwood. *The 10 Most Popular Programming Languages to Learn in 2022.* 2020. URL: https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/.

[3] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects.* toblerity.org, 2007–. URL: https://github.com/Toblerity/Shapely.

[4] Nyoman Gunantara. "A review of multi-objective optimization: Methods and its applications". In: *Cogent Engineering* 5.1 (2018), pp. 1–16. ISSN: 23311916. DOI: 10.1080/23311916.2018.1502242. URL: https://doi.org/10.1080/23311916.2018.1502242.

[5] Paul Lavrakas. *Encyclopedia of Survey Research Methods.* Thousand Oaks, California, 2008. DOI: 10.4135/9781412963947NV-0. URL: https://methods.sagepub.com/reference/encyclopedia-of-survey-research-methods.

[6] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge, UK: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5. URL: http://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[7] Neeratyoy Mallik. *OpenML: Machine Learning as a community.* 2019. URL: https://towardsdatascience.com/openml-machine-learning-as-a-community-d678306e1a7e.

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[10] Audrey Roy. "cookiecutter Documentation". In: (2020).

[11] *TIOBE Index for March 2022.* 2022. URL: tiobe.com/tiobe-index/.

A Appendix

A.1 Personal contribution

Throughout construction of the evaluation engine I have had a variety of responsibilities. I was involved in implementation of metrics and providing those metrics with documentation. Furthermore, I contributed to the testing with Pytest and creation of the placeholder object class. Lastly, I spent wuite some time doing general debugging.

A.2 tables

K-Means	n_clusters = 2
	n_clusters = 3
	n_clusters = 4
	n_clusters = 5
DBSCAN	eps=0.25, metric='euclidean'
	eps=0.25, metric='manhattan'
	eps=0.25, metric='l2'
	eps=0.50, metric='euclidean'
	eps=0.50, metric='manhattan'
	eps=0.50, metric='l2'
	eps=0.75, metric='euclidean'
	eps=0.75, metric='manhattan'
	eps=0.75, metric='l2'
MeanShift	no parameters specifies
	bandwidth = 0.5
	bandwidth = 1.0
	bandwidth = 1.5
AffinityPropagation	damping = 0.5
	damping = 0.75
	damping = 0.95

Table A.1: All models used for the analysis

	metrics	hypervolumes	correlations
0	[completeness_score, homogeneity_score]	0.809887	0.128571
1	[completeness_score, rand_score]	0.823846	0.479588
2	[fowlkes_mallows_score, homogeneity_score]	0.756245	0.531706
3	[completeness_score, adjusted_rand_score]	0.675218	0.605797
4	[completeness_score, v_measure_score]	0.714932	0.686452
5	[completeness_score, normalized_mutual_info_sc...	0.714932	0.686452
6	[completeness_score, adjusted_mutual_info_score]	0.711873	0.700762
7	[adjusted_mutual_info_score, homogeneity_score]	0.698076	0.778907
8	[fowlkes_mallows_score, rand_score]	0.722091	0.787485
9	[homogeneity_score, normalized_mutual_info_score]	0.701876	0.793728
10	[homogeneity_score, v_measure_score]	0.701876	0.793728
11	[fowlkes_mallows_score, completeness_score]	0.773755	0.804510
12	[adjusted_rand_score, homogeneity_score]	0.666560	0.805580
13	[fowlkes_mallows_score, v_measure_score]	0.622317	0.872151
14	[fowlkes_mallows_score, normalized_mutual_info...	0.622317	0.872151
15	[fowlkes_mallows_score, adjusted_mutual_info_s...	0.619808	0.879706
16	[homogeneity_score, rand_score]	0.821187	0.892425
17	[fowlkes_mallows_score, adjusted_rand_score]	0.599385	0.907797
18	[adjusted_mutual_info_score, rand_score]	0.664302	0.956298
19	[v_measure_score, rand_score]	0.666991	0.961383
20	[rand_score, normalized_mutual_info_score]	0.666991	0.961383
21	[adjusted_rand_score, adjusted_mutual_info_score]	0.551417	0.969519
22	[adjusted_rand_score, normalized_mutual_info_s...	0.553649	0.969920
23	[adjusted_rand_score, v_measure_score]	0.553649	0.969920
24	[adjusted_rand_score, rand_score]	0.642414	0.971860
25	[adjusted_mutual_info_score, normalized_mutual...	0.572513	0.999475
26	[adjusted_mutual_info_score, v_measure_score]	0.572513	0.999475
27	[v_measure_score, normalized_mutual_info_score]	0.574830	1.000000

Table A.2: All hypervolumes and correlations

A.3 Pareto front code

```
import numpy as np
import pandas as pd
from shapely.geometry import Polygon

class Pareto():
    """a class containing all functions for the construction of the pareto
       front

    Input: A list of x coordinates and a list of y coordinates"""

    def __init__(self, X, y):
        self.X = X
        self.y = y

    def pareto_efficient(self):
        """A function to find all af the points in the set of  points that
           are pareto efficient."""

        efficient_X = []
        efficient_y = []

        for i in range(len(self.X)):
```

```
            more_efficient = [t for t in range(len(self.X)) if (self.X[t] >
                self.X[i] and self.y[t] > self.y[i])]

            if len(more_efficient) == 0:
                efficient_X.append(self.X[i])
                efficient_y.append(self.y[i])

        #sorting by x coordinates for later convenience
        xs_eff, ys_eff = zip(*sorted(zip(efficient_X, efficient_y)))
        xs_eff, ys_eff = list(xs_eff), list(ys_eff)

        return xs_eff, ys_eff

    def anchor_points(self):
        """A function to find the anchor points of the current set for
            construction of the pareto front."""

        anchor_X = [0, max(self.X)]
        anchor_y = [max(self.y), 0]

        return anchor_X, anchor_y

    def frontier(self):
        """A function that uses the anchor points and Pareto efficient
            points to construct the
        entire set of points that forms the Pareto optimal frontier. """

        frontier_X, frontier_y = self.pareto_efficient()
        anchor_X, anchor_y = self.anchor_points()

        frontier_X.insert(0, anchor_X[0])
        frontier_X.append(anchor_X[1])

        frontier_y.insert(0, anchor_y[0])
        frontier_y.append(anchor_y[1])

        return frontier_X, frontier_y

    def hyper_volume(self):
        """A function that uses the frontier to find the Pareto front hyper
            volume (the area behind the frontier)"""
        frontier_X, frontier_y = self.frontier()

        #frontier_X.append(min(self.X))
        #frontier_y.append(min(self.y))
        frontier_X.append(0)
        frontier_y.append(0)


        Frontier = zip(frontier_X, frontier_y)

        return Polygon(Frontier).area
```