Bachelor Degree Project

MetaStackVis: Visually-Assisted Performance Evaluation of Metamodels in Stacking Ensemble Learning

Author: Ilya Ploshchik
Supervisors: Prof. Dr. Andreas Kerren,
Angelos Chatzimparmpas
Semester: Autumn 2022
Subject: Computer Science

Abstract

Stacking, also known as stacked generalization, is a method of ensemble learning where multiple base models are trained on the same dataset, and their predictions are used as input for one or more metamodels in an extra layer. This technique can lead to improved performance compared to single layer ensembles, but often requires a time-consuming trial-and-error process. Therefore, the previously developed Visual Analytics system, StackGenVis, was designed to help users select the set of the most effective and diverse models and measure their predictive performance. However, StackGenVis was developed with only one metamodel: Logistic Regression. The focus of this Bachelor's thesis is to examine how alternative metamodels affect the performance of stacked ensembles through the use of a visualization tool called MetaStackVis. Our interactive tool facilitates visual examination of individual metamodels and metamodels' pairs based on their predictive probabilities (or confidence), various supported validation metrics, and their accuracy in predicting specific problematic data instances. The efficiency and effectiveness of MetaStackVis are demonstrated with an example based on a real healthcare dataset. The tool has also been evaluated through semi-structured interview sessions with Machine Learning and Visual Analytics experts. In addition to this thesis, we have written a short research paper explaining the design and implementation of MetaStackVis. However, this thesis provides further insights into the topic explored in the paper by offering additional findings and in-depth analysis. Thus, it can be considered a supplementary source of information for readers who are interested in diving deeper into the subject.

Keywords: Visualization, interaction, metamodels, validation metrics, predicted probabilities, stacking, stacked generalization, ensemble learning, ML, AI.

Preface

Work on this Bachelor's thesis was an interesting scientific experience that required studying the vast field of Visual Analytics in combination with a better understanding of the principles, technologies, and Machine Learning methods. First, I would like to thank my supervisor, Angelos Chatzimparmpas, for his careful guidance and assistance throughout all project stages. I also want to thank my other supervisor, Prof. Dr. Andreas Kerren, for his support at the final stage of this project. Furthermore, many thanks to Dr. Gabriel Eilertsen, Dr. Jonas Nordqvist, Dr. Kostiantyn Kucher, and Dr. Rafael M. Martins for their valuable comments and input during the review of the visualization tool. Last but not least, I would like to thank my family for their continued support throughout this time.

Contents

1 Introduction

Stacking (or stacked generalization), as stated by Rocca [1], is an ensemble learning method that uses predictions or predictions probabilities from multiple base models (or so-called "weak learners") trained on the original dataset as input for training higher-level models, known as a metamodels. The ultimate goal of this method is to try to improve the predictive performance of supervised ML scenarios as much as possible. Methods of stacked generalization, as outlined by Wolpert [2], have been shown to lower bias and improve generalization error as compared to using individual learning algorithms. The ISOVIS research group at Linnaeus University has developed a Visual Analytics (VA) system called StackGenVis [3], that provides the additional visual functionality to end users in optimizing validation and performance metrics, data manipulation, including filtering the data features to be considered, as well as evaluating predictive performance of implemented algorithms. The current version of StackGenVis was created with Logistic Regression (LR) at the metamodel layer [4].

This Bachelor's thesis describes MetaStackVis, an *interactive and open-source visualization tool* [4] for visual impact investigation of various metamodels on stacking ensemble based on data provided by StackGenVis. We examined alternative metamodels using multiple performance validation metrics and predictive probabilities (or *confidence*). We also compared the overall predictive power of metamodels' pairs and reviewed the performance of metamodels' combinations on consistently mislabeled data instances using different voting techniques.

1.1 Background

As stated by Rocca [1], model ensemble is a ML paradigm where simple base models, also known as "weak learners", are trained to solve the same problem and then combined in a specific way to obtain better results. The main "wisdom of the crowd" idea, as pointed by Skyler [5], is that these independently trained weak models, when properly combined, can lead to more predictive and/or more stable results. Gada et al. [6] identifies following ensemble methods, based on a way of defining the base algorithms (homogeneous or heterogeneous) and an approach to ensemble: bagging, boosting, and stacking. Bagging, also known as *bootstrap aggregation*, involves training a number of similar models on different subsets of the data, and then using some kind of averaging or voting process to combine their predictions [1]. The goal is to reduce the variance of the overall model by training each base model on a different sample of the data, rather than training all of the models on the same data. This can help to improve the stability and generalization of the model. Boosting also considers homogeneous base models, but is based on a sequential process of model improvement. Gada et al. [6] defines boosting as a method then each previous model is improved by a successive model. Stacking [1], unlike boosting and bagging, usually operates with heterogeneous base models and deploys a metamodel to sum the prediction results of heterogeneous base models organized at one or more levels, as mentioned by Chatzimparmpas et al. [7]. Stacking is commonly composed of two layers: "layer 0" known as *base layer* (see Figure 1.1(c)) and "layer 1" so-called *metamodel layer* (cf. Figure 1.1(d)). However, more layers can be used if required [1]. The base layer models are trained on a training set using cross-validation techniques (that is typically k-fold cross validation [8]).

It is important to mention that prior to training, the original dataset has to be split into a specific ratio of training and testing data, as illustrated in Figure 1.1(a) and (b). As previously mentioned, the models in the base layer do not need to be homogeneous; therefore,

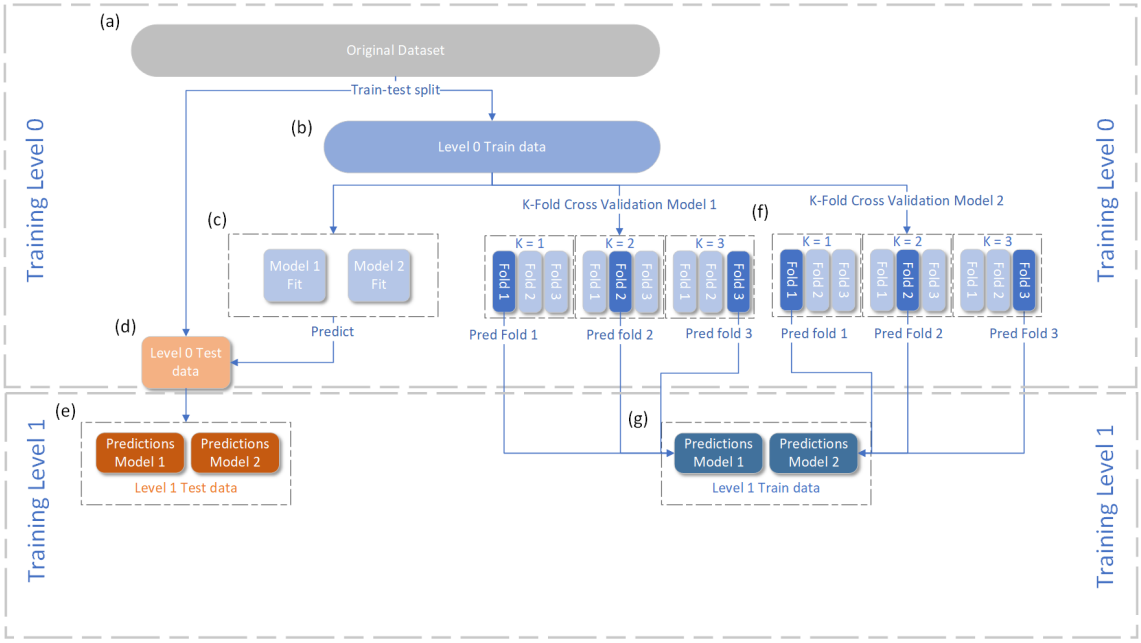

Figure 1.1: Stacked generalization with two layers. Stacked generalization is a ML ensemble technique that combines the predictions of multiple models to create a more accurate and stable prediction. In a stacked generalization with two layers, the first layer consists of a set of base models that are trained on the input data (a) split into train data shown in (b) and test data shown in (d). The base models are trained on a training set using k-fold cross validation [8], visible in (f). In (g), each fold predictions are then stacked and used as training data for the second layer (i.e., the metamodel layer). Base models are further trained on the complete dataset, as shown in (d), and used to predict the original test data to generate level 1 test data, as shown in (e).

one can choose algorithms with different strengths based on the structure and complexity of the given problem. The predictions of these models are then combined (or *stacked*) and used as training data for the second layer (i.e., a metamodel layer). Base model predictions made on the test dataset are stacked to form a test dataset for the metamodel layer. Ting and Witten [9] claim that metamodel training usually consists of a single model, such as Decision Tree (DT), Naive Bayesian (NB) Classifier, or Linear Regression.

Blending can be considered an alternative version of stacking with the main difference being the way it splits up the original dataset, as presented in Figure 1.2. Instead of relying on k-fold cross validation for the training of base models for proper data distribution in regular stacked generalization, the blending method uses a separate hold-out validation set to ensure the base layer model predictions are generalizable (see Figure 1.2(f)). As metamodel testing is performed on unseen data blending method can lead to reduced overfitting of base layer models. Thus, the original dataset is typically split into three datasets with a certain proportion [7]:

- training dataset for training the base layer models, see Figure 1.2(b);
- holdout validation set to generate predictions of base layer models, see Figure 1.2(f)—these predictions form the training dataset for the metamodel layer; and
- test dataset to generate predictions of base layer models, see Figure 1.2(d)—these predictions form the test data for the metamodel layer.

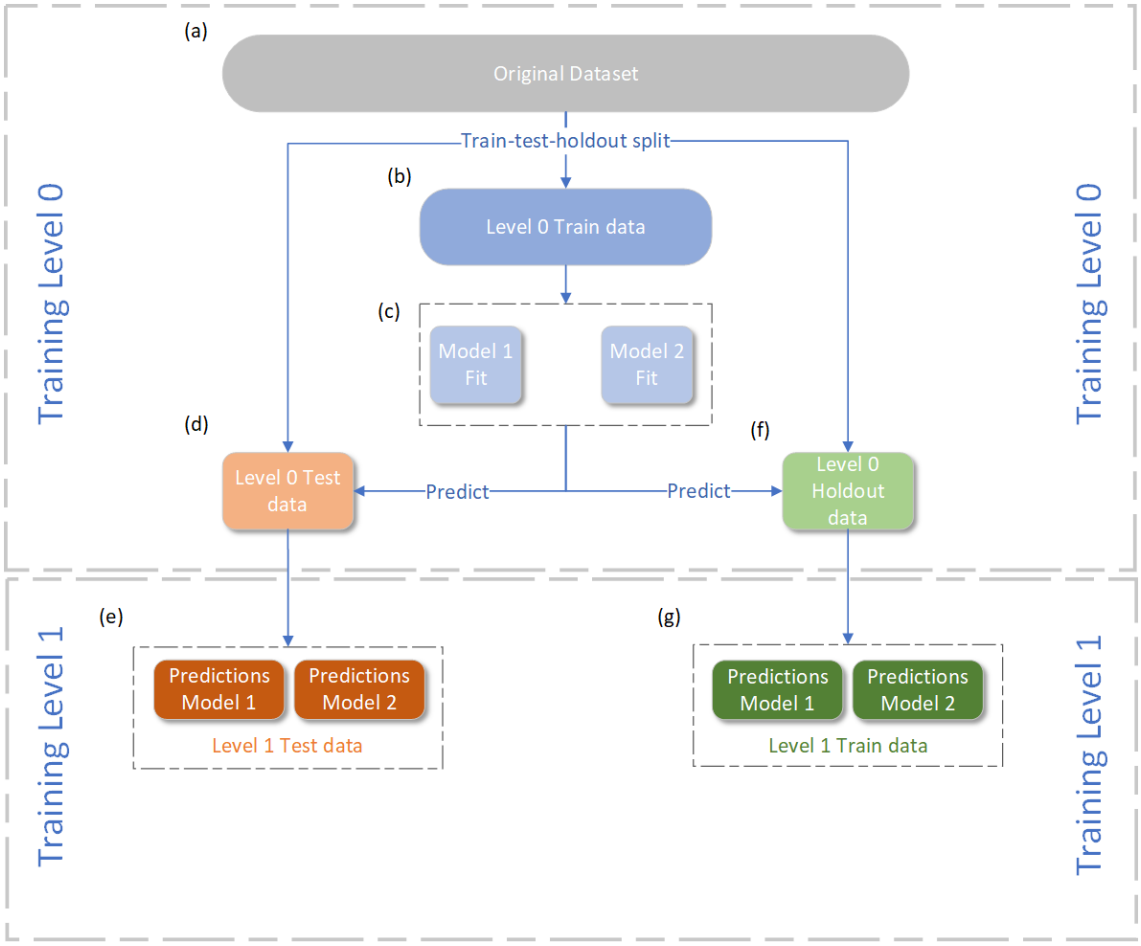

Figure 1.2: Blending with two layers. Blending technique requires the original data (a) to be split into three datasets: train data shown in (b), holdout validation data shown in (f), and test data shown in (d). Base models are first trained using train data, as shown in (c). As shown in (g), the base model predictions for the holdout validation data are used as input for the metamodel layer training data, whereas the predictions for the test data are used to evaluate the metamodel's predictive performance, see (e).

Stacking may effectively increase overall model predictive performance, and with the properly chosen algorithms both on the base layer and on the metamodel layer one can ensure getting the top-performing models. In this context, the models depict the output structure of the algorithm once it has been adjusted to the data with specific hyperparameters [4]. However, as mentioned by Chatzimparmpas et al. [3] the process of defining proper hyperparameters for the specific algorithm requires a complicated "trial-and-error" process. This process of hyperparameter optimization, using either exhaustive search algorithms (Grid Search or Random Search) or sequential model-based optimization (e.g., Bayesian Optimization), is further described in Section 2.2. This "trial-and-error" process opens additional questions, like how one can avoid these manual trial runs, monitor the overall training process, and at the same time define the best metrics to be used for performance and model validation [3].

Chatzimparmpas et al. [7] identifies various approaches, such as AutoML, which have been developed for data gathering and performance analysis, as well as hyperparameter tuning. However, these techniques are usually considered as "black boxes" without end users' interaction or visual process representation. According to Keim et al. [10], these

black boxes cannot be opened without additional data and process insights. Therefore, the emerging field of VA tries to address the same questions by taking care of the automatic evaluation of the training process through repetitive interaction and optimization of hyperparameters of the ML models. Chatzimparmpas et al. [7] point that beyond providing the automatic way of hyperparameter tuning, VA has also one additional preeminence above other automated ML tools as it provides the additional feedback to the end user by visual process representation and various interaction possibilities. That allows users with little or no coding experience to evaluate different ML algorithms and techniques. According to Sacha et al. [11], VA is the tool that supports domain experts and business users in the data exploration of large and complex datasets and provides knowledge generation by enabling tight interaction between the end user and the system. That helps end users identify patterns in data, verify the hypotheses, and make decisions based on the provided data.

That whole process of transforming data into valuable insights through the visual representation of initial data, the data wrangling process, and the end results is what separates VA from another, broader known field of Information Visualization (InfoVis). While VA is trying to address the specific analytical and data-driven questions through the combination of computation tasks and visual representation of results, InfoVis is focusing on more generic data exploration without any specific agreed-upon view as mentioned by Card [12], see Section 2.4 for additional details regarding VA systems.

In terms of ML and data analytics, VA tools can cover the whole aspect of data extraction, modification, training, and results visualization in an organized and well-structured manner. As an example, the VA system, presented in this study, gives the end user a functionality to interact with clusters of base models based on cluster performance and provide the valuable insight on single metamodel and pairwise combinations of metamodels in regard to predictive performance and probability power (or level of confidence); see Section 4 for additional details.

1.2 Problem formulation

Despite providing very powerful functionalities to the end user regarding data and modeling exploration, VA tools still face some challenges compared to traditional ML alternatives.

- Both Chatzimparmpas et al. [3] and Keim et al. [10] state that data scalability is required to support high-resolution data and continuous data. In the modern world of big data and analytics projects model fitting and hyperparameter tuning can already require extensive computing power and networking infrastructure. By adding the additional element of visual representation of data modeling, one can expect a much higher computing expense. One additional point is related to continuous data flow, as VA tools usually work with historical data (i.e., data that has already been captured and stored). In cases of continuous data flow (like machine or transport information) one can expect sufficient latency in data exploration, which can effect the reasoning behind real-time data analysis.

- Chatzimparmpas et al. [3] also mention data dimensionality to support multi-layer stacking models as one of the current VA tools' challenges. Due to the variety and complexity of modern data the performance of multi-layer stacking models can be drastically reduced as models are generally developed to handle specific data types. Certain dimensionality reduction algorithms can be applied to the data preparation step prior to training of base layer algorithms. These algorithms can include

future selection methods, such as matrix factorization with Principle Component Analysis (PCA) [13] and manifold learning methods with Multidimensional Scaling (MDS) [14] or t-distributed Stochastic Neighbor Embedding (t-SNE) [15].

- Keim et al. [10] commented on the important challenge related to the quality of data to avoid any misinterpretations by the analyst. Chatzimparmpas et al. [3] brought up the point that despite the fact that it may seem like the main advantage of VA tools and systems, the interpretability and explainability of them can still pose as a non-trivial problem. For example, choosing the appropriate visual representation for the stacking ensemble method, while including several performance indicators and combinations of various "weak learners" is a daunting task. It is important to find a balance between model performance and interpretability, as increasing a ML model's overall predictive performance can negatively impact the ability to interpret its results. The visualization tools shall be designed in a way to exclude or at least reduce the possible misinterpretation of results. That can be achieved by providing special training to respective small and medium-sized enterprises and data analysts, as well as developing visualization tools with specific usage scenarios in mind.

- Keim et al. [10] also mentioned infrastructure limitations to achieve smooth interaction with large amounts of data. This point is related to data scalability mentioned before. With an increased amount of data, visualization of the results can be accomplished at increased computing expense, which is not always technologically and rationally achievable. Therefore, when developing a VA tool/system, one needs to find a balance between functionality and performance based on the given problem.

Data scalability and data dimensionality aspects in this research project will be addressed by using the HDBSCAN technique to separate clusters of base layer models. In such a way, data analysts can easily compare the performance of metamodels on a specific cluster of base layer models. That can be done on a subset of the original data before transitioning to the full dataset. By using only a subset of base layer models, one can drastically reduce the computational time needed for fitting data while maintaining the necessary level of performance - being equal to or greater to the performance level of an ensemble trained on all base layer models' predictions. Campello et al. [16] identify the HDBSCAN algorithm as an improved version of the well-known DBSCAN density-based clustering algorithm. It has some clear advantages compared to its predecessor since it focuses more on high density clustering compared to DBSCAN. For example, DBSCAN is quite sensitive to false clusters due to the need for setting a global density threshold. Campello et al. [16] further claim that HDBSCAN allows hierarchical clustering and helps to reduce the number of clusters with low density (i.e., noisy clusters). Being a density-based algorithm, HDBSCAN tries to estimate the densities of the given data points (which are base model probabilities in our case), define clusters with high densities, and combine the models in these clusters. Our tool will also identify all left-over points (in our case, ML models), known as outliers, as a separate cluster.

We will utilize the Uniform Manifold Approximation and Projection (UMAP) technique, introduced by McInnes et al. [17] for the visual representation of model probabilities from all instances into the two-dimensional workspace, which is then used by MetaStackVis to generate visualizations of both metamodels and base models on the same plot. UMAP is a dimensionality reduction technique for visualizing high-dimensional datasets in a lower-dimensional space. It is a nonlinear method that is based on the idea of preserving the local structure of the data, which means that similar points in the original space

should be mapped to nearby points in the reduced space. UMAP works by first constructing a graph representation of the data, where the nodes of the graph are the data points, and the edges are determined by the similarity between the points. This graph representation is used in order to learn and possibly visualize a low-dimensional embedding of the data.

Due to complexity of stacking ensembles, Chatzimparmpas et al. [3] developed StackGenVis using solely LR algorithm on a metamodel level. The selection of a single classifier on the metamodel level (e.g., LR for classification problems) is a common practice for developing stacking ensembles. Although simple, this approach is a powerful way of capturing input data based on prediction results from the base layer models. However, one can argue that using a specific model will not always suit the given data or provide the most optimal solution based on performance and confidence level for a specific problem. In their work, Chatzimparmpas et al. [3] considered the evaluation of alternative metamodels' impact on the predictive performance as a further improvement of the StackGenVis system. This research opportunity is the main topic of this thesis. In addition to comparing various models' performance and prediction probability confidence on a metamodel level we will also perform a pairwise comparison of metamodels. That can be considered as a simulation of having two metamodels at the metamodel level to see where a particular pair of metamodels can provide better results than a single model.

1.3 Motivation

Several research papers have considered the use of a specific metamodel for stacked generalization. Ting and Witten [9] have proposed the DT, NB Classifier, IB1 (a variant of a lazy learning algorithm), and the Multiple Linear Regression (MLR) algorithm as metamodels for stacked generalization. In their research study, Ting and Witten [9] came to the conclusion that the MLR algorithm outperforms other alternatives in most cases. Ting and Witten [9] also proposed using base models' probabilities rather than base layer models' predictions for metamodel layer as input data. Indeed, soft voting majority voting strategy that considers predicted probabilities is usually a better approach as it provides additional information for the metamodel, especially when compared to a regular hard voting schema for metamodel training only on the predictions. Nevertheless, the idea behind grouping these probabilities to identify clusters of base-models with further overall performance improvement of the two-layer stacking ensemble yet remains uninvestigated.

Seewald [18] has further developed the MLR algorithm for metamodel layer modeling and developed the "StackingC" algorithm, which is especially efficient for multi-class problem definition. The algorithm is utilizing dimensionality reduction techniques to ensure higher insensitivity to the number of classes in the dataset. These techniques, however, expected the whole set of base layer models to be employed in reality. A different approach has been proposed by Chen and Wong [19] with their Ant Colony Optimization technique. This approach was based on first identifying the most suitable subset of base layer models and then a metamodel for a given problem using a meta-heuristic method. A similar approach has been developed by Shunmugapriya and Kanman [20] for their Artificial Bee Colony algorithm. Both previous works operated with the same type of base layer and metamodel layer algorithms: LR, KStar, OneR, PART, ZeroR, C4.5 DT, NB Classifier, Decision Stump, and K-Nearest Neighbors (KNN). This degree project will re-introduce some of the algorithms mentioned on a metamodel level, including NB, LR, and KNN classifiers, among others.

To sum up, the need for advanced visualization approaches for stacking ensemble that can provide data analysts with the vital toolset to identify the subsets of base-models with the highest performance, is potentially of great interest to the ML and visualization communities. Further development requirements are also identified in regard to the comparison of the metamodels belonging to a second layer. In detail, we facilitate the pair-wise comparison of metamodels in terms of metric-based performance and confidence/predicted probabilities.

1.4 Objectives

We have identified several challenges while reviewing the available VA tools and systems for the visualization of ML algorithms in stacked generalization. The answers to these challenges are being considered as research objectives for this Bachelor's thesis. The detailed explanation of why a research question is important can be found in Table 1.1.

Table 1.1: Open challenges for the VA tools and systems, working with multiple base models and alternative metamodels in stacking ensemble.

Research Question	Reasoning
What is the impact on the overall predictive performance when judging which alternative metamodel is suitable for a stacking ensemble?	As stated in Section 1.2, this work is considered as further contribution to the previously developed StackGenVis system by the ISOVIS group at LNU [3], in which LR was the only metamodel they experimented with. In this project, we will try to investigate the performance of 11 different algorithms on the overall performance of stacked generalization.
What is the difference in the predictive performance of a single metamodel compared to a pair of metamodels?	To the best of our knowledge, this question has not been addressed by other research papers. For pairwise model combinations, we expect that the pair of two metamodels will perform at least equally well as the best performing model in this pair. However, this hypothesis needs to be confirmed.
How heavily are metamodels' predictions getting affected by the training on a specific subset of base layer models compared to learning from all base layer models?	We expect that training a metamodel by using predictions from all base models will provide the highest possible metric-based performance and confidence. Despite that, this hypothesis requires verification, especially since some base models may underperform in certain cases due to the data structure and data complexity.

1.5 Scope/Limitation

This project will address the challenges described in Table 1.1 and support the further development of the StackGenVis system [3] (developed by the ISOVIS group at LNU). The main focus will be relevant to the development of a visualization tool with a web interface to support the visual representation of the metamodel layer's results. We define the scope of this project by dividing the research objectives into several aspects:

1. The input data will be based on generated data from the current version of the StackGenVis implementation. The data will be split into training and testing sets. In addition to that, StackGenVis provides base layer models' predicted probabilities for the test data, as well as values for eight performance metrics. In total, data from

55 top performing base models will be provided, meaning that we will explore the top five base models in terms of performance per each of the 11 ML algorithms.

2. In total, 11 ML models are to be trained on a metamodel level with a single metamodel per ML algorithm.

3. Our newly-implemented visualization tool will be limited to a specific set of hyperparameters for each of the metamodels. Hyperparameters will be automatically chosen from the hyperparameter sets defined by the best-performing base layer model for each ML algorithm, leading to 11 concrete metamodels.

4. Base models will be grouped into separate clusters according to predicted probabilities, using HDBSCAN clustering algorithm. HDBSCAN strives to group similarly performing base models, starting with clusters of higher density.

5. Each metamodel will then be trained on each cluster of base layer models, including all base layer models and the ones known as "outliers" (meaning not being assigned by HDBSCAN in any of the clusters). Depending on the number of identified clusters, the 11 metamodels will increase in number accordingly.

6. We will then present the UMAP projection of each base layer model's cluster and respective trained metamodels to see if there are any patterns in combinations of base layer and metamodel layer models in every cluster.

7. A specific view will provide a thorough overview of metamodels' metric-based performance and probability confidence, as well as facilitate the comparison of pairs of metamodels.

8. As this tool is considered as a "proof-of-concept" for testing hypotheses for the research objectives (as described in Section 1.4), we decided to proceed with only two layers of stacked generalization, i.e., only one base layer and one metamodel layer. We plan to indicate to users if the combinations of metamodels could be beneficial; however, additional layers could summarize the predictions of those powerful metamodels. If this hypothesis is confirmed, the tool can be further improved at a later stage to a multi-layer approach.

9. Although we will base our analysis on a set of previously-used benchmark datasets (see Section 4), the approach is going to be designed in a way to be generalizable to any binary classification problem.

1.6 Target group

MetaStackVis is meant to primarily target data scientists and data analysts, who would prefer to evaluate model performance not only based on performance metrics, but also based on practical model understanding [21]. Even though the visualization tool provides the possibility to evaluate the ML models' predictive performance for end users with little to no coding experience, the overall understanding of basic ML principles, performance indicators, ML algorithms, and statistics concepts are the main prerequisites.

From another perspective, MetaStackVis can be considered as a supplement to knowledge building for a given problem by ML experts and researchers who are interested in building robust and powerful ML pipelines. The proposed tool can also provide an overview of ML models' predictive power using a subset of the original data. These

valuable insights can be further developed to deploy a more inclusive toolset for ML predictors.

1.7 Outline

The organization of this report's writing allows the reader to easily follow the process's involvement from problem formulation to solution and output presentation.

Chapter 2 is related to the followed methodology in this research work. In detail, it explains the process involved, from problem formulation to data collection and analysis, including a thorough review of currently available research, related to metamodels for stacked generalization, different strategies for hyperparameter optimization, validation metrics for performance evaluation, as well as reliability, and validity aspects of this investigation, and any ethical aspects related to data collection, usage of the research results, data protection, and the privacy of involved parties.

Chapter 3 explains the MetaStackVis tool's functionalities in depth. It also gives a description of the technologies involved for data collection and model analysis, as well as any other software necessary for data exploration and visual representation of the proposed solution.

Chapter 4 demonstrates the benefits of the MetaStackVis tool and its application on a specific scenario by simulating a realistic business problem for a data scientist using a publicly available dataset.

Chapter 5 provides the assessment of our visualization tool through interview sessions with experts in the areas of ML and VA. The evaluation focuses on giving a brief overview of the positive and negative aspects concerning the workflow, visualization, and interaction of the tool. Additionally, we received feedback and recommendations for potential improvements in the MetaStackVis tool, such as the possible process optimizations, the integration with existing datasets and tools, more user-friendly visual representations, and insights into the scalability of the system in larger datasets and more complex use cases.

Chapter 6 explores the implications of the MetaStackVis tool and the possible trends that could arise from its adoption, as well as further potential improvements in its functionality and new research opportunities for the future.

Finally, Chapter 7 summarizes the results and offers suggestions for further research and development of MetaStackVis in order to make it a powerful tool for data scientists to utilize in their day-to-day work.

2 Method

The problem foundation as described in the Section 1.4 requires a systematic approach to evaluate the current challenges and perform an extensive review of currently available publications, related to the designated topic with regards to alternative solutions and available toolsets. It was therefore decided to split the Method chapter into several sections.

Section 2.1 is related to a literature review of metamodels for stacked generalization. The section tries to address the terminology and history behind the development of stacked generalization methods.

Section 2.2 captures hyperparameter optimization as required for model training to identify the optimal set of hyperparameters, leading to highest performance and the lowest error. This section covers different techniques, used for hyperparameter optimization, including exhaustive search algorithms, sequential model-based optimization and population-based hyperparameter tuning.

Section 2.3 describes metrics used to quantitatively measure the performance of the model in an objective and comparable way.

Section 2.4 describes the reviewed VA tools and systems with comparison of the tools available in our system, noting their differences and similarities.

Section 2.5 captures the aspects related to a more standardized way for data collection and exploration. While developing our system, we try to reduce uncertainty and possible misinterpretation by the end user. While operating with several layers of models and performing additional clustering and model tuning, we also need to ensure that the tool yields the same results, provided the input conditions remain the same.

Section 2.6 captures any concerns related to any leak of personal information in the provided datasets and tools. We need to ensure that we decouple and generalize the information provided in the toolkit to ensure that we follow the General Data Protection Regulation (GDPR).

2.1 Metamodels for stacked generalization

According to Wolpert [2], stacked generalization, or "stacking" is meant to be an efficient way to combine outputs, generated by a certain number of base models or *level 0* space. These outputs are further used as input data by models on a metamodel or *level 1* level to predict the value of the target variable. Even though the process can be iterated multiple times (leading to multiple stacks of ML models) [2], one can argue that it is mostly common to operate with a single metamodel layer [3]. Sesmero et al. [22] proposes that standard algorithm for stacking includes following stages:

- split the original dataset into k-fold dataset subsets, using cross-validation or other techniques;
- select one fold for testing and k-1 folds for training;
- train base model algorithms on training dataset and make predictions on a testing set; and
- train a metamodel on the new dataset, where k is the hyperparameter defining the number of folds in which the dataset will be split into [23].

The larger the number k, the more data to be allocated for model training and less for testing purposes. Operating with a higher k-number of folds reduces the overall prediction

error, but at the same time increases the computational time as more data needs to be fed for training. Anguita et al. [23] propose that, as a rule of thumb, testing with a hyperparameter of 5, 10, or 20 folds can be considered a valuable approach, heavily depending on the dataset size. We decided to proceed with 5-fold cross validation for our purpose because it was considered sufficient with the current size of the datasets being tested.

As identified by Wolpert [2], combining different learning algorithms (heterogeneous ensembles) for base ("weak") models exploits single base model's bias and therefore lead to overall bias reduction. However, as mentioned by Sesmero [22], there is no consensus among the analytics community regarding the optimal method to choose when it comes to the choice of algorithms for base layer classifiers, their parameters, the number of base classifiers, the metamodel layer algorithms, and the hyperparameters' tuning of these algorithms. At the same time, they confirmed that Stacking shall be considered a legitimate approach, compared to bagging and boosting, when it comes to the ensemble method. Our tool tries to answer some of the challenges listed by Sesmero [22], and by using the VA approach, it supports the conclusions linked to the choice of models and their parameters to use to gain optimal performance of the competing ensemble. However, as proper tuning of hyperparameters for base layer models has been sufficiently implemented by Chatzimparmpas at al. [3] in the StackGenVis solution, less focus is given to hyperparameter tuning in our toolset.

Ting and Witten [9] have addressed another crucial point, related to stacked generalization. The research group discovered that class probabilities shall be used as base layer model output data rather than proper class predictions [9]. That approach has been followed in our application, as well as to provide a smoother variant of "soft voting" of base layer model prediction results compared to "hard voting" of final prediction results in a two-class problem. Ting and Witten [9] have also discovered that among the proposed algorithms for the metamodel layer, the MLR algorithm provided the optimal way of combining not only the probabilities of base layer classifiers but also the way of "smoothing" the prediction probability confidence for different base layer models. Even though this algorithm is specifically related to regression tasks, it was successfully tested on classification problems with each class in a multiclass dataset that could be transitioned into a single regression problem. Our tool does not specifically test the MLR algorithm among the top 11 chosen algorithms, but as to some extent the MLR algorithm can be described as a method of building trees of linear models, we can expect the results to be comparable with those of Random Forest (RF) and Extra Trees (ET) classifiers.

Džeroski and Ženko [24] have further developed the concept defined by Ting and Witten [9] and evaluated the methods of using probability distributions rather than prediction results for metamodel layer input data. They have also evaluated the multi-response linear regression models as proposed by Ting and Witten [9] with additional modifications related to feature engineering of metamodel layer features. Džeroski and Ženko [25] have also investigated the replacement of the originally used multi-response linear regression model for the metamodel layer classifier by more advanced multi-response model trees (SMM5). The study confirmed that the proposed method outperformed both voting and selectBest schemes, as well as the previously used MLR stacking approach.

Seewald [18] has further investigated the approach, presented by Ting and Witten [9] and empirically confirmed, that the proposed solution of using MLR model on a metemodel level performed worse on multi-class problem compared to two class datasets. He explained it with increased dimensionality of the dataset, as there are more features to be considered on metamodel layer. Seewald [18] introduced the new algorithm (StackingC), which was based on dataset dimensionality reduction to make it non-dependent of num-

ber of classes in the dataset. The dimensionality reduction has led to a faster learning, comparing to other ensemble learning methods which also makes sense as the number of features to consider and to be fed to metamodel classifier is less. We use different approach in our toolset, rather focusing on reducing number of features to be used by the metamodel, we try to reduce the overall computing time by focusing on a best performing cluster of base models (provided this cluster outperforms the cluster with all base models), resulting in the overall stacked generalization performance gain. In case this cluster can be identified, it can drastically reduce the training time of overall stacking algorithm as it does not require training of all base layer models.

The StackingC algorithm has been further improved by Menahem [26], which shows to work better with multiclass problems. Although most algorithms considered so far, were operating with two layer stacked generalization, Menahem [26] proposed to ensemble base classifiers in three stages. First stage combined base-classifiers using specialist classifiers, where each classifier solves the specific dual-class problem, using one-against-all binarization method. During the second stage meta-classifiers learned and classified the predictions of base layer classifiers. The third stage is super-classifier layer, where one single classifier is trained using prediction results from the second layer. Menahem [26] confirmed that 3 layer stacked generalization method is a preferable solution for multiclass problems, and even though using more then 2-layer staking is not a common practice, the solution can still be considered as a valuable input to further improvement of overall stacked generalization techniques.

Todorovski and Džeroski [27] introduced a new variant of combining base layer classifiers in a stacked generalization approach by using Meta Decision Trees (MDT). They discovered that, compared to an ordinary DT, MDT leaves, instead of giving the prediction results, could specify the most optimal base-layer classifier to be used for a prediction. The idea behind MDT was to split and assign portions of the dataset to each leaf with each base-layer classifier, working only on a subset of the original data. Todorovski and Džeroski [27] have also confirmed that focusing on class probability distributions rather than class predictions provided a higher overall performance compared to ordinary DTs and several other approaches for stacking. Although we do not plan to use MDT as a metamodel in our toolset, it is interesting to compare some similar models like RF or ET Classifier to other proposed alternatives for metamodels.

In the last decade, a new generation of stacking methods, called evolutionary and swarm intelligent algorithms, like the one presented by Shunmugapriya and Kanmani [20], has been developed and is considered a new era in stacked generalization techniques. Shunmugapriya and Kanmani [20] consider these methods to be inspired by nature and represent the living organisms' meta-heuristic approach to searching and finding the optimal way (optimal solutions). These methods are considered a counterpart and evolution to more traditional heuristic approaches, there the selection of base classifiers and metamodel classifiers is highly dependent on computational power and very time consuming. One additional aspect, which authors of such systems, as Chen and Wong [19] and Shunmugapriya and Kanmani [20] point out, is that in comparison to regular methods for stacked generalization, the defined solution received by the evolutionary methods is better suited for other domain problems. Some examples of evolutionary algorithms include Ant Colony Optimization, introduced by Chen and Wong [19] and Bee Colonies Optimization, presented by Karaboga et al. [28]. As the algorithms' names suggest, the algorithms are based on a swarm intelligence search algorithm based on the behavior of honey bee swarms or ant farms when searching for food. These algorithms have proven to perform exceptionally well on various optimization problems, using the idea of growing

from the random initial population of insects into a well-organized group through several mutations over generations. In a stacked generalization approach, that requires the algorithm to start at some random multiple points and then proceed to an optimal solution. Although this solution can sound tempting to work on, our work is based on a traditional approach due to the focus on the VA aspect of stacked generalization. We focus instead on the representation of particular clusters of base models and their influence on the final solution, as well as metamodel comparison in one-to-one and pairwise comparison modes with regards to overall performance and probability confidence per cluster of base models. Though this heuristic approach requires additional computing power and longer computing time, we believe that from a visual standpoint, this tool can provide additional insights into the ordinal "black box" of stacked generalization methods and algorithms.

In total, our tool focuses on 11 different ML algorithms, both deterministic and stochastic in nature [29], that allow us to compare these different models in order to find the best solution for a given task. When given the same input, deterministic algorithms always produce the same output. There is no randomness involved in the process of these algorithms; instead, they follow a predetermined set of steps to produce their output. On the other hand, stochastic algorithms incorporate a small amount of randomness. Due to the randomness involved in how these algorithms operate, even when given the same input, they may produce different results.

K-Nearest Neighbors (KNN) Cover and Hart [30] define KNN algorithm as a supervised learning algorithm used for classification and regression tasks. In the KNN algorithm, the model makes predictions for a given sample by finding the K nearest data points in the training set and using their labels or values to make a prediction. The value of K is specified by the user, and the model uses a voting system to determine the final prediction. For example, if K=3 and the three nearest neighbors to a sample are labeled "A", "B", and "A", the model predicts that the sample belongs to class "A". KNN can be used for both classification and regression tasks. For classification, it returns the class label that is most common among the K nearest neighbors. For regression, it returns the mean or median of the values of the K nearest neighbors.

Support Vector Machines (SVMs) Hearst et al. [31] presented SVMs as a type of supervised learning algorithm that can be applied both to classification and regression problems. SVM is a linear model for classification and regression, and can be used to find the hyperplane that maximally separates the different classes in the training data. The basic idea behind SVM is to find the hyperplane in a high-dimensional space that maximally separates the different classes. Once this hyperplane is found, new samples can be easily classified by checking on which side of the hyperplane they fall. SVM can handle high-dimensional data efficiently and can also perform well when the number of dimensions is much greater than the number of samples. It is also effective in cases where the data is not linearly separable, as it can use the "kernel trick" to transform the data into a higher-dimensional space where it becomes linearly separable.

Gaussian Naive Bayes (GNB) GNB, as proposed by Agresti [32], is a classification method that uses the Bayes' theorem to predict the likelihood of an event given specific characteristics. It is a probabilistic ("naive") model that makes the assumption that the data's features are unrelated to one another and that they have a normal distribution. Based on the probability density function of the normal distribution for each feature, the algorithm determines the probability that a given sample belongs to each class in order

to make a prediction for that sample. The prediction is then made for the class with the highest probability.

The Multi-Layer Perceptron (MLP) Taud and Mas [33] reference to MLP as a type of artificial neural network that is composed of multiple layers of artificial neurons (or "perceptrons"). It is called a "multi-layer" perceptron because it typically has at least two layers: an input layer and an output layer, nut can also include additional hidden layers between the input and output layers. The input layer receives the input data, and the output layer produces the model's predictions. The hidden layers process the data and transmit it between the input and output layers. Each neuron in an MLP receives input from the previous layer, applies a linear combination of the input data with a set of weights, and then applies an activation function to produce an output. The activation function is typically chosen from a set of nonlinear functions, such as the sigmoid or ReLU function, which allows the MLP to model complex, nonlinear relationships in the data.

Logistic Regression (LR) LaValley [34] mention LR as a statistical method for predicting a binary outcome, such as the likelihood of an event occurring or the probability that an instance belongs to a particular class. It is a type of regression analysis that is used when the dependent variable is dichotomous, or has only two possible values (e.g., 0 or 1, true or false, yes or no). In LR, the relationship between the dependent variable and the independent variables is modeled using a logistic function, which is a sigmoid curve that maps any real-valued number to a value between 0 and 1. The logistic function is used to model the probability that an instance belongs to a particular class, given the values of the independent variables.

Linear Discriminant Analysis (LDA) Riffenburgh [35] already back in 1957 identified LDA as a dimensionality reduction technique that is often used in ML and statistical analysis to project a set of features onto a lower-dimensional space. It is a linear method that seeks to find a projection that maximizes the separation between different classes in the data. LDA is closely related to PCA, which is another dimensionality reduction technique that seeks to find a projection that maximizes the variance in the data. However, while PCA is an unsupervised method that ignores the class labels of the data, LDA is a supervised method that takes into account the class labels and seeks to find a projection that maximizes the separation between the different classes. To perform LDA, one need to specify the number of dimensions one wants to reduce the data to. LDA then finds the projection that maximizes the ratio of the between-class variance to the within-class variance. In other words, it tries to find a projection that maximizes the separation between the different classes while minimizing the spread within each class.

Quadratic Discriminant Analysis (QDA) Riffenburgh [35] has also proposed the QDA as a classification algorithm that is used to predict the class of an instance based on a set of features. It is similar to LDA, but unlike LDA, which assumes that the classes have the same covariance matrix, QDA allows the covariance matrix of each class to be different. This makes it more flexible than LDA, as it can model more complex relationships between the features and the classes. However, this also means that QDA requires more data to estimate the covariance matrices, and it can be less stable when the number of samples is small.

Random Forest (RF) Breiman [36] identifies RF as is an ensemble ML algorithm that can be applied both to classification and regression tasks. It is a type of DT algorithm that builds multiple decision trees and combines their predictions to make a final prediction. In RF classifier, each decision tree is trained on a separate subset of the training data. The final prediction is determined by taking the average of the predictions made by all of the trees. This process of training each tree on a different subset of the data is known as bootstrapping, and it helps to reduce the variance of the model and improve its generalization performance. The decision trees in a RF classifier are constructed using a heuristic called recursive partitioning, which involves recursively splitting the data into smaller and smaller subsets based on the values of the features. At each split, the algorithm selects the feature and the threshold value that results in the greatest reduction in impurity, as measured by a criterion such as the Gini impurity or the entropy.

Extra Trees (ET) ET (also known as Extremely Randomized Trees) as described by Geurts and Wehenkel [37] is similar to the RF algorithm, but it builds decision trees using a more randomized approach, which can lead to less bias and better generalization performance. Geurts and Wehenkel [37] show that Extra Trees can outperform traditional decision trees and other ensemble methods, such as random forests, on some tasks, and they demonstrate how Extra Trees can be used to improve the accuracy of a classifier by combining the predictions of multiple trees. In an ET classifier, each decision tree is trained on a different subset of the training data, and the final prediction is made by averaging the predictions of all the trees. Like in a RF classifier, the decision trees in an ET classifier are constructed using recursive partitioning, which involves recursively splitting the data into smaller and smaller subsets based on the values of the features. However, unlike in a RF classifier, where the feature and threshold values are chosen to maximize the reduction in impurity at each split, in an ET classifier, the feature and threshold values are chosen at random. This randomization process helps to reduce the bias of the model and improve its ability to generalize to new data.

Adaptive Boosting (AB) Rojas et al. [38] describe AB as is an ensemble ML algorithm that is used for classification and regression tasks. It works by building a sequence of weak learners, where each weak learner is a classifier that is only slightly better than random guessing. The weak learners are trained in sequence, and the predictions of each weak learner are combined to make a final prediction. In AB, the weak learners are typically decision trees with a single split, known as decision stumps. The training process involves adjusting the weights of the training instances at each iteration so that the weak learner focuses more on the instances that were misclassified by the previous weak learners. This process continues until the desired number of weak learners has been trained or a stopping criterion has been met. The final prediction of an AB classifier is made by weighting the predictions of the individual weak learners using a weighted majority vote. The weights of the weak learners are chosen based on their accuracy on the training data.

Gradient Boosting (GB) Gradient Boosting, as presented by Friedman [39] is an ensemble ML algorithm that is used for classification and regression tasks. It works by building a sequence of weak learners, where each weak learner is a classifier that moderately outperforms random estimate. The weak learners are trained in sequence, and the predictions of each weak learner are combined to make a final prediction. In Gradient Boosting, the weak learners are typically decision trees. The training process involves fitting a decision tree to the residual errors made by the previous weak learner. The residual

errors are the difference between the predicted values and the true values of the training data. The process continues until the desired number of weak learners has been trained or a stopping criterion has been met. The final prediction of a Gradient Boosting classifier is made by summing the predictions of the individual weak learners. The weights of the weak learners are chosen based on the gradient of the loss function, which is a measure of the error between the predicted values and the true values.

2.2 Hyperparameter optimization

Yu and Zhu [40] define model hyperparameters as parameters that cannot be updated during model training. The idea behind hyperparameter optimization is to identify the optimal set of hyperparameters, leading to the lowest error for a dataset (using cross-validation techniques or a dedicated validation set). These hyperparameters can be related both to the model design itself (like the number of hidden layers in the neutral model or the number of trees in a decision forest) and the model performance (like the learning rate for gradient descent or regularization for linear regression) [40]. This is a computationally costly process, as the process should either blindly try new sets of hyperparameters independently from each try or somehow try to fit the previous leanings into new tries. MetaStackVis tool does not provide an additional algorithm hyperparameter tuning, as it was clearly captured by the StackGenVis system. As our focus is on comparing algorithms on a metamodel level and not the individual model tuning, we have chosen to utilize the hyperparameters set from the top-performing base model for each ML algorithm in order to instantiate metamodels, originating from the same algorithm [4]. Hyperparameter optimization algorithms can be split into 2 main categories: Exhaustive search algorithms, see Section 2.2.1 and Sequential model-based optimization, see Section 2.2.2.

2.2.1 Exhaustive search algorithms

Yu and Zhu [40] consider grid search as a basic algorithm for hyperparameter optimization. Grid search operates with a discrete search space of hyperparameters and exhaustively searches through all possible combinations of the specified hyperparameters to determine which ones give the best performance on the given evaluation metric and selects the best result as the optimal combination of hyperparameters. Grid search is simple to understand and implement hyperparameter optimization algorithm, but it can be computationally expensive since it evaluates the performance of all possible combinations of hyperparameters. Yu and Zhu [40] have also identified that grid search is well suited for parallel processing, as trials can be run independently from each other, but suffers from dimensionality as the complexity increases in an exponential manner with the number of tested hyperparameters. Another disadvantage of grid search is its lack of sensitivity to parameter interaction effects since it only looks at individual hyperparameters and not their interactions. Despite this, it remains an effective and popular method for finding the optimal hyperparameters for a given problem due to its simplicity and systematic approach, is simple to understand and implement, but it can be computationally expensive since it evaluates the performance of all possible combinations of hyperparameters. Random search from another hand has been shown to be more effective in finding better hyperparameter combinations, but at the cost of longer run times. Bissuel [41] mentions that random search in comparison to a grid search provides the randomness to a search space instead of discrediting it with the cartesian product and allows for an even wider exploration of the hyperparameter space. As the algorithm has no formal ending, the search

process continues until a satisfactory set of hyperparameters is found, the maximum number of iterations is reached, or the maximum time allocated to search has been exceeded [40]. Bergstra and Bengio [42] point that random search can also utilize the knowledge acquired during the search process to guide future decisions using techniques like adaptive sampling. It can also be considered a general-purpose optimizer for hyperparameter tuning and can be applied to a wide range of problems despite being computationally expensive.

2.2.2 Sequential model-based optimization

Bayesian Optimization (BO) According to Yu and Zhu [40], bayesian optimization is a method aiming to find a global function minimum with a limited number of trials. The process is based on the Bayes theorem and tries to find the global minimum (or maximum) of the objectively noisy function. BO assumes that a prior belief about the function can be formulated and, with the help of sequential optimization, an optimal point can be determined with a few trials. The BO algorithm includes two key processes: a probabilistic surrogate model to model an objective function and a next point acquisition function. At each iteration, the surrogate model is trained on all previously obtained objective function outputs in order to obtain the best set of hyperparameters in terms of performance for that particular surrogate model. Next, the acquisition function, using the predictive distribution of the surrogate model, evaluates the "usefulness" of various next points, balancing between the exploitation of currently available data and the exploration of a new area of the search space (exploitation vs. exploration) [40]. The algorithm then chooses the point with the highest predicted performance as the next point to evaluate and continues this process until a predetermined number of evaluations has been reached. After the predetermined number of evaluations has been reached, the algorithm outputs the hyperparameters associated with the best-performing point. This output is then used to obtain the final performance of the objective function, which can then be compared against existing approaches. Li et al. [43] prove the BO method to identify good configurations more quickly compared to traditional search methods such as grid or random search and is well suited for complex objective functions with multiple local optima. Yu and Zhu [40] have also identified the advantage of BO over Grid Search and Random Search techniques despite the objective function being stochastic or discrete, or convex or non convex. According to Yu and Zhu [40], BO being a highly efficient hyperparameter tuning technique is still a sequential process in which new trials are run using the experience from the previous ones. Another major drawback of BO, as mentioned by Yu and Zhu [40] is its higher computational cost compared to grid or random search and other optimization techniques such as evolutionary algorithms.

Population Based Training (PBT) Mohapatra [44] describes the Population Based Training as an algorithm which aims to achieve maximum efficiency by combining the best aspects of BO, grid search, and random search while also introducing a self-adjusting population size that adjusts to the performance of the model and estimates of the search space, adding a reference that allows the algorithm to utilize more computation when exploring difficult parts of the search space and fewer computations in simpler parts of a search space, occasionally swapping out inferior agents for nearly identical ones. This process of updating hyperparameters is done in a systematic manner to find the best possible combination while simultaneously allowing flexibility in the range of searchable values. According to Nabi [45], PBT dynamically adjusts hyperparameters such as learn-

ing rate, momentum, and regularization strength over the course of training in order to improve the performance of a given model, including the potential for faster convergence and improved robustness [46]. PBT is not without flaws, though. To explore the hyperparameter space, it most importantly depends on a sizable population [47]. Furthermore, PBT is vulnerable to outlier hyperparameter configurations that could lead to sub-optimal performance [48].

2.3 Validation metrics

Liu et al. [49] define validation metric as a quantitative measure of predicted values and previously generated data in an objective way, meaning that metrics shall perform equally, provided equal conditions . As mentioned by Chatzimparmpas et al. [3], the proper set of performance metrics for different types of analytical problems is crucial and shall be adapted for specific task needs. StackGenVis [7] allows end user to operate with 8 different performance metrics and align them individually based on a given problem. The metrics used should be tailored to the task at hand and be relevant to the problem in order to avoid bias and ensure a valid evaluation of the performance of a system.The currently aligned performance metrics can be summarized into three main groups:

- Threshold
  - Accuracy
  - Precision
  - Recall
  - Geometric-Mean
  - F-beta score (with option to distinguish between F1 and F2 scores)
  - Matthews Correlation Coefficient (MCC)
- Ranking (ROC AUC)
- Probability (Log Loss)

We have decided to focus on these 8 metrics because they are thorough and provide a good overview of how well a model is performing. Additional metrics can be added to better understand the performance of a model, but focusing on the eight main metrics provides us with a good baseline. The addition of the new metrics can be considered a future improvement to both the StackGenVis and MetaStackVis tools. MetaStackVis reuses validation metrics as in StackGenVis (except Log Loss), but with additional new metric (confidence metric), tailored to understand model performance specifically in the domain of meta-stacking. The confidence metric gives a more all-encompassing understanding of the model's performance by focusing on the model overall prediction probability confidence. It is measured in terms of the probability confidence score, which is the average of the model probabilities for each data instance and can be calculated over the entire dataset.

Wardhani et al. [50] define most threshold validation metrics to be calculated from the confusion matrix with two classes. The confusion matrix provides the ratio between properly and improperly predicted values over actual data, where TP (true positives – number of predictions that the classifier got correct for the positive class) and TN (true negatives – number of predictions that the classifier got correct for the negative class) stay for a number of samples, correctly classified, and FN (false negative – number of

predictions that the classifier got wrong for the negative class) and FP (false positive – number of predictions that the classifier got wrong for the positive class) stay for a number of samples, incorrectly classified.

Table 2.2: Confusion matrix

Actual	Positive Predicted	Negative Predicted
Positive	TP	FN
Negative	FP	TN

In this case, the accuracy is a common metric, used to evaluate the performance of a classifier. Accuracy provides the percentage of correctly classified samples and is calculated, as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision is considered "positive accuracy" and identifies the percentage of properly identified positive samples over all positive predictions made. Precision is a measure of the classifier's ability to correctly identify positive examples. A classifier with high precision is one that produces few false positives, i.e., it correctly identifies most of the positive examples that it predicts:

$$Precision = \frac{TP}{TP + FP}$$

Specificity from another end is considered "negative accuracy" and identifies the percentage of properly identified negative values over all real negative values as follows. Like precision and recall, specificity is a useful metric for evaluating the performance of a binary classifier, particularly in cases where the two classes are imbalanced. It is often used in conjunction with sensitivity (also known as the true positive rate or recall) to get a more complete picture of the classifier's performance:

$$Specificity = \frac{TN}{FP + TN}$$

Recall (also known as the true positive rate or sensitivity) is a metric for identifying positive samples, but it calculates out of all possible positive predictions. Recall is a measure of the classifier's ability to find all the positive examples in the data. A classifier with high recall is the one that correctly identifies most of the positive examples:

$$Recall = \frac{TP}{TP + FN}$$

The F-beta score is considered a harmonic mean of precision and recall [50]. The metric is used to balance both precision and recall, and is a better metric for improperly identified samples than accuracy. The balance between precision and recall in this case is controlled by a beta coefficient, which allows the F-beta metric to give more weight to either false positives or false negatives [51]:

$$F_{beta} = \frac{(1 + beta^2) * Precision * Recall}{beta^2 * Precision + Recall}$$

The F-beta score is a generalization of the F1 score, which is the harmonic mean of precision and recall, and is obtained when beta is set to 1. The F-beta score is a useful metric when the relative importance of precision and recall is not equal. For example, if it is more important to have a high precision, even if it comes at the expense of a lower recall, then beta can be set to a value greater than 1. On the other hand, if it is more important to have a high recall, even if it comes at the expense of a lower precision, then beta can be set to a value less than 1.

The G-mean (also known as the geometric mean or F-measure) metric is primarily focused on overall performance on unbalanced datasets (capturing both classes with much higher and much lower numbers of samples) [50]. It is the geometric mean of the sensitivity (also known as recall or the true positive rate) and specificity (the true negative rate) of the classifier:

$$Gmean = \sqrt{Precision * Recall}$$

A lower G-mean value can be interpreted as a sign of low performance in positive sample identification, even when negative samples have been properly identified [52]. In general, G-mean is considered more "optimistic" (having a higher value) than F1-score and less sensitive to class imbalance than other metrics, such as accuracy, which can be misleading when the classes are imbalanced.

The Matthew's correlation coefficient (MCC) performs also well on unbalanced datasets and is considered to be least affected by unbalanced datasets [52]. It is known as the correlation coefficient between predicted and known samples:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{((TP + FP)(TP + FN)(TN + FP)(TN + FN))}}$$

The MCC ranges from -1 to 1, with values closer to 1 indicating a strong correlation between the predicted and true class labels. Value closer to -1 indicates a strong negative correlation, while value of 0 indicates no correlation. MetaStackVis uses the absolute range values of MCC to ensure the same scale measurement across all other metrics, meaning that MetaStackVis does not take into account, whether a correlation has a positive or negative trend, and focuses solely on correlation magnitude.

The receiver operating characteristic (ROC) curve is used to show the performance of a model at various threshold levels and is a graphical representation of the performance of a binary classifier. It plots the true positive rate (also known as sensitivity or recall) on the y-axis and the false positive rate on the x-axis for different classification thresholds. The classification threshold is the probability threshold at which an example is classified as positive or negative. By default, most classifiers classify an example as positive if the predicted probability of the positive class is greater than 0.5, but this threshold can be adjusted. One of the main benefits of the ROC measure is that, it is threshold-agnostic and instead shows how the recall (sensitivity) changes with specificity at different thresholds [53]. The ROC curve is a useful tool for evaluating the performance of a classifier, particularly when the classifier produces probability estimates for the classes. The closer the ROC curve is to the top-left corner of the plot (i.e., the point (0,1)), the better the classifier's performance. The Area Under the Curve (AUC) is measure of the classifier's overall performance and can be found by integrating all the areas under all line segments [53] with an AUC of 1, indicating a perfect classifier, and an AUC of 0.5, representing a *random guess* classifier.

The log loss function applies a different concept than the other metrics shown above and instead operates with a prediction probability of how close the predicted values are to

the actual values for a given class. The higher the difference between the actual value (for a 2-class problem, as considered 1 and 0) and the predicted probability, the higher the log loss value [54]. Log loss is a measure of the accuracy of a classifier that uses probability estimates as predictions. It is calculated as the negative log likelihood of the true labels given the predicted probabilities. The log loss metric is often used in classification tasks, where the goal is to predict the probability that an example belongs to a certain class. Function first calculates the log loss values for each single observation and then calculates the arithmetic average of all log loss values in order to come up with a single performance value. The log loss function is calculated using the following formula:

$$Logloss = -\frac{1}{N} * \sum_{i=1}^{N} [y_i * \ln p_i + (1 - y_i) * \ln(1 - p_i)]$$

where N is the number of examples, y is the true label (0 or 1), and p is the predicted probability of the positive class.

Normalized log loss, used in StackGenVis, is a variant of log loss that is normalized to have a range of 0 to 1. Normalization is a common technique used to scale a variable to a specific range, and it can be useful in cases where the scale of the variable is not relevant or where comparing variables with different scales is desired. To normalize the log loss, the result is divided by the maximum possible log loss, which is the log loss that would be obtained if the classifier always predicted the opposite class from the true label. The maximum possible log loss is given by:

$$MaxLogloss = -\frac{1}{N} * \sum_{i=1}^{N} [y_i * \ln(1 - y_i) + (1 - y_i) * \ln(y_i)]$$

The normalized log loss is then calculated as:

$$NormalizedLogloss = \frac{Logloss}{MaxLogloss}$$

This gives a value between 0 and 1, with 0, indicating a perfect classifier, and 1, indicating a classifier, that is always wrong. Normalized log loss can be useful in situations, where the scale of the log loss is not relevant and it is only important to compare the log loss of different classifiers. It can also be useful when comparing log loss across different classification tasks with different numbers of classes or different class distributions.

Log loss is a measure of the accuracy of a classifier that uses probability estimates as predictions. While, on the other hand, metrics like accuracy, recall, precision, F1 score, and G-mean are all measures of the performance of a classifier that use discrete predictions rather than probability estimates. Combining log loss with other metrics can lead to conflicting or confusing results and therefore it has been decided to exclude the log loss metric from the list of metrics, used by MetaStackVis.

2.4 Visual analytics

As described in the work of Chatzimparmpas et al. [7], the established area of VA is focusing on supporting end user (human) evaluations and decisions by providing some automatic solutions with incorporated ML methods (combined in an iterative way) through interactive visual process representation. This has led to an increased interest in the development of autonomous and intelligent systems, that are capable of providing useful VA insights, and implies that VA is more than just a set of methods for the representation and

understanding of complex data; it is also about how those methods can be used in an intelligent and autonomous manner to provide decision makers with a thorough tool to better evaluate and decide on given problems and tasks. As mentioned by Cashman et al. [55], one of the main contributors to the field of VA is exploratory data analysis (EDA). EDA is a method to explore large datasets with the aim of finding patterns and relationships, discovering underlying structures, detecting anomalies and forming hypotheses. EDA, according to Tuckey [56] allows the end user to get better insight into the available data by discovering the main data structures, identifying the most valuable features, and exploring anomalies and hypotheses. By using VA tools such as EDA, users are able to interact with their data in an efficient and intelligent manner. Cashman et al. [55] have further developed the concept of VA beyond EDA by introducing a new term, called Exploratory Model Analysis (EMA). The EMA [55] primarily focuses not on better exploration of the data used, but on better understanding and definition of the implemented ML models and the effects of modifying their parameters and parameters on overall model performance. Cashman's EMA builds on the concept of EDA by allowing users to better understand and define their ML models. Using the concepts of EDA and EMA, several VA systems, like StackGenVis [3], Beams [57], EnsembleMatrix [58], Manifold [59], Squares [60] and QUESTO [61] have been developed recently. The main goal of these systems is to provide the end user with the ability not only to explore the available data characteristics and discover main patterns and most important features, but also to perform exploratory analysis of available ML models in regards to overall model (or combination of models, like in the form of StackGenVis [3]) performance and provide insight in the iterating process of model hyperparameter tuning. These systems focus not only on providing a more detailed and visual exploratory analysis of the data, but also on using this analysis to better understand and define the implemented ML models and their effects on the overall model performance. By providing users with an intuitive and visual interface for exploring their data, these systems allow them to identify key features and patterns that can be used to better understand the models, optimize their hyperparameters, and create more accurate models. Some of these systems are further described below:

StackGenVis The ISOVIS research group at Linnaeus University has developed a VA system, called StackGenVis [3], that assists users in dynamically adapting performance metrics, managing data instances, selecting the most important features for a given dataset, choosing a set of top-performing and diverse algorithms, and measuring the predictive performance, see Figure 2.3. StackGenVis allows the end user to perform certain data pre-processing and feature engineering steps, provides the possibility to assign specific weights to model performance metrics, and provides the visual representation of each base model's performance through model exploration charts [7]. The main benefit and what distinguishes StackGenVis from similar tools is that it operates not with a single layer of models but with multi-layer models through the stacked generalization ML technique. However, as StackGenVis is focusing solely on the predictive performance of a single metamodel, it does not currently support further exploration into other aspects of other models' performance and the confidence of their predictions. This poses a limitation for end users, as it does not provide an in-depth analysis of how different models and their relative performance can be affected by the data pre-processing steps or feature engineering techniques. As a result, there is potential for further research to extend the current capabilities of StackGenVis by incorporating an exploration into the model's performance and prediction probability confidence. That particular effort would be able to provide an in-depth analysis for a comprehensive understanding of the predictive performance of the

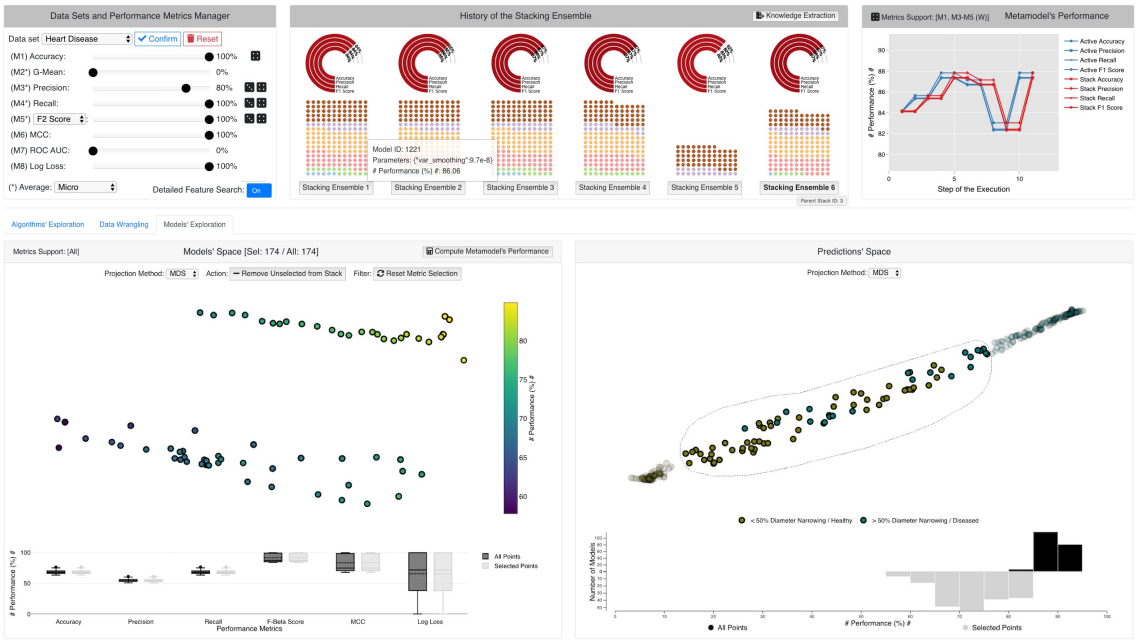

Figure 2.3: StackGenVis user interface [3]. This VA system allows users to visually explore ML models from various algorithms and select the best-performing and most diversified base models. However, it only supports LR as the metamodel. This VA system supports the functionality of loading a dataset and applying the specific weights to performance metric. StackGenVis also enables the examination of the performance history for the constructed stacking ensembles, and it visualizes the models' and predictions' spaces.

metamodel and related models, as well as how different preprocessing steps or feature engineering techniques might influence the overall model performance, which has been addressed as a main benefit of MetaStackVis. This would be a significant advancement in the capabilities of StackGenVis, providing valuable insights for end users when analyzing and understanding the predictive performance of several metamodels and their pair-wise combinations. MetaStackVis does not focus on hyperparameter tuning, as it was extensively captured by StackGenVis, but instead concentrates on further exploration and understanding of the predictive capabilities of single model and pairs of metamodels.

EnsembleLens An integrated VA solution called EnsembleLens, created by Xu et al. [62] offers an ensemble-based, user-guided evaluation of anomaly detection algorithms using multidimensional data, see Figure 2.4. By leveraging the advantages of ensemble analysis, EnsembleLens allows users to analyze multidimensional data more quickly and effectively. This system demonstrates the significance of detectors by their weights, which account for the optimized ensemble, and visualizes the connection between various anomaly detectors. Furthermore, this system highlights the anomalies, detected by different ensemble combinations, and allows users to analyze how these results are impacted by the selection of different ensemble combination algorithms. Through its intuitive interface and interactive visualizations, EnsembleLens helps users understand their data more comprehensively and quickly detect anomalies. In comparison to EnsembleLens, MetaStackvis is focusing on the overall model performance and confidence for a single metamodel as well as pairs of metamodels on a dataset as a whole. Moreover, our visualization tool enables the exploration of specific confusing data instances rather than focusing on anomalies.

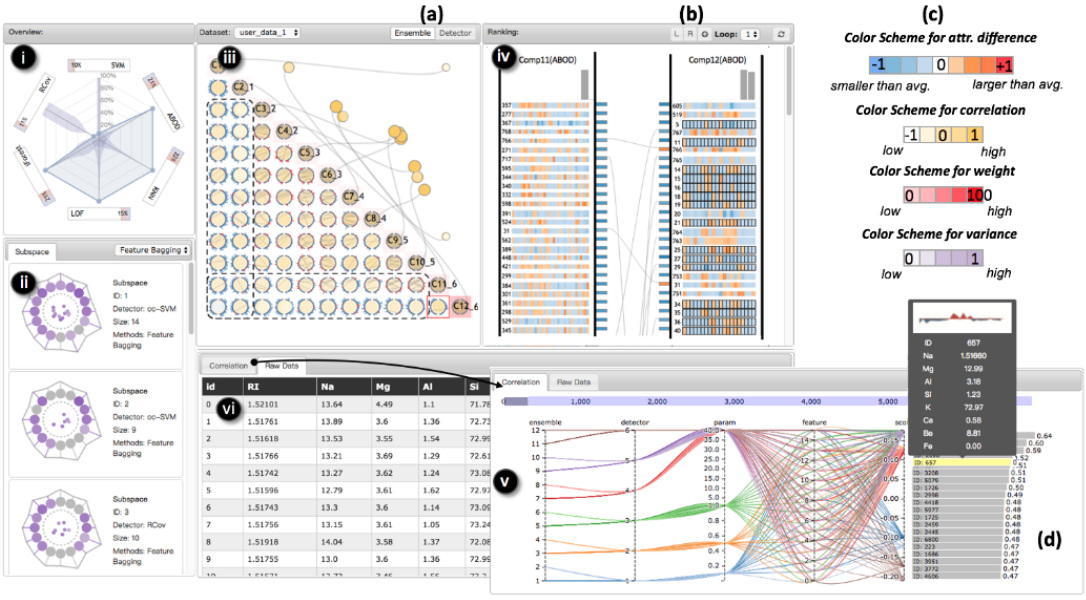

Figure 2.4: EnsembleLens main views [62]. The Detector view, as visible in (i), allows users to change the detection mode and provide feedback by labeling detected anomalous points. Feature subspace view, as shown in (ii), which displays a feature subspace and allows users to explore the data. Inspection view in (iii) containing a global inspection view and a correlation matrix view, as well as allowing users to examine the data in more detail. Ranking view in (iv) displaying rankings of different features or data points. Validation view in (v) which allows users to validate their findings and assess the quality of the data. Raw data table in (vi) for displaying raw data in a table format, with informative tool tips providing descriptions of the data.

BOOSTVis Another VA tool, particularly designed for tree boosting methods like XGBoost or LightGBM, is BOOSTVis [63]. BOOSTVis VA tool includes a comprehensive overview of the tree-boost build process, using tree projections in dimensionally reduced space for ease of visualization and perception, which provides the end user with extensive knowledge of the overall model building process, see Figure 2.5. BOOSTVis provides also the end user with functionality to explore, measure and control various model features for further subtree tuning and performance improvement for underperforming subtrees. This level of insight into the model building process makes BOOSTVis an invaluable tool for both novice and experienced data scientists alike, providing a more thorough understanding of the tree-boost build process and allowing for greater control over fine-tuning model parameters. In addition to the tree view, the confusion matrix can be provided as an interim validation tool to further explore the model's performance during the model training process. In comparison to BOOSTVis, MetaStackVis focuses on the overall model performance and the comparison of pairs of models, while BOOSTVis is geared more toward exploring and visualizing tree-boosts. MetaStackVis is an algorithm- and feature-agnostic tool, as it works with 11 different algorithms and therefore can be applied to a larger portion of classification tasks compared to BOOSTVis.

BEAMS BEAMS is a VA system that offers multi-model exploratory analysis for regression workloads [57], see Figure 2.6. Similar to other VA tools, BEAMS enables the end user to conduct preliminary data analysis and modify hyperparameters for a variety of

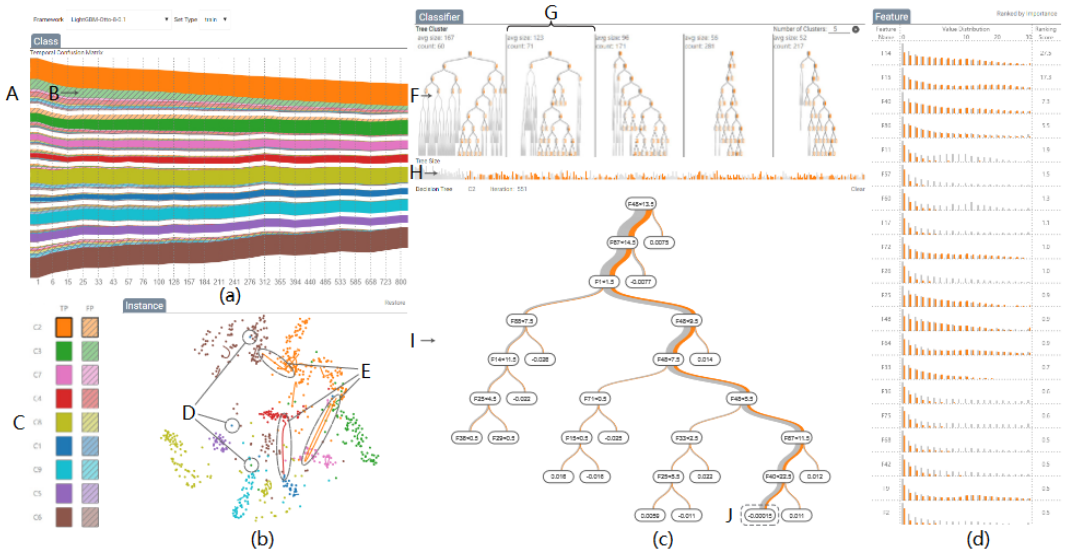

Figure 2.5: BOOSTVis main view [63]. Temporal confusion matrix in (a), displaying the evolution of model performance at the class-level. Instance view, see (b), which uses the t-SNE projection to show the relationships between instances. Classifier view, as shown in (c), which provides an overview of all the decision trees and allows users to select and examine a specific one. Feature view in (d) to show the distributions of features on selected subsets of instances.

models (such as Bayesian regression, linear regression, and LR) in order to improve overall performance and support the selection of the most effective overall regression model. The main goal of BEAMS is to give the end user an easy-to-use EDA and EMA interface by giving an overview of the data instances, outlining the key features, exploring the models that are available, iteratively fine-tuning model hyperparameters, and evaluating model performance using embedded performance metrics like residual scores, mean squared error, etc. BEAMS can supply both a single model and a model ensemble (using bagging) as an output. Even though BEAMS is a simple, fast, and straightforward way to access and improve predictive models, which allows extensive tuning of hyperparameters and focuses on specific data instances, it works with model comparison on a single level and provides limited interaction with dataset and analysis due to focusing on regression tasks only. MetaStackVis applies a different approach, and instead of focusing on a single level of model comparison and regression, it provides an interactive environment that supports meta level comparisons between different models and pairs of metamodels on any two-class classification dataset.

EnsembleMatrix EnsembleMatrix, presented by Talbot et al. [58] is an interactive VA system primarily focusing on EMA of classification tasks, see Figure 2.7. EnsembleMatrix allows users to perform a range of classification tasks, including binary and multiclass classification, feature selection, and data pre-processing EnsembleMatrix heavily relies on the use of confusion matrices to provide valuable input to the end user regarding overall model performance. As some other VA tools, EnsembleMatrix provides iteration possibilities to the end user in order to explore the models and model ensembles. However, EnsembleMatrix goes beyond providing simple iteration options, providing the user with a range of visual aids that give insight into which features and models have a larger im-

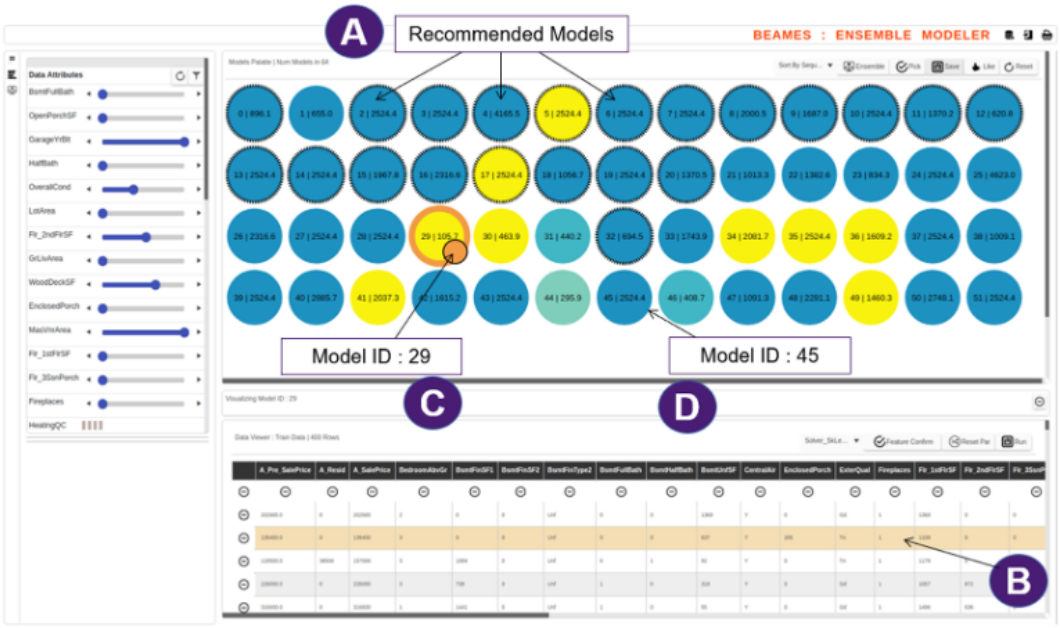

Figure 2.6: BEAMS user interface [57] provides a way to control, select, and examine multiple regression models for tasks. It uses a model view, see (A), that displays circular glyphs representing the models (color-coded by residual error). The interface also includes a data table, as presented in view (B), displaying training, test, and application datasets. A control panel in (C) that enables users to perform selection of models and valuable data points, and adjust feature weights (cf. (D)).

pact on the overall model's accuracy. These visual aids, including confusion matrices and accuracy plots, allow the user to quickly pinpoint important aspects of the data that can be used for further analysis or model refinement. EnsembleMatrix is also capable of providing intuitive and interactive features such as data exploration, feature engineering, model selection, and visualization that are not available with many traditional VA tools. As with MetaStackVis, the EnsembleMatrix focuses on classification tasks, allowing users to quickly build and analyze models with a variety of parameters, but using a different approach by focusing on single classifiers and evaluating the entire set of them in an ensemble by choosing the best individual classifiers, thus improving the overall accuracy.

Manifold One of the main features of the Manifold [59] VA tool, compared to other solutions, is that Manifold has been designed to be mode-agnostic, meaning that it is less focused on the internal logic of the model and only relies on inputs and outputs. This mode-agnostic approach allows Manifold to be applied to a wide variety of models regardless of the underlying algorithm, provided they operate in the same domain, like classification and regression tasks. The main benefit of Manifold is focusing on questionable areas and data points, which allows for the distinction of various ML models and further fine-tuning of those models through extensive feature engineering techniques. This in turn can provide a greater level of accuracy than traditional techniques, as the data can be understood more precisely and each model is tuned to best match its purpose. Manifold shares several similarities with MetaStackVis, like focusing on model pairs while reviewing datapoints that are not properly predicted by both models, see Figure 2.8. Manifold

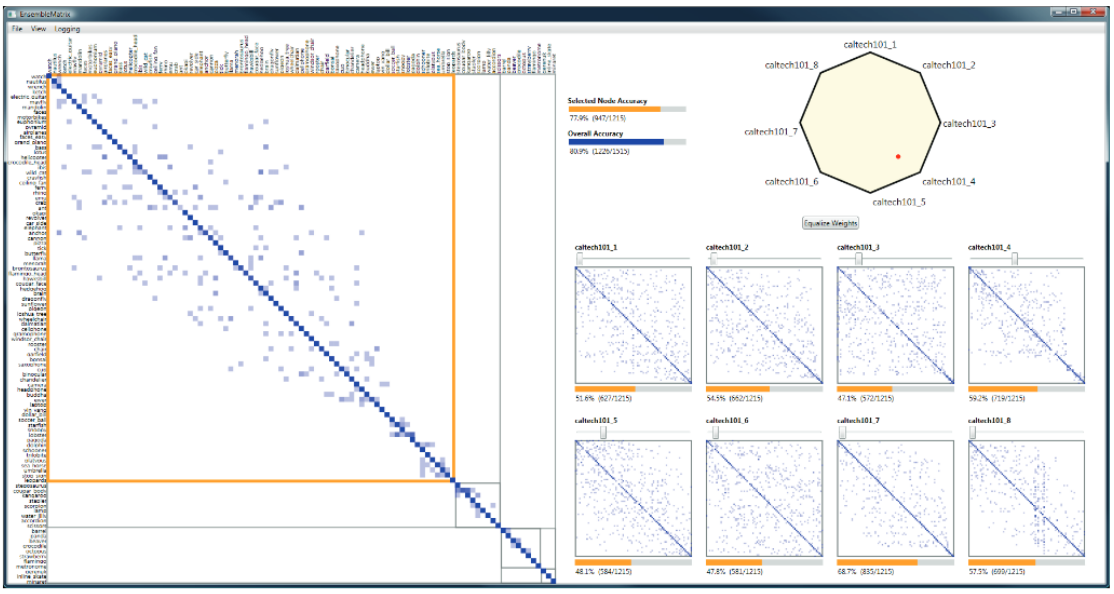

Figure 2.7: EnsembleMatrix primary view [58]. Tool classifier's "confusion matrices" are presented as square images on the right and the ensemble classifier's confusion matrix—which was built by the user—is located on the left of EnsembleMatrix's major view.

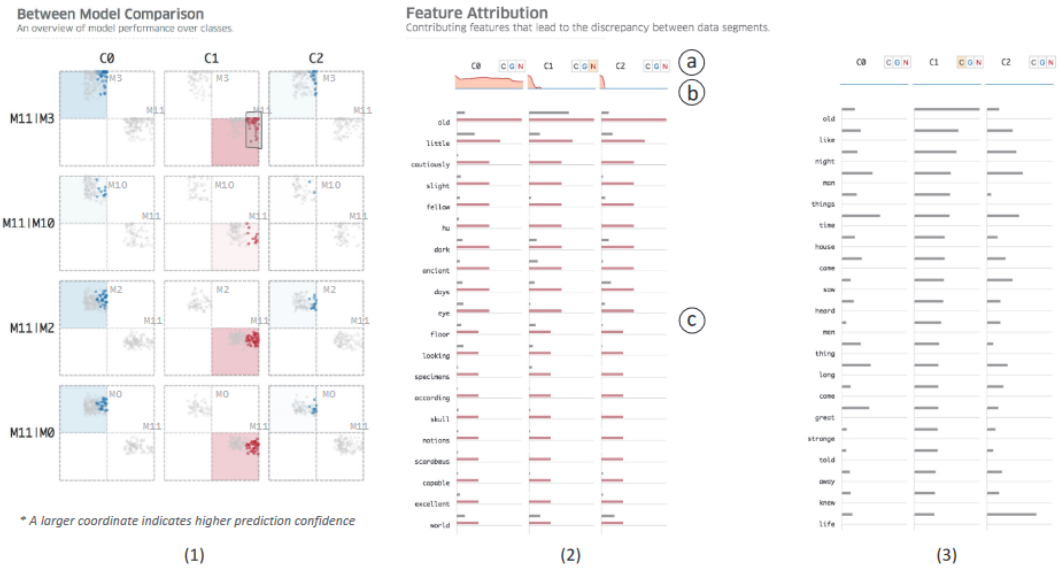

Figure 2.8: Manifold main views [59]. This VA tool includes the model comparison overview in (1), which provides a visual comparison of pairs of models using a small multiple design. The local feature interpreter view (2) which shows a feature-wise comparison between user-defined subsets, as visible in (c), and provides a measure of similarity between the feature distributions, see (b). In (a), the user can sort the features based on various metrics, and then in (3), it is possible to identify the most distinguishing features among different subsets or a specific class.

supports both multi-class classification and regression problems and allows end user to handle large-scale applications with a reduced impact on computation cost [59], but at the same time is focused on single-layer model ensembles, while MetaStackVis is specifically focused on meta layer models' performance. Manifold provides an extensive feature selection process that considers both the relative importance of each feature, as well as its contribution to model performance, while MetaStackVis relies solely on model hyperparameters provided by StackGenVis.

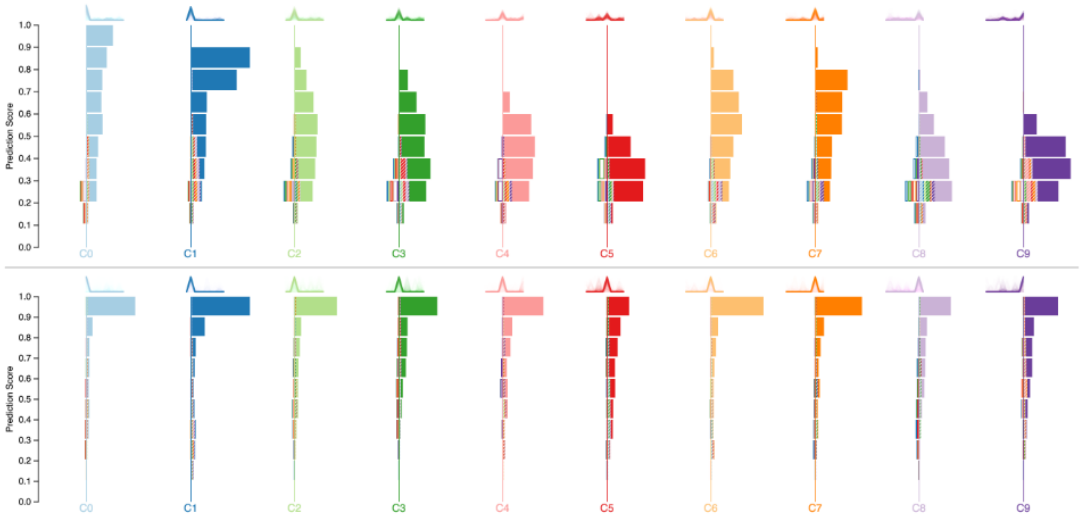

Figure 2.9: Squares overview [60] presenting the performance of classification models for two digit recognition.

Squares Probably the most related VA system to MetaStackVis is Squares [60]. Unlike MetaStackVis, Squares is working on single layer models only and is not focusing on any ensemble techniques. From another side, Squares focuses on classification problems, like MetaStackVis, but provides additional functionality by claiming to the tool to be optimized to work best with 3–20 multi-class problems [60]. Squares provides a comprehensive overview of model performance with additional characteristics related to class skewness, see Figure 2.9. As with MetaStackVis, Squares ensures the end user with the functionality to review the improperly classified particular data instances, going beyond the current functionality of MetaStackVis as it allows to distinguish instances of multiple classes. Squares provides the increased scalability for larger datasets as it allows the end user the functionality to toggle the instance presentation in the form of boxes (for smaller datasets), strips or stacks in order to fit the information on the screen. This suggests that although the two approaches are fundamentally different, they share the common goal of making it easier for users to develop and compare classifiers across multiple datasets.

QUESTO QUESTO is another VA tool, particularly working with model comparison on classification tasks, both on balanced and unbalanced datasets [61]. However, QUESTO provides a different approach to previously presented VA tools. Instead of requiring the end user to operate with pre-defined models, which restricts the interaction possibilities to changing feature or performance metrics weights, adjusting model hyperparameters, etc., QUESTO supports end users (without necessary programming or ML

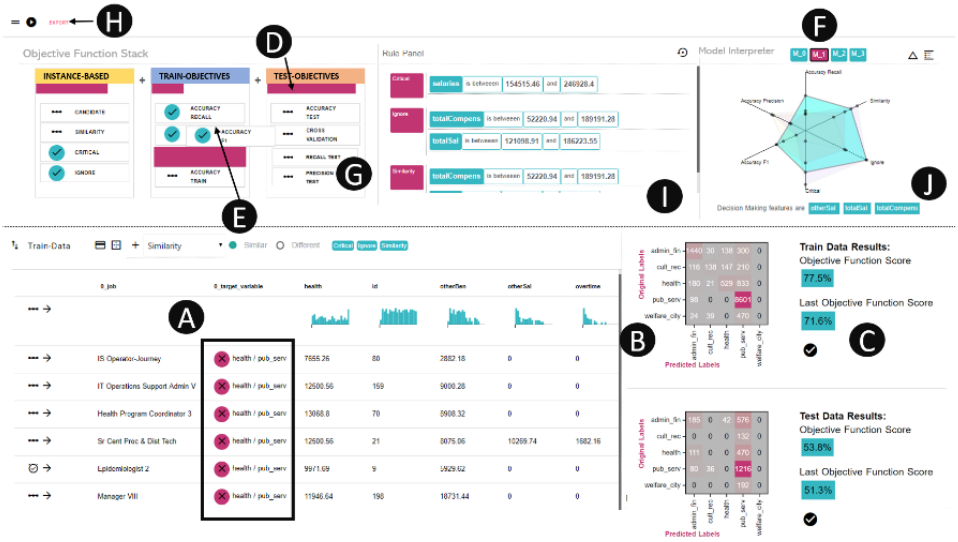

Figure 2.10: QUESTO overview [61] with the view, shown in (A), identifying predictions that are incorrect. Table in (B) that presents the number of correct and incorrect predictions made by a classification algorithm. In (C), a measure showing of how well a model is performing. In (D), we observe the controls to adjust the relative importance of different objectives. Adjustable constraints in (E) that specify the priority of different goals. Thereafter, the model in (F) has been selected. Constraints in (G) on the test data and a button (cf. (H)) to initiate the model run and export the results. A panel in (E) displaying the rules used by the model. Finally the view visible in (F), showing the features that have a significant impact on the model's predictions.

expertise) to design custom objective functions through an interactive visual interface by providing weights to different model objectives, like criticality, prediction, and recall [61], see Figure 2.10. That makes the QUESTO more similar to StackGenVis functionality than MetaStackVis, as feature selection and model adaptation by providing specific weights to model objectives process have been thoroughly implemented in StackGenVis and therefore have not been considered beneficial in MetaStackVis implementation.

2.5 Reliability and validity

The reliability of the data in our case lies in the randomness of the data split, provided by the StackGenVis tool. The train-test split process has been established to ensure that each dataset is split into two sets, namely training and testing sets, with an optimal ratio of 80/20, which means that some minor final model performance data fluctuations can occur due to the random nature of the split. That can be specifically mentioned for particularly small datasets, as with increasing data size, the train-test split process is considered reliable and unlikely to significantly affect the final model's performance.

Another reliability concern is related to the cross-validation technique used for model evaluation, where multiple small datasets are used to fit and evaluate the model's performance. The default 5-fold cross-validation split strategy is used to train final estimators in stacked classifiers, which can affect the final performance to a certain minor degree. However, the effect of this is typically negligible when the data size is large enough.

Stochastic ML algorithms, like neural networks, and boosted tree models, can also introduce variance to final performance results, which are then generally applied to a larger dataset as they use randomness as part of their learning process. Therefore, it is important to monitor the variance associated with these models and ensure that the overall result of an evaluation on multiple datasets is not heavily affected by a particular split or model run.

Additional reliability concerns can be related to general software limitation operations with floating points, resulting in rounding differences in model prediction results and model performance metrics. Even though this is considered to have a small impact on the evaluation process, it is still important to pay attention to it as, over time, such rounding errors can accumulate and result in a large difference when evaluating models.

Another important reliability concern is related to the versioning of used tools and libraries, including the Python main language library, as different versions of these libraries might have slightly different behaviors that, again, can accumulate and affect the results. In order to guarantee a reliable evaluation of multiple models, it is essential to use consistent and stable libraries and tools. The specific list of requirements for library versions used by the current version of the MetaStackVis tool is provided in Appendix A of this paper in order to ensure the most consistent results in terms of data generation, evaluation, and representation.

As the nature behind the model clustering by HDBSCAN is confirmed to be consistent in terms of results yield and is within pure mathematical computations, the validity of the model clustering can be considered highly reliable.

Other technological limitations and possible reliability related to the training and evolution of models in StackGenVis, resulting in some data fluctuations for MetaStackVis datasets, like top model probabilities and top model performance metrics, are captured by StackGenVis and beyond the scope of this paper.

2.6 Ethical considerations

There are several ethical considerations that have been taken into account when working on the practical implementation of the MetaStackVis tool, as well as conducting interviews with subject matter experts.

The input data represents freely available datasets, distributed under Creative Commons (CC) licenses, which allow for the free use, sharing, adaptation, and distribution of the content. Thus, it is important to ensure that this data is used in an ethical manner and that any usage of the content should adhere to the terms and conditions stipulated by the license. These datasets can be both biased and unbiased, so it is imperative to ensure that any analysis conducted using the MetaStackVis tool takes into account the biases, present in the data, and does not lead to false conclusions.

Implemented ML algorithms and tools, like Streamlit, are also available under open-source licenses, and thus the same principles of ethical usage should apply. It was considered not necessary to obtain consent for the use of these datasets and tools, provided that the terms and conditions are respected. It is important to mention that the Streamlit privacy policy outlines their practices for collecting, using, and sharing usage statistics and allows company to capture use age data on default installation. One shall carefully review this policy and consider any potential ethical implications before using Streamlit. If there are any concerns about the collection of usage statistics, one shall consider to opt-out of usage statistics, see https://streamlit.io/privacy-policy.

In order to ensure ethical use of the data, any analysis conducted using the MetaStackVis tool should also take into account how the data was collected and how it might be used in practice. For example, if a dataset contains personal information that was collected without the informed consent of the individuals in question, then any analysis conducted using the MetaStackVis tool shall not allow any identification or tracking of these individuals using the embedded analysis of the tool itself.

Furthermore, any interviews conducted as part of the MetaStackVis review adhere to the relevant ethical standards, and any results have been presented responsibly, ensuring that no data is overgeneralized or misinterpreted. To ensure the anonymity of any individuals involved in the review, all identifying information from the analysis conducted has been obscured in this report. These measures are intended to protect the privacy and security of those who were part of the research process, as well as provide a safe environment for the study and use of data.

3 System Overview

MetaStackVis is a tool for visualizing the behavior and predictive performance of ML base models and metamodels.

3.1 Development environment

Python Python [64] is a widely used programming language that is easy to read, understand, and use due to its simple design and flexibility. Python has a comprehensive standard library with numerous modules and functions for tasks, including extensive data manipulating modules. Python is dynamically-typed, meaning that variable types do not need to be specified when declaring them, making the code easy to write but potentially harder to catch errors. It is an interpreted language, meaning that it is not compiled to machine code and is instead run by an interpreter, making python applications easy to test and debug but potentially slower to run then the ones, made with compiled languages.

Streamlit Streamlit [65] is a powerful open-source library for creating interactive web-based data visualization and ML tools in Python. It is designed to be easy to use, allowing end to build complex, interactive applications with just a few lines of code. Streamlit uses Python on back-end and combination of HTML, CSS, and JavaScript that make up the front-end of the application to create the interactive components of application. It allows end user to create interactive widgets like sliders, buttons, and drop-down menus in order to interact with the application. One of the key benefits of Streamlit is that it abstracts away many of the details of front-end development, allowing end user to focus on the core functionality of application. This makes it easy for data scientists and ML engineers to create interactive tools without needing to have extensive web development experience.

Plotly Plotly [66] is a data visualization library for creating interactive plots and charts in Python. It is built on top of the popular JavaScript library D3.js and can be used to create a wide range of static, animated, and interactive visualizations. On the front-end, Plotly uses JavaScript and HTML to create the user interface of the plot or chart. It includes a variety of interactive features, such as hover text, zoom, pan, and select, that allow the user to interact with the plot or chart in a variety of ways. It also has a number of formatting options that provides the flexibility to customize the appearance of the plot or chart to suit specific needs. In terms of functionality, Plotly offers a wide range of chart types and plot styles that one can use for data visualization. Plotly also has a number of built-in datasets that one can use to experiment with different chart types and plot styles. It is widely used in a variety of fields, including data science, finance, and marketing, and is a popular choice for creating dashboards and other types of data visualizations.

3.2 Implementation

MetaStackVis is designed to be used in conjunction with the Python Streamlit [65] library as the main library for front- and back-end integration and the Python Plotly [66] library for data visualization and can be used to analyze a wide variety of ML models on a metamodel level. MetaStackVis provides an intuitive, user-friendly interface, that allows users to quickly and easily visualize the structure of their ML models, along with performance metrics such as accuracy and precision. MetaStackVis uses several ML algorithms from the Scikit-learn [67] Python package as metamodels. The Scikit-learn

package provides also support for performance metrics (except the G-mean metric, provided by the imbalanced-learn package [68]) for all its models, so it is easy to compare the performance of each model. The HDBSCAN [16] Python package is used to group the base models into clusters based on their predicted probabilities for all test instances. The UMAP [17] Python package is used for dimensional reduction of model probabilities to fit the probabilities from all instances into the two-dimensional workspace, which is then used by MetaStackVis to generate visualizations of both metamodels and base models on the same plot. The Python Pandas [69] and Numpy [70] packages are used for data wrangling and numerical computing, respectively. The complete list of required set packages with the accompanied versions needed for a successful launch and run of the MetaStackVis tool is listed in Appendix A.

3.3 Data exploration algorithm

The main process of data exploration incorporates the main stages, as shown in Figure 3.11. It is important to mention that stages need to be performed in a straight-forward order due to the inter dependency between stages and applications. Once all stages have been initially completed, one can go back and review previous steps in case any additional modifications or fine-tuning of parameters is desired. More detailed overviews of every single stage are provided in the following subsections. The algorithm assumes that the source code has already been downloaded from the GitHub repository `https://github.com/ilyaploshchik/MetaStackVis`, and all necessary packages with the correct versions, as stated in Appendix A, have been installed in the Python environment. One can navigate to the forked folder with the source code in the Streamlit folder and start the Streamlit web interface by running:

```
streamlit run DataLoading.py
```

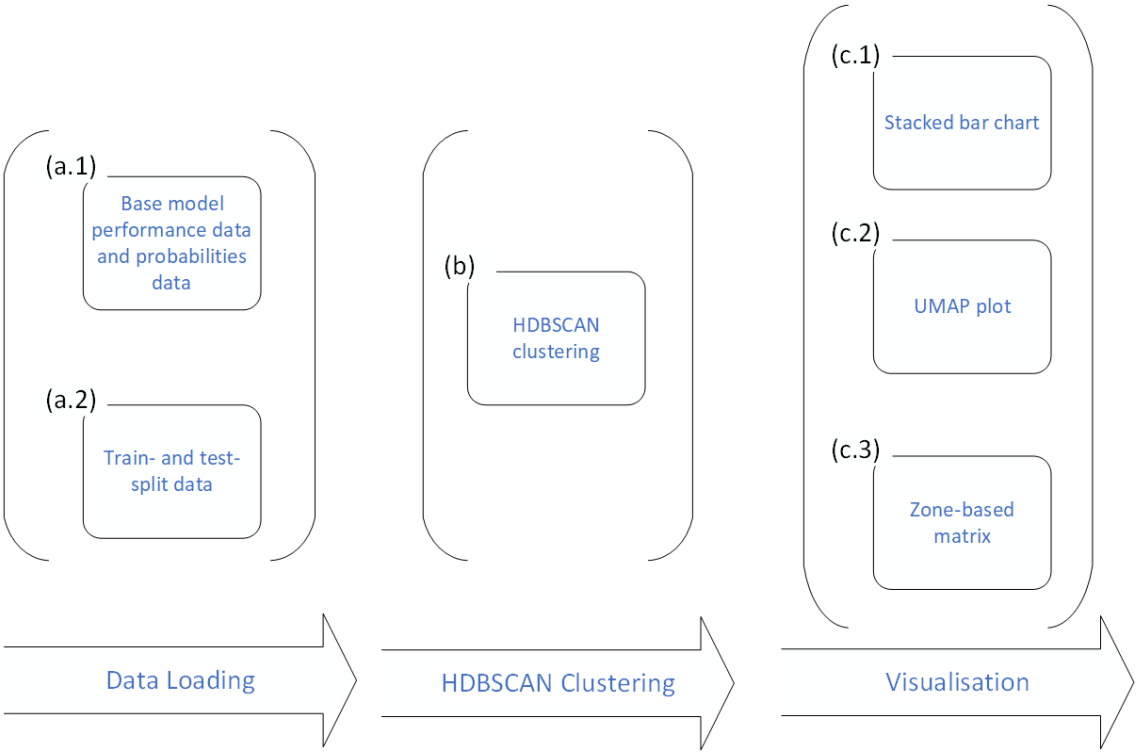

Figure 3.11: The main stages of data exploration in MetaStackVis.

To use the MetaStackVis, one needs to split the data into training and testing for both datasets containing the features and the ground truth labels (see Figure 3.11(a.1)). Afterward, the top-model metric-based performance and top-model predicted probabilities should be exported from StackGenVis, as depicted in Figure 3.11(a.2). Load all of the CSV files into the specific fields on the Data Loading page and click the button to ensure that the data is loaded correctly. The HDBSCAN clustering (cf. Figure 3.11(b)) is completed automatically once the HDBSCAN Clustering tab is activated, but one can perform any additional modifications if needed before proceeding to the Visualization tab. The user then clicks on the button to generate the main plots, and the tool automatically produces the three different visualizations, as described in Figure 3.11(c.1–c.3).

3.4 Data loading tab

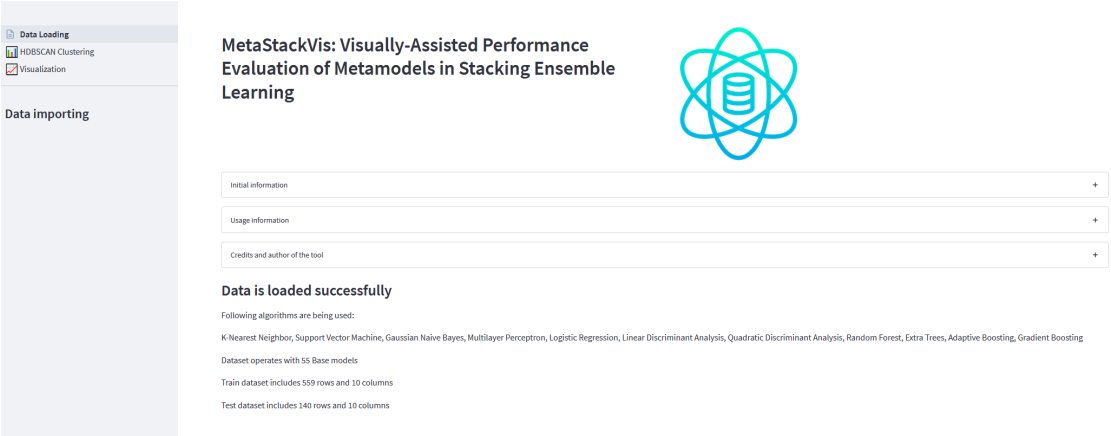

Figure 3.12: Data Loading in MetaStackVis.

Users should first experiment with the most recent publicly available version of StackGenVis [71] before eliciting data, such as predicted probabilities and performance validation metrics for the ML models arranged in two separate CSV files [4]. The hyperparameters used by each ML algorithm to create the exported ML models should be included in the latter file as well [4]. User shall also have access to train and test split data. MetaStackVis is dataset agnostic and therefore the split ratio is considered to be at the discretion of the end user, but one need to ensure that the same data has been fed to StackGenVis to generate the top model performance metrics and model prediction probabilities for the test dataset. The ratio of 80/20 has been considered sufficient given the nature of small size datasets used for tool development, and therefore is the recommended approach for further tool evaluation. Each of the 11 ML algorithms supported by StackGenVis uses pre-defined hyperparameter sets to produce actual models. These models are then evaluated for their performance on the test set data to determine the best-performing model for that particular task. To compare the performance of these models, a range of metrics are used, such as accuracy, precision, recall, and F1 scores. Based on the overall performance of each algorithm, we selected the top five base models from all of our experiments to produce a total of 55 base models. StackGenVis generated the model probabilities and model performance data for each of these 55 base models and compiled the information into 2 csv files to be consumed by MetaStackVis. Once the data has been loaded, the system provides confirmation of the successful data load along with general information about the loaded data, including the algorithms the base models are trained with, the number of

base models, and the sizes of the training and test datasets with the number of features available.

Dataset We describe MetaStackVis functionality in Section 3.5 and Section 3.6 using the Breast Cancer Wisconsin dataset downloaded from the UCI ML repository [72]. We decided to use this dataset to describe the functionality of the MetaStackVis tool because it is a publicly available dataset that allows for easy comparison of the performance of different algorithms. Additionally, it contains a large number of instances and a relatively small number of features, making it a good choice for demonstrating the capabilities of a tool without overwhelming the user with large amounts of data. The Breast Cancer Wisconsin (Diagnostic) [72] dataset is a collection of 699 clinical traits of breast cancer tumors that can help distinguish benign from malignant tumors. It includes features calculated from a digitized image of a breast mass that was sampled with a fine needle aspiration. The features give details about the image's visible cell nuclei and include the following, all ranging from 1 to 10:

- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- Marginal Adhesion
- Single Epithelial Cell Size
- Bare Nuclei
- Bland Chromatin
- Normal Nucleoli
- Mitoses

The target variable includes two classes: benign and malignant, converted into numeric values of 0 and 1, respectively. The data has been divided into training and testing sets using the StackGenVis system in an 80/20 split.

3.5 HDBSCAN clustering tab

After the data load is completed [4], HDBSCAN clustering algorithm [16] groups all 55 base models provided by StackGenVis into separate clusters based on their predicted probabilities for all test instances, see Figure 3.13.

HDBSCAN [73] is a powerful algorithm that relies on the density of points, allowing it to separate clusters with different densities, which makes it well suited for clustering the base models in MetaStackVis. Additionally, our visualization tool enables users to experiment with various hyperparameter combinations for the clustering algorithm [4], see Figure 3.13(a), while optimizing the solution to get the highest *density-based clustering validation* (DBCV) score [73] and the *coverage* score, concurrently.

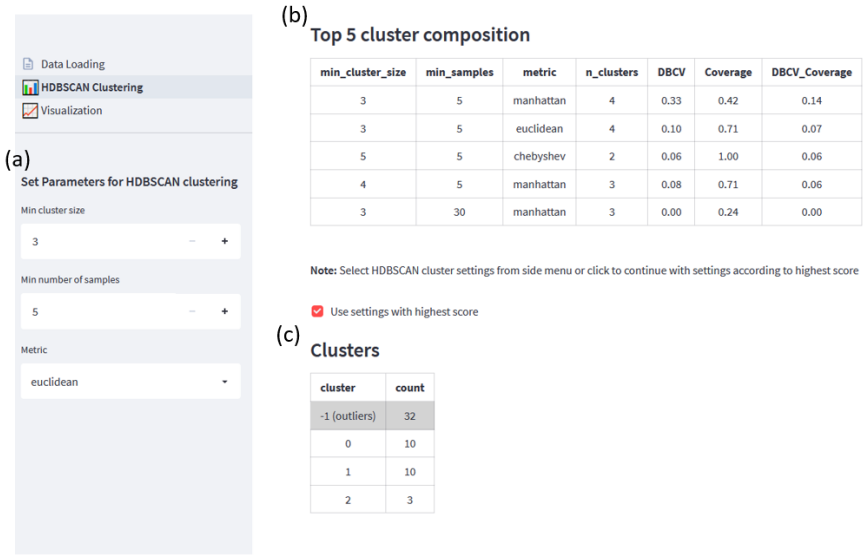

Figure 3.13: The HDBSCAN Clustering tab provides several options for manual tuning of HDBSCAN hyperparameters for cluster optimization, as seen in (a). This tab also provides the overview of the 5 top-performing cluster combinations [4] based on the DBCV score, cluster coverage, and the composition of both scores (b), as well as the best-performing cluster composition visible in (c).

- *Min cluster size* [74] specifies the minimum number of clusters that the algorithm should try to identify in the data. HDBSCAN attempts to find at least that many clusters in the data if the minimum number of clusters parameter is set to a value higher than 1. Depending on the implementation, the algorithm either returns an error or a warning if it is unable to locate the required number of clusters. In some circumstances where it is known in advance that there should be a specific number of clusters in the data, setting the minimum number of clusters parameter can be useful. HDBSCAN is a density-based clustering algorithm, so it's important to keep in mind that not in every dataset the required number of clusters can be obtained. Depending on the structure of the data, the algorithm may occasionally find fewer or more clusters than the required minimum number.

- *Min number of samples* [74] specifies the minimum number of points that are required to define a cluster. In HDBSCAN, *core points* or locations with the fewest other points within a specific distance are the centers of clusters. The least number of points that must be within this range of a point in order for that point to be regarded as a core point is known as *min samples*. A group of points won't be regarded as a valid cluster if it has fewer points than the required minimum sample size. The HDBSCAN-identified clusters' granularity may vary depending on the value of min samples. Fewer, larger clusters come from a higher value for the minimum number of samples, whereas more, smaller clusters would result from a lower value. The structure and qualities of the data being studied should be taken into consideration while selecting a suitable value for the minimum number of samples.

- *Metric* [74] is used to determine the density of points in the data and to identify clusters of high density. The Manhattan distance and Euclidean distance are popular options for the metric.The square root of the sum of the squares of the differences between the coordinates of two points is the definition of the Euclidean metric

[75], sometimes referred to as the L2 norm or the straight-line distance. A distance measurement known as the Manhattan metric [75], sometimes referred to as the L1 norm or the taxicab distance, is the total of the absolute differences between the coordinates of two places. In addition to the above mentioned metrics, MetaStackVis provides an option for the Chebyshev metric [75], often referred to as the maximum norm or the infinity norm, which is the maximum absolute difference between the coordinates of two points used in the calculation of distance. The Chebyshev metric can be advantageous when the data has a specific structure. The Chebyshev metric, for instance, can be used to locate groups of spatially distant points, even though they are not necessarily close to one another in terms of Euclidean distance. Depending on the metric selected, HDBSCAN can identify a variety of cluster sizes and shapes. Because different distance measures could be better or worse suited for different types of data, it is essential to choose an appropriate distance measure based on the characteristics of the data being studied.

The DBCV score is determined by evaluating the density within clusters and the density between clusters in order to contribute to the overall *validity index* [76] of the clusters; a higher density within a cluster and a lower density between clusters indicate a higher DBCV score compared to the opposite case. In order to calculate the coverage score, which represents the number of base models that are part of particular clusters, all base models that are part of a cluster, aside from those that emphasize outliers or noisy data, are divided by the 55 base models. This tends to provide a quality indicator of how well the data is described by the particular cluster composition.

We have also introduced a new quality metric, derived from the multiplication of the DBCV score and the coverage score. It can be considered as an overall ranking score, thus defining five top-performing cluster combinations, as shown in Figure 3.13(b). We believe that this metric provides a better estimation of the quality of the cluster composition than a single DBCV and helps to provide an insight into how well the data is described by the clusters by giving an indication of the expected quality of cluster compositions. The table is sorted in descending order based on DBCV and the coverage multiplication metric. The goal is to reduce the number of outliers by tuning the HDBSCAN hyperparameters in order to achieve the highest possible coverage score, meaning that most of the 55 base models have been allocated to a specific cluster while ensuring—as much as possible—an even cluster distribution [4]. However, this goal can be intentionally sacrificed if the gain in coverage outweighs highly the DBCV score [4].

MetaStackVis gives the end user the option of testing some hypotheses to obtain the desired cluster composition or proceeding with the tool's default cluster combination based on the highest *DBCV_Coverage score* [4]. Please note that outlier models also form a separate cluster and are further used as an input layer for metamodel training as well, see Figure 3.13(c).

After exploring the desired cluster compositions, we proceed with the default settings for HDBSCAN clustering, which provides the highest DBCV-Coverage score of 0.14. It is worth noting that the cluster composition with the highest DBCV-Coverage score is not necessarily going to be the best in terms of model coverage. In our case, we have only 42 percent of all models covered by the clusters, and 58 percent of models remain unclustered and be assigned as outliers. Some cluster compositions can obtain a much higher coverage score (up to 100 percent as for the 3rd cluster composition), but the DBCV score is dramatically reduced, representing the overall low performance quality of the HDBSCAN algorithm with non-sufficient cluster separation and low cluster density.

The results of the default cluster composition are shown in Figure 3.13(c) and in Figure 3.14(b), in parentheses. *Cluster_0* and *cluster_1* each have 10 models, while 32 models are considered as *outliers* (i.e, models not assigned to any cluster). Finally, *cluster_2* includes 3 models.

3.6 Visualization tab

Once the preferred cluster composition based on the HDBSCAN DBCV-Coverage value has been achieved, all 55 base models are split into dedicated clusters. As mentioned in Section 3.5, all models that have not been assigned to a specific cluster (outliers) form a dedicated cluster as well. For comparison reasons, we have also decided to include the cluster with all 55 models as a separate cluster in order to test the hypothesis on whether a metamodel trained on some clusters with a reduced number of models can outperform a metamodel trained using predictions from all 55 base models. Since the StackGenVis system has already extensively applied hyperparameter tuning, we chose to use the best-performing hyperparameters from each ML algorithm's base model instead of conducting additional metamodel tuning in MetaStackVis [4]. However, there are many hyperparameter tuning methods available in the literature [77] that can be used to address this limitation. Base model performance is calculated as an average ratio of performance metrics per model, provided by StackGenVis. The overall algorithm for model training in MetaStackVis is presented in the Algorithm 1.

The visualization [4], shown in Figure 3.14, includes the functionality to adapt some UMAP hyperparameters [17] for the specific cluster of models, as well as presents three different views: *stacked bar chart*, *UMAP plot*, and *zone-based matrix*.

Algorithm 1 MetaStackVis Stacked Classifier

```
for algorithm in algorithms do
    Obtain five base models for each algorithm
    Return the model with the highest performance ratio per algorithm
end for
for cluster in clusters do
    for metamodel in 11 metamodels do
        Define StackedClassifier using base models in cluster as estimators and a metamodel as the final estimator with a default cross-validation of 5
        Fit StackedClassifier to train features and train target datasets
        Predict class values for the test-target dataset
        Predict class value probabilities for the test target dataset
        Generate all 7 performance metrics and generate the average performance value. (Note: MCC is converted to an absolute value in order to be comparable with other metrics.)
        Calculate the average probability metric by taking the average of all the correct class probabilities for the test dataset.
        Calculate the overall model "rank" metric as the multiplication of average performance and average probability value.
        Sort metamodels in each cluster by rank in descending order.
    end for
end for
```

The **stacked bar chart** in 3.14(b) demonstrates the top performing metamodel per cluster, including clusters of all models, and the clusters with models that are not assigned

Figure 3.14: MetaStackVis allows for comparison of the predictive performance of different metamodels through several features: panel, shown in (a), provides the functionality to tune UMAP hyperparameters and select the cluster of interest; stacked bar chart, as shown in (b), identifies all clusters of base models, contributing to that specific cluster combination as well as the most effective metamodel for each cluster with the respective performance validation metrics and the confidence score; UMAP plot, identified in (c), enables the projection of predicted probabilities and performance of base models and metamodels in an active cluster; and the zone-based matrix in (d) that highlights the potential benefits of introducing another metalayer by contrasting the soft voting results (point grids) with the optimal outcome from merging two metamodels (gauge charts). Consequently, it provides an overview of all metamodel pairs and visualizes the performance and prediction characteristics of a single metamodel [4]

to a specific cluster by HDBSCAN. The y-axis shows performance metrics and the confidence metric, while the x-axis is used to distinguish between the top-performing models in each cluster. For every model in each cluster, seven performance metrics have been calculated, including accuracy, precision, recall, ROC AUC, Geometric Mean, Matthews correlation coefficient, and F1 score. The Matthews correlation coefficient is converted to an absolute value on a scale from 0 to 1 in order to be comparable with other metrics, see Section 2.3. The average performance metric has been calculated by taking the mean value of all metrics for a specific metamodel in each cluster. The 8th metric is the confidence score, which is calculated by aggregating the average of all class prediction probabilities for a given metamodel on a scale from 0 to 1, with 1 representing maximum confidence. The multiplied values of the mean performance metric and the confidence metric (divided by 1000) define the overall performance rank. For each cluster, only the top performing model with the highest performance rank has been returned and presented per cluster. Clusters have been further sorted in descending order by the ranking of the top-performing metamodel in each cluster.

This visualization provides an overview of metamodel performance (in percentage % format) per cluster, showing seven performance metrics and the confidence metric for the best-performing metamodel in the cluster. Confidence has been given equal weight

to mean performance, thus it was multiplied by 7 to align with all the other validation metrics. This allows for easy comparison of the two key components of overall performance. These measurements are represented in the legend using various color encodings [4]. Each bar in the graph, which is color-coded for each cluster, indicates one of these metrics. User can deselect a metric and temporarily hide, it if one believes it is unhelpful for the given problem. In general, one can observe that both LDA and QDA, see Figure 3.14(b) were among the top performing algorithms for the majority of clusters. That can be explained by several factors, like the data being relatively clean and well separated. Due to their sensitivity to class overlap, QDA and LDA typically perform better when the classes are well separated, which means that there is little to no overlap between the data points belonging to distinct classes. Assuming that the data points in each class are distributed uniformly, QDA and LDA both anticipate that the data distribution is roughly Gaussian. However, these assumptions shall be considered with certain caution for stacked generalization techniques like MetaStackVis, because, as in our case, the metamodel is trained not directly on the data but on the prediction probabilities of the base model layer. Another interesting observation is that it is not necessary to use all base models for layer 0 training. A cluster with metamodel (Figure 3.14(b)), based on QDA algorithm, trained on input data from all 55 base models' predictions, is only on the 3rd place in ranking, with both the overall performance and probability confidence being lower, compared to the most performing and confident model, based on LDA algorithm, trained on a cluster with only 10 base models' prediction probabilities. This can successfully prove the original hypothesis of obtaining the meta model classifier trained on a portion of the original data predictions, which can greatly reduce the computational expense and time.

The **UMAP plot** in Figure 3.14(c) helps to visually examine the base models (contributing to the selected active cluster) and the 11 metamodels that are trained on base models' predicted probabilities. This allows for a more in-depth analysis of the metamodels in relation to the base models [4]. Each point represents a single model, while the size defines whether the model belongs to base models or metamodels, with metamodels being 2 times larger in size than base models. The high-dimensional predicted probabilities for all datapoints for the given test dataset are projected into two-dimensional space by the UMAP [17]. The UMAP projection can be further adapted by fine tuning the following parameters, available to the end user on the left side menu of the tool:

- *Number of neighbors* in range 3 to 5 represents the size of the local neighborhood that UMAP considers when making projections;
- *Metric* (Euclidean or Manhattan): UMAP uses the metric to calculate distances between points and determines how similar or dissimilar two points are when making projections; and
- *Minimal distance* (0.2 or 0.5): UMAP uses the minimal distance parameter to determine the minimum distance that a point can be from another point in order for it to be considered valid when making projections.

In our example, the UMAP projection has been received from the model prediction probabilities of approximately 140 instances of the test dataset (which corresponds to approximately 20% of the whole data prior to the train/test split in an 80/20 ratio). Each point is accompanied by a summary window, shown by hovering over the point. The summary window includes information about model type (base or metamodel), the corresponding ML algorithm this model is based on, Model ID, the overall mean performance metric,

all seven performance metrics contributing to the mean performance metric, and the confidence score, representing the average of all the correct class probabilities for the test dataset by a given model. All points have been color-coded by the metric-based mean performance value in Figure 3.14(c), using the Viridis colormap [78] with main legend, scaling from the least performing value to the highest performing value, rounded to the closest largest or smallest even number respectively, and shown on the right-hand side of the plot. The legend on the left side of the visualization uses 11 distinct symbols to represent the various ML algorithms, while the opacity of the models corresponds to the confidence level, with a higher value illustrated with a more opaque model and a lower value depicted in a more transparent way. One can also pan and zoom to specific areas of the plot for better point distinction.

In our case for the top-performing metamodel in cluster 0, with the *number of neighbors* equal to 3, the Euclidian *metric* and the *minimal distance* being equal to 0.2, we can observe highly discernible clusters of models, including base models and metamodels. The base model cluster consists mainly of base models from the RF and ET algorithms. The ET models, including the base and metamodel, have been grouped together, while the RF base models have been clearly clustered with the KNN metamodel. This could be because both models are based on decision trees, which are a type of ML algorithm that involves constructing a tree-like model of decisions and their possible consequences. ET and KNN both use decision trees as the basis for their models, and they both involve making predictions based on the characteristics of nearby data points. Additionally, UMAP is sensitive to the underlying structure of the data, and it may be that the structure of the data is such that ET and KNN models are naturally clustered together.

The **zone-based matrix** in Figure 3.14(d) is inspired by the scatter-plot matrix [79] and offers a more complete view of the performance of the metamodels [4]. A zone-based matrix is composed of three sections: the diagonal, the lower-left portion, and the upper-right portion. A zone-based matrix represents the performance data, as well as pair-wise model comparisons for all 11 metamodels trained using the model probability data from base models of a chosen cluster, see Figure 3.14(b). Models are sorted in descending order based on their overall rank (combination of mean performance and confidence level).

The matrix diagonal line visualizes the validation metrics in the metric-based performance bar chart for the specific metamodel with specific metric values available, then hovering over the chart. A bar chart shows the confidence level as well, shown above the chart. The confidence level is also represented by color (opacity), ranked from most to least effective. Black (the most visual) color denotes the highest confidence value, while white (the least visual) color is the lowest possible.

The lower triangle shows the union of all test examples that were incorrectly classified by at least one metamodel pair, meaning the probability of predicting the correct class was less than 50% for any of the total of 11 metamodels. In our case, we have 20 such data points (20 circles) in total. Test samples that are easiest to classify are at the top of the grid (in white, if classified correctly by both metamodels) and those that are hardest to classify are at the bottom (in yellow, if classified incorrectly by both metamodels) because the points in all grids are sorted (from easiest to hardest to classify) based on the predicted probabilities of all metamodels in combination. We use the "soft majority" voting strategy [80] of combining the models' predicted probabilities for a specific entity with dark red when the row-wise (red) metamodel is unable to overcome the wrong prediction of the column-wise (blue) metamodel, meaning that even though the probability confidence of the prediction by the red model was higher than 50%, the overall confidence in the correct results was under 50%. The opposite case scenario, with column-wise (blue)

metamodel prediction probability confidence not being able to overcome row-wise (red) metamodel confidence, results in a dark blue circle. The latter light blue and light red circles represent scenarios then corresponding model probability confidence surpasses the other metamodel, resulting in the correct class prediction by a particular pair. Therefore, the use of more prominent colors emphasizes the points that show the inability of metamodels to accurately predict these points [4].

The upper triangular part, from another side, represents the "maximum potential" predictive performance that could be attained if the optimal metamodel was chosen for all the test instances (140 in our case) for each combination of metamodels [4]. The gauge chart represents the pairwise model contribution to the end result and answers the question of how well two models describe the data and predict the correct results in the event that an optimal model from each model pair is chosen for a specific data instance. The optimal model is chosen simply by operating with the highest prediction probability for every given instance of the pair of models, meaning that we do not focus on "soft voting" (mean probability) here and instead allow the end user to explore the maximum theoretical performance of the metamodel pair-wise ensemble. The gauge represents the mean validation performance (average of 7 performance metrics) for the metamodel pairwise ensemble in orange with the additional visual representation of performance and confidence color comparison: green if performance is higher than confidence and purple in the opposite case. These levels have also been shown as values under the gauge, with the confidence level presented above the performance value. Figure 3.14(d) examines metamodel pairs to show the potential for additional schemas, such as adding an extra stacking layer to combine the predictions of the first metalayer [4].

4 Hypothetical Usage Scenario

We have decided to use the Pima Indian Diabetes dataset instead of the Breast Cancer Wisconsin dataset for evaluating MetaStackVis in order to test the tool's performance and usability on a variety of data types and structures. This allows for a more comprehensive assessment of the tool's capabilities and limitations and can help identify areas for tool improvement. Even though both datasets are moderately complex with similar numbers of features and instances, we expect the Pima Indian Diabetes dataset to be more complex due to the continuous nature of its features, which may make it a more challenging dataset for MetaStackVis to handle. The Pima Indian Diabetes dataset is a collection of medical data used for ML and statistical analysis and includes information related to 268 positive and 500 negative diagnosis instances, consisting of eight attributes/features [81]. It is often used to predict the likelihood of an individual developing diabetes, based on various characteristics such as age, blood pressure, and insulin levels. The dataset contains a total of 768 observations, or records, and each observation includes the following variables:

- Age: (in years)
- Blood pressure: diastolic blood pressure (mm Hg)
- Body mass index (BMI): body mass index (weight in kg/(height in m)^2)
- Diabetes pedigree function (DFP): a function that measures the likelihood of an individual developing diabetes based on their family history
- Glucose: plasma glucose concentration after a 2-hour oral glucose tolerance test
- Insulin: 2-hour serum insulin (mu U/ml)
- Pregnancies: number of times the individual was pregnant
- Skin thickness: triceps skin fold thickness (mm)

We will try to improve the prediction of whether an individual has diabetes (1) or not (0) using ML models, particularly stacking ensembles. After exploring the data, we decide to proceed with using all 8 features of the dataset and perform the training-test split in an 80/20 % ratio. We want to utilize the functionality of StackGenVis to review and tune the performance of ML algorithms on a single layer, but prefer to further optimize the solution by reviewing the stacked classification functionality of MetaStackVis in order to try to further improve the predicted performance and probability confidence of ML algorithms. We would also like to optimize the solution by reducing overall computational time and expense by checking the hypotheses wherever we can obtain similar or better prediction results using a subset of the original base models from StackGenVis. We also want to check the hypotheses on where the pair of metamodels can provide better results compared to single metamodels in stacked generalization training. Armed with the StackGenVis and MetaStackVis tools, we are ready to explore the potential of stacked classification algorithms to achieve better prediction performance, improve overall probability confidence levels, and increase computational efficiency.

Data Loading After successful loading of train and test generated data from StackGenVis, as well as StackGenVis output data in the form of model prediction probabilities and model performance data for all 55 top base models, split by 5 top models per ML algorithm, into MetaStackVis, we verify the successful data loading on the front of MetaStackVis by reviewing a summary, showing the information about successful data loading of train and test data with the number of rows and number of features loaded, as well as other useful information, such as the ML algorithms used by base models and the total number of base models in the model performance and model probabilities datasets, see Figure 4.15.

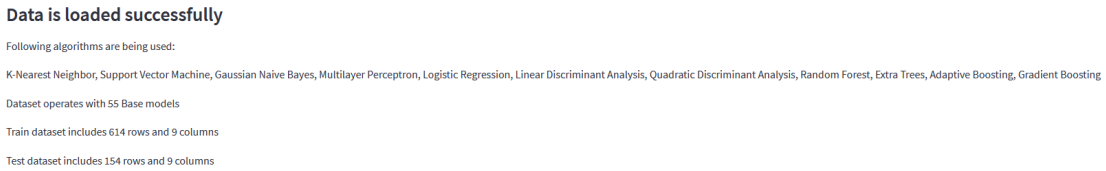

Figure 4.15: Data loading for the Pima Indian Diabetes dataset. We confirm that all 768 observations have been successfully loaded, split into 614 rows in train data and 154 instances for testing and validation purposes. We can also verify that all 11 original algorithms have been successfully used, and that data is generated by 55 base models in total.

Cluster Overview We proceed further to the HDBSCAN clustering tab and review the top 5 cluster compositions, presented by MetaStackVis, see Table 4.3.

Table 4.3: Top 5 cluster compositions with the best-found DBCV and coverage for Pima Indian Diabetes dataset. Cluster compositions are sorted in descending order by DBCV-Coverage metric with best cluster composition, consisting of 4 clusters (including outliers) and DBCVCoverage score of 0.09. Cluster composition is generated by using HDBSCAN hyperparameters as following: *minimum cluster size* of 3, *minimum number of samples* of 5, and *metric* as "chebyshev".

Top 5 cluster composition

min_cluster_size	min_samples	metric	n_clusters	DBCV	Coverage	DBCV_Coverage
3	5	chebyshev	4	0.17	0.49	0.09
8	5	chebyshev	3	0.17	0.40	0.07
3	5	euclidean	3	0.11	0.45	0.05
5	5	manhattan	3	0.08	0.55	0.05
3	20	euclidean	3	0.05	0.55	0.03

We decide to keep the setting for attaining the highest score for the HDBSCAN clustering algorithm, which is activated by default. The hyperparameters are automatically set to 3 for min_cluster_size, 5 for min_samples and the "Chebyshev" metric as shown in Table 4.3. These hyperparameters generate a cluster composition with four clusters, overall DBCV score of 0.17, and Coverage score of 0.49—meaning that 49% of all base models

have been successfully assigned to clusters. An important observation can be made to the low DBCV score of only 0.17, as it would indicate that the clusters produced by the HDBSCAN algorithm have relatively low density compared to the density of points between the clusters. DBCV score that is closer to 1 indicates better-defined clusters, while a score that is closer to 0 indicates less-defined clusters. However, we understand that the interpretation of the DBCV score can depend on the specific context and the desired properties of the clusters, and therefore decide to proceed as is.

We review the cluster composition and confirm the coverage percentage by having 28 out of 55 models identified as outliers, see Table 4.4. We also observe some uneven distribution of base models among other clusters, with cluster 0 consisting of 5 models, cluster 1 consisting of 10 models, and cluster 2 consisting of 12 models in total.

Table 4.4: Detailed cluster overview for specific cluster composition for Pima Indian Diabetes dataset with 4 clusters in total, including 28 models, identified as *outliers* and defined as a cluster as well; other clusters include 5, 10, and 12 models each respectively.

Clusters

cluster	count
-1 (outliers)	28
0	5
1	10
2	12

Detailed Exploration of Base Models and Metamodels After the HDBSCAN clustering review, we proceed to the main visualization tab and focus on the cluster composition stacked bar chart, ref Figure 4.16.

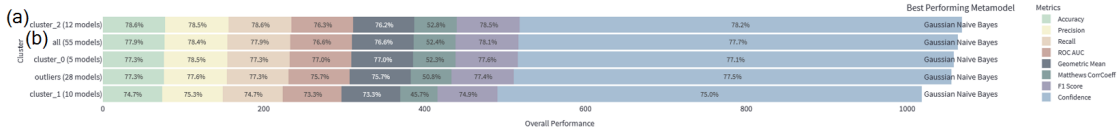

Figure 4.16: Stacked bar chart. The stacked bar chart displays the overall performance of the top-performing metamodel for each cluster, using seven validation metrics and confidence [4]. View (a) shows that even though cluster_2 consists of fewer models (i.e., 12 base models instead of 55), its best performing metamodel achieves better overall performance compared to a metamodel trained using probability predictions from all based models, as shown in view (b).

The stacked bar chart shows the best metamodel per cluster's performance and probability confidence. By knowing that clusters in a stacked bar chart are sorted in descending order by the overall top-model per cluster rank (multiplied value of mean performance and probability confidence), we can make an important observation [4] that even though fewer

number of models (i.e., 12 base models instead of 55) contribute to *cluster_2*, see Figure 4.16(a), the best performing metamodel in *cluster_2* achieves higher overall performance compared to best metamodel trained according to probability predictions from *all* based models, ref. Figure 4.16(b). Even though the overall performance difference is not that big, can also make the proper assumption that using only 12 out of 55 base models for staked classifier can drastically reduce the training time for classifier fit and can reduce the overall cost for production pipeline run.

We continue to compare the top-performing cluster 2 of base models with the cluster consisting of all models, and now focus on *cluster_all*, ref Figure 4.17, using the Manhattan metric, shown in Figure 4.17(a) from the drop-down side menu, as it is preferable for high-dimensionality data [82].

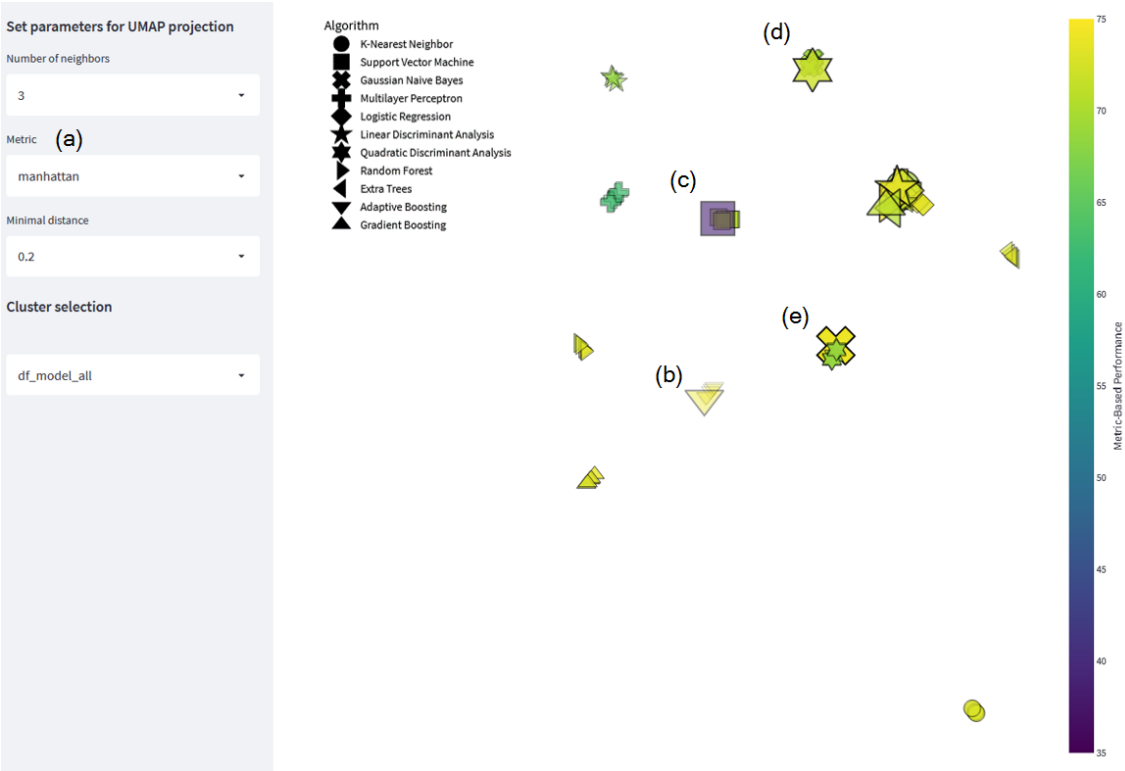

Figure 4.17: UMAP plot for *cluster_all*. UMAP [4] captures base models and metamodels providing similar predictive performance on the same data instances in views (b), (c), (d) and (e) using Manhattan metric, ref. view (a) as hyperparameter for UMAP.

An interesting observation when examining Figure 4.17 is that the base models are categorized based on their underlying algorithm and generally attain similar predictive performance, regardless of the hyperparameters selected [4] (as StackGenVis provides the top 5 performing base models per algorithm). The similar behavior can be observed for metamodels, like Figure 4.17(b) and Figure 4.17(c) are formed because metamodels predict similarly the same test instances. However, some metamodels in Figure 4.17(d) and Figure 4.17(e) have swapped places, with Gaussian NB being closer to QDA and vice versa [4]. This pattern warrants further examination.

We continue the tool review by focusing on the most performing *cluster_2*. From Figure 4.18, we may observe that the majority of metamodels contribute to the same group Figure 4.18(a). An exception here is Figure 4.18(b) with lower model performance compared to the rest of the stack, which only includes some tree-based ML algorithms such as

Figure 4.18: UMAP plot for top performing *cluster_2*. View (a) demonstrates that most of the metamodels have similar performance, view (b) highlights that tree-based ML algorithms have lower performance compared to other methods, and view (c) focuses on the two metamodels with the lowest level of confidence.

GB [4], and Figure 4.18(c) with AB and SVM with lower performance and lower prediction probability compared to the rest of the stack (as it can be observed by color and opacity level with SVM and ADA being almost transparent). We would like to investigate these algorithms further by making some assumptions for possible reasons for this behavior, as listed below:

- Incompatibility with the rest of the stack: the performance of a ML model can be affected by the models it is stacked with. If GB, ADA, or SVM are not compatible with the other models in the stack, they may not perform well.
- Over-fitting or under-fitting: GB, ADA, and SVM can all be prone to overfitting or under-fitting, depending on how they are tuned. If they are not properly tuned, they may not perform well in the stacked generalization classifier.
- Limited capacity: GB, ADA, and SVM can only model relationships between the input features and the target variable up to a certain complexity. If the relationships in the data are more complex than the model can handle, the model may not perform well.
- Lack of sufficient data: GB, ADA, and SVM can all benefit from large amounts of training data. If the data available for training is limited, the model may not have enough examples to learn from and may not perform well.

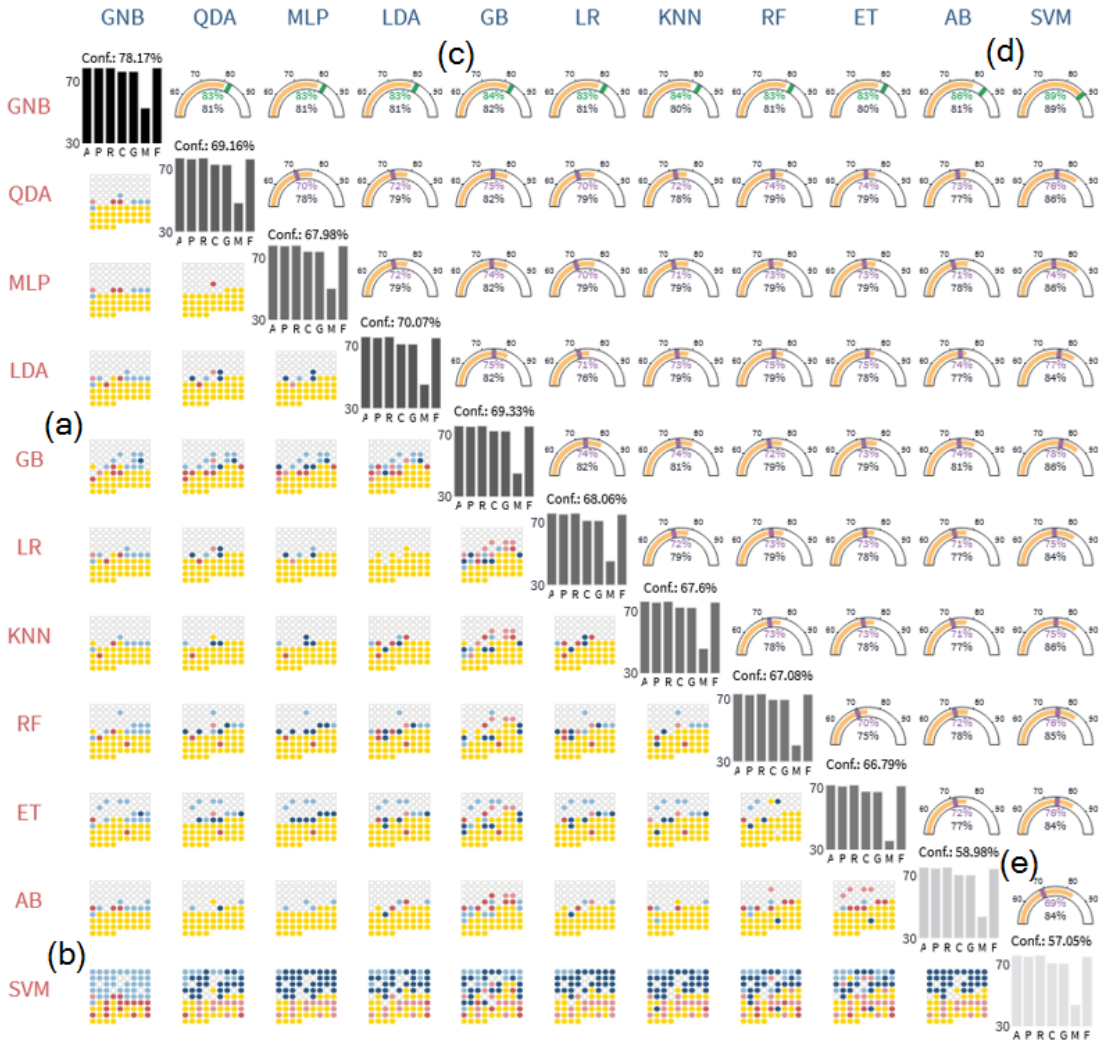

Figure 4.19: Zone-based matrix for top performing *cluster_2*. Views (b), (d) and (a),(c) show some promising metamodel combinations, while view (e) shows the validation metrics in the metric-based performance bar chart and probability confidence values for AB and SVM models

Pairwise Combination of Metamodels We can confide that the last group's metamodels (AB and SVM) have the lowest confidence according to Figure 4.19(e) by looking at probability confidence values of 58.98% and 57.05% respectively, meaning that these models are confident in their predictions in less than 60% of all instances. However, we may still see some benefit of combining diverse metamodels. The combination of GNB and SVM, for example, has both a maximum prediction probability score and performance score of 89%, as shown in Figure 4.19(d), using a theoretical maximum performance strategy (meaning that for every instance, the model with the highest probability confidence score has been chosen). GNB is able to accurately predict most simple test instances when used with a majority voting strategy, as seen in Figure 4.19(b). However, the SVM is not able to surpass the performance of the GNB for the test cases that are difficult to classify, as indicated by the dark red colors in Figure 4.19(b) [4].

Another impressive model pair is GNB and GB, which has the potential to achieve a maximum metric-based performance of 82% and a prediction probability confidence

score of 84%, see Figure 4.19(c). According to Figure 4.19(a), GNB correctly classified and outperformed GB in nine cases, while GB correctly classified one case, but neither algorithm was able to achieve a significantly higher level of probability prediction confidence than the other as the yellow instances were misclassified by both algorithms. Despite this, the overall performance of this combination is still quite strong, with most instances correctly classified. This is indicative of a successful pairing of the two methods, which can be attributed to the fact that GNB is generally good at predicting well on instances that are easier to classify, while GB's strength lies in its ability to accurately predict on more complex instances. That also confirmed the hypothesis that combining the two methods would provide a stronger performance than either one used on its own.

5 Evaluation

We conducted semi-structured, asynchronous interviews with four experts to gather qualitative feedback on the effectiveness and applicability of the MetaStackVis tool [4], using "Pima Indian Diabetes" dataset [81]. This feedback was used to identify areas for improvement in the user experience of the MetaStackVis tool.

5.1 Experts

The first ML expert (**E1**) has PhD in the field of mathematics and 4.5 years of experience with ML. He confirmed having good knowledge and experience of ensemble learning techniques, particularly in the context of reinforcement learning, and is working on research projects related to image recognition and reinforcement learning. The second ML expert (**E2**) has 7.5 years of experience in ML, and a PhD in media technology and visualization. **E2** has experience with computer graphics, image processing, ensemble learning, and deep learning, particularly methods such as generative adversarial networks (GANs). The third VA expert (**E3**) has a PhD in computer science with 12 years of experience in the field of data visualization and 6 years of experience with ML, particularly related to data mining, dimensionality reduction, and clustering methods. The fourth VA expert (**E4**) confirmed working with InfoVis and VA, mainly related to natural language processing (NLP) and network data, with approximately 10 years of experience. Additionally, he has a PhD in InfoVis sciences and two years of experience with stacking ensembles.

The first three experts reported no issues with colorblindness. Although **E4** has a mild case, he confirmed that he had no difficulty accurately identifying the color combinations provided in the MetaStackVis [4].

5.2 Methodology

As the MetaStackVis tool requires an extensive amount of time to train all the allowed classifier alternatives and present the charts, all computations have been completed prior to the interview. The experts were given a detailed explanation of the interview procedure and its purpose prior to beginning the session. Prior to the interview session, a confirmation of consent for recording was obtained from each of the experts to ensure that all of their recordings would remain confidential and would only be used for the purpose of analyzing and improving the user experience of MetaStackVis.

Each interview lasted approximately one hour, and the interviews were arranged in three main phases:

1. In the first phase, the experts were given a short introduction to MetaStackVis and the reasoning behind creating the tool.

2. After the introduction, the experts were guided through the tool interface and asked to provide comments on each step of the process while they got presented test scenarios to investigate the available dataset.

3. The experts were asked questions to better understand their experiences with the MetaStackVis tool and their overall impressions of it. They were also asked to provide any suggestions for improving the user experience of the tool.

Throughout the process, the experts were asked to verbalize their thought process and their responses were recorded to identify areas of difficulty and confusion when using

MetaStackVis. After the interviews were completed, the recorded responses were transcribed, analyzed, and organized to form a comprehensive analysis of their experiences, which is presented in Section 5.3.

5.3 Results

The main points we explicitly requested to receive feedback were the following three aspects: the overall workflow of the tool, the visualizations used, and the interaction capabilities, and the limitations as identified by the experts. The results from the real-time hands-on experience can be found below.

Workflow Overall, the experts generally had positive reviews of the tool. **E2**, **E3**, and **E4** confirmed that the overall proposed workflow for MetaStackVis makes sense and is reflected well in the spatial arrangement of the views. In particular, **E2** noted that the progressive analysis of various base model clusters and the generated metamodels was a strong point of MetaStackVis [4]. **E1** and **E2** proposed an alternate method for selecting clusters, focusing on the UMAP plot and allowing users to select clusters of interest using various base models. However, they acknowledged that using a user-controlled HDBSCAN clustering as a starting point before manual exploration would be practical. **E1** also mentioned that the MetaStackVis workflow requires a deep understanding of stacking ensembles and access to the publicly available StackGenVis source code [71] and noted that the tool may not provide additional insights into the data itself. We may plan to merge all features into a "single tool" solution, as it may be beneficial to feed the human knowledge generated by MetaStackVis back into StackGenVis, as previously suggested by **E4** [4]. **E1** proposed using a different approach for selecting clusters based on the UMAP plot, rather than the automatic clustering technique by HDBSCAN, which was suggested as a good starting point by the same expert.

Visualization and Interaction **E3** had a positive impression of the visual representations used to display the computed data. **E1** found MetaStackVis impressive due to the level of detail it provided, but acknowledged that it may be confusing for new users, as confirmed by **E2** as well [4]. **E1** suggested focusing on one metric at a time or specific ones based on the dataset, rather than having all 7 loaded at the same time, to help alleviate this issue. **E4** also suggested including more context and explanation for each plot and metric, especially for those that are more complex, to help users better understand the tool. Both **E2** and **E3** mentioned that the stacked bar chart is easy to interpret. **E3** suggested segmenting and vertically aligning each metric instead of globally sorting them, but acknowledged that this would affect the overall ranking [4]. This aligns with **E1**'s suggestion to focus on one or a group of validation metrics at a time rather than visualizing all seven simultaneously [4]. MetaStackVis provides the functionality to users to hide irrelevant metrics, but it would be beneficial to implement this feature in the future, similar to StackGenVis [4]. Specifically, **E2** noted the usefulness of the UMAP plot for understanding model combinations and suggested further developing the tool by picking models from each cluster and combining them with blending and bagging techniques for additional model optimization. **E3** was positively impressed by the projection of models based on predicted probabilities (as seen in the UMAP plots). **E3** noted that the visualization shows the successful identification of valuable information and patterns related to clustering and underlying cluster structure [4]. However, **E3** was confused by having both base models and metamodels on the same plot and suggested that it would be better

to focus solely on the metamodels. **E2** agreed on the ability of the UMAP plot to project the difference in predictive performance between base models and metamodels [4]. However, **E1** mentioned being confused by the UMAP plot, but understood the main concept of similarity. An improvement here could be to highlight the circle chart on the lower left-hand side and the corresponding gauge chart on the upper right hand side when user hovers over either one of them as it would make easier to perform comparison of the metamodels' pairs [4], as **E2** pointed out. Both **E2** and **E3** pointed that the zone-based matrix view can be challenging to understand and requires a certain amount of time to follow the concept [4]. **E1** also suggested presenting aggregated figures (like probability confidence or performance) with a sense of spread or distribution to help users better understand the data. **E3** pointed out the importance of the degree of model confidence in data exposition and agreed that the concept of sorting metamodels based on overall performance ranking eases the process of model evaluation in zone-based matrix. **E4** suggested improving the tool's interface by adding more interactive elements, such as hover-over text or filtering options. **E3** and **E4** have also supported our idea for future tool improvement by utilizing the best-performing pairs of metamodels as input data to a potential second layer of metamodels [4].

Limitations **E4** pointed out *efficiency* and *scalability* as some possible tool limitations. The first one is the amount of time needed to render all views through calculation. However, as long as everything is parallelized and pre-computed beforehand, this does not endanger interaction. [83]. In the latter instance, he emphasized that the tool's inability to visualize a much larger dataset with cases that are harder to forecast is a result of the zone-based matrix's increased space requirement. Filtering, which is applicable in situations where some metamodel pairings are doing poorly, might be a straightforward answer to this issue. By adding a filtering mechanism that allows for the removal of metamodel pairings that do not perform well, the space requirement can be reduced without compromising the accuracy or interactivity. As **E2** stated, the tool currently works only with *binary classification problems* and does not support algorithms' *hyperparameter tuning* [84] on the metamodel level [4]. **E1** referred to the important role that metamodels' confidence plays in the data exposition and was clear on missing variance indicator (confidence spread) for model confidence, but proposed the *individual visual representations of spread* instead of the currently implemented aggregated view [4]. To provide the user with an insight on how the confidence of each metamodel is changing, we can represent the spread of model confidence in a visual form, for example as a box plot or through similar alternatives. **E1** also mentioned that the data distribution could be presented on demand for better data exploration, which is related to constantly *misclassified data instances* [4]. Another limitation, as pointed out by **E4**, is related to data dependency on StackGenVis for input data and a general good understanding of stacked generalization concepts. That limitation can be resolved in a future overall solution by combining StackGenVis and MetaStackVis. **E4** has also pointed out the resolution limitation, as due to comprehensive visualization, the development team chose a 2K (2560 x 1440) resolution. Presenting MetaStackVis on a regular Full HD resolution (1920 x 1020) or lower resolution could lead to a lack of clarity and precision in data analysis, as well as an inability to spot interesting patterns.

6 Discussion

The MetaStackVis tool has identified several findings related to metamodel composition in stacked generalization, as well as provided some new opportunities for the development team related to further adaptation of the tool and integration into a more complete pipeline, which included data gathering, base model exploration, and metamodel tuning processes.

In its current state, MetaStackVis can be considered an interim solution, heavily depending on the data provided by the StackGenVis system [3], both in terms of input data and base model performance and probability information. However, even when used independently, it still offers a wide range of capabilities for comparing metamodels in stacking ensemble learning, including efficient clustering of metamodels using the HDBSCAN technique, visual representation of metamodel performance for each cluster of base models, visual representation of metamodel and base model prediction probabilities in two-dimensional UMAP space, and an extensive pair-wise metamodel comparison both in terms of "soft voting technique" and "theoretical maximum performance" of each pair.

Several research papers [3, 9, 18, 19, 20] have considered the use of a specific metamodel in stacked generalization. One could counter that applying a particular model won't always work with the data available or offer the best solution based on effectiveness and confidence level for a particular issue. According to our knowledge, there is no current research available that could be used for a proper pair-wise comparison of metamodels. That can be considered the main advantage of MetaStackVis over the existing VA tools, related to metamodel comparison in stacked generalization.

MetaStackVis obviously comes with its limitations, as the work was only considered an interim solution to further tool development related to the combination of StackGenVis and MetaStackVis. That was also in the nature of a Bachelor's project to focus on some cornerstone targets that have not been captured by the current version of StackGenVis, which we hope MetaStackVis has achieved. As a result of this project, a short research paper has been written and is currently out for review. Both the research paper and the MetaStackVis programming code on GitHub are publicly available, and we hope they can benefit the further improvement of the VA field in relation to the exploration of stacked generalization by the Compute Science and Data Science communities.

7 Conclusion

In this thesis, we presented MetaStackVis, a VA tool that provides users with visually assessment of metamodel performance in stacking ensemble learning [4]. MetaStackVis provides an efficient and intuitive way of representing the results of stacking ensemble learning models, making it possible for users to detect potential performance improvements when different metamodels are used in combination. The tool allows for a thorough examination of metamodels utilized in stacked generalization through the use of various ML and visualization techniques.

By using the HDBSCAN technique end user can successfully identify the optimal cluster composition of base models and identify the clusters based on the highest cluster density (DBCV) score and model coverage (the percentage of models successfully assigned to a specific cluster). Based on domain knowledge, one can proceed with various cluster combinations as the tool provides additional functionality for tuning HDBSCAN hyperparameters, or simply proceed with the default settings, achieving the highest possible scores.

The main visualization view provides an overview of the main metamodel performance metrics and the probability confidence metric. For ease of visualization, we only present the top-performing model per cluster and sort the clusters in descending order for ease of perception. This can provide the end user with an overview of the overall metamodel performance per cluster. The UMAP plot provides a different perspective on base models and metamodel probabilities, presented in the UMAP's two-dimensional space. Again, MetaStackVis allows the end user to explore different clusters of base models and metamodels trained on the specific cluster in a single view, as well as adapt the view by tuning the main UMAP hyperparameters, such as the number of neighbors and minimal distance.

The main metamodel pair-wise comparison chart provides overview of both single metamodel performance and probability confidence, as well as pair-wise metamodel comparison using different combinations of techniques such as "soft voting" for regular ensemble of models and the "theoretical maximum performance" showing the highest performance and probability confidence, that can be achieved by a specific combination of metamodels. We believe that giving the end user both perspectives can broaden the functional representation of model performance and provide additive views of model exploration.

As one of our objectives, we planned to further contribute to the previously developed StackGenVis system, allowing end users to have only a LR as a metamodel. The task was to investigate the performance of 11 different algorithms on the overall performance of stacked generalization. All metamodels are ranked in descending order by overall performance and probability confidence in our pair-wise model comparison chart, which allows for easy comparison of algorithms and satisfies the requirement set.

We have also tried to answer the question, "How does the performance of a single metamodel compare to that of a pair of metamodels?" This question has not been addressed by other research papers, to our knowledge, and therefore was still applicable. We could hypothesize that the pair of two metamodels perform at least equally well the best performing model in this pair or outperforms single models in terms of overall predictive performance and predictive probability confidence. That has also been confirmed by MetaStackVis, especially by reviewing the maximum theoretical *performance* of the pair of models as they are able to compensate each other's errors, leading to better overall predictive performance. We can therefore conclude that the pair of two metamodels is

better than a single model in terms of maximum *metric-based performance* and *predictive probability confidence*.

We also had the objective of verifying the performance of metamodels trained on a specific subset of base layer models compared to training metamodels using prediction data from all base layer models. Training the metamodel using predictions from all base models, resulting in the highest possible overall predictive performance and confidence, did not always have to be the correct assumption; according to our hypotheses, some base models can underperform in certain cases due to the original data structure and complexity. That hypothesis has been confirmed on some of our datasets, such as the Pima Indian Diabetes dataset [81] where we found that training metamodels on a subset of base models predictions provided better overall performance and predictive probability for the top performing metamodel in this cluster than training metamodels on predictive probabilities from all 55 base models. That provides an additional functionality as the end user can further update the production pipeline and reduce the overall computational cost and duration by only focusing on a subset of the original base models, in case metamodels trained on this subset outperform the metamodels trained on the predictions of all base models. MetaStackVis has also been assessed in terms of the applicability and efficacy by using an actual healthcare dataset and through online interviews with subject matter experts [4]. The results showed that the comparison of other metamodels with our tool showed promise. Finally, they assisted us in realizing the limitations of MetaStackVis, which can be addressed in the future, see Section 7.1.

7.1 Future work

The future work for tool improvement lies within the current limitations and possibilities for improvement, as identified while working on the tool and by subject matter experts during interview sessions. Efficiency and scalability were listed as some of the main limitations of the tool in its current stage.

From an efficiency perspective, the tool does not currently allow the end user to train metamodels on a subset of base models, but instead performs all necessary training for all clusters of base models. This clearly has an impact on overall computational time, particularly for datasets with multiple features and a large number of observations. Future revisions of MetaStackVis can reduce the impact of this limitation by focusing only on a specific cluster or subset of base models for further investigation. As a VA tool, MetaStackVis also performs a preliminary calculation of all possible scenarios for tuning the hyperparameters of HDBSCAN and UMAP to reduce the waiting time for the end user while reviewing the tool. That limitation is inherent in the nature of VA and currently cannot be eliminated except by skipping some of the hyperparameter tuning as excessive.

Scalability, on the other hand, can be further improved by code optimization. The current code is written in Python and uses the Streamlit [65] library for the back-end and front-end, as well as the Plotly [66] library for visual representation. Some integrated solutions, such as the JavaScript library D3.js [85] and React [86] for visualization, may aid in optimizing the run-time performance of a VA app. Another solution to improve the scalability of the application is to use a cloud-based platform, such as Google Cloud Platform [87] or Microsoft Azure [88], and deploy the application in a serverless architecture, which can scale the application automatically and dynamically according to the computational load.

As pointed out by some of the subject matter experts, MetaStackVis is currently developed for binary classification problems and does not allow any multi-class classification or regression tasks. That can be considered a future tool improvement if necessary.

Another observation is related to the fact that MetaStackVis does not support any additional hyperparameter tuning for metamodel algorithms and simply inherits the hyperparameters from the top performing model at the base level. That has not been considered a limitation because hyperparameter tuning has been extensively captured by the StackGenVis system and thus has not been considered as a strongly required functionality requirement for MetaStackVis, also due to computational power limitations when tuning a large number of hyperparameters for multiple metamodels. This lack of additional hyperparameter tuning allows user to more quickly obtain results by utilizing the top performing model's hyperparameters.

Finally, MetaStackVis, acting as a stand-alone tool, heavily depends on input data provided by StackGenVis. Combining these two tools can provide an end-to-end solution for data exploration, base model exploration, metamodeling, hyperparameter tuning, and focus on single metamodel and metamodel pair performance comparison, giving the user complete control over the process.

References

[1] J. Rocca. (2019, Apr. 23) Ensemble methods: Bagging, boosting, and stacking. Accessed January 4, 2023. [Online]. Available: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

[2] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.

[3] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren, "StackGenVis: Alignment of data, algorithms, and models for stacking ensemble learning using performance metrics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1547–1557, 2020.

[4] I. Ploshchik, A. Chatzimparmpas, and A. Kerren, "Metastackvis: Visually-assisted performance evaluation of metamodels," *arXiv preprint arXiv:2212.03539*, 2022.

[5] S. Dale. (2020, Nov. 13) An intuitive explanation of random forests. Accessed January 4, 2023. [Online]. Available: https://towardsdatascience.com/an-intuitive-explanation-of-random-forests-109b04bca343

[6] M. Gada, Z. Haria, A. Mankad, K. Damania, and S. Sankhe, "Super Learner: Stack generalization algorithm for AutoML," in *Proceedings of the 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE, 2021, pp. 1–6.

[7] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren, "Empirical study: Visual analytics for comparing stacking to blending ensemble learning," in *Proceedings of the 2021 23rd International Conference on Control Systems and Computer Science (CSCS)*. IEEE, 2021, pp. 1–8.

[8] J. Grus, "The bias-variance tradeoff in k-fold cross validation," in *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*, 2017.

[9] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *CoRR*, vol. abs/1105.5466, 2011. [Online]. Available: https://arxiv.org/abs/1105.5466

[10] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual analytics: Definition, process, and challenges," *Information Visualization*, pp. 154–175, 2008.

[11] D. Sacha, A. Stoffel, F. Stoffel, B. C. Kwon, G. Ellis, and D. A. Keim, "Knowledge generation model for visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1604–1613, 2014.

[12] S. Card, *Information visualization*. Lawrence Erlbaum Associates, Mahwah, NJ, 01 2008, pp. 509–543.

[13] I. T. Jolliffe, *Principal component analysis*. Springer, 2002.

[14] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[15] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[16] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2013, pp. 160–172.

[17] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," *CoRR*, vol. abs/1802.03426, 2018. [Online]. Available: https://arxiv.org/abs/1802.03426

[18] A. Seewald, "How to make stacking better and faster while also taking care of an unknown weakness," 01 2002, pp. 554–561.

[19] Y. Chen and M. L. Wong, "Optimizing stacking ensemble by an ant colony optimization approach," in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, 2011, pp. 7–8.

[20] P. Shunmugapriya and S. Kanmani, "Optimization of stacking ensemble configurations through artificial bee colony algorithm," *Swarm and Evolutionary Computation*, vol. 12, pp. 24–32, 2013.

[21] J. Krause, A. Perer, and E. Bertini, "Using visual analytics to interpret predictive machine learning models," *CoRR*, vol. abs/1606.05685, 2016. [Online]. Available: https://arxiv.org/abs/1606.05685

[22] M. P. Sesmero, A. I. Ledezma, and A. Sanchis, "Generating ensembles of heterogeneous classifiers using stacked generalization," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 1, pp. 21–34, 2015.

[23] D. Anguita, L. Ghelardoni, A. Ghio, L. Oneto, and S. Ridella, "The 'K' in K-fold cross validation," in *Proceedings of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. i6doc. com publ, 2012, pp. 441–446.

[24] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Machine Learning*, vol. 54, no. 3, pp. 255–273, 2004.

[25] S. Džeroski and B. Ženko, "Stacking with multi-response model trees," in *Proceedings of the International Workshop on Multiple Classifier Systems*. Springer, 2002, pp. 201–211.

[26] E. Menahem, L. Rokach, and Y. Elovici, "Troika – An improved stacking schema for classification tasks," *Information Sciences*, vol. 179, pp. 4097–4122, 2009.

[27] L. Todorovski and S. Džeroski, "Combining classifiers with meta decision trees," *Machine Learning*, vol. 50, pp. 223–249, 01 2003.

[28] D. Karaboga *et al.*, "An idea based on honey bee swarm for numerical optimization," Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Kayseri/Türkiye, Tech. Rep., 2005.

[29] S. Sun, "A review of deterministic approximate inference techniques for Bayesian machine learning," *Neural Computing and Applications*, vol. 23, no. 7, pp. 2039–2050, 2013.

[30] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[31] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[32] A. A. Agresti, "Probabilistic learning theory," *IEEE Transactions on Information Theory*, vol. 30, no. 6, pp. 858–866, 1984.

[33] H. Taud and J. Mas, "Multilayer perceptron (MLP)," in *Geomatic Approaches for Modeling Land Change Scenarios*. Springer, 2018, pp. 451–455.

[34] M. P. LaValley, "Logistic regression," *Circulation*, vol. 117, no. 18, pp. 2395–2399, 2008.

[35] R. H. Riffenburgh, "Linear discriminant analysis," Ph.D. dissertation, Virginia Polytechnic Institute, 1957.

[36] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[37] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[38] R. Rojas *et al.*, "AdaBoost and the super bowl of classifiers a tutorial introduction to adaptive boosting," *Freie University, Berlin, Tech. Rep*, 2009.

[39] J. Friedman, "Greedy function approximation: A gradient boosting machine https://www. salford-systems. com/doc," *GreedyFuncApproxSS. pdf*, 1999.

[40] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *CoRR*, vol. abs/2003.05689, 2020. [Online]. Available: https://arxiv.org/abs/2003.05689

[41] A. Bissuel. (2019, Apr. 16) Ensemble methods: Bagging, boosting, and stacking. Accessed January 4, 2023. [Online]. Available: https://medium.com/criteo-engineering/hyper-parameter-optimization-algorithms-2fe447525903

[42] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[43] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[44] A. Mohapatra, "Population-based algorithms for hyperparameter optimization in reinforcement learning," Apr 2022, accessed January 4, 2023. [Online]. Available: https://medium.com/mlearning-ai/population-based-algorithms-for-hyperparameter-optimization-in-reinforcement-learning-b04ce2165533

[45] J. Nabi, "Hyper-parameter tuning techniques in deep learning," Mar 2019, accessed January 4, 2023. [Online]. Available: https://towardsdatascience.com/hyper-parameter-tuning-techniques-in-deep-learning-4dad592c63c8

[46] A. Li, O. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley, and P. Gupta, "A generalized framework for population based training," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1791–1799.

[47] J. Parker-Holder and A. Kamsetty, "Population based bandits: Provably efficient online hyperparameter optimization," Nov 2020, accessed January 4, 2023. [Online]. Available: https://www.anyscale.com/blog/population-based-bandits

[48] J. Parker-Holder, V. Nguyen, and S. J. Roberts, "Provably efficient online hyperparameter optimization with population-based bandits," *Advances in Neural Information Processing Systems*, vol. 33, pp. 17 200–17 211, 2020.

[49] Y. Liu, W. Chen, P. Arendt, and H.-Z. Huang, "Toward a better understanding of model validation metrics," *Journal of Mechanical Design*, vol. 133, no. 7, 2011.

[50] N. W. S. Wardhani, M. Y. Rochayani, A. Iriany, A. D. Sulistyono, and P. Lestantyo, "Cross-validation metrics for evaluating classification performance on imbalanced data," in *Proceedings of the 2019 International Conference on Computer, Control, Informatics, and its Applications (IC3INA)*, 2019, pp. 14–18.

[51] J. Brownlee. (2020, Feb. 24) A gentle introduction to the Fbeta-measure for machine learning. Accessed January 4, 2023. [Online]. Available: https://machinelearningmastery.com/fbeta-measure-for-machine-learning

[52] J. Akosa, "Predictive accuracy: A misleading performance measure for highly imbalanced data," in *Proceedings of the SAS global forum*, vol. 12, 2017, pp. 1–4.

[53] J. Muschelli, "ROC and AUC with a binary predictor: A potentially misleading metric," *Journal of Classification*, vol. 37, no. 3, pp. 696–708, 2020.

[54] G. Dembla. (2020, Nov. 17) Intuition behind log-loss score. Accessed January 4, 2023. [Online]. Available: https://towardsdatascience.com/intuition-behind-log-loss-score-4e0c9979680a

[55] D. Cashman, S. Humayoun, F. Heimerl, K. Park, S. Das, J. Thompson, B. Saket, A. Mosca, J. Stasko, A. Endert, M. Gleicher, and R. Chang, "A user-based visual analytics workflow for exploratory model analysis," *Computer Graphics Forum*, vol. 38, pp. 185–199, 06 2019.

[56] J. W. Tukey, "Exploratory data analysis," *Reading Mass*, 1977.

[57] S. Das, D. Cashman, R. Chang, and A. Endert, "Beames: Interactive multimodel steering, selection, and inspection for regression tasks," *IEEE Computer Graphics and Applications*, vol. 39, no. 5, pp. 20–32, 2019.

[58] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan, "EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1283–1292.

[59] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert, "Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 364–373, 2018.

[60] D. Ren, S. Amershi, B. Lee, J. Suh, and J. Williams, "Squares: Supporting interactive performance analysis for multiclass classifiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, pp. 61–70, 2017.

[61] S. Das, S. Xu, M. Gleicher, R. Chang, and A. Endert, "Questo: Interactive construction of objective functions for classification tasks," in *Proceedings of the Computer Graphics Forum*, vol. 39, no. 3. Wiley Online Library, 2020, pp. 153–165.

[62] K. Xu, M. Xia, X. Mu, Y. Wang, and N. Cao, "EnsembleLens: Ensemble-based visual exploration of anomaly detection algorithms with multidimensional data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 109–119, 2018.

[63] S. Liu, J. Xiao, J. Liu, X. Wang, J. Wu, and J. Zhu, "Visual diagnosis of tree boosting methods," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 163–173, 2017.

[64] A. S. Saabith, T. Vinothraj, and M. Fareez, "Popular python libraries and their application domains," *Development*, vol. 7, no. 11, 2020.

[65] "Streamlit — The fastest way to build and share data apps," https://streamlit.io, 2020, accessed January 4, 2023.

[66] "Plotly — Python open source graphing library," https://plotly.com/python/, 2013, accessed January 4, 2023.

[67] "Scikit-learn — Machine learning in Python," https://scikit-learn.org/stable/, accessed January 4, 2023.

[68] "Imblearn — Dealing with classification with imbalanced classes in Python," https://imbalanced-learn.org/stable/, accessed January 4, 2023.

[69] "Pandas — Data analysis and manipulation tool," https://pandas.pydata.org/, accessed January 4, 2023.

[70] "Numpy — Scientific computing with Python," https://numpy.org, accessed January 4, 2023.

[71] "StackGenVis code," https://bit.ly/StackGenVis-code, 2021, accessed January 4, 2023.

[72] D. Dua and C. Graff, "UCI machine learning repository," 2017, accessed January 4, 2023. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

[73] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, and J. Sander, "Density-based clustering validation," in *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 2014, pp. 839–847.

[74] L. McInnes, J. Healy, and S. Astels, "HDBSCAN: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, no. 11, mar 2017. [Online]. Available: https://doi.org/10.21105/joss.00205

[75] J. Irani, N. Pise, and M. Phatak, "Clustering techniques and the similarity measures used in clustering: A survey," *International Journal of Computer Applications*, vol. 134, no. 7, pp. 9–14, 2016.

[76] C. Frenzel, "Tuning with HDBSCAN," https://towardsdatascience.com/tuning-with-hdbscan-149865ac2970, 9 2020, accessed January 4, 2023.

[77] M. Shahhosseini, G. Hu, and H. Pham, "Optimizing ensemble weights and hyperparameters of machine learning models for regression problems," *Machine Learning with Applications*, vol. 7, p. 100251, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827022000020

[78] Y. Liu and J. Heer, "Somewhere over the rainbow: An empirical assessment of quantitative colormaps," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, pp. 598:1–598:12.

[79] D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield, "Scatterplot matrix techniques for large N," *Journal of the American Statistical Association*, vol. 82, no. 398, pp. 424–436, 1987. [Online]. Available: https://www.jstor.org/stable/2289444

[80] J. Cai, J. L. Garner, and R. A. Walkling, "A paper tiger? An empirical analysis of majority voting," *Journal of Corporate Finance*, vol. 21, pp. 119–135, 2013.

[81] J. Smith, J. Everhart, W. Dickson, W. Knowler, and R. Johannes, "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus," in *Proceedings of the Annual Symposium Computer Application in Medical Care*. American Medical Informatics Association, 1988, pp. 261–265.

[82] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *Proceedings of the International Conference on Database Theory (ICDT)*. Springer Berlin Heidelberg, 2001, pp. 420–434.

[83] J. K. Li and K.-L. Ma, "P4: Portable parallel processing pipelines for interactive information visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 3, pp. 1548–1561, 2020.

[84] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing, 2019, pp. 3–33.

[85] "D3.js — Data-driven documents," https://d3js.org/, accessed January 4, 2023.

[86] "React.js — A JavaScript library for building user interfaces," https://reactjs.org/, accessed January 4, 2023.

[87] "Google Cloud — Build apps faster, make smarter business decisions, and connect people anywhere," https://cloud.google.com/, accessed January 4, 2023.

[88] "Microsoft Azure — Do more with less - On Azure," https://azure.microsoft.com/en-us/, accessed January 4, 2023.

A Project Environment Setup

Table 1.5: Package name and corresponding package version. This table is to ensure that forked code is using a specific version of a package that is known to be compatible with MetaStackVis. Package versions are also available in *requirements.txt* file at `https://github.com/ilyaploshchik/MetaStackVis`.

Package Name	Package Version
streamlit	1.11.0
pandas	1.4.3
scikit-learn	1.1.1
numpy	1.22.4
pillow	9.2.0
hdbscan	0.8.28
plotly	5.9.0
imbalanced-learn	0.9.1
umap-learn	0.5.3