# IterSHAP: an XAI feature selection method for small high-dimensional datasets

Master Thesis, Date: September 08, 2023

Frank van Mourik
University of Twente
The Netherlands
f.g.vanmourik@student.utwente.nl

## ABSTRACT

Small high-dimensional datasets pose challenges for achieving accurate predictive models, due to issues like overfitting and the curse of dimensionality. While complex models, like deep learning models, have been used to address these challenges, they often lack interpretability and transparency. Explainable Artificial Intelligence (XAI) is a popular field that aims to bridge this gap by developing techniques that provide insights into the decision-making process of machine learning models, therefore increasing the explainability and trustworthiness of models. However, when applying feature selection before model training, less-complex models can be used, such that interpretability and explainability are preserved, such that Explainable AI methods are not even needed. This research presents an iterative feature selection method named *IterSHAP*, which utilizes a popular XAI technique named SHAP, to increase model performance on small high-dimensional datasets. The performance of *IterSHAP* was evaluated via both a simulation-based approach and an application-based approach. The results demonstrate the effectiveness of *IterSHAP* in selecting informative features and improving classification performance on small high-dimensional datasets. The limitations of *IterSHAP* are its convergence to a local optimum when working on large datasets and its lack of computational optimization.

## 1 INTRODUCTION

Small high-dimensional datasets present unique challenges for achieving high accuracy in predictive models, such as over- and underfitting and the *curse of dimensionality* [2]. Overcoming these issues while preserving the interpretability and transparency of the underlying models remains a challenge for researchers.

To tackle these challenges, a growing trend has emerged in using complex models, such as deep learning models. However, as a consequence, the explainability and interpretability of these models are minimised, resulting in 'black-box decision making'. The rise of Explainable Artificial Intelligence (XAI) aims to bridge this gap by developing techniques to increase insights in *how* a machine learning model came to its prediction [17]. Research into XAI applications on small high-dimensional datasets, however, leaves a gap in research.

Among the many, one often utilised XAI technique is SHAP [15], short for SHapley Additive exPlanations. SHAP uses Shapley values [21] [7], a popular game theory concept, to quantify the contribution of an individual feature to the outcome of the model. This technique has proven to significantly increase the insights into the relation between the input data and the output prediction of a model. An interesting research field would be to investigate whether we can use SHAP to reduce the dimensionality of a small dataset, after which we can use a traditional, intrinsic-explainable, machine learning model for classification while maintaining model accuracy.

Previous studies have explored the usage of SHAP for feature selection, although their focus has primarily been on larger synthesized or benchmark datasets.

The remainder of this paper is structured as follows. In section 2, we will describe the research methodology used and introduce the goals and research questions of this research. In section 3, we iterate over the state-of-the-art feature selection methods and Explainable AI methods. In sections 4 and 5, we present our artifact, including its validation and evaluation. In sections 7 and 8, we will conclude our work and highlight limitations and pointers for future.

## 2 RESEARCH METHODOLOGY

This research was conducted using the Design Science Research Methodology proposed by Wieringa [29]. Figure 2 shows the Engineering Cycle from Wieringa, from which three steps are used in this research: Problem investigation/Implementation evaluation, Treatment design, and Treatment validation. By leaving out the Treatment implementation step, this research can be seen as a design cycle research [29].

First, a list of stakeholders and their goals was created, as can be seen in table 1. These goals are combined in one research goal, formulated using the design template of Wieringa [29]: **Improve** *model performance on small high-dimensional datasets* **by treating it** *with a SHAP-based feature selection method* **that satisfies** *increased model performance and model explainability* **in order to** *reduce the need for complex models.*

This research goal can be translated into the following research questions:

(1) RQ 1: How can we develop a feature selection method that uses SHAP to improve classification on small high-dimensional datasets?
(2) RQ 2: How can we improve model performance on a real-world problem to show the utility of SHAP feature selection on small high-dimensional datasets?

Figure 1 shows how sections 1 to 5 are connected to the principles from Wieringa's design cycle. Before this research, a tertiary review was conducted on Explainable AI methods, which only serves as an introduction to this research, but is not part of this research. The treatment design and treatment evaluation of this research are

| Stakeholder | Stakeholder Taxonomy [1] | Goals |
|---|---|---|
| Data Scientist and practitioners | Normal operators | Benefit from the developed feature selection method to achieve higher accuracies on smaller datasets |
| University of Twente | Knowledge Supplier | Contribute to the scientific field of the research |
| Slimstock | Sponsor | Support and advise the author with their research |
| Author | Researcher, Developer | Design and develop a feature selection method that satisfies the requirements |

**Table 1: List of Stakeholders, based on Alexander's Stakeholder Taxonomy [1]**

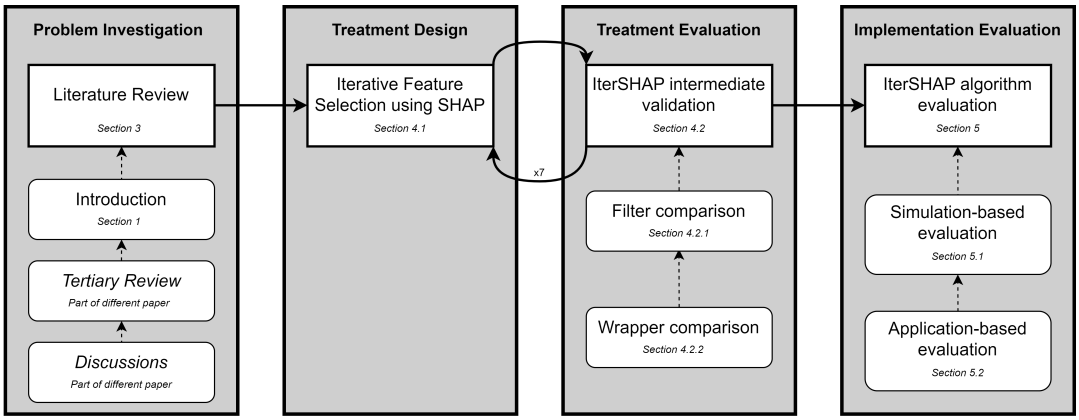



**Figure 1: The research process used during this research**

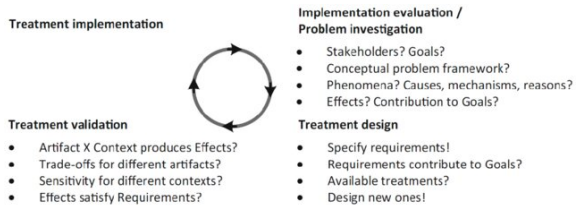



**Figure 2: The Engineering cycle, presented by Wieringa [29].**

intertwined, as *IterSHAP* was iteratively developed, which is indicated by the circular arrows in figure 1. Finally, the implementation evaluation in section 5 evaluated *IterSHAP* after development.

## 3 RELATED WORK

In this section, we will describe the relevant state-of-the-art on Explainable Artificial Intelligence and Feature Selection, as these are the two core components of our research.

### 3.1 Explainable Artificial Intelligence (XAI)

The number of published articles on Explainable AI (XAI) has been skyrocketing for five years, see figure 3. Vast amounts of these articles introduce new XAI methods, whereas others apply or benchmark said methods. One of the often-utilised XAI techniques is SHAP [15], short for SHapley Additive exPlanations.

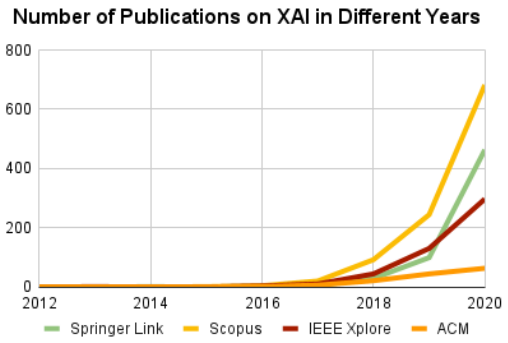



**Figure 3: Number of published articles has exploded over the past five years. Source: zoom of figure 1 of [8].**

*3.1.1 SHapley Additive exPlanations (SHAP).* SHapley Additive exPlanations (SHAP) was introduced in 2017 by Lundberg and Lee [15]. The SHAP method is based on the Shapley values, originating from the game theory introduced by L.S. Shapley [21]. SHAP aims to explain individual predictions of black-box models. However, it is also possible to retrieve global explanations by aggregating these individual predictions.

*Shapley values.* SHAP uses the formula described in equation 1, which calculates the Shapley value of feature $i$ in black-box model $f$ and input vector $x$. For instance, when predicting the mortality rate of a Covid-19 intensive-care patient[3], $i$ could be the age of the patient. The input vector $x$ is all input features combined, such as age, body mass index, physical condition, diabetes, and more.

$$\phi_i(f,x) = \sum_{z' \subseteq x'} \frac{|z'|!(M-|z'|-1)!}{M!} \left(f_x(z') - f(x(z' \setminus i)\right) \quad (1)$$

Equation 1 iterates over all subsets of the simplified data input $x'$. Simplified data input is mostly used for complex models, where the input data would otherwise become too large, for instance when analysing images. Then, the outcome of model $f$ using only that subset is calculated with our feature $i$ ($f_x(z')$) and without our feature $i$ ($f(x(z' \setminus i)]$). By subtracting those outcomes, we calculate the influence that feature $i$ has on the model's outcome.

Finally, this score is weighted according to how many features (of the total number of features $M$) were included in that subset. The intuition is that adding the feature $i$ should be weighted more if there are already many features included in the subset.

The benefit of using Shapley-based feature selection methods, is that feature dependencies, such as feature redundancy, interaction, complementarity, and substitution, are intrinsically taken into account.

*Model behaviour.* When implementing the above-mentioned mathematical equation on a machine learning model, one cannot simply leave out features, as that would change the dimensions of the model. SHAP tackles this issue by not excluding features, but by using random inputs from the training data set. The idea behind this is that by using random inputs for all subsets, the relevance of these features is sampled out by random variation, nullifying the predictive power of these features.

## 3.2 Feature Selection methods

Feature selection methods are used to select a subset of the input features as input for a model. These methods solve several challenges regarding the *curse of dimensionality* [2], interpretability of the model, and training computation time. Most model-agnostic feature selection methods can be divided into the categories: filter and wrapper methods. The third category, embedded methods, includes methods that are built into an algorithm, which we will leave out for this research, as we aim for a model-agnostic approach.

Figure 4 shows an overview of three categories.

*3.2.1 Filter methods.* Filter methods select a feature subset independent of the model used. They do this by e.g. calculating the relevance between input features and the target feature, after which the most-relevant features are selected. The main advantages of filter methods are the low computation time and robustness towards overfitting [20]. On the contrary, the main disadvantage is that usually feature dependencies, including redundancy and repetition, are not filtered out.

*Chi-square.* This feature selection filter method uses the test chi-squared statistic, see formula 2, to select $k$ features from the input

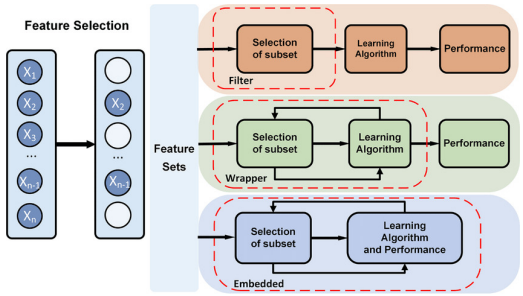



**Figure 4: The three feature selection categories [30]**

vector **x** [9]. This method only works with non-negative data and is typically used for categorical features.

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} \quad (2)$$

where $O_i$ is the observed value and $E_i$ is the expected value.

*Pearson's correlation.* The Pearson's correlation feature selection filter method selects features based on their correlation coefficient with the target feature, using formula 3 [4], where $x_i$ and $y_i$ are individual samples and $\overline{x}$ and $\overline{y}$ are the (sample) means of $X$ and $Y$ respectively. It selects the features with the highest correlation to the target feature.

$$r = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}} \quad (3)$$

*3.2.2 Wrapper methods.* Wrapper methods use some measurement, mostly statistical, to select the most relevant features. Examples of these are information gain, similarities, and distance measures. As these calculations are not performed within the training or evaluation of a model, these measures are mostly model-agnostic, meaning that they can be applied to any underlying model. Some examples of wrapper methods are Recursive Feature elimination [6], Backward Elimination [23], and Sequential Forward Selection [16].

*3.2.3 Feature selection using SHAP.* Furthermore, to the best of our knowledge, three wrapper methods using SHAP have been introduced.

*Shapicant.* The Shapicant algorithm[1] uses a permutation-based approach to select important features. It does this by training two models, one on the original training dataset and one on a shuffled dataset. The training on the shuffled dataset is executed multiple times, after which the Shapley value of the features is calculated. When a feature has significantly higher importance for the non-shuffled model than for the shuffled model, that feature is considered important.

*Borutashap.* Borutashap[2] uses Shapley values and the Boruta algorithm to determine which features are important. It does this by adding so-called *shadow features*, which are randomly shuffled

[1] https://github.com/manuel-calzolari/shapicant
[2] https://github.com/Ekeany/Boruta-Shap

features. It then hypothesised that an informative feature should be more important than the most-important shadow feature, hence only these features are selected. This method is statistically substantiated by running multiple iterations, resulting in a binomial distribution and a p-value cutt-off point [13].

*Powershap.* In 2022, Verhaeghe et al. [25] introduced Powershap: a 'power-full Shapley features selection method'. In the first component of the Powershap algorithms, the 'Explain' component, a single known random uniform feature $r$ is added to the feature set for training. The hypothesis is that all informative features (the feature that one wants to select) should on average have a higher Shapley value than the random feature $r$. Using a one-sample one-tailed student-t statistic test, it is tested for each feature whether it has a p-value lower than threshold $\alpha$, which is 0.01 by default. The features that pass this test are selected.

As Powershap outperforms both Borutashap and Shapicant [25] and has the same underlying statistical approach, Powershap will be the SHAP feature selection method we compare our research to.

## 4 ITERSHAP

*IterSHAP* is, as the name suggests, an iterative feature selection algorithm based on Shapley values. *IterSHAP* trains and evaluates a model at each step in the process, after which it selects the most-important features based on their Shapley value. This process is repeated until it has reached its lower limit, which concludes one iteration. In total, this is repeated a predefined maximum number of iterations.

In section 4.1, we will describe the architecture of the algorithm, including the considerations taken into account during the development. This can be seen as the 'treatment design' of Wieringa [29] (figure 1). In section 4.2, we will describe the intermediate validation of *IterSHAP* during the development, which is the 'treatment validation' of Wieringa [29].

### 4.1 Algorithm description

The *IterSHAP* method has three main configurable parameters, *model*, *max_iter* and *step_size*. The first parameter determines the model that is used for training, computing SHAP values, and evaluation. The default model is a RandomForestClassifier, as this is an often used classifier with decent default parameters **[cite]** RFC is good to use. The second parameter determines the maximum number of iterations that are executed. The final parameter defines the portion of features to keep at each sub-iteration of the algorithm. Algorithm 1 describes the working of *IterSHAP*. Figure 5 gives a visual overview of the working of IterSHAP.

We keep track of the model performances via the *log* dictionary. Afterwards, we iteratively train and evaluate a model on data $X$. After each training iteration, we retrieve the Shapley values of the features on an unseen data split, $X_{shap}$, and only keep the top $step\_size$ (default 0.50) of the important features. We end the iteration when the lower limit is reached.

After this iteration, we have a set of features that performed the best so far, which we call $S_{prime}$. We retrieve the smallest superset of $S_{prime}$ from the *log* (in figure 5a, this would be the subset with 50 features). We 'restore' $X$ to have only these features and define the search space for the next iteration. The upper limit of the search space is set to the size of the above-mentioned superset. The lower limit is set to the size of the largest subset of $S_{prime}$ in the *log* (12 features in the example). After the maximum allowed iterations have been executed or the search space is exhausted, the best-performing feature subset is returned. The search space is considered exhausted when the search space is smaller than three, as all possible configurations in the search space have already been executed. In the example figure 5c, the search space would become $17 - 15 = 2$, hence the search space is exhausted.

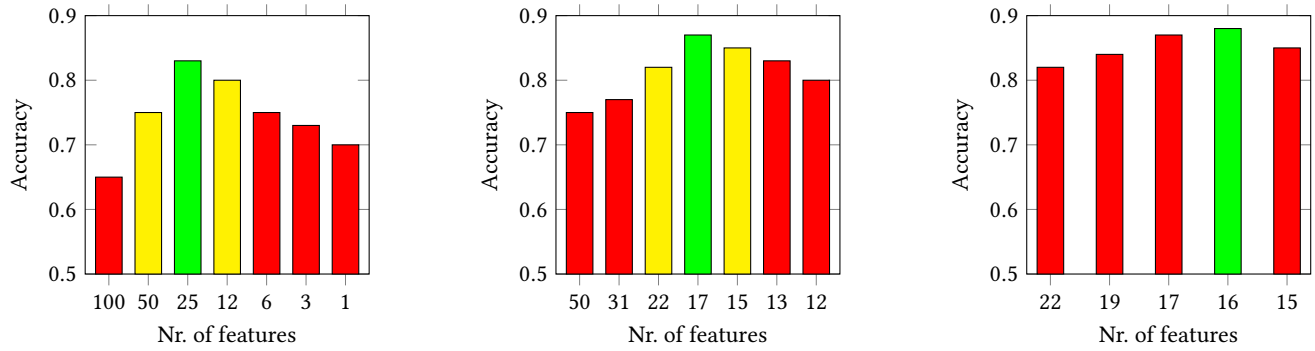
**Algorithm 1** IterSHAP algorithm

**function** IterSHAP($model, max\_iter, step\_size$)
  $X, y \leftarrow get\_data()$
  $log \leftarrow \{\}$ ▷ Tracks feature subsets and its performances
  $upper\_limit \leftarrow X.shape[1]$
  $lower\_limit \leftarrow 0$
  **for** $max\_iter$ **do**
    **while** $|X.features| > lower\_limit$ **do**
      $model.fit(X_{train}, y_{train})$
      $acc \leftarrow eval(model, X_{val}, y_{val})$
      $log.append(X.features, acc)$
      $important\_features \leftarrow getSHAP(model, X_{shap})$
      $X \leftarrow SelectBestFeatures(X)$ ▷ Select features based on step size
    **end while**
    $S_{prime} \leftarrow log(Max(acc))$ ▷ Get the set with the highest model accuracy
    $X \leftarrow X[S_{prime} + 1]$ ▷ Restore $X$ to the first superset
    $upper\_limit, lower\_limit \leftarrow get\_limits()$
  **end for**
  **return** $log(max(accuracy))$ ▷ Return the feature subset with the highest accuracy
**end function**

The three SHAP feature selection methods mentioned in section 3.2.3 are all based on shuffling/randomising features, after which the differences between training on the *real* features and the *shuffled* features are compared to select the most important features. *IterSHAP* does not shuffle or randomise the dataset at all, as it tries to iteratively select a subset of features until it finds the optimal subset. At each iteration, it selects the subset as a whole, not on an individual feature basis, which reduces the statistical need for a large dataset. Using this iterative approach always returns a subset of features that it considers most important, even on small datasets.

*4.1.1 Considerations.* In this section, we will describe the considerations made during the development of *IterSHAP*.

*Feature selection and elimination.* As numerous techniques for feature selection exist in literature [10], several options were considered. First, a division in forward and backwards feature selection was made. Forward feature selection is an iterative process of starting with an empty set of features and adding new features based on their impact on the model [26]. This is repeated until the predefined requirements are met. This process of selecting features, however, contradicts the core of SHAP, which evaluates the impact of individual features in a large set of features. Therefore, backwards feature elimination was chosen as the feature selection technique.




**(a) Iteration 1** **(b) Iteration 2** **(c) Iteration 3**

**Figure 5: Example of IterSHAP executing three iterations to find the optimal subset of 16 features**

| | Total features: 100 | | Total features: 250 | |
|---|---|---|---|---|
| | Inf. feat.: 3 | Inf. feat.: 8 | Inf. feat.: 3 | Inf. feat.: 8 |
| RFE 25% | 0.76 (1.37 s) | 0.78 (1.64 s) | 0.80 (1.43 s) | 0.66 (1.54 s) |
| RFE 35% | 0.74 (1.19 s) | 0.78 (2.18 s) | 0.80 (1.23 s) | 0.67 (1.25 s) |
| RFE 50% | 0.74 (0.99 s) | 0.78 (1.03 s) | 0.80 (1.02 s) | 0.68 (1.04 s) |
| RFE 60% | 0.73 (0.87 s) | 0.78 (0.90 s) | 0.80 (0.90 s) | 0.67 (0.91 s) |
| RFE 2 | 0.72 (1.23 s) | 0.78 (1.21 s) | 0.78 (1.26 s) | 0.66 (1.24 s) |
| RFE 5 | 0.72 (1.32 s) | 0.78 (1.22 s) | 0.80 (1.29 s) | 0.67 (1.20 s) |
| RFE 10 | 0.72 (1.38 s) | 0.78 (1.22 s) | 0.78 (1.25 s) | 0.68 (1.25 s) |
| RFE 15 | 0.73 (1.38 s) | 0.77 (1.19 s) | 0.81 (1.22 s) | 0.66 (1.18 s) |

**Table 2: Comparing Recursive Feature Elimination configurations on model accuracy and computation time, while varying the total number of (informative) features.**

Future work could be conducted on embedding SHAP in a forward feature selection method.

*4.1.2 Grouped feature elimination.* In traditional feature elimination, like Recursive Feature Elimination (RFE) [6], the least relevant feature is removed in each iteration. However, when dealing with high-dimensional datasets, retraining a model after each removed feature is computationally expensive. Furthermore, the complexity of SHAP scales factorial, as it calculates the contribution of a feature in each possible subset of features. Therefore, we investigated how grouped feature elimination would work in terms of accuracy and computation time.

Multiple grouped feature elimination configurations of RFE were tested and compared. These configurations are removing 25%, 35%, 50% or 60% of all features in each iteration. Furthermore, removing 2, 5, 10, or 15 features at each iteration were tested configurations. The results of these configurations can be found in table 2.

From these intermediate results, we concluded that there is no significant difference in accuracy between the configurations. However, a decrease in computation time of at least 30% was achieved with the 50% and 60% elimination configurations. As the 50% elimination configuration balances the accuracy and computation time the best, we used this configuration for the second part of the validation, as described in section 4.2.2.

## 4.2 Validation

While developing *IterSHAP*, we validated our work by comparing the results of *IterSHAP* with other feature selection methods in several taxonomies. In this section, we will chronologically describe these validations.

*4.2.1 Filter comparison.* As an initial validation of the working of *IterSHAP*, we compared the implementation with several state-of-the-art filter methods, as these are popular feature selection methods. These filter methods are Chi2 [9], Pearson's correlation [4], and two variations of the maximum relevance minimum redundancy (mRMR) algorithm [19] [27]. As these filter methods require a target number of features to retrieve, we set this number $K$ to equal the number of informative features in the dataset. Using this setting, the filter methods have an advantage, as they 'know' how many informative features there are. The methods were compared on two different sample sizes (100 and 1000 samples), two different feature sizes (100 and 250), and two different numbers of informative features (3 and 8). Each experiment was run five times, after which the results were averaged.

The results can be found in table 3. The first number in each cell represents the number of informative features retrieved, whereas the number between parentheses represents the number of noisy features retrieved. For all filter methods, adding these two numbers equals the total number of informative features, as explained earlier. For *IterSHAP*, however, this is not the case, as *IterSHAP* has no prior 'knowledge' on how many features to extract.

In each row, the method that extracted the highest number of informative features is highlighted. As can be seen, *IterSHAP* performs the best in all configurations. However, *IterSHAP* often tends to overselect, as indicated by the high number of noisy features returned. This is a direct consequence of the lack of 'knowledge' of *IterSHAP* in terms of the number of features to select.

As can be seen in table 3, all filter methods perform significantly better on the large sample set compared to the small sample set. Even though *IterSHAP* also performs better on the large sample set, this is less significant. This is an initial confirmation of our hypothesis that *IterSHAP* works better on small high-dimensional datasets.

| Configuration | | | Feature selection method | | | | |
|---|---|---|---|---|---|---|---|
| **Nr. samples** | **Nr F.** | **Nr. inf. F** | **IterSHAP** | **Chi2** | **Pearsson** | **mRMR once** | **mRMR iterative** |
| 100 | 100 | 3 | **2.0** (3.4) | 1.4 (1.6) | 2.0 (1.0) | 1.2 (1.8) | 1.0 (2.0) |
| | | 8 | **5.4** (8.8) | 3.8 (4.2) | 2.8 (5.2) | 1.6 (6.4) | 1.4 (6.6) |
| | 250 | 3 | **1.6** (2.4) | 1.2 (1.8) | **1.6** (1,4) | 1.2 (1.8) | 1.2 (1.8) |
| | | 8 | **5.2** (2.4) | 2.0 (6.0) | 3.4 (4.6) | 1.8 (6.2) | 1.8 (6.2) |
| 1000 | 100 | 3 | **3.0** (2.0) | 2.2 (0.8) | 1.8 (1.2) | 1.2 (1.8) | 2.0 (1.0) |
| | | 8 | **7.2** (1.6) | 5.4 (2.6) | 6.2 (1.8) | 5.0 (3.0) | 4.6 (3.4) |
| | 250 | 3 | **3.0** (6.6) | 2.2 (0.8) | 1.6 (1.4) | 1.2 (1.8) | 2.2 (0.8) |
| | | 8 | **8.0** (0.6) | 5.4 (2.6) | 5.8 (2.2) | 4.4 (3.6) | 4.4 (3.6) |

**Table 3: Comparison of retrieved informative and noisy features of *IterSHAP* with several filter feature selection methods.**

| | Nr. informative features | | | | | | |
|---|---|---|---|---|---|---|---|
| | 3 | 5 | 8 | 12 | 15 | 20 | 28 |
| RFE | 0.743 | 0.665 | 0.783 | 0.660 | 0.590 | 0.665 | 0.618 |
| *IterSHAP* | 0.825 | 0.829 | 0.803 | 0.733 | 0.644 | 0.723 | 0.658 |
| Student's T-test | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

**Table 4: Mean model accuracy with Student's t-test on *IterSHAP* and *RFE* series, varying total number of informative features. Total number of features = 100**

*4.2.2 Taxonomy comparison.* As a second validation of *IterSHAP*, we compared it with a feature selection method in the same taxonomy, that is, a wrapper feature selection method with grouped elimination. This method is RFE 0.50, the Recursive Feature Elimination configuration that eliminates the bottom 50% of all features each iteration, as discussed in section 4.1.2. We compared the two methods in seven experiments. Each dataset included 100 samples with 100 features but varied the number of important features. These options are 3, 5, 8, 12, 15, 20, and 28 important features. All experiments were executed 10 times, after which their accuracies were compared using the Student's T-test with a p-value of 0.05 [24]. As can be seen in table 4, all configurations achieved a p-value significantly smaller than 0.05, indicating that we reject the null hypothesis that these sets originate from the same object. Therefore, we can conclude that *IterSHAP* works significantly better than Random Feature Elimination on this small high-dimensional dataset.

## 5 EVALUATION

This section serves as the 'implementation evaluation', as described by Wieringa [29]. We will present the frameworks used to evaluate the performance of *IterSHAP*. These frameworks are split into a simulation-based evaluation and an application-based evaluation.

### 5.1 Simulation-based evaluation

To compare the performance of *IterSHAP* with other SHAP feature selection methods, we use a similar simulation-based evaluation setup as Verhaeghe et al. [25]. At the start of each experiment, we generate a dataset, using sklearn's 'make_classification()' method [3], with 5000 samples and a variable percentage of informative features (10%, 33%, 50%, and 90%). All other features are noisy features. Furthermore, we vary the total number of features between 20, 100, 250, and 500. This creates 16 unique experiments.

The datasets are split into four subsets: 'train' (50%), 'val' (15%), 'shap' (15%), and 'test' (20%). The 'train' set is used to train the models used in the feature selection methods. The 'val' set is used to evaluate the trained model and calculate its performance. Afterwards, the 'shap' is used to compute the Shapley values for each feature. Finally, after the feature selection, we use the 'test' dataset to see what influence the feature selection methods have on the performance of the model.

As we are not only interested in comparing the SHAP FS methods on a large dataset (5000 samples) but especially on a small dataset, we also select 160 random samples (train: 100, val: 30, shap: 30) from the total dataset which are used as a separate experimental setup. The remaining samples are used for post-feature selection evaluation. This gives us a total of 32 experiments on which we perform both *IterSHAP* and *PowerSHAP*.

Finally, we run these experiments with three underlying models. These models are 'CatBoostClassifier', 'RandomForestClassifier', and 'RidgeClassifier'. The first two are tree-based models, which do have embedded feature importance. 'RidgeClassifier' is a linear classification model that does not have feature importance inherently. The 'CatBoostClassifier' has similar hyperparameters as used by Verhaeghe et al. providing the ideal model to compare the two methods. For the other two models, no hyperparameter optimization has been applied. Each experimental setup is run in five iterations, after which the results per setup are aggregated over these runs to minimise randomness.

*5.1.1 Evaluation and testing.* There are four main evaluation metrics that we are interested in. The first one is the quality of the

[3] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html, accessed June 1st, 2023.

feature selection, see figure 7. This is measured by the total percentage of informative features that the method extracted from the dataset (the true positives). Secondly, we measure the number of noisy features extracted from the dataset, see figure 8. These are features that are not informative and are thus considered false positives.

The third evaluation metric, figure 9, is the performance of the underlying model, after only including the selected features. This metric is valuable as our main goal is to increase underlying model performance, via feature selection. A separate split is used for measuring this performance, which was not used in any of the previous steps. We plot the average model performance for each configuration. Lastly, we measure the total execution time of the feature selection process for each configuration, as SHAP-based values are computationally expensive. These results are displayed in figure 10.

## 5.2 Application-based evaluation

Next to simulation-based evaluation, we will implement *IterSHAP* in a real-world scenario to test its ability to replace complex neural networks with simple traditional models. Furthermore, to test the working of *IterSHAP* on small high-dimensional datasets, we will insert several data segments and evaluate its performance.

*5.2.1 Scenario description.* The dataset used is called DEAP [12] and concerns an EEG 32-channel signal of participants watching and rating different kinds of videos. After watching these videos, participants rated the video on four levels: valence, arousal, dominance, and liking. With this dataset, both regression as well as classification analyses can be performed. For this research, two binary classes for the valence rating are created: 'high' if valence $\bar{5}$, 'low' otherwise. This results in a simple classification problem, allowing us to focus on the feature selection process.

The data consists of 32 participants each rating 40 videos, which totals to 1280 experiments. Each experiment consists of 32 channel signals of 63 seconds each at a rate of 128 Hz. The first three seconds of each signal is discarded, as these are considered to be transient. This leaves 1280 recordings of 7680 values times 32 channels.

The dataset is available via the Queen Mary University of London [4] after signing an End User License Agreement. As the dataset was released over ten years ago, numerous feature extraction and selection methods have been developed and validated using the DEAP dataset [22] [5] [14] [18].

*5.2.2 Feature selection comparison.* Li et al. have used 'Continuous Wavelet Transformation' (CWT) for feature extraction on the DEAP dataset [14]. Afterwards, they designed 'a hybrid deep learning model that combines the 'Convolutional Neural Network (CNN)' and 'Recurrent Neural Network (RNN)'.' In our research, we will use a simple feature extraction method, after which we will apply *IterSHAP* as feature selection method. Finally, we will use a default RandomForestClassifier model for classification.

*5.2.3 Feature extraction.* For each of the 1280 experiment, we divided the 32 channel signals into chunks of 128 frames without overlap, as this matches the approach of Li et al.. Then, we extracted 12 features per chunk. These are the mean, median, standard deviation, interquartile range, maximum value, and minimum value, in both the time domain as well as the frequency domain. The frequency domain data was acquired by apply a Fast Fourier Transformation on the original chunk. This results in a tabular dataset of 384 (12x32) features and 76800 (1280x60) samples. This is used as input for the experiments.

*5.2.4 Experimental setup.* Using the preprocessed dataset, *IterSHAP* as feature selection method, and a default RandomForestClassifier as model, we aim to compare the performance of *IterSHAP* in combination with a traditional, intrinsic explainable, model with the CNN used by Li et al. In order to highlight the applicability of *IterSHAP* on small high-dimensional datasets, we apply the two methods on different segments of the original dataset, as we did in section 5.1 as well.

These segments are 1.00, 0.50, 0.25, 0.10, 0.05, 0.025, 0.01, 0.005, 0.0025, and 0.001 of the total dataset. The remainder of the data is used to test the model performance afterwards. Each experiment setup is run five times, after which the performances are averaged.

*5.2.5 Evaluation metrics.* The performances of the *IterSHAP*/RFC combination and the CNN are compared on two metrics: accuracy and execution time. The accuracy depicts the classification accuracy on an unseen segment of the dataset. The *IterSHAP* execution time is the time from the start of *IterSHAP* until the processing of the test accuracy. The CNN execution time is the time from the first epoch until the processing of the test accuracy. These run times are counterparts, as both include the process from the end of the feature extraction until the final classification of the test dataset. Therefore, it is fair to compare these two run times.

# 6 RESULTS

In this section, we present the results of the validation frameworks described in sections 5.1 and 5.2

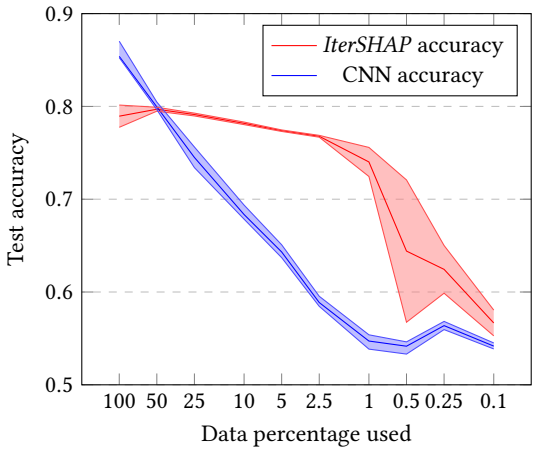
## 6.1 Simulation performance

The comprehensive results of the simulations on synthesized data can be found in appendix A. The simulation results of the Powershap algorithm on the large dataset correspond to the results presented by Verhaeghe et al., indicating that the algorithm was implemented correctly.

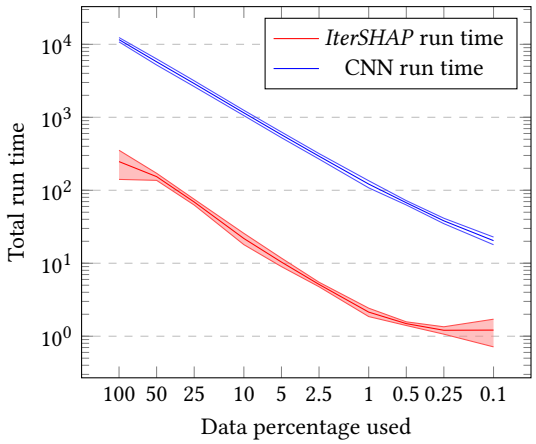
*6.1.1 Small dataset configurations.* For most configurations on the small dataset using a CatBoostClassifier, *IterSHAP* significantly outperforms Powershap in terms of informative features found (figure 7), model accuracy (figure 9), and run time (figure 10). However, *IterSHAP* tends to return more noisy features than Powershap (figure 8. This can be explained by the underlying technique used to select features, as *IterSHAP* selects a group of features at once, whereas Powershap selects features individually based on statistics.

When using a RidgeClassifier (figure 17 15, 16, and 18, *IterSHAP* outperforms Powershap on selecting informative features and model accuracy on relative low-dimensional datasets, whereas Powershap performs better on higher dimensional dataset ($\#F > 100$). In terms of run time, *IterSHAP* is significantly faster in most configurations. Only on some small dataset configurations, the run time is similar.

[4] https://www.eecs.qmul.ac.uk/mmv/datasets/deap/download.html




**(a) Semi-log plot of *IterSHAP* and CNN model accuracies**




**(b) Log-log plot of *IterSHAP* and CNN run time**

**Figure 6: Average model accuracy and run time of *IterSHAP* and an CNN on the DEAP dataset, including standard deviation.**

*6.1.2 Large dataset configurations.* When using the large dataset, *IterSHAP* tends to select all important features on the low-dimensional datasets. On the high-dimensional datasets, *IterSHAP* selects more than 90% of the important features, whereas Powershap tends to drop below 40% on the highest-dimensional dataset simulated. However, *IterSHAP* selects more noisy features than Powershap. Model accuracy after feature selection is similar for most configurations.

In terms of run time on large dataset configurations, *IterSHAP* and Powershap have similar run times, with *IterSHAP* outperforming Powershap more frequently than the other way around.

## 6.2 Application performance

The results of the application-based evaluation can be found in figure 6 and in table 5 (appendix section D). As can be seen in figure 6a, the performance of the CNN from Li et al. [14] slightly outperforms our *IterSHAP* with RFC when using the entire dataset. However, the CNN performance heavily decreases when using smaller data segments. On the contrary, *IterSHAP* remains at a decent accuracy until the moment the data segment drops below 2.5% of the original dataset, which approximately equals the observations of one participants of the DEAP study [12].

In terms of run time (figure 6b), we see that *IterSHAP* is between 20 to 100 times faster than the CNN. This can partly be explained as the CNN is trained for 50 epochs, which takes a lot of time. Furthermore, it can be observed that both *IterSHAP* and CNN scale linearly with the number of samples.

# 7 CONCLUSION

With this paper, we introduce *IterSHAP*: a novel feature selection method based on SHapley Additive exPlanations (SHAP), optimised for small high-dimensional datasets. To validate and test *IterSHAP*, we created a simulation-based experiment setup by varying the size and dimensionality of the input data. Furthermore, we varied the number of informative features. We tested *IterSHAP* against the current best-performing SHAP feature selection method *Powershap*. We found that on large datasets, *IterSHAP* has at least similar or better results than *Powershap*. On small datasets, *IterSHAP* consistently outperforms *Powershap*, which answers RQ 1.

Next to the simulation-based validation, we applied *IterSHAP* on the DEAP dataset [12]. By comparing *IterSHAP* to the CNN introduced by Li et al. [14], we showed that utilising *IterSHAP* allows for intrinsic explainable models to be used for classification. Furthermore, by varying the data portions used by the models, we showed the effectiveness of *IterSHAP* on smaller high-dimensional datasets, which answers RQ 2.

## 7.1 Contributions

Our main academic contribution is that, to our knowledge, we introduced the first feature selection method specialised for small high-dimensional datasets. Furthermore, our method guarantees to return a subset of features, without predefined cutoff point $K$ or underlying statistical needs for a large sample set. Additionally, the *IterSHAP* algorithm is model-agnostic, making sure it can be applied on any types of models.

Finally, *IterSHAP* is available as a plug-and-play Python package, compatible with a wide range of scikit-learn and other models. It can be installed via 'pip install itershap' [5].

# 8 DISCUSSION

In this section, we will describe the limitation of our research in combination with pointers for future work.

## 8.1 Descent path

The current descent path is not tested for optimization. Therefore, we do not know whether the current default step size, $\alpha$, is optimal or could be further optimised to lower run time or increase performance. We assume that the optimal $\alpha$ differs per dataset, so testing which value of $\alpha$ would generally work best can be tested using e.g.

[5] https://pypi.org/project/itershap/

the Armijo rule or any other learning rate optimisation rule. We leave this to future work.

## 8.2 Convergence to local optimum

When using a model with some intrinsic feature importance, *IterSHAP* is more likely to end up in a local optimum when a large dataset is used. This can be explained by the fact that these types of models can already determine for themselves which features contribute the most towards the target and hence will already have some form of internal feature selection. Therefore, it might happen that the model's accuracy does not change that much during the iterations of *IterSHAP* and hence the 'optimal' subset might have many noisy features. A possible solution would be a bottoms-up approach of *IterSHAP*, to mitigate the noisy features, however, due to the intrinsic design op *IterSHAP* this might be complex and computationally heavy.

## 8.3 Future work

Next to the possible improvements mentioned in sections 8.1 and 8.2, we address additional pointers for future research in this section.

*8.3.1 IterSHAP with Active learning.* As *IterSHAP* is specialised for smaller high-dimensional datasets, it could be applied to active learning, where the labelled part of the partially-labelled dataset could be considered the small high-dimensional dataset mentioned in this research. Using *IterSHAP*, highly-deviating samples, in terms of Shapley values, could be proposed to the human-in-the-loop to more efficiently label the entire dataset.

Using SHAP for active learning has been researched by Kara et al. [11], however, this was not focussed on smaller datasets.

*8.3.2 IterSHAP for regression problems.* Within this research, only classification problems were considered, to keep the research concise. Future research can be conducted into extending *IterSHAP* to cover regression problems as well.

*8.3.3 Step size optimization.* One of the parameters of *IterSHAP* is the step size, which indicates the portion of features to keep at each iteration of the algorithm. In the current implementation, the default value is 0.50, which has not been tested on whether it is optimized. Therefore, future research could be conducted into optimizing this step size.

*8.3.4 IterSHAP with filtering method for preprocessing.* As *IterSHAP* takes exponentially longer when increasing the number of features, it might be interesting to look at a preprocessing step before using *IterSHAP*. This preprocessing could be a filtering feature selection method, like Chi-square or Pearson's correlation, that is significantly faster to compute. This could also solve the potential issue of linear models when $\#features > \#samples$ [28].

*8.3.5 Applications of IterSHAP.* In this research, *IterSHAP* was applied on one specific application domain. To achieve a comprehensive assessment of *IterSHAP*'s capabilities, further investigations into diverse domains could be conducted.

*8.3.6 Software verification.* Future work includes the exploration of formal verification techniques to rigorously establish the correctness and consistency of *IterSHAP*. By subjecting IterSHAP to formal methods, its alignment with intended behavior can be formally proven, supporting its reliability in real-world applications.

*8.3.7 Performance optimization.* Currently, the algorithm is only implemented in Python, which makes it accessible and easy to read for users. However, Python is considered slow and thus future work could look into implementing *IterSHAP* in other programming languages, for instance, C, that makes use of GPU parallel computing.

## REFERENCES


[1] Ian Alexander. 2007. A Taxonomy of Stakeholders. *International Journal of Technology and Human Interaction* 1 (07 2007), 23–59. https://doi.org/10.4018/jthi.2005010102

[2] Richard Bellman. 1957. *Dynamic Programming*. Dover Publications, New York.

[3] Dimitris Bertsimas, Galit Lukin, Luca Mingardi, Omid Nohadani, Agni Orfanoudaki, Bartolomeo Stellato, Holly Wiberg, Sara Gonzalez-Garcia, Carlos Luis Parra-Calderón, Kenneth Robinson, Michelle Schneider, Barry Stein, Alberto Estirado, Lia a Beccara, Rosario Canino, Martina Dal Bello, Federica Pezzetti, Angelo Pan, and The Hellenic COVID-19 Study Group. 2020. COVID-19 mortality risk assessment: An international multi-center study. *PLOS ONE* 15, 12 (12 2020), 1–13. https://doi.org/10.1371/journal.pone.0243262

[4] Royal Society (Great Britain) and Royal Society (Great Britain). 1854. *Proceedings of the Royal Society of London*. Vol. v.7=no.[37-42] (1854-1855). London, The Society, 1855-1905, London. 648 pages. https://www.biodiversitylibrary.org/item/60968 https://www.biodiversitylibrary.org/bibliography/67976.

[5] Jing Chen, Bin Hu, Lixin Xu, Philip Moore, and Yun Su. 2015. Feature-level fusion of multimodal physiological signals for emotion recognition. In *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, Washington, DC, 395–399. https://doi.org/10.1109/BIBM.2015.7359713

[6] Xue-wen Chen and Jong Cheol Jeong. 2007. Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*. IEEE, Cincinnati, Ohio, 429–435. https://doi.org/10.1109/ICMLA.2007.35

[7] Sergiu Hart. 1989. *Shapley Value*. Palgrave Macmillan UK, London, 210–216. https://doi.org/10.1007/978-1-349-20181-5_25

[8] Mir Islam, Mobyen Ahmed, Shaibal Barua, and Shahina Begum. 2022. A Systematic Review of Explainable Artificial Intelligence in Terms of Different Application Domains and Tasks. *Applied Sciences* 12 (01 2022), 1353. https://doi.org/10.3390/app12031353

[9] Xin Jin, Anbang Xu, Rongfang Bie, and Ping Guo. 2006. Machine Learning Techniques and Chi-Square Feature Selection for Cancer Classification Using SAGE Gene Expression Profiles. In *Data Mining for Biomedical Applications*, Jinyan Li, Qiang Yang, and Ah-Hwee Tan (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 106–115.

[10] A. Jović, K. Brkić, and N. Bogunović. 2015. A review of feature selection methods with applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, Opatija, Croatia, 1200–1205. https://doi.org/10.1109/MIPRO.2015.7160458

[11] Nailcan Kara, Yagiz Levent Gume, Umit Tigrak, Gokce Ezeroglu, Serdar Mola, Omer Burak Akgun, and Arzucan Özgür. 2022. A SHAP-based Active Learning Approach for Creating High-Quality Training Data. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, Osaka, Japan, 4002–4008. https://doi.org/10.1109/BigData55660.2022.10020327

[12] Sander Koelstra, C. Mühl, Mohammad Soleymani, Jung Seok Lee, Ashkan Yazdani, Touradj Ebrahimi, Thierry Pun, Antinus Nijholt, and Ioannis Patras. 2012. DEAP: A Database for Emotion Analysis Using Physiological Signals. *IEEE transactions on affective computing* 3, 1 (2012), 18–31. https://doi.org/10.1109/T-AFFC.2011.15 eemcs-eprint-21368.

[13] Miron B. Kursa and Witold R. Rudnicki. 2010. Feature Selection with the Boruta Package. *Journal of Statistical Software* 36, 11 (2010), 1–13. https://doi.org/10.18637/jss.v036.i11

[14] Xiang Li, Dawei Song, Peng Zhang, Guangliang Yu, Yuexian Hou, and Bin Hu. 2016. Emotion recognition from multi-channel EEG data through Convolutional Recurrent Neural Network. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, Shenzhen, China, 352–359. https://doi.org/10.1109/BIBM.2016.7822545

[15] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., Long Beach, California. https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[16] A. Marcano-Cedeño, J. Quintanilla-Domínguez, M. G. Cortina-Januchs, and D. Andina. 2010. Feature selection using Sequential Forward Selection and classification applying Artificial Metaplasticity Neural Network. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, Glendale, AZ, USA, 2845–2850. https://doi.org/10.1109/IECON.2010.5675075

[17] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267 (2019), 1–38. https://doi.org/10.1016/j.artint.2018.07.007

[18] Bahareh Nakisa, Mohammad Naim Rastgoo, Dian Tjondronegoro, and Vinod Chandran. 2018. Evolutionary computation algorithms for feature selection of EEG-based emotion recognition using mobile sensors. *Expert Systems with Applications* 93 (2018), 143–155. https://doi.org/10.1016/j.eswa.2017.09.062

[19] Hanchuan Peng, Fuhui Long, and C. Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 8 (2005), 1226–1238. https://doi.org/10.1109/TPAMI.2005.159

[20] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. 2007. Filter Methods for Feature Selection - A Comparative Study. In *Ideal*. Springer, Berlin, Heidelberg, Birmingham, UK, 178–187.

[21] Lloyd S Shapley. 1953. Stochastic games. *Proceedings of the national academy of sciences* 39, 10 (1953), 1095–1100.

[22] Elham Shawky, Reda El-Khoribi, Mahmoud Shoman, and Mohamed Wahby Shalaby. 2018. EEG-Based Emotion Recognition using 3D Convolutional Neural Networks. *International Journal of Advanced Computer Science and Applications* 9 (09 2018), 329.

[23] Le Song, Alex Smola, Arthur Gretton, Karsten M. Borgwardt, and Justin Bedo. 2007. Supervised Feature Selection via Dependence Estimation. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. Association for Computing Machinery, New York, NY, USA, 823–830. https://doi.org/10.1145/1273496.1273600

[24] Student. 1908. The Probable Error of a Mean. *Biometrika* 6, 1 (1908), 1–25. http://www.jstor.org/stable/2331554

[25] Jarne Verhaeghe, Jeroen Van Der Donckt, Femke Ongenae, and Sofie Van Hoecke. 2023. Powershap: A Power-Full Shapley Feature Selection Method. In *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, Cham, 71–87.

[26] Dimitrios Ververidis and Constantine Kotropoulos. 2005. Sequential forward feature selection with low computational cost. In *2005 13th European Signal Processing Conference*. IEEE, Antalya, Turkey, 1–4.

[27] Guangfen Wei, Jie Zhao, Yanli Feng, Aixiang He, and Jun Yu. 2020. A novel hybrid feature selection method based on dynamic feature importance. *Applied Soft Computing* 93 (2020), 106337. https://doi.org/10.1016/j.asoc.2020.106337

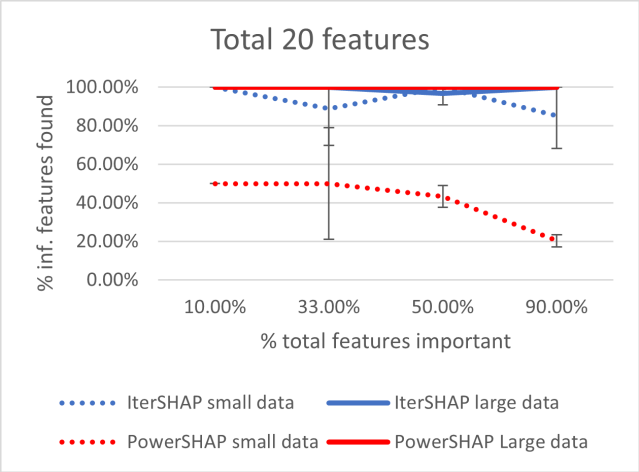
[28] Mike West. 2002. Bayesian Factor Regression Models in the "Large p, Small n" Paradigm. *Bayesian Stat.* 7 (08 2002).

[29] Roelf J. Wieringa. 2014. *Design science methodology for information systems and software engineering*. Springer, Enschede, The Netherlands. https://doi.org/10.1007/978-3-662-43839-8 10.1007/978-3-662-43839-8.

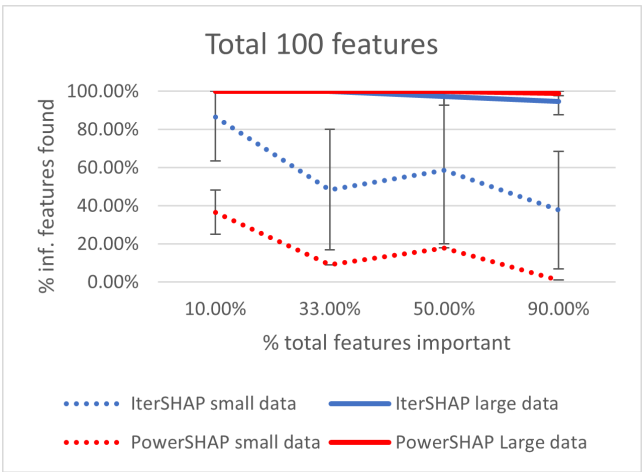
[30] Liping Xie, Zilong Li, Yihan Zhou, Yiliu He, and Jiaxin Zhu. 2020. Computational Diagnostic Techniques for Electrocardiogram Signal Analysis. *Sensors* 20, 20 (11 2020).

**APPENDICES**

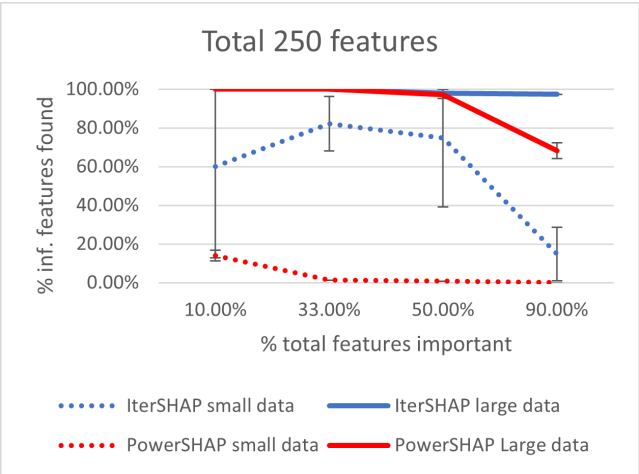
## A Performance plots CatBoostClassifier



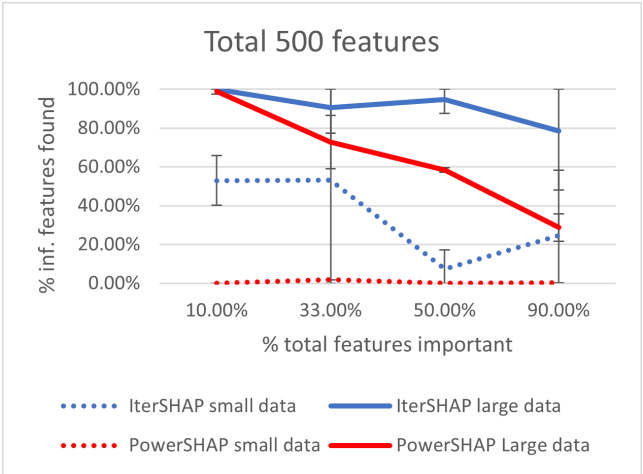

(a) Total 20 features




(b) Total 100 features

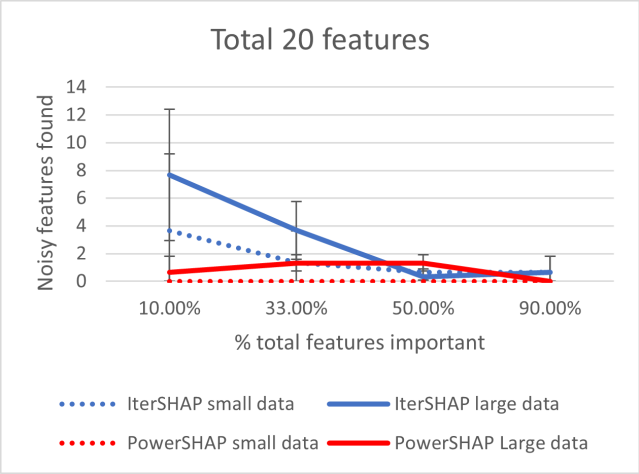



(c) Total 250 features

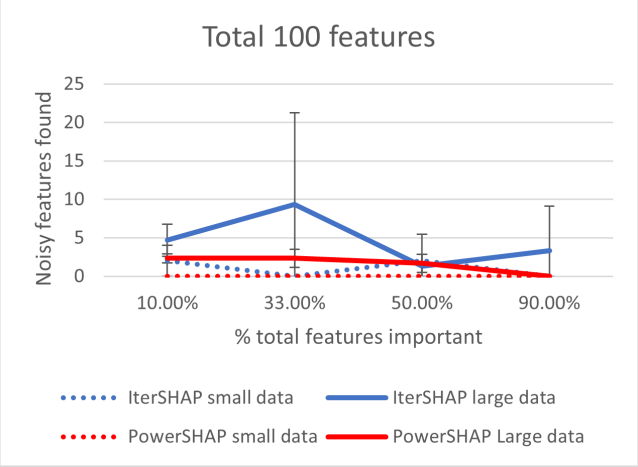



(d) Total 500 features

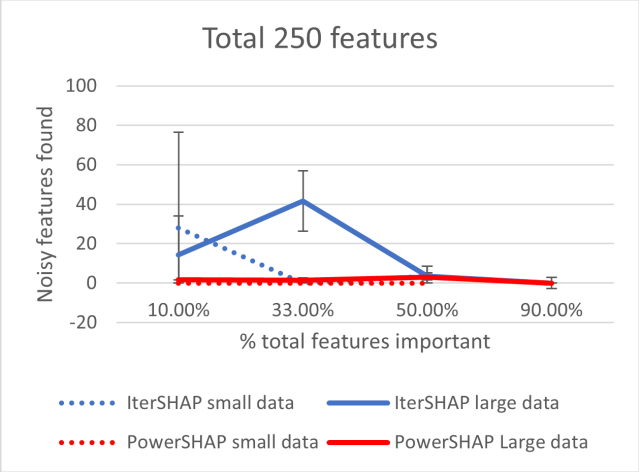
**Figure 7: The percentage of informative features found with varying total nr. of features and percentage of important features**



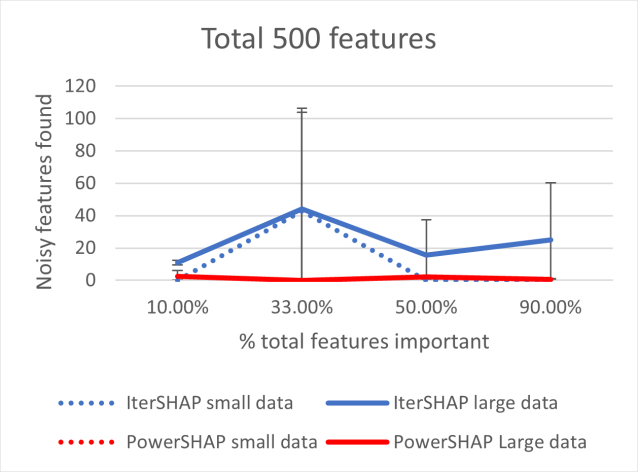

(a) Total 20 features




(b) Total 100 features

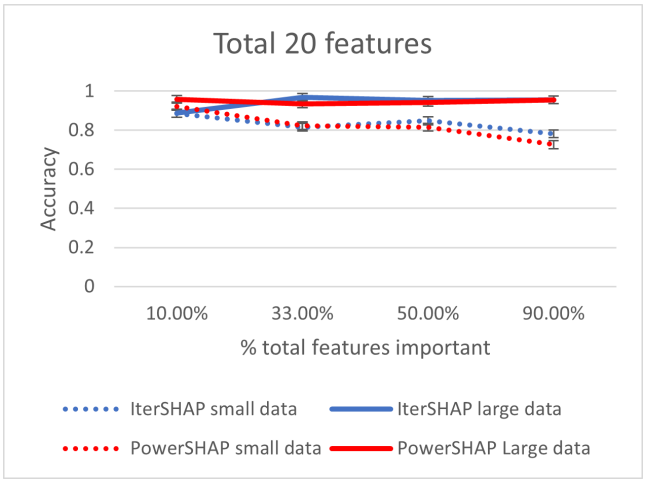



(c) Total 250 features

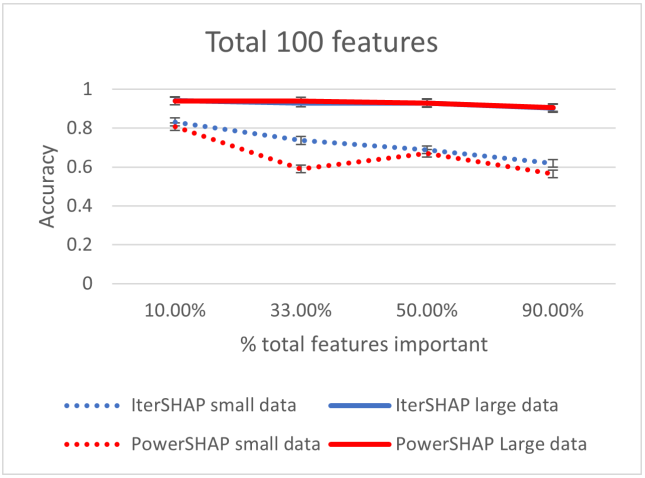



(d) Total 500 features

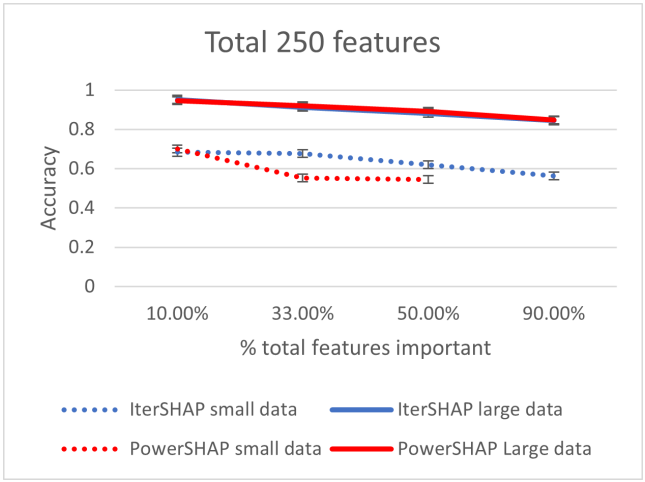
**Figure 8: The number of noisy features found with varying total nr. of features and percentage of important features**



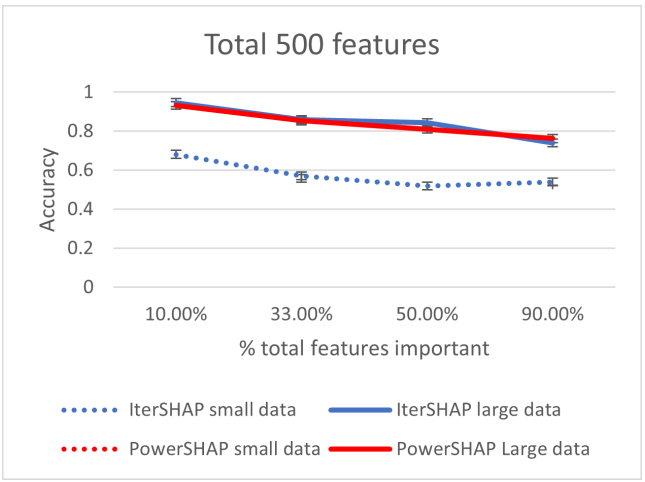

**(a) Total 20 features**




**(b) Total 100 features**

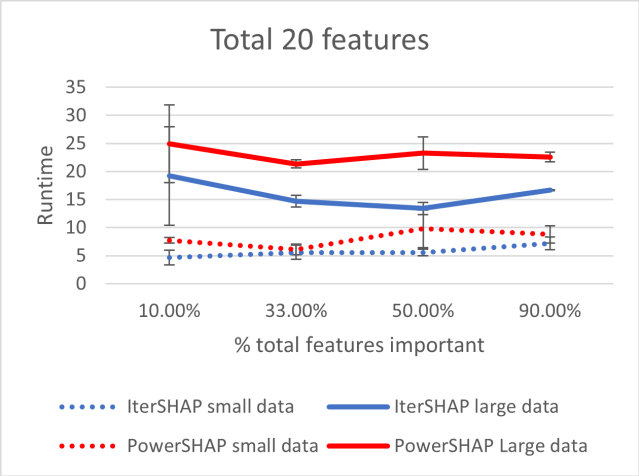



**(c) Total 250 features**

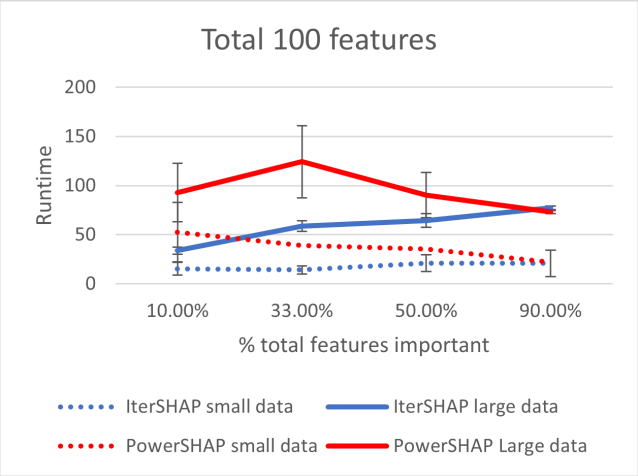



**(d) Total 500 features**

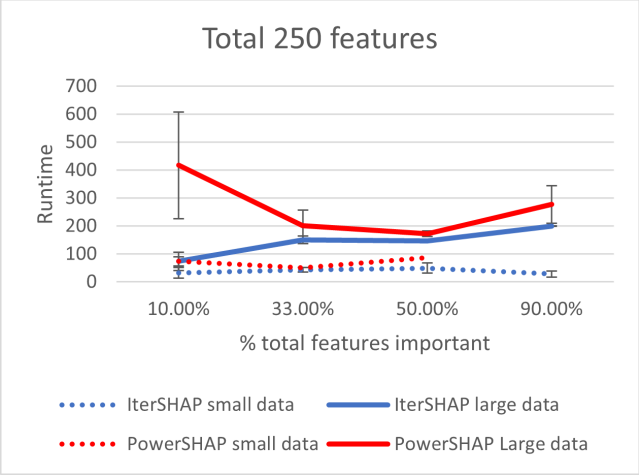
**Figure 9: The accuracy of a trained model after feature selection with varying total nr. of features and percentage of important features**



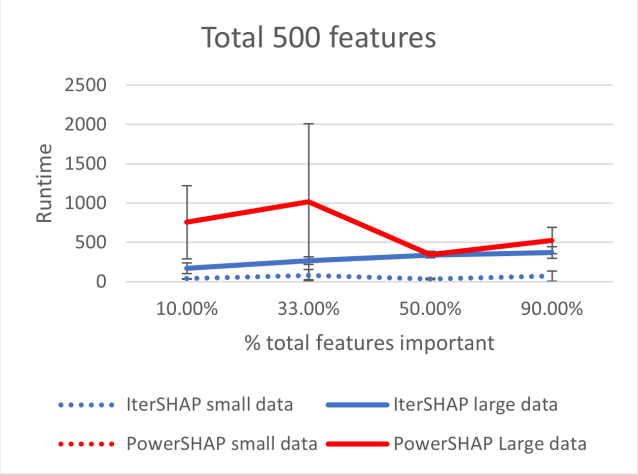

(a) Total 20 features




(b) Total 100 features



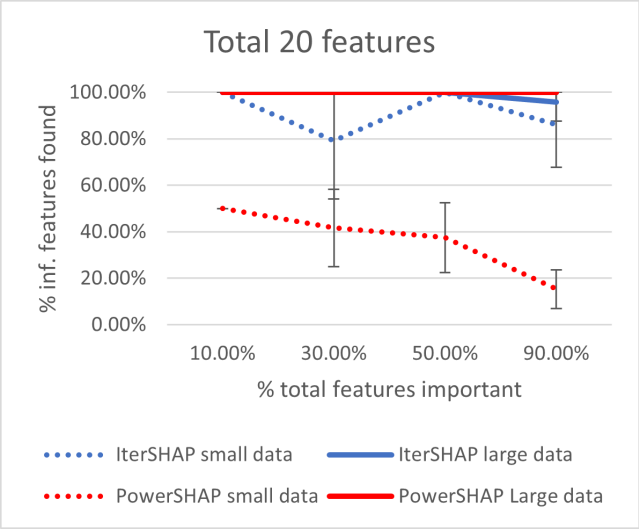

(c) Total 250 features



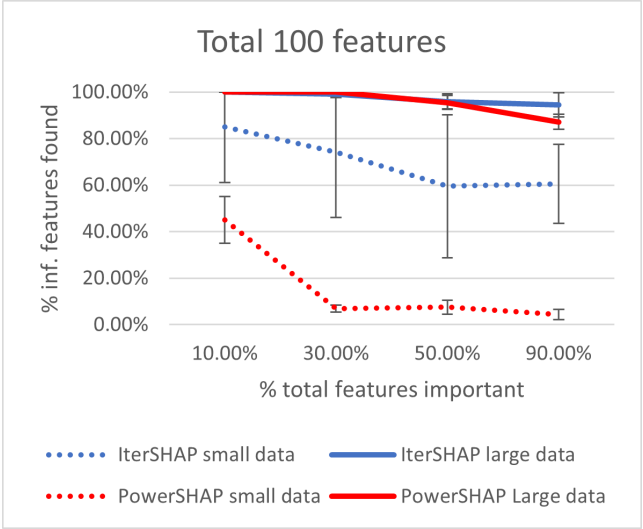

(d) Total 500 features

**Figure 10: The run time of the feature selection process with varying total nr. of features and percentage of important features**

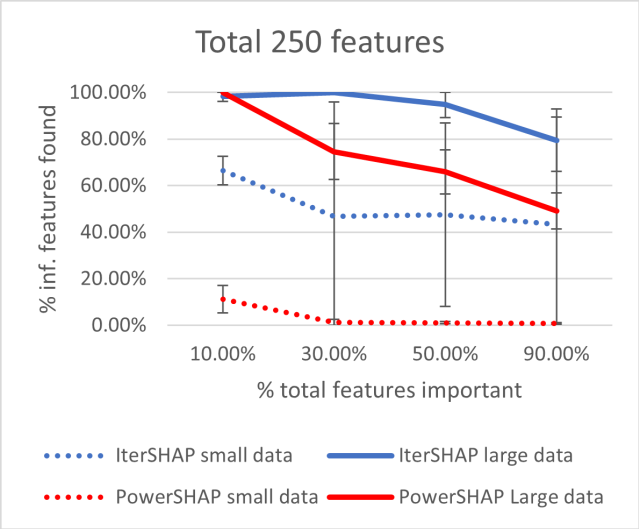
## B Performance plots RandomForestClassifier



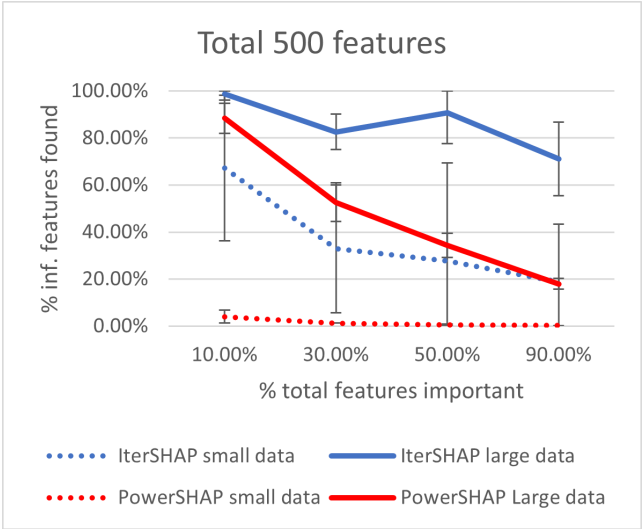

(a) Total 20 features




(b) Total 100 features

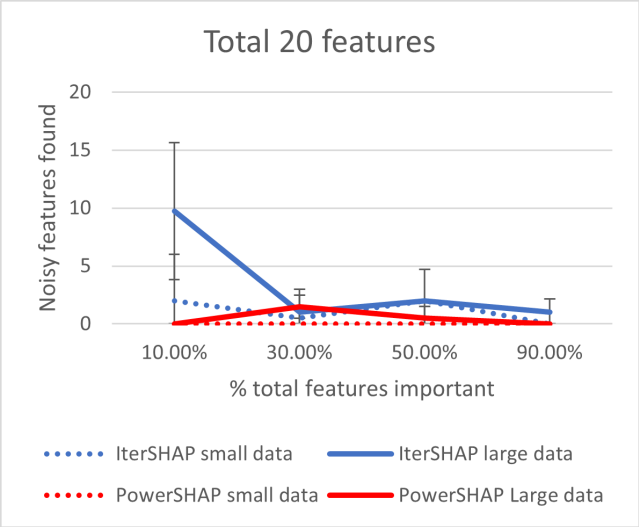



(c) Total 250 features

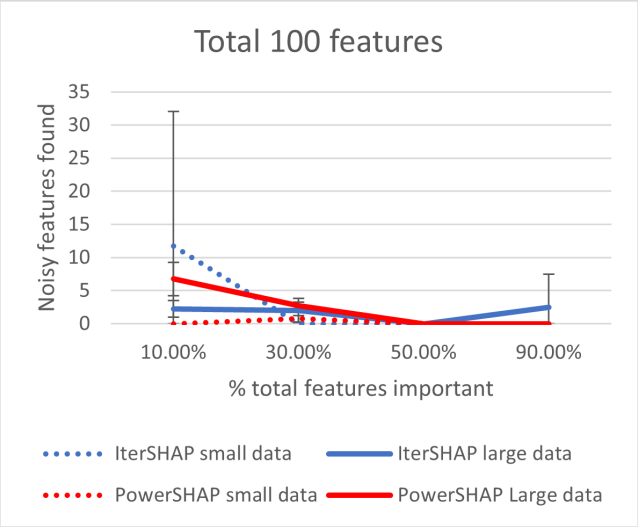



(d) Total 500 features

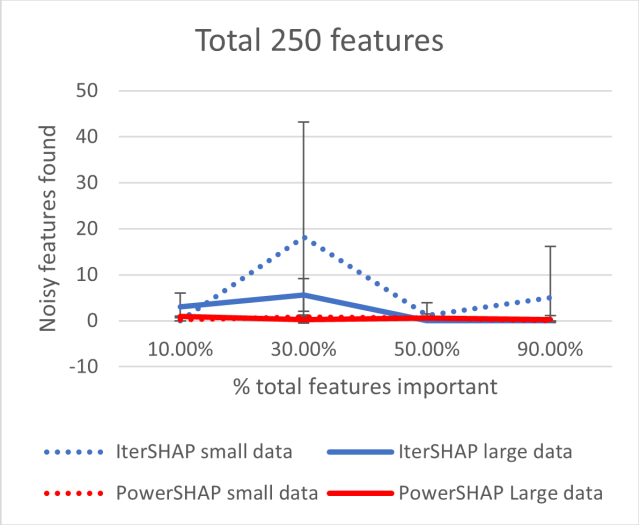
**Figure 11: The percentage of informative features found with varying total nr. of features and percentage of important features**



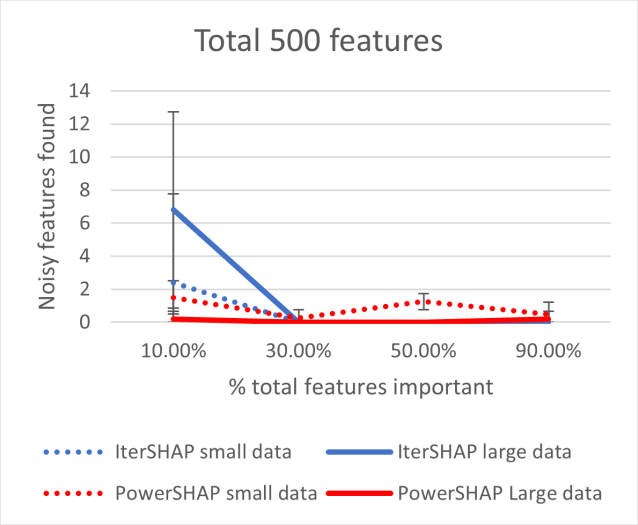

(a) Total 20 features




(b) Total 100 features

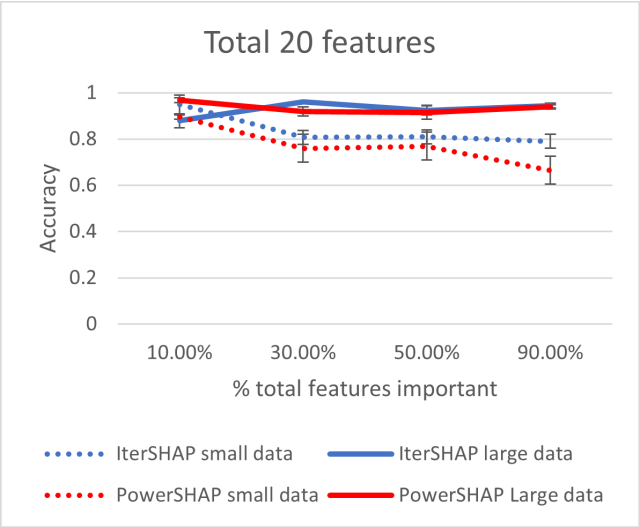



(c) Total 250 features

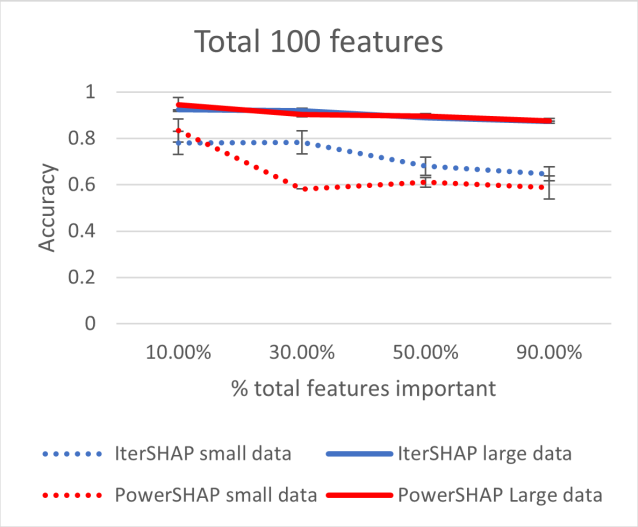



(d) Total 500 features

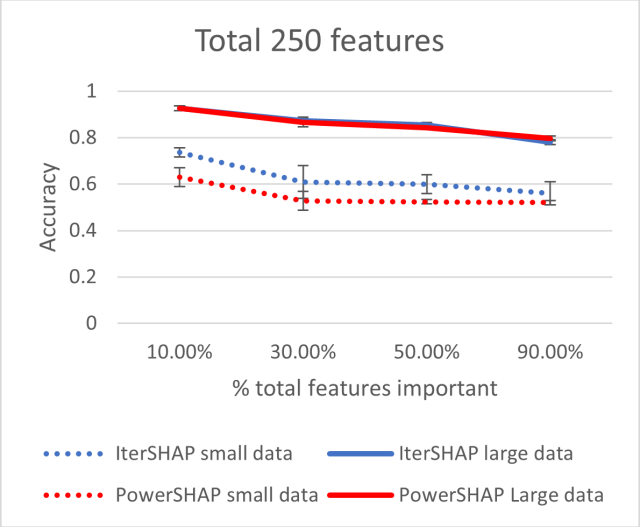
**Figure 12: The number of noisy features found with varying total nr. of features and percentage of important features**



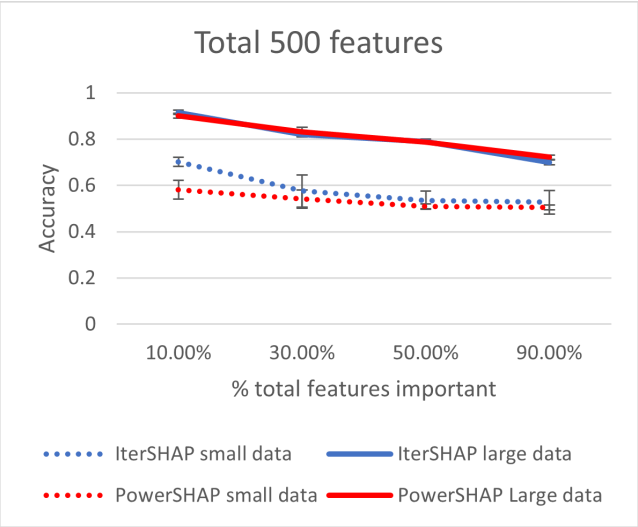

(a) Total 20 features




(b) Total 100 features

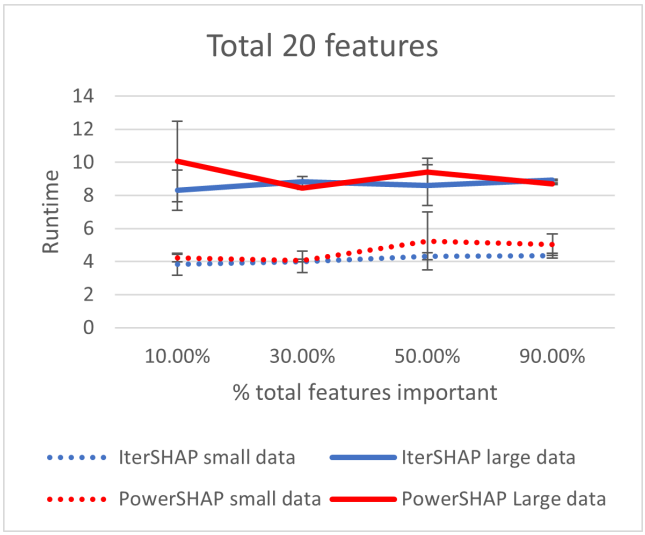



(c) Total 250 features

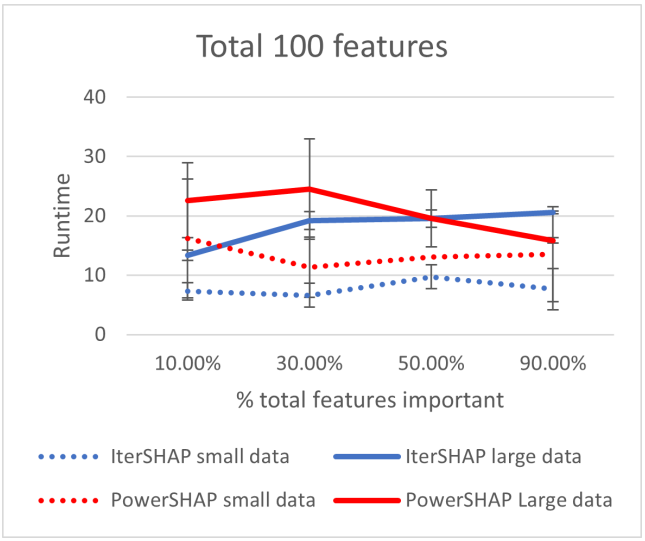



(d) Total 500 features

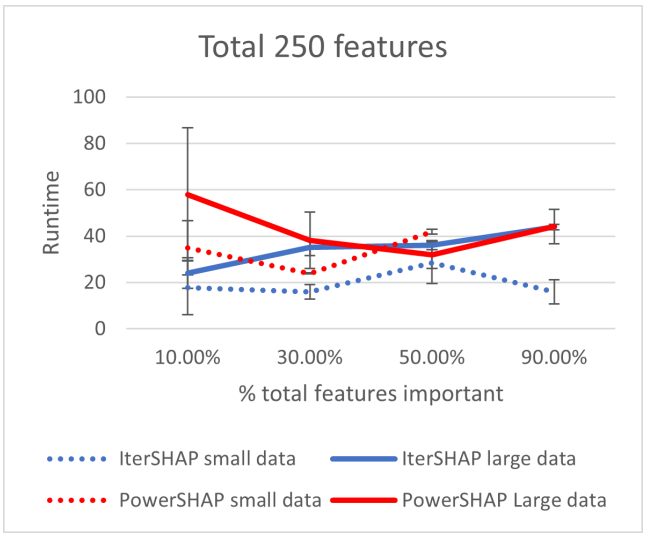
**Figure 13: The accuracy of a trained model after feature selection with varying total nr. of features and percentage of important features**



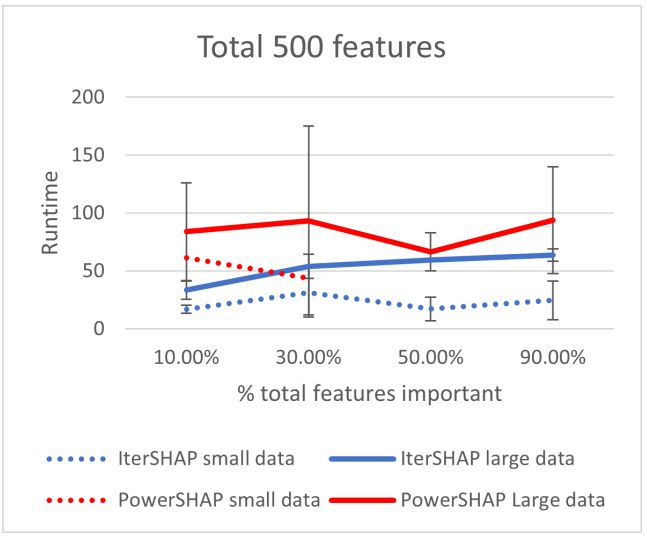

(a) Total 20 features




(b) Total 100 features



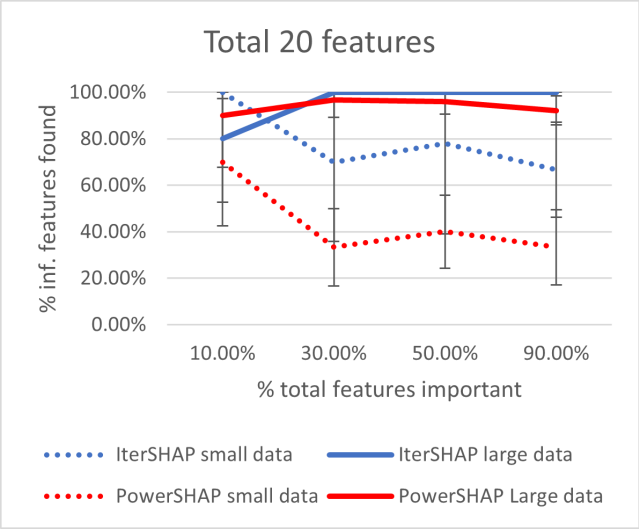

(c) Total 250 features



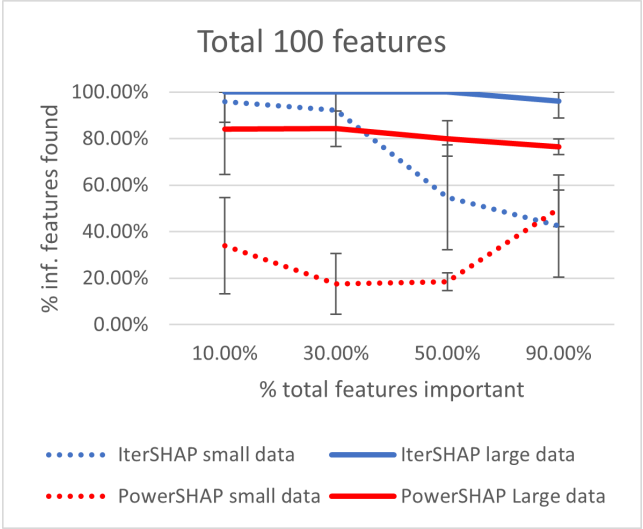

(d) Total 500 features

**Figure 14: The run time of the feature selection process with varying total nr. of features and percentage of important features**

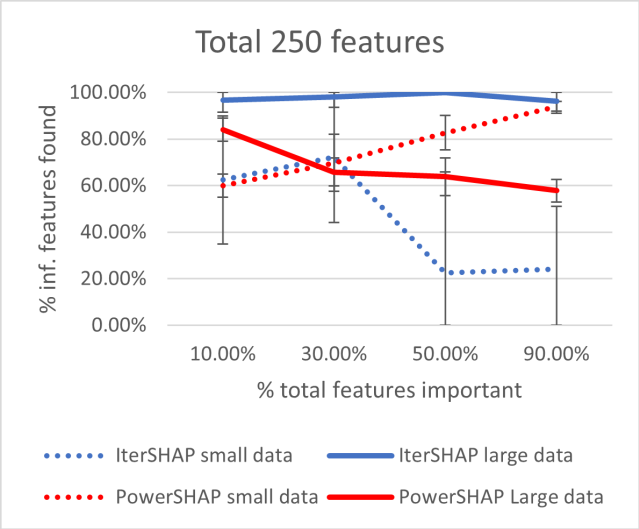
## C Performance plots RidgeClassifier



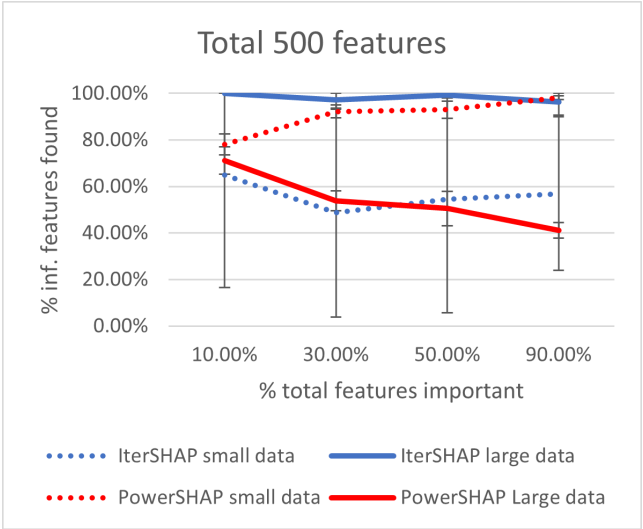

(a) Total 20 features




(b) Total 100 features

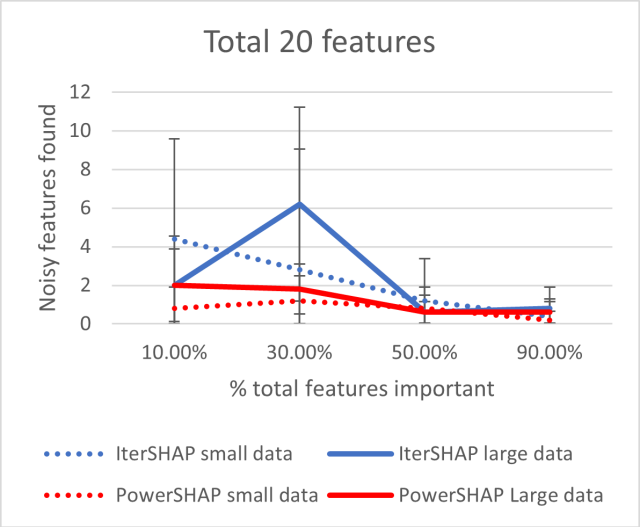



(c) Total 250 features

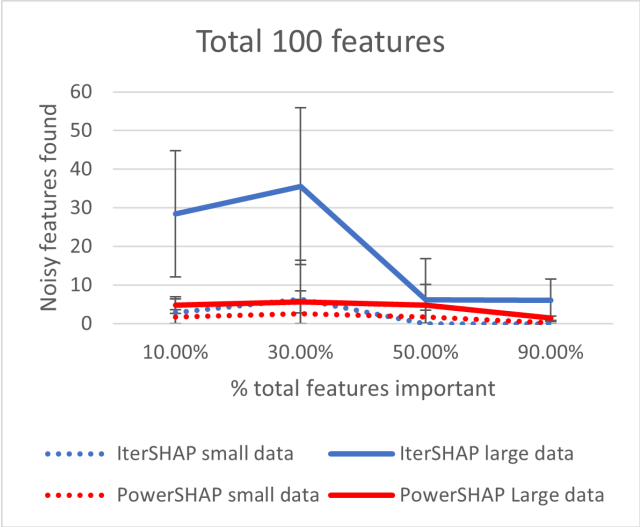



(d) Total 500 features

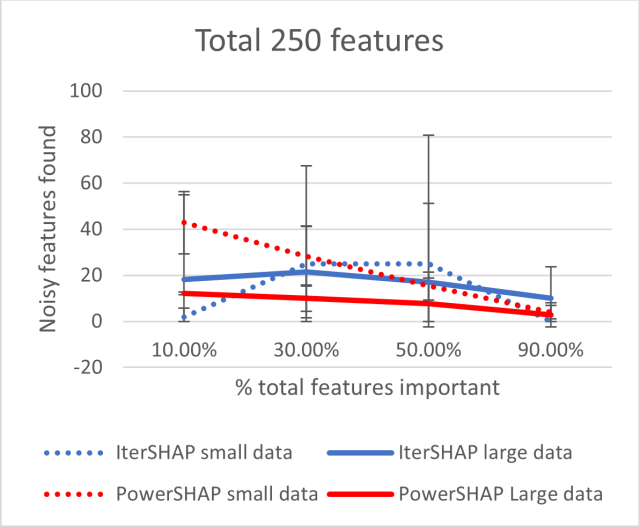
**Figure 15: The percentage of informative features found with varying total nr. of features and percentage of important features**



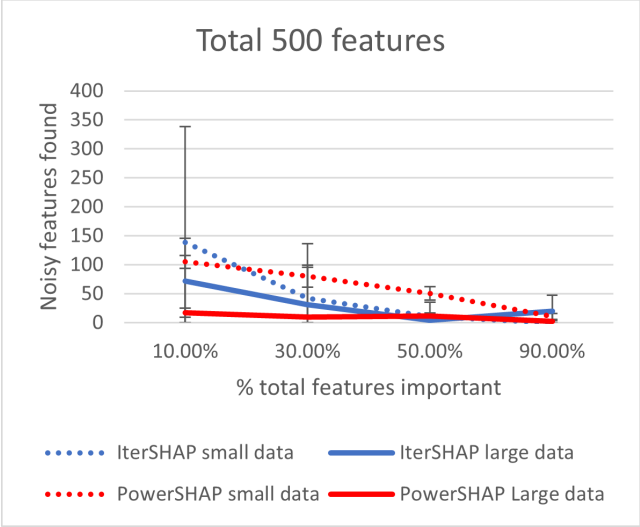

(a) Total 20 features




(b) Total 100 features

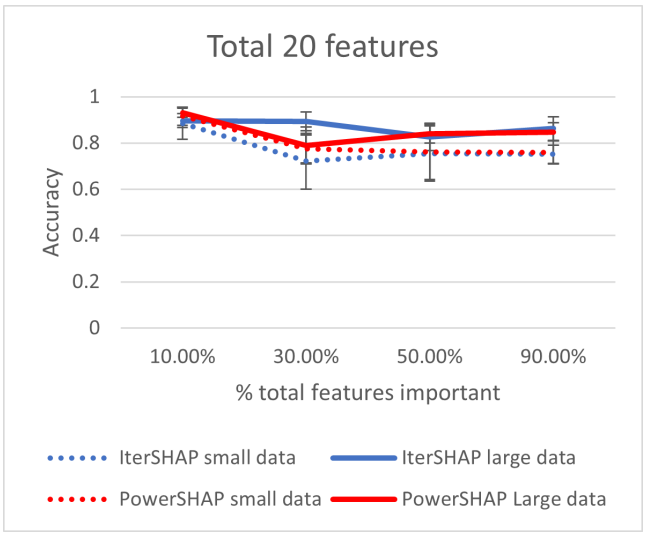



(c) Total 250 features

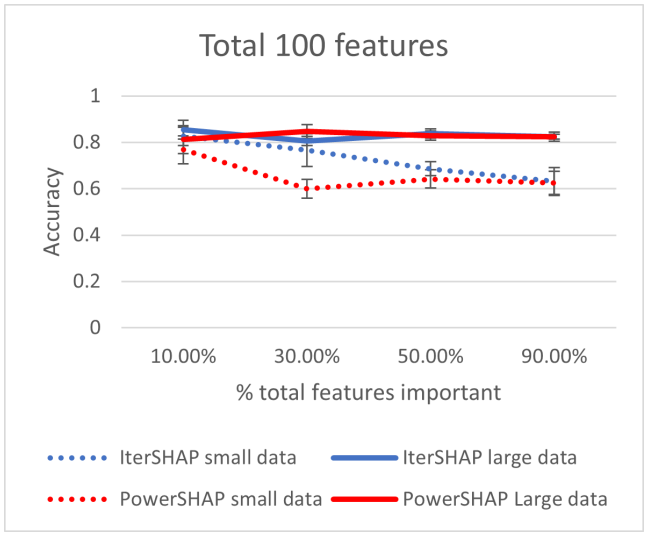



(d) Total 500 features

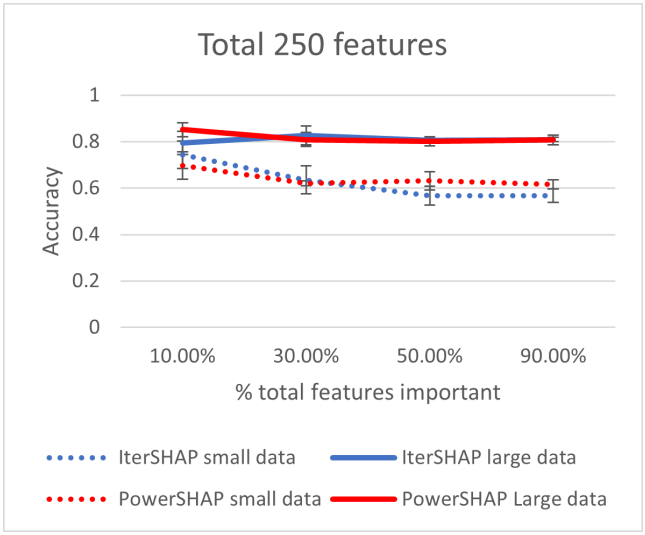
**Figure 16: The number of noisy features found with varying total nr. of features and percentage of important features**



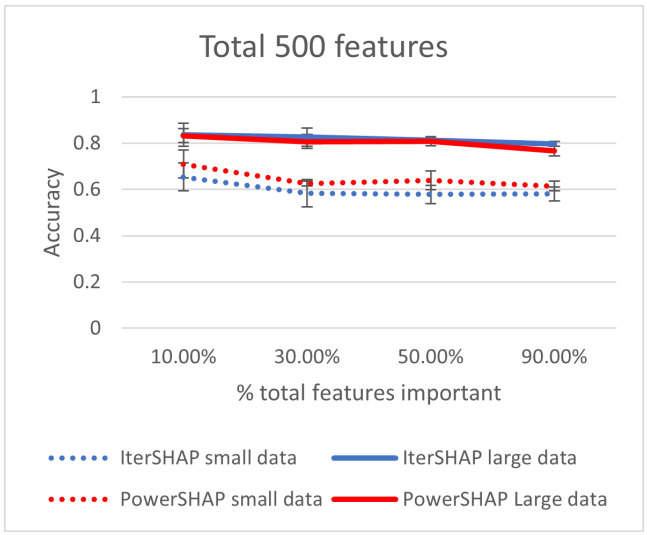

(a) Total 20 features




(b) Total 100 features

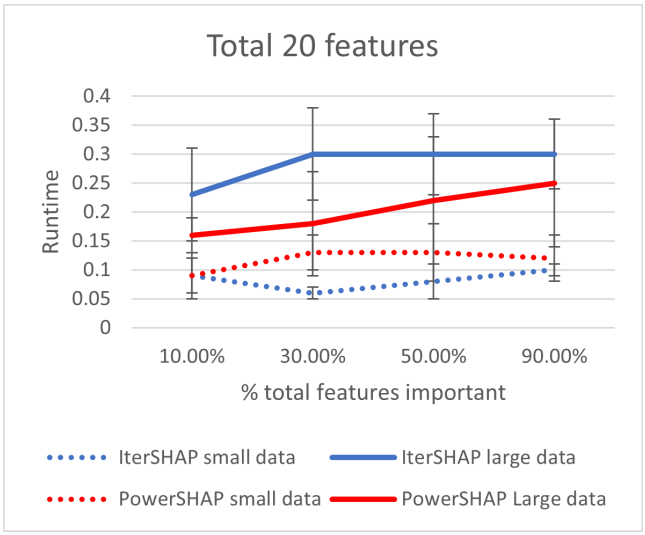



(c) Total 250 features

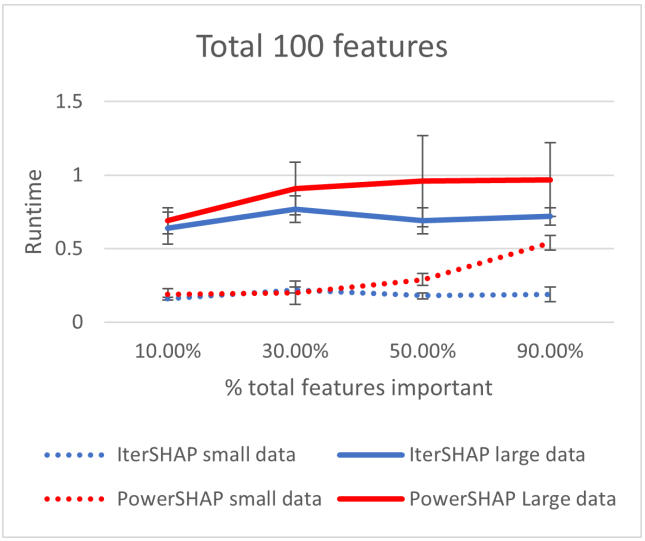



(d) Total 500 features

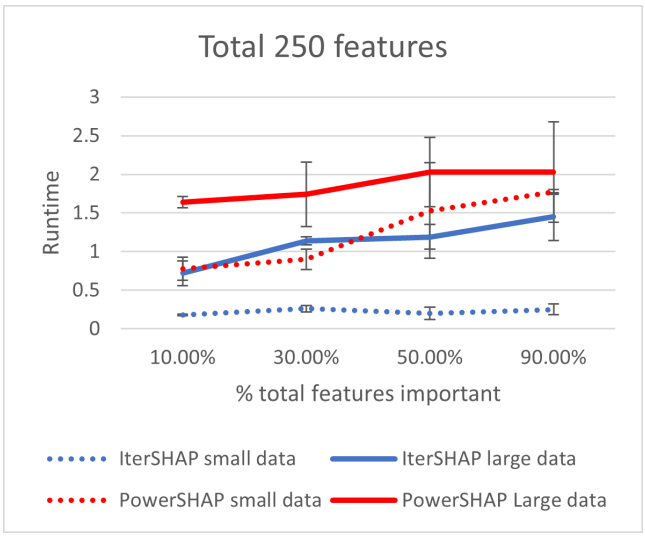
**Figure 17: The accuracy of a trained model after feature selection with varying total nr. of features and percentage of important features**



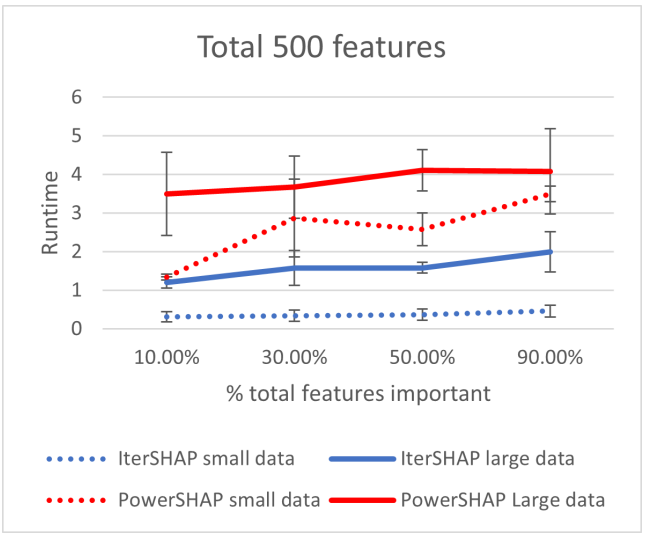

(a) Total 20 features




(b) Total 100 features




(c) Total 250 features




(d) Total 500 features

**Figure 18: The run time of the feature selection process with varying total nr. of features and percentage of important features**

## D Tabular results of *IterSHAP* vs. CNN

| | IterSHAP + RFC | | | | CNN | | | |
|---|---|---|---|---|---|---|---|---|
| Data segment | Mean acc | Std. dev. acc | Mean run time | Std. dev. run time | Mean acc | Std. dev. acc | Mean run time | Std. dev. run time |
| 100% | 0.789 | 0.012 | 247.104 | 106.211 | 0.861 | 0.009 | 11591.883 | 786.753 |
| 50% | 0.797 | 0.002 | 153.815 | 17.063 | 0.801 | 0.004 | 5760.694 | 550.764 |
| 25% | 0.791 | 0.002 | 69.059 | 6.265 | 0.745 | 0.011 | 2899.711 | 264.014 |
| 10% | 0.782 | 0.001 | 21.836 | 3.845 | 0.686 | 0.008 | 1150.883 | 95.874 |
| 5% | 0.774 | 0.001 | 10.373 | 1.383 | 0.644 | 0.007 | 576.600 | 51.438 |
| 2.5% | 0.768 | 0.001 | 5.130 | 0.366 | 0.590 | 0.005 | 293.062 | 25.328 |
| 1% | 0.740 | 0.016 | 2.141 | 0.281 | 0.546 | 0.008 | 121.530 | 13.573 |
| 0.5% | 0.644 | 0.077 | 1.489 | 0.087 | 0.540 | 0.007 | 67.489 | 4.604 |
| 0.25% | 0.624 | 0.026 | 1.207 | 0.140 | 0.564 | 0.004 | 38.113 | 3.255 |
| 0.1% | 0.567 | 0.014 | 1.213 | 0.499 | 0.542 | 0.003 | 20.397 | 2.454 |

**Table 5: Model accuracy and run time of *IterSHAP* and a CNN on the DEAP dataset.**